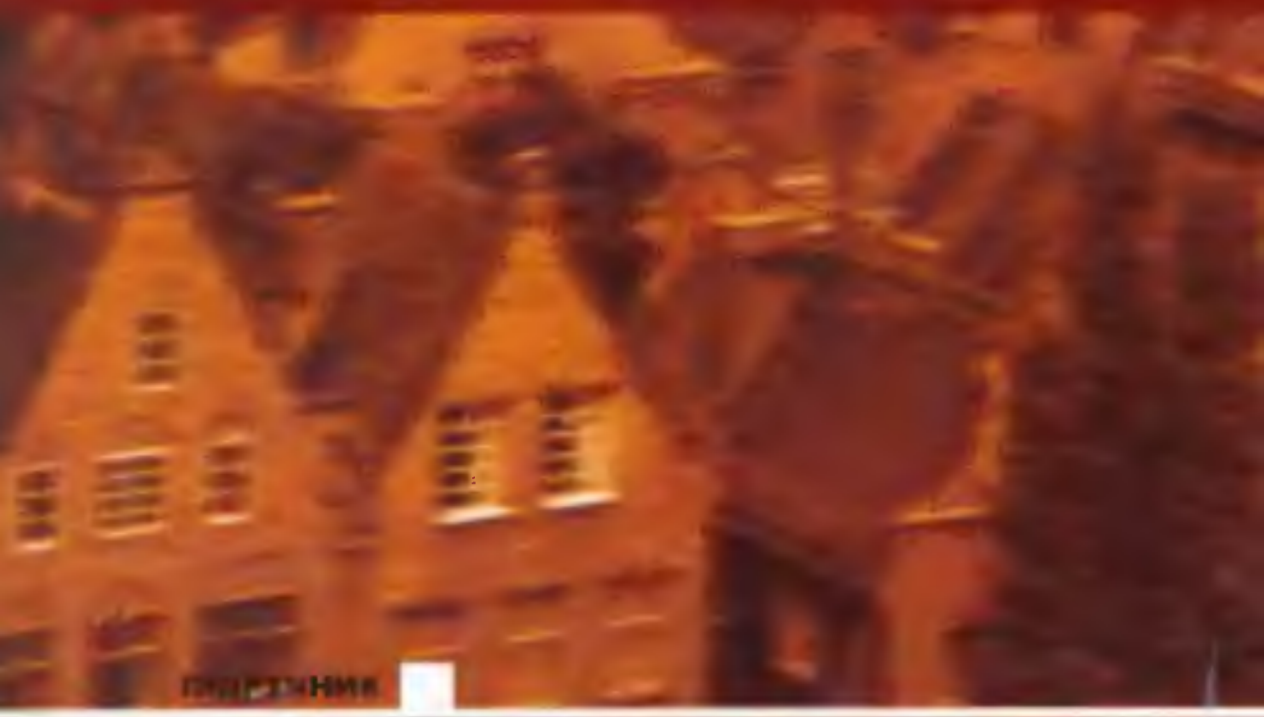
ІНФОРМАТИКА

В. В. Пасічник, В. А. Резниченко

# ОРГАНІЗАЦІЯ БАЗ ДАНИХ ТА ЗНАНЬ



ПІДРУЧНИК

ДЛЯ ВИЩИХ НАВЧАЛЬНИХ ЗАКЛАДІВ

BHV ПИТЕР

ББК 32.973-018.2я73
П19
УДК 004.6(075.8)

Рецензенти: *П. І. Андон*, доктор фізико-математичних наук, член-кореспондент НАН України, директор Інституту програмних систем НАН України;
*Г. О. Цейтлін*, доктор технічних наук, професор, завідувач кафедри програмного забезпечення автоматизованих систем Міжнародного Соломонова університету;
*В. М. Синєглазов*, доктор технічних наук, професор, директор Інституту електроніки та систем управління Національного авіаційного університету.

*Гриф надано Міністерством освіти і науки України, лист № 14/18.2-332 від 13.02.2006 р.*

**Пасічник В. В., Резніченко В. А.**

П19 Організація баз даних та знань. — К.: Видавнича група BHV, 2006. — 384 с.: іл.
ISBN 966-552-156-X

У підручнику детально і грунтовно викладено основи класичної та сучасної теорії, а також важливі практичні аспекти організації баз даних та знань. Докладно розглянуто реляційну модель даних, реляційну алгебру Кодда, реляційне числення та теорію проектування баз даних. Значну увагу приділено мові SQL, а також методиці побудови різнотипних запитів. Окремий розділ присвячено мові маніпулювання даними QBE. Викладено основи фізичної організації та захисту баз даних. Крім реляційних, у підручнику досить докладно розглядаються розподілені, паралельні та об'єктно-орієнтовані системи управління базами даних, а також бази даних в Інтернеті. Досить повно висвітлені питання організації баз знань.

Для студентів, які навчаються за напрямом «Комп'ютерні науки», може бути рекомендований студентам усіх спеціальностей, які вивчають курс «Організація баз даних та знань», «Бази даних та інформаційні системи» тощо. Підручник може стати в пригоді студентам, які навчаються за напрямом «Комп'ютерні науки» і вивчають сучасні інформаційні технології в межах дисципліни «Основи програмування та алгоритмічні мови», а також для викладачів зазначеної дисципліни.

**ББК 32.973-018.2я73**



ISBN 966-552-156-X 

# Стислий зміст

# Зміст

# Вступ

Від початку розвитку обчислювальної техніки сформувалися два основні напрями її використання.

Перший напрям – застосування комп'ютерів для виконання обчислень, які дуже складно або неможливо здійснювати вручну. Становлення цього напряму сприяло інтенсифікації методів чисельного розв'язання складних математичних задач і розвитку мов програмування, які орієнтовані на зручний запис чисельних алгоритмів.

Другий напрям – це використання засобів обчислювальної техніки в системах обробки даних та автоматизованих інформаційних системах. У широкому розумінні *інформаційна система* є програмним комплексом, функції якого полягають у забезпеченні надійного зберігання даних у пам'яті комп'ютера, виконанні специфічних для певного програмного застосування перетворень інформації та/або обчислень, наданні користувачам зручного й легкого в освоєнні інтерфейсу. Зазвичай обсяги інформації, з якими працюють такі системи, досить великі, а самі вони мають складну структуру. Класичними прикладами інформаційних систем є банківські, системи резервування авіаційних або залізничних квитків, місць у готелях тощо.

Другий напрям виник пізніше за перший, що було пов'язано з обмеженими можливостями комп'ютерів у галузі зберігання даних на зовнішніх носіях.

Системи керування даними в зовнішній пам'яті з'явилися лише після винайдення магнітних дисків. До цього прикладні програми самі визначали розташування даних на магнітній стрічці чи барабані й здійснювали обмін інформацією між оперативною та зовнішньою пам'яттю за допомогою програмно-апаратних засобів низького рівня (машинних команд або викликів відповідних програм операційної системи). Такий спосіб роботи не дозволяв або дуже ускладнював підтримку на одному зовнішньому носії кількох архівів інформації, якщо вони мали зберігатися тривалий час. Крім того, для кожної прикладної програми доводилося вирішувати проблеми іменування частин даних та їхньої структуризації в зовнішній пам'яті.

Кроком уперед був перехід до використання централізованих систем керування файлами. Для прикладної програми файл – це іменована область зовнішньої пам'яті, куди можна записувати й звідки можна зчитувати дані. Правила іменування файлів, спосіб доступу до даних, що в них зберігаються, та структура цих даних залежать зазвичай від конкретної системи керування файлами, інколи – від типу файла. Система керування файлами, або *файлова система* (ФС), яку ще називають файловою підсистемою операційної системи, виконує функції розподілу зовнішньої пам'яті, відображення імен файлів на відповідні адреси пам'яті та забезпечення доступу до даних.

Файлова система має низку особливостей, які значно обмежують її безпосереднє застосування у зберіганні й обробці великих обсягів складноструктурованих даних, призначених для колективного використання. Розглянемо найістотніші з них.

Завдяки використанню файлової системи встановлюється тісний зв'язок між фізичними даними і прикладною програмою. Файл створюється програмою. Окрім логіки прикладної задачі, програма реалізує логіку зображення даних, інтерпретує відповідні операції, переводить їх у примітивні файлові операції, обробляє файли, що містять дані. Отже, дані залежать від програми.

Кожний файл існує для забезпечення інформаційних потреб програми. Дані у файлах, як правило, не пов'язуються між собою, тобто відсутня інтеграція даних, що призводить до дублювання даних у різних файлах і, можливо, до суперечності стану системи.

У файлових системах відсутня динаміка, оскільки прикладні програми розробляються для виконання наперед відомих запитів і завдань. Вони не можуть виконати нетипові запити, які зазвичай вимагають отримання відповіді в режимі реального часу.

Як правило, дані з одних і тих самих файлів не можна спільно використовувати, оскільки подібна практика може призвести до некоректного змінення даних.

Дані у файлах використовуються неефективно. Це є прямим наслідком того, що файли й дані, які в них містяться, створюються і обробляються для задоволення інформаційних потреб конкретних програм, а не для адекватного відображення усієї предметної області.

Саме для подолання обмежень, пов'язаних із використанням файлових систем, і були розроблені системи баз даних.

## Від видавництва

Свої зауваження, пропозиції та запитання надсилайте за адресою електронної пошти pg@bhv.kiev.ua, а також залишайте на сайті http://www.osvita.info. На цьому ж сайті можна отримати детальну інформацію про видання серії «Інформатика» для вищих навчальних закладів.

Інформацію про всі книжки Видавничої групи BHV ви знайдете на сайті http://www.bhv.kiev.ua.

# Розділ 1

# Системи баз даних. Основні поняття й архітектура

- Архітектура баз даних
- Системи керування базами даних
- Історія розвитку баз даних

У літературних джерелах існує чимало визначень поняття бази даних, які текстуально відрізняються одне від одного через те, що в них акцентується увага на особливостях процесів уведення та зберігання даних, але за своєю суттю ці визначення досить близькі. Розглянемо одне з них.

*База даних* (БД) – це сукупність взаємопов'язаних (звичайно складноструктурованих) даних, яку можна спільно використовувати та керування якою здійснюється централізовано.

Перелічимо основні властивості БД:

- допущення для даних такої мінімальної надлишковості, яка сприяє їхньому оптимальному використанню в кількох програмних застосуваннях;
- незалежність даних від програм;
- наявність засобів для підтримки цілісності бази даних та захисту від неавторизованого доступу.

*Система керування базами даних* (СКБД) це програмне забезпечення для ефективного, зручного і безпечного зберігання даних у БД, організації пошуку в ній та виведення даних на вимогу користувачів. Крім зазначених, СКБД має підтримувати ще низку функцій, про які йтиметься далі.

*Банк даних* – це система, що складається з БД, СКБД і прикладного програмного забезпечення, призначена для інформаційного обслуговування користувачів. Також уживається термін *система баз даних*. По суті, це певна прикладна система, яка використовує базу даних (відтак і СКБД, що підтримує цю БД) для вирішення конкретних завдань зберігання й обробки даних.

## 1.1. Основні поняття баз даних

Найближчим попередником, що мав неабиякий вплив на появу й подальше формування баз даних, були файлові системи. Саме тому розглядати можливості, достоїнства й характерні особливості баз даних найлегше, порівнюючи їх з файловими системами.

## 1.1.1. Порівняння баз даних із файловими системами

Розглянемо детально переваги, які надають користувачам бази даних і яких не мають файлові системи.

### Взаємопов'язаність даних

Локалізація певної групи даних програми загалом полегшує доступ до інших груп даних цього ж застосування. Внаслідок орієнтації БД на велику кількість сеансів використання виникає необхідність у підтримці різноманітних зв'язків між даними. Розглянемо приклад, який ілюструє вказану властивість у порівнянні з простою файловою системою.

Припустимо, що є мішок з різнокольоровими кульками. Завдання полягає в тому, аби знайти кульку певного кольору, витягаючи їх по одній із мішка. Якщо операції витягування кульок з мішка є повністю незалежними, то цей приклад є аналогом технології роботи в межах простої файлової системи з послідовним доступом

Припустимо, що потрібно знайти зелену кульку. Експериментатор наосліп вибирає й витягує з мішка по одній кульці, доки не побачить шуканий колір. При цьому попередня кулька ніяк не впливає на вибір наступної. Можливий інший спосіб пошуку. Експериментатор витягує з мішка червону кульку, до якої прив'язані кілька мотузок різних кольорів, зокрема й зелена. Тоді наступну кульку він витягає за зелену мотузку, і це буде шуканий об'єкт. Якщо ж червона кулька не матиме зеленої мотузки, то експериментатор може потягти за жовту мотузку і, витягши жовту кульку, вже від неї шукати зелену.

Для такої роботи необхідно, щоб експериментатор знав структуру зв'язків між кульками Якщо припустити, що кульки зв'язані між собою різнокольоровими мотузками, то у такому випадку пошук можна було б прискорити, використавши на черговому кроці мотузку потрібного кольору. В наведеному вище прикладі продемонстровано технологію роботи зі взаємопов'язаними даними, характерну для БД.

### Мінімальна надлишковість

Вимога мінімізації надлишковості полягає в тому, що в базі має зберігатися мінімальна кількість копій одних і тих самих даних. Це необхідно у зв'язку з орієнтацією БД на кілька застосувань. Надлишкові копії використовуються для підтримки зв'язків між даними.

Розглянемо такий приклад. У відділі кадрів певного підприємства зберігаються дані про співробітників. Користувачами цієї інформації є адміністрація, профспілкова організація та бухгалтерія підприємства. Адміністрацію цікавлять дані про кваліфікацію, професійний рівень і досвід роботи, профспілки використовують дані соціально-побутового характеру, а бухгалтерія обробляє дані, потрібні для нарахування заробітної плати, обчислення величини податку тощо. Хоча така інформація й різнорідна, та все ж має багато спільного — всім користувачам потрібно знати службовий номер, прізвище, ім'я, по батькові співробітника, його рік народження та дані про умови праці. Крім загальної інформації, бухгалтерії та проф-

спілкам також необхідні відомості про сімейний стан і склад сім'ї. Якщо для зберігання даних застосувати технологію файлової системи, можливі два протилежні варіанти:

а) незалежні один від одного файли, відсортовані згідно з потребами того чи іншого користувача, передбачають значну надлишковість даних;

б) всі дані розташовані в одному файлі, відсортованому так, як потрібно одному з користувачів (скажімо, адміністрації), надлишковість при цьому практично відсутня, але зручно працювати тільки певному користувачеві, інші ж опиняться у становищі «попелюшки без надії на черевичок».

Концепція БД полягає в створенні компромісу між вищеописаними варіантами – тобто надлишковість є, але вона мінімальна.

Зайва надлишковість має певні недоліки. По-перше, зберігання кількох копій призводить до додаткових витрат пам'яті. По-друге, доводиться виконувати численні операції оновлення для надлишкових копій. Крім того, оскільки різні копії даних можуть відповідати різним стадіям оновлення, то інформація, що зберігається в системі, на певний час може стати суперечливою.

Розглянемо приклад, який ілюструє подібну ситуацію. Співробітниця вийшла заміж і змінила прізвище, про що повідомила відділ кадрів. Кадровик зафіксував ці зміни у своєму файлі й запустив на виконання програму оновлення для файлової системи профспілкового комітету. Якщо в цей час трапиться збій системи, то в двох різних файлах одна й та сама людина значитиметься під різними прізвищами, до того ж матиме неоднаковий сімейний стан.

### Незалежність даних від програм

Незалежність даних від програм означає можливість зміни структури даних без зміни програм, що її використовують. Під це поняття підпадає й рівень самоінтерпретованості даних. Для файлової системи він найнижчий, оскільки дані файлів інтерпретуються виключно через прикладні програми, що їх використовують. БД, окрім самих даних, зберігає також їхні описи, а бази знань (найвищий рівень самоінтерпретованості) містять дані, описи даних та процедури їхньої обробки.

Проілюструємо цю ситуацію на дещо абстрактному прикладі. Припустимо, ви збираєтеся переглянути фільм у кінотеатрі, а для того щоб прибути на місце, плануєте скористатися послугами таксі. Поінформованість та досвід водія таксі відповідають рівню незалежності, тобто можливості діяти за тим чи іншим із наведених нижче варіантів. В одному випадку вам достатньо вказати лише назву фільму, і водій все зробить сам. В іншому – додати назву кінотеатру. Наступне зниження рівня – визначення адреси, а ще далі – вказівки дорогою («їхати прямо, звернути наліво» тощо).

За цією аналогією діє й користувач – що вище ступінь незалежності даних, то менше треба задавати (й знати) «процедурної» інформації щодо доступу до них. Зауважимо, що сучасні файлові системи мають певний (хоча й досить низький) рівень незалежності: для доступу до файла достатньо вказати його ім'я, інформація про доріжки та сектори не потрібна

### Цілісність та захист від неавторизованого доступу

Під цілісністю бази даних розуміють несуперечність (відповідність певним умовам) даних, що в ній зберігаються. Наприклад, для кадрових даних рік народження співробітника не може бути більшим за рік призначення на посаду або поточний рік. Для уникнення суперечливих ситуацій при модифікації бази даних співвідношення між даними контролюються спеціальними засобами підтримки цілісності БД. Умови, що накладаються на дані та яких дотримуються за будь-котрих оновлень, розробляються спеціальною службою – *адміністраторами баз даних* (АБД), а СКБД надають необхідні інструментальні засоби

Оскільки БД орієнтована на широке коло застосувань, то зрозумілим є існування засобів захисту від неавторизованого доступу (навмисного чи ненавмисного) користувачів до даних. Серед них – система паролів та ідентифікацій користувачів, а також розподіл даних і користувачів на групи з різними правами.

Крім наведених, БД мають ще низку переваг порівняно з файловими системами.

- **Інтегроване зберігання даних.** Сукупність даних розглядається як єдине ціле, незалежно від характеру використання та способів зберігання. Тобто база даних є інформаційною моделлю всієї предметної області.
- **Централізоване керування.** Передбачає адміністрування бази даних, що дає можливість забезпечити ефективне зберігання даних, усунення їхньої суперечності, підтримку єдиної політики у використанні даних та необхідного рівня їхньої безпеки, збалансовування суперечливих вимог різних користувачів.
- **Ефективне керування доступом до даних.** СКБД надає ефективні механізми керування доступом до даних, незважаючи на складність структур, у яких вони зберігаються.
- **Скорочення часу розробки програмних систем.** За наявності засобів створення і обробки даних, які надаються СКБД, усі зусилля розробників програмних систем концентруються на розв'язанні прикладних задач.
- **Спільне використання даних.** Передбачає одночасний доступ до інформації багатьох користувачів та ефективний розподіл ресурсів
- **Відновлення бази даних.** Жодна непередбачена ситуація – навіть перебій програмного чи апаратного забезпечення – не може вивести з ладу базу даних.
- **Дотримання стандартів.** Формати зберігання даних, технологія їхньої обробки, вхідні й вихідні форми можуть бути розроблені з урахуванням єдиних стандартів підприємства або інших нормативних документів.

## 1.1.2. Функції адміністратора бази даних

Функціонування складної структури на зразок СКБД потребує наявності певних служб, відповідальних за підтримання експлуатаційних характеристик баз даних. Чи не найважливішою з них є служба адміністратора бази даних, функції якої перелічено нижче.

- Проектування й розробка описів даних на різних рівнях та їхнє узгодження.
- Розробка структур зберігання і стратегій доступу до даних згідно з вимогами ефективного (чи економічного) зберігання, швидкої обробки та виведення даних за бажанням користувача.
- Реструктуризація та реорганізація БД у випадку зміни вимог до характеристик зберігання й обробки даних.
- Проектування та застосування механізмів захисту даних.
- Реєстрація користувачів (імен, паролів), визначення їхніх прав доступу та повноважень.
- Розробка й використання механізмів резервного копіювання даних та їхнє відновлення під час перебоїв.
- Настроювання роботи БД (підвищення продуктивності механізмів обробки даних, підтримання планованої надлишковості, забезпечення ефективності їхнього зберігання).
- Систематичне наглядання за використанням бази даних.

### 1.1.3. Основні вимоги до систем керування базами даних

До сучасних систем керування базами даних висуваються такі вимоги.

- **Простота і гнучкість створення застосувань.** СКБД має значно полегшувати процес створення і супроводу застосувань, що працюють з базами даних
- **Багаторазове й багатоаспектне використання даних.** Користувачі, які інакше розуміють одні й ті самі дані, мають отримати змогу використовувати їх по-різному.
- **Простота, легкість і гнучкість у використанні.** Користувачі мають легко дізнаватися, які дані є в їхньому розпорядженні, й отримувати простий доступ до даних (усі складнощі бере на себе СКБД).
- **Простота і гнучкість зміни, розширення й настроювання бази даних.** Будь-які зміни в БД, додавання до неї інформації чи її розширення повинні виконуватися незалежно від застосувань, що вже існують і використовують базу даних. До того ж ці операції мають виконуватися максимально просто й ефективно.
- **Ефективність і гнучкість зберігання й обробки даних.** СКБД має забезпечувати можливість простого й ефективного збільшення обсягів даних без порушення наявних способів їхнього використання, а також легкого виконання реорганізації і реструктуризації даних, СКБД повинна підтримувати заплановану надлишковість даних або надавати можливість її знижувати. Запити на отримання даних мають виконуватися зі швидкістю, потрібною для їхнього ефективного використання.

- **Низька вартість зберігання й використання даних.** СКБД мають забезпечувати нижчу вартість зберігання і використання даних, ніж будь-які інші інформаційні системи
- **Захист від несанкціонованого доступу, викривлення і знищення.** Система має забезпечувати необхідний рівень захисту даних. Інформація має бути захищена від перебоїв, впливу форс-мажорних ситуацій, а також від некомпетентного чи зловмисного звернення до даних осіб, які можуть оновити, викривити або ж видалити їх.
- **Підтримання необхідного рівня незалежності даних.** Як правило, від СКБД вимагається наявність механізмів підтримання логічної та фізичної незалежності даних.
- **Підтримання необхідного рівня цілісності даних.** Засоби опису і підтримання обмежень цілісності мають бути достатніми для опису правил, законів і обмежень, що діють у предметній області. Якщо ті чи інші обмеження не підтримуються безпосередньо, то СКБД має надавати можливості для їхнього опису і підтримання через відповідні засоби програмування.
- **Розвинені засоби адміністрування.** Передбачається, що СКБД надає всі необхідні інструменти для підтримання функцій адміністрування баз даних.

## 1.2. Архітектура баз даних

Термін «архітектура» може мати різні тлумачення. Наприклад, коли вказуються функціональні модулі системи й способи їхньої взаємодії, йдеться про *функціональну архітектуру*. Спосіб реалізації функцій системи, її компоненти та взаємозв'язки між ними фіксує *архітектура реалізації* системи. У цьому контексті можна також згадати *архітектуру технічних засобів* систем. Говорячи ж про термін «архітектура БД», ми матимемо на увазі архітектуру інформаційного забезпечення.

Архітектура БД уперше була специфікована дослідницькою групою ANSI/X3/SPARC Study Group on Data Base Management Systems (ANSI – Американський національний інститут стандартів, X3 – його комітет обчислювальної техніки й обробки інформації, SPARC – підкомітет ANSI/X3 з планування стандартів ANSI). Метою дослідницької групи було визначення галузей і технологій баз даних, в яких було б доречно проводити стандартизацію, а також вироблення рекомендацій для роботи в кожній із таких галузей. Група дійшла висновку, що, можливо, єдиним аспектом систем баз даних, який можна стандартизувати, є інтерфейси. Нею було докладено чимало зусиль для визначення загальної архітектури системи баз даних. Запропонована цією групою архітектура стала класичною та є актуальною й донині.

Основною ідеєю специфікації ANSI SPARC є виділення трьох архітектурних рівнів бази даних, а саме: *зовнішнього*, *концептуального* та *внутрішнього* (рис. 1.1).

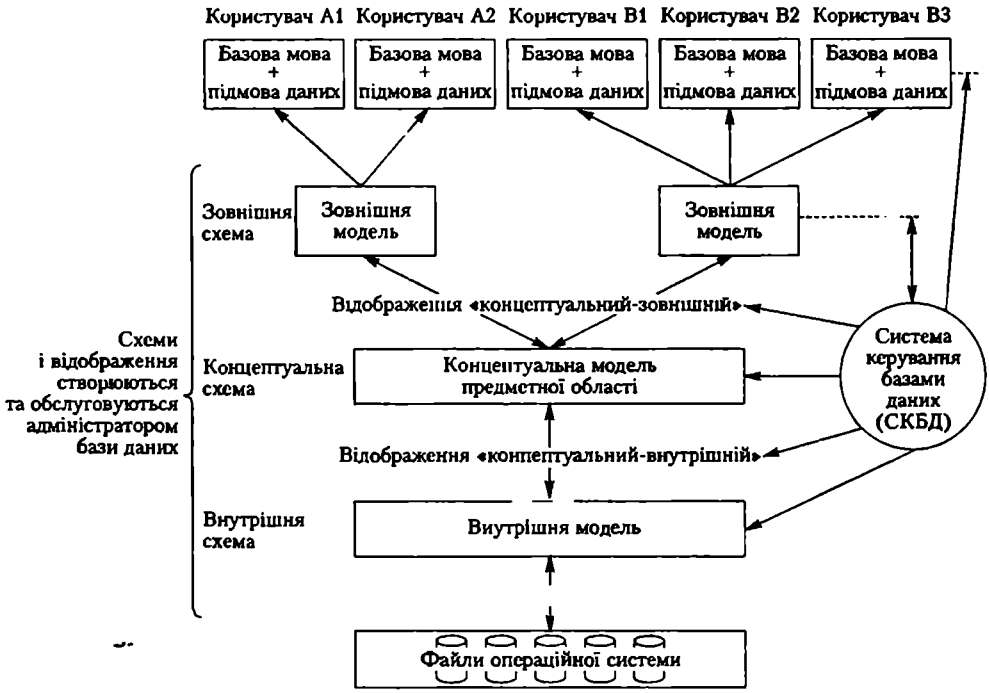



**Рис 1.1.** Трирівнева архітектура СКБД

## 1.2.1. Концептуальний рівень

На концептуальному рівні здійснюється інтегрований опис предметної області, для якої розробляється БД, незалежно від її сприйняття окремими користувачами та способів реалізації в комп'ютерній системі. Дамо означення основних понять, що використовуються на концептуальному рівні.

*Предметна область* (ПО) – частина реального світу, для якої здійснюється концептуальне моделювання.

*Концептуальна модель* ПО формальне зображення сукупності думок, які характеризують можливі стани ПО, а також переходи з одного стану в інший (включно з класифікацією наявних у ПО сутностей, чинних правил, законів, обмежень тощо)

*Концептуальне моделювання* ПО процес побудови концептуальної моделі ПО, яка б відображувала ПО з урахуванням вимог, висунутих до цього процесу

*Концептуальна схема* фіксація концептуальної моделі ПО засобами конкретних мов моделей даних. У СКБД концептуальна модель подається у вигляді концептуальної схеми.

Опишемо властивості концептуальної моделі (схеми) й характерні особливості концептуального моделювання.

- **Спільне та однозначне тлумачення предметної області всіма зацікавленими особами.** До розробки складної бази даних залучається великий колектив: експерти, системні аналітики, проектувальники, розробники, ті, хто займається впровадженням і супроводом. Усі вони повинні однозначно розуміти, чим є ПО, в чому зміст використаних понять, як вони взаємопов'язані між собою, які обмеження висуваються до моделі ПО тощо. Спільність понять має забезпечувати концептуальна модель.
- **Концептуальна схема відображує лише концептуально важливі аспекти ПО, виключаючи будь-які аспекти зовнішнього або внутрішнього відображення даних.** Ця модель не повинна відображувати конкретні потреби окремих користувачів або застосувань. Вона має фіксувати, чим є ПО в цілому, а не з точки зору інтересів або потреб користувачів. Для отримання цілісного уявлення про ПО її модель має інтегрувати думки, погляди та інтереси окремих користувачів, але саме інтегрувати, а не виражати їхні конкретні побажання.
- **Визначення допустимих меж еволюції бази даних.** У процесі експлуатації база даних може розвиватися, проте цей розвиток може відбуватися тільки в межах, допустимих для концептуальної схеми.
- **Відображення зовнішніх схем на внутрішню.** Саме через концептуальну схему зовнішні дані відображуються на внутрішні, й навпаки. У такий спосіб створюється єдина основа для опису даних і підтримки цих відображень.
- **Забезпечення незалежності даних.** Наявність відображень *концептуальний-зовнішній* і *концептуальний-внутрішній* дає змогу вирішувати проблему логічної та фізичної незалежності даних. Будь-які зміни в тій чи іншій зовнішній моделі не повинні спричиняти зміни в концептуальній або внутрішній моделях. У цьому випадку має змінитися тільки відповідне відображення «концептуальний-зовнішній». Аналогічно, будь-які зміни у внутрішній моделі не зачіпають концептуальну модель і моделі зовнішнього рівня, а тільки приводять до змін відображення «концептуальний-внутрішній».
- **Централізоване адміністрування.** Саме через концептуальну схему здійснюється адміністрування баз даних.
- **Стійкість.** Концептуальна схема не має підладжуватися до вимог тих чи інших користувачів (зовнішній рівень) або до вимог зберігання даних (внутрішній рівень). Будучи моделлю ПО, вона має змінюватися тільки тоді, коли входить у суперечність із нею.

Існує багато мов, які претендують на роль мов концептуального моделювання ПО. Найпопулярнішими і широковживаними є мови, що належать до класу так званих графічних мов, які оперують поняттями «сутність-атрибут-зв'язок» (Entity-Relationship language).

## 1.2.2. Зовнішній рівень

Через зовнішній рівень користувачі та застосування отримують доступ до бази даних. Мета зовнішнього рівня – надати користувачу/застосуванню лише ті дані,

які йому потрібні (а отже, до яких дозволений доступ) і в потрібному вигляді. Це індивідуальний рівень користувача, яким може бути кінцевий користувач, програміст чи застосування. Кожен з них має свою мову спілкування: для кінцевого користувача – це спеціальна мова запитів, для програміста – одна з мов програмування, розширена командами звернення до СКБД, для застосувань – це, як правило, певний стандартний інтерфейс звернення до бази даних через СКБД.

*Зовнішня модель* – це засоби зображення концептуальної моделі ПО з урахуванням інтересів конкретних користувачів або застосувань. Кожна зовнішня модель подається в СКБД у вигляді зовнішньої схеми.

Зовнішній рівень виконує такі функції.

- Забезпечує зображення даних зручним для людини або застосування способом. Ступінь незалежності зовнішнього зображення від концептуального рівня визначається потужністю засобів опису відображення «концептуальний-зовнішній».
- Сприяє вирішенню проблеми безпеки (захисту) даних. Надаючи користувачу лише ті дані, що його цікавлять, ми залишаємо поза межами його доступу решту даних.
- Сприяє вирішенню проблеми логічної незалежності даних. Це досягається завдяки відображенню «концептуальний-зовнішній», що встановлює відповідність між концептуальною схемою і конкретною зовнішньою схемою. Потужність його засобів визначає ступінь логічної незалежності застосувань від даних.

## 1.2.3. Внутрішній рівень

*Внутрішня модель* є відображенням концептуальної моделі ПО з урахуванням способів зберігання даних і методів доступу до них. Внутрішня модель відображується в СКБД у вигляді внутрішньої схеми. Внутрішня модель – це не модель фізичної пам'яті з характеристиками конкретних пристроїв зберігання даних (циліндри, доріжки тощо); вона описується у вигляді нескінченної абстрактної лінійної пам'яті, яка може структуруватися за допомогою інших абстрактних понять на зразок блоків, кластерів, індексів тощо.

Доступ до фізичної пам'яті надається за допомогою опису відображень внутрішньої моделі на фізичну пам'ять операційної системи.

Загалом внутрішній рівень виконує такі функції.

- Забезпечує настроювання бази даних для підвищення продуктивності обробки даних, опису й підтримки планованої надлишковості.
- Дає змогу описувати й підтримувати структури зберігання та методи доступу.
- Сприяє вирішенню проблеми фізичної незалежності даних: зміни у внутрішній схемі не повинні призводити до змін у зовнішній схемі.
- Сприяє вирішенню проблеми безпеки (захисту) даних.
- Вирішує проблему відображення даних на структури ОС, у яких дані зберігаються (до таких структур належать зокрема файли).

### 1.2.4. Відображення

Відображення зовнішнього рівня на концептуальний і концептуального рівня на внутрішній показані на рис. 1.1. Відображення «зовнішній-концептуальний» визначає відповідність між зовнішнім рівнем і концептуальним. Незалежність зовнішньої схеми, відтак і ступінь логічної незалежності даних, обумовлюються потужністю засобів опису цих відображень. Тобто можна описувати або змінювати зовнішню схему тільки в тих межах, які допускає це відображення.

Подібне можна сказати і про відображення «концептуальний-внутрішній», яке встановлює відповідність між концептуальною і внутрішньою моделями. Потужність його засобів визначає ступінь фізичної незалежності застосувань від даних Будь-які зміни у фізичній структурі не повинні призводити до змін у концептуальній моделі – змінюється лише відображення «концептуальний-внутрішній».

За створення і ведення схем усіх рівнів (концептуальної, зовнішньої і внутрішньої), а також відображень відповідає адміністратор бази даних. Окрім того, він виконує багато інших функцій, про які йшлося в підрозділі 1.1.2.

## 1.3. Функції систем керування базами даних

Система керування базами даних надає засоби для здійснення таких операцій

- **Визначення даних (зовнішні, внутрішня і концептуальна схеми) та відображень у певний формальний спосіб і реалізація цих визначень у вигляді відповідних об'єктів.** СКБД має обробляти вказівки мови опису даних (МОД) для опису концептуальної схеми, підмови опису даних (ПМОД) для опису зовнішніх схем і мови опису збережених даних (МОЗД) для опису внутрішньої схеми.
- **Маніпулювання даними.** СКБД має сприймати, інтерпретувати й обробляти запити користувачів на вибирання, оновлення і видалення наявних даних або на додавання нових даних до бази. Ці запити мають бути сформульовані мовою запитів (МЗ) або мовою маніпулювання даними (ММД).

Архітектура ANSI/SPARC не підтримується в повному обсязі жодною з сучасних СКБД, особливо, якщо це стосується концептуального рівня. У більшості наявних систем концептуальна схема насправді є простим об'єднанням усіх окремих зовнішніх схем, доповненим засобами гарантування безпеки даних і правилами забезпечення цілісності. Зовнішні схеми підтримуються за допомогою так званих віртуальних таблиць. Щодо внутрішнього рівня, то він, як правило, підтримується різноманітними механізмами опису структур зберігання даних і методів доступу до них.

У сучасних СКБД зазвичай підтримується єдина інтегрована мова, що містить усі необхідні засоби для роботи з БД, починаючи від її створення, і забезпечує базовий, призначений для користувача, інтерфейс баз даних. Стандартною мовою найпоширеніших на сьогодні реляційних СКБД є мова SQL (Structured Query Language – мова структурованих запитів).

## 1.4. Історія розвитку баз даних

Історію розвитку баз даних можна поділити на чотири періоди.

### Період становлення — 60-ті роки

Ідея застосування файлів для зберігання даних спільного використання виникла наприкінці 50-х років. Проте саме у 60-ті роки з'явився сам термін «база даних» і було створено кілька систем баз даних. Цього ж десятиліття з'явилася класифікація БД за структурами даних, що в них використовуються, — почали вирізняти системи баз даних з ієрархічною й мережною структурами.

*Ієрархічні системи*, в яких базова структура даних мала деревоподібний вигляд, досягли найвищої ефективності функціонування, але виразові можливості цих систем лишилися відносно низькими. Системам зі структурами даних *типу мережі*, навпаки, вдалося надати значно кращих виразових можливостей, але вони програють в ефективності функціонування, тобто для плідної експлуатації таких систем від користувача вимагався (і вимагається) значно вищий рівень кваліфікації. Останніми з'явилися СКБД *реляційного типу*, які характеризувалися найпростішою структурою даних (таблиця, або так званий плоский файл) з одного боку та високим рівнем мов маніпулювання даними — з іншого. Це зробило їхні виразові можливості максимально потужними, але знизило ефективність функціонування.

Використання в мейнфреймах магнітних дисків замість магнітних стрічок сприяло створенню в середині 60-х років перших систем керування базами даних, з яких найрозвиненішою виявилася система IMS фірми IBM, що підтримувала ієрархічну структуру даних. Головний ідеолог мережного підходу Ч. Бахман 1963 року розробив першу промислову систему баз даних IDS, орієнтовану на мережну організацію даних.

Асоціація CODASYL, що створила мову програмування COBOL, у 1967 році організувала робочу групу з питань баз даних, яка узагальнила мовні специфікації систем БД. (Відповідні звіти були опубліковані в 1969 та 1971 роках, які за найменуванням робочої групи — Data Base Task Group — отримали назви DBTG 69 і DBTG 71.) Взявши за основу реалізовану в системі IDS мережну структуру даних та методи навігації нею, група DBTG істотно розвинула й обґрунтувала мережну модель.

Одним з типових представників систем, що відповідали пропозиції CODASYL DBTG, була система Integrated Database Management System (IDMS) компанії Cullinet Software, призначена для використання на мейнфреймах IBM.

У цей же період чітко окреслилися два підходи до проблеми замкненості систем баз даних.

- Створення *систем замкненого типу*, що містять у своєму складі не традиційні мови програмування, а непроцедурні мови запитів. Головна мета розробників полягала в тому, аби з цими системами міг працювати пересічний оператор, а не лише фахівець з програмування. До таких систем належали TDMS і UL/1.
- Створення *систем з базовою мовою*. Такі системи, окрім власне мов маніпулювання даними, надають мовні й інструментальні засоби розробки прикладних

програм з використанням наявних мов програмування. Цього принципу дотримувалася група DBTG.

Наприкінці періоду становлення виник термін *інформаційно-керуюча система* (ІКС). У той час під цим терміном розуміли орієнтовану на пошук даних систему баз даних, що забезпечувала можливість роботи з віддаленого термінала.

### Період розвитку — 70-ті роки

Концепція баз даних отримала широке розповсюдження завдяки поліпшенню характеристик апаратного забезпечення комп'ютерів. Успішно впроваджувалися системи, орієнтовані на підтримку ієрархічної та мережної структур даних

Тривала далі робота групи CODASYL DBTG. Була специфікована система мов для баз даних CODASYL, яка складалася з кількох груп мовних специфікацій

У 1975 році з'явився звіт робочої групи ANSI/X3/SPARC, у якому розглядалося питання про стандартизацію баз даних і СКБД, а також про те, що саме може підлягати стандартизації. Група вирішила, що ця проблема стосується лише інтерфейсів, які можуть існувати між різними компонентами СКБД, самі ж програмні компоненти стандартизації не підлягають. Учені скерували свої подальші зусилля на виявлення таких інтерфейсів і, врешті-решт, запропонували концепцію трирівневої архітектури баз даних, яка стала класичною і дотепер не втратила актуальності.

Період розвитку більш відомий завдяки створенню реляційної моделі даних, яку 1970 року запропонував співробітник інституту фірми IBM у Сан-Хосе Е. Ф. Кодд Протягом десятиліття всебічно досліджувались теоретичні й прикладні питання цієї моделі, розроблялись експериментальні реляційні СКБД. Плідна праця дала можливість створити формальну теорію баз даних, яка до цього мала описовий характер. Протягом кількох років багато провідних фірм проводили експериментальні дослідження, спрямовані на винайдення прототипів реляційних СКБД, підвищення їхньої ефективності та функціональності. І нарешті наприкінці десятиліття були створені перші промислові реляційні СКБД.

### Період зрілості — 80-ті роки

Реляційна модель отримала повне теоретичне обґрунтування. Було розроблено великі реляційні СКБД · Oracle, Informix та інші. Промислові реляційні системи почали використовуватися в усіх сферах людської діяльності. У такий спосіб реляційні системи практично витіснили зі світового ринку попередні СКБД ієрархічного та мережного типів.

У цей період проводились також теоретичні й експериментальні дослідження в галузі баз знань та були створені численні експертні системи. У більшості випадків бази знань розроблялися на основі реляційних СКБД.

Подальший розвиток реляційних СКБД відбувався за такими напрямами.

- **Зручність застосування.** З поширенням персональних комп'ютерів постало принципове питання щодо зручності використання програм, яке також стосувалося СКБД. У зв'язку з цим у СКБД почали інтенсивно застосовуватися засоби інтерфейсу користувача.

- **Багатоплановість.** Зростання попиту на бази даних у нетрадиційних галузях їхнього застосування (системи автоматизації проектування, видавнича справа тощо) привело до виникнення потреби у зберіганні й обробці в базах даних зображень, звуків та повнотекстової, а не лише символьної інформації.

### Постреляційний період — з початку 90-х років

Розпочалися інтенсивні дослідження з питань дедуктивних та об'єктно-орієнтованих баз даних, які сприяли створенню дослідницьких прототипів таких систем Так, 1991 року з'явилася ODMG (Object Data Management Group) - група з питань керування об'єктними базами даних, яка посіла особливе місце в галузі стандартизації об'єктно-орієнтованих СКБД. В 1993 році група видала свій перший стандарт ODMG-93, а в 1995 був опублікований його вдосконалений варіант

З розвитком інтернет-технологій розробники спрямували свої зусилля на впровадження баз даних в Інтернет. Щодо приєднання СКБД разом з базами даних до «всесвітньої павутини», окреслилися різні підходи – починаючи від найпростіших «публікацій» баз даних в Інтернеті й завершуючи розробкою веб-серверів, які надають користувачам Інтернету весь спектр послуг, що стосуються роботи з базами даних на сервері. Інтенсивного розвитку набули дослідження й розробки, присвячені маніпулюванню структурами даних в Інтернеті.

Слід зазначити, що, незважаючи на розвиток більш передових технологій баз даних, реляційні БД і досі залишаються домінуючими як у настільних системах, так і на промисловому рівні.

## Контрольні запитання та завдання

1. Якими є основні функції файлової системи?
2. Що таке база даних? Чим вона відрізняється від файлової системи?
3. Коли потрібно використовувати систему керування базами даних замість звичайних засобів ведення файлів, що входять до складу операційної системи?
4. Що таке незалежність даних від програм?
5. Що таке цілісність бази даних?
6. Які функції виконує адміністратор бази даних?
7. Що таке захист даних?
8. Які основні вимоги висуваються до системи баз даних?

# Розділ 2
# Моделі даних

- Поняття та класифікація моделей даних
- Ієрархічна модель даних
- Мережна модель даних
- Історія реляційної моделі

## 2.1. Поняття про моделювання даних

### Дані та їхня семантика

Слово «дані» походить від латинського «datum» — факт, проте дані не завжди відповідають конкретним чи навіть реальним фактам. Іноді вони неточні або описують те, чого насправді не існує. *Даними* ми вважатимемо опис будь-якого явища (чи ідеї), що викликає зацікавленість через певні потреби. З даними нерозривно пов'язана їхня інтерпретація (або семантика), тобто той зміст, який їм приписується.

Дані описуються тією чи іншою мовою і фіксуються на певному носії (скажімо, папері). Зазвичай дані (факти) та їхня семантична інтерпретація фіксуються спільно. Проте в деяких випадках дані й інтерпретація розділяються. Так, у зведеній статистичній таблиці зверху, в її шапці, міститься інформація стосовно того, чим є дані (заголовок таблиці, описова інформація, назви стовпців тощо), а нижче розташовуються власне рядки чисел.

Застосування комп'ютерів для введення й обробки даних сприяло ще більшому розділенню даних та їхньої інтерпретації. Оскільки комп'ютери оперують даними як такими, велика частина інтерпретуючої інформації взагалі явно не фіксується. Програма одержує певні вхідні дані, обробляє їх і видає результат у вигляді сукупності вихідних даних.

З розвитком обчислювальної техніки з'явилася можливість фіксувати за допомогою комп'ютерів семантику даних, тобто закладати інтерпретацію в програми, що оперують даними. Проте за умов спільного використання даних різноманітними прикладними програмами цей підхід можна застосовувати лише частково. Інакше процес написання програм, які містять схожі, хоча й не ідентичні механізми інтерпретації, стає неефективним. У подібній ситуації доцільно зв'язувати дані з механізмами інтерпретації, що має забезпечувати уніфікованість подання семантичної інформації. У результаті змінюється роль даних: їх уже не можна розглядати лише як сукупність бітів, вони отримують певне семантичне навантаження

Засоби інтерпретації мають бути гнучкими, аби разом з незмінними параметрами даних відображувати й аспекти їхньої еволюції. Цього досягають двома спо-

собами. По-перше, можна відтворювати різні погляди на одні й ті самі дані. Наприклад, розглядати певну особу з точки зору кадрової системи як службовця, з погляду виробничих інтересів – як виконавця робіт, а з боку медичного обслуговування – як пацієнта. По-друге, різні дані можна подавати одноманітно. Так, адміністратори, клерки, агенти, секретарі незалежно від роду своєї діяльності можуть розглядатися в кадровій системі як службовці.

### Моделювання даних

Існує багато типів моделей фізичні, математичні, економічні тощо, які відображують різні аспекти реального світу. Модель даних відображує уявлення про реальний світ. Проте важливо, аби обсяг знань і семантика даних, відтворені в моделі, були адекватні способу використання даних. Вважатимемо, що *модель даних* це сукупність структури даних, операцій над ними (операції маніпулювання даними) та обмежень цілісності. Іншими словами, модель визначає, в який спосіб відбувається об'єднання даних у структури різної складності, які існують обмеження на значення даних і як здійснюється оперування цими даними

Основою для будь-якої структури даних є відображення елементарної одиниці даних у вигляді такої трійки: <об'єкт, властивість об'єкта, значення властивості>. Сукупність взаємопов'язаних між собою елементарних одиниць даних може відображуватися різноманітними способами, що приводить до формування різних структур, а відтак - різних моделей даних. Моделі даних поділяються на два класи: сильно та слабко типізовані

У *сильно типізованих моделях* усі дані мають належати до певної категорії, або типу. Якщо дані не підпадають під жодну з категорій, їх потрібно типізувати штучно. Деякі моделі будуються у такий спосіб, що категорії визначаються наперед і не можуть змінюватися динамічно. У цьому випадку модельований світ начебто вміщується в гамівну сорочку. Наприклад, категорія «службовець» – строго фіксована, й усі її об'єкти повинні мати однакові властивості та структуру

Сильно типізовані моделі мають значні переваги, бо дають змогу побудувати абстракції властивостей даних і дослідити їх у термінах категорій. Більшість моделей, що використовуються в автоматизованих системах, зокрема й базах даних, належать до сильно типізованих. (Це стосується й усіх моделей даних, що розглядаються далі.)

Для *слабко типізованих моделей* належність даних до тієї чи іншої категорії не має жодного значення. Категорії використовуються настільки, наскільки це доцільно в кожному конкретному випадку. Окремі дані можуть існувати як незалежно, так і у зв'язку з іншими. Інформація про категорії (якщо вони використовуються) розглядається як додаткова.

На відміну від сильно типізованих моделей, слабко типізовані забезпечують інтеграцію даних і категорій. Найкращі можливості такої інтеграції надаються численням предикатів, яке у багатьох моделях даних використовується для зображення знань, що не підтримуються базовими засобами моделювання.

Відомі три класичні моделі баз даних: ієрархічна, мережна та реляційна. (У базах знань використовуються інші моделі, але про них мова йтиме в розділі 15.)

## 2.2. Ієрархічна модель даних

Ієрархічна модель даних уперше була задіяна в системі IMS (Information Management System – інформаційна керуюча система) у межах проекту висадки на Місяць. У першій ієрархічній системі були повністю реалізовані функції СКБД, а саме: мови визначення та маніпулювання даними, опис і підтримка обмежень цілісності, паралелізм, відновлення, а також механізми ефективної обробки запитів. Варто сказати, що IMS і досі використовується на мейнфреймах.

Згодом було розроблено ще декілька ієрархічних СКБД, і кожна з них привносила в модель свою специфіку, зумовлену способом реалізації системи. Далі будуть розглянуті найбільш загальні та принципові аспекти моделі.

### 2.2.1. Ієрархічна структура даних

Ієрархічна структура даних визначається ієрархічною впорядкованістю своїх компонентів (або вузлів), тобто кожен вузол має не більше одного «батька» – старшого за ієрархією вузла.

Структура складається зі *схем елементів даних* (описова інформація) та їхніх *екземплярів*. Інакше кажучи, схема задає логічну структуру (або тип) елементу даних, а екземпляр – його значення.

Елементарним значенням структури є поіменоване *поле даних*, а його екземпляр – це елементарне значення.

*Схема сегмента* (яку називатимемо також просто *сегментом*) – це поіменована впорядкована сукупність імен полів. Сегмент є одиницею доступу до даних ієрархічної структури під час взаємодії зовнішньої та оперативної пам'яті. *Екземпляр сегмента* - впорядкована сукупність значень полів.

*Ієрархічна схема даних* – це ієрархічно впорядкована сукупність сегментів, що має певні властивості:

- на найвищому рівні ієрархії розташований єдиний сегмент, що називається *кореневим*;
- кожен інший сегмент, окрім кореневого, зв'язаний з одним і тільки одним сегментом вищого рівня, який є для цього сегмента *батьківським* (початковим);
- сегмент може бути зв'язаний з одним або кількома сегментами нижчого рівня, які називаються *дочірніми* (породженими);
- сегменти, що підпорядковані одному батьківському сегменту, називаються *близнюками*;
- сегменти, що не мають дочірніх, вважаються *листковими*, або їх ще називають *листками*.

*Ієрархічний шлях* (або просто шлях) - це послідовність сегментів, починаючи з кореневого, де кожний попередній є «батьком» наступного. *Рівень* сегмента визначається як кількість сегментів, що містяться на шляху, який веде від кореня до даного сегмента.

Для ієрархічної схеми використовується така графічна нотація.

- Кожний сегмент зображується у вигляді пойменованого прямокутника. Усередині прямокутника записуються імена полів
- Ієрархічний зв'язок між сегментами позначається лініями зі стрілками, що проведені від батьківського сегмента до дочірнього. Батьківські сегменти, як правило, розміщують над дочірніми.

Приклад графічного зображення простої ієрархічної схеми даних наведений на рис. 2.1, *а*. Якщо немає необхідності уточнювати сегменти полями, що зазвичай робиться під час загального аналізу ієрархічної структури предметної області, то в прямокутнику сегмента зазначається його ім'я (рис. 2.1, *б*).

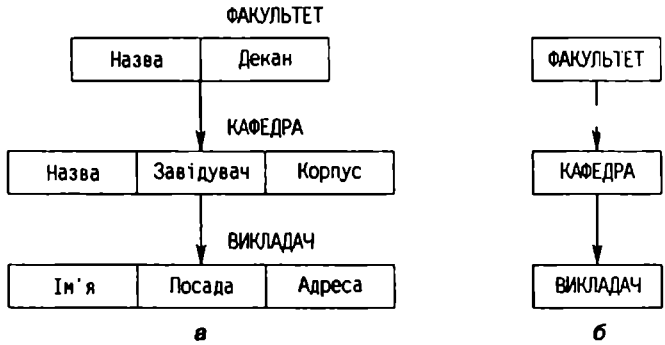



**Рис. 2.1.** Графічне зображення схеми ієрархічної структури даних: з уточненнями (*а*); без уточнень (*б*)

*Екземпляр ієрархічної схеми даних* складається з одного екземпляра кореневого сегмента і, можливо, кількох екземплярів дочірніх сегментів для кожного екземпляра батьківського сегмента. Припускається існування таких зв'язків між екземплярами сегментів:

- кожний екземпляр будь-якого сегмента підпорядкований одному екземпляру батьківського сегмента;
- екземпляр будь-якого сегмента (окрім кореневого) не може існувати без відповідного екземпляра батьківського сегмента;
- кожний екземпляр сегмента зв'язаний (підпорядковує собі) з усіма екземплярами дочірніх сегментів;
- екземпляри одного сегмента, зв'язані з одним екземпляром батьківського сегмента, можуть бути зв'язані між собою в ланцюжок, що дає змогу виконувати їхнє послідовне перебирання у межах усіх сегментів, породжених з одного початкового.

У такий спосіб ієрархічна впорядкованість сегментів створює зв'язок «один-до-багатьох» між екземплярами батьківського і дочірнього сегментів.

Приклад екземплярів ієрархічної схеми наведений на рис. 2.2.

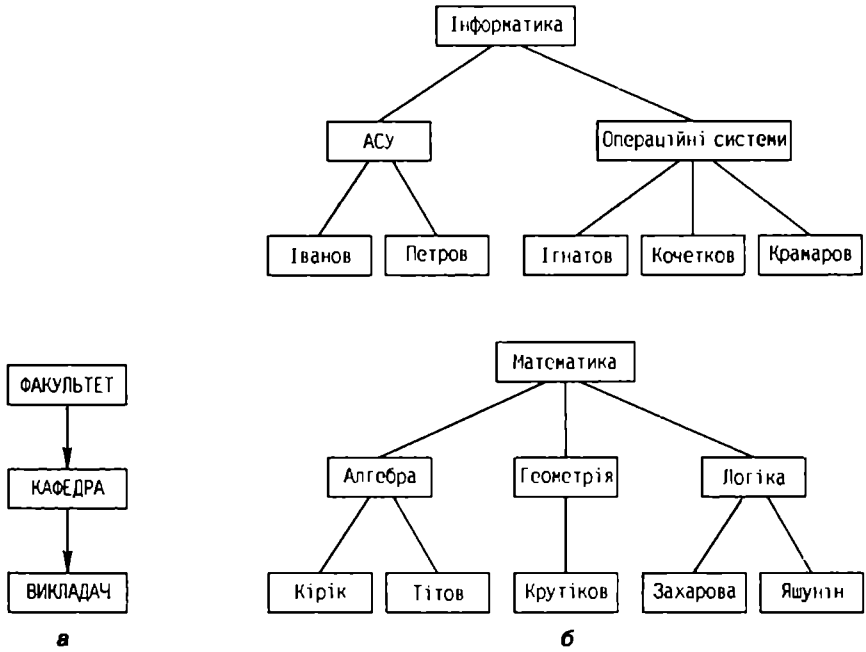



**Рис. 2.2.** Схема (*а*) та її екземпляри (*б*) в ієрархічній структурі даних

Ієрархічна схема інколи має розгалуження, як це показано на рис. 2.3. У подібному випадку на рівні схеми батьківський сегмент може зв'язуватися з кількома дочірніми сегментами


**Рис. 2.3.** Ієрархічна схема з розгалуженням

*Ієрархічна структура даних* — це сукупність ієрархічної схеми даних та всіх можливих екземплярів цієї схеми. Сукупність ієрархічних структур даних називається *ієрархічною базою даних.*

## 2.2.2. Операції над ієрархічною структурою

Операції, що виконуються над ієрархічною структурою, поділяють на дві групи. операції пошуку (чи вибирання) та операції оновлення даних (або маніпулювання ними). У цьому підрозділі операції, які формують семантичну основу мови маніпулювання даними тієї чи іншої СКБД, описуються на концептуальному рівні без уточнення синтаксису та інших другорядних деталей.

Зазначимо кілька базових принципів, характерних для мов маніпулювання даними ієрархічних систем.

- Об'єктом маніпулювання є екземпляр сегмента.
- У результаті успішного виконання пошукових операцій визначається *поточний екземпляр сегмента*, який може відігравати роль стартової позиції для операцій маніпулювання; спочатку поточним є кореневий екземпляр сегмента.
- Пошук необхідного сегмента відбувається за «навігаційним» принципом: необхідно прокласти шлях від кореневого екземпляра сегмента (або ж поточного) до шуканого, перевіряючи умови, накладені на значення полів сегментів, які розташовані на шляху, що будується. Операція в ієрархічній моделі визначається шляхом програмування «навігації» структурою даних.

В описаних далі операціях було використано мову DL/1 системи IMS. синтаксис якої тут значно спрощений.

### Вибирання даних

Необхідний екземпляр сегмента вибирається в ієрархічній структурі за допомогою команди «навігації» структурою. Екземпляр сегмента є одиницею навігації. Для визначення необхідного сегмента навігації може накладатися умова на значення полів сегмента. Відносно поточного сегмента можна переміщуватися ієрархічною структурою вверх, вниз і вбік.

Навігацію ієрархічною структурою можна здійснювати з метою її впорядкування, а потім переміщуватися нею згідно зі встановленим порядком. Порядок переміщення встановлюється, починаючи з кореня або будь-якого первинного сегмента. Далі рухатись можна зверху донизу і зліва направо.

Доступ до даних здійснюється за допомогою команди GET, різновиди якої розглядаються нижче.

- GET UNIQUE — *пряме вибирання*. Вибирання першого сегмента вказаного типу, що відповідає умові пошуку. За допомогою цієї операції фіксується початкова позиція для подальшої навігації ієрархією.
- GET NEXT — *послідовне вибирання*. Вибирання сегмента, розташованого відразу за поточним, згідно з наявним порядком розташування сегментів у базі даних. Задаючи різні значення параметрів команди, можна отримати
  - сегмент, який розташований за поточним (при цьому тип сегмента може виявитися довільним);
  - наступний сегмент заданого типу;
  - наступний сегмент заданого типу, що відповідає заданій умові.
- GET NEXT WITHIN PARENT — послідовне вибирання в межах поточного батьківського сегмента.

Розглянемо декілька найпростіших прикладів вибирання даних. Користуватимемось ієрархічною схемою, що зображена на рис. 2.1.

```
ФАКУЛЬТЕТ(Назва, Декан)
КАФЕДРА(Назва, Завідувач, Корпус)
ВИКЛАДАЧ(Ім'я, Посада, Адреса)
```

Команда GET UNIQUE має такий синтаксис:

```
GET UNIQUE <тип сегмента>
[WHERE <умова>]
```

За цією командою знаходиться перший сегмент вказаного типу, що відповідає вказаній умові. Тут <умова> — це предикат, у якому можуть зазначатися будь-які типи сегментів, що є батьківськими до типу <тип сегмента>, та сам вказаний тип сегмента. Фраза WHERE може бути відсутня, тоді команда знаходить перший екземпляр сегмента вказаного типу.

Розглянемо кілька прикладів.

Перший екземпляр кафедри в корпусі 5 знаходиться у такий спосіб:

```
GET UNIQUE КАФЕДРА
WHERE КАФЕДРА.Корпус = 5;
```

Знайдений сегмент стає поточним. Для знаходження професора кафедри АСУ факультету інформатики слід записати:

```
GET UNIQUE ВИКЛАДАЧ
WHERE ФАКУЛЬТЕТ.Назва = "інформатики" AND
 КАФЕДРА.Назва = "АСУ" AND
 ВИКЛАДАЧ.Посада = "професор";
```

Пошук першого професора здійснюється згідно з прийнятою впорядкованістю екземплярів сегментів.

Якщо необхідно знайти кілька сегментів, що відповідають умові пошуку, слід скористатися командою GET NEXT (вона має синтаксис, аналогічний GET UNIQUE). За допомогою цієї команди знаходиться наступний сегмент, що відповідає заданій умові пошуку. Наприклад, щоб послідовно знайти всіх професорів кафедри математики факультету інформатики, слід виконати таку команду:

```
GET UNIQUE ВИКЛАДАЧ
WHERE ФАКУЛЬТЕТ.Назва = "інформатики" AND
 КАФЕДРА.Назва = "математики" AND
 ВИКЛАДАЧ.Посада = "професор";
NT: GET NEXT ВИКЛАДАЧ
 WHERE ФАКУЛЬТЕТ.Назва = "інформатики" AND
 КАФЕДРА.Назва = "математики" AND
 ВИКЛАДАЧ.Посада = "професор";
GOTO NT
```

Тут спочатку фіксується перший сегмент типу «викладач кафедри математики факультету інформатики», а потім за допомогою команд GET NEXT, що виконуються в циклі, здійснюється послідовне перебирання сегментів.

Наведені вище різновиди команди GET здійснюють пошук конкретного екземпляра сегмента в межах усієї ієрархічної структури. Коли необхідно віднайти дані в межах піддерева, пошук його кореневого сегмента здійснюється за допомогою команд GET UNIQUE та/або GET NEXT. Знайдений у такий спосіб екземпляр сегмента стає поточним батьківським сегментом для того піддерева, що нас цікавить. Подальший пошук у піддереві виконується за допомогою команди GET NEXT WITHIN PARENT, яка має синтаксис, аналогічний розглянутим раніше різновидам команди

GET. Наприклад, для знаходження всіх професорів факультету інформатики можна виконати наведені нижче команди.

```
GET UNIQUE ФАКУЛЬТЕТ
WHERE ФАКУЛЬТЕТ.Назва = "інформатики";
...NT: GET NEXT WITHIN PARENT ВИКЛАДАЧ
WHERE ВИКЛАДАЧ.Посада = "професор";
...
GOTO NT
```

Отже, операція в ієрархічній моделі програмує «навігацію» структурою даних.

### Маніпулювання даними

Маніпулювання даними передбачає додавання, заміну та видалення екземплярів сегментів специфікованих типів.

Додати новий екземпляр сегмента можна лише за умови, що в ієрархічній структурі вже є екземпляр його батьківського сегмента. У команді додавання INSERT задається повний ієрархічний шлях до цього батьківського сегмента, тип сегмента, що підлягає додаванню, та дані екземпляра сегмента, який додається Наведемо приклад операції додавання нового екземпляра сегмента КАФЕДРА:

```
КАФЕДРА.Назва = "САПР";
КАФЕДРА.Завідуючий = "Іванов";
КАФЕДРА.Корпус = 3;
INSERT КАФЕДРА
WHERE ФАКУЛЬТЕТ.Назва = "інформатики";
```

Додати сегмент ВИКЛАДАЧ можна так:

```
ВИКЛАДАЧ.Ім'я = "Петров".
ВИКЛАДАЧ.Посада = "асистент";
ВИКЛАДАЧ.Адреса = "Проспект Миру 13/17";
INSERT ВИКЛАДАЧ
WHERE ФАКУЛЬТЕТ.Назва = "інформатики" AND КАФЕДРА.Назва = "САПР";
```

Щоб замінити сегмент, його слід спочатку вибрати за допомогою команди GET HOLD, яка вказує, що шуканий сегмент буде модифікований, потім змінити у знайденого сегмента певні поля й застосувати команду REPLACE. Наприклад, у такий спосіб можна змінити сегмент «адреса викладача Петрова з кафедри САПР»:

```
GET HOLD UNIQUE ВИКЛАДАЧ
WHERE ФАКУЛЬТЕТ.Назва = "інформатики" AND
 КАФЕДРА.Назва = "САПР" AND
 ВИКЛАДАЧ.Ім'я = "Петров";
 ВИКЛАДАЧ.Адреса = "вул. Теремківська 1/19";
REPLACE;
```

Видалення сегмента здійснюється за тією ж схемою, що й заміна: потрібно віднайти сегмент за допомогою команди GET HOLD, потім видалити його командою DELETE. При цьому прикладна програма може проаналізувати значення полів знайденого сегмента і прийняти остаточне рішення щодо його подальшого існування.

Наприклад, видалення інформації про викладача Петрова кафедри САПР факультету інформатики виконується так:

```
GET HOLD UNIQUE ВИКЛАДАЧ
 WHERE ФАКУЛЬТЕТ.Назва = "інформатики" AND
 КАФЕДРА.Назва = "САПР" AND
 ВИКЛАДАЧ.Ім'я = "Петров";
DELETE;
```

Видалення екземпляра сегмента приводить до видалення всіх екземплярів його дочірніх сегментів, оскільки в ієрархічній структурі жодний екземпляр сегмента не може існувати без свого «батька». Отже, якщо ви видаляєте екземпляр сегмента КАФЕДРА, то автоматично видаляються всі екземпляри сегментів, що відповідають викладачам даної кафедри.

## 2.2.3. Переваги та недоліки ієрархічної моделі

До переваг ієрархічної моделі належать:

- розвинені низькорівневі засоби керування даними в зовнішній пам'яті;
- можливість побудови ефективних прикладних систем;
- економне використання пам'яті.

Зазначимо, що ієрархічна модель має також певні недоліки, які описані нижче.

- **Асиметрія пошуку за симетричними запитами.** Програми пошуку викладачів за кафедрами та кафедр за викладачами принципово відрізняються: в першому випадку структура ієрархії збігається зі структурою пошуку, а в другому — ні.
- **Залежність між пошуком та відповідністю ієрархічної структури наявним зв'язкам у предметній області.** Якщо ієрархічна структура відповідає структурі зв'язків ПО, то записувати пошукові вирази набагато легше.
- **Низький рівень мови запитів і маніпулювання даними.** Оскільки маємо справу з мовою програмування процедурного типу, результатом однієї пошукової операції є один екземпляр сегмента даних, або ж один екземпляр ієрархічного шляху до заданого екземпляра сегмента. Це вказує на низьку селективну потужність мови.
- **Аномалії додавання, видалення та оновлення даних.** Не можна здійснити операцію додавання сегмента ВИКЛАДАЧ без зазначення сегмента КАФЕДРА; не можна видалити сегмент КАФЕДРА, не видаляючи сегментів ВИКЛАДАЧ; після оновлення даних один і той самий екземпляр сутності зображується у вигляді багатьох екземплярів об'єктів бази даних.
- **Дублювання даних.** Якщо об'єкти предметної області мають зв'язки типу «один-до-одного» або «один-до-багатьох», то ієрархічна структура дає змогу зображувати дані без дублювання; проте, якщо є зв'язки типу «багато-до-багатьох», то дублювання даних під час відображення в ієрархічній моделі неминуче.

Крім згаданих основних недоліків ієрархічної моделі слід зазначити також складність реалізації гнучких механізмів захисту даних, цілісності та несуперечності й «дружніх» інтерфейсів користувача.

Основні недоліки ієрархічної модел пов'язані з тим, що не всі предметні області мають чітко виражену ієрархічну структуру Наведений на рис. 2.2 приклад добре узгоджується з ієрархічною структурою. Проте, якщо ми розглянемо предметну область із сутностями «викладач», «дисципліна», «лекція» в ситуації, коли один викладач читає лекції з багатьох дисциплін і одна й та сама дисципліна читається багатьма викладачами, то вона «погано» піддається ієрархічній структуризації. Справа в тому, що між викладачами і дисциплінами існує зв'язок типу «багато-до-багатьох», який не є адекватним ієрархічній структурі даних (докладніше про це мова йтиме в розділах, присвячених логічному проектуванню).

Для відображення зв'язків цього типу була запропонована мережна модель даних.

## 2.3. Мережна модель даних

Мережна модель даних є розширенням ієрархічної моделі й призначена для адекватного моделювання зв'язків між сутностями типу «багато-до-багатьох». Домінуючий вплив на розвиток мережної моделі даних і побудованих на її основі систем баз даних мали пропозиції групи CODASYL DBTG (див. розділ 1).

Окрім формальної нотації для мережної моделі (мова опису даних – МОД) та пов'язаних з нею певних ключових концепцій, DBTG запропонувала МОД підсхеми для означення зовнішнього відображення концептуальної схеми бази даних та мову опису збережених даних (МОЗД) для означення способів зберігання даних на носіях. Сама концептуальна схема описується за допомогою МОД. Запропонована була й мова маніпулювання даними (ММД) для написання прикладних програм, що взаємодіють з базою даних у термінах зовнішньої схеми (підсхеми).

### 2.3.1. Мережна структура даних

Мережна структура даних є сукупністю схеми та екземпляра схеми. У свою чергу мережна схема формується з полів даних, типів записів і наборів, які також мають свої екземпляри. Власне з екземплярів полів, записів та наборів складається екземпляр схеми.

Елементарною одиницею даних мережної (так само, як ієрархічної) структури є поіменоване *поле даних*.

*Тип запису* – це поіменована впорядкована сукупність імен полів. Екземпляр запису (аналог сегмента в ієрархічній структурі даних) – це впорядкована сукупність значень полів запису. Екземпляр запису є одиницею доступу до даних мережної структури.

*Набір* – поіменований дворівневий ієрархічний зв'язок типів записів. Набір є основною конструкцією мови, запропонованої CODASYL DBTG. Із дворівневих наборів можуть будуватися багаторівневі ієрархії та мережні структури. Кожний *тип набору* – це сукупність зв'язків між двома або кількома типами записів, де один тип запису оголошується власником, а інший (або кілька інших) – членами типу набору. Екземпляр набору містить один екземпляр запису-власника

і довільну кількість екземплярів кожного типу запису-члена набору. Отже, набір описує дворівневий ієрархічний зв'язок типу «один-до-багатьох».

Тип запису КАФЕДРА (рис. 2.4, *а*) є власником типу набору, а типи ДИСЦИПЛІНА і ВИКЛАДАЧ — члени типу набору.

На рис. 2.4, *б* зображений екземпляр цього типу набору. Він містить один екземпляр типу запису КАФЕДРА і декілька типів записів ДИСЦИПЛІНА та ВИКЛАДАЧ.

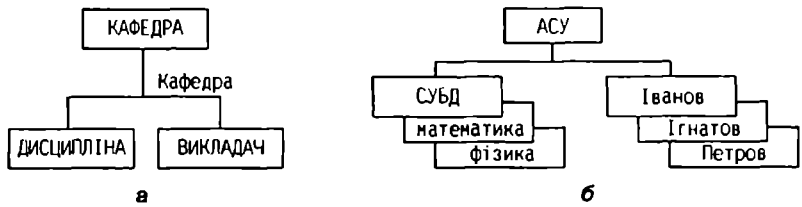



**Рис. 2.4.** Приклад опису набору «Кафедра»: тип набору (*а*); екземпляр набору (*б*)

Типи наборів можуть використовуватися для створення багаторівневих ієрархічних або мережних структур.

Для отримання багаторівневої ієрархії потрібно більше одного набору Тип запису, що є власником на нижньому рівні ієрархії, має бути також оголошений членом типу набору вищого рівня

Трирівнева ієрархія у вигляді двох типів наборів: Кафедра і Наукові праці показана на рис. 2.5. Тип запису ВИКЛАДАЧ є власником у типі набору Наукові праці та членом у типі набору Кафедра.


**Рис. 2.5.** Трирівнева ієрархічна структура

Для опису будь-якої $n$-рівневої ієрархії потрібно принаймні $n-1$ наборів.

Один тип запису може мати кілька батьківських записів, якщо вони є власниками різних типів наборів, тобто запис може бути членом багатьох наборів і мати декілька записів-власників. Так формуються мережні структури

Приклад схеми мережної структури даних, що складається з п'яти типів наборів даних, наведений на рис. 2.6. Мережну структуру формують набори Прослуховує, Читається і Читає, а тип запису ЛЕКЦІЯ є їхнім членом.

Отже, одні й ті самі типи записів можуть бути зв'язані в різні набори.

Мережна структура дає змогу моделювати зв'язки типу «багато-до-багатьох». Такий зв'язок, наприклад, існує між викладачами та дисциплінами: викладач читає багато дисциплін, і дисципліна може викладатися багатьма викладачами.

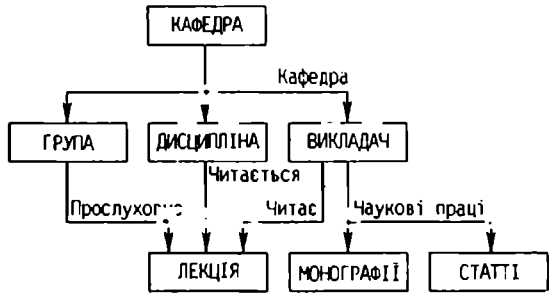
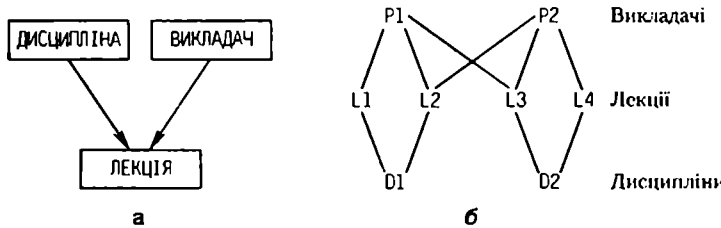



**Рис. 2.6.** Мережна структура даних

Зв'язок згаданого типу моделюється введенням нового типу запису ЛЕКЦІЯ і встановленням двох зв'язків (наборів) типу «один-до-багатьох» (рис. 2.7).

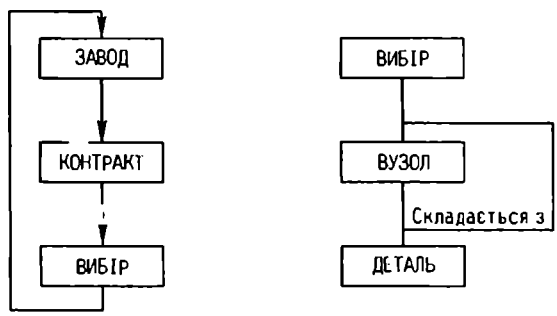



а б

**Рис. 2.7.** Моделювання зв'язку типу «багато-до-багатьох»: схема (а); екземпляр схеми (*б*)

Мережна структура може відображувати цикли та петлі. *Циклом* називається конфігурація, в якій предок типу запису є водночас його нащадком. Приклад циклічної мережної структури наведений на рис. 2.8. *а*. Тут вироби певного заводу є сировиною для виробів іншого заводу.

*Петля* – це структура, де один тип запису є одночасно власником і членом в одному типі набору. Структура виробів описана на рис. 2.8, *б*. Виріб складається з вузлів і деталей; у свою чергу вузли можуть складатися з інших вузлів і деталей. Тоді навколо типу запису ВУЗОЛ утворюється петля.




а б

**Рис. 2.8.** Приклади мережних структур: цикл (*а*); петля (*б*)

Мережна структура CODASYL не допускає відображення петель. У разі їхнього виникнення пропонується вводити додаткові типи записів, які дають змогу вирішити проблему «розвузлування».

Категорії схем, які зустрічаються в мережних структурах даних, зображені на рис. 2.9. Як бачимо, мережна структура даних здатна моделювати в повному обсязі ієрархічну структуру та власне мережні схеми. Окрім того, у загальному випадку вона дає можливість описувати петлі та цикли.

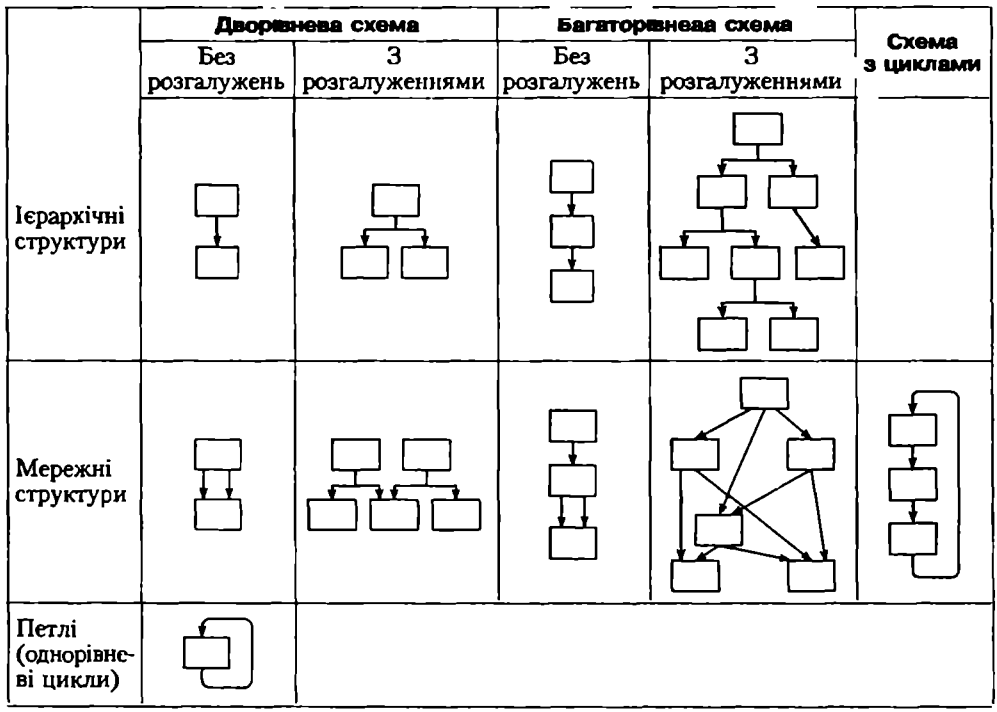



**Рис. 2.9.** Категорії схем мережної структури даних

## 2.3.2. Операції над мережною структурою

Операції над мережною структурою даних концептуально аналогічні до операцій над ієрархічною структурою. Мова маніпулювання даними в мережній структурі також є низькорівневою, тобто її використання передбачає програмування процесів «навігації» структурою Кожний тип запису в цій моделі, на відміну від ієрархічної, може мати поточний екземпляр запису, відносно якого можна організовувати подальше перебирання екземплярів. окрім того, кожний тип набору може мати поточний екземпляр набору.

Згідно з пропозиціями CODASYL, навігація структурою та вибирання даних виконуються в два етапи. Спочатку за допомогою послідовності операторів FIND визначається розташування екземпляра запису необхідного типу, тобто бажаний запис стає поточним. Потім командою GET можна вибрати поточний запис, вка-

зуючи (в разі потреби) його поля. Команда FIND насправді є сукупністю команд, об'єднаних спільною метою, — визначити місцезнаходження конкретного запису за вказаною стратегією. Оператори FIND дають змогу:

- знаходити запис за ключем або значеннями полів;
- послідовно переглядати записи вказаного типу, щоб знайти всі записи із заданими значеннями полів;
- послідовно переглядати всі записи-члени екземпляра набору;
- переглядати записи-члени екземпляра набору, які мають специфіковані значення в певних полях;
- знаходити власника даного запису в наборі.

Комерційні системи зазвичай підтримують мови, що дають можливість досить гнучко здійснювати навігацію мережею. Покажемо стисло, як це робиться в мережній СКДБ IDMS. Розглянемо оператори, що дають змогу встановити поточні записи, незалежно від зв'язків (наборів), у яких вони беруть участь. Усі наведені приклади використовують структуру, зображену на рис. 2.4. Оператор

```
FIND ім'я запису RECORD USING ідентифікатор
```

визначає пошук запису за значенням ключа бази даних. Тут ідентифікатор — змінна, що містить значення ключа бази даних. Оператор

```
FIND [NEXT DUPLICATE] ім'я запису RECORD [WHERE кваліфікація]
```

керує пошуком запису за значенням його елементу.

Розглянемо такий запит:

```
FIND ВИКЛАДАЧ RECORD WHERE Посада = "Професор"
GET ВИКЛАДАЧ RECORD
Початок циклу, виконувати до стану "запис відсутній"
 FIND NEXT DUPLICATE ВИКЛАДАЧ RECORD
 GET ВИКЛАДАЧ RECORD
Кінець циклу
```

Під час його виконання здійснюється перегляд списку всіх професорів, незалежно від членства запису ВИКЛАДАЧ у тому чи іншому наборі. Наступні три навігаційні операції дають змогу встановлювати поточні записи з урахуванням їхньої належності до будь-яких наборів Оператор

```
FIND {NEXT | PRIOR | FIRST | LAST | n} <ім'я запису> RECORD <ім'я набору> SET
```

дає можливість специфікувати елемент набору, який вибирається. Наприклад, команда

```
FIND FIRST ВИКЛАДАЧ RECORD КАФЕДРА SET
```

робить поточним перший запис типу ВИКЛАДАЧ, що належить до поточного екземпляра набору КАФЕДРА.

Команда

```
FIND OWNER RECORD <ім'я набору> SET
```

задає вибирання екземпляра запису-власника, що належить до екземпляра набору вказаного типу. Ця команда, як й інші. застосовується до поточного екземпляра набору. Нарешті команда

```
FIND <ім'я запису> RECORD VIA [CURRENT] <ім'я набору> SET WHERE <кваліфікація>
```

забезпечує вибір запису за його членством у наборі й значенням елементу даних.

Як і в ієрархічній моделі даних, мова маніпулювання мережною структурою надає можливість вставляти, модифікувати й видаляти екземпляри записів відповідних типів з урахуванням їхньої належності до тих чи інших наборів.

### 2.3.3. Переваги та недоліки мережної моделі

Для мережної моделі даних властиві майже ті самі переваги та недоліки, що й для ієрархічної моделі. Принциповою відмінністю мережної структури від ієрархічної є можливість безпосередньо відображувати складніші типи зв'язків. Натомість ієрархічні системи простіші в реалізації та експлуатації.

## 2.4. Історія реляційної моделі даних

Базам даних постійно загрожувала небезпека перетворитися на громіздкі й дуже складні для використання системи. Адже наявність сотень типів записів, зв'язаних сотнями типів наборів, робили схему бази даних заплутаною, а її використання – неймовірно складним завданням

Уникнути складності ієрархічної та мережної моделей вдалося в реляційній моделі даних, що у певному розумінні була поверненням до файлових структур.

Структура даних у реляційній моделі може бути проінтерпретована як звичайна двовимірна таблиця, що складається зі стовпців і рядків. Проте порівняно з ієрархічною та мережною структурами рівень мови маніпулювання цією структурою значно вищий. Це дало змогу більш глибоко й усебічно вирішити багато проблем, пов'язаних із базами даних, а саме: вибирання даних, їхня незалежність, цілісність та захист.

Реляційна структура даних уперше дала можливість залучити до дослідження практично всіх питань, пов'язаних з базами даних, формальні математичні дисципліни (теорію множин, теорію відношень та числення предикатів першого порядку), шо перетворило науку про бази даних, яка раніше багато в чому мала описовий характер, на точну математичну дисципліну.

Реляційні системи не відразу набули широкого розповсюдження. Незважаючи на те, що основні теоретичні результати в цій галузі були отримані ще в 70-х роках і саме тоді з'явилися перші прототипи реляційних СКБД, тривалий час реалізація таких систем вважалася неможливою. Проте поступове накопичення методів організації реляційних баз даних, а також алгоритмів керування ними сприяли тому, що вже в середині 80-х років реляційні системи майже цілком витіснили зі світового ринку попередні СКБД ієрархічного та мережного типів.

Нині основним предметом критики реляційних СКБД є не їхня недостатня ефективність, а властива цим системам певна обмеженість під час використання в так званих нетрадиційних галузях (найпоширенішими прикладами є системи автоматизації проектування), де потрібні гранично складні структури даних. Ще один недолік реляційних баз даних полягає у відсутності засобів, які б могли адекватно відображувати семантику предметної області. Іншими словами, можливості відображення знань про семантичну специфіку предметної області в реляційних системах вельми обмежені. Сучасні дослідження в галузі постреляційних систем головним чином присвячені саме усуненню цих недоліків.

У результаті досліджень у галузі штучного інтелекту та баз знань було створено модель семантичних мереж і фреймову модель даних. За останнє десятиліття були проведені дослідження й розробки в галузі дедуктивних, об'єктно-орієнтованих, розподілених баз даних та баз даних в Інтернеті. Інформацію про особливості моделей даних, що підтримують згадані вище типи баз даних, можна знайти у відповідних розділах цієї книги.

## Контрольні запитання та завдання

1. Що таке модель даних? Які моделі даних ви знаєте?
2. Чим відрізняються слабко типізовані моделі даних від сильно типізованих?
3. Дайте означення ієрархічної структури даних.
4. Які є операції маніпулювання ієрархічною структурою даних? Наведіть приклади їхнього використання.
5. Дайте означення мережної структури даних.
6. Які є операції маніпулювання мережною структурою даних? Наведіть приклади їхнього використання.
7. Які категорії схем мережної структури ви знаєте?

# Розділ 3
# Реляційна модель даних

- ✦ Означення реляційної структури даних
- ✦ Реляційна алгебра
- ✦ Реляційне числення
- ✦ Реляційна повнота та селективна потужність

## 3.1. Реляційна структура даних

Теоретичні основи реляційної моделі баз даних були закладені Е. Коддом на початку 70-х років XX століття. На відміну від поширених на той час систем з ієрархічними чи мережними типами структур даних, реляційний підхід запропонував спрощені структури даних – *реляції*, або *таблиці*, та розширив можливості мови маніпулювання даними. У науковій літературі, присвяченій реляційним базам даних, на означення того, що було названо вище реляцією або таблицею, часто застосовується термін *відношення*. Кожен із цих термінів має свої переваги й недоліки. Терміном «відношення» у математичній теорії відношень позначається дещо інше поняття і тому тлумачення цього терміну в контексті теорії баз даних є неоднозначним. Недолік терміну «реляція» полягає в його недавньому іншомовному походженні. Слово «таблиця» часто вживається в різних значеннях і має більш прикладний наголос. Тому для теоретичних розглядів будемо вживати термін «реляційне відношення», а в прикладах – «відношення» або «таблиця».

Нехай $V$ – основний алфавіт, тобто певна скінченна множина; $\omega$ – певний виділений елемент ($\omega \notin V$), який означає «невизначено», або *NULL-значення*.

Позначимо через $D'_1, D'_2, \ldots, D'_{n1}$ підмножини $V^*$ – множини всіх слів в алфавіті $V$. *Домени* визначаються як $D_i = D'_i \cup \{\omega\}$.

Нехай $\Omega$ – множина імен. Введемо однозначне, але не обов'язково ін'єктивне (тобто взаємно однозначне) відображення $N: \Omega \to \{D_1, D_2, \ldots, D_n\}$, яке іменує домени. Кожному домену може бути приписано кілька імен, але кожне ім'я може бути приписане лише одному домену.

Нехай $\Omega_i = N^{-1}(D_i)$ – множина всіх імен домена $D_i$. Безпосередньо з означення випливає, що множини $\Omega_i$ мають такі властивості:

$$\Omega_i \cap \Omega_j = \varnothing; \quad \cup_{(i=1,\ldots,n)} \Omega_i = \Omega.$$

*Атрибутом* називається пара $(A, D_i)$, де $A \in \Omega_i$, тобто атрибут означується як іменований домен; у різних атрибутів домени можуть бути однакові, а імена мають відрізнятися.

Нехай $\Re \subseteq (A_1,D_{i1}) \otimes (A_2,D_{i2}) \otimes ... \otimes (A_k,D_{ik})$ – відношення над декартовим добутком атрибутів. Введемо над множиною таких відношень розбиття на класи еквівалентності: одному класу належатимуть відношення, які відрізняються лише порядком компонентів у декартовому добутку. Представник такого класу, що називається *реляційним відношенням*, або *таблицею* $R((A_1, D_{i1}), (A_2, D_{i2}), ..., (A_k,D_{ik}))$, і є головним об'єктом наших досліджень.

У класичному алгебраїчному розумінні відношення визначається як підмножина декартового добутку деяких множин, але у реляційному відношенні суттєвим є не місце розташування компонента (стовпця), а його ім'я. У табл. 3.1 порівнюються класичний алгебраїчний та реляційний підходи.

**Таблиця 3.1.** Порівняння класичного алгебраїчного та реляційного підходів

| Поняття | Класичне тлумачення | Реляційне тлумачення |
|---|---|---|
| Компонент | Довільна множина або домен | Атрибут, тобто пара {ім'я, домен} |
| Позиція у декартовому добутку | Є суттєвою | Не є суттєвою |
| Звернення або доступ до компонента | За номером позиції | За іменем атрибута |

Розглянемо приклад. Таблиця розкладу руху потягів чи літаків, умовно кажучи, складається з двох частин: наповнення та опису структури таблиці. Наповнення – це ті номери рейсів потягів чи літаків та час їхнього відправлення, що занесені у відповідні клітинки таблиці й періодично змінюються, а структура таблиці описується заголовками стовпців. Згідно з термінологією баз даних і реляційного підходу, наповнення таблиць називають *даними* Інколи буває так, що таблиця розкладу містить лише порожні стовпці. Такий об'єкт фахівець з баз даних міг би назвати *схемою*.

Реляційне відношення $R((A_1, D_{i1}), (A_2, D_{i2}), ..., (A_k, D_{ik}))$ у реляційній моделі даних зображується через схему й екземпляр відношення. Схема записується у вигляді $R(A_1 : D_{i1}, A_2 : D_{i2}, ..., A_k : D_{ik})$ або просто $R(A_1, A_2, ..., A_k)$, якщо зв'язок атрибутів з доменами відомий апріорі.

Сукупність схем реляційних відношень називають *схемою бази даних*, або *реляційною схемою*.

Схема реляційного відношення має такі властивості:

- реляційне відношення має ім'я;
- імена атрибутів у межах схеми одного реляційного відношення мають бути унікальними;
- порядок атрибутів у схемі реляційного відношення не є суттєвим, оскільки звернення до атрибута здійснюється за його іменем, а не за номером.

Приклад схеми реляційного відношення: ВИКЛАДАЧ(#Ід, ПІБ, Адреса). Тут ВИКЛАДАЧ – це ім'я реляційного відношення, а #Ід, ПІБ, Адреса – імена його атрибутів.

*Екземпляр реляційного відношення* це його наповнення. Точніше, екземпляр є множиною кортежів, а *кортеж* – це множина значень $\{v_1, ..., v_k\}$, $v_1 \in D_1, ..., v_k \in D_k$. Екземпляр відношення має такі властивості.

- порядок кортежів довільний;
- кортежі, як елементи множини, мають бути унікальними в межах реляційного відношення.

Реляційне відношення може бути зображене у вигляді таблиці. Приклад табличного відображення реляційного відношення наведено на рис. 3.1.

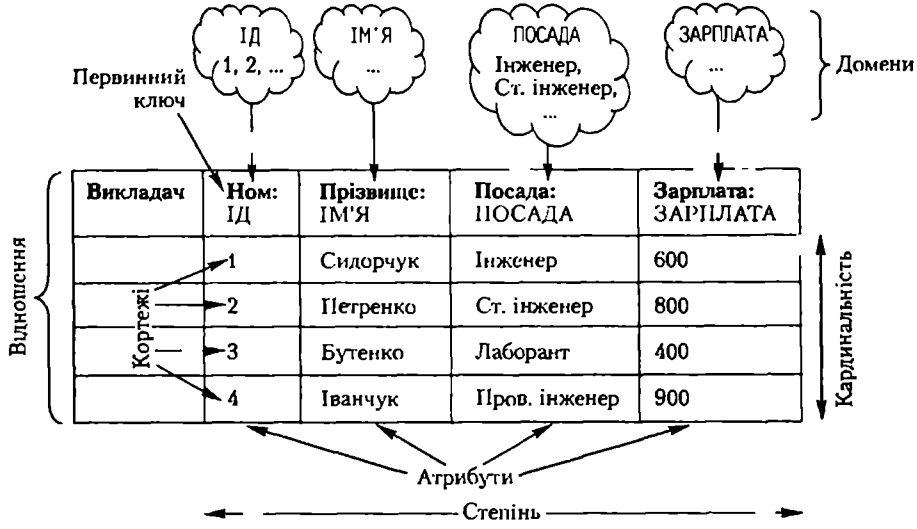



**Рис. 3.1.** Приклад табличного зображення реляційного відношення

*Таблиця* – це пойменоване двовимірне зображення відношення; вона складається з одного чи більше пойменованих стовпців і нуля або більше рядків. Назва таблиці відповідає імені реляційного відношення, імена стовпців – іменам атрибутів, а рядки – кортежам.

Відповідність термінів, що вживаються під час розгляду реляційних відношень та їхніх табличних зображень, вказана в табл. 3.2.

**Таблиця 3.2.** Відповідність термінів, що вживаються під час розгляду реляційних відношень та їхніх табличних зображень

| Формальний термін | Неформальний еквівалент |
|---|---|
| Відношення | Таблиця |
| Кортеж | Рядок, запис |
| Кардинальність | Кількість рядків |
| Атрибут | Стовпець, поле |
| Степінь | Кількість стовпців |
| Первинний ключ | Ідентифікатор |
| Домен | Область допустимих значень |

### Поняття ключа

У будь-якому реляційному відношенні можна виділити таку множину атрибутів, що набори відповідних їм значень однозначно ідентифікуватимуть кортежі відношення. Це випливає з того, що відношення є різновидом множини, а отже, не може містити кортежів, які повторюються, відтак – уся множина атрибутів реляційного відношення унікально ідентифікує кортежі. Множина атрибутів, що однозначно ідентифікують кортежі реляційного відношення, називається *ключем*.

Відношення може мати декілька різних ключів. Наприклад, у реляційному відношенні ВИКЛАДАЧ(#Ід, ПІБ, Адреса) ключами є #Ід, пара (ПІБ, Адреса) й усі три атрибути (#Ід, ПІБ, Адреса). Ключ називається *простим*, якщо складається з одного атрибута (#Ід), і *складеним* – якщо з кількох атрибутів, наприклад (ПІБ, Адреса). Ключ називають *надлишковим*, якщо певна його підмножина також є ключем. Наприклад, ключ (#Ід, ПІБ, Адреса) є надлишковим тому, що містить атрибут #Ід, який також є ключем. Ключ, що не є надлишковим, називають *мінімальним*. Іноді надлишковий ключ називають *суперключем*, а мінімальний *можливим ключем*. Реляційне відношення може мати багато можливих ключів, але тільки один із них є *первинним*.

Первинний ключ має такі властивості:

- кожне реляційне відношення має один і лише один первинний ключ;
- значення всіх атрибутів первинного ключа не можуть бути невизначеними, оскільки він має унікально ідентифікувати всі кортежі будь-якого екземпляра реляційного відношення;
- значення первинного ключа не можуть повторюватися, але допускаються повторення значень частини складеного первинного ключа;
- апріорі значення первинного ключа не впливають на порядок кортежів у табличному зображенні реляційного відношення, хоча інколи таблиці впорядковують за ключем;
- первинний ключ не впливає на доступ до кортежів, який може бути здійснено за значенням будь-якого атрибута чи набору атрибутів незалежно від того, чи є він первинним ключем.

Якщо розглядається певне реляційне відношення, то сукупність атрибутів, що є первинним ключем іншого реляційного відношення, називається *зовнішнім (сторонніми) ключем*. За допомогою зовнішніх ключів у реляційній моделі встановлюються зв'язки між реляційними відношеннями (точніше, між їхніми кортежами).

Зовнішні ключі мають такі властивості:

- значення зовнішнього ключа завжди посилається на певне значення відповідного первинного ключа, тобто будь-яке значення зовнішнього ключа має бути значенням первинного ключа іншого відношення;
- значення зовнішнього ключа, на відміну від значень первинного ключа, можуть бути невизначеними й повторюватися в межах реляційного відношення.

Розглянемо приклад. Нехай до реляційного відношення ВИКЛАДАЧ додано атрибут #Каф – код кафедри, де працює викладач: ВИКЛАДАЧ(#Ід, ПІБ, Адреса, #Каф).

Крім того, є ще одне відношення КАФЕДРА(#Каф, Назва, Керівник, Місцезнаходження). Первинним ключем відношення ВИКЛАДАЧ є #Ід, а відношення КАФЕДРА — #Каф; цей самий атрибут у реляційному відношенні ВИКЛАДАЧ є зовнішнім ключем. Перша властивість зовнішнього ключа означає: якщо в таблиці ВИКЛАДАЧ є кортеж, у якому атрибуту #Каф відповідає певне значення, наприклад К1, то в таблиці КАФЕДРА обов'язково має бути кортеж, у якому атрибуту #Каф також відповідає значення К1. Саме завдяки рівності значень, що відповідають цим двом атрибутам, встановлюється зв'язок між кортежами різних реляційних відношень.

## 3.2. Реляційна алгебра

До складу *реляційної моделі даних*, крім структури даних, мають входити операції маніпулювання даними. З усіх таких операцій складається *мова запитів*. Найбільш відомими мовами запитів у реляційній моделі є реляційна алгебра та реляційне числення. Перша з цих мов буде розглянута в даному підрозділі, різновиди другої — в підрозділах 3.3 та 3.4.

У класичному розумінні алгебра визначається як пара, що складається з основної множини і множини операцій (сигнатури), при цьому аргументи й результат кожної операції належать основній множині.

*Реляційна алгебра* є алгеброю в строгому класичному розумінні її визначення. Елементами основної множини є реляційні відношення. У зв'язку з цим операції алгебри можуть вкладатися одна в одну, тобто аргументом певної операції може бути результат виконання іншої операції. Це дає можливість записувати запити довільного рівня складності у вигляді виразів, що містять вкладені одна в одну операції.

### 3.2.1. Операції реляційної алгебри

Сигнатура реляційної алгебри Кодда складається з восьми операцій. Перш ніж детально розглянути ці операції, введемо поняття сумісності реляційних відношень. Це поняття є необхідним, оскільки деякі операції (а саме: теоретико-множинні операції об'єднання, перетину та різниці) визначені лише для сумісних реляційних відношень.

Реляційні відношення $R_1(A_1, ..., A_n)$ і $R_2(B_1, ..., B_k)$ називаються *сумісними*, якщо:

1) у них однакова кількість атрибутів, тобто $k = n$;
2) можна встановити взаємно однозначну відповідність між доменами атрибутів першої та другої реляцій, тобто існує таке бієктивне відображення

$$S:\{1, ..., k\} \to \{1, ..., k\},$$

що

$$N(A_i) = N(B_{S(i)}),\ i = 1, ..., k,$$

тобто домени зіставлених атрибутів однакові.

**ПРИМІТКА**

Для зручності вважатимемо, що зіставлені атрибути сумісних відношень повинні мати однакові імена.

Дамо також означення кількох властивостей бінарних операцій:

- операція φ є комутативною, якщо $A \varphi B = B \varphi A$;
- операція φ є асоціативною, якщо $(A \varphi B) \varphi C = A \varphi (B \varphi C)$;
- операція φ є дистрибутивною з операцією θ, якщо $A \varphi (B \theta C) = (A \varphi B) \theta (A \varphi C)$.

Даючи означення бінарним операціям реляційної алгебри, ми будемо вказувати, які з цих властивостей вони мають.

**ПРИМІТКА**

Оскільки різні відношення можуть містити атрибути з однаковими іменами, то під час виконання бінарних операцій у кінцевому відношенні можуть повторюватися імена атрибутів. Для забезпечення унікальності імен атрибутів вони уточнюються іменами відповідних відношень згідно з таким синтаксисом: `<ім'я відношення>.<ім'я атрибута>`.

Під час розгляду операцій реляційної алгебри атрибути позначатимемо великими літерами з початку латинського алфавіту: $A$, $B$, ..., а множини атрибутів – великими літерами з середини латинського алфавіту: $L$, $M$, ... .

Отже, розглянемо операції реляційної алгебри.

### Об'єднання

Нехай $L$ – певна множина атрибутів. *Об'єднанням* сумісних реляційних відношень $R_1$ і $R_2$ зі схемами $R_1(L)$ і $R_2(L)$ (позначається як $R_1 \cup R_2$) називається таке реляційне відношення $R$ зі схемою $R(L)$, що містить кортежі обох поєднуваних відношень, але без повторень

$$R(L) = R_1(L) \cup R_2(L) = \{r \mid r \in R_1 \vee r \in R_2\}.$$

Операція комутативна, асоціативна й дистрибутивна щодо перетину.

**Приклад**

$R_1$

| $A$ | $B$ |
|---|---|
| $a_1$ | $b_1$ |
| $a_1$ | $b_2$ |
| $a_2$ | $b_3$ |

$R_2$

| $A$ | $B$ |
|---|---|
| $a_1$ | $b_1$ |
| $a_2$ | $b_1$ |

$R_1 \cup R_2$

| $A$ | $B$ |
|---|---|
| $a_1$ | $b_1$ |
| $a_1$ | $b_2$ |
| $a_2$ | $b_1$ |
| $a_2$ | $b_3$ |

### Перетин

Припустимо, що $L$ - певна множина атрибутів. *Перетином* сумісних реляційних відношень $R_1$ і $R_2$ зі схемами $R_1(L)$ і $R_2(L)$ (позначається як $R_1 \cap R_2$) називається

таке реляційне відношення $R$ зі схемою $R(L)$, яке містить кортежі, що входять до складу обох операндів:

$$R(L) = R_1(L) \cap R_2(L) = \{r \mid r \in R_1 \ \& \ r \in R_2\}.$$

Операція комутативна, асоціативна й дистрибутивна щодо об'єднання

**Приклад**

$R_1$

| $A$ | $B$ |
|---|---|
| $a_1$ | $b_1$ |
| $a_1$ | $b_2$ |
| $a_2$ | $b_3$ |

$R_2$

| $A$ | $B$ |
|---|---|
| $a_1$ | $b_1$ |
| $a_2$ | $b_1$ |

$R_1 \cap R_2$

| $A$ | $B$ |
|---|---|
| $a_1$ | $b_1$ |

**Різниця**

Нехай $L$ – певна множина атрибутів. *Різницею* сумісних реляційних відношень $R_1$ і $R_2$ зі схемами $R_1(L)$ і $R_2(L)$ (позначається як $R_1 - R_2$) називається реляційне відношення $R$ зі схемою $R(L)$, що містить ті кортежі з першого операнда $R_1$, яких немає у другому операнді $R_2$:

$$R(L) = R_1(L) - R_2(L) = \{r \mid r \in R_1 \ \& \ r \notin R_2\}.$$

Операція не комутативна, не асоціативна й не дистрибутивна з іншими операціями.

**Приклад**

$R_1$

| $A$ | $B$ |
|---|---|
| $a_1$ | $b_1$ |
| $a_1$ | $b_2$ |
| $a_2$ | $b_3$ |

$R_2$

| $A$ | $B$ |
|---|---|
| $a_1$ | $b_1$ |
| $a_2$ | $b_1$ |

$R_1 - R_2$

| $A$ | $B$ |
|---|---|
| $a_1$ | $b_2$ |
| $a_2$ | $b_3$ |

Зауважимо, що $R \cap S = R - (R - S)$.

Зазначимо деякі особливості теоретико-множинних операцій.

- У реляційній алгебрі, на відміну від алгебри множин, не використовується операція доповнення, оскільки певні домени можуть бути нескінченними або містити дуже багато значень і в результаті операції доповнення можна отримати або нескінченне відношення, або відношення з дуже великою кількістю кортежів
- Вимога сумісності операндів зумовлена тим, що без цього обмеження результатом теоретико-множинних операцій могли б бути різноструктурні кортежі, а не реляційні відношення.

Розглянемо операції, які визначені лише в реляційній алгебрі.

### Проекція

*Проекцією* реляційного відношення $R$ зі схемою $R(A_1, ..., A_k)$ за атрибутами $A_{i1}, ..., A_{in}$, де $\{A_{i1}, ..., A_{in}\} \subset \{A_1, ..., A_k\}$, що позначається $R[A_{i1}, ..., A_{in}]$, називається таке відношення $S$ зі схемою $S(A_{i1}, ..., A_{in})$, кортежі якого отримані з кортежів відношення $R$ шляхом видалення значень, що не належать атрибутам, за якими виконується проекція. При цьому в кінцевому відношенні повторні екземпляри кортежів видаляються.

Якщо $r$ є кортежем відношення $R$, то записом $r[L]$, де $L$ – підмножина атрибутів відношення $R$, позначимо множину тих елементів кортежу $r$, що відповідають значенням атрибутів з $L$. Тоді наведене вище визначення проекції може бути записане у такий спосіб:

$$S = R[A_{i1}, ..., A_{in}] = \{r[A_{i1}, ..., A_{in}] \mid r \in R\}.$$

Операція проекції також записується як $\pi_{A_{i1}, ..., A_{in}}(R)$.

З теоретичної точки зору операція проекції не є «чистою», оскільки список атрибутів не належить основній множині (тобто не є реляційним відношенням), а тому не може бути операндом. За «чистого» теоретичного підходу операцію проекції слід вважати бінарною, а на практиці вона розглядається як унарна з параметрами.

Такі ж самі проблеми мають місце в усіх операціях, де операндами є списки атрибутів.

**Приклад**

$R$

| $A$ | $B$ | $C$ |
|---|---|---|
| $a_1$ | $b_1$ | $c_1$ |
| $a_1$ | $b_2$ | $c_1$ |
| $a_2$ | $b_3$ | $c_1$ |
| $a_2$ | $b_4$ | $c_2$ |

$R[A, C]$

| $A$ | $C$ |
|---|---|
| $a_1$ | $c_1$ |
| $a_2$ | $c_1$ |
| $a_2$ | $c_2$ |

### Обмеження (селекція)

Спочатку дамо означення θ-порівнянності атрибутів. Нехай θ є одним з операторів порівняння: $=, \neq, <, \leq, >, \geq$ (набір операторів можна розширити). Атрибути $A$ і $B$ одного й того самого чи різних відношень називаються θ-*порівнянними*, якщо для будь-яких значень $a \in A$ і $b \in B$ результат операції $a\ \theta\ b$ є визначеним (істинним або хибним). Інакше кажучи, ця операція визначена на відповідних атрибутах. Набори атрибутів $L = (A_1, ..., A_k)$ та $M = (B_1, ..., B_n)$ називаються θ-порівнянними, якщо $k = n$ і $A_i$ θ-порівнянне з $B_i$ $(i = 1, 2, ..., k)$. Тоді вираз $L\ \theta\ M$ розуміють так:

$$L\ \theta\ M = (A_1\ \theta\ B_1)\ \&...\&\ (A_k\ \theta\ B_k).$$

Тепер дамо означення операції обмеження.

Нехай $L$ і $M$ – набори θ-порівнянних атрибутів схеми відношення $R$. Тоді *обмеженням* реляційного відношення $R$ за умовою $L$ θ $M$, що позначається $R[L\ \theta\ M]$, називається реляпійне відношення, кортежі якого відповідають умові $L$ θ $M$:

$$S = R[L\ \theta\ M] = \{r \mid r \in R\ \&\ r[L]\ \theta\ r[M]\}$$

**ПРИМІТКА**

Множина $M$ може складатися як з атрибутів, так і з констант.

Операція обмеження також записується як $\sigma_{L\theta M}(R)$.

**Приклад**

$R$

| $A$ | $B$ | $C$ |
|---|---|---|
| $a_1$ | $b_1$ | $c_1$ |
| $a_1$ | $b_2$ | $c_1$ |
| $a_2$ | $b_3$ | $c_1$ |
| $a_2$ | $b_4$ | $c_2$ |

$R[A = a_2]$

| $A$ | $B$ | $C$ |
|---|---|---|
| $a_2$ | $b_3$ | $c_1$ |
| $a_2$ | $b_4$ | $c_2$ |

## Декартів добуток

*Декартовим добутком* реляційних відношень $R$ і $S$ зі схемами $R(A_1, A_2, ..., A_n)$ та $S(B_1, B_2, ..., B_m)$ відповідно, що позначається $R \times S$, називається відношення $Q$ зі схемою $Q(A_1, A_2, ..., A_n, B_1, B_2, ..., B_m)$, яке містить усі можливі з'єднання кортежів відношення $R$ з кортежами відношення $S$:

$$Q = R \times S = \{(r, s) \mid r \in R\ \&\ s \in S\}.$$

Операція комутативна й асоціативна.

**Приклад**

$R1$

| $A$ | $B$ |
|---|---|
| $a_1$ | $b_1$ |
| $a_1$ | $b_2$ |
| $a_2$ | $b_3$ |

$R_2$

| $C$ | $D$ |
|---|---|
| $c_1$ | $d_1$ |
| $c_2$ | $d_1$ |

$R = R_1 \times R_2$

| $A$ | $B$ | $C$ | $D$ |
|---|---|---|---|
| $a_1$ | $b_1$ | $c_1$ | $d_1$ |
| $a_1$ | $b_1$ | $c_2$ | $d_1$ |
| $a_1$ | $b_2$ | $c_1$ | $d_1$ |
| $a_1$ | $b_2$ | $c_2$ | $d_1$ |
| $a_2$ | $b_3$ | $c_1$ | $d_1$ |
| $a_2$ | $b_3$ | $c_2$ | $d_1$ |

### З'єднання

Припустимо. що відношення $R$ має схему $R(L. M)$, а відношення $S$ – схему $S(N, P)$. Нехай множини атрибутів $M$ і $N$ - θ-порівнянні. Тоді *з'єднанням*, або *θ-з'єднанням*, відношень $R$ і $S$ за умовою $M$ θ $N$, що позначається як $R[M$ θ $N]S$, називається відношення $Q$ зі схемою $Q(L, M, N. P)$, кортежі якого можна отримати з'єднанням тих кортежів відношень $R$ і $S$, на яких виконується умова $M$ θ $N$:

$$Q = R[M \theta N]S = \{(r, s) \mid r \in R \ \& \ s \in S \ \& \ r[M] \ \theta \ s[N]\}.$$

Під час з'єднання атрибути, за якими виконується така операція, повторюються в кінцевому реляційному відношенні. Операція комутативна й асоціативна.

Іноді операція з'єднання позначається як $R \overset{F}{\bowtie} S$, де $F$ – умова з'єднання.

З'єднання за умовою рівності називається *еквіз'єднанням*. З'єднання за умовою рівності, коли один з порівнюваних атрибутів (чи група порівнюваних атрибутів) видаляється з кінцевого відношення, називається *природним з'єднанням*; на його позначення використовується символ «*». Наприклад, якщо задані відношення $R(A, B, C, D)$ і $S(C, D. E)$, то в результаті виконання операції $Q = R * S$ отримаємо реляційне відношення $Q(A, B, C, D, E)$.

Серед операцій θ-з'єднання виділяють операцію *напівз'єднання*, за якої з результату видаляються всі атрибути одного з відношень, що з'єднуються. Вона записується як $R[M$ θ $N)S$ і формально визначається так:

$$R[M \theta N)S = \{r \mid r \in R \ \& \ s \in S \ \& \ r[M] \ \theta \ s[N]\}.$$

Операція напівз'єднання не розширює можливостей реляційної алгебри, оскільки вона виражається через з'єднання і проекцію в такий спосіб:

$$R[M \theta N)S = (R[M \theta N]S)[L, M].$$

**Приклад**

$R$

| $A$ | $B$ | $C$ | $D$ |
|---|---|---|---|
| $a_1$ | $b_1$ | $c_1$ | $d_1$ |
| $a_1$ | $b_1$ | $c_2$ | $d_1$ |
| $a_1$ | $b_2$ | $c_1$ | $d_1$ |
| $a_2$ | $b_3$ | $c_1$ | $d_1$ |
| $a_2$ | $b_4$ | $c_2$ | $d_3$ |

$S$

| $C$ | $D$ | $E$ |
|---|---|---|
| $c_1$ | $d_1$ | $e_2$ |
| $c_2$ | $d_1$ | $e_3$ |
| $c_2$ | $d_1$ | $e_1$ |

$R[C, D = C, D]S$

| $A$ | $B$ | $C$ | $D$ | $C$ | $D$ | $E$ |
|---|---|---|---|---|---|---|
| $a_1$ | $b_1$ | $c_1$ | $d_1$ | $c_1$ | $d_1$ | $e_2$ |
| $a_1$ | $b_2$ | $c_1$ | $d_1$ | $c_1$ | $d_1$ | $e_2$ |
| $a_2$ | $b_3$ | $c_1$ | $d_1$ | $c_1$ | $d_1$ | $e_2$ |
| $a_1$ | $b_1$ | $c_2$ | $d_1$ | $c_2$ | $d_1$ | $e_3$ |
| $a_1$ | $b_2$ | $c_2$ | $d_1$ | $c_2$ | $d_1$ | $e_3$ |
| $a_1$ | $b_1$ | $c_2$ | $d_1$ | $c_2$ | $d_1$ | $e_1$ |
| $a_1$ | $b_2$ | $c_2$ | $d_1$ | $c_2$ | $d_1$ | $e_1$ |

$R * S$

| $A$ | $B$ | $C$ | $D$ | $E$ |
|---|---|---|---|---|
| $a_1$ | $b_1$ | $c_1$ | $d_1$ | $e_2$ |
| $a_1$ | $b_2$ | $c_1$ | $d_1$ | $e_2$ |
| $a_2$ | $b_3$ | $c_1$ | $d_1$ | $e_2$ |
| $a_1$ | $b_1$ | $c_2$ | $d_1$ | $e_3$ |
| $a_1$ | $b_2$ | $c_2$ | $d_1$ | $e_3$ |
| $a_1$ | $b_1$ | $c_2$ | $d_1$ | $e_1$ |
| $a_1$ | $b_2$ | $c_2$ | $d_1$ | $e_1$ |

### Ділення

Нехай задано відношення зі схемою $R(M, N)$. *Образом реляційного відношення $R$ за кортежем $t_1 \in R[M]$* називається така множина кортежів $t_2 \in R[N]$, для яких зчеплення $(t_1, t_2)$ належить відношенню $R$. Образ $R$ за кортежем $t_1$ позначається $I_R(t_1)$ і формально визначається у такий спосіб:

$$I_R(t_1) = \{ t_2 \mid t_2 \in R[N] \ \& \ (t_1, t_2) \in R \}.$$

**Приклад**

$R$

| $A$ | $B$ | $C$ |
|---|---|---|
| $a_1$ | $b_1$ | $c_1$ |
| $a_1$ | $b_1$ | $c_2$ |
| $a_1$ | $b_3$ | $c_2$ |
| $a_2$ | $b_1$ | $c_4$ |

$I_R(a_1)$

| $B$ | $C$ |
|---|---|
| $b_1$ | $c_1$ |
| $b_1$ | $c_2$ |
| $b_3$ | $c_2$ |

$I_R(a_2)$

| $B$ | $C$ |
|---|---|
| $b_1$ | $c_4$ |

$I_R(a_1, b_1)$

| $C$ |
|---|
| $c_1$ |
| $c_2$ |

$I_R(c_2)$

| $A$ | $B$ |
|---|---|
| $a_1$ | $b_1$ |
| $a_1$ | $b_3$ |

Нехай задано відношення $R$ і $S$ зі схемами $R(M, N)$ та $S(K, L)$, для яких проекції $R[N]$ та $S[K]$ є сумісними. *Діленням* відношення $R$ на відношення $S$ за наборами атрибутів $N$ і $K$ (позначається $R[N \div K]S$) називається операція, результатом якої є відношення $Q$ зі схемою $Q(M)$, що складається з таких кортежів $t \in R[M]$, образи $I_R(t)$ яких містять усі кортежі проекції $S[K]$, тобто:

$$Q = R[N \div K]S = \{t \mid t \in R[M] \ \& \ I_R(t) \supseteq S[K]\}.$$

Можна показати. шо операція ділення виражається через інші операції алгебри в такий спосіб.

$$R[N \div K]S = R[M] - ((R[M] \times S[K]) - R)[M].$$

Операція не комутативна й не асоціативна.

**Приклад**

*R*

| *A* | *B* | *C* |
|---|---|---|
| $a_1$ | $b_1$ | $c_1$ |
| $a_1$ | $b_1$ | $c_2$ |
| $a_1$ | $b_3$ | $c_2$ |
| $a_2$ | $b_1$ | $c_4$ |

*S*

| *C* | *D* |
|---|---|
| $c_1$ | $d_1$ |
| $c_1$ | $d_2$ |
| $c_2$ | $d_1$ |
| $c_2$ | $d_3$ |

*S*[*C*]

| *C* |
|---|
| $c_1$ |
| $c_2$ |

*R*[*C* ÷ *C*]*S*

| *A* | *B* |
|---|---|
| $a_1$ | $b_1$ |

## 3.2.2. Приклади застосування реляційної алгебри

Нехай задано реляційну схему бази даних вищого навчального закладу:

```
ФАКУЛЬТЕТ(#F, Назва, Декан, Корпус, Фонд)
КАФЕДРА(#D, #F, Назва, #ЗАВІДУВАЧ, Корпус, Фонд)
ВИКЛАДАЧ(#T, #D, Прізвище, Посада, Тел)
ГРУПА(#G, #D, Курс, Номер, Кількість, #КУРАТОР)
ПРЕДМЕТ(#S, Назва)
АУДИТОРІЯ(#R, Номер, Корпус, Місткість)
ЛЕКЦІЯ(#T, #G, #S, #R, Тип, День, Тиждень)
```

На рис. 3.2 зображено зв'язки між відношеннями описаної схеми

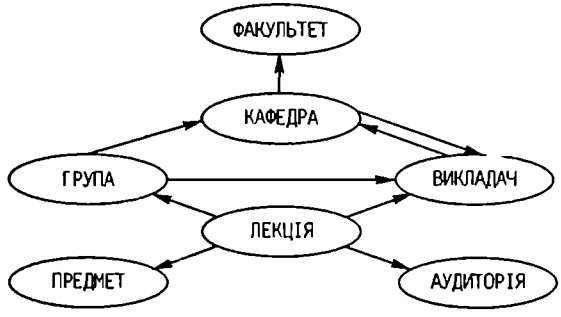



**Рис. 3.2.** Зв'язки між відношеннями в реляційній схемі вищого навчального закладу

Приклади згрупуємо за операціями, застосування яких у них демонструється.

### Проекція

**Запит 3.1**

Вивести список усіх викладачів вузу разом з їхніми телефонними номерами.

```
ВИКЛАДАЧ[Прізвище, Тел]
```

**Запит 3.2**

Вивести список усіх факультетів та відомості про те, хто є деканом кожного з них.

```
ФАКУЛЬТЕТ[Назва, Декан]
```

### Обмеження

**Запит 3.3**

Хто є деканом факультету інформатики?

```
(ФАКУЛЬТЕТ[Назва="інформатики"])[Декан]
```

Використання константи «інформатики» у наведеному прикладі операції θ-обмеження є відступом від теоретичної «чистоти» реляційної моделі, але розробники мов на основі реляційної алгебри використовували подібні відступи задля зручності користувача та практичної ефективності. Надалі ми будемо застосовувати подібні відступи, але вже без коментарів. Наведемо також теоретично «чистий» варіант згаданого вище запиту. З цією метою використаємо допоміжну таблицю ДОП(Назва) з єдиним кортежем інформатики.

```
(ФАКУЛЬТЕТ[Назва=Назва]ДОП)[Декан]
```

**Запит 3.4**

Вивести список усіх професорів навчального закладу.

```
(ВИКЛАДАЧ[Посада="професор"])[Прізвище]
```

### З'єднання, обмеження, проекція

**Запит 3.5**

Вивести назви факультетів разом із назвами відповідних кафедр.

Особливістю цього запиту порівняно з попередніми є те, що шукані значення розташовані в різних таблицях, тому операція з'єднання поєднує два операнди, а проекція виділяє два шукані стовпці.

```
(ФАКУЛЬТЕТ[#F=#F]КАФЕДРА)[ФАКУЛЬТЕТ.Назва, КАФЕДРА.Назва]
```

Відношення, отримане в результаті з'єднання таблиць ФАКУЛЬТЕТ та КАФЕДРА, має два атрибути Назва. Відтак, виконуючи проекцію, слід уточнити імена атрибутів іменами відношень.

**Запит 3.6**

Вивести список усіх кафедр факультету інформатики.

Необхідність використання операції з'єднання зумовлена тим, що аргумент пошуку (факультет інформатики) і поле результату перебувають в різних таблицях.

```
((ФАКУЛЬТЕТ[#F=#F]КАФЕДРА))[ФАКУЛЬТЕТ Назва="інформатики"])[КАФЕДРА.Назва]
```

**Запит 3.7**

Вивести список усіх викладачів кафедри АСУ разом із номерами їхніх телефонів.

```
((КАФЕДРА[#D=#D]ВИКЛАДАЧ)[КАФЕДРА.Назва="АСУ"])[Прізвище, Тел]
```

**Запит 3.8**

Вивести список усіх викладачів факультету інформатики разом з номерами їхніх телефонів.

```
(((ФАКУЛЬТЕТ[#F=#F]КАФЕДРА)[#D=#D]ВИКЛАДАЧ)[ФАКУЛЬТЕТ.Назва="інформатики"])
[Прізвище, Тел]
```

Для обчислення цього запиту спочатку здійснюється з'єднання трьох відношень (ФАКУЛЬТЕТ, КАФЕДРА, ВИКЛАДАЧ) за рівністю первинних і зовнішніх ключів, потім вибираються ті кортежі, які стосуються факультету інформатики, і нарешті здійснюється проекція за необхідними атрибутами.

**Запит 3.9**

Вивести список номерів усіх груп першого курсу кафедри АСУ.

```
((ГРУПА[#D=#D]КАФЕДРА)[Назва="АСУ", Курс=1])[Номер]
```

**Запит 3.10**

Цей запит є уточненням попереднього. Вивести список номерів усіх груп першого курсу кафедри АСУ разом із прізвищами кураторів цих груп.

```
(((ГРУПА[#D=#D]КАФЕДРА))[#КУРАТОР=#T]ВИКЛАДАЧ)[Назва="АСУ" & Курс=1])[Номер, Прізвище]
```

«Чистий» варіант для подібного запиту передбачає введення допоміжного відношення ДОП1(Назва,Курс) з єдиним кортежем "АСУ,1".

```
(((ГРУПА[#D=#D]КАФЕДРА))[#КУРАТОР=#T]ВИКЛАДАЧ)[Назва, Курс = Назва, Курс]ДОП1)
[Номер, Прізвище]
```

Зауважте, що у першому варіанті елементарні порівняння з'єднані логічною зв'язкою & (можливе також зв'язування за допомогою диз'юнкції ∨). Таке з'єднання логічних виразів підвищує гнучкість використання операцій θ-з'єднання і θ-обмеження.

**Запит 3.11**

Вивести список лекцій, на яких кількість студентів у групі перевищує кількість місць в аудиторії. У списку вказати номер аудиторії, номер групи, дисципліну, що читається, тиждень та день тижня.

```
((((ЛЕКЦІЯ[#G=#G]ГРУПА)[#S=#S]ПРЕДМЕТ)[#R=#R]АУДИТОРІЯ)
[ГРУПА.Кількість>АУДИТОРІЯ.Місткість]) [АУДИТОРІЯ.Номер, ГРУПА.Номер, ПРЕДМЕТ.Назва,
Тиждень, День]
```

Послідовність виконання операцій запиту можна зобразити графічно, як дерево, що його листки позначають відношення, а інші вершини — операції (рис. 3.3).

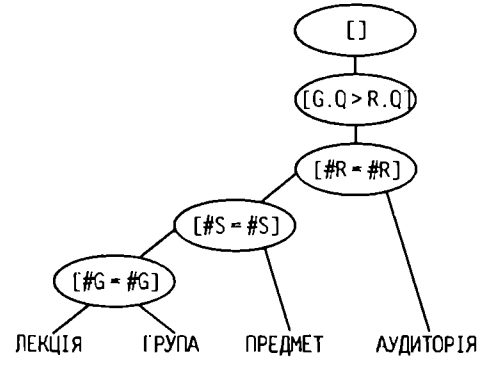



**Рис. 3.3.** Схема виконання запиту

Будуйте схему виконання запиту в тих випадках, коли виникають проблеми з його записом формальною мовою.

### Теоретико-множинні операції

**Запит 3.12**

Вивести список усіх корпусів, де розташовуються кафедри факультету інформатики, а також корпус, де розміщений деканат

```
(ФАКУЛЬТЕТ[Назва="інформатики"])[Корпус] ∪ ((ФАКУЛЬТЕТ [#F=#F]
КАФЕДРА)[ФАКУЛЬТЕТ.Назва="інформатики"])[КАФЕДРА.Корпус]
```

У першій частині запиту (до символу «∪») ми знаходимо номер корпусу, де розташовується деканат, тобто сам факультет інформатики, а в другій частині отримуємо перелік номерів корпусів, у яких розміщуються кафедри факультету. Операція об'єднання зводить в одну таблицю (без повторень) вміст двох сумісних між собою таблиць-операндів.

**Запит 3.13**

Вивести список усіх корпусів, де розташовуються кафедри факультету інформатики або факультету систем автоматизованого проектування, а також деканати цих факультетів.

```
(ФАКУЛЬТЕТ[Назва="інформатики"])[Корпус] ∪
((ФАКУЛЬТЕТ[#F=#F]КАФЕДРА)[ФАКУЛЬТЕТ.Назва="інформатики"])[КАФЕДРА.Корпус] ∪
(ФАКУЛЬТЕТ[Назва="САПР"])[Корпус] ∪
((ФАКУЛЬТЕТ[#F=#F]КАФЕДРА)[ФАКУЛЬТЕТ.Назва="САПР"]) [КАФЕДРА.Корпус]
```

**Запит 3.14**

Хто з викладачів кафедри АСУ не читає лекцій?

Цей запит доцільно будувати за методом «від супротивного», тобто від усієї множини викладачів (точніше, їхніх прізвищ) віднімається множина тих, які читають хоча б одну лекцію.

```
((КАФЕДРА[#D=#D]ВИКЛАДАЧ)[КАФЕДРА.Назва="АСУ"])[ВИКЛАДАЧ.Прізвище] -
((КАФЕДРА[#D=#D]ВИКЛАДАЧ )[#T=#T]ЛЕКЦІЯ)[КАФЕДРА.Назва="АСУ"])[ВИКЛАДАЧ.Прізвище]
```

### Операція ділення

**Запит 3.15**

Вивести номери тих викладачів, які викладають в усіх групах.

```
((ЛЕКЦІЯ[#T, #G])[#G ÷ #G]ГРУПА)[#T]
```

Необхідність використання операції ділення у цьому запиті зумовлена тим, що тут має місце множинне порівняння. Для кожного викладача (#T) потрібно сформувати його образ, тобто множину груп, у яких він викладає, а потім перевірити, чи збігається цей образ із множиною всіх груп, тобто з другим операндом. Зауважимо, що для дільника і діленого потрібно детально продумувати список атрибутів, оскільки це може вплинути на результат виконання запиту.

**Запит 3.16**

Цей запит є уточненням попереднього. Вивести номери тих викладачів, які викладають принаймні в усіх групах першого курсу.

```
((ЛЕКЦІЯ[#T, #G])[#G ÷ #G](ГРУПА[Курс=1])[#T]
```

Наявність у тексті запиту слова «принаймні» означає, що шукані викладачі можуть викладати ще в якихось групах. Якщо потрібно знайти коди викладачів, які викладають у всіх групах першого курсу і тільки в них, то від множини викладачів, знайдених за даним запитом, необхідно відняти викладачів, які викладають хоча б у одній групі не першого курсу, тобто таку множину:

```
((ЛЕКЦІЯ[#T,#G])[#G=#G]((ГРУПА[Курс≠1][#G]))[#T]
```

Подумайте, як би виглядав запит у термінах реляційної алгебри у випадку, коли замість «принаймні» стояло б слово «лише»

**Запит 3.17**

Цей запит є уточненням попереднього. Вивести номери тих викладачів, які викладають принаймні в усіх групах першого курсу кафедри АСУ

```
((ЛЕКЦІЯ[#T, #G])[#G ÷ #G]((ГРУПА[#D=#D]КАФЕДРА)[Назва="АСУ" & Курс=1]))[#T]
```

**Запит 3.18**

Цей запит є уточненням попереднього. Вивести прізвища тих викладачів, які викладають принаймні у всіх групах першого курсу кафедри АСУ.

```
(((ЛЕКЦІЯ[#T, #G])[#G ÷ #G]((ГРУПА[#D=#D]КАФЕДРА)[Назва="АСУ",
Курс=1]))[#T])[#T=#T]ВИКЛАДАЧ)[ВИКЛАДАЧ.Прізвище]
```

**Використання реляційних відношень-констант**

**Запит 3.19**

Вивести прізвища тих викладачів, у яких є лекції в усі робочі дні тижня.

Створимо реляційне відношення-константу

```
РОБОЧІ-ДНІ-ТИЖНЯ(День)("Пн", "Вт", "Ср", "Чт", "Пт")
```

Тоді запит матиме такий вигляд:

```
((ЛЕКЦІЯ[#T, День])[День ÷ День]РОБОЧІ-ДНІ-ТИЖНЯ)[#T=#T]ВИКЛАДАЧ)[ВИКЛАДАЧ.Прізвище]
```

## 3.2.3. Властивості операцій реляційної алгебри. Еквівалентні перетворення

Опишемо основні властивості операцій реляційної алгебри, на яких базуються правила еквівалентних перетворень її виразів. Два вирази реляційної алгебри називаються *еквівалентними*, якщо за будь-яких значень реляційних відношень, що входять до їхнього складу, результати обчислення виразів збігаються. Правила еквівалентних перетворень дають змогу вирішувати проблему оптимізації виконання запитів реляційної алгебри.

Наведемо основні властивості операцій реляційної алгебри.

1. Комутативність, асоціативність та дистрибутивність теоретико-множинних операцій об'єднання, перетину і різниці.
2. Ідемпотентність проекцій

   Нехай $L$ і $M$ – множини атрибутів деякого реляційного відношення $R$. Якщо $L \subseteq M$, то

   $$\pi_L(\pi_M R) = \pi_L R.$$

3. Дистрибутивність проекції з теоретико-множинними операціями, декартовим добутком, з'єднанням, селекцією.

   Нехай $K$ – певна підмножина атрибутів реляційного відношення $R$, $P$ – підмножина атрибутів реляційного відношення $S$ і $N = K \cup P$. Тоді:

   а) $\pi_N(R \cup S) = \pi_N R \cup \pi_N S$;
   б) $\pi_N(R \cap S) = \pi_N R \cap \pi_N S$;
   в) $\pi_N(R - S) = \pi_N R - \pi_N S$;
   г) $\pi_N(R \times S) = \pi_K R \times \pi_P S$,
   д) $\pi_N(R \overset{F}{\bowtie} S) = \pi_K R \overset{F}{\bowtie} \pi_P S$ (якщо в умові $F$ використовуються лише атрибути з множини $N$);
   е) $\pi_K \sigma_F R = \sigma_F \pi_K R$ (якщо в умові $F$ використовуються лише атрибути з множини $K$).

4. Ідемпотентність (комутативність) селекцій:

   $$\sigma_F(\sigma_G(R)) = \sigma_G(\sigma_F(R)) = \sigma_{F\&G}(R).$$

5. Комутативність селекції з декартовим добутком.

   а) $\sigma_F(R \times S) = \sigma_F(R) \times S$ (якщо в умові $F$ використовуються лише атрибути відношення $R$);
   б) $\sigma_{F_1\&F_2}(R \times S) = \sigma_{F_2}(\sigma_{F_1}(R \times S)) = \sigma_{F_1}(R) \times \sigma_{F_2}(S)$ (якщо всі атрибути з $F_1$ містяться у відношенні $R$ і всі атрибути з $F_2$ міститься у відношенні $S$);
   в) $\sigma_{F_1\&F_2\&F_3}(R \times S) = \sigma_{F_3}(\sigma_{F_1}(R) \times \sigma_{F_2}(S))$ (якщо всі атрибути з $F_1$ містяться у відношенні $R$ і всі атрибути з $F_2$ міститься у відношенні $S$).

6. Комбінування селекції з декартовим добутком та θ-з'єднанням:

   а) $\sigma_{K\theta P}(R \times S) = R \overset{K\theta P}{\bowtie} S$;
   б) $\sigma_G(R \overset{F}{\bowtie} S) = R \overset{G\theta F}{\bowtie} S$ (атрибути з $G$ належать відношенню $R$, атрибути з $F$ – відношенню $S$).

7. Дистрибутивність селекції з теоретико-множинними операціями.

   Нехай атрибути з умови $F$ входять до складу як реляційного відношення $R$, так і реляційного відношення $S$. Тоді:

   а) $\sigma_F(R \cup S) = \sigma_F(R) \cup \sigma_F(S)$;
   б) $\sigma_F(R \cap S) = \sigma_F(R) \cap \sigma_F(S)$;
   в) $\sigma_F(R - S) = \sigma_F(R) - \sigma_F(S)$.

8 Комутативність селекції і з'єднання:

а) якщо всі атрибути з логічного виразу $G$ містяться в реляційному відношенні $R$, то

$$\sigma_G(R \overset{F}{\bowtie} S) = \sigma_G(R) \overset{F}{\bowtie} S;$$

б) якщо $G = G_1 \& G_2$, всі атрибути з $G_1$ містяться у реляційному відношенні $R$ і всі атрибути з $G_2$ – у реляційному відношенні $S$, то

$$\sigma_G(R \overset{F}{\bowtie} S) = (\sigma_{G_1}(R) \overset{F}{\bowtie} \sigma_{G_2}(S)).$$

9. Комутативність та асоціативність добутку:

а) $R \times S = S \times R$ – комутативність;

б) $R \times (S \times T) = (R \times S) \times T$ – асоціативність.

10. Комутативність та асоціативність з'єднання:

а) $R \overset{F}{\bowtie} S = S \overset{F}{\bowtie} R$ – комутативність;

б) $R \overset{F}{\bowtie} (S \overset{G}{\bowtie} T) = (R \overset{F}{\bowtie} S) \overset{G}{\bowtie} T$ – асоціативність

11. Дистрибутивність з'єднання з теоретико-множинними операціями:

а) $R \overset{F}{\bowtie} (S \cup T) = (R \overset{F}{\bowtie} S) \cup (R \overset{F}{\bowtie} T)$;

б) $R \overset{F}{\bowtie} (S \cap T) = (R \overset{F}{\bowtie} S) \cap (R \overset{F}{\bowtie} T)$;

в) $R \overset{F}{\bowtie} (S - T) = (R \overset{F}{\bowtie} S) \quad (R \overset{F}{\bowtie} T)$.

## 3.2.4. Оптимізація обчислення виразів реляційної алгебри

Описані вище властивості операцій реляційної алгебри дають змогу вирішувати завдання логічної оптимізації алгебраїчних виразів. Під терміном *логічна оптимізація* ми маємо на увазі оптимізацію, що дає можливість прискорити обчислення реляційних виразів незалежно від способів реалізації реляційних відношень. На відміну від алгебри числових виразів, складність виконання формул реляційної алгебри залежить не лише від кількості операцій, але й від розміру операндів.

Розглянемо приклад оптимізаційних перетворень реляційного виразу. Нехай задано вираз:

$$\pi_A(\sigma_{B=C\&D=9}(R(A, B) \times S(C, D))).$$

Можливі шляхи послідовних еквівалентних перетворень, що оптимізують обчислення цього виразу, наведені на рис. 3.4. Оптимізаційні перетворення здійснюються згідно з властивостями, описаними в підрозділі 3.2.3.

Наведемо тепер основні правила оптимізації виразів реляційної алгебри.

1. Кожна селекція $\sigma_{F_1\&\ldots\&F_n}(E)$ згідно з властивістю 4 подається у вигляді послідовності селекцій $\sigma_{F_1}(\ldots(\sigma_{F_n}(E))\ldots)$.
2. Кожна селекція переміщується деревом виразу вниз наскільки це можливо (властивості 3е, 5, 7, 8). У такий спосіб зменшується кардинальність відношень

3. Розташовані поруч селекції і декартові добутки замінюються з'єднаннями, якщо це допускається властивістю 6.
4. Кожна проекція переміщується деревом виразу вниз наскільки це можливо (властивості 2, 3). У такий спосіб зменшується степінь відношень.
5. Кожен каскад селекцій і проекцій перетворюється на одиничну селекцію, одиничну проекцію чи селекцію з наступною проекцією (властивості 2, 3е, 4). Перетворення може суперечити евристиці «роби проекцію якомога раніше», однак ефективніше виконати всі можливі операції селекції і проекції за один перегляд відношення, ніж здійснювати кілька переглядів

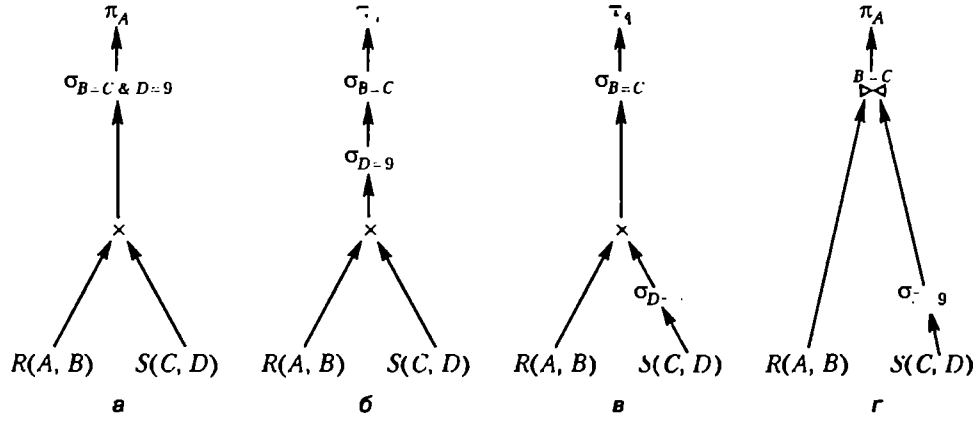



**Рис. 3.4.** Приклад еквівалентних перетворень, що оптимізують вирази реляційної алгебри: *а* — вихідний вираз обчислюється так, як він записаний; *б* — селекція поділяється на каскад селекцій відповідно до властивості 4; *в* — одна з отриманих селекцій опускається нижче декартового добутку відповідно до властивості 5а; *г* — декартів добуток і селекція замінюються на з'єднання згідно з властивістю 6.

## 3.3. Реляційне числення Кодда (зі змінними-кортежами)

Реляційне числення Кодда є одним із наріжних каменів у теорії реляційних баз даних. У СКБД, що існували до появи реляційного підходу, було багато засобів для обробки даних і формулювання запитів. Їхня розробка та впровадження скеровувались особливостями певної предметної області або кількох областей. У дореляційну епоху не було створено теоретичного критерію щодо повноти мовних засобів, призначених для формування запитів у межах певного класу мов. Ситуація видавалася досить складною: з одного боку множина запитів на основі природної мови (неформалізована область), з іншого — мовні засоби СКБД, тісно пов'язані з досить складними структурами даних.

Реляційне числення Кодда є аналогом математичної теорії — числення предикатів 1-го порядку — і є формальною семантичною основою для цілої родини мов програмування, точніше мов запитів. Принагідно зауважимо, що не кожна з широковживаних нині мов програмування має семантичну основу такого ж рівня

формальності. Основне завдання реляційного числення полягає у формулюванні спеціального числення предикатів, яке б інтерпретувалося реляційними відношеннями. Вузьким місцем цього підходу є те, що у традиційному численні предикатів 1-го порядку передбачається явне існування області інтерпретації *всіх* можливих змінних і тому немає жодних проблем з інтерпретацією формул з запереченнями чи з кванторами загальності. У реляційній моделі з такою інтерпретацією можуть бути проблеми. Наприклад, якщо схема містить елементарний предикат `Студент(x)`, то можуть виникнути ускладнення під час інтерпретації формули `¬Студент(x)` чи `∀x Студент(x)` скінченними реляційними відношеннями. Застосування формул подібного типу призводить до побудови нескінченної області інтерпретації предикатів (нескінченних відношень). Було навіть уведено поняття *безпечної формули* числення предикатів – формули, що інтерпретується скінченним відношенням.

Отже, з множини формул числення предикатів варто вибрати таку підмножину, під час інтерпретації якої не було б необхідності звертатися до області інтерпретації змінних і яка давала б змогу, крім того, одержувати формули, що інтерпретуються скінченними відношеннями. Перед реляційним численням ставилося завдання: при максимально спрощеному синтаксисі зберегти виразову потужність числення предикатів.

Відомі два реляційні числення: реляційне числення зі змінними-кортежами (Кодда) і реляційне числення зі змінними доменами (Піротта). Перше з них ми опишемо детально, а друге – оглядово.

## 3.3.1. Означення реляційного числення

Нехай задано множину *елементарних предикатних символів* $P_1$, $P_2$, ..., $Q_1$, $Q_2$, .... Ці предикати інтерпретуються скінченними реляційними відношеннями (або таблицями). Також задано множину *кортежних змінних* $r_1$, $r_2$, ..., $s_1$, $s_2$, .... Кортежні змінні інтерпретуються кортежами, або рядками таблиць. Вираз типу $r_i[j]$ називається *зрізом* змінної $r_i$ за $j$-м компонентом. Вирази типу $P_i \cdot r_j$ називаються *термами значень*. Такий вираз означає, що змінна $r_j$ набуває значень з відношення $R_i$, яке є інтерпретацією предиката $P_i$. За допомогою терма значень явно фіксується область значень змінної $r_j$. Вираз типу $r_i[m]\ \theta\ r_i[n]$ чи $r_i[m]\ \theta\ b$ (де $\theta$ один із бінарних предикатів порівняння, $r_i[m]$ – зріз, а $b$ - константа) називається *термом з'єднань*. Такі вирази визначають умови з'єднання різних відношень. Приклади термів з'єднань:

$$r_1[3] < r_2[1];\ \ r_1[2] = a;\ \ r_3[4] \neq r_3[1]$$

### Правильно побудовані формули

За допомогою п'яти аксіом визначимо поняття *правильно побудованої формули* (ППФ). Водночас введемо поняття *вільних* та *зв'язаних змінних*

1. Усі терми значень і терми з'єднань є ППФ. Усі їхні змінні - вільні.
2. Якщо $F$ – ППФ, то $\neg F$ – ППФ. Змінні, що були вільними в $F$, залишаються вільними в $\neg F$.

3. Якщо $F_1$ і $F_2$ - ППФ та їхні спільні змінні вільні у кожній з формул, то $F_1 \vee F_2$, $F_1 \& F_2$ – ППФ. Змінні будуть вільними у $F_1 \vee F_2$, $F_1 \& F_2$, якщо вони були вільними принаймні в одній із формул.
4. Якщо $F$ – ППФ і $r$ – вільна у ній змінна, то $\forall r(F)$, $\exists r(F)$ ППФ: а $r$ у пих формулах стає зв'язаною змінною.
5. Інших ППФ немає.

Приклади ППФ:

$$P_1 \cdot r_2 \vee (r_2[1] < r_3[2]);$$

$$\forall r_2 ((P_1 \cdot r_1 \& P_2 \cdot r_2) \vee (r[1] \neq r[2])).$$

**Правильно визначені формули**

За допомогою термів значень типу $P_i \cdot r_j$ задається область допустимих значень (область інтерпретації) кортежних змінних. У даному випадку це відношення, що зв'язується з предикатом $P_i$. Але визначити область значень змінної одним відношенням недостатньо. Її потрібно розширити настільки, наскільки це можливо в реляційній моделі. Це досягається завдяки поняттю *правильно визначеної формули* (ПВФ). Спочатку введемо поняття безкванторної ПВФ, а потім ПВФ з кванторами.

Безкванторна ППФ $F$ називається *правильно визначеною над змінною r*, якщо виконуються такі умови.

1. Єдиною змінною в $F$ є $r$.
2. Усі предикатні символи, що входять до складу формули, інтерпретуються сумісними відношеннями.
3. Усі терми в $F$ є термами значень.
4. Символ «¬» може бути розташований у $F$ лише після символу «&»

Приклади формул, правильно визначених над змінною $r$:

$$P_1 \cdot r \vee P_2 \cdot r \vee P_3 \cdot r \& P_4 \cdot r; \quad P_1 \cdot r \vee P_2 \cdot r \& \neg P_3 \cdot r.$$

Якщо предикати $P_1, P_2, P_3, P_4$ інтерпретуються таблицями $R_1, R_2, R_3, R_4$, то в першому прикладі змінна $r$ набуває значень з області $(R_1 \cup R_2 \cup R_3) \cap R_4$, а в другому – з області $(R_1 \cup R_2) - R_3$.

Тепер визначимо поняття ПВФ на кванторах. Нехай $F$ – формула, правильно визначена над змінною $r$, а $G$ – формула, що містить $r$ як вільну змінну, але не містить термів зиачень типу $P_i \cdot r$. Тоді формули:

$$\exists r(F \& G); \quad \forall r(F \supset G)$$

називаються *правильно визначеними на кванторах* існування й загальності. Тут символом «⊃» позначено операцію імплікації, тобто запис $F \supset G$ означає: «якщо формула $F$ є істинною, то і формула $G$ є істинною».

Формула $F$ специфікує ту область, у межах якої діє змінна, що використовується у кванторах. Тому формули $\exists r(F \& G)$ та $\forall r(F \supset G)$ будемо записувати так:

$$\exists F(G) \text{ і } \forall F(G).$$

### Формули з областю визначення

У поняття формули з областю визначення об'єднаємо формули, правильно визначені над кортежними змінними, з формулами, правильно визначеними на кванторах, а також з термами з'єднань. Формула $W$ називається *формулою з областю визначення*, якщо вона має вигляд

$$W = U_1 \ \& \ U_2 \ \&...\&U_n \ \& \ V, \ n \geq 1$$

і її складовим притаманні такі властивості.

1. Кожна формула $U_i$ є правильно визначеною над змінною $r_i$.
2. Формула $V$ або порожня, або задовольняє такі умови:
   а) у ній можуть бути тільки правильно визначені квантори;
   б) кожна вільна змінна з $V$ є вільною змінною принаймні в одній із формул $U_1, U_2, ..., U_n$;
   в) у формулі $V$ немає термів значень з вільними змінними.

Поняття формули з областю визначення можна інтерпретувати так. Якщо предикати $U_1, U_2, ...,U_n$ інтерпретуються відношеннями $S_1, S_2, ..., S_n, ...$, то конструкція $U_1 \ \& \ U_2 \ \&...\&U_n$ інтерпретується декартовим добутком $S_1 \times S_2 \times...\times S_n$.

Формула $V$ задає умову з'єднання відношень $S_1, S_2, ..., S_n, ....$ Зокрема, терми значень задають умови з'єднань на окремих атрибутах відношень, а квантори – умови кванторних зв'язувань.

Приклади формул з областю визначення наведено в табл. 3.3 (вважаємо, що предикати $P_1$, $P_2$, $P_3$ інтерпретуються відношеннями $R_1$, $R_2$, $R_3$).

**Таблиця 3.3.** *Приклади формул з областю визначення*

| Формула | Інтерпретація |
|---|---|
| $P_1 \cdot r_1 \ \& \ P_2 \cdot r_3 \ \& \ P_3 \cdot r_3$ | Декартів добуток відношень $R_1, R_2, R_3$ |
| $P_1 \cdot r_1 \ \& \ (r_1[3] = b)$ | Обмеження відношення $R_1$ умовою, що його третій стовпець має дорівнювати $b$ |
| $P_1 \cdot r_1 \ \& \ P_3 \cdot r_3 \ \& \ (r_1[2] = r_3[1])$ | З'єднання відношень $R_1$ і $R_3$ за умовою, що другий стовпець $R_1$ дорівнює першому стовпцю $R_3$ |
| $P_1 \cdot r_1 \ \& \ P_2 \cdot r_2 \ \& \ \forall P_3 \cdot r_3 (r_1[2] = r_3[1] \ \& \ r_1[3] = r_2[1])$ | Ділення відношення $R_1$ на $R_3$ за умовою $r_1[2] = r_3[1]$ і з'єднання результату ділення з відношенням $R_2$ за умовою $r_1[3] = r_2[1]$ |

### ALPHA-вирази

Вираз $(t_1, t_2, ..., t_k) : W$ називається *простим ALPHA-виразом*, якщо виконані такі умови:

- $W$ – формула з областю визначення,
- *цільовий список* $t_1, t_2, ..., t_k$ є послідовністю кортежних змінних і зрізів. Ці змінні мають бути вільними у $W$.

Довільний *ALPHA-вираз* визначається аксіоматично в такий спосіб:

- простий ALPHA-вираз є ALPHA-виразом;
- нехай $t = (t_1, t_2, ..., t_k)$. Якщо $t : W_1$ і $t : W_2$ – ALPHA-вирази, то вирази $t : W_1 \vee W_2$, $t : W_1 \,\&\, W_2$ і $t : W_1 \,\&\, \neg W_2$ теж є ALPHA-виразами;
- інших ALPHA-виразів немає.

## 3.3.2. Приклади використання реляційного числення

Використаємо реляційну схему з попереднього підрозділу і подібні приклади запитів Вкажемо імена предикатів, що відповідатимуть реляційним відношенням:

```
Fac – ФАКУЛЬТЕТ(#F, Назва, Декан, Корпус, Фонд)
Dep – КАФЕДРА(#D, #F, Назва, #ЗАВІДУВАЧ, Корпус, Фонд)
Tch – ВИКЛАДАЧ(#T, #D, Прізвище, Посада, Тел)
Grp – ГРУПА(#G, #D, Курс, Номер, Кількість, #КУРАТОР)
Sbj – ПРЕДМЕТ(#S, Назва)
Rom – АУДИТОРІЯ(#R, Номер, Корпус, Місця)
Lec – ЛЕКЦІЯ(#T, #G, #S, #R, Тип, День, Тиждень)
```

**ПРИМІТКА**

Для того щоб підвищити наочність формул, у зрізах змінної вигляду $r_i[j]$ замість номерів атрибутів будемо використовувати їхні імена.

На реляційному численні базується ціла родина мов запитів високого рівня. У подальших прикладах будемо застосовувати реляційне числення або одну з запропонованих Коддом мов – мову ALPHA – з дещо спрощеним синтаксисом.

**Запит 3.20**

Вивести список усіх викладачів вузу разом з їхніми телефонними номерами.

```
(r[3].r[5]): Tch.r
```

Так виглядає запит у термінах реляційного числення. `Tch.r` – це терм значення, що зв'язує кортежну змінну `r` з предикатом `Tch`, який, у свою чергу, представляє реляційне відношення ВИКЛАДАЧ. Якщо в цільовому списку використати імена атрибутів, а не їхні номери, то отримаємо такий вираз:

```
(r[І'мя].r[Тел]): Tch.r
```

У мовах програмування зазвичай спочатку оголошують змінну, а вже потім її застосовують, тому в мові ALPHA замість термів значень використовують спеціальний оператор `RANGE`, що оголошує змінну `r` до її використання у пошуковому операторі `GET`. Крім того, змінна оголошується безпосередньо за реляційним відношенням без участі унарного предиката:

```
RANGE ВИКЛАДАЧ r
GET W(r[І'мя].r[Тел]):
```

W специфікує таблицю результату або віртуальний пристрій, куди спрямовується результат.

За замовчуванням замість змінної r можна використати анонімну змінну, вказавши в тексті запиту ім'я відповідної таблиці. Остаточно запит виглядатиме так:

```
GET W(ВИКЛАДАЧ.Прізвище, ВИКЛАДАЧ.Тел):
```

Замість квадратних дужок тут використовується крапка.

Зауважимо, що в результаті за всіма кортежами початкової таблиці ВИКЛАДАЧ буде сформовано таблицю з двох стовпців (відсутність умови означає її істинність).

**Запит 3.21**

Хто є деканом факультету інформатики?

Реляційне числення:

```
(r[Декан]): Fac.r & (r[Назва]="інформатики")
```

Мова ALPHA:

```
GET W(ФАКУЛЬТЕТ.Декан):
(ФАКУЛЬТЕТ.Назва="інформатики")
```

За вказаною умовою буде відібрано один кортеж (припускається, що назви факультетів унікальні) і з нього буде отримано одне значення

**Запит 3.22**

Вивести список усіх викладачів кафедри АСУ разом із номерами їхніх телефонів

Реляційне числення:

```
(t[Прізвище], t[Тел]): Dep.d & Tch.t(d[#D]=t[#D] & d[Назва]="АСУ")
```

Мова ALPHA:

```
RANGE КАФЕДРА D
GET W(ВИКЛАДАЧ.Прізвище, ВИКЛАДАЧ.Тел):
∃D(D.#D=ВИКЛАДАЧ.#D & D.Назва="АСУ")
```

Цей запит дещо складніший за попередні. Підкванторна формула досить проста — це два терми з'єднання, перший з яких з'єднує два реляційні відношення, а другий — накладає умову на назву кафедри. Необхідність використання квантора зумовлена тим, що у цільовому списку специфіковані фрагменти кортежів лише однієї таблиці, тому друга є зв'язаною. Для того щоб визначити, який саме квантор слід застосувати, необхідно змінити текст запиту так, щоб квантор «зазвучав». Попередній запит може бути трансформований у такий спосіб:

«Знайти прізвища та телефони викладачів, для яких існує (як місце роботи) кафедра АСУ». Слово «існує» якраз і вказує на квантор існування.

**Запит 3.23**

Знайти прізвища викладачів, які викладають вищу математику.

Реляційне числення:

```
(t[Прізвище]): Tch.t & Sbj.s & Lec.l & (t[#T]=l[#T] & l[#S]=s[#S] & s[Назва]="Вища
математика")
```

Мова ALPHA:

```
RANGE ПРЕДМЕТ S
RANGE ЛЕКЦІЯ L

GET W(ВИКЛАДАЧ.Прізвище):
∃S(∃L((S.Назва="Вища математика") & (S.#S=L.#S) & (L.#T= ВИКЛАДАЧ.#T)))
```

Для того щоб написати такий запит, спочатку потрібно перекласти його початковий текст «квазіукраїнською» мовою, тобто так, щоб «зазвучали» квантори. Наприклад: «Знайти прізвища викладачів таких, що для них існує лекція така, що існує навчальний предмет з назвою «Вища математика», який викладається на цій лекції». Зауважимо, що порядок входження термів з'єднання до складу підкванторної формули не важливий. Не має значення також порядок розташування однотипних кванторів, тому їх можна не розділяти дужками. Якщо ж є обидва типи кванторів, то порядок їхнього запису впливає на результат.

**Запит 3.24**

Хто з викладачів кафедри АСУ не читає лекцій?

Реляційне числення:

```
(t[Прізвище]): ((Dep.d & Tch.t & (d[#D]=t[#D] &
d[Назва]="АСУ")) &¬(Dep.d & Tch.t & ∃Lec.l(l[#T]=t[#T] & d[#D]=t[#D]))
```

Мова ALPHA:

```
RANGE КАФЕДРА D
RANGE ЛЕКЦІЯ L

GET W(ВИКЛАДАЧ.Прізвище): ¬(∃D ∃L(D.#D=ВИКЛАДАЧ.#D & ВИКЛАДАЧ.#T=L.#T))
```

Пояснимо формулу, записану в термінах реляційного числення. Спочатку ми знаходимо всіх викладачів кафедри АСУ (формула зліва від &¬), потім – усіх викладачів, які мають хоча б одну лекцію. Логічна зв'язка &¬ інтерпретується як теоретико-множинна різниця цих двох множин. Зверніть увагу на конструкцію ∃Lec.l у другій частині виразу. Нам потрібно, щоб формули ліворуч і праворуч від &¬ мали однакові вільні змінні (тобто, щоб відношення, які їх інтерпретують, були сумісними), тому ми зв'язуємо змінну l квантором існування.

В ALPHA такий запит найпростіше формувати методом від супротивного; потрібно спочатку написати обернений запит («Хто з викладачів кафедри обчислювальної техніки читає лекції»), а потім перед формулою поставити знак заперечення.

**Запит 3.25**

Вивести прізвища тих викладачів, які викладають в усіх групах.

Реляційне числення:

```
(t[Прізвище]): Tch.t & ∀Grp.g ∃Lec.l(g[#G]=l[#G] &
l[#T]=t[#T])
```

Мова ALPHA:

```
RANGE ГРУПА G
RANGE ЛЕКЦІЯ L

GET W(ВИКЛАДАЧ.Прізвище):
∀G ∃L(G.#G=L.#G & L.#T=ВИКЛАДАЧ.#T)
```

Перекладаємо «квазіукраїнською» мовою: «Знайти прізвища таких викладачів, що для всіх (для кожної з) груп існує лекція, яку один із них читає».

Зауважимо, що останній запит належить до класу складних, оскільки під час його виконання необхідно порівнювати множину груп, у яких проводить заняття той чи інший викладач, з множиною всіх груп.

Розглянемо ще один запит з множинним порівнянням.

**Запит 3.26**

Знайти прізвища викладачів, які проводять заняття принаймні в усіх тих групах, де проводить заняття викладач Іванчук.

Для реалізації таких запитів доцільно декомпонувати їх на підзапити, реалізувати кожен підзапит окремо, а потім об'єднати окремі формули в одну.

**Підзапит 1.** Знайти коди груп, де проводить заняття викладач Іванчук.

```
∃L ∃T(T.Прізвище="Іванчук" & T.#T=L.#T & L.#G=G.#G)
```

Змінна G є вільною в даному підзапиті.

**Підзапит 2.** Знайти прізвища викладачів, які проводять заняття у підгрупі з заданим кодом.

```
∃L1(ВИКЛАДАЧ.#T=L1.#T & L1.#G=const)
```

Тепер поєднуємо підзапити, підставивши G.#G замість const, щоб ототожнити вхід другого підзапиту з виходом першого. За умови початкового запиту, якщо якась група належить множині, специфікованій першим підзапитом, то ця ж група має належати й множині результатів другого підзапиту (якщо ... то ...), тобто виконується операція імплікації (⊃).

Отже, складений запит на мові ALPHA виглядатиме так:

```
RANGE ГРУПА G
RANGE ВИКЛАДАЧ T
RANGE ЛЕКЦІЯ L, L1

GET W(ВИКЛАДАЧ.Прізвище): ∀G((∃L ∃T(T.Прізвище="Іванчук" & T.#T=L.#T & L.#G=G.#G)) ⊃
∃L1(ВИКЛАДАЧ.#T=L1.#T & L1.#G=G.#G))
```

Якби у початковому запиті замість слів «принаймні в усіх тих ...» стояло «лише у тих ...», то реалізація запиту була б майже такою самою, тільки символ імплікації був би спрямований у протилежний бік; якби стояли слова «ті й тільки ті ...», то замість імплікації потрібно було б застосувати операцію еквівалентності (~).

У даному прикладі виникла необхідність оголошувати більше однієї змінної для одного реляційного відношення; це пов'язано з тим, що змінні мають набувати значень незалежно одна від одної.

Розглянемо кілька прикладів, пов'язаних з використанням у мові ALPHA вбудованих функцій.

**Запит 3.27**

Визначити кількість кафедр на факультеті інформатики.

```
RANGE ФАКУЛЬТЕТ F

GET W(COUNT(КАФЕДРА.#D)):
∃F(F.Назва="інформатики" & F.#F=КАФЕДРА.#F)
```

Згідно з порядком обчислення такого запиту спочатку визначається множина кафедр на факультеті інформатики, а потім за допомогою функції COUNT підраховується їхня кількість.

**Запит 3.28**

Знайти кількість груп на факультеті інформатики та загальну кількість студентів у цих групах.

```
RANGE FACULTY F
RANGE DEPARTMENT D

GET W(COUNT(GROUP.#G), TOTAL(GROUP.Кількість)):
∃F∃D(F.Назва="інформатики" & F.#F=D.#F & GROUP.#D=D.#D)
```

Як і в попередньому прикладі, тут спочатку визначається певна множина кортежів, а потім до них застосовуються функції COUNT і TOTAL. COUNT підраховує кількість елементів у множині, а TOTAL підсумовує значення атрибута Кількість. Множина-аргумент для функції TOTAL має бути числовою, тобто на її елементах має бути визначеною операція додавання.

## 3.4. Реляційне числення Пірота (зі змінними доменами)

Як альтернативу реляційному численню зі змінними-кортежами А. Пірот запропонував *реляційне числення зі змінними доменами* (*P*-числення) і описав на його базі мову FQL. *P*-числення засновано на багатотиповому прикладному численні предикатів й інтерпретації виразів як доменів бази даних.

У *P*-численні кожному типу змінної відповідає певний домен. Передбачається, що в схемі бази даних визначені всі домени й описані зв'язки атрибутів реляційних відношень із доменами. Для кожного типу змінної визначається набір допустимих операцій, зокрема операції порівняння допускаються тільки для змінних одного типу, що дає змогу більш ефективно контролювати коректність запиту.

Прив'язування змінних до реляційної бази даних можна здійснити, використовуючи спеціальну елементарну формулу – *модельний предикат*. За його допомогою задається зв'язок змінної з атрибутом певного реляційного відношення та накладаються умови на належність значень змінної кортежам цього відношення.

У *P*-численні використовується поняття обмежених кванторів загальності й існування.

## 3.5. Реляційна повнота та селективна потужність

Однією з найважливіших характеристик будь-якої мови запитів (як частини мови маніпулювання даними) є її *селективна потужність*, тобто можливості вибирання даних за різними критеріями. Це поняття майже неможливо формалізувати. Мова з високою селективною потужністю дає змогу сформулювати більшість запитів (користуючись тільки власними засобами) у такий спосіб, що у відповідь на них видається лише релевантна інформація, тобто та й тільки та, яку можна використовувати без сторонніх доповнень чи урізань. Для мови з низькою селективною потужністю релевантні шукані дані видаються лише для запитів із простими критеріями відбору, а для більш складних випадків потрібно застосовувати додаткові сторонні засоби, наприклад програми, записані універсальною мовою, чи такі, що дають змогу візуально переглядати таблиці.

Селективну потужність різних мов запитів зазвичай оцінюють за допомогою певної мови-зразка. Таким зразком можуть бути різновиди реляційного числення чи реляційної алгебри. Для оцінки селективної потужності декларативних чи непроцедурних мов більш придатні мови-зразки на базі числення, алгебраїчні моделі зручніші для більш низькорівневих мов, зокрема таких, що допускають вбудовування однієї конструкції в іншу. Тому доцільним є використання мов-зразків обох типів. При цьому бажано, щоб обрані зразки мали однакову селективну потужність, тобто, щоб будь-яка формула числення мала реалізацію у вигляді скінченної послідовності операцій реляційної алгебри і, навпаки, щоб будь-яка скінченна послідовність операцій реляційної алгебри могла бути подана у вигляді формули реляційного числення.

Кодд увів поняття реляційно повної мови – мови, що дає змогу описати максимальну множину формул, які уможливлюють отримання наявної в базі даних інформації. Конструктивно визначити таку множину формул неможливо, тому потрібно використовувати її неформально, або ввести аксіоматично, що й зроблено в *тезі Кодда*.

### Теза Кодда

Реляційне числення Кодда є реляційно повним. (Сам Кодд називав це тезою про реляційну повноту). Мова називається *реляційно повною*, якщо для будь-якого скінченного набору реляційних відношень $R_1$, $R_2$, ..., $R_n$ вона дає змогу визначити будь-яке реляційне відношення, що може бути отримане з $R_1$, $R_2$, ..., $R_n$ за допомогою виразів реляційного числення Кодда.

Реляційна повнота реляційної алгебри обґрунтовується за допомогою запропонованого Коддом *алгоритму редукції*, який кожному виразу реляційного числення зіставляє скінченну послідовність операцій реляційної алгебри.

Визначення повноти селективних можливостей мов запитів у реляційній моделі дає критерій встановлення повноти будь-якої мови запитів. Кожна нова мова запитів реляційної моделі перевіряється на повноту. Природно, що такі мови можуть мати й інші можливості (наприклад, використання агрегатних функцій). Реляційна повнота – це необхідний мінімум можливостей мови.

## Контрольні запитання та завдання

1. Які є відмінності між математичним і реляційним відношеннями?
2. Що таке ключ у реляційному відношенні?
3. Чому операції реляційної алгебри можуть вкладатися одна в одну? Що це дає?
4. Що називається реляційною структурою даних?
5. Нехай задано відношення $R_1$ і $R_2$, де $R_1$ містить $n_1$ кортежів, а $R_2$ – $n_2$, причому $n_2 > n_1 > 0$. Вкажіть мінімально й максимально можливу кількість кортежів у відношеннях, одержаних у результаті виконання операцій реляційної алгебри, що наведені нижче. Якими мають бути схеми відношень $R_1$ і $R_2$?

$$R_1 \cup R_2,\ R_1 \cap R_2,\ R_1 - R_2,\ R_1 \times R_2,\ R_1[A],\ R_1[A = a],\ R_1[A \div \mathrm{B}]R_2.$$

6. Щодо розглянутої в розділі реляційної схеми для бази даних вищого навчального закладу сформулюйте такі запити мовою реляційної алгебри.
   а) Хто з деканів навчального закладу не є завідувачем кафедрою?
   б) Хто з викладачів факультету інформатики не є куратором?
   в) Хто з викладачів викладає всі типи лекцій (використайте відношення-константу)?
   г) Хто з викладачів читає лекції принаймні з усіх тих предметів, що й викладач Іванчук?
   д) Хто з викладачів читає лекції з тих і тільки тих предметів, що й викладач Іванчук?
   е) Хто з викладачів читає лекції лише з тих предметів, що й викладач Іванчук?
   є) Вивести назви кафедр, викладачі яких проводять лекції в аудиторії 307, разом із тижнем та днем тижня, коли ці лекції відбуваються.
7. Опишіть основні властивості операцій реляційної алгебри.
8. У чому полягає сутність еквівалентних перетворень реляційних виразів?
9. Які є правила оптимізаційних перетворень виразів реляційної алгебри?
10. Наведіть графічні приклади еквівалентних перетворень запитів.
11. Що таке безпечна формула?
12. Чим реляційне числення зі змінними-кортежами відрізняється від числення предикатів?
13. Дайте визначення таким основним поняттям реляційного числення Кодда:
    а) терм значення та терм з'єднання;
    б) формула, правильно визначена над кортежною змінною;
    в) формула, правильно визначена на кванторах;
    г) формула з областю визначення;
    д) ALPHA-вираз.

14. Запишіть такі запити в реляційному численні Кодда.
    а) На якому факультеті працює Іванчук?
    б) Вивести прізвище куратора групи 407.
    в) Хто очолює кафедру, на якій викладає Петренко?
    г) Вивести список імен усіх викладачів, які читають лекції з дисципліни «БД».
    д) Які викладачі читають лекції в аудиторії 205 і які саме дисципліни в цій аудиторії ними читаються?
    е) Знайти назви дисциплін, що читаються всіма тими викладачами, які мають лекції з дисципліни «БД».

# Розділ 4

# Мова SQL

- Засоби пошуку даних
- Засоби маніпулювання даними
- Операції над схемою бази даних
- Віртуальні таблиці та індекси
- Транзакції
- Додаткові можливості мови

## 4.1. Історія мови SQL та огляд її можливостей

Історія SQL починається з 70-х років XX століття, коли в дослідницькій лабораторії IBM у штаті Каліфорнія було розроблено першу версію цієї мови. Назва SQL є абревіатурою від Structured Query Language (структурована мова запитів), й іноді її вимовляють як «sequel» (первісна назва). Спочатку ця мова була реалізована в реляційній СКБД DB2 виробництва IBM. На відміну від мов третього покоління (COBOL, C), які з'явилися в той самий час, мова SQL не є процедурною. Непроцедурна мова – це мова, в якій описується, що потрібно одержати, а не як це зробити.

Особливість реляційних СКБД полягає у тому, що вони надають множинно-орієнтовану мову маніпулювання базами даних, тобто результатом дії мовного оператора є таблиця, яка містить множину даних. Більшість сучасних реляційних СКБД використовують саме мову SQL.

Американський інститут національних стандартів (American National Standards Institute – ANSI) та Міжнародна організація стандартів (International Standards Organization – ISO) займаються описом і підтримкою стандартів цієї мови. Усі сучасні СКБД підтримують певний стандарт, проте є й відхилення, які в кожному конкретному випадку специфікуються в документації програмного продукту. Окрім того, у багатьох системах розроблено розширення SQL, що дають змогу використовувати мову запитів у середовищі програмування.

SQL надає такі можливості:

- створювати й видаляти таблиці бази даних, а також змінювати заголовки таблиць;
- вставляти, змінювати й видаляти рядки в таблицях;
- виконувати пошук даних у багатьох таблицях та впорядковувати результати цього пошуку;

- описувати процедури підтримки цілісності;
- визначати та змінювати інформацію про захист даних.

Керуючись стандартами ANSI-92 та ANSI-99, розглянемо можливості SQL на численних прикладах. Усі запити конструюватимуться для тієї ж бази даних, що була використана під час розгляду реляційної алгебри та реляційного числення:

```
ФАКУЛЬТЕТ(#F, Назва, Декан, Корпус, Фонд)
КАФЕДРА(#D, #F, Назва, #ЗАВІДУВАЧ, Корпус, Фонд)
ВИКЛАДАЧ(#T, #D, Прізвище, Посада, Тел)
ГРУПА(#G, #D, Курс, Номер, Кількість, #КУРАТОР)
ПРЕДМЕТ(#S, Назва)
АУДИТОРІЯ(#R, Номер, Корпус, Місткість)
ЛЕКЦІЯ(#T, #G, #S, #R, Тип, День, Тиждень)
```

# 4.2. Засоби пошуку даних

## 4.2.1. Основні конструкції мови, призначені для вибирання даних

Основна конструкція, призначена у мові SQL для вибирання даних, складається з фраз SELECT і FROM. Фраза FROM вказує, з якої таблиці потрібно вибрати дані, а фраза SELECT — які саме атрибути (стовпці) з цієї таблиці мають бути вибрані. Запит

```
SELECT   Назва
FROM     ФАКУЛЬТЕТ
```

здійснює виведення назв факультетів. Ці дві фрази обов'язково мають бути в будь-якому запиті.

### Виведення окремих стовпців

У фразі SELECT можна зазначати список імен стовпців. Передбачається, що результат виведення буде впорядкований за стовпцями відповідно до того, як розташовані імена у фразі:

```
SELECT   Номер, Курс, Кількість
FROM     ГРУПА
```

### Виведення всіх стовпців

Якщо необхідно вивести всі стовпці таблиці, то у фразі SELECT використовується символ *:

```
SELECT   *
FROM     КАФЕДРА
```

### Неповторювані рядки

Хоча в реляційних відношеннях не має бути повторюваних рядків (дублікатів), у SQL за замовчуванням встановлено, що всі дублікати рядків у таблиці-результаті виводяться. Щоб унаслідок виконання запиту одержати унікальні (неповторювані) значення, потрібно використовувати модифікатор DISTINCT (за замовчуванням застосовується модифікатор ALL). Наприклад, щоб отримати список усіх

типів лекцій, які читаються у вузі, і щоб кожен тип виводився лише один раз, потрібно записати:

```
SELECT DISTINCT   Тип
FROM              ЛЕКЦІЯ
```

Без модифікатора DISTINCT ми одержали б список із кількох сотень рядків (його довжина дорівнювала б кількості всіх лекцій у вузі).

Зазначимо, що весь запит можна розмістити в одному рядку.

### Перевизначення імен стовпців

Фраза SELECT надає можливість перевизначити імена стовпців кінцевої таблиці. Для цього після імені стовпця вихідної таблиці необхідно зазначити ім'я стовпця кінцевої таблиці (з використанням необов'язкової фрази AS). Наприклад, у наведеному нижче запиті перевизначаються імена обох стовпців:

```
SELECT   Назва AS Назва_факультету, Декан AS Декан_факультету
FROM     ФАКУЛЬТЕТ
```

### Умова вибирання

Для запису умови вибирання використовується фраза WHERE. У ній зазначено, якій умові мають відповідати вихідні дані. Алгоритм обробки запиту з фразою WHERE є таким:

- вибрати рядок із таблиці;
- перевірити його відповідність вказаній умові;
- якщо рядок відповідає умові, то вивести значення стовпців, вказаних у фразі SELECT.

Цей запит виводить список усіх професорів вузу:

```
SELECT   Прізвище
FROM     ВИКЛАДАЧ
WHERE    Посада = "професор"
```

## 4.2.2. Вирази, умови та оператори

У мові SQL існує багато різновидів *виразів*, у яких використовуються дані різних типів — рядки, числа, логічні значення. Конструкція SELECT...FROM...WHERE також є виразом, не кажучи вже про просте згадування імені стовпця у фразі SELECT.

*Умова* — це вираз, що повертає логічне значення — TRUE чи FALSE. Умовні вирази обов'язково використовуються у фразі WHERE, а також можуть застосовуватися в інших фразах, наприклад SELECT. Прикладом умовного виразу є конструкція Посада = <професор>.

*Оператори* — це конструкції, що використовуються у виразах для означення певних дій над даними. Є кілька типів операторів: арифметичні, логічні, теоретико-множинні, оператори порівняння, оператори над рядками. У конкретних системах списки цих операторів можуть відрізнятися, зокрема бути ширшими, ніж наведений далі список.

- Арифметичні оператори: +, -, *, /.
- Оператор зчеплення рядків ||.
- Теоретико-множинні оператори: UNION, INTERSECT, EXCEPT (або MINUS).
- Логічні оператори: AND, OR, NOT.
- Оператори порівняння: >, <, =, <>, >=, <=.

Оператори порівняння, їхнє призначення та приклади використання наведені в табл. 4.1.

**Таблиця 4.1.** Оператори порівняння

| Оператор | Призначення | Приклад |
|---|---|---|
| = | Перевірка на рівність | SELECT * FROM КАФЕДРА<br>WHERE Фонд = 1500 |
| !=, ^=, <>, ¬= | Перевірка на нерівність | SELECT * FROM КАФЕДРА<br>WHERE Фонд != 1500 |
| >, < | Перевірка, чи одне значення більше або менше іншого | SELECT * FROM КАФЕДРА<br>WHERE Фонд > 1500<br>SELECT * FROM КАФЕДРА<br>WHERE Фонд < 1500 |
| >=, <= | Перевірка, чи одне значення не менше або не більше іншого | SELECT * FROM ФАКУЛЬТЕТ<br>WHERE Фонд >= 1500<br>SELECT * FROM ФАКУЛЬТЕТ<br>WHERE Фонд <= 1500 |
| IN | Перевірка на належність елементу множині | SELECT * FROM ВИКЛАДАЧ<br>WHERE Посада IN<br>("професор","доцент")<br>SELECT * FROM ВИКЛАДАЧ<br>WHERE Посада IN<br>(SELECT Посада<br>FROM ВИКЛАДАЧ<br>WHERE tel =<br>"5261815") |
| NOT IN | Еквівалентний != ALL. Перевірка на неналежність елементу множині | SELECT * FROM ВИКЛАДАЧ<br>WHERE Посада NOT IN<br>("професор","доцент")<br>SELECT * FROM ВИКЛАДАЧ<br>WHERE Посада NOT IN<br>(SELECT Посада<br>FROM ВИКЛАДАЧ<br>WHERE tel =<br>"5261815") |
| ANY, SOME | Використовуються разом із предикатом =, !=, >, <, <= або >=. Перевіряють, чи істинний цей предикат хоча б для одного елементу множини, заданої у правій частині оператора, відносно елементу, заданого в лівій частині | SELECT * FROM ФАКУЛЬТЕТ<br>WHERE Фонд = ANY<br>(SELECT Фонд<br>FROM ФАКУЛЬТЕТ<br>WHERE Корпус = 7) |

**Таблиця 4.1** (*продовження*)

| Оператор | Призначення | Приклад |
|---|---|---|
| ALL | Використовується разом з предикатом =, !=, >, <, <= або >=. Перевіряє, чи істинний цей предикат для всіх елементів множини, заданої у правій частині оператора, відносно елементу, заданого в лівій частині | SELECT * FROM ФАКУЛЬТЕТ WHERE Фонд >= ALL(1400,3000) |
| EXISTS | Повертає TRUE, якщо підзапит, до якого застосовується оператор, містить хоча б один рядок | SELECT Назва FROM Факультет WHERE EXISTS (SELECT * FROM Кафедра WHERE Кафедра.#F = Факультет. #F) |
| IS [NOT] NULL | Перевірка на рівність значенню NULL | SELECT Назва FROM Кафедра WHERE Фонд IS NULL |

Розглянемо на прикладах використання цих операторів.

**Запит 4.1**

Визначити групи, де кількість студентів становить від 40 до 50 осіб, а також курси, які їм відповідають.

```
SELECT   Номер, Курс
FROM     ГРУПА
WHERE    Кількість >= 40 AND Кількість <= 50
```

**Запит 4.2**

Визначити прізвища й посади професорів та асистентів.

```
SELECT   Прізвище, Посада
FROM     ВИКЛАДАЧ
WHERE    Посада = "професор" OR Посада = "асистент"
```

## 4.2.3. Вибирання з кількох таблиць

Щоб вибрати дані з багатьох таблиць, необхідно перелічити їхні імена у фразі FROM. Якщо у фразі WHERE не зазначено умову з'єднання таблиць, то обчислюється *декартів добуток* усіх таблиць із фрази FROM. Наприклад, задано дві таблиці

*Table*1

| *ROW* | *REMARK* |
|---|---|
| $row_1$ | $table_1$ |
| $row_2$ | $table_1$ |
| $row_3$ | $table_1$ |

*Table*2

| *ROW* | *REMARK* |
|---|---|
| $row_1$ | $table_2$ |
| $row_2$ | $table_2$ |
| $row_3$ | $table_2$ |

Результатом виконання запиту

```
SELECT   *
FROM     TABLE1, TABLE2
```

буде таблиця

| *Table1.ROW* | *Table1.REMARK* | *Table2.ROW* | *Table2.REMARK* |
|---|---|---|---|
| $row_1$ | $table_1$ | $row_1$ | $table_2$ |
| $row_1$ | $table_1$ | $row_2$ | $table_2$ |
| $row_1$ | $table_1$ | $row_3$ | $table_2$ |
| $row_2$ | $table_1$ | $row_1$ | $table_2$ |
| $row_2$ | $table_1$ | $row_2$ | $table_2$ |
| $row_2$ | $table_1$ | $row_3$ | $table_2$ |
| $row_3$ | $table_1$ | $row_1$ | $table_2$ |
| $row_3$ | $table_1$ | $row_2$ | $table_2$ |
| $row_3$ | $table_1$ | $row_3$ | $table_2$ |

Як бачимо, кожен рядок першої таблиці з'єднався з кожним рядком другої. Зазвичай таблиці поєднуються за певної умови. Наприклад, для визначення назв факультетів та відповідних їм кафедр слід записати:

```
SELECT   ФАКУЛЬТЕТ.Назва, КАФЕДРА.Назва
FROM     ФАКУЛЬТЕТ, КАФЕДРА
WHERE    ФАКУЛЬТЕТ.#F = КАФЕДРА.#F
```

### Уточнення імен стовпців

У фразах SELECT і WHERE імена стовпців можна уточнювати іменами таблиць. Якщо в поєднуваних таблицях є стовпці, що мають однакові імена, то посилаючись на такий стовпець у запиті, його ім'я потрібно уточнювати іменем таблиці.

### Псевдоніми таблиць

Таблиці можуть мати довгі назви, тому мова надає можливість зв'язувати з кожною таблицею певний псевдонім і надалі посилатися на таблицю за ним. Зіставлення таблиці з псевдонімом здійснюється у фразі FROM. Наприклад, попередній запит можна записати в такий спосіб:

```
SELECT   f.Назва, d.Назва
FROM     ФАКУЛЬТЕТ f, КАФЕДРА d
WHERE    f.#F = d.#F
```

Фраза WHERE може використовуватися для зазначення способу з'єднання таблиць або умови відбору рядків до кінцевої таблиці. Зокрема, запит

```
SELECT   КАФЕДРА.Назва
FROM     КАФЕДРА, ВИКЛАДАЧ
WHERE    КАФЕДРА.#D = ВИКЛАДАЧ.#D AND
         ВИКЛАДАЧ.Прізвище = "Іванов"
```

виводить назву кафедри, де працює викладач Іванов. (Зауважимо, що пошук викладачів ведеться в усьому вузі, тому будуть знайдені всі кафедри, де працюють викладачі на прізвище Іванов). Зверніть увагу на те, що умова пошуку задається на одній таблиці, а результат виводиться з іншої.

### Стовпці, що обчислюються

У фразі SELECT можна сформувати новий стовпець. У ньому записуватимуться результати обчислення виразу, в якому використовуються значення з інших стовпців таблиць, що з'єднуються. Наведений нижче запит видає дані про всі кафедри факультету інформатики разом з їхніми фондами та інформацією про те, яку частку становить фонд кафедри від загального фонду факультету.

```
SELECT   d.Назва, d.Фонд, (d.Фонд / f.Фонд) * 100
FROM     ФАКУЛЬТЕТ f, КАФЕДРА d
WHERE    f.#F = d.#F AND F.Назва = "інформатики"
```

### Еквіз'єднання

Якщо таблиці з'єднуються за рівністю значень в одній чи кількох парах стовпців, до того ж із кожної таблиці вибираються всі стовпці, то таке з'єднання відповідає операції еквіз'єднання реляційної алгебри. Наприклад:

```
SELECT   f.*, d.*
FROM     ФАКУЛЬТЕТ f, КАФЕДРА d
WHERE    f.#F = d.#F
```

### Природне з'єднання

Операція *природного з'єднання* здійснюється з'єднанням двох чи кількох таблиць за рівністю значень в одній чи кількох парах стовпців із наступним видаленням повторюваних стовпців (необхідні стовпці вказуються у фразі SELECT). Наприклад:

```
SELECT   f.#F, f.Назва, f.Декан, f.Корпус, f.Фонд, d.#D, d.Назва, d.#ЗАВІДУВАЧ,
         d.Корпус, d.Фонд
FROM     ФАКУЛЬТЕТ f, КАФЕДРА d
WHERE    f.#F = d.#F
```

Хоча стовпці Назва, Корпус, Фонд є в обох таблицях, однак вони не розглядаються як повторювані, оскільки мають різне семантичне навантаження. Стовпцем, який повторюється, є той єдиний, за яким виконується з'єднання, — стовпець #F, що виконує роль зовнішнього ключа в таблиці КАФЕДРА.

### θ-з'єднання

Це з'єднання за будь-якою умовою. θ-з'єднання виконується не за первинним і зовнішнім ключами, а за іншими стовпцями. Раніше наведені різновиди з'єднання є окремими випадками θ-з'єднання.

### З'єднання таблиці зі своєю копією

Інколи необхідно з'єднати таблицю із самою собою. Для цього у фразі FROM потрібно двічі зазначити назву таблиці з різними псевдонімами, щоб можна було

на кожен її екземпляр посилатися окремо. Розглянемо такий приклад. Необхідно перевірити, чи є в таблиці ФАКУЛЬТЕТ пари рядків, де назви факультетів збігаються, а ключі #F різні.

```
SELECT   f1.Назва, f1.#F, f2.#F
FROM     ФАКУЛЬТЕТ f1, ФАКУЛЬТЕТ f2
WHERE    f1.Назва = f2.Назва AND f1.#F != f2.#F
```

З'єднання можна виконувати безпосередньо у фразі FROM, скориставшись одним із різновидів оператора JOIN (наприклад, F1 **INNER JOIN** F2 **ON** F1.F# = F2.F#). Слід пам'ятати, що за певних обставин потрібно саме виконувати JOIN-з'єднання, а не обчислювати декартів добуток таблиць. Наприклад, декартів добуток двох таблиць з десятками тисяч рядків у кожній обчислюватиметься протягом кількох годин, а обсяг кінцевої таблиці буде вимірюватися сотнями гігабайтів!

Розглянемо кілька запитів, що вже використовувались як приклади у розділі 3. Запити будемо записувати як мовою SQL, так і за допомогою реляційної алгебри.

**Запит 4.3**

Визначити всі кафедри факультету інформатики.

Мова SQL:

```
SELECT   КАФЕДРА.Назва
FROM     ФАКУЛЬТЕТ, КАФЕДРА
WHERE    ФАКУЛЬТЕТ.#F = КАФЕДРА.#F AND
         ФАКУЛЬТЕТ.Назва = "інформатики"
```

Реляційна алгебра:

```
((ФАКУЛЬТЕТ[#F = #F]КАФЕДРА)[ФАКУЛЬТЕТ.Назва = "інформатики"])[КАФЕДРА.Назва]
```

**Запит 4.4**

Визначити всіх викладачів кафедри АСУ та їхні телефонні номери.

Мова SQL:

```
SELECT   Прізвище, Тел
FROM     КАФЕДРА, ВИКЛАДАЧ
WHERE    КАФЕДРА.#D = ВИКЛАДАЧ.#D AND КАФЕДРА.Назва = "АСУ"
```

Реляційна алгебра:

```
((КАФЕДРА[#D = #D]ВИКЛАДАЧ)[КАФЕДРА.Назва = "АСУ"])[ВИКЛАДАЧ.Прізвище, Тел]
```

**Запит 4.5**

Вивести список усіх викладачів факультету інформатики разом з їхніми телефонними номерами.

Мова SQL:

```
SELECT   Прізвище, Тел
FROM     ФАКУЛЬТЕТ, КАФЕДРА, ВИКЛАДАЧ
WHERE    ФАКУЛЬТЕТ.#F = КАФЕДРА.#F AND КАФЕДРА.#D = ВИКЛАДАЧ.#D AND
         ФАКУЛЬТЕТ.Назва = "інформатики"
```

Реляційна алгебра:

```
(((ФАКУЛЬТЕТ[#F = #F]КАФЕДРА)[#D = #D]ВИКЛАДАЧ)[ФАКУЛЬТЕТ.Назва = "інформатики"])
[ВИКЛАДАЧ.Прізвище, Тел]
```

**Запит 4.6**

Визначити номери груп першого курсу кафедри АСУ.

Мова SQL:

```
SELECT   Номер
FROM     ГРУПА, КАФЕДРА
WHERE    ГРУПА.#D = КАФЕДРА.#D AND Назва = "АСУ" AND Курс = 1
```

Реляційна алгебра:

```
((ГРУПА[#D = #D]КАФЕДРА)[Назва = "АСУ" & Курс = 1])[Номер]
```

**Запит 4.7**

Визначити номери груп першого курсу кафедри АСУ та прізвища їхніх кураторів.

Мова SQL:

```
SELECT   Номер, Прізвище
FROM     ГРУПА, КАФЕДРА, ВИКЛАДАЧ
WHERE    ГРУПА.#D = КАФЕДРА.#D AND ГРУПА.#КУРАТОР = ВИКЛАДАЧ.#T AND Назва = "АСУ" AND
         Курс = 1
```

Реляційна алгебра:

```
(((ГРУПА[#D = #D]КАФЕДРА)[#КУРАТОР = #T]ВИКЛАДАЧ)[Назва = "АСУ" & Курс= 1])[Номер,
  Прізвище]
```

**Запит 4.8**

Визначити лекції, на яких кількість студентів у групі перевищує кількість місць в аудиторії. Вивести номери аудиторій та груп, назви дисциплін, що читаються, а також дні й тижні проведення лекцій.

Мова SQL:

```
SELECT   АУДИТОРІЯ.Номер, ГРУПА.Номер, ПРЕДМЕТ.Назва, Тиждень, День
FROM     ЛЕКЦІЯ, ГРУПА, ПРЕДМЕТ, АУДИТОРІЯ
WHERE    ЛЕКЦІЯ.#G = ГРУПА.#G AND
         ЛЕКЦІЯ.#S = ПРЕДМЕТ.#S AND
         ЛЕКЦІЯ.#R = АУДИТОРІЯ.#R AND
         ГРУПА.Кількість > АУДИТОРІЯ.Місткість
```

Реляційна алгебра:

```
((((ЛЕКЦІЯ[#G = #G]ГРУПА)[#S = #S]ПРЕДМЕТ)[#R = #R]АУДИТОРІЯ)[ГРУПА.Кількість >
АУДИТОРІЯ.Місткість])[АУДИТОРІЯ.Номер, ГРУПА.Номер, ПРЕДМЕТ.Назва, Тиждень, День]
```

## 4.2.4. Використання агрегатних функцій

SQL містить функції, що дають змогу обчислювати різні статистичні характеристики. Такі функції називаються *агрегатними*. Стандарт ANSI визначає певний набір функцій, однак у СКБД він зазвичай значно розширюється. Наприклад, діалект SQL, що використовується в СКБД Oracle, містить понад 120 функцій.

З усієї множини функцій SQL ми розглянемо лише ті, що перелічені в стандарті ANSI. А саме:

- COUNT – повертає кількість рядків у таблиці;
- SUM – повертає суму всіх значень у стовпці;
- AVG – повертає середнє арифметичне всіх значень у стовпці;
- MAX – повертає найбільше значення у стовпці;
- MIN – повертає найменше значення у стовпці.

Окрім COUNT(*), кожна з цих функцій оперує (як аргументом) сукупністю значень стовпця певної таблиці та повертає єдине значення. Для функцій SUM і AVG стовпець-аргумент повинен містити числові значення.

Слід зазначити, що у даному випадку стовпець-аргумент – це стовпець віртуальної таблиці, в якій можуть міститися дані не лише зі стовпця базової таблиці, але й отримані шляхом функціонального перетворення та/або зв'язування символами арифметичних операцій значень з одного або кількох стовпців. Вирази, що визначають стовпець такої таблиці, мають різний рівень складності, але не можуть містити інших SQL-функцій. Наприклад, вираз AVG(Ставка + Надбавка/2) є припустимим, а вираз AVG(SUM(...)) – ні. SQL-функції можуть входити до складу виразів, наприклад SUM(Фонд)/Count(*).

Аргументам усіх функцій, окрім COUNT(*), може передувати модифікатор DISTINCT (різний), що вказує на необхідність видалення дублікатів перед застосуванням функції. Функція COUNT(*) використовується для обчислення кількості всіх рядків таблиці з урахуванням дублікатів.

Агрегатні функції можна застосовувати лише у фразах SELECT та HAVING. Якщо в запиті немає фрази GROUP BY, то область дії агрегатної функції поширюється на все кінцеве реляційне відношення, а в разі її наявності – на створювані нею групи. Розглянемо використання цих функцій на прикладах.

**Запит 4.9**

Визначити кількість кафедр на факультеті інформатики.

```
SELECT   COUNT(*)
FROM     ФАКУЛЬТЕТ, КАФЕДРА
WHERE    ФАКУЛЬТЕТ.#F = КАФЕДРА.#F AND
         ФАКУЛЬТЕТ.Назва = "інформатики"
```

Результат цього запиту може виглядати так:

```
COUNT(*)
--------
4
```

**Запит 4.10**

Визначити кількість предметів, що читаються.

```
SELECT   COUNT(*) Number_Of_Subjects
FROM     ПРЕДМЕТ
```

Тепер результат виглядає так:

```
Number_Of_Subjects
------------------
217
```

**Запит 4.11**

Визначити місткість усіх аудиторій у корпусі 5.

```
SELECT   SUM(Місткість) Total_Number_Of_Seats_In_Building_5
FROM     АУДИТОРІЯ
WHERE    Корпус = 5
```

**Запит 4.12**

Визначити кількість студентів факультету інформатики.

```
SELECT   SUM(ГРУПА.Кількість) Number_Of_IT_Faculty_Students
FROM     ФАКУЛЬТЕТ, КАФЕДРА, ГРУПА
WHERE    ФАКУЛЬТЕТ.#F = КАФЕДРА.#F AND
         КАФЕДРА.#D = ГРУПА.#D AND
         ФАКУЛЬТЕТ.Назва = "інформатики"
```

**Запит 4.13**

Визначити найбільший фонд із фінансування серед кафедр факультету інформатики.

```
SELECT   MAX(КАФЕДРА.Фонд) MAX_Department_Fund_In_IT_Faculty
FROM     ФАКУЛЬТЕТ, КАФЕДРА
WHERE    ФАКУЛЬТЕТ.#F = КАФЕДРА.#F AND
         ФАКУЛЬТЕТ.Назва = "інформатики"
```

Можна задавати кілька агрегатних функцій, що використовуються у виразах.

**Запит 4.14**

Визначити місткість усіх аудиторій корпусу 5, кількість місць у найменшій та найбільшій аудиторіях, середню місткість.

```
SELECT   SUM(Місткість) Total_Number_Of_Seats_In_Building_5,
         MIN(Місткість) Seats_In_The_Smallest_Room,
         MAX(Місткість) Seats_In_The_Largest_Room,
         SUM(Місткість)/COUNT(*) Average_Number_Of_Seats
FROM     АУДИТОРІЯ
WHERE    Корпус = 5
```

Якщо в запиті немає фрази GROUP BY, то використання імен стовпців разом із агрегатними функціями у фразі SELECT є неприпустимим.

Наприклад, у результаті виконання запиту

```
SELECT   Корпус, SUM(Місткість) Total_Number_Of_Seats_In_Building
FROM     АУДИТОРІЯ
```

може бути виведене повідомлення про помилку:

```
Dynamic SQL Error
-SQL error code = -104
-invalid column reference
```

Агрегатні функції не використовуються у фразі WHERE. В результаті виконання запиту

```
SELECT   Назва
FROM     КАФЕДРА
WHERE    Фонд = MAX(Фонд)
```

може бути виведене повідомлення про помилку:

```
ERROR at line 3:
ORA-00934: group function is not allowed here
```

### Невизначені значення в агрегатних функціях

У стовпці-аргументі агрегатної функції перед застосуванням будь-якої функції, окрім COUNT(*), виключаються всі невизначені значення. Сформулювавши запит

```
SELECT   COUNT(Корпус), COUNT(DISTINCT Корпус), COUNT(*)
FROM     ФАКУЛЬТЕТ
```

ви можете отримати, наприклад, таку відповідь:

```
COUNT(Корпус)  COUNT(DISTINCT Корпус)  COUNT(*)
-------------  ----------------------  --------
15             11                      17
```

Результат виконання запиту слід інтерпретувати так: усього є 17 факультетів; для 15 з них задані значення корпусу (два факультети в полі корпусу містять NULL), і серед цих 15 значень корпусів лише 11 є різними, тобто неповторюваними.

Якщо виявляється, що аргумент – порожня множина, функція COUNT набуває значення 0, а інші – NULL.

## 4.2.5. Фраза GROUP BY. Групування таблиці за рядками

Фраза GROUP BY дає змогу поділити множину рядків, одержаних після застосування фрази WHERE (тобто після фільтрації), на групи за ознакою рівності значень в одному чи кількох стовпцях. У цьому випадку агрегатні функції, що використовуються у фразі SELECT, діють не в усьому кінцевому реляційному відношенні, а лише в межах кожної групи. За наявності фрази GROUP BY кожний вираз у списку фрази SELECT повинен набувати єдиного значення для всієї групи, тобто він може бути:

- константою;
- агрегатною функцією;
- таким самим виразом, що й у фразі GROUP BY;
- виразом, що побудований з перелічених вище виразів.

Розглянемо кілька прикладів.

**Запит 4.15**

Визначити кількість студентів на кожній кафедрі факультету інформатики.

```
SELECT   КАФЕДРА.Назва, SUM(ГРУПА.Кількість) Number_Of_Students_In_The_Departments
FROM     ФАКУЛЬТЕТ, КАФЕДРА, ГРУПА
WHERE    ФАКУЛЬТЕТ.#F = КАФЕДРА.#F AND
         КАФЕДРА.#D = ГРУПА.#D AND
         ФАКУЛЬТЕТ.Назва = "інформатики"
GROUP BY КАФЕДРА.Назва
```

**Запит 4.16**

Визначити кількість викладачів на кожному факультеті.

```
SELECT   ФАКУЛЬТЕТ.Назва, COUNT(*) Number_Of_Teachers_In_The_Faculty,
         SUM (ГРУПА.Кількість) Number_Of_Students_In_The_Departments
FROM     ФАКУЛЬТЕТ, КАФЕДРА, ВИКЛАДАЧ
WHERE    ФАКУЛЬТЕТ.#F = КАФЕДРА.#F AND
         КАФЕДРА.#D = ВИКЛАДАЧ.#D
GROUP BY ФАКУЛЬТЕТ.Назва
```

**Запит 4.17**

Для кожного викладача визначити кількість дисциплін, які він читає.

```
SELECT   ВИКЛАДАЧ.Прізвище, COUNT(DISTINCT #S) Number_Of_Subjects
FROM     ВИКЛАДАЧ, ЛЕКЦІЯ
WHERE    ВИКЛАДАЧ.#T = ЛЕКЦІЯ.#T
GROUP BY ВИКЛАДАЧ.Прізвище
```

**Запит 4.18**

Отримати фонди факультетів з таблиці ФАКУЛЬТЕТ, а також обчислити їх як суми фондів кафедр.

```
SELECT   ФАКУЛЬТЕТ.Назва, ФАКУЛЬТЕТ.Фонд, SUM(КАФЕДРА.Фонд) Departments_Funds
FROM     ФАКУЛЬТЕТ, КАФЕДРА
WHERE    ФАКУЛЬТЕТ.#F = КАФЕДРА.#F
GROUP BY ФАКУЛЬТЕТ.Назва, ФАКУЛЬТЕТ.Фонд
```

## 4.2.6. Фраза HAVING. Умова вибирання для груп рядків

Фраза HAVING має таке ж значення для груп, що й WHERE для рядків усієї таблиці: вона дає змогу задавати умови для груп рядків, сформованих фразою GROUP BY. Фраза HAVING може використовуватися лише за наявності фрази GROUP BY; вирази в HAVING повинні набувати єдиного значення для групи. В умовах, що формулюються у фразі HAVING, можна використовувати агрегатні функції, які діють у межах створюваних груп, а також стовпці, за якими виконується групування. Розглянемо приклади.

**Запит 4.19**

Визначити факультети, де власний фонд перевищує сумарний фонд усіх кафедр.

```
SELECT   ФАКУЛЬТЕТ.Назва
FROM     ФАКУЛЬТЕТ, КАФЕДРА
WHERE    ФАКУЛЬТЕТ.#F = КАФЕДРА.#F
GROUP BY ФАКУЛЬТЕТ.Назва, ФАКУЛЬТЕТ.Фонд
HAVING   ФАКУЛЬТЕТ.Фонд > SUM(КАФЕДРА.Фонд)
```

**Запит 4.20**

Визначити факультети, в яких власний фонд перевищує сумарний фонд усіх кафедр на 2000

```
SELECT   ФАКУЛЬТЕТ.Назва
FROM     ФАКУЛЬТЕТ, КАФЕДРА
WHERE    ФАКУЛЬТЕТ.#F = КАФЕДРА.#F
GROUP BY ФАКУЛЬТЕТ.Назва, ФАКУЛЬТЕТ.Фонд
HAVING   (ФАКУЛЬТЕТ.Фонд - SUM(КАФЕДРА.Фонд)) > 2000
```

## 4.2.7. Фраза ORDER BY. Впорядкування рядків

У фразі ORDER BY перелічуються стовпці кінцевої таблиці, за значеннями яких слід відсортувати її рядки, а також встановити порядок цього сортування: за зростанням – ASC чи спаданням – DESC. Зауважимо, що впорядковувати таблиці можна за значеннями лише тих стовпців, які перелічені у фразі SELECT. За замовчуванням здійснюється впорядкування за зростанням.

**Запит 4.21**

Вивести прізвища викладачів у порядку спадання.

```
SELECT   Прізвище
FROM     ВИКЛАДАЧ
ORDER BY Прізвище DESC
```

**Запит 4.22**

Вивести в порядку зростання кількості викладачів на всіх кафедрах навчального закладу.

```
SELECT   КАФЕДРА.Назва, COUNT(*) Number_Of_Faculty_Teachers
FROM     КАФЕДРА, ВИКЛАДАЧ
WHERE    КАФЕДРА.#D = ВИКЛАДАЧ.#D
GROUP BY КАФЕДРА.Назва
ORDER BY Number_Of_Faculty_Teachers
```

## 4.2.8. Порядок обчислення запитів

У запитах фрази вживають у певному порядку: SELECT, FROM, WHERE, GROUP BY, HAVING, ORDER BY. Обчислення запиту здійснюється дещо в інший спосіб:

- обчислюється декартів добуток рядків з таблиць, зазначених у фразі FROM;
- до отриманої єдиної таблиці застосовуються умови з фрази WHERE, які формулюються так, що їхня істинність перевіряється на рядку єдиної таблиці;

- отримані рядки групуються відповідно до умови, записаної у фразі GROUP BY;
- до згрупованих рядків застосовуються умови, задані у фразі HAVING: вибираються лише ті групи, що відповідають цим умовам;
- рядки (групи) впорядковуються відповідно до фрази ORDER BY;
- виводяться зазначені у фразі SELECT стовпці чи вирази.

## 4.2.9. Підзапити

*Підзапит* — це запит, результат виконання якого є аргументом іншого запиту. Підзапити називають ще *вкладеними запитами*. Для того щоб вкласти підзапит у запит, потрібно зазначити його у фразі WHERE чи HAVING у правому аргументі одного з таких предикатів: IN, EXISTS =, <>, <, <=, >, >=. Існують прості (незалежні) й корельовані (залежні) вкладені підзапити. Інколи корельовані підзапити називають також інвертованими, підкреслюючи тим самим зворотний напрямок їхнього обчислення.

*Простий (незалежний) підзапит* — це підзапит, обчислення якого здійснюється незалежно від обчислення зовнішнього запиту. Такі підзапити обробляються «знизу вверх»: першим обробляється вкладений підзапит, а множина значень, отримана в результаті його виконання, використовується під час обробки зовнішнього запиту.

*Корельований (залежний, пов'язаний) підзапит* — це підзапит, обчислення якого залежить від процесу обчислення в зовнішньому запиті. Такі підзапити обробляються «зверху вниз»: спочатку вибирається поточний рядок із таблиці зовнішнього запиту й за значеннями його полів виконується обчислення підзапиту (тобто під час перевірки умов обчислення підзапиту використовуються значення полів із зовнішнього запиту). Далі перевіряється умова WHERE щодо входження поточного рядка зовнішнього запиту до результату.

### Повернення одного значення

У разі використання предикатів, що порівнюють два значення ( =, ≠, <, ≤, >, ≥), підзапит має повертати одне значення. У протилежному випадку видається повідомлення про помилку. Розглянемо приклади.

**Запит 4.23**

Визначити кафедри вузу, що розташовані в тому ж корпусі, що й кафедра АСУ.

```
SELECT   Назва
FROM     КАФЕДРА
WHERE    Корпус = (SELECT   Корпус
                   FROM     КАФЕДРА
                   WHERE    Назва = "АСУ")
```

Внутрішній запит визначає корпус кафедри АСУ; він використовується в предикаті порівняння (=) зовнішнього запиту.

У підзапиті можна застосовувати агрегатну функцію, що гарантує повернення єдиного значення.

**Запит 4.24**

Визначити факультети, фонд яких перевищує сумарний фонд усіх кафедр факультету інформатики.

```
SELECT   Назва
FROM     ФАКУЛЬТЕТ
WHERE    Фонд > (SELECT   SUM(КАФЕДРА.Фонд)
                 FROM     ФАКУЛЬТЕТ, КАФЕДРА
                 WHERE    ФАКУЛЬТЕТ.#F = КАФЕДРА.#F AND
                          ФАКУЛЬТЕТ.Назва = "інформатики")
```

Як уже зазначалося, підзапити можуть вкладатися у фразу HAVING. Прикладом є запит 4.25.

**Запит 4.25**

Визначити кафедри, де студентів навчається більше, ніж на кафедрі інженерії програмного забезпечення (ІПЗ).

```
SELECT   КАФЕДРА.Назва, SUM(ГРУПА.Кількість)
FROM     КАФЕДРА, ГРУПА
WHERE    КАФЕДРА.#D = ГРУПА.#D
GROUP BY КАФЕДРА.Назва
HAVING   SUM(ГРУПА.Кількість) >
         (SELECT   SUM(ГРУПА.Кількість)
          FROM     КАФЕДРА, ГРУПА
          WHERE    КАФЕДРА.#D = ГРУПА.#D AND
                   КАФЕДРА.Назва = "ІПЗ")
```

### Повернення багатьох значень

Якщо використовується предикат, що перевіряє належить (IN) чи ні (NOT IN) окреме значення множині, то підзапит може повертати множину значень. Розглянемо приклад.

**Запит 4.26**

Визначити, хто з викладачів факультету інформатики працює також на інших факультетах.

```
SELECT   ВИКЛАДАЧ.Прізвище
FROM     ФАКУЛЬТЕТ, КАФЕДРА, ВИКЛАДАЧ
WHERE    ФАКУЛЬТЕТ.#F = КАФЕДРА.#F AND
         КАФЕДРА.#D = ВИКЛАДАЧ.#D AND
         ФАКУЛЬТЕТ.Назва = "інформатики" AND
         ВИКЛАДАЧ.Прізвище IN
         (SELECT   ВИКЛАДАЧ.Прізвище
          FROM     ФАКУЛЬТЕТ, КАФЕДРА, ВИКЛАДАЧ
          WHERE    ФАКУЛЬТЕТ.#F = КАФЕДРА.#F AND
                   КАФЕДРА.#D = ВИКЛАДАЧ.#D AND
                   ФАКУЛЬТЕТ.Назва ≠ "інформатики")
```

### Корельованість (зв'язаність) підзапиту із запитом

У деяких випадках спосіб обчислення підзапиту залежить від значення поточного рядка зовнішнього запиту. Так, у підзапиті запиту 4.27 у фразі FROM згадується лише реляційне відношення КАФЕДРА, однак у фразі WHERE ми посилаємося на відношення ФАКУЛЬТЕТ. Це означає, що відбувається звернення до зовнішнього запиту. За наявності подібного звернення обчислення здійснюється в такий спосіб:

- фіксується поточний рядок зовнішнього реляційного відношення;
- для обчислення умови фрази WHERE виконується обчислення підзапиту, при цьому використовуються значення з поточного рядка зовнішнього запиту.

Обробка корельованого підзапиту має повторюватися для кожного з рядків таблиці, обчисленої у фразі FROM зовнішнього підзапиту, а не виконуватися лише раз, як у незв'язаному підзапиті. Розглянемо кілька прикладів корельованих підзапитів.

**Запит 4.27**

Визначити факультети, фонд кожного з яких менший, ніж сума фондів всіх його кафедр.

```
SELECT   Назва
FROM     ФАКУЛЬТЕТ
WHERE    Фонд < (SELECT   SUM(Фонд)
                 FROM     КАФЕДРА
                 WHERE    КАФЕДРА.#F = ФАКУЛЬТЕТ.#F)
```

**Запит 4.28**

Визначити викладачів, які не є кураторами груп.

```
SELECT   Прізвище
FROM     ВИКЛАДАЧ
WHERE    NOT EXISTS (SELECT   *
                     FROM     ГРУПА
                     WHERE    ВИКЛАДАЧ.#T = ГРУПА.#КУРАТОР)
```

## 4.2.10. Використання предикатів ANY, ALL, EXISTS та IN

### Предикати ANY та ALL

Ключові слова ANY та ALL розміщують після символів однієї з операцій порівняння =, ≠, >, <, <=, >=, щоб перевірити, чи є предикат істинним принаймні для одного (для всіх) значень множини, заданої в дужках після слова ANY (ALL), стосовно елементу, записаного зліва від символів порівняння. Розглянемо приклад.

**Запит 4.29**

Визначити кафедри, фонди яких більші, ніж хоча б у однієї з кафедр факультету інформатики.

```
SELECT   Назва
FROM     КАФЕДРА
WHERE    Фонд >
```

```
ANY (SELECT   КАФЕДРА.Фонд
     FROM     ФАКУЛЬТЕТ, КАФЕДРА
     WHERE    ФАКУЛЬТЕТ.#F = КАФЕДРА.#F AND
              ФАКУЛЬТЕТ.Назва = "інформатики")
```

Якби у формулюванні запиту замість слів «хоча б у однієї з» були вжиті слова «у всіх», то під час реалізації запиту мовою SQL замість слова ANY слід було б записати слово ALL.

### Предикат EXISTS

EXISTS є одноаргументним предикатом, що повертає значення TRUE, коли підзапит, до якого він застосовується, містить хоча б один рядок.

**Запит 4.30**

Визначити викладачів, які читають хоча б один курс лекцій.

```
SELECT   Прізвище
FROM     ВИКЛАДАЧ
WHERE    EXISTS (SELECT   *
                 FROM     ЛЕКЦІЯ
                 WHERE    ЛЕКЦІЯ.#T = ВИКЛАДАЧ.#T)
```

Якби нас цікавили викладачі, що не читають жодної лекції, то замість слова EXISTS слід було б записати NOT EXISTS.

### Предикат IN

Предикат IN перевіряє, чи належить елемент множині. Лівий операнд предиката має бути виразом, результат обчислення якого є окремим значенням (не множиною). Лівий операнд предиката IN може бути константою чи іменем поля. Правий операнд має специфікувати множину, він може бути SELECT-запитом або константою-множиною, що зображується взятим у дужки списком своїх елементів, – ("Іванов", "Петров", "Ігнатов"). Вираз x IN ("Іванов", "Петров", "Ігнатов") еквівалентний виразу x = "Іванов" OR x = "Петров" OR x = "Ігнатов". Якщо до предиката IN застосувати заперечення, він матиме вигляд NOT IN. Розглянемо приклади.

**Запит 4.31**

Визначити факультети, що розташовані в корпусах 1, 3, 5, 11.

```
SELECT   Назва
FROM     ФАКУЛЬТЕТ
WHERE    Корпус IN (1, 3, 5, 11)
```

**Запит 4.32**

Визначити факультети, які розташовані в тих самих корпусах, що й факультети інформатики або економіки.

```
SELECT   Назва
FROM     ФАКУЛЬТЕТ
WHERE    Корпус IN (SELECT   Корпус
                    FROM     ФАКУЛЬТЕТ
                    WHERE    Назва = "інформатики" OR
                             Назва = "економіки")
```

Якщо потрібно визначити факультети, розташовані не в тих корпусах, що факультети інформатики й економіки, у даному запиті замість предиката IN слід застосувати предикат NOT IN.

## 4.2.11. Використання теоретико-множинних операторів

У мові SQL означено три теоретико-множинні оператори – UNION, INTERSECT, EXCEPT, що дають змогу об'єднувати, перетинати й віднімати множини. Аргументами цих операторів є таблиці, що мають бути сумісними. Сумісність таблиць означає, що вони мають однакову кількість стовпців, і типи даних відповідних пар стовпців є сумісними. Розглянемо кілька прикладів.

**Запит 4.33**

Вивести прізвища викладачів, які мають лекції в понеділок і вівторок.

```
SELECT   Прізвище
FROM     ВИКЛАДАЧ, ЛЕКЦІЯ
WHERE    ВИКЛАДАЧ.#Т = ЛЕКЦІЯ.#Т AND
         ЛЕКЦІЯ.День = "понеділок"
INTERSECT
SELECT   Прізвище
FROM     ВИКЛАДАЧ, ЛЕКЦІЯ
WHERE    ВИКЛАДАЧ.#Т = ЛЕКЦІЯ.#Т AND
         ЛЕКЦІЯ.День = "вівторок"
```

Операцію перетину можна також зобразити за допомогою предиката IN.

```
SELECT   Прізвище
FROM     ВИКЛАДАЧ, ЛЕКЦІЯ
WHERE    ВИКЛАДАЧ.#Т = ЛЕКЦІЯ.#Т AND
         ЛЕКЦІЯ.День = "понеділок" AND
         Прізвище IN (SELECT   Прізвище
                      FROM     ВИКЛАДАЧ, ЛЕКЦІЯ
                      WHERE    ВИКЛАДАЧ.#Т =  ЛЕКЦІЯ.#Т AND
                               ЛЕКЦІЯ.День = "вівторок")
```

**Запит 4.34**

Визначити коди викладачів, що читають лекції з курсу «Бази даних», але не читають лекцій з курсу «Програмування».

```
SELECT   t.#T
FROM     ВИКЛАДАЧ t, ЛЕКЦІЯ l, ПРЕДМЕТ s
WHERE    t.#T = l.#T AND l.#S = s.#S AND
         s.Назва = "Бази даних"
EXCEPT
SELECT   t.#T
FROM     ВИКЛАДАЧ  t,  ЛЕКЦІЯ l, ПРЕДМЕТ s
WHERE    t.#T = l.#T AND l.#S = s.#S AND
         s.Назва = "Програмування"
```

Операція різниці може бути зображена також за допомогою предиката NOT IN.

```
SELECT   t.#T
FROM     ВИКЛАДАЧ t, ЛЕКЦІЯ l, ПРЕДМЕТ s
WHERE    t.#T = l.#T AND l.#S = s.#S AND
         s.Назва = "Бази даних" AND
         t.#T NOT IN
               (SELECT   t.#T
                FROM     ВИКЛАДАЧ t, ЛЕКЦІЯ l, ПРЕДМЕТ s
                WHERE    t.#T = l.#T AND l.#S = s.#S AND
                         s.Назва = "Програмування")
```

**Запит 4.35**

Визначити назви всіх факультетів та кафедр.

```
SELECT   Назва
FROM     ФАКУЛЬТЕТ
UNION
SELECT   Назва
FROM     КАФЕДРА
```

## 4.2.12. Запити, в яких реалізується квантор загальності

Стандартна SQL не містить предикатів, що перевіряють чи є одна множина підмножиною іншої. За допомогою цих предикатів можна було б легко записувати запити, еквівалентні тим запитам реляційного числення, в яких використовується квантор ∀. У SQL для реалізації таких запитів застосовуються інші засоби.

**Запит 4.36**

Визначити викладачів, що читають лекції в усіх групах.

Цей запит можна сформулювати так: «Визначити викладачів, для яких не існує таких груп, де вони не читають лекції». Перша частина запиту «Визначити викладачів, для яких не існує таких груп, де...» реалізується в такий спосіб:

```
SELECT   t.Прізвище
FROM     ВИКЛАДАЧ t
WHERE    NOT EXISTS (SELECT   *
                     FROM     ГРУПА g
                     WHERE...)
```

Тепер слід записати вираз, який містить умову щодо групи. Для цього сформулюємо допоміжний запит: «Визначити групи, де не веде занять викладач з кодом X». Він подібний до вихідного запиту (за винятком того, що у даному запиті зазначається конкретний викладач):

```
SELECT   *
FROM     ГРУПА g
WHERE    #G NOT IN (SELECT   #G
                    FROM     ЛЕКЦІЯ l
                    WHERE    l.#T = "X")
```

Об'єднаємо обидва запити, вклавши другий запит у перший у такий спосіб:

- фраза WHERE другого запиту замінює фразу WHERE вкладеного підзапиту першого запиту;
- у фразі WHERE другого запиту замість коду конкретного викладача (X) використовується код викладача з першого запиту (t.#T).

У результаті одержимо:

```
SELECT   t.Прізвище
FROM     ВИКЛАДАЧ t
WHERE    NOT EXISTS
         (SELECT   *
          FROM     ГРУПА g
          WHERE    #G NOT IN (SELECT   #G
                              FROM     ЛЕКЦІЯ l
                              WHERE    l.#T = t.#T))
```

### 4.2.13. Використання невизначених значень

Якщо під час введення даних не внести значення до поля таблиці, то СКБД розмістить там NULL-значення, або *невизначене значення*. Аналогічне значення може бути введене до поля таблиці під час виконання операції заміни даних. Мова SQL надає можливість маніпулювати невизначеними значеннями. При цьому NULL-значення не вважається рівним іншому null-значенню, тобто NULL = NULL не дорівнює TRUE (дорівнює логічному NULL), і NOT(NULL = NULL) також дорівнює логічному NULL. Незважаючи на це, два невизначені значення розглядаються як дублікати, коли необхідно видалити дублікати, і оператор SELECT DISTINCT дасть у результаті не більше одного NULL-значення. Для запису умов з невизначеними значеннями в SQL введені предикати IS та IS NOT, які можна використовувати з константою NULL. Наприклад, якщо потрібно підрахувати середню величину фонду кафедри за умови, що фонд задано, запит буде таким:

```
SELECT   AVG(Фонд)
FROM     КАФЕДРА
WHERE    Фонд IS NOT NULL
```

Коли необхідно визначити кафедри, для яких не вказано величину фонду, потрібно записати:

```
SELECT   Назва
FROM     КАФЕДРА
WHERE    Фонд IS NULL
```

## 4.3. Засоби маніпулювання даними

Ми розглядали, в який спосіб дані вибираються з БД. Далі ми обговоримо, як дані можна вводити до БД, редагувати та видаляти. Мова йтиме про оператори SQL, що дають змогу маніпулювати даними: INSERT, UPDATE і DELETE.

## 4.3.1. Додавання рядків до таблиці. Оператор INSERT

Оператор INSERT дає змогу вводити дані до БД. Є два різновиди цього оператора:

- INSERT...VALUES
- INSERT...SELECT

### Оператор INSERT...VALUES

Цей оператор дає можливість додати до таблиці один рядок і має такий формат:

```
INSERT INTO <ім'я таблиці> (<поле 1> [, <поле 2>]...)
VALUES (<значення 1> [, <значення 2>]...)
```

Виконання цього оператора потребує дотримання певних правил:

- типи даних, що вставляються, мають узгоджуватися з типами даних відповідних стовпців;
- розміри даних мають відповідати розмірам стовпців;
- порядок даних у фразі VALUES має відповідати порядку стовпців у фразі INSERT INTO.

Наведемо кілька прикладів.

**Запит 4.37**

Додати рядок до таблиці ФАКУЛЬТЕТ.

```
INSERT INTO ФАКУЛЬТЕТ (#F, Назва, Декан, Корпус, Фонд) VALUES (15, "інформатики",
"Сидоров", 5, 25000)
```

Список імен стовпців не є обов'язковим, якщо вставляються значення всіх стовпців. У цьому випадку порядок запису значень має збігатися з порядком стовпців у таблиці.

**Запит 4.38**

Додати рядок до таблиці КАФЕДРА.

```
INSERT INTO КАФЕДРА
VALUES (03, 15, "АСУ", "Петренко", "5", 5500)
```

Якщо значення стовпців не відомі, то слід використовувати NULL-значення.

**Запит 4.39**

Додати рядок до таблиці ВИКЛАДАЧ.

```
INSERT INTO ВИКЛАДАЧ
VALUES (173, 13, "Резніченко", NULL, "526-1В-15")
```

Багато СКБД дають змогу означувати стовпці із властивістю UNIQUE. В межах таблиці значення такого стовпця мають бути унікальними. Якщо це обмеження порушується, під час додавання рядків можуть виникнути помилки. Зазначимо, що стовпці із властивістю UNIQUE не можуть містити NULL-значень.

### Оператор INSERT...SELECT

Цей оператор дає змогу додати до таблиці множину рядків (результат виконання запиту) і, таким чином, дозволяє копіювати інформацію з однієї чи кількох таблиць до іншої. Часто за допомогою оператора INSERT...SELECT дані копіюються до похідних таблиць, що створюються з метою підвищення продуктивності виконання тих чи інших операцій над базою даних. Оператор має такий формат:

```
INSERT INTO <ім'я таблиці> (<список полів>)
SELECT      <список полів>
FROM        <ім'я таблиці>
WHERE       <умова пошуку>
```

Вихідні результати стандартного оператора SELECT є вхідними даними для оператора INSERT. Наведемо приклад.

**Запит 4.40**

Додати до таблиці ТИМЧАСОВА, що має стовпці Назва_факультету, Назва_кафедри, Прізвище_викладача, наявні в базі даних відомості про викладачів, а також про кафедри та факультети, де вони працюють.

```
INSERT INTO ТИМЧАСОВА (Назва_факультету, Назва_кафедри, Прізвище_викладача)
SELECT   ФАКУЛЬТЕТ.Назва, КАФЕДРА.Назва, ВИКЛАДАЧ.Прізвище
FROM     ФАКУЛЬТЕТ, КАФЕДРА, ВИКЛАДАЧ
WHERE    ФАКУЛЬТЕТ.#F = КАФЕДРА.#F AND
         КАФЕДРА.#D = ВИКЛАДАЧ.#D
```

Для оператора INSERT...SELECT мають виконуватися додаткові правила:

- оператор SELECT не може вибирати рядок із таблиці, до якої здійснюється додавання;
- кількість стовпців у фразі INSERT INTO має збігатися з кількістю стовпців у фразі SELECT;
- типи даних стовпців у фразі INSERT INTO мають збігатися з типами даних стовпців у фразі SELECT.

## 4.3.2. Оновлення даних. Оператор UPDATE

Оператор UPDATE надає можливість змінювати значення у наявних рядках. Його синтаксис такий:

```
UPDATE   <ім'я таблиці>
SET      <поле 1> = <вираз 1>
         [, <поле 2> = <вираз 2>]...
         [WHERE <умова пошуку>]
```

### Оновлення за умовою

Усі рядки таблиці, які задовольняють задану у фразі WHERE умову, змінюються згідно з фразою SET.

**Запит 4.41**

Встановити фонд факультету інформатики рівним 250 300.

```
UPDATE   ФАКУЛЬТЕТ
SET      Фонд = 250300
WHERE    Назва = "інформатики"
```

### Безумовне оновлення

Якщо фразу WHERE не задано, то оновлюються всі рядки.

**Запит 4.42**

Встановити фонд усіх факультетів рівним 260 500.

```
UPDATE   ФАКУЛЬТЕТ
SET      Фонд = 26050
```

### Неконстантне оновлення

Стовпцю може присвоюватися не константа, а вираз, що обчислюється на поточному рядку.

**Запит 4.43**

Збільшити всім факультетам фонд фінансування на 10 %.

```
UPDATE   ФАКУЛЬТЕТ
SET      Фонд = Фонд + Фонд/10
```

**Запит 4.44**

Збільшити всім кафедрам факультету інформатики фонд фінансування на 5 %.

```
UPDATE   КАФЕДРА
SET      Фонд = Фонд + Фонд/20
WHERE    #F IN (SELECT   #F
                FROM     ФАКУЛЬТЕТ
                WHERE    ФАКУЛЬТЕТ.Назва = "інформатики")
```

## 4.3.3. Видалення рядків таблиці. Оператор DELETE

Оператор DELETE дає змогу видаляти рядки таблиці й має такий синтаксис:

```
DELETE FROM <ім'я таблиці>
[WHERE умова]
```

Залежно від наявності та змісту фрази WHERE можна видалити один рядок, множину рядків, усі рядки або не видалити жодного.

Назвемо кілька особливостей використання оператора DELETE.

- Оператор не дає змоги видаляти окремі поля (використовуйте для цього оператор UPDATE), видаляючи рядок повністю.
- Застосування оператора DELETE, як і INSERT та UPDATE, може призвести до порушення цілісності бази даних.

- Якщо у фразі WHERE використовується вкладений підзапит, то у фразі FROM цього підзапиту не можна зазначати таблицю, з якої видаляються рядки. Це стосується також операторів INSERT та UPDATE.
- Оператор видаляє лише рядки таблиці, а не саму таблицю. Для видалення всієї таблиці слід застосувати оператор DROP TABLE.

Наведемо приклади.

**Запит 4.45**

Видалити відомості про лекції, що читаються в суботу та неділю.

```
DELETE FROM   ЛЕКЦІЯ
WHERE         День IN ("субота","неділя")
```

**Запит 4.46**

Видалити всі рядки з реляційного відношення ПРЕДМЕТ.

```
DELETE FROM   ПРЕДМЕТ
```

**Запит 4.47**

Видалити з таблиці ПРЕДМЕТ ті предмети, з яких не читається лекцій.

```
DELETE FROM   ПРЕДМЕТ
WHERE         #S != ALL (SELECT   #S
                         FROM     ЛЕКЦІЯ)
```

## 4.4. Операції над схемою бази даних

У цьому підрозділі ми вивчимо можливості мови SQL щодо створення баз даних і таблиць, а також розглянемо основні операції маніпулювання даними: зміна структури таблиці, додавання та видалення стовпців і рядків, видалення таблиці та бази даних.

### 4.4.1. Створення бази даних. Оператор CREATE DATABASE

Операція створення бази даних може бути або елементарною дією, або ж вимагати складних маніпуляцій, залежно від потреб користувача та обраної СКБД. Багато сучасних СКБД оснащені графічними засобами, що дають змогу визначати схему бази, проте універсальним способом створення бази даних є застосування операторів SQL. Синтаксис типового оператора створення бази даних є таким:

```
CREATE DATABASE  <ім'я бази даних>
```

Оскільки синтаксис цього оператора змінюється залежно від системи, ми його не будемо деталізувати. У більшості СКБД оператор CREATE DATABASE дає змогу виконати такі основні дії:

- делегувати повноваження на створення бази даних тому чи іншому користувачу;
- визначити місце (диск, каталог), де розташовуватиметься база даних;
- зарезервувати певний обсяг дискового простору для подальшого зберігання рядків таблиць та іншої інформації.

## 4.4.2. Створення таблиці. Оператор CREATE TABLE

Процедура створення таблиці є більш стандартизованою. Базовий синтаксис відповідного оператора є таким:

```
CREATE TABLE  <ім'я таблиці>
(<поле 1> <тип даних 1> [ NOT NULL ]
 [, <поле 2> <тип даних 2> [ NOT NULL ]]...)
```

Після ключових слів CREATE TABLE задаються ім'я таблиці та імена стовпців з типами даних і деякими іншими характеристиками. Специфікатор NOT NULL забороняє введення у стовпець NULL-значень і застосовується зокрема до ключових полів.

### Типи даних

Кожна система має свої типи даних. Наведемо типи даних у СКБД Oracle (табл. 4.2).

**Таблиця 4.2.** Типи даних у СКБД Oracle

| Тип даних | Коментарі |
|---|---|
| CHAR | Текстові рядки довжиною від 1 до 255 символів. Довжина рядків фіксована |
| VARCHAR2 | Текстові рядки довжиною від 1 до 2000 символів. Довжина рядків змінна |
| DATE | Дата: рік, місяць, день, година, хвилина, секунда |
| LONG | Рядок символьних даних змінної довжини до 2 Гбайт |
| LONG RAW | Двійкові дані довжиною до 2 Гбайт |
| NUMBER | Додатні чи від'ємні числа з фіксованою чи плаваючою комою |
| RAW | Двійкові дані довжиною до 255 байтів |
| ROWID | Шістнадцятковий рядок, що містить унікальну адресу рядка в таблиці |

### Унікальні значення

Певні системи надають можливість зазначати, що те чи інше поле має містити унікальні (не повторювані) значення. Це особливо важливо для ключових полів. Інші системи, наприклад Oracle та SQL Server, дають змогу оголошувати унікальні (UNIQUE) індекси. Є системи, в яких визначено тип даних, що його значення автоматично надаються створюваним рядкам. Ці значення є унікальними в межах таблиці протягом усього часу її існування. В Oracle таким типом даних є ROWID. Окрім того, в Oracle модифікатор UNIQUE застосовується для позначення полів, що разом забезпечують унікальність записів у межах таблиці.

**Запит 4.48**

Створити таблицю ФАКУЛЬТЕТ (застосовуємо типи даних Oracle).

```
CREATE TABLE ФАКУЛЬТЕТ
(#F NUMBER (10),
 Назва CHAR(50) NOT NULL,
 Декан CHAR(25),
 Корпус CHAR(5),
 Фонд NUMBER(6,2))
```

### Створення таблиці на базі існуючої

Для створення таблиць найчастіше використовують команду CREATE TABLE. Однак деякі системи надають альтернативний спосіб означення таблиць із використанням формату та даних наявної таблиці. Цей метод корисний для вибирання даних із таблиці з метою тимчасового зберігання та модифікації. В СКБД Oracle синтаксис цієї команди виглядає так:

```
CREATE TABLE  <нова таблиця>(<список полів>)
AS (SELECT    <список полів>
    FROM      <стара таблиця>
   [WHERE...])
```

Ця команда дає змогу створити нову таблицю з типами даних полів наявної таблиці та за необхідності перейменувати поля.

## 4.4.3. Модифікація таблиці. Оператор ALTER TABLE

Можливі ситуації, коли вже створена таблиця не відповідає зміненим вимогам до бази даних. Команда ALTER TABLE дає змогу змінити структуру таблиці після її створення: додати до вже визначеної таблиці стовпець або змінити наявний.

Синтаксис команди такий:

```
ALTER TABLE   <ім'я таблиці>
[ADD <поле> <означення поля>|
 ALTER <поле> <параметри>|
 DROP <поле> <параметри>]
```

## 4.4.4. Видалення таблиці. Оператор DROP TABLE

У SQL є команда, призначена для видалення з бази даних усієї таблиці. Оператор DROP TABLE видаляє таблицю з усіма зв'язаними з нею віртуальними таблицями й індексами. Найчастіше він застосовується до тимчасових таблиць, які видаляються через певний час після створення. Синтаксис команди такий:

```
DROP TABLE   <ім'я таблиці>
```

## 4.4.5. Видалення бази даних. Оператор DROP DATABASE

Команда DROP DATABASE застосовується для видалення всієї бази даних і має такий синтаксис:

```
DROP DATABASE   <ім'я бази даних>
```

# 4.5. Віртуальні таблиці та індекси

У цьому підрозділі ми розглянемо два засоби SQL, які дають змогу зображувати дані не в тому вигляді, в якому вони зберігаються в БД. Мова йтиме про віртуальні таблиці та індекси.

## 4.5.1. Використання віртуальних таблиць

*Віртуальна таблиця* – це пойменована таблиця, одержана в результаті виконання оператора SELECT з можливою заміною імен стовпців.

Віртуальна таблиця застосовується для створення складних запитів. Вона може використовуватися в операторах SELECT, INSERT, INPUT, UPDATE, DELETE, хоча фізично не займає місця в базі даних, як звичайна таблиця. Синтаксис створення віртуальної таблиці такий:

```
CREATE VIEW   <віртуальна таблиця> [(<список полів>)] AS
              SELECT   <список полів>
              FROM     <імена таблиць>
             [WHERE ...]
             [WITH CHECK OPTION]
```

Необов'язкова фраза WITH CHECK OPTION (з контролем) вказує на те, що для операцій INSERT та UPDATE над цією таблицею має здійснюватися контроль, який забезпечує виконання умов, заданих у фразі WHERE підзапиту.

### Проста віртуальна таблиця

Це віртуальна таблиця, що є копією вихідної, але має інше ім'я:

```
CREATE VIEW   КОПІЯ_ФАКУЛЬТЕТУ AS
SELECT   *
FROM     ФАКУЛЬТЕТ
```

У результаті виконання наведеного оператора створюється віртуальна таблиця КОПІЯ_ФАКУЛЬТЕТУ, що є точною копією таблиці ФАКУЛЬТЕТ. Можна також створювати віртуальні таблиці на базі інших віртуальних таблиць.

### Вибирання стовпців

До віртуальної таблиці можна копіювати окремі стовпці вихідної таблиці:

```
CREATE VIEW   ДЕКАНИ_ФАКУЛЬТЕТІВ (Назва, Декан) AS
SELECT   *
FROM     ФАКУЛЬТЕТ
```

### Перейменування стовпців

Синтаксис створення віртуальних таблиць дає змогу перейменовувати стовпці вихідної таблиці. Для цього необхідно явно зазначити стовпець у фразі SELECT та імена стовпців у фразі CREATE VIEW:

```
CREATE VIEW   ВИКЛАДАЧ_ПРІЗВИЩЕ_ПОСАДА (Прізвище, Положення) AS
SELECT   Прізвище, Посада
FROM     ВИКЛАДАЧ
```

### Складні конструкції

Фраза WHERE у складі оператора CREATE VIEW може містити підзапити. Наведемо приклад.

**Запит 4.49**

Створити віртуальну таблицю зі списком викладачів факультету інформатики, які працюють також на інших факультетах

```
CREATE VIEW   ВИКЛАДАЧ_2 (Прізвище, Посада) AS
SELECT        ВИКЛАДАЧ.Прізвище, Посада
FROM          ФАКУЛЬТЕТ, КАФЕДРА, ВИКЛАДАЧ
WHERE         ФАКУЛЬТЕТ.#F = КАФЕДРА.#F AND
              КАФЕДРА.#D = ВИКЛАДАЧ.#D AND
              ФАКУЛЬТЕТ.Назва = "інформатики" AND
              ВИКЛАДАЧ.Прізвище IN
                (SELECT   ВИКЛАДАЧ.Прізвище
                 FROM     ФАКУЛЬТЕТ, КАФЕДРА, ВИКЛАДАЧ
                 WHERE    ФАКУЛЬТЕТ.#F = КАФЕДРА.#F AND
                          КАФЕДРА.#D = ВИКЛАДАЧ.#D
                          ФАКУЛЬТЕТ.Назва ≠ "інформатики")
```

### Обмеження, накладені на структуру запиту в операторі CREATE VIEW

SQL накладає певні обмеження на структуру запиту, що використовується для означення віртуальної таблиці: в ньому забороняється застосовувати оператор UNION та фразу ORDER BY.

### Зміна даних через віртуальні таблиці

Віртуальні таблиці можна використовувати в операторах UPDATE, INSERT, DELETE для зміни даних у БД, але в цьому разі слід дотримуватися таких правил:

- забороняється застосовувати оператор DELETE до віртуальних таблиць, визначених на багатьох базових таблицях;
- директиву INSERT дозволяється використовувати лише в тому випадку, якщо віртуальна таблиця містить усі значення NOT NULL-стовпця базової таблиці;
- під час додавання чи оновлення через віртуальну таблицю всі записи, що оновлюються, мають належати одній і тій самій фізичній таблиці;
- забороняється додавати чи оновлювати записи через віртуальну таблицю, визначену із застосуванням модифікатора DISTINCT;
- не дозволяється оновлювати віртуальні стовпці (тобто стовпці, значення в яких є результатами обчислення виразів).

### Застосування віртуальних таблиць

Віртуальна таблиця може використовуватися для захисту, конвертування й забезпечення логічної незалежності даних, а також для спрощення складних запитів.

- **Забезпечення логічної незалежності.** Одне з основних завдань, вирішити яке дають змогу віртуальні таблиці, полягає в забезпеченні незалежності користувацьких програм від змін у логічній структурі бази даних, зумовлених розширенням таблиці та/або зміною розташування стовпців (наприклад, під час розщеплення таблиць).

- **Захист даних.** Надаючи користувачам доступ до певної інформації лише через віртуальні таблиці, можна вирішити проблему захисту даних.
- **Конвертування даних.** Віртуальні таблиці стають у нагоді, коли постає потреба надати користувачу дані не в тому форматі, в якому вони зберігаються в базі даних.
- **Спрощення конструкцій складних запитів.** Складний запит, що має ієрархічну структуру, легше сформулювати, використовуючи віртуальні таблиці як допоміжні.

### Видалення віртуальної таблиці

Для видалення віртуальної таблиці застосовується оператор DROP VIEW. Його синтаксис такий:

```
DROP VIEW  <ім'я віртуальної таблиці>
```

Слід пам'ятати, що під час видалення віртуальної таблиці видаляються також всі інші таблиці, у визначеннях яких дана таблиця використовувалась

## 4.5.2. Використання індексів

Одним зі способів подання даних не в тому порядку, в якому вони зберігаються, є використання індексів. Вони забезпечують:

- задоволення вимоги унікальності записів;
- підтримку логічної упорядкованості даних відповідно до значень одного чи кількох полів;
- оптимізацію виконання запитів.

З погляду користувача, індекс – це перелік стовпців таблиці, за значеннями яких записи логічно впорядковуються. З погляду СКБД, індекс – це механізм, що дає змогу значно підвищити швидкість доступу до записів за індексованими полями та забезпечує ефективну перевірку унікальності значень індексованих полів.

### Визначення індексу

Базовий синтаксис оператора визначення індексу є таким:

```
CREATE INDEX  <ім'я індексу>
ON  <ім'я таблиці> (<поле 1> [,<поле 2>]...)
```

У багатьох СКБД оператор визначення індексу доповнюється іншими конструкціями.

### Правила використання індексів

Використовуючи індекси, варто брати до уваги такі міркування.

- У таблицях невеликих розмірів індекси майже не забезпечують підвищення продуктивності.
- Продуктивність значно підвищується в тих випадках, коли стовпці містять переважно неповторювані дані чи багато NULL-значень.
- Завдяки індексам оптимізується виконання запитів, що видають невелику кількість рядків (до 25 %).

- Пам'ятайте, що індекси прискорюють пошук даних, однак сповільнюють процес їхнього оновлення, що стає особливо відчутним під час одночасного оновлення великої кількості рядків. У подібних випадках перед оновленням індекс потрібно видаляти, а після завершення даної операції — відновлювати.
- Зберігання індексів потребує значних обсягів пам'яті. Якщо СКБД дає змогу керувати пам'яттю, слід відвести частину пам'яті під індекси.
- Потрібно завжди індексувати поля, що використовуються для з'єднання таблиць. Це значно прискорює виконання запитів.
- Не слід індексувати поля, які регулярно оновлюються.
- Не бажано зберігати індекси разом із таблицями на одному фізичному пристрої. Розподіл цих об'єктів між носіями інформації знижує навантаження на них та прискорює виконання запитів.

### Складені індекси

SQL дає змогу створювати індекс за кількома полями. Наприклад. оператор

```
CREATE INDEX   КАФЕДРА_ЗАВІДУВАЧ_НАЗВА
ON   КАФЕДРА(#ЗАВІДУВАЧ, Назва)
```

створює індекс у таблиці КАФЕДРА за полями #ЗАВІДУВАЧ і Назва.

У складених індексах слід спочатку зазначати поля, що використовуються найчастіше. Складені індекси потрібно застосовувати тоді, коли зазначені в них поля використовуються для опису умови вибирання даних.

### Використання фрази UNIQUE

Фраза UNIQUE вказує, що значення індексу мають бути унікальними. Наприклад, оператор

```
CREATE UNIQUE INDEX   ГРУПА_ІД
ON   ГРУПА(#G)
```

вимагає, щоб у таблиці ГРУПА значення поля #G були неповторюваними.

### Порядок сортування полів у індексі

У деяких СКБД надається можливість зазначати порядок сортування полів, що індексуються. Наприклад, оператор

```
CREATE INDEX   ФАКУЛЬТЕТ_НАЗВА_ДЕКАН
ON   ФАКУЛЬТЕТ(Назва DESC, Декан)
```

вказує на необхідність індексування таблиці ФАКУЛЬТЕТ за стовпцем Назва у порядку спадання, а за стовпцем Декан — у порядку зростання. За замовчуванням сортування здійснюється в порядку зростання.

### Видалення індексу

Видалення індексу виконується командою DROP INDEX, що має такий синтаксис:

```
DROP INDEX   <ім'я індексу>
```

# 4.6. Транзакції

*Транзакція* – це сукупність операцій з маніпулювання базою даних, що мають розглядатися як атомарна дія. Під терміном «атомарна дія» ми маємо на увазі таку дію, що повністю виконується або скасовується як єдине ціле, тобто щодо неї підтримується принцип «усе або нічого». Різні СКБД мають свою специфіку щодо підтримки транзакцій. Тому ми використовуватимемо гіпотетичний узагальнений синтаксис таких операцій.

## 4.6.1. Початок і завершення транзакції

Транзакція активізується командою

```
BEGIN TRANSACTION   [<ім'я транзакції>]
```

і завершується командою

```
COMMIT [TRANSACTION]
```

за якою відбувається фактична зміна стану бази даних згідно з командами, що містяться в транзакції. Наприклад, якщо є така транзакція:

```
BEGIN TRANSACTION
   INSERT INTO   ФАКУЛЬТЕТ VALUES (1, "інформатики", "Іванов", "2/b", 42000)
   INSERT INTO   ФАКУЛЬТЕТ VALUES (1, "економіки", "Петров", "3/a", 47000)
COMMIT
```

то в таблицю ФАКУЛЬТЕТ будуть вставлені або два рядки, або жодного – залежно від успішності виконання всієї транзакції.

## 4.6.2. Скасування транзакції. Точки збереження

Після здійснення транзакції зазвичай надається можливість з'ясувати, чи успішним було її виконання. Транзакцію можна скасувати навіть у разі її успішного виконання, застосувавши команду ROLLBACK (але вона має бути виконана до команди COMMIT). У цьому випадку база даних набуває того ж вигляду, який вона мала до виконання транзакції. Можна також вказати так звані точки збереження (savepoint), або контрольні точки.

Загальний синтаксис команди RDLLBACK є таким:

```
ROLLBACK [TRANSACTION] [TO SAVEPOINT <ім'я точки збереження>]
```

Розглянемо приклад.

```
BEGIN TRANSACTION
   INSERT INTO   ФАКУЛЬТЕТ VALUES (1, "інформатики",
                           "Іванов", "2/b", 42000)
ROLLBACK TRANSACTION
   INSERT INTO   ФАКУЛЬТЕТ VALUES (1, "економіки",
                           "Петров", "3/a", 47000)
COMMIT
```

У таблицю ФАКУЛЬТЕТ буде вставлено лише відомості про факультет економіки, оскільки після додавання першого рядка відбувається скасування транзакції (додавання першого рядка анулюється).

Після застосування команди COMMIT усі дії, що є в транзакції, виконуються. Відміна транзакції призводить до скасування всіх дій, виконаних до моменту застосування команди ROLLBACK TRANSACTIDN.

За допомогою *точок збереження* можна скасовувати не всі виконані з початку транзакції дії, а лише їх частину. Синтаксис оголошення точки збереження є таким:

```
SAVEPOINT <ім'я точки збереження>
```

Розглянемо приклад.

```
BEGIN TRANSACTION
   UPDATE КАФЕДРА
      SET Фонд = Фонд + Фонд/10 WHERE Корпус = "2/3"
   SAVEPOINT save_it
   DELETE FROM КАФЕДРА WHERE Корпус = "2/3"
   ROLLBACK TO SAVEPOINT save_it
COMMIT
```

Видалення рядків не відбудеться, оскільки здійснюється повернення до точки збереження save_it, що означена до команди видалення.

## 4.7. Тригери

*Тригер* — це SQL-оператор, що активізується під час виконання певних операцій над об'єктами бази даних. Об'єктами бази даних є таблиці, а операціями — додавання, видалення та заміна рядків. Тригери — це один із механізмів підтримки цілісності бази даних.

У найпростішому випадку синтаксис оголошення тригера є таким:

```
CREATE TRIGGER <ім'я тригера>
   {BEFORE | AFTER} <операції над таблицею> [OF <список полів>] ON <ім'я таблиці>
   [WHEN (<умова>)]
<оператори SQL>
```

Якщо умова у фразі WHEN є істинною або ця фраза відсутня, до (BEFORE) або після (AFTER) виконання операції INSERT, UPDATE чи DELETE над таблицею, зазначеною після слова ON, буде виконано вказані нижче оператори SQL. Коли таких операторів кілька, їх слід помістити між ключовими словами BEGIN ATOMIC та END. Конструкція другого рядка означення тригера називається *реченням ініціювання*, WHEN — *умовою ініціювання*, а <оператори SQL> — *дією тригера*.

Розглянемо приклади застосування тригерів.

**Запит 4.50**

Після видалення інформації про кафедру видалити інформацію про всіх викладачів кафедри.

```
CREATE TRIGGER   Кафедрат_Видалення
AFTER DELETE ON  КАФЕДРА
```

```
DELETE FROM     ВИКЛАДАЧ
WHERE           ВИКЛАДАЧ.#D = КАФЕДРА.#D
```

**Запит 4.51**

Після видалення рядка з відомостями про викладача встановити значення NULL в полі #Куратор тих записів із таблиці ГРУПА, що відповідають групам, де згаданий викладач був куратором.

```
CREATE TRIGGER  Викладач_Видалення
AFTER DELETE ON ВИКЛАДАЧ
UPDATE          ГРУПА
SET             #Куратор = NULL
WHERE           ГРУПА.#T = ВИКЛАДАЧ.#T
```

### Доступ до старих і нових значень рядків

Якщо виконується оновлення рядків таблиці, то в тригері допускається звернення до старих і нових значень рядків, що оновлюються. Для звернення до нового (старого) рядка слід вказати кваліфікатор NEW (DLD) перед іменем стовпця. Такі кваліфікатори дозволяється використовувати як в умові тригера, так і в описі його дії. Розглянемо приклади.

**Запит 4.52**

Після додавання чи оновлення рядка в таблиці КАФЕДРА встановити значення поля Фонд у таблиці ФАКУЛЬТЕТ рівним сумі фондів усіх кафедр відповідного факультету.

```
CREATE TRIGGER   ФАКУЛЬТЕТ_Фонд
  AFTER INSERT OR UPDATE ON   КАФЕДРА

BEGIN
  UPDATE   ФАКУЛЬТЕТ
  SET      Фонд = SELECT   SUM(Фонд)
                  FROM     КАФЕДРА
                  WHERE    КАФЕДРА.#F = КАФЕДРА.NEW.#F AND
                           ФАКУЛЬТЕТ.#F =  КАФЕДРА.NEW.#F
END
```

### Тригери й транзакції

Дії, які виконуються під час основної операції та в тригері, становлять єдину транзакцію.

1. Перед виконанням додавання, оновлення та видалення неявно ініціюється команда BEGIN TRANSACTION.
2. Реалізується операція додавання/оновлення/видалення.
3. Ініціюється та виконується тригер;
4. Тригер скасовує транзакцію або за замовчуванням його дія завершується.

**Запит 4.53**

Дозволити додавання рядка з відомостями про кафедру лише в тому випадку, коли існує рядок із даними про факультет, якому належить кафедра (кафедри без факультету не буває).

```
CREATE TRIGGER   Кафедра_Додавання
BEFORE INSERT ON КАФЕДРА
WHEN NOT EXISTS (SELECT   *
                 FROM ФАКУЛЬТЕТ
                 WHERE ФАКУЛЬТЕТ.#F = КАФЕДРА.#F)
BEGIN
  ROLLBACK TRANSACTION
END
```

### Вкладеність тригерів

Тригери бувають вкладеними. Це означає, що маніпулювання рядком однієї таблиці може ініціювати тригер, який маніпулює рядками іншої таблиці. У свою чергу, маніпулювання рядками другої таблиці може ініціювати тригер, що маніпулює рядками третьої таблиці тощо. Вкладеність тригерів може призвести до «зациклення».

Обмеження на використання тригерів:

- тригери не визначаються для віртуальних таблиць;
- після видалення таблиці всі зв'язані з нею тригери також видаляються;
- тригери визначаються лише для створених таблиць.

В усіх наведених прикладах тригери використовувалися для підтримки цілісності бази даних. Але існує багато інших можливостей використання тригерів. Наведемо лише деякі з них:

- автоматичне обчислення значень похідних стовпців;
- запобігання виконанню недозволених транзакцій;
- виконання складних перевірок з метою захисту даних,
- підтримка складних обмежень цілісності;
- реєстрація подій, що відбуваються в базі даних;
- збирання статистики щодо доступу до таблиць бази даних.

## 4.8. Додаткові можливості

### Тимчасові таблиці

Деякі СКБД надають можливість створювати *тимчасові таблиці*, які є звичайними таблицями, що існують лише до завершення сеансу роботи користувача з базою даних. Як правило, такі таблиці входять до складу спеціальної службової бази даних, призначеної для їхнього зберігання. Синтаксис директив для роботи з тимчасовими таблицями аналогічний синтаксису відповідних команд для звичайних таблиць, за винятком того, що є конструкції, призначені для створення тимчасових таблиць. Тимчасові таблиці потрібні для зберігання тимчасових даних, які потрібні лише протягом поточного сеансу зв'язку із СКБД.

### Курсори

*Курсор* є покажчиком на окремий запис тієї чи іншої таблиці Курсори використовуються переважно в збережених процедурах, а також у програмах, що працюють з базою даних й ініціюють виконання операторів SQL. Використання курсорів дає можливість.

- вибрати певну сукупність даних (на зразок тимчасової таблиці, що містить результати виконання оператора SELECT);
- ініціювати послідовний перегляд сукупності записів;
- проаналізувати конкретний запис, на який вказує курсор;
- виконати зовнішню операцію над поточним записом перш ніж перейти до наступного.

Ще одним варіантом застосування курсору є тимчасове зберігання результатів запиту для подальшого використання. Якщо зовнішня програма вимагає багаторазового використання певного набору записів, то створюється курсор, яким оперують замість багаторазового виконання оператора SELECT.

Для використання курсору потрібно:

- створити курсор;
- відкрити курсор для використання в застосуванні чи збереженій процедурі;
- ініціювати за допомогою курсору послідовне перебирання записів з метою їхньої обробки;
- закрити курсор після завершення роботи з ним (з можливим наступним відкриттям у разі потреби);
- видалити курсор

На відміну від таблиць, індексів, тригерів та збережених процедур, курсори не є об'єктами бази даних.

### Збережені процедури

*Збережені процедури* – це об'єкти бази даних, які зазвичай містять велику кількість директив SQL. Такі процедури можуть містити сукупність команд (складних запитів, операцій оновлення, додавання та видалення), що часто використовуються як єдине ціле. Збережена процедура дає змогу звернутися до неї як до функції, замість того, щоб послідовно записувати команди SQL, які вона містить. Значне зниження навантаження на канали зв'язку під час роботи користувачів із сервером теж є важливою перевагою збережених процедур, адже замість окремих багаторазових звернень до бази даних виконується єдине звернення до збереженої процедури.

### Розширення SQL

Будь-яка СКБД розширює стандарт мови SQL, причому в багатьох випадках SQL трансформується в мову програмування, що надає усі можливості SQL. Наприклад, у Microsoft SQL Server таким розширенням є мова Transact-SQL, в Oracle –

мова PL/SQL. Кожна з них має переваги звичайних мов програмування, підтримує всі можливості стандарту ANSI SQL, а також розширює його.

Крім того, багато СКБД надають засоби для застосування SQL у традиційних мовах програмування. Існують два методи такого використання мови.

*Статична SQL* припускає вкладання безпосередньо в програмний код речення, що не може бути змінене під час виконання програми. Зазвичай статична SQL вимагає використання предкомпілятора. Наприклад, Oracle має предкомпілятори для використання SQL у мовах C, Pascal, Ada, COBOL та FORTRAN.

*Динамічна SQL* дає змогу програмувати побудову SQL-запитів, що створюються під час виконання програми і передаються СКБД, яка, в свою чергу, передає знайдені дані змінним програми.

## Контрольні запитання та завдання

1. Які можливості надає мова SQL?
2. Як можна вивести всі стовпці таблиці?
3. Для чого використовується модифікатор DISTINCT у фразі SELECT?
4. Як перевизначити імена стовпців таблиці, отриманої в результаті виконання запиту?
5. Що таке агрегатні функції? Перелічіть їх.
6. Чи можна використовувати у фразі SELECT водночас імена стовпців та агрегатні функції, якщо немає фрази GROUP BY?
7. Яке призначення фрази GROUP BY?
8. Чи можна застосовувати агрегатні функції у фразі WHERE?
9. Які вирази можна використовувати у фразі SELECT, коли наявна фраза GROUP BY?
10. Для чого призначена фраза HAVING?
11. Для чого призначена фраза ORDER BY? За якими стовпцями можна впорядковувати кінпеве відношення?
12. Які є типи вкладених запитів? Чим вони відрізняються?
13. Як використовуються невизначені значення?
14. Назвіть два різновиди команди INSERT.
15. Яке призначення команди UPDATE?
16. Що можна видалити за допомогою команди DELETE?
17. Які можливості надають команди CREATE TABLE, ALTER TABLE і DROP TABLE?
18. Що таке віртуальні таблиці? Як вони використовуються?
19. Для чого призначені транзакпії?
20. Що таке тригери? Які можливості вони надають?

# Розділ 5

# Мова QBE

- Вибирання даних
- Модифікація таблиць бази даних
- Варіант мови QBE у СКБД Paradox

QBE (Query By Example – запит за зразком) є графічною мовою запитів до реляційних баз даних. Формулюючи QBE-запит, користувач вписує у *бланки таблиць* бази даних одну з можливих відповідей на запит, а інтерпретуюча система за аналогією відшукує всі можливі відповіді. Операції задаються в табличній формі, тому можна сказати, що QBE має двовимірний синтаксис. У мові є можливість, якої не надають інші мови реляційної моделі, зокрема алгебра й числення, – можливість формулювати запити ієрархічної структури.

З огляду на графічну природу, мова має певні обмеження, але її виразові засоби є достатньо потужними. QBE надає користувачу можливість побудувати запит тим способом, що видається йому найзручнішим, порядок заповнення таблиць, рядків і комірок довільний. Як свідчать результати спеціальних досліджень, користувачі, які не є професійними програмістами, з усіх мов, що забезпечують взаємодію з базами даних, найкраще засвоюють саме QBE.

Ми розглянемо, як засобами цієї мови виражаються операції реляційної алгебри та деякі конструкції мови SQL. Приклади будемо формулювати, користуючись базою даних, розглянутою в попередніх розділах:

```
ФАКУЛЬТЕТ(#F, Назва, Декан, Корпус, Фонд)
КАФЕДРА(#D, #F, Назва, #ЗАВІДУВАЧ, Корпус, Фонд)
ВИКЛАДАЧ(#T, #D, Прізвище, Посада, Тел)
ГРУПА(#G, #D, Курс, Номер, Кількість, #КУРАТОР)
ПРЕДМЕТ(#S, Назва)
АУДИТОРІЯ(#R, Номер, Корпус, Місткість)
ЛЕКЦІЯ(#T, #G, #S, #R, Тип, День, Тиждень)
```

## 5.1. Вибирання даних

Засобами мови QBE можна змоделювати майже всі конструкції реляційної алгебри, числення та мови SQL. Відповідні запити будуть розглянуті в підрозділах 5.1.1–5.1.13. Крім того, QBE дозволяє формулювати ієрархічні запити, про які йтиметься в підрозділі 5.1.14.

## 5.1.1. Вибирання окремих стовпців

Щоб вибрати значення з окремих стовпців таблиці (за термінологією реляційної алгебри – виконати операцію проекції), ці стовпці потрібно позначити символами P. (print) у бланку таблиці. Наприклад, для виведення назв факультетів разом з їхніми корпусами бланк таблиці ФАКУЛЬТЕТ потрібно заповнити так:

| ФАКУЛЬТЕТ | #F | Назва | Декан | Корпус | Фонд |
|---|---|---|---|---|---|
| | | P. | | P. | |

За замовчуванням дублікати отриманих у результаті проекції кортежів не видаляються. Щоб результат не містив кортежів, які повторюються, слід у полі під іменем таблиці записати символи UNQ. (unique – унікальні). Запит

| ВИКЛАДАЧ | #T | #D | Прізвище | Посада | Телефон |
|---|---|---|---|---|---|
| UNQ. | | | P. | | P. |

виводить унікальні кортежі таблиці ВИКЛАДАЧ з полями Прізвище і Телефон.

Щоб вивести всі поля таблиці, слід проставити символи P. в усіх полях бланка, або в полі під іменем таблиці. Наприклад:

| КАФЕДРА | #D | #F | Назва | #ЗАВІДУВАЧ | Корпус | Фонд |
|---|---|---|---|---|---|---|
| P. | | | | | | |

## 5.1.2. Вибирання за умовою

### Кон'юнктивна селекція

У найпростішому випадку умова вибирання має такий вигляд:

```
<ім'я поля 1> θ <значення 1> AND <ім'я поля 2> θ <значення 2> AND ... AND
<ім'я поля n> θ <значення n>
```

де θ – один із предикатів порівняння (=, !=, >, >=, <, <=). Умова записується так, що предикати разом зі своїми значеннями вказуються у відповідних полях бланка таблиці. Наприклад, для отримання назв кафедр, що розташовуються в корпусі 5 і мають фонд більше 20 000, та кодів їхніх завідувачів бланк таблиці КАФЕДРА потрібно заповнити так:

| КАФЕДРА | #D | #F | Назва | #ЗАВІДУВАЧ | Корпус | Фонд |
|---|---|---|---|---|---|---|
| | | | P. | P. | = 5 | > 20000 |

Предикат = встановлюється за замовчуванням і тому перед значенням у клітинці його можна не записувати. Якщо всі компоненти запиту розташовані в одному рядку, інтерпретуюча система QBE вважатиме, що всі елементарні умови зв'язані логічними зв'язками «і».

### Диз'юнктивна селекція

Якщо умова містить логічні зв'язки AND, OR і NOT, вираз слід записати в диз'юнктивній нормальній формі:

(<літерал 11> AND ... AND <літерал 1k>) OR ... OR (<літерал p1> AND ... AND <літерал pn>)

Тут <літерал ij> позначає елементарний предикат вигляду <ім'я поля> θ <значення> чи його заперечення. Кожну з кон'юнкцій потрібно записати в окремому рядку бланка. Розглянемо приклад.

**Запит 5.1**

Вивести назви кафедр, що: (мають код менше 100 й розташовані в корпусі 3) або (розташовані в корпусі 7 і мають фонд більше 30 000).

| КАФЕДРА | #D | #F | Назва | #ЗАВІДУВАЧ | Корпус | Фонд |
|---|---|---|---|---|---|---|
| | < 100 | | P | | 3 | |
| | | | P. | | 7 | > 30000 |

Якщо заповнено два чи більше рядків, які не зв'язані між собою однією й тією ж змінною (про змінні йтиметься нижче), то умови, що задаються на цих рядках, з'єднуються логічним «або». Елементарні умови, що задаються в межах одного рядка, з'єднуються логічним «і», як і у випадку однорядкового запиту. Інтерпретуюча система QBE об'єднує в один потік всі значення, що отримуються в одному стовпці за умовами різних рядків.

### Диз'юнктивна селекція на одному полі

Диз'юнктивна селекція, що застосовується до одного поля, має такий вигляд:

(<літерал 11> AND ... AND <літерал 1k>) OR .. OR (<літерал p1> AND ... AND <літерал pn>)

Тут літерали містять ім'я лише одного поля. Така умова може бути записана з використанням змінних

**Запит 5.2**

Вивести назви кафедр, фонд яких (більше 2000 і менше 3000) або (більше 4000 і менше 5000 і не дорівнює 4500).

Отже, потрібно реалізувати таку диз'юнктивну нормальну форму:

(Фонд > 2000 AND Фонд < 3000) OR (Фонд >4000 AND Фонд < 5000 AND Фонд != 4500)

Реалізуємо умову Фонд > 2000 AND Фонд < 3000:

| ФАКУЛЬТЕТ | #F | Назва | Декан | Корпус | Фонд |
|---|---|---|---|---|---|
| | _x | P. | | P. | > 2000 |
| | _x | | | | < 3000 |

Вираз _x у полі #F — це *змінна*. Наявність однакової змінної у двох рядках свідчить про те, що умова в полі Фонд стосується одного й того ж факультету (зауважте, ми використали змінну _x у полі, яке є первинним ключем таблиці ФАКУЛЬТЕТ).

Другий кон'юнкт (Фонд >4000 AND Фонд < 5000 AND Фонд != 4500) виражається у такий самий спосіб. Отже, наведена вище умова зображується так:

| ФАКУЛЬТЕТ | #F | Назва | Декан | Корпус | Фонд |
|---|---|---|---|---|---|
| | _x | P. | | P. | > 2000 |
| | _x | | | | < 3000 |
| | _y | P | | P. | > 4000 |
| | _y | | | | < 5000 |
| | _y | | | | != 4500 |

### Предикат IN, застосований до переліку констант

Якщо умова має вигляд: «номер корпусу дорівнює 3, або 4, або 6, або 7», то диз'юнкція записується в чотирьох рядках таблиці. Ця умова також може бути записана за допомогою предиката IN: Корпус IN (3, 4, 6, 7). На бланку цей запис відобразиться так:

| КАФЕДРА | #D | #F | Назва | #ЗАВІДУВАЧ | Корпус | Фонд |
|---|---|---|---|---|---|---|
| | | | P. | | IN (3, 4, 6, 7) | |

## 5.1.3. Використання змінних

Змінні в мові QBE потрібні для зв'язування значень, що задаються різними рядками одного чи кількох бланків таблиць. Змінні записуються у вигляді рядкових літералів, яким передує символ підкреслення. У цьому підрозділі розглянемо використання змінних для виявлення зв'язків усередині однієї таблиці.

**Запит 5.3**

Вибрати прізвища викладачів, які обіймають ту саму посаду, що й Іванов.

| ВИКЛАДАЧ | #T | #D | Прізвище | Посада | Телефон |
|---|---|---|---|---|---|
| | | | P. | _x | |
| | | | Іванов | _x | |

Інтерпретуюча система QBE, обробляючи другий рядок бланка-запиту, зв'яже зі змінною _x множину посад, які мають викладачі з прізвищем Іванов, а потім виведе (1-й рядок) прізвища тих викладачів, чиї посади наявні у множині _x. Зауважте, що порядок рядків у бланку-запиті не впливає на результат.

**Запит 5.4**

Якщо викладачі Іванов і Петров мають однаковий номер телефону, вивести його.

| ВИКЛАДАЧ | #T | #D | Прізвище | Посада | Телефон |
|---|---|---|---|---|---|
| | | | Іванов | | P._x |
| | | | Петров | | _x |

За двома рядками таблиці, незалежно одна від одної, формуються множини, які зв'язуються кожна зі своїм примірником змінної _x, потім виводиться результат перетину цих двох множин. Отже, якщо існують такі пари рядків, що в одному з них зазначене прізвище «Іванов», в іншому — «Петров» і телефони у цих рядках однакові, то виводяться номери таких телефонів.

Рядки можна зв'язувати не тільки за предикатом рівності.

**Запит 5.5**

Вибрати всю інформацію про кафедри, що мають фонд більший, ніж кафедра АСУ.

| КАФЕДРА | #D | #F | Назва | #ЗАВІДУВАЧ | Корпус | Фонд |
|---|---|---|---|---|---|---|
| P | | | АСУ | | | > _x<br>_x |

Після формування множини, зв'язаної зі змінною _x, виводяться рядки, в яких значення поля Фонд більше хоча б за одне значення з цієї множини.

Наведемо ще один приклад вибирання даних з використанням зв'язку всередині однієї таблиці

**Запит 5.6**

Одержати коди викладачів, які викладають більше одного предмета.

| ЛЕКЦІЯ | #T | #G | #S | #R | Тип | День | Тиждень |
|---|---|---|---|---|---|---|---|
| | P._x<br>_x | | _y<br>!= _y | | | | |

Запит може бути проінтерпретований у такий спосіб: «Вибрати коди викладачів, які читають курс лекцій _y і також читають ще один курс лекцій, що відрізняється від _y». Аналогічно можна виражати умови типу: «... викладають більше двох предметів», «... викладають більше трьох предметів» тощо. Вони можуть бути також виражені за допомогою агрегатних функцій.

## 5.1.4. Запити за кількома таблицями

Засобами QBE можна формулювати запити, що передбачають з'єднання двох чи кількох таблиць. Для цього слід заповнити бланки таблиць, використовуючи спільні змінні для їхнього зв'язування. Розглянемо кілька прикладів.

**Запит 5.7**

Вибрати назви всіх кафедр факультету інформатики.

| ФАКУЛЬТЕТ | #F | Назва | Декан | Корпус | Фонд |
|---|---|---|---|---|---|
| | _x | інформатики | | | |

| КАФЕДРА | #D | #F | Назва | #ЗАВІДУВАЧ | Корпус | Фонд |
|---|---|---|---|---|---|---|
| | | _x | P. | | | |

**Запит 5.8**

Вибрати прізвища викладачів, які є кураторами груп.

| ВИКЛАДАЧ | #T | #D | Прізвище | Посада | Телефон |
|---|---|---|---|---|---|
| | _y | | P. | | |

| ГРУПА | #G | #D | Курс | Номер | Кількість | #КУРАТОР |
|---|---|---|---|---|---|---|
| | | | | | | _y |

**Запит 5.9**

Вибрати ті кафедри факультету інформатики, фонд яких перевищує фонд факультету.

| ФАКУЛЬТЕТ | #F | Назва | Декан | Корпус | Фонд |
|---|---|---|---|---|---|
| | _x | інформатики | | | _y |

| КАФЕДРА | #D | #F | Назва | #ЗАВІДУВАЧ | Корпус | Фонд |
|---|---|---|---|---|---|---|
| | | _x | P. | | | > _y |

У даному випадку змінна _x використовується для зв'язування двох таблиць за первинними та зовнішніми ключами, а предикат >_y вказує, що фонд кафедри перевищує фонд факультету.

**Запит 5.10**

Вивести прізвища викладачів-асистентів, які проводять лабораторні заняття.

| ВИКЛАДАЧ | #T | #D | Прізвище | Посада | Телефон |
|---|---|---|---|---|---|
| | _x | | P. | асистент | |

| ЛЕКЦІЯ | #T | #G | #S | #R | Тип | День | Тиждень |
|---|---|---|---|---|---|---|---|
| | _x | | | | лабораторна | | |

У цьому запиті обидві таблиці з'єднуються змінною _x за рівністю первинного та зовнішнього ключів #T, виконується селекція за посадою «асистент» і типом лекції «лабораторна», а потім з таблиці результату вибираються значення поля Прізвище.

## 5.1.5. Використання бланка умови

Інколи незручно або неможливо задавати умову в полях бланка однієї чи кількох таблиць. Якщо в предикаті використовуються різні атрибути або якщо один чи два операнди двомісного предиката є виразами над різними полями таблиці, то таку

умову неможливо записати в одному полі бланка. Прикладом подібної умови може бути: Надбавка > 2 * Зарплата

У цьому випадку для запису умови можна використовувати допоміжний бланк.

Наприклад, для отримання списку кафедр із фондом, що перевищує фонд кафедр «СКБД» чи «АСУ», бланком умови можна скористатися в такий спосіб:

| КАФЕДРА | #D | #F | Назва | #ЗАВІДУВАЧ | Корпус | Фонд |
|---|---|---|---|---|---|---|
| | | | P. | | | _x |
| | | | СКБД | | | _y |
| | | | АСУ | | | _z |

| Бланк умови |
|---|
| _x > _y OR _x > _z |

Бланк умови може виконувати ту саму функцію, що й фраза WHERE у мові SQL, тобто в ньому можна задавати умови з'єднання таблиць, селекції рядків у таблиці, умови, за якими виконується вкладання одного запиту в інший.

## 5.1.6. Використання полів імен таблиць

### Ідентифікатор друку в полі імені таблиці

Ми вже розглядали приклад використання поля імені таблиці, коли в ньому записували символи P. для специфікації відображення всіх полів таблиці. Загалом, якщо в полі імені таблиці зазначена якась дія, то вона поширюється на всю таблицю. Наприклад, у результаті виконання запиту

| ВИКЛАДАЧ | #T | #D | Прізвище | Посада | Телефон |
|---|---|---|---|---|---|
| P. | | | | | |

буде виведений вміст усієї таблиці, а запит

| ВИКЛАДАЧ | #T | #D | Прізвище | Посада | Телефон |
|---|---|---|---|---|---|
| P. | | | | != професор | |

виводить усі рядки таблиці, окрім тих, що стосуються професорів. Зауважимо, що в таблиці результату дублікати рядків не видаляються. Тому для виведення лише унікальних значень потрібно використовувати конструкцію UNQ.

**Запит 5.11**

Вивести перелік усіх посад та телефони деяких працівників, що їх обіймають.

| ВИКЛАДАЧ | #T | #D | Прізвище | Посада | Телефон |
|---|---|---|---|---|---|
| UNQ. | | | | P. | P. |

### Заперечення в полі імені таблиці

У полі імені таблиці можна також записувати логічну операцію заперечення NOT (або ¬), що перевіряє, чи є порожньою множина рядків, отримана в результаті виконання запиту. Ця операція еквівалентна предикату NOT EXISTS у SQL.

**Запит 5.12**

Отримати прізвища доцентів, які не читають жодної лекції в понеділок (тобто множина лекцій, що читаються ними в цей день, порожня).

| ВИКЛАДАЧ | #T | #D | Прізвище | Посада | Телефон |
|---|---|---|---|---|---|
| | _y | | P. | доцент | |

| ЛЕКЦІЯ | #T | #G | #S | #R | Тип | День | Тиждень |
|---|---|---|---|---|---|---|---|
| N | _y | | | | | понеділок | |

Інтерпретувати запит можна в такий спосіб: якщо для чергового рядка таблиці ВИКЛАДАЧ, у якому поле Посада дорівнює «доцент», не існує такого рядка з таблиці ЛЕКЦІЯ, що належить до того ж викладача (тобто в цих рядках значення поля #T збігаються), а поле День дорівнює «понеділок», то такий рядок таблиці ВИКЛАДАЧ належить результату. Інакше кажучи, будуть виведені прізвища всіх доцентів, що не читають лекцій у понеділок.

Заперечення може використовуватися в одному з кількох рядків бланка.

**Запит 5.13**

Вибрати коди предметів, кожен із яких читається лише одним викладачем.

| ЛЕКЦІЯ | #T | #G | #S | #R | Тип | День | Тиждень |
|---|---|---|---|---|---|---|---|
| | _v | | P._x | | | | |
| N_ | != _y | | _x | | | | |

## 5.1.7. Використання додаткових полів

Якщо потрібно вивести значення, яке обчислюється за допомогою значень із двох або більше полів, то необхідно застосувати додаткове не поіменоване поле.

**Запит 5.14**

Вивести різницю між кількістю студентів у групі та номером групи, поділену на номер курсу (даний запит не має реального змісту, ми його записуємо лише для демонстрації можливостей мови).

| ГРУПА | #G | #D | Курс | Номер | Кількість | #КУРАТОР | |
|---|---|---|---|---|---|---|---|
| | | | _z | _y | _x | | P. (_x-_y)/_z |

У мові QBE можна виводити значення полів тільки однієї таблиці, тому для виведення полів із різних таблиць потрібно використати додаткові поля.

**Запит 5.15**

Вивести прізвища викладачів-доцентів, які мають лекційні заняття, разом із днем та тижнем проведення занять.

| ВИКЛАДАЧ | #T | #D | Прізвище | Посада | Телефон | | |
|---|---|---|---|---|---|---|---|
| | _x | | P. | доцент | | P _y | P._z |

| ЛЕКЦІЯ | #T | #G | #S | #R | Тип | День | Тиждень |
|---|---|---|---|---|---|---|---|
| | _x | | | | лекція | _y | _z |

Зауважимо, що додаткові поля не можуть використовуватися для формування умов. Умови формулюються у бланку умов.

## 5.1.8. Теоретико-множинні предикати

Множина кортежів реляційного відношення специфікується за допомогою ключового слова ALL. Наявність цього слова в полі означає, що формується множина всіх значень поля в таблиці.

**Запит 5.16**

Знайти коди викладачів, що читають усі ті й лише ті предмети, що й викладач з кодом 100.

| ЛЕКЦІЯ | #T | #G | #S | #R | Тип | День | Тиждень |
|---|---|---|---|---|---|---|---|
| | P. | | ALL._y | | | | |
| | 100 | | ALL._y | | | | |

Запис ALL._y у другому рядку формує множину кодів предметів, що читаються викладачем з кодом 100, і ця множина ідентифікується змінною _y. Записи в першому рядку означають, що коли для поточного кортежу множина зв'язаних із ним кодів предметів дорівнює множині, сформованій відповідно до другого рядка (про це свідчить використання одних і тих самих змінних в обох рядках), то цей поточний кортеж включається у результат.

**Запит 5.17**

Вибрати прізвища викладачів, які читають усі ті й лише ті предмети, що й викладач Іванов.

| ВИКЛАДАЧ | #T | #D | Прізвище | Посада | Телефон |
|---|---|---|---|---|---|
| | _x | | P. | | |

| ЛЕКЦІЯ | #T | #G | #S | #R | Тип | День | Тиждень |
|---|---|---|---|---|---|---|---|
| | _x | | ALL._y | | | | |
| | _z | | ALL._y | | | | |

| ВИКЛАДАЧ | #T | #D | Прізвище | Посада | Телефон |
|---|---|---|---|---|---|
| | _z | | Іванов | | |

Нестроге включення однієї множини в іншу (⊇) записується в такий спосіб: менша множина вказується як ALL._x, а та, що повинна її містити, специфікується так: ALL._x. . (символи «. .» начебто замінюють слова «і, можливо, ще щось»).

**Запит 5.18.**

Вибрати коди викладачів, які читають принаймні всі ті предмети, що й викладач з кодом 100.

| ЛЕКЦІЯ | #T | #G | #S | #R | Тип | День | Тиждень |
|---|---|---|---|---|---|---|---|
| | P.<br>100 | | ALL._y. .<br>ALL._y | | | | |

Наявність у мові предикатів порівняння множин і перевірки того, чи є одна множина підмножиною іншої, дає можливість безпосередньо виражати запити з множинними порівняннями.

**Запит 5.19**

Вибрати прізвища викладачів, які читають усі типи лекцій.

| ВИКЛАДАЧ | #T | #D | Прізвище | Посада | Телефон |
|---|---|---|---|---|---|
| | _x | | P. | | |

| ЛЕКЦІЯ | #T | #G | #S | #R | Тип | День | Тиждень |
|---|---|---|---|---|---|---|---|
| | _x | | | | ALL._y<br>ALL._y | | |

## 5.1.9. Упорядкування результатів

Дані в таблиці-результаті можуть бути впорядковані за значеннями полів у порядку зростання чи спадання. Для цього необхідно до команди P дописати вказівку сортування: P.AO (Ascending Order, упорядкування у зростаючому порядку) або P.DO (Descending Order, упорядкування у спадному порядку).

**Запит 5.20.**

Вивести назви всіх факультетів, відсортувавши їх в алфавітному порядку.

| ФАКУЛЬТЕТ | #F | Назва | Декан | Корпус | Фонд |
|---|---|---|---|---|---|
| | | P.AO | | | |

Якщо кортежі необхідно відсортувати за кількома полями, то відразу за порядком сортування вказується вкладеність полів сортування, наприклад P.AO(1) — перше поле сортування, P.AO(2) — друге тощо.

**Запит 5.21**

Відсортувати факультети за номерами корпусів у порядку спадання, а потім за їхніми назвами – у порядку зростання.

| ФАКУЛЬТЕТ | #F | Назва | Декан | Корпус | Фонд |
|---|---|---|---|---|---|
| | | P.AO(2) | | P.DO(1) | |

## 5.1.10. Проміжні таблиці

Мова QBE не надає достатньо можливостей для вкладання одних виразів в інші, як, наприклад, реляційна алгебра та SQL, натомість вона дає можливість створювати проміжні таблиці запитів і використовувати їх в інших конструкціях того ж запиту. У багатьох випадках проміжні таблиці підвищують наочність запитів Іноді поділ запиту на кілька послідовних підзапитів значно полегшує їхнє формулювання. Проміжні таблиці можна використовувати також у тих випадках, коли поля виведення належать різним таблицям.

Повернімося до запиту 5.8, що встановлює зв'язок між викладачами і кураторами, і розширимо список вихідних полів.

**Запит 5.22**

Вибрати прізвища викладачів, які є кураторами груп, номери відповідних груп. кількість студентів у них і назви кафедр кураторів.

| ВИКЛАДАЧ | #T | #D | Прізвище | Посада | Телефон |
|---|---|---|---|---|---|
| | _y | _x | _tchNm | | |

| КАФЕДРА | #D | #F | Назва | #ЗАВІДУВАЧ | Корпус | Фонд |
|---|---|---|---|---|---|---|
| | _x | | _dpNm | | | |

| ГРУПА | #G | #D | КУРС | Номер | Кількість | #КУРАТОР |
|---|---|---|---|---|---|---|
| | | | | _grNm | _grQty | _y |

| РЕЗУЛЬТАТ | Викл_Прізвище | Каф_Назва | Група_Номер | Кількість |
|---|---|---|---|---|
| P. | _tchNm | _dpNm | _grNm | _grQty |

Ми створили нову таблицю РЕЗУЛЬТАТ, що має необхідні для виведення поля, в бланках яких вказані змінні з відповідних полів вихідних таблиць.

**Запит 5.23**

Вибрати прізвища викладачів, які є кураторами тих груп, що прослуховують усі типи лекцій, номери цих груп, кількість студентів у них і назви кафедр кураторів

Відмінність цього запиту від попереднього полягає в тому, що групи мають прослуховувати всі типи лекцій. Реалізувати цю умову можна за допомогою тимчасової таблиці з додатковим полем Група_#G.

| ВИКЛАДАЧ | #T | #D | Прізвище | Посада | Телефон |
|---|---|---|---|---|---|
| | _y | _x | _tchNm | | |

| КАФЕДРА | #D | #F | Назва | #ЗАВІДУВАЧ | Корпус | Фонд |
|---|---|---|---|---|---|---|
| | _x | | _dpNm | | | |

| ГРУПА | #G | #D | КУРС | Номер | Кількість | #КУРАТОР |
|---|---|---|---|---|---|---|
| | _g | | | _grNm | _grQty | _y |

| РЕЗУЛЬТАТ | Викл_Прізвище | Каф_Назва | Група_#G | Група_Номер | Кількість |
|---|---|---|---|---|---|
| | P._tchNm | P._dpNm | _g | P._grNm | P._grQty |

| ЛЕКЦІЯ | #T | #G | #S | #R | Тип | День | Тиждень |
|---|---|---|---|---|---|---|---|
| | | _g | | | ALL._t<br>ALL._t | | |

Введення додаткового поля Група_#G у таблицю РЕЗУЛЬТАТ необхідне для того, щоб за допомогою змінної _g сформулювати умову «групи, що прослуховують усі типи лекцій» з використанням бланка таблиці ЛЕКЦІЯ.

## 5.1.11. Агрегатні функції

Мова QBE надає можливість використання агрегатних функцій CNT, SUM, MIN, MAX, AVG. Щоб використати одну з таких функцій, потрібно вказати, що вона застосовується до множини значень поля

**Запит 5.24**

Визначити сумарну кількість студентів у вузі.

| ГРУПА | #G | #D | КУРС | Номер | Кількість | #КУРАТОР |
|---|---|---|---|---|---|---|
| | | | | | P.SUM.ALL | |

**Запит 5.25**

Визначити кількість викладачів, що є кураторами груп першого курсу.

| ГРУПА | #G | #D | КУРС | Номер | Кількість | #КУРАТОР |
|---|---|---|---|---|---|---|
| | | | 1 | | | P.CNT.UNQ.ALL |

Слово ALL вказує на те, що множина кодів кураторів має формуватися на базі всієї таблиці ГРУПА, UNQ — на те, що елементи цієї множини не повинні повторюватись (припускається, що викладачі можуть бути кураторами в кількох групах), CNT — це ім'я агрегатної функції підрахунку кількості елементів у множині.

Як і в SQL, разом з агрегатним значенням не можна виводити значення окремих полів.

Наведемо ще кілька прикладів використання агрегатних функцій.

**Запит 5.26**

Визначити кількість викладачів на кафедрі АСУ.

| КАФЕДРА | #D | #F | Назва | #ЗАВІДУВАЧ | Корпус | Фонд |
|---|---|---|---|---|---|---|
| | _x | | АСУ | | | |

| ВИКЛАДАЧ | #T | #D | Прізвище | Посада | Телефон |
|---|---|---|---|---|---|
| | | _x | P.CNT.UNQ.ALL | | |

**Запит 5.27**

Який найменший фонд фінансування серед кафедр факультету інформатики?

| ФАКУЛЬТЕТ | #F | Назва | Декан | Корпус | Фонд |
|---|---|---|---|---|---|
| | _x | інформатики | | | |

| КАФЕДРА | #D | #F | Назва | #ЗАВІДУВАЧ | Корпус | Фонд |
|---|---|---|---|---|---|---|
| | | _x | | | | P.MIN.ALL |

**Запит 5.28**

Якою є сумарна місткість усіх аудиторій корпусу 5, кількість місць у найменшій і найбільшій аудиторіях, середня місткість аудиторій у цьому корпусі?

| АУДИТОРІЯ | #R | Номер | Корпус | Місткість |
|---|---|---|---|---|
| | _y | | 5 | _x |

| РЕЗУЛЬТАТ | Сума | Найменша | Найбільша | Середня |
|---|---|---|---|---|
| P. | SUM._x | MIN._x | MAX._x | SUM._x / CNT._y |

Зауважте, що в цьому прикладі використовується допоміжна таблиця

## 5.1.12. Групування рядків таблиць

Мова QBE дає змогу групувати рядки, тобто містить аналог фрази GROUP BY, що використовується в SQL. Для того щоб здійснити групування за значеннями певного поля, до цього поля потрібно ввести вказівку групування G; групувати можна за значеннями кількох полів. Під час групування дозволяється виводити лише поля групування й агрегатні значення. Можна також накладати умову на вибір рядків для групування.

**Запит 5.29**

Обчислити кількість студентів першого курсу на кожній кафедрі факультету інформатики.

| ФАКУЛЬТЕТ | #F | Назва | Декан | Корпус | Фонд |
|---|---|---|---|---|---|
| | _x | інформатики | | | |

| КАФЕДРА | #D | #F | Назва | #ЗАВІДУВАЧ | Корпус | Фонд |
|---|---|---|---|---|---|---|
| | _y | _x | G._z | | | |

| ГРУПА | #G | #D | КУРС | Номер | Кількість | #КУРАТОР | |
|---|---|---|---|---|---|---|---|
| | | _y | 1 | | P.SUM.ALL | | P._z |

**Запит 5.30**

Обчислити фонди факультетів, отримані з таблиці ФАКУЛЬТЕТ, та суми фондів їхніх кафедр.

| ФАКУЛЬТЕТ | #F | Назва | Декан | Корпус | Фонд |
|---|---|---|---|---|---|
| | _x | G._y | | | G._z |

| КАФЕДРА | #D | #F | Назва | #ЗАВІДУВАЧ | Корпус | Фонд | | |
|---|---|---|---|---|---|---|---|---|
| | | _x | | | | P.SUM.ALL | P._y | P._z |

## 5.1.13. Предикати на групах рядків

У мові QBE можна задавати умови на групах рядків (аналог фрази HAVING у SQL). Як правило, умови на групах задаються в окремому бланку умови. Наведемо кілька прикладів.

**Запит 5.31**

Вибрати ті факультети з корпусу 11, фонд яких перевищує сумарний фонд усіх кафедр на 5000.

| ФАКУЛЬТЕТ | #F | Назва | Декан | Корпус | Фонд |
|---|---|---|---|---|---|
| | _x | P.G. | | 11 | _y |

| КАФЕДРА | #D | #F | Назва | #ЗАВІДУВАЧ | Корпус | Фонд |
|---|---|---|---|---|---|---|
| | | _x | | | | SUM.ALL._z |

| Бланк умови |
|---|
| _y > _z + 5000 |

**Запит 5.32**

Вибрати прізвища тих викладачів, для яких кількість лекцій, що читаються на 1-му тижні, перевищує більш ніж на 3 кількість лекцій, що читаються викладачем Івановим.

| ВИКЛАДАЧ | #T | #D | Прізвище | Посада | Телефон |
|---|---|---|---|---|---|
| | _x | | P | | |
| | _y | | Іванов | | |

| ЛЕКЦІЯ | #T | #G | #S | #R | Тип | День | Тиждень |
|---|---|---|---|---|---|---|---|
| | G._x | | | | | | 1 |
| | G._y | | | | | | |

| Бланк умови |
|---|
| CNT.ALL._x > CNT.ALL._y + 3 |

**Запит 5.33**

Вибрати прізвища викладачів, які мають на першому тижні заняття типу «лекція» в усі робочі дні (тобто дні, коли читається принаймні одна лекція).

| ВИКЛАДАЧ | #T | #D | Прізвище | Посада | Телефон |
|---|---|---|---|---|---|
| | _x | | P. G. | | |

| ЛЕКЦІЯ | #T | #G | #S | #R | Тип | День | Тиждень |
|---|---|---|---|---|---|---|---|
| | _x | | | | лекція | _y | 1 |
| | | | | | | _z | |

| Бланк умови |
|---|
| CNT.UNQ.ALL._y = CNT.UNQ.ALL._z |

За допомогою бланка ВИКЛАДАЧ і першого рядка бланка ЛЕКЦІЯ задаються умова з'єднання двох таблиць, умова групування, виведене поле, визначається умова селекції та специфікується змінна _y, що потім використовується в бланку умови Значення виразу CNT.UNQ.ALL._y дорівнює кількості неповторюваних днів тижня, коли викладач має лекції. Другий рядок бланка ЛЕКЦІЯ використовується лише для визначення змінної _z, що також вказується в бланку умови. Вираз CNT.UNQ.ALL._z визначає кількість усіх робочих днів тижня.

## 5.1.14. Ієрархічні запити

Розглянемо таблицю BOSS(#Mgr, #Emp), що містить відомості про підпорядкованість службовців певної організації. Службовець, який має номер #Emp, працює

під керівництвом начальника, який має номер #Mgr. Припустимо, що структурі підпорядкування притаманні такі властивості:

- жодний службовець не є своїм власним начальником чи начальником будь-якого зі своїх начальників (петель і циклів немає);
- жодний службовець має не більше одного безпосереднього начальника (відсутність множинного підпорядкування).

Ця структура називається деревоподібною. У мові QBE є можливість формулювати специфічні запити до таких об'єктів, які не можуть бути виражені в інших реляційних мовах, зокрема в реляційній алгебрі та численні. Розглянемо далі приклади.

**Запит 5.34**

Одержати номери службовців, які є безпосередніми підлеглими службовця 100 (підлеглими на першому рівні).

| BOSS | #Mgr | #Emp |
|---|---|---|
| | 100 | P. |

**Запит 5.35**

Одержати номери службовців, які є підлеглими службовцю 100 на другому рівні (тобто є безпосередніми підлеглими підлеглих службовця 100).

| BOSS | #Mgr | #Emp |
|---|---|---|
| | 100 | _x |
| | _x | P. |

Мова QBE надає можливість скороченого запису.

| BOSS | #Mgr | #Emp |
|---|---|---|
| | 100 | P. (2L) |

Скорочений запис має суттєві переваги: якщо необхідно обробити $n$ рівнів дерева, то можна задати номер рівня, замість того щоб використовувати $n-1$ рядків бланка. У наведеному прикладі (2L) позначає «другий рівень підлеглості». Замість 2 можна записати будь-яке натуральне число.

Використання подібних фраз у стовпці #Mgr дає можливість рухатися ієрархією вверх.

**Запит 5.36**

Одержати номер начальника, що перебуває на три рівні вище від службовця 200.

| BOSS | #Mgr | #Emp |
|---|---|---|
| | P (3L) | 200 |

Зазначимо, що коефіцієнти рівнів ієрархії не є засобом, що принципово розширює виразні можливості QBE, адже використання таких коефіцієнтів може бути замінене записами у відповідній кількості рядків у бланку таблиці. Проте QBE дозволяє також використовувати на місці коефіцієнта рівнів змінну, що суттєво збільшує потужність мови.

**Запит 5.37**

Одержати номери службовців, які є підлеглими службовцю 100 на всіх рівнях *підлеглості.*

| BOSS | #Mgr | #Emp |
|---|---|---|
| | 100 | P. (_6L) |

Подібний запит не можна виразити в жодній із розглянутих у попередніх розділах реляційних мов. Зауважте, що символ підкреслення в записі рівня (_6L) вказує на використання змінної (тобто вираз _6 є іменем змінної).

QBE надає можливість вибирати дані з найнижчого або найвищого рівня, тобто довжина ієрархічного шляху до шуканих підлеглих (керівників) має бути максимальною.

**Запит 5.38**

Одержати номери службовців, які є підлеглими службовцю 100 на найнижчому рівні.

| BOSS | #Mgr | #Emp |
|---|---|---|
| | 100 | P. (MAX._6L) |

Можна також шукати дані на кінцевих вершинах дерева

**Запит 5.39**

Вибрати номери тих службовців, які є підлеглими службовцю 100 і самі не мають підлеглих.

| BOSS | #Mgr | #Emp |
|---|---|---|
| | 100 | P (LAST.L) |

У цьому прикладі за допомогою функції LAST ми шукаємо службовців на кінцевих вершинах дерева під службовцем 100

## 5.2. Модифікація таблиць бази даних

Група операцій, яку буде розглянуто в цьому підрозділі, дає змогу додавати, замінювати й видаляти рядки таблиць. Можливості цих операцій обмежені порівняно з мовою SQL через те, що в них немає розвинених засобів модифікації з урахуванням поточного стану бази даних.

## 5.2.1. Додавання рядків

На позначення операції додавання рядка в полі імені таблиці записується символ I (Insert).

### Просте додавання

У найпростішому випадку поля рядка, що додається, заповнюються значеннями. Зауважте, що значення ключових полів мають бути присутніми завжди. Якщо значення того чи іншого поля відсутнє, воно набуває значення NULL. Наведемо приклад додавання рядка в таблицю:

| ВИКЛАДАЧ | #T | #D | Прізвище | Посада | Телефон |
|---|---|---|---|---|---|
| I. | 120 | 17 | Іванов | професор | 211-15-67 |

### Додавання з посиланнями на інші таблиці

У рядку, що додається, як значення тих чи інших полів можуть використовуватися змінні, які визначаються у бланках інших таблиць.

**Запит 5.40**

Додати рядок із відомостями про викладача Петрова, що працює на кафедрі АСУ

| ВИКЛАДАЧ | #T | #D | Прізвище | Посада | Телефон |
|---|---|---|---|---|---|
| I. | 121 | _x | Петров | доцент | 211-15-87 |

| КАФЕДРА | #D | #F | Назва | #ЗАВІДУВАЧ | Корпус | Фонд |
|---|---|---|---|---|---|---|
| | _x | | АСУ | | | |

Наведемо ще один приклад, дещо складніший

**Запит 5.41**

Додати рядок із відомостями про викладача Ігнатова, який працює на тій же кафедрі, що й викладач Іванов, обіймає таку саму посаду, що й викладач Петров. і не має телефону.

| ВИКЛАДАЧ | #T | #D | Назва | Посада | Телефон |
|---|---|---|---|---|---|
| I. | 122 | _x | Ігнатов | _y | |
| | | _x | Іванов | | |
| | | | Петров | _y | |

### Копіювання за умовою

Користуючись змінними, до таблиці можна скопіювати дані, які вибрані з іншої таблиці.

**Запит 5.42**

Нехай таблиця ВИКЛАДАЧ2 аналогічна за структурою таблиці ВИКЛАДАЧ. Для того щоб скопіювати у ВИКЛАДАЧ2 з ВИКЛАДАЧ рядки, що стосуються викладачів-доцентів, слід записати:

| ВИКЛАДАЧ2 | #T | #D | Прізвище | Посада | Телефон |
|---|---|---|---|---|---|
| I. | _x | _y | _z | доцент | _u |

| ВИКЛАДАЧ | #T | #D | Прізвище | Посада | Телефон |
|---|---|---|---|---|---|
| | _x | _y | _z | доцент | _u |

## 5.2.2. Оновлення рядків

Для оновлення рядків у бланку таблиці слід насамперед записати умову, за якою відбиратимуться рядки, що будуть оновлені. Після цього в полях, що мають бути оновлені, слід записати ознаку оновлення U., за якою вказати константу або вираз оновлення.

**Запит 5.43**

Встановити для кафедри «Бази даних» факультету інформатики фонд рівним 30 000.

| КАФЕДРА | #D | #F | Назва | #ЗАВІДУВАЧ | Корпус | Фонд |
|---|---|---|---|---|---|---|
| | | _x | Бази даних | | | U.30000 |

| ФАКУЛЬТЕТ | #F | Назва | Декан | Корпус | Фонд |
|---|---|---|---|---|---|
| | _x | інформатики | | | |

**Запит 5.44**

Збільшити фонд на 5 % всім факультетам, розташованим у корпусі 11.

| ФАКУЛЬТЕТ | #F | Назва | Декан | Корпус | Фонд |
|---|---|---|---|---|---|
| | | | | 11 | _x, U._x * 1.05 |

**Запит 5.45**

Усім факультетам, які мають фонд фінансування менший, ніж 30 000, збільшити фонд на 10 %.

| ФАКУЛЬТЕТ | #F | Назва | Декан | Корпус | Фонд |
|---|---|---|---|---|---|
| | | | | | < 30000, _x, U._x * 1.1 |

**Запит 5.46**

Усім факультетам, фонд фінансування яких менший чи дорівнює фонду фінансування факультету економіки, збільшити фонд на 10 %.

| ФАКУЛЬТЕТ | #F | Назва | Декан | Корпус | Фонд |
|---|---|---|---|---|---|
| | | економіки | | | _x, U._x * 1.1 >= _x |

**Запит 5.47**

У таблиці ФАКУЛЬТЕТ усі корпуси 5 замінити на 12.

| ФАКУЛЬТЕТ | #F | Назва | Декан | Корпус | Фонд |
|---|---|---|---|---|---|
| | _x | | | 5 | |
| U. | _x | | | 12 | |

Ознака оновлення U. вказує, який із двох рядків відповідає новим даним.

## 5.2.3. Видалення рядків

Видалення ідентифікується ознакою D. у полі імені таблиці чи всередині її бланка. У першому випадку видаляються рядки, що відповідають заданій умові. У другому випадку в усіх рядках, що відповідають умові, видаляються значення тих полів, в яких записано ознаку видалення. Існує можливість видаляти окремі рядки таблиці, множину рядків, а також видаляти множини зв'язаних рядків у багатьох таблицях.

**Запит 5.48**

У таблиці ВИКЛАДАЧ видалити номери телефонів усіх викладачів, кафедри яких розташовуються в корпусі 7 (видалення значення поля).

| ВИКЛАДАЧ | #T | #D | Прізвище | Посада | Телефон |
|---|---|---|---|---|---|
| | | _x | | | D. |

| КАФЕДРА | #D | #F | Назва | #ЗАВІДУВАЧ | Корпус | Фонд |
|---|---|---|---|---|---|---|
| | _x | | | | 7 | |

**Запит 5.49**

З таблиці ЛЕКЦІЯ видалити відомості, що стосуються занять типу «лабораторна», які проводяться у суботу і неділю.

| ЛЕКЦІЯ | #T | #G | #S | #R | Тип | День | Тиждень |
|---|---|---|---|---|---|---|---|
| D. | | | | | лабораторна | IN (сб, нд) | |

**Запит 5.50**

У зв'язку з переведенням до іншого вузу кафедри лінгвістики разом з викладачами і студентами, видалити з таблиці КАФЕДРА рядок з відомостями про цю кафедру, з таблиці ВИКЛАДАЧ – відомості про всіх викладачів кафедри, з таблиці ГРУПА – відомості про всі групи студентів кафедри, з таблиці ЛЕКЦІЯ – відомості про лекції, що читаються цим групам, а також коди викладачів, що були переведені.

| КАФЕДРА | #D | #F | Назва | #ЗАВІДУВАЧ | Корпус | Фонд |
|---|---|---|---|---|---|---|
| D. | _x | | лінгвістики | | | |

| ВИКЛАДАЧ | #T | #D | Прізвище | Посада | Телефон |
|---|---|---|---|---|---|
| D. | _y | _x | | | |

| ГРУПА | #G | #D | КУРС | Номер | Кількість | #КУРАТОР |
|---|---|---|---|---|---|---|
| D. | _z | _x | | | | |

| ЛЕКЦІЯ | #T | #G | #S | #R | Тип | День | Тиждень |
|---|---|---|---|---|---|---|---|
| D. | D. _y | _z | | | | | |

# 5.3. Варіант мови QBE в СКБД Paradox

СКБД Paradox підтримує різновид мови QBE, який є одним із найбільш близьких до класичного варіанту цієї мови, викладеного у попередніх підрозділах. З різновидом цього діалекту QBE можна також ознайомитися за допомогою утиліти Data-Base Desktop, яка входить до складу інтегрованих середовищ програмування фірми Borland (Delphi. C++ Builder та ін.).

## 5.3.1. Пошукові запити

Будуючи запит, система за допомогою діалогового вікна з'ясовує у користувача ім'я таблиці, за якою буде формуватися запит; якщо потрібно кілька таблиць, їх можна буде додати пізніше. Одержавши ім'я таблиці, система будує її бланк (чи схему), до якого користувач має ввести запит. Розглянемо кілька прикладів.

**Запит 5.51**

Вивести назви кафедр, що розташовуються в корпусі 5 і мають фонд більший, ніж 20 000, а також коди їхніх завідувачів.

| КАФЕДРА | #D | #F | Назва | #ЗАВІДУВАЧ | Корпус | Фонд |
|---|---|---|---|---|---|---|
| | | | √ | √ | = 5 | > 20000 |

Замість символів P. у СКБД Paradox використовується символ √.

**Запит 5.52**

Визначити назви факультетів, фонд яких не менше 2000 і не більше 3000, або більше 5000, а також номери їхніх корпусів.

| ФАКУЛЬТЕТ | #F | Назва | Декан | Корпус | Фонд |
|---|---|---|---|---|---|
| | | √ | | √ | (>= 2000 , <= 3000 ) DR >5000 |

Символ «,» використовується замість логічної зв'язки AND.

**Запит 5.53**

Вибрати назви всіх кафедр факультету інформатики.

| ФАКУЛЬТЕТ | #F | Назва | Декан | Корпус | Фонд |
|---|---|---|---|---|---|
| | _x | інформатики | | | |

| КАФЕДРА | #D | #F | Назва | #ЗАВІДУВАЧ | Корпус | Фонд |
|---|---|---|---|---|---|---|
| | | _x | √ | | | |

Розглянемо запити з множинними порівняннями.

**Запит 5.54**

Знайти коди викладачів, які читають лекції всіх типів.

| ЛЕКЦІЯ | #T | #G | #S | #R | Тип | День | Тиждень |
|---|---|---|---|---|---|---|---|
| SET | √ | | | | _y<br>EVERY _y | | |

У цьому запиті спочатку зі змінною _y зв'язується множина всіх типів лекцій, а потім для кожного коду лектора (другий рядок) перевіряється, чи є множина лекцій, які він читає, надмножиною множини _y; якщо так, код лектора виводиться.

**Запит 5.55**

Знайти коди викладачів, які читають лише ті лекції, що проводяться в п'ятницю.

| ЛЕКЦІЯ | #T | #G | #S | #R | Тип | День | Тиждень |
|---|---|---|---|---|---|---|---|
| SET | √ | | | | _y<br>ONLY _y | "п'ятниця" | |

Запит виконується аналогічно до попереднього. Головна відмінність полягає в тому, що здійснюється перевірка, чи є множина типів лекцій, які викладач читає, підмножиною множини _y.

**Запит 5.56**

Знайти коди викладачів, які читають ті й лише ті лекції, що проводяться в п'ятницю.

| ЛЕКЦІЯ | #T | #G | #S | #R | Тип | День | Тиждень |
|---|---|---|---|---|---|---|---|
| SET | √ | | | | _y<br>EXACTLY _y | "п'ятниця" | |

Аналогічно до попереднього запиту зі змінною _y зв'язується множина типів лекцій, які читаються в п'ятницю, потім для кожного коду лектора (другий рядок) перевіряється, чи дорівнює множина типів лекцій, які він читає, множині _y; якщо умова виконана, код лектора виводиться.

**Запит 5.57**

Знайти прізвища викладачів, які читають лекції тих й лише тих типів, що проводяться у п'ятницю.

| ЛЕКЦІЯ | #T | #G | #S | #R | Тип | День | Тиждень |
|---|---|---|---|---|---|---|---|
| SET | √G _x | | | | _y<br>EXACTLY _y | "п'ятниця" | |

| ВИКЛАДАЧ | #T | #D | Прізвище | Посада | Телефон |
|---|---|---|---|---|---|
| | _x | | √ | | |

Стосовно бланка таблиці ВИКЛАДАЧ ніяких додаткових зауважень непотрібно, оскільки тут, як і в багатьох попередніх запитах, для кожного значення змінної _x виводиться відповідне значення зі стовпця Прізвище. Використання у бланку ЛЕКЦІЯ комбінації символів √G _x пов'язане з тим, що рядки цієї таблиці потрібно згрупувати за кодом лектора, не виводячи його.

Результати всіх пошукових запитів у СКБД Paradox записуються до таблиці зі стандартним ім'ям ANSWER.

## 5.3.2. Запити дії

Відмінності запису запитів дії у СКБД Paradox порівняно з класичним варіантом несуттєві й мають переважно синтаксичний характер.

**Запит 5.58**

Встановити для кафедри «Бази даних» факультету інформатики фонд рівним 30 000.

| КАФЕДРА | #D | #F | Назва | #ЗАВІДУВАЧ | Корпус | Фонд |
|---|---|---|---|---|---|---|
| | | _x | Бази даних | | | changeto 30000 |

| ФАКУЛЬТЕТ | #F | Назва | Декан | Корпус | Фонд |
|---|---|---|---|---|---|
| | _x | інформатики | | | |

**Запит 5.59**

Збільшити фонд на 5 % всім факультетам, розташованим у корпусі 11

| ФАКУЛЬТЕТ | #F | Назва | Декан | Корпус | Фонд |
|---|---|---|---|---|---|
| | | | | 11 | _x, changeto_x * 1.05 |

**Запит 5.60**

Усім факультетам, в яких фонд фінансування менший за 3000, збільшити фонд на 10 %.

| ФАКУЛЬТЕТ | #F | Назва | Декан | Корпус | Фонд |
|---|---|---|---|---|---|
| | | | | | < 30000, _x, changeto _x * 1.1 |

На відміну від класичної моделі, у запитах на видалення та додавання даних у СКБД Paradox замість кодів I та D використовуються повні назви, тобто Insert та Delete.

## Контрольні запитання та завдання

1. Для чого в мові QBE використовуються змінні?
2. В який спосіб використовують бланки умов?
3. З якою метою в мові застосовуються поля імен таблиць?
4. Опишіть різні варіанти використання додаткових полів таблиць.
5. Як у мові описуються теоретико-множинні предикати?
6. Для чого використовуються проміжні таблиці?
7. Які можливості QBE відсутні в SQL?
8. Чи є мова QBE реляційно повною? Обґрунтуйте відповідь.
9. Задано таку базу даних:

```
ПОСТАЧАЛЬНИК(SID, Прізвище, Місто)
ВИРОБИ(PID, Назва, Колір)
ЗАМОВЛЕННЯ(SID, PID, Кількість)
```

Реалізуйте на мові QBE такі запити.

а) вивести прізвища та назви міст постачальників, які отримали замовлення на червоні олівці та білий папір у кількості не менше 150 одиниць кожного виробу;

б) вивести назви жовтих виробів, які були замовлені у постачальників з Києва, Житомира та Вінниці;

в) вивести прізвища та назви міст постачальників, які отримали замовлення на червоні та чорні вироби у кількості не менше 200;

г) вивести назви виробів, які були замовлені у постачальника Іванова та іншого постачальника у кількості, яка перевищує кількість замовлення цього виробу в Іванова не менш як на 300;

д) вивести всю наявну інформацію про постачальників, які отримали замовлення на зелені вироби;

е) для кожного замовлення на червоні вироби вивести назву виробу і замовлену кількість;

є) вивести назви виробів (в алфавітному порядку), які одночасно були замовлені у Іванова і Петрова;

ж) вивести назви виробів, які були замовлені у Іванова, але не були замовлені у Петрова;

з) вивести загальну кількість виробів, що були замовлені;

и) для кожного постачальника вивести його прізвище та загальну кількість замовлених у нього виробів червоного кольору;

і) для кожного виробу червоного кольору вивести середній обсяг замовлення (тобто скільки в середньому їх замовляється);

ї) вивести прізвища постачальників, які отримали замовлення на всі вироби.

# Розділ 6

# Теорія нормалізації реляційної моделі даних

- Функціональні залежності
- Нефункціональні залежності
- Нормальні форми реляційних відношень
- Проектування схеми бази даних у реляційній моделі

Завдяки теорії нормалізації схем відношень у реляційній моделі даних встановлюється, в який спосіб схема реляційних відношень може бути перетворена на іншу схему, що еквівалентна в певному розумінні початковій і в певному розумінні є кращою за неї. В межах цієї теорії формулюються критерії еквівалентності й оцінювання якості схем реляційних відношень, а також описуються механізми еквівалентних перетворень названих схем, які надають можливість підвищити їхню якість.

У цьому розділі наводяться базові означення й розглядаються властивості функціональної та багатозначної залежностей, означуються нормальні форми реляційних відношень і формулюється основне завдання проектування схем реляційної моделі даних.

## 6.1. Функціональні залежності

Поняття *функціональної залежності* є базовим у теорії проектування реляційних баз даних. На нього спираються як критерії якості схем баз даних, так і методи їхнього поліпшення. Властивості функціональних залежностей означуються аксіоматично. В даному підрозділі наводиться відповідна система аксіом, а також даються всі означення, необхідні для того, щоб у наступних підрозділах розглянути методи проектування баз даних.

### 6.1.1. Основні поняття

Нехай задано відношення $R$, яке містить набори атрибутів $A$ і $B$. У відношенні $R$ набір атрибутів $B$ *функціонально залежить* від $A$ і $A$ функціонально визначає $B$ тоді й лише тоді, коли кожному значенню проекції $R[A]$ у будь-який момент часу відповідає точно одне значення проекції $R[B]$.

Означена вище функціональна залежність позначається як $R.A \to R.B$. Якщо належність $A$ і $B$ відношенню $R$ відома апріорі, то пишеться $A \to B$. Формально функціональна залежність означується так:

$$R.A \to R.B \Leftrightarrow \forall r_1 \in R \; \forall r_2 \in R(r_1[A] = r_2[A] \Rightarrow r_1[B] = r_2[B]).$$

Поняття функціональної залежності може бути звужене на той випадок, коли $A$ і $B$ є окремими атрибутами.

Зауважимо, що наявність функціональної залежності є властивістю схеми, а не того чи іншого екземпляра відношення, і відображує семантику предметної області, що моделюється. Одним із базових понять у теорії нормалізації є поняття ключового набору атрибутів відношення. Розглядаються кілька різновидів ключів.

Набір $K$ атрибутів відношення $R$ називається *можливим ключем*, або *квазіключем* відношення $R$, якщо:

а) кожний атрибут відношення $R$ функціонально залежить від $K$;

б) жоден атрибут з набору $K$ не може бути видалений так, щоб не порушувалась властивість (а).

У формульному вигляді це означення записується так. Нехай $M$ – повний набір атрибутів відношення $R$. Підмножина атрибутів $K$ відношення $R$ є можливим ключем, якщо:

а) $\forall A \subseteq M(R.K \to R.A)$;

б) $\forall A \subset K(\exists B \subseteq M \mid R.A \not\to R.B)$.

Символ $\not\to$ вказує на те, що функціональна залежність відсутня.

У будь-якому відношенні існує принаймні один можливий ключ, оскільки набір усіх атрибутів відношення задовольняє властивість (а), а потім цей набір можна «стиснути» так (відкидаючи надлишкові атрибути), щоб він задовольняв властивість (б).

Оскільки у відношенні може існувати більше одного ключа, то один із них називається *первинним*. Будь-який із можливих ключів може бути первинним.

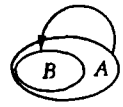
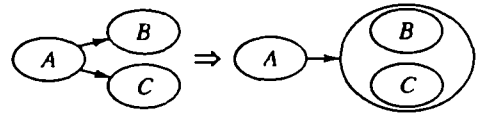
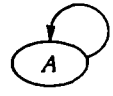
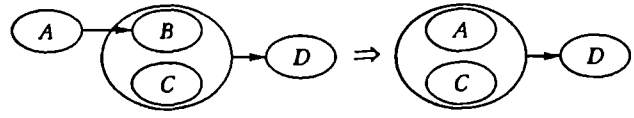
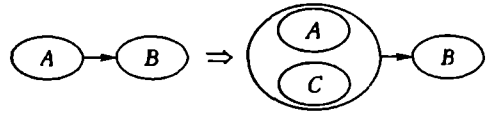
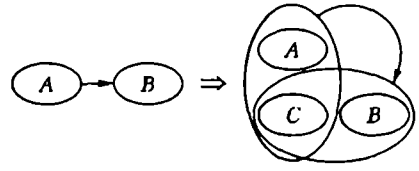
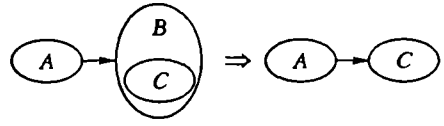
Множина атрибутів, що містить можливий ключ, називається *суперключем*. Атрибути, які входять до складу можливого ключа відношення, називаються *ключовими*; атрибути, які належать первинному ключу, – *первинними*, решта – *непервинними*, або *вторинними*.

## 6.1.2. Аксіоматика функціональних залежностей

Стосовно заданого реляційного відношення $R$ ми можемо розглядати множину функціональних залежностей $F$, які визначені на ньому.

Як довів У. Армстронг, множина функціональних залежностей $F$ має певні властивості, що перелічені в табл. 6.1. У лівому стовпці таблиці згадані властивості записані в аналітичному вигляді, у правому – в графічному. Більшість тверджень тут наведено у формі «якщо ... , то ... », яку слід розуміти так: якщо деякі функціональні залежності належать множині $F$, то цій множині належать і ті залежності, що вказані після слова «то».

**Таблиця 6.1.** Властивості функціональних залежностей

| Властивість | Графічне позначення |
|---|---|
| 1) Транзитивність: якщо $A \to B$ і $B \to C$, то $A \to C$ |  |
| 2) Проективність: якщо $B \subseteq A$, то $A \to B$ |  |
| 3) Адитивність (об'єднання): якщо $A \to B$ і $A \to C$, то $A \to (B, C)$ |  |
| 4) Рефлексивність: $A \to A$ |  |
| 5) Псевдотранзитивність: якщо $A \to B$ і $(B, C) \to D$, то $(A, C) \to D$ |  |
| 6) Продовження: якщо $A \to B$, то $(A, C) \to B$ для будь-якого атрибута $C$ |  |
| 7) Поповнення: якщо $A \to B$ то $(A, C) \to (B, C)$ для будь-якого атрибута $C$ |  |
| 8) Декомпозиція: якщо $A \to B$ і $C \subseteq B$, то $A \to C$ |  |

Не всі з наведених властивостей є незалежними, а саме властивості (4)–(8) виводяться з (1), (2), (3), які утворюють повну систему аксіом функціональних залежностей.

## 6.1.3. Логічне виведення функціональних залежностей

Нехай на відношенні $R$ визначено множину функціональних залежностей $F$ та залежність $A \to C$, яка не належить $F$. Залежність $A \to C$ *логічно випливає* з множини $F$, якщо вона може бути виведена з $F$ за допомогою аксіом функціональних залежностей. Кажуть також, що залежність $A \to C$ *виводиться* з $F$, або є *логічним наслідком* $F$.

Наприклад, якщо $R = (A, B, C)$ і множина $F$ складається з залежності $A \to B$. то з $F$ логічно випливають такі залежності:

- $(A, C) \to B$ – за властивістю продовження;
- $(A, C) \to (B, C)$ – за властивістю поповнення

Припустимо, що на відношенні $R$ задано множину функціональних залежностей $F$. Множина всіх функціональних залежностей, кожна з яких є логічним наслідком $F$, називається *логічним замиканням* $F$, вона позначається як $F^+$. Очевидно, що $F \subseteq F^+$ і $F^+ = F^{++}$.

Усі функціональні залежності, що належать замиканню, можуть бути отримані з початкової множини $F$ застосуванням властивостей (1), (2), (3), тому ці властивості іноді називають *правилами виведення*.

Множина функціональних залежностей $F$ є *повною*, якщо $F = F^+$.

Дві множини залежностей $F$ і $G$ називаються *логічно еквівалентними*, якщо $F^+ = G^+$.

Нехай задано множину функціональних залежностей $F$. Множина функціональних залежностей $G$ є *базисом*, або *мінімальним покриттям* множини $F$, якщо $G$ є такою підмножиною $F$, що $G^+ = F^+$, а жодна підмножина $G$ цієї властивості не має.

## 6.2. Нормальні форми реляційних відношень

Реляційне відношення перебуває в тій чи іншій нормальній формі, якщо задані на ньому функціональні залежності задовольняють певні умови. Відношення, які не перебувають у відповідній нормальній формі, мають певні небажані властивості, або *аномалії*. Відтак виникає потреба в нормалізації, орієнтованій на те, щоб позбавити реляційні відношення цих властивостей. Аномалії виникають унаслідок того, що реляційні відношення можуть містити надлишкові функціональні залежності, тобто кількість атрибутів відношення може бути завеликою, і тоді постає питання про коректність його схеми. *Коректною* вважається схема, що не містить небажаних функціональних залежностей. Виправлення некоректних схем здійснюється за допомогою процедури декомпозиції (розкладання) реляційного відношення на множину інших відношень. Мета цієї процедури – побудова множини таблиць без небажаних функціональних залежностей та аномалій. Це і є суттю процесу нормалізації. Іншими словами, *нормалізація* – це зворотний процес заміни даної схеми реляційних відношень іншою схемою, в якій відношення мають просту й коректну форму. Зворотність нормалізації означає, що після її здійснення зберігається можливість повернення відношення до початкового стану.

Е. Кодд спочатку визначив три рівні нормалізації, які він назвав першою, другою і третьою нормальними формами (1НФ, 2НФ і 3НФ). Згодом Р. Фейджин визначив четверту нормальну форму, в якій перебувають деякі відношення, що мають 3НФ. Нарешті, була визначена ще одна форма, п'ята. Ці форми та відповідні залежності будуть розглянуті в підрозділах 6.2.1–6.2.3.

## 6.2.1. Складені домени і перша нормальна форма

Реляційна модель не допускає використання складених (неатомарних) доменів, тобто на перетині рядка і стовпця у таблиці має стояти атомарне значення, неподільне з точки зору всіх його можливих застосувань. Реляційне відношення перебуває в *першій нормальній формі* (1НФ), якщо всі його атрибути мають атомарні (прості) домени, відтак значення елементів таблиці є простими. Реляційне відношення називається *нормалізованим*, якщо воно перебуває в 1НФ.

На рис. 6.1 зображено структуру відношення СЛУЖБОВЕЦЬ, побудованого на атомарних доменах НС (номер службовця) і Прізвище та складеному домені Діти. Ця структура є деревоподібною і тому відношення не перебуває в 1НФ.

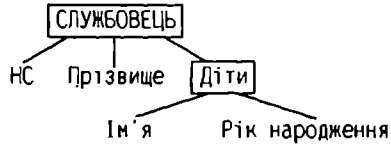



**Рис. 6.1.** Структура відношення, що не перебуває в 1НФ

Розглянемо екземпляр відношення СЛУЖБОВЕЦЬ.

| НС | Прізвище | Діти | |
|---|---|---|---|
| | | Ім'я | Рік народження |
| 1 | Іванов | Анна | 1987 |
| | | Петро | 1979 |
| 2 | Петров | Юлія | 1989 |
| | | Анна | 1991 |
| | | Іван | 1994 |
| 3 | Попов | Ігор | 1991 |

У 1НФ цей екземпляр відношення зображуватиметься у вигляді двох таблиць:

СЛУЖБОВЕЦЬ

| НС | Прізвище |
|---|---|
| 1 | Іванов |
| 2 | Петров |
| 3 | Попов |

Діти

| НС | Ім'я | Рік народження |
|---|---|---|
| 1 | Анна | 1987 |
| 1 | Петро | 1979 |
| 2 | Юлія | 1989 |
| 2 | Анна | 1991 |
| 2 | Іван | 1994 |
| 3 | Ігор | 1991 |

Реляційна модель даних означена лише для тих відношень, що перебувають у першій нормальній формі.

## 6.2.2. Неповні функціональні залежності та друга нормальна форма

Нехай на реляційному відношенні $R$ задано набори атрибутів $A$ і $B$. Функціональна залежність $R.A \rightarrow R.B$ називається *повною*, якщо $B$ не залежить функціонально від жодного піднабору $C \subset A$, що не містить $B$.

На рис. 6.2 атрибут Аудиторія неповно залежить від набору атрибутів Викладач, Група, Предмет, оскільки він залежить і від одного атрибуту Предмет, що входить до складу цього набору.

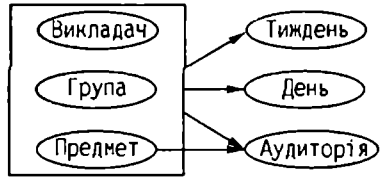



**Рис. 6.2.** Реляційне відношення ЛЕКЦІЇ, що містить неповну функціональну залежність

Реляційному відношенню ЛЕКЦІЇ властиві аномалії, що пов'язані з операціями маніпулювання даними. Пояснимо, чому ці аномалії виникають.

Якщо в предметній області є функціональна залежність $A \rightarrow B$, це свідчить про наявність множини сутностей, складених з атрибутів ($A$, $B$), причому група атрибутів $A$ є унікальним ідентифікатором сутностей цієї множини (ключем), а $B$ – їхньою властивістю. Оскільки атрибут Аудиторія функціонально залежить від атрибута Предмет, то в нашій предметній області на цій парі атрибутів визначена окрема множина сутностей. Проте ми не зможемо незалежно маніпулювати сутностями даної множини, оскільки це призведе до таких аномалій.

### Аномалія додавання

Ми не зможемо відобразити той факт, що в конкретній аудиторії читаються лекції з конкретного предмета доти, доки не будуть визначені викладач і група

### Аномалія видалення

Видаляючи рядок з таблиці ЛЕКЦІЇ, ми можемо видалити унікальну (таку, якої більше ніде не міститься) інформацію про те, що певний предмет читається в певній аудиторії.

### Аномалія оновлення

Одне й те саме значення атрибута Аудиторія для даного предмета повторюється багато разів. Тому, якщо предмет став читатися в іншій аудиторії, необхідно буде замінити номер аудиторії в усіх рядках таблиці ЛЕКЦІЇ.

З поняттям повної функціональної залежності пов'язане поняття другої нормальної форми.

Реляційне відношення перебуває в *другій нормальній формі* (2НФ), якщо воно *перебуває в першій нормальній формі й усі його неперервинні атрибути функціо*нально повно залежать від певного можливого ключа.

Друга нормальна форма нейтралізує аномалії, пов'язані з неповними функціональними залежностями. Звести відношення до другої нормальної форми можна лише шляхом його поділу (декомпозиції) принаймні на два інших. Але цей поділ має відповідати таким умовам:

- відношення, які отримуємо в результаті, є проекціями початкового;
- початкове реляційне відношення можна відновити з кінцевих за допомогою операції природного з'єднання без втрат даних;
- під час декомпозиції не втрачаються функціональні залежності (тобто множини залежностей початкового й кінцевого відношень є еквівалентними)

Шлях до вирішення цієї проблеми дає теорема, доведена І. Хітом

**Теорема Хіта**

Якщо у реляційному відношенні $R$ з наборами атрибутів $A$, $B$, $C$ ($B \cap C = \varnothing$) задано функціональну залежність $R.A \rightarrow R.B$, то його можна декомпонувати на два відношення – $R[A, B]$ і $R[A, C]$ так, що завдяки операції природного з'єднання проекцій за групами атрибутів $A$ початкове відношення $R$ відновлюється без втрат даних

Алгоритм зведення схеми відношення до 2НФ є таким. Нехай задано реляційне відношення $R$ із множиною атрибутів $M$. Якщо в $R$ є неповна функціональна залежність $R.A \rightarrow R.B$ непервинного атрибута $B$ від можливого ключа $A$, то $R$ декомпонується на дві проекції: $R[A, B]$ і $R[M \quad B]$ ($M - B$ – це теоретико-множинна різниця). Якщо отримані в результаті таблиці все ще не перебувають у другій нормальній формі, до них знову застосовується це правило, доки всі відношення не досягнуть 2НФ.

Наприклад, для наведеного вище реляційного відношення ЛЕКЦІЇ алгоритм нормалізації приведе до таких двох таблиць: ЛЕКЦІЇ(Викладач, Група, Предмет, Пара, День) і ЩО-ДЕ(Предмет, Аудиторія). Декомпозиція виконана на основі теореми Хіта за функціональною залежністю Предмет → Аудиторія.

Початкову таблицю можна відновити без втрат з отриманих проекцій шляхом природного з'єднання за атрибутом Предмет. Це твердження обґрунтовується тим, що таблиця ЛЕКЦІЇ містить у повному складі той самий ключ, що й початкове реляційне відношення, тому під час проектування зберігаються всі кортежі. Значення атрибута Аудиторія можна однозначно відновити з іншої таблиці, оскільки має місце функціональна залежність Предмет → Аудиторія.

Процес зведення реляційного відношення до другої нормальної форми у певних випадках не є однозначним. У зв'язку з цим вводиться таке поняття.

Декомпозиція схеми реляційного відношення до другої нормальної форми називається *оптимальною*, якщо в результаті утворюється мінімально можлива кількість таблиць-проекцій.

## 6.2.3. Транзитивні залежності й третя нормальна форма

Поняття функціональної транзитивної залежності добре відоме в класичній теорії множин та теорії відношень. У реляційному підході використовується більш вузьке тлумачення цього терміна.

Набір атрибутів *C транзитивно залежить* від набору атрибутів *A*, якщо існує такий набір атрибутів *B*, що $A \to B$, $B \to C$ і $B \not\to A$. Останньою умовою реляційний варіант означення відрізняється від класичного.

Наявність у реляційному відношенні транзитивної залежності призводить до тих самих аномалій, що й наявність неповної функціональної залежності. Якщо в $R(A, B, C)$ є функціональні залежності $A \to B$ та $B \to C$, то таке відношення містить інформацію про дві множини сутностей: $(A, B)$ та $(B, C)$, що призводить до аномалій додавання, видалення й оновлення.

З поняттям функціональної транзитивної залежності пов'язане поняття третьої нормальної форми

Реляційне відношення перебуває в *третій нормальній формі* (3НФ), якщо воно перебуває в другій нормальній формі й не містить транзитивних функціональних залежностей непервинних атрибутів від можливих ключів.

Декомпозиція реляційного відношення на кілька відношень у третій нормальній формі має задовольняти ті самі властивості, що й декомпозиція до другої нормальної форми. Алгоритм зведення до 3НФ базується на теоремі Хіта. Розглянемо його детальніше.

Нехай задано відношення $R$ з атрибутами (або наборами атрибутів) $A$, $B$, $C$, де $A$ - можливий ключ, і є функціональні залежності $R.A \to R.B$, $R.B \to R.C$. Тоді таблиця $R$ декомпонується на дві проекції: $R[A, B]$ та $R[B, C]$. Якщо отримані в результаті відношення все ще не перебувають у третій нормальній формі, до них знову застосовується зазначене правило.

Третя нормальна форма вимагає відсутності транзитивної залежності саме *непервинних*, а не всіх атрибутів відношення. Тому відношення НАВЧАННЯ, функціональні залежності якого зображені на рис. 6.3, перебуває в 3НФ

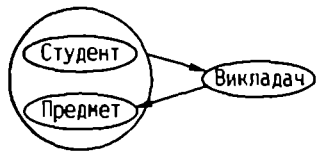



**Рис. 6.3.** Відношення НАВЧАННЯ в 3НФ

Відношенню в 3НФ також властиві аномалії додавання, видалення й оновлення. У зв'язку з цим вводиться поняття посиленої 3НФ і нормальної форми Бойса-Кодда.

Якщо відношення перебуває в третій нормальній формі та не містить неповних і транзитивних залежностей будь-яких атрибутів від можливих ключів, то воно перебуває в *посиленій 3НФ*

Реляційне відношення перебуває в *нормальній формі Бойса-Кодда* (НФБК), коли воно має таку властивість: якщо довільний атрибут відношення, що не належить набору атрибутів $A$, залежить функціонально від $A$, то й усі атрибути відношення функціонально залежать від $A$.

Поняття посиленої 3НФ і НФБК еквівалентні, й тому їх часто вживають як синоніми.

Алгоритм зведення до посиленої третьої нормальної форми такий самий, як і до 3НФ. Наприклад, у результаті нормалізації зображеного на рис. 6.3 відношення (воно не перебуває у посиленій 3НФ) буде отримано два такі відношення: ВИКЛАДАННЯ(Студент, Викладач) і ДИСЦИПЛІНА(Викладач, Предмет).

## 6.3. Нефункціональні залежності

Найбільш сильним критерієм якості схеми реляційної бази даних, який можна сформулювати виходячи з поняття функціональної залежності, є нормальна форма Бойса-Кодда. Проте між атрибутами відношення можуть існувати не лише функціональні залежності. Узагальненням цього поняття є багатозначна залежність, через яку означується четверта нормальна форма. Так само, узагальненням поняття багатозначної залежності є залежність за з'єднанням – це поняття застосовується в означенні п'ятої нормальної форми. 5НФ є остаточною щодо операції проекції, тобто якщо відношення перебуває у 5НФ, воно не містить аномалій, які можуть бути усунені шляхом його поділу на проекції. Це не означає, що відношення у 5НФ зовсім не містить аномалій. Тому існують й інші нормальні форми, які використовуються рідше і розгляд яких виходить за межі завдань, що розглядаються в даному посібнику.

### 6.3.1. Багатозначні залежності

У реляційній моделі функціональних залежностей недостатньо для відображення всіх семантичних особливостей предметних областей. У зв'язку з цим було введено поняття *багатозначної залежності*. Розглянемо це поняття на прикладі відношення ЛЕКЦІЯ з атрибутами Предмет, Викладач, Підручник (рис. 6.4). Воно перебуває в 3НФ, оскільки до складу його ключа входять усі три атрибути, між якими немає функціональних залежностей. Воно перебуває також у НФБК Інтерпретація цього реляційного відношення є такою: «Даний предмет викладають певні викладачі й для читання лекцій з цього предмета рекомендовані вказані підручники».


**Рис. 6.4.** Відношення ЛЕКЦІЯ в НФБК

Ми не випадково зобразили атрибути відношення ЛЕКЦІЯ у вигляді ієрархічної структури, оскільки саме вона найадекватніше відображує зміст зв'язків між цими атрибутами. У нас немає конкретної інформації про зв'язок викладачів з підручниками, ми не знаємо за яким підручником той чи інший викладач викладає конкретну дисципліну. Проте під час наповнення цієї таблиці ми маємо в кожному її рядку вказати значення всіх трьох атрибутів, оскільки ці атрибути є ключем, а значення ключових атрибутів завжди мають бути визначеними. Отже, така ситуація є аномальною.

Дану проблему можна вирішити так: якщо з предметом, наприклад «БД», пов'язані викладачі Іванов і Петров та підручники «Основи БД», «Вступ до БД» і «Теорія БД», то у відношенні кожний із цих викладачів повинен бути пов'язаний (асоційований) з кожним підручником.

ЛЕКЦІЯ

| Предмет | Викладач | Підручник |
|---|---|---|
| БД | Іванов | Основи БД |
| БД | Іванов | Вступ до БД |
| БД | Іванов | Теорія БД |
| БД | Петров | Основи БД |
| БД | Петров | Вступ до БД |
| БД | Петров | Теорія БД |

Інакше кажучи, ми повинні констатувати таку семантику зв'язку між викладачами і підручниками: «Цей предмет викладають такі викладачі й для читання лекцій з цього предмета всі вони використовують усі вказані підручники».

Таким чином, якщо в предметній області відсутній безпосередній зв'язок між атрибутами *A* і *B*, але необхідно зафіксувати такий зв'язок у реляційному відношенні, то єдиним коректним рішенням є відображення того факту, що всі значення атрибута *A* пов'язані з усіма значеннями атрибута *B* і навпаки.

Незалежність викладачів від підручників і навпаки, а також тлумачення такої незалежності згідно з наведеною вище тезою свідчить, що у відношенні ЛЕКЦІЯ множина підручників, пов'язаних із конкретним предметом, збігається з множиною підручників, які пов'язані з парою значень (Предмет, Викладач). Так, із предметом «Бази даних» пов'язані три підручники, ці ж три підручники пов'язані з парами значень (Бази даних, Іванов) і (Бази даних, Петров). І навпаки, множина викладачів, пов'язаних із конкретним предметом, збігається з множиною викладачів, пов'язаних із парою значень (Предмет, Підручник).

Таке реляційне відношення надлишкове і спричиняє певні труднощі під час виконання операцій маніпулювання даними. Прізвище кожного лектора є значенням атрибута Викладач стільки разів, скільки є підручників, і про це слід пам'ятати, виконуючи операції додавання, видалення й заміни викладачів. Ті самі проблеми мають місце і для підручників

У зв'язку з цим було введено поняття багатозначної залежності.

Нехай задано реляційне відношення *R* з атрибутами (або наборами атрибутів) *A*, *B*, *C*. Набір атрибутів *B* ***багатозначно залежить*** від *A* (або *A* багатозначно визначає *B*), і це позначається як $A \rightarrow\!\!\!\rightarrow B$, якщо для будь-яких заданих значень атрибутів *A* існує множина зв'язаних значень атрибутів *B* і склад цієї множини не залежить від значень атрибутів *C*.

Наприклад, у відношенні ЛЕКЦІЯ атрибути Викладач і Підручник багатозначно залежать від атрибута Предмет (Предмет $\rightarrow\!\!\!\rightarrow$ Викладач, Предмет $\rightarrow\!\!\!\rightarrow$ Підручник).

Дамо більш строге означення поняття багатозначної залежності.

Нехай реляційне відношення *R* означено на множині атрибутів *M*. Припустимо також, що *A* та *B* – підмножини *M*, що можуть перетинатися. Позначимо через

$B_R(a)$ множину проекцій за атрибутами $B$ тих кортежів відношення $R$, проекція яких за атрибутами $A$ дорівнює $a$, тобто:

$$B_R(a) = \{b \mid \exists\, r \in R,\ r[A] = a \ \&\ r[B] = b\}.$$

Нехай $C = M \quad (A \cup B)$. Тоді в $R$ має місце багатозначна залежність $A \to\!\!\to B$, якщо

$$\forall ac \in R[A.C]\ B_R(ac) = B_R(a), \text{ де } a \in R[A].$$

Іншими словами, сукупність значень атрибутів $B$, яка належить кортежам відношення $R$ із заданим значенням $a$ атрибутів $A$, присутня також у множині кортежів відношення $R$ із заданими значеннями атрибутів $A$ і $C$. Отже, множина значень $B$ для заданого $a$ не залежить від значень $C$.

Будь-яка функціональна залежність є багатозначною, але не навпаки. Функціональна і багатозначна залежності відрізняються, адже функціональна залежність $A \to B$ визначається тільки через $A$ і $B$, існування ж багатозначної залежності $A \to\!\!\to B$ є властивістю всієї сукупності атрибутів $R$.

### Аксіоми багатозначної залежності

Наведемо аксіоми, якими визначаються властивості багатозначних залежностей.

Нехай реляційне відношення $R$ складається з атрибутів (або набору атрибутів) $A$, $B$, $C$, причому $C = M - (A \cup B)$, де $M$ – уся множина атрибутів $R$. Тоді справедливими є такі аксіоми:

1) Якщо $A \to\!\!\to B$, то $A \to\!\!\to C$ – доповнення.

2) Якщо $A \to\!\!\to B$ і $V \subseteq W$, то $A \cup W \to\!\!\to B \cup V$ – поповнення.

3) Якщо $A \to\!\!\to B$ і $B \to\!\!\to C$, то $A \to\!\!\to C$ – транзитивність.

Існують дві аксіоми, що пов'язують багатозначні й функціональні залежності.

4) Якщо $A \to B$, то $A \to\!\!\to B$ – реплікація

5) Якщо $A \to\!\!\to B$ і $Z \subseteq B$, і для деякого $W$, що не перетинається з $B$, маємо $W \to Z$, то $A \to Z$ – з'єднання.

Нарешті, багатозначна залежність має такі додаткові властивості (тут літерами $A$, $B$, $C$ позначені довільні множини атрибутів певного відношення $R$).

6) Якщо $A \to\!\!\to B$ і $A \to\!\!\to C$, то $A \to\!\!\to B \cup C$ – об'єднання.

7) Якщо $A \to\!\!\to B$ і $W \cup B \to\!\!\to Z$, то $W \cup A \to\!\!\to Z - (W \cup B)$ – псевдотранзитивність.

8) Якщо $A \to\!\!\to B$ і $A \cup B \to\!\!\to C$, то $A \to\!\!\to C - B$ – змішана псевдотранзитивність.

9) Якщо $A \to\!\!\to B$ і $A \to\!\!\to C$,
то $A \to\!\!\to B \cap C$, $A \to\!\!\to B - C$, $A \to\!\!\to C - B$ – перетин і різниця.

### Четверта нормальна форма

Узагальнення нормальної форми Бойса-Кодда, що називається четвертою нормальною формою, базується на понятті багатозначної залежності. Перш ніж сформулювати означення 4НФ, введемо поняття тривіальної багатозначної залежності.

Багатозначна залежність $A \rightarrow\!\!\!\rightarrow B$ у відношенні $R$ називається *тривіальною*, якщо $B \subseteq A$ або $R$ визначено на множині атрибутів $A \cup B$. Тривіальні багатозначні залежності властиві будь-яким відношенням.

Реляційне відношення $R$ перебуває в *четвертій нормальній формі* – 4НФ, якщо з існування в ньому нетривіальної багатозначної залежності $X \rightarrow\!\!\!\rightarrow B$ випливає, що $X$ є суперключем $R$.

Нагадаємо, що множина $X$ атрибутів відношення $R$ називається суперключем. якщо від $X$ функціонально залежать усі атрибути $R$.

Теорема, доведена Р. Фейджином, вказує спосіб зведення відношення до 4НФ.

**Теорема Фейджина**

Нехай реляційне відношення $R$ складається з атрибутів (або множин атрибутів) $A$, $B$, $C$. Залежність $A \rightarrow\!\!\!\rightarrow B$ має місце у відношенні $R$ тоді й лише тоді, коли $R = R[A, B] * R[A, C]$ ($R$ є природним з'єднанням проекцій $R[A, B]$ та $R[A, C]$).

**Алгоритм зведення до 4НФ**

Припустимо, що задане відношення $R$ з атрибутами $A$, $B$, $C$, де $A$ не є суперключем $R$, і є багатозначна залежність $R.A \rightarrow\!\!\!\rightarrow R.B$. Тоді відношення $R$ підлягає декомпозиції на два відношення: $R[A. B]$ та $R[A. C]$. Якщо отримані відношення все ще не перебувають у 4НФ, до них знову застосовується це правило.

**Приховані багатозначні залежності**

Під час зведення схеми відношення $R$ до четвертої нормальної форми слід пам'ятати, що відношення може містити приховану багатозначну залежність, тобто таку залежність, яка відсутня в самому відношенні, але може існувати в його проекціях за деякими атрибутами (для функціональних залежностей така ситуація неможлива). Тобто під час декомпозиції відношення за багатозначною залежністю слід проводити подальший аналіз отриманих відношень на предмет наявності в них нових багатозначних залежностей

## 6.3.2. Залежності за з'єднанням

Нехай $M_1, M_2, ..., M_n$ – довільні множини атрибутів $R$. Відношення $R$ має *залежність за з'єднанням* щодо $M_1, M_2, ..., M_n$, що позначається як $*(M_1, M_2, ..., M_n)$, якщо воно дорівнює природному з'єднанню його проекцій за $M_1, M_2, ..., M_n$, тобто $R = R[M_1] * R[M_2] *...* R[M_n]$.

Залежність за з'єднанням є *тривіальною*, якщо $M_i = R$ для певного $i$.

Для функціональних і багатозначних залежностей існують системи аксіом, які мають властивість повноти. На жаль, для залежності за з'єднанням повна система аксіом відсутня.

Кожна залежність за з'єднанням вигляду $*(M_1, M_2)$ у відношенні зі схемою $R(M_1, M_2)$ є еквівалентною багатозначним залежностям $R[M_1 \cap M_2] \rightarrow\!\!\!\rightarrow R[M_1]$ та $R[M_1 \cap M_2] \rightarrow\!\!\!\rightarrow R[M_2]$. Проте існують залежності за з'єднанням, які не є еквівалентними багатозначним залежностям. Найпростішим прикладом залежності за з'єднанням на схемі $R = (A, B, C)$ є $*(A \cup B, B \cup C, A \cup C)$.

Наведена далі таблиця містить залежність за з'єднанням $*(A \cup B, B \cup C, A \cup C)$, що можна перевірити, обчисливши вираз $R[A, B] * R[A, C] * R[B, C]$

Жодної нетривіальної багатозначної залежності в ній немає. У цьому можна переконатися, перевіривши, що жодна із залежностей $A \to\to B$, $A \to\to C$, $B \to\to A$, $B \to\to C$, $C \to\to A$, $C \to\to B$ не задовольняється.

| $A$ | $B$ | $C$ |
|---|---|---|
| $a_1$ | $b_1$ | $c_2$ |
| $a_2$ | $b_1$ | $c_1$ |
| $a_1$ | $b_2$ | $c_1$ |
| $a_1$ | $b_1$ | $c_1$ |

Між багатозначною залежністю і залежністю за з'єднанням існує така аналогія: якщо багатозначна залежність встановлює відсутність безпосередньої залежності між двома парами атрибутів (множин атрибутів), то залежність за з'єднанням вказує на незалежність деякої сукупності атрибутів (множини атрибутів).

### П'ята нормальна форма

Реляційне відношення $R$ перебуває в *п'ятій нормальній формі* (5НФ) тоді й тільки тоді, коли в кожній його нетривіальній залежності за з'єднанням $*(M_1, M_2, ..., M_n)$ будь-яка множина $M_i$ є можливим ключем $R$. Цю нормальну форму також називають *проекційно-з'єднувальною нормальною формою.*

Оскільки будь-яка пара багатозначних залежностей з однаковою лівою частиною є також залежністю за з'єднанням, то будь-яка схема відношення, що перебуває в 5НФ, також перебуває і в 4НФ.

## 6.4. Проектування схеми реляційної бази даних

Теорія проектування реляційних баз даних ґрунтується на тезі, що будь-яка предметна область може бути зображена у вигляді одного універсального реляційного відношення, яке містить усю множину атрибутів предметної області.

Таке відображення предметної області є надто суперечним з погляду на доцільність його практичного використання. Але ми цю тезу будемо використовувати з теоретичної точки зору, як підґрунтя для формальних міркувань, погоджуючись, що на практиці вона буде застосовуватись тоді, коли це буде доцільно.

Формалізуємо задачу проектування реляційної схеми. Нехай задано реляційну схему $S_0$ одного (універсального) реляційного відношення $R$:

$$S_0 = \{R = <U, G>\},$$

де $U$ – множина атрибутів, а $G$ – множина функціональних і багатозначних залежностей. Необхідно знайти еквівалентну реляційну схему сукупності відношень $R_1, R_2, ..., R_n$:

$$S_D = \{R_i = <U_i, G_i>, i = 1, 2, ..., n\},$$

яка була б у певному розумінні кращою за схему $S_0$.

У цьому визначенні слід уточнити, чим є процедура зображення відношень у вигляді сукупності інших, що таке «еквівалентні реляційні схеми» та що означає «одна реляційна схема краща за іншу». (Ці уточнення здійснюються нижче )

## 6.4.1. Процедура декомпозиції схеми реляційного відношення

*Декомпозицією* реляційного відношення $R$ зі схемою $R(M)$, де $M$ – множина атрибутів, називається процедура поділу $R$ на множину відношень $R_1$, $R_2$, ..., $R_n$ зі схемами $R_1(M_1)$, $R_2(M_2)$, ..., $R_n(M_n)$, що відповідають таким умовам:

- $R_1 \cup R_2 \cup ... \cup R_n = R$. Інакше кажучи, будь-який атрибут з $R$ має міститися принаймні в одному з відношень зі схемою $R_i$, а множина атрибутів будь-якого $R_i$ має бути підмножиною атрибутів $R$.
- Усі відношення $R_i$, $i = 1, 2, ..., n$, мають бути проекціями відношення $R$ на атрибути, що містяться в $R_i$, тобто $R_i = R[M_i]$, $i = 1, 2, ..., n$.

Ми припускаємо, що поділ універсального відношення здійснюється саме шляхом його декомпозиції.

Нехай задано реляційне відношення $R$, після декомпозиції якого отримані відношення $R_1$, $R_2$, ..., $R_n$. Ця декомпозиція має властивість *з'єднання без втрат*, якщо $R$ є природним з'єднанням відношень $R_1$, $R_2$, ..., $R_n$, тобто $R = R_1 * R_2 *...* R_n$.

Бажано, щоб декомпозиція мала властивість з'єднання без втрат. Це є гарантією того, що будь-яке відношення може бути відновлене з його проекцій.

Інша бажана властивість декомпозиції полягає в тому, що множина залежностей $F$ (функціональних і багатозначних) для відношення $R$ має виводитися з проекцій, що були отримані в результаті декомпозиції.

Зауважте, що декомпозиція може мати властивість з'єднання без втрат, не зберігаючи при цьому залежності.

## 6.4.2. Еквівалентність відношень

Є різні підходи до означення еквівалентності. Неформально дві сукупності відношень називаються еквівалентними, якщо відображують одну й ту саму інформацію. Точне тлумачення поняття еквівалентності залежить від того, що розуміти під інформацією, яка відображується відношеннями. Дамо такі два означення еквівалентності.

1. Дві сукупності відношень відображують одну й ту саму інформацію, якщо вони визначені на одних і тих самих атрибутах і в них збережені всі залежності даних (функціональні й багатозначні). Це так звана *еквівалентність за залежностями*. Тобто якщо $S_0 = \{R = <U, G>\}$ – початкова схема, а $S_D = \{R_i = <U_i, G_i>, i = 1, 2, ..., n\}$ – кінцева (див. означення цих схем вище), то

$$U = \bigcup_{i=1}^{n} U_i, \quad G^+ = \left( \bigcup_{i=1}^{n} G_i \right)^+.$$

Отже, множина залежностей відношення $R$ породжується множинами залежностей проекцій $R_i$.

2. Дві сукупності відношень відображують одну й ту саму інформацію, якщо вони містять одні й ті самі дані.

   У нашому випадку під час декомпозиції $R$ на $R_1$, $R_2$, ..., $R_n$ початкове і кінцеві відношення містять одні й ті самі дані, якщо ця декомпозиція має властивість з'єднання без втрат. Це так звана *еквівалентність за даними*.

### Декомпозиція без втрат даних

Якщо в результаті декомпозиції відношення $R(U)$ зі збереженням функціональної (багатозначної) залежності утворюються відношення $R_1(U_1)$ та $R_2(U_2)$, то такий поділ забезпечує з'єднання без втрат тоді й тільки тоді, коли:

$$U_1 \cap U_2 \rightarrow (\rightarrow\!\!\!\rightarrow)\ U_1 - U_2$$

або

$$U_1 \cap U_2 \rightarrow (\rightarrow\!\!\!\rightarrow)\ U_2 - U_1.$$

У разі, коли потрібно здійснити декомпозицію відношення на більше, ніж два нових, можна застосовувати *метод таблиці* (в цьому методі використовується лише поняття функціональної залежності).

Нехай задано: множина функціональних залежностей та початкове й кінцеві відношення. Процедура полягає в побудові таблиці, рядки якої відповідають отриманим у результаті поділу відношенням, а стовпці – атрибутам початкового відношення $(A_1, ..., A_n)$. Комірка заповнюється символами $a_j$, якщо елемент, розташований в $i$-му рядку й $j$-му стовпці, відповідає атрибуту $A_j$ відношення $R_i$, інакше в комірці записується $b_{ij}$. Після заповнення таблиці здійснюється перегляд усіх атрибутів, що розташовані в лівій частині кожної функціональної залежності $X \rightarrow Y$ Якщо для атрибутів з $X$ знайдуться рядки, де у відповідних місцях розташовані $a_j$, то елементи $b_{ik}$ цих рядків, відповідні стовпцям атрибутів з $Y$, замінюються на $a_k$ Якщо в результаті з'явиться рядок таблиці, повністю заповнений $a_j$, то з'єднання здійснено без втрат; у протилежному випадку – це з'єднання з втратами.

Нехай задано відношення $R$ $(A, B, C, D)$ і функціональні залежності $A \rightarrow C$, $B \rightarrow C$, $CD \rightarrow B$, $C \rightarrow D$. Після декомпозиції схема є такою: $R_1(A, B)$, $R_2(B, D)$, $R_3(A, B, C)$, $R_4(B, C, D)$

Початкова таблиця виглядає так:

| | $A$ | $B$ | $C$ | $D$ |
|---|---|---|---|---|
| $R_1$ | $a_1$ | $a_2$ | $b_{13}$ | $b_{14}$ |
| $R_2$ | $b_{21}$ | $a_2$ | $b_{23}$ | $a_4$ |
| $R_3$ | $a_1$ | $a_2$ | $a_3$ | $b_{34}$ |
| $R_4$ | $b_{41}$ | $a_2$ | $a_3$ | $a_4$ |

Розглянемо функціональну залежність $A \to C$. Таблиця перетвориться на таку:

| | $A$ | $B$ | $C$ | $D$ |
|---|---|---|---|---|
| $R_1$ | $a_1$ | $a_2$ | $a_3$ | $b_{14}$ |
| $R_2$ | $b_{21}$ | $a_2$ | $b_{23}$ | $a_4$ |
| $R_3$ | $a_1$ | $a_2$ | $a_3$ | $b_{34}$ |
| $R_4$ | $b_{41}$ | $a_2$ | $a_3$ | $a_4$ |

Розглянемо $B \to C$:

| | $A$ | $B$ | $C$ | $D$ |
|---|---|---|---|---|
| $R_1$ | $a_1$ | $a_2$ | $a_3$ | $b_{14}$ |
| $R_2$ | $b_{21}$ | $a_2$ | $a_3$ | $a_4$ |
| $R_3$ | $a_1$ | $a_2$ | $a_3$ | $b_{34}$ |
| $R_4$ | $b_{41}$ | $a_2$ | $a_3$ | $a_4$ |

Обробка залежності $CD \to B$ результатів не дає. Розглядаємо $C \to D$:

| | $A$ | $B$ | $C$ | $D$ |
|---|---|---|---|---|
| $R_1$ | $a_1$ | $a_2$ | $a_3$ | $a_4$ |
| $R_2$ | $b_{21}$ | $a_2$ | $a_3$ | $a_4$ |
| $R_3$ | $a_1$ | $a_2$ | $a_3$ | $a_4$ |
| $R_4$ | $b_{41}$ | $a_2$ | $a_3$ | $a_4$ |

Є рядок з усіма $a_j$. Отже, декомпозицію проведено без втрат.

### Декомпозиція зі збереженням залежностей

Здійснення декомпозиції без втрат даних не завжди гарантує збереження залежностей. І навпаки, не кожна декомпозиція, що зберігає залежності, зберігає можливість з'єднання без втрат. Проілюструємо це на прикладах.

Розглянемо відношення НАВЧАННЯ(Студент, Предмет, Викладач), яке ми наводили при обговоренні НФБК. Це відношення містить залежності (Студент, Предмет) → Викладач і Викладач → Предмет.

При поділі НАВЧАННЯ на відношення ВИКЛАДАННЯ(Студент, Викладач) і ДИСЦИПЛІНА(Викладач, Предмет) втрат даних не буде, оскільки поділ здійснено за залежністю Викладач → Предмет, проте залежність (Студент, Предмет) → Викладач у результаті буде втрачено.

Можна також навести інший приклад декомпозиції, що зберігає залежності, але не дані. Нехай у відношенні $R(A, B, C, D)$ є одна залежність $A \to B$, тоді декомпозиція відношення на $R_1(A, B)$ і $R_2(C, D)$ цю залежність збереже, але дані – ні (зв'язок між усіма чотирма атрибутами втрачається).

### Еквівалентність нормальних форм

Виявляється, що будь-яка схема відношення може бути зведена до НФБК так, щоб декомпозиція мала властивість з'єднання без втрат. Можливе також зведення до 3НФ без втрат даних та зі збереженням залежностей. Проте схема відношення може виявитися такою, що не зводиться до НФБК зі збереженням залежностей. Наприклад, відношення $R$ ($A$, $B$, $C$) з функціональними залежностями $(A, B) \rightarrow C$, $C \rightarrow B$, яке перебуває в 3НФ, може бути зведено до нормальної форми Бойса-Кодда в єдиний спосіб: $R_1$ ($A$, $C$) і $R_2$ ($B$, $C$). Тут еквівалентність за даними зберігається, а еквівалентність за функціональними залежностями – ні, оскільки втрачається залежність $(A, B) \rightarrow C$.

Отже, використовуючи нормальні форми під час проектування, слід пам'ятати, що до 3НФ перетворення можна виконувати зі збереженням еквівалентності за залежностями і даними. Проте під час отримання нормальної форми Бойса-Кодда може втрачатися еквівалентність за функціональними залежностями.

## 6.4.3. Критерій якості реляційної схеми

Спробуємо розібратися, що означає вислів «одна реляційна схема краща за іншу». Тут можна говорити про критерії незалежності відображень зв'язків і ненадлишковості даних і, як наслідок, про відсутність можливих аномалій маніпулювання даними. Фактично якість схеми можна оцінити за допомогою нормальних форм відношень. Що вищою є нормальна форма, в якій перебуває сукупність відношень, то більш незалежними є відображені в її відношеннях зв'язки між атрибутами. Бажано також, щоб кількість кінцевих реляційних відношень була щонайменшою

# Контрольні запитання та завдання

1. Дайте означення функціональної залежності та сформулюйте аксіоми, яким такі залежності відповідають.
2. Коли відношення перебуває у першій нормальній формі? Опишіть алгоритм зведення до 1НФ
3. Визначте неповну функціональну залежність і другу нормальну форму. Опишіть алгоритм зведення до 2НФ
4. Що таке третя нормальна форма? Опишіть алгоритм зведення до 3НФ.
5. Чим відрізняється 3НФ від НФБК?
6. Дайте означення багатозначної залежності та сформулюйте аксіоми, яким такі залежності відповідають.
7. Що таке четверта нормальна форма? Опишіть алгоритм зведення до 4НФ
8. Опишіть процес проектування схеми реляційної бази даних.
9. У чому полягає процедура декомпозиції схеми реляційних відношень?
10. До якої нормальної форми зберігається еквівалентність відношень за залежностями і даними?

# Розділ 7

# Проектування баз даних

- Методологія проектування бази даних
- Етапи проектування бази даних
- ER-моделювання предметної області

## 7.1. Методологія проектування бази даних

Процес створення такої структури бази даних, яка б відповідала вимогам користувачів, називається проектуванням бази даних. Його можна порівняти зі зведенням нової будівлі: визначення вимог, проектування, конструювання і, нарешті, реалізація.

*Життєвий цикл системи баз даних* є концепцією, в межах якої корисно й зручно розглядати розвиток такої системи. Він, як і життєвий цикл будь-якої програмної системи, складається з двох основних фаз: *проектування* та *реалізації*

Фаза проектування поділяється на такі етапи:

- визначення стратегії;
- аналіз предметної області;
- концептуальне моделювання;
- логічне й фізичне проектування.

Фаза реалізації складається з таких пунктів:

- власне програмна реалізація;
- документування;
- дослідне впровадження;
- промислова експлуатація.

*Методологія проектування баз даних* – це сукупність принципів, методів, інструментів і засобів, що застосовуються для послідовного розроблення структури бази даних. Оскільки система баз даних складається з програм і даних, методологія проектування баз даних розглядається як невід'ємна частина загальної методології проектування програмних систем.

До методології проектування баз даних висуваються певні вимоги. Прийнятною вважається база даних, яка відповідає вимогам користувачів (ефективність, адаптивність, незалежність, захищеність, цілісність тощо) і вимогам до апаратного забезпечення. Методологія має бути достатньо гнучкою, доступною розробникам

із різним досвідом проектування, що використовують різні моделі даних і різне програмне забезпечення СКБД.

Методологія проектування баз даних визначає:

- процес проектування;
- методику виконання розрахунків і критеріїв оцінювання альтернативних рішень на кожному етапі проектування;
- інформаційні вимоги як вихідні дані для процесу проектування;
- засоби опису вихідних даних і відображення результатів кожного етапу проектування.

### Процес проектування

Для баз даних можна застосувати ітеративне низхідне проектування. Процес проектування добре структурований, оскільки кожний його етап завершується певним результатом, а також тому, що допускається ітеративне повторення попередніх етапів, якщо отриманий результат не відповідає вимогам замовника або системним вимогам. Це дає можливість переглядати й змінювати проектні рішення на будь-якому етапі.

З проектуванням тісно пов'язане експертне оцінювання проекту. Мета експертизи – знайти помилки й виправити їх на ранніх етапах проектування. Зазвичай експертиза виконується після завершення кожного з етапів.

### Критерії оцінювання

Оцінювання необхідне для ухвалення рішень за наявності альтернатив. Труднощі у визначенні критеріїв і виборі альтернатив пов'язані з тим, що часто розробляється кілька проектів структури бази даних і потрібно оцінити, який з них є кращим. Зробити це буває досить складно.

Критерії є кількісні (час обробки запитів, вартість операцій маніпулювання даними, витрати пам'яті тощо) та якісні (гнучкість, адаптивність, сприйнятливість та сумісність).

### Інформаційні вимоги

Визначаючи вимоги до інформації, врахуйте, що є інформація, яка стосується структури даних (опис даних та зв'язків безвідносно до конкретних способів їхнього використання й обробки), та інформація про спосіб використання даних (опис вимог до обробки даних).

### Засоби опису

Це мовні засоби, призначені для опису результатів виконання кожного етапу проектування. А саме, йдеться про такі засоби.

- Природна мова, якою строго означуються всі необхідні для опису результатів проектування поняття. Використовується, як правило, на етапі визначення стратегії.
- Стандартні форми, анкети та бланки. Використовуються переважно на етапі аналізу.

- Спеціальні формалізовані мови концептуального моделювання (семантичні мережі, числення предикатів та ER-мови). Використовуються переважно на етапі концептуального моделювання.
- Формалізовані мова означення даних (МОД) і мова маніпулювання даними (ММД). Використовуються на етапі логічного проектування. Зазвичай з цією метою застосовують мову SQL.

## 7.2. Етапи проектування бази даних

У даному підрозділі будуть детально розглянуті всі чотири етапи, на які поділяється фаза проектування бази даних, від визначення стратегії до логічного й фізичного проектування. Припускатимемо, що база даних розробляється для певної організації-замовника.

### 7.2.1. Визначення стратегії

Метою етапу визначення стратегії є формування спільно з замовником прикладних моделей, вироблення переліку рекомендацій і ухвалення узгодженого плану, складеного з урахуванням наявних організаційних, фінансових і технічних обмежень, що відображує як поточні, так і майбутні потреби організації.

#### Опис

Детальний аналіз структури організації може бути початковою базою для розроблення перспективного плану створення системи, але витрати на його проведення навряд чи будуть економічно виправданими. Як правило, стратегія розроблення інформаційної системи визначається в результаті узагальненого аналізу, на підставі якого потім будується великомасштабна модель прикладної області. Стратегія має визначатися в достатньо стислі терміни з тим, щоб результати проектування не втрачали актуальності.

Результати цього етапу мають узгоджуватися одне з одним і бути достатньо чітко сформульовані, щоб замовник міг легко співвіднести запропоновану стратегію зі своїми завданнями і зрозуміти, які саме чинники обумовили ухвалення тих чи інших рішень. Окрім того, йому має бути викладена перспектива подальшого аналізу, уточнення й перегляду стратегічних рішень.

#### Результати

Основними результатами цього етапу мають бути

- опис напрямів прикладної діяльності, зокрема формулювання її цілей і завдань, визначення пріоритетів, обмежень, критичних чинників успіху та ключових показників ефективності;
- опис цілей і завдань автоматизації, витрат і можливого виграшу;
- узагальнена діаграма сутностей і зв'язків;
- узагальнена ієрархічна схема завдань (виробничих та управлінських);

- рекомендації щодо майбутньої реалізації та подолання можливих труднощів;
- визначення меж і окреслення сфери застосування системи баз даних;
- можлива архітектура системи;
- поетапний план проектування бази даних

Такий підхід до моделювання предметної області передбачає її відображення з трьох різних точок зору:

- загальний напрям прикладної діяльності;
- прикладні завдання;
- інформаційні потреби.

Побудовані на цьому етапі моделі мають бути зрозумілими для замовника, а для того щоб досягти повного узгодження різних точок зору на прикладну область і можливі напрями діяльності, проводяться групові координаційні наради

Стратегії еволюціонують і розвиваються, обставини й завдання з часом можуть змінюватися, відтак неможливо запропонувати директивний метод моделювання стратегій. Тому важливо поєднувати неупередженість до нових рішень зі здатністю швидко оцінювати альтернативні напрямки діяльності з урахуванням заданих обмежень і пріоритетів.

#### Ключові чинники успіху

На першому етапі слід виділити насамперед такі чинники успіху:

- використання всіх можливих засобів, що дають змогу підвищити рівень знань про предметну область;
- активна участь у розробленні стратегії осіб, які добре розуміють справжні потреби організації;
- проведення плідних нарад із ретельним розглядом усіх питань

### 7.2.2. Аналіз предметної області

Підсумки етапу визначення стратегії є вихідними даними для етапу аналізу, де вони ретельно перевіряються, уточнюються і деталізуються, для того щоб забезпечити предметній області адекватність моделі, гарантувати можливість реалізації рішень і сформувати тверде підґрунтя для етапів концептуального моделювання, логічного й фізичного проектування.

Цей етап є найменше вивченим, найважчим і найтривалішим. Проте він найважливіший, оскільки саме на ньому формується більшість проектних рішень

#### Опис

Аналіз предметної області складається з аналізу даних та аналізу завдань. Аналіз даних передбачає документування всіх атрибутів. Аналіз завдань може потребувати застосування різноманітних методів побудови діаграм для дослідження зв'язків і способів використання даних, подій, станів даних, а також детального опису алгоритмів.

Вивчається потреба в заходах із контролю та захисту даних, їхньому резервному копіюванні та відновленні. Має бути проведений детальний аналіз наявних систем та інших чинників, що впливають на процес впровадження системи. Потрібно виявити всі обмеження і припущення, що можуть вплинути на подальше проектування, використання ресурсів і терміни проведення робіт.

### Підхід

На цьому етапі аналітики й користувачі працюють пліч-о-пліч, встановлюючи й перевіряючи вимоги. Аналіз предметної області передбачає:

- проведення бесід з користувачами;
- перегляд усіх документів та бланків, які обробляються і формуються організацією;
- аналіз потоків документів;
- аналіз способів вирішення завдань організації;
- фіксація правил, обмежень та законів, що діють у предметній області.

### Результати

До ключових результатів етапу аналізу належать:

- узгоджена діаграма сутностей і зв'язків;
- відомості про обсяги даних, частоту виконання завдань, очікуваний користувачем рівень продуктивності;
- деталізовані й узгоджені описи завдань;
- первинний варіант стратегії впровадження;
- опис заходів з ревізії і контролю даних, резервного копіювання й відновлення;
- загальний опис процедур, що не автоматизуються;
- критерії прийнятності, якості, гнучкості та продуктивності;
- попереднє оцінювання обсягів системи;
- узгоджений підхід до здійснення етапу проектування й фази реалізації;
- уточнений план розроблення системи.

### Ключові чинники успіху

- активна участь користувачів;
- ретельна перевірка достовірності, повноти й несуперечності даних;
- виявлення всіх питань та припущень, що мають ключове значення для проектування і впровадження;
- встановлення точних характеристик ключових завдань і даних;
- жорсткий контроль за ходом робіт, концентрація зусиль на виконанні календарних планів і дотриманні запланованих термінів.

## 7.2.3. Концептуальне моделювання предметної області

Етап концептуального моделювання полягає в побудові опису предметної області в термінах формальної мови, наприклад у термінах моделі сутностей і зв'язків. Ідеї побудови концептуальної моделі предметної області беруть свій початок із публікації робочої групи ANSI/SPARC, присвяченій архітектурі СКБД.

### Опис

На базі змістовного опису предметної області, отриманого в результаті її аналізу, розроблюється строгий формальний опис її інформаційного забезпечення.

### Результати

До ключових результатів етапу концептуального моделювання належать:

- формальний опис інформаційного забезпечення предметної області;
- докладний і строгий опис сховищ даних;
- детальний опис потоків даних;
- детальний опис ієрархії й специфікація завдань, що вирішуються;
- детальний опис чинних у предметній області правил і обмежень.

### Ключові чинники успіху

До ключових чинників успіху належать:

- глибоке знання і практичний досвід використання мов опису концептуальної моделі,
- знання методів проектування реляційної моделі та/або інших моделей даних.

## 7.2.4. Логічне й фізичне проектування

Етап проектування полягає в пошуку і визначенні якнайкращого способу реалізації та виконання вимог, сформульованих на етапі аналізу. При цьому має забезпечуватись належний рівень сервісу в умовах певного технологічного середовища, що відповідає ухваленим рішенням щодо рівня автоматизації.

*Логічне проектування* – це розроблення структур зберігання, методів доступу й логічної структури системи баз даних без прив'язки до конкретної СКБД.

*Фізичне проектування* – це проектування бази даних у конкретній СКБД

### Опис

Під час виконання цього етапу модель сутностей і зв'язків перетворюється на схему бази даних і специфікації зберігання даних на зовнішніх носіях. Прикладні задачі перетворюються на модулі й процедури з необхідними засобами ревізії/контролю та резервного копіювання й відновлення. Проектуються формати звітів, визначаються міжмодульні зв'язки. Виходячи із завдань, сформульованих на попередніх етапах, створюються проектні рішення з архітектури комунікаційної мережі. Для полегшення процесу пошуку проектних рішень можуть застосовуватися прототипи. Нарешті, на етапі проектування розроблюються програмні специфікації і план тестування системи, а отримана інформація й нові погляди на майбутню систему застосовуються для доопрацювання та уточнення стратегії її реалізації.

**Підхід**

Аналітики, розробники і проектувальники баз даних спілкуються з користувачами менше, ніж на етапі аналізу, проте вони повинні надати їм для перевірки результати своєї роботи або запропонувати на вибір різні варіанти проектних рішень. Корисним є створення прототипів, проте воно має розглядатися лише як засіб, а не самоціль.

**Результати**

До ключових результатів етапу проектування належать:

- архітектура системи;
- схеми системних модулів;
- логічна і фізична схеми;
- схема бази даних і файлів;
- детальні часові та ємнісні характеристики;
- програмні специфікації;
- специфікації неавтоматизованих процедур;
- чорновий варіант посібника для користувача;
- узгоджена стратегія впровадження, що складається з планів приймання і здачі системи, організаційної підготовки, заходів зі збирання даних, переходу на нову систему та встановлення обладнання;
- план випробувань системи;
- чорновий варіант експлуатаційної документації;
- уточнений план розроблення системи.

**Ключові чинники успіху**

У результаті виконання даного етапу має бути створений проект, що забезпечує задоволення прикладних вимог з урахуванням наявних технічних обмежень. Ключовими чинниками успіху в досягненні цієї мети є:

- знання можливостей апаратного й програмного забезпечення;
- розуміння прикладних потреб;
- ухвалення обґрунтованих компромісних рішень;
- виявлення і вирішення потенційних проблем.

## 7.3. ER-моделювання предметної області

Мова ER-моделювання (від англ. Entity-Relationship – сутність-зв'язок) - це графічна мова, призначена для опису інформаційних потреб організації. Мова базується на концепції, згідно з якою інформаційне забезпечення будь-якої предметної області зображується як сукупність взаємозв'язаних об'єктів Процес моделювання полягає у виділенні об'єктів (сутностей предметної області), визначенні їхніх властивостей і виявленні зв'язків між ними.

Моделювання сутностей і зв'язків, що здійснюється переважно на етапах розроблення стратегії, аналізу і концептуального моделювання, має за основну мету створення точної та адекватної моделі інформаційних потреб організації.

## 7.3.1. Основні поняття

Далі будуть наведені означення базових понять, розглянуті основні властивості та формальні позначення сутностей, зв'язків, атрибутів, а також надані рекомендації і правила щодо креслення ER-діаграм.

### Сутність

*Сутність* — це реальний або уявний об'єкт, інформація про який має бути зібрана чи збережена. Графічно сутність зображується поіменованим прямокутником із заокругленими кутами. *Ім'я сутності* подається в однині й пишеться великими літерами (рис. 7.1, *а*).

Прямокутник, що зображує сутність, може бути будь-якої форми і розміру, основні вимоги — достатність місця для однозначного відображення імені (бажано не вживати скорочень) і зручність креслення ER-діаграми. Часто буває зручно подовжити прямокутник, щоб можна було провести до нього більше ліній зв'язку, уникаючи зайвих перетинів.

Ім'я сутності має бути таким, щоб посилатися на тип або клас об'єктів, а не на окремий екземпляр. У нашому прикладі Хітроу або Орлі не можуть бути іменами сутностей, сутність — це АЕРОПОРТ, а згадані назви іменують екземпляри сутності (рис. 7.1, *б*).

Якщо різні слова з одним і тим самим значенням використовуються як імена сутності, то можна застосовувати синоніми. Одне з імен вибирається як первинне, а його синоніми записуються через похилу риску.


*а* *б*

**Рис. 7.1.** Графічне позначення сутності: загальний вигляд (*а*), сутність та її екземпляри (*б*)

Перелічимо найважливіші властивості сутностей.

- Будь-який предмет або об'єкт може бути відображений лише однією сутністю, тобто сутності завжди є взаємовиключними.
- Кожна сутність має бути унікально ідентифікована, тобто має існувати спосіб незалежної ідентифікації кожного екземпляра сутності, що дає змогу відрізняти його від інших її екземплярів.

### Зв'язок

*Прикладний зв'язок*, або просто *зв'язок* — це поіменована асоціація двох або більшої кількості сутностей.

Зв'язок двох сутностей або сутності з самою собою називається *бінарним*. Зазначимо, що будь-який зв'язок між більшою кількістю сутностей на стадії логічного проектування має бути змодельований за допомогою додаткової таблиці, яка з'єднується з іншими бінарними зв'язками. Тому надалі розглядатимемо лише бінарні зв'язки. З кожного боку бінарний зв'язок має такі характеристики:

- ім'я;
- множинність, або потужність;
- обов'язковість – зв'язок може бути обов'язковим або факультативним.

Розрізняють дві *множинності зв'язку* – «один» і «багато». Якщо зв'язок між сутностями *A* і *B* з боку сутності *A* має множинність «один», то це означає, що кожний екземпляр *B* асоціюється даним зв'язком не більше ніж з одним екземпляром *A*. І навпаки, якщо екземпляр *B* може асоціюватися певним зв'язком із довільною кількістю екземплярів *A*, то зв'язок з боку сутності *A* має множинність «багато».

Зв'язок між сутностями *A* і *B* є *обов'язковим* з боку сутності *A*, якщо кожен її екземпляр повинен асоціюватися даним зв'язком з певним екземпляром (певними екземплярами) сутності *B*. Якщо участь у зв'язку екземплярів *A* не є обов'язковою, зв'язок з боку *A* називається *факультативним*

На діаграмах бінарні зв'язки зображуються лініями, що сполучають два прямокутники сутностей або рекурсивно один і той же прямокутник з самим собою. Множинність «один» зображується одинарним кінцем лінії, множинність «багато» – потрійним («пташина лапка»). Обов'язковий зв'язок зображується неперервною лінією, факультативний – пунктирною.


**Рис. 7.2.** Зображення бінарного зв'язку на ER-діаграмі

На рис. 7.2 зображено зв'язок типу «багато-до-одного», що є обов'язковим з лівого боку і факультативним з правого.

**Іменування й інтерпретація зв'язків**

Біля кожного закінчення зв'язку маленькими літерами записується ім'я (рис. 7.3).

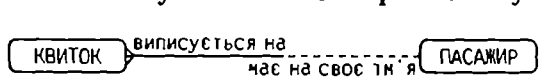



**Рис. 7.3.** Іменований зв'язок

Бінарному зв'язку відповідають два предикати, або твердження, які цей зв'язок інтерпретують. Один з цих предикатів дає можливість «прочитати» зв'язок зліва направо, інший – справа наліво.

Діаграма, що наведена на рис. 7.3, зліва направо і справа наліво читається так:

Будь-який КВИТОК завжди виписується на одного й лише одного ПАСАЖИРА

Будь-який ПАСАЖИР може мати на своє ім'я один або кілька КВИТКІВ

Слід зазначити, що обов'язкові зв'язки інтерпретуються за допомогою прислівника «завжди», факультативні – за допомогою дієслова «може». Множинність «багато» означає «один або кілька», а множинність «один» – «один і лише один».

Важливо також розуміти, що зв'язок означується саме для сутностей, а не для їхніх окремих екземплярів. Для довільних множин екземплярів зв'язаних сутностей предикати, які інтерпретують зв'язок, повинні бути істинними.

**Рекурсивні зв'язки**

Зв'язок сутності з самою собою називається *рекурсивним*. Таким зокрема є зв'язок типу «один-до-багатьох» для сутності ДЕТАЛЬ, якщо припускати, що деталі можуть складатися з інших деталей (рис. 7.4).

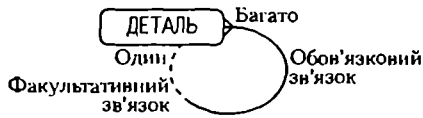


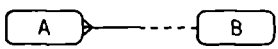
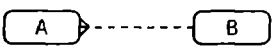
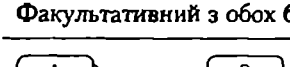
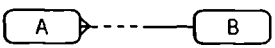

**Рис. 7.4.** Рекурсивний зв'язок

Наведений на рисунку рекурсивний зв'язок утворює нескінченну ієрархію.

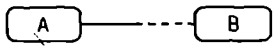
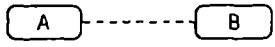
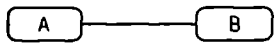
**Припустимі й неприпустимі зв'язки**

У табл. 7.1–7.6 наведено інформацію про найпоширеніші типи зв'язків, що відрізняються множинністю й факультативністю. Деякі з них застосовуються часто, інші – в особливих ситуаціях, а треті є неприпустимими.

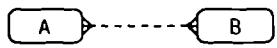
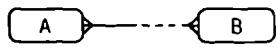
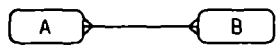
**Таблиця 7.1.** Зв'язки типу «багато-до-одного»

| Різновид зв'язку | Коментар |
|---|---|
|   Обов'язковий з боку множинності «багато», факультативний з боку множинності «один» | Конфігурація є найпоширенішою; екземпляр *A* може існувати виключно в контексті одного й лише одного екземпляра *B*, а екземпляри *B* можуть існувати як в асоціації з екземплярами *A*, так і незалежно від них |
|   Факультативний з обох боків | Конфігурація використовується не дуже часто. Екземпляри *A* та *B* можуть існувати, не будучи пов'язаними між собою |
|   Обов'язковий з обох боків | Конструкція є дуже жорсткою: жоден екземпляр *B* не може бути створений без створення хоча б одного пов'язаного з ним екземпляра *A*, і навпаки |
|   Факультативний з боку множинності «багато», обов'язковий з боку множинності «один» | Конфігурація трапляється рідко, але необхідна для відображення ситуації, коли *B* є штучним об'єктом, який завжди містить певну множину екземплярів *A*. Останні в цьому випадку можуть існувати абсолютно незалежно. Під час більш ретельного розгляду часто виявляється, що такі зв'язки мають множинність «багато-до-багатьох» |

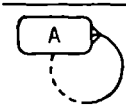
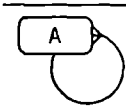
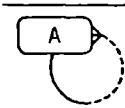
**Таблиця 7.2.** Зв'язки типу «один-до-одного»

| Різновид зв'язку | Коментар |
|---|---|
|  З одного боку обов'язковий, з іншого — факультативний | Конфігурація найчастіше відображує той факт, що сутність *A* є різновидом сутності *B* |
|  Факультативний з обох боків | Конфігурація трапляється нечасто |
|  Обов'язковий з обох боків | Конфігурація трапляється вкрай рідко. (Майже завжди це помилка!). Під час детальнішого розгляду зв'язку такого типу виявляється, що *A* і *B* - це фактично різні відображення або підмножини одного й того ж поняття чи об'єкта, які отримали різні імена і, мабуть, різні атрибути та зв'язки |

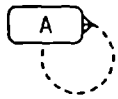
**Таблиця 7.3.** Зв'язки типу «багато-до-багатьох»

| Різновид зв'язку | Коментар |
|---|---|
|  Факультативний з обох боків | Ця конструкція є дуже поширеною на ранніх етапах аналізу предметної області |
|  З одного боку обов'язковий, з іншого — факультативний | Конфігурація трапляється рідко. Подібні зв'язки підлягають подальшому уточненню |
|  Обов'язковий з обох боків | Неприпустимий варіант. Такий зв'язок означає, що жоден екземпляр *A* не може існувати без екземплярів *B*, і навпаки. Важко уявити ситуацію, коли така конструкція відповідала б дійсності |

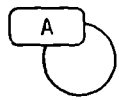
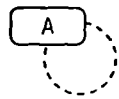
**Таблиця 7.4.** Рекурсивні зв'язки типу «багато-до-одного»

| Різновид зв'язку | Коментар |
|---|---|
|  Обов'язковий з боку множинності «багато», факультативний з боку множинності «один» | Неприпустима конфігурація, що описує ієрархію без кореневої вершини |
|  Обов'язковий з обох боків | Неприпустима конфігурація, що описує ієрархію без кореневої і кінцевих вершин |
|  Факультативний з боку множинності «багато», обов'язковий з боку множинності «один» | Неприпустима конфігурація, що описує ієрархію без кінцевих вершин |

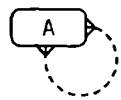
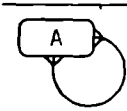
**Таблиця 7.4** (*продовження*)

| Різновид зв'язку | Коментар |
|---|---|
|  | Дуже популярна конфігурація (так званий «факультативний поросячий хвостик»). Відображує просту ієрархію з довільною кількістю рівнів; застосовується для зображення адміністративної підлеглості, опису складу виробів тощо |
| Факультативний з обох боків | |

**Таблиця 7.5.** Рекурсивні зв'язки типу «один-до-одного»

| Різновид зв'язку | Коментар |
|---|---|
|  | Неприпустима конфігурація |
| З одного боку обов'язковий, з іншого — факультативний | |
|  | Неприпустима конфігурація |
| Обов'язковий з обох боків | |
|  | Конфігурація трапляється рідко. Може використовуватися для зображення зв'язків, які вказують на альтернативні варіанти |
| Факультативний з обох боків | |

**Таблиця 7.6.** Рекурсивні зв'язки типу «багато-до-багатьох»

| Різновид зв'язку | Коментар |
|---|---|
|  | Конфігурація часто трапляється на ранніх етапах аналізу предметної області. Зазвичай зображує структуру такого типу: «будь-який елемент може складатися з одного або кількох елементів і будь-який елемент може входити до складу одного або кількох елементів» |
| Факультативний з обох боків | |
|  | Неприпустима конфігурація |
| З одного боку обов'язковий, з іншого — факультативний | |
|  | Неприпустима конфігурація |
| Обов'язковий з обох боків | |

### Атрибут

*Атрибут* – це деталь або аспект якісного чи кількісного опису сутностей, іхньої ідентифікації, класифікації або відображення їхнього стану. Атрибутом може бути текст, число, картинка, відчуття, запах тощо. У найпростішому випадку можна обмежитися текстовими й числовими атрибутами.

#### Зображення атрибутів

Ім'я атрибута записується всередині прямокутника сутності маленькими літерами в однині, можливо, з прикладами значень. Атрибути необов'язково показувати на діаграмі сутностей і зв'язків, проте додавання до сутностей кількох атрибутів у період формування моделі, як правило, є дуже корисним методом. Особливо це відчутно під час розмежування загальних сутностей та їхніх різновидів (рис. 7.5).

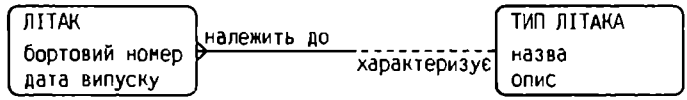



**Рис. 7.5.** Зображення атрибутів

#### Визначення атрибутів та їх належності сутностям

Атрибут має описувати сутність, якій він належить. Це може видаватися очевидним, проте з цим пов'язана найбільша кількість помилок під час визначення атрибутів. Наприклад, атрибутом якої сутності є «номер місця»: квитка, посадкового талона, повітряного човна або місця в літаку? Очевидно, це атрибут сутності МІСЦЕ, але на практиці ми часто зустрічаємося з його багаторазовими повтореннями, наприклад на посадковому талоні, що на рис. 7.6 зображений як самостійна сутність. Повторення пояснюються тим, що атрибут може використовуватися як зовнішній ключ для моделювання зв'язків.

Не слід використовувати імені сутності в найменуванні її атрибутів. Це зайве, оскільки атрибут описує тільки одну сутність. У наведеному прикладі існування дефініції «номер місця» вказує на наявність сутності МІСЦЕ, яку потім можна описувати за допомогою атрибута номер та інших атрибутів (скажімо, атрибута тип).

Імена атрибутів можуть читатися так: <Ім'я атрибута> <Ім'я сутності у родовому відмінку>. Наприклад, номер МІСЦЯ. У класичному прикладі з відділами і службовцями «номер відділу» не є атрибутом сутності СЛУЖБОВЕЦЬ, він є атрибутом відділу і тому має бути визначений як номер ВІДДІЛУ.

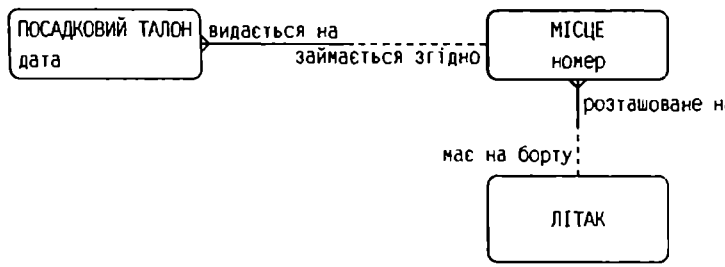



**Рис. 7.6.** Встановлення належності атрибута певній сутності

Кожний екземпляр сутності може містити лише одне значення кожного атрибута — це вимога першої нормальної форми. Якщо це не так, слід визначити нову сутність, додати до неї «спірний» атрибут, і з'єднати її з вихідною сутністю зв'язком типу «багато-до-одного». Звернімося до прикладу з місцями на повітряному човні, де на початковому етапі моделювання можна було б побудувати діаграму, наведену на рис. 7.7, *а*. Користуючись вказаним вище правилом, отримаємо іншу діаграму (рис. 7.7, *б*).

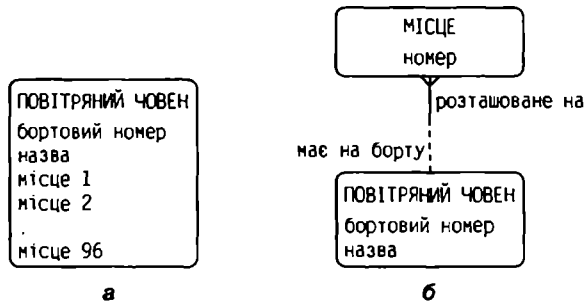



**Рис. 7.7.** Зведення до першої нормальної форми: атрибут, що повторюється, вказує на сутність, якої бракує (*а*); додавання сутності, якої не вистачало (*б*)

Якщо атрибут має самостійний зміст, власні зв'язки та атрибути, він може стати сутністю. Так, у попередньому прикладі ми використовували бортовий номер як атрибут сутності ПОВІТРЯНИЙ ЧОВЕН. Це завжди буде виправдано, якщо тільки ми не розглядатимемо підсистему реєстрації повітряних суден авіакомпаніями. У цьому випадку можуть знадобитися атрибути «дата реєстрації», «місце реєстрації», «зареєстрований власник» тощо. Тоді діаграма з попереднього рисунку трансформується (рис. 7.8).

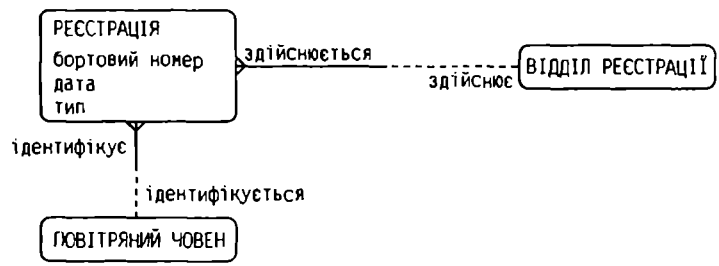



**Рис. 7.8.** Атрибут, перетворений на сутність

### Факультативні й обов'язкові атрибути

Атрибут, значення якого може бути відсутнє, називається *факультативним*. Перед його ім'ям записується символ «°». Атрибут, значення якого має бути завжди відоме називається *обов'язковим* і позначається символом «*». Екземпляр сутності може бути визначений тоді й лише тоді, коли відомі значення всіх його обов'язкових атрибутів.

**Унікальний ідентифікатор**

Кожна сутність має однозначно ідентифікуватися за допомогою комбінації атрибутів та/або зв'язків. Тому серед можливих атрибутів та/або зв'язків сутності завжди мають бути такі, які дають змогу її ідентифікувати. Унікальний ідентифікатор позначається на ER-діаграмі символом «#» перед ім'ям кожного атрибута, що входить до його складу, і поперечними рисками на лініях відповідних зв'язків. Усі атрибути унікального ідентифікатора мають бути обов'язковими.

Значення інших атрибутів сутності мають залежати від усього унікального ідентифікатора, а не від його частин (вимога другої нормальної форми). Видаліть атрибути, значення яких залежать лише від певної частини ідентифікатора. Наявність таких атрибутів зазвичай вказує на те, що слід створити додаткову сутність.

Атрибути мають залежати лише від унікального ідентифікатора (вимога третьої нормальної форми). Якщо існують атрибути, що залежать не тільки від унікального ідентифікатора, необхідно створити нову сутність, якій ці атрибути належатимуть.

Слід видалити також ті атрибути, які не залежать від ідентифікатора сутності (йдеться лише про функціональну залежність). Наприклад, є посадковий талон із зазначеним на ньому прізвищем пасажира. Чи залежить прізвище пасажира від унікального ідентифікатора посадкового талона? Очевидно, що ні. (Адже пасажир не змінює свого прізвища, коли йому видають новий посадковий талон). Якщо той чи інший атрибут не залежить від ідентифікатора, це, ймовірно, означає наявність пропущеної сутності та/або зв'язку, якому належить цей атрибут. Розглянемо рис. 7.9, де зображені зв'язки та унікальні ідентифікатори кількох сутностей.

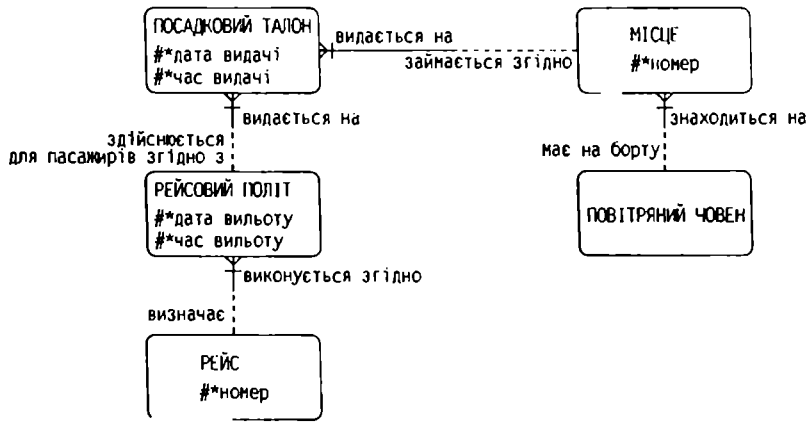



**Рис. 7.9.** Відображення унікальних ідентифікаторів

Згідно з рисунком для однозначної ідентифікації посадкового талона необхідні:

- дата і час видачі;
- зв'язок з місцем, відтак і номер місця;
- зв'язок з повітряним човном;
- зв'язок з рейсовим польотом, відтак відповідні дата і час вильоту;
- номер рейсу, оскільки до складу унікального ідентифікатора рейсового польоту входить зв'язок з рейсом.

### Типи та екземпляри

Дуже важливо чітко розуміти, що всі визначення сутності, зв'язку, атрибута й унікального ідентифікатора, які ми щойно розглянули, є визначеннями типу, або класу, а не екземпляра. Екземпляри сутностей і зв'язків відображуються не на ER-моделі, а в самій базі даних. Решта понять і визначень у моделюванні сутностей і зв'язків також стосуються типів, а не об'єктів.

## 7.3.2. Рекомендації та правила побудови діаграм

Готуючись до обговорення проектних рішень із замовниками системи, потрібно побудувати діаграму відповідного фрагмента предметної області. Переглядаючи цей фрагмент разом із вашим співрозмовником, ви часто знаходитимете помилки. Це абсолютно природно, адже побудова діаграми складної предметної області є ітеративним процесом, на першому етапі якого майже ніколи не отримують кінцевий результат.

Діаграму слід спланувати в такий спосіб, щоб прямокутники сутностей розміщувалися на одних рівнях, а лінії зв'язків були прямими, горизонтальними або вертикальними. Кількість перетинів потрібно звести до мінімуму, також слід уникати побудови діаграм з великою кількістю близько розташованих паралельних ліній. Необхідно використовувати більше вільного місця, щоб не створювалося враження громіздкості.

### Розпізнавання зображень

Більшість людей здатні миттєво розпізнавати зорові образи. Розробник моделі може використовувати цю особливість, створюючи діаграми, що відрізняються за формою і конфігурацією. Це дасть змогу легко повертатися до раніше узгодженої моделі та обговорювати її з максимальною продуктивністю. Якщо намалювати кілька різних діаграм однакової конфігурації, то здатність до швидкого відновлення деталей в пам'яті буде втрачено.

### Текст

Тексти мають виключати двозначне тлумачення. Слід уникати скорочень і жаргону. Щоб текст легше читався, більшість написів має розташовуватися горизонтально. Для зручності написи можна розміщувати у кількох рядках. Два імені зв'язку слід розташовувати на протилежних кінцях лінії зв'язку по різні її боки.

Центрування і вирівнювання тексту вліво або вправо потрібно здійснювати одноманітно, це значно підвищить якість документів і зручність їх сприйняття.

### Множинність зв'язку

Закінчення зв'язку типу «багато» («пташину лапку») бажано розміщувати ліворуч від лінії зв'язку або над нею. Це поліпшує якість сприйняття моделі та її адекватність.

### Розміри та форма прямокутників

Розміри та форма прямокутників, що зображують сутності, не мають особливого значення, їх можна змінюватн — подовжувати, збільшувати або зменшувати.

## 7.3.3. Складніші поняття ER-моделювання

У мові ER-моделювання використовуються й більш складні поняття. А саме, це поняття, пов'язані із:

- сутностями – підтипи, супертипи, базисні й перехідні сутності;
- зв'язками – моделювання зв'язків типу «багато-до-багатьох», взаємовиключність, непереміщуваність, кваліфікована множинність, надлишковість, каскадне видалення й оновлення;
- унікальними ідентифікаторами визначення за допомогою дуг, що виключають одна одну;
- доменами – означення, характеристики і застосування;
- атрибутами – похідні атрибути

### Сутності

#### Підтипи

*Підтип* – це різновид певної сутності. Сутність може поділятися на два або більше підтипи, що мають спільні атрибути та/або зв'язки, які явно визначаються один раз на вищому рівні. Підтипи можуть мати власні атрибути та/або зв'язки і, в свою чергу, поділятися на підтипи нижчих рівнів. Підтип сутності неявно успадковує всі атрибути, зв'язки і властивості відповідної сутності вищого рівня супертипу.

#### Супертипи

*Супертип* – це сутність, що має підтипи. Одна й та сама сутність може бути супертипом однієї і підтипом іншої сутності. Підтипи сутності мають утворювати повну систему множин, тобто будь-який екземпляр супертипу має належати принаймні одному з підтипів. У багатьох випадках це правило буде приводити до означення додаткового підтипу з ім'ям на кшталт ІНША СУТНІСТЬ. Наприклад, в діаграмі, зображеній на рис. 7.10, введено сутність ІНШИЙ ПОВІТРЯНИЙ ЧОВЕН.

На ER-діаграмі прямокутники підтипів розташовуються всередині прямокутників їхніх супертипів (рис. 7.10).

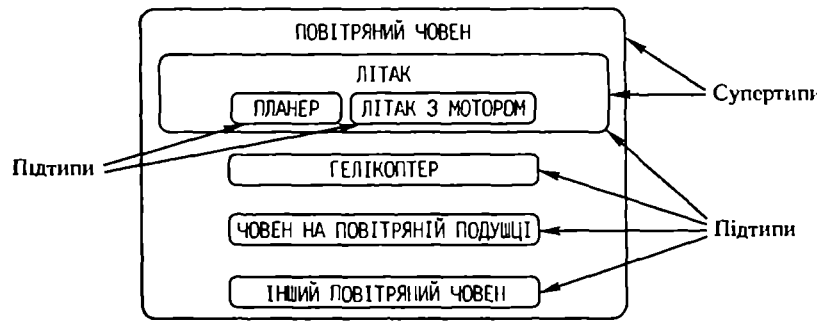



**Рис. 7.10.** Зображення підтипів і супертипів

Як і всі інші сутності, підтипи й супертипи можуть брати участь у зв'язках. Приклад встановлення зв'язків між підтипами та іншими сутностями наведений на рис. 7.11.

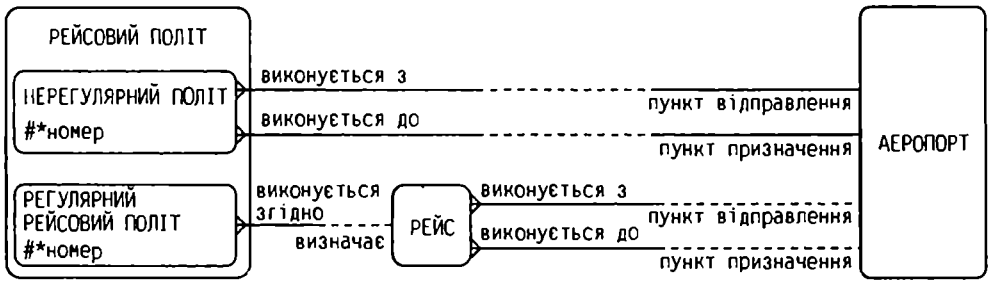



**Рис. 7.11.** Встановлення зв'язків між підтипами та іншими сутностями

### Базисні сутності

*Базисна сутність* – це сутність, з якою не з'єднано жодного обов'язкового закінчення зв'язку. Така сутність використовується переважно з метою доозначення інших сутностей – саме тому базисна сутність, як правило, розташована в закінченні «один» кількох зв'язків типу «багато-до-одного». Базисні сутності можна уявити як сутності, що можуть існувати самі по собі, без зв'язків або посилань на інші сутності.

### Перехідні сутності

*Перехідна сутність* вводиться виключно для моделювання зв'язку типу «багато-до-багатьох» між двома іншими сутностями. Екземпляри перехідної сутності можуть існувати лише в контексті зв'язків з відповідними базисними сутностями (рис. 7.12).

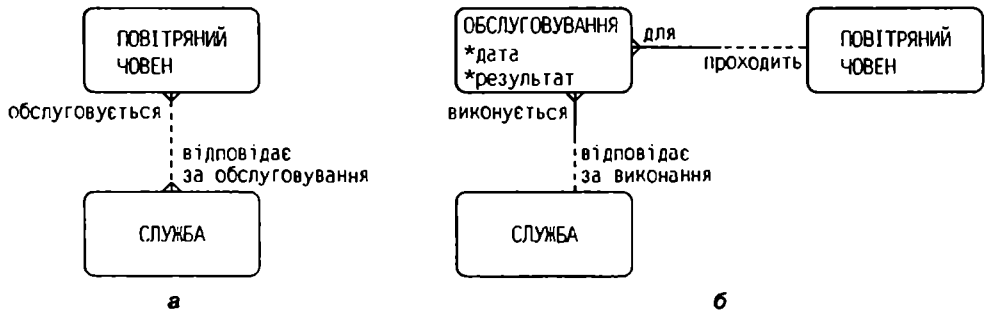



**Рис. 7.12.** Моделювання зв'язку типу «багато-до-багатьох»: початковий вигляд (*а*); кінцевий вигляд (*б*)

## Зв'язки

### Моделювання зв'язків типу «багато-до-багатьох»

Зв'язки типу «багато-до-багатьох» часто створюються на етапах визначення стратегії та аналізу предметної області. Під час завершення етапу аналізу вони мають бути змодельовані за допомогою двох зв'язків типу «багато-до-одного» і нової перехідної сутності, яка розділяє закінчення вихідного зв'язку (див. рис. 7.12).

**Взаємовиключність**

Два або більше зв'язків однієї і тієї ж сутності можуть виявитися взаємовиключними. Цей факт відображується поперечною дугою, що перетинає закінчення всіх взаємовиключних зв'язків, з невеликими кружечками в місцях перетину (рис. 7.13).

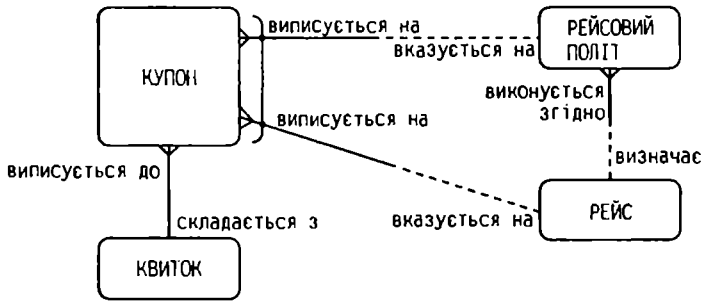



**Рис. 7.13.** Зображення взаємовиключних зв'язків

Якщо між взаємовиключними зв'язками на діаграмі зустрічається зв'язок, який не стосується цього обмеження, відповідна дуга може перетинати закінчення такого зв'язку, але без кружечка в місці перетину.

Взаємовиключні зв'язки мають такі властивості:

- всі закінчення зв'язків, які перетинаються поперечною дугою, мають бути або обов'язковими, або факультативними;
- закінчення зв'язку може перетинати лише одна поперечна дуга;
- майже завжди дуги перетинають закінчення «багато»;
- поперечні дуги не можуть перетинати зв'язки, що йдуть від різних сутностей, зокрема зв'язки, що проведені від підтипу та його супертипу;
- якщо хоча б один із зв'язків, що перетинаються поперечною дугою, входить до складу унікального ідентифікатора, то й решта зв'язків, що перетинаються цією дугою, мають бути частинами унікальних ідентифікаторів.

**Непереміщувані зв'язки**

Як правило, зв'язок екземпляра однієї сутності з екземпляром іншої можна розірвати. Після цього для вихідного екземпляра можна встановити інший зв'язок

Якщо це неприпустимо, то зазначають, що зв'язок (закінчення зв'язку) є таким, що не може бути переміщеним. Подібне закінчення зв'язку позначається ромбом (рис. 7.14). Недоцільно переміщувати посадковий талон з одного квитка в інший, проте можливо потрібно буде переписати його на інший рейс, якщо, наприклад, кількість пасажирів перевищить кількість посадкових місць.

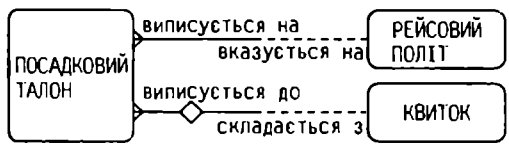



**Рис. 7.14.** Приклад зв'язку, що не може бути переміщений

### Кваліфікована множинність

Іноді буває дуже важливо мати нагоду вказати межі для множинності зв'язку, її стандартне, максимальне, середнє або мінімальне значення. Надалі така інформація може виявитися вкрай необхідною для проектувальників. У таких позначеннях використовуються символи операцій порівняння: =, >, >=, <, <= (рис. 7.15).

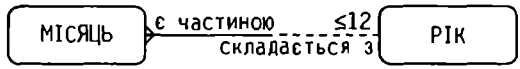



**Рис. 7.15.** Кваліфікована множинність

### Надлишкові зв'язки

Зв'язки, які можуть бути виведені з інших зв'язків, називаються *надлишковими*. ER-діаграма не повинна містити надлишкових зв'язків. Хоча у базах даних надлишковість є поширеним засобом досягнення необхідної швидкодії системи, подібні рішення мають ухвалюватися проектувальником і не повинні прийматися системним аналітиком На діаграмі, що зображена на рис. 7.16, два надлишкові зв'язки перекреслено.

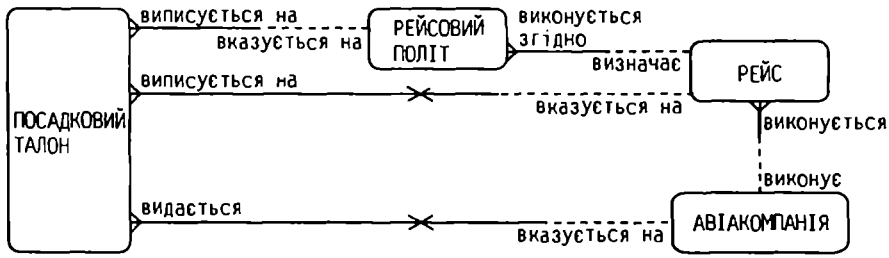



**Рис. 7.16.** Приклад надлишкових зв'язків

### Каскадне видалення

У реальному світі ми, коли втрачаємо про щось усі відомості, часто неявно втрачаємо й відомості про якісь інші взаємопов'язані речі. Наприклад, якщо ми видалимо всі відомості про квиток, ми неявно видалимо і всі відомості про його посадкові талони, оскільки вони, як «діти», існують виключно в контексті їхніх зв'язків з квитками («батьками»). Це явище відоме як *каскадне видалення* і має місце для деяких сутностей, розміщених у закінченнях «багато». У ситуаціях, коли каскадне видалення неприпустиме, часто діє таке правило: забороняється видаляти незалежні сутності («батьків»). Наприклад, не можна видалити екземпляр сутності ЕКІПАЖ доти, доки існують члени цього екіпажу.

Для відображення різних правил видалення сутності використовується покажчик каскадного видалення:

- X – видаляти всіх «дітей» під час видалення «батька»;
- C – заборонити видалення «батька», якщо існують «діти»;
- N видалення «батьків» і «дітей» може здійснюватися незалежно.

Правило заборони (C) зазвичай має місце в тих випадках. коли залежна сутність («дитина») обов'язково має бути пов'язана з «батьком», а незалежне видалення (N) – тоді, коли обидва закінчення зв'язку є факультативними.

#### Каскадне оновлення

Під час зміни унікального ідентифікатора (первинного ключа) «батька» його нове значення автоматично копіюється до всіх зовнішніх ключів, які він утворює. Це і є *каскадне оновлення*.

### Унікальні ідентифікатори

Як уже зазначалось, унікальні ідентифікатори можуть бути довільними комбінаціями атрибутів та/або закінчень зв'язків. Припустимо, що один із кількох взаємовиключних зв'язків є частиною унікального ідентифікатора. Тоді унікальний ідентифікатор для множини екземплярів, які не беруть участі в цьому зв'язку, може виявитися недовизначеним. Щоб цього не сталося, має виконуватися така умова: якщо певний зв'язок є частиною унікального ідентифікатора, то й взаємовиключні з ним зв'язки теж мають бути частинами унікальних ідентифікаторів.

### Домени

*Домен* – це набір прикладних обмежень, форматів, правил контролю за допустимістю значень та інших властивостей, спільних для групи атрибутів. Наприклад, доменом може бути:

- перелік значень;
- діапазон значень;
- перелік або діапазон значень з обмеженнями;
- будь-яка комбінація перелічених вище об'єктів.

Визначення доменів доцільно вводити для атрибутів, що часто використовуються, на зразок адреси, поштового коду, оцінки успішності, окладу тощо.

Домен зображується у вигляді окремого опису правил і обмежень, а також переліку атрибутів, які належать цьому домену.

### Атрибути

*Похідні атрибути* – це атрибути, яким ані користувачі, ані зовнішні застосування не присвоюють значень. Їхні значення виводяться (обчислюються), коли в цьому виникає потреба. Наприклад, до сутності КВИТОК можна було б додати похідний атрибут «фактично сплачена вартість» зі значеннями, що обчислюються як різниця повної вартості та наданої знижки.

З поняттям похідного атрибута пов'язані певні труднощі, які необхідно вирішити, перш ніж використовувати його під час проектування бази даних.

Зі зміною значення одного з атрибутів, за допомогою яких визначається похідний атрибут, його значення також буде змінюватися. Залежно від того, як спроектована база даних, нове значення може обчислюватися або кожного разу, коли виникає така потреба, або ж кожного разу, коли відбувається його фактична зміна.

Усвідомлення цієї проектної дилеми й досягнення чіткого взаєморозуміння між аналітиками та проектувальниками дасть змогу широко застосовувати похідні атрибути під час моделювання предметних областей.

## 7.3.4. Супутні поняття

Інформаційне моделювання пов'язане з такими поняттями, як сутність, атрибут, зв'язок і домен. Існує ще декілька термінів, які слід розуміти проектувальникам прикладних систем, а саме:

- потоки даних;
- сховища даних;
- прикладні задачі й процеси;
- події.

### Потоки і сховища даних

Багато системних аналітиків і проектувальників використовують схему потоків даних, яка дає змогу моделювати зв'язки, наявні між прикладними задачами, шляхом зображення даних, що надходять на вхід і вихід кожної задачі. Вміст будь-якого сховища даних має бути об'єднанням усіх асоційованих з ним потоків даних.

Означення всіх потоків і сховищ даних здійснюється в термінах імен атрибутів, що належать сутностям ER-моделі. В окремих випадках в означеннях можуть також використовуватися похідні атрибути (наприклад, прибуток), які будуть обчислюватися через інші атрибути.

### Прикладні задачі й процеси

Будь-яка прикладна задача є означенням дій, які певний об'єкт виконує або має виконувати, незалежним від способу, в який це відбувається. Прикладна задача записується у вигляді фрази, що починається з дієслова, формулюється в термінах сутностей або їхніх синонімів та описує зміни даних в термінах імен атрибутів.

### Події

Виконання більшості прикладних застосувань ініціюється настанням різноманітних подій. Наприклад, календарні події (тридцять днів з дня виписування рахунку-фактури), події-зміни, коли змінюються деякі деталі, аспекти або надходить нова інформація про стан справ у прикладній області, системні події, коли завершення дії одного або кількох застосувань є сигналом до початку виконання нового застосування. Кожна така подія має бути означена й може містити опис умов, за яких вона відбудеться і які мають формулюватися в термінах імен атрибутів.

## 7.3.5. Нормалізація даних

*Нормалізація даних* — це процес, за допомогою якого будується модель даних відповідно до певних вимог. Для моделей зв'язків і сутностей такою вимогою є мінімізація дублювання даних з метою забезпечення гнучкості й можливості відображення моделі в різних схемах проектування баз даних.

Для того аби перевірити, що модель сутностей і зв'язків, в якій усі сутності унікально ідентифіковані, перебуває в третій нормальній формі, необхідно перевірити її відповідність кільком умовам.

### Попередня умова

Всі сутності мають бути унікально ідентифіковані комбінацією атрибутів та/або зв'язків.

### Перша нормальна форма

Слід видалити атрибути або групи атрибутів, що повторюються. Якщо атрибут може набувати більше одного значення водночас або є більше одного атрибута з тим самим ім'ям, то необхідно означити нову сутність, яка описується цим атрибутом. Унікальний ідентифікатор нової сутності складається з атрибутів, що перейшли до нього, і зв'язку типу «багато-до-одного» з вихідною сутністю.

Наприклад, видаляючи групи атрибутів з відомостями про членів екіпажу з сутності РЕЙСОВИЙ ПОЛІТ (рис. 7.17) і створюючи при цьому нову сутність ЧЛЕН ЕКІПАЖУ з атрибутами імені й ролі, сполучену зв'язком типу «багато-до-одного» з вихідною сутністю, ми зводимо відношення РЕЙСОВИЙ ПОЛІТ до першої нормальної форми.

Отже, зведення до першої нормальної форми є процедурою виявлення та ідентифікації попередньо упущених сутностей і зв'язків.

### Друга нормальна форма

Слід видалити атрибути, залежні тільки від частини унікального ідентифікатора.

Якщо сутність має унікальний ідентифікатор, що складається більш ніж з одного атрибута та/або зв'язку і якщо певний атрибут залежить тільки від частини цього складеного ідентифікатора, тоді цей атрибут і та частина ідентифікатора, від якої він залежить, мають стати основою для формування нової сутності. Нова сутність ідентифікується успадкованою частиною унікального ідентифікатора вихідної сутності та має з нею зв'язок типу «один-до-багатьох».

Наприклад, назви авіакомпанії та аеропорту, тип повітряного судна і його місткість залежать від номера рейсового польоту, але не залежать від дати і часу польоту, тому ми створюємо нову сутність РЕЙС з фіксованим номером рейсу, за допомогою якої визначається розклад одного або кількох рейсових польотів у певний проміжок часу (рис. 7.17).

Отже, зведення до другої нормальної форми також є процедурою виявлення й ідентифікації попередньо упущених сутностей і зв'язків.

### Третя нормальна форма

Слід видалити атрибути, залежні від атрибутів, що не є частиною унікального ідентифікатора.

Якщо існує атрибут, який залежить від іншого атрибута, що не є частиною унікального ідентифікатора, то залежний атрибут разом із тим атрибутом, від якого він безпосередньо залежить, мають становити основу для формування нової сутності, яка буде сполучена зв'язком типу «один-до-багатьох» з вихідною сутністю Унікальним ідентифікатором нової сутності є той атрибут, від якого залежить інший атрибут. Наприклад, місткість повітряного судна залежить від його типу й тому ці два атрибути слід виділити в окрему сутність (рис. 7.17).

Третя нормальна форма є останньою, за допомогою якої виявляються упущені сутності та зв'язки.

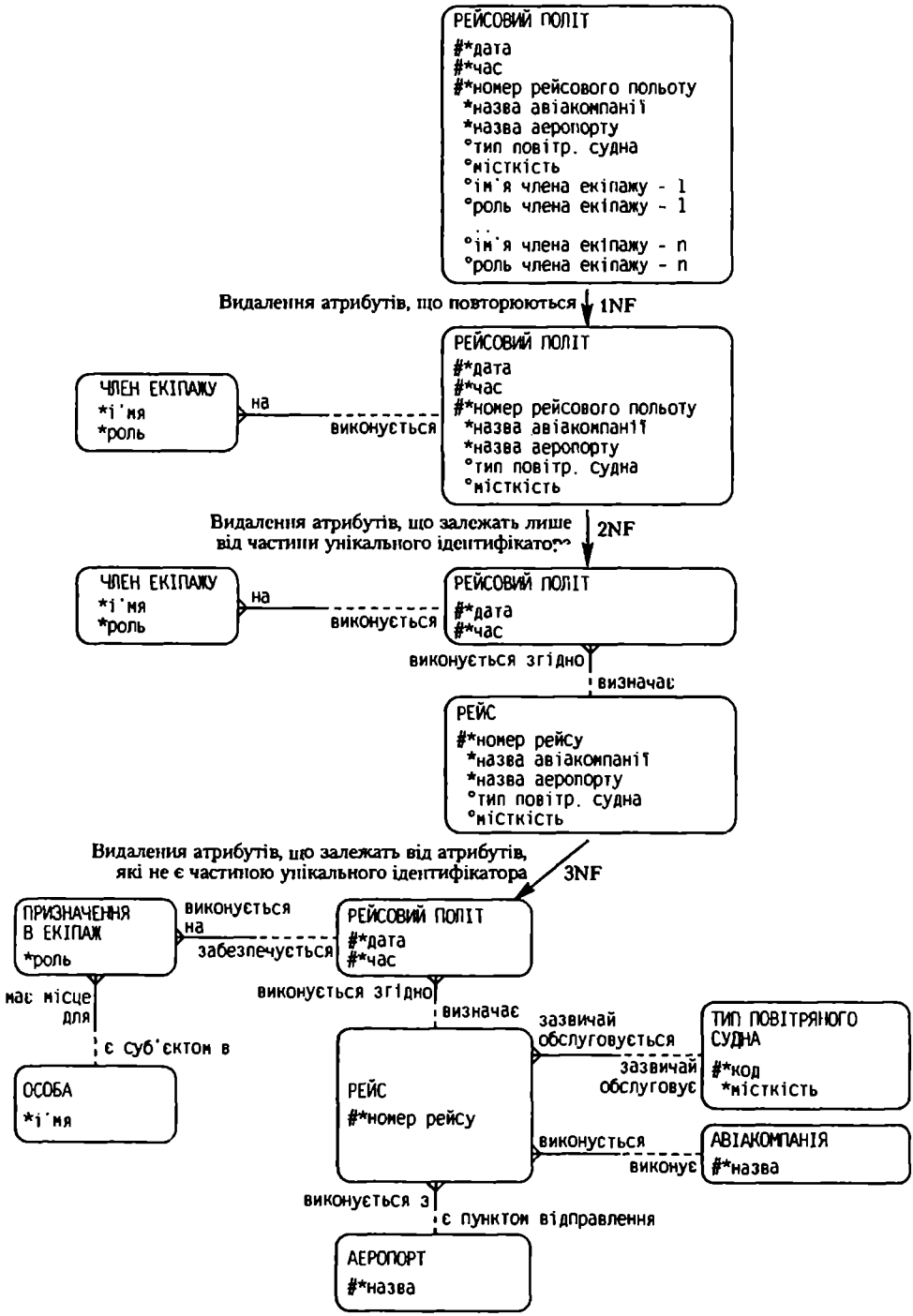



**Рис. 7.17.** Нормалізація даних в ER-моделі

#### Інтуїтивна нормалізація

Якщо ви ретельно проаналізуєте кінцеву модель, то виявите, що існують такі незалежні й важливі речі, щодо яких необхідно зберігати інформацію, тобто такі сутності, як АЕРОПОРТ, АВІАКОМПАНІЯ, ОСОБА тощо. Аналітик також усвідомлює, що назва авіакомпанії може бути атрибутом лише авіакомпанії. Базуючись на таких міркуваннях, можна виділити ще кілька сутностей (див. рис. 7.17).

Констатуючи наявність атрибутів у сутності, ретельно перевіряйте, чи не є атрибут сутністю, і якщо так, формуйте з нього та інших атрибутів нову сутність.

### 7.3.6. Проектування реляційної бази даних

На базі ER-моделі може бути спроектована схема реляційної бази даних. Алгоритм такого проектування з використанням мови SQL наведено нижче.

#### Крок 1. Перетворення сутностей на таблиці

Кожна *проста* сутність перетворюється на таблицю. *Простою* є сутність, яка не є підтипом або не має підтипів. Іменем може бути будь-яке з імен сутності

#### Крок 2. Перетворення атрибутів на стовпці

Кожний атрибут перетворюється на стовпець із тим самим ім'ям. Під час такого перетворення можна обрати формат стовпця.

Факультативні атрибути стають NULL-стовпцями, обов'язкові – NOT NULL-стовпцями.

#### Крок 3. Зображення унікальних ідентифікаторів ключами таблиць

Усі складові певного унікального ідентифікатора сутності стають первинним ключем таблиці. Нагадаємо, що сутність може мати більше одного унікального ідентифікатора, тому оберіть той, який використовується найчастіше.

Сутність може унікально ідентифікуватися комбінацією атрибутів та/або зв'язків. Якщо в ідентифікаторі сутності використовується зв'язок, додайте до таблиці стовпці, що відповідають унікальному ідентифікатору сутності, яка зв'язана з даною сутністю. Додані стовпці становитимуть частину первинного ключа. (Така процедура може виявитися рекурсивною, і побудова таблиці триватиме доти, доки унікальний ідентифікатор чергової сутності не буде складатися лише з атрибутів). Під час цього процесу імена зв'язків та/або імена сутностей використовуються разом з іменами атрибутів для отримання унікальних імен стовпців. Такі стовпці будуть зовнішніми ключами.

#### Крок 4. Перетворення зв'язків типів «багато-до-одного» та «один-до-одного» на зовнішні ключі

Зв'язки типів «багато-до-одного» та «один-до-одного» породжують зовнішні ключі. Іншими словами, унікальний ідентифікатор сутності, розташованої в закінченні зв'язку з множинністю «один», необхідно додати як стовпець до таблиці, розташованої з боку множинності «багато».

Факультативним з боку множинності «багато» зв'язкам відповідають NULL-стовпці; обов'язковим – NOT NULL-стовпці.

Нижче наведено приклад проектування фрагмента ER-діаграми з рис. 7.18.

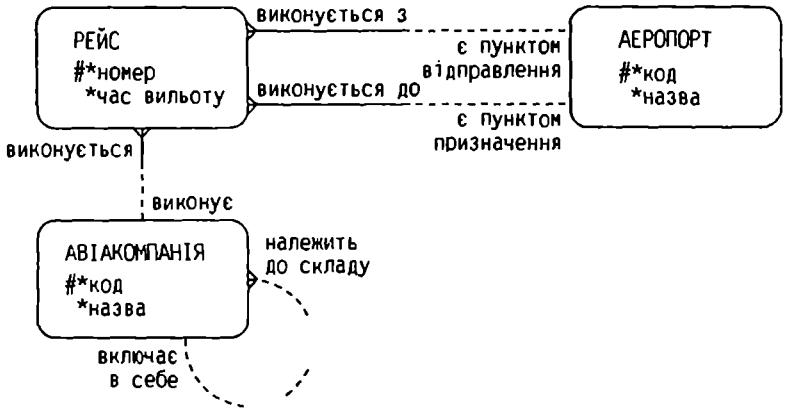



**Рис. 7.18.** Фрагмент ER-діаграми

| АЕРОПОРТИ | | | | |
|---|---|---|---|---|
| Код | char | 4 | not null | ← Первинний ключ |
| Назва | char | 40 | not null | |

| АВІАКОМПАНІЇ | | | | |
|---|---|---|---|---|
| Код | char | 4 | not null | ← Первинний ключ |
| Назва | char | 40 | not null | |
| Код батьківської компанії | char | 4 | null | ← Зовнішній ключ з рекурсивного зв'язку, що кваліфікується іменами сутності та зв'язку |

| РЕЙСИ | | | | |
|---|---|---|---|---|
| Номер | integer | 4 | not null | ← Первинний ключ |
| Код авіакомпанії | char | 4 | not nu | ← Зовнішній ключ з авіакомпанії |
| Код аеропорту з | char | 4 | not null | ← Зовнішні ключі з аеропорту |
| Код аеропорту в | char | 4 | not null | |
| Час вильоту | date | | null | |

Кроки 1–4 виконуються майже в кожному процесі проектування Інколи потрібно виконати й допоміжні кроки 5–7.

### Крок 5. Проектування за наявності підтипів

*Підтип сутності* — це також сутність з її власними атрибутами та зв'язками; вона успадковує всі атрибути та зв'язки батьківської сутності (супертипу), а також усіх сутностей, що розташовані на вищих рівнях ієрархії.

Є два альтернативні способи відображення підтипів у вигляді таблиць: поєднання всіх атрибутів в одній таблиці та створення окремої таблиці для кожного підтипу. Кожен із цих методів має свої переваги й недоліки.

**Все в одній таблиці**

Основна таблиця створюється для сутності-супертипу, що є вершиною ієрархії. Можуть також створюватися віртуальні таблиці для кожного з підтипів. Як і раніше, атрибути й зв'язки типу «багато-до-одного» приводять до створення стовпців даних і зовнішніх ключів.

*Віртуальна таблиця* — це спосіб звернення до підмножини даних із базової таблиці так, ніби це інша таблиця. Віртуальна таблиця може бути обмежена підмножиною стовпців та/або підмножиною рядків базової таблиці і в ній можуть бути змінені імена стовпців. Подібні прості віртуальні таблиці використовуються як для оновлення, так і для пошуку даних. Процедура створення віртуальних таблиць застосовується до кожного підтипу, всіх його підтипів тощо.

У базовій таблиці всі стовпці, створені для підтипів, оголошуються як NULL-стовпці. Обов'язковість атрибута (або зв'язку) для одного підтипу не має застосовуватися до іншого — тому всі такі стовпці мають стати факультативними з можливою підтримкою згаданого обмеження цілісності в прикладному програмному забезпеченні або за допомогою спеціальної віртуальної таблиці.

До базової таблиці потрібно додати принаймні ще один стовпець з опцією not-null для визначення типу; він стає частиною первинного ключа.

Віртуальна таблиця створюється для кожного з підтипів з тією метою, щоб надати можливість доступу лише до необхідних даних згідно з вимогами прикладних задач, що визначають спосіб маніпулювання сутностями, які є підтипами. Подібні віртуальні таблиці мають містити стовпці підтипу, ключі всіх його підтипів та супертипу. До оголошення віртуальної таблиці додається фраза WHERE для перевірки значення поля Тип та контролю коректності зовнішніх ключів.

Розглянемо приклад на рис. 7.19. До складу первинного ключа введено поле Тип, значення якого вказує на те, що рядок таблиці належить супертипу ("ЗАМ") чи одному з його підтипів (рядок замовлення квитка, "РЗК", або інший рядок замовлення, "ІРЗ").

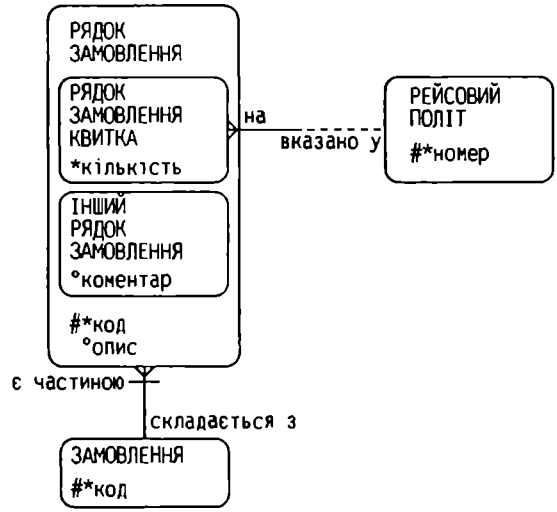
Наведемо приклади описів віртуальних таблиць на мові SQL:

```
CREATE VIEW ІНШІ_РЯДКИ_ЗАМОВЛЕННЯ AS
  SELECT Код_рядка, Код_замовлення, Опис, Коментар, Тип
  FROM РЯДКИ_ЗАМОВЛЕННЯ
  WHERE Тип="ІРЗ"

CREATE VIEW РЯДКИ_ЗАМОВЛЕННЯ_КВИТКА AS
  SELECT Код_рядка, Код_замовлення, Опис, Кількість, Номер_рейсу, Тип
  FROM РЯДКИ_ЗАМОВЛЕННЯ
  WHERE Тип="РЗК" AND Кількість NOT NULL AND
    EXISTS (SELECT *
            FROM РЕЙСОВІ_ПОЛЬОТИ
            WHERE РЕСОВІ_ПОЛЬОТИ Номер = РЯДКИ_ЗАМОВЛЕННЯ.Номер_рейсу)
```

За допомогою обох віртуальних таблиць перевіряються значення у стовпці Тип. Друга з цих таблиць вимагає обов'язкової наявності значення у стовпці Кількість

та гарантує існування номера рейсу в таблиці РЕЙСОВІ_ПОЛЬОТИ, що збігається зі значенням Номер_рейсу.




*а*

| РЯДКИ_ЗАМОВЛЕННЯ | | | |
|---|---|---|---|
| Код_рядка | integer | 4 | not null |
| Код_замовлення | integer | 9 | not null |
| Тип | char | 3 | not null |
| Опис | char | 40 | null |
| Коментар | char | 40 | null |
| Кількість | integer | 3 | null |
| Номер_рейсу | integer | 4 | null |

Первинний ключ

*б*

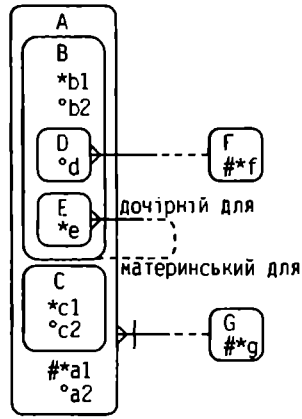
**Рис. 7.19.** Відображення підтипів у одній таблиці: ER-діаграма (*а*); таблиця (*б*)

**Окрема таблиця для кожного підтипу**

Для кожного з підтипів створюється таблиця, в якій зберігатимуться його екземпляри. Якщо є багато рівнів підтипів, то можна побудувати таблиці тільки для найнижчого рівня. У цьому випадку таблиці всіх інших рівнів отримують як віртуальні таблиці.

Таблиця, що створюється для підтипу, містить стовпці для кожного атрибута і зв'язку типу «багато-до-одного». Окрім того, створюються стовпці для всіх атрибутів і зв'язків типу «багато-до-одного» супертипу даного типу (і так далі вгору за ієрархією підтипів). При цьому властивості факультативності й належності до первинного ключа успадковуються.

На рис. 7.20 наведено приклад відображення підтипу найнижчого рівня.




*а*

| E | |
|---|---|
| e | not null |
| b1 | not null |
| b2 | null |
| a1 | not null |
| G_g | not null |
| a2 | null |
| материнський_a1 | null |
| материнський_G_g | null |

З підтипу E, обов'язковий
З підтипу B, обов'язковий
З підтипу B, факультативний
Первинний ключ з супертипу A
З супертипу A, факультативний
Зі зв'язку між E та B, повторення первинного ключа B з факультативною опцією

*б*

**Рис. 7.20.** Відображення підтипу E: ER-діаграма (*а*); таблиця (*б*)

В аналогічний спосіб можна побудувати таблиці для підтипів D та C. Тоді віртуальні таблиці для сутностей типів B та A означуватимуться так:

```
CREATE VIEW B AS
  SELECT b1, b2, a1, a2, G_g
  FROM D
  UNION
  SELECT b1, b2, a1, a2, G_g
  FROM E

CREATE VIEW A AS
  SELECT a1, a2, G_g
  FROM D
  UNION
  SELECT a1, a2, G_g
  FROM E
  UNION
  SELECT a1, a2, G_g
  FROM C
```

### Крок 6. Зв'язки, що виключають одне одного

Є два основні методи проектування баз даних з урахуванням використання зв'язків, що виключають одне одного: спільний домен та явні зовнішні ключі.

#### Спільний домен

Якщо всі зовнішні ключі належать одному домену, то для проектування набору взаємовиключних зв'язків створюються два стовпці: ідентифікатор зв'язку та ідентифікатор екземпляра сутності. Стовпець ідентифікатора зв'язку потрібен для розрізнення зв'язків. Стовпець ідентифікатора екземпляра сутності є зовнішнім ключем.

Для таблиці ПОСАДКОВИЙ_ТАЛОН стовпці, що реалізують зв'язки з сутностями РЕЙСОВИЙ ПОЛІТ або РЕЙС (рис. 7.21), можуть бути такими:

```
Пов'язаний_з  char 4 not null
Номер_рейсу_польоту  integer 4 not null
```

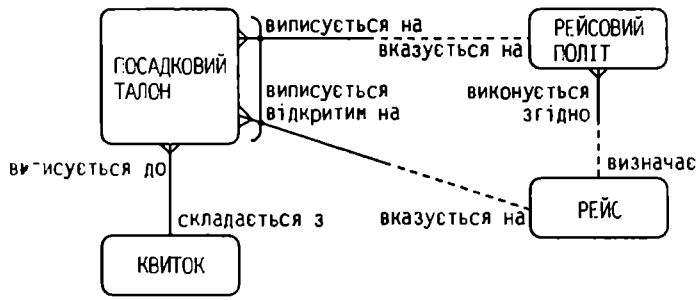



**Рис. 7.21.** ER-діаграма із взаємовиключними зв'язками

Стовпець Пов'язаний_з ідентифікує зв'язок, а стовпець Номер_рейсу_польоту — екземпляр сутності. Оскільки обидва зв'язки є обов'язковими, то й стовпці мають NOT NULL-опції. Значеннями стовпця Пов'язаний_з можуть бути, наприклад, рядки "РПОЛ" і "РЕЙС".

#### Явні зовнішні ключі

Якщо зовнішні ключі належать різним доменам, то потрібно створити стовпці зовнішніх ключів для всіх взаємовиключних зв'язків, і зробити всі ці стовпці факультативними. При цьому прикладні застосування мають підтримувати таке обмеження: може бути встановлене значення не більше ніж одного зовнішнього ключа для всіх взаємовиключних зв'язків, а якщо ці зв'язки є обов'язковими, то потрібно встановити точно одне таке значення.

У попередньому прикладі ми припустили, що ключі сутностей РЕЙСОВИЙ ПОЛІТ та РЕЙС належать до одного домену. Насправді вони з різних доменів, ключ РЕЙСУ — це його номер, а ключ сутності РЕЙСОВИЙ ПОЛІТ — номер рейсу, дата та час. У цьому випадку стовпцями зовнішніх ключів для зв'язування з рейсом і рейсовим польотом можуть бути:

```
Номер_рейсу  integer 4 null
Дата_польоту  date null
Час_відльоту  time null
```

### Крок 7. Похідні атрибути

Під час проектування слід ухвалити рішення про відображення похідних атрибутів. Врахуйте, що атрибут слід робити похідним лише тоді, коли в цьому є необхідність, оскільки це може призвести до порушення важливого правила — атрибути завжди стають стовпцями.

Якщо застосування багаторазово здійснюють доступ до значення атрибута, а зміни відбуваються рідко, то не слід обчислювати значення атрибута щоразу під час його використання. Краще створити для похідного атрибута стовпець й оновлювати його значення щоразу під час змінення відповідних вихідних даних. Цей підхід є виправданим за таких умов.

- Похідне значення змінюється рідко.
- Вартість обчислень дуже висока. Зазвичай це відбувається тоді, коли обчислення виконуються більш ніж на одному рядку бази даних

## Контрольні запитання та завдання

1. Назвіть основні фази та етапи життєвого циклу системи баз даних.
2. Які вимоги висуваються до методології проектування баз даних?
3. Якими є основні результати етапу визначення стратегії?
4. Назвіть основні результати етапу аналізу предметної області.
5. Які основні результати дає етап концептуального моделювання?
6. Назвіть основні результати етапів логічного та фізичного проектування бази даних.
7. Наведіть приклади неприпустимих зв'язків
8. Наведіть приклади, коли в процесі проектування атрибути і зв'язки перетворюються на сутності.
9. Наведіть приклади. коли зв'язок входить до складу унікального ідентифікатора сутності.
10. Що таке підтип і супертип? Наведіть приклади.
11. Що таке базисні й перехідні сутності?
12. У процесі проектування були виявлені зв'язки типу «багато-до-багатьох». Про що це свідчить?
13. Що таке взаємовиключні зв'язки? Наведіть приклади.
14. Що називається каскадним видаленням та оновленням?
15. Що таке нормалізація даних? Назвіть етапи нормалізації.
16. Опишіть кроки перетворення ER-моделі на реляційну базу даних.

# Розділ 8

# Цілісність даних

- Поняття про обмеження цілісності та їхня класифікація
- Декларативні обмеження цілісності
- Динамічні обмеження цілісності
- Семантичні обмеження цілісності
- Підтримка цілісності у разі виникнення перебоїв

## 8.1. Поняття про обмеження цілісності

Терміном *цілісність даних* позначають достовірність і точність інформації, що зберігається в базі. Цілісність досягається забезпеченням відповідності даних певним додатковим обмеженням, крім тих, які накладаються схемою бази на структуру даних та їхні типи.

*Обмеження цілісності* – це правила, які обмежують усі можливі стани бази даних, а також переходи з одного стану в інший. Таким чином, обмеження цілісності визначають множину «допустимих» станів і переходів між ними. База даних перебуває в *цілісному стані*, якщо вона відповідає всім визначеним для неї вимогам цілісності.

### Класифікація обмежень цілісності

Обмеження цілісності класифікують за:

- походженням;
- способом підтримки;
- терміном перевірки;
- областю дії;
- можливістю обмежувати переходи бази даних з одного стану в інший.

За походженням обмеження цілісності поділяють на:

- структурні;
- семантичні.

*Структурні обмеження цілісності* — це обмеження, які випливають із властивостей структури даних, що зберігаються в базі.

*Семантичні обмеження цілісності* — це обмеження, що накладаються предметною областю, яка моделюється.

За способом підтримки виділяють такі класи обмежень цілісності:

- декларативні;
- процедурні.

*Декларативні обмеження цілісності* – це обмеження, що фіксують умови, яким має відповідати база даних. Завдання СКБД - не допускати порушення цих умов. Зазвичай декларативні обмеження цілісності визначаються мовою опису структури даних. Прикладом такого обмеження є: «Фонд зарплати факультету має бути рівним сумі фондів зарплати всіх його кафедр». Декларативні обмеження цілісності специфікуються фразою CONSTRAINT в описі таблиці або її полів.

*Процедурні обмеження цілісності* – це описи дій, спрямованих на забезпечення цілісності. Прикладом такого обмеження є: «Під час зміни фонду зарплати будь-якої з кафедр автоматично змінити фонд зарплати факультету так, щоб він дорівнював сумі фондів зарплати усіх кафедр». Процедурні обмеження цілісності специфікуються тригерами.

За часом перевірки обмеження цілісності поділяються на такі, що:

- перевіряються негайно;
- мають відкладену перевірку.

*Обмеження, що перевіряються негайно*, перевіряються безпосередньо у момент виконання операції, яка може порушити цілісність. Наприклад, неповторюваність назви факультету перевіряється під час додавання нового рядка до таблиці, яка містить інформацію про факультети, або під час заміни імені факультету в існуючому рядку таблиці. Якщо обмеження порушується, то операція блокується.

*Обмеження цілісності з відкладеною перевіркою* використовуються у тому випадку, коли для підтримання бази даних у несуперечному стані потрібно виконати дві або більше операції. Прикладом обмеження цілісності, яке не може бути перевірено негайно, є декларативне обмеження щодо фондів заробітної плати факультету та його кафедр. Після додавання нової кафедри з заданим фондом фінансування база даних переходить у суперечний стан, оскільки фонд фінансування факультету стає не рівним сумі фондів фінансування його кафедр. Для того щоб повернути базу даних у несуперечний стан, потрібно виконати ще одну операцію – оновити фонд фінансування факультету. У такому випадку ці дві операції об'єднуються в одну транзакцію і обмеження цілісності перевіряється після її завершення.

За областю дії розрізняють обмеження, що стосуються:

- відношення;
- атрибута;
- зв'язків між відношеннями;
- зв'язків між атрибутами.

За можливістю обмежувати переходи бази даних з одного стану в інший обмеження цілісності поділяються на:

- статичні;
- динамічні.

*Статичні обмеження цілісності* накладають обмеження на можливі стани бази даних. Наприклад, декларативні обмеження цілісності, що описуються далі, є статичними.

*Динамічні обмеження цілісності* задають обмеження на можливі переходи бази даних з одного стану в інший.

У підрозділах 8.2–8.4 ми розглянемо окремо кілька найважливіших класів обмежень цілісності. А саме, ці підрозділи будуть присвячені декларативним, динамічним та семантичним обмеженням. Вибір пояснюється тим, що виділені за різними критеріями класи обмежень значною мірою збігаються. Так, декларативні обмеження цілісності найчастіше бувають структурними і статичними, а динамічні – процедурними.

Слід зазначити, що механізми захисту даних, про які йтиметься в наступному розділі, також сприяють підтриманню цілісності бази даних. Обмежуючи доступ або маніпулювання даними для тих чи інших користувачів, ми забезпечуємо «недоступність» певних даних. Не все можна описати обмеженнями цілісності. Неможливо вказати і проконтролювати всі допустимі зміни бази даних. Тому, обмежуючи доступ до бази, ми значно знижуємо можливість неправильної зміни її стану. Якщо тій чи іншій особі заборонено змінювати стан бази даних, а дозволено лише вибирати дані, то вона буде не в змозі порушити цілісність.

## 8.2. Декларативні обмеження цілісності

Під час розгляду декларативних обмежень цілісності постають такі питання:

- на які об'єкти моделі даних поширюються обмеження цілісності;
- якими є ці обмеження;
- як ті чи інші обмеження цілісності специфікуються;
- які існують механізми підтримання цілісності.

У реляційних базах даних до об'єктів, на які поширюються обмеження цілісності, належать такі:

- відношення;
- атрибути;
- зв'язки між відношеннями;
- зв'язки між атрибутами.

У сучасних СКБД є й багато інших об'єктів бази даних, щодо яких специфікуються обмеження цілісності

Розглянемо, як задаються обмеження цілісності для об'єктів реляційних СКБД Специфікація буде подана мовою PL/SQL, що застосовується в СКБД Oracle і в якій для специфікації структурних обмежень цілісності використовується фраза CONSTRAINT (складова частина речень CREATE TABLE і ALTER TABLE).

### Цілісність відношень

У реляційній СКБД цілісність відношень визначається за допомогою первинного ключа, для якого мають виконуватися такі обмеження цілісності:

- атрибути первинного ключа не можуть містити NULL-значень;
- значення первинного ключа (як окремого атрибута або сукупності атрибутів) не можуть дублюватися в межах відношення.

Цілісність за первинним ключем специфікується так:

```
CONSTRAINT <ім'я обмеження> PRIMARY KEY (<перелік полів>)
```

де <перелік полів> — поля, що складають первинний ключ. Ця специфікація вказує, які саме поля є ключовими. Наприклад:

```
CREATE TABLE КАФЕДРА
  ( #D NUMBER(2),
    Назва VARCHAR2(9),
    Завідувач VARCHAR2(10),
    CONSTRAINT DepPK PRIMARY KEY (#D) )
```

Для специфікації можливих ключів використовується фраза UNIQUE.

Механізм підтримки цілісності відношень реалізує СКБД

### Цілісність атрибутів

У реляційній СКБД цілісність атрибутів може забезпечуватися:

- зазначенням типів даних та їхніх розмірів (обов'язкова властивість);
- визначенням, чи є обов'язковим значення атрибута (NULL/NOT NULL);
- визначенням, чи може значення атрибута дублюватися (UNIQUE);
- зміною значення атрибута після його введення;
- встановленням умов, яким мають відповідати значення атрибута.

Для похідних атрибутів цілісність має гарантуватися процедурою їхнього обчислення. Цілісність атрибутів специфікується фразами NULL/NOT NULL і UNIQUE у визначенні атрибута (поля).

Якщо унікальними мають бути значення не окремих полів, а сукупності полів, то це вказується так:

```
CONSTRAINT <ім'я обмеження> UNIQUE (<перелік полів>)
```

Цілісність атрибутів специфікується також зазначенням обмеження

```
CONSTRAINT <ім'я обмеження> CHECK (<умова>)
```

Умова визначається на атрибутах відношення. Фраза CHECK може також вказуватися в означенні атрибута.

Розглянемо приклад специфікації цілісності атрибутів:

```
CREATE TABLE ГРУПА
  ( #G NUMBER(3),
    #D NUMBER(3),
    Курс NUMBER(1) CHECK (Course IN(1,2,3,4,5)),
    Номер NUMBER(3) CHECK (Номер > 0),
```

```
Кількість NUMBER(2) CHECK (Кількість > 0),
#Куратор NUMBER(3) UNIQUE,
CONSTRAINT GrpPK PRIMARY KEY (#G),
CONSTRAINT GrpUNQ UNIQUE (Курс, Номер))
```

Зазначимо, що коли UNIQUE вказується для групи атрибутів, то йдеться про унікальність саме групи атрибутів, кожний з них окремо не обов'язково має бути унікальним. За допомогою фрази UNIQUE специфікуються також можливі ключі таблиці, тоді вона має використовуватися разом з обмеженням NOT NULL.

В Oracle відсутнє обмеження цілісності типу NOT CHANGE, яке вказує на те, що значення атрибута не мають змінюватися.

Механізм підтримки цілісності атрибутів реалізує СКБД.

### Цілісність зв'язків між відношеннями

Цілісність зв'язків між відношеннями визначається зовнішніми ключами. Під час специфікації зовнішнього ключа вказується відповідний йому первинний ключ іншого відношення. Такий первинний ключ називається *референційним*.

Відношення, на яке здійснюється посилання за допомогою зовнішнього ключа, називається *батьківським*, а те, яке посилається, – *дочірнім*.

Якщо зовнішній ключ певного відношення може містити NULL-значення, це означає, що зв'язок даного відношення з іншим є факультативним. NOT NULL-специфікація стовпців зовнішнього ключа свідчить, що зв'язок є обов'язковим

Означення зовнішнього ключа свідчить про наявність *цілісності за посиланням*: значення зовнішнього ключа має посилатися на значення первинного ключа іншого відношення. Тобто ситуація, коли зовнішній ключ має значення, відсутнє серед значень відповідного первинного ключа, розглядається як ситуація, що порушує цілісність за посиланням.

Цілісність за посиланням в Oracle специфікується так:

```
CONSTRAINT <назва обмеження> FOREIGN KEY (<перелік полів 1>)
REFERENCES <ім'я таблиці>(<перелік полів 2>)
[ON DELETE {CASCADE | SET NULL}]
```

де <перелік полів 1> – поля, що становлять зовнішній ключ, <ім'я таблиці> – ім'я таблиці референційного ключа, <перелік полів 2> – поля, з яких складається референційний ключ. В Oracle допускається, щоб <перелік полів 2> був не первинним ключем, а списком довільних полів, які мають специфікацію UNIQUE.

Механізм підтримки цілісності за посиланням реалізує СКБД. Розглянемо, як діє механізм під час видалення рядка з батьківської таблиці. Можливі три варіанти:

- рядок батьківської таблиці може бути видалений лише в тому випадку, якщо немає зовнішніх ключів, що посилаються на значення референційного ключа цього рядка;
- під час видалення рядка батьківської таблиці видаляються також усі рядки в інших таблицях, значення зовнішніх ключів яких посилаються на значення референційного ключа цього рядка (каскадне видалення);
- під час видалення рядка батьківської таблиці всі посилання на значення видаленого референційного ключа замінюються на NULL.

Відсутність фрази [ON DELETE {CASCADE | SET NULL}] вказує на перший варіант. Фраза ON DELETE CASCADE в специфікації референційної цілісності вказує на другий варіант, а ON DELETE SET NULL - на третій.

#### Цілісність зв'язків між атрибутами

Цілісність зв'язків між атрибутами визначається умовою, якій має відповідати сукупність атрибутів одного або кількох відношень. Такі умови специфікуються *тригерами*. Наприклад, аби вказати, що кількість студентів на лекції не може перевищувати кількість місць в аудиторії, слід означити такий тригер:

```
CREATE TRIGGER Лекція_Вставка_Оновлення
BEFORE INSERT, UPDATE ON ЛЕКЦІЯ
  WHEN (SELECT Місця FROM АУДИТОРІЯ WHERE АУДИТОРІЯ.#R =
  ЛЕКЦІЯ.#R) < (SELECT Кількість FROM ГРУПА WHERE ГРУПА.#G = ЛЕКЦІЯ.#G)
BEGIN
  ROLLBACK TRANSACTION
END
```

## 8.3. Динамічні обмеження цілісності

*Динамічні обмеження цілісності* – це обмеження, які встановлюють залежність між різними частинами бази даних у різні моменти часу. Виділяють такі різновиди динамічних обмежень:

- ситуаційно орієнтовані;
- операційно орієнтовані.

#### Ситуаційно-орієнтовані обмеження цілісності

*Ситуаційно-орієнтовані обмеження* задаються у вигляді вимог, яким мають відповідати послідовні стани бази даних, тобто завдяки таким обмеженням фіксуються допустимі переходи бази даних з одного стану в інший.

Приклад можливих переходів значення атрибута Сімейний_стан можна побачити на рис. 8.1.

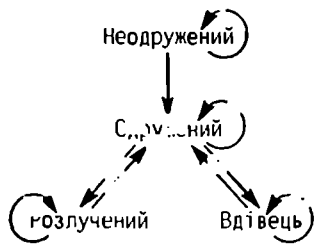



**Рис. 8.1.** Можливі переходи значення атрибута «Сімейний стан»

Для специфікації ситуаційно-орієнтованих обмежень застосовуються ті самі засоби, що й для специфікації структурних обмежень. Окрім того, використовується можливість посилатися на значення бази даних до її оновлення й після.

Це досягається за допомогою уточнюючих фраз OLD та NEW, що вживаються в посиланнях на значення, які зберігалися в базі даних до й після зміни її стану.

Зазвичай для опису динамічних обмежень використовуються тригери, які ініціюються під час зміни стану бази даних командами INSERT, UPDATE та DELETE. Наприклад, допустимі переходи атрибута Сімейний_стан можуть бути специфіковані у вигляді такого тригера:

```
CREATE TRIGGER Особа_Сімейний_Стан
BEFORE INSERT, UPDATE ON ОСОБА
WHEN (OLD.Сімейний_стан = 'Неодружений' AND
      NEW.Сімейний_стан IN ('Розлучений', 'Вдівець') ) OR
(OLD.Сімейний_стан = 'Одружений' AND
NEW.Сімейний_стан = 'Неодружений') OR
(OLD.Сімейний_стан = 'Розлучений' AND
NEW.Сімейний_стан IN ('Неодружений','Вдівець') ) OR
(OLD.Сімейний_стан = 'Вдівець' AND
NEW.Сімейний_стан IN ('Неодружений','Розлучений')
BEGIN
  ROLLBACK TRANSACTION
END
```

Це обмеження можна специфікувати в означенні таблиці ОСОБА за умови, що допускається використання кваліфікаторів NEW і OLD:

```
CONSTRAINT Сімейний_стан CHECK
(OLD.Сімейний_стан = 'Неодружений' AND NEW.Сімейний_стан = 'Одружений') OR
(OLD.Сімейний_стан = 'Одружений' AND
NEW.Сімейний_стан = IN ('Розлучений', 'Вдівець') ) OR
(OLD.Сімейний_стан = 'Розлучений' AND NEW.Сімейний_стан = 'Одружений') OR
(OLD.Сімейний_стан = 'Вдівець' AND NEW.Сімейний_стан = 'Одружений')
```

Зазначимо, що новий стан бази даних може залежати не лише від попереднього стану, але й від ланцюжка попередніх станів. Наприклад, у навчальному закладі може діяти таке правило: «На посаду професора можна приймати тільки тих, хто викладав у навчальному закладі на посадах викладача, старшого викладача і доцента». Як правило, СКБД не надають можливості зберігати передісторію станів. Якщо таке обмеження формулюється, то потрібно спроектувати базу даних так, щоб множина станів атрибутів зберігалася у спеціальних відношеннях. Зокрема для наведеного вище прикладу достатньо буде замість атрибута Посада у відношенні ВИКЛАДАЧ(#Т #D, Назва, Посада, Тел) ввести нове відношення ВИКЛАДАЧ_ПОСАДА(#Т, Посада, Дата), в якому зберігатимуться всі посади, на яких викладач працював раніше.

### Операційно-орієнтовані обмеження цілісності

*Операційно-орієнтовані обмеження* цілісності задаються у вигляді допустимих послідовностей операцій. Допустимість операції або послідовності операцій може залежати від поточного стану бази даних. Наведемо приклад операційно-орієнтованого обмеження цілісності: «Додавання в базу даних інформації про те, що х є чоловіком у допустиме лише в тому випадку, коли і х, і у в даний момент не є одруженими». Зазвичай подібні обмеження реалізуються за допомогою означення тригерів для відповідних операцій маніпулювання таблицями. Наприклад, якщо

є відношення ПОДРУЖЖЯ(Чоловік, Дружина) та ОСОБА(Прізвище, Сімейний_стан), то описане вище обмеження можна специфікувати так:

```
CREATE TRIGGER Подружжя_Перевірка
  BEFORE INSERT ON ПОДРУЖЖЯ
  WHEN (ПОДРУЖЖЯ.Чоловік = ОСОБА.Прізвище AND
    ОСОБА.Сімейний_Стан = 'Одружений') AND NEW
   (ПОДРУЖЖЯ.Жінка = ОСОБА.Прізвище AND ОСОБА.Сімейний_Стан = 'Одружена')
BEGIN
  ROLLBACK TRANSACTION
END
```

## 8.4. Семантичні обмеження цілісності

*Семантичні обмеження цілісності*, або прикладні правила цілісності, – це правила, які характеризують обмеження, що діють у предметній області. Прикладами семантичних обмежень можуть бути:

- описане вище правило зміни стану атрибута Сімейний_стан;
- «лист, що надійшов, вважається обробленим, коли з ним ознайомлені всі зацікавлені особи і щодо нього ухвалене відповідне рішення»;
- «розмір посадових окладів має варіюватися в межах від 3000 до 5000 грн.»;
- «студент може перейти на наступний курс або залишитися на тому самому, але не на попередньому»;
- «одна й та ж особа не може бути завідувачем двох кафедр».

Для специфікації семантичних обмежень використовують фразу CONSTRAINT і тригери.

У деяких випадках для підтримки семантичних обмежень цілісності пишуться спеціальні процедури, які зберігаються в СКБД, або спеціальні процедури, що входять до складу зовнішнього застосування.

## 8.5. Підтримка цілісності у разі виникнення перебоїв

Нагадаємо, що цілісність бази даних - це її відповідність модельованій предметній області у будь-який момент часу. Механізми опису обмежень цілісності забезпечують підтримку цілісності з «логічної» точки зору. Проте перебої в програмному або апаратному забезпеченні також можуть призвести до порушення цілісності (а в деяких випадках до повного руйнування бази даних). Для забезпечення цілісності бази даних на цей випадок пропонуються такі механізми:

- періодичне створення резервної копії бази даних;
- ведення журналу всіх змін стану бази даних.

Загальна схема підтримки цілісності на випадок перебоїв є такою. У певний момент часу створюється резервна копія бази даних. Починаючи з цього моменту в журналі фіксуються всі зміни, що виконуються в базі. Якщо в якийсь момент часу база даних виявляється настільки зіпсованою, що її неможливо відновити, то береться резервна копія і до неї застосовуються всі зафіксовані в журналі операції. У такий спосіб резервна копія стає *актуальною*.

## Контрольні запитання та завдання

Після вивчення матеріалу цього розділу пропонуємо вам виконати наведене далі завдання.

Для бази даних вищого навчального закладу, означеної в розділах 3, 4 та 5, запишіть мовою SQL такі обмеження цілісності.

У таблиці ФАКУЛЬТЕТ:

- #F — первинний ключ;
- Назва — можливий ключ, обов'язковий;
- Фонд — не може бути від'ємним;
- Корпус - може набувати значень з множини {"1", "2а", "2б", "3", "4", "5", "6"}.

У таблиці КАФЕДРА:

- #D — первинний ключ;
- #F — зовнішній ключ, обов'язковий, посилається на #F з таблиці ФАКУЛЬТЕТ (каскадне видалення);
- Назва, #ЗАВІДУВАЧ — у сукупності становлять можливий ключ;
- Корпус — може набувати значень з множини {"1", "2а", "2б", "3", "4", "5", "6"}.

У таблиці ВИКЛАДАЧ:

- #T — первинний ключ;
- #D — зовнішній ключ, не обов'язковий, посилається на #D з таблиці КАФЕДРА (каскадне видалення);
- Посада — може набувати значень зі списку {«лаборант», «викладач», «доцент», «професор»}.

У таблиці ГРУПА:

- #G — первинний ключ;
- #D — зовнішній ключ, обов'язково посилається на #D з КАФЕДРА (каскадне видалення);
- Курс — обов'язковий, набуває значень з множини {1, 2, 3, 4, 5};
- Номер — обов'язковий, додатне число;
- Кількість — обов'язковий, допустимі значення належать інтервалу від 0 до 100;
- #Куратор — необов'язковий, зовнішній ключ, посилається на #T з таблиці ВИКЛАДАЧ;

У таблиці АУДИТОРІЯ:

- #R — первинний ключ;
- Корпус — може набувати значень з множини {"1", "2а", "2б", "3", "4", "5", "6"};
- Місткість — обов'язковий, додатне число;

У таблиці ЛЕКЦІЯ:

- #T #G #S #R — у сукупності утворюють первинний ключ;
- #T — обов'язковий зовнішній ключ, посилається на #T з ВИКЛАДАЧ;
- #G — обов'язковий зовнішній ключ, посилається на #G з ГРУПА;
- #S — обов'язковий зовнішній ключ, посилається на #S з ПРЕДМЕТ;
- #R — обов'язковий зовнішній ключ, посилається на #R з АУДИТОРІЯ;
- Тип — обов'язковий, набуває значень з множини {"лекція", "практикум", "лабораторна", "консультація"};
- День — обов'язковий, набуває значень з множини {"пн", "вт", "ср", "чт", "пт", "сб", "нд"};
- Тиждень — обов'язковий, набуває значень з множини {1, 2}.

# Розділ 9

# Захист баз даних

- ✦ Реєстрація користувачів та керування правами доступу
- ✦ Специфікація повноважень в Oracle
- ✦ Обов'язкові методи захисту
- ✦ Ведення журналів доступу
- ✦ Обхід системи захисту

## 9.1. Безпека даних

Дані в системах баз даних мають зберігатися з гарантуванням конфіденційності та безпеки. Інформація не може бути загубленою або викраденою. Під безпекою даних у базі розуміють захист даних від випадкового або спланованого доступу до них осіб, які не мають на це права, від несанкціонованого розкриття, зміни або видалення.

Безпека даних підтримується комплексом заходів і засобів.

- ✦ *Організаційно-методичні заходи* передбачають розроблення інструкцій та правил, які регламентують доступ до даних та їхнє використання, а також створення відповідних служб і підрозділів, які стежать за дотриманням цих правил.
- ✦ *Правові та юридичні заходи* передбачають юридичне закріплення прав і обов'язків щодо зберігання, використання й передавання в електронному вигляді даних, які підлягають захисту, на рівні державних законів та інших нормативних документів.
- ✦ *Технічні засоби захисту* — це комплекс технічних засобів, які сприяють вирішенню проблеми захисту даних.
- ✦ *Програмні засоби захисту* — це комплекс математичних, алгоритмічних і програмних засобів, що сприяють вирішенню проблеми захисту даних.

Далі йтиметься лише про програмні засоби захисту.

*Система захисту* — це сукупність заходів, що вживаються в системі баз даних для гарантування необхідного рівня безпеки.

У сучасних СКБД підтримується один з двох найбільш розповсюджених методів забезпечення захисту даних: вибірковий чи обов'язковий.

*Вибірковий метод захисту* передбачає, що користувачі мають різні права (привілеї, повноваження) доступу до різних або одних тих самих об'єктів бази даних.

*Обов'язковий метод захисту* передбачає, що кожному об'єкту бази даних надається певний рівень секретності, а кожному користувачу – певний рівень допуску. Доступ до об'єкта даних є лише в тих користувачів, які мають відповідний для цих даних рівень допуску.

Зазначимо, що вибірковий метод гнучкіший, аніж обов'язковий

Безпека даних може гарантуватися такими механізмами.

- **Реєстрація користувачів.** Будь-який користувач для отримання доступу до бази даних має бути зареєстрований у системі під певним ім'ям і певним паролем.
- **Керування правами доступу.** Адміністратор може визначити, яким користувачам до яких даних дозволяється доступ і які саме операції над цими даними (вибирання, введення, зміну чи видалення) він може виконувати.
- **Ідентифікація та підтвердження автентичності всіх користувачів або застосувань, що отримують доступ до бази даних.** Будь-який користувач або застосування, звертаючись до системи баз даних, мають вказати своє ім'я і пароль Ім'я ідентифікує користувача, а пароль підтверджує автентичність імені. Ці два кроки – ідентифікація та підтвердження автентичності – виконуються лише один раз під час з'єднання з базою даних і залишаються чинними до завершення сеансу роботи з базою даних конкретного користувача чи застосування.
- **Автоматичне ведення журналів доступу до даних.** У цих журналах протоколюються операції, виконані над даними користувачами, з метою подальшого аналізу на випадок отримання доступу до бази в обхід системи захисту.
- **Шифрування даних на зовнішніх носіях.** Здійснюється криптографічними методами на випадок несанкціонованого копіювання даних із зовнішніх носіїв.

Припускається, що адміністратор бази даних має всі необхідні повноваження на виконання функцій, пов'язаних із захистом даних.

### Довірче й адміністративне керування доступом

*Довірче керування доступом до даних* – це такий тип керування, коли система захисту дає змогу звичайним користувачам не лише отримувати доступ до певних даних, але й передавати повноваження на доступ до них іншим користувачам без адміністративного втручання. *Адміністративне керування доступом до даних* – це таке керування, за якого система захисту дає змогу передавати повноваження на доступ до даних лише спеціально авторизованому користувачу (адміністратору).

## 9.2. Реєстрація користувачів

Будь-який користувач для отримання доступу до бази даних має бути зареєстрований у системі під певним ім'ям і паролем. Реєстрація необхідна для того, щоб знати, з ким має справу система у кожний момент часу. Зазначимо, що під користувачем розуміється не лише фізична особа, але й будь-яке джерело, що в змозі звернутися до бази даних (прикладна програма, операційна система, інтернет-застосування тощо).

Спрощений вигляд конструкції, що застосовується для означення користувача, є таким:

```
CREATE USER <користувач> UDENTIFIED <пароль>
```

Тут <користувач> — ім'я користувача, а <пароль> - пароль.

Спочатку користувач не має жодних повноважень. Щоб користувач міг виконувати ті чи інші операції над базою даних, йому необхідно передати відповідні повноваження.

## 9.3. Керування правами доступу

Визначити права доступу користувачів — це означає зафіксувати інформацію стосовно:

- осіб, яким надаються права доступу;
- умов надання прав доступу;
- об'єктів, на які поширюються права доступу;
- операцій, щодо яких специфікуються права доступу;
- можливості передавання прав доступу іншим особам.

Схематично аспекти керування правами доступу зображені на рис. 9.1.

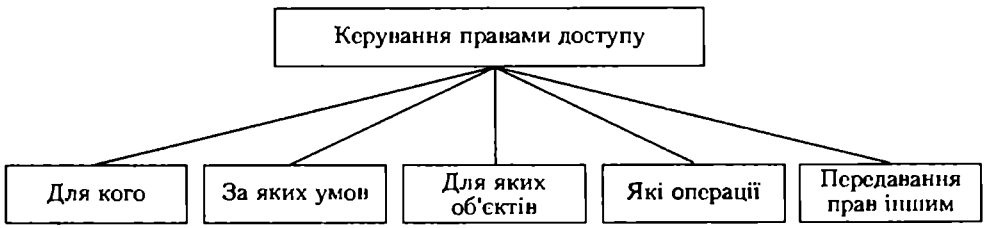



**Рис. 9.1.** Складові прав доступу

Розглянемо коротко всі ці аспекти.

### 9.3.1. Кому надаються права доступу

Права доступу надаються всім, хто звертається до бази даних. Це можуть бути користувачі, прикладні програми, операційні системи тощо. Кожен, хто звертається до бази даних, має насамперед вказати своє ім'я і пароль. Для СКБД не важливо, хто саме звертається до бази даних, головне, щоб усі, хто хоче отримати можливість працювати з нею, були заздалегідь зареєстровані в системі

У деяких випадках може знадобитися виділити користувачів системи, які мають однакові повноваження. Наприклад, усі співробітники відділу кадрів можуть мати одні й ті ж права, а співробітники планового відділу — інші, причому також однакові. Для реалізації такого розподілу прав вводиться поняття ролі.

*Роль* — це сукупність повноважень, які можуть передаватися користувачам або іншим ролям. Можна присвоїти повноваження ролям, а згодом приписувати

ролі користувачам. Коли користувачу присвоєна роль, він має ті повноваження, які приписані ролі.

Роль має всі повноваження, які присвоєні їй явно, й усі повноваження, які передані їй іншими ролями.

Спрощений синтаксис означення ролі є таким:

```
CREATE ROLE <роль> IDENTIFIED <пароль>
```

Тут <роль> — ім'я ролі.

Спочатку роль є порожньою, тобто не має жодного повноваження.

## 9.3.2. Умови надання прав доступу

Іноді доцільно специфікувати додаткові умови, за дотримання яких користувачам надаються певні права доступу. Йдеться про умови, які не визначаються іншими складовими прав доступу (кому надається доступ, до яких даних, які операції дозволяються).

Приклади додаткових умов:

- часові характеристики (наприклад, «права доступу діють лише між 16 і 17 годинами першого понеділка кожного місяця»);
- локалізація комп'ютерів у локальній мережі (наприклад, «права доступу діють лише для комп'ютерів, установлених у плановому відділі»).

У більшості СКБД немає засобів явного опису додаткових умов, що обмежують права доступу. За необхідності такі умови можуть бути специфіковані у прикладних системах.

## 9.3.3. Об'єкти, на які поширюються права доступу

Зазвичай у контексті прав доступу розрізняють об'єкти двох класів: системні й об'єкти бази даних. До системних об'єктів належать: база даних, кластери, тригери, транзакції тощо. До об'єктів бази даних належать таблиці, віртуальні таблиці та процедури. Крім того, в таблицях і віртуальних таблицях можуть додатково вказуватися стовпці, щодо яких специфікуються права доступу.

У деяких випадках виникає необхідність специфікувати рядки певної таблиці, що стосуються особи, для якої визначаються права доступу. Наприклад:

- будь-який користувач може змінювати у відношенні СЛУЖБОВЕЦЬ значення полів лише того рядка, який стосується його самого (тобто він може змінювати лише свої особисті дані), за винятком величини його зарплати;
- у відношенні СЛУЖБОВЕЦЬ змінювати зарплату може лише той користувач, який є начальником відділу даного службовця.

У розглянутій ситуації виникає необхідність посилатися на поточного користувача в умові вибирання записів із віртуальної таблиці. Для цього в деяких СКБД надається можливість використовувати спеціальну константу, яка ідентифікує поточного користувача (тобто користувача, який ініціює виконання відповідного запиту на вибирання або оновлення даних).

### 9.3.4. Операції, щодо яких специфікуються права доступу

До операцій, стосовно яких специфікуються права доступу, належать стандартні операції з маніпулювання об'єктами бази даних, а саме:

- означення, переозначення й видалення таблиці або віртуальної таблиці;
- вибирання, додавання, видалення, оновлення рядків у таблиці або віртуальній таблиці:
- виконання збережених процедур.

У кожній конкретній СКБД наведений список операцій над об'єктами бази даних може розширюватися.

### 9.3.5. Можливість передавання прав доступу іншим особам

Інколи користувачу надаються не лише права на маніпулювання тими чи іншими об'єктами бази даних, але й можливість передавати ці права іншим. Наприклад, користувачу передається право створити таблицю, вводити в неї дані, змінювати і видаляти їх, навіть видалити всю таблицю. Він стає повноправним власником таблиці і може з нею робити що завгодно. Тому не дивно, що йому надають дозвіл на передавання будь-яких прав на цю таблицю іншим користувачам.

## 9.4. Специфікація повноважень в СКБД Oracle

Розглянемо, як означуються права доступу до даних в СКБД Oracle. Інструкція, що дозволяє означити користувача, була описана в підрозділі 9.2. Щойно створений користувач не має жодних повноважень. Надання повноважень користувачу здійснюється командою GRANT.

#### Передавання повноважень користувачам або ролям

Команда GRANT дає змогу передати повноваження користувачам або ролям.

Спрощений синтаксис команди є таким:

```
GRANT { <операція> | ALL} [(<список стовпців>)] ON <список об'єктів> TO
    {<користувач> | <роль> | PUBLIC} WITH GRANT OPTION
```

Специфікатор <операція> вказує, щодо яких операцій описується повноваження. Такими операціями можуть бути: SELECT, INSERT, UPDATE, DELETE та інші. Фраза ALL вказує, що повноваження передаються щодо всіх операцій.

У списку об'єктів зазначаються всі об'єкти бази даних, щодо яких специфікуються повноваження. Цей список містить посилання на таблиці й віртуальні таблиці. За допомогою параметра <список стовпців> можна додатково вказати стовпці таблиць.

Повноваження може бути передане користувачу, ролі або всім, хто вказується у фразі TO.

Якщо повноваження передається користувачу, він одержує право виконувати операції згідно з цим повноваженням. Коли повноваження передається ролі, вона ним поповнюється.

Фраза WITH GRANT OPTION означає, що одержувач повноваження отримує також право передавати це повноваження іншому користувачу або ролі

Розглянемо кілька прикладів надання й передавання прав доступу:

1. Надати користувачу на ім'я Джон права на виконання будь-яких операцій над таблицею ФАКУЛЬТЕТ і дозвіл на передавання цих прав іншим користувачам.

```
GRANT ALL ON ФАКУЛЬТЕТ TO Джон WITH GRANT OPTION
```

2. Надати всім користувачам право переглядати дані з таблиці Лекція.

```
GRANT SELECT ON ЛЕКЦІЯ TO PUBLIC
```

3. Надати користувачу на ім'я Джон право змінювати стовпець Фонд у таблиці КАФЕДРА.

```
GRANT UPDATE (Фонд) ON КАФЕДРА TO Джон
```

## 9.5. Обов'язкові методи захисту

Обов'язкові методи захисту, або методи обов'язкового керування доступом застосовуються до баз, в яких дані мають статичну або жорстку структуру, характерну, наприклад, для урядових або військових організацій. Як уже наголошувалося, основна ідея полягає в тому, що кожний об'єкт даних має певний рівень секретності, а кожний користувач – певний рівень доступу. Передбачається, що ці рівні утворюють строгий ієрархічний порядок (наприклад, цілком таємно, таємно, для службового користування тощо).

Останнім часом методи обов'язкового керування доступом набули популярності в США, оскільки згідно з вимогами міністерства оборони цієї країни такий метод керування доступом має підтримуватися в будь-якій системі, що використовується в цьому відомстві. Тому розробникам і постачальникам СКБД довелося докласти чимало зусиль для розробки подібних методів керування доступом. Вимоги до такого керування подано в двох важливих публікаціях міністерства, які неформально називаються «оранжевою» та «рожевою» книгами (Orange Book та Lavender Book). В «оранжевій» книзі наведено перелік вимог до безпеки певного «надійного обчислювального середовища», а в «рожевій» книзі ці вимоги інтерпретуються стосовно систем керування базами даних.

## 9.6. Ведення журналів доступу

Не буває невразливих систем захисту. Настирливий зловмисник завжди може знайти спосіб подолати всі системи контролю і захисту. Тому під час роботи з дуже важливими даними необхідно реєструвати у відповідному журналі всі події, що стосуються функціонування системи захисту. У зв'язку з цим система захисту має надавати можливість виконувати наведені далі дії.

- Реєструвати всі спроби отримання доступу до системи баз даних, у тому числі й безуспішні. Ця реєстрація має бути максимально повною (повинні реєструватися відомості про те, хто отримував доступ або намагався його отримати, з якого термінала або вузла мережі, о котрій годині тощо).
- Реєструвати дії користувачів з використання всіх ресурсів системи, зокрема й даних, а також інші дії та події, які можуть вплинути на захист даних.
- Надавати користувачам, які мають адміністративні повноваження, можливість переглядати й аналізувати результати реєстрації, виявляти небезпечні ситуації, встановлювати причини їхнього виникнення та знаходити користувачів, відповідальних за порушення політики безпеки.

Навіть сама лише констатація факту, що в системі підтримується реєстрація всіх дій користувачів, у деяких випадках має суттєве значення для запобігання несанкціонованому проникненню в систему.

## 9.7. Обхід системи захисту

До носіїв, на яких записані дані, можна отримати доступ в обхід системи захисту. Якщо отримати доступ до файлів операційної системи, які містять дані з бази, то можна прочитати вміст цих файлів, оскільки фірми-розробники переважної більшості СКБД створюють формати зберігання даних такими, що вони є доступними широкому колу користувачів і застосувань. Більше того, той, хто отримав доступ до файлів бази, може змінити їхній вміст або навіть видалити їх.

Найефективнішими засобами боротьби з такою загрозою є використання методів криптографії, тобто шифрування інформації. Коли дані записані в базі у зашифрованому вигляді, тоді той, хто отримав доступ до них в обхід системи захисту, стикається з проблемою дешифрування. В ідеальному випадку метод шифрування має бути таким, щоб витрати на дешифрування перевищували виграш від володіння інформацією або щоб час дешифрування перевищував час, протягом якого дані розглядаються як секретні.

У системах розподілених баз даних, коли інформація пересилається каналами зв'язку, небезпечними є різні форми перехоплення даних, що передаються. Єдиним контрзаходом, який дає змогу запобігти подібним перехопленням, є шифрування даних перед їхнім передаванням каналами зв'язку.

## Контрольні запитання та завдання

1. За допомогою яких засобів підтримується безпека даних?
2. Чим відрізняється довірче керування доступом від адміністративного?
3. Як специфікуються повноваження в Oracle?
4. У чому полягає ідея обов'язкових методів захисту?
5. В який спосіб ведення журналів доступу може підвищити безпеку даних?
6. Як захистити дані, доступ до яких можливий поза системою захисту СКБД?

# Розділ 10

# Розподілені бази даних

- Архітектура розподілених баз даних
- Розподілене зберігання даних
- Обчислення розподілених запитів
- Обробка розподілених транзакцій

## 10.1. Основні означення

*Розподілена база даних* (РБД) – це множина логічно взаємозалежних баз даних. розподілених у комп'ютерній мережі.

*Розподілена система керування базами даних* (РСКБД) – це програмне забезпечення, яке керує РБД і надає такі механізми доступу до них, що їх застосування дає користувачу можливість працювати з РБД як з однією цілісною базою даних.

*Розподілена система баз даних* (РСБД) – це РБД разом із РСКБД.

Не слід плутати РСБД з централізованою базою даних, що використовується в мережі (рис. 10.1). У цьому випадку база даних розташована на одному з комп'ютерів, а всі інші мають доступ до неї через комунікаційну мережу. Не є розподіленою також база даних, що працює в середовищі багатопроцесорних комп'ютерів. У цьому випадку ми маємо справу з паралельною БД.

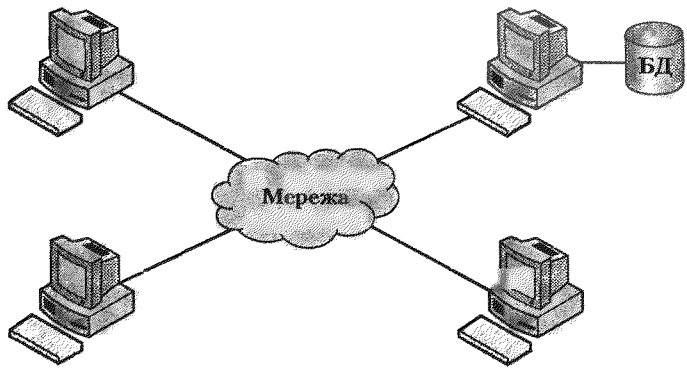



**Рис. 10.1.** Централізована база даних у комп'ютерній мережі

Архітектура розподіленої СКБД наведена на рис. 10.2. Кожний з вузлів мережі містить свою базу даних, однак вони розглядаються як логічно єдина база, а не як сукупність розкиданих у мережі файлів. Усі дані є логічно взаємозалежними.

Розподілена СКБД – це повноцінна СКБД, що виконує всі необхідні функції з керування даними.

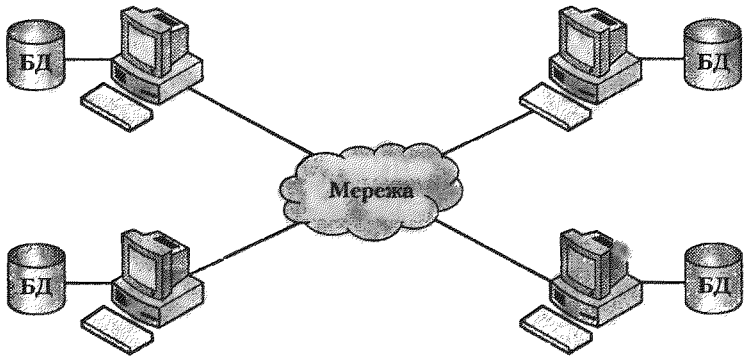



**Рис. 10.2.** Розподілена база даних

Залежно від типу програмного забезпечення розрізняють два типи РСКБД:

- однорідні;
- неоднорідні.

### Однорідні РСКБД

В *однорідних РСКБД* передбачається, як мінімум, що на всіх вузлах використовуються однотипні СКБД (наприклад, реляційні), які мають схожі функціональні можливості. Як максимум, припускається, що на всіх вузлах використовуються однакові технічні засоби, тобто однакові типи комп'ютерів і програмного забезпечення Це стосується операційних систем, програмного забезпечення СКБД та моделей даних, що підтримуються.

### Неоднорідні РСКБД

У *неоднорідних РСКБД* вузли базуються на різних програмно-технічних платформах, які можуть містити різні типи СКБД. Окрім того, такі СКБД можуть підтримувати різні моделі даних. У цьому випадку ускладнюється вирішення проблеми їхньої взаємодії.

### Прозорість доступу до даних

Однією з важливих проблем РСКБД є досягнення логічної незалежності даних від місця зберігання, тобто *прозорість доступу до даних*. Це означає, що користувач повинен мати можливість сприймати всі необхідні йому дані як єдине ціле, не зважаючи на те, в який спосіб вони розподілені у мережі.

Розглянемо такий приклад. Нехай база даних містить відношення СЛУЖБОВЕЦЬ, ПРОЕКТ, РОБОТА з інформацією стосовно службовців, проектів, що розроблюються, та участі службовців у проектах компанії, яка має філії в Римі, Лондоні, Парижі й Токіо. Бази даних цих філій містять дані згідно зі схемою, зображеною на рис. 10.3. Будь-який користувач у кожному з вузлів цієї мережі має сприймати логічну

структуру бази даних так, ніби всі три відношення містяться в одній локальній базі даних. Розглянемо приклад.

```
SELECT    ПРОЕКТ.Назва, СЛУЖБОВЕЦЬ.Назва
FROM      СЛУЖБОВЕЦЬ, ПРОЕКТ, РОБОТА
WHERE     СЛУЖБОВЕЦЬ.#E = РОБОТА.#E AND РОБОТА.#P = ПРОЕКТ.#P AND
          (ПРОЕКТ.Місце = 'Париж' OR ПРОЕКТ.Місце ='Токіо') AND
          СЛУЖБОВЕЦЬ.Зарплата > 40000 AND СЛУЖБОВЕЦЬ.Зарплата < 60000
```

Цей запит, згідно з яким вибираються назви проектів, що виконуються в Парижі чи Токіо, й імена службовців, які беруть у них участь, із зарплатою в діапазоні від 40 000 до 60 000, буде виконано навіть тоді, коли його формулює користувач, що перебуває в Лондоні.

З погляду користувача існує єдина розподілена СКБД і єдина база даних, у якій є всі дані, що визначені у віртуальній таблиці користувача

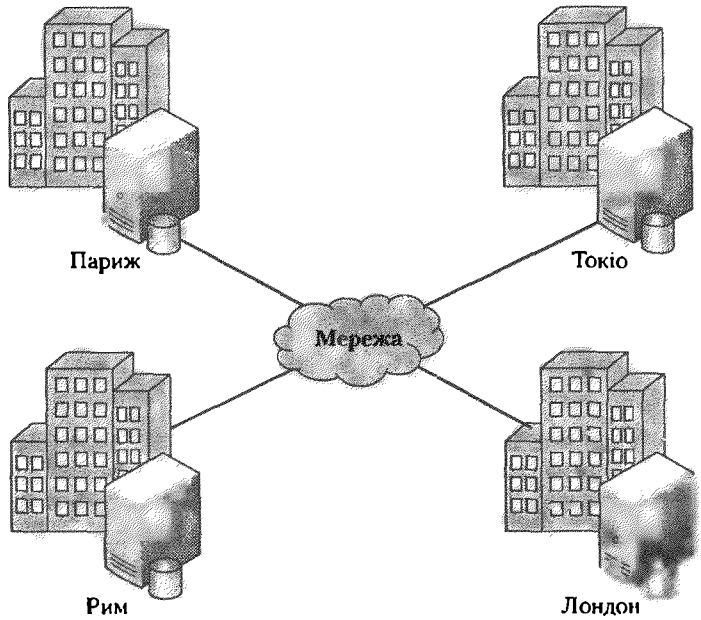



**Рис. 10.3.** Приклад розміщення даних у РБД

## 10.2. Логічна архітектура розподілених баз даних

*Логічна архітектура розподілених баз даних* — це архітектура логічно взаємозалежних даних. Вона дещо нагадує архітектуру ANSI/SPARC, яка розглядалася в розділі 1. Графічне зображення логічної архітектури розподілених баз даних

наведене на рис. 10.4. Особливість архітектури РБД, порівняно з архітектурою ANSI/SPARC, полягає в тому, що виникає ще один рівень, *глобальний концептуальний*, завданням якого (як і в моделі ANSI/SPARC) є зображення концептуальної моделі предметної області в цілому. На цьому рівні описується характер розподілу даних.

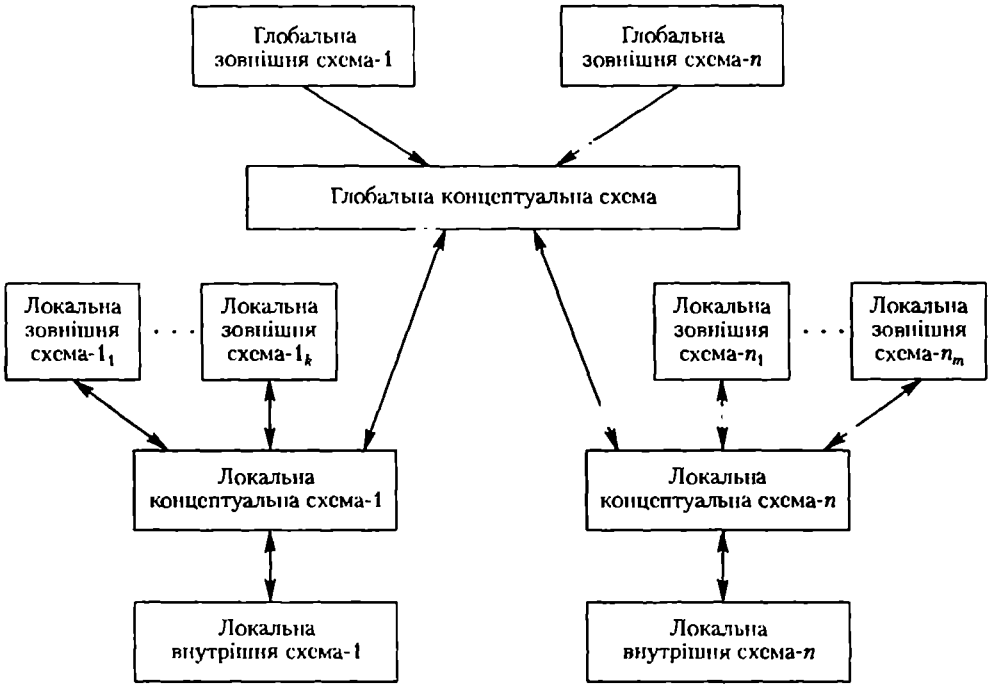



**Рис. 10.4.** Логічна архітектура розподіленої СКБД

На *локальному концептуальному рівні* здійснюється локальний опис ПО. Тобто схема цього рівня містить опис тільки тієї частини предметної області, що є специфічною для конкретного вузла розподіленої ПО; дані інших вузлів розподіленої ПО в схемі не вказуються. Відображення «глобальний-концептуальний/локальний-концептуальний» дають змогу зобразити глобальну концептуальну схему у вигляді сукупності локальних концептуальних схем, і навпаки.

*Глобальна зовнішня схема* зображує дані, необхідні користувачам і застосуванням, у бажаному для них вигляді, незалежно від способу розподілу даних за вузлами. Це завдання вирішується завдяки тому, що глобальні зовнішні схеми базуються на глобальній концептуальній схемі.

*Локальні зовнішні схеми* базуються на локальних концептуальних схемах, відтак вони можуть посилатися лише на ті дані, що розташовані на відповідному вузлі розподіленої ПО.

*Локальні внутрішні схеми* — це схеми зберігання даних на конкретних вузлах розподіленої ПО. Вони пов'язані з відповідними локальними концептуальними схемами.

## 10.3. Архітектура програмно-технічних засобів розподілених СКБД

### Властивості архітектури

Архітектура програмно-технічного комплексу розподілених СКБД має три головні характеристики:

- розподіленість;
- неоднорідність;
- автономність.

Розглянемо ці характеристики детальніше.

#### Розподіленість

Спосіб розподілу компонентів системи баз даних за комп'ютерами мережі визначається тим, чи є в мережі єдиний комп'ютер з повноваженнями розподіленої СКБД, або ж їх кілька. Якщо таких комп'ютерів кілька, то чи є між ними взаємодія тощо.

#### Неоднорідність

Важливою характеристикою є міра однорідності програмно-технічних засобів СКБД. Ідеться про технічне забезпечення (типи комп'ютерів), комунікаційні засоби зв'язку, операційні системи і типи баз даних (моделі даних, мови запитів, алгоритми керування транзакціями тощо).

#### Автономність

Автономність характеризує, наскільки самостійно компоненти СКБД можуть виконувати свої функції. До питань автономності належать:

- **автономність проектних рішень** (наскільки самостійно компоненти СКБД можуть розв'язувати задачі, які були спроектовані для них);
- **автономність вирішенням комунікаційних проблем** (наскільки самостійно компоненти СКБД можуть встановлювати зв'язки з іншими компонентами);
- **автономність обчислень** (наскільки самостійно компоненти СКБД можуть приймати рішення щодо виконання локальних операцій).

Діаметрально протилежними підходами до вирішення цих питань є створення єдиної централізованої СКБД та встановлення СКБД у кожному з вузлів мережі.

Зазначимо, що перелічені характеристики мають не лише розподілені СКБД, але й усі розподілені системи обробки даних.

### Різновиди архітектури

Основними різновидами архітектури програмно-технічних засобів розподіленої СКБД є:

- клієнт-серверна архітектура;
- архітектура з багатьма незалежними серверами;
- архітектура із взаємодіючими серверами;
- архітектура одноранговой мережі.

Розглянемо ці типи архітектури детальніше

***Клієнт-серверна архітектура***

Така архітектура передбачає наявність єдиного комп'ютера-сервера і багатьох комп'ютерів-клієнтів, що взаємодіють між собою через канали зв'язку. На сервері розташована СКБД та інтегрована (централізована) база даних. Ніякого розподілу баз даних за вузлами мережі немає. На клієнтських комп'ютерах виконуються застосування, які працюють із серверною базою даних, а також розміщене програмне забезпечення для зв'язку з віддаленою СКБД. Як клієнти, так і сервер оснащені комунікаційним програмним забезпеченням.

**Архітектура з багатьма незалежними серверами**

Ця архітектура передбачає існування багатьох серверів, що мають доступ до своїх локальних баз даних. У такому випадку на комп'ютерах клієнтів має зберігатись інформація стосовно того, які дані на яких серверах розташовані, а також має бути розміщене програмне забезпечення, що дає змогу декомпозувати запити для їхнього виконання на різних серверах і потім об'єднати результати.

**Архітектура із взаємодіючими серверами**

У цьому випадку передбачається, що кожний із серверів містить повну інформацію стосовно того, які дані на яких серверах зберігаються, а також здатний обробляти розподілені запити. У разі потреби один сервер може звернутися до іншого для одержання необхідних даних. Кожен клієнт має свій сервер, до якого звертається для виконання операцій над розподіленою базою даних.

**Архітектура одноранговоі мережі**

Усі комп'ютери мережі є серверами. На кожному комп'ютері розміщено розподілену СКБД і базу даних, з кожного комп'ютера можна надіслати до іншого запит на отримання необхідних даних.

## 10.4. Розподілене зберігання даних

У розподіленій СКБД дані розподіляються за вузлами мережі. Існують два основні механізми розподіленого зберігання даних:

- фрагментація;
- реплікація

Розглянемо їхню реалізацію в реляційній моделі даних.

### 10.4.1. Фрагментація

Суть фрагментації полягає в тому, щоб поділити логічну базу даних на фрагменти з метою зберігання кожного фрагмента на певному вузлі мережі. Одиницями фрагментації можуть бути відношення та *складені відношення*. У випадку, коли одиницею фрагментації є відношення, вирішується проблема, яке відношення в якій базі даних має зберігатися. За іншого підходу допускається, що будь-яке відношення може бути зображене у вигляді сукупності фрагментів, що розподіляються за різними базами даних.

Фрагментацію, що здійснюється розподілом відношень за базами даних, теоретично здійснити нескладно, тому розглянемо проблему фрагментації власне відношень.

### Фрагментація відношень

Завдання фрагментації відношень формулюється в такий спосіб. Нехай задане відношення $R$. Його потрібно зобразити у вигляді сукупності відношень $R_1, ..., R_n$ так, щоб ця сукупність відповідала критеріям ефективності (за часом доступу, пам'яттю, завантаженістю комп'ютерів тощо).

Фрагментація є *коректною*, якщо вона повна, не містить перетинів і може бути реконструйована. Пояснимо ці терміни.

Декомпозиція відношення $R$ на фрагменти $R_1, R_2, ..., R_n$ є *повною* тоді й лише тоді, коли кожен елемент даних з $R$ належить якомусь із відношень $R_i$.

Декомпозиція відношення $R$ на фрагменти $R_1, R_2, ..., R_n$ може бути *реконструйована*, якщо існує такий реляційний вираз $\varphi(R_1, R_2, ..., R_n)$, що $R = \varphi(R_1, R_2, ..., R_n)$.

Декомпозиція відношення $R$ на фрагменти $R_1, R_2, ..., R_n$ *не містить перетинів*, якщо будь-який елемент даних з $R$ міститься не більш ніж в одному фрагменті.

Є три типи фрагментації відношень:

- горизонтальна;
- вертикальна;
- змішана

Розглянемо кожний з цих різновидів фрагментації.

#### Горизонтальна фрагментація

Горизонтальна фрагментація полягає в розподілі кортежів відношення за фрагментами. Формально горизонтальну фрагментацію можна визначити в такий спосіб. Нехай задане відношення $R$ і на ньому визначений предикат $F_i$. Тоді *горизонтальний фрагмент* $R_i$ відношення визначається так:

$$R_i = \sigma_{F_i}(R).$$

Тобто горизонтальний фрагмент $R_i$ — це множина кортежів $R$, що задовольняють умову $F_i$.

Коректність горизонтальної фрагментації може бути встановлена в такий спосіб. Нехай на відношенні $R$ задано множину предикатів $F = \{F_i, i = \overline{1,k}\}$. Ця множина породжує відповідну множину фрагментів $R_1, ..., R_k$. Множина $F$ є повною стосовно $R$ (відтак і горизонтальна фрагментація теж є повною), якщо

$$R_i = \bigcup_{i=\overline{1,k}} \sigma_{F_i}(R).$$

Фрагментація є повною тоді й лише тоді, коли $F_1 \vee F_2 \vee ... \vee F_k$ є завжди істинною формулою. Таку фрагментацію можна реконструювати за щойно наведеною формулою. Якщо для будь-якої пари $F_i$ і $F_j$ з множини $F$, $i \neq j$ формула $F_i \,\&\, F_j$ є завжди хибною, то фрагментація, породжувана предикатами з множини $F$, не міститиме перетинів.

Розглянемо кілька прикладів. Нехай задане відношення ПРОЕКТ(Рном, Назва, Тип, Вартість). Очевидно, що множина предикатів

```
{ Вартість < 300000, Вартість = 300000, Вартість > 300000}
```

породжує повну фрагментацію відношення ПРОЕКТ, яка не містить перетинів. Предикати

```
{ Вартість < 300000, Вартість > 200000}
```

породжують повну фрагментацію, що містить перетин. А предикати

```
{ Вартість < 200, Вартість > 300}
```

породжують неповну фрагментацію, яка не містить перетинів. Нарешті предикати

```
{Тип = 'держзамовлення', Тип = 'приватний', Тип = 'ініціативний'}
```

визначають фрагментацію без перетинів, а їхня повнота залежить від домену атрибута Тип. Усі чотири фрагментації можуть бути реконструйовані.

**Породжена горизонтальна фрагментація**

Якщо два відношення сполучені зв'язком типу «один-до-багатьох», то під час фрагментації відношення в закінченні «один» (відношення-власник) може породжувати фрагментацію відношення, що розташоване в закінченні «багато» (відношення-член). А саме, розташовуючи той чи інший кортеж відношення-власника на одному з вузлів, можна на ньому ж розташувати всі ті кортежі відношення-члена, що зв'язані з вихідним кортежем. Це буде доцільним, оскільки в багатьох запитах кортежі відношення-власника обробляються разом із відповідними кортежами відношення-члена.

Розглянемо приклад. Нехай задано відношення

```
ПРОЕКТИ(#Рном, Назва, Тип, Вартість)
СЛУЖБОВЦІ(#Сном, #Рном, Ім'я)
```

Зв'язок між ними, встановлений за допомогою зовнішнього ключа #Рном у відношенні СЛУЖБОВЦІ, вказує, в яких проектах беруть участь службовці.

Нехай відношення ПРОЕКТИ поділяється на два фрагменти згідно з такими предикатами:

```
{Вартість < 300000, Вартість >= 300000}
```

Відповідно до цих предикатів породжуються два фрагменти відношення СЛУЖБОВЦІ. Перший із них міститиме всіх службовців, які беруть участь у проектах вартістю менше 30 000, а другий — службовців, які беруть участь у проектах вартістю не менше 30 000.

Породжена фрагментація формально може бути визначена в такий спосіб. Нехай задано відношення $R$ та $S$ і у відношенні $S$ продубльований атрибут $A$ відношення $R$. Нехай відношення $R$ поділене на горизонтальні фрагменти $\{R_1, R_2, ..., R_k\}$ згідно з предикатами $\{F_1, F_2, ..., F_k\}$, визначеними на атрибуті $A$. Поділ відношення $S$ на фрагменти $\{S_1, S_2, ..., S_k\}$ називається *породженою горизонтальною фрагментацією*, якщо будь-який фрагмент $S_i$ визначається за такою формулою:

$$S_i = \sigma_{S[A]=R_i[A]}(S).$$

Розглянемо властивості повноти, можливості реконструкції і відсутності перетинів стосовно породженої фрагментації.

Якщо у відношенні $S$ атрибут $A$ є обов'язковим зовнішнім ключем з відношення $R$, то визначена вище породжена горизонтальна фрагментація є повною. Що стосується можливості реконструкції, то відношення $S$ відновлюється з породженої фрагментації так само, як і під час звичайного відновлення відношення з фрагментів. Якщо зв'язок між відношеннями $R$ і $S$ має множинність «один-до-багатьох», то фрагментація відношення $S$, породжена фрагментацією без перетинів відношення $R$, також не містить перетинів.

Переваги горизонтальної фрагментації:

- дає змогу паралельно обробляти фрагменти відношення;
- дає можливість поділяти відношення так, щоб кортежі розташовувалися там, де вони будуть використовуватися найчастіше.

**Вертикальна фрагментація**

Суть вертикальної фрагментації полягає в тому, що відношення поділяється на дві чи більше проекцій, тобто схема відношення поділяється на певну множину підсхем.

Для відновлення вихідного відношення з фрагментів необхідно, щоб усі підсхеми містили первинний ключ. Існує інший підхід, коли під час поділу схеми до кожного фрагмента автоматично додається поле з ідентифікатором кортежу. Значення цього ідентифікатора зазвичай встановлюються системою автоматично.

Розглянемо приклад. Нехай задано відношення ПОСТАЧАННЯ(#Рном, Постачальник, Обладнання, Проект).

ПОСТАЧАННЯ

| #Рном | Постачальник | Обладнання | Проект |
|---|---|---|---|
| Р1 | Іванов | Двигун | АН-24 |
| Р2 | Іванов | Двигун | ЯК-40 |
| Р3 | Іванов | Шасі | АН-24 |
| Р4 | Петров | Двигун | АН-24 |
| Р5 | Петров | Електрообладнання | ЯК-40 |
| Р6 | Петров | Електрообладнання | АН-70 |

Вертикальна фрагментація з використанням первинного ключа може виглядати так:

ПОСТАЧАННЯ-1

| #Рном | Постачальник |
|---|---|
| Р1 | Іванов |
| Р2 | Іванов |
| Р3 | Іванов |
| Р4 | Петров |
| Р5 | Петров |
| Р6 | Петров |

ПОСТАЧАННЯ-2

| #Рном | Обладнання | Проект |
|---|---|---|
| Р1 | Двигун | АН-24 |
| Р2 | Двигун | ЯК-40 |
| Р3 | Шасі | АН-24 |
| Р4 | Двигун | АН-24 |
| Р5 | Електрообладнання | ЯК-40 |
| Р6 | Електрообладнання | АН-70 |

Розглянемо умови коректності вертикальної фрагментації.

Нехай задано відношення $R$, що визначене на множині атрибутів $A$ з ключем $K$. Це відношення поділене на вертикальні фрагменти $F_R = \{R_1, R_2, ..., R_n\}$, кожний з яких має атрибути $A_{R_i}$. Повнота вертикальної фрагментації означає, що

$$A = \bigcup_{i=1,n} A_{R_i}.$$

Можливість реконструкції вертикальної фрагментації означає, що

$$R = (R_1 * R_2 *...* R_n)[A].$$

Природне з'єднання виконується за первинним ключем чи ідентифікатором кортежу. Відсутність перетинів означає, що будь-який атрибут з множини $A$, за винятком первинного ключа та ідентифікатора кортежу, не входить до двох чи більше фрагментів водночас.

Переваги вертикальної фрагментації:

- дає змогу поділяти кортежі відношення так, щоб їхні частини розташовувалися там, де вони використовуватимуться найчастіше;
- дає змогу проводити паралельну обробку відношень;
- наявність ідентифікатора кортежу дає можливість здійснювати ефективне з'єднання вертикальних фрагментів.

**Змішана фрагментація**

Змішана фрагментація передбачає послідовне застосування вертикальної і горизонтальної фрагментацій. Приклад змішаної фрагментації зображений на рис. 10.5.

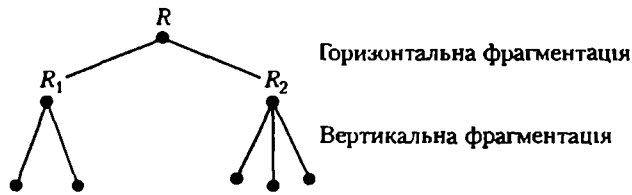



**Рис. 10.5.** Приклад змішаної фрагментації

## Розподіл даних за вузлами мережі

Після отримання усіх необхідних фрагментів відношень постає проблема розподілу цих фрагментів за вузлами мережі. Єдиних рекомендацій стосовно того, як це робити, немає.

Нехай задано:

$F = \{F_1, F_2, ..., F_n\}$ – множину фрагментів логічної бази даних;

$S = \{S_1, S_2, S_m\}$ – множину вузлів розподіленої СКБД;

$Q = \{Q_1, Q_2, ..., Q_k\}$ – множину застосувань, що мають функціонувати в мережі.

Потрібно знайти оптимальний розподіл фрагментів $F$ за вузлами мережі $S$ за умови, що відомий розподіл застосувань $Q$ за вузлами мережі.

Визначаючи оптимальність, слід враховувати різні параметри, зокрема:

- вартість передавання, зберігання й обробки даних;
- часові характеристики;
- продуктивність;
- обмеження (наприклад, ємнісні характеристики вузлів мережі).

Для того щоб розподілити фрагменти за вузлами мережі, необхідна додаткова інформація, яка стосується

- бази даних та розмірів фрагментів;
- застосувань (місць їхнього розташування, частоти використання тих чи інших фрагментів для вибирання даних, частоти використання фрагментів для відновлення даних);
- вузлів мережі (вартості зберігання й обробки даних у вузлах);
- мережі (вартості та часових характеристик передавання даних між двома вузлами).

## 10.4.2. Реплікація

*Реплікація* – це механізм розподілу даних за вузлами, що дозволяє зберігати копії тих самих даних на різних вузлах мережі з метою прискорення пошуку і підвищення стійкості до відмов. Відношення чи фрагмент є *репліковаим*, якщо його копії зберігаються на двох або більше вузлах (копії ще називають *репліками*). За *повної реплікації* відношення його копії зберігаються на всіх вузлах мережі. Допускається ситуація, коли вся база даних зберігається на всіх вузлах мережі – це називається *повною реплікацією* бази даних.

Переваги реплікації:

- доступність (у разі перебою в роботі вузла, що містить відношення $R$, його доступність на інших вузлах зберігається);
- паралелізм (виконання запитів до відношення $R$ може бути розпаралелено за всіма репліками відношення);
- зниження вартості передавання даних (відношення $R$ доступне локально в усіх вузлах, де є його репліки).

Недоліки реплікації:

- підвищується вартість зберігання, створення і відновлення даних;
- підвищуються вимоги до ресурсів;
- ускладнюється підтримання цілісності даних, наприклад одночасне відновлення різних реплік одного й того ж відношення

### Механізми реплікації

Для реалізації реплікації використовуються три сервери: видавець, дистриб'ютор і передплатник.

*Видавцем* називають сервер, що надає розміщені на ньому дані для копіювання на інші сервери. Окрім створення копії даних, видавець відстежує внесені до його бази даних зміни і готує нову копію.

*Дистриб'ютором* називається сервер, що підтримує розподілену базу даних. Він виконує роль посередника, копіює всі публікації, підготовлені видавцем, і пересилає їх передплатникам. Дистриб'ютором може бути виділений сервер або сервер, сконфігурований як видавець чи передплатник. Конкретні функції, що їх виконує дистриб'ютор, залежать від методів реплікації.

*Передплатником* називається сервер, що отримує копії даних, надані видавцем. Механізми зміни даних передплатником відрізняються від механізмів зміни даних видавцем.

Є два методи відновлення даних передплатників:

- **Реплікація за запитом.** Передплатник періодично звертається до дистриб'ютора із запитом про зміни, що відбулися з моменту останнього з'єднання.
- **Примусова реплікація.** Дистриб'ютор сам встановлює з'єднання з передплатником і пересилає йому необхідні дані.

Залежно від методу реплікації, передплатники можуть чи не можуть вносити зміни в репліковані дані. У найпростішому випадку змінювати дані може тільки видавець, у складніших моделях реплікації — передплатники і видавці. Змінені дані, отримані від усіх передплатників, синхронізуються і поєднуються з даними видавця, а потім розсилаються передплатникам.

### Моделі реплікації

Є такі моделі реплікації:

- реплікація моментальних знімків;
- реплікація транзакцій.

*Реплікація моментальних знімків* є найпростішою моделлю реплікації. *Моментальний знімок* - це повна копія даних, обраних для реплікації; вона розсилається передплатникам.

Під час *реплікації транзакцій* використовується журнал транзакцій бази даних. Обрані транзакції копіюються в базу даних дистриб'ютора зі збереженням інформації про послідовність їхнього виконання, потім розсилаються серверам-передплатникам і виконуються на них у тому ж порядку, в якому виконувалися на сервері-видавці.

Цей механізм зменшує завантаження мережі, його рекомендується використовувати у великих базах даних з невеликою кількістю змін.

### Топологія реплікацій

Топологія реплікацій описує характер взаємозв'язків між учасниками реплікації:

- реплікація «один-до-багатьох» передбачає наявність одного видавця і кількох передплатників;
- реплікація «багато-до-одного» має місце, коли дані від кількох видавців пересилаються одному передплатнику;
- реплікація «багато-до-багатьох» означає, що дані від кількох видавців пересилаються кільком передплатникам.

## 10.5. Обчислення розподілених запитів

Для централізованих СКБД критерієм вартості обчислення запитів є кількість операцій доступу до зовнішньої пам'яті. У розподілених СКБД слід брати до уваги й інші аспекти, а саме:

- вартість передавання даних мережею,
- можливість обчислювати частини запиту в різних вузлах паралельно.

### 10.5.1. Обчислення запитів на нефрагментованих відношеннях

#### Обчислення декартового добутку

Нехай відношення $R_1, R_2, ..., R_n$ розподілені за вузлами $S_1, S_2, ..., S_n$ відповідно. Результат запиту $R_1 \times R_2 \times ... \times R_n$, ініційованого у вузлі $T$, розміщується також у вузлі $T$. Очевидно, що оптимальним є обчислення, коли всі відношення пересилаються до вузла $T$, де й обчислюється декартів добуток.

#### Обчислення з'єднання відношень

Нехай відношення $R_1, R_2, ..., R_n$ розподілені за вузлами $S_1, S_2, ..., S_n$ відповідно. Результат запиту $R_1 \bowtie R_2 \bowtie ... \bowtie R_n$, ініційованого у вузлі $T$, розміщується також у вузлі $T$.

Розглянемо можливі стратегії обчислення запиту.

- Переслати копії всіх $n$ відношень до вузла $T$ і там виконати обчислення
- Переслати копію $R_1$ до вузла $S_2$, виконати з'єднання $R_1 \bowtie R_2$, переслати його результат на вузол $S_3$ і з'єднати його з відношенням $R_3$, і так далі до вузла $S_n$ включно. Кінцевий результат пересилається з вузла $S_n$ до вузла $T$.
- Можливі також варіанти попередньої стратегії, коли змінюється порядок використання вузлів $S_1, S_2, ..., S_n$.
- У деяких випадках найефективнішою є стратегія напівз'єднання, що розглядатиметься далі.

Обираючи стратегію, слід взяти до уваги такі фактори:

- обсяги даних, що пересилаються;
- вартість пересилання даних між вузлами;
- відносна швидкість обробки даних на кожному з вузлів.

#### Стратегія напівз'єднання

Нехай відношення $R_1$ розташоване у вузлі $S_1$, а відношення $R_2$ у вузлі $S_2$, і вони мають спільний атрибут $A$. Необхідно обчислити вираз $R_1 \bowtie R_2$, де з'єднання виконується за атрибутом $A$, і розмістити результат у вузлі $S_1$.

Алгоритм обчисленя виразу $R_1 \bowtie R_2$

1. Обчислити вираз $R_1' \leftarrow \pi_A(R_1)$ у вузлі $S_1$.
2. Переслати $R_1'$ з вузла $S_1$ на вузол $S_2$.

3. Обчислити вираз $R'_2 \leftarrow R_2 \bowtie R'_1$ у вузлі $S_2$.
4. Переслати з $S_2$ на $S_1$.
5. Обчислити $R_1 \bowtie R'_2$ у $S_1$. Це і буде результатом операції $R_1 \bowtie R_2$.

Напівз'єднання відношення $R_1(C)$ з відношенням $R_2$, що позначається як $R_1 \ltimes R_2$, обчислюється в такий спосіб:

$$R_1 \ltimes R_2 = \pi_C(R_1 \bowtie R_2).$$

Тобто $R_1 \ltimes R_2$ - це множина кортежів відношення $R_1$, які беруть участь у формуванні з'єднання $R_1 \bowtie R_2$. З'єднання може бути виконане за допомогою операцій напівз'єднання:

$$R \bowtie S = (R \ltimes S) \bowtie S = (S \ltimes R) \bowtie R = (R \ltimes S) \bowtie (S \ltimes R).$$

В описаному вище алгоритмі $R'_2 = R_2 \bowtie R_1$. На випадок з'єднання кількох відношень даний алгоритм розширюється послідовним застосуванням серії напівз'єднань

Алгоритм напівз'єднання виявляється ефективнішим за пересилання $R_2$ на вузол $S_1$, якщо сумарний обсяг відношень $R'_1$ та $R'_2$ є меншим за обсяг відношення $R_2$.

## 10.5.2. Обчислення запитів на фрагментованих відношеннях

В усіх наведених далі прикладах будуть використовуватися такі відношення:

EMP(#ENO, ENAME, TITLE) — службовці
PROJ(#PNO, PNAME, BUDGET, LOC) — проекти
ASG(#ENO, #PNO, RESP, DUR) — розподіл службовців за проектами

Нехай відношення ASG и EMP розподілені за вузлами у такий спосіб:

Вузол 1: $ASG_1 = \sigma_{\#ENO \le \text{'E3'}}(ASG)$
Вузол 2: $ASG_2 = \sigma_{\#ENO > \text{'E3'}}(ASG)$
Вузол 3: $EMP_1 = \sigma_{\#ENO \le \text{'E3'}}(EMP)$
Вузол 4: $EMP_2 = \sigma_{\#ENO > \text{'E3'}}(EMP)$

У вузлі 5 потрібно зберегти результат виконання такого запиту:

```
SELECT   ENAME
FROM     EMP, ASG
WHERE    EMP.#ENO = ASG.#ENO AND DUR > 37
```

На рис. 10.6 зображені дві стратегії обчислення цього запиту.

У першому випадку обчислення переміщуються в ті вузли, де розташовані дані, а в другому — дані переміщуються в той вузол, де необхідно отримати результат обчислень. У більшості випадків перший варіант виявляється кращим. У зв'язку з цим постає проблема: як декомпозувати реляційний запит, знаючи розподіл даних за вузлами. Далі розглянемо спосіб вирішення цієї проблеми

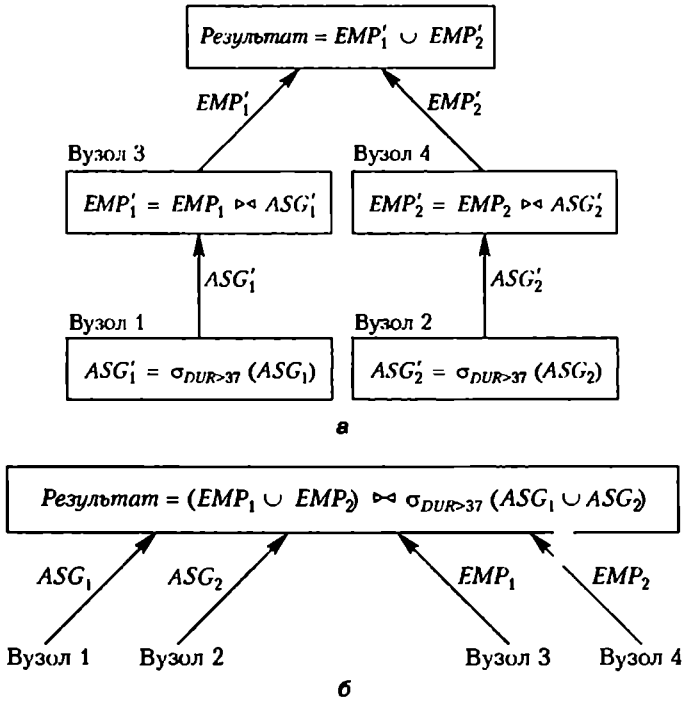



**Рис. 10.6.** Стратегії обчислення запиту з фрагментованими відношеннями: переміщення обчислень у вузли (*а*); переміщення відношень у вузли (*б*)

### Відношення з горизонтальною фрагментацією

Основна ідея обчислення запитів на відношеннях з горизонтальною фрагментацією полягає в тому, що відношення $R$, поділене на фрагменти $\{R_1, R_2, ..., R_n\}$, можна зобразити у вигляді об'єднання фрагментів, тобто

$$R = R_1 \cup R_2 \cup ... \cup R_n.$$

Розглянемо приклад. Нехай задано запит, за яким потрібно вибрати всіх службовців, крім Dio, що працювали над проектом «CAD/CAM» 12 чи 24 місяці:

```
SELECT  Ename
FROM    PROJ, ASG, EMP
WHERE   ASG.#ENO = EMP.#ENO  AND
        ASG.#PNO = PROJ.#PNO AND
        Ename ≠ 'Dio' AND
        PROJ.Name = 'CAD/CAM' AND
        (Dur=12 OR Dur=24)
```

Нехай відношення *EMP* поділене на такі фрагменти:

$EMP_1 = \sigma_{\#ENO \le \text{'E3'}}(EMP)$

$EMP_2 = \sigma_{\text{'E3'} < \#ENO \le \text{'E6'}}(EMP)$

$EMP_3 = \sigma_{\#ENO \ge \text{'E6'}}(EMP)$

*ASG* поділено на фрагменти $ASG_1$ і $ASG_2$ у такий спосіб:

$ASG_1 = \sigma_{\#ENO \leq \text{'E3'}}(ASG)$;

$ASG_2 = \sigma_{\#ENO > \text{'E3'}}(ASG)$.

Тоді схема обчислення запиту матиме такий вигляд, як зображено на рис. 10.7.

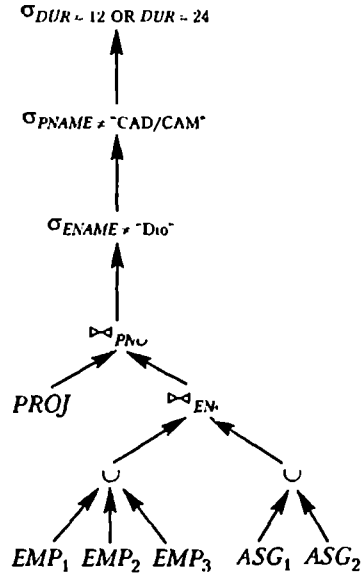



**Рис. 10.7.** Схема обчислення запиту на відношеннях з горизонтальною фрагментацією

Для подальшої оптимізації обчислення запиту можна скористатися еквівалентними перетвореннями, описаними в підрозділі 3.2.3.

### Обчислення селекції

Якщо в запиті до відношення, поділеного на горизонтальні фрагменти, застосовується селекція, то деякі з фрагментів можуть не брати в ній участі, й тому їх можна видалити.

Формально це можна записати так. Нехай задане відношення $R$ із фрагментами $\{R_1, R_2, ..., R_k\}$, де $R_j = \sigma_{P_j}(R)$, $j = \overline{1,k}$. Тоді для довільної умови $F$

$$\sigma_F(R) = \sigma_F(R_1 \cup R_2 \cup ... \cup R_k) = \sigma_F(R_1) \cup \sigma_F(R_2) \cup ... \cup \sigma_F(R_k).$$

Тоді $\sigma_{P_i} = \varnothing$, якщо $\forall x \in R: \neg(P_i\{x\} \& F\{x\})$. Отже, якщо певні $\sigma_F(R_i)$ виявляться порожніми множинами, їх можна не брати до уваги.

Розглянемо приклад. Нехай задано запит:

```
SELECT   *
FROM     EMP
WHERE    #ENO ='E5'
```

Для нього можна виконати перетворення, зображені на рис. 10.8.

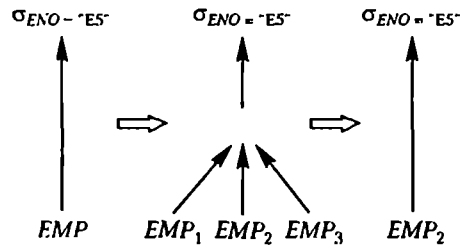



**Рис. 10.8.** Оптимізація обчислення селекції

### Обчислення з'єднання

Основна ідея полягає в тому, щоб:

- зобразити відношення, що з'єднуються, у вигляді об'єднань їхніх фрагментів;
- перемістити операцію з'єднання до фрагментів;
- видалити з'єднання тих чи інших фрагментів, якщо наперед відомо, що з'єднання порожні;
- виконати інші допустимі перетворення, які сприяють підвищенню ефективності обчислення запиту.

Припустимо, що задано відношення $R$ і $S$ із фрагментами $\{R_1, R_2, ..., R_k\}$ та $\{S_1, S_2, ..., S_m\}$ відповідно. Тоді мають місце такі співвідношення:

$$R \bowtie S = (R_1 \cup R_2 \cup ... \cup R_k) \bowtie (S_1 \cup S_2 \cup ... \cup S_m) =$$

$$(R_1 \bowtie (S_1 \cup S_2 \cup ... \cup S_m)) \cup (R_2 \bowtie (S_1 \cup S_2 \cup ... \cup S_m)) \cup ...$$
$$... \cup (R_k \bowtie (S_1 \cup S_2 \cup ... \cup S_m)) =$$

$$= (R_1 \bowtie S_1) \cup (R_1 \bowtie S_2) \cup ... \cup (R_1 \bowtie S_m) \cup (R_2 \bowtie S_1) \cup (R_2 \bowtie S_2) \cup ...$$
$$... \cup (R_2 \bowtie S_m) \cup ... \cup (R_k \bowtie S_1) \cup (R_k \bowtie S_2) \cup ... \cup (R_k \bowtie S_m)$$

Розглянемо приклад. Нехай схема обчислення запиту має такий вигляд, як зображено на рис. 10.7. Тоді можна опустити операцію з'єднання якомога нижче. На найнижчому рівні отримаємо шість пар фрагментів відношень, з'єднання яких можна обчислювати паралельно (рис 10.9).

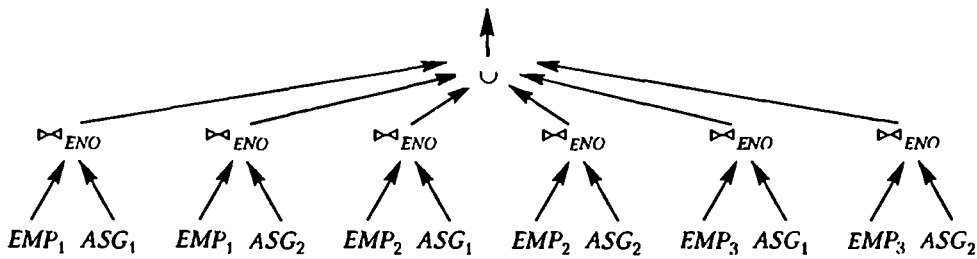



**Рис. 10.9.** Розпаралелювання схеми виконання запиту

Знаючи семантику фрагментів, можна помітити, що з'єднання деяких пар фрагментів дають порожні множини, тому їх можна видалити зі схеми обчислень. У результаті отримаємо схему, що зображена на рис. 10.10.

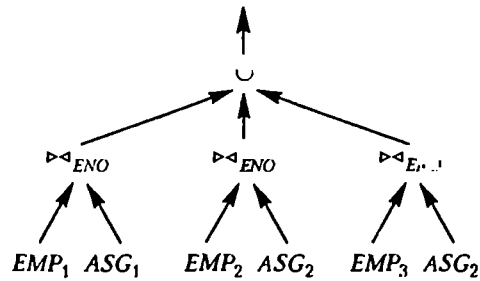



**Рис. 10.10.** Схема виконання запиту з видаленими «порожніми» вершинами

### Відношення з вертикальною фрагментацією

Основна ідея обчислення запитів на відношеннях з вертикальною фрагментацією полягає в тому, що відношення $R(A)$, яке є поділеним на вертикальні фрагменти $\{R_1, R_2, ..., R_n\}$, можна подати у вигляді природного з'єднання фрагментів, тобто $R = (R_1 * R_2 * ... * R_n)[A]$

За подальшої оптимізації можуть застосовуватися всі допустимі еквівалентні перетворення виразів реляційної алгебри та евристичні алгоритми, що враховують семантику запитів.

Однією з таких евристик є поняття явно порожньої проекції.

Нехай у відношенні $R$, заданому на множині атрибутів $A = \{A_1, A_2, ..., A_n\}$, виділений вертикальний фрагмент $R_i = \pi_{A'}(R)$, де $A' \subseteq A$. Проекція $\pi_B(R_i)$ називається *явно порожньою*, якщо $B \cap A' = \varnothing$, тобто жоден з атрибутів $B$ не належить множині атрибутів $A'$.

Якщо серед проекцій відношення є явно порожні, їх можна видалити.

Розглянемо приклад. Нехай задано вертикальні проекції:

$EMP_1 = \pi_{\#ENO,\ Ename}(EMP)$,

$EMP_2 = \pi_{\#ENO,\ Title}(EMP)$,

а також задано запит:

```
SELECT   Ename
FROM     EMP
```

Тоді оптимізація обчислення запиту може виконуватися згідно зі схемою, наведеною на рис. 10.11.

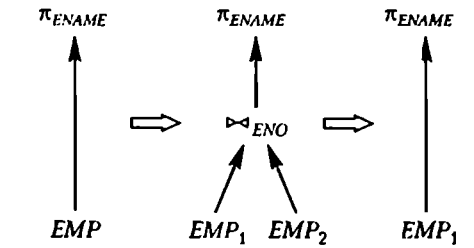



**Рис. 10.11.** Оптимізація схеми виконання запиту на вертикально фрагментованому відношенні

### Відношення з породженою горизонтальною фрагментацією

Основними правилами обчислення запитів на таких відношеннях є:

- перенесення з'єднань всередину об'єднань;
- застосування правил обчислення з'єднань для горизонтальних фрагментів.

Розглянемо приклад.

Основна фрагментація:

$EMP_1$: $\sigma_{TITLE="Hacker"}(EMP)$

$EMP_2$: $\sigma_{TITLE\neq"Hacker"}(EMP)$

Породжена фрагментація:

$ASG_1$: $ASG \ltimes_{\#ENO} EMP$

$ASG_2$: $ASG \ltimes_{\#ENO} EMP_2$

Запит:

```
SELECT   *
FROM     EMP, ASG
WHERE    ASG.#ENO=EMP.#ENO AND EMP.Title = 'Hacker'
```

На рис. 10.12 проілюстровано процес оптимізації схеми обчислення запиту.

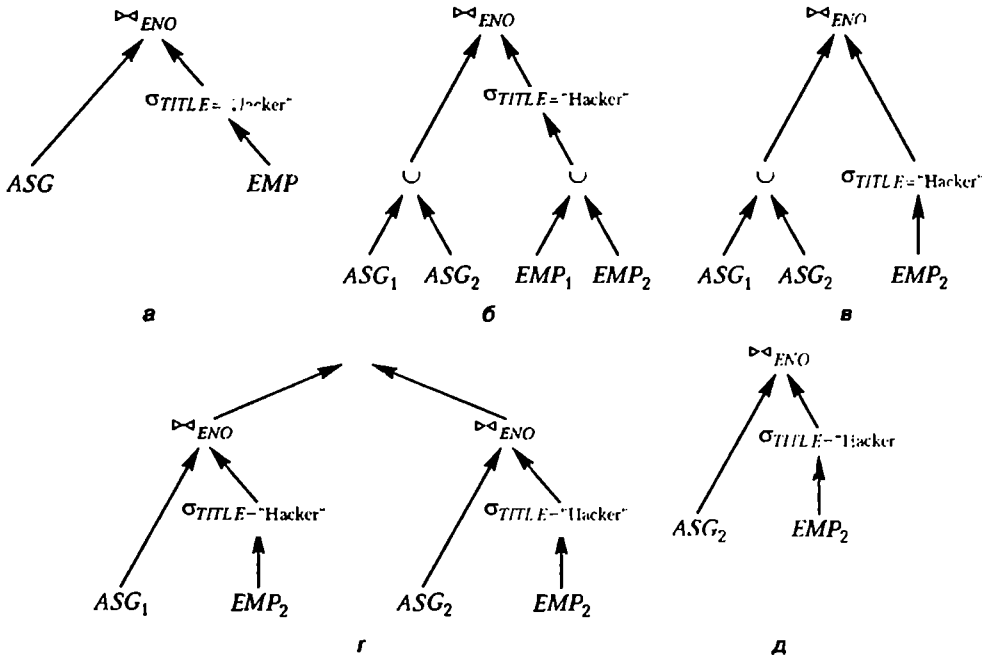



**Рис. 10.12.** Оптимізація схеми виконання запиту на відношеннях з породженою горизонтальною фрагментацією: схема виконання вихідного запиту (а); зображення відношень їхніми фрагментами (*б*); опускання селекції, $EMP_1$ видаляється, оскільки селекція на ньому дає порожнє відношення (*в*); перенесення з'єднання всередину об'єднання (*г*); видалення порожніх проміжних відношень — всього лівого піддерева (*д*)

### Відношення зі змішаною фрагментацією

Для обчислення запитів на відношеннях, які мають змішану фрагментацію, слід застосовувати правила, описані для горизонтальної і вертикальної фрагментацій, а саме.

- видаляти порожні відношення, що породжуються селекціями горизонтальних фрагментів;
- видаляти явно порожні вирази, що породжуються проекціями вертикальних фрагментів;
- опускати з'єднання під об'єднання з метою ізоляції й видалення явно порожніх з'єднань.

Нехай задано змішану фрагментацію:

$EMP_1 = \sigma_{\#ENO \leq "E4"}(\pi_{\#ENO,ENAME}(EMP))$

$EMP_2 = \sigma_{\#ENO > "E4"}(\pi_{\#ENO,ENAME}(EMP))$

$EMP_3 = \pi_{\#ENO,\ TITLE}(EMP)$

Відношення EMP виражається через свої фрагменти у такий спосіб:

$$EMP = (EMP_1 \cup EMP_2) \bowtie EMP_3$$

Розглянемо запит:

```
SELECT   Ename
FROM     EMP
WHERE    #ENO = 'E5'
```

Процес оптимізації схеми виконання цього запиту проілюстровано на рис. 10.13.

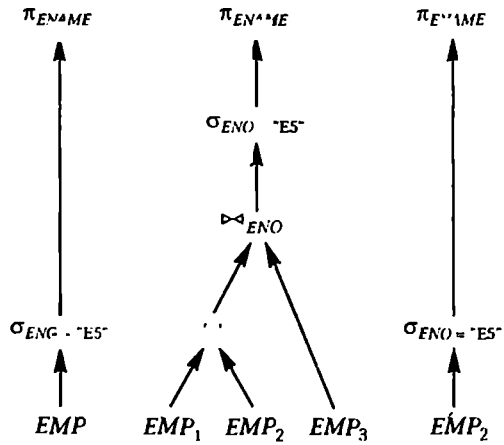



**Рис. 10.13.** Оптимізація схеми виконання запиту на відношеннях зі змішаною фрагментацією

У кінцевому виразі фрагмент $EMP_1$ відсутній, оскільки селекція на ньому дає порожнє відношення, а фрагмент $EMP_3$ відсутній, оскільки проекція на цьому фрагменті дає явно порожній вираз.

# 10.6. Обробка розподілених транзакцій

*Транзакція* – набір команд, що виконується як єдине ціле. У транзакції або всі команди будуть виконані, або жодна з них не виконається. Якщо хоча б одна з команд транзакції не може бути виконана, здійснюється *відкочування* (відновлюється стан системи, в якому вона перебувала до початку виконання транзакції).

## 10.6.1. Вимоги ACID

Транзакції мають задовольняти вимоги ACID (Atomicity, Consistency, Isolation, Durability – атомарність, несуперечність, ізольованість, довговічність), що гарантують правильність і надійність роботи системи.

### Атомарність

*Атомарність* передбачає таке:

- виконуються всі операції транзакції або жодна з них не виконується;
- якщо виконання транзакції було перерване, то всі зроблені транзакцією зміни мають бути скасовані

Дії, спрямовані на підтримку атомарності транзакції під час її аварійного завершення у зв'язку з помилками введення/виведення, перевантаженнями системи чи блокуваннями, називаються *відновленням транзакції*.

Дії, спрямовані на забезпечення атомарності під час виходу з ладу системи, називаються *відновленням у разі виникнення перебоїв*.

### Несуперечність

*Несуперечність* означає, що транзакція, яка працює з несуперечною базою даних, після завершення роботи залишає її також у несуперечному стані. Транзакція не повинна порушувати цілісності бази даних

Для вирішення проблем одночасного доступу інститут ANSI розробив спеціальний стандарт, який визначає чотири рівні блокування (кожний вищий рівень передбачає виконання умов усіх нижчих рівнів)

- **Рівень 0.** Заборона «забруднення» даних. На цьому рівні вимагається, щоб змінювати дані могла лише одна транзакція. Якщо іншій транзакції необхідно змінити ці ж дані, то вона має очікувати завершення першої транзакції.
- **Рівень 1.** Заборона некоректного зчитування. Якщо певна транзакція розпочала змінювати дані, то жодна інша транзакція не зможе зчитати ці дані доти, доки перша не завершиться.
- **Рівень 2.** Заборона неповторюваного зчитування. Якщо певна транзакція зчитує дані, то жодна інша транзакція не зможе їх змінити. Отже, під час повторного зчитування дані перебуватимуть у початковому стані.
- **Рівень 3.** Заборона «фантомів». Якщо транзакція звертається до даних, то жодна інша транзакція не зможе додати чи видалити рядки, що можуть бути зчитані під час виконання транзакції. Реалізація цього рівня блокування виконується блокуванням діапазону ключів. Це блокування накладається не на конкретні рядки таблиці, а на рядки, що відповідають певній логічній умові.

### Ізольованість

Властивість *ізольованості* означає, що на роботу транзакції не мають впливати інші транзакції. Транзакція «бачить» дані в тому стані, в якому вони перебували до початку роботи іншої транзакції, або в тому стані, в якому вони перебувають після її завершення. Одна транзакція не може переглядати проміжні стани даних, що використовуються іншими транзакціями. Якщо транзакція зчитує кілька разів ті самі дані, то вона повинна отримувати їх щоразу в тому стані, в якому вони були під час першого зчитування. Ще одним аспектом ізольованості є неможливість роботи з неповними результатами – незавершена транзакція не може передавати свої результати іншим транзакціям до підтвердження свого успішного завершення.

Наведемо приклад порушення ізольованості. Нехай задано дві транзакції:

```
T1: Read(x)      T2: Read(x)
    x ← x+1          x ← x+1
    Write(x)         Write(x)
    Commit           Commit
```

Розглянемо дві можливі послідовності їхнього виконання.

```
    T1:  Read(x)       T1.  Read(x)
    T1:  x ← x+1       T1:  x ← x+1
    T1:  Write(x)      T2:  Read(x)
    T1:  Commit        T1:  Write(x)
    T2:  Read(x)       T2:  x ← x+1
    T2:  x ← x+1       T2.  Write(x)
    T2:  Write(x)      T1:  Commit
    T2:  Commit        T2   Commit
Результат: x = x+2          x = x+1
```

### Довговічність

Після того, як було підтверджено успішне завершення роботи транзакції (Commit), система має гарантувати, що її результати не будуть втрачені, незважаючи на можливі перебої. Це й називається *довговічністю*. Для забезпечення довговічності застосовуються механізми відновлення бази даних.

## 10.6.2. Загальна схема роботи розподілених транзакцій

Розглянемо питання, що виникають під час обробки розподілених транзакцій.

- Якою є структура, або модель транзакцій? Розрізняють просту лінійну і вкладену моделі. Остання припускає вкладеність одних транзакцій в інші.
- Як підтримувати внутрішню несуперечність бази даних під час виконання розподілених транзакцій (якими є алгоритми підтримання обмежень цілісності)?
- Якою є реалізація протоколів надійності, локального відновлення та глобального підтвердження?
- Як синхронізувати виконання транзакцій, що працюють одночасно? (За це відповідають алгоритми керування одночасним доступом).
- Як керувати взаємною несуперечністю реплікованих даних? (За це відповідають протоколи керування реплікацією).

Несуперечність та ізольованість досягаються керуванням паралельним доступом, а атомарність і довговічність підтримуються засобами відновлення транзакцій.

Транзакція може здійснювати доступ до даних на різних вузлах. Кожен вузол містить *монітор розподіленої обробки*, що відповідає за всю обробку розподілених даних на вузлі, а також два компоненти, що стосуються транзакцій менеджер і планувальник транзакцій.

До функцій *менеджера транзакцій* належить:

- ведення журналу для відновлення бази даних у разі виникнення перебоїв;
- участь у координації транзакцій, що одночасно виконуються на його вузлі й інших вузлах.

Кожний з вузлів має свого *планувальника виконання транзакцій*, що складається з *координатора транзакцій* і *менеджера блокувань*. До його функцій належать:

- запуск транзакції, що ініціюється на його вузлі;
- блокування транзакцій;
- поширення підтранзакцій на вузли, де вони виконуватимуться;
- координація завершення транзакції, що ініціюється на вузлі планувальника, за умови, що всі підтранзакції завершені успішно (`Commit`) чи аварійно (`Abort`).

Взаємозв'язки розглянутих компонентів зображені на рис. 10.14.

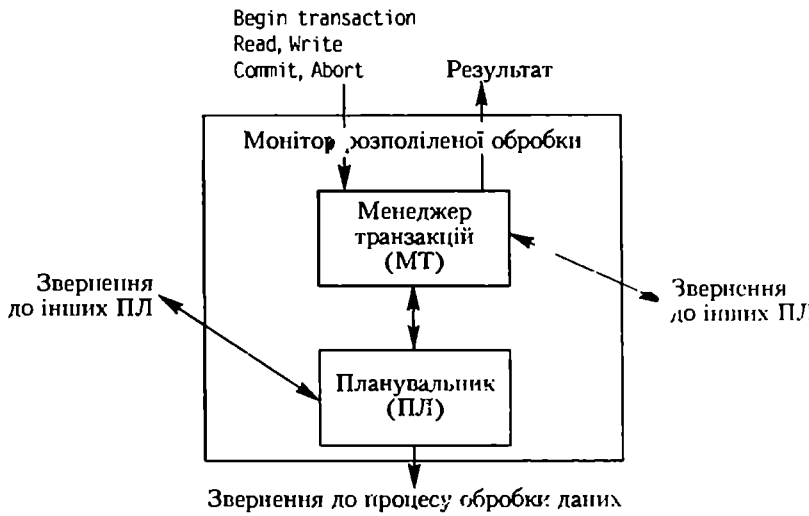



**Рис. 10.14.** Схема функціонування транзакції

## 10.6.3. Керування одночасним доступом. Блокування даних

У розподілених СКБД може функціонувати багато серверів зі своїми базами даних і виконуватися багато транзакцій водночас, тому виникає проблема керування одночасною роботою багатьох транзакцій, особливо в разі одночасного доступу до тих самих даних та їхнього відновлення. Проблема полягає в синхронізації

транзакцій, що одночасно виконуються, яка здійснюється з метою зберегти несуперечність бази і водночас досягти максимального паралелізму.

Виділяють два типи алгоритмів керування одночасним доступом:

- песимістичні,
- оптимістичні.

До песимістичного типу належать такі алгоритми.

- Алгоритми, що базуються на методах двофазного блокування (Two-Phase Locking – 2PL):
  - централізований (на первинному вузлі) 2PL;
  - розподілений 2PL.
- Алгоритми впорядкування за часовими мітками (Timestamp Ordering – TO):
  - базовий алгоритм упорядкування за часовими мітками;
  - багатоверсійні часові оцінки;
  - консервативні часові оцінки.
- Гібридні алгоритми.

До оптимістичного типу належать алгоритми, що базуються на:

- блокуванні;
- часових оцінках.

### Алгоритми двофазного блокування

Для отримання доступу до даних транзакції надсилають запити на іхнє блокування за допомогою менеджера блокувань. Блокування необхідні для керування доступом до даних, що використовуються в транзакціях.

Блокування може поширюватися на операцію зчитування (*блокування спільного доступу*) або на операцію запису (*монопольне блокування*). Блокування на зчитування і на запис, що здійснюються різними транзакціями, конфліктують між собою, оскільки ці операції не сумісні одна з одною. Сумісними є лише різні блокування на зчитування.

Загальний зміст алгоритму двофазного блокування є таким:

- перед зчитуванням необхідно здійснити блокування на зчитування, щоб заборонити блокування на запис;
- перед відновленням необхідно забезпечити блокування на запис;
- розблокування можна здійснювати тільки після виконання всіх блокувань.

### Алгоритм двофазного блокування

Опишемо алгоритм двофазного блокування:

- транзакція блокує об'єкт перед його використанням;
- якщо об'єкт заблокований іншою транзакцією, то транзакція, яка потребує блокування, переходить у стан очікування,
- коли транзакція завершує процес певного блокування, вона не може ініціювати інше блокування.

Схематично блокування цього типу зображене на рис. 10.15.

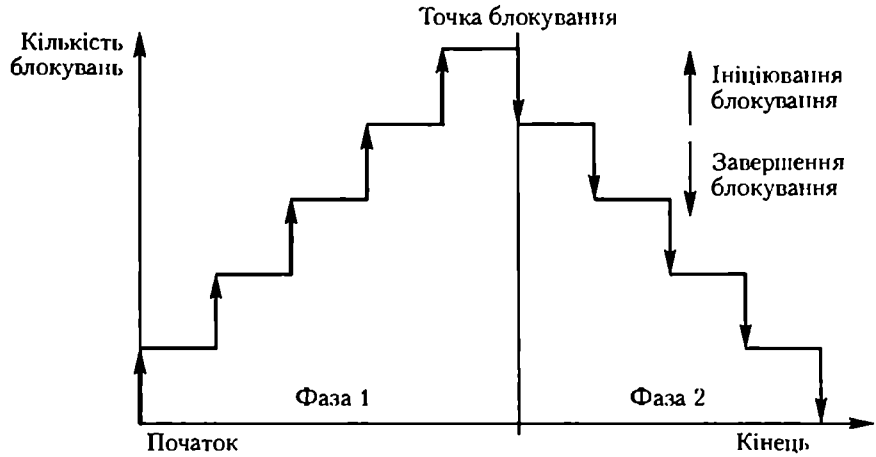



**Рис. 10.15.** Двофазне блокування

### Алгоритм строгого двофазного блокування

Різновидом алгоритму 2PL є *строгий алгоритм 2PL*. У цьому випадку, як показано на рис. 10.16, блокування об'єктів продовжується до завершення виконання транзакції.

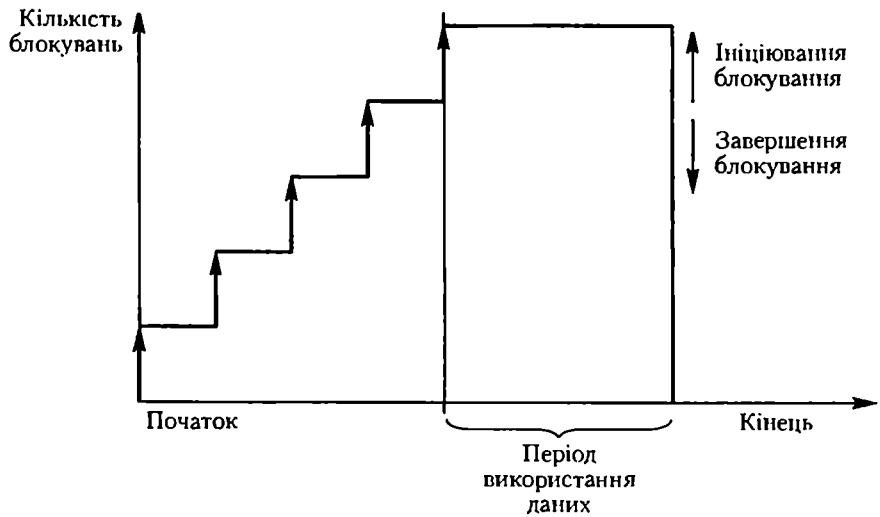



**Рис. 10.16.** Строге двофазне блокування

### Централізоване двофазне блокування

У даному випадку в усій розподіленій системі є лише один менеджер блокувань. За потреби блокування будь-якого об'єкта з будь-якого вузла слід звернутися до цього менеджера (рис. 10.17).


**Рис. 10.17.** Централізоване двофазне блокування

### Розподілене двофазне блокування

Кожний з вузлів розподіленої системи має свого менеджера блокувань, що керує запитами на блокування даних на його вузлі. Схематично розподілене 2PL зображене на рис. 10.18.

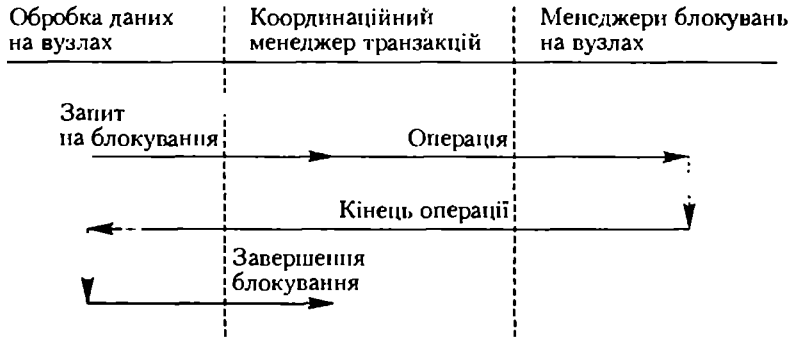



**Рис. 10.18.** Розподілене двофазне блокування

Зазначимо, що транзакція може зчитувати кожну з реплікованих копій необхідного їй об'єкта, одержуючи повідомлення про успішність блокування зчитування щодо будь-якої копії. Запис об'єкта до бази даних вимагає наявності блокування запису для всіх копій об'єкта.

### Алгоритми впорядкування транзакцій за часовими мітками

Альтернативний метод керування одночасним доступом грунтується на використанні часових міток. Основна ідея методу: якщо транзакція $T_1$ розпочалася раніше транзакції $T_2$, то система забезпечує такий режим виконання, ніби $T_1$ була повністю виконана до початку $T_2$. Для цього кожній транзакції $T$ надається часова мітка *ts*, що відповідає, наприклад, моменту початку $T$. Під час виконання операції над об'єктом $r$ транзакція $T$ позначає його своєю часовою міткою і типом операції

(зчитування чи змінення). Перед початком операції над об'єктом $r$ транзакція $T_1$ виконує такі дії.

- Перевіряє, чи не завершилася транзакція $T$, що позначила цей об'єкт. Якщо $T$ завершилася, $T_1$ позначає об'єкт $r$ своєю часовою міткою і виконує операцію.
- Якщо транзакція $T$ не завершилася, $T_1$ перевіряє конфліктність операцій. У випадку, коли операції неконфліктні, біля об'єкта $r$ ставиться часова мітка з меншим значенням, і транзакція $T_1$ виконує свою операцію.
- Якщо операції $T_1$ і $T$ конфліктують, а $ts(T) > ts(T_1)$ (тобто транзакція $T$ є «молодшою», ніж $T_1$), то виконується відкочування $T$, і $T_1$ продовжує роботу.
- Якщо операції $T_1$ і $T$ конфліктують, а $ts(T) < ts(T_1)$ ($T$ «старша» за $T_1$), то $T_1$ одержує нову часову мітку і розпочинається заново.

Серед недоліків методу часових міток можна назвати вищу ймовірність відкочування транзакцій, ніж у випадку використання двофазного блокування. Це пов'язане з тим, що конфліктність транзакцій визначається більш «грубо». Крім того, у розподілених системах не дуже легко створювати глобальні часові мітки так, щоб їх усі можна було порівняти.

Але в розподілених системах ці недоліки окупаються тим, що не потрібно розпізнавати тупики, а побудова графа очікування в таких системах коштує дуже дорого (про тупики й графи очікування йтиметься в підрозділі 10.6.4).

Опишемо більш точно механізм упорядкування транзакцій за часовими мітками

1. Транзакції $T_i$ приписується унікальна глобально визначена часова мітка $ts(T_i)$.
2. Під час виконання транзакції менеджер транзакцій присвоює часові мітки всім операціям, що ініціюються транзакцією. А саме, будь-якому елементу даних присвоюються такі часові мітки: $rts(x)$ – найбільша часова мітка серед усіх операцій зчитування елементу даних $x$ і $wts(x)$ – найбільша часова мітка серед усіх операцій запису $x$.
3. Конфліктні ситуації розв'язуються з урахуванням порядку часових міток.

Формально базовий алгоритм впорядкування за часовими мітками можна записати так:

```
for Ri(x)
if ts(Ti) < wts(x)
then reject Ri(x)
else
   begin
      accept Ri(x)
      rts(x) ← ts(Ti)
   end
for Wi(x)
if ts(Ti) < rts(x) and ts(Ti) < wts(x)
then reject Wi(x)
else
```

```
begin
  accept Wi(x)
  wts(x) ← ts(Ti)
end
```

Тут і далі символами $R_i(x)$ і $W_i(x)$ позначені довільні операції зчитування й запису, що входять до складу транзакції $T_i$.

### Алгоритм консервативного впорядкування за часовими мітками

Згідно з базовим алгоритмом операції виконуються тоді, коли вони надходять. Алгоритм консервативного впорядкування затримує кожну з операцій доти, доки не з'явиться впевненість, що її можна буде виконати. Затримки можуть призводити до тупиків.

### Алгоритм багатоверсійних часових оцінок

У цьому алгоритмі не здійснюється заміна значень у базі даних, але створюються нові значення. Алгоритм діє так.

- Операція $R_i(x)$ транслюється в операцію зчитування однієї з версій об'єкта $x$. Обрана версія об'єкта $x$ (скажімо, $x_m$) може визначатися, наприклад, тим, що її $ts(x_m)$ є найбільшою часовою міткою, меншою за $ts(T_i)$.
- Операція $W_i(x)$ транслюється в $W_i(x_s)$, де $x_s$ – версія об'єкта $x$, і виконується, якщо планувальник ще не обробив жодну таку $R_j(x_r)$, що $ts(T_i) < ts(x_r) < ts(T_j)$.

Використання часових міток виключає можливість виникнення тупикових ситуацій. У цьому випадку одночасне виконання транзакцій еквівалентне послідовному виконанню, що визначається часовими мітками. Але, хоча тупикові ситуації й виключаються, все ж таки можуть виникнути надлишкові повторні виконання і відмови.

### Оптимістичні алгоритми керування одночасним доступом

Відмінності між схемами песимістичного й оптимістичного керування одночасним доступом зображені на рис. 10.19. За песимістичного підходу передбачається можливість тупиків, тому перевірки виконуються якомога раніше, наприклад, коли надходить команда зчитування. За оптимістичного підходу всі необхідні перевірки відкладаються на якомога пізніший термін, наприклад вони можуть виконуватися перед командами запису.


**Рис. 10.19.** Відмінності між схемами керування одночасним доступом: песимістичне керування (*а*); оптимістичне керування (*б*)

Розподілені транзакції можна розглядати як набори підтранзакцій, кожна з яких виконується на власному вузлі. $T_{ij}$ позначає транзакцію $T_i$, що виконується на вузлі $j$; транзакції виконуються незалежно на кожному з вузлів доти, доки не будуть завершені їхні фази зчитування. Усім підтранзакціям по завершенні фаз зчитування присвоюються часові мітки. У фазі перевірки виконується описаний нижче тест. Якщо на одній із підтранзакцій тест не проходить, то здійснюється відкочування всієї транзакції.

Тест на допустимість виконання операції запису проводиться згідно з таким алгоритмом.

1. Якщо всі транзакції $T_k$, де $ts(T_k) < ts(T_{ij})$, завершують запис до початку фази зчитування $T_{ij}$ (рис. 10.20, *а*), результат тесту позитивний, тобто транзакції виконуються послідовно.
2. Якщо існує така транзакція $T_k$, що $ts(T_k) < ts(T_{ij})$, і її фаза запису завершується тоді, коли $T_{ij}$ перебуває у фазі зчитування (рис. 10.20, *б*), то результат тесту позитивний, якщо $WS(T_k) \cap RS(T_{ij}) = \varnothing$. Тут:
   - $RS(T)$ позначає множину об'єктів, що зчитуються транзакцією $T$,
   - $WS(T)$ позначає множину об'єктів, що записуються транзакцією $T$.

   Це означає, що фази запису і зчитування транзакцій $T_k$ і $T_{ij}$ перетинаються, але $T_{ij}$ не зчитує елементів даних, що записуються транзакцією $T_k$.
3. Якщо існує така транзакція $T_k$, що $ts(T_k) < ts(T_{ij})$, і її фаза запису завершується після початку фази запису $T_{ij}$ (рис. 10.20, *в*), то результат тесту позитивний, якщо $WS(T_k) \cap RS(T_{ij}) = \varnothing$ і $WS(T_k) \cap WS(T_{ij}) = \varnothing$. Це означає, що транзакції перетинаються, але не потребують доступу до спільних елементів даних.

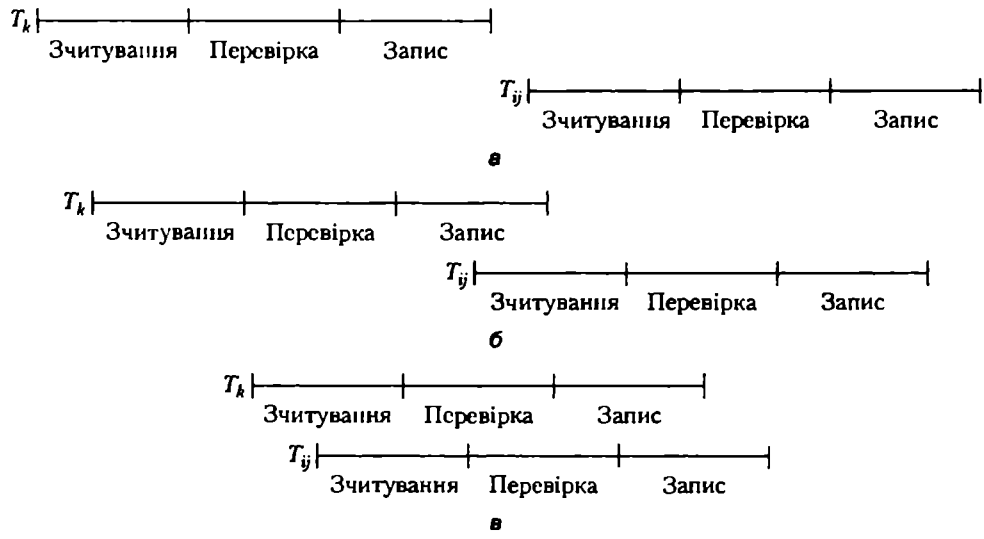



**Рис. 10.20.** Перевірка допустимості операцій запису: транзакції не перетинаються (*а*); фаза зчитування однієї транзакції перетинається з фазою запису іншої (*б*); фази запису різних транзакцій перетинаються (*в*)

## 10.6.4. Керування одночасним доступом. Нейтралізація тупиків

Транзакція буде в тупику, якщо її виконання заблоковане, і вивести її з такого стану можна лише завдяки зовнішньому втручанню. Тупикові ситуації виникають, як правило, в тому випадку, коли є транзакції, що працюють паралельно і потребують доступу до одних тих самих ресурсів. Наприклад, нехай транзакція $T_1$ заблокувала ресурс $P_1$, а транзакція $T_2$ – ресурс $P_2$. Якщо у якийсь момент часу транзакції $T_1$ знадобився ресурс $P_2$, то $T_1$ переходить у стан очікування звільнення ресурсу $P_2$. Якщо транзакція $T_2$, не звільнивши ресурс $P_2$, потребує доступу до ресурсу $P_1$, то вона також переходить у стан очікування звільнення ресурсу $P_1$. Виконання обох транзакцій блокується вони в *тупику*.

До тупиків можуть призводити алгоритми планування паралельної обробки, що базуються на механізмах блокування, а також алгоритми, що базуються на часових оцінках з очікуванням.

### Граф очікування

Ситуацію, коли одна транзакція може очікувати звільнення ресурсу, що використовується іншою транзакцією, можна зобразити у вигляді орієнтованого графа, вершинами якого є транзакції. Спрямований зв'язок між вершинами $T_1$ і $T_2$ у цьому графі існує в тому випадку, коли транзакція $T_1$ очікує звільнення ресурсу, захопленого транзакцією $T_2$. Очевидно, що дві транзакції є в тупику, якщо зв'язки між ними формують цикл. Більше того, у тупику можуть перебувати кілька транзакцій (рис. 10.21). Тут цикл містить три транзакції: $T_1$ очікує звільнення ресурсу, що використовується $T_2$, $T_2$ очікує звільнення ресурсу, захопленого $T_3$, а $T_3$, у свою чергу, – ресурсу, що використовується $T_1$.

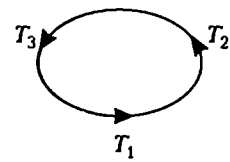



**Рис. 10.21.** Взаємне блокування трьох транзакцій

Зауважимо, що немає суттєвої залежності між виникненням тупикових ситуацій і тим, чи розташовані транзакції і блоковані ними дані в одному вузлі.

Можна назвати чотири механізми керування тупиками.

- **Ігнорування.** Це найпростіший спосіб вирішення проблеми. Все віддається «на відкуп» користувачу. Він перериває виконання однієї із заблокованих транзакцій (якщо така можливість надається системою) або перезавантажує систему.
- **Запобігання.** Гарантується, що тупики ніколи не виникнуть. У момент ініціювання транзакція перевіряється на можливість потрапляння в тупик. Якщо такі транзакції не виконуватимуться, ніяких засобів запобігання виникненню тупиків не буде потрібно.

- **Уникнення.** Під час роботи транзакції виявляються потенційні тупики і вживаються заходи для їхнього уникнення.
- **Виявлення й нейтралізація.** Тупики допускаються, однак існують механізми їхнього виявлення і нейтралізації.

Три останні механізми розглянемо детальніше

### Запобігання тупикам

Єдиний можливий варіант вирішення проблеми тупиків для даного випадку – попереднє оголошення (чи виявлення) усіх необхідних для транзакції ресурсів. Це має бути зроблено за ініціювання транзакції. Система має гарантувати, що в жодних інших ресурсах, окрім виявлених під час ініціювання транзакції, потреби не виникатиме. Всі необхідні ресурси попередньо резервуються. Транзакція запускається на виконання лише в тому випадку, якщо всі потрібні ресурси доступні для резервування. Як правило, коли ресурсом є інформація з бази даних, цей варіант складно реалізувати.

Даний механізм слід використовувати, коли система не має засобів відкочування транзакцій. Серед недоліків можна назвати значне зниження ефекту паралелізму та складність визначення необхідних ресурсів. Переваги полягають у тому, що попереднє оголошення не вимагає механізмів відкочування і повторного запуску транзакцій.

### Уникнення тупиків

У цьому випадку немає потреби виявляти всі необхідні ресурси та попередньо їх резервувати. Транзакція працює без жодних обмежень доти, доки не став потрібен зайнятий ресурс. Якщо така ситуація склалася, транзакція переходить у стан очікування звільнення ресурсу. Зазвичай фіксується часовий інтервал можливого очікування.

Під час роботи в середовищі баз даних цей механізм більш привабливий за попередній.

Розглянемо два алгоритми уникнення тупиків.

#### Алгоритм без витіснення (Wait-Die)

**Wait-Die правило:** якщо транзакція $T_i$ потребує блокування елементу даних, що вже заблокований транзакцією $T_j$, то $T_i$ переходить у стан очікування, якщо $ts(T_i) < ts(T_j)$. Якщо ж $ts(T_i) > ts(T_j)$, то $T_i$ завершується і запускається знову з тією ж часовою міткою.

$$\textbf{if } ts(T_i) < ts(T_j) \textbf{ then } T_i \text{ waits } \textbf{else } T_i \text{ diesS}.$$

В алгоритмі без витіснення $T_i$ ніколи не витісняє $T_j$.

#### Алгоритм із витісненням (Wound-Wait)

**Wound-Wait правило:** якщо транзакція $T_i$ потребує блокування елементу даних, що вже заблокований транзакцією $T_j$, то $T_i$ переходить у стан очікування, якщо $ts(T_i) > ts(T_j)$. Якщо $ts(T_i) < ts(T_j)$, то $T_j$ завершується, а $T_i$ одержує право заблокувати елемент даних.

$$\text{if } ts(T_i) < ts(T_j) \text{ then } T_j \text{ is wounded else } T_i \text{ waits.}$$

У цьому алгоритмі $T_i$ витісняє $T_j$ за певних умов.

### Виявлення й нейтралізація тупиків

Якщо потрібно отримати доступ до тих самих елементів даних, транзакції не відкочуються, їм дозволяється переходити в стан очікування Необхідно періодично аналізувати, що очікують транзакції, і виявляти тупикові ситуації. Основний механізм пошуку тупиків – аналіз графа очікувань (Wait For Graph – WFG) і виявлення в ньому циклів.

Алгоритми виявлення тупиків бувають трьох типів:

- централізовані;
- розподілені;
- ієрархічні.

#### Централізоване виявлення тупиків

Один із вузлів мережі виділяється як єдиний детектор тупиків (Deadlock Detector – DD). Кожний із менеджерів блокувань у вузлах надсилає свій WFG у центральний вузол, де всі WFG зливаються в єдиний глобальний WFG, який аналізується на наявність циклів.

#### Ієрархічне виявлення тупиків

Є декілька детекторів тупиків, що поєднуються в ієрархічну схему. Кожний із детекторів співпрацює з детектором, розташованим у батьківському вузлі. Якщо тупик не виявляється на локальному вузлі, то його WFG пересилається на вищий рівень.

#### Розподілене виявлення тупиків

Усі вузли мають свої детектори тупиків й усі вони можуть кооперуватися один з одним. Опишемо одну зі схем розподіленого виявлення тупиків.

- Локальні WFG формуються на кожному з вузлів і за необхідності пересилаються на будь-який інший вузол. Кожен локальний WFG модифікується і пересилається в описаний далі спосіб.
- Оскільки кожний із вузлів одержує можливі цикли з інших вузлів, то отримані графи з'єднуються з локальним WFG. Ребра локального WFG, які вказують, що локальні транзакції очікують транзакції на інших вузлах, поєднуються з тими ребрами з віддалених WFG, які свідчать, що віддалені транзакції також перебувають у стані очікування.

Кожен локальний детектор тупиків:

- відшукує цикли, що не містять віддалених ребер (проблема тупиків для таких циклів вирішується локально);
- відшукує цикли, до складу яких входять віддалені ребра (для таких циклів проблема вирішується спільно з відповідними вузлами)

### 10.6.5. Керування одночасним доступом. Відновлення після перебоїв

Щоб відновити базу даних після перебою, необхідно, аби кожна дія транзакції над базою даних була зафіксована в *журналі транзакцій*.


**Рис. 10.22.** Реєстрація змін у журналі транзакцій

У журналі записується інформація, необхідна для відновлення несуперечності системи у випадку виникнення перебоїв. Ця інформація може містити:

- ідентифікатор транзакції;
- тип операції;
- елементи, до яких був здійснений доступ,
- попереднє значення (стан) елементу даних;
- нове значення (стан) елементу даних.

Якщо стався перебій у ситуації, що зображена на рис. 10.23, то загальний механізм відновлення має бути таким:

- усі дії, виконані транзакцією $T_1$, мають бути відновлені (REDO), якщо в цьому є необхідність ($T_1$ – це транзакція, яка була завершена, але результати її дій не були зафіксовані в постійній базі даних);
- усі дії, які було виконано незавершеною транзакцією $T_2$, мають бути скасовані (UNDO).


**Рис. 10.23.** Виконання транзакцій до і під час перебою

#### Протокол відновлення

Відновлення (REDO) здійснених раніше операцій означає їхнє повторне виконання. Під час відновлення використовується інформація з журналу і виконуються операції, що були відмінені у зв'язку з перебоєм в системі

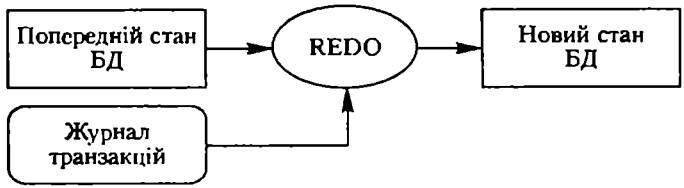



**Рис. 10.24.** Протокол відновлення (REDO)

### Протокол скасування/відкочування

Відкочування – це відновлення того стану об'єкта, який він мав до виконання транзакції. На основі нового стану бази даних й інформації з журналу відновлюються попередні значення.

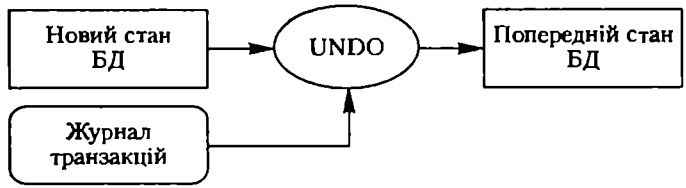



**Рис. 10.25.** Протокол відкочування (UNDO)

### Журналізація і буферизація

Журналізація змін тісно пов'язана не лише з керуванням транзакціями, але й з буферизацією сторінок бази даних в оперативній пам'яті. Оскільки відмінність між швидкістю роботи процесорів і оперативної пам'яті та пристроїв зовнішньої пам'яті існувала, існує й буде існувати завжди, буферизація сторінок бази даних в оперативній пам'яті – єдиний реальний спосіб досягнення високої швидкодії СКБД.

Якби запис про зміну стану бази даних, що мав надійти до журналу за її модифікації, негайно записувався у зовнішню пам'ять, це дуже сповільнило б роботу системи. Тому записи в журнал теж буферизуються: під час нормальної роботи чергова сторінка записується в зовнішній пам'яті журналу лише за умови його переповнення.

Є два види буферів: журналу та сторінок оперативної пам'яті, що містять інформацію з бази даних. Вміст і тих, й інших буферів може копіюватися в зовнішню пам'ять. Проблема полягає у виробленні загальної політики копіювання, що забезпечувала б можливості відновлення стану бази даних після перебоїв.

Проблема не виникає під час відкочування окремих транзакцій, оскільки в цих випадках вміст оперативної пам'яті не втрачається і можна скористатися як буфером журналу, так і буферами сторінок оперативної пам'яті. Але якщо виник перебій і вміст буферів утрачено, для відновлення бази даних необхідно, щоб стан журналу і бази даних у зовнішній пам'яті був узгодженим.

Основний принцип узгодженої політики копіювання буфера журналу і буферів сторінок оперативної пам'яті у зовнішню пам'ять полягає в тому, що запис про зміну об'єкта бази даних має потрапляти до зовнішньої пам'яті журналу раніше, ніж змінений об'єкт буде записано до зовнішньої пам'яті бази даних. Відповідний

протокол журналізації (і керування буферизацією) називається протоколом WAL (Write Ahead Log – випереджальний запис до журналу).

### Протокол WAL

Припустимо, що транзакція *T* змінила сторінку *P* в оперативній пам'яті, після чого події відбувалися в такому порядку.

1. Сторінка *P* була записана до постійної пам'яті.
2. Відповідний журнальний запис було збережено в зовнішній пам'яті.
3. До завершення транзакції у системі відбувся перебій.

Ми можемо здійснити відкочування транзакції (UNDO), відновивши попередній стан сторінки *P* за допомогою збереженого запису про транзакцію.

Тепер припустимо, що послідовність подій була такою.

1. Сторінка *P* була записана до постійної пам'яті.
2. До збереження відповідного журнального запису в постійній пам'яті у системі відбувся перебій.

У цьому випадку немає можливості здійснити відкочування транзакції, оскільки в журналі немає відповідного запису.

Уникнути таких ситуацій дає можливість протокол WAL.

1. Перед початком транзакції записати в зовнішню пам'ять UNDO-інформацію з журналу.
2. Після успішного завершення транзакції записати в зовнішню пам'ять REDO-інформацію з журналу.
3. Оновити постійні дані

Якщо система виходить з ладу до того, як транзакції успішно завершуються, то всі операції мають бути скасовані (UNDO). Якщо транзакція завершилася, але перед записом її результатів до постійної пам'яті стався перебій, то її дії слід повторити (REDO), для цього використовуються нові значення даних з журналу.

Загальна схема ведення журналу зображена на рис. 10.26.

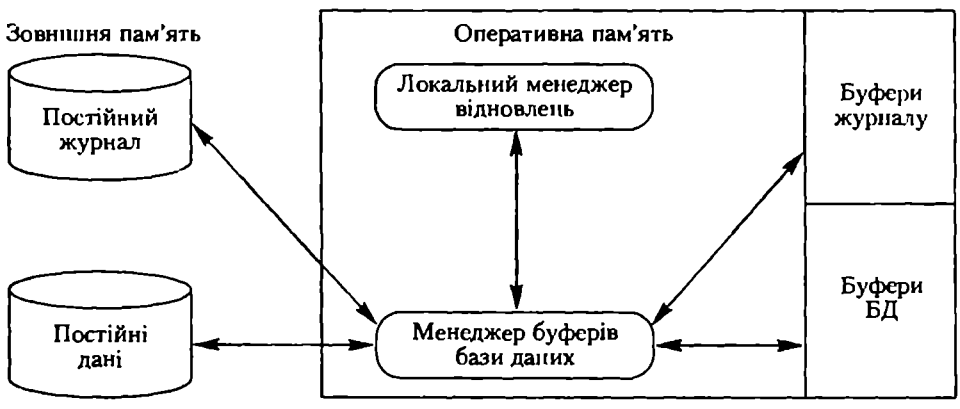



**Рис. 10.26.** Схема ведення журналу

Протокол WAL гарантує, що коли в зовнішній пам'яті бази даних модифіковано певний об'єкт, то в зовнішній пам'яті журналу обов'язково є запис, що відповідає цій операції. Але якщо в зовнішній пам'яті журналу міститься запис про зміну об'єкта бази даних, то сам змінений об'єкт може бути відсутнім у зовнішній пам'яті бази даних.

Додаткова вимога до копіювання буферів полягає в тому, що кожна успішно завершена транзакція має бути реально зафіксована в зовнішній пам'яті. Який би перебій не стався, система має бути спроможною відновити стан бази даних з результатами всіх зафіксованих до моменту перебою транзакцій.

Простим рішенням було б копіювання буфера журналу до зовнішньої пам'яті, після якого здійснюватиметься масове копіювання буферів сторінок бази даних, що змінювалися даною транзакцією. Досить часто так і роблять, але це призводить до значних додаткових витрат під час фіксації транзакції

Мінімальною вимогою, що гарантує можливість відновлення останнього несуперечного стану бази даних, є копіювання до зовнішньої пам'яті під час фіксації транзакції всіх журнальних записів про зміну бази даних цією транзакцією. Останнім записом у журналі, виконаним від імені транзакції, є запис про її завершення.

Розглянемо, як можна виконувати відновлення бази даних у різних ситуаціях, якщо в системі згідно з протоколом WAL підтримується спільний для всіх транзакцій журнал зі спільною буферизацією записів.

### Відновлення після перебоїв

Одна з основних проблем, пов'язаних із відновленням після перебоїв полягає в тому, що кожна логічна операція зміни бази даних може змінювати кілька її фізичних блоків, наприклад сторінку даних і кілька сторінок індексів. Сторінки бази даних буферизуються в оперативній пам'яті й копіюються до зовнішньої пам'яті незалежно. Незважаючи на застосування протоколу WAL, після перебою набір сторінок у зовнішній пам'яті бази даних може виявитися неузгодженим, тобто частина сторінок у зовнішній пам'яті відповідатиме об'єкту до зміни, частина – після зміни. До такого стану об'єкта не застосовні операції логічного рівня.

Стан зовнішньої пам'яті бази даних називається *фізично узгодженим*, якщо набори сторінок усіх об'єктів узгоджені, тобто відповідають стану об'єкта після зміни або до зміни.

Вважатимемо, що в журналі вказуються точки фізичної узгодженості бази даних – моменти часу, в які у зовнішній пам'яті містяться узгоджені результати операцій, що вже завершилися, й відсутні результати операцій, які ще не завершилися. В ці моменти буфер журналу має бути записаний у зовнішню пам'ять. Такі точки позначаються абревіатурою tpc (Time of Physical Consistency)

На момент м'якого перебою транзакції можуть перебувати у станах, що зображені на рис. 10.27.

Для відновлення узгодженого на момент tpc стану зовнішньої пам'яті необхідно здійснити таку обробку транзакцій.

- Для транзакції $T_1$ жодних дій виконувати не потрібно. Вона завершилася до моменту tpc, і всі її результати відображені в зовнішній пам'яті бази даних.

- Для транзакції $T_2$ потрібно повторно виконати ті операції, що залишилися. У зовнішній пам'яті не відображені операції, що виконувалися транзакцією $T_2$ після моменту tpc. Отже, повторне виконання операцій $T_2$ коректне й приведе до логічно узгодженого стану бази даних (оскільки транзакція $T_2$ успішно завершилася до моменту перебою, у журналі містяться записи про всі зміни, здійснені цією транзакцією).
- Для транзакції $T_3$ потрібно виконати у зворотному напрямку першу частину операцій. У зовнішній пам'яті бази даних повністю відсутні результати операцій $T_3$, що були виконані після моменту tpc. З іншого боку, в зовнішній пам'яті зафіксовані результати операцій $T_3$, що були виконані до моменту tpc. Отже, зворотна інтерпретація операцій $T_3$ коректна й приведе до узгодженого стану бази даних (оскільки транзакція $T_3$ не завершилася до моменту м'якого перебою, під час відновлення необхідно усунути всі наслідки її виконання).
- Для транзакції $T_4$, що встигла початися після моменту tpc і завершитися до моменту перебою, потрібно повторити виконання операцій у прямому напрямі.
- Нарешті, для $T_5$, що почалася після моменту tpc і не встигла завершитися до моменту перебою, жодних дій виконувати не потрібно Результати операцій цієї транзакції не відображені в зовнішній пам'яті бази даних.

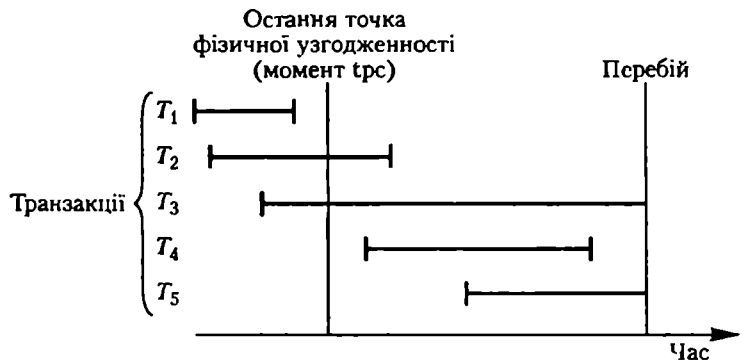



**Рис. 10.27.** Варіанти станів транзакцій на момент перебою

## 10.6.6. Керування одночасним доступом. Надійність обробки розподілених транзакцій

Для підвищення надійності функціонування розподілених транзакцій слід вирішити три проблеми.

- **Підтвердження завершення виконання транзакцій.** Питання полягає в тому, як виконати команду COMMIT для розподілених транзакцій, коли обчислювальні процедури транзакції розподілені за вузлами мережі. Основна проблема – забезпечення атомарності й довговічності виконання транзакції в цілому.
- **Припинення роботи розподіленої транзакції.** Проблема полягає в тому, як реагуватимуть на перебій різні вузли, що обробляють розподілену транзакцію.

Наявність перебоїв не має змушувати вузли, де стався перебій, очікувати відновлення з метою завершення транзакції.

- **Відновлення.** Під час незалежного відновлення вузол, на якому стався перебій, може самостійно подолати його наслідки без одержання додаткової інформації ззовні. Як правило, незалежне відновлення припускає припинення транзакції без блокування.

### Протоколи підтвердження завершення транзакції

Протокол підтвердження завершення транзакції використовується для гарантування атомарності й довговічності розподіленої транзакції. Тобто транзакція, що виконується на багатьох вузлах, має бути успішно завершена на всіх вузлах або анульована на всіх вузлах. Не допускається, щоб транзакція була успішно завершена на одних вузлах і аварійно на інших.

Відомі два протоколи завершення роботи розподілених транзакцій:

- двофазного підтвердження (Two-Phase Commit – 2PC) найбільш розповсюджений;
- трифазного підтвердження (Three-Phase Commit – 3PC) – складніший і дорожчий у реалізації, але позбавлений недоліків протоколу 2PC.

### Протокол двофазного підтвердження завершення транзакції

**Фаза 1.** Одержання даних для ухвалення рішення.

1. Координатор звертається до всіх учасників з вимогою підготуватися до завершення їхніх частин загальної транзакції з метою запису результатів у базу даних:
   - координатор звертається до всіх учасників з запитом підготуватися (PREPARE) до завершення транзакції;
   - координатор додає запис begin_commit у журнал, ініціює запис журналу в постійну пам'ять і переходить у стан очікування відповідей.
2. При одержанні повідомлення від координатора менеджер транзакцій учасника визначає, чи може він завершити (COMMIT) транзакцію:
   - якщо ні, то записує abort до свого журналу, відсилає повідомлення ABORT координатору і переходить до аварійного завершення транзакції;
   - якщо транзакція може бути завершена, то додає запис ready до журналу, записує всі дії транзакції до постійної пам'яті, відсилає повідомлення COMMIT координатору і переходить у стан READY.

**Фаза 2.** Ухвалення рішення на підставі отриманих даних.

1. На підставі прийнятих від усіх учасників повідомлень координатор вирішує долю транзакції. Якщо всі учасники надіслали повідомлення COMMIT, координатор додає до журналу запис commit, якщо ж хоча б один учасник надіслав повідомлення ABORT, координатор записує до журналу abort. У будь-якому випадку цей запис журналу відображується в постійній пам'яті. Будучи записаним у постійну пам'ять, він не підлягає скасуванню (навіть якщо відбудеться перебій).
2. Координатор надсилає повідомлення всім учасникам для інформування про прийняте рішення (COMMIT чи GLOBAL ABORT)

в) Учасники приймають відповідні рішення локально. Після запису до журналу сигналу abort або commit учасники надсилають оповіщення (ACK) координатору.

г) Отримавши оповіщення від усіх учасників, координатор записує до журналу повідомлення про завершення загальної транзакції.

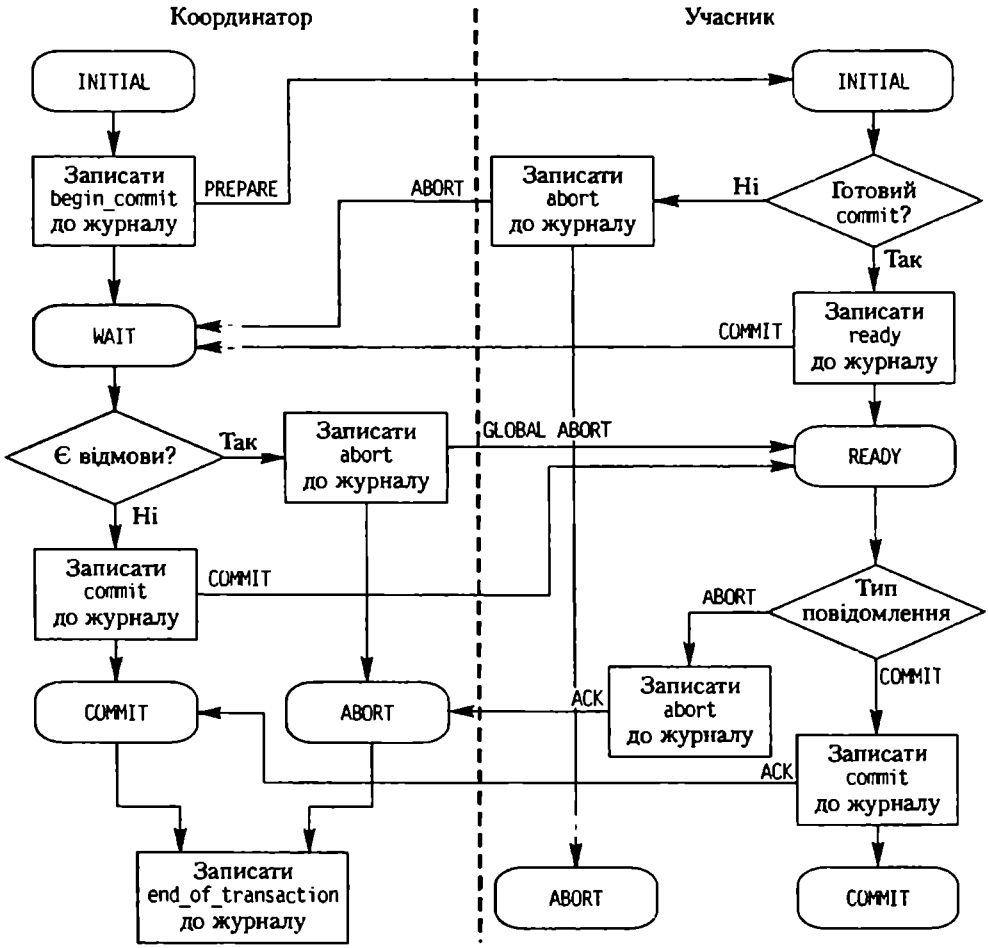



**Рис. 10.28.** Схема функціонування протоколу 2PC

### Протокол трифазного підтвердження завершення транзакції

Протокол 2PC має такі недоліки.

- Припускається блокування. Перехід у стан READY передбачає, що учасники очікують повідомлення координатора. Якщо у координатора стався перебій, вузли будуть заблоковані до завершення процедури відновлення. Зазначимо, що блокування зменшують доступність вузлів.
- Незалежне відновлення є неможливим

Для усунення цих недоліків був розроблений протокол трифазного підтвердження, в якому перша фаза аналогічна 2PC, а друга поділяється на дві: фазу 2 і фазу 3. У фазі 2 координатор приймає рішення, як і в 2PC, і повідомляє про це рішення учасникам. Але у випадку позитивного рішення (тобто коли всі учасники на першій фазі відповіли COMMIT) координатор надсилає повідомлення PRE-COMMIT. Учасники, які відповіли на першій фазі COMMIT, очікують від координатора повідомлення PRE_COMMIT або ABORT. Якщо вони отримали PRE_COMMIT, то відповідають оповіщенням ACK На фазі 3 координатор збирає сигнали ACK від усіх учасників і приймає рішення про завершення транзакції. Для цього він надсилає повідомлення COMMIT усім учасникам і завершує свою роботу. Учасники, після отримання COMMIT від координатора, завершують свої транзакції.

У протоколі 3PC знання про попереднє рішення щодо завершення може використовуватися для завершення транзакцій учасників незалежно від можливого перебою у вузлі координатора.

## Контрольні запитання та завдання

1. Що таке однорідні й неоднорідні розподілені бази даних?
2. Чи відповідає логічна архітектура розподілених баз даних архітектурі ANSI/SPARC?
3. Що таке розподіленість, неоднорідність й автономність баз даних?
4. Які механізми розподіленого зберігання даних ви знаєте?
5. Які різновиди фрагментації баз даних ви знаєте?
6. Що таке реплікація? Які існують механізми й моделі реплікації?
7. Опишіть алгоритми розподіленого виконання операції з'єднання відношень.
8. Які є способи обчислення запитів на фрагментованих відношеннях?
9. Що таке правила ACID виконання транзакцій?
10. Які алгоритми керування одночасним доступом ви знаєте?
11. У чому суть алгоритмів впорядкування транзакцій за часовими мітками?
12. Що таке оптимістичні алгоритми керування одночасним доступом?
13. Що називається тупиком? Які механізми керування тупиками ви знаєте?
14. Які існують механізми відновлення бази даних після перебоїв?
15. Розкажіть про протокол двофазного підтвердження завершення транзакції.

# Розділ 11

# Паралельні бази даних

- Архітектура багатопроцесорних систем
- Розподіл даних
- Паралельна обробка даних

## 11.1. Основні поняття паралельної обробки даних

Під час паралельної обробки даних велика задача поділяється на кілька менших, які одночасно виконуються на кількох вузлах комп'ютерної системи. Відтак вихідна задача вирішується значно швидше.

Ефективність реалізації паралельної обробки даних залежить:

- від способу поділу задачі на підзадачі, які можна виконувати водночас (паралельно);
- від можливості виявлення підзадач, які мають виконуватися послідовно, та підтримки послідовної схеми їхнього виконання.

Система паралельної обробки даних має такі властивості:

- кожний вузол системи може виконувати задачі одночасно з іншими вузлами;
- система може синхронізувати виконання задач (підзадач);
- вузли системи спільно використовують ресурси, тобто дані, а також диски та інші пристрої.

До ключових аспектів паралельної обробки даних належать наступні.

- **Прискорення обчислень та масштабованість** – це два основні критерії ефективності паралельної обробки даних. Обчислювальні задачі в паралельний спосіб мають виконуватись швидше, ніж у послідовний. Отже, можна казати про прискорення обчислень завдяки паралелізму. Терміном «масштабованість» позначають наявність легкого способу підвищувати потужність системи так, щоб вона була здатна обробити зростаючі обсяги даних. Рівень масштабованості можна виміряти за допомогою *коефіцієнту масштабування*, що показує, у скільки разів прискорення обчислень перевищує збільшення апаратних витрат.
- **Синхронізація** – це координація обробки задач, що виконуються одночасно, вона необхідна для отримання коректного результату обчислень. Запорукою успішної паралельної обробки є такий поділ на підзадачі, коли потрібна незначна синхронізація. Внаслідок зменшення синхронізації зростає швидкодія

та збільшується коефіцієнт масштабування. Потреба в синхронізації залежить від можливості спільного використання ресурсів, обсягів і кількості задач.

- **Блокування** – це механізм керування доступом до ресурсів, за допомогою якого досягається синхронізація виконання задач.
- **Передавання повідомлень** – це один із ключових аспектів паралельної обробки даних, оскільки вона виконується ефективно лише в тому випадку, коли вузли системи з'єднані високопродуктивними каналами зв'язку.

Технологія паралельних баз даних надає низку переваг прикладним системам, які їх використовують.

- **Висока продуктивність.** Що більше вузлів, то швидше виконуються задачі.
- **Підвищена працездатність.** За відмови одного з вузлів його функції може взяти на себе інший, і система залишиться працездатною. Отже, порівняно з однопроцесорною системою, дані будуть більш доступними.
- **Підвищена гнучкість.** Можна настроювати конфігурацію технічних засобів відповідно до потреб прикладних задач, нарощувати обчислювальні потужності системи або, навпаки, зменшувати їх
- **Збільшення кількості користувачів.** Технологія паралельних баз даних надає можливість подолати наслідки обмеженості ресурсів пам'яті, що дозволяє суттєво збільшити кількість користувачів, які обслуговуються водночас.

Дані можуть бути розподілені за різними дисками з метою розпаралелювання операцій введення/виведення, що виконуються різними процесорами. Окремі операції (наприклад, сортування чи з'єднання) можна виконувати в паралельному режимі. Високорівневість мови, на якій формулюються запити (наприклад, мови SQL), також сприяє розпаралелюванню її виразів.

## 11.2. Архітектура багатопроцесорних систем

Існує кілька типів архітектури багатопроцесорних систем. Ця архітектура визначає спосіб взаємодії між процесором, оперативною пам'яттю і дисковим простором. Найбільш загальними її різновидами є архітектури:

- зі спільною пам'яттю;
- зі спільною зовнішньою пам'яттю;
- без спільної пам'яті.

Коротко опишемо ці типи архітектури

### Архітектура зі спільною пам'яттю

Багато процесорів мають спільну оперативну й дискову пам'ять, до якої вони здійснюють спільний доступ (рис. 11.1). На цьому рисунку символами $P_1$, ..., $P_n$ позначені процесори, літерою $M$ – оперативна пам'ять, літерою $D$ – диски (які називають також *розділами*)

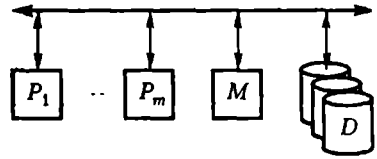



**Рис. 11.1.** Архітектура зі спільною пам'яттю

Основною перевагою архітектури зі спільною пам'яттю є те, що повідомлення та дані не передаються комунікаційними каналами. Процесори обмінюються даними через спільну пам'ять. Це також полегшує синхронізацію процесів. Ще однією перевагою такої архітектури є те, що нерівномірне розподілення навантаження процесорів можна усунути з мінімальними додатковими витратами. Недоліки таких систем пов'язані з проблемами спільного використання ресурсів. Лише за незначної кількості процесорів кожен з них може отримати доступ до бажаної ділянки пам'яті.

### Архітектура без спільної пам'яті

Ця архітектура протилежна попередній. Кожен процесор має свою власну оперативну й дискову пам'ять (рис. 11.2).

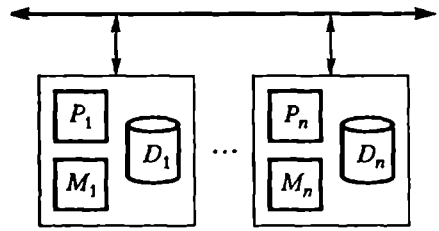



**Рис. 11.2.** Архітектура без спільної пам'яті

В архітектурі без спільної пам'яті процесори передають один одному повідомлення та дані через комунікаційну мережу. Основна перевага подібної архітектури полягає в можливості масштабування до сотень і тисяч процесорів без втручання кожного з них у роботу всіх інших.

Недоліком таких систем є нерівномірне розподілення навантаження процесорів та їхня залежність від пропускної здатності комунікаційної мережі. Якщо дані розподілені асиметрично, то паралельне виконання операцій сортування та з'єднання є малоефективним, оскільки залежить від способу координації обміну даними між вузлами. Ще один недолік подібної архітектури пов'язаний з ситуацією, коли виходить з ладу один із процесорів. адже дані, якими цей процесор володіє, стають недоступними.

Архітектура без спільної пам'яті може бути побудована з використанням комп'ютерів, з'єднаних комунікаційною мережею. У цьому випадку така архітектура є найдешевшим варіантом реалізації паралельних систем баз даних. У більшості досліджень у галузі паралельних баз даних за основу береться саме такий варіант архітектури.

### Архітектура зі спільною зовнішньою пам'яттю

У цій архітектурі (рис 11.3) кожний процесор має власну оперативну пам'ять, але всі вони працюють зі спільним дисковим простором.

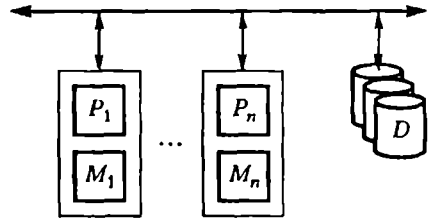



**Рис. 11.3.** Архітектура зі спільною зовнішньою пам'яттю

У деяких випадках за допомогою архітектури зі спільною зовнішньою пам'яттю можна вирішувати проблеми, пов'язані з недоліками архітектури інших типів. Наприклад, у цій архітектурі можна вирішити проблему відмови вузла, що виникає в системах без спільної пам'яті, оскільки диски, на яких зберігаються дані, доступні всім процесорам.

## 11.3. Розподіл даних

Розподіл даних бази за багатьма дисками здійснюється для того, щоб дати можливість багатьом процесорам отримувати одночасний доступ до даних, розміщених на зовнішніх носіях. Цим забезпечується паралелізм під час виконання операцій введення/виведення.

Відомі два варіанти розміщення даних бази на багатьох дисках:

- розміщення на різних дисках відношень;
- розміщення на різних дисках кортежів одного відношення, або горизонтальний поділ відношень.

Як розміщувати на різних дисках цілі відношення, є цілком очевидним, тому далі розглянемо кілька методів розміщення на різних дисках кортежів одного відношення.

### 11.3.1. Методи розподілу кортежів відношення

Розглянемо основні методи розподілу кортежів відношення за різними дисками (припустимо, що є $n$ дисків). Опишемо їхні переваги й недоліки, що виявляються під час обчислення таких запитів.

- Послідовне сканування всього відношення.
- Пошук кортежа за значенням атрибута — *точковий запит*. Наприклад, $RA = 26$.
- Пошук усіх кортежів, для яких значення заданого атрибута належить вказаному інтервалу — *запит за діапазоном*. Наприклад, $10 < RA < 25$.

### Циклічний розподіл

Кортежі відношення розподіляються за дисками циклічно: *i*-й кортеж потрапляє на диск з номером $i \bmod n$ Цей метод найкраще застосовувати для повного перегляду всього відношення. Диски міститимуть приблизно однакову кількість кортежів, а отже, навантаження, пов'язане з доступом до даних, розподілятиметься за всіма дисками рівномірно.

Недоліки такого методу полягають у тому, що складно обробляти запити за діапазоном, відсутня кластеризація (групування) даних і кортежі розкидані по всіх розділах.

### Хешування

Вибирається один чи кілька атрибутів, значення яких використовуються як значення функції хешування. Функція хешування має набувати значень в діапазоні від 1 до *n*. Якщо ця функція на кортежі набуває значення *j*, то він розміщується на диску *j*.

Якщо припустити, що хешування здійснюється за ключами і функція хешування вдало підібрана, то кортежі будуть рівномірно розподілені за дисками. Тому під час пошуку кортежів навантаження між дисками теж буде розподілятися рівномірно

Для точкових запитів, коли умова задається на атрибуті хешування, подібний метод може бути досить ефективним, оскільки він надає можливість переглянути лише один диск (усі інші диски в цьому випадку залишаться вільними для виконання інших запитів). Окрім того, індекс за атрибутом хешування може бути локальним стосовно диска, що дає змогу більш ефективно виконувати пошук і відновлення.

Недолік методу полягає у відсутності кластеризації, тому за такого розподілу даних важко здійснювати пошук за діапазоном значень.

### Ранжування

У цьому методі обирається *атрибут* чи *атрибути ранжування*. Вся множина допустимих значень атрибутів ранжування упорядковується й поділяється на *n* груп (тим самим створюється *вектор ранжування*). Кортежі *i*-ї групи потрапляють на *i*-й диск. Наприклад, якщо значеннями атрибута ранжування є літери алфавіту і записи мають бути розміщені на 4 дисках, то можна означити такий вектор ранжування: *А–Е, Ж–Л, М–С, Т–Я*.

Використання цього методу гарантує кластеризацію даних шляхом групування за значеннями певних атрибутів.

Метод може застосовуватися для послідовного доступу та виконання точкових запитів за атрибутом ранжування. Він також буде ефективним для запитів за діапазоном, коли умова визначається на атрибутах ранжування, оскільки зменшується кількість дисків, до яких необхідно отримати доступ.

Недолік методу полягає в можливості нерівномірного розподілу даних на дисках. Наприклад, може виникнути така ситуація, коли диски, що містять кортежі зі значеннями вектора ранжування *Т–Я*, будуть майже порожні.

### 11.3.2. Проблема нерівномірного розподілу даних

Бажано, щоб кортежі відношення розподілялися за розділами рівномірно, оскільки саме в такий спосіб досягатиметься рівномірний розподіл навантаження на диски під час виконання операцій введення/виведення. А це, в свою чергу, сприяє ефективнішому розпаралелюванню. Однак можливі ситуації, коли одні диски містять багато кортежів, а інші — мало. Такий розподіл, звичайно, є небажаним. Нерівномірний розподіл може бути обумовлений такими факторами.

- **Нерівномірний розподіл значень атрибутів.** Значення деяких атрибутів у відношенні можуть розподілятися нерівномірно. Наприклад, атрибут Зарплата взагалі не має значень поза межами певного діапазону, а всередині нього, скоріш за все, значення розподілені нерівномірно. Якщо такий атрибут використовується в хешуванні чи ранжуванні, то кортежі відношення будуть розподілятися за дисками нерівномірно.
- **Нерівномірний розподіл значень у групах ранжування.** Застосування методу ранжування за неправильного формування груп може призвести до нерівномірного розподілу кортежів.

#### Забезпечення рівномірного розподілу в методі ранжування

Для забезпечення рівномірного розподілу кортежів слід створити так звані *збалансовані групи ранжування*. Якщо припустити, що атрибут ранжування формує ключ відношення, то можна:

- відсортувати кортежі відношення за значеннями цього атрибута;
- поділити кортежі відношення на $n$ груп так, щоб усі групи містили приблизно однакові кількості кортежів, і сформувати вектор ранжування на базі значень атрибута ранжування в кожній із груп ($n$ — це кількість розділів).

## 11.4. Паралельна обробка запитів

Аналізуючи проблему паралельної обробки запитів, ми розглянемо два підходи:

- розпаралелювання обробки даних між запитами;
- розпаралелювання обробки самих запитів

### 11.4.1. Розпаралелювання між запитами

Цей підхід передбачає, що запити розподіляються та обробляються водночас багатьма процесорами мультипроцесорної системи. У такий спосіб підвищується загальна кількість транзакцій, що обробляються за одиницю часу.

Розпаралелювання обробки даних між запитами є найпростішим способом підтримки паралельної обробки баз даних, особливо в тому разі, коли йдеться про архітектуру мультипроцесорних систем зі спільною пам'яттю. Значно складніше реалізувати паралельну обробку запитів у мультипроцесорній архітектурі зі спільною дисковою пам'яттю чи в архітектурі без спільної пам'яті. Це пояснюється

тим, що необхідно координувати блокування і ведення журналів, а також відстежувати можливість відновлення даних у локальних буферах різними процесорами

Існують два різновиди паралелізму між запитами:

- незалежний;
- конвеєрний.

*Незалежний паралелізм* передбачає, що запити певною мірою можна виконати незалежно, проте й тоді потрібно синхронізувати їхнє виконання За *конвеєрного паралелізму* проміжні результати одного запиту передаються іншому. У цьому випадку маємо певні переваги: система не очікує на завершення виконання одного запиту для початку виконання іншого.

## 11.4.2. Розпаралелювання обробки запиту

Цей підхід передбачає розроблення механізмів, що дають змогу розпаралелити виконання окремих запитів на багатьох процесорах і вирішити проблему прискорення обробки великих обсягів даних. Існують два підходи до вирішення цієї проблеми.

- **Розпаралелювання обчислення операцій.** Дає можливість розпаралелити виконання певної операції незалежно від інших.
- **Паралелізм між операціями.** Вирішує проблему паралельним виконанням операцій.

Перший підхід, як правило, дає кращий результат з точки зору маштабування, оскільки кількість кортежів, що обробляються кожною з операцій, є набагато більшою, ніж кількість операцій у запиті.

Для архітектури без спільної пам'яті таке розпаралелювання досягається завдяки розподілу даних за процесорами, або *декластеризації даних*. Коли виконується запит, кожний процесор працює зі своїми локальними кластерами даних, а згодом результати роботи процесорів об'єднуються в результуюче відношення.

Методи декластеризації даних (циклічний, хешування, ранжування) були розглянуті в підрозділі 11.3.

Усі теоретичні дослідження з розпаралелювання обробки запитів стосуються лише запитів у реляційній моделі даних та операцій реляційної алгебри. Саме розпаралелюванню операцій реляційної алгебри присвячений підрозділ 11.4.3, а в підрозділі 11.4.4 розглядається паралелізм між такими операціями.

## 11.4.3. Розпаралелювання операцій реляційної алгебри

Далі ми будемо розглядати лише запити на вибирання даних, що реалізуються в архітектурі багатопроцесорних систем без спільної пам'яті. Припустимо, що система складається з $n$ процесорів $P_0, ..., P_{n-1}$ і $n$ дисків $D_1, ..., D_{n-1}$, де диск $D_i$ зв'язаний із процесором $P_i$.

Зазначимо, що архітектура без спільної пам'яті може бути ефективно промодельована на системах зі спільною пам'яттю і спільною зовнішньою пам'яттю,

а отже, й алгоритми, орієнтовані на системи без спільної пам'яті, можуть бути виконані на системах з іншою архітектурою.

### Паралельне сортування

Сортування не є операцією реляційної алгебри, проте є основною допоміжною операцією під час виконання більшості реляційних запитів. Тому далі йтиметься про способи реалізації сортування на багатопроцесорній системі.

Розглянемо метод *сортування ранжуванням*.

1. Вибираємо для сортування процесори $P_0$, ..., $P_m$, де $m \le n-1$.
2. Створюємо $m$-елементний вектор ранжування, що поділяє відношення, яке сортується, на $m$ груп за значеннями атрибутів сортування.
3. Згідно з вектором ранжування розподіляємо кортежі відношення за $m$ процесорами. Усі кортежі з $i$-ї групи приписуються процесору $P_i$ і розміщуються на диску $D_i$.
4. Всі процесори сортують свої групи кортежів незалежно.
5. Виконується операція злиття відсортованих розділів. Злиття є тривіальним, оскільки всі значення атрибутів сортування на диску $D_i$ менші (або більші, якщо сортування виконувалося за спаданням), ніж на диску $D_{i+1}$.

### Паралельне виконання операції з'єднання

Операція з'єднання вимагає перевірки певної умови на парах кортежів. Якщо для пари кортежів умова виконується, то перевірена пара додається до відношення-результату.

Алгоритми паралельного з'єднання передбачають розподіл усіх можливих пар за різними процесорами. Кожний із процесорів локально перевіряє виконання умови з'єднання на своїх парах кортежів. На завершальному етапі результати обчислень, виконаних кожним із процесорів, об'єднуються.

### З'єднання шляхом розподілу кортежів за групами

Описаний далі алгоритм можна застосувати до операцій еквіз'єднання та природного з'єднання. Нехай $R$ і $S$ є вихідними відношеннями і необхідно виконати з'єднання $R \bowtie_{R.A=S.B} S$.

1. Кортежі кожного з відношень $R$ і $S$ методом ранжування або хешування поділяються на $n$ груп, що позначаються $R_0$, ..., $R_{n-1}$ і $S_0$, ..., $S_{n-1}$. Поділ $R$ і $S$ виконується за їхніми атрибутами з'єднання, $R.A$ і $S.B$.
2. Групи $R_i$ і $S_i$ надсилаються процесору $P_i$. Кожний із процесорів локально обчислює з'єднання своїх груп кортежів. У цьому випадку може застосовуватися будь-який зі стандартних методів з'єднання.
3. Результати з'єднань, що були виконані всіма процесорами, записуються на диск, де зберігатиметься загальний результат.

Графічно цей алгоритм зображений на рис. 11.4.

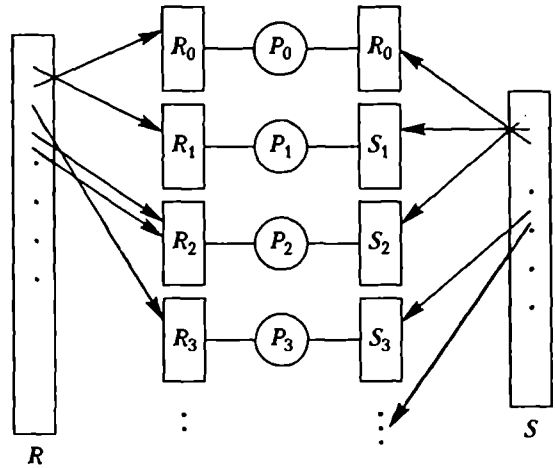



**Рис. 11.4.** З'єднання відношень шляхом розподілу їхніх кортежів за групами

### З'єднання за допомогою фрагментації з реплікацією

З'єднання поділом відношення на групи кортежів неможливе, коли умовою з'єднання не є операція рівності, наприклад коли ця умова має вигляд $R.A > S.B$. У цьому випадку паралельне з'єднання може бути реалізоване методом *фрагментації з реплікацією*. Існують методи *симетричної* (рис. 11.5, *а*) та *асиметричної* фрагментації з реплікацією (рис. 11.5, *б*). Алгоритм симетричної фрагментації з реплікацією є таким.

1. Будь-якими методами відношення $R$ поділяється на $n$ фрагментів $R_0, R_1, ..., R_{n-1}$, а відношення $S$ — на $m$ фрагментів $S_0, S_1, ..., S_{m-1}$.
2. Використовується $m \times n$ процесорів. Процесор $P_{ij}$ незалежно від інших з'єднує фрагменти $R_i$ та $S_j$. З цією метою фрагменти $R_i$ реплікуються на процесори $P_{i,0}, P_{i,1}, ..., P_{i,m-1}$, $i = \overline{0, n-1}$, а фрагменти $S_j$ — на процесори $P_{0,j}, P_{1,j}, ..., P_{n-1,j}$, $j = \overline{0, m-1}$. З'єднання на кожному із процесорів виконується довільним методом.
3. Результати з'єднань, що були виконані всіма процесорами, записуються на диск, де зберігатиметься загальний результат.

Особливістю асиметричної фрагментації з реплікацією є те, що фрагментується одне відношення, наприклад $R$, а інше, $S$, реплікується на всі процесори. Після виконання з'єднання на кожному із процесорів результат передається на процесор, що об'єднує результати.

Обидва варіанти обчислення з'єднання можна застосовувати до будь-яких умов з'єднання, оскільки вони передбачають, що для будь-якої пари кортежів відношень $R$ та $S$ умова з'єднання буде перевірена.

Зазвичай цей метод менш ефективний порівняно з поділом на групи, оскільки передбачає реплікацію одного з відношень (асиметричний варіант) чи використання великої кількості процесорів (симетричний варіант). Однак у деяких випадках варіант асиметричної фрагментації і реплікації виявляється кращим. Наприклад, нехай відношення $S$ невелике, а $R$ — велике і вже фрагментоване. У цьому випадку

ефективнішою може бути реплікація $S$ на всі процесори замість повторного поділу на групи відношень $R$ і $S$ за атрибутами, які беруть участь у з'єднанні.

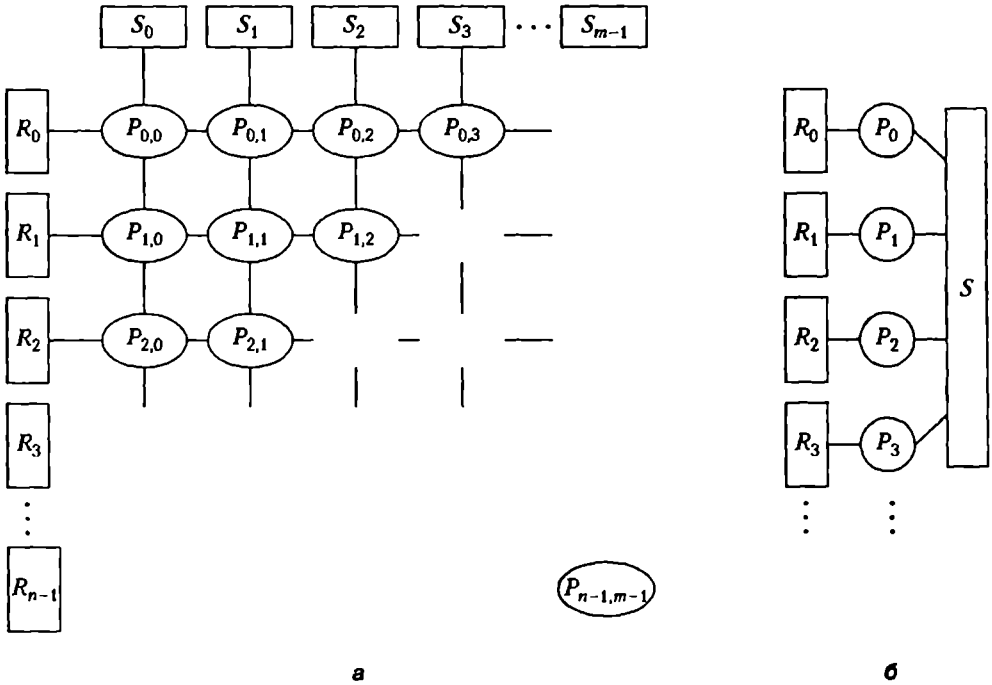



**Рис. 11.5.** З'єднання за допомогою фрагментації з реплікацією: симетрична фрагментація з реплікацією (*а*); асиметрична фрагментація з реплікацією (*б*)

### Паралельне з'єднання з циклічним гніздуванням

Припустимо, що відношення $S$ менше за відношення $R$, $S$ і $R$ поділені на фрагменти і для кожного фрагмента відношення $R$ за атрибутом з'єднання визначено індекс. Застосовуємо асиметричну фрагментацію з реплікацією, де відношення $S$ реплікується. Для цього скористаємося наявним розподілом відношення $R$.

Кожен процесор $P_j$, в якому збережена частина відношення $S$, зчитує збережені в $D_j$ кортежі відношення $S$ і реплікує їх на кожний процесор $P_i$. Після завершення цієї фази відношення $S$ репліковане на всі диски, де зберігаються кортежі відношення $R$. Затим кожен процесор виконує індексоване з'єднання відношення $S$ з $i$-м фрагментом відношення $R$.

### Паралельне виконання селекції

Якщо умова селекції має вигляд $a_i = v$, де $a_i$ – атрибут, а $v$ – значення, і якщо відношення $R$ поділене на фрагменти за значеннями атрибута $a_i$, то селекція виконується на одному процесорі.

Якщо умова селекції має вигляд $u_1 \le a_i \le u_2$ (тобто селекція у діапазоні значень) і відношення розподілене за процесорами методом ранжування за значеннями

атрибута $a_i$, то селекція обчислюється на тих процесорах, що містять кортежі зі вказаного діапазону значень.

В інших випадках селекція виконується паралельно на всіх процесорах, де розташовані кортежі відношення.

#### Видалення дублікатів значень

Для видалення значень, що повторюються, застосовується будь-який метод паралельного сортування. Можна також поділити відношення на фрагменти (ранжування чи хешування) і видалити дублікати локально на кожному з процесорів

#### Паралельне виконання проекції

Проекція без видалення дублікатів може бути виконана під час паралельного зчитування кортежів із дисків. Якщо необхідно видалити дублікати, можна скористатися будь-яким методом сортування.

#### Паралельне групування й обчислення агрегатних функцій

Поділяємо відношення на фрагменти за атрибутами групування й локально на кожному з процесорів обчислюємо агрегатні значення. У деяких випадках можна зменшити кількість пересилань кортежів шляхом часткового обчислення агрегатних значень перед поділом відношення на фрагменти.

### 11.4.4. Паралелізм між операціями реляційної алгебри

#### Конвеєрне виконання операцій

Розглянемо з'єднання чотирьох відношень $R_1 \bowtie R_2 \bowtie R_3 \bowtie R_4$. У конвеєрній архітектурі на процесорах $P_1$–$P_3$ його можна обчислити в такий спосіб:

- $P_1$ обчислює $temp_1 = R_1 \bowtie R_2$;
- $P_2$ обчислює $temp_2 = temp_1 \bowtie R_3$;
- $P_3$ обчислює $temp_2 \bowtie R_4$.

Усі ці операції можна виконати паралельно шляхом пересилання результатів обчислень на одному процесорі іншому, незважаючи на те, що «попередній» процесор ще не повністю виконав з'єднання.

Конвеєрні обчислення дають змогу уникнути збереження проміжних результатів на диску. Проте конвеєрно не можна виконувати операції, які не дають результату доти, доки не будуть отримані всі вихідні дані (до таких операцій належать зокрема обчислення агрегатних функцій та сортування).

#### Незалежне паралельне виконання операцій

Знову розглянемо з'єднання чотирьох відношень $R_1 \bowtie R_2 \bowtie R_3 \bowtie R_4$. Нехай між процесорами $P_1$–$P_3$ обчислення розподілені так:

- $P_1$ обчислює $temp_1 = R_1 \bowtie R_2$;
- $P_2$ обчислює $temp_2 = R_3 \bowtie R_4$;
- $P_3$ обчислює $temp_1 \bowtie temp_2$.

У цьому випадку $P_1$ і $P_2$ можуть працювати одночасно й незалежно, а $P_3$ очікує на результати обчислення перших двох процесорів. $P_3$ може також виконувати обчислення в конвеєрному режимі стосовно $P_1$ і $P_2$.

## Контрольні запитання та завдання

1. Від чого залежить ефективність паралельної обробки даних?
2. У чому полягає відмінність між прискоренням обчислень та масштабуванням?
3. Які переваги дає розпаралелювання обробки даних?
4. Назвіть різновиди архітектури багатопроцесорних систем. Які переваги та недоліки має кожен із них?
5. Опишіть методи розподілу даних.
6. У чому полягає суть нерівномірного розподілу даних?
7. Що таке розпаралелювання між запитами та всередині запиту?
8. Опишіть алгоритм паралельного сортування даних.
9. Які є алгоритми паралельного з'єднання відношень?
10. Опишіть методи розпаралелювання операцій реляційної алгебри.

# Розділ 12

# Дедуктивні бази даних

- Основні поняття дедуктивних баз даних
- Мова DATALOG
- Обчислення нерекурсивних і рекурсивних Datalog-програм
- Обчислення правил із запереченнями

## 12.1. Основні поняття дедуктивних баз даних

Напрямок дедуктивних баз даних, як однієї з галузей теорії баз даних, почав інтенсивно розвиватися в останні 20 років. Вихідним пунктом для нього стала теорія логічного програмування. В основу цих двох напрямків покладено числення предикатів першого порядку.

Як логічна програма, так і дедуктивна база даних – це послідовність формул числення предикатів, що мають спеціальний вигляд. Однак у першому випадку сукупність таких формул інтерпретується як послідовність операцій, спрямованих на вирішення конкретної обчислювальної задачі, а в другому – як послідовність перетворень даних певної структури. Особливості другого напрямку були породжені специфічними для баз даних проблемами, які не розглядаються в логічному програмуванні.

Дедуктивна база даних реалізується як звичайна (як правило, реляційна) база даних разом із множиною логічних правил, що вказують способи породження нових даних. Наприклад, нехай база даних містить відношення

```
Родич(Батько, Син)
```

і такі правила:

```
Дід – це батько батька.
Онук – це син сина
```

До такої бази даних, що містить правила породження понять «дід» і «онук», можна ставити запити типу: «Хто дід Івана?», «Визначити (знайти) всіх онуків Петра» тощо.

Далі у цьому розділі описуються основні способи визначення змісту дедуктивних баз даних і розглядається мова опису дедуктивних баз даних Datalog. Крім того, досліджуються три класи Datalog-програм: нерекурсивні, рекурсивні та з запереченнями, а також описуються правила визначення (обчислення) їхнього змісту.

## 12.2. Інтерпретація логічних правил

Розглянемо такі логічні правила:

```
бос(E, M) :- керівник(E, M)
бос(E, M) :- бос(E, N) & керівник(N, M)
```

Ліва та права частини логічного правила відокремлюються символами :-. Як у лівій, так і в правій частині правила розташовані предикати з вільними змінними. Правило інтерпретується так: якщо істинним є вираз, записаний у його правій частині, то істинним є також предикат у лівій частині правила. Наприклад, якщо змінні *E*, *M*, *N* у правій частини наведених вище правил будуть заміщені істинними виразами, то й вирази, записані у лівих частинах, теж будуть істинним для тих самих заміщень змінних.

Існують три альтернативні способи визначення змісту правил:

- теоретико-довідний;
- теоретико-модельний;
- визначення за допомогою обчислень.

У простих випадках, як у наведеному вище прикладі, всі три методи визначення змісту правил дають одну відповідь. Однак для складних логічних правил гарантії однозначності відповіді немає.

У разі *теоретико-довідної інтерпретації* правила розглядаються як аксіоми, що використовуються у доведеннях. Тобто на підставі фактів, що є в базі даних, відшукується множина інших фактів, що могли б бути доведені з використанням заданих правил.

*Теоретико-модельна інтерпретація* передбачає, що правила розглядаються як спосіб визначення можливих світів чи моделей. У результаті інтерпретації сукупності предикатів істинне чи хибне значення присвоюється кожному можливому екземпляру предиката, причому змінним присвоюються значення з нескінченного домену. Інтерпретація визначається множиною екземплярів, на яких предикати є істинними. Інтерпретація є *моделлю* для множини правил, якщо в разі цієї інтерпретації всі логічні правила стають істинними незалежно від того, які значення з домену присвоєні змінним, що використовуються у правилах.

Визначення змісту *за допомогою обчислень* передбачає, що надається алгоритм, який для кожного потенційно можливого факту (тобто для предиката, у якому змінні замінені значеннями) визначає, є він істинним чи хибним.

У цьому розділі правила будуть описуватись у вигляді послідовностей операцій реляційної алгебри.

Виникає питання, який із наведених способів інтерпретації логічних програм є найкращим. Логіки, наприклад, не сприймають серйозно обчислювальний зміст правил. Однак для тих, хто замислюється над реалізацією баз знань, питання ефективного обчислення правил є принциповим, оскільки логічні правила не можна використовувати як програми доти, доки не буде визначений спосіб їхнього обчислення.

## 12.3. Мова DATALOG

Дамо означення основних понять, необхідних для розгляду логічної моделі даних Datalog (від англ. Database Logic). Datalog – це мова запитів, що базується на мові програмування Prolog. Проте Datalog відрізняється від мови Prolog за деякими аспектами.

- У Datalog не можна використовувати функціональні символи як аргументи предикатів. Наприклад, у Datalog неприпустиме таке означення суми:

```
Sum(x,0,x)
Sum(x,s(y),s(z)) :- Sum(x,y,z)
```

  Тут *s* є функціональним символом, що позначає наступне число. Як значення аргументів предикатів у Datalog використовуються константи і змінні.

- Зміст Datalog-програм розкривається за допомогою теоретико-модельного або теоретико-довідного підходу.

### Атомарні формули

Запити, сформульовані мовою Datalog, будуються з *атомарних формул*, що є предикатами з кількома аргументами, наприклад $p(A_1, ..., A_n)$. Аргументами в Datalog можуть бути константи або змінні. Для позначення констант і предикатів ми використовуватимемо ідеитифікатори, що починаються з малої літери, а для позначення змінних і відношень – ідентифікатори, що починаються з великої літери. З кожним предикатним символом зв'язується число, що вказує кількість аргументів предиката. Крім того, ми користуватимемося записом вигляду $p^k$, якщо потрібно вказати, що $p$ є $k$-місним предикатом.

Мова запитів Datalog означена для реляційної моделі даних. Предикатні символи в Datalog позначають відношення. На відміну від реляційної алгебри, в мові Datalog припускається, що компоненти кортежів відношення впорядковані за аргументними місцями відповідного предикатного символу, і посилання на атрибут може здійснюватись за його номером. Наприклад, якщо $p$ є предикатним символом, то в записі $p(X, Y, Z)$ змінна $X$ буде позначати перший компонент кортежу відношення, що відповідає предикату $p$.

Атомарна формула позначає відношення, яке зіставляється з відношенням предикатного символу за такими правилами:

- якщо всі аргументи атомарної формули є змінними й усі вони позначаються різними ідентифікаторами, то відношення атомарної формули збігається з відношенням предикатного символу;
- якщо певний аргумент атомарної формули є константою, цій формулі відповідає відношення, отримане з відношення предикатного символу вибиранням лише тих кортежів, компоненти яких містять цю константу;
- якщо два чи більше аргументи позначаються однаковими ідентифікаторами змінних, то атомарній формулі відповідає відношення, отримане з відношення предикатного символу вибиранням лише тих кортежів, відповідні компоненти яких рівні.

Наприклад, розглянемо такі відношення:

```
Замовлення(Ім'я, Адреса, Товар, Кількість)
Видача(Ім'я, Товар, Замовлено, Видано)
```

Атомарна формула замовлено(Джон, Адреса, Товар, Кількість) зображує відношення $\sigma_{s_1='Джон'}$ (*Замовлення*), а атомарна формула видано(Ім'я, Товар, X, X) відображує відношення $y_{s_3=s_4}$ (*Видача*). Зазначимо, що тут і далі в цьому розділі в операціях реляційної алгебри символами $s_i$ позначатиметься $i$-й атрибут відношення.

У Datalog не використовуються імена атрибутів відношень, однак замість них застосовуються змінні типу Адреса, Товар, що мають інформативні імена. Як і в реляційній алгебрі, ми маємо пам'ятати семантичний зміст кожної позиції у списку аргументів предиката.

### Екстенсійні та інтенсійні предикати

У Datalog існують два способи означення відношень. Предикат, відношення якого зберігається в базі даних, називається предикатом *екстенсійної бази даних* (ЕБД), а предикат, визначений за допомогою логічних правил, — предикатом *інтенсійної бази даних* (ІБД). Ми вважатимемо, що будь-який предикатний символ позначає відношення ЕБД або відношення ІБД, але не обидва разом. Відповідні їм відношення будуть називатися ЕБД- чи ІБД-відношеннями.

У реляційній моделі всі відношення є ЕБД-відношеннями. Можливість створення віртуальних відношень у моделях типу реляційної подібна до можливості Datalog визначати відношення ІБД. Однак, порівняно з логічними правилами, можливості з визначення віртуальних відношень, що надаються реляційними СКБД, виявляються слабшими.

### Вбудовані предикати

Ми будемо використовувати атомарні формули з предикатами порівняння $=, \geq, \neq$, які називаються *вбудованими предикатами*. Атомарні формули зі вбудованими предикатами записуватимуться у звичайній інфіксній нотації, наприклад $X < Y$, а не $<(X, Y)$. Інші атомарні формули та їхні предикати називаються *звичайними* на підкреслення того, що вони не є вбудованими.

Вбудовані предикати не обов'язково зображують скінченні відношення. Можна було б припустити, що предикат $X < Y$ відповідає відношенню, яке містить усі такі кортежі $(x, y)$, що $x < y$, однак це невдале припущення, оскільки побудована множина пар є нескінченною; більш того, не зрозуміло, значення з якого домену мають набувати $x$ та $y$. У зв'язку з цим множини значень усіх змінних формули зі вбудованим предикатом мають бути обмеженими за областю зміни іншої атомарної формули з правої частини цього правила. Наприклад, змінна може бути обмежена через те, що вона входить до складу атомарної формули, якій відповідає ЕБД- відношення. За таких умов вбудовані атомарні формули можуть інтерпретуватися як механізм селекції в іншому відношенні чи у з'єднанні кількох відношень. Більш докладно це питання розглядатиметься під час вивчення безпечних правил наприкінці підрозділу 12.3.

### Диз'юнкти і хорнівські диз'юнкти

*Літералом* називається атомарна формула або атомарна формула з запереченням. Ми позначатимемо атомарну формулу з запереченням як $\neg p(A_1, ..., A_n)$. Атомарна формула з запереченням називається *негативним літералом*, а без заперечення *позитивним літералом*. *Диз'юнкт* — це диз'юнкція (логічне АБО) скінченної кількості літералів. *Хорнівським диз'юнктом* називається диз'юнкт не більше ніж з одним позитивним літералом. Сукупність хорнівських диз'юнктів називається *логічною програмою*.

Отже, хорнівськими диз'юнктами є такі різновиди атомарних формул:

- єдиний позитивний літерал, наприклад $p(X, Y)$, що може розглядатись як факт;
- один або більше негативних літералів і жодного позитивного (такі диз'юнкти називаються обмеженнями цілісності й далі детально не розглядатимуться);
- один позитивний літерал і один чи більше негативних (такі диз'юнкти називаються *правилами*).

Диз'юнкти третього типу називаються правилами, оскільки вони подаються у вигляді імплікації, тобто хорнівський диз'юнкт $\neg p_1 \vee \neg p_2 \vee ... \vee \neg p_n \vee q$ логічно еквівалентний виразу $p_1 \,\&\, p_2 \,\&...\&\, p_n \Rightarrow q$.

Надалі для позначення хорнівських диз'юнктів ми будемо використовувати стиль мови Prolog, а саме записуватимемо $q :- p_1 \,\&...\&\, p_n$ замість $p_1 \,\&...\&\, p_n \Rightarrow q$.

*Головою правила* $q :- p_1 \,\&...\&\, p_n$ будемо називати $q$, а *тілом* — $p_1 \,\&...\&\, p_n$. Усі $p_i$ є *підцілями правила*.

### Інтерпретація правил

Змінні, що згадуються лише в тілі, інтерпретуються за допомогою квантора існування в межах тіла, інші змінні інтерпретуються за допомогою квантора загальності в межах правила

**Приклад 12.1**

Розглянемо приклад логічного правила:

```
брат/сестра(X,Y) :- батько(X,Z) & батько(Y,Z) & X ≠ Y;
```

Дане правило означає, що для всіх $X$ та $Y$ справедливим є таке твердження: $X$ є братом/сестрою для $Y$, якщо існує такий $Z$, що $Z$ є батьком для $X$ і для $Y$, а також $X$, $Y$ є різними об'єктами.

Зазначимо, що всі змінні в межах правила можна коректно інтерпретувати за допомогою квантора загальності. Зокрема вказане вище правило може бути прочитане так: для всіх $X$, $Y$, $Z$, якщо $Z$ є батьком як $X$, так і $Y$, та $X$ не збігається з $Y$, то $X$ є братом/сестрою для $Y$.

### Графи залежностей і рекурсія

У межах логічної програми одні предикати залежать від інших. Ці залежності зображуються за допомогою *графа залежностей*. Кожна його вершина відображує звичайний предикат, дуга свідчить про існування правила, підціллю якого є предикат, із якого виходить дуга, а головою — предикат, до якого спрямована дуга.

Логічна програма є *рекурсивною*, якщо граф залежностей містить принаймні один цикл. Усі предикати, що входять до складу циклів, називаються *рекурсивними*.

Існування циклу, що складається з однієї дуги, початок і кінець якої збігаються, також свідчить про рекурсивність логічної програми. Такі цикли в рекурсивних програмах трапляються частіше за цикли, що містять багато вершин.

Логічна програма з ациклічним графом залежностей називається *нерекурсивною*. Всі предикати в нерекурсивній програмі є *нерекурсивними*. Нерекурсивними ми будемо називати також ті предикати, що належать рекурсивній програмі, але не входять до складу жодного з циклів.

**Приклад 12.2**

Запишемо логічну програму, що складатиметься з правила, наведеного у прикладі 12.1, та ще кількох правил.

```
брат/сестра(X,Y) :- батько(X,Z) & батько(Y,Z) & X ≠ Y          (1)
кузен(X,Y) :- батько(X,Xp) & батько(Y,Yp) & брат/сестра(Xp,Yp)  (2)
кузен(X,Y) :- батько(X,Xp) & батько(Y,Yp) & кузен(Xp,Yp)       (3)
родич(X,Y) :- брат/сестра(X,Y)                                 (4)
родич(X,Y) :- родич(X,Z) & батько(Y,Z)                         (5)
родич(X,Y) :- родич(Z,Y) & батько(X,Z)                         (6)
```

Граф залежностей для цієї логічної програми наведений на рис. 12.1. Ми не створюємо вершини для вбудованого предиката ≠. У графі є два цикли, один з яких містить єдину вершину родич, інший — єдину вершину кузен. Отже, ці два предикати є рекурсивними, а предикати брат/сестра і батько — нерекурсивними. Очевидно, що ЕБД-відношення типу батько мають бути нерекурсивними. Фактично ЕБД-предикатами є такі предикати, вершини яких не мають вхідних дуг, відтак вони не можуть бути рекурсивними. У цілому програма є рекурсивною, оскільки містить рекурсивні предикати.

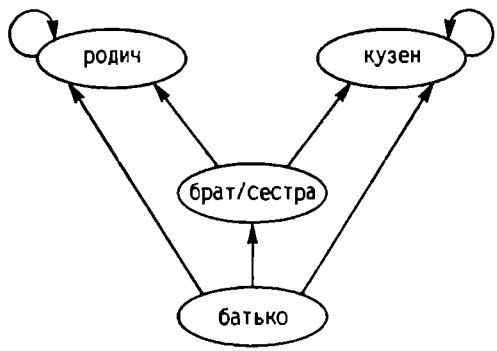



**Рис. 12.1.** Граф залежностей для логічної програми

### Безпечні правила

Якщо ми хочемо, щоби Datalog-правила інтерпретувалися як операції над скінченними відношеннями, то на структуру правил необхідно накласти обмеження. Одним із джерел появи нескінченних відношень є змінні, що входять до складу ли-

ше вбудованих предикатів. Іншим таким джерелом є змінні, що належать тільки голові правила. Для уникнення появи нескінченних відношень необхідно, щоби будь-який звичайний (невбудований) предикат, що згадується в тілі, відповідав скінченному відношенню. Ми маємо бути впевненими, що для кожної змінної $X$ існує така скінченна множина значень $V_X$, що їй належать усі значення $X$, за яких тіло предиката може бути істинним. Дамо формальне означення поняття *обмеженої змінної* для заданого правила.

1. Змінна, що вказана в тілі правила як аргумент звичайного предиката, є обмеженою.
2. Змінна $X$, що вказана в підцілі $X = a$ чи $a = X$, де $a$ — константа, є обмеженою.
3. Змінна $X$, що вказана в підцілі $X = Y$ чи $Y = X$, де $Y$ — обмежена змінна, є обмеженою.

Означення 1 і 2 є основними, а означення 3 може застосовуватися рекурсивно.

Правило називається *безпечним*, якщо всі його змінні обмежені. Правило (1) з прикладу 12.2 є безпечним, адже змінні $X$, $Y$, $Z$ обмежені, оскільки входять до складу двох підцілей з предикатом батько. Зазначимо, що вбудований предикат X≠Y не може призвести до породження нескінченної множини братів/сестер, тому що $X$ і $Y$ уже обмежені як компоненти відношення батько. Всі інші правила з прикладу 12.2 також є безпечними.

Розглянемо правило

```
q(X,Y) :- p(X,Z) & W=a & Y=W
```

Тут $X$ і $Z$ обмежені на підставі означення 1, оскільки вони входять до складу першої підцілі тіла, змінна $W$ обмежена за означенням 2, а змінна $Y$ — за означенням 3. Зауважте, що відношення для правила є рівним

$$\pi_{s_1}(p) \times \{a\}$$

Очевидно, що це відношення буде скінченним за умови скінченності відношення, яке відповідає $p$.

## 12.4. Обчислення нерекурсивних Datalog-програм

Розглянемо нерекурсивні Datalog-програми. Для цього простого класу програм усі три можливі тлумачення (теоретико-модельне, теоретико-довідне та обчислювальне) збігаються. Ми покажемо спосіб перетворення нерекурсивних Datalog-правил на вирази реляційної алгебри. Ці вирази визначають відношення для предикатів ІДБ і становлять водночас *єдину мінімальну модель правил* та множину ІДБ-фактів, виведених із правил і бази даних.

У цьому підрозділі ми почнемо викладати матеріал із теоретико-довідної точки зору, а потім покажемо, що правила коректно інтерпретуються їхньою єдиною мінімальною моделлю.

Якщо правила не є рекурсивними, можна частково впорядкувати вершини графа залежностей $p_1, ..., p_n$: якщо є дуга з $p_i$ у $p_j$, то $i < j$. Опишемо, як згідно з побудованим порядком обчислити відношення для $p_1, ..., p_n$.

Припустимо, що всі правила для $p_i$ вже обчислені. Обчислення відношення для предиката $p_i$ поділяється на два кроки.

1. Для голови $p_i$ кожного правила $r$ обчислюється відношення, що відповідає тілу правила. У цьому відношенні кожній змінній з правила $r$ відповідає один атрибут, а саме відношення дорівнює природному з'єднанню відношень, що відповідають усім підцілям $r$, якщо вважати атрибутами цих відношень змінні, вказані у відповідних позиціях підцілей. Оскільки правила є нерекурсивними, можна припустити, що всі відношення для підцілей уже обчислені.
2. Відношення для предиката $p_i$ обчислюються виконанням проекцій відношень для всіх правил, що визначають $p_i$, за компонентами, які відповідають змінним із голів цих правил, з наступним об'єднанням відношень, що задовольняють правила з $p_i$ в голові.

Описані обчислення складніші за просте виконання з'єднань і проекцій. Потрібно враховувати наявність констант в аргументах та можливість входження однієї і тієї ж змінної в різні аргументи однієї чи кількох підцілей або голови. Ці моменти будуть враховані в алгоритмі 12.2, що описаний нижче.

## 12.4.1. Відношення, обумовлені тілом правила

Насамперед ми маємо визначити множину значень змінних, за яких тіло правила буде істинним. Із теоретико-довідної точки зору заміна змінних такими значеннями є саме тією заміною, яка дає можливість стверджувати, що голова правила також стає істинною. Відношення для правила $r$ має схему $X_1, ..., X_m$, де $X_i$ є змінними тіла $r$, що перелічуються в певному порядку. Ми будемо припускати, що дане відношення містить кортеж $(a_1, ..., a_m)$ тоді й лише тоді, коли при підстановці $a_i$ замість $X_i$, $1 \le i \le m$, усі підцілі правила стають істинними. Припустимо, що $p_1, ..., p_n$ є списком усіх предикатів тіла правила $r$ і $P_1, ..., P_n$ є такими відношеннями: $P_i$ складається з усіх кортежів $(a_1, ..., a_k)$, для яких $p_i(a_1, ..., a_k)$ є істинним предикатом. Підціль $s$ правила $r$ стає істинною для цієї підстановки, якщо:

- $s$ є звичайною підціллю та стає за цієї підстановки рівною $p(b_1, ..., b_k)$, де $(b_1, ..., b_k)$ є кортежем відношення $P$, яке відповідає предикату $p$;
- $s$ є вбудованою підціллю та за цієї підстановки набуває вигляду істинного виразу $b\ \theta\ c$, де $\theta$ - символ операції порівняння.

**Приклад 12.3**

Нижче наведено приклад неформальної побудови відношення для тіла правила. Формальний метод буде описано в алгоритмі 12.1.

Розглянемо правило

```
q(X,Y) .- p(a,X) & r(X,Z,X) & s(Y,Z)
```

Припустимо, що ми обчислили відношення $P$, $R$, $S$ для підцілей $p$, $r$, $s$. Для першої підцілі необхідно побудувати відношення $T$ з атрибутом $X$, кортежі якого міститимуть лише другий компонент $P$. При цьому слід відібрати лише ті кортежі $P$, значення першого компонента яких дорівнює $a$:

$$T(X) = \pi_{s_2}(\sigma_{s_1=a}(Q)).$$

Ми повинні обмежити відношення $R$ у такий спосіб, щоб його перший і третій компоненти, що відображуються в предикаті другої підцілі однією і тією ж змінною $X$, були рівними. Тому означимо відношення

$$U(X,Y) = \pi_{s_1,s_2}(\sigma_{s_1=s_3}(R))$$

Відношення для тіла наведеного вище правила визначається виразом

$$T(X) * U(X,Z) * S(Y,Z)$$

Цей вираз визначає множину кортежів $(x, y, z)$, які роблять істинним тіло правила, тобто таку множину кортежів, що:

- $(a, x)$ належить $P$;
- $(x, z, x)$ належить $R$;
- $(y, z)$ належить $S$.

Тепер наведемо формальний алгоритм побудови виразу реляційної алгебри, що обчислює відношення для тіла правила.

**Алгоритм 12.1**

**Вхід.** Тіло Datalog-правила $r$, що складається з підцілей $s_1, ..., s_n$, які містять змінні $X_1, ..., X_m$. Для кожної підцілі $s_i$, яка є звичайним предикатом, уже обчислено відношення $R_i$.

**Вихід.** Вираз реляційної алгебри, який ми назвемо EVAL-RULE($r$, $R_1, ..., R_n$). Зазначимо, що не обов'язково всі відношення будуть операндами цього виразу, оскільки деякі з підцілей можуть бути вбудованими предикатами й не мати відповідних відношень. Отримане в результаті обчислення EVAL-RULE відношення $R(X_1, ..., X_m)$ міститиме ті й лише ті кортежі $(a_1, ..., a_m)$, для яких за підстановки $a_j$ замість $X_j$, $1 \le j \le m$, усі підцілі стають істинними.

**Метод.** Вираз EVAL-RULE будується у чотири етапи.

1 Для кожного звичайного предиката $s_i$ обчислимо вираз $Q_i = \pi_{V_i}(\sigma_{F_i}(R_i))$.

Тут $V_i$ є множиною компонентів, побудованою за таким принципом: якщо є хоча б одне входження змінної $X$ у множину аргументів предиката $s_i$, то ця змінна один раз входить і у $V_i$; $F_i$ є кон'юнкцією таких умов:

- якщо в позиції $k$ предиката $s_i$ міститься константа $a$, то $F_i$ містить терм $\$_k = a$ (тут $\$_k$ позначає $k$-й атрибут відношення);
- якщо в позиціях $k$ та $j$ предиката $s_i$ міститься та сама змінна, то $F_i$ містить терм $\$_k = \$_j$. Зазначимо, що немає потреби будувати терми для всіх подібних

пар. Наприклад, якщо $X$ міститься в позиціях 2, 5, 9, 14, то достатньо додати терми $\$_2 = \$_5$, $\$_5 = \$_9$, $\$_9 = \$_{14}$.

Спеціальним випадком є така ситуація, коли $F_i$ не містить термів, наприклад $s_i = p(X, Y)$. У цьому випадку $F_i$ стає тотожно істинною умовою і $Q_i = R_i$.

2. Для кожної змінної $X$, що не належить жодній зі звичайних підцілей, обчислимо вираз $D_X$. Результатом цього виразу є унарне відношення, яке містить усі такі значення, що їх могла б набувати змінна $X$ у разі здійснення присвоювань, які відповідають усім підцілям правила $r$. Оскільки $r$ є безпечним правилом, то існує така змінна $Y$, до якої прирівнюється змінна $X$ через послідовність однієї чи більше підцілей (або предикатів =) і яка є обмеженою або її прирівнюванням до константи $a$, або входженням як аргумент у звичайну підціль. Розглянемо ці два випадки:
   - якщо $Y = a$ є підціллю, то $D_X$ є виразом-константою $\{a\}$;
   - якщо $Y$ є аргументом $j$ звичайної підцілі $s_i$, то $D_X = \pi_j(R_i)$.
3. Нехай $E$ є природним з'єднанням усіх $Q_i$, визначених у пункті 1, а $D_X$ задані згідно з пунктом 2. У цьому з'єднанні ми розглядаємо $Q_i$ як відношення, чиї атрибути є змінними, що згадуються в $s_i$, а $D_X$ розглядаємо як відношення з атрибутом $X$. Оскільки жодна зі змінних $X$, для яких будуються відношення $D_X$, не є атрибутом жодного з $Q_i$, то фактично природне з'єднання для $D_X$ зводиться до декартового добутку
4. EVAL-RULE($r$, $R_1$, ..., $R_n$) = $\sigma_F(E)$, де $F$ дорівнює кон'юнкції виразів $X\,\theta\,Y$, що відповідають усім вбудованим підцілям $X\,\theta\,Y$, а $E$ є виразом, побудованим згідно з правилами, наведеними в пункті 3. Якщо тіло не містить вбудованих підцілей, то EVAL-RULE($r$, $R_1$, ..., $R_n$) = $E$.

Алгоритм 12.1 є коректним, оскільки відношення $R$ містить такі й лише такі кортежі $(a_1, ..., a_n)$, що за умови підстановки всіх $a_j$ замість $X_j$ усі підцілі $s_i$ стають істинними.

## 12.4.2. Зведені правила

Під час обчислення відношення для правила можуть виникнути ускладнення, якщо аргументами предиката $p$, записаного в голові правила, є константи або змінні, що повторюються, наприклад $p(a, X, X)$. У зв'язку з цим ми введемо поняття зведених правил. Правила для предиката $p$ є *зведеними*, якщо всі їхні голови однакові й мають вигляд $p(X_1, ..., X_k)$, де $X_1, ..., X_k$ – різні змінні.

Зводити правила дуже легко: слід створити нові змінні для кожного з аргументів предиката голови, а в тіло ввести вбудовані підцілі, які б забезпечували заміну однієї змінної на іншу або константи на змінну. Припустимо, що маємо правило $r$ з головою $p(Y_1, ..., Y_m)$, де $Y_i$ можуть бути змінними або константами, і при цьому імена змінних можуть повторюватися. Замінимо голову правила $r$ на $p(X_1, ..., X_k)$, де $X_i$ є новими неповторюваними змінними, а в тіло правила $r$ додамо підцілі $X_i = Y_i$, де $i = \overline{1,k}$. Якщо $Y_i$ є змінною, то замість введення підцілі $X_i = Y_i$ можна замінити всі входження в тіло змінної $Y_i$ на змінну $X_i$.

**Твердження 12.1**

Припустимо, що $r$ – правило, а $r'$ – зведений вигляд цього правила. Якщо $r$ – безпечне правило, то безпечним є і правило $r'$. Правила $r$ і $r'$ є еквівалентними, оскільки існує можливість таким чином перейменувати змінні $r$, що голова цього правила буде істинною тоді й тільки тоді, коли істинною буде голова правила $r'$, і навпаки.

## 12.4.3. Обчислення відношень для нерекурсивних програм

Після того як правила певного предиката будуть зведені, потрібно спроектувати відношення для тіла кожного правила за змінними голови й об'єднати отримані проекції. Цей принцип використовується в алгоритмі інтерпретації нерекурсивної програми за допомогою операцій реляційної алгебри.

**Алгоритм 12.2**

**Вхід.** Нерекурсивна Datalog-програма і відношення для кожного з ЕБД-предикатів, що означуються в програмі.

**Вихід.** Вираз реляційної алгебри для кожного ІБД-предиката $p$. Даний вираз визначає відношення для $p$ у термінах відношень $R_1$, ..., $R_m$ для ЕБД-предикатів.

**Метод.** Вираз будується у п'ять етапів.

1. Почнемо зі зведення всіх правил, потім побудуємо граф залежностей для вихідної програми й упорядкуємо предикати $p_1$, ..., $p_n$ так, щоб для всіх дуг графа залежностей програми, які ведуть з $p_i$ у $p_j$, виконувалася нерівність $i < j$. Таку впорядкованість можна встановити, оскільки вихідна програма є нерекурсивною і граф залежностей не містить циклів.

2. Для кожного $p_i$, $i = 1, ..., n$, побудуємо відповідний вираз реляційної алгебри. Якщо $p_i$ – ЕБД-предикат, то його відношення $P_i$ є заданим. Якщо ж $p_i$ – ІБД-предикат, тоді його відношення будується згідно з правилами, зазначеними в пунктах 3–5.

3. Щоб віднайти вираз $E_r$, який обчислює відношення $R_r$ для тіла кожного правила $r$, головою якого є $p_i$, застосуємо алгоритм 12.1. Вираз $E_r$ дає можливість обчислити відношення $R_r$ у термінах відношень для предикатів тіла.

4. Оскільки програма нерекурсивна, то відношення для всіх предикатів, що вказуються в тілі $r$, можуть бути виражені через ЕБД-відношення. Підставивши у $E_r$ вирази для кожного ІБД-відношення, отримаємо новий вираз $F_r$, в якому згадуються лише ЕБД-відношення.

5. Якщо це необхідно, перейменуємо змінні так, щоб голови всіх правил для $p_i$ мали однаковий вигляд. Спроектувавши відношення, що відповідають тілам правил, за змінними голови й об'єднавши отримані проекції, матимемо відношення $P_i$.

**Приклад 12.3**

Аби проілюструвати роботу алгоритму 12.2, розглянемо абстрактний приклад. Припустимо, що є такі правила:

```
p(a,Y) :- r(X,Y)                    (1)
p(X,Y) :- s(X,Z) & r(Z,Y)           (2)
q(X,X) :- p(X,b)                    (3)
q(X,Y) :- p(X,Z) & s(Z,Y)           (4)
```

Тут $r$ і $s$ є ЕБД-предикатами, для яких задані відношення $R$ і $S$. Предикати $p$ і $q$ є ІБД-предикатами, для яких необхідно обчислити відношення $P$ і $Q$.

Почнемо зі зведення, у результаті якого отримаємо такий набір правил:

```
p(X,Y) :- r(X,Y) & X=a              (1)
p(X,Y) :- s(X,Z) & r(Z,Y)           (2)
q(X,Y) :- p(X,b) & X=Y              (3)
q(X,Y) :- p(X,Z) & s(Z,Y)           (4)
```

Оскільки предикат $q$ залежить від предиката $p$, допустимим упорядкуванням правил є таке, за якого спочатку обчислюється $p$, а потім $q$. Відношення для тіла правила (1), відповідно до алгоритму 12.1, зображує вираз

$$R(X,Y) * D_a(X).$$

Тут $D_a(X)$ — відношення-константа, яке має один атрибут $X$ і один рядок зі значенням $a$; для тіла правила (2) відношення має вигляд $S(X, Z) * R(Z, Y)$.

Обидва відношення перед об'єднанням мають бути спроектовані за атрибутами $X$, $Y$. Перше з цих відношень проектування не потребує. Отже, в результаті отримуємо такий вираз для відношення $P$ предиката $p$:

$$P(X,Y) = \pi_{X,Y}(R(X,Y) * D_a(X)) \cup \pi_{X,Y}(S(X,Z) * R(Z,Y)).$$

Обчислимо відношення для тіла правила (3). Згідно з алгоритмом 12.1, вираз для підцілі $p(X, b)$ є таким:

$$\pi_X(\sigma_{Z=b}(P(X,Z))),$$

де $Z$ є вільною змінною, відсутньою в проекції. Результатом цього виразу є відношення з одним атрибутом $X$. Оскільки $Y$ прирівнюється до $X$, тільки значення $X$ можуть бути значеннями $Y$, тому можна взяти аргумент, де зустрічається $X$, а саме перший аргумент відношення $P$, як домен для $Y$. Цей домен виражається так:

$$\pi_Y(P(Y,W)).$$

Тут $W$ є ще однією вільною змінною. Після обчислення декартового добутку виразу для $p(X, b)$ і домену значень $Y$, здійснюємо селекцію отриманого відношення за умовою $X = Y$ у зв'язку з існуванням у правилі (3) підцілі $X = Y$. Отже, вираз для тіла правила (3) набуває вигляду:

$$\sigma_{X=Y}(\pi_X(\sigma_{Z=b}(P(X,Z))) \times \pi_Y(P(Y,W))).$$

Виразом для правила (4) є

$$P(X, Z) * S(Z, Y).$$

Отже, відношення $Q$ для предиката $q$ визначається так:

$$Q(X,Z) = \sigma_{X=Z}(\pi_X(\sigma_{Z=b}(P(X,Z))) \times \pi_Y(P(Y,W))) \cup \pi_{X,Y}(P(X,Z) * S(Z,Y)).$$

У наведеному виразі необхідно зробити заміну: підставити замість відношення $P$ його вираз, отриманий вище. У результаті остаточний вираз для $Q$ описуватиметься тільки в термінах ЕБД-відношень $R$ і $S$.

**Твердження 12.2**

Алгоритм 12.2 дозволяє коректно обчислити відношення для кожного з предикатів, а вирази, які за цим алгоритмом будуються для кожного з ІБД-предикатів, визначають:

- множину фактів для предиката, що можуть бути доведені на основі вмісту екстенсійної бази даних;
- єдину мінімальну модель правил.

## 12.5. Обчислення рекурсивних програм

Алгоритм 12.2 не застосовний до рекурсивних Datalog-програм, оскільки для них неможливо встановити порядок обчислення предикатів. Тобто, якщо в графі залежностей є цикл, то який би предикат з нього ми не обчислювали, він обов'язково міститиме правило, для однієї з підцілей якого вираз ще не відомий.

Проте теоретико-довідний підхід можна застосовувати й до рекурсивних програм, оскільки є можливість виводити певні факти з правил і далі використовувати їх для виведення інших фактів. Якщо ми розпочнемо процес виведення з відношень екстенсійної бази даних і будемо використовувати лише Datalog-правила, то множина всіх виведених фактів буде скінченною і всі вони матимуть вигляд $p(a_1, ..., a_k)$, де $p$ — ІБД-предикат, а $a_1, ..., a_n$ константи з бази даних.

Розглянемо Datalog-програму, де використовуються ЕБД-відношення $R_1, ..., R_k$, а обчислити потрібно ІБД-відношення $P_1, ..., P_m$. Для кожного $i$, $1 \le i \le m$, виразимо множину виведених фактів для предиката $p_i$, якому відповідає ІБД-відношення $P_i$, у вигляді

$$P_i := \text{EVAL}(p_i, R_1, ..., R_k, P_1, ..., P_m).$$

Тут EVAL є об'єднанням EVAL-RULE (визначених за алгоритмом 12.1) для кожного з правил для $p_i$, спроектованих за змінними голови. Якщо розпочати обчислення з порожніх значень для $P_i$ і рекурсивно застосовувати наведену вище формулу для всіх $i$, то ми досягнемо такого моменту, коли в процесі обчислення нові факти до жодного з відношень $P_i$ не будуть додаватися. Отже, наприкінці обчислення символ присвоювання стає рівністю, тобто множина виведених ІБД-фактів для всіх $i$ відповідає рівнянням

$$P_i = \text{EVAL}(p_i, R_1, ..., R_k, P_1, ..., P_m).$$

Рівняння, отримані з Datalog-програми у наведений вище спосіб, називатимемо *Datalog-рівняннями*. Розв'язками цих рівнянь є відношення, кортежі яких задовольняють правила Datalog-програми.

## 12.5.1. Нерухомі точки для Datalog-рівнянь

Отримання Datalog-рівнянь заміною у Datalog-правилах символів :- на символ рівності інтуїтивно виправдується тим, що зміст правил означає не більше й не менше, ніж те, що може бути доведене з використанням правил. Було б добре, якби існував єдиний розв'язок для множини Datalog-рівнянь, однак у загальному випадку таких розв'язків може бути багато. Нехай задано множину відношень для ЕБД-предикатів, наприклад $R_1$, ..., $R_k$. *Нерухомою точкою* Datalog-рівнянь стосовно $R_1$, ..., $R_k$ називається розв'язок, що відповідає ІБД-предикатам цих рівнянь

Нерухома точка рівнянь зі змінними $P_1$, ..., $P_m$ стосовно ЕБД-відношень $R_1$, ..., $R_k$ разом із цими відношеннями є моделлю правил, з якої утворюються Datalog-рівняння. Нехай $M$ — модель, у якій істинні лише ті факти, що відповідають кортежам відношень $P_1$, ..., $P_m$ і $R_1$, ..., $R_k$. Тоді будь-яка заміна змінних на константи, що робить істинним тіло правила, робить істинною й голову цього правила. Якщо голова правила набуває істинного значення $p(a_1, ..., a_n)$, то кортеж $(a_1, ..., a_n)$ має належати відношенню ІБД-предиката $p$, інакше наявне ІБД-відношення не буде нерухомою точкою Datalog-рівнянь.

Однак не кожна модель множини Datalog-правил є нерухомою точкою відповідних Datalog-рівнянь, оскільки модель може визначати надто багато фактів і деякі з них вказуються в лівих і не вказуються в правих частинах правил (далі буде наведений відповідний приклад). З іншого боку, ми будемо продовжувати цікавитися насамперед нерухомими точками й моделями, мінімальними в тому розумінні, що вони не містять строгої підмножини фактів, що також є нерухомою точкою.

Виявляється, що Datalog-програми, які не містять правил із запереченнями, мають єдину мінімальну модель, яка містить задані ЕБД-відношення, і ця модель є єдиною мінімальною нерухомою точкою стосовно ЕБД-відношень відповідних Datalog-рівнянь. Як і в нерекурсивному випадку, найменша нерухома точка збігається з множиною фактів, що можуть бути доведені за допомогою правил програми стосовно заданого стану бази даних.

Нехай $P_1$, ..., $P_m$ є змінними Datalog-рівнянь, що відповідають ІБД-предикатам $p_1$, ..., $p_m$, припустимо також, що задані відношення $R_1$, ..., $R_k$, які відповідають ЕБД-предикатам $r_1$, ..., $r_k$.

Розв'язком, чи нерухомою точкою стосовно ЕБД-відношень $R_1$, ..., $R_k$ є таке присвоєння змінним $P_1$, ..., $P_m$ значень конкретних відношень $P_1'$, ..., $P_m'$, що вони задовольнятимуть Datalog-рівняння. Якщо $S_1 = \{P_1', ..., P_m'\}$ і $S_2 = (P_1'', ..., P_m'')$ є двома розв'язками для заданої множини рівнянь, то ми будемо казати, що $S_1 \leq S_2$, якщо для будь-якого $i$, $1 \leq i \leq m$, відношення $P_i'$ є підмножиною відношення $P_i''$. Тоді $S_0$ є *найменшою нерухомою точкою* множини рівнянь стосовно ЕБД-відношень $R_1$, ..., $R_k$, якщо $S_0 \leq S$ для будь-якого розв'язку $S$. $S_0$ називається *мінімальною нерухомою точкою*, якщо не існує ніякої іншої нерухомої точки $S$, такої, що $S \leq S_0$. Зазначимо, якщо існує найменша нерухома точка, то існує і єдина мінімальна нерухома

точка. Однак може існувати кілька непорівнянних мінімальних нерухомих точок, і в цьому випадку найменша нерухома точка відсутня.

## 12.5.2. Розв'язування рекурсивних Datalog-рівнянь

Припустимо, що всі $P_i$ є порожніми, а $R_i$ задані як ЕБД-відношення. Для одержання нових значень ІБД-відношень застосуємо процедуру EVAL до поточних значень ІБД-відношень і постійних значень ЕБД-відношень. Даний процес повторюватимемо доти, доки хоча б одне $P_i$ змінюється. Послідовність ІБД-відношень має збігатися, оскільки EVAL є монотонною операцією. Тобто, якщо до якогось із аргументів операції додаються кортежі. кількість кортежів у відношенні-результаті не зменшується.

Формалізуємо алгоритм розв'язування рекурсивних Datalog-рівнянь.

**Алгоритм 12.3**

**Вхід.** Сукупність Datalog-правил з ЕБД-предикатами $r_1$, ..., $r_k$ та ІБД-предикатами $p_1$, ..., $p_m$, а також список відношень $R_1$, ..., $R_k$, що є значеннями ЕБД-предикатів.

**Вихід.** Розв'язок, що є найменшою нерухомою точкою для Datalog-рівнянь.

**Метод.** Перетворимо правила на рівняння зі змінними $P_1$, ..., $P_m$, що відповідають ІБД-предикатам. Рівняння для $P_i$ має такий вигляд:

$$P_i = \text{EVAL}(p_i, R_1, ..., R_k, P_1, ..., P_m).$$

Ініціалізуємо всі $P_i$ порожніми відношеннями. Для отримання нових значень до всіх $P_i$ застосуємо процедуру EVAL. Цей процес будемо повторювати доти, доки хоча б одне $P_i$ змінюється. Отриманий результат і буде шуканим. Нижче алгоритм записаний у вигляді програми.

```
for i := 1 to m do
    P_i := ∅;
  repeat
    for i := 1 to m do
      Q_i := P_i                          /* збереження старих значень P_i */
    for i := 1 to m do
      P_i := EVAL(p_i, R_1, ..., R_k, Q_1, ..., Q_m);
  until P_i = Q_i, 1 ≤ i ≤ m;
for i := 1 to m do
  output P_i
```

## 12.5.4. Монотонність

Для того щоб довести збіжність алгоритму 12.3 у загальному випадку та його збіжність до найменшої нерухомої точки, якщо вона існує, необхідно показати, що багаторазове застосування процедури EVAL приводить до одержання для кожного ІБД-предиката послідовності відношень, кардинальність яких до певного

моменту збільшується, а згодом перестає зростати й залишається фіксованою. Інакше кажучи, слід показати, що процедурі EVAL відповідає монотонна функція. Формалізуємо це поняття.

Нехай $f(P_1, ..., P_m)$ – функція, аргументи і значення якої є відношеннями. Нехай $S_1 = \{P_1', ..., P_m'\}$ і $S_2 = \{P_1'', ..., P_m''\}$ є двома наборами відношень для реляційних аргументів функції $f$. Припустимо, що $S_1 \leq S_2$, тобто для будь-якого $i$, $1 \leq i \leq m$, $P_i'$ є підмножиною (не обов'язково строгою) $P_i''$. Тоді $f$ буде монотонною, якщо з $S_1 \leq S_2$ випливає $f(S_1) \leq f(S_2)$.

Монотонні функції широко застосовуються в теорії реляційних баз даних, оскільки всі основні операції реляційної алгебри, за винятком різниці, є монотонними. Зокрема монотонними операціями є природне з'єднання, селекція і проекція, а оскільки суперпозиція монотонних операцій буде монотонною, відтак і процедура EVAL є монотонною. Таким чином, за алгоритмом 12.3 визначається найменша нерухома точка для рівнянь, до яких він застосовується.

## 12.6. Обчислення правил із запереченнями

Часто виникають ситуації, коли для вираження певних типів зв'язків у логічних правилах необхідно використовувати заперечення. Правила, у підцілях яких містяться негативні літерали, не є хорнівськими диз'юнктами, але багато з викладених вище ідей застосовні до цього ширшого класу правил. Зрозуміло, що в загальному випадку, інтерпретуючи правила з однією чи більше підцілями, які є запереченнями, слід визначити доповнення до відношень, які відповідають підцілям, що заперечуються, і обчислити відношення правила так, як це робиться в алгоритмі 12.1.

Щоб обчислити доповнення, необхідно знати домени для атрибутів, стосовно яких береться доповнення. Саме тому в реляційній алгебрі застосовується операція різниці множин, а не доповнення. Однак навіть якщо ми визначимо універсальну множину всіх можливих кортежів, стосовно якої ми будемо визначати доповнення відношення, то все-таки ми будемо стикатися із ситуаціями, коли доповнення виявлятиметься нескінченним відношенням. У цьому випадку ми не зможемо застосовувати операції типу селекції чи з'єднання до доповнення відношення і не зможемо застосувати алгоритм 12.1 до правил із запереченнями безпосередньо.

Інтерпретуючи правило з підцілями, що заперечуються, потрібно відповісти на питання, чи належать змінні, які входять до складу таких підцілей, до інших, звичайних (тобто не вбудованих) підцілей без заперечень. У наведеному далі прикладі ми покажемо, що коли на поставлене вище питання дається позитивна відповідь, то існує прийнятна інтерпретація правила, а також продемонструємо, в чому полягає суть проблеми, коли змінні належать лише підцілям, що заперечуються. Далі буде висвітлено ще одну проблему, пов'язану з підцілями, що заперечуються, а саме можливу відсутність найменшої нерухомої точки для логічної програми. Оскільки немає механізмів доведення фактів, що заперечуються, то теоретико-довідним підходом в цьому випадку скористатися неможливо, і ми спробуємо визначити процедуру вибору такої моделі логічної програми, що буде інтерпретувати її зміст.

**Приклад 12.4**

Розглянемо таке правило:

```
самотній-чоловік(X) :- чоловік(X) & ~одружений(X,Y)
```

Припустимо, що чоловік є ЕБД-предикатом з очевидним змістом, а одружений — також ЕБД-предикат, який означає, що $X$ є чоловіком $Y$.

Однією з інтерпретацій даного правила є твердження, що $X$ є самотнім, якщо він чоловік, і не існує такої особи $Y$, з якою одружений $X$. Однак, якщо ми обчислимо відношення для правила шляхом з'єднання відношення чоловік(X) з відношенням, що є доповненням до відношення одружений, тобто з множиною таких пар $(X,Y)$, що $X$ не одружений з $Y$ і виконаємо проекцію за атрибутом $X$, то виявиться, що до самотніх належать ті й тільки ті чоловіки, які одружені відразу з усіма особами з універсума.

Для того щоб уникнути такої невідповідності інтерпретацій, необхідно заборонити використання змінних у підцілях, що заперечуються, якщо ці змінні не згадуються в інших звичайних підцілях без заперечень. Ця вимога не обмежує виразові можливості мови, оскільки завжди можна переписати правило без використання подібних змінних (за умови, що ми приймаємо таку інтерпретацію доповнення для виразу $\sim q(X_1, ..., X_n)$: не існує таких значень для змінних, що використовуються лише в підцілях із запереченнями, за підстановки яких вираз $q(X_1, ..., X_n)$ був би істинним). Щоб коректно означити поняття самотнього чоловіка, приклад 12.4 необхідно переписати так:

```
одружений-чоловік(X) :- одружений(X,Y);
самотній-чоловік(X) :- чоловік(X) & ~одружений-чоловік(X)
```

## 12.6.1. Множинність мінімальних нерухомих точок

Зображення заперечень за допомогою операції різниці відношень вирішує не всі потенційні проблеми, пов'язані з підцілями, що заперечуються. За наявності правил із запереченнями може виявитися, що не існує найменшої нерухомої точки, хоча існує кілька мінімальних нерухомих точок. У зв'язку з цим постає питання, яким є зміст логічної програми за відсутності єдиної найменшої нерухомої точки.

**Приклад 12.5**

Розглянемо правила:

```
p(X) :- r(X) & ~q(X)        (1)
q(x) .- r(X) & ~p(X)        (2)
```

Нехай $P$, $Q$, $R$ є відношеннями для ІБД-предикатів $p$, $q$ і ЕБД-предиката $r$. Припустимо, що $R$ складається з єдиного кортежу, наприклад $R = \{1\}$. Розглянемо два розв'язки даної логічної програми:

$$S_1 = \{P = \varnothing, Q = \{1\}\};$$
$$S_2 = \{P = \{1\}, Q = \varnothing\}.$$

Легко переконатися, що $S_1$ та $S_2$ є розв'язками рівнянь $P = R - Q$ і $Q = R - P$ Правила (1) і (2) логічно еквівалентні, однак вказані два реляційні рівняння, що їм відповідають, не є еквівалентними між собою: можна визначити такі множини $P$, $Q$, $R$, що вони задовольнятимуть одне з рівнянь, але не інше. Такі розбіжності між логічно еквівалентними формулами, що виявляються під час перетворення логічних правил на обчислювані вирази, не є наслідком помилковості моделі. Вони вимагають розробки способів інтерпретації широкого класу логічних правил із запереченнями.

Для визначеної вище пари розв'язків не виконується як нерівність $S_1 \le S_2$, так і нерівність $S_2 \le S_1$. Крім того, не існує такого розв'язку $S$, відмінного від $S_1$ та $S_2$, що $S \le S_1$ чи $S \le S_2$, адже для такого $S$ має виконуватися $P = \varnothing$ і $Q = \varnothing$, але тоді $S$ не буде розв'язком. Отже, $S_1$ та $S_2$ є мінімальними нерухомими точками для правил із прикладу 12.5, які не мають найменшої нерухомої точки.

## 12.6.2. Стратифіковані заперечення

Щоб вирішити проблему заперечень за наявності багатьох мінімальних нерухомих точок, ми будемо допускати наявність лише стратифікованих заперечень.

Правило, голова якого містить предикат $p$, а тіло – підціль $q$, що заперечується, є *стратифікованим*, якщо в графі залежностей відсутній шлях з $p$ до $q$ (структура графа залежностей не змінюється з уведенням підцілей, що заперечуються).

Як буде показано в прикладі 12.6, стратифікованість заперечень не гарантує існування найменшої нерухомої точки для множини правил. Однак це обмеження надає можливість вибору серед багатьох мінімальних нерухомих точок такої, яка й буде інтерпретуватися як зміст логічної програми зі стратифікованими запереченнями.

**Приклад 12.6**

Розглянемо такі правила:

```
p(X) :- r(X)                    (1)
p(X) :- p(X)                    (2)
q(X) :- s(X) & ~p(X)            (3)
```

Дана множина правил є стратифікованою. Нехай ЕБД-предикатам $r$ і $s$ відповідають відношення $R$ і $S$, а ІБД-предикатам $p$ і $q$ – відношення $P$ і $Q$. Припустимо, що $R = \{1\}$ і $S = \{1,2\}$ Тоді розв'язками рівнянь $P = P \cup R$ і $Q = S - P$ будуть зокрема такі набори відношень:

$$S_1 = \{P = \{1\}, Q = \{2\}\};$$

$$S_2 = \{P = \{1,2\}, Q = \varnothing\}.$$

Зазначимо, що, якби не було правила (2), перше рівняння набуло б вигляду $P = R$ і розв'язок рівнянь був би єдиним.

Обидва розв'язки $S_1$ і $S_2$ є мінімальними. Хоча для даних правил відсутня найменша нерухома точка, розв'язок $S_1$ можна вважати «природнішим», оскільки кожний із кортежів цього розв'язку може бути отриманий підстановкою відомих

фактів до тіл правил і виведення фактів, що відповідають головам правил. Далі ми побачимо, що правильний вибір способу інтерпретації стратифікованої програми приведе до створення $S_1$, а не $S_2$.

## 12.6.3. Пошук стратифікації

Не кожна логічна програма з запереченнями є стратифікованою, тому корисно мати алгоритм перевірки на стратифікованість. Наведений нижче алгоритм не лише виконує перевірку, але й стратифікує правила, тобто групує предикати в *страти*, що є такими максимальними множинами предикатів, для яких виконуються наведені нижче умови:

- якщо предикат $p$ є головою правила, тіло якого містить предикат із запереченням $q$, то $q$ розташовується в страті нижчого рівня, ніж той, в якому розташовується $p$;
- якщо предикат $p$ є головою правила з тілом, що містить предикат $q$, який не заперечується, то страт для $p$ має той самий або вищий рівень, ніж страт для $q$.

Страти встановлюють порядок, у якому мають обчислюватися відношення для ІБД-предикатів. Дотримуючись цього порядку, ми можемо інтерпретувати підцілі, які заперечуються, у такий спосіб, ніби вони є ЕБД-відношеннями.

Опишемо алгоритм тестування на стратифікованість і пошуки стратифікації.

**Алгоритм 12.4**

**Вхід.** Множина Datalog-правил, можливо з підцілями, що заперечуються.

**Вихід.** Відповідь на питання, чи є правила стратифікованими і, якщо це так, то побудова стратифікації.

**Метод.** Побудова стратифікації здійснюється у п'ять етапів.

1. *stratum*[$p$] := 1 для кожного інтенсійного предиката $p$.
2. Для кожного правила $r$ із предикатом $p$ у голові виконуємо дії, що зазначені в пунктах 3 та 4.
3. Для кожної підцілі з предикатом $q$, що заперечується, виконуємо присвоювання *stratum*[$p$] := max(*stratum*[$p$], 1 *stratum*[$q$] + 1).
4. Для кожної підцілі з предикатом $q$, що не заперечується, виконуємо присвоювання *stratum*[$p$] := max(*stratum*[$p$], *stratum*[$q$] + 1).
5. Якщо немає змін у масиві *stratum*, або хоча б один із елементів *stratum* перевищує кількість предикатів, переходимо до пункту 2, інакше завершуємо алгоритм.

Якщо в правилах відсутні підцілі, що заперечуються, то алгоритм 12.4 зупиниться після першої ітерації, розмістивши всі предикати у страті 1. Якщо логічна програма може бути стратифікована, то алгоритм 12.4 зупиниться під час його застосування до цієї програми без створення страта номер якого перевищує кількість предикатів у програмі.

Загалом алгоритм 12.4 коректно визначає, чи може бути Datalog-програма з запереченнями стратифікована.

## 12.6.4. Безпечність і стратифіковані правила

Для того щоб інтерпретувати правила із запереченнями, необхідна не лише їхня стратифікованість, але й безпечність. У підрозділі 12.3 ми визначили, що правило є безпечним, якщо всі його змінні є обмеженими. Нагадаємо, що змінна є обмеженою, якщо вона входить до складу звичайних підцілей без заперечень або прирівнюється до константи чи іншої обмеженої змінної (можливо, через ланцюжок рівностей). Якщо ми маємо підцілі, що заперечуються, то означення безпечності не змінюється. Тобто змінна, що входить до складу підцілей із запереченням або прирівнюється до таких змінних, не може бути обмеженою.

Коли правила є безпечними і стратифікованими, то існує природний спосіб вибору з множини можливих нерухомих точок такої, яка і буде розглядатися як зміст правил. Для цього будемо обробляти страти відповідно до їхнього упорядкування, починаючи зі страта найнижчого рівня.

Припустимо, що ми обробляємо предикат $p$ зі страта $i$. Якщо правило для $p$ містить підціль із предикатом $q$, що належить страту нижчого рівня, ніж $i$, то ми вже маємо відношення для $q$. Це відношення є ЕБД-відношенням або має бути обчислене під час обробки страта нижчого рівня, ніж $i$. Жодні підцілі предикатів рівня $i$ не можуть належати стратам рівня вище $i$, а якщо підціль заперечується, то її предикат належить страту нижчого рівня, ніж $i$.

Наслідком такої властивості стратифікації є те, що правила для предикатів страта $i$ є рекурсивними означеннями відношень для всіх предикатів страта $i$ у термінах відношень для ЕБД-предикатів і всіх відношень для ІБД-предикатів зі стратів нижчого рівня. Оскільки рівняння для ІБД-предикатів страта $i$ не містять підцілей, що заперечуються, то для їхнього обчислення можна застосувати алгоритм 12.3.

Спосіб створення відношення для підцілі із запереченням $\neg q(X_1, ..., X_n)$ правила $r$ має бути таким, щоб дане відношення було скінченним і відповідало певній підцілі, що не заперечується. Означимо DOM як об'єднання всіх значень, що вказуються в ЕБД-відношеннях, і констант у самих правилах. Якщо правила є безпечними, то жодне значення, що не належить ЕБД-відношенням і не вказується у правилах (тобто не належить DOM), не може з'являтися в такій підстановці констант замість змінних, яка б відповідала істинним значенням тіл цих правил. Тому ми нічого не втратимо, якщо будемо припускати, що відношення підцілі, яка заперечується, складатиметься з кортежів, що містять значення з DOM.

Отже, якщо $Q$ є вже обчисленим відношенням для $q$ (або просто ЕБД-відношенням для цього предиката), то відношення для підцілі $\neg q(X_1, ..., X_n)$ матиме такий вигляд:

$$\underbrace{\text{DOM} \times ... \times \text{DOM}}_{n \text{ разів}} - Q.$$

Якщо виконати аналогічні заміни для всіх підцілей із запереченнями правил зі страта $i$ та застосувати алгоритм 12.3 для обчислення найменшої нерухомої точки для ІБД-предикатів одного страта, то одержимо результат, що буде збігатися з результатом, отриманим шляхом замінення підцілей, що заперечуються, нескінченними відношеннями, наприклад відношенням для предиката $\neg q$, яке містить усі кортежі, що не належать $Q$.

Розглянемо алгоритм обчислення відношень для безпечної стратифікованої Datalog-програми.

**Алгоритм 12.5**

**Вхід.** Datalog-програма, правила якої безпечні, зведені та стратифіковані, а також відношення для ЕБД-предикатів програми

**Вихід.** Відношення для ІБД-предикатів, що визначають разом з ЕБД-відношеннями мінімальну нерухому точку Datalog-програми.

**Метод.** Спочатку стратифікуємо програму за алгоритмом 12.4 і будуємо DOM, проектуючи всі ЕБД-відношення на кожний з їхніх атрибутів, об'єднуючи всі такі проекції, а також константи, що вказуються у правилах. Потім послідовно для кожного страта $i$ (починаючи зі страта найнижчого рівня) виконуємо описані далі дії. Слід врахувати, що коли ми досягнемо страта $i$, то відношення для ІБД-предикатів рівнів, нижчих, ніж $i$, виявляться обчисленими, відтак усі необхідні нам відношення будуть відношеннями для ЕБД-предикатів. Якщо правило зі страта $i$ містить підціль, що заперечується, то відношення самої підцілі вже обчислено.

1. Розглянемо всі підцілі правила зі страта $i$, які не заперечуються. Якщо така підціль є ЕБД-предикатом чи ІБД-предикатом зі страта нижчого рівня, то для таких предикатів використовуємо вже обчислені відношення.
2. Для кожної підцілі правила зі страта $i$, яка заперечується, вже має бути обчислене відношення $Q$ самої підцілі (тобто підцілі без заперечення), оскільки страти для предикатів підцілей, що заперечуються, є нижчого рівня, ніж $i$. Відношення для підцілі з запереченням обчислюємо за формулою DOM $\times \ldots \times$ DOM $- Q$.
3. Використовуємо алгоритм 12.3 для обчислення відношень ІБД-предикатів зі страта $i$. При цьому інтерпретуємо всі підцілі, відношення яких були отримані у пункті 2 або 3, так, ніби вони є ЕБД-відношеннями

## 12.6.5. Найкраща нерухома точка

Нерухома точка, що обчислюється за алгоритмом 12.5, називається *найкращою нерухомою точкою* чи *моделлю*. Алгоритм 12.5 обчислює одну з мінімальних нерухомих точок для множини безпечних стратифікованих правил із запереченнями. Хоча ми й не довели, що результат алгоритму 12.5 є мінімальною нерухомою точкою, той факт, що отримано саме нерухому точку, легко довести методом математичної індукції за рівнем страта.

Найкраща нерухома точка $S$ має такі властивості.

- Якщо $S_1$ – інша нерухома точка, то відношення в $S$ для будь-якого предиката $p$ зі страта 1 є строгою підмножиною його відношення в $S_1$.
- Для всіх $i > 1$, якщо $S_1$ є нерухомою точкою, що збігається з $S$ за відношеннями для предикатів зі стратів рівня менше $i$, то відношення для предикатів страта $i$, що належать $S$, є підмножинами відповідних відношень, які належать $S_1$.

З цих властивостей випливає, що $S$ є мінімальною нерухомою точкою.

## Контрольні запитання та завдання

1. Назвіть три способи визначення змісту дедуктивних правил.
2. У чому полягають основні відмінності мови Datalog від мови Prolog?
3. Що таке екстенсійна та інтенсійна бази даних?
4. Дайте означення атомарної формули.
5. Дайте визначення диз'юнкту і хорнівського диз'юнкту.
6. Що таке рекурсивне правило? Наведіть приклади.
7. Яке правило називається безпечним? Наведіть приклади.
8. Що таке відношення, обумовлене тілом правила? Наведіть алгоритм обчислення відношення для тіла правила через операції реляційної алгебри.
9. Що називається зведеним правилом? Як зводити правила?
10. Наведіть алгоритм обчислення нерекурсивних правил за допомогою операцій реляційної алгебри.
11. Що таке нерухома точка для Datalog-рівнянь?
12. Що таке монотонна функція? Які операції реляційної алгебри є монотонними?
13. У чому полягають проблеми обчислення правил із запереченнями?
14. Що таке стратифіковане заперечення?
15. Наведіть алгоритм пошуку стратифікації
16. Як обчислювати безпечні стратифіковані Datalog-програми?

# Розділ 13

# Бази даних в Інтернеті

- Бази даних на основі XML
- Мови запитів
- Генерація описів DTD зі схеми бази даних і навпаки
- Публікування баз даних в Інтернеті
- Робота з базами даних через мережу Інтернет

## 13.1. Основи XML

Мова XML (eXtensible Markup Language – розширювана мова розмітки) була розроблена і підтримується консорціумом W3C. Вона розроблялася як мова розмітки документів, а не як мова опису баз даних. Розширюваність є головною відмінністю XML від іншої популярної мови розмітки – HTML. Спочатку фахівці вважали, що ця мова замінить HTML як мову публікації веб-документів, проте наявність у мові засобів визначення нових тегів, а також можливість створювати вкладені структури тегів дозволила використовувати XML для зображення даних складної структури, а не тільки документів. У зв'язку з цим мова стала інтенсивно використовуватися в застосуваннях, що здійснюють обмін даними, а не просто як замінник HTML. Завдяки відкритості і розширюваності XML стала основою для нового покоління форматів зображення даних в Інтернеті.

Формати попередніх поколінь базувалися на «плоских» текстах, що складалися з рядків. XML дає змогу описувати структуровані дані, структура яких може бути довільною або фіксуватися за допомогою схем XML-даних. Окрім того, розроблені мови з організації пошуку в документах, записаних мовою XML.

### 13.1.1. Базові поняття XML

#### Елементи

Будь-який XML-документ складається з *елементів*. Елемент починається *тегом* <ім'я тега> і закінчується тегом </ім'я тега>, що називаються *початковим тегом* і *кінцевим тегом* відповідно. Між цими двома тегами розташовується вміст елементу. В елементі можуть міститися інші елементи. Вкладеність елементів має бути коректною, тобто внутрішній елемент повинен закінчуватися раніше зовнішнього. Наприклад, із двох вказаних нижче записів перший є коректним, а другий – ні

```
<факультет>...<кафедра>...</кафедра>...</факультет>
<факультет>...<кафедра>...</факультет>...</кафедра>
```

Окрім інших елементів, елемент може містити текст, наприклад:

```
<кафедра>
  Кафедру було створено в жовтні 2000 року. Першим завідувачем кафедри був професор
  П.І  Іванов.
  <назва_кафедри>комп'ютерних наук</назва_кафедри>
  <завідувач>І.П. Петров</завідувач>
  <фонд>2000.00</фонд>
</кафедра >
```

### Атрибути

Елементи можуть мати *атрибути*, значення яких вказуються всередині початкового тега елементу згідно з таким форматом: `ім'я_атрибута=значення`. Елемент може мати кілька атрибутів, але кожне ім'я атрибута має бути унікальним у межах елементу. Наведемо приклад означення атрибутів:

```
<факультет корпус="5/1" назва="інформатики">
```

Зазначимо, що атрибути і вкладені елементи взаємозамінні. Наприклад, наведена щойно конструкція рівнозначна такій:

```
<факультет>
  <корпус>5/1</корпус>
  <назва>інформатики</назва>
</факультет>
```

Атрибути і вкладені елементи можуть мати різне семантичне навантаження: в контексті документа атрибути ідентифікують елементи й відображують властивості елементу в цілому, а за допомогою вкладених елементів визначається структура документа.

### Порожні елементи

*Порожнім* називається елемент, що не має вмісту, і зокрема не містить інших елементів. Є два еквівалентних способи позначення порожнього елементу: `<ім'я_елементу[атрибути]></ім'я_елементу>`, або `<ім'я_елементу[атрибути]/>` Наприклад:

```
<кафедра назва_кафедри="комп'ютерних_наук" завідувач="І.П. Петров" фонд="2000.00"/>
```

### Розділи CDATA

Щоб символи текстового рядка, який визначає вміст елементу, не розпізнавалися як символи розмітки, вони записуються в так званому CDATA-розділі. Він може розміщуватися всюди, де допускається записувати рядкові дані. CDATA-розділ починається із запису `<![CDATA]` і закінчується символами `]]>`. Всередині можуть вказуватись довільні символи, разом із тими, які використовуються для розмітки

### Посилання на об'єкти

Документи XML можуть містити посилання на різноманітні об'єкти, зокрема на розділи тексту в тому ж самому або в іншому документі. Посилання є рядком, що починається з символу амперсанда (&) і закінчується крапкою з комою. Наприклад, наведені нижче посилання доцільно записати на початку журнальної статті

```
<стаття>
  &вступ;
  &тіло;
  &меню;
  &висновок;
  &ресурси;
</стаття>
```

### Простори імен

XML-дані можуть передаватися з однієї організації до іншої. Імена деяких елементів в різних організаціях можуть інтерпретуватися по-різному, внаслідок чого документ буде розтлумачено некоректно. Використання певного унікального рядка як доповнення до імені елементу допомагає вирішити цю проблему. Такий рядок позначає *простір імен* і вказується перед іменем елементу згідно з таким синтаксисом: `ім'я_простору:ім'я_елементу`. Передбачається, що ім'я `xml` є зарезервованим, і користувач не може його вживати для власних потреб. Наведемо приклад використання простору імен:

```
<NAU:кафедра>
  <NAU:назва_кафедри>комп'ютерних наук<NAU:назва_кафедри>
  <NAU:завідувач>І.П. Петров</NAU:завідувач>
  <NAU:фонд>2000.00</NAU:фонд>
</NAU:кафедра>
```

Зазначимо, що останнім часом почали створюватися служби реєстрації і підтримки просторів імен для різних співтовариств розробників і користувачів XML-ресурсів.

## 13.1.2. Опис структури документа

У схемі бази даних описується, яка саме інформація може бути збережена в базі даних, як вона структурована і якими є типи значень, що зберігаються. У документах XML описувати структуру даних не обов'язково. Проте, щоб XML-дані міг автоматично інтерпретувати той, кому вони призначені, між ним і тим, хто ці дані надсилає, має існувати попередня домовленість щодо способу тлумачення даних. У зв'язку з цим у мові XML передбачена можливість описувати структуру даних у вигляді *схем XML-даних*. Є два механізми (мови) опису таких схем:

- DTD (Document Type Definition – визначення типу документа), який сьогодні використовується дуже широко;
- XML Schema – досить новий механізм, який наразі використовується рідше.

### Мова DTD

Тип XML-документа може бути визначений за допомогою мови DTD, що дає змогу вказати:

- які саме елементи можуть зустрічатися в документі,
- які атрибути може або повинен мати елемент;
- які вкладені елементи може або повинен містити елемент і якою є їхня кількість.

Загальний синтаксис DTD-опису є таким:

```
<!DOCTYPE ім'я_документа [
  <!ELEMENT ім'я_елементу (специфікація_вмісту)>
  <!ATTLIST ім'я_елементу специфікація_атрибутів>
  ...
]>
```

**Специфікація вмісту елементу**

Структура елементу документа XML специфікується означенням типу елементу і списку його атрибутів. Головна мета специфікації вмісту елементу полягає в тому, аби вказати, елементи яких типів можуть бути вкладені до нього. Ця специфікація може містити:

- Список розділених пробілами імен елементів, які може або повинен містити даний елемент. Порядок елементів у переліку не суттєвий.
- Взятий у круглі дужки список імен вкладених елементів, що розділяються комами. Порядок запису імен елементів у списку відповідає порядку запису вкладених елементів у XML-документі.
- Взятий у круглі дужки список імен вкладених елементів, що розділяються символами |. Ці елементи є альтернативними, тобто один з елементів списку може або повинен бути вкладений у елемент, вміст якого специфікується.
- Ключове слово #PCDATA (Parsed Character data), яке вказує, що елемент може містити лише рядки символів, але не вкладені елементи. В ієрархії елементів розділи PCDATA розташовані на найнижчому рівні.
- Ключове слово EMPTY, яке означає, що елемент є порожнім.
- Ключове слово ANY, яке вказує на те, що вміст елементу може бути довільним.

Зазначимо, що власне вміст елементів розглядається в XML лише як рядки символів і не вказується в DTD.

Наведемо приклад специфікації типів елементів:

```
<!ELEMENT кафедра (назва_кафедри завідувач фонд)>
<!ELEMENT назва_кафедри (#PCDATA)>
<!ELEMENT завідувач (#PCDATA)>
<!ELEMENT фонд (#PCDATA)>
```

Тут вказується, що елемент кафедра містить невпорядковану послідовність елементів назва_кафедри, завідувач і фонд. З іншого боку, специфікація

```
<!ELEMENT кафедра ((назва_кафедри, завідувач) фонд)>
```

означає, що вкладений елемент назва_кафедри має передувати вкладеному елементу завідувач.

Наведемо приклад специфікації вмісту елементу з альтернативами.

```
<!ELEMENT університет (факультет | інститут)>
```

Тут вказано, що елемент університет може містити або елемент факультет, або елемент інститут.

Будь-яке ім'я вкладеного елементу, а також будь-який різновид їхнього переліку можуть уточнюватися специфікаторами, що вказуються після імені вкладеного елементу або після їхнього переліку:

- + — не менше одного елементу;
- * — нуль або більше елементів;
- ? — не більше одного елементу

Якщо специфікатор відсутній, то це означає, що вкладений елемент повинен вказуватися точно один раз. Наприклад, специфікація

```
<!ELEMENT університет ((назва, ректор) (факультет | інститут)+ фонд?)>
```

означає, що до складу елементу університет входить точно по одному елементу назва і ректор (до того ж назва має вказуватися раніше ректора) один або більше елементів факультет або інститут, і не більше одного елементу фонд.

Загалом специфікація типів елементів документа, що описує структуру університету, може виглядати так:

```
<!ELEMENT університет ((назва, ректор) (факультет | інститут)+, фонд?)>
<!ELEMENT назва (#PCDATA>)
<!ELEMENT ректор (#PCDATA>)
<!ELEMENT факультет (назва_факультету декан кафедра+ фонд?)>
<!ELEMENT назва_факультету (#PCDATA)>
<!ELEMENT декан (#PCDATA)>
<!ELEMENT фонд (#PCDATA)>
<!ELEMENT кафедра (назва_кафедри завідувач примітки)>
<!ELEMENT назва_кафедри (#PCDATA)>
<!ELEMENT завідувач (#PCDATA)>
<!ELEMENT примітки (ANY)>
<!ELEMENT інститут (назва_інституту, лабораторія?)>
<!ELEMENT інститут назва (#PCDATA>)
<!ELEMENT лабораторія (#PCDATA>)
```

**Специфікація атрибутів**

Атрибут дає можливість зіставити елементу пару «ім'я/значення». Специфікувати атрибути можна лише в початковому тегу елементу. Оголошення списку атрибутів може використовуватися для

- визначення множини атрибутів, допустимих для елементу заданого типу;
- встановлення обмежень на значення атрибутів;
- надання атрибутам значень за замовчуванням.

Як уже зазначалося вище, синтаксис оголошення списку атрибутів є таким:

```
<!ATTLIST ім'я_елементу специфікація_атрибутів>
```

Тут специфікація_атрибутів — список атрибутів, для кожного з яких задається:

- ім'я;
- тип;
- обов'язковість чи необов'язковість атрибута;
- значення за замовчуванням

Атрибут може мати такі типи:

- CDATA – рядковий літерал.
- ID – ідентифікатор. Значення атрибута цього типу має унікально ідентифікувати елемент серед усіх інших елементів XML-документа. Елемент не може мати більше одного атрибута цього типу. Атрибуту типу ID можна надати значення за замовчуванням #IMPLIED або #REQUIRED.
- IDREF – посилання на ідентифікатор. Значення цього атрибута посилається на атрибут-ідентифікатор одного з елементів XML-документа, тобто значення IDREF має відповідати одному з атрибутів ID.
- IDREFS – значенням цього атрибута є список IDREF.

Після оголошення типу атрибута можна вказати специфікатор, який дозволяє визначити, є атрибут обов'язковим чи ні, а також надати йому значення за замовчуванням:

- #REQUIRED – атрибут має бути специфікований у всіх елементах даного типу.
- #IMPLIED – атрибут не є обов'язковим.
- #FIXED значення_атрибута – встановлює, що атрибут може мати лише вказане значення, що є значенням за замовчуванням. При цьому сам атрибут не є обов'язковим.

Опис типу документа з використанням атрибутів ID, IDREF і IDREFS може виглядати так:

```
<!DOCTYPE університет [
  <!ELEMENT факультет(назва_факультету, декан)>
  <!ATTLIST факультет факультет_ID ID #REQUIRED
            кафедри_факультету IDREFS #REQUIRED>
  <!ELEMENT кафедра(назва_кафедри, завідувач)>
  <!ATTLIST кафедра кафедра_ID ID #REQUIRED
            належить_факультету IDREF #REQUIRED>
...
]>
```

Відповідний цьому опису документ може виглядати так:

```
<університет>
  <факультет факультет_ID="IT"
   кафедри_факультету="D1 D2">
    <назва_факультету>інформатики</назва_факультету>
    <декан>Сідоров</декан>
  </факультет>
  <факультет факультет_ID="CS" кафедри_факультету="D3">
    <назва_факультету>Комп'ютерних наук</назва_факультету>
    <декан>Петров</декан>
  </факультет>
  <кафедра кафедра_ID="D1" належить_факультету="IT">
    <назва_кафедри>Технології програмування</назва_кафедри>
    <завідувач>Ігнатов</завідувач>
```

```
  </кафедра>
  <кафедра кафедра_ID="D2" належить_факультету="IT">
    <назва_кафедри>Комп'ютерний контроль</назва_кафедри>
    <завідувач>Іванов</завідувач>
  </кафедра>
  <кафедра кафедра_ID="D3" належить_факультету="CS">
    <назва_кафедри>Бази даних</назва_кафедри>
    <завідувач>Крамаров</завідувач>
  </кафедра>
</університет>
```

Головним недоліком мови DTD є її слабка типізованість, зокрема:

- не розрізняються типи вмісту елементів і значень атрибутів, які можуть бути лише рядками;
- ідентифікатори ID і IDREF не типізуються, а отже. вони можуть посилатися на будь-які елементи, що є в документі.

Зауважимо, що мова DTD стала основою для серії спеціалізованих мов, наприклад мови MathML, призначеної для опису математичних документів, ChemML – для опису документів з хімії, BSML – для опису документів з біології тощо

### Мова XML Schema

XML Schema є більш потужною мовою опису типів XML-документів, якщо порівнювати з DTD. Окрім іншого, ця мова підтримує:

- означення типів елементів і атрибутів, наприклад `integer`, `string` та ін.;
- обмеження діапазонів допустимих значень;
- означення типів користувача:
- простори імен;
- успадкування об'єктів.

Перевагою мови XML Schema є також те, що в ній використовується не спеціалізований синтаксис, а власне XML. Проте ця мова є набагато складнішою, ніж DTD.

### Таблиці стилів XSL

У Веб користуються популярністю також каскадні таблиці стилів (Cascading Style Sheet – CSS). Вони дають змогу змінювати форматування відомих тегів HTML і визначати нові.

CSS можуть застосовуватися і для форматування документів XML, але це не дуже вдалий вибір. Головна перевага XML полягає у тому, що формат документа подається у вигляді деревоподібної структури, аби можна було здійснювати певні маніпуляції над даними. На жаль, CSS не здатні взаємодіяти з деревом і можуть тільки форматувати документи XML «як вони є». Ви можете вивести документ на екран в будь-якому форматі, який вам подобається, але не можете здійснити вибіркове подання його даних без застосування мови сценаріїв. Більш того, для використання CSS вам доведеться вивчити синтаксис ще однієї мови.

Названі вище обмеження привели до створення мови XSL (eXtensible Style Language). Це додаток до XML із своєю власною семантикою (фіксованим набором елементів), отже, він може бути використаний для створення таблиць стилів (шаблонів документів), зрозумілих будь-якій програмі, що працює з XML.

У таблицях стилів XSL описується, як документи XML мають бути відформатовані або перетворені на інші формати, зокрема HTML, RTF, PDF, PostScript, SGML, TeX. Але таблиці стилів XML – це щось більше, ніж просто мова форматування або перетворювач форматів; вона також надає механізм для маніпулювання даними. Наприклад, дані можна сортувати, виконувати за ними пошук, видаляти або додавати безпосередньо з вікна браузера.

#### Об'єктна модель документа

Значний потенціал XML як проміжного програмного забезпечення підсилюється об'єктною моделлю документа (Document Object Model – DOM), версія 1.0 якої була прийнята як рекомендації консорціуму W3C у жовтні 1998 року. DOM визначає логічну структуру документів, способи доступу до їхніх елементів і способи маніпулювання елементами документів.

DOM ніяк не впливає на написання XML- і HTML-документів. Замість означення набору структур даних вона зображує документи відповідно до об'єктної моделі, такої, як деревоподібна структура, що складається з вузлів. Недоцільно використовувати DOM просто для перегляду XML-документів за допомогою браузера. DOM використовується в тих випадках, коли потрібно змінити XML-документ або звернутися до його даних. На сервері DOM може застосовуватися для обробки отриманих від клієнта XML-файлів. Окрім того, DOM може використовуватися програмістами як проміжний рівень для перетворення даних з формату бази даних на формат XML. Якщо інтерфейси DOM правильно реалізовані, користувачам не потрібно буде знати, що дані зберігаються в якому-небудь іншому форматі, а не в XML.

### 13.1.3. Мови запитів і перетворення XML-даних

За допомогою мов цього класу можна:

- перетворювати документи, подані в одній XML-схемі, на іншу, а також XML на HTML і навпаки;
- формулювати запити до XML-документів.

Дві названі можливості тісно взаємопов'язані і забезпечуються одними й тими ж засобами.

Широко відомими мовами запитів і перетворень XML-даних є:

- XPath – досить проста мова, основою якої є вирази шляхів доступу;
- XSLT – проста мова, призначена для переведення документів з однієї XML-схеми на іншу, а також для перекладу з XML на HTML;
- XQuery – мова запитів із розвиненими можливостями;
- XML-QL, Quilt, XQL – мови, що базуються на стандарті XQuery.

Мови запитів і перетворень ґрунтуються на *деревоподібній моделі XML-документів*. Це означає, що XML-документ може бути зображений у вигляді деревоподібної структури, у вузлах якої розташовані елементи або атрибути. Дерево будується за такими правилами:

- для кожної вершини елементу нащадками є вершини вкладених елементів або атрибутів даного елементу;
- текстове значення елементу зображується у вигляді вершини, що є нащадком вершини елементу;
- вершини, що є нащадками даної вершини, впорядковуються згідно з їхнім порядком у XML-документі;
- вершини елементів і атрибутів (за винятком кореневої вершини) мають єдину вершину-предка, яка є вершиною елементу;
- коренева вершина має єдину вершину-нащадка, яка є кореневим елементом документа.

## 13.2. Бази даних на основі XML

На сучасному етапі розвитку Інтернету ще не створено спеціалізованих систем керування базами даних. Це обумовлено тим, що мережа Інтернет ще не висунула таких вимог до систем баз даних, які б виходили за межі можливостей наявних СКБД. Відтак усі інтернет-застосування, які передбачають зберігання й обробку великих обсягів складноструктурованих даних, використовують СКБД загального призначення

Завжди була гострою проблема зберігання і передавання Інтернетом не лише документів довільної форми, але й даних довільної структури. У зв'язку з цим інтенсивно розроблялися стандарти передавання даних і мови, що їх підтримують. У результаті виникла мова XML, що може використовуватись як мова опису даних, що передаються Інтернетом, а також як мова зображення баз даних в Інтернеті.

Є два способи інтеграції інтернет-технологій з технологіями СКБД: публікація баз даних в Інтернеті та робота інтернет-застосувань з базами даних наявних СКБД. Публікація баз даних здійснюється за допомогою спеціальних засобів наявних СКБД, що вибирають підмножини даних з баз і відображують їх у форматах, поширених в Інтернеті (як правило, це формати HTML і XML). Про такий підхід йтиметься в підрозділі 13.3. Інший спосіб інтеграції баз даних з мережею Інтернет – розроблення інтернет-застосувань, що працюють з наявними СКБД, – стисло аналізується в підрозділі 13.4.

### 13.2.1. Риси баз даних в технології XML

Перед обговоренням баз даних на основі XML необхідно відповісти на питання: «Чи є XML базою даних?» Якщо чітко дотримуватися означення бази даних, то XML базою даних не є. Хоча XML-документ містить структуровані дані, без додаткового програмного забезпечення, що призначене для обробки цих даних, він є базою даних не більше, ніж будь-який інший текстовий файл.

Якщо під XML мати на увазі не лише XML-документи, але й технології, за допомогою яких здійснюється їхня обробка, відповідь буде «так, але». «Так», тому що технологія XML має багато характерних рис систем баз даних, насамперед це стосується багаторівневого структурованого зображення XML-даних. Але в середовищі XML принцип багаторівневості втілюється недостатньо строго. У XML-документі співіснують аспекти фізичного й логічного зображення даних. І це не дає змоги реалізувати найважливішу ідею технологій баз даних – принцип незалежності даних.

Як і в системах баз даних, у стандартах платформи XML йдеться про модель даних для зображення інформаційних ресурсів.

У середовищі XML є аналог схеми бази даних, роль якого відіграє опис DTD (Document Type Definition Означення типу документа) або опис XML-документів відповідно до стандарту XML Schema. Можна вести мову також про аналог концептуальної схеми бази даних, роль якої відіграє RDF-специфікація. Нарешті, як і в системах баз даних, на платформі XML розробляються мови запитів (XQuery, XPath, XQL, XML-QL, QUILT та інші).

У контексті баз даних на основі XML важливо звернути увагу на розроблений консорціумом W3C стандарт Document Object Model (DOM) об'єктної моделі для XML-документів, який визначає функції інтерфейсу прикладного програмування для систем, що підтримують інформаційні ресурси XML.

Проте сумніви в тому, що технологія XML доведена до рівня бази даних залишаються, оскільки багато функцій, властивих класичним базам даних, пій технології не притаманні. А саме, йдеться про:

- ефективне збереження даних;
- індексацію;
- централізоване керування;
- гарантування безпеки інформації;
- обробку транзакцій;
- підтримання цілісності даних;
- спільне використання інформації;
- тригери;
- запити до багатьох документів.

XML може використовуватись як база даних у середовищах, де не оброблюються великі обсяги інформації, немає великої кількості користувачів і відсутня потреба у високій продуктивності. Проте вирішити завдання, що передбачають підтримку великої кількості користувачів та дотримання жорстких вимог до цілісності даних і продуктивності, XML не зможе.

## 13.2.2. Дані, документи і бази даних

Існують два основні типи документів XML: *документи, орієнтовані на зображення даних*, і *документи, орієнтовані на зображення документів*. Для роботи з доку-

ментами цих типів призначені *засоби обробки даних* і *засоби обробки документів* відповідно.

Документи, орієнтовані на зображення даних, характеризуються чіткою й формалізованою структурою, частим повторенням однакових елементів даних, порядок вказування котрих, як правило, несуттєвий. Прикладом можуть бути XML-документи, що містять дані про замовлення на продаж товарів, графіки польотів, меню ресторанів, протоколи наукових досліджень тощо. Документи, орієнтовані на дані, зазвичай створюються для комп'ютерної обробки; можливості XML використовуються тут мінімально — він відіграє роль транспортного засобу для даних.

Документ, орієнтований на дані, може виглядати так, як описано у наведеному нижче лістингу.

```
<SalesOrder SDNumber="12345">
  <Customer CustNumber="543">
    <CustName>Компанія АБВ</CustName>
    <Street>Хрещатик, 1</Street>
    <City>Київ</City>
    <Zip>01001</Zip>
    <PostCode>60609</PostCode>
  </Customer>
  <OrderDate>14082005</OrderDate>
  <Item ItemNumber="1">
    <Part PartNumber="123">
      <Description>
        <p><b>Гайковий ключ:</b><br/> нержавіюча сталь, виливок, гарантія 2 роки.</p>
      </Description>
      <Price>9.95</Price>
    </Part>
    <Quantity>10</Quantity>
  </Item>
  <Item ItemNumber="2">
    <Part PartNumber="456">
      <Description>
        <p><b>Сепаратор:<b><br/> алюміній, гарантія 1 рік.</p>
      </Description>
      <Price>13.27</Price>
    </Part>
    <Quantity>5</Quantity>
  </Item>
</SalesOrder>
```

Багато XML документів, орієнтованих на дані, містять великі обсяги текстової інформації. Наприклад, веб-сторінка книги містить переважно текст, але її структура може визначатися засобами мови XML.

Веб-сайт може бути побудований на базі простого, орієнтованого на дані документа XML, що містить інформацію, відповідну кожній сторінці, яка видобувається з бази даних, а також таблиць стилів XSL, призначених для форматування тексту. Будь-який веб-сайт, що на ньому динамічно створюються документи HTML шляхом заповнення шаблону даними з бази, може бути замінений орієнтованими на дані XML-документами й однією або кількома таблицями стилів XSL.

Як приклад можна розглянути наведений далі документ, що містить інформацію про рейсові польоти.

```
<FlightInfo>
  <Airline>Авіалінії України</Airline> здійснюють на тиждень
  <Count>три</Count> безпосадкові польоти з
  <Origin>Києва</Origin> у <Destination>Нью-Йорк</Destination>.
  Час відправлення в <Day>понеділок</Day> о
  <Departure>09:15</Departure>, у <Day>середу</Day> о
  <Departure>11:15</Departure> та в <Day>п'ятницю</Day> о
  <Departure>13:15</Departure>. Час прибуття – через 9 годин
  після відправлення
</FlightInfo>
```

Дані про рейсові польоти можна зберігати й у такому XML-документі

```
<Flights>
  <Airline>Авіалінії України</Airline>
  <Origin>Київ</Origin>
  <Destination>Нью-Йорк</Destination>
  <Flight>
    <Day>понеділок</Day>
    <Departure>09:15</Departure>
    <Arrival>18:15</Arrival>
  </Flight>
  <Flight>
    <Day>середа</Day>
    <Departure>11:15</Departure>
    <Arrival>20:15</Arrival>
  </Flight>
  <Flight>
    <Day>п'ятниця</Day>
    <Departure>13:15</Departure>
    <Arrival>22:15</Arrival>
  </Flight>
</Flights>
```

Документи, орієнтовані на зображення документів, не мають жорсткої структури, не вирізняються глибиною деталізації даних (мінімальний незалежний елемент даних розміщується на рівні змішаного інформаційного повідомлення або всього документа). Крім того, вони характеризуються великим обсягом змішаного інформаційного наповнення. Порядок, у якому згадуються однорідні елементи, майже завжди важливий. До цього класу документів належать книги, повідомлення електронної пошти, рекламні оголошення і майже всі документи у форматі XHTML. Подібні документи призначені, як правило, для читання людьми, а не для комп'ютерної обробки.

Наведемо приклад документа, орієнтованого на зображення документів.

```
<Product>
  <Intro>
    <ProductName>Гайковий ключ</ProductName> фірми
    <Developer>Київський завод металовиробів</Developer> подібний
```

```
<Summary>розвідному гайковому ключу меншого розміру.</Summary>
  </Intro>
  <Description>
  <Para>Гайкові ключі <i>двох типів</i> виготовляються з <b>високоякісної нержавіючої
   сталі</b>. Ручка має гумовий наконечник. Допустиме налаштування ширини розтискання.</Para>
<Para>Ви можете:</Para>
    <List>
      <Item><Link URL="Order.html">Замовити гайковий ключ</Link></Item>
      <Item><Link URL="Wrenches.htm">Отримати додаткову інформацію</Link></Item>
      <Item><Link URL="Catalog.zip">Завантажити каталог</Link></Item>
    </List>
<Para>Ключ буде коштувати для Вас <b>лише 19.99 гривень</b> і якщо Ви замовите його
 зараз, то отримаєте<b>молоток для делікатної роботи</b> у подарунок.</Para>
  </Description>
</Product>
```

На практиці не завжди можна чітко розмежувати різні типи документів. Наприклад, документи, орієнтовані на зображення даних (такі як рахунок-фактура), можуть містити неструктуровані дані з незначною деталізацією (наприклад, опис товарів). Документи, орієнтовані на зображення документів (наприклад, посібник користувача), можуть містити дані з чіткою структурою (назва посібника, дані про організацію, в якій написано посібник, його авторів, ключові слова, УДК, дату видання). Охарактеризувавши документи як орієнтовані на дані або на документи, легше зрозуміти, з чим доведеться мати справу — з даними або документами, відтак значно полегшиться вирішення питання вибору системи для їхнього збереження й обробки.

### 13.2.3. Бази даних з дворівневим доступом на основі XML

Бази даних на основі XML можуть надавати ієрархічний дворівневий доступ до даних. Складовими ієрархії є і структура каталогів файлової системи, і структура елементів документа XML.

Дворівневий доступ — це найпростіший механізм зберігання даних. Переваги БД з дворівневим доступом полягають у тому, що вони невеликі, зручні у створенні та супроводі, а також забезпечують доступ до даних за допомогою широкого набору інструментальних засобів. Недоліком такого підходу є те, що зовсім не підтримуються або підтримуються недостатньо основні функції БД, а саме: захист даних та маніпулювання ними, організація пошуку, відновлення, керування доступом, цілісність тощо. Дворівневим доступом варто користуватися в тому випадку, коли ви маєте справу з простими слабко структурованими документами і застосування специфічних функцій СКБД не є суттєвим.

Під час розробки БД з дворівневим доступом слід чітко визначити, які компоненти документів будуть зображені у вигляді структури каталогів операційної системи, а які — у вигляді структури документа XML. Граничним є варіант, коли вся інформація зображена у вигляді структури каталогів, а файли містять лише окремі найпростіші фрагменти документів. Йдеться про БД з дворівневим доступом

*з високим ступенем деталізації даних*, оскільки дані в ній зображені у вигляді численних дрібних фрагментів (рис. 13.1).

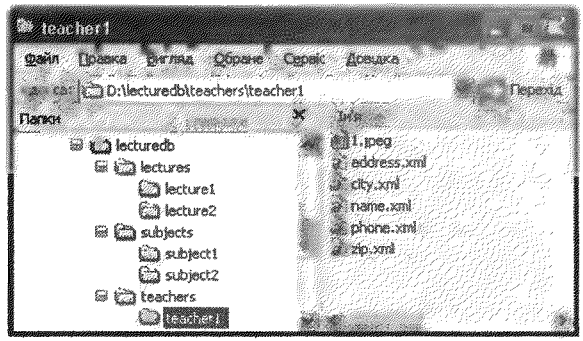



**Рис. 13.1.** Дворівнева база даних з високим ступенем деталізації даних

Іншою крайністю є створення єдиного документа XML, що містить усю базу даних. Це так звана база даних *з низьким ступенем деталізації даних*. З точки зору операційної системи дані в ній не структуровані. Проте до таких даних можна легко звертатися за допомогою текстових редакторів, інструментальних засобів XML та будь-якої мови програмування.

Наведемо приклад бази даних з низьким ступенем деталізації даних

```
<?xml version="1.0"?>
<lecturedb>
  <teachers>
    <teacher id="1">
      <name>Reznichenko</name>
      <address>Teremkovskaya street, 1</address>
      <city>Kiev</city>
      <zip>03187</zip>
      <phone>2665139</phone>
      <picture img="1.jpeg"/>
    </teacher>
  </teachers>
  <subjects>
    <subject id="1">
      <name>databases</name>
      <hours>36</hours>
      <session>5</session>
      <description>It is taught to students of the Computer science faculty.
      </description>
    </subject>
    <subject id="2">
      <name>SQL</name>
      <hours>18</hours>
      <session>5</session>
      <description>It is a laboratory work for the database discipline.
      </description>
```

```
    </subject>
  </subjects>
  <lectures>
    <lecture id="1">
      <teacher id="1"/>
      <subject id="2"/>
      <week>1</week>
      <day>monday</day>
      <lesson>3</lesson>
      <room>401</room>
    </lecture>
    <lecture id="2">
      <teacher id="1"/>
      <subject id="2"/>
      <week>2</week>
      <day>friday</day>
      <lesson>1</lesson>
      <room>505</room>
    </lecture>
  </lectures>
</lecturedb>
```

Найбільш зважений підхід полягає у виборі проміжного варіанта, що називається базою даних з *середнім ступенем деталізації даних* (рис. 13.2).

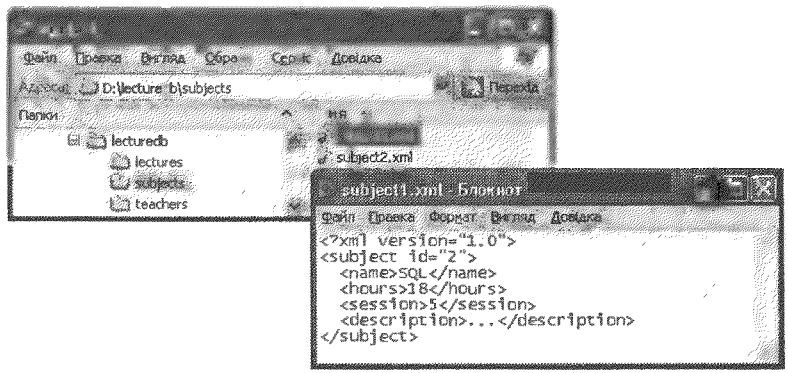



**Рис. 13.2.** Дворівнева база даних з середнім ступенем деталізації даних

Якщо база даних на основі XML поділяється на фрагменти з метою її деталізації, то важливим є питання пошуку так званих *точок поділу*, які можна застосовувати для фрагментації документів не лише під час використання файлових систем, але й у разі збереження документів в об'єктно-орієнтованих і реляційних базах даних.

## 13.3. Бази даних із вбудованою підтримкою XML

*Бази даних із вбудованою підтримкою XML* (native XML databases) розробляються спеціально для збереження і маніпулювання документами XML. Для таких баз

даних існує поняття логічної моделі документа XML, в межах якої документи означуються й зберігаються, а також здійснюється їхній пошук і маніпулювання ними. Модель документа має містити елементи, атрибути, а також описувати впорядкованість даних у документі. БД із вбудованою підтримкою XML не висувають вимог до моделі фізичного зображення документів, тому можуть реалізовуватись як незалежно, так і на основі наявних реляційних, об'єктно-орієнтованих, ієрархічних або інших баз даних. Головне, щоб зовнішній інтерфейс був орієнтований саме на документи XML, а не на модель, яка підтримується СКБД. Системи баз даних із вбудованою підтримкою XML, подібно до інших систем баз даних, забезпечують виконання усіх властивих СКБД функцій, таких як захист даних, підтримка їхньої цілісності, резервного копіювання та відновлення. мов запитів, транзакцій, тригерів, доступу багатьох користувачів тощо.

Використання БД із вбудованою підтримкою XML є найефективнішим у разі збереження й обробки документів XML, орієнтованих на зображення документів. Саме такі БД підтримують упорядкованість усередині документа, коментарі, розділи CDATA, що звичайно не реалізується в СКБД з іншими моделями даних, розширеними для підтримки XML. БД із вбудованою підтримкою XML дають змогу використовувати мови запитів на основі XML й формулювати запити типу «Вивести документи, де третій абзац після початку кожного розділу містить текст, виділений жирним шрифтом». Очевидно, що подібні запити важко формулювати в SQL.

БД із вбудованою підтримкою XML можуть зберігати слабко структуровані дані, даючи при цьому змогу підвищувати швидкість пошуку та обробляти документи, що не мають DTD або інших схем XML (документи без схем). Тобто такі бази даних дозволяють зберігати й обробляти будь-які документи XML, що не містять опису відображень. Під час використання реляційних або об'єктно-орієнтованих баз даних для збереження документів XML необхідно попередньо описати відображення схеми XML на схему бази даних. Відсутність або необов'язковість попереднього настроювання на схему XML у БД із вбудованою підтримкою XML виявляється досить корисною у застосуваннях типу пошукових веб-систем. Пошуковий механізм цих застосувань обробляє документи XML, що можуть мати найрізноманітніші схеми або навіть зовсім не супроводжуватися схемами.

### 13.3.1. Різновиди баз даних із вбудованою підтримкою XML

Є два різновиди БД із вбудованою підтримкою XML:

- орієнтовані на зберігання тексту (текстові XML-БД);
- орієнтовані на модель (модельні XML-БД).

#### Текстові бази даних із вбудованою підтримкою XML

У текстових XML-БД документи XML зберігаються у вигляді текстів. Сховищем таких текстів можуть бути файли операційної системи, поля таблиці реляційної бази даних, що дають змогу зберігати рядкові об'єкти великого розміру, або спеціальні репозиторії текстової інформації.

Характерною ознакою всіх орієнтованих на зберігання тексту БД із вбудованою підтримкою XML є наявність індексів, що дають можливість легко й швидко виконувати пошук у XML-документах. У БД на основі XML цього типу пошук здійснюється одноразовим переглядом індексу. Коли індекс переглянуто, необхідний фрагмент документа можна вибирати одноразовим зчитуванням, якщо цей фрагмент зберігається на диску у вигляді безперервної послідовності байтів. З іншого боку, збирання документа з його складових частин, яке здійснюється в реляційних системах баз даних і в деяких орієнтованих на модель БД із вбудованою підтримкою XML, вимагає багаторазового перегляду індексу і багаторазових звернень до диска.

Текстові XML-БД подібні до ієрархічних БД тим, що обидві значно перевершують реляційну БД за продуктивністю в тому випадку, коли логіка пошуку даних відповідає попередньо визначеній ієрархічній структурі. Якщо логіка пошукового виразу і форма виведення даних не відповідають ієрархічній структурі документа, то текстові XML-БД, як і ієрархічні, у швидкодії значно програють реляційним БД.

### Модельні бази даних із вбудованою підтримкою XML

Іншою категорією є БД, орієнтовані на модельне зображення документів XML. У таких БД документ XML зображується у внутрішній об'єктній моделі й саме в такому вигляді зберігається. Спосіб збереження моделі документів XML залежить від специфіки реалізації XML-БД.

У деяких XML-БД моделі документів зберігаються в реляційних чи об'єктно-орієнтованих базах даних. Наприклад, збереження моделі DOM у реляційній базі даних може привести до побудови таких таблиць, як `Elements`, `Attributes`, `PCDATA`, `Entities` та `EntityReferences`. В інших орієнтованих на модель базах даних із вбудованою підтримкою XML використовуються фізичні структури, спеціально розроблені для оптимального збереження і маніпулювання моделлю документа XML. Модельні XML-БД, побудовані на основі баз даних іншого типу, мають гірші характеристики продуктивності, особливо щодо пошуку даних, аніж ті, які використовують власні, спеціально розроблені під обрану модель, формати збереження даних.

Модельні XML-БД, у яких для збереження даних використовуються спеціальні фізичні структури, характеризуються продуктивністю пошуку даних, порівнянною з продуктивністю текстових XML-БД під час пошуку за ієрархічною структурою. Це пояснюється тим, що більшість модельних XML-БД використовують покажчики для навігації структурою документа. Показники продуктивності текстових і модельних XML-БД залежать від форми подання результатів. Текстові XML-БД набагато швидше виводять документи в текстовому вигляді, а модельні скоріше виводять документи, наприклад, у вигляді DOM-дерев (за умови, що їхня модель документа легко відображується в DOM).

Як і текстові XML-БД, модельні XML-БД працюють менш ефективно під час пошуку й виведення даних у форматах, що відрізняються від того, який обраний для збереження документів.

## 13.3.2. Огляд функцій і можливостей БД із вбудованою підтримкою XML

У цьому розділі ми коротко обговоримо ключові аспекти, що стосуються БД із вбудованою підтримкою XML.

### Колекції документів

У багатьох БД із вбудованою підтримкою XML дані зберігаються у вигляді колекцій. Колекції відіграють ту саму роль, що й таблиці в реляційній базі даних або каталоги у файловій системі. Припустимо, що БД із вбудованою підтримкою XML використовується для зберігання документів, які містять відомості про викладачів вузу. У цьому випадку можна визначити колекцію викладачів для того, щоб запити, які стосуються викладачів, обробляли лише документи цієї колекції.

Інший приклад: необхідно зберігати документацію щодо всіх програмних продуктів, які використовуються на підприємстві. У цьому випадку можна визначити ієрархію колекцій. На першому рівні можна побудувати колекцію із загальною інформацією про програмні продукти, а на другому — підколекції, кожна з яких містить документацію, що стосується програмних продуктів певного типу.

### Мови запитів

БД із вбудованою підтримкою XML мають підтримувати принаймні одну спеціалізовану мову запитів. Найбільш популярними мовами є XPath (з можливим розширенням для побудови запитів за багатьма документами) і XQuery. Деякі БД із вбудованою підтримкою XML оснащені власними спеціалізованими мовами запитів. Вельми ймовірно, що в майбутньому всі БД із вбудованою підтримкою XML будуть підтримувати мову XQuery.

Оновлення й видалення документів

У БД із вбудованою підтримкою XML застосовується багато різних стратегій оновлення і видалення документів. До них належать:

- просте видалення або заміна наявних документів;
- модифікація документів з використанням певної моделі DOM;
- використання спеціалізованих мов з метою модифікації документів або їхніх фрагментів.

Багато з цих методів маніпулювання документами є специфічними для конкретних XML-БД. Проте існують кілька стандартних мов маніпулювання документами XML. Наприклад, мова XUpdate, в якій для ідентифікації вершин використовуються вирази мови XPath, призначена для оновлення XML-документів. Крім того, членами робочої групи W3C XQuery було запропоновано кілька розширень мови XQuery засобами оновлення і видалення документів XML та їхніх фрагментів. Незабаром мова XQuery має бути офіційно розширена можливостями з оновлення документів.

### Транзакції, блокування й одночасний доступ

Практично всі БД із вбудованою підтримкою XML дозволяють використовувати транзакції з можливістю відкочування. Однак блокування здійснюється на рівні

всього документа, а не на рівні окремих вершин у його структурі. У зв'язку з цим багатокористувацький режим підтримується на досить низькому рівні. Чи справді це впливає на ефективність роботи в даному режимі, залежить від конкретного застосування і від того, що розуміти під поняттям «документ». Розглянемо кілька прикладів.

- Документом є розділ технічної документації. Під час написання документації обов'язки між її розробниками були розподілені так, що кожний працював над своїм розділом. Блокування на рівні документа (тобто розділу) навряд чи призведе до виникнення проблем, пов'язаних з одночасним доступом, оскільки малоймовірно, що один і той самий розділ оновлюватиметься кількома авторами водночас.
- Документ містить інформацію про резервування авіаквитків на один рейсовий політ. Великою є ймовірність того, що за наявності багатьох авіакас попереднього резервування і продажу квитків блокування на рівні документа призведе до виникнення проблеми одночасного доступу.
- Документ містить інформацію, що використовується в послідовному технологічному процесі, наприклад під час укладання фінансової угоди. На кожному кроці цього процесу здійснюється зчитування даних із документа і додавання до нього нових даних. Наприклад, один із кроків може полягати в перевірці платоспроможності клієнта і введенні у документ даних про нове кредитування. Інший крок може полягати в перевірці заборгованості клієнта за іншими контрактами і введенні або зміні значення сумарної заборгованості. Якщо блокування здійснюється на рівні вершин структури документа, то багато операцій, подібних до згаданих вище, можуть виконуватися паралельно. Якщо блокування здійснюється на рівні документа, то операції можуть виконуватися лише послідовно. Це може призвести до неприпустимих затримок у роботі застосувань, що інтенсивно використовуються.

Блокування на рівні вершини XML-структури зазвичай вимагає одночасного блокування «батька» цієї вершини, а також «батька» цього «батька» і так до самого кореня. Щоб пояснити, чому потрібне багаторівневе блокування, розглянемо такий приклад. Нехай транзакція виконує оновлення листкової вершини. Якщо не було заблоковано її батьківську вершину, то інша транзакція може в цей час видалити дану батьківську вершину, що, у свою чергу, призведе до видалення листкової вершини, яка оновлюється першою транзакцією. Таким чином, кожного разу під час оновлення вершини слід заблокувати її саму, всі її дочірні вершини та всіх її предків аж до кореневої вершини. Отже, буде заблоковано певний шлях від кореня до листка, а вершини, розташовані на інших шляхах в ієрархічній структурі документа XML, залишаться доступними для інших транзакцій.

### Інтерфейс прикладного програмування

Майже всі БД із вбудованою підтримкою XML надають інтерфейс прикладного програмування. Це, як правило, ODBC-подібний інтерфейс, що містить методи для з'єднання з базою даних, аналізу метаданих, виконання запитів та перегляду результату. Результат виконання запиту може повертатися у вигляді XML-тексту,

дерева DOM або в інших форматах. Якщо запити можуть повертати багато документів, то надаються методи їхнього ітеративного перебирання.

Багато БД із вбудованою підтримкою XML надають можливості виконувати запити і повертати результати з використанням протоколу HTTP.

### Ідентичність логічного документа його оригіналу, що зберігається в базі даних

Характерною ознакою БД із вбудованою підтримкою XML є ідентичність документа, що вибирається з БД, його оригіналу. Це дуже важлива властивість для застосувань, що оперують поняттям документа, і для яких розділи CDATA, використання сутностей, коментарі й команди обробки становлять невід'ємну частину документа.

Ідентичність логічних і збережених документів є менш важливою для застосувань, що орієнтуються на обробку не самих документів, а наявних у них даних. Для таких застосувань у документі XML важливими є: атрибути, текст та ієрархічна структура. У цьому випадку під час передавання даних між документами XML і базами даних важливо зберегти ідентичність не всього документа, а лише зазначених вище складових. В окремих випадках для орієнтованих на дані застосувань важливо зберігати порядок дочірніх елементів і даних PCDATA у межах батьківського елементу. У загальному випадку застосування, що орієнтовані на обробку даних, не підтримують впорядкованість елементів документа, команди обробки, коментарі й фізичні структури зберігання даних (посилання на сутності, розділи CDATA і т. д.), тому їх не можна використовувати для збереження й обробки документів.

Усі БД із вбудованою підтримкою XML можуть підтримувати ідентичність елементів документів, порядку їхнього розташування, атрибутів і даних PCDATA. Більш повне забезпечення ідентичності збережених і логічних документів залежить від реалізації конкретних XML-БД. Як правило, текстові XML-БД підтримують повну ідентичність документів, а модельні XML-БД – ідентичність на рівні моделі документа.

### Індексування

У БД із вбудованою підтримкою XML індекси використовуються для прискорення пошуку. Існують три типи індексів. *Індекси значень* дають можливість індексувати значення атрибутів або тексту. Вони дозволяють здійснювати швидкий пошук за запитами типу «Знайти всі елементи або атрибути, що містять значення `база даних XML`». *Індекси структури* дають можливість індексувати розташування елементів і атрибутів та прискорювати пошук за запитами на кшталт «Знайти всі елементи `<заголовок>`». Разом індекси значень і структури дають змогу відповідати на запити типу «Знайти всі елементи `<заголовок>`, що містять значення `база даних XML`». Нарешті, *повнотекстові індекси* дають можливість індексувати всі слова в тексті та значеннях атрибутів і використовуються при реалізації запитів типу «Знайти документи, що містять усі слова з фрази `база даних XML`», або спільно зі структурними індексами для обробки запитів типу «Знайти документи, що містять усі слова з фрази `база даних XML` в елементах `<заголовок>`». Як правило, в базах

даних із вбудованою підтримкою XML застосовуються індекси значень і структури, а інколи й повнотекстові індекси.

### Збереження зовнішніх об'єктів

Під час збереження документів XML у базі даних принциповим є спосіб обробки зовнішніх об'єктів. Можна вибрати зовнішній об'єкт і зберегти його разом з основним документом, або ж залишити в документі, що зберігається, тільки посилання на зовнішній об'єкт. У кожному окремому випадку кращим може виявитися як перший, так і другий спосіб.

Розглянемо випадок, коли документ містить посилання на CGI-програму, що надає інформацію про погоду на поточний день. Якщо документ, що зберігається, використовується як веб-сторінка з інформацією про сьогоднішню погоду, то було б помилковим, зберігаючи документ, замінити посилання на зовнішній об'єкт його значенням, оскільки в цьому випадку збережена сторінка вже не міститиме реальних даних наступного дня. З іншого боку, якщо документ є частиною колекції хронологічних даних про погоду і якщо він має містити дані про погоду на день його збереження, то буде помилковим зберігати документ разом із посиланням на зовнішній об'єкт, оскільки в цьому випадку документ буде містити прогноз погоди не на той день, коли він був збережений, а на той день, коли до нього звертаються

БД із вбудованою підтримкою XML не можуть самостійно обирати спосіб збереження зовнішніх об'єктів, вони лише надають можливість робити такий вибір у кожному конкретному випадку.

## 13.3.3. Нормалізація у БД із вбудованою підтримкою XML

Головною метою нормалізації є зображення даних у такому вигляді, щоби кожен факт зберігався в базі лише один раз. Нормалізація надає низку переваг, зокрема вона дозволяє економити дисковий простір та підтримувати цілісність даних. Нормалізація є одним з ключових аспектів технології проектування реляційних баз даних і «гарячою» темою в обговореннях щодо запам'ятовування даних у БД із вбудованою підтримкою XML.

Для більшості документів XML, орієнтованих на зображення документів, питання нормалізації не постає так гостро. Наприклад, розглянемо колекцію документів, що описують товари, виготовлені певною компанією. У кожному документі з такої колекції є лише невелика частина даних, спільних для всіх документів: назва, адреса і телефон компанії, її товарний знак, інформація про права власності. Обсяг цих даних є незначним порівняно із загальним обсягом документів, тому можна вважати, що в базі даних одні й ті самі факти повторно майже не відображуються.

Як і в реляційних базах даних, у БД із вбудованою підтримкою XML немає нічого такого, що змушувало б створювати лише строго нормалізовані бази даних. Тобто можна створювати такі ж слабко нормалізовані XML-БД, як і реляційні бази. Тому завдання правильного проектування БД із вбудованою підтримкою XML покладено на плечі проектувальників: перш ніж прийняти певні проектні рішення,

вони мають ретельно вивчити структуру документів і способи їхнього використання. БД із вбудованою підтримкою XML мають одну перевагу в порівнянні з реляційними БД: вони не підтримують схеми бази даних, тому можуть зберігати схожі документи, що мають різні схеми.

Нормалізація даних у БД із вбудованою підтримкою XML багато в чому подібна до їхньої нормалізації в реляційних БД, тобто слід проектувати документи так, щоб вони не містили даних, які повторюються. Одна з відмінностей БД із вбудованою підтримкою XML від реляційних полягає в тому, що XML допускає багатозначність, а реляційна модель — ні. Це дає змогу скористатися такими прийомами нормалізації даних у БД із вбудованою підтримкою XML, які не дозволені в реляційних БД.

Наприклад, розглянемо замовлення на продаж товарів. Воно містить заголовок, що складається з номера замовлення й дати, а також багато рядків, у кожному з яких вказано найменування товару, кількість замовлених одиниць, вартість одиниці й сумарну вартість товару. У реляційній БД заголовок і рядки з відомостями про товар мають зберігатися в різних таблицях, у XML-БД всі ці дані без надлишковості зберігаються в одному документі, оскільки ієрархічна структура документа XML дозволяє одному батьківському елементу мати багато дочірніх.

На жаль, структура реальних документів не завжди є строго ієрархічною. Наприклад, якщо документи замовлень на товари мають містити інформацію про покупця (його ім'я, адресу доставки та телефон), то є два рішення. Можна дублювати інформацію про покупця в кожному з документів, який його стосується. Це призводить до виникнення надлишковості даних і пов'язаних з нею проблем. Можна зберігати інформацію про покупця окремо, використовувати посилання XLink на відповідних покупців у документах на замовлення товарів і виконувати з'єднання двох документів під час вибирання даних. Такий підхід вимагає підтримки в XML-БД XML-механізму XLink або наявності у мові запитів механізму з'єднання документів.

Теорія й практика розробки БД із вбудованою підтримкою XML не дає однозначних відповідей на питання щодо необхідності нормалізації. Навіть реальні реляційні БД часто не «доводяться» до високої нормальної форми задля досягнення необхідної продуктивності, тому наявність недостатньо нормалізованих XML-БД не є настільки суттєвим недоліком, як це може видатися на перший погляд. Проте проектувальнику завжди слід приймати виважене рішення щодо проектування XML-БД.

## 13.3.4. Цілісність посилань у БД із вбудованою підтримкою XML

З поняттям нормалізації тісно пов'язане поняття цілісності посилань, що стосується коректності посилань на взаємозалежні дані та є важливою складовою підтримання цілісного стану бази даних. Цілісність посилань означає, що не можна посилатися на відсутні дані. Для користувачів бази даних не дуже приємною буде ситуація, коли замовлення на продаж товару міститиме номер покупця, який відсутній у списку покупців. У цьому випадку відділ доставки не знатиме, за якою

адресою слід доставити замовлений товар, а бухгалтерія не знатиме, кому надіслати рахунок.

У реляційній БД цілісність посилань гарантує, що значення зовнішнього ключа відношення завжди посилається на існуюче значення первинного ключа цього або іншого відношення. У XML-БД цілісність посилань означає, що покажчики в документах XML вказують на існуючі документи або їхні фрагменти.

Покажчики в документах XML створюються за допомогою різних механізмів: атрибутів ID/IDREF, полів key/keyref (як це визначається в схемах XML), XLink, специфічних для певних мов елементів та атрибутів посилань, наприклад атрибута ref елементу <element> у схемах XML, а також специфічних для тих чи інших XML-БД механізмів посилань. Використання специфічних для мов елементів і атрибутів посилань є розповсюдженою практикою, а специфічні для XML-БД механізми посилань використовуються рідко.

Цілісність посилань у БД із вбудованою підтримкою XML буває двох типів:

- цілісність посилань внутрішніх покажчиків (покажчиків на елементи того ж документа);
- цілісність посилань зовнішніх покажчиків (покажчиків на елементи інших документів).

Цілісність посилань внутрішніх покажчиків, що використовують нестандартні механізми опису, XML-БД не підтримується, оскільки вони не оснащені засобами ідентифікації таких покажчиків. Цілісність посилань внутрішніх покажчиків, що використовують стандартні механізми опису, наприклад, атрибути ID/IDREF або поля key/keyref, підтримується частково. Це пояснюється тим, що більшість БД із вбудованою підтримкою XML здійснюють перевірку цілісності посилань лише під час додавання або оновлення документа. Якщо оновлюється весь документ, то можна легко перевірити коректність внутрішніх посилань. Однак якщо оновлення здійснюється на рівні вершин структури документа, тобто відбувається вставка, заміна або видалення окремих вершин, то слід додатково забезпечити підтримку цілісності посилань внутрішніх покажчиків. Це роблять лише кілька існуючих систем баз даних на основі XML.

Підтримувати цілісність посилань зовнішніх покажчиків значно складніше. Якщо зовнішній покажчик посилається на ресурс, збережений у цій же базі даних, то проблема забезпечення цілісності вирішується. Однак якщо є посилання на ресурс за межами бази даних, наприклад, документ містить посилання XLink, що посилається на документ із зовнішнього веб-сайту, то система не має жодної можливості перевірити, чи насправді цей документ там є. В такому випадку підтримувати цілісність посилання за допомогою одних лише механізмів XML-БД неможливо.

Очевидно, в майбутньому більшість БД із вбудованою підтримкою XML забезпечуватимуть цілісність посилань внутрішніх покажчиків, що використовують стандартні механізми опису. Можна також очікувати, що БД із вбудованою підтримкою XML будуть підтримувати цілісність зовнішніх покажчиків, які посилаються на документи тієї ж бази даних. Сьогодні відповідальність за підтримку цілісності покажчиків, як внутрішніх, так і зовнішніх, у більшості випадків перекладено на застосування, що використовують XML-БД.

## 13.4. XML-БД на основі баз даних іншого типу

Реалізація XML-БД на основі існуючих традиційних систем баз даних (реляційних або об'єктно-орієнтованих) надає низку переваг, оскільки в таких системах реалізовані всі функції, необхідні для роботи з базами даних. XML-БД на основі інших баз даних можна створювати для підтримки обох типів документів XML – орієнтованих на дані та орієнтованих на зображення документів. Однак такий підхід є більш вдалим для реалізації XML-БД, орієнтованих на дані. Тому далі ми звертатимемо увагу переважно на документи цього типу.

### Відображення документів у базах даних

Для того щоб передавати дані з документів XML у бази даних, необхідно вміти відображувати елементи схем документів (DTD, XML Schema, RELAX NG і т. д.) в елементи схем баз даних.

Відображення між документами і базами даних зазвичай виконується для елементів, атрибутів і тексту. У цьому разі опускаються інформація, що стосується фізичної структури (наприклад, означення сутностей, розділи CDATA та інформація щодо кодування), і деякі дані про логічну структуру (наприклад, команди обробки, коментарі, порядок обробки елементів у межах їхнього батьківського елементу).

Це обумовлено насамперед тим, що для бази даних, яка лежить в основі XML-документа, такі відомості не є суттєвими. З точки зору системи баз даних не має значення, зберігаються номери замовлень у документі до чи після дати замовлення, як не має значення й те, чи зберігається ім'я покупця в розділі CDATA як зовнішня сутність, чи безпосередньо в елементі PCDATA. Важливо лише, щоб відповідні дані були передані з документа XML до бази даних.

Одним з наслідків такого відображення є те, що збережений документ не ідентичний оригіналу, тобто зі збереженого в базі даних документа неможливо відновити в точності вихідний XML-документ.

Для відображення схеми документа XML на схему бази даних використовуються, як правило, два методи:

- табличне відображення;
- об'єктно-реляційне відображення.

### Табличне відображення

Табличне відображення застосовується для передавання даних між документами XML і реляційною базою даних. Це відображення моделює документи XML у вигляді однієї або кількох таблиць, тому структура XML-документа має виглядати як у наведеному нижче прикладі.

```
<база даних>
  <таблиця>
    <рядок>
      <стовпець1>...</стовпець1>
```

```
      <стовпець2>...</стовпець2>
      ...
    </рядок>
    <рядок>
      ...
    </рядок>
    ...
  </таблиця>
  <таблиця>
    ...
  </таблиця>
    ...
</база даних>
```

У випадку табличного відображення є можливість зазначати, запам'ятовуються значення з таблиць як дочірні елементи чи як атрибути, а також можна вказувати, які імена використовувати для кожного з елементів чи атрибутів. Можливості такого відображення можуть розширюватися за допомогою метаданих таблиць та/або стовпців. Поняття «таблиця», що використовується в цьому відображенні, не має чіткої інтерпретації. Тобто під час передавання даних із бази в документ XML таблицею може вважатися будь-яка множина даних, отримана в результаті виконання запиту, а під час передавання XML-даних у базу таблиця може зображуватися реальною таблицею бази даних або віртуальною таблицею, що оновлюється.

Табличне відображення є корисним, наприклад, під час передавання даних між двома реляційними базами Очевидним недоліком такого відображення є те, що воно не може використовуватися для документів XML, структура яких не відповідає таблиці.

### Об'єктно-реляційне відображення

Об'єктно-реляційне відображення використовується реляційними та об'єктно-орієнтованими базами даних, що підтримують XML. У цьому випадку дані XML-документа мають відображувати деревоподібну структуру об'єктів. У такій моделі *елементи складних типів*, що містять інші елементи, а також змішані повідомлення, відображуються в класи. *Елементи простих типів*, що містять лише атрибути і дані PCDATA, відображуються у вигляді скалярних атрибутів. Потім модель відображується на реляційну базу даних традиційними методами об'єктно-реляційного відображення або за допомогою об'єктних віртуальних таблиць SQL 3. Тобто класи відображуються в таблиці, скалярні атрибути – у стовпці, атрибути. значеннями яких є об'єкти – в пари «первинний ключ/зовнішній ключ».

Назва цього відображення – «об'єктно-реляційне» – не зовсім коректна, оскільки дерево об'єктів може бути відображене також в об'єктно-орієнтовану або ієрархічну базу даних. Відображення одержало таку назву у зв'язку з тим, що для більшості програмних продуктів, які підтримують цей тип відображення, кінцевими структурами є саме реляційні бази даних

Важливо врахувати, що об'єктна модель, яка використовується за цього відображення, не є об'єктною моделлю документа (DOM). DOM створює модель документа як такого і є однаковим для всіх документів XML, що відповідають заданій схемі. Наприклад, документ замовлень на продаж товарів, наведений у підрозділі 13.1.2, може бути зображений у вигляді дерева об'єктів, що належать чотирьом класам: `SalesOrder`, `Customer`, `Item` і `Part`, зі структурою, що зображена на рис. 13.3.

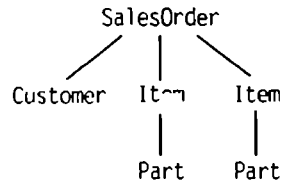



**Рис. 13.3.** Дерево об'єктів для XML-документа

Дерево DOM для цього документа може бути сконструйоване з об'єктів `Element`, `Attr` та `Text` (рис. 13.4).

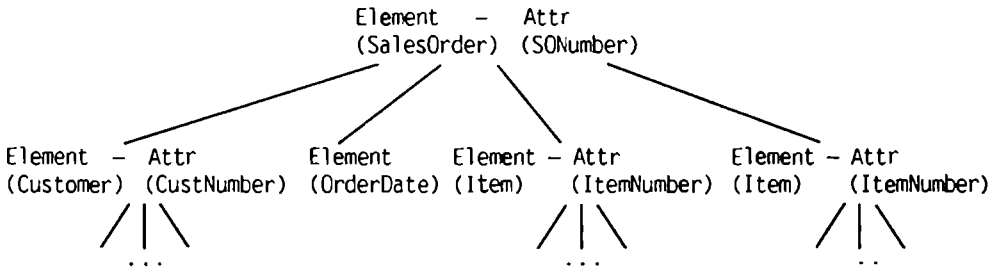



**Рис. 13.4.** Дерево DOM для XML-документа

Чи насправді об'єкти створюються в процесі відображення, залежить від обраного підходу. В одних випадках створюються класи й у процесі відображення дані документів XML спочатку перетворюються на об'єкти цих класів, а потім – на структури бази даних. В інших випадках дані відображуються безпосередньо в базу даних, а об'єктна модель використовується лише як засіб, що полегшує розуміння процесу конвертування. Зв'язування документів XML з об'єктами називається *зв'язуванням даних XML*.

## 13.5. Мови запитів

Один з найпростіших підходів до відображення даних, що зберігаються в базі, полягає в їхній безпосередній інтерпретації відповідно до тієї моделі, за правилами якої вони були введені. Проте структура документа XML, як правило, відрізняється від структури бази даних, тому під час передавання даних з бази в документи XML доцільно скористатися спеціальною проміжною мовою, такою як XSLT – мова перетворення таблиць стилів.

Проблему формування документів на основі даних, що зберігаються в базі, може вирішити реалізація мови запитів, вирази якої повертатимуть документи XML. Як правило, такі вирази базуються на використанні фрази SELECT, вкладеної в шаблони. Очікується, що питання вибору документів з бази даних буде вирішено з реалізацією мов XQuery і SQL/XML. На жаль, майже всі XML-мови запитів мають лише засоби вибору документів (зокрема XQuery 1.0 і перша версія SQL/XML) і в них відсутні засоби вставки, заміни і видалення даних

### Мови запитів на основі шаблонів

Багато мов запитів, що будують XML-документи на основі даних з реляційних баз, використовують шаблони. У цих мовах відсутній будь-який наперед визначений спосіб відображення структури бази даних у структурі документа. Замість цього команди вибирання даних вбудовуються в шаблон, що обробляється проміжним програмним забезпеченням, призначеним для передавання даних. Наприклад у шаблоні, що наводиться далі, елемент <SelectStmt> використовується для включення в нього фрази SELECT, а записи вигляду $ім'я_стовпця визначають, де мають бути розмішені результати застосування шаблону:

```
<?xml version="1.0"?>
  <FlightInfo>
    <Introduction>Є місця на такі рейси:</Introduction>
    <SelectStmt>SELECT Airline, FltNumber  Depart. Arrive FROM Flights</SelectStmt>
    <Flight>
      <Airline>$Airline</Airline>
      <FltNumber>$FltNumber</FltNumber>
      <Depart>$Depart</Depart>
      <Arrive>$Arrive</Arrive>
    </Flight>
    <Conclusion> Ми сподіваємося. Ви знайдете те. що Вам необхідно.</Conclusion>
  </FlightInfo>
```

Після застосування цього шаблону можна отримати такий XML-документ:

```
<?xml version="1.0"?>
  <FlightInfo>
    <Introduction> Є місця на такі рейси:</Introduction>
    <Flights>
      <Flight>
        <Airline>Авіалінії України</Airline>
        <FltNumber>123</FltNumber>
        <Depart>17 серпня 2005: 16:05</Depart>
        <Arrive>18 серпня 2005 01:21</Arrive>
      </Flight>
    </Flights>
    <Conclusion> Ми сподіваємося. Ви знайдете те. що Вам необхідно.</Conclusion>
  </FlightInfo>
```

Мови запитів на основі шаблонів відзначаються високою гнучкістю. Вони надають такі можливості:

- набори значень, що вибираються, можна розмістити в будь-якому місці документа, котрий формується, зокрема їх можна використовувати як параметри послідовних фраз SELECT, не обмежуючись лише форматуванням результатів, як показано в наведеному вище прикладі;
- можна використовувати змінні й означувати функції;
- підтримуються цикли і конструкції if-then;
- підтримується параметризація речень SELECT, наприклад за допомогою параметрів HTTP.

Мови запитів на основі шаблонів використовуються переважно для передавання даних з реляційної бази у XML-документ.

### Мови запитів на основі SQL

У мовах запитів, призначених для формування XML-документів, але розроблених на основі SQL, використовується модифікований варіант фрази SELECT, результат виконання якої перекладається на XML. Є кілька мов, що були розроблені й реалізовані в конкретних XML-БД. У найпростішому випадку в таких мовах використовуються вкладені фрази SELECT, що обробляються згідно з принципами об'єктно-реляційного відображення. Існують також мови, що перетворюють об'єкти SQL 3 безпосередньо на елементи XML.

У 2000 році кілька компаній об'єдналися з метою розробки стандарту, що інтегруватиме XML і SQL. Нині загальновизнаним є стандарт ISO, що одержав назву XML/SQL. У мові XML/SQL введено тип даних XML і реалізовано функції, що дають змогу будувати елементи й атрибути XML на основі даних із реляційних баз. Наприклад, нижче мовою XML/SQL записано запит, результатом якого є таблиця з двох стовпців:

```
SELECT Orders.SONumber,
       XMLELEMENT(NAME "Order",
       XMLATTRIBUTES(Orders.SONumber AS SONumber),
       XMLELEMENT(NAME "Date", Orders.Date),
       XMLELEMENT(NAME "Customer", Orders.Customer)) AS xmldocument
FROM   Orders
```

Перший стовпець містить номер замовлення, другий – документ XML. Кожен рядок містить один документ XML, що створюється з даних відповідного рядка таблиці замовлень. Наприклад, документ для рядка з номером замовлення 123 може мати такий вигляд:

```
<Order SONumber="123">
  <Date>10/29/02</Date>
  <Customer>Gallagher Industries</Customer>
</Order>
```

### Мови запитів на основі XML

На відміну від мов запитів, що використовують шаблони, і мов на основі SQL, які працюють із реляційними базами даних, мови запитів на основі XML можуть застосовуватися до будь-якого документа XML. Для забезпечення взаємодії цих мов із реляційними базами дані мають бути відображені у XML-форматі, що дає змогу формулювати запити до віртуальних документів XML.

У мові XQuery може використовуватись і табличне відображення, і об'єктно-реляційне. Під час використання табличного відображення кожна таблиця розглядається як окремий документ, а з'єднання таблиць (документів) специфікується у самому запиті, подібно до того, як це робиться в SQL. Під час використання об'єктно-реляційного відображення ієрархія таблиць інтерпретується як єдиний документ, причому з'єднання таблиць специфікуються в самому відображенні.

У мові XPath для побудови запитів за багатьма таблицями слід застосовувати об'єктно-реляційне відображення. Це пояснюється тим, що XPath не підтримує з'єднання документів і дозволяє використовувати табличні відображення лише в однотабличних запитах.

## 13.6. Генерація описів DTD зі схеми бази даних і навпаки

Найпоширеніше питання, що виникає під час передавання даних між XML-документами і базою даних, – як створити XML DTD на основі схеми бази даних і навпаки. Слід зазначити, що найчастіше DTD та реляційна схема генеруються одноразово.

Нижче у спрощеному вигляді описано процедуру генерації реляційної схеми з DTD.

- Для кожного елементу, що містить не лише дані PCDATA, створюється таблиця і стовпець з первинним ключем.
- Для змішаного вмісту створюється окрема похідна таблиця, в якій зберігаються дані PCDATA, і яка зв'язана з батьківською таблицею елементу за допомогою зовнішнього ключа.
- Для кожного атрибута з єдиним значенням і кожного вкладеного елементу, який може вказуватися в межах батьківського елементу один раз і містить лише дані PCDATA, створюється стовпець у похідній таблиці. Якщо похідний елемент або атрибут необов'язкові, стовпцю приписується властивість, що дозволяє зберігати NULL-значення.
- Для кожного багатозначного атрибута і кожного вкладеного елементу, що вказується багато разів і містить лише дані PCDATA, створюється окрема таблиця, рядки якої зв'язані з рядками батьківської таблиці за допомогою зовнішнього ключа.
- Для кожного вкладеного елементу та змішаного вмісту створюється зв'язок з батьківською таблицею за допомогою зовнішнього ключа

Далі стисло описано процедуру генерації DTD з реляційної схеми.

- Для кожної таблиці створюється елемент.
- Для кожного стовпця таблиці створюється атрибут або вкладений елемент, що містить лише дані PCDATA.
- Для кожного зв'язку, що моделюється за допомогою первинного й зовнішнього ключа, створюється вкладений елемент.

Ці процедури мають кілька недоліків. Так, на підставі DTD не можна чітко визначити типи даних та розміри полів. Будь-яке припущення, зроблене за результатами аналізу документа-зразка, може бути легко спростоване уміщенням у документ даних іншого типу або іншого розміру. (Рішенням цієї проблеми є використання типів даних у документах схем XML.) З іншого боку, під час генерації DTD з реляційної схеми не можна визначити порядок вкладених елементів або встановити, чи варто взагалі відображувати в XML-документі стовпець (який, наприклад, може бути внутрішнім ідентифікатором в базі даних).

Незважаючи на ці недоліки, наведені алгоритми є базою для процедур генерації DTD з реляційних схем і навпаки.

## 13.7. Публікування баз даних в Інтернеті

Для того щоб опублікувати базу даних в Інтернеті, необхідно на основі даних, які зберігаються в базі, згенерувати HTML-файл і розмістити його в одній із папок на веб-сервері. В такий спосіб користувачі мережі отримають можливість переглядати опубліковані дані за допомогою звичайних браузерів. У подальшому, коли опубліковані дані зміняться в базі, можна буде сформувати цю сторінку ще раз, щоби на ній відображувалась актуальна інформація.

Подібний спосіб публікування даних в Інтернеті є досить простим і зручним під час роботи зі звітами стандартної форми. Його недолік полягає у неможливості отримання з бази довільних даних у довільній формі.

Розглянемо, як генеруються HTML-сторінки в Microsoft SQL Server 2000.

Microsoft SQL Server 2000 оснащений майстром Web Assistant Wizard, що дає змогу створювати стандартні HTML-сторінки, які містять інформацію з бази даних. Тобто він дозволяє ввести в інтерактивному режимі всі необхідні параметри, після чого виконує системну процедуру, яка й генерує HTML-файл.

Створюючи веб-сторінку, слід вказати відомості про базу даних, що публікуватиметься, метод підготовки даних, місце на сервері, де розташовуватиметься сторінка, та розклад публікації даних. Розглянемо детальніше етапи процесу публікування баз даних в Інтернеті за допомогою СКБД Microsoft SQL Server 2000.

- **База даних для публікації.** У SQL Server активними можуть бути багато баз даних. У зв'язку з цим слід вказати ту з них, яка буде джерелом даних.
- **Метод підготовки даних.** SQL Server надає три способи вибору даних з таблиць вказаної бази.
  - **Вибір таблиці вручну.** У цьому випадку явно вказується єдина таблиця, з якої будуть вибиратися дані. Для даної таблиці можна зазначити список

стовпців, що публікуватимуться, а також задати умову відбору рядків. Побудовані в такий спосіб проекція й селекція фіксують множину даних, які публікуватимуться.

- **Вибирання даних за допомогою процедури.** Заздалегідь створюється процедура, яка здійснює вибирання даних з бази. Її ім'я вказується майстру. Якщо процедура має параметри, то можна задати їхні значення.
- **Вибирання даних за допомогою запиту.** У цьому випадку майстер пропонує ввести SQL-запит, що організує вибирання даних, призначених для публікації. Можна ввести більше одного запиту, тоді дані, що повертає кожний запит, публікуватимуться окремо.

- **Місце розташування сторінки, що публікується.** Вказується папка веб-сервера, де має розташовуватися сторінка. Якщо дана папка розташована не на веб-сервері, то кожного разу після створення сторінки її потрібно буде переміщувати в цільову папку вручну.
- **Розклад публікування даних.** Можна вказати, як часто відповідна сторінка буде автоматично публікуватися. Пропонуються такі варіанти:
  - одноразове створення сторінки після завершення роботи майстра;
  - публікування кожного разу, коли відповідні дані змінюються в базі;
  - публікування вручну, шляхом запуску на виконання процедури, що генерує сторінку;
  - одноразове створення сторінки у вказані дату і час;
  - оновлення сторінки відповідно до вказаного часового графіка.
- **Форматування HTML-сторінки.** Надається можливість вказати мінімальний набір властивостей сторінки, що формується. Наприклад, можна вказати кількість рядків, що виводитимуться на одній сторінці.

По завершенні роботи майстер, як мінімум, створить HTML-сторінку, а як максимум, окрім сторінки дасть завдання відповідній службі SQL Server, яка буде автоматично створювати сторінку за заданим графіком з урахуванням параметрів, встановлених користувачем під час роботи майстра.

Після публікування даних доступ до сторінки можна буде отримати за допомогою будь-якого браузера.

## 13.8. Робота з базами даних через мережу Інтернет

У міру зростання популярності Інтернету як базового середовища для різноманітних інформаційних систем висуваються все вищі вимоги до механізмів, що реалізують взаємодію користувачів і застосувань з базами даних через мережу Інтернет. Розробники сучасних СКБД прагнуть задовольнити вимоги користувачів, надаючи різні механізми пошуку й обробки даних в Інтернеті.

Загальна схема доступу до бази даних через Інтернет наведена на рис. 13.5.

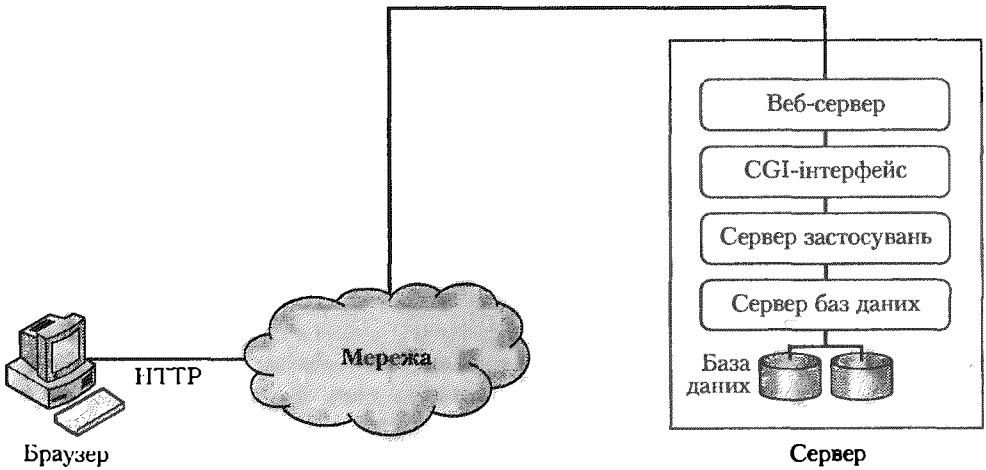



**Рис. 13.5.** Загальна схема доступу до бази даних через Інтернет

Користувач за допомогою браузера формує запит на звернення до бази даних. Цей запит містить:

- ім'я комп'ютера, на якому розташований веб-сервер;
- ім'я CGI-скрипта, якому має бути передано запит для його подальшого передавання СКБД;
- власне текст запиту, що обробляється СКБД.

Запит надходить на вказаний веб-сервер за протоколом передавання гіпертексту HTTP. Сервер, розпізнавши в запиті звернення до CGI-скрипта, надсилає запит йому. CGI (Common Gateway Interface – загальний шлюзовий інтерфейс) специфікує правила взаємодії веб-серверів з прикладними програмами. У нашому випадку CGI-скрипт, прийнявши запит, перетворить його на звернення до бази даних і передасть серверу застосувань, серверу баз даних або СКБД для подальшої обробки. Це може бути запит на оновлення або вибирання даних. Якщо це був вибірковий запит, то отримавши дані з бази, CGI-скрипт передає їх веб-серверу (відповідно до інтерфейсу CGI), який, у свою чергу, пересилає їх браузеру.

Далі ми розглянемо, як доступ до баз даних через Інтернет організовано в пакеті Cold Fusion та в СКБД Microsoft SQL Server 2000.

## 13.8.1. Веб-інтерфейс баз даних у пакеті Cold Fusion

Пакет Cold Fusion для Windows від компанії Allaire призначений для швидкого розроблення інтерактивних динамічних веб-документів. У ньому використовуються такі технології, як HTML, CGI, SQL, ODBC.

Розроблення застосувань з використанням Cold Fusion не вимагає програмування на таких мовах як, наприклад, Perl, C/C++, Visual Basic або Delphi. Технологія, що використовується в Cold Fusion, полягає у вбудовуванні в стандартний HTML-файл спеціальних тегів для роботи з базами даних.

### Шаблони

Файли, які обробляє Cold Fusion, називаються шаблонами. У шаблоні вказується, яке саме звернення до бази даних має бути виконано, і що буде повернено в результаті. Шаблони готуються заздалегідь у вигляді текстових файлів і потім інтерпретуються системою. Звернення до системи з браузера передбачає явне зазначення в URL імені шаблону з можливими параметрами Шаблони містять HTML- і DBML-теги.

- **HTML-теги** використовуються для форматування як фіксованої частини документа, так і результатів запитів. З їхньою допомогою, наприклад, формується загальний вигляд веб-сторінок відповідей на запити.
- **DBML-теги** – це спеціальні теги в мові розмітки DBML системи Cold Fusion, призначені для роботи з базами даних. Вони використовуються для формування запитів до бази даних, а також визначають, де і як будуть відображені результати запитів.

  Основні можливості Cold Fusion реалізуються за допомогою чотирьох тегів:

  - DBQUERY – виконання SQL-запиту до бази даних;
  - DBINSERT і DBUPDATE – створення і модифікація записів у базі даних;
  - DBOUTPUT – відображення результату запиту.

### Загальна схема взаємодії з базами даних

Пакет Cold Fusion дає можливість динамічно генерувати HTML-документи на основі запитів користувача. Ці запити передаються в CGI-скрипт (записаний у файлі dbml.exe), який пересилає їх в ядро Cold Fusion Engine, що обробляє дані відповідно до заданого шаблону і генерує HTML-документ, який надсилається користувачу.

На рис. 13.6. зображено схему обробки запиту в Cold Fusion.

Розглянемо процес обробки запиту користувача детальніше.

1. Коли користувач натискає кнопку Submit у формі, вибирає гіпертекстове посилання в документі або набирає URL із запитом в полі адреси браузера, то браузер надсилає запит на веб-сервер.
2. Веб-сервер, якщо в запиті вказаний DBML-шаблон, запускає процес Cold Fusion, передаючи йому дані, отримані від клієнта.
3. Cold Fusion приймає дані, обробляє теги DBML у шаблоні, зокрема здійснює підготовку запиту до бази даних і форматування сторінки відповіді.
4. Cold Fusion взаємодіє з базою даних через інтерфейс ODBC.
5. Cold Fusion динамічно генерує HTML-документ, що містить результат виконання запиту до бази даних, і повертає його веб-серверу. Cold Fusion може також динамічно генерувати поштове повідомлення і надсилати його через поштовий SMTP-сервер.
6. Веб-сервер повертає веб-клієнту згенерований HTML-документ

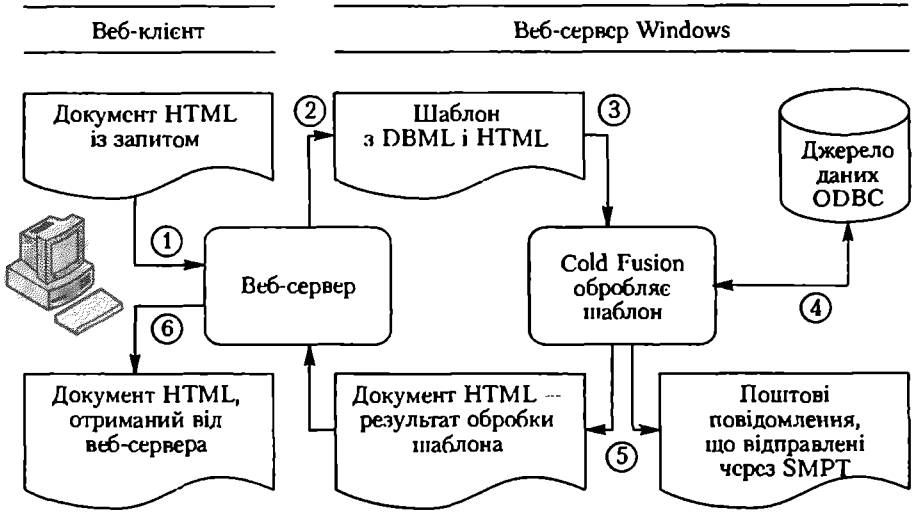



**Рис. 13.6.** Схема обробки запиту клієнта в Cold Fusion

### Передавання параметрів

Під час звернення до CGI-програми має бути означений стандартний параметр Template, що вказує на певний шаблон, наприклад:

```
<FORM ACTION="cgi-shl/dbml.exe?Template=example.dbm">
  <INPUT TYPE="HIDDEN" NAME="user_id" VALUE="5">
  <INPUT TYPE="SUBMIT" VALUE="Enter">
</FORM>
```

Якщо параметри передаються через URL-рядок, вони ініціалізуються після запису адреси шаблону за такою формою: параметр=значення. Запис, що позначає ініціалізацію одного параметра, відділяється від інших таких записів символом &. Наприклад, гіпертекстове посилання, наведене нижче, надсилає параметр з ім'ям user_id і значенням 5 до шаблону example.dbm:

```
<A HREF="cgi-shl/dbml.exe?Template=example.dbm&user_id=5">
```

### Додавання і модифікація даних

Нехай означено шаблон для додавання даних, що має ім'я example.dbm:

```
<DBINSERT DATASOURCE="Persons DB" TABLENAME="Persons"
FORMFIELDS="Name,Phone,Birthday">
<HTML>
  <HEAD><TITLE>Підтвердження</TITLE></HEAD>
  <BODY>
    <H1>Запис створено!</H1>
  </BODY>
</HTML>
```

Тег <DBINSERT> означає, що це шаблон для додавання даних. Його атрибуту DATASOURCE присвоєне ім'я джерела даних ODBC (Persons DB), значення атрибута

TABLENAME вказує на ім'я таблиці (Persons), за допомогою атрибута FORMFIELDS визначається список імен полів, до яких потрібно ввести дані. Значення таких полів будуть параметрами під час звернення до системи через цей шаблон. Звернення можна виконати з браузера, наприклад через таку форму:

```
<HTML>
  <HEAD>
    <TITLE>Приклад введення даних для створення запису</TITLE>
  </HEAD>
  <BODY>
    <FORM ACTION="/cgi-shl/dbml.exe?Template=example.dbm" METHOD="POST">
      Прізвище : <INPUT TYPE="Text" NAME="Name">
      Телефон : <INPUT TYPE="Text" NAME="Phone">
      Дата народження : <INPUT TYPE="Text" NAME="Birthday">
    </FORM>
  </BODY>
</HTML>
```

Відповідно до наведеного вище шаблону буде виконано додавання нового рядка до таблиці Особи із значеннями полів Name, Phone, Birthday, отриманими з форми, а браузеру буде повернено повідомлення Запис створено!.

За необхідності модифікації бази даних використовується тег <DBUPDATE>.

### Вибирання даних і виведення результатів

Для виконання запитів до бази даних використовується тег <DBQUERY>. Припустимо, шаблон prs.dbm містить такий тег <DBQUERY>:

```
<DBQUERY NAME="PrsByID" DATASUORCE="Person DB" SQL="SELECT * FROM Persons WHERE Id=#id#>
```

Тут NAME — ім'я запиту, а SQL — текст запиту, причому запис #id# визначає місце, де має вказуватися значення вихідного параметра. Припустимо також, що браузер надсилає URL

```
/cgi-shl/dbml.exe?Template=prs.dbm&Id=22
```

Тоді до СКБД буде передано такий SQL-вираз:

```
SELECT * FROM Persons WHERE Id = 22
```

Для виведення даних, отриманих в результаті виконання запиту, визначеного в DBQUERY, застосовується тег <DBOUTPUT>. Усередині цього тега, пов'язаного з конкретним запитом, може міститися звичайний текст, теги HTML, а також посилання на поля, визначені в запиті. Під час обробки шаблону вміст тега <DBOUTPUT> надсилається клієнту для кожного запису, що повертається в результаті виконання запиту, з заміщенням відповідних значень параметрів і полів. Загальний вигляд тега <DBOUTPUT> є таким:

```
<DBOUTPUT QUERY="ім'я запиту" MAXROWS=n>
  Текст, теги HTML, посилання на поля і параметри
</DBOUTPUT>
```

Атрибут QUERY застосовується для зазначення імені запиту, результат виконання якого буде використовуватися, а атрибут MAXROWS визначає максимальну кількість записів цього запиту, які будуть передані для виведення.

Наприклад, для виведення результату виконання запиту з іменем PrsByID, що відображує ім'я людини і її телефон, та розділяє записи горизонтальною лінією, може використовуватися така конструкція:

```
<DBOUTPUT QUERY="PrsByID" MAXROWS=50>
  <HR>
  #Name# ( Телефон: #Phone# )
</DBOUTPUT>
```

Результат обробки даного тега може бути таким:

```
<HR>
Іванов Іван Петрович (Телефон: 224-26-73)

<HR>
Петров Петро Ігнатович (Телефон. 444-B2-43)

<HR>
Резніченко Валерій Анатолійович (Телефон: 266-1B-15)
```

### Додаткові можливості

Ми описали лише базові можливості системи Cold Fusion, коротко перелічимо додаткові:

- виведення результатів виконання запиту у вигляді таблиці, означеної мовою HTML,
- використання у шаблонах змінних середовища CGI;
- механізм перевірки коректності заповнення полів HTML-форми;
- спеціальні функції для відображення даних у потрібному форматі;
- скерування запиту або результату на інший URL;
- вміщення шаблону в інші шаблони. що полегшує структуризацію обробки запитів;
- динамічне визначення SQL-виразів у шаблоні;
- підтримка транзакцій;
- вкладання тегів <DBOUTPUT> один в один з метою структуризації областей виведення.

## 13.8.2. Доступ до баз даних SQL Server 2000 через веб-інтерфейс

СКБД SQL Server 2000 підтримує мову XML. У розпорядженні користувачів є технологія, що дає змогу за допомогою браузера не лише переглядати дані в базах, але й змінювати їх.

Використання технології XML вимагає попереднього конфігурування віртуального каталогу для веб-серверу, до якого будуть звертатися користувачі під час вибирання даних, а також зазначення методу формування наборів даних, що повертатимуться.

Для відкриття доступу до даних засобами XML необхідно створити віртуальний каталог, до якого будуть звертатися користувачі. У віртуальному каталозі може зберігатися така інформація: ім'я віртуального каталогу, метод автентифікації, адреса джерела даних та метод доступу до них

### Ім'я віртуального каталогу

Ім'я віртуального каталогу буде використовуватися для звернення до нього користувачів. Так, якщо на сервері nau як ім'я віртуального каталогу обрано university, то для звернення до нього в рядку адреси браузера слід набрати http://nau/university

Один віртуальний каталог можна використовувати для доступу до багатьох наборів даних. Зокрема, для роботи з даними використовуються спеціальні XML-файли, що містять опис усіх дій, які необхідно виконати під час здійснення доступу до даних. Окрім цього, користувач безпосередньо в полі адреси браузера може вказати запит, за допомогою якого має бути сформовано початковий набір даних.

Коли джерело даних визначається безпосередньо в полі адреси браузера, то для роботи з даними використовувати файли не обов'язково. Інакше необхідно вказати шлях до папки з XML-файлами, за допомогою яких буде готуватися набір даних для публікації.

### Метод автентифікації

Веб-сервер має з'єднуватися з SQL-сервером, а для цього необхідно здійснити автентифікацію. Існують три способи автентифікації

- Використовувати єдине ім'я і пароль для під'єднання всіх користувачів через віртуальний каталог. Цей варіант зручний у тому випадку, коли через віртуальний каталог надається доступ до загальнодоступних даних із бази.
- Використовувати ім'я і пароль сеансу користувача у Windows. У цьому випадку слід узгоджувати паролі доступу до операційної системи з паролями доступу до СКБД.
- Вимагати від користувача передавання імені та пароля під час першого звернення до віртуального каталогу (ці дані будуть передаватися на SQL-сервер відкритим текстом).

### Адреса джерела даних

Для організації з'єднання з конкретною базою даних необхідно вказати ім'я SQL-сервера і бази даних. Лише до даних із вказаної бази вказаного сервера буде дозволено доступ через віртуальний каталог.

### Метод доступу до даних

Доступ до даних, що містяться у вказаній користувачем базі, здійснюється одним з таких методів:

- за запитом з URL;
- за допомогою шаблонів запитів;
- за допомогою запитів XPath.

Розглянемо стисло кожен із цих методів.

### Запити з URL

Цей метод передбачає звернення до бази даних з використанням довільних запитів, які вказуються в полі адреси браузера. Наприклад, якщо на сервері nau є віртуальний каталог university, пов'язаний із базою даних Лекції, то для вибирання даних зі стовпців Name та Addr таблиці Tchr можна ввести в полі адреси браузера такий рядок:

```
http://nau/univ?sql=select+'':select+Name,+Addr+from+Tchr+for+xml+raw;select+'</ROOT>'
```

У результаті може бути отриманий такий XML-текст:

```
<ROOT>
  <row name='Ivanov' address='Kiev, Teremkovskaya,1,19'/>
  ...
</ROOT>
```

Переваги цього методу полягають у тому, що він дає можливість формулювати довільні запити, а недоліки — в тому, що користувач повинен знати мову SQL. Окрім того, під час повторного виконання запиту його слід вводити ще раз. Цей недолік нейтралізується, якщо використовувати шаблони запитів.

### Шаблони запитів

*Шаблон* — це заздалегідь підготовлений файл з описом того, як мають відображуватися дані під час виведення їх у вікні браузера, і власне текстом запиту, який створює множину даних, що відображуються в браузері.

Наведемо приклад шаблону, який дозволяє створити XML-файл з переліком назв та ідентифікаційних кодів компаній-замовників (повний вигляд схеми використаної бази даних для даного прикладу є несуттєвим)

```
<?xml version="1.0" encoding="windows-1252" ?>
<Schema xmlns="urn:schemas-microsoft-com:xml-data"
  xmlns:dt="urn:schemas-microsoft-com:datatypes"
  xmlns:x="urn:schemas-microsoft-com:xml-data-ex"
  xmlns:sql="urn:schemas-microsoft-com:xml-sql">
  <ElementType name="CustomerList" sql:relation="Customers"
   sql:key-fields="CustomerID">
    <group order="many"><element type="Customer"/></group>
  </ElementType>
  <ElementType name="CompanyName" sql:relation="Customers"/>
  <ElementType name="CustomerID" sql:relation="Customers"/>
  <ElementType name="Customer" sql:relation="Customers">
    <element type="CompanyName" sql:field="CompanyName"/>
    <element type="CustomerID" sql:field="CustomerID"/>
  </ElemetType>
</Schema>
```

Звертаючись до шаблонів через браузер, необхідно використовувати так звані *віртуальні імена*. Віртуальне ім'я — це псевдонім імені необхідного файла разом зі шляхом доступу до нього через каталоги операційної системи на сервері. Припустимо, що наведений вище шаблон зберігається на сервері nau і йому зіставлене

віртуальне ім'я template.xml. Тоді для виконання записаного в шаблоні запиту в рядок адреси браузера можна ввести, наприклад, такий текст:

```
http://nau/university/template.xml
```

У результаті обробки шаблону отримаємо XML-файл:

```
<ROOT xmlns::sql="urn:schemas-microsoft-com:xml-sql">
  <Customer>
    <CompanyName>Alfreds Futterkiste</CompanyName >
    <CustomerID>ALFKI</CustomerID>
  </Customer>
  <Customer>
  ...
  </Customer>
  ...
</ROOT>
```

**Запити XPath**

Мова XPath призначена для адресації частин XML-документів. Як і під час роботи з шаблонами, для звернення до XML-документів за допомогою XPath застосовуються віртуальні імена. Наприклад, певному XML-файлу, що містить відомості про викладачів, зіставлене віртуальне ім'я myschema. Тоді в полі адреси браузера можна ввести такий рядок:

```
http://nau/university/myschema/Teacher[@TeacherID='5']/@Photo
```

У цьому рядку записаний запит мовою XPath. Для досягнення того ж результату за допомогою мови SQL необхідно ввести такий текст:

```
http://nau/university?sql=select+photo+from+Teacher+where+TeacherID=5
```

У результаті на браузер буде передано рисунок, що міститься у стовпці photo таблиці Teacher у рядку, що відповідає викладачу з ідентифікатором 5.

## Контрольні запитання та завдання

1. Назвіть характерні риси систем баз даних, що притаманні й технології XML.
2. Які функції СКБД не підтримує технологія XML?
3. Які існують основні типи документів XML? У чому полягає відмінність між ними?
4. Назвіть три основні різновиди баз даних на основі XML.
5. Якою є архітектура баз даних з дворівневим доступом на основі XML? Які переваги та недоліки таких БД?
6. Назвіть відмінні риси баз даних із вбудованою підтримкою XML.
7. Які є різновиди БД із вбудованою підтримкою XML?
8. В який спосіб забезпечується нормалізація та цілісність посилань даних у БД із вбудованою підтримкою XML?

9. Які є способи відображення схеми документа XML на схему бази даних?
10. Назвіть типи мов запитів для баз даних на основі XML.
11. Опишіть процедуру перетворення опису DTD на схему бази даних і навпаки.
12. Що таке публікація баз даних в Інтернеті? Як вона виконується?
13. Опишіть загальну схему доступу до баз даних через Інтернет.
14. Опишіть принцип обробки веб-документа в пакеті Cold Fusion.
15. Як здійснюється доступ до баз даних через веб-інтерфейс у СКБД SQL Server 2000?

# Розділ 14

# Об'єктно-орієнтовані бази даних

- Об'єктно-орієнтована модель даних
- Мова опису об'єктів
- Об'єктна мова запитів
- Архітектура об'єктно-орієнтованих СКБД
- Зображення об'єктної моделі в реляційній базі даних

## 14.1. Сучасний стан досліджень у галузі об'єктно-орієнтованих баз даних

Дослідження у сфері об'єктно-орієнтованих баз даних (ООБД) були розпочаті у зв'язку з необхідністю розробити ефективний механізм, що дозволяє об'єктно-орієнтованим застосуванням зберігати об'єкти після завершення своєї роботи. У будь-якій СКБД користувачами можуть бути не лише люди, а й програми, відтак програми можуть звертатися до СКБД як для збереження даних, так і для їхнього подальшого вибирання. Майже всі сучасні мови програмування є об'єктно-орієнтованими, а СКБД – реляційними. У зв'язку з цим нагальною стає потреба у створенні об'єктно-орієнтованих СКБД або розробці механізмів відображення між двома моделями даних: об'єктно-орієнтованою та реляційною. Інакше кажучи, необхідно надати програмістам, які працюють в об'єктно-орієнтованому середовищі, прозорий механізм збереження і вибирання даних із баз даних.

*Об'єктно-орієнтована СКБД* (ООСКБД) – це СКБД, що базується на об'єктно-орієнтованій моделі даних.

Нині ООСКБД приділяється багато уваги як з теоретичної, так і з практичної точок зору. Можна виділити три характерні риси сучасного стану досліджень у цій галузі:

- відсутність загальноприйнятої моделі даних;
- відсутність єдиної формальної теорії;
- активна експериментаторська діяльність.

Класичні роботи Кодда специфікували системи реляційних баз даних (модель даних і мову запитів), для систем об'єктно-орієнтованих баз даних подібна специфікація відсутня. Це не означає, що не існує жодної повної моделі об'єктно-орієнтованої бази даних. Таких моделей є багато, але жодна з них не може вважатися

загальноприйнятим стандартом. Нині нема єдиної думки про те, чим є об'єктно-орієнтована система програмування, не кажучи вже про ООСКБД. Окрім того, відсутній надійний теоретичний базис, що робить практично неможливим досягнення спільного погляду на модель даних.

Ведуться численні експериментальні дослідження. Можна сподіватися, що разом із відповідною моделлю виникнуть її життєздатні реалізації. Можливо, незабаром за взірець буде взята певна система, але не тому, що вона буде найоптимальнішою, а тому, що вона виявиться першою системою, яка надасть багато функціональних можливостей, відповідних до вимог ринку.

Особливе місце в розвитку проблематики об'єктно-орієнтованих СКБД посідає діяльність групи з управління об'єктними базами даних ODMG (Object Data Management Group) - некомерційного консорціуму виробників об'єктних баз даних та інших організацій, зацікавлених у виробленні стандартів зі зберігання об'єктів у базах даних.

ODMG діяла з 1991 по 2001 рік. У 1993 році група випустила свій перший стандарт – ODMG-93. 1995 року був опублікований удосконалений варіант цього стандарту. Стандарти ODMG – це стандарти зі зберігання об'єктів в базах даних. Ці стандарти стосуються трьох галузей: баз даних (мова SQL), об'єктної моделі даних (стандарти OMG Object Management Group - консорціуму, що розробляє стандарти міжкомпонентної взаємодії) та об'єктно-орієнтованих мов програмування (стандарти ANSI з мов програмування C++, Smalltalk і Java).

Складовими частинами стандартів ODMG є:

- об'єктно-орієнтована модель (Object Oriented Model – OOM);
- мова опису об'єктів (Object Definition Language – ODL);
- об'єктна мова запитів (Object Query Language – OQL);
- описи методів зв'язування OOM з об'єктно-орієнтованими мовами програмування C++, SmallTalk, Java (Language Bindings to C++, SmallTalk, Java).

Далі ми зробимо короткий огляд об'єктно-орієнтованої моделі, що була запропонована ODMG, а також опишемо мови ODL й OQL.

## 14.2. Об'єктно-орієнтована модель ODMG

В об'єктно-орієнтованій моделі дані та методи, що їх обробляють, об'єднуються в структури, які називаються *об'єктами*. Типи об'єктів називаються *класами*. З точки зору баз даних є такі важливі особливості OOM:

- підтримка структур даних, що мають довільний рівень складності;
- ідентифікованість та унікальність об'єктів;
- належність об'єктів класам;
- інкапсуляція;
- успадкування та ієрархії класів;
- поліморфізм.

### Складні структури даних

Наявність складноструктурованих даних не є відмінною рисою ООМ, проте існування ООМ без механізмів породження структур даних довільної складності важко уявити. Складні об'єкти будуються з простіших за допомогою *конструкторів*. Найпростішими об'єктами є: числа, символи, символьні рядки довільної довжини, булеві змінні тощо. Існують різні конструктори складних об'єктів (кортежів, множин, мультимножин, списків та масивів). Мінімальний набір конструкторів, який повинна мати система, – це конструктори множин, списків і кортежів. Множини необхідні, оскільки завдяки їм набори об'єктів реального світу зображуються в природний спосіб. Кортежі надають спосіб зображення властивостей сутностей. Списки та масиви потрібні для відображення впорядкованості об'єктів реального світу, а також для зображення широкого кола математичних об'єктів.

Будь-який конструктор має бути застосовним до будь-якого об'єкту (наприклад, повинна надаватися можливість побудови множини з масивів або масиву з множин). Конструктори реляційної моделі не мають такої властивості, оскільки конструкція множини може бути застосована лише до кортежів, а конструкція кортежу – лише до атомарних значень. Навіть реляційна модель, що підтримує ненормалізовані відношення (тобто відношення, які не перебувають у першій нормальній формі), не має вказаної властивості, оскільки конструкцією верхнього рівня завжди має бути відношення.

Маніпулювання складними об'єктами забезпечується відповідними *операціями*, які часто розповсюджуються на всі компоненти таких об'єктів. Прикладом може бути вибирання чи видалення складного об'єкту або створення його копії. Існує можливість визначати додаткові операції над складними об'єктами.

### Ідентифікованість, унікальність і стан об'єктів

Кожний об'єкт є унікальним, тобто забезпечується унікальна *ідентифікація об'єктів* (для мов програмування унікальними ідентифікаторами можуть бути адреси пам'яті, за якими зберігаються об'єкти).

*Стан об'єкта* – це поточне значення, приписане об'єкту. Об'єкт може мати єдиний стан протягом свого життєвого циклу або переходити з одного стану в інший. Оскільки об'єкти мають властивість інкапсуляції (що буде розглянута нижче), то стан об'єкта є абстракцією, яка визначається лише через його поведінку (методи).

Унікальність об'єкта не залежить від його стану. Два об'єкти, що перебувають в одному й тому ж стані, є рівними, але не ідентичними. Не може існувати двох або більше екземплярів одного об'єкта (двох копій одного й того самого об'єкта з одним і тим самим ідентифікатором) У моделі з ідентифікованістю об'єктів об'єкт існує незалежно від свого значення. Отже, є два поняття еквівалентності об'єктів: об'єкти можуть бути ідентичними (бути одним і тим самим об'єктом) або вони можуть бути рівними (перебувати в одному й тому самому стані). У цьому контексті слід розглянути такі два аспекти: розрізнення та змінення об'єктів.

### Розрізнення об'єктів

У моделі з об'єктами, що ідентифікуються, два об'єкти можуть спільно використовувати компоненти (зокрема інші об'єкти). Отже, схематичним відображенням

складного об'єкта є граф, натомість у системі без ідентифікованості об'єктів — це дерево. Розглянемо такий приклад: людина має ім'я, вік і дітей. Припустимо, що Іван і Марія мають сина на ім'я Петро. У реальному житті можуть мати місце дві ситуації: Іван і Марія є батьками однієї і тієї самої дитини або кожний із них має по сину. У моделі з ідентифікованістю об'єктів адекватно відображуються обидві ситуації.

### Змінення об'єктів

Припустимо, що Іван і Марія є батьками хлопчика на ім'я Петро. Тоді зміни, що стосуються сина Марії, будуть виконуватися з об'єктом Петро, відтак і з сином Івана.

### Поведінка об'єктів

*Поведінка об'єкта* — це сукупність операцій (методів), які він надає. Лише через ці операції розкривається семантика об'єкта. Виконання операцій є єдиним способом взаємодії між об'єктами. Всі можливі операції об'єкта утворюють його *інтерфейс*. Лише використовуючи операції, можна змінити стан об'єкта.

### Класи об'єктів

В об'єктно-орієнтованій моделі клас узагальнює спільні риси об'єктів, що мають однакові властивості, й відповідає поняттю абстрактного типу даних. Клас означує спосіб реалізації множини об'єктів, встановлюючи їхню структуру, поведінку та інтерфейс, тобто спосіб запам'ятовування інформації про їхні стани. Проте власне стан має запам'ятовувати сам об'єкт.

Клас є водночас фабрикою та сховищем об'єктів. Як *фабрика об'єктів* клас використовується для створення нових об'єктів. Термін *сховище об'єктів* означає, що до класу приєднується набір об'єктів, які є його екземплярами. Отже, класи використовуються для створення об'єктів і маніпулювання ними.

Однією з основних властивостей класу, відтак і його об'єктів, є інкапсуляція.

### Інкапсуляція

*Інкапсуляція* вимагає, щоб дані та програмні коди для маніпулювання даними були приховані. З цієї точки зору об'єкт поділяється на інтерфейсну й реалізаційну частини. *Інтерфейсна частина* є специфікацією набору операцій, допустимих над об'єктом. Лише ця частина об'єкта видима для методів інших об'єктів. *Реалізаційна частина* складається з даних, що описують стан об'єкта, і процедур, що реалізують операції над об'єктом. Інкапсуляція специфікується на рівні оголошення класу. Наприклад, оголошення класу об'єктів з іменем Товар мовою C++ або Java може виглядати так:

```
public class Товар {
// Дані
  string назва;
  float  ціна;
  int    кількість_на_складі;
// Методи
  public Товар() { /*.*/ };
  public int Є_на_складі() { /*.*/ };
```

```
  public int Надходження(int n) { /*.*/ };
  public int Продаж(int n) { /*.*/ };
  public float Сумарна_ціна(int n) { /*.*/ };
}
```

Дані цього класу приховані від методів інших класів, які могли б їх використовувати. Лише методи класу, які, у свою чергу, використовують дані, є доступними для будь-яких методів інших класів.

### Успадкування

*Успадкування* є механізмом, що дає змогу створювати нові класи з використанням даних і методів інших класів. Це дає можливість деякі властивості, спільні для багатьох класів, описувати в *базовому класі*. Наприклад, якщо необхідно додати новий клас `Одяг`, що повністю збігається з класом `Товар`, за винятком наявності додаткових відомостей про розмір, мовою Java це можна записати так:

```
public class Одяг extends Товар {
int розмір;
}
```

У такий спосіб ми вказали, що `Одяг` — це `Товар`, тому перший має всі дані й методи другого, а також власні дані, що зберігаються в змінній `розмір`. Успадкування дає змогу будувати ієрархію класів.

### Поліморфізм

Принцип *поліморфізму* є розширенням принципу успадкування й дає змогу переозначувати методи в успадкованих класах. Наприклад, є кілька різновидів товарів, для яких властиві специфічні правила обчислення ціни. Зокрема товарами можуть бути продукти, на які не діють націнки, та одяг, на який націнка встановлена. Всі товари успадковують метод `Сумарна_ціна()` з базового класу `Товар`, але правила обчислення ціни можуть бути різними. Це означає, що кожний клас, який є нащадком класу `Товар`, може мати власну реалізацію методу `Сумарна_ціна()`. Тому вираз `x.Сумарна_ціна()` може означати різні правила обчислення ціни залежно від того, екземпляром якого класу є об'єкт `x`.

## 14.3. Мова опису об'єктів ODL ODMG

Будь-яка СКБД має мову опису даних (МОД), що використовується для опису схем баз даних. Мова опису об'єктів ODL ODMG розглядається як розширення МОД, призначене для опису об'єктів, їхніх атрибутів, зв'язків та операцій. Основою цієї мови стала мова IDL (Interface Definition Language), розроблена групою OMG.

Мова ODL є абстрактною в тому розумінні, що згенерована ODL-схема має бути незалежною від мов програмування та конкретної СКБД. У зв'язку з цим в ODL розглядаються лише аспекти означення об'єктів різних типів і повністю ігноруються питання реалізації методів. Згенерована ODL-схема може вільно переміщуватися між СКБД, що підтримують концепцію ODMG, використовуватися програмами, записаними різними мовами програмування, і навіть транслюватися в конкретні МОД, наприклад ті, які базуються на стандарті SQL-1999.

## 14.3.1. Основні положення

ODL – це мова, призначена насамперед для специфікації класів. Вона підтримує об'єктну модель ODMG і не є мовою програмування. Більше того, ODL незалежна від мов програмування. Основна мета розробки цієї мови – створити єдину основу для опису об'єктів і тим самим забезпечити перенесення схем об'єктних даних між різними ООСКБД. ODL також можна використовувати для відображення об'єктних даних у різні мови програмування, як це показано на рис. 14.1.

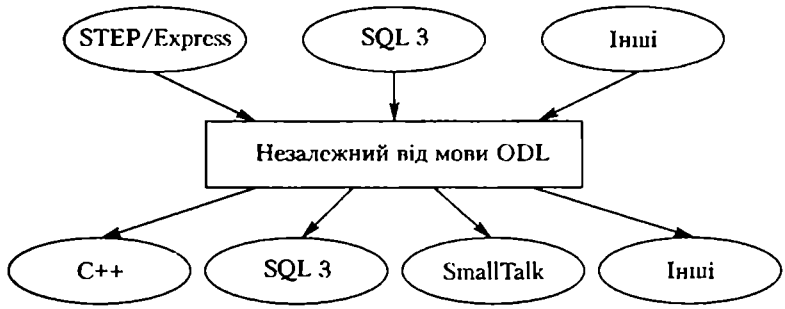



**Рис. 14.1.** Відображення об'єктних даних у мови програмування за допомогою ODL

Основні положення об'єктної моделі даних ODMG:

- базовим поняттям моделі є об'єкт;
- поведінка об'єкта визначається за допомогою множини його операцій;
- стан об'єкта визначається за допомогою множини його властивостей:
- об'єкти належать до класів, саме через класи вони специфікуються;
- клас має інтерфейсну та реалізаційну частини;
- клас є об'єктом;
- ODL специфікує класи.

В об'єктній моделі ODMG мова йде насамперед про типи, а вже потім про класи. Клас розглядається як різновид типу, що має одну реалізацію. У загальному випадку допускається, що тип має кілька реалізацій. Множинність реалізацій типу необхідна для підтримки неоднорідних баз даних, розподілених у мережі, та середовищ з різними мовами програмування й компіляторами. Нас не пікавить реалізація, тому далі основна увага приділятиметься класам.

### Система типів ODL

Базовими типами мови є: `integer`, `float`, `string`, `boolean`, перелічувані типи, що створюються за допомогою ключового слова `enum`, та класи. Похідні типи створюються за допомогою конструкторів типів. Є конструктор `Struct` для структур і чотири конструктори для типів колекцій: `Set`, `Bag`, `List`, `Array`. Опис структур і колекцій наведено далі.

### Об'єкти і літерали

Мовою опису об'єктів обробляються об'єкти та літерали. І об'єкти, і літерали поділяються на атомарні та структуровані. І об'єкти, і літерали ідентифіковані,

проте літерал ідентифікується своїм значенням, а об'єкт - ідентифікатором OID (Object IDentifier).

**Об'єкти**

Основні характеристики об'єктів.

- OID унікально ідентифікує об'єкт, відрізняючи його від інших об'єктів тієї предметної області, де він був створений. Будь-який об'єкт має лише один OID, але може мати більше одного *імені*. Об'єкти можуть ідентифікуватися предикатами, визначеними на їхніх *властивостях*.
- Видалення об'єкта не призводить до рекурсивного видалення пов'язаних із ним об'єктів.

На рис. 14.2 наведена класифікація об'єктів.


**Рис. 14.2.** Класифікація об'єктів у ODL

**Літерали**

Літерали — це об'єкти, екземпляри яких не можна змінювати. Для наперед визначених типів літералів не можна змінювати операції.

Класифікація літералів наведена на рис. 14.3.

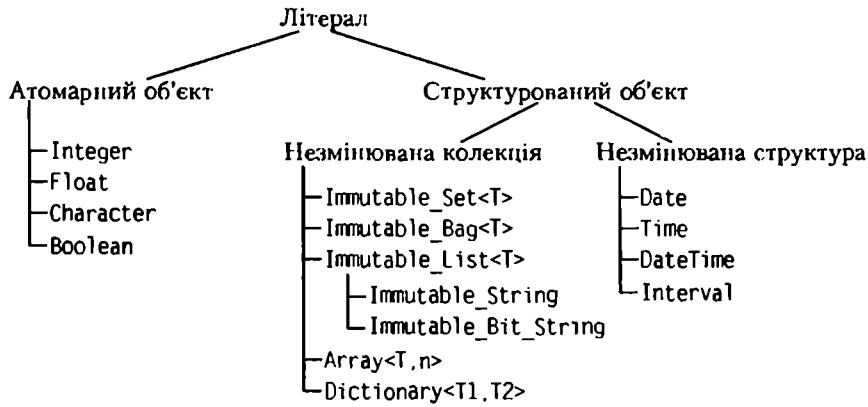



**Рис. 14.3.** Класифікація літералів у ODL

### Колекції

Об'єкти і літерали можуть створюватися за допомогою колекцій. Колекція містить елементи, які належать одному типу. Передбачається, що для всіх колекцій означені такі стандартні операції:

- `insert element()` — додати елемент до колекції;
- `remove element()/remove element at()` — видалити елемент із колекції;
- `replace element at()/retrieve element at()` — замінити елемент колекції;
- `select element()/select()` — знайти елемент колекції;
- `create iterator()` — створити *ітератор колекції* (механізм послідовного перебирання елементів);
- `create index()/drop index()` — створити/видалити *індекс колекції* (механізм прискорення пошуку елементів колекції).

Є такі вбудовані типи колекцій: `Set`, `Bag`, `List`, `Array`, `Dictionary`.

`Set` (множина) — невпорядкована сукупність однотипних елементів, дублювання яких не допускається. Множина специфікується так: `Set<тип>`, де `тип` — це тип елементів множини. Над множинами означені такі стандартні операції:

- `union()/intersection()/difference()` — об'єднання, перетин і різниця;
- `is_subset?()/is_proper_subset?()` — предикати перевірки, чи є колекція підмножиною (строгою підмножиною) іншої колекції;
- `is_superset?()/is_proper_superset?()` — операції, симетричні двом попереднім

`Bag` (мультимножина) — невпорядкована сукупність елементів, які можуть дублюватися. Специфікація мультимножини виглядає так: `Bag<тип>`, де `тип` — тип елементів мультимножини.

`List` (список) впорядкована сукупність елементів, дублювання яких допускається. Список специфікується так: `List<тип>`, де `тип` — тип елементів списку. Елементи списку сортуються відповідно до порядку їхнього додавання. Для списків означені такі стандартні операції:

- `insert_element_after()/insert_element_before()` — вставляння елементу;
- `remove_element_after()/remove_element_before()` — видалення елементу;
- `replace_element_at()/retrieve_element_at()` — заміна/пошук.

`Array` (масив) — одновимірний масив заданої довжини, що специфікується так: `Array<тип, число>`, де `тип` — тип елементів масиву, `число` — ціле додатне число, що визначає розмір масиву. Стандартні операції над масивами:

- `insert_element_at()/remove_element_at()` — вставляння/видалення елементу;
- `replace_element_at()/retrieve_element_at()` — заміна/пошук елементу;
- `resize()` — зміна розмірності масиву.

`Dictionary` (словник) — множина пар вигляду «ключ, значення», що специфікується так: `Dictionary<тип_1, тип_2>`, де `тип_1` тип ключів, `тип_2` — тип значень. Ключі не можуть повторюватися. Словник організується так, аби пошук значення за заданим ключем виконувався ефективно.

Наведемо приклади специфікацій колекцій:

```
Array<string 10>;

Struct Point{
  float X;
  float B;
};

Set<Point>;

List<integer>.
```

### Структури

Структура — це непойменований набір елементів. Структура специфікується так:

```
Struct ім'я {тип_1 ім'я_1, ..., тип_n ім'я_n}
```

Наведемо приклади специфікацій структур і класу, елементом якого є структура:

```
Struct PhoneNumber{
   integer CountryCode,
   integer AreaCode,
   integer PersonCode
};

Struct Address{
   string Street,
   string City,
   PhoneNumber Phone
};

class Person{
   attribute string Name;
   attribute Address PersonAddress;
};
```

Перелічимо основні стандартні операції над структурами:

- get_element_value() — отримання значення елементу;
- set_element_value() — встановлення значення елементу;
- copy() — копіювання екземпляра структури.

## 14.3.2. Специфікація класів

Синтаксис оголошення класу в мові ODL є таким:

```
class ім я [: Список_імен_суперкласів]
          [(Список_характеристик)]
{
  [Список_властивостей]
  [Список_методів]
};
```

Будь-який клас повинен мати ім'я, яке має бути унікальним у межах схеми, що описується. Рекомендується іменувати клас іменником в однині.

Список_імен_суперкласів - це список імен класів, які є суперкласами даного класу. Клас може мати будь-яку кількість суперкласів: один, багато або жодного. Підклас успадковує всі властивості і операції суперкласу.

### Характеристики класу

Елементами списку характеристик класу можуть бути екстенти і ключі.

*Екстент* (extent) — це множина всіх екземплярів даного класу. Якщо клас *A* є підкласом класу *B*, то екстент *A* буде підмножиною екстента *B*. Рекомендується іменувати екстент тим самим іменником, що і його клас, але в множині. Екстент може бути відсутній. Клас, для якого екстент не оголошено, не може мати екземплярів і називається *інтерфейсом*. Приклад оголошення екстента:

```
class Professor:Employee (extent professors) {
...
};
```

*Ключ* — це атрибут або група атрибутів, значення яких мають бути унікальними для всіх об'єктів класу. Ключі специфікуються ключовим словом key або keys, за яким записується ім'я атрибута, що становить ключ, наприклад key passport. Якщо ключ складається з кількох атрибутів, їхні імена беруться в круглі дужки, наприклад key(room, hours). Допускається, що клас може мати кілька альтернативних ключів, тоді задається кілька списків, наприклад key(room, hours), (dept, number).

Оголошувати ключі не обов'язково, оскільки для унікальної ідентифікації об'єктів цілком достатньо OID.

Специфікуючи клас, можна використовувати не більше одного ключового слова extent та key (keys). Ці ключові слова можна розташовувати в довільному порядку.

### Список властивостей

У списку властивостей специфікуються атрибути та/або зв'язки.

#### Специфікація атрибутів

Специфікація атрибута має такий синтаксис:

```
attribute тип ім'я;
```

Типом атрибута може бути один з атомарних типів або структурований, створений за допомогою рекурсивного застосування конструкторів колекцій та/або структури.

Передбачається, що для будь-якого атрибута можна застосувати вбудовані операції встановлення та отримання його значення: set value(значення) та get value(). Наведемо приклади специфікацій атрибутів:

```
attribute string name;

attribute Set<Struct Degree{
                string degree_name,
                Year degree_year}> degree;
```

#### Специфікація зв'язків

Специфікація зв'язків необхідна для встановлення зв'язків між об'єктами даного класу та об'єктами іншого класу. Оголошення зв'язку має такі складові:

- ім'я цільового класу (з можливими конструкторами колекцій);
- ідентифікатор зв'язку;
- ідентифікатор зворотного зв'язку, який зазначається після ключового слова inverse.

Зв'язки означуються між двома класами об'єктів, тобто підтримуються лише бінарні зв'язки. Зв'язок зображується у вигляді інверсної пари, тобто ідентифікується в обох напрямах. Є можливість відображення таких різновидів зв'язків:

- «один-до-одного» — обидва закінчення зв'язку є класами;
- «один-до-багатьох» — закінчення «один» зв'язується з колекцією, а закінчення «багато» — з класом;
- «багато-до-багатьох» — обидва закінчення зв'язку зв'язуються з колекціями.

Слід зазначити, що для кожного зв'язку вказується два оголошення: по одному в кожному з класів, які зв'язуються. Якщо зв'язок має множинність «багато-до-багатьох», то в обох оголошеннях після ключового слова relationship вказуються колекції. Після ключового слова inverse завжди вказується тільки ідентифікатор зворотного зв'язку, якому передує специфікатор класу, де цей зв'язок оголошено: ім'я_класу::ім'я_зв'язку.

Приклади специфікації зв'язків:

```
relationship List<Section> has_sections
inverse Section::is_section_of;

relationship Set<Course> has_prerequisites
inverse Course::is_presiquite_for;

relationship Course is_section_of
inverse Course::has_section;
```

Для зв'язків означені такі вбудовані операції:

- add_one_to_one(theFirstObj, theSecObj) — встановлення зв'язку між об'єктами;
- remove_one_to_one(theFirstObj, theSecObj) — видалення зв'язку між об'єктами;
- traverse(fromObj, toObj) — перехід від одного об'єкта до іншого

Приклади:

```
relationship Set<Student> advises inverse Student::advisor;
relationship Set  teaching_assistants inverse TA::works_for;
relationship Department department inverse Department::faculty;
```

#### Список операцій

ODL є сумісним з OMG IDL за специфікацією операцій. Синтаксис оголошення операції є таким:

```
тип_значення_що_повертається ім'я_операції([список_аргументів])
[raises(список_виключних ситуацій)];
```

Приклади:

```
grant_tenure();
raise(inlegible_for_tenure);
hire(in Person);
fire(in Professor) raises(no_such_employee);
```

Наведемо приклад специфікацій класів.

**Приклад**

Нижче наведено ER-діаграму фрагмента предметної області «вищий навчальний заклад», а також специфікацію цього фрагмента мовою ODL. Ця специфікація буде використана під час розгляду мови OQL.

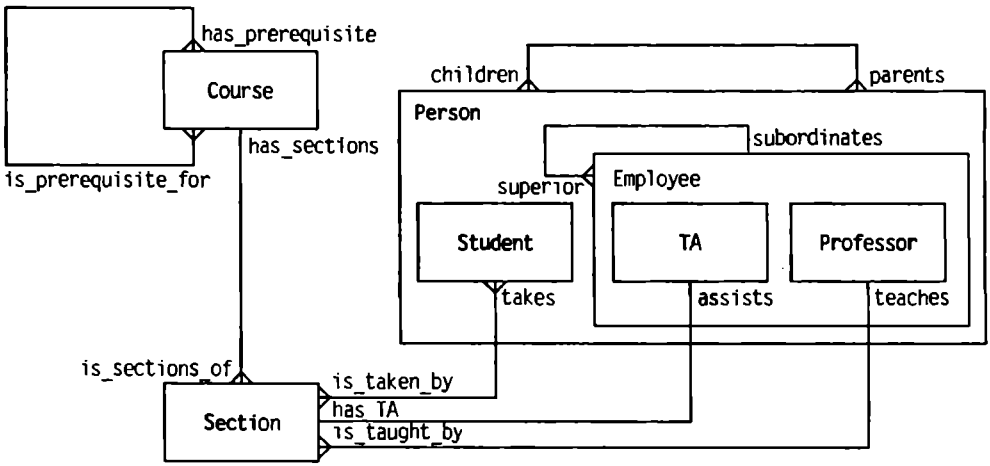



**Рис. 14.4.** ER-діаграма для об'єктно-орієнтованої моделі

```
class Course(extent courses keys name, number) {
  attribute string name;
  attribute string number;
  relationship List<Section> has_sections
      inverse Section::is_section_of;
  relationship Set<Course> has_prerequisites
      inverse Course::is_presiquite_for;
  relationship Set<Course> is_prerequisite_for
      inverse Course::has_presiquites;
  offer(in Semister) raises(already_offered);
  drop(in Semister) raises(not_offered);
};

class Person(extent persons key passport) {
  attribute string name;
  attribute string passport;
  attribute char sex;
  attribute integer faculty_id;
```

```
  attribute date birthdate;
  attribute string activities;
  attribute Struct{string street, string city, string
                   state, string zip} address;
  relationship Set<Person> children inverse Person::parents;
  relationship Set<Person> parents inverse Person::children;
  integer age();
  Person sibling();
  boolean hasAncestor(in Person);
  facultyName();
};

class Employee: Person(extent employees) {
  attribute integer salary;
  relationship Set<Employee> subordinates inverse
  Employee::superior;
  relationship Employee superior inverse Employee::subordinates;
  integer seniority();
  string activities();
};

class Professor: Employee(extent professors) {
  attribute Set<Struct{string degree_name, Year degree_year}> degree;
  attribute string rank;
  relationship Set<Section> teaches
      inverse Section::is_taught_by;
};

class TA: Employee(extent TAs) {
  relationship Set<Section> assists
      inverse Section::has_TA;
};

class Student: Person(extent students) {
  attribute integer faculty_id;
  relationship Set<Section> takes
    inverse Section::is_taken_by;
};

class Section(extent sections) {
  attribute string title;
  atrribute integer secnumber;
  relationship Set<Student> is_taken_by
      inverse Student::takes;
  relationship Course is_section_of
      inverse Course::has_section;
  relationship TA has_TA
      inverse TA::assists;
  relationship Professor is_taught_by
      inverse Professor::teaches;
};
```

## 14.4. Об'єктна мова запитів OQL ODMG

OQL ODMG – це незалежна мова запитів до об'єктної моделі даних ODMG, синтаксис якої базується на мові SQL. Окрім того, передбачається можливість її використання в мовах програмування.

Мова запитів орієнтована на побудову виразів, її конструкції мають такі властивості:

- будь-який запит є виразом, що має тип – об'єкт або літерал;
- вирази та операції над ними можуть вкладатися одне в одне;
- результатом виконання запиту є об'єкти, що належать типам, означеним у моделі ODMG, і можуть брати участь у формуванні виразів.

Мова має високорівневі примітиви для маніпулювання множинами, об'єктами, структурами, масивами і списками. У ній відсутні оператори оновлення, замість них використовуються операції, визначені для об'єктів. Передбачається, що всі створювані об'єкти мають OID, а літерали унікально ідентифікуються своїм значенням.

### 14.4.1. Запити OQL

В OQL запитом може бути будь-який вираз, що повертає об'єкт, колекцію об'єктів, літерал або колекцію літералів. Оскільки запит – це вираз, а будь-який вираз має тип, то й будь-якому запиту ставиться у відповідність тип. Наприклад, ім'я екстента є запитом і повертає множину об'єктів, що містяться в ньому. Так, вираз persons має тип Set<persons> і повертає множину об'єктів з екстента persons.

Пойменований об'єкт також є допустимим запитом, наприклад, якщо один із об'єктів класу Person пойменований як Dean, то такі два запити:

```
Dean, Dean.address
```

відшукують даний об'єкт і його адресу. Типи наведених запитів є такими: Person і Struct{street: string, city: string, state:string, zip:string}.

Найпоширенішим різновидом запиту, як і в SQL, є select-запит. Наприклад, наведений нижче запит знаходить множину осіб чоловічої статі, й при цьому типом значення, що повертається, буде Bag<Person>:

```
SELECT   x
FROM     persons x
WHERE    x.sex = 'M'
```

Зазначимо, що select-вираз повертає колекцію типу Bag, тобто мультимножину (множина зі значеннями, що повторюються). Використання слова DISTINCT приводить до того, що тип колекції, яка повертається, – Set, тобто множина. Наведений нижче запит повертає список номерів факультетів для людей на ім'я Pete, тобто літерал типу Set<integer>.

```
SELECT DISTINCT x.faculty_id
FROM     persons x
WHERE    x.name = 'Pete'
```

Зауважте, що в запитах слід звертатися саме до екстентів, а не до класів.

## 14.4.2. Обчислення проміжних результатів

Запит можна не лише сформулювати для виконання, але й визначити для подальшого використання в інших запитах (певний аналог віртуальних таблиць, створюваних командою CREATE VIEW у мові SQL). Це досягається за допомогою означення запиту, що має такий вигляд:

```
define ім'я_запиту as вираз
```

Наприклад

```
define Joe as element(SELECT   x
                      FROM     students x
                      WHERE    x.name = 'Joe')
```

Тут element є перетворювачем типу, що зводить *тип* одноелементної колекції до типу елементу. Отже, Joe є іменем об'єкта класу students із значенням атрибута name рівним 'Joe'. Тепер до об'єкта Joe можна звернутися, наприклад, для отримання значень його атрибутів. Так, вирази

```
Joe.birthdate,  Joe.faculty_id,  Joe.passport
```

повертають дату народження, номер факультету і номер паспорта студента на ім'я Joe.

Ім'я запиту розглядається як *елементарний вираз*, як і змінні, атоми та поіменовані об'єкти. Приклади елементарних виразів: 27, nil, students, Joe.

## 14.4.3. Вирази конструювання

Ці вирази дають змогу конструювати об'єкти, структури (структуровані літерали) та колекції. Синтаксис конструювання об'єктів є таким.

```
ім'я_типу(ім'я_атрибута_1 : вираз_1, ..., ім'я_атрибута_n : вираз_n)
```

Наведемо приклади:

```
Person(name: 'Pete', birthdate: '3/28/56', sex: 'M')

Person(name: 'Joan', birthdate: '8/23/65', sex: 'F',
address: Struct(street: '154 Main St.', city: 'Fairview', state: 'NH', zip: '0888'))
```

Тут створюються об'єкти класу Person (пропущені атрибути не ініціалізуються).

Синтаксис конструювання структурованого літерала є таким:

```
Struct(ім'я_атрибута_1 : вираз_1, ..., ім'я_атрибута_n . вираз_n)
```

Наприклад, запис Struct(name: 'Pete', age:25) приводить до створення структурованого літерала. Елементи name та age одержують типи своїх виразів (значень).

Також можна застосовувати конструктори колекцій, як це показується нижче.

- Set(вираз_1, ..., вираз_n) — конструювання множини.
  Наприклад, Set(1,2,3).
- Bag(вираз_1, ..., вираз_n) — конструювання мультимножини.
  Наприклад, Bag(1,1,2,3).

- `List(вираз_1, ..., вираз_n)` — конструювання списку. Наприклад, `List(1,1,2,3,3)`.
- `Array(вираз_1, ..., вираз_n)` — конструювання масиву. Наприклад, `Array(1,1,2,3,3)`.

### Використання конструкторів типів

Для зазначення типу результату є можливість явно застосовувати конструктори типів. У запиті, що наводиться нижче, утворюється множина двоелементних структур. Першим елементом структури є значення типу `date` (дата народження), а другим — значення типу `char` (стать). Подібні структури формуються для всіх об'єктів з екстента `persons`, у яких атрибут `name` має значення `Pete`.

```
SELECT DISTINCT Struct(a: x.birthdate, s: x.sex)
FROM    persons x
WHERE   x.name = 'Pete'
```

Якщо записати фразу `SELECT` як `SELECT DISTINCT a:x.birthdate, s:x.sex`, тип результату не зміниться, тобто якщо у фразі `SELECT` є список елементів, то він формує значення типу `Struct` (сам запит повертає дані типу `Bag` або `Set`, залежно від наявності фрази `DISTINCT`). Звернемо увагу на те, що у фразі `SELECT` відбувається перейменування полів: вказується ім'я поля, а потім через двокрапку записується шлях до елементу об'єкта. Якби в попередньому запиті поля у фразі `SELECT` не перейменовувалися, тобто фраза мала б вигляд `SELECT DISTINCT x.birthdate, x.sex`, тип результату був би `Set<Struct(birthdate: string, sex: char)>`.

У мові OQL є можливість визначати власні типи на базі стандартних і використовувати їх для визначення типу результату пошуку. При цьому у створених конструкторах можна використовувати select-запити. Наприклад, припустимо, що є таке означення типу:

```
typedef vectint: Bag<integer>;
```

Тоді можна оголосити конструктор `vectint`, який створює і повертає об'єкт типу `vectint`, що містить результати виконання запиту:

```
vectint(SELECT DISTINCT x.faculty_id
        FROM    persons x
        WHERE   x.name='Pete')
```

### Шляхи доступу

OQL є навігаційною мовою, оскільки вона дає змогу перемщуватися за посиланнями від поточного об'єкта до інших. Для цього використовується вираз шляху доступу, що складається з перелічених далі компонентів.

1. Ім'я змінної об'єкта.
2. Крапка (.) або стрілка (->), що позначає *оператор доступу до атрибута* (ці позначення є синонімічними).
3. Посилання на атрибут, що може бути або його ім'ям, або шляхом доступу, тобто послідовністю довільної довжини, що складається з імен атрибутів і операторів доступу.

Наприклад, припустимо, що змінна p ініціалізована об'єктом типу persons. Тоді обидва наведені нижче вирази повертають назву міста, в якому проживає певна людина:

```
p.address.city

p->address->city
```

Вираз шляху не може містити імен змінних типу колекції, оскільки шлях має бути однозначним. Наприклад, якщо children є колекцією, шлях p.children.address.city недопустимий. Для проходження багатьма шляхами використовуйте директиву SELECT:

```
SELECT   c.address.city
FROM     p.children c
```

### Фраза FROM

У фразі FROM вказується колекція об'єктів/літералів, з яких вибираються значення. Під час виконання запиту ця колекція послідовно переглядається, при цьому застосовується умова з фрази WHERE і вибираються значення атрибутів, вказаних у фразі SELECT. З об'єктами фрази FROM може бути зв'язана змінна. Якщо persons є ім'ям екстента або пойменованою колекцією, то обидва наведені нижче варіанти фрази FROM вказують на те, що змінна x послідовно ініціалізувалася об'єктами з persons:

```
FROM persons AS x
FROM persons x
```

Для того щоб у фразі FROM вказати ім'я колекції, можна використовувати всі засоби мови OQL зі специфікації колекцій, зокрема, можна вказувати:

- імена екстентів;
- шляхи доступу;
- select-вирази.

Використання імен екстентів і шляхів доступу розглядалося в попередніх прикладах. Наведемо приклад використання у фразі FROM select-виразів.

**Запит 14.1**

Вивести міста, в яких мешкають онуки p.

```
SELECT   g.address.city
FROM     (SELECT c.children g FROM p.children c)
```

Як і в SQL, у фразі FROM можна використовувати список колекцій. Наведений вище запит може бути записаний так:

```
SELECT   g.address.city
FROM     p.children c, c.children g
```

Зауважте, що змінна c, визначена за допомогою першого виразу шляху доступу, використовується в другому виразі.

Наведемо ще один приклад вкладання змінних у фразі FROM.

**Запит 14.2**

Вивести імена викладачів, що мають звання full professor, і студентів, яким вони викладають.

```
SELECT   couple(professor:z.name, student:x.name)
FROM     students x, x.takes y, y.is_taught_by z
WHERE    z.rank='full professor'
```

Цей запит слід інтерпретувати так.

- Змінна x набуває значень з екстента students. Це незалежна змінна.
- Змінна y не є незалежною. Для конкретного поточного значення змінної x областю значень змінної y є ті об'єкти з екстента Section, з якими пов'язаний поточний об'єкт зі students зв'язком takes (тобто ті розділи курсу лекцій, що прослуховує студент).
- Область значень змінної z визначається змінною y так: це ті об'єкти з екстента professors, які пов'язані з поточним об'єктом Section зв'язком teaches (відповідно до означення схеми бази даних будь-який розділ курсу лекцій читає лише один професор).

Зазначимо, що порядок переліку колекцій у фразі FROM є суттєвим.

### Фраза WHERE. Вираз з'єднання об'єктів

У фразі WHERE, як і в SQL, задається умова вибирання. Зазначимо, що умова може накладатися на атрибути шляхів доступу, що мають довільну глибину.

**Запит 14.3**

Визначити адреси тих людей, для яких:

- бабуся/дід мешкає на вулиці Main Street;
- бабуся/дід має двох або більше дітей;
- бабуся/дід, батько і дитина мають різні zip-коди адреси проживання

```
SELECT   g.address
FROM     persons p, p.children c, c.children g
WHERE    p.address.street = 'Main Street' AND
         COUNT(p.children) >= 2 AND
         c.address.zip != p.address.zip AND
         g.address.zip != c.address.zip
```

Реляційну операцію з'єднання двох або більше відношень можна застосовувати до об'єктів: об'єкти різних (або одного й того ж) типів можуть бути з'єднані за певною умовою.

**Запит 14.4**

Вибрати службовців, які працюють під керівництвом своїх дітей.

```
SELECT   e
FROM     employee e, e.children c, e.superior s
WHERE    c.passport = s.passport
```

У фразі WHERE відбувається з'єднання кожної дитини службовця з його керівником за умови, що вони (дитина службовця і його керівник) є однією особою Зі службовців, у яких є принаймні одна така дитина, формується вихідний результат, який має тип Bag<Employee>.

### Використання методів

У мові OQL можна звертатися до методів так само, як і до атрибутів. При цьому тип значення, яке повертає метод, має бути сумісним із тим OQL-виразом, де воно використовується. Не допускається звернення до методу, який повертає значення типу void. Якщо метод не має аргументів, під час звернення до нього після його імені або записуються символи (), або не записується жодних символів. Оскільки age – метод без аргументів з класу Person, що повертає вік особи, то наведений нижче запит виводить імена й вік усіх осіб, яким за 40 років, і які проживають у місті New York:

```
SELECT   p.name, p.age()
FROM     persons p
WHERE    p.address.city = 'New York  AND p.age() > 40
```

Якщо метод повертає складений об'єкт, його можна використовувати як зв'язок. Наприклад, метод sibling класу Person повертає братів і сестер для об'єкту цього класу. Наведений нижче запит виводить усю інформацію, що стосується осіб, яким за 60 років і які мають братів на ім'я Pete:

```
SELECT   p
FROM     persons p
WHERE    p.age > 60 AND p.sibling.name = 'Pete'
```

**Запит 14.5**

Вивести всю інформацію про людей віком до 30 років, які мають братів або сестер віком від 20 років.

```
SELECT   p
FROM     persons p
WHERE    p.age < 30 AND p.sibling.age > 20
```

Якщо метод має аргументи, то значення, які ним надаватимуться, слід розміщувати в круглих дужках.

**Запит 14.6**

Вивести імена, вік і предків усіх людей, відомості про яких зберігаються в базі.

Припустимо, що аргументом методу hasAncestor(q) є об'єкт Person, а повертає цей метод логічне значення, що вказує, чи є ідентифікована аргументом особа предком для особи, щодо якої застосований метод. Тоді реалізація запиту виглядатиме так:

```
SELECT   p.name, p.age, q.name, q.age
FROM     persons p, persons q
WHERE    p.hasAncestor(q)
```

### Агрегатні функції

Агрегатні функції мають такий синтаксис:

```
ім'я_агрегатної_функції(колекція_літералів)
```

Імена агрегатних функцій стандартні й запозичені з SQL: MAX, MIN, AVG, SUM, COUNT. Аргументом функції COUNT може бути будь-яка колекція. Наведемо приклади:

```
AVG(SELECT e.seniority FROM employees e)
COUNT(SELECT e FROM employees e)
COUNT(Employees)
MAX(SELECT p.age FROM persons p)
```

Для сумісності з SQL допускається використання агрегатних функцій у стилі цієї мови. У табл. 14.1 розглянуто реалізацію конструкцій, що містять агрегатні функції, мовами SQL та OQL.

**Таблиця 14.1.** Використання агрегатних функцій в мовах SQL та OQL

| Мова SQL | Мова OQL |
|---|---|
| `SELECT COUNT(*) FROM...` | `COUNT(SELECT * FROM...)` |
| `SELECT ім'я_функції(атрибути) FROM...` | `ім'я_функції(SELECT атрибути FROM...)` |
| `SELECT ім'я_функції(DISTINCT атрибути) FROM...` | `ім'я_функції(DISTINCT (SELECT атрибути FROM...))` |

Агрегатна функція може використовуватися у фразах SELECT, WHERE, GROUP BY, HAVING. Розглянемо приклади.

**Запит 14.7**

Визначити середню зарплату всіх службовців.

```
AVG(SELECT e.salary
    FROM   employees e)
```

**Запит 14.8**

Визначити мінімальну зарплату для тих службовців, які мають брата або сестру віком від 30 років.

```
SELECT MIN(SELECT e.salary
           FROM   TAs e
           WHERE  e.sibling.age > 30)
FROM   TAs
```

**Запит 14.9**

Визначити кількість службовців, у яких зарплата вища середньої.

```
SELECT COUNT(e)
FROM   employees e
WHERE  e.salary > AVG(SELECT x.salary FROM employees x)
```

### Фрази GROUP BY і HAVING

OQL надає ті самі засоби для групування, що і мова SQL. Загальний синтаксис запиту, в якому здійснюється групування, є таким:

```
SELECT...
FROM...
[WHERE...]
GROUP BY атрибути_групування
[HAVING предикат_на_групах]
```

Обчислення запиту виконується так:

1. Вибираються елементи, що відповідають умові, зазначеній у фразі WHERE.
2. Відібрані елементи групуються за наборами значень усіх атрибутів групування.
3. Із груп відбираються ті елементи, для яких записаний у фразі HAVING предикат є істинним.
4. У фразі SELECT можуть вказуватися лише атрибути групування та агрегатні функції, що діятимуть у межах групи

Розглянемо приклади.

**Запит 14.10**

Для кожного факультету, де працюють службовці, визначити кількість службовців, зарплата яких вища за середню.

```
SELECT   e.facultyName, COUNT(*)
FROM     employees e
WHERE    e.salary > AVG(SELECT x.salary
                        FROM   employees x)
GROUP BY e.facultyName
HAVING   COUNT(SELECT *
               FROM   employees y
               WHERE  y.facultyName = e.facultyName) > 0
```

**Запит 14.11**

Для кожного міста, де мешкає більше двох працівників чоловічої статі, обчислити їхню середню зарплату.

```
SELECT   e.address.city, AVG(e.salary)
FROM     employees e
WHERE    e.sex = "M"
GROUP BY e.address.city
HAVING   COUNT(SELECT *
               FROM   partition x)>2
```

Типом результату запиту є Bag<Struct(city: string, salary: integer)>.

Звернемо увагу на використання службового слова partition — це ім'я поточної колекції, одержаної під час групування.

**Запит 14.12**

Вивести середню зарплату працівників на тих факультетах, де вона перевищує 30 000.

```
SELECT   e.facultyName,
         AVG_salary: AVG(SELECT x.salary
                         FROM   partition x)
FROM     employees e
GROUP BY facultyName
HAVING   AVG(SELECT x.e.salary
             FROM   partition x) > 30000
```

Типом результату є Bag<Struct(facultyName: string, AVG_salary: integer)>.

**Запит 14.13**

Для кожного міста визначити кількості службовців, що в ньому мешкають і заробляють менше 10 000, від 10 000 до 50 000 і більше 50 000

Стратегія реалізації запиту є такою. В екстенті employees слід виконати групування за чотирма елементами:

- назвою міста, де мешкає службовець;
- предикатом, який є істинним, коли зарплата менша 10 000;
- предикатом, який є істинним, коли зарплата перебуває в межах від 10 000 до 50 000;
- предикатом, який є істинним, коли зарплата більша 50 000.

Слід вивести значення елементів групування і підрахувати кількості елементів у групах. Мовою OQL запит записується так:

```
SELECT   City, Low, Medium, High, NumberOfEmployees: Count(*)
FROM     employees e
GROUP BY City: e.address.city,
         Low: e.salary <= 1000,
         Medium: e.salary > 10000 AND e.salary < 50000,
         High   e.salary >=50000
```

Після виведення результатів виконання запиту на екран монітора ми можемо отримати таку таблицю:

| City | Low | Medium | High | NumberOfEmployees |
|---|---|---|---|---|
| New York | TRUE | FALSE | FALSE | 57 |
| New York | FALSE | TRUE | FALSE | 33 |
| New York | FALSE | FALSE | TRUE | 12 |
| Paris | TRUE | FALSE | FALSE | 21 |
| Paris | FALSE | TRUE | FALSE | 113 |
| Paris | FALSE | FALSE | TRUE | 87 |
| ... | ... | ... | ... | ... |

### Впорядкування результату запиту

Вираз, за яким здійснюється впорядкування, задається у фразі ORDER BY. Результат select-запиту впорядковується за такими правилами:

- літерали впорядковуються відповідно до їхніх значень;
- об'єкти впорядковуються за літеральними значеннями атрибутів, розташованих у кінцях шляхів доступу.

Розглянемо приклади.

**Запит 14.14**

Упорядкувати працівників за статтю, а працівників однієї статі — за віком. Упорядкування здійснювати в зростаючому порядку.

```
SELECT   e
FROM     employees e
ORDER BY e.sex, e.age()
```

**Запит 14.15**

Упорядкувати працівників факультету інформаційних технологій за zip-кодом поштової адреси у спадному порядку

```
SELECT   e.name
FROM     employees e
WHERE    e.facultyName = 'IT'
ORDER BY e.address.zip DESC
```

Наявність у запиті фрази ORDER BY приводить до того, що типом результату стає List.

### Використання невизначених значень

OQL, як і SQL, дозволяє використовувати невизначені значення. Для цього застосовується тризначна логіка, що оперує значеннями TRUE, FALSE та UNDEFINED. Виконуються такі правила:

- посилання на невизначений об'єкт/літерал повертає логічне значення UNDEFINED;
- існують дві стандартні функції is_defined(expr) та is_undefined(expr) з такою семантикою: is_defined(expr) = FALSE, якщо expr = UNDEFINED, інакше is_defined(expr) = TRUE; is_undefined(expr) = NOT(is_defined(expr));
- якщо в результаті обчислення предиката, зазначеного у фразі WHERE чи HAVING select-запиту, отримано значення UNDEFINED, воно інтерпретується так само, як і значення FALSE;
- UNDEFINED може входити до складу будь-якої колекції, що конструюється явно чи неявно;
- UNDEFINED є допустимим значенням під час обчислення агрегатної функції COUNT;
- результат будь-якої іншої операції, принаймні одним з операндів якої є вираз зі значенням UNDEFINED, дорівнює UNDEFINED.

Наведемо приклад використання невизначених значень.

**Запит 14.16**

Визначити службовців, які не мають керівників.

```
SELECT   e
FROM     employees e
WHERE    is_undefined(e.superior)
```

### Область дії змінних

Фраза FROM допускає оголошення змінних на відповідних колекціях. Такі змінні можуть означуватися явно (наприклад, FROM employees e) або неявно (наприклад, FROM employees). У другому випадку ім'я колекції employees використовується як змінна у посиланнях на атрибути об'єктів. Область дії такої змінної — увесь select-запит разом із підзапитами.

Фраза GROUP BY неявно вводить ім'я групи — partition, точніше ім'я поточної колекції групи, і допускає оголошення імен для атрибутів, за якими здійснюється групування. Область дії цих імен — фрази SELECT, GROUP BY, HAVING разом із підзапитами. Наведемо приклад.

**Запит 14.17**

Для кожного міста, де проживають службовці, старші 20 років, визначити їхню середню зарплату.

```
SELECT   cty, AVG(e.salary)
FROM     employees e
WHERE    e.age > 20
GROUP BY cty: e.address.city
HAVING   COUNT(SELECT *
               FROM partition x)>0
```

У фразі SELECT використані посилання на змінну cty, що оголошена у фразі GROUP BY та змінну e, що оголошена у фразі FROM.

### Використання кванторів

У мові OQL є два квантори: існування та загальності. Вираз із квантором існування виглядає так:

```
EXISTS x IN колекція : логічний_вираз
```

Вираз набуває істинного значення, якщо множина елементів колекції, для яких логічний вираз є істинним, не порожня. Логічний вираз має бути предикатом, що залежить від змінної x.

Квантор загальності застосовується в такий спосіб:

```
FOR ALL x IN колекція · логічний_вираз
```

Вираз набуває істинного значення, якщо для всіх елементів колекції логічний вираз є істинним.

Наведемо приклади виразів з кванторами:

```
EXISTS x IN employees : x.salary > 60000
```

```
FOR ALL x IN students : x.student_id > 0
```

### Запити як функції бази даних

У мові OQL є можливість означувати параметризовані запити і надавати їм імена. Такі запити розглядаються як функції, що містять вільні змінні. Передбачається, що означення таких запитів додаються до схеми об'єктно-орієнтованої бази даних. Синтаксис означення параметризованого запиту такий:

```
define q(x1, x2, ..., xn) as e(x1, x2, ..., xn)
```

Тут q — ім'я запиту (функції), що означується. Воно не має збігатися з іменами наявних об'єктів, методів, функцій, класів; q(x1, x2, ..., xn) — прототип функції, що додається до схеми бази даних; e(x1, x2, ..., xn) — вираз OQL (зазначимо, що це може бути довільний вираз OQL, а не лише select-вираз); x1, x2, ..., xn — вільні змінні в e. Тип виразу q(...) визначається типом об'єкта e.

Наведемо приклад.

**Запит 14.18**

Визначити множину осіб, вік і стать яких задає користувач.

```
define persons_with_age_and_sex(x,y) as
  SELECT   p
  FROM     persons p
  WHERE    p.name = x AND p.sex = y
```

Слід відрізняти функції бази даних, тобто запити, що означуються, від методів класів. Перші не прив'язані до жодного класу й існують самостійно, а другі оголошуються як елементи класів.

В окремих випадках функції, що означуються, можуть не мати аргументів. Так, у підрозділі 14.4.2 було означено функцію без аргументів Joe.

### Суперпозиція функцій

Деякі функції (запити) можна вкладати одне в одного. Наведемо приклад.

**Запит 14.19**

Визначити вулицю, для якої середня зарплата службовців віком до 30 років, що на ній мешкають, є найнижчою.

1. Визначимо службовців віком до 30 років

```
define employeed_persons() as
  SELECT   e30
  FROM     employees e30
  WHERE    e30.age<30
```

2. Сформуємо мультимножину Bag<Struct(street: string, ave_salary: integer)>, де для кожної вулиці, на якій мешкають службовці віком до 30 років, буде визначено їхню середню зарплату.

```
define salary_map() as
  SELECT   street, ave_salary
                  AVG(SELECT x.e.salary
                      FROM partition x)
  FROM     employeed_persons() e
  GROUP BY street: e.address.street
```

3. Відсортуємо результати попереднього підзапиту за величиною зарплати.

```
define sorted_salary_map() as
SELECT   s
FROM     salary_map() s
ORDER BY s.ave_salary
```

Сортування результатів запиту salary_map приводить до того, що його тип Bag перетворюється на List.

4. За допомогою стандартної функції first визначимо вулицю з першого елементу списку, тобто вулицю, якій відповідає найменша середня зарплата.

```
first(sorted_salary_map()).street
```

Усе це можна записати у вигляді одного запиту:

```
first(SELECT   street, ave_salary:
                  AVG(SELECT e.salary FROM partition)
                     FROM (SELECT   e30
                           FROM     employees e30
                           WHERE    e30.age<30) e
                     GROUP BY street: e.address.street
                     ORDER BY ave_salary).street
```

### Перетворення і зведення типів

У мові OQL є спеціальні функції та операції, які здійснюють перетворення і зведення типів об'єктів. Ці функції та операції описані в табл. 14.2.

**Таблиця 14.2.** Функції перетворення і зведення типів

| Функція | Зміст |
|---|---|
| element(e) | Вибір єдиного елементу з колекції. Наприклад:<br>`element(SELECT x FROM professors x WHERE x.name = 'Turing')`<br>Тут select-вираз повертає мультимножину типу Bag(Professor), яка складається з одного об'єкта. Застосування до цієї колекції функції element приводить до вибору згаданого єдиного об'єкта. Тип результату — Professor |
| listtoset(e) | Перетворення списку на множину, наприклад результатом функції listtoset(list(1,2,3,2)) є множина (1,2,3) |
| flatten(e) | Об'єднання вкладених структур. Діє лише на першому рівні вкладеності. Наприклад:<br>`flatten(list(set(1,2,3).set(3,4,5,6).set(7))) = set(1,2,3,4,5,6,7)`<br>`flatten(list(list(1,2),list(1,2,3))) = list(1,2,1,2,3)`<br>`flatten(set(list(1,2),list(1,2,3))) = set(1,2,3)` |
| (t)e | Зведення типу: тип виразу e зводиться до типу t. Наприклад:<br>`SELECT (Employee) p FROM persons p WHERE 'has a job' IN p.activities`<br>Тут у фразі SELECT тип Person змінної p зводиться до типу Employee. Тип результату — Bag<Employee> |

**Спеціальні можливості, які надаються для роботи зі списками**

Щоб можна було обробляти списки, мовою OQL надаються такі додаткові можливості:

- вибір *i*-го елементу списку, наприклад `List(a, b, c, d)[2]` повертає елемент b;
- вибір діапазону елементів і створення з них нового списку, наприклад `List(a, b, c, d)[2 : 4]` повертає список `List(b, c, d)`;
- вибір першого й останнього елементів списку: `first(e)`, `last(e)`;
- конкатенація списків, так `List(1,2) + List(2,3)` повертає список `List(1,2,2,3)`.

# 14.5. Архітектура ООСКБД

Є різні підходи до створення ООСКБД. Крім розробки власне об'єктно- орієнтованих СКБД, є численні способи поєднання об'єктно-орієнтованої та реляційної моделей даних. Усі ці підходи ми розглянемо далі.

## 14.5.1. Розширення реляційних СКБД

Реляційні СКБД надають можливість звертатися до них програмам, написаним різними, зокрема об'єктно-орієнтованими, мовами програмування. У цьому випадку об'єктно-орієнтовані прикладні програми виконують усі функції, пов'язані з відображенням об'єктної моделі в реляційну, тобто перетворюють об'єкти на структури даних, які можуть бути безпосередньо записані в табличні БД, підтримують властивості успадкування, інкапсуляції, зв'язування з об'єктами їхніх методів.

РСКБД бере на себе єдину функцію – зберігання даних, які пов'язані з об'єктами, причому зберігання у вигляді реляційних таблиць, усе інше виконує прикладна програма.

Даний підхід передбачає включення до складу РСКБД засобів, які полегшують процес відображення об'єктів у базі даних і маніпулювання ними. Тобто сама РСКБД удосконалюється, полегшуючи обробку об'єктів, але залишається при цьому реляційною. До можливих розширень РСКБД належать такі.

- Надання можливості автоматичного створення унікальних ідентифікаторів кортежів відношень. Наприклад, в Oracle є тип `rowid`, який виконує цю функцію. Такий тип сприяє вирішенню проблеми підтримки OID, хоча й не вирішує її повністю, оскільки OID має бути унікальним у всій базі даних, а не в межах одного відношення, як `rowid`.
- Створення механізмів означення нових типів даних (такі механізми є в мовах програмування). Введення нових типів, навіть якщо вони відповідають досить складним структурам даних, не суперечить реляційній моделі. Інша річ, що реляційна модель в «чистому» вигляді не надає можливості працювати зі складеними структурами даних, вони розглядаються як атомарні. Але стандарт

SQL-3 допускає означення структурованих типів і елементів типу масиву та використання їх в означенні таблиці, наприклад:

```
CREATE TYPE Publisher AS
          (name varchar(20),
           branch varchar(20))

CREATE TYPE Book AS
          (title varchar(20),
           author-array varchar(20) array[10],
           pub-date date,
           publisher Publisher))

CREATE TABLE books OF Book
```

**Успадкування типів і відповідних їм таблиць**

У SQL-3 допускається успадкування типів:

```
CREATE TYPE Person AS
          (name varchar(20),
           address varchar(20))

CREATE TYPE Student UNDER Person AS
          (degree varchar(20),
           department varchar(20))

CREATE TYPE Teacher UNDER Person AS
          (salary integer,
           department varchar(20))
```

Допускається означувати ієрархії таблиць на базі ієрархії типів, наприклад, нехай задане означення таблиці:

```
CREATE TABLE people OF Person
```

Тоді можна означити підтаблиці:

```
CREATE TABLE students OF Student UNDER people
CREATE TABLE teachers OF Teacher UNDER people
```

Кожний кортеж з підтаблиць (students і teachers) неявно присутній в супертаблиці (тобто people).

Перевага підходу, який базується на розширенні реляційних СКБД, полягає в тому, що надається можливість використовувати всю потужність реляційних систем баз даних. Недолік – слабка розвиненість засобів зображення об'єктів і маніпулювання ними, багато з цих функцій виконують прикладні програми.

## 14.5.2. Створення самостійних ООСКБД

Об'єктно-орієнтовані СКБД реалізують гнучку модель даних, яка базується на тій же парадигмі, що й об'єктно-орієнтовані мови програмування. ООБД забезпечують глибшу інтеграцію з об'єктно-орієнтованими застосуваннями, ніж реляційні бази даних, і мінімізують обсяг роботи з програмування збереження і вибирання об'єктно-орієнтованих даних.

Переваги використання однакових моделей у застосуванні та базі даних виявляються тоді, коли об'єктно-орієнтовані моделі є складними. Якщо ієрархія успадкування є багаторівневою, колекції перетворюють граф об'єкта на павутину, застосуванням складно використовувати поліморфізм та посилання, об'єктна база даних робить такі застосування меншими і їхня продуктивність підвищується. Це знижує витрати на написання і налагодження програмного коду й підвищує загальну продуктивність застосувань

### Підтримка об'єктно-орієнтованих концепцій

Самостійні ООСКБД забезпечують повну підтримку об'єктно-орієнтованої парадигми. Це передбачає безпосередню інтеграцію з об'єктно-орієнтованими мовами програмування, підтримку об'єктних типів, зв'язків між об'єктами та операцій бази даних, які інтерпретують об'єкти (найчастіше об'єкти інтерпретуються як записи)

Прикладами об'єктно-орієнтованих операцій бази даних можуть бути: створення посилань на об'єкти і завантаження з бази даних об'єктів, на які є посилання, забезпечення блокування на об'єктному рівні й повернення об'єктів як результатів запитів або операцій з курсорами.

#### Інкапсуляція

Стабільна (persistent) поведінка класу є незалежною від проектування бази даних, яке не потребує змінення реалізацій класів.

#### Успадкування

Об'єкт інтерпретується одним записом незалежно від того, наскільки глибокою є ієрархія успадкування. Всі дані об'єкту запам'ятовуються в єдиному записі. Коли об'єкт вибирається з бази даних, усі дані базового класу та всі дані похідних класів завжди доступні.

#### Поліморфізм

Концепція полиморфізму тісно пов'язана з успадкуванням. Стосовно об'єктно-орієнтованих баз даних це означає, що можна оперувати різними класами за допомогою спільного базового класу, одержуючи дані з об'єктів необхідного похідного класу. Тобто об'єкт базового класу може мати тип похідного класу. Якщо ви послідовно переглядаєте всі екземпляри базового класу, то будете одержувати всі об'єкти цього класу, незалежно від того, якого вони типу. Видалення екземпляра базового класу призводить до видалення всього об'єкту, включаючи дані похідних об'єктів.

#### Ідентифікованість об'єкта

ООСБД поєднують ідентифікованість об'єкта в базі даних з ідентифікованістю об'єкта в оперативній пам'яті. Якщо виконується збереження об'єкта, то ООСБД «знає», чи відповідає він об'єкту з бази даних (чи є він у базі даних), а під час вибирання об'єкта з бази даних - чи є він у пам'яті. Програмісту не потрібно власноруч підтримувати відповідність між об'єктами бази даних і об'єктами в пам'яті.

#### Посилання на об'єкти

Самостійні ООСКБД дають можливість створювати в пам'яті посилання на об'єкти, а потім відображувати їх у базі даних і навпаки.

### Означення схеми ООБД

Суттєвою відмінністю між реляційною і об'єктно-орієнтованою базами даних є та легкість, з якою будується схема об'єктної БД. Оскільки ООБД підтримує зв'язки типу ієрархій успадкування і посилання між об'єктами, то досить легко безпосередньо скопіювати описи об'єктно-орієнтованих класів у схему бази даних. Схеми бази даних будуються безпосередньо за означеннями стабільних класів.

### Маніпулювання об'єктами в ООБД

Об'єктно-орієнтованим застосуванням набагато легше оперувати даними, що зберігаються в базі, коли використовується ООБД, а не реляційна база даних. Далі будуть розглянуті способи реалізації базових операцій з обробки даних у тому випадку, коли об'єктно-орієнтоване застосування використовує ООБД.

#### Збереження об'єктів

Оскільки об'єкт зберігається як єдиний елемент, то стає набагато легше реалізувати стабільну поведінку в методах класу.

Наприклад, ООСКБД FastObjects автоматично генерує стабільні методи стабільних класів. Окрім того, автоматично забезпечуються ідентифікація стабільних об'єктів, збереження членів базового і похідних класів, узгодження даних зі схемою бази даних та робота з іншими об'єктами, на які є посилання.

Клас реалізує специфічну стабільну поведінку й дає змогу операціям збереження даних мати об'єктно-орієнтований інтерфейс. Стабільні об'єкти створюються і оголошуються як стабільні в контексті транзакції і запам'ятовуються в базі явно або неявно, коли транзакція успішно завершується.

#### Вибирання об'єктів

Вибирання об'єктів з бази даних складається з двох етапів:

- пошук об'єкта;
- отримання об'єкта.

Отримання даних є складовою частиною стабільної поведінки класу. У FastObjects відповідні стабільні методи генеруються автоматично. Пошук об'єктів може здійснюватися так:

- множина об'єктів дістається в результаті виконання запиту;
- виконується безпосередній пошук поіменованого кореневого об'єкта;
- об'єкти знаходяться шляхом навігації посиланнями на об'єкти;
- об'єкти відшукуються шляхом ітеративного перегляду екстенту.

ООСКБД FastObjects дає змогу виконувати операції над посиланнями на об'єкти без отримання дозволів об'єктів. Завдяки цьому можна ітеративно переглядати екстент об'єктів класу або працювати з колекціями без додаткових витрат, пов'язаних із вибиранням об'єктів, на які є посилання. Коли здійснюється доступ до певного об'єкта бази, він у разі потреби блокується, після чого створюється його відображення в оперативній пам'яті, яке заповнюється даними.

#### Оновлення об'єктів

Оновлення об'єктів здійснювати простіше, ніж їхнє запам'ятовування. Під час оновлення не потрібно виконувати спеціальну операцію, що вимагає зв'язування

копії об'єкта в пам'яті з його зображенням у базі даних, оскільки ідентифікованість об'єктів підтримується для обох копій. Взаємодія між об'єктом і базою даних є частиною стабільної поведінки.

Коли стан знайденого об'єкта змінюється в пам'яті, він зберігається в базі даних після успішного завершення транзакції. Якщо транзакція завершується аварійно, стан об'єкта в базі залишається незмінним.

#### Видалення об'єктів

Якщо об'єкту відповідає запис, його видалення є досить простим: слід звернутися до функції бази даних, що здійснює видалення, і передати їй посилання на об'єкт.

#### Запити

Наявність об'єктно-орієнтованих баз даних, які підтримують зв'язки між об'єктами, вимагає модифікації стандарту SQL, оскільки ця мова розроблена для реляційної моделі. До складу стандарту ODMG включено мову запитів до об'єктів (OQL), огляд засобів якої було зроблено у підрозділі 14.4.

Синтаксис OQL подібний до синтаксису SQL, проте припускає використання об'єктно-орієнтованих зв'язків, наприклад посилань між об'єктами. Оскільки OQL підтримує об'єктно-орієнтовані зв'язки, то немає необхідності з'єднувати таблиці за допомогою операції JOIN або будувати віртуальні таблиці. Все це дає можливість легко будувати запити й швидко їх виконувати.

#### Переваги та недоліки

Перевагою ООСКБД є їхня повна узгодженість із об'єктно-орієнтованою парадигмою програмування, що знімає всі проблеми, пов'язані зі зберіганням і маніпулюванням об'єктами у базі даних.

Основний недолік пов'язаний з тим, що для самостійних ООСКБД слід вирішувати весь комплекс проблем, пов'язаних із СКБД, які вже вирішені в наявних реляційних СКБД.

### 14.5.3. Об'єктно-реляційні СКБД

Особливість даного підходу полягає в тому, що на базі наявних реляційних СКБД реалізується об'єктно-орієнтований інтерфейс. Робота з цим інтерфейсом здійснюється так само, як і в ООСКБД, але всі проблеми, пов'язані зі створенням і веденням баз даних, вирішуються в реляційній СКБД.

Основна проблема, пов'язана зі створенням такого інтерфейсу, – відображення об'єктно-орієнтованої моделі в реляційну. Є кілька способів інтеграції об'єктного і реляційного підходів, що будуть розглянуті далі.

#### Об'єктно-реляційний шлюз

*Об'єктно-реляційний шлюз* автоматично виділяє об'єкти програми й зберігає їх у реляційній базі даних.

Об'єктно-орієнтоване застосування працює як звичайний користувач СКБД (рис. 14.5). Такий варіант дає змогу програмістам повністю сконцентруватися на об'єктно-орієнтованому проектуванні.

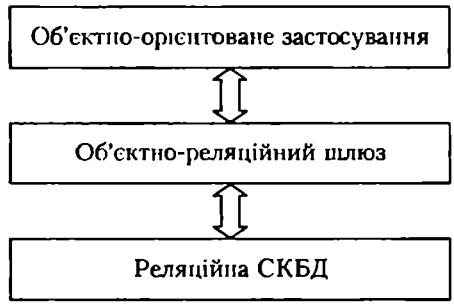



**Рис. 14.5.** Використання об'єктно-реляційного шлюзу

Деякі об'єктно-орієнтовані СКБД, наприклад GemStone, можуть самі виконувати роль потужного об'єктно-реляційного шлюзу, дозволяючи об'єктно-орієнтованим застосуванням звертатися до реляційних СКБД. Деякі об'єктно-реляційні шлюзи, наприклад Oadapter для СКБД Oracle, можна використовувати як ПО, що з'єднує об'єктно-орієнтоване застосування з РСКБД.

### Об'єктно-реляційний прошарок між об'єктною та реляційною СКБД

У разі використання *об'єктно-орієнтованого прошарку* програма взаємодіє з БД за допомогою мови ООСКБД, а прошарок замінює всі об'єктно-орієнтовані елементи цієї мови на їхні реляційні еквіваленти (рис. 14.6). За це доводиться розплачуватися продуктивністю. Окрім іншого, прошарок має перетворювати об'єкти на набори зв'язаних відношень, генерувати унікальні OID об'єктів і передавати ці дані до реляційної БД.


**Рис. 14.6.** Використання об'єктно-реляційного прошарку

### Уніфікована об'єктно-реляційна СКБД

Цей підхід передбачає створення гібридних об'єктно-реляційних СКБД, що можуть зберігати як табличні дані, так і об'єкти. Вважається, що майбутнє саме за гібридними СКБД. Сьогодні розробники реляційних СКБД починають додавати до своїх продуктів об'єктно-орієнтовані засоби.

З іншого боку, виробники об'єктних СКБД усвідомлюють, що об'єктно-орієнтовані бази даних у найближчому майбутньому не замінять реляційних СКБД. У зв'язку з цим вони створюють прошарки для підтримки реляційних баз даних або інші специфічні об'єктно-реляційні інтерфейси.

### Порівняння підходів

На рис. 14.7 наведено схематичне порівняння різних підходів до реалізації СКБД, що підтримують об'єктно-орієнтовану парадигму. Як бачимо, об'єктно-реляційний підхід має значні переваги порівняно з іншими.

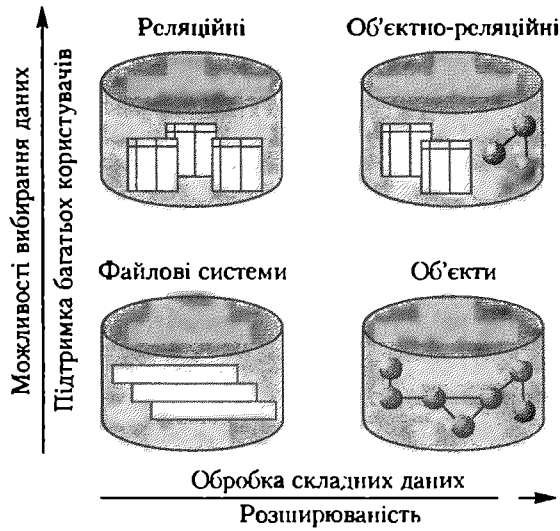



**Рис. 14.7.** Порівняння підходів до реалізації об'єктно-орієнтованих СКБД

## 14.6. Зображення об'єктної моделі в реляційній базі даних

Обговоримо проблеми, які виникають у разі подання об'єктної моделі в реляційній базі даних, а саме під час проектування схеми, маніпулювання об'єктами, формулювання запитів та їх виконання.

### 14.6.1. Проектування реляційної схеми для зберігання об'єктів

Схема бази даних визначає спосіб зберігання даних. У нашому випадку необхідно створити таблиці для зберігання об'єктів, що використовуються в об'єктно-орієнтованих застосуваннях. Обговоримо проблему створення схем, враховуючи основні властивості об'єктно-орієнтованої моделі.

### Інкапсуляція

Ключовою ідеєю в об'єктно-орієнтованому проектуванні є об'єднання програмного коду і даних для створення об'єкта. Оскільки реляційна модель даних не відображує відповідних понять, то для кожного класу об'єктів необхідно написати функції зі збереження даних об'єкта в базі і їх отримання звідти.

Реляційні СКБД не підтримують механізмів інкапсуляції, тобто «приховування» даних, за винятком механізму їх захисту. Найчастіше програма, що отримала доступ до бази, має один рівень повноважень. Усе, що потрібно програмі, стає загальнодоступним для неї. Область доступу програми можна обмежити шляхом створення загальнодоступних та приватних віртуальних таблиць для даних класу. Проте це допомагає лише в тому випадку, коли автор програми весь час пам'ятатиме, яку віртуальну таблицю в якій частині програми використовувати.

### Успадкування

Реляційна база даних не підтримує ієрархії успадкування, але її можна змоделювати кількома способами, що були детально розглянуті в підрозділі 7.3.5. При цьому класи-предки слід вважати супертипами, а класи-нащадки – підтипами

### Поліморфізм

Реляційна база даних не підтримує поліморфізму. Для забезпечення коректної реконструкції об'єкта під час зчитування даних інформацію про його тип має збе рігати застосування.

### Ідентифікація об'єктів

Аби гарантувати, що в різних рядках таблиці не зберігаються однакові OID, слід зробити поле OID первинним ключем. Це дає змогу використовувати первинний ключ як спосіб ідентифікації об'єктів. На жаль, як тільки об'єкт буде завантажено в оперативну пам'ять, його ідентифікованість втрачається. Тому немає захисту від завантаження в пам'ять кількох копій одного об'єкта, а це може призвести до суперечливих оновлень.

### Посилання об'єктів на об'єкти

Посилання об'єктів на об'єкти моделюються за допомогою зовнішніх ключів. Будь-яке посилання на об'єкт зображується у вигляді зовнішнього ключа певної таблиці. Під час збереження в базі даних об'єкта програма має переглянути всі його зв'язки з іншими об'єктами і гарантувати, що для кожного об'єкта, на який є посилання, створюється відповідний зовнішній ключ. Якщо об'єкт вибирається з бази даних, необхідно переглянути всі його зовнішні ключі, вибрати відповідні ним об'єкти й у коректний спосіб ініціалізувати посилання на них.

## 14.6.2. Маніпулювання об'єктними даними

У попередньому підрозділі ми обговорили проблеми проектування реляційної схеми для зберігання об'єктно-орієнтованих даних. Було зазначено, що такі поняття, як ідентифікованість об'єктів, типи даних і посилання між об'єктами підтримуються не самою реляційною моделлю даних, а лише додатковими механізмами.

Використання реляційної бази даних для зберігання, пошуку, оновлення і видалення об'єктів також є нелегким завданням. Нижче розглядаються методи маніпулювання об'єктно-орієнтованими даними в реляційній базі.

### Зберігання об'єктів

#### Визначення таблиць, що застосовуватимуться для зберігання даних об'єкта

Більшість об'єктів запам'ятовуються в багатьох таблицях у зв'язку з тим, що вони входять до складу ієрархії успадкування, мають колекції посилань і можуть містити дані складних типів, зокрема інші об'єкти.

Оголошення кожного класу має містити детальну інформацію про фрагмент схеми бази даних і спосіб зберігання в ній об'єктів класу.

#### Створення первинних ключів таблиць

Первинні ключі таблиць, які використовуються для зберігання складових частин об'єкта, надають можливість зв'язувати рядки, що відповідають одному об'єкту. Використання первинного ключа гарантує, що OID буде унікальним і тому дані, які належать різним об'єктам, не переплутаються.

Необхідність стеження за значеннями багатьох первинних ключів зв'язує структуру бази даних зі структурами класів, які ускладнюються у зв'язку з необхідністю включення додаткових даних, що не стосуються логіки застосування.

#### Створення або відновлення зовнішніх ключів під час вставки об'єкта

Необхідно перетворити кожне посилання в об'єкті, що вставляється, на зовнішній ключ, який посилається на збережений у базі даних об'єкт. Створення ключа є необхідним, оскільки посилання на об'єкти, що зберігаються в оперативній пам'яті, безпосередньо не відображуються в базі даних.

#### Отримання й збереження даних об'єкта

Інкапсуляція вимагає, щоб лише об'єкт мав доступ до своїх приватних даних. У зв'язку з цим необхідно реалізувати метод класу, який отримує дані об'єкта та зберігає їх у реляційній базі. Цей метод має бути виконаний у межах однієї транзакції, що гарантуватиме збереження даних об'єкта.

#### Збереження об'єктів, на які є посилання

Коли зберігається певний об'єкт, може виникнути необхідність зберегти ті об'єкти, на які він посилається. Залежно від семантики зв'язків між об'єктами, збереження об'єктів, на які є посилання, слід виконувати в тій самій транзакції, що й збереження вихідного об'єкта, або в іншій.

З кожним об'єктом може бути зв'язана «мітка оновлення», яка дозволяє вказати, що об'єкт має бути збережений у базі даних або змінений у ній. Це дасть можливість зменшити кількість зайвих операцій під час збереження об'єктів. на які є посилання.

#### Зв'язування посилань у пам'яті з ключами

Важливо запобігти створенню багатьох копій одного й того ж об'єкта в оперативній пам'яті. Велика кількість копій може порушити семантику об'єктної моделі. Це означає, що застосування повинно мати спосіб визначення, чи завантажений об'єкт із бази даних у пам'ять. Для цього можна зберегти в пам'яті таблицю, яка

міститиме відомості про зв'язки між первинними ключами та ідентифікаторами об'єктів у пам'яті.

### Вибирання об'єктів

#### Отримання інформації про тип об'єкта

Для того щоб правильно побудувати об'єкт, необхідно знати ієрархію успадкування класів. Наприклад, об'єкт `Одяг` може бути вибраний як `Товар`, але в цьому випадку не буде отримано інформації про розмір одягу

#### Визначення, чи міститься об'єкт у пам'яті

Якщо є таблиця відповідності первинних ключів об'єктів та їхніх ідентифікаторів у пам'яті, за нею можна визначити, чи міститься об'єкт у пам'яті. Якщо він там є, то за вимогою на зчитування даних із бази потрібно повернути посилання на об'єкт, а не створювати в пам'яті його нову копію.

#### Створення нового об'єкта

Якщо в базі даних збережена вся інформація про тип об'єкта, то можна викликати відповідний конструктор класу для створення нового екземпляра, і потім заповнити об'єкт даними з бази. Цей об'єкт буде мати відповідні йому методи, й поліморфізм підтримуватиметься.

#### Ініціалізація нового об'єкта даними з бази

Як уже було зазначено, має існувати метод, який отримує дані об'єкта й зберігає їх у базі даних. Так само має бути розроблений метод, що може знайти дані об'єкта в базі і відновити його стан у пам'яті.

#### Завантаження у пам'ять об'єктів, на які є посилання

Якщо об'єкт посилається на інші об'єкти, то залежно від семантики цих посилань може виявитися необхідним завантажити у пам'ять зазначені об'єкти та ініціалізувати посилання на них у початковому об'єкті.

### Оновлення об'єктів

Оновлення об'єктів аналогічне їхньому запам'ятовуванню, проте під час виконання оновлення необхідно знати первинний ключ об'єкта, що оновлюється. Оновлення має виконуватися в межах певної транзакції.

### Видалення об'єктів

Слід стежити за тим, щоб відомості про об'єкт коректно видалялися з усіх таблиць, де вони зберігаються. Видалення має виконуватися в межах певної транзакції.

## 14.6.3. Виконання запитів

Специфікація SQL-запиту, що обробляє об'єктні дані, є досить складною, оскільки необхідно враховувати семантику даних об'єкта і структуру таблиць, у яких ці данні зберігаються Для цього потрібно пам'ятати, які таблиці використовуються для зберігання даних об'єкта і в який спосіб використовуються первинні й зовнішні ключі для встановлення зв'язків між цими таблицями.

Проблема стає ще складнішою, коли в запиті використовується об'єкт, на який є посилання. У цьому випадку обидва об'єкти мають взаємодіяти, оскільки жоден із них «не знає», якою є відображена в базі структура зв'язків іншого об'єкта.

Результатом запиту є таблиця, що містить відомості більше ніж про один об'єкт. Тому необхідно знайти спосіб переглядати результати запиту. Для цього зокрема можна скористатися курсорами. У деяких випадках може виникнути потреба повернути значення ключів знайдених об'єктів, в інших випадках необхідно завантажити у пам'ять об'єкти й повернути посилання на них. Під час пошуку слід уникати створення в пам'яті багатьох копій об'єктів.

### 14.6.4. Недоліки й обмеження, пов'язані із зображенням об'єктної моделі в реляційній базі даних

Ми показали, як можна використовувати реляційну базу даних для зберігання об'єктів й описали способи реалізації таких важливих операцій, як запам'ятовування, вибирання, оновлення й видалення даних, а також формулювання запитів. Розгляд лише цих базових операцій показує, що зображення об'єктної моделі в реляційній базі даних призводить до виникнення проблем із програмуванням об'єктно-орієнтованих застосувань і має низку недоліків, які будуть описані далі.

#### Необхідність підтримки двох різних парадигм

Реляційна та об'єктна моделі відрізняються одна від одної і тому в застосуваннях необхідно підтримувати дві різні моделі даних. Створивши ієрархічну структуру класів, слід спроектувати під них реляційну схему бази даних. Після створення схеми бази даних необхідно повернутися до проектування класів і визначити, яким чином реалізувати стабільну поведінку кожного з них з урахуванням створеної схеми бази даних.

#### Складність відображення між двома проектними рішеннями

Створення відображення між об'єктно-орієнтованим застосуванням і реляційною схемою бази даних — це не тривіальне завдання. Реляційна модель може лише апроксимувати об'єктно-орієнтовану ієрархію, тому багато її переваг втрачаються під час збереження і вибирання даних. Необхідно розробити складний програмний продукт для збереження об'єктної моделі в таблицях, подальшого вибирання даних із них та відновлення об'єктної моделі в оперативній пам'яті Виникає також проблема продуктивності, пов'язана з необхідністю постійного реконструювання складних зв'язків між об'єктами.

#### Порушення інкапсуляції, пов'язане з тим, що класи залежать від схеми бази даних

Реалізація стабільної поведінки кожного класу шляхом запам'ятовування даних у реляційній базі вимагає написання коду, який «поділяє» об'єкт і зберігає значення його полів у конкретних таблицях реляційної схеми, спроектованих спеціально для цього. Це приводить до утворення залежності між реалізаціями класу й реляційною схемою. Якщо необхідно змінити один з таких компонентів, то слід

змінити й інший. Тобто змінення реалізації класу вимагає перетворення схеми бази даних, а змінення схеми бази даних означає необхідність компіляції нової версії застосування.

### Перекладення на застосування відповідальності за безпеку та цілісність даних

Оскільки реляційна база даних «не розуміє» об'єктно-орієнтованої парадигми, то типові для мови програмування механізми захисту даних і правила поведінки не відтворюються в базі даних. Трансляцію поведінки об'єкта в операції над реляційною базою може здійснювати лише створений програмістом код. Ваша програма має точно відтворити об'єкт, зчитуючи його дані з бази. Вона повинна гарантувати коректне зчитування з бази даних усіх вершин ієрархії об'єктів. Лише ваше застосування може підтримувати унікальну ідентифікацію об'єкта і гарантувати необхідне блокування записів під час надання спільного доступу до даних об'єкта.

## Контрольні запитання та завдання

1. Перелічіть основні складові об'єктно-орієнтованої моделі даних.
2. Назвіть основні положення об'єктної моделі даних ODMG.
3. Які типи даних означені в мові опису об'єктів ODL ODMG?
4. Які є різновиди об'єктів у ODL ODMG?
5. Що таке «колекція»? Які є вбудовані типи колекцій?
6. Що таке «структура»? Які стандартні операції означені для структури?
7. Що таке «клас»? Які є характеристики класу?
8. Як специфікуються зв'язки між двома класами? Які є типи зв'язків?
9. Що таке «вирази конструювання?
10. Як використовуються конструктори типів?
11. Що таке «шлях доступу»?
12. Які елементи можна вказувати у фразі FROM?
12. Як використовуються методи у мові OQL?
14. В яких фразах можуть використовуватися агрегатні функції?
15. Якою є семантика фрази GROUP BY?
16. Для чого потрібна фраза HAVING?
17. Як впорядкувати результати запиту?
18. Як використовуються невизначені значення в мові OQL?
19. Які квантори є в мові OQL?
20. У який спосіб можна означити запит як функцію бази даних?
21. Як здійснюється перетворення типів у мові OQL?
22. Які є підходи до реалізації об'єктно-орієнтованих СКБД?

# Розділ 15

# Бази знань

- Моделі зображення знань: продукційна, фреймова, семантичні мережі
- Розширення реляційної моделі даних
- Нечіткі дані
- Механізми виведення даних

## 15.1. Коли дані стають знаннями

### Метадані — дані про дані

Що більше ми знаємо, чим є дані, то інформативнішими вони є. Розширивши відомості про дані, можна надати їм статусу знань.

Нехай у базі є дані «17.01.2005». Якщо дата — це єдина інформація, яку можна почерпнути з бази про ці дані, то вони є малоінформативними. Якщо ж із бази даних можна буде встановити, що це — дата здачі іспитів студентами 305, 306 і 307 груп 3 курсу кафедри інженерії програмного забезпечення факультету комп'ютерних наук Національного авіаційного університету, то ця дата стає набагато інформативнішою. У даному прикладі семантика дати розширюється завдяки встановленню зв'язків з іншими поняттями, такими як іспит, студент, група, кафедра, університет. Звичайна база даних дає змогу встановлювати зв'язки одних сутностей з іншими. Проте в базі не можуть зберігатися неповні, нечіткі, суперечливі дані та інформація про те, що дані є саме такими.

### Активність даних

Будь-які зміни певних даних можуть спричинити зміни інших даних, які, у свою чергу, приводять до подальших змін і т. д. Наприклад, поява нового співробітника приведе до зміни кількості співробітників відповідного підрозділу, штатного розкладу і фонду заробітної платні, відбудеться перерозподіл обсягів робіт тощо. Зміна будь-яких даних обумовлюється настанням певної події, яка може не лише приводити до цієї зміни, але й ініціювати сукупність інших дій.

### Дані у просторі та часі

Сутності будь-якої предметної області мають багато властивостей і найважливішими з них є ті, що пов'язані з існуванням сутностей у просторі та часі. Такі властивості мають фіксуватися в базі даних незалежно від того, чи вказується це проектувальником системи явно. Якщо йдеться про простір, то СКБД, що претендує

на інтелектуальність, має «розуміти», що означає «поставити на», «посунути якомога ближче до», «пересунути в крайній лівий кут», «обернути на певний кут навколо певної осі» тощо. Якщо СКБД відображує часові характеристики, то в ній має фіксуватися час настання подій. Проблема полягає не в тому, щоб запрограмувати ті чи інші просторові операції або явним способом визначити часові атрибути сутностей (дата введення, дата видалення, дата заміни), а в тому, щоби відтворити семантику простору і часу в самій системі.

### Ускладнення моделі даних

Реляційна модель відображує найпростіші структури даних і надає високорівневу мову маніпулювання даними. У реальному світі ми маємо справу з об'єктами довільної складності. Отже, в реляційній базі даних доводиться зображувати складні об'єкти у вигляді сукупностей простих, хоча це й не сприяє адекватному відображенню предметної області.

Розглянемо простий приклад. Реляційна модель дає змогу зобразити відношення R(Батько, Дитина), в якому для кожної особи вказується її дитина, причому єдина (відношення має перебувати в 1НФ). Якщо в людини кілька дітей, ми повинні зображувати це дублюванням значень поля Батько, відтак і дублюванням екземплярів відповідної сутності. Можна стверджувати, що таке відношення не адекватно відображує предметну область, і про це мають пам'ятати всі, хто працюють з базою даних. Природнішим було б зображувати предметну область у вигляді відношення R(Батько, Діти), коли з кожним батьком асоціюються всі його діти. Іншими словами, для адекватнішого зображення семантики предметної області бажано відмовитися від нормалізованості відношень, зберігаючи при цьому високорівневу мову маніпулювання даними. Таке ускладнення структури даних насамперед має відображуватися на концептуальному й зовнішньому рівнях. Внутрішній рівень СКБД може підтримуватися звичайними нормалізованими відношеннями за умови, що є формальні алгоритми опису відображень між рівнями.

### Надання семантики зв'язкам між сутностями

Реляційна модель дає змогу зв'язувати сутності за допомогою зовнішніх ключів, але ці зв'язки можуть мати лише невелику кількість властивостей (наприклад, обов'язковість існування батьківської сутності для підлеглої). Проте багато дослідників зазначають, що реально існує понад 200 типів зв'язків із різною семантикою, які бажано відображувати в базі даних. Уже згадувалося про необхідність відображення семантики двох фундаментальних типів властивостей об'єктів реального світу — просторових і часових. Взагалі, що більше типів зв'язків з притаманними їм властивостями фіксується в базі даних, то більш інформативною є така база.

### Виведення нових даних

У базі даних зберігаються лише дані або факти, що стосуються моделюваної предметної області. Будь-які дані зв'язані з багатьма похідними даними, які можуть бути отримані шляхом відповідного виведення. Традиційні бази даних не дають змоги описувати правила виведення нових даних (фактів, понять) з наявних. Перші кроки в цьому напрямку були зроблені в дедуктивних базах даних.

Проте в них використовується лише один метод виведення – дедуктивний, а людина в своїх міркуваннях застосовує й інші методи. Використання ширшого набору методів виведення даних сприяє трансформації баз даних у бази знань.

## 15.2. Постулати систем баз даних

Традиційні системи керування базами даних висувають кілька вимог до того, якою має бути модель предметної області, що фіксується в базі. Ці вимоги можна охарактеризувати за допомогою чотирьох постулатів.

### Постулат єдності

Цей постулат стверджує, що існує єдина предметна область, яка вивчається однією особою, в результаті чого формується єдина база даних. У цьому твердженні слово «єдиний» не можна сприймати категорично. Предметна область може бути поділена на сукупність предметних областей, що в певних ситуаціях можуть розглядатися незалежно. Проте, враховуючи мету вивчення предметної області, вона має розглядатися як єдине ціле. Тобто вивчається не сукупність предметних областей (взаємозв'язаних, але різних), а одна предметна область. Далі база даних проектується колективом, але це колектив однодумців в тому розумінні, що перед ними поставлено завдання сформувати не множину різних уявлень про предметну область, а єдине адекватне уявлення. Цей постулат не допускає існування, відтак і фіксації, в базі даних множини уявлень про предметну область, сформованих у свідомості різних осіб, оскільки їхні погляди на предметну область, що певним чином доповнюють одне одного, можуть базуватися на різних поняттях і суперечити одне одному. Проте база даних має бути несуперечною.

### Постулат про категоричність

Цей постулат можна викласти словами з Біблії: «Але таким буде слово ваше: так, так; ні, ні; а що понад це, то від лукавого». Категоричність свідчить про те, що факти, які виводяться з бази даних, можуть бути лише істинними або хибними. Ніяка інша ситуація, окрім «знаю» або «не знаю», неможлива. Більше того, неможливі відтінки істинних значень на кшталт «цілком можливо», «в дев'яти випадках з десяти істинно», «ймовірно істинно» тощо.

### Постулат достовірності

Суть постулату полягає в тому, що знання, закладені в базу даних, відповідають дійсності. Даний постулат означає, що істинність або хибність визначень не викликає сумнівів. Усе, що істинне в базі даних, насправді має місце в предметній області, а все що хибне – не має й не може мати місця. Цей постулат відкидає ситуацію, коли знання в базі даних можуть не відповідати реаліям. Із нього також випливає, що самі знання про предметну область є несуперечними (жодний факт не може бути оцінений як істинний і як хибний водночас). Достовірність вимагає беззастережної віри в істинність фактів, що виводяться з бази даних, не лише тому, що система баз даних не припускається помилок, але й тому, що всі закладені в неї знання вважаються істинними.

**Постулат про всезнання**

Цей постулат стверджує, що проектувальник бази даних вивчив предметну область у обсязі, достатньому для досягнення поставлених практичних цілей. Він знає все, що йому слід було б знати, і в нього немає жодних сумнівів у достатності своїх знань. Тому вважається, що спроектована база даних повністю відображує предметну область.

Звичайно, розглянуті постулати ідеалізують реальну ситуацію. Колектив, який вивчає предметну область, може й не прийти до спільних позицій. І тоді рішення мають бути компромісними, а компроміс позбавляє впевненості у правильності ухвалених рішень. У цьому випадку було б доцільно організувати базу даних так, щоб вона акумулювала думки всіх експертів і дозволяла вирішувати завдання у неоднозначно визначеному середовищі. Не завжди можна категорично думати про предмети та явища світу, що вивчається. Наші знання можуть бути нечіткими, неповними, суперечливими, але традиційні бази даних, як правило, відповідають розглянутим вище постулатам і тому цих властивостей знань не відображують.

## 15.3. Моделі зображення знань

Існує багато різних моделей, призначених для зображення знань. Класичними серед них можна вважати такі:

- формально-логічна модель;
- продукційна модель;
- семантичні мережі;
- фреймова модель.

Деякі автори включають у цей список об'єктну модель та модель сутностей і зв'язків.

Що стосується класифікації знань, то зазвичай їх поділяють на *декларативні* й *процедурні*. На ці категорії поділяються й моделі зображення знань. Підкласами декларативних моделей є *логічні* та *структурні* моделі.

Раніше в комп'ютерних системах використовувався процедурний спосіб зображення знань. Він передбачав зображення знань у вигляді алгоритмів, за якими укладаються програми. Такі програми керують і обробляють дані, що зберігаються у файлових системах або базах даних. Для зміни чи поповнення «розчинених» у програмі знань, програма має бути переписана.

Під декларативними знаннями розуміють знання типу «*A* це *B*», зберігання яких є характерним для баз даних. Це факти на кшталт «у годину пік на вулиці багато машин», «запалена плита гаряча», «скарлатина - інфекційне захворювання».

Розглянемо детальніше різновиди моделей зображення знань.

### 15.3.1. Формально-логічна модель

Логіка предикатів першого порядку стала теоретичним базисом науки про штучний інтелект на початку її становлення. Дослідники в галузі баз знань віддають

перевагу математичній логіці у зв'язку з тим, що вона має повну математичну формалізацію, а також тому, що вона дозволяє в межах однієї й тієї ж моделі подавати знання і способи мислення.

Згідно з логічною моделлю база знань розглядається як теорія, що містить багато фактів довільної складності, які є даними бази фактів та аксіомами бази знань. У цій теорії діють механізми виведення, що дають змогу одержувати нові знання з наявних. У класичній логіці використовується дедуктивний механізм виведення (він дає можливість на підставі фактів $A$ і $A \supset B$ зробити висновок про існування факту $B$). Можуть застосовуватися й інші механізми виведення (абдуктивний, традуктивний та індуктивний), але це вже не буде класичною логікою.

Наприклад, якщо задано факти $A$, $A \supset B$, $B \supset C$, то за допомогою дедуктивного виведення можна отримати такі нові факти: $B$, $C$, $A \vee B$, $A \vee C$, $B \vee C$ тощо.

#### Обмеження

Застосування логіки для моделювання знань пов'язане з певними обмеженнями:

- відсутністю механізмів додавання й видалення аксіом;
- відсутністю механізмів «відстежування» переходу від однієї теорії до іншої;
- відсутністю принципів керування;
- негнучкістю у тому розумінні, що мова моделювання знань є повністю декларативною.

Є альтернативні способи використання логіки, які знімають ці обмеження.

### 15.3.2. Продукційна модель

#### База фактів і база правил

У продукційній моделі, або моделі, заснованій на правилах, знання зображуються у вигляді сукупності фактів і правил.

Факти нічим не відрізняються від даних, вони можуть бути лише елементарними. Кажучи про елементарність, ми маємо на увазі відсутність суттєвої структурованості фактів. Максимально допустима структурованість — це можливість зображення у вигляді фактів описів сутностей, що мають атрибути, яким надаються елементарні значення. Факти утворюють *базу фактів*.

Правила описують процедурні знання і подають їх у вигляді пропозицій: «якщо істинна певна умова, то виконати певну дію».

Під *умовою* розуміють пропозицію-зразок, за якою виконується пошук у базі фактів. Умову іноді називають *посилкою правила*, а дії, що виконуються в разі успішного завершення пошуку за умовою, - *дією правила*. Дію також називають *висновком правила*. Дія може бути *проміжною* або *термінальною* (*цільовою*).

Проміжні дії — це окремі кроки в ланцюжку можливих (або необхідних) дій. Такою дією може бути, наприклад, ініціювання діалогу з людиною, внесення змін у базу фактів або породження додаткових умов. Термінальні дії завершують роботу системи.

Сукупність правил утворює *базу правил*. *База знань* у продукційній моделі – це сукупність бази фактів і бази правил.

Побудова бази знань на основі продукційної моделі є домінуючим підходом в експертних системах. Наприклад, у медичній експертній системі правила `if...then` можуть використовуватися для встановлення взаємозв'язків між симптомами і діагнозами. Під час виведення реальний симптом зіставляється з тими, які є в лівих частинах правил і в разі збігу права частина відповідного правила вважається можливим діагнозом. Якщо є інші правила, що містять у лівих частинах отриманий можливий діагноз, то він розглядається як проміжний симптом. У цьому випадку здійснюється подальше виведення, яке триває доти, доки не буде отримано результат, з якого нічого вже не можна вивести. Якщо більше немає правил, на основі яких можна зробити виведення з отриманого можливого діагнозу, то він розглядається як «остаточний». На будь-якому кроці такого виведення може виявитися кілька застосовних правил і тоді породжується дерево виведення, що визначає множину діагнозів.

### Машина виведення

Правила продукційної моделі не впорядковані. Кожне з них існує незалежно від інших правил. У зв'язку з цим потрібний спеціальний механізм, який керуватиме перебиранням правил. Такий механізм називається *машиною виведення*.

Машина виведення складається з двох компонентів:

- *Компонент виведення* реалізує власне дедуктивне виведення. Тобто, якщо в базі фактів є факт `A`, а в базі правил є правило `If A then B`, то робиться висновок про необхідність застосування дії `B`.
- *Компонент керування*, або *інтерпретатор правил* керує процесом перебирання фактів і застосування правил.

Інтерпретатор правил працює за описаним нижче алгоритмом.

1. **Зіставлення.** Здійснюється пошук множини правил, посилки яких зіставляються хоча б з одним фактом із бази фактів. Усі правила з цієї множини є застосовними до поточної бази фактів. Якщо правил у цій множині більше одного, то кажуть, що множина правил є *конфліктною* (у тому розумінні, що будь-яке з правил можна застосувати і невідомо, яке саме).
2. **Вибір.** Алгоритм роботи інтерпретатора є циклічним. На кожній ітерації циклу може бути застосовано лише одне правило. Якщо правил більше одного, інтерпретатор має вирішити конфлікт, тобто обрати з правил найвідповідніше. Вибір здійснюється на основі критерію, який може встановлюватися ззовні.
3. **Виконання.** Відібране правило запускається на виконання (спрацьовує). Суть спрацьовування полягає у виконанні дії, описаної у висновку правила. Такими діями можуть бути:
   - коректування критерію вибору правил;
   - запис, видалення або коректування фактів у базі фактів;
   - запис, видалення або коректування правил у базі правил;
   - виконання інших дій (ведення діалогу з людиною, перевірка цілісності тощо).

Схематично процес інтерпретації правил зображено на рис. 15.1.

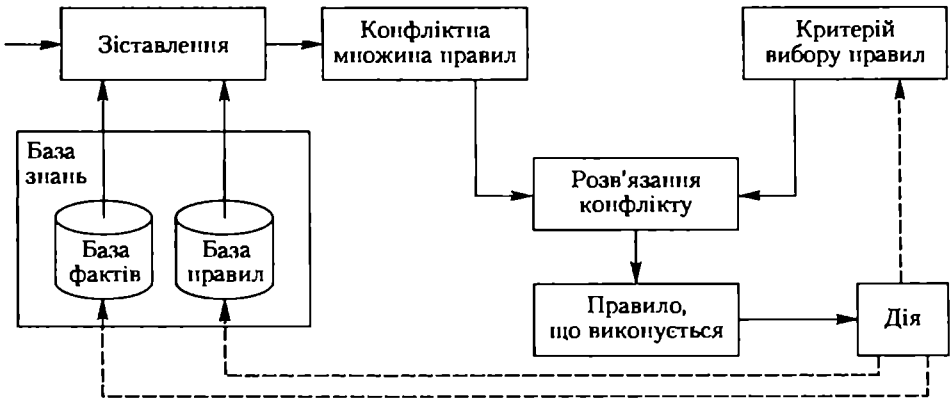



**Рис. 15.1.** Інтерпретація правил

У цій схемі робота машини виведення залежить від стану бази знань і критерію вибору правил. Існує також варіант організації машини виведення, за якого враховується передісторія її роботи, тобто поведінка механізму виведення на попередніх ітераціях.

### Стратегії керування виведенням

Машина виведення має вирішувати, як перебирати факти і правила бази знань, а також на підставі якого критерію слід вибирати правила з конфліктної множини. Відповідні рішення приймаються згідно з обраною стратегією керування виведенням. Зазвичай основні складові цієї стратегії вбудовані в машину виведення і їх не можна змінити.

Розробляючи стратегію керування виведенням, слід вирішити такі питання.

- Яку точку в просторі станів обрати як початкову? Від вибору цієї точки залежить спосіб виконання пошуку — в прямому чи зворотному напрямку.
- Якою має бути стратегія перебирання правил — углибину чи вшир?

#### Пряме і зворотне виведення

У системах з *прямим виведенням* за відомими фактами відшукується факт, який з них випливає (рис. 15.2, *а*). Якщо такий факт вдається знайти, то він записується в базу фактів. Пряме виведення називають також *виведенням, керованим даними*, або *виведенням, керованим посилками правил*

Під час *зворотного виведення* спочатку висувається гіпотеза, а потім механізм виведення ніби повертається назад, переходячи до фактів і намагаючись знайти в базі фактів ті, які підтверджують гіпотезу (рис. 15.2, *б*). Якщо гіпотеза не підтверджується фактами з бази фактів, одна з її можливих посилок вважається гіпотезою, що деталізує початкову і є стосовно неї *підціллю*. Далі відшукуються факти, які могли б підтвердити дану підціль, тобто процес рекурсивно повторюється. Виведення цього типу називається також *виведенням, керованим цілями*.

### Пошук углибину та вшир

Під час пошуку вглибину черговою підціллю стає та, що відповідає більш детальному рівню опису задачі. Виконуючи пошук ушир, машина виведення спочатку спробує знайти розв'язок серед можливих варіантів одного рівня і потім, за необхідності, перейде на наступний рівень деталізації.

На рис. 15.2 графічно зображено пошук углибину та вшир за умов прямого й зворотного виведення.

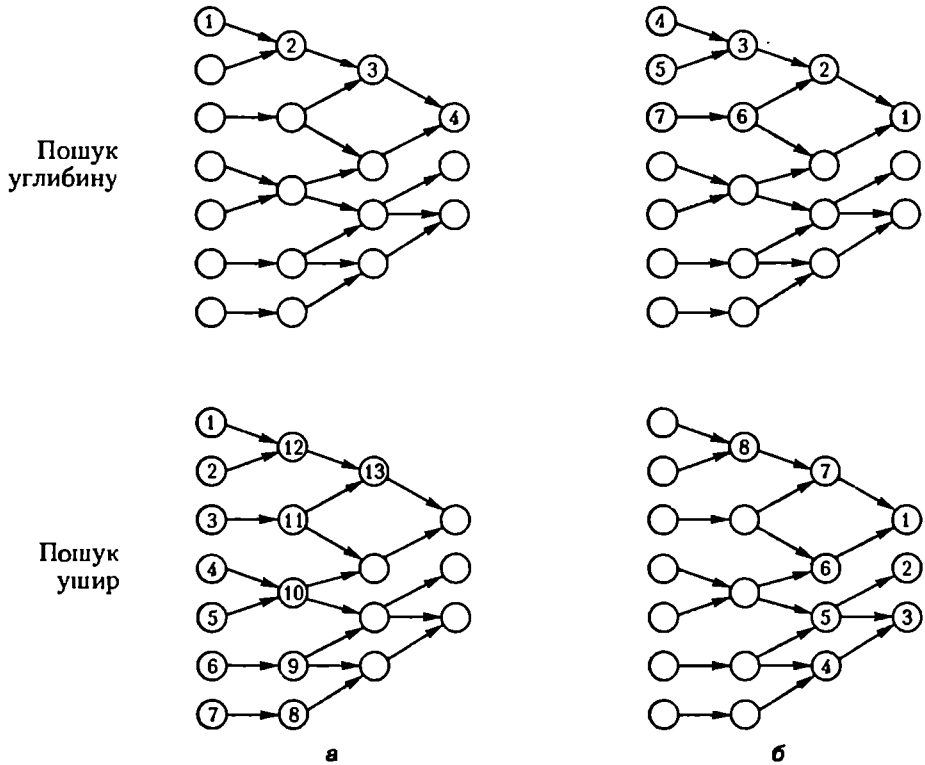



**Рис. 15.2.** Стратегії виведення: пряме виведення (*а*); зворотне виведення (*б*)

## Проблеми

Продукційна модель знімає обмеження, характерні для логіки, проте з нею пов'язані інші проблеми: нескінченні цикли, можлива суперечність знань і непрозорість поведінки машини виведення.

*Нескінченні цикли* виникають у тому випадку, коли машина виведення повертається до правил, які вже були переглянуті. Це можливо, наприклад, за наявності таких правил:

```
If A then B; If B then C; If C then A.
```

*Суперечливі знання* з'являються тоді, коли додавання нових правил призводить до суперечності тим фактам, які можна було отримати раніше.

*Непрозорість поведінки* обумовлена тим, що немає жодних принципів, які б встановлювали порядок перегляду правил і їхнього застосування в тому випадку, коли може бути застосовано кілька правил. Унаслідок цього досить важко обробляти продукційні бази знань великого обсягу, оскільки навіть за умов коректності всіх наявних правил хибний порядок їхнього виконання може призвести до помилок, що важко виявляються.

Частково зняти обмеження, характерні для формально-логічної та продукційної моделей, можна шляхом структуризації бази знань. Названі моделі допускають зображення в базі знань лише елементарних фактів. Структуризація фактів приводить до створення груп взаємопов'язаних фактів, тобто певних абстракцій. Структурні абстракції можуть мати свою семантику, щодо якої застосовуються правила виведення.

Семантичні мережі та фрейми, що розглядатимуться далі, найчастіше використовуються у моделях. які підтримують структурні абстракції.

## 15.3.3. Семантичні мережі

Знаннями можна називати описи зв'язків між абстрактними поняттями і сутностями, що є конкретними об'єктами реального світу. Поняття і зв'язки між ними можна описати мережею, що складається з вузлів і дуг. Вузли в такій мережі зображують сутності й поняття, а дуги відповідають зв'язкам між ними; усі вузли й дуги можуть бути позначені мітками, які вказують, що саме вони описують. Така форма зображення знань називається *семантичною мережею.*

Семантичні мережі як моделі даних були розроблені дослідниками, які працювали над проблемами штучного інтелекту. Базові структури можуть бути зображені у вигляді графу, множина вершин і дуг якого утворює мережу як, наприклад, у мережній моделі даних. Проте семантичні мережі призначені для зображення і систематизації знань загального характеру. Поштовхом до розвитку моделей цього класу стали насамперед проблеми алгоритмізації процесів розпізнавання природної мови. Використання семантичних мереж було спочатку пов'язане з моделюванням людської пам'яті, яка має асоціативну природу. Асоціації зображувалися за допомогою дуг, що з'єднують вершини мережі.

### Структура семантичної мережі

Будь-яка семантична мережа є графом. Вершини такого графа можуть зображувати предмети (значення екземплярів або сутностей), а дуги — твердження щодо предметів (зв'язки між предметами). Наприклад, одна вершина може зображувати сутність Іван, інша — Петро, а дуга між ними — твердження Іван – брат Петра.

Одиниця інформації в цій моделі може розглядатися як предмет (сутність) або твердження. Наприклад, колір «зелений» може відповідати предмету або твердженню. Зв'язок «бути братом» може розглядатися як твердження або предмет із певними властивостями (наприклад, «це дійсний факт» або «хтось вірить у це»). Можливість інтерпретації об'єкта як предмета або твердження має аналогії в інших моделях даних, де дані інтерпретуються як типи сутностей, їхні атрибути або зв'язки.

Виразові можливості семантичної мережі, що зображує лише предмети й твердження, достатньо обмежені. Виразова потужність моделі підвищується завдяки диференціації вершин і дуг мережі за категоріями.

Категорії вершин встановлюються відповідно до предметів, що зображуються ними. Наведемо один із найважливіших способів такої категоризації. Виділимо чотири категорії вершин:

- концепти (поняття);
- події;
- характеристики (властивості);
- значення

*Концепти* – категорія об'єктів, заради яких будується модель. Концепти використовуються для специфікації сутностей модельованої предметної області (Іван, Петро, місто Київ).

*Події* відповідають діям у предметній області, що зображується. Наприклад, дія Іван вчить Петра зображується вершиною події для дії вчить і двома вершинами-концептами Іван і Петро (рис. 15.3).

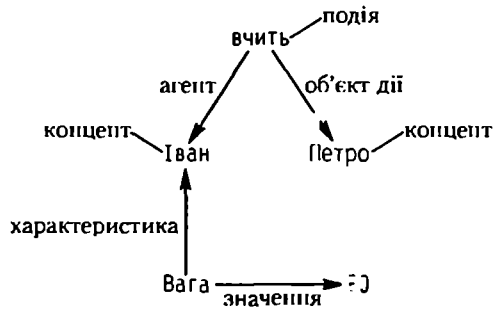



**Рис. 15.3.** Приклад семантичної мережі

Дуги, що сполучають вершини-події та вершини-концепти, відповідають ролям концептів у подіях (наприклад, у події вчить концепт Іван є *агентом*, а концепт Петро – *об'єктом дії*).

*Характеристики* – це вершини, що відповідають властивостям концепту. Наприклад, вага і зріст Івана – характеристики концепту Іван.

Нарешті, *значення* – це вершини, що відповідають значенням, яких можуть набувати характеристики. Так, характеристика Вага, що належить концепту Іван і має значення 80, визначає, що вага Івана становить 80 кг.

Додатковим засобом забезпечення виразової потужності семантичних мереж даних є розподіл вершин за типами. Зазвичай *вершини-концепти*, що зображують об'єкти, відрізняють від *вершин-класів*, що зображують типи. Наприклад, Іван – концепт, а ОСОБА – клас. Концепт може належати більш ніж одному класу. Отже, різні ролі об'єкта зображуються його належністю до різних класів.

Відмінність між класом і концептом близька до відмінності між типом і екземпляром в інших моделях даних, але у зв'язку з цим слід зробити два зауваження.

По-перше, в семантичних мережах зображуються як класи, так і концепти. В інших моделях даних типи специфікуються в схемі, а екземпляри зображуються в базі даних. По-друге, в семантичних мережах концепти зв'язуються з різними класами. У мережній та ієрархічній моделях даних це не допускається.

Відмінність між вершинами-концептами і вершинами-класами обумовлює виділення трьох різновидів дуг. Дуга, що сполучає два концепти, відповідає *твердженню*. Дуга між класом і концептом зображує *породження екземпляра класу*. Нарешті, дуга, що зв'язує два класи, характеризується як *бінарний зв'язок* (сам по собі він може розглядатися як клас). На рис. 15.4 наведено приклад семантичної мережі. Тут вершини Іван і Петро — це концепти, ШКОЛА, ДИТИНА — класи, дуга брат — твердження. Дуги, що зв'язують Івана і Петра з класом ДИТИНА, зображують зв'язок екземплярів з класом. Дуга відвідує зображує бінарний зв'язок між класами ДИТИНА і ШКОЛА.


**Рис. 15.4.** Різновиди дуг і вершин у семантичній мережі

Класи можуть бути сполучені ієрархічно за допомогою зв'язків «ціле/частина» і «узагальнення/різновид» (рис. 15.5). Наприклад, вік є «частиною» студента, а студент є «різновидом» особи. Ієрархія класів використовується для того, щоб вказати на успадкування властивостей одного класу іншим. Клас успадковує всі атрибути (властивості) класу, розташованого вище за ієрархією «узагальнення/різновид». За зв'язком «клас/екземпляр» концепт успадковує всі атрибути класу, до якого він належить.

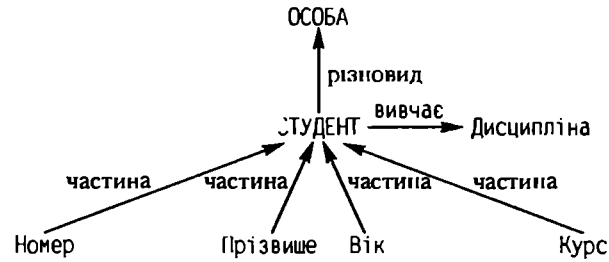



**Рис. 15.5.** Ієрархія класів

З метою структуризації і забезпечення успадкування властивостей бінарний зв'язок класів можна розглядати, у свою чергу, як клас, і на множині таких класів також можна встановлювати ієрархічні зв'язки «ціле/частина» і «узагальнення/різновид». Наприклад, визначивши на класі ОСОБА бінарні зв'язки Взаємини і Шлюб, можна встановити, що вони, у свою чергу, зв'язані відношенням «узагальнення/

різновид» (Шлюб є різновидом зв'язку Взаємини і навпаки, Взаємини є узагальненням зв'язку Шлюб). Якщо бінарні зв'язки розглядати як вершини графу семантичної мережі, то їх також можна з'єднувати дугами (рис. 15.6).

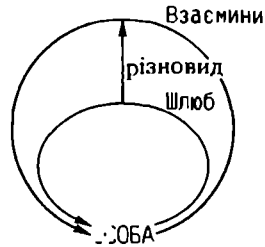



**Рис. 15.6.** Зв'язки між зв'язками

Клас, екземплярами якого є інші класи, називається *метакласом*. Оскільки метаклас є класом, то він може брати участь в ієрархічних зв'язках типу «ціле/частина» і «узагальнення/різновид». Коренем цієї ієрархії може бути вершина КЛАС, що зображує клас усіх класів.

Отже, семантичні мережні моделі даних достатньо складні. Вершини семантичної мережі можуть зображувати екземпляри, класи і метакласи, а дуги – твердження, породження екземплярів і бінарні зв'язки. Крім того, на бінарних зв'язках, класах і метакласах можуть бути означені ієрархічні зв'язки типу «ціле/частина» і «узагальнення/різновид».

### Обмеження цілісності в семантичних мережах

У семантичних мережних моделях даних специфікація обмежень цілісності є складовою частиною специфікації самої мережі. Обмеження – це деякі твердження, що стосуються концептів або класів. У більшості випадків семантичні мережі підтримують мову числення предикатів для опису дуг, що зв'язують концепти та/або класи, а також для опису довільних обмежень. За її допомогою можна, наприклад, вказати на те, що бінарний зв'язок є функціональним, або формалізувати обмеження «кожна дисципліна має бути пов'язана принаймні з одним студентом».

### Операції в семантичних мережах

Немає стандартного набору операцій над семантичними мережами, як і немає стандартної семантичної мережної моделі даних. У зв'язку з цим розглянемо операції, означені в семантичній мережній моделі PSN (Procedural Semantic Network). Вони бувають двох типів: операції над класами й екземплярами та операції над бінарними зв'язками.

Існують чотири основні операції над класами й екземплярами.

- Створення екземпляра класу та встановлення приналежності екземпляра одного класу іншому. Так, наприклад, новий екземпляр класу СТУДЕНТ може бути введений шляхом його створення або вказівкою, що певний екземпляр класу ОСОБА має бути зарахований також до класу СТУДЕНТ.
- Скасування належності екземпляра до певного класу або повне видалення екземпляра.

- Вибирання всіх екземплярів, що належать певному класу.
- Визначення приналежності екземпляра тому чи іншому класу.

Тепер перелічимо основні операції над бінарними зв'язками.

- Встановлення зв'язку між двома класами.
- Розривання зв'язку між двома класами.
- Вибирання всіх екземплярів, які через заданий бінарний зв'язок з'єднані з заданим екземпляром.
- Перевірка наявності зв'язків між двома зв'язками.

Якщо в семантичній мережі є ієрархія абстракцій, операції в ній успадковуються. Нехай, наприклад, клас ДИТИНА пов'язаний з класом ОСОБА зв'язком «узагальнення/різновид» і накладено таке обмеження: до класу ДИТИНА належать усі особи, молодші 18 років. У результаті під час створення екземпляра класу ДИТИНА буде автоматично створюватися екземпляр класу ОСОБА. Аналогічно, під час створення екземпляра класу ОСОБА, що відповідає вказаному обмеженню, буде також створюватися екземпляр класу ДИТИНА. У семантичній мережній моделі успадкування розглядається як невід'ємна частина операцій і відіграє значну роль у моделюванні правил, які людина застосовує автоматично.

Правила успадкування властиві також об'єктно-орієнтованій моделі даних і реалізованим на її основі об'єктно-орієнтованим СКБД.

Операції над класами і бінарними зв'язками є основою для маніпулювання даними у семантичній мережі. Проте ці операції надто примітивні й обмежені для того, аби бути засобом реалізації запитів

### Виведення в семантичних мережах

Виведення створюються за дугами, що мають спільні вузли. Вони базуються переважно на властивостях тих чи інших типів зв'язків між вершинами мережі. Якщо, наприклад, відношення, поіменоване є, належить типу «узагальнення/різновид» (яке має властивість транзитивності) і в мережі зафіксовано, що футболіст є спортсмен і спортсмен є людина, то з цього робиться висновок, що футболіст є людина. Ймовірно, кращі результати в галузі виведення в семантичних мережах можна буде отримати в разі їх комбінування з іншими моделями знань.

### Висновки

Створення семантичних мережних моделей — це спроба забезпечити інтегроване зображення даних, категорій даних, властивостей категорій і операцій над даними. У цьому полягає принципова відмінність семантичних мереж від інших моделей даних.

У більшості моделей дані, що зберігаються в базі, чітко відокремлюються від категорій даних (схеми) і допускаються лише операції над даними. Властивості категорій, їхня структура, правила успадкування зазвичай виходять за межі понять, якими оперують моделі даних. У більшості випадків модель даних орієнтована на підтримку структурованої сукупності фактів і семантичних властивостей, що виражаються структурно або за допомогою невеликого набору обмежень, а також операцій над фактами (даними).

До семантичної мережі, як правило, включаються елементи знань, що дають змогу полегшити інтерпретацію даних. Різноманітність структур семантичної мережної моделі даних сприяє реалізації різних способів зображення семантики. Тобто одна й та сама інформація може бути зображена різними способами.

Семантична мережа має певні недоліки. Один з них полягає в тому, що її механізм виведення може призводити до некоректних результатів у зв'язку з можливістю виходу за межі певного контексту.

Для прикладу розглянемо рис. 15.7. На питання «Чи є літак членом творчого колективу?» ми отримаємо позитивну відповідь. Щоб уникнути цього, необхідно встановити обмеження, що область дії виведення стосовно літака не може розповсюджуватися далі мультиагентної системи. Проте специфікація в семантичній мережі подібних обмежень є проблематичною. Можливим виходом з даної ситуації є інтеграція семантичних мереж з іншими моделями зображення знань, наприклад фреймовою моделлю.

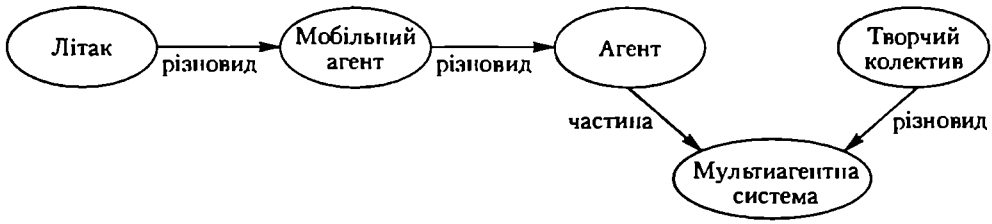



**Рис. 15.7.** Приклад семантичної мережі

Інший недолік семантичних мереж — імовірність комбінаторного вибуху в разі отримання негативної відповіді. Наприклад, щоб відповісти на питання «Чи є база даних частиною мультиагентної системи?» ми отримаємо відповідь «ні», але лише після того, як будуть перебрані всі можливі асоціації.

## 15.3.4. Фреймова модель

Фреймова модель даних є методом зображення знань, який базується на теорії фреймів, запропонованій М. Мінським у 1975 році. Ця теорія розроблялася, зокрема, і для вирішення проблеми виходу за межі контексту.

У фреймовій моделі одиницею зображення є *фрейм* (об'єкт). Він є формою зображення певної ситуації, яку можна (або доцільно) описувати сукупністю понять і сутностей.

У різних галузях науки використовується поняття абстрактного зображення. Наприклад, слово «кімната» породжує зображення кімнати: «житлове приміщення зі стінами, підлогою, стелею, вікнами і дверима». В цьому описі є «гнізда» — невизначені значення певних атрибутів, наприклад, кількість стін, вікон і дверей, висота стелі. У теорії фреймів таке зображення кімнати називається фреймом кімнати.

Розрізняють *фрейми-зразки* (прототипи, що є описами) і *фрейми-екземпляри* (конкретні значення прототипів).

У фреймі сутності предметної області зображуються у вигляді пар «атрибут-значення», або так званих *слотів*. Слоти, у свою чергу, можуть містити *фасети*, або обмеження на множину допустимих значень. Так, у фреймі AGENT з наведеного нижче опису слоти Planner, Knowledge-Store і Security-Level містять фасети. Зокрема для слота Security-Level є два фасети: Range і If-Changed. Range містить пов'язане з ним значення, а If-Changed — ім'я процедури, яка викликається під час зміни значення. Процедури типу If-Changed називаються *демонами*. Демон активізується автоматично, коли в його слоті здійснюється підстановка або порівняння значень.

```
Generic AGENT Frame
  Specialization-of: PROGRAM
  Generalization-of: (MOBILE-AGENT, INTERFACE-AGENT)
Planner:
  Range: (FORWARD-CHAINING, BACKWARD-CHAINING)
Knowledge-Store:
  Range: (Database, Flat-File)
  Default: Database
Security-Level:
  Range: (SECRET, UNCLASSIFIED, TOP-SECRET)
  If-Changed:(ERROR: No permission to change classification)
```

Потужність фреймової моделі обумовлена тим, що в ній використовується властивість успадкування фреймів і демонів. Успадкування фреймів підтримується завдяки наявності в будь-якому фреймі полів Specialization-of і Generalization-of. Так, фрейм AGENT є різновидом фрейму PROGRAM (не наведеного в прикладі), оскільки його ім'я вказане як значення поля Specialization-of. Оскільки AGENT є PROGRAM, то він неявно успадковує всі слоти фрейму PROGRAM. У свою чергу, слоти фреймів PROGRAM та AGENT успадковуються фреймами MOBILE-AGENT та INTERFACE-AGENT, оскільки імена цих фреймів указані в полі Generalization-of фрейму AGENT, а отже, згадані фрейми є різновидами AGENT.

Слот у складі фрейму-екземпляра може набувати значень у такі способи:

- наданням значення за замовчуванням, що вказується у фреймі-зразку;
- через успадкування властивостей від іншого фрейма;
- за формулою, вказаною в описі слота;
- через з'єднану зі слотом процедуру;
- під час діалогу з користувачем;
- з бази даних.

На сьогодні розроблені спеціальні мови для зображення знань у вигляді фреймів: FRL (Frame Representation Language), KRL (Knowledge Representation Language), інструментальні засоби підтримки цих мов, а також фрейм-орієнтовані експертні системи.

### Недоліки

Як і інші моделі, фреймова модель має певні недоліки. Перший з них полягає в тому, що фрейми не можуть так легко зображувати евристичні знання, як продукційна модель. Фрейми також не можна пристосувати до нових ситуацій або об'єктів. Окрім того, підходи, що дають змогу описувати структуру предметної

області (семантичні мережі та фрейми), вимагають, щоб механізм виведення залежав від семантики структурних зв'язків. Відтак створення нових зв'язків зі своєю семантикою вимагає заміни або настроювання машини виведення. Одним зі шляхів вирішення цієї проблеми є об'єднання логічного й структурного підходів, про що йтиметься нижче.

### 15.3.5. Об'єктне зображення знань

Об'єктне зображення знань базується на об'єктно-орієнтованій концепції, яка зараз широко використовується на основних етапах життєвого циклу розробки програмного забезпечення – в аналізі, проектуванні й реалізації. За основу для об'єктного відображення даних у базах даних можна взяти пропозиції робочої групи ODMG (див. розділ 14).

В об'єктному відображенні ефективно використовується процедурність демонів фрейма. Замість того, щоб незалежно посилатися на процедури з різних атрибутів, об'єкти інкапсулюють процедури разом з даними, які вони обробляють. Об'єктно-орієнтований підхід до зображення знань пропонує всі ті переваги з абстракції, інкапсуляції, поліморфізму і успадкування, що й мови програмування. За винятком успадкування, ці переваги притаманні лише об'єктно-орієнтованому підходу й роблять його ефективнішим за фреймову модель. Проте об'єктно-орієнтовані знання мають принаймні один недолік порівняно з фреймами: неможливість зручного пристосування до змін.

### 15.3.6. Гібридні моделі

Від області застосування залежить, який із засобів слід використовувати для зображення знань: логіку, правила, семантичні мережі, фрейми чи об'єкти. Можливо, жодна з відповідних моделей у чистому вигляді не спроможна повною мірою вирішити поставлені завдання. У такому разі можна використовувати гібридні моделі. Вони можуть бути простим об'єднанням двох чи більше моделей або бути настільки складними, що як просте об'єднання моделей їх не можна класифікувати.

Наприклад, метою створення продукційно-фреймової моделі може бути об'єднання потужності продукційних правил з перевагами структурного зображення фактів. Однією з особливостей такої моделі є те, що в ній виведення фактів може виконуватися демонами фреймів, а не машиною виведення. Крім того, таке сполучення дає фреймам можливість вдаватися до допомоги інших фреймів, коли вони не в змозі самостійно вирішити проблему.

За такого об'єднання завжди слід чітко усвідомлювати, що одна з моделей буде домінуючою, а інша – виконуватиме роль супутньої. Наприклад, для продукційно-фреймової моделі можливі такі реалізації:

- будується фреймова модель даних з процедурною складовою – демонами, поданими у вигляді продукційних правил;
- будується продукційна система, в якій є можливість зображувати факти у вигляді елементарних конструкцій та фреймів.

## 15.3.7. Розширена реляційна модель даних

Після появи реляційної моделі проводилися численні дослідження щодо можливостей її розширення. Реляційна модель розширюється для того, щоби вона могла більш адекватно відображувати семантику предметних областей, тобто була інтелектуальнішою. Семантично-орієнтовані моделі даних дають змогу обробляти запити й інші повідомлення в «інтелектуальний» спосіб. Окрім того, подібні моделі можуть бути ефективними посередниками між людиною і комп'ютером. У цьому підрозділі ми викладемо основні положення розширення реляційної моделі даних, що були запропоновані Е. Коддом. Відмінність цієї моделі даних від звичайної реляційної полягає у можливості використання *невизначених значень*.

Невизначені значення даних можуть тлумачитися по-різному. Спосіб маніпулювання такими значеннями залежить від способу їх інтерпретації. Е. Кодд запропонував розглядати два типи невизначених значень:

- значення, які невідомі на даний момент;
- незастосовні значення.

Проте даний підхід не здобув широкого визнання. Тому найчастіше в реляційній алгебрі розглядаються лише значення, які невідомі на даний момент. Невизначене значення цього типу позначається літерою ω.

З використанням невизначених значень пов'язані кілька питань, наприклад, якого значення набуває вираз $x = y$, коли $x$, або $y$, або обидва ці значення є невизначеними? Очевидно, що цей вираз не може бути ані істинним, ані хибним, а має бути невизначеним. Тому для порівняння невизначених даних необхідна тризначна логіка з такими значеннями: $t$ – істина, $f$ – хибність, ω – невизначене значення (невизначене логічне значення позначається тим самим символом, що й невизначене значення даних, оскільки, по-перше, даними в базі можуть бути й логічні значення, і, по-друге, природно зображувати в базі даних усі невизначені значення однаково)

Тризначна логіка висловлювань ґрунтується на таблицях істинності логічних операцій:

| AND | $f$ | ω | $t$ |
|---|---|---|---|
| $f$ | $f$ | $f$ | f |
| ω | $f$ | ω | ω |
| $t$ | $f$ | ω | $t$ |

| OR | $f$ | ω | $t$ |
|---|---|---|---|
| $f$ | $f$ | ω | $t$ |
| ω | ω | ω | $t$ |
| $t$ | $t$ | t | $t$ |

| | NOT |
|---|---|
| $f$ | $t$ |
| ω | ω |
| $t$ | $f$ |

Значення виразів з кванторами існування й загальності обчислюються шляхом ітеративного застосування операцій OR чи AND.

Значення виразу $x\,\theta\,y$, де $\theta$ – один із предикатів $=, \neq, <, >, \geq, \leq$, становить ω, якщо або $x$, або $y$, або вони обидва дорівнюють ω.

Що стосується предикатів належності елементу множині ($\in$) і включення множини в множину ($\subset$), то вважається, що вирази $\omega \in S$ і $\{\omega\} \subset S$ мають значення ω, якщо $S$ не порожня множина (навіть якщо $S$ містить тільки ω). Якщо ж $S$ – порожня множина, то ці предикати рівні $f$.

Будь-яка множина може містити не більше одного значення ω, будь-яке відношення може містити не більше одного кортежу з усіма значеннями ω. Ці правила *поширюються на всі теоретико-множинні операції реляційної алгебри* (об'єднання, перетин та різницю).

Розглянемо, як невизначені значення обробляються іншими операціями реляційної алгебри.

Операції декартового добутку і проекції залишаються без змін

Операція θ-з'єднання тепер поділяється на два різновиди.

- *TRUE θ-з'єднання* (θ-з'єднання за істинністю) – під час з'єднання вибираються лише ті рядки, на яких умова з'єднання набуває значення *t*. Така інтерпретація використовується в звичайній реляційній алгебрі. Позначення операції також не відрізняється від її позначення в звичайній реляційній алгебрі.
- *MAYBE θ-з'єднання* (θ-з'єднання за невизначеністю) – під час з'єднання вибираються лише ті рядки, на яких умова з'єднання набуває значення ω. Для позначення операції після θ-символу вказується символ ω («=ω», «≠ω», «*ω»).

Нижче наведені приклади використання обох різновидів операції θ-з'єднання.

*R*

| *A* | *B* |
|---|---|
| *a* | ω |
| ω | 2 |
| ω | 1 |

*S*

| *C* |
|---|
| ω |
| 2 |

*R* [*B* = *C*] *S*

| *A* | *B* | *C* |
|---|---|---|
| ω | 2 | 2 |

*R* [*B* = ω *C*] *S*

| *A* | *B* | *C* |
|---|---|---|
| *a* | ω | ω |
| *a* | ω | 2 |
| ω | 2 | ω |
| ω | 1 | ω |

*R* [*B* < *C*] *S*

| *A* | *B* | *C* |
|---|---|---|
| ω | 1 | 2 |

*R* [*B* < ω *C*] *S*

| *A* | *B* | *C* |
|---|---|---|
| *a* | ω | ω |
| *a* | ω | 2 |
| ω | 2 | ω |
| ω | 1 | ω |

Операція *селекції* також поділяється на два різновиди.

- *TRUE-селекція* (селекція за істинністю).
- *MAYBE-селекція* (селекція за невизначеністю).

У результаті TRUE-селекції вибираються кортежі, для яких умова відбору є істинною, а в результаті MAYBE-селекції – кортежі, для яких умова відбору є невизначеною. Позначаються ці операції символами «θ» і «θω».

Те ж можна сказати про операцію ділення. *TRUE-ділення* (ділення за істинністю) має стандартну інтерпретацію і позначення, а під час виконання *MAYBE- ділення* (ділення за невизначеністю) здійснюється перевірка включення множини у множину з урахуванням значень ω. MAYBE-ділення позначається символами «÷ω».

Три різновиди теоретико-множинних операцій, що описані далі, називаються *зовнішніми*. Вони дають змогу застосувати операції об'єднання, перетину і різниці до двох відношень, які не сумісні за об'єднанням. При цьому результат операції є відношенням.

Нехай задано відношення $R(A, B)$ і $S(B, C)$, де $A$, $B$, $C$ – множини атрибутів. Нехай $R_1(A, B, C) = R \times (C : \omega)$, $C_1(A, B, C) = (A : \omega) \times S$, де запис вигляду $(C : \omega)$ позначає відношення-константу, що складається з атрибута $C$ з єдиним значенням ω. Тоді *зовнішні об'єднання*, *перетин* і *різниця* відношень $R$ і $S$ означуються так:

$$R \textcircled{\cup} S = R_1 \cup S_1, \quad R \textcircled{\cap} S = R_1 \cap S_1, \quad R \textcircled{-} S = R_1 - S_1.$$

Розглянемо приклади використання цих операцій.

*R*

| *A* | *B* | *C* |
|---|---|---|
| $a_1$ | $b_1$ | $c_2$ |
| $a_1$ | $b_2$ | $c_1$ |
| $a_2$ | $b_1$ | $c_2$ |
| ω | $b_1$ | ω |

*S*

| *B* | *D* |
|---|---|
| $b_2$ | $d_2$ |
| $b_3$ | $d_1$ |
| $b_1$ | ω |

*R*⊙*S*

| *A* | *B* | *C* | *D* |
|---|---|---|---|
| $a_1$ | $b_1$ | $c_2$ | ω |
| $a_1$ | $b_2$ | $c_1$ | ω |
| $a_2$ | $b_1$ | $c_2$ | ω |
| ω | $b_1$ | ω | ω |
| ω | $b_2$ | ω | $d_2$ |
| ω | $b_3$ | ω | $d_1$ |

*R*⊛*S*

| *A* | *B* | *C* | *D* |
|---|---|---|---|
| ω | $b_1$ | ω | ω |

*R*⊖*S*

| *A* | *B* | *C* | *D* |
|---|---|---|---|
| $a_1$ | $b_1$ | $c_2$ | ω |
| $a_1$ | $b_2$ | $c_1$ | ω |
| $a_2$ | $b_1$ | $c_2$ | ω |

У класичній реляційній моделі операції θ-з'єднання та еквіз'єднання призводять до втрат інформації, коли проекції за атрибутами з'єднання відношень, що беруть участь в операції, не рівні. Для зберігання інформації незалежно від рівності цих проекцій, вказані операції модифікуються в описаний далі спосіб.

Нехай задано відношення $R(A, B_1)$ і $S(B_2, C)$, де атрибути $B_1$ і $B_2$ означені на одному домені. Нехай

$T = R\,[B_1\ \theta\ B_2]\,S$,

$R_1 = R - T\,[A, B_1]$ – рядки з $R$, що втрачаються в $T$,

$S_1 = S - T\,[B_2, C]$ – рядки з $S$, що втрачаються в $T$

Тоді *зовнішнє θ-з'єднання* означується в такий спосіб:

$$\mathrm{R}\,[B_1 ⊛ B_2\,]S = T \cup (R_1 \times (B_2, C : (ω,ω))) \cup ((A, B_1 : (ω,ω)) \times S_1).$$

Тут запис вигляду $(B_2, C : (ω,ω))$ позначає відношення-константу з атрибутами $B_2$ і $C$ та єдиним кортежем $(ω,ω)$.

Якщо покласти $T = R\,[B_1 = B_2]\,S$, а відношення $R_1$ і $S_1$ означити так, як це зроблено вище, то *зовнішнє природне з'єднання* означуватиметься так:

$$R\,[B_1 ⊙ B_2]\,S = \pi_{AB_1C}(T) \cup (R_1 \times (C : ω)) \cup ((A : ω) \times S_1).$$

Наведемо приклади застосування цих операцій.

*R*

| *R#* | *RC* |
|---|---|
| $r_1$ | $c_4$ |
| $r_2$ | $c_2$ |
| $r_4$ | $c_1$ |
| $r_6$ | $c_1$ |
| $r_7$ | $c_3$ |

*S*

| *SC* | *S#* |
|---|---|
| $c_1$ | $s_1$ |
| $c_2$ | $s_2$ |
| $c_2$ | $s_3$ |
| $c_5$ | $s_4$ |

*R* [*RC* ⊖ *SC*] *S*

| *R#* | *RC* | *SC* | *S#* |
|---|---|---|---|
| $r_1$ | $c_4$ | ω | ω |
| $r_2$ | $c_2$ | $c_2$ | $s_2$ |
| $r_2$ | $c_2$ | $c_2$ | $s_3$ |
| $r_4$ | $c_1$ | $c_1$ | $s_1$ |
| $r_6$ | $c_1$ | $c_1$ | $s_1$ |
| $r_7$ | $c_3$ | ω | ω |
| ω | ω | $c_5$ | $s_4$ |

*R* [*RC* ⊙ *SC*] *S*

| *R#* | *RC* | *S#* |
|---|---|---|
| $r_1$ | $c_4$ | ω |
| $r_2$ | $c_2$ | $s_2$ |
| $r_2$ | $c_2$ | $s_3$ |
| $r_4$ | $c_1$ | $s_1$ |
| $r_6$ | $c_1$ | $s_1$ |
| $r_7$ | $c_3$ | ω |
| ω | $c_5$ | $s_4$ |

## 15.4. Розширення семантики даних. Нечіткі дані

Дані, як значення властивостей об'єктів, можуть мати якісні характеристики (наприклад, дуже високий, високий, низький, дуже низький тощо). Ці характеристики не є чітко визначеними, не можуть бути однозначно інтерпретовані, але містять важливу інформацію. Окрім того, деякі людські знання не можна інтерпретувати як абсолютно істинні чи абсолютно хибні, а можна лише говорити про їхню істинність з певним ступенем упевненості у правильності припущення.

В обох згаданих вище випадках ми маємо справу з нечіткістю зображення даних. Як зобразити подібні знання формально, не руйнуючи властивості нечіткості? Для вирішення цієї проблеми на початку 70-х років Л. Заді запропонував формальний алгебраїчний та логічний апарат, що базується на понятті нечіткої множини. Теорія нечітких множин — це етап на шляху до зближення точності класичної математики і всепроникної неточності реального світу. Колись поява класичної логіки була кроком уперед у боротьбі з нечіткістю й розпливчастістю зображення людських знань. Тепер виникла необхідність створення теорії, яка б давала змогу формально описувати нестрогі, нечіткі поняття і дозволяла зрозуміти процес міркувань, під час якого такі поняття використовуються.

### Основні поняття теорії нечітких множин

*Нечітка множина* є сукупністю базової множини, наприклад $A$, і *функції належності* елементів базовій множині $\mu_A(x)$. Множина значень функції належності називається *множиною належностей*, вона є підмножиною інтервалу [0; 1].

Отже, нечітка множина — це сукупність пар вигляду: $\{x \mid \mu_A(x)\}$. Будемо позначати нечітку множину через $\tilde{A}$. Функція належності виражає суб'єктивний ступінь упевненості в тому, що певне значення базової множини належить нечіткій множині. Інакше кажучи, $\mu_A(x)$ — це суб'єктивно оцінена ймовірність того, що $x \in A$, яка ще називається *ступенем належності* $x$ нечіткій множині $\tilde{A}$. *Носієм* нечіткої множини $\tilde{A}$ називається множина таких $x$, що $\mu_{\tilde{A}}\{x\} > 0$.

Наприклад, нехай $U = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$. Тоді нечітку множину, що відповідає поняттю «трохи» можна визначити в такий спосіб: *трохи* = {(1|0), (2|0), (3|0,5), (4|0,8), (5|1), (6|1), (7|0,8), (8|0,5), (9|0), (10|0)}.

Під порожньою нечіткою множиною розуміють таку множину $\varnothing$, функція належності якої тотожньо дорівнює нулю.

Для нечітких множин означуються традиційні теоретико-множинні операції — включення, доповнення, об'єднання, перетин та інші. У табл. 15.1 перелічені деякі з них.

**Таблиця 15.1.** Операції над нечіткими множинами

| Позначення | Інтерпретація | Означення |
|---|---|---|
| $\tilde{A} \subseteq \tilde{B}$ | $\tilde{A}$ міститься в $\tilde{B}$ | $\forall x(\mu_{\tilde{A}}(x) \le \mu_{\tilde{B}}(x))$ |
| $\overline{\tilde{A}}$ | Доповнення до $\tilde{A}$ | $\forall x(\mu_{\overline{\tilde{A}}}(x) = 1 - \mu_{\tilde{A}}(x))$ |
| $\tilde{A} \cup \tilde{B}$ | Об'єднання $\tilde{A}$ і $\tilde{B}$ | $\forall x(\mu_{\tilde{A} \cup \tilde{B}}(x) = \max[\mu_{\tilde{A}}(x), \mu_{\tilde{B}}(x)])$ |
| $\tilde{A} \cap \tilde{B}$ | Перетин $\tilde{A}$ і $\tilde{B}$ | $\forall x(\mu_{\tilde{A} \cup \tilde{B}}(x) = \max[\mu_{\tilde{A}}(x), \mu_{\tilde{B}}(x)])$ |

| Позначення | Інтерпретація | Означення |
|---|---|---|
| $\tilde{A} \times \tilde{B}$ | Алгебраїчний добуток $\tilde{A}$ і $\tilde{B}$ | $\forall x(\mu_{\tilde{A}\times\tilde{B}}(x) = \mu_{\tilde{A}}(x) \cdot \mu_{\tilde{B}}(x))$ |
| $\tilde{A} + \tilde{B}$ | Алгебраїчна сума $\tilde{A}$ і $\tilde{B}$ | $\forall x(\mu_{\tilde{A}+\tilde{B}}(x) = \mu_{\tilde{A}}(x) + \mu_{\tilde{B}}(x) - \mu_{\tilde{A}}(x) \cdot \mu_{\tilde{B}}(x))$ |
| $\tilde{A}^{\alpha}$ | Піднесення до степеня $\alpha$ для будь-якого дійсного $\alpha$ | $\forall x(\mu_{\tilde{A}^{\alpha}}(x) = \mu_{\tilde{A}}^{\alpha}(x))$ |

Звичайна (чітка) множина – це частковий випадок нечіткої множини, для якої функція належності має такий вигляд:

$$\mu_A(x) = \begin{cases} 0, & x \notin A, \\ 1 & x \in A. \end{cases}$$

Для функції належності вказаного вигляду означені вище операції збігаються з відповідними операціями класичної теорії множин. Тому теорія нечітких множин є узагальненням традиційної теорії множин.

Операції об'єднання, перетину й доповнення для нечітких множин мають такі властивості.

- $\left.\begin{array}{l} \tilde{A} \cup \tilde{B} = \tilde{B} \cup \tilde{A} \\ \tilde{A} \cap \tilde{B} = \tilde{B} \cap \tilde{A} \end{array}\right\}$ – комутативність;
- $\left.\begin{array}{l} \tilde{A} \cup (\tilde{B} \cup \tilde{C}) = (\tilde{A} \cup \tilde{B}) \cup \tilde{C} \\ \tilde{A} \cup (\tilde{B} \cup \tilde{C}) = (\tilde{A} \cup \tilde{B}) \cup \tilde{C} \end{array}\right\}$ – асоціативність;
- $\left.\begin{array}{l} \tilde{A} \cap (\tilde{B} \cup \tilde{C}) = (\tilde{A} \cap \tilde{B}) \cup (\tilde{A} \cap \tilde{C}) \\ \tilde{A} \cup (\tilde{B} \cap \tilde{C}) = (\tilde{A} \cup \tilde{B}) \cap (\tilde{A} \cup \tilde{C}) \end{array}\right\}$ – дистрибутивність;
- $\left.\begin{array}{l} \tilde{A} \cup \tilde{A} = \tilde{A} \\ \tilde{A} \cap \tilde{A} = \tilde{A} \end{array}\right\}$ – ідемпотентність;
- $\left.\begin{array}{l} \tilde{A} \cup \varnothing = \tilde{A} \\ \tilde{A} \cap \varnothing = \varnothing \end{array}\right\}$ – об'єднання і перетин з порожньою множиною;
- $\left.\begin{array}{l} \overline{\tilde{A} \cup \tilde{B}} = \overline{\tilde{A}} \cap \overline{\tilde{B}} \\ \overline{\tilde{A} \cap \tilde{B}} = \overline{\tilde{A}} \cup \overline{\tilde{B}} \end{array}\right\}$ - правила де Моргана для нечітких множин

### Нечіткі відношення

Поняття відношення, що означується через поняття множини, відіграє фундаментальну роль у теорії реляційних баз даних. Його можна узагальнити, означивши через поняття нечіткої множини.

Припустимо, що $P$ – декартів добуток $n$ множин. Нечітка множина, базовою множиною якої є підмножина $P$, називається *нечітким n-арним відношенням*. Нечіткі відношення зображуються у вигляді множини пар, кожна з яких складається з упорядкованого набору з $n$ елементів і числа з інтервалу [0; 1] – *ступеня належності* цього набору даному нечіткому відношенню. Як і звичайні відношення,

нечіткі відношення позначаються літерами латинського алфавіту, але з символом «~» зверху.

Наприклад, нехай задані базові множини $U_1 = U_2 = \{1, 2, 3, 4\}$. Означимо нечітке відношення «приблизно рівні»: $\tilde{R} = \{(1,1)|1; (1,2)|0,5; (1,3)|0; (1,4)|0; (2,1)|0,5; (2,2)|1; (2,3)|0.5; (2.4)|0: (3,1)|0; (3,2)|0,5; (3,3)|1; (3,4)|0,5; (4,1)|0; (4,2)|0; (4,3)|0,5; (4,4)|1\}$.

Операції додавання, доповнення, об'єднання, перетину, алгебраїчного добутку і суми над нечіткими відношеннями збігаються з відповідними операціями над нечіткими множинами. Крім того, вводиться операція max-min-композиції. Якщо $\tilde{R}_1 \subseteq X \times Y$ і $\tilde{R}_2 \subseteq Y \times Z$, то *max-min-композицією* нечітких відношень $\tilde{R}_1$ і $\tilde{R}_2$ називається нечітке відношення $\tilde{R}_2 \circ \tilde{R}_1$, функція належності якого визначається за допомогою виразу

$$\mu_{\tilde{R}_2 \circ \tilde{R}_1}(x,z) = \max_y[\min(\mu_{\tilde{R}_1}(x,y), \mu_{\tilde{R}_2}(y,z))], \text{ де } x \in X, y \in Y, z \in Z.$$

До основних властивостей, які можуть мати нечіткі бінарні відношення, належать симетричність, рефлективність, транзитивність. Дамо формальні означення цих властивостей, розглянувши відношення $\tilde{R}$, означене на декартовому добутку $U \times U$.

Нечітке бінарне відношення $\tilde{R}$ називається *симетричним*, якщо

$$\forall(x,y) \in U \times U \quad (\mu_{\tilde{R}}(x,y) = \mu \Rightarrow \mu_{\tilde{R}}(y,x) = \mu).$$

Нечітке бінарне відношення $\tilde{R}$ називається *рефлексивним*, якщо

$$\forall(x,x) \in U \times U \quad (\mu_{\tilde{R}}(x,x) = 1).$$

Нехай $x, y, z \in U$, тоді нечітке бінарне відношення $\tilde{R}$ називається *транзитивним*, якщо

$$\forall(x,y),(y,z),(x,z) \in U \times U \quad (\mu_{\tilde{R}}(x,z) \geq \max_y[\min(\mu_{\tilde{R}}(x,y), \mu_{\tilde{R}}(y,z))]).$$

Прикладом симетричного й рефлексивного нечіткого відношення є відношення «*y* близьке до *x*». Нечіткі відношення «*y* набагато більше *x*», «*x* – далекий родич *y*» є транзитивними, а відношення «*x* схоже на *y*» не є таким, оскільки з того, що *x* схоже на *y* і *y* схоже на *z* може й не випливати, що *x* схоже на *z*.

Нечіткі відношення передпорядку й порядку означуються так само, як і у випадку звичайних бінарних відношень. Наприклад, нечітким відношенням передпорядку називається нечітке бінарне відношення, що має властивості транзитивності й рефлексивності. Всі означення, що пов'язані з відношеннями порядку на звичайних множинах, можна безпосередньо переносити на нечіткі відношення порядку.

Серед нечітких відношень виділяють також відношення подібності та відмінності. Відношення подібності – це узагальнення відношення толерантності для звичайних відношень. Для відношення відмінності аналогів у теорії звичайних відношень немає.

Відношенням *подібності* називається нечітке бінарне відношення $\tilde{R}$, що має такі властивості:

- рефлексивність: $\forall(x,x) \in U \times U \quad (\mu_{\tilde{R}}(x,x) = 1)$;
- симетричність: $\forall(x,x) \in U \times U \quad (\mu_{\tilde{R}}(x,y) = \mu_{\tilde{R}}(y,x))$.

Відношенням *відмінності* називається нечітке бінарне відношення $\tilde{R}$, що має такі властивості:

- антирефлексивність: $\forall(x,x) \in U \times U \quad (\mu_{\tilde{R}}(x,x) = 0)$;
- симетричність: $\forall(x,y) \in U \times U \quad (\mu_{\tilde{R}}(x,y) = \mu_{\tilde{R}}(y,x))$.

### Лінгвістичні змінні

Л. Заді ввів одне з головних понять у нечіткій логіці – поняття лінгвістичної змінної. Воно дає можливість маніпулювати нечіткими поняттями й категоріями, якими ми так часто послуговуємося в реальному житті, здійснювати виведення з посилань, що використовують нечіткі поняття тощо. На понятті лінгвістичної змінної базується теорія наближених міркувань.

*Лінгвістична змінна* – це змінна, значення якої визначається набором вербальних (тобто словесних) характеристик певної властивості.

Значеннями лінгвістичних змінних можуть бути не лише числа, але й фрази та вислови природної чи формальної мови. Розглянемо це поняття докладніше на прикладі лінгвістичної змінної Вік. Значеннями змінної Вік є поняття, що позначають вік людини, наприклад: молодий, немолодий, дуже молодий, старий, немолодий і нестарий, дуже-дуже старий тощо. Всі ці значення є нечіткими поняттями, які не можна зобразити конкретним числом. Більше того, оцінювання віку людини подібними поняттями є суб'єктивним. Вказані вище значення будемо називати *лінгвістичними значеннями* змінної Вік. Множина лінгвістичних значень певної змінної називається її *терм-множиною*. Будемо позначати її через $T(\chi)$, де аргумент $\chi$ – це ім'я лінгвістичної змінної.

Кожне з лінгвістичних значень змінної Вік можна зобразити у вигляді нечіткої підмножини універсальної множини $U = \{1, 2, 3, ..., 110\}$ (елементами $U$ є можливі значення віку людини). На рис. 15.8 показано один із способів надання лінгвістичних значень змінній Вік.

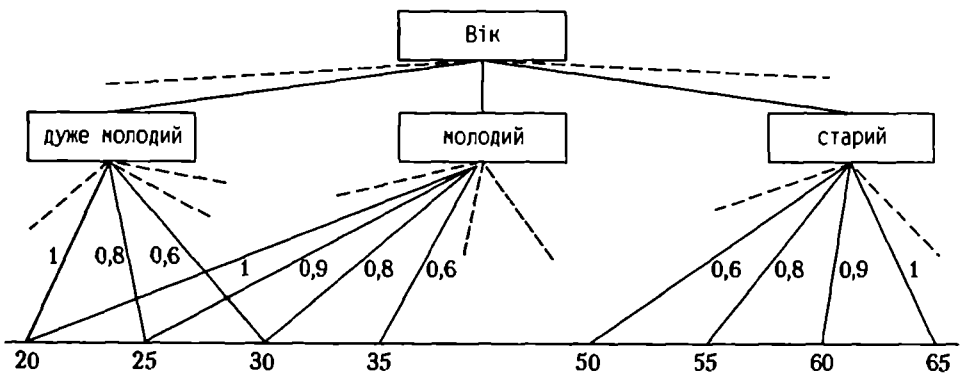



**Рис. 15.8.** Зображення лінгвістичної змінної

## 15.5. Механізми виведення даних

У базах знань, окрім дедуктивного механізму виведення, про який ми вже згадували в цьому розділі, можуть підтримуватися й інші. Далі ми коротко опишемо ще два механізми виведення: індуктивний та за аналогією.

### 15.5.1. Індуктивне виведення

*Індуктивне виведення* – це виведення від часткового до загального, виведення з наявних даних (прикладів) загального правила, що їх пояснює.

Розглянемо простий приклад. Нехай $f(x)$ – певний невідомий многочлен. Подивимося, як виводиться $f(x)$, якщо послідовно задаються (як приклади) пари значень $(0, f(0))$, $(1, f(1))$, .... Припустимо, що цими парами є (0, 1), (1, 1), (2, 3), (3, 7), (4, 13), (5, 21). Спочатку задається пара (0, 1), і можна покласти $f(x) = 1$. Потім задається пара (1, 1), яка відповідає запропонованій функції $f(x) = 1$. Отже, у цей момент немає необхідності змінювати виведення. Нарешті, нехай задається пара (2, 3). Тепер наше виведення не відповідає вихідним даним, тому відмовимося від нього й після кількох спроб і помилок виведемо нову функцію $f(x) = x^2 - x + 1$, що відповідає всім заданим фактам (0, 1), (1, 1), (2, 3). Далі, ми переконуємося, що й наступні наявні в нас факти (3, 7), (4, 13), (5, 21) підтверджують правильність виведення, тому немає необхідності його змінювати.

Як видно з цього прикладу, під час виведення в кожен момент часу враховуються всі дані, отримані до цього моменту. Дані, отримані пізніше, можуть не відповідати цьому виведенню. У такому разі виведення доводиться змінювати Отже, у загальному випадку індуктивне виведення – це необмежено довгий процес.

Для здійснення індуктивного виведення необхідно визначити:

- множину правил виведення;
- метод зображення правил;
- спосіб зображення прикладів;
- метод виведення:
- критерій правильності виведення.

Існують такі варіанти індуктивного виведення:

- на повних прикладах;
- на позитивних (правильних) прикладах (тобто тих прикладах, шо відповідають шуканому узагальнюючому правилу);
- на негативних прикладах (тобто тих, які спростовують узагальнююче правило).

З повних даних, що містять позитивні й негативні приклади, завжди можна робити індуктивні висновки. Проте, користуючись лише позитивними даними, у деяких випадках індуктивні висновки робити не можна. Тоді слід розглядати негативні дані.

Серед типових областей застосування індуктивного виведення відзначимо напівавтоматичне зображення знань в експертних системах, автоматичний синтез програм за прикладами в галузі інтелектуального програмування, напівавтоматичне налагодження програм.

### 15.5.2. Виведення за аналогією

Принцип аналогії полягає у виведенні висновку, подібного посилці. Пояснимо сформульований принцип на прикладі (рис. 15.9). Нехай задано два об'єкти, $S_1$ і $S_2$. Щодо об'єкта $S_1$ відомі факти $\alpha_1, \alpha_2, ..., \alpha_n$, а щодо об'єкта $S_2$ – факти $\beta_1, \beta_2, ..., \beta_n$. Нехай між фактами об'єктів $S_1$ і $S_2$ задано відношення подібності $\varphi$, тобто $\alpha_i \varphi \beta_i$ для $1 \le i \le n$. Нехай існує виведення з фактів $\alpha_1, \alpha_2, ..., \alpha_n$ факту $\alpha$. Тоді *аналогією* називається виведення з фактів $\beta_1, \beta_2, .... \beta_n$ такого факту $\beta$, що $\alpha\varphi\beta$.

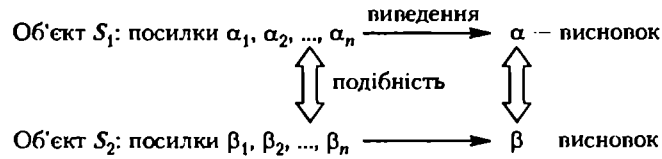



**Рис. 15.9.** Принцип аналогії

Щоб отримати конкретні механізми виведення за аналогією, слід уточнити такі аспекти розглянутої загальної схеми:

- як здійснюється виведення факту $\alpha$ з фактів $\alpha_1, \alpha_2, ..., \alpha_n$;
- чим є відношення подібності $\varphi$;
- як за заданим $\varphi$ здійснюється виведення такого $\beta$, що $\alpha \varphi \beta$.

Аналогія – більш природний механізм виведення для людини, ніж індукція. Класичними прикладами виведення за аналогією є прогнози і методи розв'язання задач. Аналогія застосовується в дослідженнях з машинного перекладу, використовується під час автоматизації доведення теорем та розроблення програм. Згідно з принципом аналогії нові факти прогнозуються (чи навіть виводяться) шляхом перетворення уже відомих знань.

## Контрольні запитання та завдання

1. Що перетворює дані на знання?
2. На яких постулатах базуються традиційні бази даних та інформаційні системи?
3. Опишіть формально-логічну модель зображення знань.
4. У чому полягає суть продукційної моделі зображення знань?
5. Як зображуються знання в семантичній мережі?
6. Що таке фреймова модель зображення знань?
7. Які додаткові переваги надає розширена реляційна модель даних?
8. Що таке нечіткі дані?
9. У чому полягає суть механізму індуктивного виведення?
10. Що називається виведенням за аналогією?

# Література


1. Андон Ф. И., Яшунин А. Е., Резниченко В. А. Логические модели интеллектуальных информационных систем. – К.: Наук. думка, 1999. – 396 с.
2. Атре Ш. Структурный подход к организации баз данных. – М.: Финансы и статистика, 1983. – 320 с.
3. Берзтисс А. Т. Структуры данных. – М.: Статистика, 1974. – 408 с.
4. Бойко В. В., Савинков В. М. Проектирование баз данных информационных систем. – М.: Финансы и статистика, 1989. – 351 с.
5. Брудно В. А., Скворцов Д. П., Финн В. К., Цаленко М. Ш. Базы данных с неполной информацией // Семиотика и информатика. – 1985. – Вып. 25. – С. 5–45
6. Гаврилова Т. А., Хорошевский В. Ф. Базы знаний интеллектуальных систем. – СПб.: Питер, 2001. – 384 с.
7. Глушков В. М., Цейтлин Г. Е., Ющенко Е. Л. Алгебра, языки, программирование. – К.: Наук. думка, 1974. – 328 с.
8. Грабер М. Введение в SQL. – М.: Лори, 1996. – 376 с.
9. Дейт К. Введение в системы баз данных. – М.: Вильямс, 2005. – 1328 с.
10. Джексон Г. Проектирование реляционных баз данных для использования с микроЭВМ. – М.: Мир, 1991. – 252 с.
11. Дрибас В. П. Реляционные модели баз данных. – Минск: Изд-во БГУ им. В.И. Ленина, 1982. – 192 с.
12. Заде Л. А. Понятие лингвистической переменной и его применение к принятию приближенных решений. – М.: Мир, 1976. – 165 с.
13. Клини С. К. Математическая логика. – М.: Мир, 1973. – 480 с.
14. Кофман А. Введение в теорию нечетких множеств. – М.: Радио и связь, 1982. – 432 с.
15. Мартин Дж. Организация баз данных в вычислительных системах. – М.: Мир, *1980. – 662 с.*
16. Мейер Д. Теория реляционных баз данных / Пер. с англ. – М.: Мир, 1987. – 608 с.
17. Минский М. Фреймы для представления знаний. – М.: Энергия, 1979. – 152 с.
18. Нагао М., Катаяма Т., Уэмура С. Структуры и базы данных / Пер. с япон. – М.: Мир, 1986. – 197 с.
19. Озкарахан Э. Машины баз данных и управление базами данных / Пер. с англ. – М.: Мир, 1989. 696 с.
20. Осуга С. Обработка знаний / Пер. с япон. – М.: Мир, 1989. – 293 с.
21. Представление и использование знаний / Пер. с япон.; Под ред. Х. Уэно, М. Исидзуки. – М.: Мир, 1989. – 220 с.
22. Приобретение знаний / Пер с япон.; Под ред. С. Осуга, Ю. Саэки. – М.: Мир, 1990. – 194 с.
23. Системы управления базами данных и знаний: Справ. изд. / А. Н. Наумов, А. М. Вендров, В. К. Иванов и др.; Под ред. А. Н. Наумова. – М.: Финансы и статистика, 1991. – 352 с.
24. Стогний А. А., Пасичник В. В. Реляционные модели баз данных. – М.: ЦНТИ «Атоминтформ», 1983. – 296 с.

25. Тиори Т., Фрай Дж. Проектирование структур баз данных: В 2 кн. – М.: Мир, 1985. – Кн. 1. – 287 с.; Кн. 2. – 320 с.

26. Горша-Молина Г., Ульман Дж., Уидом Дж. Системы баз данных. Полный курс – М.: Вильямс, 2003. – 1088 с.

27. Хаббард Дж. Автоматизированное проектирование баз данных. – М.: Мир, 1984. – 294 с.

28. Харрингтон Д. Проектирование объектно-ориентированных баз данных / Пер. с англ. – М.: ДМК Пресс, 2001. – 272 с

29. Цаленко М. Ш. Семантические и математические модели баз данных. – М.: ВИНИТИ, 1985. – 208 с. (Итоги науки и техники. Сер. Информатика – Т. 9)

30. Цикритзис Д., Лоховски Ф. Модели данных. – М.: Финансы и статистика, 1985. – 344 с.

31. Чери С., Готлиб Г., Танака Л. Логическое программирование и базы данных. – М.: Мир, 1992. – 352 с.

32. Язык описания данных КОДАСИЛ. – М.: Статистика, 1981. – 183 с.

33. Aho A. V., Beeri C., Ullman J. D The theory of joins in relational databases // ACM Transactions on Database Systems. – 1979. – Vol. 4, N 3. – P. 297–314.

34. ANSI/X3/SPARC Study Group on Data Base Management Systems: Interim Report. 75-02-08 // ACM SIGMOD Newsletter, FDT. - 1975. – Vol. 7, N 2. · P. 1–140

35. Armstrong W. W. Dependency structures of database relationships: Proc. IFIP Congress, North Holland, Amsterdam. – 1974. – P. 580–583.

36. Berri C., Bernstein P. A., Goodman N. A sophisticate's introduction to database normalization theory: Proc. of 4th Int. Conf. on Very Large data Bases, Western Berlin, Germany (September 13–15). – 1978. – P. 113–124.

37. Beeri C., Fagin R., Howard J. H. A complete axiomatization for functional and multivalued dependencies: Proc. ACM SIGMOD Conf. - 1977. P. 47–61.

38. Chen P. P.-S. The Entity-Relationship Model. Towards a Unified View of Data // ACM Transactions on Database Systems. – 1976. – Vol. 1, N 1. – P. 9–36.

39. Codd E. F. A Relational Model of Data for Large Shared Data banks // Communications ACM. – 1970. – Vol. 13, N 6. – P. 377–383.

40. Codd E. F. Relational Completeness of Data Base Sublanguage. Data Base Systems. Courant Computer Science Symposium 6 (May 24–25, 1971). – S.l.: Prentice-Hall, 1972. – P. 65–98

41. Codd E. F. A Data Base Sublanguage Founded on the Relational Calculus // Proc. ACM SIGFIDET, 1971. Workshop on Data Description, Access and Contol, P. 35–68.

42. Codd E. F. Further Normalization of the Data Base Relational Model: Data Base Systems, Courant Computer Science Symposia Series.– Vol. 6: Englewood Cliffs.– N.Y.: Prentice Hall, 1972.– P. 33–64.

43. Codd E. F. Extending a Database Relational Model to Capture More Meaning // ACM Transactions on Database Systems. – 1979. – Vol. 4, N 4. – P. 397–434.

44. Fagin R. Multivalued Dependencies and a New Normal Form for Relational Databases // ACM Transactions on Database Systems. – 1977. – Vol. 2, N 3. – P. 262–278.

45. Heath I. J. Unacceptable File Operations in Relational Data Base // Proc. ACM SIGFIDET, 1971. Workshop on Data Description, Access and Control. P. 302–318.

46. Teach Yourself SQL in 21 days. – 2d ed. – http://vic.nsau.edu.ru/drosophila/tech/asl21/.

# Алфавітний покажчик

Навчальне видання

**Пасічник Володимир Володимирович,**
**Резніченко Валерій Анатолійович**

**ОРГАНІЗАЦІЯ БАЗ ДАНИХ ТА ЗНАНЬ**

Підручник

Керівник проекту І. О. Завадський
Редактор І. В. Замоцпа
Коректор Н. М. Тонконог
Комп'ютерна верстка З. В. Лобач

ТОВ «Видавнича група BHV»
Свідоцтво про внесення до Державного реєстру України
суб'єктів видавничої справи
ДК №175 від 13.09.2000 р.

Підписано до друку 14.02.06. Формат 70×100 $^{1}/_{16}$.
Папір офсетний. Гарнітура Petersburg. Друк офсетний.
Ум. друк. арк. 30,96. Обл.-вид. арк. 29,03
Наклад 3000 прим. Зам. №30212

В. В. ПАСІЧНИК, В. А. РЕЗНІЧЕНКО

# ОРГАНІЗАЦІЯ БАЗ ДАНИХ ТА ЗНАНЬ

Затверджено Міністерством освіти і науки України як підручник для студентів вищих навчальних закладів, які навчаються за напрямами «Комп'ютерні науки», «Комп'ютеризовані системи, автоматика і управління», «Комп'ютерна інженерія», «Прикладна математика»

Серія «ІНФОРМАТИКА»
За загальною редакцією академіка НАН України
М. З. Згуровського

Київ
Видавнича група BHV
2006