МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

**ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
МІСЬКОГО ГОСПОДАРСТВА імені О. М. БЕКЕТОВА**

О. Б. Костенко, І. О. Гавриленко

ОРГАНІЗАЦІЯ БАЗ ДАНИХ ТА ЗНАНЬ

КОНСПЕКТ ЛЕКЦІЙ

(для студентів денної і заочної форм навчання першого (бакалаврського) рівня вищої освіти за спеціальністю 126 – Інформаційні системи та технології)

**Харків
ХНУМГ ім. О. М. Бекетова
2021**

Костенко О. Б. Організація баз даних та знань : конспект лекцій (для студентів денної та заочної форм навчання першого (бакалаврського) рівня вищої освіти за спеціальністю 126 – Інформаційні системи та технології) / О. Б. Костенко, І. О. Гавриленко ; Харків. нац. ун-т міськ. госп-ва ім. О. М. Бекетова. – Харків : ХНУМГ ім. О. М. Бекетова, 2021. – 92 с.

Автори:
канд. фіз.-мат. наук, доц. О. Б. Костенко,
ст. викл. І. О. Гавриленко

Рецензент

М. Ю. Карпенко, кандидат технічних наук, доцент кафедри комп'ютерних наук та інформаційних технологій (Харківський національний університет міського господарства імені О. М. Бекетова)

Рекомендовано кафедрою комп'ютерних наук та інформаційних технологій, протокол № 1 від 28 серпня 2020 р.

Конспект лекцій складено відповідно до вимог кредитно-модульної системи організації навчального процесу та узгоджено з орієнтовною структурою змісту навчальної дисципліни, рекомендованою Європейською кредитно-трансферною Системою (ECTS)

ЗМІСТ

ВСТУП

На сьогоднішній день бази даних є невід'ємною частиною більшості інформаційних систем. Спостерігається стрімкий розвиток нових технологій, платформ реалізації і середовищ розробки прикладних програм баз даних.

Розробка БД залишається одним з головних напрямів сучасних інформаційних технологій. Вони дозволяють структурувати інформацію, зберігати і отримувати її оптимальним для користувача чином. Знання основ цього напряму є необхідним фахівцям у галузі інформаційних систем і технологій.

Мета курсу полягає у вивченні принципів організації і побудови баз даних, систем управління базами даних, методів проєктування баз даних і засобів використання систем управління базами даних як складових елементів комп'ютерних систем.

У конспекті лекцій коротко викладено основи сучасної теорії баз даних. Докладно розглянуто реляційну модель даних, яка являється основою практично усіх комерційних систем управління базами даних і найбільш поширена на даний час. Велику увагу приділено мові SQL, оскільки SQL являється стандартною базовою мовою при роботі з базами даних. Деякі аспекти роботи систем управління базами даних також розглядаються в даному курсі.

ЗМІСТОВИЙ МОДУЛЬ 1 МОДЕЛЮВАННЯ ДАНИХ

ТЕМА 1 СИСТЕМИ БАЗ ДАНИХ. ОСНОВНІ ПОНЯТТЯ Й АРХІТЕКТУРА

1.1 Історія розвитку баз даних

В історії обчислювальної техніки можна прослідкувати розвиток двох основних областей її використання. Перша область – застосування обчислювальної техніки для виконання чисельних розрахунків, які занадто довго або взагалі неможливо робити вручну. Розвиток цієї галузі сприяв інтенсифікації методів чисельного рішення складних математичних задач, появі мов програмування, орієнтованих на зручний запис чисельних алгоритмів, становленню зворотнього зв'язку із розробниками нової архітектури ЕОМ. Характерною особливістю даної сфери застосування обчислювальної техніки є наявність складних алгоритмів обробки, які застосовуються до простих за структурою даних, обсяг яких порівняно невеликий.

Друга область, яка безпосередньо відноситься до цієї теми, – це використання засобів обчислювальної техніки в автоматичних або автоматизованих інформаційних системах. Інформаційна система являє собою програмно-апаратний комплекс, що забезпечує виконання таких функцій:

− надійне зберігання інформації в пам'яті комп'ютера;

− виконання специфічних для даного застосування перетворень інформації і обчислень;

− надання користувачам зручного і легкого освоювання інтерфейсу.

Зазвичай такі системи мають справу з великими обсягами інформації, що має досить складну структуру. Класичними прикладами інформаційних систем є банківські системи, автоматизовані системи управління підприємствами, системи резервування авіаційних або залізничних квитків, місць в готелях і т. і.

Друга область використання обчислювальної техніки виникла декілька пізніше першої. Це пов'язано з тим, що на зорі обчислювальної техніки можливості комп'ютерів по зберіганню інформації були дуже обмеженими. Говорити про надійне і довготривале зберігання інформації можна тільки при наявності запам'ятовуючих пристроїв, що зберігають інформацію після виключення електричного живлення. Оперативна (основна) пам'ять комп'ютерів цією властивістю зазвичай не володіє. У перших комп'ютерах використовувалися два види пристроїв зовнішньої пам'яті – магнітні стрічки і

барабани. Ємність магнітних стрічок була досить велика, але по своїй фізичній природі вони забезпечували послідовний доступ до даних. Магнітні ж барабани (вони найближче до сучасних магнітним дискам з фіксованими головками) давали можливість довільного доступу до даних, але мали обмежений об'єм інформації.

Ці обмеження не були дуже істотними для винятково чисельних розрахунків. Навіть якщо програма повинна обробити (або провести) великий обсяг інформації, при програмуванні можна продумати розташування цієї інформації в зовнішній пам'яті (наприклад, на послідовній магнітній стрічці), що забезпечує ефективне виконання цієї програми. Однак в інформаційних системах сукупність взаємопов'язаних інформаційних об'єктів фактично відображає модель об'єктів реального світу. А потреба користувачів в інформації, адекватно відображає стан реальних об'єктів, вимагає порівняно швидкої реакції системи на їх запити. І в цьому випадку наявність порівняно повільних пристроїв зберігання даних, до яких відносяться магнітні стрічки і барабани, було недостатнім.

Можна припустити, що саме вимоги нечислових застосувань викликали появу знімних магнітних дисків з рухомими головками, що стало революцією в історії обчислювальної техніки. Ці пристрої зовнішньої пам'яті володіли істотно більшою ємністю, ніж магнітні барабани, забезпечували задовільну швидкість доступу до даних в режимі довільної вибірки, а можливість зміни дискового пакету на пристрої дозволяла мати практично необмежений архів даних.

З появою магнітних дисків почалася історія систем управління даними у зовнішній пам'яті. До цього кожна прикладна програма, якою потрібно було зберігати дані в зовнішній пам'яті, сама визначала розташування кожної порції даних на магнітній стрічці або барабані і виконувала обміни між оперативною пам'яттю і пристроями зовнішньої пам'яті за допомогою програмно-апаратних засобів низького рівня (машинних команд або викликів відповідних програм операційної системи). Такий режим роботи не дозволяє або дуже утрудняє підтримку на одному зовнішньому носії декількох архівів довго тривало зберігається. Крім того, кожній прикладній програмі доводилося вирішувати проблеми іменування частин даних і структуризації даних у зовнішній пам'яті.

Перший етап – бази даних на великих ЕОМ. Історія розвитку СУБД налічує більше 30 років. У 1968 році була введена в експлуатацію перша промислова СУБД система IMS фірми IBM. У 1975 році з'явився перший стандарт асоціації по мовам систем обробки даних – Conference of Data System Languages

(CODASYL), який визначив ряд фундаментальних понять в теорії систем баз даних, які і досі є основними для мережевої моделі даних.

У подальший розвиток теорії баз даних великий внесок був зроблений американським математиком Е. Ф. Коддом, який є творцем реляційної моделі даних. У 1981 році Е. Ф. Кодд отримав за створення реляційної моделі і реляційної алгебри престижну премію Тьюринга Американської асоціації з обчислювальної техніки.

Менше двох десятків років минуло з цього моменту, але стрімкий розвиток обчислювальної техніки, зміна її принципової ролі в житті суспільства, бум персональних ЕОМ і, нарешті, поява потужних робочих станцій і мереж ЕОМ вплинуло також і на розвиток технології баз даних. Можна виділити чотири етапи в розвитку даного напрямку в обробці даних. Однак необхідно зауважити, що все-таки немає жорстких часових обмежень в цих етапах: вони плавно переходять один в інший і навіть співіснують паралельно, але тим не менше виділення цих етапів дозволить чіткіше охарактеризувати окремі стадії розвитку технології баз даних, підкреслити особливості, специфічні для конкретного етапу.

Перший етап розвитку СУБД пов'язаний з організацією баз даних на великих машинах типу IBM 360/370, ЄС-ЕОМ і міні-ЕОМ типу PDP11 (фірми Digital Equipment Corporation – DEC), різних моделях HP (фірми Hewlett Packard).

Бази даних зберігалися у зовнішній пам'яті центральної ЕОМ, користувачами цих баз даних були завдання, що запускаються в основному в пакетному режимі. Інтерактивний режим доступу забезпечувався за допомогою консольних терміналів, які не володіли власними обчислювальними ресурсами (процесором, зовнішньою пам'яттю) і служили тільки пристроями введення-виведення для центральної ЕОМ. Програми доступу до БД писалися на різних мовах і запускалися як звичайні числові програми. Потужні операційні системи забезпечували можливість умовно паралельного виконання всієї безлічі завдань. Ці системи можна було віднести до систем розподіленого доступу, тому що база даних була централізованою, зберігалася на пристроях зовнішньої пам'яті однієї центральної ЕОМ, а доступ до неї підтримувався від багатьох користувачів-завдань.

Особливості цього етапу розвитку виражаються в наступному:

− усі СУБД базуються на потужних мультипрограмних операційних системах (MVS, SVM, RTE, OSRV, RSX, UNIX), тому в основному

підтримується робота з централізованою базою даних в режимі розподіленого доступу;

− функції управління розподілом ресурсів в основному здійснюються операційною системою (ОС);

− підтримуються мови низького рівня маніпулювання даними, орієнтовані на навігаційні методи доступу до даних;

− значна роль відводиться адмініструванню даних;

− проводяться серйозні роботи по обґрунтуванню і формалізації реляційної моделі даних, і була створена перша система (System R), що реалізує ідеологію реляційної моделі даних;

− проводяться теоретичні роботи по оптимізації запитів і управлінню розподіленим доступом до централізованої БД, було введено поняття транзакції;

− результати наукових досліджень відкрито обговорюються у пресі, йде потужний потік загальнодоступних публікацій, що стосуються всіх аспектів теорії і практики баз даних, і результати теоретичних досліджень активно впроваджуються в комерційні СУБД.

З'являються перші мови високого рівня для роботи з реляційною моделлю даних. Однак відсутні стандарти для цих перших мов.

Другий етап – епоха персональних комп'ютерів. Персональні комп'ютери стрімко увірвалися в наше життя і буквально перевернули наше уявлення про місце і роль обчислювальної техніки в житті суспільства. Тепер комп'ютери стали ближче і доступніше кожному користувачеві. Зник благоговійний страх рядових користувачів перед незрозумілими і складними мовами програмування. З'явилося безліч програм, призначених для роботи непідготовлених користувачів. Ці програми були прості у використанні і інтуїтивно зрозумілі: це перш за все різні редактори текстів, електронні таблиці та інші. Простими і зрозумілими стали операції копіювання файлів і перенесення інформації з одного комп'ютера на інший, роздрук текстів, таблиць та інших документів. Системні програмісти були відсунуті на другий план. Кожен користувач міг себе відчути повним господарем цього потужного і зручного пристрою, що дозволяє автоматизувати багато аспектів діяльності. І, звичайно, це позначилося і на роботі з базами даних. З'явилися програми, які називалися системами управління базами даних і дозволяли зберігати значні обсяги інформації, вони мали зручний інтерфейс для заповнення даних, вбудовані засоби для генерації різних звітів. Ці програми дозволяли автоматизувати багато облікові функції, які раніше велися вручну. Постійне зниження цін на персональні комп'ютери зробило їх доступними не тільки для організацій і фірм, а й для окремих користувачів.

Комп’ютери стали інструментом для ведення документації та власних облікових функцій. Це все зіграло як позитивну, так і негативну роль в області розвитку баз даних. Удавана простота і доступність персональних комп’ютерів і їх програмного забезпечення породила безліч дилетантів. Ці розробники, вважаючи себе знавцями, стали проєктувати недовговічні бази даних, які не враховували багатьох особливостей об’єктів реального світу. Багато було створено систем-одноденок, які не відповідали законам розвитку і взаємозв’язку реальних об’єктів. Однак доступність персональних комп’ютерів змусила користувачів з багатьох областей знань, які раніше не застосовували обчислювальну техніку в своїй діяльності, звернутися до них. І попит на розвинені зручні програми обробки даних змушував постачальників програмного забезпечення поставляти все нові системи, які прийнято називати настільними (desktop) СУБД. Значна конкуренція серед постачальників змушувала удосконалювати ці системи, пропонуючи нові можливості, покращуючи інтерфейс і швидкодію систем, знижуючи їх вартість. Наявність на ринку великої кількості СУБД, що виконують подібні функції, зажадало розробки методів експорту-імпорту даних для цих систем і відкриття форматів зберігання даних.

Але і в цей період з’являлися аматори, які всупереч здоровому глузду розробляли власні СУБД, використовуючи стандартні мови програмування. Це був тупиковий варіант, тому що подальший розвиток показав, що перенести дані з нестандартних форматів в нові СУБД було набагато важче, а в деяких випадках вимагало таких трудовитрат, що краще було б все розробити наново, але дані все рівно треба було переносити на нову більш перспективну СУБД. І це теж було результатом недооцінки тих функцій, які повинна була виконувати СУБД.

Особливості цього етапу:

− усі СУБД були розраховані на створення БД в основному з монопольним доступом. І це зрозуміло. Комп’ютер персональний, він не був приєднаний до мережі, і база даних на ньому створювалася для роботи одного користувача. У рідкісних випадках передбачалася послідовна робота декількох користувачів, наприклад, спочатку оператор, який вводив бухгалтерські документи, а потім головбух, який визначав проводки, відповідні первинним документам;

− більшість СУБД мали розвинений і зручний призначений для користувача інтерфейс. У більшості існував інтерактивний режим роботи з БД як в межах опису БД, так і в рамках проєктування запитів. Крім того, більшість СУБД пропонували розвинений і зручний інструментарій для розробки готових додатків без програмування. Інструментальне середовище складалося з готових

елементів програми у вигляді шаблонів екранних форм, звітів, етикеток (Labels), графічних конструкторів запитів, які досить просто могли бути зібрані в єдиний комплекс;

− у всіх настільних СУБД підтримувався тільки зовнішній рівень представлення реляційної моделі, тобто тільки зовнішній табличний вигляд структур даних;

− при наявності високорівневих мов маніпулювання даними типу реляційної алгебри і SQL в настільних СУБД підтримувалися низькорівневі мови маніпулювання даними на рівні окремих рядків таблиць;

− у настільних СУБД були відсутні засоби підтримки посилальної і структурної цілісності бази даних. Ці функції повинні були виконувати додатки, проте недосконалість засобів розробки додатків іноді не дозволяла це зробити, і в цьому випадку ці функції повинні були виконуватися користувачем, вимагаючи від нього додаткового контролю при введенні і зміні інформації, що зберігається в БД;

− наявність монопольного режиму роботи фактично призвело до виродження функцій адміністрування БД і в зв'язку з цим – до відсутності інструментальних засобів адміністрування БД;

− і, нарешті, остання і зараз дуже позитивна особливість – це порівняно скромні вимоги до апаратного забезпечення з боку настільних СУБД. Цілком працездатні застосування, розроблені, наприклад, на Clipper, працювали на PC 286. В принципі, їх навіть важко назвати повноцінними СУБД. Яскраві представники цього сімейства - дуже широко використовувалися до недавнього часу СУБД Dbase (DbaseIII +, DbaseIV), FoxPro, Clipper, Paradox.

Третій етап – розподілені бази даних. Добре відомо, що історія розвивається по спіралі, тому після процесу «персоналізації» почався зворотний процес – інтеграція. Множиться кількість локальних мереж, все більше інформації передається між комп'ютерами, гостро постає завдання узгодженості даних, що зберігаються і обробляються в різних місцях, але логічно один з одним пов'язаних, виникають завдання, пов'язані з паралельною обробкою транзакцій – послідовностей операцій над БД, що переводять її з одного несуперечливого стану в інший. Успішне вирішення цих завдань призводить до появи розподілених баз даних, що зберігають всі переваги настільних СУБД і в той же час дозволяють організувати паралельну обробку інформації і підтримку цілісності БД.

Особливості етапу:

− практично всі сучасні СУБД забезпечують підтримку повної реляційної моделі, а саме: допустимими є тільки дані, представлені у вигляді відношень реляційної моделі (структурна цілісність); мови маніпулювання даними високого рівня (мовна цілісність); контроль за дотриманням посилальної цілісності протягом всього часу функціонування системи і гарантія неможливості з боку СУБД порушити ці обмеження (посилальна цілісність);

− більшість сучасних СУБД розраховані на багатоплатформну архітектуру, тобто вони можуть працювати на комп'ютерах з різною архітектурою і під різними операційними системами, при цьому для користувачів доступ до даних, керованим СУБД на різних платформах, практично невиразний;

− необхідність підтримки роботи багатьох користувачів бази даних і можливість децентралізованого зберігання даних зажадали розвитку засобів адміністрування БД з реалізацією загальної концепції засобів захисту даних;

− потреба в нових реалізаціях викликала створення серйозних теоретичних праць з оптимізації реалізацій розподілених БД і роботі з розподіленими транзакціями і запитами з впровадженням отриманих результатів в комерційні СУБД;

− для того щоб не втратити клієнтів, які раніше працювали на настільних СУБД, практично всі сучасні СУБД мають засоби підключення клієнтських додатків, розроблених з використанням настільних СУБД, і кошти експорту даних з форматів настільних СУБД другого етапу розвитку;

− саме до цього етапу можна віднести розробку ряду стандартів в рамках мов опису і маніпулювання даними починаючи з SQL89, SQL92, SQL99 і технологій по обміну даними між різними СУБД, до яких можна віднести і протокол ODBC (Open DataBase Connectivity), запропонований фірмою Microsoft;

− саме до цього етапу можна віднести початок робіт, пов'язаних з концепцією об'єктно-орієнтованих БД (ООБД). Представниками СУБД, що належать до другого етапу, можна вважати MS Access 97 і всі сучасні сервери баз даних Oracle7.3, Oracle 8.4 MS SQL6.5, MS SQL7.0, System 10, System 11, Informix, DB2, SQL Base і інші сучасні сервери баз даних, яких зараз налічується кілька десятків.

Четвертий етап – перспективи розвитку систем управління базами даних. Цей етап характеризується появою нової технології доступу до даних – *інтранет*. Основна відмінність цього підходу від технології клієнт-сервер

полягає в тому, що відпадає необхідність використання спеціалізованого клієнтського програмного забезпечення. Для роботи з віддаленою базою даних використовується стандартний браузер Інтернету, наприклад Microsoft Internet Explorer або Netscape Navigator, і для кінцевого користувача процес звернення до даних відбувається аналогічно ковзанню по Всесвітній Павутині При цьому вбудований в завантажуються користувачем HTML-сторінки код, написаний зазвичай на мові Java, Java-script, Perl і інших, відстежує всі дії користувача і транслює їх в низькорівневі SQL-запити до бази даних, виконуючи, таким чином, ту роботу, якої в технології клієнт-сервер займається клієнтська програма. Зручність даного підходу призвела до того, що він став використовуватися не тільки для віддаленого доступу до баз даних, але і для користувачів локальної мережі підприємства. Прості завдання обробки даних, які пов'язані зі складними алгоритмами, які вимагають узгодженої зміни даних у багатьох взаємопов'язаних об'єктах, досить просто і ефективно можуть бути побудовані по даній архітектурі. В цьому випадку для підключення нового користувача до можливості використовувати наявне завдання не потрібна установка додаткового клієнтського програмного забезпечення. Однак алгоритмічно складні завдання рекомендується реалізувати в архітектурі «клієнт-сервер» з розробкою спеціального клієнтського програмного забезпечення.

У кожного з перерахованих вище підходів до роботи з даними є свої переваги і свої недоліки, які і визначають область застосування того чи іншого методу, і в даний час всі підходи широко використовуються.

1.2 Архітектура та основні технічні характеристики систем управління базами даних

Активна діяльність щодо пошуку засобів усуспільнення безупинно зростаючого обсягу інформації призвела до створення на початку 60-х років спеціальних програмних комплексів, що називаються «Системи управління базами даних» (далі – СУБД). Основна особливість СУБД – це наявність процедур для введення і збереження не тільки самих даних, але і описів їх структури. Файли, які забезпечені описом збережених у них даних і знаходяться під управлінням СУБД, стали називати банки даних, а потім «Бази даних» (далі – БД).

СУБД повинна надавати доступ до даних будь-яким користувачам, включаючи і тих, які практично не мають і (або) не хочуть мати уявлення про:

– фізичне розміщення в пам'яті даних і їх описів;

– механізми пошуку запитуваних даних;

– проблеми, що виникають при одночасному запиті одних і тих же даних багатьма користувачами (прикладними програмами);

– засоби забезпечення захисту даних від некоректних оновлень і (або) несанкціонованого доступу;

– підтримку баз даних в актуальному стані та безлічі інших функцій СУБД.

При виконанні основних функцій СУБД повинна використовувати різні описи даних. А як створювати ці описи?

Природно, що проєкт бази даних треба починати з аналізу предметної області та виявлення вимог до неї окремих користувачів (співробітників організації, для яких створюється база даних). Детальніше цей процес буде розглянутий нижче, а тут зазначимо, що проєктування зазвичай доручається людині або групі осіб – адміністратору бази даних (далі – АБД). Ним може бути як спеціально призначений співробітник організації, так і майбутній користувач бази даних, добре обізнаний із машинною обробкою даних.

Узагальнивши приватні уявлення про вміст бази даних, отримані шляхом опитування користувачів, і свої уявлення про дані, які можуть знадобитися в майбутніх додатках, АБД спочатку створює узагальнений неформальний опис створюваної бази даних. Такий опис, що виконується з використанням природної мови, математичних формул, таблиць, графіків і інших засобів, зрозумілих усім людям, які працюють над проєктуванням бази даних, називають інфологічною моделлю даних.

Така людино-орієнтована модель повністю незалежна від фізичних параметрів середовища зберігання даних. Зрештою цим середовищем може бути пам’ять людини, а не обчислювальна машина. Тому інфологічна модель не повинна змінюватися до тих пір, доки зміни в реальному світі не зажадають зміни в ній деякого визначення, щоб ця модель продовжувала відображати предметну область.

Решта моделей є компьютеро-орієнтованими. За їх допомогою СУБД дає змогу програмам і користувачам здійснювати доступ до збережених даних лише за їх іменами, не піклуючись про фізичне розташування цих даних. Потрібні дані відшукуються СУБД на зовнішніх запам’ятовуючих пристроях з фізичної моделі даних.

Оскільки зазначений доступ здійснюється за допомогою конкретної СУБД, моделі повинні бути описані мовою опису даних цієї СУБД. Такий опис,

створюваний АБД за інфологічною моделлю даних, називають даталогічною моделлю даних.

Основна мета СУБД у тому, щоб запропонувати користувачеві абстрактне представлення даних, приховавши конкретні особливості збереження і управління ними. Оскільки база даних є спільною ресурсом, то кожному користувачеві може знадобитися своє уявлення про характеристики інформації, яка зберігається базою даних. Для задоволення цих потреб архітектура більшості СУБД будується з урахуванням так званої архітектури ANSI-SPARC, яку розглянемо нижче.

Концепції багаторівневої інформаційної архітектури стали основою сучасної технології баз даних. Ці ідеї пов'язують із опублікованим у 1975 року звітом робочої групи з баз даних ANSI/X3/SPARC (Комітету з планування стандартів Американського національного інституту стандартів). У цьому звіті була запропонована узагальнена трирівнева модель інформаційної архітектури системи бази даних, куди входять концептуальний, внутрішній і зовнішній рівні. Така модель описує архітектуру будь-якої СУБД.

Концептуальний рівень архітектури – це структурний рівень, що визначає логічну схему бази даних. На даному рівні виконується концептуальне проєктування бази даних, яке включає аналіз інформаційних потреб користувачів та визначення потрібних їм елементів даних. Він служить на підтримку єдиного «погляду» на бази даних, загального для всіх її додатків й незалежного від нього. Представлення бази даних на концептуальному рівні системи описується концептуальної схемою бази даних.

Внутрішній рівень архітектури – це рівень, що визначає фізичний вид бази даних, найбільш близький до фізичного зберігання і пов'язаний зі способами збереження інформації на фізичних пристроях зберігання. Це єдиний рівень інформаційної архітектури, де база даних насправді представлена цілком у «матеріалізованому» вигляді. Опис представлення бази даних на цьому рівні архітектури називається внутрішньої схемою чи схемою зберігання. На внутрішньому рівні зберігається наступна інформація:

– інформація про розподіли дискового простору для зберігання даних і індексів;

– опис подробиць збереження записів (із зазначенням реальних розмірів елементів даних, що зберігаються);

– інформація про розміщення записів;

– інформація про стиснення даних і методах їх шифрування.

Зовнішній рівень архітектури – це структурний рівень БД, що визначає призначені для користувача представлення даних. Кожна користувальницька група отримує своє власне уявлення даних в БД. Кожне таке представлення даних дає орієнтований на користувача опис елементів даних, з яких складається представлення даних, і відношень між ними. Його можна безпосередньо вивести з концептуальної схеми. Сукупність таких користувальницьких представлень даних і дає зовнішній рівень. Опис таких представлень називають зовнішніми схемами. У системі бази даних може водночас підтримуватися кілька зовнішніх схем щодо різноманітних груп користувачів.

Слід зазначити, що у запропонованій архітектурній моделі необхідно підтримувати відповідність між уявленнями бази даних на суміжних рівнях архітектури СУБД. У моделі ANSI/X3/SPARC для цієї мети служать механізми міжрівневого відображення даних «зовнішній-концептуальний» і «концептуальний-внутрішній». Саме ці механізми забезпечують абстракцію даних у системі, визначають ступінь незалежності даних.

Основним призначенням трирівневої архітектури СУБД є забезпечення незалежності від даних. Це означає, що на нижніх рівнях неможливо впливати на верхні рівні. Розрізняють два типу незалежності від даних: логічний й фізичний.

Логічна незалежність від даних означає повну захищеність зовнішніх схем змін, внесених в концептуальну схему. Такі зміни концептуальної схеми, як додавання чи видалення нових сутностей, атрибутів чи зав’язків, має здійснюватися без внесення змін – у вже існуючі зовнішні схеми чи переписування прикладних програм.

Фізична незалежність від даних означає захищеність концептуальної схеми змін, внесених у внутрішню схему. Такі зміни внутрішньої схеми, як використання різних файлових систем чи структур зберігання, різних пристроїв зберігання, модифікація індексів чи змішування, має здійснюватися без необхідності внесення змін – у концептуальну чи зовнішню схеми. Користувачем можуть бути здійснені зміни лише у загальну продуктивності системи.

До основних технічних характеристик СУБД належать:

1. Продуктивність – визначається часом виконання запитів, швидкістю пошуку інформації, часом імпортування бази даних з інших форматів, часом генерації звіту та ін.

2. Забезпечення цілісності даних на рівні бази даних – наявність засобів, які дозволяють упевнитися, що інформація в базі коректна і повна.

3. Гарантування безпеки – шифрування прикладних програм, шифрування даних, захист паролем, обмеження рівня доступу до бази даних чи до таблиці.

4. Робота в багатокористувацьких середовищах – припускає можливість блокування бази даних, файлу, запису, поля, а також ідентифікацію станції, встановила блокування, обробку транзакцій – послідовності операцій користувача над базою даних, що зберігає її логічну цілісність, роботу з мережевими операційними системами.

5. Можливість імпорту та експорту інформації, підготовленої іншими програмними засобами.

6. Інструментальні засоби розробки прикладних програм (мови програмування, засоби генерації додатків, реалізації меню, форм, звітів). Реалізація мовних засобів здійснюється не однаково – для розробників програмних продуктів мова представляється в явній синтаксичній формі (наприклад, в Access – мова Access Basic, в FoxPro і dBase – мова xBase). Для користувачів функції мови часто доступні непрямим чином – через меню, діалогові сценарії і т. д. При цьому синтаксичні конструкції мови формуються автоматично і передаються на виконання.

Мовні засоби використовуються для виконання двох функцій:

– опису представлення бази даних, що включає опис структури бази даних і обмежень цілісності, які накладаються на неї, іноді забезпечує можливість обмеження доступу до даних;

– мови маніпулювання даними – запитує операції над даними БД.

Деякі мови забезпечують як можливість опису даних, так і маніпулювання ними, до їх числа належить SQL.

7. Доступ до даних за допомогою мови SQL (Structured Query Language) – структурована мова запитів – ця мова є міжнародним стандартом мови запитів і особливо важлива при проведенні роботи з корпоративними даними.

1.2 Логічна та фізична структура баз даних

У базі даних відображається інформація про певну предметну область. Предметною областю називається частина реального світу, що представляє інтерес для даного дослідження (використання). В автоматизованих інформаційних системах відображення предметної області представлено моделями даних декількох рівнів. Кількість рівнів моделей буде залежати від особливостей СУБД. Далі розглядаються питання баз даних, що підтримують структуровані моделі даних. Незалежно від того, чи підтримуються в явному

вигляді окремо моделі логічного та фізичного рівня, з точки зору методології все одно можна виділити ці рівні моделей і відповідні їм етапи проєктування баз даних.

Даталогічна модель бази даних. Даталогічна модель є моделлю логічного рівня і являє собою відображення логічних зав'язків між елементами даних безвідносно до їх змісту і середовищі зберігання. Ця модель будується в термінах інформаційних одиниць, допустимих в тій конкретній СУБД, в середовищі якої ми проєктуємо базу даних. Етап створення даталогічної моделі називається даталогічним проєктуванням. Опис логічної структури бази даних на мові СУБД називається схемою.

Фізична модель бази даних. Для прив'язки даталогічної моделі до середовища зберігання використовується модель даних фізичного рівня (для стислості часто звана фізичною моделлю). Ця модель визначає запам'ятуючі пристрої, які використовуються, способи фізичної організації даних у середовищі зберігання. Модель фізичного рівня також будується з урахуванням можливостей, що надаються СУБД. Опис фізичної структури бази даних називається схемою зберігання. Відповідний етап проєктування БД називається фізичним проєктуванням.

СУБД володіють різними можливостями з фізичної організації даних, у зв'язку з чим складність і трудомісткість фізичного проєктування, набір виконуваних кроків розрізняються для конкретних систем. До числа робіт, що виконуються на етапі фізичного проєктування, відносяться:

– вибір типу носія, способу організації даних, методів доступу;

– визначення розміру фізичного блоку;

– управління розміщенням даних на зовнішньому носії;

– управління вільною пам'яттю;

– визначення доцільності стиснення даних і використовуваних методів стиснення;

– оцінка фізичної моделі даних.

До фізичного проєктування відносяться проблеми, пов'язані з буферизацією – визначення числа і розмірів буферів, які використовуються при передачі даних із зовнішньої пам'яті у внутрішню, закріплення файлів.

У цей час спостерігається тенденція до скорочення робіт на стадії фізичного проєктування. Іноді ці роботи взагалі бувають приховані від проєктувальника.

Зовнішня модель. У деяких СУБД, крім опису загальної логічної структури бази даних, є можливість описати логічну структуру БД з точки зору конкретного

користувача. Така модель називається зовнішньою, а її опис називається підсхемою. Якщо СУБД «підтримує» схему, схему зберігання і підсхему, то вона є СУБД з трирівневої архітектурою.

Якщо СУБД підтримує рівень підсхем, то перед проєктувальником постає завдання їх визначення. Це теж можна розглядати як етап проєктування БД.

Зовнішня модель не завжди є точною складовою частиною схеми. Деякі СУБД допускають відмінності в типах даних, визначених у схемі і підсхемі, і забезпечують їх перетворення, допускаються різний логічний порядок проходження елементів у схемі і підсхемі, введення в підсхему віртуальних полів і т. д. Якщо визначена підсхема, то користувач має доступ тільки до тих даних, які відображені у відповідній підсхемі, що є одним зі способів захисту інформації від несанкціонованого доступу.

У підсхемі часто задається не тільки логічна структура частини бази даних з точки зору конкретного користувача, а й допустимі режими обробки в рамках цієї підсхеми, що служить додатковим механізмом захисту інформації від руйнування.

Використання апарату підсхем полегшує роботу користувача, так як він повинен знати структуру не всієї бази даних, а тільки тієї її частини, яка має безпосереднє відношення до нього. Крім того, ця структура пристосована до його потреб.

У тих випадках, коли СУБД в явному вигляді не підтримує підсхеми, перераховані функції можуть виконувати інші компоненти системи. Близьким до поняття підсхеми є поняття «погляд» (view), яке в даний час широко використовується в англомовній літературі з реляційним СУБД.

ТЕМА 2 МОДЕЛІ ДАНИХ. РЕЛЯЦІЙНА МОДЕЛЬ ДАНИХ

2.1 Класифікація моделей даних

Одними з основоположних в концепції баз даних є узагальнені категорії «дані» і «модель даних».

Поняття *«дані»* в концепції баз даних – це набір конкретних значень, параметрів, що характеризують об’єкт, умова, ситуацію або будь-які інші фактори. Дані не володіють певною структурою, дані стають інформацією тоді, коли користувач задає їм певну структуру, тобто усвідомлює їх смисловий зміст. Тому центральним поняттям в області баз даних є поняття моделі. Не існує однозначного визначення цього визначення, у різних авторів ця абстракція

визначається з деякими відмінностями, але тим не менш можна виділити щось загальне в цих визначеннях.

Модель даних – це деяка абстракція, яка, будучи застосовною до конкретних даних, дозволяє користувачам і розробникам трактувати їх уже як інформацію, тобто відомості, що містять не тільки дані, але і взаємозв'язок між ними.

Відповідно до розглянутої раніше трирівневої архітектури виникає поняття моделі даних стосовно кожного рівня. І дійсно, фізична модель даних оперує категоріями, що стосуються організації зовнішньої пам'яті і структур зберігання, використовуваних в даному операційному середовищі. На даний момент в якості фізичних моделей використовуються різні методи розміщення даних, засновані на файлових структурах: це організація файлів прямого та послідовного доступу, індексних файлів і інвертованих файлів, файлів, що використовують різні методи хешування, взаємопов'язаних файлів. Крім того, сучасні СУБД широко використовують сторінкову організацію даних. Фізичні моделі даних, засновані на сторінкової організації, є найбільш перспективними.

Найбільший інтерес викликають моделі даних, що використовуються на концептуальному рівні. По відношенню до них зовнішні моделі називаються підсхемами і використовують ті ж абстрактні категорії, що і концептуальні моделі даних.

Крім трьох розглянутих рівнів абстракції при проєктуванні БД існує ще один рівень, що передує їм. Модель цього рівня повинна виражати інформацію про предметну область у вигляді, незалежному від використовуваної СУБД. Ці моделі називаються *інфологічними*, або *семантичними*, і відображають в природній і зручній для розробників та інших користувачів формі інформаційно-логічний рівень абстрагування, пов'язаний з фіксацією і описом об'єктів предметної області, їх властивостей і їх взаємозв'язків.

Інфологічні моделі даних використовуються на ранніх стадіях проєктування для опису структур даних в процесі розробки програми, а *даталогічні моделі* вже підтримуються конкретною СУБД.

Документальні моделі даних відповідають уявленню про слабоструктуровану інформацію, орієнтовану в основному на вільні формати документів, текстів природною мовою.

Моделі, засновані на мовах розмітки документів, пов'язані перш за все зі стандартною спільною мовою розмітки – SGML (Standart Generalised Markup Language), яка була затверджена ISO як стандарт ще в 80-х роках.

Ця мова призначена для створення інших мов розмітки, вона визначає припустимий набір тегів (посилань), їх атрибути і внутрішню структуру документа. Контроль за правильністю використання тегів здійснюється за допомогою спеціального набору правил, званих DTD-описами, які використовуються програмою клієнта при розборі документа. Для кожного класу документів визначається свій набір правил, що описують граматику відповідної мови розмітки. За допомогою SGML можна описувати структуровані дані, організовувати інформацію, що міститься в документах, представляти цю інформацію в деякому стандартизованому форматі. Але зважаючи на деяку свою складність мова SGML використовувалася в основному для опису синтаксису інших мов (найбільш відомою з яких є HTML), і мало які додатки працювали з SGML-документами безпосередньо.

Набагато більш проста і зручна, ніж SGML, мова HTML дозволяє визначати оформлення елементів документа і має якийсь обмежений набір інструкцій – тегів, за допомогою яких здійснюється процес розмітки. Інструкції HTML в першу чергу призначені для управління процесом виведення вмісту документа на екрані програми-клієнта і визначають цим самим спосіб представлення документа, але не його структуру. В якості елемента гіпертекстової бази даних, описуваної HTML, використовується текстовий файл, який може легко передаватися по мережі з використанням протоколу HTTP. Ця особливість, а також те, що HTML є відкритим стандартом і величезна кількість користувачів має можливість застосовувати можливості цієї мови для оформлення своїх документів, безумовно, вплинули на зростання популярності HTML і зробили його сьогодні головним механізмом подання інформації в Інтернеті.

Однак HTML сьогодні вже не задовольняє повною мірою вимогам, що пред’являються сучасними розробниками до мов подібного роду. І їй на зміну була запропонована нова мову гіпертекстової розмітки, потужна, гнучкий і, одночасно з цим, зручна мова XML. У чому ж полягають її переваги?

XML (Extensible Markup Language) – це мова розмітки, що описує цілий клас об’єктів даних, званих XML-документами. Вона використовується як засіб для опису граматики інших мов і контролю за правильністю складання документів. Тобто сама по собі XML не містить ніяких тегів, призначених для розмітки, вона просто визначає порядок їх створення.

Тезаурусні моделі засновані на принципі організації словників, містять певні мовні конструкції і принципи їх взаємодії в заданій граматиці. Ці моделі ефективно використовуються в системах-перекладачах, особливо багатомовних

перекладачах. Принцип зберігання інформації в цих системах і підпорядковується тезаурусним моделям.

Дескрипторні моделі – найпростіші з документальних моделей, вони широко використовувалися на ранніх стадіях використання документальних баз даних. У цих моделях кожному документу відповідав дескриптор – описувач. Цей дескриптор мав жорстку структуру і описував документ відповідно до тих характеристик, які потрібні для роботи з документами в документальній БД, яка розробляється. Наприклад, для БД, що містить опис патентів, дескриптор містив назву області, до якої належав патент, номер патенту, дату видачі патенту і ще ряд ключових параметрів, які заповнювалися для кожного патенту. Обробка інформації в таких базах даних велася виключно за дескрипторів, тобто за тими параметрами, які характеризували патент, а не за самим текстом патенту.

2.2 Інфологічна модель даних як інструмент відтворення предметної області

Для того щоб спроєктувати структуру бази даних, необхідно мати вихідну інформацію про предметну область. Бажано, щоб ця інформація була представлена в формалізованому вигляді. Інформація, необхідна для проєктування БД, мало залежить від особливостей СУБД. Опис предметної області, виконаний без орієнтації на використовувані надалі програмні і технічні засоби, називається *інфологічною моделлю предметної області*. Інфологічна модель даних – це опис, виконаний із використанням природної мови, математичних формул, таблиць, графіків і інших засобів, зрозумілих усім людям, що працюють над проєктуванням бази даних.

Інфологічна модель предметної області будується першою. Попередня інфологічна модель будується ще на передпроєктної стадії і потім уточнюється на більш пізніх стадіях проєктування. Потім на її основі будується даталогічна модель. Фізична та зовнішня моделі після цього можуть будуватися в будь-якій послідовності, в тому числі і паралельно.

Основною вимогою до інфологічної моделі, що випливають з її призначення, є вимога адекватного відображення предметної області. Інфологічна модель повинна бути несуперечливою.

Незважаючи на те, що реальний світ, який відображається в інфологічної моделі, є за своєю природою нескінченним, інфологічна модель є кінцевою, що забезпечується чітким обмеженням предметної області. У зв’язку з цим інфологічна модель повинна мати властивість легкої розширюваності, що

забезпечує введення нових даних без зміни раніше визначених. Те ж саме можна сказати і про видалення даних. У зв'язку з великою розмірністю реальних інфологічних моделей повинна забезпечуватися можливість композиції і декомпозиції моделі.

Інфологічна модель повинна легко сприйматися різними категоріями користувачів. Бажано, щоб інфологічну модель будував фахівець, який працює в цій предметній області, а не проєктувальник систем машинної обробки даних, Інфологічна модель повинна також легко і однозначно сприйматися всіма фахівцями, які надалі беруть участь в процесі проєктування баз даних та програмного забезпечення.

Метою інфологічного моделювання є забезпечення найбільш природних для людини способів збору і представлення тієї інформації, яку передбачається зберігати в створюваній базі даних. Тому інфологічну модель даних намагаються будувати за аналогією з природною мовою. Основними конструктивними елементами інфологічних моделей є сутності, зв'язки між ними і їх властивості (атрибути).

Сутність – будь-який помітний об'єкт (об'єкт, який ми можемо відрізнити від іншого), інформацію про який необхідно зберігати в базі даних. Необхідно розрізняти такі поняття, як тип сутності й екземпляр сутності. Поняття тип сутності відноситься до набору однорідних особистостей, предметів, подій або ідей, які виступають як ціле. Екземпляр сутності відноситься до конкретної речі в наборі. Наприклад, типом сутності може бути МІСТО, а екземпляром – Київ. Сутності бувають:

1) стрижневі – це незалежні сутності, що не є ні характеристиками, ні асоціаціями;

2) асоціативні – це зв'язок виду «багато-до-багатьох» між двома або більш сутностями або екземплярами сутності. Асоціації розглядаються як повноправні сутності:

– вони можуть брати участь в інших асоціаціях і позначеннях точно так же, як стрижневі сутності;

– вони можуть мати властивості, тобто мати не тільки набір ключових атрибутів, необхідних для вказівки зв'язків, а й будь-яке число інших атрибутів, що характеризують зв'язок.

3) характеристичні – це зв'язок виду «багато-до-одного» або «одна-до-одного» між двома сутностями (окремий випадок асоціації). Мета характеристики в рамках розглянутої предметної області полягає в описі чи уточнення деякої іншої сутності. Необхідність в них виникає в зв'язку з тим, що

сутності реального світу мають іноді багатозначні властивості. Наприклад, книга може мати кілька характеристик перевидання.

4) підклас асоціативних сутностей – позначення – це зв'язок виду «багато-до-одного» або «одна-до-одного» між двома сутностями і відрізняється від характеристики тим, що не залежить від позначається сутності.

Атрибут – пойменована характеристика сутності. Найменування повинно бути унікальним для конкретного типу сутності, але може бути однаковим для різного типу сутностей. Використовуються для визначення того, яка інформація повинна бути зібрана про сутність.

Ключ – це мінімальний набір атрибутів, за значеннями яких можна однозначно знайти необхідний екземпляр сутності. Мінімальність означає, що виключення з набору будь-якого атрибута не дозволяє ідентифікувати сутність по що залишилися. Кожна сутність має хоча б одним можливим ключем. Один з них приймається за первинний ключ.

Сурогатний ключ – автоматично згенероване значення, ніяк не пов'язане з інформаційним змістом сутності.

Один з можливих природних ключів або сурогатний ключ приймається за первинний ключ. При виборі первинного ключа слід віддавати перевагу несоставним ключам або ключам, складеним з мінімального числа атрибутів. Недоцільно також використовувати ключі з довгими текстовими значеннями (краще використовувати цілочисельні атрибути).

Не допускається, щоб первинний ключ стрижневої сутності (будь-який атрибут, який бере участь в первинному ключі) брав невизначене значення. Інакше виникне суперечлива ситуація: з'явиться екземпляр, який не володіє індивідуальністю, отже, не існуючий екземпляр стрижневий суті. З тих же причин необхідно забезпечити унікальність первинного ключа.

Особливості зовнішніх ключів полягають у наступному:

– якщо сутність В пов'язує суті А і Б, то вона повинна включати зовнішні ключі, відповідні первинним ключам сутностей А і Б;

– якщо сутність Б характеризує сутність А, то вона повинна включати зовнішній ключ, відповідний первинному ключу суті А.

Виникає перше питання: чи може даний зовнішній ключ приймати невизначені значення (NULL-значення). Інакше кажучи, чи може існувати деякий екземпляр сутності даного типу, для якого невідома цільова сутність, що вказується зовнішнім ключем. Відповідь на це питання не залежить від примхи проєктувальника бази даних, а визначається фактичним чином дій, прийнятим в тій частині реального світу, яка повинна бути представлена в базі даних.

Друге питання: що повинно статися при спробі видалення екземпляра сутності, на первинний ключ якої посилається зовнішній ключ? Для даної операції існує три можливості:

– операція каскадується, щоб видалити також всі поставки видаляється постачальника;

– операція обмежується – видаляються лише ті постачальники, які ще не здійснювали поставок. Інакше операція видалення відкидається;

– операція встановлюється – для всіх поставок видаляється постачальника зовнішній ключ встановлюється в невизначене (NULL) значення, а потім цей постачальник видаляється. Така можливість, звичайно, не застосовується, якщо даний зовнішній ключ не повинен містити NULL-значень.

Третє питання: що повинно відбуватися при спробі оновлення первинного ключа сутності, на яку посилається деякий зовнішній ключ? Цей випадок для визначеності знову розглянемо докладніше. Є ті ж три можливості, як і при видаленні:

– операція каскадується з тим, щоб оновити також і зовнішній ключ в поставках цього постачальника;

– операція обмежується – оновлюються первинні ключі лише тих постачальників, які ще не здійснювали поставок. Інакше операція оновлення відкидається;

– операція встановлюється – для всіх поставок поновлюваного постачальника зовнішній ключ встановлюється в невизначене значення, а потім оновлюється первинний ключ постачальника. Така можливість, звичайно, не застосовується, якщо даний зовнішній ключ не повинен містити NULL- значень.

Таким чином, для кожного зовнішнього ключа в проєкті проєктувальник бази даних повинен специфікувати не тільки поле або комбінацію полів, що складають цей зовнішній ключ і сутність, яка ідентифікується цим ключем, але також і відповіді на зазначені раніше питання (три обмеження, які відносяться до цього зовнішньому ключу).

Для характеристик, існування яких залежить від сутностей, що характеризуються, розглянуті раніше обмеження на зовнішній ключ повинні Вони специфіковані в такий спосіб:

– NULL-значення не є допустимими;

– видалення з сутності, яка характеризується, каскадується;

– оновлення (первинний ключ сутності) каскадується.

Зазначені специфікації представляють залежність по існуванню характеристичних сутностей.

Зв’язок – асоціювання двох або більше сутностей. Одне з основних вимог до організації бази даних – це забезпечення можливості відшукання одних сутностей за значеннями інших, для чого необхідно встановити між ними певні зв’язки. А так як в реальних базах даних нерідко містяться сотні або навіть тисячі сутностей, то теоретично між ними може бути встановлено більше мільйона зв’язків. Наявність такої великої кількості зв’язків і визначає складність інфологічних моделей.

При визначенні інфологічної моделі необхідно брати до уваги таке:

– база даних повинна задовольняти актуальним інформаційним потребам організації. Одержувана інформація повинна за структурою й утриманням відповідати розв’язуваним задачам;

– база даних повинна забезпечувати одержання необхідних даних за прийнятний час, тобто відповідати заданим вимогам продуктивності;

– база даних повинна задовольняти виявленим і знову виникаючим вимогам усіх користувачів;

– база даних повинна легко розширюватися при реорганізації і розширенні предметної області;

– база даних повинна легко змінюватися при зміні програмного й апаратного середовища.

Інфологічна модель даних «сутність-зв’язок». Інфологічна модель відображає реальний світ у деякій зрозумілі людині концепції, цілком незалежні від параметрів середовища збереження даних. Існує множина підходів до побудови таких моделей: графові моделі, семантичні мережі, модель «сутність-зв’язок» і т. д. Найбільше популярної з них виявилася модель сутність-зв’язок» або названа ще ER-моделлю.

На використанні різновидів ER-моделі заснована більшість сучасних підходів до проєктування баз даних (головним чином, реляційних). Модель була запропонована Ченом у 1976 р. Моделювання предметної області базується на використанні графічних діаграм, що включають невеличке число різнорідних компонентів. У зв’язку з наочністю уявлення концептуальних схем баз даних ER-моделі одержали широке поширення в системах CASE, що підтримують автоматизоване проєктування реляційних баз даних. У них сутності зображуються позначеними прямокутниками, асоціації (зв’язки) – позначеними ромбами або шестикутниками, атрибути – позначеними овалами, а зв’язки між ними – ненаправленими ребрами, над якими може проставлятися ступінь зв’язку (1 або буква, що заміняє слово «багато») і необхідне пояснення.

Між двома сутностям, наприклад, А и В можливі чотири види зв’язків.

Перший тип – зв’язок ОДИН-ДО-ОДНОГО (1:1): у кожний момент часу кожному представнику (примірнику) сутності А відповідає 1 або 0 представників сутності У. Наприклад, студент може не «заробити» стипендію, одержати звичайну або одну з підвищених стипендій. Або, допустимо, у визначений момент часу один клієнт може зробити тільки одне замовлення. У цьому випадку між об’єктами «клієнт» і «замовлення» встановлюється взаємозв’язок «один до одного», що позначається одинарними стрілками.

Між даними, що зберігаються в об’єктах «клієнт» і «замовлення», буде існувати взаємозв’язок, у котрому кожний запис в однім об’єкті буде однозначно вказувати на запис в іншому об’єкті. Ні в одному, ні в іншому об’єкті не може існувати запису, не пов’язаного з якийсь записом в іншому об’єкті.

Другий тип – зв’язок ОДИН-ДО-БАГАТЬОХ (1:М): одному представнику сутності А відповідають 0, 1 або декілька представників сутності В. Наприклад, квартира може пустувати, у ній може жити один або декілька мешканців. Або, наприклад, у визначений момент часу один клієнт може стати володарем декількох моделей автомобілів, при цьому декілька клієнтів не можуть бути володарями одного автомобіля. Взаємозв’язок «один до багатьох» можна позначити за допомогою одинарної стрілки в напрямку до «одному» і подвійної стрілки в напрямку до «багатьох». У цьому випадку одного запису даних першого об’єкта (його часто називають батьківським або основним) буде відповідати декілька записів другого об’єкта (дочірнього або підпорядкованого). Взаємозв’язок «один до багатьох» дуже поширений при розробці реляційних баз даних.

Третій тип – зв’язок БАГАТО-ДО-ОДНОГО (М:1): одному представнику сутності В відповідають 0, 1 або декілька представників сутності А.

Зазвичай немає ніякої різниці між зв’язком ОДИН-ДО-БАГАТЬОХ і БАГАТО-ДО-ОДНОГО, тому що між двома сутностями можливі зв’язки в обох напрямках і все залежить від того, з якими сутностями пов’язані дані.

Четвертий тип – зв’язок БАГАТО-ДО-БАГАТЬОХ (N:M): одному представнику сутності В відповідають 0, 1 або декілька представників сутності А и одночасно одному представнику сутності А відповідають 0, 1 або декілька представників сутності В. Це також обумовлено тим, що між двома сутностями можливі зв’язки в обох напрямках.

2.3 Реляційна модель даних та її властивості

Наприкінці 60-х років з'явилися роботи, в яких обговорювалися можливості застосування різних табличних даталогічних моделей даних, тобто можливості використання звичайних і природних засобів подання даних. Найбільш значною з них була стаття співробітника фірми IBM доктора Е. Кодда (Codd EF, A Relational Model of Data for Large Shared Data Banks. CACM 13 : 6, June 1970), де, ймовірно, вперше був застосований термін «реляційна модель даних».

Як математик за освітою, Е. Кодд запропонував використовувати для обробки даних апарат теорії множин. Він показав, що будь-яке представлення даних зводиться до сукупності двовимірних таблиць особливого виду, відомого в математиці як відношення (англ. «relation»).

Найменша одиниця даних реляційної моделі – це окреме атомарне (що не розкладається) для даної моделі значення даних. Так, в одній предметній області прізвище, ім'я та по батькові можуть розглядатися як єдине значення, а в іншій – як три різних значення. Так, в однієї предметній області прізвище, ім'я та по батькові можуть бути розглянути як єдине значення, а в іншій – як три різних значення.

Доменом називається безліч атомарних значень одного і того ж типу. Якщо значення двох атрибутів беруться з одного і того ж домена, то, ймовірно, мають сенс порівняння, які використовують ці два атрибути. Якщо ж значення двох атрибутів беруться з різних доменів, то їх порівняння, ймовірно, позбавлене сенсу.

Відношення на доменах D1, D2, ..., Dn (не обов'язково, щоб всі вони були різні) складається із заголовка і тіла.

Заголовок складається з такої фіксованої множини атрибутів A1, A2, ..., An, що існує взаємно однозначна відповідність між цими атрибутами Ai і доменами Di (i = 1,2, ... , n), що їх визначають.

Тіло складається зі змінюваної в часі множини кортежів, де кожен кортеж складається в свою чергу з безлічі пар атрибут-значення (Ai: Vi), (i = 1,2, ..., n), по одній такій парі для кожного атрибута Ai в заголовку. Для будь-якої заданої пари атрибут-значення (Ai: Vi) Vi є значенням з єдиного домену Di, який пов'язаний з атрибутом Ai.

Ступінь відношення – це число її атрибутів. Відношення ступеня один називають унарним, ступеня два – бінарним, ступеня три – тринарним, ..., ступеня n – n-арним. Кардинальне число або потужність відношення – це число

його кортежів. Кардинальне число відношення змінюється в часі на відміну від його ступеня.

Оскільки відношення – це множина, а множини за визначенням не містить співпадаючих елементів, то ніякі два кортежі відношення не можуть бути дублікатами один одного в будь-який довільно-заданий момент часу.

Нехай R – відношення з атрибутами A1, A2, ..., An. Кажуть, що множина атрибутів K = (Ai, Aj, ..., Ak) відношення R є можливим ключем R тоді і тільки тоді, коли задовольняються дві незалежні від часу умови.

1. Унікальність: у довільний заданий момент часу ніякі два різних кортежа R не мають одного і того ж значення для Ai, Aj, ..., Ak.

2. Мінімальність: жоден з атрибутів Ai, Aj, ..., Ak не може бути виключений з K без порушення унікальності.

Кожне відношення володіє хоча б одним можливим ключем, оскільки комбінація всіх його атрибутів задовольняє умові унікальності. Один з можливих ключей, якій обраний довільним образом, приймається за його первинний ключ. Інші можливі ключі, якщо вони є, мають назву альтернативні ключі.

Вищезазначені та деякі інші математичні поняття явилися теоретичною базою для створення реляційних СУБД, розробки відповідних мовних засобів і програмних систем, що забезпечують їх високу продуктивність, і створення основ теорії проєктування баз даних. Однак для масового користувача реляційних СУБД можна з успіхом використовувати неформальні еквіваленти цих понять:

Відношення – Таблиця (іноді Файл),

Кортеж – Рядок (Запис),

Атрибут – Стовпець (Поле).

При цьому приймається, що «запис» означає «екземпляр запису», а «поле» означає «ім’я» і «тип поля».

Реляційна база даних – це сукупність відношень, що містять всю інформацію, яка повинна зберігатися в базі даних. Однак користувачі можуть сприймати таку базу даних як сукупність таблиць. Існують наступні правила.

1. Кожна таблиця складається з однотипних рядків (записів) і має унікальне ім’я.

2. Рядки (записи) мають фіксовану кількість полів (стовпців) і значень (множинні поля і повторювані групи неприпустимі). Інакше кажучи, в кожній позиції таблиці на перетині рядка і стовпця завжди є в точності одне значення або нічого.

3. Рядки таблиці обов’язково відрізняються один від одного хоча б єдиним значенням, що дозволяє однозначно ідентифікувати будь-який рядок такої таблиці.

4. Стовпцям таблиці присвоюються імена і в кожному з них розміщуються однорідні значення даних (дати, прізвища, цілі числа або грошові суми).

5. Повний інформаційний зміст бази даних представляється у вигляді явних значень даних і такий метод подання є єдиним. Зокрема, не існує будь-яких спеціальних «зв’язків» або покажчиків, що з’єднують одну таблицю з іншою.

6. При виконанні операцій з таблицею її рядки і стовпці можна обробляти в будь-якому порядку безвідносно до їх інформаційного змісту. Цьому сприяє наявність імен таблиць і їх стовпців, а також можливість виділення будь-якого їх рядка або будь-якого набору рядків із зазначеними ознаками.

ТЕМА 3 ТЕОРІЯ НОРМАЛІЗАЦІЇ РЕЛЯЦІЙНОЇ МОДЕЛІ ДАНИХ

3.1 Правила нормалізації баз даних

Нормалізація – це розбиття таблиці на дві або більше, що володіють кращими властивостями при включенні, зміні і видаленні даних. Остаточна мета нормалізації зводиться до отримання такого проєкту бази даних, в якому кожен факт з’являється лише в одному місці, тобто виключена надмірність інформації. Це робиться не стільки з метою економії пам’яті, скільки для виключення можливої суперечливості збережених даних.

Як вказувалося раніше, кожна таблиця в реляційній БД задовольняє умові, відповідно до якої в позиції на перетині кожного рядка і стовпчика таблиці завжди знаходиться єдине значення, і ніколи не може бути безліч таких значень. Будь-яка таблиця, яка задовольняє цій умові, називається нормалізованою. Фактично, ненормалізовані таблиці, тобто таблиці, що містять повторювані групи, навіть не допускаються в реляційної БД.

Всяка нормалізована таблиця автоматично вважається таблицею в першій нормальній формі, скорочено 1НФ. Таким чином, строго кажучи, «нормалізована» і «така, що знаходиться в 1НФ» означають одне і те ж. Однак на практиці термін «нормалізована» часто використовується в більш вузькому сенсі – «повністю нормалізована», який означає, що в проєкті не порушуються ніякі принципи нормалізації.

Тепер, окрім 1НФ, можна визначити подальші рівні нормалізації – другу нормальну форму (2НФ), третю нормальну форму (3НФ) і т. д. По суті, таблиця міститься в 2НФ, якщо вона міститься в 1НФ і задовольняє, крім того, деяку додаткову умову, суть якої буде розглянуто нижче. Таблиця міститься в 3НФ, якщо вона міститься в 2НФ і, крім цього, задовольняє ще іншу додаткову умову і т. д.

Таким чином, кожна нормальна форма є в певному сенсі більш обмеженою, але й більш бажаною, ніж попередня. Це пов'язано з тим, що (N + 1)-а нормальна форма не володіє деякими неприваблівими особливостями, які властиві N-й нормальній формі. Загальний сенс додаткової умови, що накладається на (N + 1)-у нормальну форму стосовно N-ї нормальної форми, полягає у виключенні цих неприваблівих особливостей.

Теорія нормалізації ґрунтується на наявності тієї чи іншої залежності між полями таблиці. Визначено два види таких залежностей: функціональні і багатозначні.

Функціональна залежність. Поле B таблиці функціонально залежить від поля A тієї ж таблиці в тому і тільки в тому випадку, коли в будь-який заданий момент часу для кожного з різних значень поля A обов'язково існує тільки одне з різних значень поля B. Зазначимо, що тут допускається, що поля A та B можуть бути складовими.

Повна функціональна залежність. Поле B знаходиться в повній функціональної залежності від складеного поля A, якщо воно функціонально залежить від A і не залежить функціонально від будь-якої підмножини поля A.

Багатозначна залежність. Поле A багатозначно визначає поле B тій самій таблиці, якщо для кожного значення поля A існує певна безліч відповідних значень B.

Нормальні форми. Вище було надано визначення першої нормальної форми (1НФ). Наведемо тут більш чітке її визначення, а також визначення інших нормальних форм.

Таблиця міститься в першій нормальній формі (1НФ) тоді і тільки тоді, коли жоден із її рядків не містить у будь-якому своєму полі більше одного значення і жодне з її ключових полів не порожнє.

Таблиця міститься в другій нормальній формі (2НФ), якщо вона задовольняє визначення 1НФ, і всі її поля, що не входять у первинний ключ, пов'язані повною функціональною залежністю з первинним ключем.

Для спрощення нормалізації таблиць доцільно використовувати таку рекомендацію. Під час проведення нормалізації таблиць, у які введені цифрові

або інші замінники складових і (або) текстових первинних і зовнішніх ключів, варто хоча б подумки підміняти їх вихідними ключами, а після закінчення нормалізації знову відновлювати.

Таблиця міститься в третій нормальній формі (3НФ), якщо вона задовольняє визначення 2НФ і ні одне з її неключових полів не залежить функціонально від будь-якого іншого неключового поля.

У зв'язку з недоречностями, які можуть виникати не тільки через введення кодованих первинних ключів, теоретики реляційних систем Кодд і Бойс обґрунтували і запропонували більш чітке визначення 3НФ, яке враховує, що в таблиці може бути декілька можливих ключів.

Таблиця міститься в нормальній формі Бойса – Кодда (далі – НФБК), якщо і тільки якщо будь-яка функціональна залежність між його полями зводиться до повної функціональної залежності від можливого ключа.

У наступних нормальних формах (4НФ і 5НФ) враховуються не тільки функціональні, але і багатозначні залежності між полями таблиці. Для їх опису познайомимося з поняттям повної декомпозиції таблиці.

Повною декомпозицією таблиці називають таку сукупність довільної кількості її проєкцій, поєднання яких повністю співпадає із вмістом таблиці.

Таблиця міститься в п'ятій нормальній формі (5НФ) тоді і тільки тоді, коли в кожній її повної декомпозиції всі проєкції містять можливий ключ. Таблиця, яка не має жодної повної декомпозиції, також міститься в 5НФ.

Четверта нормальна форма (4НФ) є окремим випадком 5НФ, коли повна декомпозиція повинна бути з'єднанням рівно двох проекцій. Дуже не просто підібрати реальну таблицю, що може бути надана в 4НФ, але не була б в 5НФ.

Процедура нормалізації. Як уже зазначалося, нормалізація – це розбиття таблиці на декілька частин, що володіють кращими властивостями при оновленні, включенні і видаленні даних. Тепер можна подати й інше визначення: нормалізація – це процес послідовної заміни таблиці її повними декомпозиціями до тих пір, поки всі вони не будуть знаходитися в 5НФ. На практиці ж досить привести таблиці до НФБК і з великою гарантією вважати, що вони перебувають в 5НФ. Зрозуміло, цей факт потребує перевірки, проте поки не існує ефективного алгоритму такої перевірки. Тому зупинимося лише на процедурі приведення таблиць до НФБК.

Ця процедура ґрунтується на тому, що єдиними функціональними залежностями в будь-якій таблиці повинні бути залежності виду K→F, де K – первинний ключ, а F – деяке інше поле. Зауважимо, що це випливає з визначення первинного ключа таблиці, відповідно до якого K→F завжди має місце для всіх

полів цієї таблиці. «Один факт в одному місці» говорить про те, що не мають сили ніякі інші функціональні залежності. Мета нормалізації полягає саме в тому, щоб позбутися від усіх цих «інших» функціональних залежностей, тобто таких, які мають інший вигляд, ніж K→F.

Якщо скористатися рекомендацією, наведеною вище, і підмінити на час нормалізації коди первинних (зовнішніх) ключів на вихідні ключі, то, по суті, слід розглянути лише два випадки:

1. Таблиця має складовою первинний ключ виду, скажімо, (К1, К2), і включає також поле F, яке функціонально залежить від частини цього ключа, наприклад, від К2, тільки не від повного ключа. В цьому випадку рекомендується сформувати іншу таблицю, яка містить К2 і F (первинний ключ – К2), і видалити F з початкової таблиці: замінити Т (К1, К2, F), первинний ключ (К1, К2), ФЗ К2→F на Т1 (К1, К2), первинний ключ (К1, К2), і Т2 (К2, F), первинний ключ К2.

2. Таблиця має первинний (можливий) ключ К, поле F1, яке не є можливим ключом, що, звичайно, функціонально залежить від К, і інше неключове поле F2, яке функціонально залежить від F1. Рішення те ж саме, що і раніше – формується інша таблиця, яка містить F1 і F2, з первинним ключем F1, і F2 видаляється з початкової таблиці: замінити Т (K, F1, F2), первинний ключ К, ФЗ F1→F2 на Т1 (K, F1), первинний ключ К, і Т2 (F1, F2), первинний ключ F1.

Для будь-якої заданої таблиці, повторюючи застосування двох розглянутих правил, майже у всіх практичних ситуаціях можна отримати в кінцевому рахунку безліч таблиць, які знаходяться в «остаточній» нормальній формі і, таким чином, не містять будь-яких функціональних залежностей виду, відмінного від K→F.

Для виконання цих операцій необхідно спочатку мати в якості вхідних даних будь-які «великі» таблиці (наприклад, універсальні відношення). Але нормалізація нічого не говорить про те, як отримати ці великі таблиці.

3.2 Теоретико-множинні та спеціальні операції реляційної алгебри

Реляційна алгебра – це множина операцій, що виконуються над відношеннями і мають за мету утворення нових відношень або їх станів. Реляційна алгебра визначає операції, які однаковим чином реалізуються в усіх базах даних реляційного типу, незалежно від їх змісту і технологій, за допомогою яких вони реалізовані. Тобто реляційна алгебра представляє собою процедурну мову обробки реляційних таблиць.

Об'єднанням двох відношень називається відношення, що містить безліч кортежів, що належать або першому, або другому вихідним відношенням, або обом відношенням одночасно.

Припустимо, задані два відношення $R_1 = \{r_1\}$, $R_2 = \{r_2\}$, де r_1 і r_2 – відповідно кортежі відношень R_1 і R_2, тоді поєднанням є

$$R_1 \cup R_2 = \{r \mid r \in R_1 \vee r \in R_2\},$$

де r – кортеж нового відношенням; ∨ – операція логічного складання «або».

Об'єднання дозволяє комбінувати дані з двох таблиць з однаковими множинами атрибутів. Однакові множини атрибутів потрібні для того, щоб результатом виконання операції об'єднання була реляційна таблиця.

Перетином відношень називається відношення, яке містить безліч кортежів, що належать одночасно і першому і другому відношенням R_1 і R_2:

$$R_3 = R_1 \cap R_2 = \{ r \mid r \in R1 \wedge r \in R_2 \},$$

де ∧ – операція логічного множення (логічне «і»).

Операція перетину дозволяє ідентифікувати рядки, спільні для двох таблиць.

Різницею відношень R_1 і R_2 називається відношення, що містить безліч кортежів, які належать R_1 і не належать R_2:

$$R_5 = R_1 \setminus R_2 = \{r \mid r \in R_1 \wedge r \notin R_2\}.$$

Операція різниці дозволяє ідентифікувати ті рядки, які є в одній таблиці, але відсутні в іншій.

Крім трьох перелічених операцій у рамках реляційної алгебри визначена ще одна теоретико-множинна операція – *розширений декартовий добуток*. Ця операція не накладає ніяких додаткових умов на схеми вихідних відношень, тому операція розширеного декартового добутку є допустимою для будь-яких двох відношень. Але перш ніж визначити саму операцію введемо додатково поняття конкатенації, або зчеплення, кортежів.

Зчепленням або *конкатенацією* кортежів $c = <c_1, c_2, ..., c_n>$ і $q = <q_1, q_2, ..., q_m>$ називається кортеж, отриманий шляхом додавання значень другого до кінця першого. Зчеплення кортежів c і q позначається, як (c , q):

$$(c, q) = <c_1, c2, ... , c_n, q_1, q_2, ..., q_m>,$$

де n – кількість елементів в першому кортежі c; m – кількість елементів у другому кортежі q.

Усі попередні операції не змінювали ступеня або арності відношень – це випливає з визначення еквівалентності схем відношень. Операція декартового добутку змінює ступінь результуючого відношення.

Операцію декартового добутку, з урахуванням можливості перестановки атрибутів у відношенні, можна вважати симетричною. Зазвичай операція розширеного декартового добутку використовується для отримання деякого універсуму, тобто відношення, яке характеризує всі можливі комбінації між елементами окремих множин. Однак самостійного значення результат виконання операції зазвичай не має, він бере участь в подальшій обробці.

Першою *спеціальною операцією реляційної алгебри* є горизонтальний вибір, або операція фільтрації, або операція обмеження відношень. Для визначення цієї операції необхідно ввести додаткові позначення.

Припустимо, що а – булевський вираз, складений з термів порівняння за допомогою зв'язків І (), АБО (), НЕ () і, можливо, дужок. Як терми порівняння допускаються:

терм А оп а,

де А – ім'я деякого атрибута, що набуває значення з домену D; а – константа, взята з того самого домена D; а D; оп – одна з допустимих для цього домена D операцій порівняння;

терм А оп В,

де А, В – імена деяких θ-порівнянних атрибутів, тобто атрибутів, які набувають значення з одного і того самого домену D.

Тоді результатом операції вибору, або фільтрації, заданої на відношенні R у вигляді булевського виразу, визначеного на атрибутах відношення R, називається відношення R [], що включає ті кортежі з вихідного відношення, для яких істинна умова вибору або фільтрації:

R [(r)] = {r | r R (r) = "Істина"}

Операція фільтрації є однією з основних під час роботи з реляційною моделлю даних.

Наступною спеціальної операцією є операція *проєктування*.

Нехай R – відношення, SR = (A1, ..., An) – схема відношення R.

Позначимо через В підмножина [Ai]; В {Ai}. При цьому нехай B^1 – безліч атрибутів з {Ai}, що не увійшли до В.

Якщо В = $\{A^1_i, A_i^2,..., A_i^k\}$, В = $\{A^1, A^2_j, ..., A^k_j\}$ та r = < a^1_i, a^2_i,...,a_i >, $a^k_i \in A^k_{ii}$, то r [B], s = < a^1_j, a^2_j, ... , a^m_j > ; $a^m_j \in A^m_j$.

Проєкцією відношень R на набір атрибутів В (позначається R [B]) називається відношення зі схемою, яка відповідає набору атрибутів В SR [B] = В, який містить кортежі, одержувані з кортежів вихідного відношення R шляхом видалення з них значень, які не належать атрибутам з набору В.

R[B] = { r[B] }

За визначенням відношень усі дублюючі кортежі видаляються з результуючого відношення.

Операція проєктування, яку називають також операцією вертикального вибору, дозволяє отримати тільки необхідні характеристики об'єкта, що моделюється. Найчастіше операція проєктування вживається як проміжний крок в операціях горизонтального вибору, або фільтрації. Крім того, вона використовується самостійно на заключному етапі отримання відповіді на запит.

Наступною спеціальної операцією реляційної алгебри є *операція умовного з'єднання*. На відміну від розглянутих спеціальних операцій реляційної алгебри: фільтрації і проєктування, які є унарними, тобто виробляються над одним відношенням, операція умовного з'єднання є бінарною, тобто вихідними для неї є два відношення, а результатом – одне.

Припустимо, R = {r}, Q = {q} – вихідні відношення. SR, SQ – схеми відношень R і Q відповідно.

$S_R = (A_1, A_2, \dots, A_k)$; $S_Q = (B_1, B_2, \dots, B_m)$, де A_i, B_j – імена атрибутів в схемах відношень R і Q відповідно. При цьому вважаємо, що задані набори атрибутів A і B:

$$A \subseteq \{A\}\ i = 1, k;\ B \subseteq \{B_j\}\ j = 1, m$$

і ці набори складаються з θ-порівнянних атрибутів.

Тоді з'єднанням відношень R і Q за умови β буде підмножина декартовий добуток відношень R і Q, кортежі якого задовольняють умові β, оскільки він розглядався як одночасне виконання умов:

– r.Ai θi Bi: i = 1, k, де k – число атрибутів, що входять в набори A і B, а θi – конкретна операція порівняння;

– Ai θi Bi D; θi – i-й предикат порівняння, який визначається з безлічі допустимих на домені Di операцій порівняння.

$$R\ [\ \beta]\ Q = \{\ (r,q)\ |\ (r, q)\ |\ r.A\ \theta_i\ q.B_i = \text{"Истина"},\ i=1,k\}$$

Останньою операцією, що включається в набір операцій реляційної алгебри, є *операція ділення*. Для визначення операції ділення розглянемо спочатку поняття безлічі образів.

Припустимо, R – відношення зі схемою SR = (A1, A2 ,…, Ak); A – деякий набір атрибутів A {Ai} i = 1, k, A1 – набір атрибутів, що не входять у множину A.

Перетин множин A і A^1 є порожнім: $A \cap A^1 = 0$; об'єднання множин є рівному множині всіх атрибутів вихідного відношення $A \cup A^1 = S_R$.

Тоді множиною образів елемента в проекції R [A] називається множина таких елементів у проекції R [A], для яких зчеплення (x, y) є кортежами відношень R, тобто $QA(x) = \{y \mid y \in R[A] \wedge (x, y) \in R\}$ – множина образів.

Дамо тепер визначення операції ділення. Нехай є два відношення R і T відповідно зі схемами: $S_R = (A_1, A_2, \ldots, A_k)$; $S_T = (B_1, B_2, \ldots, B_m)$; A і B – набори атрибутів цих відношень, однакової довжини (без повторень); $A \subseteq S_R$; $B \subseteq S_T$. Атрибути A^1 – це атрибути з R, що не увійшли в множину A.

Перетин множин $A \cap A^1 = \varnothing$ – є порожнім і $A \cup A^1 = S$. Проєкції R [A] і T [B] сумісні по об'єднанню, тобто мають еквівалентні схеми: $S_{R[A]} \sim S_{T[B]}$.

Тоді операція ділення ставить у відповідність відношенням R і T відношення Q = R [A: B] T, кортежі якого є тими елементами проекції $R[A^1]$, для яких T [B] входить в побудовані для них множини образів:

$$R[A:B]T = \{r \mid r \in R[A^1] \wedge T[B] \subseteq \{y \mid y \in R[A] \wedge (r, y) \in R\}\}.$$

Операція ділення зручна тоді, коли потрібно порівняти деяку множину характеристик окремих атрибутів.

ЗМІСТОВИЙ МОДУЛЬ 2 МОВИ ЗАПИТІВ

ТЕМА 4 МОВА QBE

4.1 Загальна характеристика мови запитів QBE

Сучасні бази даних зберігають великі об'єми інформації, тому обробляти її вручну, послідовно переглядаючи і редагуючи дані в таблицях, стає досить важко. Для підвищення ефективності користування базами даних застосовують запити, які дозволяють виконувати множинну обробку даних, тобто одночасно вводити, редагувати та видаляти велику кількість записів, а також вибирати дані з таблиць.

Запит – це засіб отримання інформації з бази даних.

Запит являє собою спеціальним чином описану вимогу, яка визначає склад операцій, що виконуються над базою даних. До таких операцій відносять операції по вибірці, видаленню або модифікації даних.

Для підготовки запитів з допомогою різних СУБД найчастіше використовують дві основні мови опису запитів:

- мова QBE (Query By Example) – мова запитів за зразком;
- мова SQL (Structured Query Language) – структурована мова запитів.

Мова QBE була розроблена Моше Злуфом, співробітником дослідницького центру компанії IBM, в 1970-х роках і призначалася для користувачів, зацікавлених у вибірці інформації з баз даних. Ця мова отримала в користувачів настільки широке визнання, що в даний час в тій чи іншій мірі вона реалізована практично у всіх популярних СУБД, включаючи і Microsoft Access.

Засоби підтримки мови QBE в СУБД Microsoft Access досить прості в експлуатації і в той же час представляють користувачам досить широкий спектр можливостей роботи з даними. Засоби мови QBE можуть використовуватися для введення запитів до інформації, що зберігається в одній або кількох таблицях, а так само для визначення набору полів, які повинні бути присутніми в результуючій таблиці. СУБД Microsoft Access при створенні запиту з використанням засобів QBE неявно формує еквівалентний оператор мови SQL, призначений для виконання зазначених дій.

В мові QBE використовується візуальний підхід до організації доступу до інформації в базі даних, який передбачає заповнення форми запиту, яку надає СУБД. Такий спосіб задання запиту забезпечує високий ступінь наочності і не потребує вказання алгоритму виконання операцій, адже достатньо лише описати зразок очікуваного результату. В кожній з сучасних СУБД є свій варіант мови QBE.

На мові QBE можна створювати запити однотабличні та багатотабличні (вибирають і обробляють дані з декількох зв’язаних таблиць).

За допомогою запитів мовою QBE можна виконати такі основні операції над даними бази даних:

– вибірку даних;
– обчислення над даними;
– додавання нових записів;
– видалення записів;
– модифікацію (зміну) даних.

У випадку перших двох операцій результатом виконання запиту є таблиця, яку називають таблиця-відповідь, а у випадку інших операцій – оновлена вихідна таблиця.

Вибірка, вставка, видалення та модифікація можуть виконуватись безумовно або відповідно до певних умов, які задаються за допомогою логічних виразів. Обчислення над даними задаються за допомогою арифметичних виразів і породжують в таблицях-відповідях нові поля, які називають обчислюваними.

Форма запиту має вигляд таблиці, ім’я і назви полів якої співпадають з іменем і назвами полів відповідної вихідної таблиці. Щоб дізнатись імена

доступних таблиць БД, в мові QBE передбачений запит на вибірку імен таблиць. Назви полів вихідної таблиці можуть вводитись в шаблон вручну або автоматично. В другому випадку виконується запит на вибірку заголовків стовпців.

4.2 Опис запитів мовою QBE

Запит QBE являє собою одну або декілька таблиць, рядки яких містять вимоги до значень рядків таблиці, одержуваної в результаті виконання запиту.

Відповідно до синтаксису запиту назву таблиці (відношення) поміщають в заголовок першого стовпця (сам стовпець залишається порожнім завжди, крім випадку короткого запису включення до результату всіх стовпців таблиці). Для кожного атрибута, що бере участь у запиті, в таблицю включається свій стовпець, у заголовку якого міститься ім'я атрибута. Наприклад, шаблон запиту для таблиці STUDENT, атрибутами якої є ПІБ, Група та Курс студента, матиме такий вигляд:

STUDENT	ПІБ	Група	Курс

Значення атрибутів (зразок, змінна або результат запиту) поміщаються у відповідні стовпці.

Для опису змінних в умовах відбору записів, а також для зв'язування шаблонів в запитах використовуються змінні – елементи прикладу. Елемент прикладу грає роль ідентифікатора змінної (як у мові програмування) і задається за допомогою символьно-цифрової послідовності. Довжина та склад елементу прикладу можуть бути довільними: головне, щоб при використанні у різних місцях шаблону він мав однаковий вигляд.

Вибірка даних з умовою. Вибір записів з умовою в загальному випадку може бути заснований на точному співпадінні, частковому співпадінні або порівнянні:

1. Точне співпадіння задається введенням констант у відповідних полях шаблону.

2. Часткове співпадіння задається за допомогою елементів прикладу.

3. Умова порівняння записується за допомогою операцій порівняння.

Використання результуючих функцій. При запису логічних виразів на мові QBE можуть застосовуватись результуючі функції: CNT (лічильники кількості), SUM (сума), AVG (середнє), MIN (мінімум), MAX (максимум), UN (унікальний) та ALL (всі значення, в тому числі й ті, що повторюються). Перші п'ять із них є

статистичними, а останні дві визначають характер вибірки: включати чи не включати у вибірку повторювані значення.

Обчислення в запитах. За допомогою запитів мовою QBE можна вибирати дані з таблиць і проводити обчислення. Вид обчислень задається за допомогою виразу в шаблоні. У виразах, окрім звичних арифметичних операцій та дужок, можуть використовуватися вбудовані функції: AVG, CNT, MAX, MIN, SUM.

Операції додавання, видалення та модифікації призводять до зміни вихідної таблиці. Вид операції записується в шаблоні під ім'ям таблиці, а константи і умовні вирази вказуються за тими ж правилами, що в операціях вибірки.

За допомогою запиту в СУБД Microsoft Access можна:

- вибрати записи, які задовольняють умові відбору;
- включити в результуючу таблицю задані користувачем поля;
- провести обчислення для кожного запису;
- згрупувати записи з однаковими значеннями в один запис з одночасним виконанням над ними групових функцій;
- провести оновлення, додавання або видалення записів в таблицях;
- створити нову таблицю, використовуючи дані з існуючих таблиць.

Побудова більшості запитів реалізуються за допомогою «Майстрів», що спрощують розробку запиту. Якщо створений запит не задовольняє вимогам, то можна скористатися «Конструктором», або створити заготовку запиту за допомогою «Майстра», яку потім підправити в режимі «Конструктора». Рекомендується створювати запити в режимі «Конструктора». В цьому випадку при створенні запиту буде використана мова QBE, при необхідності цей запит можна вдосконалити на мові SQL. Access автоматично генерує код SQL при створенні запиту на QBE і навпаки.

Складання запиту в режимі «Конструктора» включає визначення:

- таблиць і полів таблиць;
- виду запиту (вибірка, додавання, видалення, перехресний запит, SQL-запит);
- умов відбору записів;
- параметрів відображення результатів виконання запиту (показ полів, сортування значень).

Всі ці дії виконуються в запитальній формі, яка відноситься до форми запиту на мові QBE. Запитальна форма включає три основні елементи: заголовок (ім'я й тип запиту); область таблиць, їх полів і зв'язків між таблицями; бланк запиту за зразком.

Для вказівки таблиць, які використовуються в запиті, потрібно помістити їх в запитальну форму і вказати зв'язки між ними. Включення об'єктів в запит проводиться у вікні «Добавление таблицы», яке викликається автоматично (при створенні запиту) або примусово при роботі із запитом шляхом натиснення кнопки панелі інструментів.

При створенні запиту за умовчанням надається заготовка запиту на вибірку. Змінити вид запиту можна за допомогою пункту «Запрос» головного меню, де можливі види запитів, перераховані як підпункти меню.

Умови відбору записів задаються в бланку у вигляді виразу як:

− літерали – конкретні значення, які сприймає Access так, як вони записані. Текстові рядки беруться в «лапки»;

− константи – незмінні значення: True, False, Null;

− ідентифікатор – посилання на значення поля, укладається в квадратні дужки: [ФІО];

− оператори порівняння і логічні оператори =, <, >, <>, <=, >=, Between 10 And 100 – задає інтервал для числового значення або дати, In («Математика»; «Фізика») виконує перевірку на рівність будь-якому значенню із списку, Like «*Математика» – дозволяє використовувати зразки, що використовують символи шаблону, при пошуку в текстових полях, Or, Not.

Групові операції у запитах дозволяють виділити в таблиці групи записів з однаковими значеннями у вказаних полях і використовувати для цих груп статистичну функцію:

Sum – сума значень поля для групи;

Avg – середнє від всіх значень поля в групі;

Max, Min – максимальне і мінімальне значення поля в групі;

Count – число значень поля в групі без урахування порожніх значень.

Для створення запиту з використанням групових операцій необхідно при відкритому в режимі «Конструктора» запиті вибрати пункт меню «Групповые операции» або натиснути однойменну кнопку на панелі інструментів.

У запиті над полями можуть проводитися обчислення. Результат утворює обчислюване поле в таблиці запиту. При кожному запуску запиту проводяться обчислення з поточними значеннями полів.

Вираз вводиться в бланк запиту в порожнє поле рядка **«Поле».** Після натиснення **«Enter»** перед введеним виразом додається ім'я поля. Наприклад, Загальна вартість: [Кількість]*[ЦінаОд].

В Access визначено близько 150 вбудованих функцій, які можна використовувати в обчисленнях і умовах відбору. За призначенням їх можна згрупувати:

- «Функции даты и времени» – використовуються при обробці дат і часу. Повертають дату і час повністю або частково. Наприклад, функція Date формує поточну дату. Month – виділяє місяць із значення поля, що містить дату;
- «Функции обработки текста» – використовуються при роботі з символьними рядками;
- «Функции преобразования типа данных» – дозволяють задавати тип даних, щоб уникнути автоматичного підбору даних системою;
- «Математические и тригонометрические функции» – виконують операції над числами, які неможливо виконати за допомогою стандартних арифметичних операторів;
- «Финансовые функции» – служать для розрахунку відсотка повернення по інвестиціях, амортизаційних відрахувань і т. п.;
- «Статистические функции» – використовуються для обчислення середнього, мінімального і максимального значень і т. д.

ТЕМА 5 МОВА SQL

5.1 Мова запитів SQL як універсальний інструмент доступу до бази даних

SQL (Structured Query Language) – Структурована Мова Запитів – стандартна мова запитів по роботі з реляційними БД. Мова SQL з'явилася після реляційної алгебри, і її прототип був розроблений в кінці 70-х років в компанії IBM Research. Він був реалізований в першому прототипі реляційної СУБД фірми IBM System R. Надалі ця мова застосовувалася в багатьох комерційних СУБД і в силу свого широкого поширення поступово став стандартом «де-факто» для мов маніпулювання даними в реляційних СУБД.

Перший міжнародний стандарт мови SQL був прийнятий в 1989 р (далі ми будемо називати його SQL / 89 або SQL1). Іноді стандарт SQL1 також називають стандартом ANSI / ISO, і переважна більшість доступних на ринку СУБД підтримують цей стандарт повністю. Однак розвиток інформаційних технологій, пов'язаних з базами даних, і необхідність реалізації додатків зажадали незабаром доопрацювання і розширення першого стандарту SQL.

В кінці 1992 був прийнятий новий міжнародний стандарт мови SQL, який в подальшому будемо називати SQL / 92 або SQL2. І він не позбавлений

недоліків, але в той же час є істотно більш точним і повним, ніж SQL / 89. На даний момент більшість виробників СУБД внесли зміни в свої продукти так, щоб вони в більшій мірі задовольняли стандарту SQL2.

У 1999 році з'явився новий стандарт, названий SQL3. Якщо відмінності між стандартами SQL1 і SQL2 багато в чому були кількісними, то стандарт SQL3 відповідає якісним серйозним перетворенням. У SQL3 введені нові типи даних, при цьому передбачається можливість завдання складних структурованих типів даних, які в більшій мірі відповідають об'єктній орієнтації. Нарешті, додано розділ, який вводить стандарти на події і тригери, які раніше не порушувалися в стандартах, хоча давно вже широко використовувалися в комерційних СУБД. У стандарті визначені можливості чіткої специфікації тригерів як сукупності події і дії. Як дії можуть виступати не тільки послідовність операторів SQL, а й оператори управління ходом виконання програми. В рамках управління транзакціями відбулося повернення до старої моделі транзакцій, що допускає точки збереження (savepoints), і можливість вказівки в операторі відкату ROOLBACK точок повернення дозволить відкочувати транзакцію не в початок, а в проміжну раніше збережену точку. Таке рішення підвищує гнучкість реалізації складних алгоритмів обробки інформації.

Для постачальників СУБД стандарт – це дороговказ, що гарантує правильний напрямок робіт. А ось ефективність реалізації стандарту – це гарантія успіху.

SQL можна в повній мірі віднести до традиційних мов програмування, він не містить традиційні оператори, що керують ходом виконання програми, оператори опису типів і багато іншого, він містить тільки набір стандартних операторів доступу до даних, що зберігаються в базі даних. Оператори SQL вбудовуються в базову мову програмування, якою може бути будь-яка стандартна мова типу C ++, PL, COBOL і т. д. Крім того, оператори SQL можуть виконуватися безпосередньо в інтерактивному режимі.

На відміну від реляційної алгебри, де були представлені тільки операції запитів до БД, SQL є повною мовою, в ній присутні не тільки операції запитів, але і оператори, відповідні Data Definition Language (DDL) – мови опису даних. Крім того, мова містить оператори, призначені для управління (адміністрування) БД.

Оператори визначення даних DDL:

– CREATE TABLE – Створити таблицю – Створює нову таблицю в БД;
– DROP TABLE – Видалити таблицю – Видаляє таблицю з БД;

– ALTER TABLE – Змінити таблицю – Змінює структуру існуючої таблиці або обмеження цілісності, що задаються для даної таблиці;

– CREATE VIEW – Створити уявлення – Створює віртуальну таблицю, відповідну деякого SQL-запиту;

– ALTER VIEW – Змінити уявлення – Змінює раніше створене подання;

– DROP VIEW – Видалити уявлення – Видаляє раніше створене подання;

– CREATE INDEX – Створити індекс – Створює індекс для деякої таблиці для забезпечення швидкого доступу по атрибутам, що входять в індекс;

– DROP INDEX – Видалити індекс – Видаляє раніше створений індекс.

Оператори маніпулювання даними Data Manipulation Lanquaqe (DML):

– DELETE – Видалити рядки – Видаляє одну або кілька рядків, відповідних умовам фільтрації, з базової таблиці. Застосування оператора узгоджується з принципами підтримки цілісності, тому цей оператор не завжди може бути виконаний коректно, навіть якщо синтаксично він записаний правильно;

– INSERT – Вставити рядок – Вставляє один рядок у базову таблицю. Допустимі модифікації оператора, при яких відразу кілька рядків можуть бути перенесені з однієї таблиці або запиту в базову таблицю;

– UPDATE – Оновити рядок – Обновляє значення одного або декількох стовпців в одному або декількох рядках, відповідних умовам фільтрації.

Мова запитів Data Query Lanquaqe (DQL):

– SELECT – Вибрати рядки – Оператор, який замінює всі оператори реляційної алгебри і дозволяє сформувати результуюче відношення, відповідне запитом.

– Засоби управління транзакціями:

– COMMIT – Завершити транзакцію – Завершити комплексну взаємозв’язану обробку інформації, об’єднану в транзакцію;

– ROLLBACK – Відкинути транзакцію – Скасувати зміни, проведені в ході виконання транзакції;

– SAVEPOINT – Зберегти проміжну крапку виконання транзакції – Зберегти проміжний стан БД, позначити його для того, щоб можна було в подальшому до нього повернутися.

Засоби адміністрування даних:

– ALTER DATABASE – Змінити БД – Змінити набір основних об’єктів в базі даних, обмежень, що стосуються всієї бази даних;

– ALTER DBAREA – Змінити область зберігання БД – Змінити раніше створену область зберігання;

– ALTER PASSWORD – Змінити пароль – Змінити пароль для всієї бази даних;

– CREATE DATABASE – Створити БД – Створити нову базу даних, визначивши основні параметри для неї;

– CREATE DBAREA – Створити область зберігання – Створити нову область зберігання і зробити її доступною для розміщення даних;

– DROP DATABASE – Видалити БД – Видалити існуючу базу даних (тільки в тому випадку, коли ви маєте право виконати цю дію) ;

– DROP DBAREA – Видалити область зберігання БД – Видалити існуючу область зберігання (якщо в ній на зараз не розташовуються активні дані) ;

– GRANT – Надати права – Надати права доступу на ряд дій над деякими об'єктом БД;

– REVOKE – Позбавити прав – Позбавити прав доступу до деякого об'єкту або деяких дій над об'єктом.

Програмний SQL:

– DECLARE – Визначає курсор для запиту – Задає деяке ім'я і визначає пов'язаний з ним запит до БД, який відповідає віртуальному набору даних;

– OPEN – Відкрити курсор – Формує віртуальний набір даних, відповідає опису зазначеного курсору і поточному стану БД;

– FETCH – Вважати рядок з безлічі рядків, визначених курсором – Зчитує черговий рядок, задану параметром команди з віртуального набору даних, відповідає відкритому курсору;

– CLOSE – Закрити курсор – Припиняє доступ до віртуального набору даних, відповідному зазначеним курсору;

– PREPARE – Підготувати оператор SQL до динамічного виконання Згенерувати план виконання запиту, відповідного заданому оператору SQL;

– EXECUTE – Виконати оператор SQL, раніше підготовлений до динамічного виконання. Виконує раніше підготовлений план запиту.

У мові SQL / 89 підтримуються наступні *типи даних*:

– • CHARACTER (n) або CHAR (n) – символьні рядки постійної довжини в n символів. При завданні даного типу під кожне значення завжди відводиться n символів, і якщо реальне значення займає менш, ніж n символів, то СУБД автоматично доповнює відсутні символи пробілами;

– NUMERIC [(n, m)] – точні числа, тут n – загальна кількість цифр в числі, m - кількість цифр коду точки;

– DECIMAL [(n, m)] – точні числа, тут n – загальна кількість цифр в числі, m - кількість цифр коду точки;

– DEC [(n, m)] – те саме, що і DECIMAL [(n, m)];

– INTEGER або INT – цілі числа;

– SMALLINT – цілі числа меншого діапазону;

– FLOAT [(n)] – числа великої точності, збережені в формі з плаваючою крапкою. Тут n – число байтів, що резервуються під зберігання одного числа. Діапазон чисел визначається конкретною реалізацією;

– REAL – речовинний тип чисел, який відповідає числам з плаваючою точкою, меншою точності, ніж FLOAT;

– DOUBLE PRECISION специфікує тип даних з певною в реалізації точністю більшої, ніж визначена в реалізації точність для REAL.

Оператор вибору SELECT.

Мова запитів (Data Query Language) в SQL складається з єдиного оператора SELECT. Цей єдиний оператор пошуку реалізує всі операції реляційної алгебри. Один і той же запит може бути реалізований декількома засобами, і, будучи всі правильними, вони, тим не менш, можуть істотно відрізнятися за часом виконання, і це особливо важливо для великих баз даних.

Синтаксис оператора SELECT виглядає так:

SELECT [ALL | DISTINCT] (<Список полів> | *)
FROM <Список таблиць>
[WHERE <Предикат-умова вибірки або з'єднання>]
[GROUP BY <Список полів результату>]
[HAVING <Предикат-умова для групи>]
[ORDER BY <Список полів, за якими упорядкувати висновок>]

Тут ключове слово **ALL** означає, що в результуючий набір рядків включаються всі рядки, що задовольняють умовам запиту. Значить, в результуючий набір можуть потрапити однакові рядки. І це порушення принципів теорії відношень (на відміну від реляційної алгебри, де за замовчуванням передбачається відсутність дублікатів в кожному результуючому відношенні). Ключове слово **DISTINCT** означає, що в результуючий набір включаються тільки різні рядки, тобто дублікати рядків результату не включаються в набір.

Символ «*» (зірочка) означає, що в результуючий набір включаються всі стовпці з вихідних таблиць запиту.

У розділі **FROM** задається перелік вихідних відношень (таблиць) запиту.

У розділі **WHERE** задаються умови відбору рядків результату або умови з'єднання кортежів вихідних таблиць, подібно операції умовного з'єднання в реляційній алгебрі.

У розділі **GROUP BY** задається список полів угруповання.

У розділі **HAVING** задаються предикати-умови, що накладаються на кожну групу.

У частині **ORDER BY** задається список полів упорядкування результату, тобто список полів, який визначає порядок сортування в результуючому відношенні. Наприклад, якщо першим полем списку буде вказана Прізвище, а другим Номер групи, то в результуючому відношенні спочатку будуть зібрані в алфавітному порядку студенти, і якщо знайдуться однакові прізвища, то вони будуть розташовані в порядку зростання номерів груп.

У вираженні умов розділу **WHERE** можуть бути використані наступні предикати:

– предикати порівняння {=, <>,>, <,> =, <=}, які мають традиційний сенс;

– предикат Between A and B приймає значення між A і B. Предикат правдивий, коли зрівнюване значення потрапляє в заданий діапазон, включаючи межі діапазону. Одночасно в стандарті заданий і протилежний предикат Not Between A and B, який правдивий тоді, коли зрівнюване значення не влучає у заданий інтервал, включаючи його кордони;

– предикат входження в безліч IN (безліч) правдивий тоді, коли зрівнюване значення входить в безліч заданих значень. При цьому безліч значень може бути задано простим перерахуванням або вбудованим підзапитом. Одночасно існує протилежний предикат NOT IN (множина), який правдивий тоді, коли зрівнюване значення не входить в задану множину;

– предикати порівняння зі зразком LIKE і NOT LIKE. Предикат LIKE вимагає завдання шаблону, з яким порівнюється задане значення, предикат істинний, якщо зрівнюване значення відповідає шаблону, і хибне в іншому випадку. Предикат NOT LIKE має протилежний зміст.

Розглянемо детально перші три рядки оператора SELECT.

SELECT – ключове слово, яке повідомляє СУБД, що ця команда – запит. Всі запити починаються цим словом з подальшим пропуском. За ним може слідувати спосіб вибірки – з видаленням дублікатів (DISTINCT) або без видалення (ALL, мається на увазі – за замовчуванням). Потім слідує список перерахованих через кому стовпців, які вибираються запитом з таблиць, або символ «*» (зірочка) для вибору всього рядка. Будь-які стовпці, не перераховані тут, не будуть включені в результуюче відношення, відповідне виконання команди. Це, звичайно, не означає, що вони будуть видалені або їх інформація буде стерта з таблиць, тому що запит не вплине на інформацію в таблицях – він тільки показує дані.

• FROM – ключове слово, подібно SELECT, яке повинно бути представлено в кожному запиті. Воно супроводжується пропуском і потім іменами таблиць, використовуваних як джерело інформації. У разі якщо вказано більше одного імені таблиці, неявно мається на увазі, що над перерахованими таблицями здійснюється операція декартовий добуток. Таблицями можна привласнити імена-псевдоніми, що буває корисно для здійснення операції з'єднання таблиці з самою собою або для доступу з вкладеного підзапиту до поточного запису зовнішнього запиту (вкладені підзапити тут не розглядаються).

• WHERE – ключове слово, за яким йде предикат-умова, що накладається на запис в таблиці, з яким вона повинна задовольняти, щоб потрапити до вибірки, аналогічно операції селекції в реляційній алгебрі.

Усі наступні розділи оператора SELECT необов'язкові.

5.2 Запити SQL на вибірку даних

У наш час збільшується попит на комп'ютерні системи, що розробляються в архітектурі «клієнт/сервер». Такі системи дозволяють надавати ресурси і інформацію сервера інших комп'ютерів, які перебувають з ним у мережі. Популярність цього підходу забезпечує більшу надійність розроблюваних систем, а також сприяє зменшенню витрат на виробництво і зміст такого виду систем. Одним з найбільш популярних використанням цього підходу є розробка СУБД в архітектурі «клієнт/сервер».

Основними завданнями системи SQL-сервер є організація одночасного доступу до даних великої кількості користувачів, а також маніпуляція інформацією, що зберігається в базі даних SQL-сервер, що підтримує реляційну модель даних.

У системах, організованих в архітектурі «клієнт/сервер», підтримується колективний доступ до даних. Тут кожен комп'ютер здійснює операції, пов'язані зі зберіганням, доступом і обробкою даних.

Система SQL-сервер зберігає створювані об'єкти у відповідних файлах на диску комп'ютера-сервера. При цьому для таких об'єктів як база даних, створюються спеціальні таблиці. У них зберігається інформація про різні елементи бази даних – індексах, таблицях користувачів і т.д. Файли бази даних зберігаються з розширенням «***MDF**», а системні файли з розширенням «***LDF**».

Основні операції, пов'язані з управлінням роботою SQL-сервера, здійснюються за допомогою ряду утиліт, що входять до складу системи:

– **SQL Server Query Analyzer –** утиліта, що дозволяє виконувати команди мови запитів Transact-SQL;

– **SQL Server Service Manager –** утиліта, що виконує роботу, пов'язану з запуском, зупинкою і тимчасовим призупиненням роботи SQL-сервера;

– **SQL Server Enterprise Manager –** утиліта, що надає користувачеві можливості адміністрування SQL-сервера, доступу до всіх його об'єктів, а також запуску різних утиліт і додатків (у тому числі вищеописаних).

SQL Server Query Analyzer надає можливість виконання операторів Transact-SQL в базі даних SQL-сервера. Ця утиліта функціонує в середовищі Windows, що робить її зручною у використанні. При запуску система в діалоговому вікні запитує ім'я SQL-сервера, ім'я користувача і його пароль. Скориставшись введеної інформацією, система здійснює підключення утиліти до даного SQL-серверу.

В процесі роботи часто потрібно зберігати запити для подальшого використання. В цьому випадку утиліта дозволяє створити спеціальний пакетний файл з розширенням **«*sql»,** який буде містити в собі текст введеного SQL-запиту. Основні команди маніпулювання з даними файлами знаходяться в меню «File» утиліти.

SQL Server Enterprise Manager дозволяє виконувати всі основні операції адміністрування SQL-сервера. З її допомогою можна здійснювати запуск всіх утиліт і додатків, що входять до складу SQL-сервера. Наявність утиліти на комп'ютері дозволяє здійснювати конфігурацію і віддалених серверів, тобто утиліта може запускатися не тільки на самому сервері, а й на комп'ютері робочої станції.

Основне вікно системи дуже схоже на «Провідник» у Windows. У лівій частині знаходяться основні об'єкти SQI-сервера. Використання символів «+» і «–» зліва від назви об'єкта дозволяє розкривати його складові, що в свою чергу дає можливість їх редагування. При виборі необхідного об'єкта опції настройки його параметрів відображаються в правій частині вікна утиліти.

Для перевірки сказаного виконаємо дії, що дозволяють переглянути системну таблицю «sysdatabases», в якій знаходиться перелік всіх баз даних SQL-сервера. Отже, після запуску утиліти слід розкрити групу **SQL Server Group,** скориставшись натисканням миші на символ «+» зліва від назви групи. Тут буде представлений перелік всіх доступних SQL-серверів. Аналогічно розкриємо групу потрібного SQL-сервера. Зверніть увагу на те, що якщо обраний SQL-сервер знаходився в не занедбаному стані, то буде автоматично викликана утиліта **SQL Server Service Manager,** за допомогою якої і буде запущений SQL-

сервер. Також в цьому випадку можлива поява діалогового вікна запиту імені і пароля користувача. Після цього утиліта здійснить підключення до даного SQL-сервера і відобразить список об'єктів, з яких він складається.

Аналогічно можна розкрити об'єкт **Databases**, у якому необхідно відобразити об'єкти бази даних «master». Зверніть увагу на іконки, розташовані зліва від назви об'єкта. Використання такого графічного відображення спрощує роботу з SQL-сервер.

Отже, у групі об'єктів бази даних тепер відсутні символи розкриття («+» і «–»), а це говорить про те, що обраний найнижчий рівень даного дерева перегляду. Після вибору об'єкта **Tables** утиліта відобразить всі таблиці знаходяться в базі даних «master».

Відмінною рисою роботи з утилітою є змінюються панель інструментів і команди меню. Ці зміни здійснюються в залежності від обраного в даний момент об'єкта. Це в першу чергу відноситься до меню **Action** і її графічного відображення на панелі інструментів.

У нашому випадку слід вибрати в списку таблиць бази даних «master» системну таблицю **sysdatabases**. Для перегляду її вмісту використовується команда **Open Table / Return all rows** меню **Action.** Альтернативним способом виконати те ж саме є вибір відповідної команди з контекстного меню даного об'єкта, що викликається натисканням правої кнопки миші.

Меню «View» даної утиліти управляє зовнішнім виглядом, що відображається. При цьому користувачеві доступні наступні команди:

Console Tree – виводить або прибирає дерево об'єктів, розташоване в лівій частині вікна утиліти (при цьому спосіб перегляду об'єктів буде віддалено нагадувати роботу із засобом «Мій комп'ютер» Windows);

Description Bar – виводить або прибирає з екрана інформаційну панель, розташовану під меню утиліти;

Status Bar – виводить або прибирає рядок стану, розташовану в нижній частині вікна утиліти;

Toolbars – управління панелями інструментів;

Large – відображення об'єктів у вигляді великих іконок;

Small – відображення об'єктів у вигляді маленьких іконок;

List – відображення об'єктів у вигляді списку;

Detail – відображення об'єктів у вигляді таблиці (найбільш повна інформація про об'єкти представляється саме в цьому режимі).

За допомогою команд меню **Tools** можна провести запуск всіх утиліт і додатків, що входять до складу SQL-сервера, здійснити деякі налаштування самої утиліти, наприклад, змінити розмір і тип шрифту.

Створення бази даних. Для створення бази даних за допомогою **SQL Server Enterprise Manager** на першому етапі необхідно виконати деякі настройки самої утиліти. Для цього, після її запуску, виберіть у лівому списку об'єктів групу **SQL Server Group**, в якій клацніть на імені потрібного SQL-сервера. Після цього, скориставшись командою **Edit SQL Server Registration properties** меню **Action**, встановіть наступні налаштування підключення утиліти до даного SQL-сервера:

Server – за допомогою цього списку визначається база даних, для якої виробляються настройки;

Use Windows NT authentication – при підключенні утиліти до SQL-сервера в якості імені та пароля користувача передаються ім'я і пароль облікового запису користувача в системі Windows;

Use SQL Server authentication – для встановлення назви і пароля користувача SQL-сервера. При цьому в поля «Login Name» і «Pass-word» слід ввести відповідно ім'я і пароль користувача SQL-сервера;

Server Group – вибір групи серверів баз даних;

Display SQL Server state in console – відображення стану запуску SQL-сервера.

Show system databases and system objects – вся системна інформація в системі SQL-сервер (наприклад, перелік баз даних, імена і паролі користувачів і т.д.) зберігається в спеціальних системних базах даних;

Automatically start SOL Server when connecting – вибір цієї опції визначає, чи слід виконувати запуск SQL-сервера при спробі підключення до нього за допомогою даної утиліти.

Варто також зазначити, що процедура створення бази даних в SQL-сервері вимагає наявності прав адміністратора сервера, в зв'язку з чим необхідно упевнитися в тому, що при підключенні було використано ім'я користувача **sa**.

Наступним кроком буде вибір групи **Databases** в списку використовуваного SQL-сервера. Результатом цієї дії буде відображення в правій частині діалогового вікна утиліти всіх наявних баз даних на використовуваному сервері. Вибір команди **New Database** меню **Action** дозволяє створити нову базу даних в використовуваному сервері.

Якщо в процесі використання SQL-сервера є необхідність у видаленні раніше створеної бази даних, то в цьому випадку виконують такі дії:

– виберіть базу даних у списку об'єктів **Databases** SQL-сервера;

– виконайте команду **Delete** меню **Action.**

Таблиці бази даних. Виберіть в списку об'єктів бази даних групу **Tables**, після чого в правій частині утиліти **SQL Server Enterprise Manager** буде відображено список всіх її таблиць в тому числі і системних. Виконайте команду **New Table** меню **Action**, після чого на екрані з'явиться запит введення імені створюваної таблиці. Для зручності рекомендується використання символів у верхньому регістрі в назві таблиць, що дозволить візуально відрізняти призначені для користувача таблиці від системних.

Для прикладу скористаємося структурою таблиці **STUDENTS,** в якій знаходиться інформація про студентів навчального закладу (табл. 5.1).

Таблиця 5.1 – Структура таблиці **STUDENTS** бази даних **EDUCATION**

Column Name	Datatype	Length	Precision	Scale
Таблиця STUDENTS				
SNUM	int	4	10	0
SPAM	char	20	0	0
SIMA	char	10	0	0
SOICH	char	15	0	0
STIP	small-money	4	10	4

На запит введення імені слід ввести **STUDENTS,** після чого підтвердити введення натисканням кнопки **ОК.** Потім утиліта відобразить на екрані вікно дизайнера таблиць. У колонку **Column Name** необхідно ввести назву стовпчика таблиці (в нашому випадку **SNUM**), після чого визначити його тип даних, скориставшись колонкою **Datatype** вікна дизайнера. Тут в випадаючому списку відображається перелік усіх доступних типів даних визначених у SQL-сервері. Після вибору типу даних для створюваного поля система автоматично підставить для нього параметри **Length** (розмір поля), **Precision** (десятковий розмір) і **Scale** (точність числового типу даних).

Якщо необхідно провести видалення рядка в дизайнері таблиць, то для цього потрібно клацнути мишею на кнопку, розташовану зліва від цього рядка, що призведе до її виділення. Натискання кнопки **Delete** викликає діалогове вікно для видалення таблиці.

За потреби внесення змін в структуру таблиці після її створення дизайнер таблиць можна завжди викликати, скориставшись командою **Design Table** меню **Action,** попередньо обравши таблицю в списку.

Наступним етапом розробки структури навчальної бази даних **EDUCATION** буде внесення інформації в створені таблиці. Для зміни вмісту таблиць за допомогою утиліти **SQL Server Enterprise Manager** необхідно виконати наступні дії:

– вибрати необхідну таблицю в списку;

– виконати команду **Open Table / Return all rows** меню **Action.**

Використання кнопки **Show / Hide SOL Panel** дозволяє виводити на екран або прибрати панель введення SQL-команд. При цьому діалогове вікно редактора розбивається на дві частини для введення необхідних команд SQL. Після завершення введення цих команд слід оновити результати в таблиці, скориставшись кнопкою **RWJ,** розташованої на панелі інструментів редактора. Тут також можна прибрати або вивести з екрану панель результатів запиту, скориставшись кнопкою **Show / Hide Remits Panel.** Якщо в процесі написання SQL-команди з'являється необхідність переконатися в коректності цієї команди, то слід скористатися кнопкою **Verify SQL,** яка здійснить перевірку і виведе відповідні попередження, не звертаючись при цьому до таблиці бази даних.

Розглянемо процес створення представлень на прикладі навчальної бази даних **EDUCATION.** Тут необхідно створити представлення, яке буде зв'язувати таблиці **PREDMET** (табл. 5.2) і **TEACHERS** (табл. 5.3), іншими словами, буде відображати інформацію про навчальні курси і викладачів, які проводять по ним заняття.

Таблиця 5.2 – Структура таблиці **PREDMET** бази даних **EDUCATION**

Column Name	Datatype	Length	Precision	Scale
Таблиця PREDMET				
PNUM	smallint	2	5	0
PNAME	char	20	0	0
TNUM	intl	4	10	0
HOURS	smallint	2	5	0
COURS	tinyint	1	3	0

Таблиця 5.3 – Структура таблиці **TEACHERS** бази даних **EDUCATION**

Column Name	Datatype	Length	Precision	Scale
Таблиця TEACHERS				
TNUM	mt	4	10	0
TFAM	char	20	0	0
TIMA	char	10	0	0
TOTCH	char	15	0	0
TDATE	small-datetime	4	0	0

Щоб створити представлення, яке буде відображати повну інформацію з бази даних **EDUCATION,** слід визначити перелік студентів з оцінками, проставленими їм з різних предметів, а також список викладачів, які проводили заняття за даними курсам. Іншими словами, в представленні має бути присутня наступна інформація:

– поля **TFAM, TIMA, TOTCH** таблиці **TEACHERS;**
– поля **PNAME, HOURS** таблиці **PREDMET;**
– поля **OCENKA, UDATE** таблиці **USP;**
– поля **SFAM, SIMA, SOTCH** таблиці **TEACHERS.**

SQL-команда для цього подання повинна виглядати так:

```
SELECT STUDENTS.SFAM,
STUDENTS.SIMA, STUDENTS.SOTCH,
PREDMET . PNAME , PREDMET . HOURS ,
USP.OCENKA, USP. UDATE, TEACHERS . TFAM,
TEACHERS . TIMA, TEACHERS . TOTCH FROM TEACHERS INNER JOIN
PREDMET ON
TEACHERS . TNUM = PREDMET . TNUM INNER JOIN
USP ON PREDMET. PNUM = USP. PNUM
INNER JOIN
STUDENTS ON USP.SNUM = STUDENTS. SNUM
```

Команди SQL та вивід таблиці студентів:

```
SELECT SNUM, SFAM, SIMA, SOTCH, STIP FROM STUDENTS
SELECT  *  FROM  STUDENTS
```

Примітка: при явній вказівці полів можна змінити їх послідовність.

«*» Виводить поля так, як вони розташовані в таблиці.

Ключові слова:

DISTINCT опускає записи, де всі вибрані поля ідентичні. Якщо замість **DISTINCT** вказати **ALL**, то це буде мати протилежний ефект і збережеться дублювання рядків виводу.

SELECT DISTINCT STIP FROM STUDENTS;

Вивід:

25.50

17.00

0.00

WHERE – пропозиція команди **SELECT**, яке дозволяє встановлювати предикати, умова яких може бути або вірним або невірним для будь-якого запису таблиці. Команда витягує тільки ті записи з таблиці, для яких таке твердження істинне.

SQL володіє наступним набором операторів порівняння і зв'язування:

«=» дорівнює чомусь;

«>» більше ніж;

«<» менше ніж;

«> =» більше ніж або дорівнює:

«<=» менше ніж або дорівнює;

«<>» не дорівнює.

Стандартними булевими операторами, які використовуються в SQL, є **AND, OR** і **NOT**. Нагадаємо, як вони працюють:

AND використовує два операнда в формі «A AND B» і оцінює їх по відношенню до істини: чи правильні вони обидва;

OR використовує два операнда в формі «A OR B» і оцінює на істинність: чи вірний один з них;

NOT використовує один операнд в формі «NOT A» і замінює його значення з «ИСТИНА» на «ЛОЖЬ», або навпаки.

Приклади:

```
SELECT   SFAM, STIP
FROM STUDENTS
WHERE   STIP  > 0
```

Результат:

Поляков 25.50

Старова 17.00

Нагорний 25.50

У пропозиції **SELECT** у доповнення до традиційних реляційних і булевих операторів, які розглянуті вище, можуть бути використані спеціальні оператори **IN, BETWEEN. LIKE,** і **IS NULL**.

Оператор **IN** визначає набір значень, в який дане значення має бути включено.

```
SELECT  *
FROM STUDENTS
WHERE  STIP  IN  (17.00,  25.50).
```

BETWEEN визначає діапазон значень, в який повинні вміщатися шукані значення, що і робить предикат вірним.

```
SELECT  SNUM,  OCENKA
FROM USP
WHERE  OCENKA BETWEEN  3  AND  5
```

Оператор **LIKE** застосовний тільки до полів типу **CHAR** або **VARCHAR**, в яких він шукає підрядки, тобто він шукає символи і перевіряє, чи збігаються вони з умовою.

```
SELECT TFAM, TIMA, TOTCH
FROM TEACHERS
WHERE TFAM LIKE ‘K%’
```

Багатотабличні запити. При багатотабличних запитах таблиці, представлені у вигляді списку в пропозиції **FROM**, відокремлюються один від одного комами. Предикат запиту може посилатися до будь-якого стовпця будь-якої пов’язаної таблиці і, отже, може використовуватися для зв’язку між ними.

Поставити у відповідність викладачеві навчальні предмети, які він веде:

```
SELECT TEACHERS.TFAM, PREDMET.PNAME
FROM TEACHERS, PREDMET
WHERE TEACHERS.TNUM = PREDMET.TNUM
```

Успішність студентів:

```
SELECT STUDENTS.SFAM, STUDENTS.SNUM,
USP.PNUM, USP.SNUM
FROM STUDENTS INNER JOIN USP ON STUDENTS.SNUM = USP.SNUM
```

5.3 Підсумкові та підпорядковані SQL-запити

Отримання підсумкових значень (агрегатні функції):

COUNT – проводить підрахунок кількості рядків або ні – NULL значень полів, які вибрав запит;

SUM – розраховує арифметичну суму всіх вибраних значень даного поля;

AVG – виробляє усереднення всіх обраних значень даного поля;

MAX – знаходить і повертає найбільше з усіх вибраних значень даного поля;

MIN – знаходить і повертає найменше з усіх вибраних значень даного поля.

Кількість студентів, які здавали навчальні предмети по таблиці успішності:

```
SELECT COUNT (DISTINCT SNUM) FROM USP
```

Команда **GROUP BY** дозволяє визначати підмножину значень в поле в термінах іншого поля і застосовувати функцію.

Визначити найменшу оцінку, отриману кожним студентом:

```
SELECT SNUM, MIN (OCENKA) FROM USP GROUP BY SNUM
```

Пропозиція **HAVING**, визначає критерії, використовувані для видалення певних груп з виведення.

Приклад:

```
SELECT  SNUM,  MIN  (OCENKA.)
FROM USP
GROUP  BY  SNUM
HAVING SNUM IN  (3412,  3413)
```

Сортування результатів запиту. Для упорядкування виведення полів таблиць SQL використовує команду **ORDER BY**, дозволяючи сортувати вивід запиту відповідно до значень в тій або іншій кількості обраних стовпців. Якщо вказується кілька полів, то стовпці виведення упорядковуються один всередині іншого, при цьому можна визначати зростання (**ASC**) або спадання (**DESC**) для кожного стовпця. За умовчанням встановлено зростання.

```
SELECT *
FROM STUDENTS
ORDER BY SFAM ASC
```

Кількість студентів, які отримують ту чи іншу стипендію, але з упорядкуванням за зменшенням розмірів їх стипендій:

```
SELECT COUNT (DISTINCT SNUM), ‘студ. отр. стипендію, STIP, ‘у.е.’
FROM  STUDENTS
GROUP BY  STIP
ORDER BY  STIP DESC
```

5.4 Індексування таблиць

Важливим методом, здатним суттєво підвищити швидкість виконання запитів до реляційних баз даних, вважається індексування таблиць. У результаті індексування базових таблиць та представлень утворюються індекси.

Індекси – це об'єкти бази даних, що підвищують продуктивність запитів, допомагаючи СУБД знайти потрібний кортеж. Індекс нагадує за своїм призначенням книжковий зміст, по якому читач швидко відшукує потрібну інформацію. Індекс бази даних формується зі значень одного або декількох атрибутів таблиці, які у цьому випадку називаються ключем індексу, і покажчиків на відповідні кортежі. При виконанні запиту з індексом оптимізатор запитів використовує ключ індексу для пошуку потрібного кортежу.

Диспетчер індексів структурує індекс у вигляді збалансованого дерева, або «B-дерева» (англ. «Balanced tree», «B-tree»). Кожен об'єкт у структурі дерева являє собою групу відсортованих ключів індексу, яка називається сторінкою індексу. Упорядкування індексу за значеннями його ключів також підвищує продуктивність запитів. СУБД починає виконання всіх запитів на пошук даних з коріння «B-дерева», потім просувається далі по стовбуру, доки не досягне листового рівня.

Глибина дерева залежить від числа кортежів таблиці, а ширина дерева – від розміру ключа індексу. Дерево, створене для таблиці з великою кількістю кортежів і великим ключем індексу, є широким та глибоким. Це небажано, оскільки, чим менше дерево, тим швидше повертається результат пошуку.

Для оптимальної продуктивності запитів слід створювати індекси на тих атрибутах таблиці, які часто застосовуються в запитах на отримання даних. Небажано створювати індекс для кожного атрибута таблиці, оскільки при модифікації індексованої таблиці індекс обов'язково оновлюється. Отже, збільшення кількості індексів може негативно вплинути на продуктивність бази даних. Тому, перш ніж створювати індекс, варто переконатися, що запланований виграш у продуктивності запитів переважить додаткову витрату ресурсів системи на його супровід.

Існує два типи індексів: кластерні та некластерні. *Кластерний індекс* містить кортежі таблиці, а *некластерний* – покажчики на кортежі. Якщо для таблиці побудований кластерний індекс, то некластерний можна використовувати як допоміжний при пошуку даних. У більшості випадків для таблиці спочатку слід створювати кластерний індекс, а потім – один або декілька некластерних.

Таблиця або представлення може мати лише один кластерний індекс, оскільки ключ кластерного індексу фізично впорядковує таблицю, або представлення. Цей тип індексів особливо ефективний при виконанні запитів. Підходом до сортування кортежів кластерний індекс нагадує словник з його алфавітним порядком сортування слів і наявністю визначень після кожного слова.

При створенні обмеження **PRIMARY KEY** в таблиці, де ще немає кластерного індексу, СУБД використовує для створення ключа кластерного індексу атрибут з первинним ключем таблиці. Якщо в таблиці вже є кластерний індекс, то для атрибуту з обмеженням **PRIMARY KEY** створюється некластерний індекс. Атрибут з первинним ключем корисний для індексу, оскільки в ньому гарантовано містяться унікальні значення. У цьому випадку розмір «B-дерева» менше, ніж при використанні надмірних значень, отже механізм пошуку працює ефективніше.

Індекси можуть створюватися не тільки для атрибутів з обмеженнями, а й для будь-якого атрибуту або комбінації атрибутів таблиці чи представлення. Унікальність кластерного індексу забезпечується за допомогою внутрішніх механізмів. Тому при створенні неунікального кластерного індексу для атрибуту з надмірними значеннями СУБД генерує для значень атрибутів, що дублюються, унікальні ідентифікатори, які використовуються як вторинний ключ сортування. Щоб уникнути додаткової роботи з підтримки унікальних значень атрибутів з надмірними значеннями, слід віддати перевагу кластерним індексам на атрибутах з обмеженням **PRIMARY KEY**.

Кількість некластерних індексів кінцева (в SQL останніх версій можна створити до 250 некластерних індексів або 249 некластерних і 1 кластерний). Якщо в таблиці немає кластерного індексу, таблиця є неврегульованою і має назву купа. Некластерний індекс, створений для купи, містить покажчики на записі таблиці. Кожен елемент сторінки індексу містить ідентифікатор запису («row ID», «RID») – покажчик на табличний кортеж у купі, що містить номер елемента зовнішньої пам’яті, з якої починається сторінка. За наявності кластерного індексу сторінки некластерного індексу містять замість RID-ідентифікаторів ключі кластерного індексу.

Крім типу (кластерний або некластерний), індекс має ряд інших властивостей. Згідно з ним кожен індекс можна визначити як:

1. Унікальний індекс. В унікальному індексі ключі та відповідні їм значення атрибутів повинні бути унікальними.

2. Складний індекс. Складним називається індекс, ключ якого утворений декількома атрибутами таблиці.

3. Індекс з упорядкуванням ключів – це індекс, в якому ключі впорядковані за збільшенням або за зменшенням значень.

4. Повнотекстовий індекс – цей індекс дозволяє виконувати повнотекстові запити для пошуку в базі даних рядків символьних даних.

Для створення індексів використовують наступні оператори SQL: **CREATE INDEX, CREATE TABLE** і **ALTER TABLE.**

При застосуванні **CREATE INDEX** слід вказати ім'я індексу, таблицю або представлення, а також атрибут (атрибути), для яких створюється індекс. Також можна, але необов'язково задати унікальність індексу, його тип (кластерний або некластерний), порядок сортування кожного атрибуту, властивості індексу та розташування групи файлів для зберігання індексу. Основні конструкції оператора **CREATE INDEX** мають наступний вигляд:

CREATE

UNIQUE CLUSTERED | NONCLUSTERED INDEX ім'я індексу ON ім'я таблиці | ім'я представлення (ім'я атрибуту)

WITH властивість індексу ON група файлів

До необов'язкових належать конструкції **UNIQUE, CLUSTERED** або **NONCLUSTERED.** Також необов'язково задавати властивості індексу за допомогою конструкції **WITH** і групи файлів, в якій створюється індекс (використовуючи другу конструкцію **ON**). Для необов'язкових параметрів встановлюється значення за замовчуванням, а саме:

– створити некластерний індекс;

– упорядкувати всі кортежі індексу за збільшенням ключового атрибуту;

– розміщувати всі результати сортування під час створення індексу в групі файлів за замовчуванням.

Наприклад, в наступному операторі:

CREATE INDEX ОпераціяРахунок ON Операція(Рахунок)

для всіх необов'язкових конструкцій застосовуються параметри за умовчанням: у таблиці «Операція» створюється неунікальний і некластерний індекс «ОпераціяРахунок», ключем якого є атрибут «Рахунок».

У багатьох СУБД під час визначення в таблиці обмежень **PRIMARY KEY** та **UNIQUE** індекс створюється автоматично. Ці обмеження визначаються під час створення або модифікації таблиці. Оскільки оператори **CREATE TABLE** і **ALTER TABLE** дають можливість задавати ці обмеження, їх можна

використовувати для створення індексів. Основні конструкції оператора **CREATE TABLE,** що пов'язані із створенням індексів, наступні:

CREATE TABLE ім'я таблиці
(ім'я атрибуту тип даних
CONSTRAINT ім'я обмеження
PRIMARY KEY | UNIQUE
CLUSTERED | NONCLUSTERED
WITH властивість індексу ON група файлів)
Наприклад:
CREATE TABLE Клієнт
(КлієнтІдн int PRIMARY KEY NONCLUSTERED,
ЛюдинаІдн int UNIQUE CLUSTERED,
Статус varchar(40),)

Синтаксис оператора **ALTER TABLE** для створення чи модифікації обмежень **PRIMARY KEY,** або **UNIQUE,** аналогічний **CREATE TABLE.** Невелика відмінність полягає у необхідності вказувати в операторі **ALTER TABLE,** що буде відбуватися з обмеженням: зміна, додавання або видалення.

У випадку, якщо індекс більше непотрібен або пошкоджений, він підлягає видаленню. Зокрема, необхідно видаляти індекси для таблиць, значення ключів яких почали часто змінюватись. В іншому випадку СУБД витрачатиме зайві обчислювальні ресурси на супровід таких індексів. Синтаксис видалення індексів такий:

DROP INDEX ім'я таблиці. ім'я індексу

В операторі **DROP INDEX** варто вказати ім'я таблиці або представлення. Один оператор **DROP INDEX** може видаляти декілька індексів, розділених комами.

Шляхом послідовного використання команд **DROP INDEX** та **CREATE INDEX** можна змінювати, в разі потреби, властивості індексу. Проте для великих таблиць перебудова індексів у такий спосіб може вимагати надто багато процесорного ресурсу. Зокрема, у випадку, якщо для таблиці або представлення створений кластерний індекс, то всі некластерні індекси цієї таблиці використовують посилання на його ключ. При видаленні кластерного індексу за допомогою оператора **DROP INDEX** усі некластерні індекси перебудовуються на застосування замість ключа кластерного індексу посилання на RID-ідентифікатори. Коли згодом відновити кластерний індекс за допомогою оператора **CREATE INDEX,** то всі некластерні індекси знову перебудовуються на застосування його ключа.

Зважаючи на такі незручності, на практиці застосовують більш досконалі способи реорганізації індексів. Оператор **DBCC DBREINDEX** перебудовує один або декілька індексів таблиці чи представлення. Для перебудови всіх індексів слід примусити **DBCC DBREINDEX** перебудувати кластерний індекс. Таким чином реалізується перебудова всіх індексів таблиці або представлення. Інший спосіб – запустити оператор без вказівки імені, при цьому також перебудовуються всі індекси. Оператор **DBCC DBREINDEX** особливо корисний для перебудови індексів, створених обмеженнями **PRIMARY KEY** і **UNIQUE,** оскільки на відміну від **DROP INDEX** перед перебудовою необов'язково видаляти обмеження.

Іноді, внаслідок зміни угод про іменування або при невідповідності існуючих індексів угодам про іменування здійснюється їх перейменування. Перейменувати індекс можна видаливши його і створивши наново. Однак сучасні СУБД надають простіші способи. Зокрема, SQL дозволяє перейменувати індекс засобами системної збереженої процедури **sp_rename,** наприклад:

sp_rename @objname = 'Операція.ОпераціяРахунок', @newname = 'ІндексРахунок', @objtype = 'INDEX'

У значення вхідного параметра **@objname** вноситься ім'я таблиці. Якщо цього не зробити, процедура не знайде індекс, який потрібно перейменувати. Проте ім'я таблиці виключене з вхідного параметра **@newname,** оскільки воно береться з параметра **@objname.** Для вхідного параметра **@objtype** слід задати значення **INDEX,** інакше процедура не знайде потрібний тип об'єкта, який потрібно перейменувати.

Тепер, після того, як було розглянуто що таке індекс, як його створити і як ним управляти, слід звернути увагу на правила, що дозволяють визначити, коли слід створювати індекс і які властивості індексу потрібно налаштувати для підвищення продуктивності бази даних. Оскільки в таблиці або представленні може бути лише один кластерний індекс, продумати його конструкцію важливіше, ніж некластерного.

Індекси створюються для підтримки різних типів запитів до бази даних, що найчастіше виконуються користувачами. При цьому оптимізатор запитів для пошуку даних застосовує один або декілька індексів. Індекси здатні підвищити ефективність виконання запитів таких типів:

1. Запити, в яких критерій пошуку, що задається конструкцією **WHERE,** точно відповідає вказаному значенню.

Наприклад:

SELECT Прізвище, [Ім'я], [По батькові]

FROM Людина

WHERE ЛюдинаІдн = 5

Якщо конструкція **WHERE** забезпечує повернення одного певного значення – для таких запитів варто вибрати кластерний індекс. У запитах, які повертають декілька унікальних записів, краще використовувати некластерний індекс. Наприклад, якщо запросити в базі даних відомості про людей, і як критерій вибірки застосувати тільки ім'я, запит точно відповідних значень, імовірно, поверне декілька кортежів.

2. Запити із знаками підстановки, в яких для пошуку значень застосовується конструкція **LIKE** і критерій пошуку, починаються із знаку відсотка («%»). Наприклад:

SELECT Прізвище, [Ім'я], [По батькові]

FROM Людина

WHERE LIKE '%ко'

Індекси не підвищують ефективність виконання подібних запитів, оскільки ключі індексу починаються з конкретного символьного або числового значення.

3. Запити діапазону значень – запити на пошук послідовності значень, наприклад:

SELECT Прізвище, [Ім'я], [По батькові] FROM Людина

WHERE LIKE 'М%' AND ''Н%'

Для цього типу запитів краще вибрати кластерні індекси, оскільки їхні ключі фізично впорядковані. Тому, якщо знайдений перший запис, імовірно, що решта записів діапазону розташовані поряд.

4. Запити з упорядкованими результатами, що використовують конструкцію **ORDER BY.** Якщо якийсь атрибут або їх комбінація часто впорядковується як ключ для такого сортування, слід подумати про реалізацію сортування за допомогою індексу.

ЗМІСТОВИЙ МОДУЛЬ 3 ПРОЄКТУВАННЯ ТА ЗАХИСТ БАЗ ДАНИХ

ТЕМА 6 ПРОЄКТУВАННЯ БАЗ ДАНИХ

Проєкт реляційної бази даних – це набір взаємозв'язаних відношень, в яких визначені всі атрибути, задані первинні ключі відношень і задані деякі додаткові властивості відношень. Проєкт бази даних (далі – БД) повинен бути дуже точний і вивірений. Фактично проєкт бази даних – це фундамент майбутнього

програмного комплексу, який буде використовуватися досить довго і багатьма користувачами.

Процес проєктування БД являє собою послідовність переходів від неформального словесного опису інформаційної структури предметної області до формалізованого опису об'єктів предметної області в термінах деякої моделі. Загалом, можна виокремити наступні етапи проєктування:

1. Системний аналіз і словесний опис інформаційних об'єктів предметної області.

2. Проєктування інфологічної моделі предметної області – частково формалізований опис об'єктів предметної області в термінах деякої семантичної моделі, наприклад, в термінах ER-моделі.

3. Даталогічне або логічне проєктування БД, тобто опис БД в термінах прийнятої даталогічної моделі даних.

4. Фізичне проєктування БД, тобто вибір ефективного розміщення БД на зовнішніх носіях для забезпечення найбільш ефективної роботи програми.

Якщо ми врахуємо, що між другим і третім етапами необхідно прийняти рішення, з використанням якої стандартної СУБД буде реалізовуваний наш проєкт, то умовно процес проєктування БД можна уявити послідовністю виконання п'яти відповідних етапів. Розглянемо докладніше етапи проєктування БД.

Системний аналіз предметної області. З точки зору проєктування БД в рамках системного аналізу, необхідно здійснити перший етап, тобто провести докладний словесний опис об'єктів предметної області і реальних зв'язків, які присутні між описаними об'єктами. Бажано, щоб даний опис дозволяв коректно визначити всі взаємозв'язки між об'єктами предметної області.

Загалом, застосовують два підходи до вибору складу і структури предметної області:

– функціональний підхід – він реалізує принцип руху «від завдань» і застосовується тоді, коли заздалегідь відомі функції певної групи осіб і комплексів задач, для обслуговування інформаційних потреб яких створюється розглянута БД. У цьому випадку ми можемо чітко виділити мінімальний необхідний набір об'єктів предметної області, які повинні бути описані.

– предметний підхід – інформаційні потреби майбутніх користувачів БД жорстко не фіксуються. Вони можуть бути багатоаспектними і вельми динамічними. Ми не можемо точно виділити мінімальний набір об'єктів предметної області, які необхідно описувати. В опис предметної області в цьому випадку включаються такі об'єкти і взаємозв'язки, які найбільш характерні і

найбільш істотні для неї. БД, конструюється при цьому, називається предметною, тобто вона може бути використана при вирішенні безлічі різноманітних, заздалегідь не визначених завдань. Конструювання предметної БД в деякому сенсі здається набагато більш привабливим, однак труднощі загального охоплення предметної області з неможливістю конкретизації потреб користувачів може привести до надмірно складної схеми БД, яка для конкретних завдань буде неефективною.

Зазвичай на практиці використовують певний компромісний варіант, який, з одного боку, орієнтований на конкретні завдання або функціональні потреби користувачів, а з іншого боку, враховує можливість нарощування нових додатків.

Системний аналіз повинен закінчуватися докладним описом інформації про об'єкти предметної області, яка потрібна для вирішення конкретних завдань і яка повинна зберігатися в БД, формулюванням конкретних завдань, які будуть вирішуватися з використанням даної БД з коротким описом алгоритмів їх вирішення, описом вихідних документів, які повинні генеруватися в системі, описом вхідних документів, які є підставою для заповнення даними БД.

Даталогічне проєктування. У реляційних БД даталогічне або логічне проєктування призводить до розробки схеми БД, тобто сукупності схем відношень, які адекватно моделюють абстрактні об'єкти предметної області та семантичні зв'язки між цими об'єктами. Основою аналізу коректності схеми є так звані функціональні залежності між атрибутами БД. Деякі залежності між атрибутами відношень є небажаними через наявність побічних ефектів і аномалій, які вони викликають при модифікації БД. При цьому під процесом модифікації БД ми розуміємо внесення нових даних в БД або видалення деяких даних з БД, а також оновлення значень деяких атрибутів.

Однак етап логічного або даталогічного проєктування не закінчується проєктуванням схеми відношень. У загальному випадку в результаті виконання цього етапу повинні бути отримані такі підсумкові документи:

– опис концептуальної схеми БД в термінах обраної СУБД;
– опис зовнішніх моделей в термінах обраної СУБД;
– опис декларативних правил підтримки цілісності бази даних;
– розробка процедур підтримки семантичної цілісності бази даних.

Однак перед тим як описувати побудовану схему в термінах обраної СУБД, треба вибудувати цю схему. Необхідно побудувати коректну схему БД, орієнтуючись на реляційну модель даних. Коректною назвемо схему БД, в якій відсутні небажані залежності між атрибутами відношень.

Процес розробки коректної схеми реляційної БД називається *логічним проєктуванням* БД.

Проєктування схеми БД може бути виконано двома шляхами:

− шляхом декомпозиції (розбиття), коли вихідна множина відношень, що входять в схему БД, замінюється іншою множиною відношень (число їх при цьому зростає), що є проекціями вихідних відношень;

− шляхом синтезу, тобто шляхом компонування із заданих вихідних елементарних залежностей між об'єктами предметної області схеми БД.

Класична технологія проєктування реляційних баз даних пов'язана з теорією нормалізації, заснованої на аналізі функціональних залежностей між атрибутами відношень. Поняття функціональної залежності є фундаментальним в теорії нормалізації реляційних баз даних.

Функціональні залежності визначають стійкі відношення и між об'єктами і їх властивостями в аналізованій предметній області. Саме тому процес підтримки функціональних залежностей, характерних для даної предметної області, є базовим для процесу проєктування.

Процес проєктування з використанням декомпозиції являє собою процес послідовної нормалізації схем відношень, при цьому кожна наступна ітерація відповідає нормальній формі вищого рівня і має кращі властивості в порівнянні із попередньою.

Кожній нормальній формі відповідає деякий певний набір обмежень, і ставлення знаходиться в деякій нормальній формі, якщо задовольняє властивого їй набору обмежень.

Інфологічна модель використовується на ранніх стадіях розробки проєкту. Якщо розуміти мову умовних позначень, які відповідають категоріям ER-моделі, то її можна легко «читати», отже, вона доступна для аналізу програмістам-розробникам, що будуть розробляти окремі програми. Вона має однозначну інтерпретацію, на відміну від деяких пропозицій природної мови, і тому тут не може бути ніякого нерозуміння з боку розробників.

Всі фахівці завжди вважають за краще висловлювати свої думки на деякій формальній мові, яка забезпечує однозначне їх трактування. Такою мовою для програмістів раніше була мова алгоритмів. Будь-який алгоритм мав однозначну інтерпретацію. Він міг бути реалізований на різних мовах програмування, але сам алгоритм був і залишався одним і тим же. У перші роки розвитку обчислювальної техніки широко видавалися збірники алгоритмів для широко поширених математичних задач. Ці збірники програмістами прочитувалися як захоплюючі детективні романи, і вони вже справжнім програмістам були

зрозумілі, хоча фахівці інших профілів дивилися на ці збірники як на видання на іноземних, невідомих їм, мовами. Для опису алгоритмів могли використовуватися різні формалізми. Одним з таких формалізмів була метамова, в якій використовувалися слова з природньої мови і кожен міг прочитати ці слова, але зміст самого алгоритму міг зрозуміти тільки той, хто володів знаннями трактування алгоритмів.

Ось такою умовною загальноприйнятою мовою опису бази даних та стала мова ER-моделі. Для ER-моделі існує алгоритм однозначного перетворення її в реляційну модель даних, що дозволило в подальшому розробити безліч інструментальних систем, що підтримують процес розробки інформаційних систем, що базуються на технології баз даних. І у всіх цих системах існують засоби опису інфологічної моделі БД з можливістю автоматичної генерації тієї даталогічної моделі, на якій має бути реалізовуваний проєкт в подальшому.

Розглянемо правила перетворення ER-моделі на реляційну:

1. Кожній сутності ставиться у відповідність відношення реляційної моделі даних. При цьому імена суті і відношення можуть бути різними, тому що на імена сутностей можуть не накладатися додаткові синтаксичні обмеження, крім унікальності імені в рамках моделі. Імена відношень можуть бути обмежені вимогами конкретної СУБД, найчастіше ці імена є ідентифікаторами в деякій базовій мові, вони обмежені по довжині і не повинні містити пробілів і деяких спеціальних символів. Наприклад, сутність може бути названа «Книжковий каталог», а відповідне їй ставлення бажано назвати, наприклад, BOOKS (без пробілів і латинськими буквами).

2. Кожен атрибут сутності стає атрибутом відповідного ставлення. Перейменування атрибутів має відбуватися відповідно до тих же правил, що і перейменування відношень в п.1. Для кожного атрибута задається конкретний припустимий в СУБД тип даних і обов’язковість або необов’язковість даного атрибута (тобто допустимість або неприпустимість NULL значень для нього).

3. Первинний ключ сутності стає PRIMARY KEY відповідного ставлення. Атрибути, що входять в первинний ключ відношення, автоматично отримують властивість обов’язковості (NOT NULL).

4. У кожне відношення, відповідне підпорядкованої сутності, додається набір атрибутів основної суті, є первинним ключем основної суті. У відношенні, яке відповідає підпорядкованій сутності, цей набір атрибутів стає зовнішнім ключем (FOREING KEY).

5. Для моделювання необов’язкового типу зв’язку на фізичному рівні у атрибутів, відповідних зовнішньому ключу, встановлюється властивість

допустимості невизначених значень (ознака NULL). При обов'язковому типі зв'язку атрибути отримують властивість відсутності невизначених значень (ознака NOT NULL).

6. Для відображення категоризації сутностей при переході до реляційної моделі можливі кілька варіантів представлення. Можливо створити тільки одне відношення для всіх підтипів одного супертипа. У нього включають всі атрибути всіх підтипів. Однак тоді для ряду примірників ряд атрибутів не матиме сенсу. І навіть якщо вони будуть мати невизначені значення, то будуть потрібні додаткові правила розрізнення одних підтипів від інших. Перевагою такого подання є те, що створюється за все одне відношення.

7. При другому способі для кожного підтипу і для супертипа створюються свої окремі відношення. Недоліком такого способу подання є те, що створюється багато відношень, проте переваг у такого способу більше, так як ви працюєте тільки зі значущими атрибутами підтипу. Крім того, для можливості переходів до підтипів від супертипа необхідно в супертип включити ідентифікатор зв'язку.

8. Додатково при описі відношення між типом і підтипами необхідно вказати тип дискримінатора. Дискримінатор може бути взаємовиключним або ні. Якщо встановлено даний тип дискримінатора, то це означає, що один екземпляр сутності супертипа пов'язаний тільки з одним екземпляром сутності підтипу і для кожного екземпляра сутності супертипа існує нащадок. Крім того, необхідно вказати для другого способу, успадковується чи тільки ідентифікатор супертипа в підтипи або успадковуються всі атрибути супертипа.

Теорія нормалізації, яку ми розглядали раніше стосовно реляційної моделі, може бути застосована і до моделі «сутність-зв'язок». Тому нормалізацію можна проводити і на рівні інфологічної (семантичної) моделі і сенс її аналогічний нормалізації реляційної моделі. Алгоритм приведення семантичної моделі до 5-ї нормальної форми може бути таким.

Крок 1. Проаналізувати схему на присутність сутностей, які приховано моделюють кілька різних взаємопов'язаних класів об'єктів реального світу (саме це відповідає стану ненормалізованих відношень). Якщо таке виявлено, то розділити кожну з цих сутностей на кілька нових сутностей і встановити між ними відповідні зв'язки, отримана схема буде знаходитися в першій нормальній формі. Перейти до кроку 2.

Крок 2. Проаналізувати всі сутності, що мають складові первинні ключі, на наявність неповних функціональних залежностей не первинне атрибутів від атрибутів можливого ключа. Якщо такі залежності виявлені, то розділити дані сутності на 2, визначити для кожної сутності первинні ключі і встановити між

ними відповідні зв’язки. Отримана схема буде знаходитися в другій нормальній формі. Перейти до кроку 3.

Крок 3. Проаналізувати неключові атрибути всіх сутностей на наявність транзитивних функціональних залежностей. При виявленні таких розщепити кожну сутність на кілька таким чином, щоб ліквідувати транзитивні залежності. Схема знаходиться в третій нормальній формі. Перейти до кроку 4.

Крок 4. Проаналізувати всі сутності на наявність детермінантів, які не є можливими ключами. При виявленні подібних розщепити сутність на дві, встановивши між ними відповідні зв’язки. Отримана схема відповідає нормальній формі Бойса-Кодда. Перейти до кроку 5.

Крок 5. Проаналізувати всі сутності на наявність багатозначних залежностей. Якщо будуть виявлені суті, у яких є більше однієї багатозначної залежності, то розщепити такі сутності на дві, встановивши між ними відповідні зв’язки. Отримана схема буде перебувати в четвертій нормальній формі. Перейти до кроку 6.

Крок 6. Проаналізувати суті на наявність в них залежностей проекції-з’єднання. При виявленні таких розщепити сутність на необхідну кількість взаємопов’язаних сутностей і встановити між ними необхідні зв’язки. Отримана таким чином схема буде перебувати в п’ятій нормальній формі і, будучи формально перетвореною у реляційну схему за вказаними вище принципам, дасть реляційну схему також в п’ятій нормальній формі.

ТЕМА 7 ЦІЛІСНІСТЬ ДАНИХ. ЗАХИСТ БАЗ ДАНИХ

7.1 Методи захисту баз 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.

Функції безпеки даних:

– функція безпеки;
– функція секретності.

Для реалізації цих функцій використовуються одні й ті ж технічні методи захисту даних в базі. Забезпечення секретності реалізується ще й за допомогою додаткових заходів.

Функція безпеки – захист даних від ненавмисного доступу до даних і можливість їх спотворення з боку користувачів та адміністрації, а також при збоях апаратних і програмних. Забезпечення функції безпеки – внутрішнє завдання БД, оскільки пов'язана із забезпеченням нормального функціонування.

Функція секретності – захист даних від навмисного доступу користувачів або осіб, які виконують експлуатацію, або від сторонніх осіб. Забезпечення секретності, передбачає поділ всієї збереженої в БД інформації на загальнодоступні дані та дані, що використовуються конфіденційно, або безпосередньо містять секретну інформацію, або закодовану алгоритмічно).

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 СУБД підтримується один з двох найбільш розповсюджений 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, ніж обов'язковий.

Особливості методів забезпечення безпеки даних в базі:

– СУБД не допускає виконання операції над БД користувачеві, який не отримав на це відповідні права;

– санкціонування доступу до даних виконується адміністратором БД.

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 додатків, що отримують доступ до бази даних. Будь-який користувач або додаток, 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 додатку;

– 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 на зовнішні носії.

При вирішенні питань безпеки даних в обов'язки адміністратора БД входить наступне:

– класифікація даних відповідно з їх використанням;

– визначення прав доступу окремих користувачів до певних груп даних в базі та обмежень на характер операцій;

– організація системи контролю доступу до даних;

– тестування новостворюваних засобів безпеки даних;

– періодична перевірка правильності функціонування системи безпеки даних;

– дослідження випадків порушення безпеки даних з метою їх запобігання в подальшому;

– дослідження сучасних засобів безпеки даних та їх використання для вдосконалення засобів функціонуючої БД;

– АБД або відповідальний за забезпечення безпеки даних повинен добре знати склад, структуру і характеристики системи, так як збої або порушення можуть виникати у різних місцях.

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).

Фізичний захист. Основним заходом захисту даних від розкриття в разі фізичного вилучення частини БД (викрадення носія, контроль передачі даних по лінії зв'язку між системою і віддаленим терміналом тощо) є використання спеціальних методів кодування даних. Найпростішим методом є перекомпоновка символів у кортежі, записі, повідомленні. Інший метод передбачає заміну символу (групи символів) іншим символом (або групою символів) цього ж або іншого алфавіту тощо. Інший аспект ФЗ пов'язаний з безпекою даних від збоїв апаратних і програмних засобів. У цьому випадку здебільшого використовуються засоби ОС – зберігання поколінь даних, формування контрольних точок, введення системних журналів і т. п.

7.2 Адміністрування баз даних

Адміністрування бази даних – це функція управління базою даних. Особа, яка відповідальна за адміністрування БД, має назву «*Адміністратор бази даних*» (далі – АБД) або «Database Administrator» (далі – DBA).

Функція «адміністрування даних» стала активно розглядатися і визначатися як цілком самостійна з кінця 60-х років. Практичне значення це мало для підприємств, що використовують обчислювальну техніку в системах інформаційного забезпечення для своєї щоденної діяльності. Спеціалізація цієї функції з плином часу удосконалювалася, але якісні зміни в цій галузі стали відбуватися з початком використання так званих інтегрованих баз даних. Одна така база даних могла використовуватися для вирішення багатьох завдань.

Адміністрування баз даних передбачає виконання функцій, спрямованих на забезпечення надійного та ефективного функціонування системи баз даних, адекватності змісту бази даних інформаційним потребам користувачів, відображення в базі даних актуального стану предметної області.

Залежно від складності та обсягу банку даних, від особливостей використовуваної СУБД служба адміністрації бази даних може відрізнятися як за складом і кваліфікацією фахівців, так і за кількістю працюючих у цій службі.

Адміністратор бази даних виконує роботи зі створення та забезпечення функціонування БД протягом усіх етапів життєвого циклу системи.

У складі адміністрації бази даних повинні бути системні аналітики, проєктувальники структур даних і зовнішнього по відношенню до банку даних інформаційного забезпечення, проєктувальники технологічних процесів обробки даних, системні та прикладні програмісти, оператори, фахівці з технічного обслуговування.

Адміністратори бази даних виконують багато різноманітних функцій.

1. Аналіз предметної області. Опис предметної області, виявлення обмежень цілісності, визначення статусу інформації, визначення потреб користувачів, визначення статусу користувачів, визначення відповідності «дані – користувач», визначення об'ємно-часових характеристик обробки даних.

2. Проєктування структури бази даних. Визначення складу і структури інформаційних одиниць, що складають базу даних, завдання зв'язків між ними, вибір методів впорядкування даних і методів доступу до інформації, опис структури БД мовою обробки даних.

3. Завдання обмежень цілісності при описі структури бази даних і процедур обробки БД. Завдання обмежень цілісності, властивих предметної області, визначення обмежень цілісності, викликаних структурою бази даних, розробка процедур забезпечення цілісності БД при введенні і коригуванні даних, забезпечення обмежень цілісності при паралельній роботі користувачів у режимі багатьох користувачів.

4. Первісне завантаження та ведення бази даних. Розробка технології первинного завантаження та проведення (зміни, доповнення, видалення записів) БД, проєктування форм введення, створення програмних модулів, підготовка вихідних даних, введення та контроль введення.

5. Захист даних від несанкціонованого доступу:

– забезпечення парольного входу в систему: реєстрація користувачів, призначення і зміна паролів;

– забезпечення захисту конкретних даних: визначення прав доступу груп користувачів і окремих користувачів, визначення допустимих операцій над даними для окремих користувачів, вибір або створення програмно-технологічних засобів захисту даних; шифрування інформації з метою захисту даних від несанкціонованого використання;

– тестування засобів захисту даних;

– фіксація спроб несанкціонованого доступу до інформації;

– дослідження випадків порушення захисту даних і проведення заходів щодо їх запобігання.

6. Захист даних від руйнувань. Одним із засобів захисту від втрати даних є резервування. Використовується як при фізичному псуванні файлу, так і у випадках, якщо в БД внесені небажані незворотні зміни.

7. Забезпечення відновлення БД. Розробка програмно-технологічних засобів відновлення БД, організація ведення системних журналів.

8. Аналіз звернень користувачів до БД. Збір статистики звернень користувачів до БД, її збереження та аналіз (хто з користувачів, до якої інформації, як часто звертався, які виконував операції, час виконання запитів, аналіз причин безуспішних (в т.ч. і аварійних) звернень до БД.

9. Аналіз ефективності функціонування бази даних і розвиток системи: аналіз показників функціонування системи (час обробки, обсяг пам’яті, вартісні показники), реорганізація та реструктуризація баз даних, зміна складу баз даних, розвиток програмних і технічних засобів.

10. Робота з користувачами. Збір інформації про зміни в предметній області, про оцінку користувачами роботи бази даних, визначення регламенту роботи користувачів з базою даних, навчання та консультування користувачів.

11. Підготовка і підтримка системних програмних засобів. Збір і аналіз інформації про СУБД та інші прикладні програми, придбання програмних засобів, їх установка, перевірка працездатності, підтримка системних бібліотек, розвиток програмних засобів.

12. Організаційно-методична робота. Вибір та створення методики проєктування БД, визначення цілей і напрямів розвитку системи, планування етапів розвитку бази даних, розробка і випуск організаційно-методичних матеріалів.

Існує безліч різних способів організації зовнішньої пам’яті баз даних. Як і всі рішення, прийняті при організації баз даних, конкретні методи організації зовнішньої пам’яті необхідно вибирати в тісному зв’язку з усіма іншими рішеннями.

Безпосереднє управління даними у зовнішній пам’яті. Ця функція включає забезпечення необхідних структур зовнішньої пам’яті як для зберігання безпосередніх даних, що входять до БД, так і для службових цілей, наприклад, для прискорення доступу до даних у деяких випадках (звичайно для цього використовуються індекси). У деяких реалізаціях СУБД активно використовуються можливості існуючих файлових систем, в інших робота проводиться аж до рівня пристроїв зовнішньої пам’яті. Але в розвинених СУБД користувачі в будь-якому випадку не зобов’язані знати, чи використовує СУБД файлову систему, а якщо використовує, то як організовані в ній файли. Зокрема, СУБД підтримує власну систему іменування об’єктів БД. Це дуже важливо, оскільки імена об’єктів бази даних відповідають іменам об’єктів предметної області.

Управління буферами оперативної пам’яті. СУБД зазвичай працюють з БД значного розміру; принаймні цей розмір звичайно істотно перевищує доступний об’єм оперативної пам’яті. Зрозуміло, якщо при зверненні до будь-якого елементу даних буде проводитися обмін із зовнішньою пам’яттю, то вся система буде працювати зі швидкістю пристрою зовнішньої пам’яті. Єдиним же способом реального збільшення цієї швидкості є буферизація даних в оперативній пам’яті. І навіть якщо операційна система проводить загальносистемну буферизацію (як у випадку ОС UNIX), цього недостатньо для цілей СУБД, яка має в своєму розпорядженні набагато більше інформації про корисність буферизації тієї чи іншої частини БД. Тому в розвинених СУБД підтримується власний набір буферів оперативної пам’яті з власною дисципліною заміни буферів. При управлінні буферами основної пам’яті доводиться розробляти і застосовувати узгоджені алгоритми буферизації, журналізації і синхронізації.

Управління транзакціями. Транзакція – це послідовність операцій над БД, розглянутих СУБД як єдине ціле. Або транзакція успішно виконується, і СУБД фіксує зміни БД, вироблені нею, у зовнішній пам’яті, або жодна з цих змін ніяк не відбивається в стані БД. Поняття транзакції необхідне для підтримки логічної цілісності БД.

Таким чином, підтримка механізму транзакцій – обов’язкова умова навіть однокористувацьких СУБД. Але поняття транзакції набагато істотніше під багатокористувацьких СУБД. Та властивість, що кожна транзакція починається при цілісному стані БД і залишає цей стан цілісним після свого завершення, робить дуже зручним використання поняття транзакції як одиниці активності користувача по відношенню до БД.

З управлінням транзакціями в багатокористувацькій СУБД пов’язані важливі поняття серіалізації транзакцій і серіального плану виконання суміші транзакцій. Під *серіалізацією трансакцій*, що виконуються паралельно, розуміється такий порядок планування їх роботи, при якому сумарний ефект суміші транзакцій відповідний ефекту їх деякого послідовного виконання. *Серіальний план виконання суміші транзакцій* – це такий спосіб їх спільного виконання, який призводить до серіалізациі транзакцій. Якщо вдається домогтися дійсно серіального виконання суміші транзакцій, то для кожного користувача, з ініціативи якого утворена транзакція, присутність інших транзакцій буде непомітною.

Існує кілька базових алгоритмів серіалізациі транзакцій. У централізованих СУБД найбільш поширені алгоритми, які засновані на синхронізаційних захопленнях об’єктів БД. При використанні будь-якого алгоритму серіалізациі можливі ситуації конфліктів між двома або більше транзакціями з доступу до об’єктів БД. У цьому випадку для підтримки серіалізації необхідно виконати відкат (ліквідувати всі зміни, зроблені в БД) однієї або більше транзакцій. Це один із випадків, коли користувач багатокористувацької СУБД може реально відчути присутність в системі транзакцій інших користувачів.

Журналізація. Одна з основних вимог до СУБД – надійне зберігання даних у зовнішній пам’яті. Під надійністю зберігання розуміється те, що СУБД повинна бути в змозі відновити останній узгоджений стан БД після будь-якого апаратного або програмного збою. Зазвичай розглядаються два можливих види апаратних збоїв: так звані м’які збої, які можна трактувати як раптову зупинку роботи комп’ютера (наприклад, аварійне вимкнення живлення), і жорсткі збої, що характеризуються втратою інформації на носіях зовнішньої пам’яті. Прикладами програмних збоїв можуть бути аварійне завершення роботи СУБД (через помилки у програмі або деякого апаратного збою) або аварійне завершення користувальницької програми, в результаті чого деяка транзакція залишається незавершеною. Першу ситуацію можна розглядати як особливий вид м’якого апаратного збою, а при виникненні останнього потрібно ліквідувати наслідки лише однієї транзакції.

Але в будь-якому випадку для відновлення БД потрібно розташовувати деяку додаткову інформацію. Іншими словами, підтримка надійного зберігання даних в БД вимагає надмірності зберігання даних, причому та їх частина, яка використовується для відновлення, повинна зберігатися особливо надійно.

Найбільш поширений метод підтримки такої надмірної інформації – ведення журналу змін БД.

Журнал – це особлива частина БД, яка є недоступною користувачам СУБД і яка підтримується особливо ретельно (іноді підтримуються дві копії журналу, розташовувані на різних фізичних дисках). В неї надходять записи про всі зміни основної частини БД. У різних СУБД зміни БД журналізуються на різних рівнях: іноді запис у журналі відповідає деякій логічній операції зміни БД (наприклад, операції видалення рядка з таблиці реляційної БД), а часом запис відповідає мінімальній внутрішній операції модифікації сторінки зовнішньої пам'яті. У деяких системах одночасно використовуються обидва підходи.

У всіх випадках дотримуються стратегії «попереджуючого» запису в журнал (так званого протоколу Write Ahead Log – WAL). Ця стратегія полягає в тому, що запис про зміну будь-якого об'єкта БД повинен потрапити в зовнішню пам'ять журналу раніше, ніж змінений об'єкт потрапить на зовнішню пам'ять основної частини БД. Якщо в СУБД коректно дотримується протокол WAL, то за допомогою журналу можна вирішити всі проблеми відновлення БД після будь-якого збою.

Найпростіша ситуація відновлення – індивідуальний відкат транзакції. Для цього не потрібний загальносистемний журнал змін БД. Достатньо для кожної транзакції підтримувати локальний журнал операцій модифікації БД, виконаних у цій транзакції, і виробляти відкат транзакції виконанням зворотних операцій, ідучи від кінця локального журналу. У деяких СУБД так і роблять, але в більшості систем локальні журнали не підтримують, а індивідуальний відкат транзакції виконують за загальносистемним журналом, для чого всі записи від однієї транзакції пов'язують зворотним списком (від кінця до початку).

При м'якому збої у зовнішній пам'яті основної частини БД можуть знаходитися об'єкти, модифіковані транзакціями, які не закінчилися до моменту збою, і можуть бути відсутні об'єкти, модифіковані транзакціями, які до моменту збою успішно завершилися (унаслідок використання буферів оперативної пам'яті, вміст яких при м'якому збої пропадає). При дотриманні протоколу WAL у зовнішній пам'яті журналу повинні гарантовано знаходитися записи, пов'язані з операціями модифікації обох видів об'єктів. Метою процесу відновлення після м'якого збою є стан зовнішньої пам'яті основної частини БД, який виник би при фіксації в зовнішній пам'яті змін всіх трансакцій, що завершилися, і який не містив би жодних слідів незакінчених транзакцій. Щоб цього досягти, спочатку виробляють відкат незавершених транзакцій («undo»), а потім повторно

відтворюють («redo») ті операції завершених транзакцій, результати яких не відображені у зовнішній пам’яті.

Для відновлення БД після жорсткого збою використовують журнал і архівну копію БД. Архівна копія – це повна копія БД до моменту початку заповнення журналу. Для нормального відновлення БД після жорсткого збою необхідно, щоб журнал не пропав. До збереження журналу у зовнішній пам’яті в СУБД пред’являються особливо підвищені вимоги. Відновлення БД полягає в тому, що виходячи з архівної копії за журналом, відтворюється робота всіх транзакцій, які закінчилися до моменту збою. Можна навіть відтворити роботу незавершених транзакцій і продовжити їх роботу після кінця відновлення. Однак у реальних системах це зазвичай не робиться, оскільки процес відновлення після жорсткого збою є досить тривалим.

ТЕМА 8 КЛАСИФІКАЦІЯ БАЗ ДАНИХ. ОГЛЯД КЛІЄНТ-СЕРВЕРНИХ ТЕХНОЛОГІЙ

При розміщенні БД на персональному комп’ютері, який не перебуває у мережі, БД завжди використовується в монопольному режимі. Навіть якщо БД використовують кілька користувачів, вони можуть працювати з нею тільки послідовно, і тому запитань про підтримку коректної модифікації БД в цьому випадку тут не буває, вони вирішуються організаційними заходами – тобто визначенням необхідної послідовності роботи конкретних користувачів з відповідною БД. Однак навіть в деяких настільних БД потрібно враховувати послідовність зміни даних при обробці, щоб отримати коректний результат: так, наприклад, при запуску програми балансного бухгалтерського звіту всі бухгалтерські проводки – фінансові операції повинні бути вирішені заздалегідь до запуску кінцевого додатку.

Робота на ізольованому комп’ютері з невеликою базою даних зараз стає вже нехарактерною для більшості додатків. БД відображає інформаційну модель реальної предметної області, вона зростає за обсягом і різко збільшується кількість завдань, що вирішуються за її використанням, і відповідно до цього збільшується кількість додатків, що працюють з єдиною базою даних. Комп’ютери об’єднуються в локальні мережі, і необхідність розподілу додатків, що працюють з єдиною базою даних по мережі, є безсумнівною.

У процесі побудови БД для невеликої торгової фірми, у вас з’являється ряд специфічних користувачів БД, які мають свої бізнес-функції і територіально

можуть перебувати в різних приміщеннях, але всі вони повинні працювати з єдиною інформаційною моделлю організації, тобто з єдиною базою даних.

Паралельний доступ до однієї БД декількох користувачів, в тому випадку якщо БД розташована на одній машині, відповідає режиму розподіленого доступу до централізованої БД. Такі системи називаються *системами розподіленої обробки даних*.

Якщо ж БД розподілена по декількох комп'ютерах, розташованих в мережі, і до неї можливий паралельний доступ декількох користувачів, то ми маємо справу з паралельним доступом до розподіленої БД. Подібні системи називаються *системами розподілених баз даних*.

Визначимо термінологію, яка нам буде потрібна для подальшого застосування.

Користувач БД – програма або людина, яка звертається до БД на мові маніпулювання даними.

Запит – процес поводження користувача до БД з метою введення, отримання або зміни інформації в БД.

Транзакція – порядок модифікації даних в БД, яка переводить БД з одного несуперечливого стану в інший несуперечливий стан.

Логічна структура БД – визначення БД на фізично незалежному рівні, найближче відповідає концептуальній моделі БД.

Топологія БД = Структура розподіленої БД – схема розподілу фізичної БД по мережі.

Локальна автономність – означає, що інформація локальної БД і пов'язані з нею визначення даних належать локальному власнику і їм управляються.

Віддалений запит – запит, який виконується з використанням модемного зв'язку.

Можливість реалізації вилученої транзакції – обробка однієї транзакції, що складається з безлічі SQL-запитів на одному віддаленому вузлі.

Підтримка розподіленої транзакції допускає обробку транзакції, що складається з декількох запитів SQL, які виконуються на декількох вузлах мережі (віддалених або локальних), але кожен запит в цьому випадку обробляється тільки на одному вузлі, тобто запити не є розподіленими. При обробці однієї розподіленої транзакції різні локальні запити можуть оброблятися в різних вузлах мережі.

Розподілений запит – запит, при обробці якого використовуються дані з БД, розташовані в різних вузлах мережі.

Системи розподіленої обробки даних в основному пов'язані з першим поколінням БД, які будувалися на мультипрограмних операційних системах і використовували централізоване зберігання БД на пристроях зовнішньої пам'яті центральної ЕОМ і термінальний розрахований на багато користувачів режим доступу до неї. При цьому для користувача термінали не мали власних ресурсів, тобто процесорів і пам'яті, які могли б використовуватися для зберігання і обробки даних. Першою повністю реляційною системою, що працює в режимі багатьох користувачів, була СУБД SYSTEM R, розроблена фірмою IBM, саме в ній були реалізовані як мову маніпулювання даними SQL, так і основні принципи синхронізації, що застосовуються при розподіленій обробці даних, які до сих пір є базисними практично у всіх комерційних СУБД.

Загальна тенденція руху від окремих mainframe-систем до відкритих розподілених систем, що об'єднує комп'ютери середнього класу, отримала назву DownSizing. Цей процес зробив величезний вплив на розвиток архітектур СУБД і поставив перед їх розробниками ряд складних завдань. Головна проблема полягала в технологічній складності переходу від централізованого управління даними на одному комп'ютері і СУБД, яка використала власні моделі, формати представлення даних і мови доступу до даних і т. д. До розподіленої обробки даних в неоднорідному обчислювальному середовищі, що складається зі з'єднаних в глобальну мережу комп'ютерів різних моделей і виробників.

Водночас відбувався зустрічний процес – UpSizing. Бурхливий розвиток персональних комп'ютерів, поява локальних мереж також зробили серйозний вплив на еволюцію СУБД. Високі темпи зростання продуктивності і функціональних можливостей PC привернули увагу розробників професійних СУБД, що призвело до їх активного поширення на платформі настільних систем.

Сьогодні взяла гору тенденція створення інформаційних систем на такій платформі, яка точно відповідала б її масштабам і завданням. Вона отримала назву RightSizing (розміщення рівно в той розмір, який необхідний).

Однак і в даний час великі ЕОМ зберігаються і співіснують з сучасними відкритими системами. Причина цього проста – свого часу в апаратне і програмне забезпечення великих ЕОМ були вкладені величезні кошти: в результаті багато хто продовжує їх використовувати, не дивлячись на морально застарілу архітектуру. У той же час перенесення даних і програм з великих ЕОМ на комп'ютери нового покоління сам по собі представляє складну технічну проблему і вимагає значних витрат.

8.1 Моделі «клієнт-сервер» в технології баз даних

Обчислювальна модель «клієнт-сервер» початково пов'язана з парадигмою відкритих систем, яка з'явилася в 90-х роках і швидко еволюціонувала. Сам термін «клієнт-сервер» початково застосовувався до архітектури програмного забезпечення, яке описувало розподіл процесу виконання за принципом взаємодії двох програмних процесів, один з яких в цій моделі називався «клієнтом», а інший – «сервером». Клієнтський процес запитував деякі послуги, а серверний процес забезпечував їх виконання. При цьому передбачалося, що один серверний процес може обслужити безліч клієнтських процесів.

Раніше додаток (для користувача програма) не поділявся на частини, він виконувався деяким монолітним блоком. Але виникла ідея більш раціонального використання ресурсів мережі. Дійсно, при монолітному виконанні використовуються ресурси тільки одного комп'ютера, а інші комп'ютери в мережі розглядаються як термінали. Але тепер, на відміну від епохи main-фреймів, все комп'ютери в мережі володіють власними ресурсами, і розумно так розподілити навантаження на них, щоб максимально використовувати їх ресурси.

Основний принцип технології «клієнт-сервер» стосовно до технології баз даних полягає в поділі функцій стандартного інтерактивного додатку на 5 груп, що мають різну природу:

− функції введення і відображення даних (Presentation Logic);

− прикладні функції, що визначають основні алгоритми вирішення завдань програми (Business Logic);

− функції обробки даних всередині програми (Database Logic);

− функції управління інформаційними ресурсами (Database Manager System);

− службові функції, які відіграють роль зв'язок між функціями перших чотирьох груп.

Презентаційна логіка (Presentation Logic) як частина програми визначається тим, що користувач бачить на своєму екрані, коли працює додаток. Сюди відносяться всі інтерфейсні екранні форми, які користувач бачить або заповнює в ході роботи програми, до цієї ж частини відноситься все те, що виводиться користувачеві на екран як результати вирішення деяких проміжних завдань або як довідкова інформація, тому основні завдання презентаційної логіки такі:

– формування екранних зображень;
– читання і запис в екранні форми інформації;
– управління екраном;
– обробка рухів миші і натискання клавіш клавіатури.

Деякі можливості для організації презентаційної логіки додатків надає знакі-орієнтований інтерфейс, що задається моделями CICS (Customer Control Information System) і IMS / DC фірми IBM і моделлю TSO (Time Sharing Option) для централізованої main-фреймовой архітектури. Модель GUI – графічного призначеного для користувача інтерфейсу, підтримується в операційних середовищах Microsoft's Windows, Windows NT, в OS / 2 Presentation Manager, X-Windows і OSF / Motif.

Бізнес-логіка, або логіка власне додатків (Business processing Logic), – це частина коду програми, яка визначає власне алгоритми вирішення конкретних завдань програми. Зазвичай цей код пишеться з використанням різних мов програмування, таких як C, C ++, Cobol, SmallTalk, Visual-Basic.

Логіка обробки даних (Data manipulation Logic) – це частина коду програми, яка пов'язана з обробкою даних всередині програми. Даними управляє власне СУБД (DBMS). Для забезпечення доступу до даних використовують мову запитів і засоби маніпулювання даними стандартної мови SQL.

Зазвичай оператори мови SQL вбудовуються в мови 3-го або 4-го покоління (3GL, 4GL), які використовуються для написання коду програми.

Процесор управління даними (Database Manager System Processing) – це власне СУБД, яка забезпечує зберігання і управління базами даних. В ідеалі функції СУБД повинні бути приховані від бізнес-логіки додатка, однак для розгляду архітектури додатку нам треба їх виділити в окрему частину програми.

У централізованій архітектурі (Host-based processing) ці частини програми розташовуються в єдиному середовищі і комбінуються усередині однієї програми, що виконується.

У децентралізованій архітектурі ці завдання можуть бути по-різному розподілені між серверним і клієнтським процесами. Залежно від характеру розподілу можна виділити наступні моделі розподілів:
– розподілена презентація (Distribution presentation, DP);
– віддалена презентація (Remote Presentation, RP);
– розподілена бізнес-логіка (Remote business logic, RBL);
– розподілене управління даними (Distributed data management, DDM);
– віддалене управління даними (Remote data management, RDA).

Ця умовна класифікація показує, як можуть бути розподілені окремі завдання між серверним і клієнтськими процесами. У цій класифікації відсутня реалізація віддаленої бізнес-логіки. Дійсно, вважається, що вона не може бути видалена сама по собі повністю. Вважається, що вона може бути розподілена між різними процесами, які в загалі-то можуть виконуватися на різних платформах, але повинні коректно кооперуватися (взаємодіяти) один з одним.

Дворівневі моделі. Дворівнева модель фактично є результатом розподілу п'яти зазначених функцій між двома процесами, які виконуються на двох платформах: на клієнті і на сервері. У чистому вигляді майже ніяка модель не існує, проте розглянемо найбільш характерні особливості кожної дворівневої моделі.

Модель віддаленого управління даними. Модель файлового сервера. Модель віддаленого управління даними також називається моделлю файлового сервера (File Server, FS). У цій моделі презентаційна логіка і бізнес-логіка розташовуються на клієнті. На сервері розташовуються файли з даними і підтримується доступ до файлів. Функції управління інформаційними ресурсами в цій моделі знаходяться на клієнті.

У цій моделі файли бази даних зберігаються на сервері, клієнт звертається до сервера з файловими командами, а механізм управління всіма інформаційними ресурсами, власне база мета-даних, знаходиться на клієнті.

Переваги цієї моделі в тому, що ми вже маємо поділ монопольного додатки на два взаємодіючих процесу. При цьому сервер (серверний процес) може обслуговувати безліч клієнтів, які звертаються до нього з запитами. Власне СУБД повинна перебувати в цій моделі на клієнті.

Алгоритм виконання запиту клієнта наступний. Формується запит клієнта. СУБД переводить цей запит в послідовність файлових команд. Кожна файлова команда викликає перекачування блоку інформації на клієнта, далі на клієнті СУБД аналізує отриману інформацію, і якщо в отриманому блоці не міститься відповідь на запит, то приймається рішення про перекачку наступного блоку інформації і т. д.

Перекачування інформації з сервера на клієнт проводиться до тих пір, поки не буде отримана відповідь на запит клієнта.

Недоліками цієї моделі варто вважати:

– високий мережевий трафік, який пов'язаний з передачею по мережі безлічі блоків і файлів, необхідних додатку;

– вузький спектр операцій маніпулювання з даними, який визначається тільки файловими командами;

– відсутність адекватних засобів безпеки доступу до даних (захист тільки на рівні файлової системи).

Модель віддаленого доступу до даних. У моделі віддаленого доступу (Remote Data Access, RDA) база даних зберігається на сервері. На сервері же знаходиться ядро СУБД. На клієнті розташовується презентаційна логіка і бізнес-логіка програми. Клієнт звертається до сервера з запитами на мові SQL.

Переваги цієї моделі:

− перенесення компонента уявлення і прикладного компонента на клієнтський комп'ютер істотно розвантажив сервер БД, зводячи до мінімуму загальне число процесів в операційній системі;

− сервер БД звільняється від невластивих йому функцій; процесор або процесори сервера цілком завантажуються операціями обробки даних, запитів і транзакцій. (Це стає можливим, якщо відмовитися від терміналів, які не мають ресурсів, і замінити їх комп'ютерами, які виконують роль клієнтських станцій, які володіють власними локальними обчислювальними ресурсами);

− різко зменшується завантаження мережі, так як по ній від клієнтів до сервера передаються не запити на вхід-видобуток в файлової термінології, а запити на SQL, і їх обсяг істотно менше. У відповідь на запити клієнт отримує тільки дані, релевантні запиту, а не блоки файлів, як в FS-моделі.

Основна перевага RDA-моделі – уніфікація інтерфейсу «клієнт-сервер», стандартом при спілкуванні програми-клієнта і сервера стає мова SQL.

Недоліки моделі:

− запити на мові SQL при інтенсивній роботі клієнтських додатків можуть істотно завантажити мережу;

− так як в цій моделі на клієнті розташовується і презентаційна логіка, і бізнес-логіка програми, то при повторенні аналогічних функцій в різних додатках код відповідної бізнес-логіки повинен бути повторений для кожного клієнтського додатку. Це викликає зайве дублювання коду додатків;

− сервер в цій моделі грає пасивну роль, тому функції управління інформаційними ресурсами повинні виконуватися на клієнті. Дійсно, наприклад, якщо нам необхідно виконувати контроль страхових запасів товарів на складі, то кожний додаток, що пов'язаний зі зміною стану складу, після виконання операцій модифікації даних, що імітують продаж або видалення товару зі складу, має виконувати перевірку на обсяг залишку, і в разі , якщо він менше страхового запасу, формувати відповідну заявку на поставку необхідного товару. Це ускладнює клієнтську програму, з одного боку, а з іншого – може викликати необґрунтоване замовлення додаткових товарів кількома додатками.

Модель сервера баз даних. Для того щоб позбутися від недоліків моделі віддаленого доступу, повинні бути дотримані такі умови:

1. Необхідно, щоб БД в кожен момент відображала поточний стан предметної області, який визначається не тільки власне даними, але і зв'язками між об'єктами даних. Тобто дані, які зберігаються в БД, в кожен момент часу не повинні суперечити одна одній.

2. БД повинна відображати деякі правила предметної області, закони, за якими вона функціонує («business rules»). Наприклад, завод може нормально працювати тільки в тому випадку, якщо на складі є деякий достатній запас (страховий запас) деталей певної номенклатури, деталь може бути запущена у виробництво тільки в тому випадку, якщо на складі є в наявності достатньо матеріалу для її виготовлення, і т. д.

4. Необхідний постійний контроль за станом БД, відстеження всіх змін і адекватна реакція на них. Наприклад, при досягненні деяких вимірюваних параметрів критичного значення має відбутися відключення певної апаратури, при зменшенні товарного запасу нижче допустимої норми повинна бути сформована заявка конкретного постачальника на поставку відповідного товару.

5. Необхідно, щоб виникнення деякої ситуації в БД чітко і оперативно впливало на хід виконання прикладної задачі.

6. Однією з найважливіших проблем СУБД є контроль типів даних. На даний момент СУБД контролює синтаксично тільки стандартно-допустимі типи даних, тобто такі, які визначені в DDL (data definition language) – мові опису даних, який є частиною SQL. Однак в реальних предметних областях у нас діють дані, які несуть в собі ще й семантичну складову, наприклад, це координати об'єктів або одиниці різних метрик, наприклад робочий тиждень на відміну від реалії має відразу після п'ятниці понеділок.

Цю модель підтримують більшість сучасних СУБД: Informix, Ingres, Sybase, Oracle, MS SQL Server. Основу даної моделі становить механізм збережених процедур як засіб програмування SQL-сервера, механізм тригерів як механізм відстежування поточного стану інформаційного сховища і механізм обмежень на призначені для користувача типи даних, який іноді називається механізмом підтримки доменної структури.

У цій моделі бізнес-логіка розділена між клієнтом і сервером. На сервері бізнес-логіка реалізована у вигляді збережених процедур – спеціальних програмних модулів, які зберігаються в БД і управляються безпосередньо СУБД. Клієнтський додаток звертається до сервера з командою запуску збереженої процедури, а сервер виконує цю процедуру і реєструє всі зміни в БД, які в ній

передбачені. Сервер повертає клієнту дані, релевантні його запиту, які потрібні клієнту або для виведення на екран, або для виконання частини бізнес-логіки, яка розташована на клієнті. Трафік обміну інформацією між клієнтом і сервером різко зменшується.

Централізований контроль в моделі сервера баз даних виконується з використанням механізму тригерів. Тригери також є частиною БД.

Термін «тригер» взято з електроніки і семантично дуже точно характеризує механізм відстеження спеціальних подій, які пов'язані зі станом БД. Тригер в БД є як би деяким тумблером, який спрацьовує при виникненні певної події в БД. Ядро СУБД проводить моніторинг всіх подій, які викликають створені і описані тригери в БД, і при виникненні відповідної події сервер запускає відповідний тригер. Кожен тригер являє собою також деяку програму, яка виконується над базою даних. Тригери можуть викликати збережені процедури.

Механізм використання тригерів передбачає, що при спрацюванні одного тригера можуть виникнути події, які викличуть спрацювання інших тригерів. Цей потужний інструмент вимагає тонкого і узгодженого застосування, щоб не вийшов нескінченний цикл спрацювання тригерів.

У даній моделі сервер є активним, тому що не тільки клієнт, але і сам сервер, використовуючи механізм тригерів, може бути ініціатором обробки даних в БД.

І збережені процедури, і тригери зберігаються в словнику БД, вони можуть бути використані декількома клієнтами, що істотно зменшує дублювання алгоритмів обробки даних в різних клієнтських додатках.

Для написання збережених процедур і тригерів використовується розширення стандартної мови SQL, так звана вбудована SQL.

Недоліком даної моделі є дуже велике завантаження сервера. Дійсно, сервер обслуговує безліч клієнтів і виконує наступні функції:

– здійснює моніторинг подій, пов'язаних з описаними тригерами;

– забезпечує автоматичне спрацювання тригерів при виникненні пов'язаних з ними подій;

– забезпечує виконання внутрішньої програми кожного тригера;

– запускає збережені процедури по запитах користувачів;

– запускає збережені процедури з тригерів;

– повертає необхідні дані клієнта;

– забезпечує всі функції СУБД: доступ до даних, контроль і підтримку цілісності даних в БД, контроль доступу, забезпечення коректної паралельної роботи всіх користувачів з єдиною БД.

Якщо на сервер перекладено більшу частину бізнес-логіки додатків, то вимоги до клієнтів в цій моделі різко зменшуються. Іноді таку модель називають моделлю з «тонким клієнтом», на відміну від попередніх моделей, де на клієнта покладалися набагато серйозніші завдання. Ці моделі називаються моделями з «товстим клієнтом».

Для розвантаження сервера була запропонована трирівнева модель.

Модель сервера додатків. Ця модель є розширенням дворівневої моделі і в ній вводиться додатковий проміжний рівень між клієнтом і сервером. Цей проміжний рівень містить один або кілька серверів додатків.

У цій моделі компоненти програми діляться між трьома виконавцями:

– *клієнт* забезпечує логіку уявлення, включаючи графічний користувальницький інтерфейс, локальні редактори; клієнт може запускати локальний код додатка клієнта, який може містити звернення до локальної БД, розташованої на комп'ютері-клієнті. Клієнт виконує комунікаційні функції front-end частини програми, які забезпечують доступ клієнту в локальну або глобальну мережу. Додатково реалізація взаємодії між клієнтом і сервером може включати в себе управління розподіленими транзакціями, що відповідає тим випадкам, коли клієнт також є клієнтом менеджера розподілених транзакцій;

– *сервери додатків* складають новий проміжний рівень архітектури. Вони спроєктовані як виконання загальних незавантажуваних функцій для клієнтів. Сервери додатків підтримують функції клієнтів як частин взаємодіючих робочих груп, підтримують мережеву доменну операційну середу, зберігають і виконують найбільш загальні правила бізнес-логіки, підтримують каталоги з даними, забезпечують обмін повідомленнями і підтримку запитів, особливо в розподілених транзакцій;

– *сервери баз даних* в цій моделі займаються виключно функціями СУБД: забезпечують функції створення та ведення БД, підтримують цілісність реляційної БД, забезпечують функції сховищ даних (warehouse services). Крім того, на них покладаються функції створення резервних копій БД і відновлення БД після збоїв, управління виконанням транзакцій і підтримки застарілих (успадкованих) додатків (legacy application).

Ця модель більш гнучка порівняно з дворівневою моделлю. Найбільш помітні переваги моделі сервера додатків в тих випадках, коли клієнти виконують складні аналітичні розрахунки над базою даних, які відносяться до області OLAP-додатків. (On-line analytical processing.) У цій моделі велика частина бізнес-логіки клієнта ізольована від можливостей вбудованого SQL, реалізованого в конкретній СУБД, і може бути виконана на стандартних мовах

програмування, таких як C, C ++, SmallTalk, Cobol. Це підвищує переносимість системи, її масштабованість.

Функції проміжних серверів можуть бути в цій моделі розподілені в рамках глобальних транзакцій шляхом підтримки XA-протоколу (X / Open transaction interface protocol), який підтримується більшістю постачальників СУБД.

Моделі серверів баз даних. У період створення перших СУБД технологія «клієнт-сервер» тільки зароджувалася. Тому спочатку в архітектурі систем не було адекватного механізму організації взаємодії процесів типу «клієнт» і процесів типу «сервер». А в нинішніх СУБД він є фактично основним і від ефективності його реалізації залежить ефективність роботи системи в цілому.

Розглянемо еволюцію типів організації подібних механізмів. В основному цей механізм визначається структурою реалізації серверних процесів, і часто він називається архітектурою сервера баз даних.

Спочатку, як ми вже відзначали, існувала модель, коли управління даними (функція сервера) і взаємодія з користувачем були поєднані в одній програмі. Це можна назвати нульовим етапом розвитку серверів БД.

Потім функції управління даними були виділені в самостійну групу – сервер, проте модель взаємодії користувача з сервером відповідала парадигмі «один-до-одного», тобто сервер обслуговував запити тільки одного користувача (клієнта), і для обслуговування декількох клієнтів потрібно було запустити еквівалентну кількість серверів.

Виділення сервера в окрему програму було революційним кроком, який дозволив, зокрема, помістити сервер на одну машину, а програмний інтерфейс з користувачем – на іншу, здійснюючи взаємодію між ними по мережі. Однак необхідність запуску великого числа серверів для обслуговування безлічі користувачів сильно обмежувала можливості такої системи.

Для обслуговування великої кількості клієнтів на сервері повинно бути запущено велику кількість одночасно працюючих серверних процесів, а це різко підвищувало вимоги до ресурсів ЕОМ, на якій запускалися усі серверні процеси. Крім того, кожен серверний процес в цій моделі запускався як незалежний, тому якщо один клієнт сформував запит, який був тільки що виконаний іншим серверним процесом для іншого клієнта, то запит проте виконувався повторно. У такій моделі вельми складно забезпечити взаємодію серверних процесів. Ця модель найпростіша, і історично вона з’явилася першою.

Проблеми, що виникають в моделі «один-до-одного», вирішуються в архітектурі «систем з виділеним сервером», який здатний обробляти запити від багатьох клієнтів. Сервер єдиний володіє монополією на керування даними і

взаємодіє одночасно з багатьма клієнтами. Логічно кожен клієнт пов'язаний з сервером окремої ниткою («thread»), або потоком, за яким пересилаються запити. Така архітектура отримала назву багатопотокової односерверної («multi-threaded»).

Вона дозволяє значно зменшити навантаження на операційну систему, яка виникає при роботі великої кількості користувачів («trashing»).

Крім того, можливість взаємодії з одним сервером багатьох клієнтів дозволяє в повній мірі використовувати колективні об'єкти (починаючи з відкритих файлів і закінчуючи даними із системних каталогів), що значно зменшує потреби в пам'яті і загальне число процесів операційної системи. Наприклад, системою з архітектурою «один-до-одного» буде створено 100 копій процесів СУБД для 100 користувачів, тоді як системі з багатопотоковою архітектурою для цього знадобиться тільки один серверний процес.

Однак таке рішення має свої недоліки. Так як сервер може виконуватися тільки на одному процесорі, виникає природне обмеження на застосування СУБД для мультипроцесорних платформ. Якщо комп'ютер має, наприклад, чотири процесори, то СУБД з одним сервером використовують тільки один з них, не завантажуючи залишені три.

У деяких системах ця проблема вирішується введенням проміжного диспетчера. Подібна архітектура називається архітектурою віртуального сервера («virtual server»).

У цій архітектурі клієнти підключаються не до реального сервера, а до проміжної ланки, названу диспетчером, який виконує тільки функції диспетчеризації запитів до актуальних серверів. У цьому випадку немає обмежень на використання багатопроцесорних платформ. Кількість актуальних серверів може бути погоджено з кількістю процесорів в системі.

Однак і ця архітектура не позбавлена недоліків, тому що тут в систему додається новий шар, який розміщується між клієнтом і сервером, що збільшує витрату ресурсів на підтримку балансу завантаження актуальних серверів («load balancing») і обмежує можливості управління взаємодією «клієнт-сервер». По-перше, стає неможливим направити запит від конкретного клієнта конкретному серверу, по-друге, сервери стають рівноправними – немає можливості встановлювати пріоритети для обслуговування запитів.

Подібна організація взаємодії клієнт-сервер може розглядатися як аналог банку, де є кілька вікон касирів, і спеціальний банківський службовець – адміністратор залу (диспетчер) направляє кожного нового відвідувача (клієнта) до вільного касиру (актуальному сервера). Система працює нормально, поки всі

відвідувачі рівноправні (мають рівні пріоритети), проте варто лише з’явитися відвідувачам з вищим пріоритетом, які повинні обслуговуватися в спеціальному вікні, як виникають проблеми. Облік пріоритету клієнтів особливо важливий в системах оперативної обробки транзакцій, однак саме цю можливість не може надати архітектура систем з диспетчеризацією.

Сучасне рішення проблеми СУБД для мультипроцесорних платформ полягає в можливості запуску кількох серверів бази даних, в тому числі і на різних процесорах. При цьому кожен із серверів повинен бути багатопотоковим. Якщо ці дві умови виконані, то є підстави говорити про багатопотокову архітектуру з декількома серверами.

Вона також може бути названа багатонитковою мультисерверною архітектурою. Ця архітектура пов’язана з питаннями розпаралелювання виконання одного користувальницького запиту декількома серверними процесами.

Існує кілька можливостей розпаралелювання виконання запиту. В цьому випадку для користувача запит розбивається на ряд підзапитів, які можуть виконуватися паралельно, а результати їх виконання потім об’єднуються в загальний результат виконання запиту. Тоді для забезпечення оперативності виконання запитів їх підзапити можуть бути спрямовані окремим серверним процесам, а потім отримані результати об’єднані в загальний результат. В даному випадку серверні процеси не є незалежними процесами, такими, що розглядалися раніше. Ці серверні процеси прийнято називати нитками (treads), і управління нитками безлічі запитів користувачів вимагає додаткових витрат від СУБД, проте при оперативній обробці інформації в сховищах даних такий підхід найбільш перспективний.

Розглядають кілька шляхів розпаралелювання запитів.

Горизонтальний паралелізм. Цей паралелізм виникає тоді, коли зберігається в БД інформація розподіляється по декількох фізичних пристроях зберігання – декількох дисках. При цьому інформація з одного відношення розбивається на частини по горизонталі. Цей вид паралелізму іноді називають розпаралелюванням або сегментацією даних. І паралельність тут досягається шляхом виконання однакових операцій, наприклад фільтрації, над різними фізичними збереженими даними. Ці операції можуть виконуватися паралельно різними процесами, вони незалежні. Результат виконання цілого запиту складається з результатів виконання окремих операцій.

Час виконання такого запиту при відповідній сегментації даних істотно менше, ніж час виконання цього ж запиту традиційними способами одним процесом.

Вертикальний паралелізм. Цей паралелізм досягається конвеєрним виконанням операцій, які становлять запит користувача. Цей підхід вимагає серйозного ускладнення в моделі виконання реляційних операцій ядром СУБД. Він передбачає, що ядро СУБД може зробити декомпозицію запиту, базуючись на його функціональних компонентах, і при цьому ряд підзапитів може виконуватися паралельно, з мінімальною зв'язком між окремими кроками виконання запиту.

Дійсно, якщо ми розглянемо, наприклад, послідовність операцій реляційної алгебри:

R5 = R1 [A, C]

R6 = R2 [A, B, D]

R7 = R5 [A> 128]

R8 = R5 [A] R6,

то операції першу і третю можна об'єднати і виконати паралельно з операцією два, а потім виконати над результатами останню четверту операцію.

Загальний час виконання подібного запиту, звичайно, буде істотно менше, ніж при традиційному способі виконання послідовності з чотирьох операцій.

Найбільш активно застосовуються види паралелізму в OLAP-додатках, де ці методи дозволяють істотно скоротити час виконання складних запитів над дуже великими обсягами даних.

СПИСОК РЕКОМЕНДОВАНИХ ДЖЕРЕЛ

1. Берко А. Ю. Системи баз даних та знань : підручник / А. Ю. Берко, О. М. Верес, В. В. Пасічник. – Львів : Магнолія – 2006, 2015. – Книга 1. Організація баз даних та знань. – 440 с.

2. Павлиш В. А. Основи інформаційних технологій і систем : підручник / В. А. Павлиш, Л. К. Гліненко, Н. Б. Шаховська. – Львів : Львівська політехніка, 2018. – 619 с.

3. Шаховська Н. Б. Системи штучного інтелекту : навч. посібник / Н. Б. Шаховська, Р. М. Камінський, О. Б. Вовк. – Львів : Львівська політехніка, 2018. – 391 с.

4. Костріков С. В. Інформаційні технології в БД. Навчально-методичний посібник / С. Костріков. – Харків : РВВ ХНУ, 2015. – 56 с.

5. Основы использования SQL в серверных системах : учеб. пособие / Власюк А. Г. и др. – Киев : Компринт, 2017. – 125 с.

6. Цеслів О. В. Технологія проєктування та адміністрування баз даних і сховищ даних : навч. посібник / О. В. Цеслів, А. С. Коломієць ; Нац. техн. ун-т України «Київ. політехн. ін-т ім. Ігоря Сікорського». – Київ : КПІ ім. І. Сікорського : Політехніка, 2017. – 281 с.

Наукове видання

КОСТЕНКО Олександр Борисович,
ГАВРИЛЕНКО Ірина Олександрівна

ОРГАНІЗАЦІЯ БАЗ ДАНИХ ТА ЗНАНЬ

КОНСПЕКТ ЛЕКЦІЙ

(для студентів денної і заочної форм навчання першого (бакалаврського) рівня вищої освіти за спеціальністю 126 – Інформаційні системи та технології)

Відповідальний за випуск *О. Б. Костенко*
За авторською редакцією
Комп'ютерний набір і верстання *І. О. Гавриленко*

План 2020, поз. 134Л

Підп. до друку 08.06.2021. Формат 60 × 84/16.
Ум. друк. арк. 5,3.

Видавець і виготовлювач:
Харківський національний університет
міського господарства імені О. М. Бекетова,
вул. Маршала Бажанова, 17, Харків, 61002.
Електронна адреса: office@kname.edu.ua
Свідоцтво суб'єкта видавничої справи:
ДК № 5328 від 11.04.2017.