Міністерство освіти і науки України
Національний університет водного господарства та
природокористування

Н. О. Харів

БАЗИ ДАНИХ ТА ІНФОРМАЦІЙНІ СИСТЕМИ

Навчальний посібник

Рівне-2018

УДК 681.3 (075.8)
Х 20

Рекомендовано вченою радою Національного університету водного господарства та природокористування (протокол №2 від 23 лютого 2018 р.)

Рецензенти:
Тулашвілі Ю. Й., доктор педагогічних наук, професор Національного університету водного господарства та природокористування (м. Рівне);
Бомба А. Я., доктор технічних наук, професор Рівненського державного гуманітарного університету.

Харів Н. О.
Х 20 Бази даних та інформаційні системи: навчальний посібник / Н. О. Харів. – Рівне : НУВГП, 2018. – 127 с.

Навчальний посібник містить теорію проектування баз реляційних баз даних, опис та приклади використання мови SQL, а також завдання для лабораторних робот.

Посібник призначено для студентів спеціальностей 113 „Прикладна математика”, 122 „Комп’ютерні науки”, 121 „Інженерія програмного забезпечення”, а також може використовуватись для самостійного вивчення студентами різних галузей знань, учнями шкіл, слухачами комп’ютерних курсів.

УДК 681.3(075.8)

ЗМІСТ

ВСТУП

Нині кожен фахівець, який працює у галузях, пов'язаних з комп'ютерними технологіями, повинен знати принципи проектування структури бази даних, вміти грамотно та ефективно працювати із системами керування базами даних, створювати сучасні програмні продукти, що використовуються в процесах інформаційного обслуговування виробничо-господарської діяльності.

Навчальний посібник складається з трьох розділів. У першому розділі розглядаються теоретичні основи проектування баз даних: архітектура баз даних, їх моделювання, реляційна теорія баз даних.

Другий розділ висвітлює питання щодо практичної реалізації баз даних, зокрема створення і функціонування бази даних у клієнт-серверній програмі *Firebird – IBExpert*. Особливу увагу приділено вивченню і використанню команд мови SQL у роботі з базою даних.

У кінці кожного розділу наведено приклади контрольних запитань для самоперевірки.

Складовими третього розділу є лабораторні роботи, які рекомендується виконати для отримання практичних умінь і навиків у побудові і користуванні бази даних на тему „Університет".

Також, навчальний посібник містить словник термінів, що стосуються баз даних, і список команд SQL, синтаксис яких можна легко знайти за вказаною сторінкою.

Розділ І. Теоретичні основи бази даних

1. Вступ в теорію баз даних

1.1. Основні поняття баз даних та інформаційні системи

Сучасні комп'ютерні технології дозволяють вирішувати поставлені завдання в різних галузях людської діяльності. Практично в усіх випадках виникає потреба зберігання, накопичення, опрацювання певних обсягів інформації.

Одним із способів вирішення даних проблем є обробка даних з використанням баз даних та інформаційних системах.

Сучасна ***комп'ютерна інформаційна система*** – це сукупність апаратних, програмних, мережевих та інформаційних ресурсів, які в процесі свого сумісного функціонування забезпечують розв'язок кола задач з отримання, накопичення, збереження, передачі, перетворення, відображення інформації в певній галузі людської діяльності.

В складі інформаційних систем розрізняють такі складові:

- ***апаратне або технічне забезпечення*** (*hardware*) – набір технічних засобів, пристроїв та елементів, що утворюють фізичне середовище інформаційної системи та забезпечують технічну реалізацію її функцій;
- ***програмне забезпечення*** (*software*) – множина програмних засобів для реалізації методів, алгоритмів, сервісів, які підтримують роботу інформаційних систем;
- ***мережеве забезпечення*** (*netware*) – це специфічна категорія як апаратних, програмних та інших засобів, які забезпечують функціонування інформаційних систем у мережевих середовищах;
- ***інформаційне забезпечення*** (*data ware*) – множина всіх значень, величин, показників, які застосовуються інформаційною системою у процесах її застосування.

База даних (БД) – це будь-яка пов'язана між собою за певними ознаками інформація, що зберігається і організовується певним чином, як правило у вигляді таблиць.

Система керування базами даних (СКБД) – це програмне забезпечення, призначене для створення, ведення та сумісного використання бази даних багатьма користувачами.

Основна функція СКБД – це надання можливості користувачам працювати з базою даних, використовуючи зручний інтерфейс, не вдаючись у деталі на рівні апаратури.

1.2. Класифікація СКБД

СКБД класифікують за різними ознаками:

1) За ступенем розподіленості:

Централізовані або ***локальні***. Всі частини бази даних зберігаються на одному комп'ютері.

Розподілені. Частини бази даних можуть розміщуватись на двох або більше комп'ютерах.

2) За вмістом: історичні, наукові, географічні, мультимедійні тощо.

3) За архітектурою:

Файл-сервер. В мережі виділяється один комп'ютер (сервер файлів), на якому зберігається централізовано база даних. На кожному комп'ютері користувача встановлена СКБД для роботи з цією базою даних. Доступ до бази даних здійснюється через локальну мережу. Прикладами є *Microsoft Access*, *FoxPro*, *Paradox*, *dBase*.

Клієнт-сервер. Систему керування базою даних можна розглядати як систему, що складається з двох частин: сервера і набору клієнтів. ***Сервером*** називається сама СКБД разом із базою даних, що встановлена на одному центральному комп'ютері. ***Клієнтами*** є програми, що надають можливість роботи з базою даних. Прикладом є *Firebird – IBExpert*, *Microsoft SQL Server*.

4) За моделлю даних: ієрархічні, мережеві, реляційні, об'єктно-орієнтовані.

5) За сферою застосування: документографічні й документальні (органи влади та управління); бази даних з

промислової, будівельної та сільськогосподарської продукції; бази даних в економіці (статистична, кредитно-фінансова, зовнішньоторговельна); бази даних транспортних систем; бази і банки наукових даних; бази даних у галузі культури і мистецтва; лінгвістичні бази даних тощо.

1.3. Архітектура баз даних

Архітектура баз даних розроблена дослідницькою організацією ANSI/SPARC (*American National Standards Institute / Standards Planning and Requirements Committee* – Американський національний інститут стандартів / Комітет планування стандартів) в 1975 році.

Виділяють три рівня архітектури, на кожному з яких описується відповідна модель бази даних:

1) ***Внутрішній рівень*** (або *фізичний*) – рівень системного програмування, найбільш наближений до збереження інформації на фізичних пристроях. Він описує, яким саме чином розміщуються там дані.
2) ***Зовнішній рівень*** (або *користувацький*) – зв'язаний із способами представлення даних безпосередньо для користувача за допомогою різних складових. Користувач може маніпулювати даними в СКБД за допомогою спеціальної мови.
3) ***Концептуальний рівень*** (або *логічний*) – рівень задач розробника та адміністратора баз даних, описує дані і їх зв'язки, використовуючи певну модель даних. Даний рівень є перехідним від внутрішнього до зовнішнього. Це представлення всієї інформації бази даних в більш абстрактній формі порівняно з внутрішнім рівнем, і в більш цілій формі порівняно із зовнішнім, де конкретні дані виділяються певному користувачу.

Концептуальна модель передбачає два етапи реалізації: інфологічний та даталогічний.

Інфологічна модель – це опис майбутньої бази даних, що відображає інформацію про предметну область у вигляді, незалежному від засобів і технологій реалізації проекту за

допомогою природної мови, формул, графіків, діаграм, таблиць та інших засобів, зрозумілих як розробникам БД, так і звичайним користувачам.

Даталогічна модель – опис структури в термінах конкретної СУБД чи технології, які вибираються для реалізації бази даних на основі інфологічної моделі.

Дана архітектура дозволяє забезпечити логічну незалежність при роботі з даними між рівнями 2 і 3 і фізичну – між рівнями 1 і 3. Логічна незалежність дозволяє змінювати будь-яку складову бази даних без коригування іншої. Фізична незалежність дозволяє можливість перенесення бази даних з одних носіїв на інші.

1.4. Основні види моделей даних

Модель даних – спосіб подання даних у вигляді певної структури та засоби маніпулювання ними.

1. ***Ієрархічна модель даних***. Представляє собою сукупність елементів, зв'язаних між собою за певними правилами. Об'єкти ієрархічної моделі утворюють орієнтований граф (типу дерево). Основними поняттями є рівень, вузол і зв'язок.

Властивості:

- Кожен вузол, що не є вершиною, зв'язаний з одним вузлом вищого рівня.
- Існує тільки одна вершина.
- Зв'язок не може бути встановлений між вузлами через рівень.
- Зв'язок між вузлами одного рівня не визначається.

Приклади:

1) Файлова структура організації даних (комп'ютер, диски, папки, файли).
2) Структура університету (вуз, факультети, спеціальності, групи, студенти) (рис.1.1).

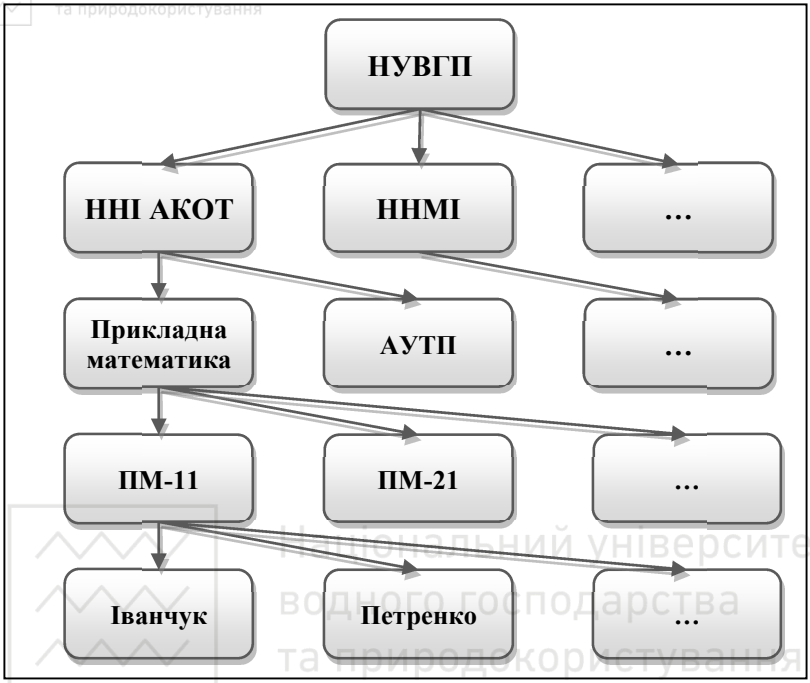


Рис.1.1. Ієрархічна структура даних

Переваги:

1) Простота організації.
2) Невеликий об’єм пам’яті.

Недоліки:

1) Відсутність універсальності. Не всяку інформацію можна виразити через таку модель.
2) Доступ до даних виключно через вершину зверху вниз.

2. **Мережева модель даних.** Елементами цієї моделі є рівень, вузол і зв’язок. Відмінність від ієрархічної полягає в тому, що елемент певного рівня може бути зв’язаний з будь-якою кількістю елементів сусіднього рівня і не існує підпорядкованості рівнів один одному.

Властивості:

- Зв'язок не може бути встановлений між вузлами через рівень.
- Зв'язок між вузлами одного рівня не визначається.

Приклад:

Структура конструкторського бюро (працівники, проекти, замовники) (рис. 1.2).

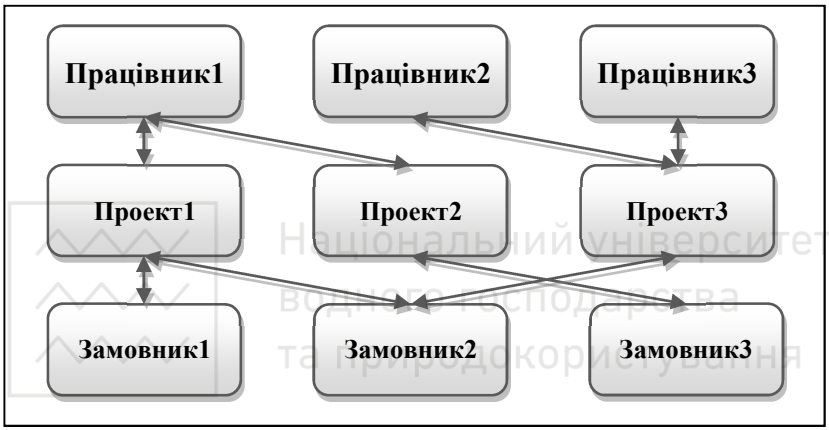


Рис. 1.2. Мережева структура даних

Переваги:

1) Більш універсальна.
2) Можливість доступу до даних більш розширена.

Недоліки:

1) Складність організації і реалізації.
2) Доступ до даних почерговий.

3. ***Реляційна модель даних***. Дані організовані у вигляді таблиць, між якими встановлені зв'язки. На даний час реляційна модель найбільш популярна і, усі сучасні СУБД орієнтовані на таке представлення даних. Надалі розглядатиметься саме така модель даних більш детально.

Контрольні запитання:

1) Дайте означення інформаційних систем.
2) Дайте означення бази даних і системи керування базою даних.
3) Як класифікують бази даних?
4) Охарактеризуйте кожен рівень архітектури бази даних.
5) Які моделі баз даних розглядаються на концептуальному рівні?
6) Чим даталогічна модель даних відрізняється від інфологічної?
7) Опишіть ієрархічну модель даних.
8) Опишіть мережеву модель даних.

2. Проектування бази даних

2.1. Процес розробки бази даних

Метою розробки будь-якої бази даних є збереження і використання інформації про деяку предметну область.

При розробці бази даних розрізняють декілька рівнів моделювання, за допомогою яких відбувається перехід від предметної області до конкретної реалізації бази даних засобами вибраної СКБД. Можна виділити наступні кроки:

1) Дослідження предметної області.
2) Модель предметної області.
3) Логічна модель даних.
4) Фізична модель даних.
5) Власне сама база даних.

Предметна область – це частина реального світу, дані про яку реалізуються в базі даних.

Предметна область є поняттям не фізичним, а інформаційним, і до неї відносяться поняття і показники, що характеризують об’єкти і відношення між ними у відповідній галузі.

Склад предметної області визначають такі фактори:

1) завдання, для вирішення яких створюється база даних;

2) місце застосування;

3) процеси, у яких використовується база даних.

Наприклад, у якості предметної області може бути організація роботи магазин, облік ліків аптеки, бухгалтерія підприємства, облік товарів на складі і т.д.

Модель предметної області – це систематизовані знання про предметну область. Це можуть бути текстовий опис, інструкції, правила, поставлені завдання, які повинні бути реалізовані. Модель предметної області описує скоріш за все процеси, які відбуваються в предметній області, і дані, які використовуються цими процесами. Від того, наскільки правильно змодельована предметна область, залежить процес розробки бази даних.

Логічна модель даних будується в термінах інформаційних одиниць, але без прив'язки до конкретної СКБД. В логічній моделі визначаються сутності, задаються ключі, встановлюються зв'язки між сутностями. Логічна модель може бути представлена у вигляді ER-діаграми.

Фізична модель даних – це модель бази даних з врахуванням виду моделі і конкретної СКБД. Будується на основі логічної моделі і відображається у вигляді схеми даних. Вибирається вид моделі, на сучасному етапі – це реляційна база даних. Сутності переходять у відношення. Встановлюються зв'язки між відношеннями. Відбувається нормалізація бази даних. Визначаються типи даних. Усуваються зв'язки „багато-до-багатьох" (∞:∞), шляхом введення додаткових таблиць. Накладаються обмеження.

Заключний етап – реалізація бази даних за допомогою СКБД. Відношення, розроблені у фізичній моделі даних, реалізуються у вигляді таблиць. Обмеження на таблиці задаються за допомогою засобів СКБД таких, як первинні і зовнішні ключі, унікальні ключі, умови на ввід даних і т.д. Остаточна реалізація бази даних полягає у створенні інших об'єктів бази даних таких, як форми, запити, звіти, процедури та інші.

2.2. Модель „сутність-зв'язок" (*ER-model*)

Моделювання структури бази даних відбувається за допомогою семантичного моделювання. У якості інструмента використовується побудова ***ER-діаграм*** (*ER – Entity Relationship*) „Сутність-зв'язок".

Перший варіант моделі був запропонований Ченом в 1976. Пізніше були запропоновані інші моделі, але в основі завжди використовувалось графічне представлення інформації.

Основні поняття ER-діаграми (за Баркером):

1) ***Сутність*** – це клас однотипних об'єктів, інформація про які використовується в моделі. Відображається у вигляді прямокутника з назвою  . ***Екземпляр сутності*** – це конкретний представник сутності. Наприклад, студент Іваненко. Екземпляри повинні бути різними.
2) ***Атрибут сутності*** – це поіменована характеристика. Наприклад, прізвище, ім'я, стипендія, дата народження і т.д. Атрибути відображаються у прямокутнику під назвою.

   Ключ сутності – атрибут сутності, який є унікальним. На діаграмі завжди підкреслюється або позначається символом #. Обов'язкові атрибути відзначені *. Вони не допускають невизначених (Null) значень.

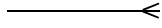
   Схематично сутність разом з атрибутами можна відобразити так:

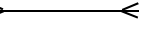
Студент
Номер
*Прізвище
Ім'я
Стипендія
Група

3) ***Зв'язок*** встановлюється між двома сутностями.

Типи зв'язків:

один -до-одного


один -до-багатьох

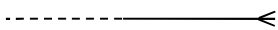

багато-до-багатьох

Модальність зв'язків:

має

може мати

Зв'язок може мати різну модальність:



Зв'язок може читатися як зліва направо, так і навпаки:

Зліва направо: кожен робітник може мати одного або більше дітей, або не мати зовсім.

Справа наліво: кожна дитина належить одному робітнику.

ER-діаграма будується на початкових стадіях проектування бази даних і в процесі роботи може змінюватись.

Після розробки ER-діаграма перетворюється у реляційну схему даних, яка точно відображає таблиці, задані ключі, поля, зв'язки. Схема даних використовується для реалізації бази даних за допомогою СКБД.

Контрольні запитання:

1) Які кроки включає процес розробки бази даних?
2) Що таке предметна область?
3) Дайте означення моделі предметної області.
4) Чим характеризується логічна модель даних?
5) Дайте означення фізичної моделі даних.
6) Для чого використовуються ER-діаграми?
7) Назвіть основні складові ER-діаграми?
8) Які типи зв'язків використовуються в ER-діаграмах?
9) Як відображаються зв'язки в ER-діаграмах?

3. Основи проектування реляційних баз даних

3.1. Означення в реляційній теорії баз даних

Реляційна база даних – це база даних, яка базується на реляційній моделі даних. Слово ,,реляційний" походить від англ. *relation* – відношення. Теоретичні основи реляційної моделі даних були закладені британським вченим Е. Коддом на початку 70-років XX століття.

За Е. Коддом, це логічна модель даних, що

- описує структури даних у вигляді наборів відношень, що можуть змінюватись в часі;
- підтримує операції реляційної алгебри;
- задовольняє спеціальним правилам, що забезпечують цілісність даних.

Фундаментальною структурою даних реляційної моделі є *N*-арне відношення.

Нехай D_1, D_2, …, D_n – довільні скінченні множини.

Декартовим добутком цих множин $D_1 x D_2 x \ldots x D_n$ є множина елементів виду $d=< d_1, d_2, \ldots, d_n>$, де $d_i \in D_i$, $i=1, \ldots, n$.

N-арне відношення – це підмножина декартового добутку множин D_1, D_2, …, D_n ($n \geq 1$).

Елементи відношення називаються ***кортежами***. Кількість кортежів визначають ***кардинальне число***.

Атрибутами є впорядковані пари (a_i, d_i), де a_i – ім'я атрибуту, d_i – множина значень. Множина усіх допустимих однорідних значень атрибута називається ***доменом***. Необов'язково всі елементи домену повинні використовуватись в одному відношенні. Кількість атрибутів називається ***степенем*** відношення (рис. 3.1).

Відношення складається із заголовка і тіла. ***Заголовок*** містить список назв атрибутів, а ***тіло*** – значення атрибутів, або, з іншої точки зору, кортежі відношення.

На практиці відношення можна представити у вигляді двовимірної таблиці, де зберігаються дані.

Рядок таблиці називають ***записом*** (відповідно – це кортеж відношення), а стовпці – ***полями*** (відповідно – це

атрибути відношення). Кожен рядок таблиці описує об'єкт або сутність. Стовпець описує одну характеристику об'єкта.

В теорії баз даних використовують означення: відношення, кортеж, атрибут, кардинальне число, степінь, а неформально – аналоги: таблиця, запис, поле, кількість записів, кількість полів.

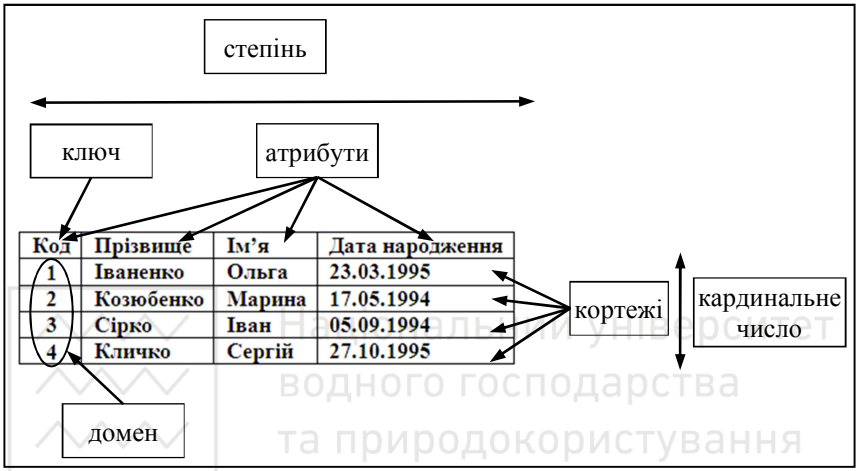


Рис. 3.1. Відношення

Відношення мають такі властивості:

- в одному відношенні не існує однакових кортежів;
- атрибути мають унікальні імена в рамках одного відношення;
- значення кожного атрибуту є логічно неподільною одиницею;
- розташування кортежів у відношенні не має значення;
- порядок слідування атрибутів у відношенні не має значення.

3.2. Реляційна алгебра

Формальною основою РБД є ***реляційна алгебра*** (РА), яка базується на теорій множин, де розглядаються спеціальні оператори над відношеннями.

Основних операторів в реляційній алгебрі 8. Вона замкнута відносно поняття відношення, тобто результатом виконання будь-якої операції реляційної алгебри над відношенням є відношення.

Всі операції реляційної алгебри розподілені на дві групи:

- **теоретико-множинні операції** над даними: об'єднання, перетин, різниця, декартовий добуток;
- **спеціальні реляційні операції**: вибірка, проекція, з'єднання і ділення.

Результатом кожної операції має бути відношення. В цьому полягає ***властивість замкнутості***.

Теоретико-множинні операції об'єднання, перетину та різниці відносяться до сумісних відношень.

Відношення називаються ***сумісними***, якщо в них однакова кількість атрибутів (степені відношень рівні) і домени зіставлених атрибутів співпадають.

Властивості бінарних операцій:

- Операція **φ** є ***комутативною***, якщо $A \varphi B = B \varphi A$;
- Операція **φ** є ***асоціативною***, якщо

$$(A \varphi B) \varphi C = A \varphi (B \varphi C);$$

- Операція **φ** є ***дистрибутивною*** з операцією **θ**, якщо

$$A \varphi (B \theta C) = (A \varphi B) \theta (A \varphi C).$$

Теоретико-множинні операції:

1. Об'єднання

Об'єднанням сумісних реляційних відношень R_1 і R_2 називають таке реляційне відношення, що містить кортежі обох відношень, але без повторень.

$$R = R_1 \cup R_2 = \{r \mid r \in R_1 \cup r \in R_2\}$$

Приклад:

R_1

A	B
a_1	b_1
a_1	b_2
a_2	b_3

R_2

A	B
a_1	b_1
a_2	b_1

$R_1 \cup R_2$

A	B
a_1	b_1
a_1	b_2
a_2	b_1
a_2	b_3

Операція об'єднання є комутативною, асоціативною і дистрибутивною щодо перетину.

2. Перетин

Перетином сумісних реляційних відношень $\boldsymbol{R_1}$ і $\boldsymbol{R_2}$ називають таке реляційне відношення, яке містить кортежі, що входять до складу обох відношень.

$$\boldsymbol{R = R_1 \cap R_2 = \{r \mid r \in R_1 \,\&\, r \in R_2\}}$$

Операція перетину є комутативною, асоціативною і дистрибутивною щодо об'єднання.

Приклад:

R_1

A	B
a_1	b
a_1	b_2
a_2	b_3

R_2

A	B
a_1	b_1
a_2	b_1

$R_1 \cap R_2$

A	B
a_1	b_1

3. Різниця

Різницею сумісних реляційних відношень $\boldsymbol{R_1}$ і $\boldsymbol{R_2}$ називають таке реляційне відношення, що містить такі кортежі з першого відношення $\boldsymbol{R_1}$, яких немає у другому відношенні $\boldsymbol{R_2}$.

$$\boldsymbol{R = R_1 - R_2 = \{r \mid r \in R_1 \,\&\, r \notin R_2\}}$$

Операція різниці властивостями комутативності, асоціативності і дистрибутивності не володіє.

Приклад

R1

A	***B***
a_1	b_1
a_1	b_2
a_2	b_3

R2

A	***B***
a_1	b_1
a_2	b_1

R1 - R2

A	***B***
a_1	b_2
a_2	b_3

4. Декартовий добуток

Декартовим добутком реляційних відношень $R(A_1, A_2, \ldots, A_n)$ і $S(B_1, B_2, \ldots, B_m)$ називають таке реляційне відношення $Q(A_1, A_2, \ldots, A_n, B_1, B_2, \ldots, B_m)$, яке містить усі можливі з'єднання кортежів відношення R з кортежами відношення S.

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

Операція декартового добутку комутативна і асоціативна.

Приклад:

R

A	***B***
a_1	b_1
a_1	b_2
a_2	b_3

S

C	***D***
c_1	d_1
c_2	d_1

R×S

A	***B***	***C***	***D***
a_1	b_1	c_1	d_1
a_1	b_1	c_2	d_1
a_1	b_2	c_1	d_1
a_1	b_2	c_2	d_1
a_2	b_3	c_1	d_1
a_2	b_3	c_2	d_1

Спеціальні реляційні операції:

5. Вибірка (селекція)

Вибіркою **(*селекцією*)** реляційного відношення ***R*** називають таке реляційне відношення, що містить кортежі, що задовольняють певній умові.

Нехай ***R*** – задане відношення, ***L*** і ***M*** – атрибути даного відношення. **θ** – оператор порівняння ($=, \neq, <, >, \leq, \geq$) такий, що умова ***L*** **θ** ***M*** при заданих ***L*** і ***M*** дає значення істинно чи хибно. Тоді селекцією або **θ**-вибіркою буде відношення, що має такий самий заголовок, що і відношення ***R***, і тіло, що містить множину кортежів, для яких умова ***L*** **θ** ***M*** істинна.

$$\boldsymbol{S = R[L\,\theta\,M] = \{r \mid r \in R_1 \,\&\, r[L]\,\theta\, r[M]\}}$$

Приклад:

R

A	***B***	***C***
a_1	b_1	c_1
a_1	b_2	c_1
a_2	b_3	c_1
a_2	b_4	c_2

***R*[A=a_2]**

A	***B***	***C***
a_2	b_3	c_1
a_2	b_4	c_2

6. Проекція

Проекцією реляційного відношення ***R*** називають таке реляційне відношення, кортежі якого отримані з кортежів відношення ***R*** шляхом видалення значень, що не належать атрибутам, за якими виконується проекція. При цьому повторення кортежів видаляються.

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

A_{i1}, …, A_{in} – атрибути, за якими проводиться проекція.

Приклад:

R

A	**B**	**C**
a_1	b_1	c_1
a_1	b_2	c_1
a_2	b_3	c_1
a_2	b_4	c_2

***R*[*A*, *C*]**

A	**C**
a_1	c_1
a_2	c_1
a_2	c_2

7. З'єднання

Нехай ***R*(*L*, *M*)** і ***S*(*N*, *P*)** – задані відношення, **M** і **N** – атрибути даних відношень, зв'язані оператором порівняння **θ**.

З'єднанням реляційних відношень ***R*(*L*, *M*)** і ***S*(*N*, *P*)** за умовою $M\ \theta\ N$ називають таке реляційне відношення ***Q***, що містить атрибути **(*L*, *M*, *N*, *M*, *N*, *P*)**, кортежі якого можна отримати з'єднанням тих кортежів відношень ***R*** і ***S***, на яких виконується умова $M\ \theta\ N$. Під час з'єднання атрибути, за якими виконується така операція, повторюються в кінцевому реляційному відношенні

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

Операція з'єднання асоціативна і комутативна.

Приклад:

R

A	***B***	***C***	***D***
a_1	b_1	c_1	d_1
a_1	b_1	c_2	d_1
a_1	b_2	c_1	d_1
a_2	b_3	c_1	d_1
a_2	b_4	c_2	d_3

S

C	***D***	***E***
c_1	d_1	e_2
c_2	d_1	e_3
c_2	d_1	e_1

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

A	***B***	***C***	***D***	***C***	***D***	***E***
a_1	b_1	c_1	d_1	c_1	d_1	e_2
a_1	b_2	c_1	d_1	c_1	d_1	e_2
a_1	b_1	c_1	d_1	c_1	d_1	e_2
a_1	b_1	c_2	d_1	c_2	d_1	e_3
a_1	b_2	c_2	d_1	c_2	d_1	e_3
a_1	b_1	c_2	d_1	c_2	d_1	e_1
a_1	b_2	c_2	d_1	c_2	d_1	e_1

З'єднання за умовою рівності називається ***еквіз'єднанням***. З'єднання за умовою рівності, коли порівнювані атрибути, які повторюються, видаляються з кінцевого відношення, називається ***природнім з'єднанням R*S= Q(L, M, N, P)***.

R*S

A	***B***	***C***	***D***	***E***
a_1	b_1	c_1	d_1	e_2
a_1	b_2	c_1	d_1	e_2
a_1	b_1	c_1	d_1	e_2
a_1	b_1	c_2	d_1	e_3
a_1	b_2	c_2	d_1	e_3
a_1	b_1	c_2	d_1	e_1
a_1	b_2	c_2	d_1	e_1

8. Ділення

Образом реляційного відношення ***R(M, N)*** за кортежем $t_1 \in R[M]$ називають таку множину кортежів $t_2 \in R[N]$, для яких зчеплення (t_1, t_2) належить відношенню ***R***.

$$I_R(t_1) = \{t_2 \mid t_2 \in R_1\ [N] \,\&\, (t_1, t_2) \in R\}$$

Приклад:

R

A	*B*	*C*
a_1	b_1	c_1
a_1	b_1	c_2
a_1	b_3	c_2
a_2	b_1	c_4

$I_R(a_1)$

B	*C*
b_1	c_1
b_1	c_2
b_3	c_2

$I_R(a_2)$

B	*C*
b_1	c_4

$I_R(a_1, b_1)$

C
c_1
c_2

$I_R(c_2)$

A	*B*
a_1	b_1
a_1	b_3

Нехай задано ***R(M, N)*** і ***S(K, L)***, для яких проекції ***R[N]*** і ***S[K]*** є сумісними.

Діленням реляційного відношення ***R*** на ***S*** за наборами ***N*** і ***K*** називається операція, результатом якої є відношення ***Q(M)***, що складається з таких кортежів $t \in R[M]$, образи $I_R(t)$ яких містять усі кортежі проекції $S[K]$.

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

Приклад:

R

A	***B***	***C***
a_1	b_1	c_1
a_1	b_1	c_2
a_1	b_3	c_2
a_2	b_1	c_4

S

C	***D***
c_1	d_1
c_1	d_2
c_2	d_1
c_2	d_1

***S*[*C*]**

C
c_1
c_2

R*[*C*÷*C*]*S

A	***B***
a_1	b_1

3.3. Поняття ключа. Зв'язки між відношеннями

Особливе місце в реляційних базах даних відводиться *цілісності даних*. У будь-якому реляційному відношенні можна виділити таку множину атрибутів, набори відповідних значень яких однозначно ідентифікують кортежі відношень. Таку множину називають ***первинним ключем*** відношення. Первинний ключ може бути ***простим***, якщо він складається з одного атрибуту, і ***складеним***, якщо він складається з двох і більше атрибутів.

Первинний ключ має такі властивості:

- кожне реляційне відношення має один і тільки один первинний ключ;
- значення всіх атрибутів первинного ключа не можуть бути невизначеними;
- значення первинного ключа не можуть повторюватись, хоча допускаються повторення значень частин складеного первинного ключа;
- значення первинного ключа не визначають порядок кортежів у відношенні, хоча використовуються для сортування кортежів.

Атрибут, який має властивості первинного ключа, і не виконує його роль, називається ***унікальним***.

Якщо атрибути, які є первинним ключем одного відношення, присутні в другому відношенні, то вони використовуються для зв'язку з першим відношенням, і називаються ***зовнішнім ключем*** відношення. На відміну від первинного ключа, зовнішній ключ може допускати повторення і невизначені (NULL) значення. Відношення, що містить первинний ключ, називається ***головним***, а відповідне відношення із зовнішнім ключем – ***підпорядкованим***.

Зв'язки між відношеннями можуть бути трьох типів:

1) **один-до-одного (1-1)**. Кожному кортежу (запису) відношення **A** в певний момент часу відповідає один або жодного кортежу (запису) відношення **B**. Наприклад, кожна людина має одну адресу проживання.
2) **один-до-багатьох (1-∞)**. Кожному кортежу відношення **A** в певний момент часу відповідає кілька кортежів **B**. При цьому кожен кортеж відношення **B** пов'язаний з одним кортежем відношення **A**. Наприклад, один студент складає іспити по різним предметам. Зв'язок **багато-до-одного** є зворотнім відносно даного.
3) **багато-до-багатьох (∞-∞)**. У будь-який момент часу множині кортежів відношення **A** відповідає множина кортежів відношення **B**. Наприклад, студенти відвідують заняття з різних предметів. На практиці такий зв'язок реалізується через зв'язки **один-до-багатьох** і **багато-до-одного** за допомогою додаткового відношення.

Відношення без кортежів із встановленими зв'язками утворюють ***схему бази даних***.

3.4. Цілісність бази даних

На зв'язки між відношеннями накладаються певні ***правила цілісності***, які допомагають зберігати базу даних в стабільному і робочому стані.

Помилкова ситуація може виникати при додаванні, зміні чи видаленні кортежів з відношень. Якщо дія, що виконується, суперечить правилам цілісності бази даних, то СКБД видає повідомлення про помилку і виконання дії відхиляється, в іншому випадку – отримується позитивний результат.

1) **Додавання кортежів**. Відбувається спочатку у головне відношення. Додавання кортежів у підпорядковане відношення з новим зовнішнім ключем приводить до помилки.
2) **Модифікація кортежів**. Найчастіше пов'язана із зміною первинного і зовнішнього ключів. При цьому можливі три операції:
   а. **Без будь-якої дії** (*NO ACTION*). Модифікація первинного ключа відбувається у головному відношенні, якщо у підпорядкованому відношенні немає відповідних кортежів. При наявності таких їх потрібно видалити, бо інакше це приведе до помилкової ситуації.
   б. **Каскадування** (*CASCADE*). Модифікація первинного ключа у головному відношенні призводить до відповідної зміни зовнішніх ключів у підпорядкованому відношенні.
   в. **NULL-значення** (*SET NULL*). Модифікація первинного ключа у головному відношенні призводить до встановлення NULL-значення у зовнішніх ключах підпорядкованого відношення.
3) **Видалення кортежів.** При цьому можливі три операції:
   а. **Без будь-якої дії** (*NO ACTION*). Видалення кортежів з головної таблиці неможливе, якщо є відповідні кортежі у підпорядкованому відношенні.

б. **Каскадування** (*CASCADE*). Видалення кортежів у головному відношенні призводить до видалення відповідних кортежів у підпорядкованому відношенні.

в. **NULL-значення** (SET NULL). Видалення кортежів у головному відношенні призводить до встановлення NULL-значення у зовнішніх ключах підпорядкованого відношення.

3.5. Нормалізація бази даних

Нормалізація – це процес заміни даної схеми реляційних відношень іншою схемою, в якій відношення мають просту і коректну форму.

В теорії реляційних баз даних нормалізація включає таку послідовність нормальних форм:

- Перша, друга і третя нормальні форми (1НФ, 2НФ, 3НФ), які розроблені Ф. Коддом;
- Нормальна форма Бойса-Кодда (НФБК);
- Четверта нормальна форма (4НФ);
- П'ята нормальна форма або нормальна форма проекції-з'єднання (5НФ або НФПЗ).
  4НФ і 5НФ визначені Р. Фейджиним.

Основні властивості нормальних форм:

- кожна наступна форма краща за попередню;
- при переході до наступної нормальної форми властивості попередньої нормальної форми зберігаються.

Реляційне відношення перебуває в тій чи іншій нормальній формі, якщо задані на ньому функціональні залежності задовольняють певним умовам.

Перехід від початкового стану бази даних до нормалізованого виконується в декілька кроків.

1-й крок. Зведення відношення до першої нормальної форми.

Для відношення повинні виконуватись наступні умови:

1) відсутність повторень атрибутів і кортежів;
2) відсутність впорядкування атрибутів і кортежів;
3) елементарність значень атрибутів у кортежах відношення;
4) усі кортежі відношення повинні мати однакову структуру;
5) імена атрибутів мають бути різними, значення кожного атрибуту повинні мати однаковий тип.

Перша нормальна форма

Реляційне відношення перебуває в ***першій нормальній формі*** (**1НФ**), якщо всі його атрибути мають атомарні (прості) домени, тобто на перетині рядка і стовпця таблиці має стояти значення, неподільне з точки зору всіх його можливих застосувань.

2-й крок. Перехід до другої нормальної форми.

Нехай дано відношення ***R***, яке містить набори атрибутів ***A*** і ***B***. У відношенні ***R*** набір атрибутів ***B функціонально залежить*** від ***A*** і ***A*** функціонально визначає ***B*** тоді і лише тоді, коли кожному значенню проекції ***R*[*A*]** у будь-який момент часу відповідає точно одне значення проекції ***R*[*B*]**. Іншими словами, для будь-якого кортежу відношення ***R*** кожному значенню з набору атрибутів ***A*** відповідає одне значення набору атрибутів ***B***.

Функціональна залежність позначається як ***R. A→R. B***.

Приклад:

Нехай ***R*** – це відношення **STUDENTS**. ***A*** – код студента, а ***B*** – набір всіх атрибутів студента. Тоді ***A → B***, так як ***A*** являє собою первинний ключ, який однозначно ідентифікує кожен запис в таблиці **STUDENTS**.

Функціональна залежність ***R. A→R. B*** називається ***повною***, якщо ***B*** не залежить функціонально від жодного піднабору ***C⊂A***, іншими словами атрибут перебуває у повній функціональній залежності, якщо він залежить від усього ключа і не залежить від його складових.

Друга нормальна форма

Реляційне відношення перебуває в ***другій нормальній формі*** (**2НФ**), якщо воно перебуває в першій нормальній формі і всі його неключові атрибути функціонально повно залежать від первинного ключа.

Означення другої нормальної форми можна сформулювати ще так:

Реляційне відношення перебуває в ***другій нормальній формі*** (**2НФ**), якщо воно перебуває в першій нормальній формі і немає неключових атрибутів залежних функціонально від складових первинного ключа.

З цього можна зробити висновок, що, якщо первинний ключ є простим, то відношення автоматично знаходиться у 2НФ.

Реляційні відношення можуть містити небажані або надлишкові функціональні залежності, тобто кількість атрибутів відношення може бути завеликою або можуть виникати різні аномалії при додаванні (інформація може бути невизначена), видаленні (інформація може втрачатись) і оновленні кортежів (однакову інформацію необхідно оновлювати багато раз для різних кортежів).

Виправлення даних недоліків відбувається за допомогою ***декомпозиції***, тобто розкладання реляційного відношення на множину інших відношень без небажаних функціональних залежностей. Таке розбиття не повинно приводити до втрати інформації. Воно має бути оберненим, тобто можливе зворотне з'єднання. При цьому має виконуватись теорема Хіта.

Теорема Хіта

Якщо у відношенні ***R*(*A*, *B*, *C*)** з наборами атрибутів ***A***, ***B*** і ***C*** задано функціональну залежність $A \rightarrow B$, то його можна декомпонувати на два відношення ***R1*[*A*, *B*]** і ***R*[*A*, *C*]** таких, що завдяки операції природного з'єднання проекцій відношення ***R*** за атрибутами ***R*[*A*, *B*]** і ***R*[*A*, *C*]** отримується початкове відношення ***R***.

$$R[A, B]*R[A, C]=R(A, B, C).$$

Нехай у відношенні ***R*(*A**, *B**, *C, D*)** атрибут ***C*** залежить від усього ключа, який складається з атрибутів ***A**** і ***B****, тобто є складеним. Атрибут ***D*** залежить лише від ***B***, тоді ***D*** знаходиться у неповній функціональній залежності, яку необхідно усунути за допомогою декомпозиції.

В результаті декомпозиції виділяються два відношення:

***R1*(*A**, *B**, *C*)** і ***R2*(*B**, *D*)**.

3-й крок. Перехід до третьої нормальної форми.

Транзитивна залежність – це залежність між неключовими атрибутами.

Нехай задано відношення ***R*(*A**, *B*, *C*)**, в якому атрибут ***C*** не залежить безпосередньо від ключового атрибута ***A****, а залежить від неключового атрибута ***B***, який залежить від ***A****, тоді ***C*** транзитивно залежить від ***A****.

Транзитивні залежності вилучаються за допомогою декомпозиції. У результаті отримуються два нових відношення ***R1*(*A**, *B*)** і ***R2*(*B**, *C*)**.

Третя нормальна форма

Реляційне відношення перебуває в ***третій нормальній формі*** (**3НФ**), якщо воно перебуває у другій нормальній формі і не містить транзитивних залежностей непервинних атрибутів від можливих ключів, тобто всі його непервинні атрибути функціонально незалежні один від одного.

Відношення, яке знаходиться у 3НФ, не може містити розрахункових полів.

Нормальна форма Бойса-Кодда (НФБК)

Вимоги ***нормальної форми Бойса-Кодда*** аналогічні вимогам 3НФ, тільки єдиний ключ у кожному відношенні є простим – складається лише з одного атрибуту.

4-й крок. Перехід до четвертої нормальної форми.

Багатозначна залежність – це різновид функціональної залежності. Атрибут ***B*** знаходиться у багатозначній залежності від атрибута ***A*** тоді, коли одному значенню атрибута ***A*** відповідає багато значень атрибута ***B***. Наприклад, між атрибутами група : код студента=1:∞, так як в одній групі навчається багато студентів.

Існують поняття тривіальної і нетривіальної багатозначної залежності.

Залежність типу A – >>B і B – >>A є ***тривіальною***, а залежність типу A – >>B і $B \neq$ >>A – ***нетривіальною***.

Четверта нормальна форма

Реляційне відношення перебуває у ***четвертій нормальній формі*** (**4НФ**), якщо воно перебуває у третій нормальній формі і в структурі багатозначної залежності, яка визначена на множині атрибутів, є лише тривіальні чи такі нетривіальні багатозначні залежності, для яких ліва частина є ключем.

Атрибут ***A*** багатозначно визначає атрибут ***B*** у відношенні ***R*(*A**, *B*, *C*)**, якщо ***B*** залежить лише від ***A*** при будь-яких його комбінаціях з іншими атрибутами відношення.

Якщо у відношенні ***R*(*A**, *B*, *C*)** наявні багатозначні залежності, тобто ***A*** – >>***B*** і ***A*** – >>***C***, то воно має бути розкладене на два інших відношення ***R1*(*A*, *B*)** і ***R*(*A*, *C*)**.

5-й крок. Перехід до п’ятої нормальної форми.

Для виявлення багатозначних залежностей потрібний значно глибший семантичний аналіз атрибутів.

Відношення, яке містить більш як три багатозначні залежності, потребує переходу до п’ятої нормальної форми.

П’ята нормальна форма

Реляційне відношення перебуває у ***п’ятій нормальній формі*** (**4НФ**), якщо воно перебуває у четвертій нормальній формі і при декомпозиції з 4НФ отримано такі проекції, кожна з яких містила б не менше як один можливий ключ і щонайменше один неключевий атрибут початкового відношення.

Для виявлення багатозначних залежностей потрібний значно глибший семантичний аналіз атрибутів. На практиці 4НФ і 5НФ використовуються при проектуванні складних баз даних.

Отже, нормалізація відношень усуває між атрибутами такі залежності: неповні функціональні, транзитивні, нетривіальні багатозначні. Завдяки чому, виключається дублювання даних і можливість виникнення аномалій при виконанні операцій поповнення, заміни та вилучення даних.

3.6. Приклад проектування бази даних Univer

Основною метою бази даних є організація навчальної роботи факультету навчального закладу.

Предметна область, яка досліджується, – це навчальний процес.

Початковими даними є особові справи студентів, навчальний план, розподіл навантаження, екзаменаційні відомості.

В результаті отриманої інформації визначаються завдання, виконання яких повинна забезпечувати база даних:

1) Містити інформацію про студентів, предмети, іспити, викладачів.
2) Можливості введення, перегляду, модифікації і видалення даних за допомогою спеціальних форм.
3) Виконання необхідних запитів.
4) Зручний вивід результатів роботи.

Кожен з даних пунктів конкретизується і доповнюється.

Надалі розробляється структура або модель бази даних. У якості інструмента можна використати ER-діаграму „Сутність-зв’язок” або схему даних

У результаті розробки потрібно отримати таку інформацію про предметну область:

1) Список сутностей предметної області;
2) Список атрибутів сутностей;
3) Опис зв'язків між сутностями.

Формується відношення, що містить інформацію про дату іспиту, студента, предмет, оцінку, викладача. Відповідно до 1НФ Кодда відбувається деталізація атрибутів. Наприклад, атрибут ПІБ доцільно поділити на три атрибути: прізвище, ім'я, по батькові.

В результаті отримане відношення можна записати у вигляді: USPISH(NOM, SNOM, SFAM, SIMA, SOTCH, GRUPA, PNOM, PNAME, SEMESTR, VNOM, VFAM,OCINKA), де

NOM – номер запису про зданий іспит;
SNOM – номер залікової книжки;
SFAM – прізвище студента;
SIMA – ім'я студента;
SOTCH – по батькові студента;
GRUPA – група, в якій навчається студент;
PNOM – номер предмета у навчальному плані;
PNAME – назва предмета;
SEMESTR – номер семестру, в якому викладається предмет;
VNOM – табельний номер викладача;
VFAM – прізвище викладача;
OCINKA – оцінка за іспит.

Оскільки, дане відношення передбачає атрибути, що характеризують студентів, їх оцінок, предметів і викладачів, то для даного відношення необхідно задати складний первинний ключ (NOM, SNOM, PNOM, VNOM).

На другому кроці здійснюється перехід до 2НФ. Якщо за даним відношенням скласти таблицю даних, то зразу ж буде видно їх надлишковість, що призводить до аномалій додавання, оновлення або знищення записів. Надалі аналізуються відповідні залежності з метою вилучення неповних функціональних залежностей.

Наприклад, атрибути SIMA, SOTCH, GRUPA залежать функціонально від атрибута SNOM. Дані атрибути можна виділити в окреме відношення STUDENTS з ключем SNOM.

За допомогою декомпозиції з USPISH виділяються відношення STUDENTS(SNOM*, SIMA, SOTCH, GRUPA), PREDMET(PNOM*, PNAME, VNOM, VFAM, SEMESTR) і отримується відношення USPISH(NOM*, SNOM, PNOM, OCINKA). Аналогічно застосовується декомпозиція до відношення PREDMET, з якого виділяється відношення VYKLAD(VNOM*, VFAM) і залишається PREDMET(PNOM*, PNAME, VNOM, SEMESTR).

Оскільки, усі ключі є простими, можна зробити висновок, що в кожному відношенні встановлена повна функціональна залежність усіх атрибутів від ключа, що відповідає 2НФ. Усі відношення відображаються на ER-діаграмі відповідно до вимог, встановлюються зв'язки між відношеннями.

В процесі проектування дані можуть уточнюватись, доповнюватись. У цьому випадку кожне відношення знову перевіряється на відповідність 1НФ і 2НФ.

Так, доповнимо відношення додатковими атрибутами, оскільки раніше дана інформація не була актуальною.

Відношення STUDENTS атрибутами:

FORM – форма навчання;
FOTO – фото студента;
STIP – стипендія.

Відношення VYKLAD атрибутами:

VIMA – ім'я викладача;
VOTCH – по батькові викладача;
KAF – кафедра, на якій працює викладач;
POSADA – посада викладача;
OKLAD – заробітна плата.

Відношення PREDMET атрибутами:

GOD – кількість годин, відведених на вивчення предмета.

Виконується перевірка на відповідність 1НФ і 2НФ.

При переході до 3НФ визначаються транзитивні залежності. Якщо прийняти до уваги, що стипендія студенту нараховується тільки відповідно до форми навчання, то виникає транзитивна залежність атрибута STIP від SNOM У цьому випадку виділяється окреме відношення з копією атрибута FORM як ключем і атрибутом STIP. Але, враховуючи, що стипендія залежить від оцінок, може бути підвищена, нараховуватись пільги, виплачуватись не тільки студентам державної форми навчання, а й платної як грант, а наведена база є навчальною і не потребує такого уточнення, то така залежність не встановлена.

Аналогічні міркування можна застосувати і до атрибута OKLAD відношення VYKLAD.

ER-діаграма не враховує особливості конкретної СКБД, і тому варто перейти до побудови схеми даних, більш наближеної до практичної реалізації: Враховуються допустимі імена таблиць і полів, визначаються типи даних, накладаються умови цілісності, усувається зв'язок ,,багато-до-багатьох" ∞:∞, який не підтримується реляційними базами даних. Для реалізації такої таблиці необхідно створити додаткову таблицю, яка буде відігравати роль зв'язкової. Зв'язкова таблиця повинна містити первинні ключі таблиць, між якими встановлюється зв'язок.

Така схема даних може використовуватися для побудови бази даних за допомогою конкретної СКБД.

Сучасні СКБД можуть автоматично відтворювати діаграми, які відображають модель побудованої бази даних (рис. 3.2-3.3).

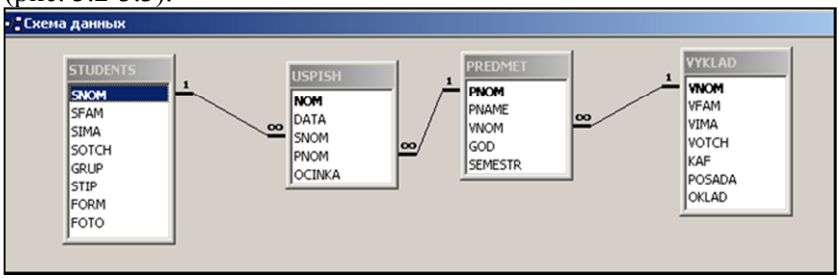


Рис. 3.2. Схема даних, побудована *Microsoft Access*

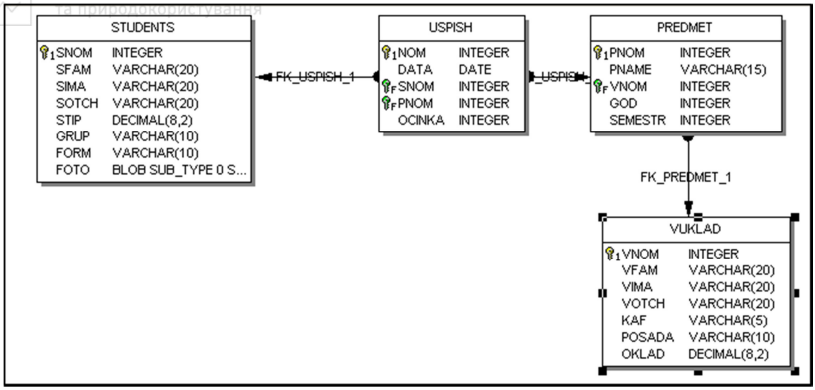


Рис. 3.3. Схема даних, побудована *IBExpert*

Для створення фізичної моделі бази даних Univer пропонується використати СКБД типу ,,клієнт-сервер” *IBExpert-Firebird*. В результаті реалізації буде отримано файл univer.fdb.

Контрольні запитання:

1) Дайте означення реляційної бази даних.
2) Що таке відношення?
3) Які є складові відношення?
4) У якому вигляді реалізуються відношення у сучасних базах даних?
5) Які операції реляційної алгебри ви знаєте?
6) Назвіть характеристики первинного ключа відношення?
7) Як встановлюються зв’язки між відношеннями?
8) Які є типи зв’язків?
9) Для чого використовується нормалізація бази даних?
10) Скільки є нормальних форм?
11) Що таке декомпозиція?
12) В чому полягає суть теореми Хіта?

Розділ ІІ. Практичні основи баз даних

4. СКБД IBExpert – Firebird

4.1. Основні відомості про архітектуру клієнт-сервер

В даний час стрімко розвиваються системи керування базами даних, які побудовані на основі архітектури „клієнт-сервер”.

В якості серверної частини виступає програма, що виконує комплекс дій по управлінню даними: виконання запитів, збереження і резервне копіювання даних, підтримку цілісності бази даних, перевірку прав користувачів, ведення журналу транзакцій.

Клієнт являє собою додаток, що полегшує роботу користувача, виконує запити до сервера і отримує інформацію від нього.

Така технологія є досить продуктивною через мінімізацію обсягу інформації, яка передається мережею, краще забезпечує захист інформації від несанкціонованого доступу та цілісність даних, забезпечує одночасну роботу багатьох користувачів з однією базою даних.

Серверна і клієнтська частини можуть розміщуватись як на одному комп’ютері, так і на різних.

Найбільш популярним сервером баз даних є *Interbase* і його клони *Firebird*, *Yaffil*. Як клієнтські програми широко використовуються *IBExpert* (www.ibexpert.net), *IBAdmin* (www.sqlly.com/ibadmin2.html), *IBManager*(www.ibmanager.com), *IBWorkbench* (www.InterBaseworkbench.com).

В даному посібнику розглядається як сервер програма *Firebird 3.0* (www.firebirdsql.org), як клієнт – *IBExpert* (www.ibexpert.net).

4.2. Інсталяція програми Firebird 3.0

Firebird SQL Server – сервер баз даних, заснований на відкритому коді *Interbase* 6.0. Дистрибутиви *Firebird* існують під *Windows*, *Linux*, *Unix*, *Solaris*, *MacOS*, під 32-х і 64-х бітну

архітектуру. *Firebird SQL Server* розповсюджується безкоштовно і не має ліцензійних обмежень.

Інсталяція програми *Firebird 3.0* за замовчуванням відбувається у папку **C:\Program Files\Firebird\Firebird_3_0**. Існує 3 варіанти установки:

1) **Classic server** – на кожне клієнтське підключення створюється окремий серверний процес, підтримка багатопроцесорних машин, немає захисника *Guardian* для управління сервером.

2) **SuperClassic server** – використовує єдиний серверний процес, рекомендується для 64-розрядних систем, для багатопроцесорних машин.

3) **Super server** – всі клієнтські підключення обслуговуються одним серверним процесом, завдяки цьому використовується менший об'єм пам'яті при більшій швидкодії, для однопроцесорних машин.

Дані архітектури використовують однаковий формат файлів бази даних, тому можна легко переключитись на іншу архітектуру.

Під час інсталяції пропонується вибрати архітектуру **Super server**, вказати необхідні налаштування: запуск захисника (*Guardian*), спосіб роботи сервера як додатку (*Application*) чи як сервісної служби (*Service*); автоматичний старт при завантаженні комп'ютера, копіювання клієнтської бібліотеки в каталог **System32**, створення бібліотеки **GDS32.dll** (рис. 4.1).

Сервіс захисника (*Guardian*) контролює роботу сервіса і перезапускає його у разі аварійної ситуації.

На наступному кроці необхідно задати пароль для системного адміністратора SYSDBA. Зазвичай, це *masterkey*. У цілях безпеки бази даних пропонується задавати інший пароль.

Після закінчення установки буде завантажений серверний процес *Firebird* як сервіс (служба), який буде запускатись автоматично при наступних завантаженнях комп'ютера.

Для запуску, зміни налаштування і зупинки сервера можна скористуватись **Панель управління (*Control panel*) ⇒ Адміністрування (*Administrative Tools*) ⇒ Сервіси (*Services*)**.

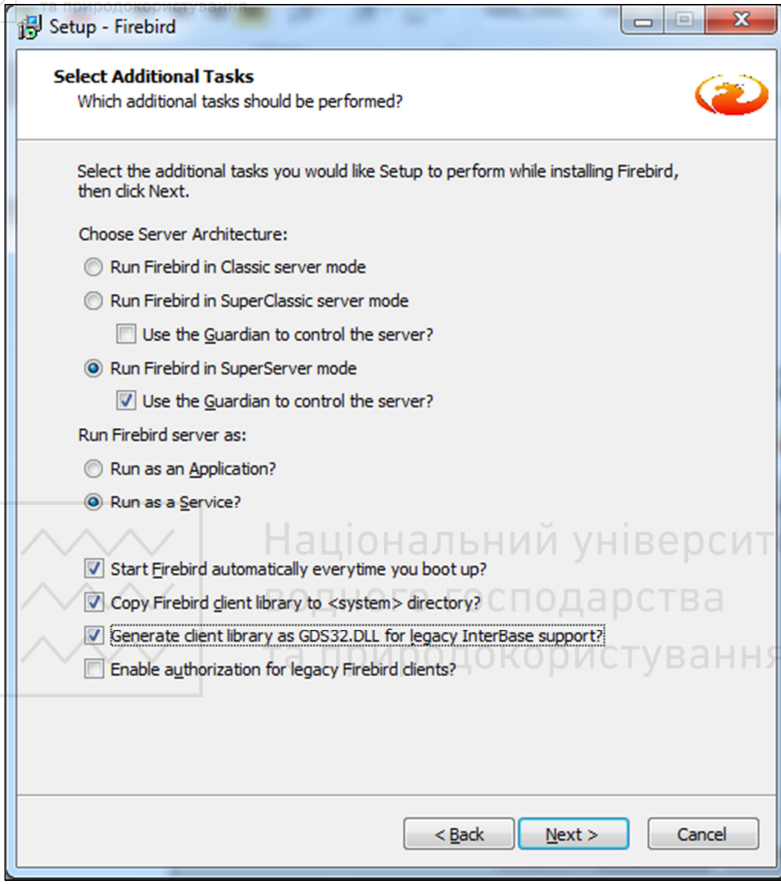


Рис. 4.1. Налаштування сервера *Firebird* при інсталяції

Якщо сервер *Firebird* завантажується як додаток, то на панелі завдань з'являється іконка додатку або іконка захисника (*Guardian*), за яким можна відслідковувати роботу сервера. Якщо іконка блимає або відображається в червоних тонах, то відбувається перезавантаження сервера або збій в роботі.

Для роботи з базою даних використовується утиліта **isql.exe**, що знаходиться у папці **C:\Program Files\Firebird\Firebird_3_0\bin.**

4.3. Клієнтська програма IBExpert

IBExpert – оболонка з графічним інтерфейсом користувача, призначена для розробки і адміністрування баз даних *Interbase* і *Firebird*.

IBExpert містить багато компонентів, які полегшують роботу користувача: візуальний редактор для всіх об'єктів бази даних, редактор *SQL*, редактор скриптів, компілятор для збережених процедур і тригерів, будівник запитів і схеми баз даних, дизайнер баз даних і т.д.

Запуск програми *IBExpert* здійснюється за допомогою ярлика 

.

Для створення бази даних необхідно виконати команду **Database ⇒ Create Database**.

У вікні **Create Database** необхідно вказати таку інформацію:

Server – тип підключення до сервера. Може бути локальним або віддаленим. Якщо база даних буде створена на тому ж самому комп'ютері, де інстальований *Firebird*, то вибирається тип *Local*.

Server name – назва комп'ютера в мережі, де встановлений *Firebird*.

Protocol – мережевий протокол, який використовується для підключення до сервера. За замовчуванням це TCP/IP, отже в назві сервера можна вказувати пряму IP-адресу сервера.

Database – місце розташування і назва файлу бази даних. Тип файлу бази даних може бути ***.fdb** (*Firebird database*) або ***.gdb** (*InterBase database*).

Username – ім'я адміністратора бази даних. Є стандартним для всіх баз даних, його змінювати не можна. Це ім'я – SYSDBA.

Password – за замовчуванням це – masterkey. Після створення бази даних пароль рекомендується змінити.

Page Size – розмір сторінки бази даних, рекомендується 8 192 або 16 384 байта.

Charset – кодування. Визначає, символи якого алфавіту будуть використовуватись в базі даних за замовчуванням. Якщо передбачається використання кирилиці, то бажано використовувати WIN1251.

SQL Dialect – для нових баз даних необхідно вибирати Dialect 3 (рис. 4.2).

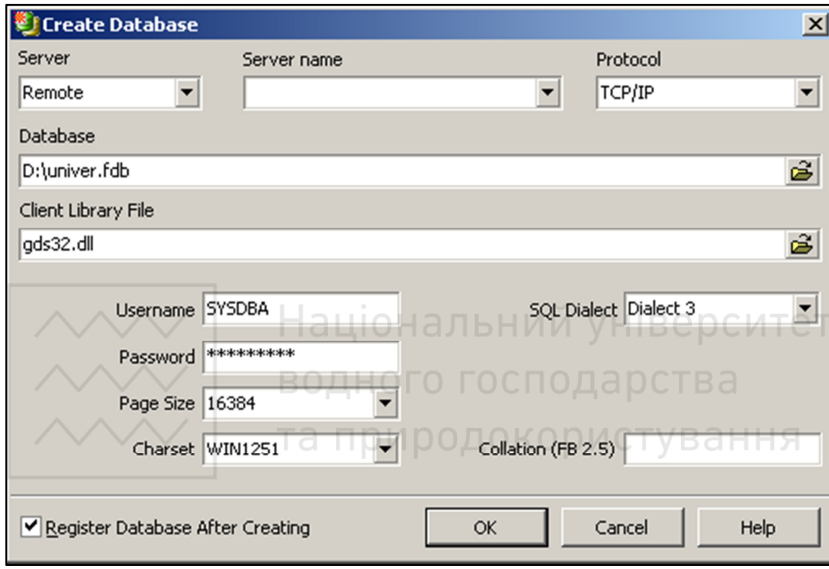


Рис. 4.2. Вікно створення бази даних

Після натиснення кнопки **ОК**, автоматично відкривається вікно реєстрації, в якому додатково необхідно задати версію сервера і псевдонім бази даних.

Якщо база даних переноситься з іншого комп'ютера, то її необхідно просто зареєструвати командою **Database ⇒ Register Database**.

Для роботи з базами даних призначені рядок меню і панель інструментів. Додаткова інформація відображається у рядку стану. На панелі ***Windows Bar*** розміщуються відкриті об'єкти *IBExpert* (рис. 4.3).

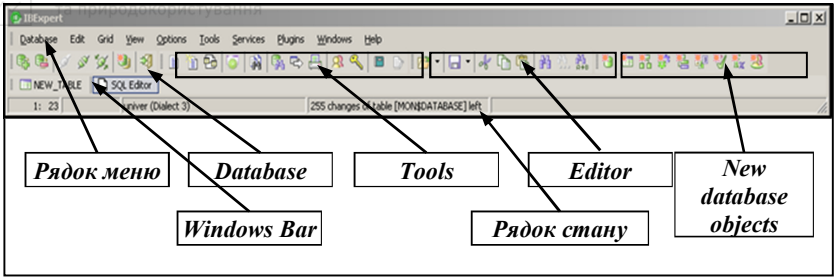


Рис. 4.3. Панелі інструментів *IBExpert*

Під рядком стану у лівій частині екрану знаходиться ***Database Explorer*** з приєднаним вікном ***SQL Assistent***.

Для початку роботи до бази даних необхідно приєднатись за допомогою команди **Database ⇒ Connect to Database**.

У вікні ***Database Explorer*** відображається дерево об'єктів бази даних, які можна створити або, які вже створені (рис. 4.4).

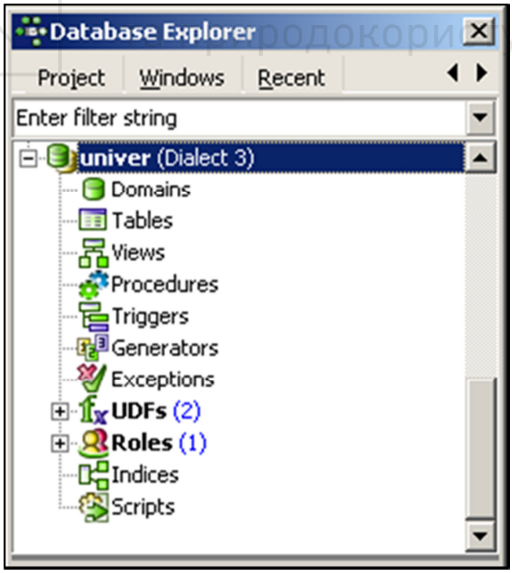


Рис. 4.4. Об'єкти бази даних

Будь-який об'єкт бази даних можна створити за допомогою:

1) команд меню **Database**: **Database⇒ New … .** Після задання необхідних даних потрібно натиснути кнопку **Compile**  . Після чого генерується *SQL*-код операції і результат виконання: *назва операції – Successful* – успішно або вказуються помилки. Для підтвердження виконаних дій необхідно натиснути кнопку **Commit**, для відміни – **Rollback**.

2) команд ***SQL***, використовуючи ***SQL Editor*** (команда **Tools ⇒ SQL Editor**, вкладка **Edit**). Після почергового вводу літер виводиться підказка-список з усіма командами чи об'єктами.

Потрібно зауважити, що в ***SQL Editor*** набирається і виконується тільки одна команда. Крапка з комою (;) в кінці команди не є обов'язковою. Для виконання команди необхідно натиснути кнопку **Execute**  , для внесення змін у базу даних – кнопку **Commit Transaction**  , для відміни команди – кнопку **Rollback Transaction**  . Інформація про успішно виконані операції зберігається на вкладці **History**. Для повторного виконання чи коригування необхідно клацнути вибраний рядок два рази лівою кнопкою миші і команда відобразиться на вкладці **Edit**.

Надалі пропонується використовувати ***SQL Editor*** паралельно з роботою в *IBExpert* для кращого вивчення команд мови *SQL*.

Контрольні запитання:

1) Що таке архітектура „клієнт-сервер"?
2) Яка програма є сервером баз даних?
3) Яка програма є клієнтом баз даних?
4) Які панелі інструментів є в програмі *IBExpert*?
5) Як створити базу даних?
6) Яка інформація необхідна для реєстрації бази даних?

5. Створення бази даних

5.1. Мова запитів SQL в середовищі СКБД IBExpert - Firebird

SQL (*Structured Query Language*) – структурована мова запитів, яка надає засоби створення і обробки даних в реляційних базах даних і є основною базовою мовою в різних СУБД.

Мова з'явилася відразу після появи реляційної алгебри, а її прототип був розроблений в кінці 70-х років в компанії IBMResearch. Він був реалізований в першому прототипі реляційної СКБД фірми IBM. Пізніше ця мова стала застосовуватись в багатьох комерційних СКБД і поступово стала стандартом для мов маніпулювання реляційними БД.

Американський інститут національних стандартів (*American National Standards Institute – ANSI*) та Міжнародна організація стандартів (*International Standards Organization – ISO*) займаються описом і підтримкою стандартів цієї мови. Перший міжнародний стандарт ***SQL*** був прийнятий в 1986 р. – *SQL1*. Розвиток інформаційних систем вимагав удосконалення ***SQL***. В 1992 р. розроблений стандарт *SQL2*, який фактично став проміжним етапом при переході від *SQL1* до *SQL3*. *SQL3*, який прийнятий в 1999, передбачає певні вдосконалення, зокрема введені нові типи даних і деякі об'єктно-орієнтовані можливості.

Команди мови ***SQL*** можна поділити на три категорії:

DDL (*Data Definition Language*) – ***мова визначення даних*** – складається з команд, які створюють, змінюють і знищують об'єкти (таблиці, індекси, представлення і так далі) у базі даних (**CREATE**, **ALTER**, **DROP**)

DML (*Data Manipulation Language*) – ***мова маніпулювання даними*** – це набір команд, що забезпечують додавання, модифікацію чи видалення даних в об'єктах бази даних, а також організацію запитів даних (**INSERT**, **UPDATE**, **DELETE**, **SELECT**).

DCL (*Data Control Language*) – ***мова керування даними*** – складається із засобів, які визначають чи дозволити користувачеві виконувати певні чи дії ні (**GRANT**, **REVOKE**).

По своїй суті ***SQL*** призначений тільки для опису взаємодії з базою даних, тому не містить засобів, необхідних для розробки програм і може використовуватись як їх частина.

В залежності від призначення ***SQL*** може використовувати один із рівнів реалізації:

1) ***ISQL*** (*Interactive SQL*) – ***інтерактивний SQL*** – дає можливість користувачам виконувати *SQL*-команди і отримувати результати за допомогою відповідних утиліт або спеціальних стандартних засобів для роботи з базами даних. В *InterBase* такими утилітами є Isql.exe і Wisql32.exe, у *Firebird* – Isql.exe.

2) ***Статичний SQL*** – фіксований, тобто заздалегідь підготовлений і записаний, а не сформований в процесі виконання програмного додатку *SQL*-код. Існує два види статистичного ***SQL***: вбудований і модульний. ***Вбудований SQL*** – ***ESQL*** (*Embedded SQL*) – це код ***SQL***, який вмонтований в текст програми, написаної на тій чи іншій алгоритмічній мові. *InterBase* підтримує вбудований ***SQL***. При використанні ***модульного SQL*** модулі, написані командами ***SQL***, приєднуються до додатку на етапі компонування.

3) ***DSQL*** (*Dynamic SQL*) – ***динамічний SQL*** – код, що генерується додатком під час його виконання. Динамічний ***SQL*** замінює статистичний в тих випадках, якщо необхідний код не може бути написаний під час створення додатку, а залежить від дій користувача під час роботи цього додатку.

4) ***PSQL*** (*Procedural*) – **процедурний *SQL*** – це код *SQL*, що включає конструкції **FOR**, **WHILE**, **IF**, які використовуються при написанні процедур, тригерів, скриптів.

В *InterBase* використовуються всі 4 рівні реалізації ***SQL***. Необхідно пам'ятати, що на різних рівнях доступні не всі команди ***SQL***.

5.2. Створення бази даних

Для створення бази даних з використанням утиліти *isql* (*isql* означає *interactive SQL* – інтерактивний *SQL*) в інтерактивному режимі, необхідно з папки **C:\Program Files\ Firebird\ Firebird_2_5\ BIN** запустити утиліту ***ISQL***.

В *isql* кожна команда закінчується крапкою з комою (;). Команду можна розбивати на декілька рядків, натискуючи клавішу <*Enter*>.

Для завершення роботи необхідно в командному рядку набрати команду **QUIT**;.

При наведенні правопису команд прийняті такі узгодження:

- службові слова виділені жирним шрифтом і написані великими літерами (**CREATE)**;
- слова, виділені курсивом і написані малими літерами (*пароль*), є ідентифікаторами, заданими користувачем;
- параметри в квадратних дужках ([ ]) можуть не задаватись;
- вертикальна лінія (|) визначає тільки один із можливих варіантів використання;
- фігурні дужки визначають обов'язкове включення однієї з конструкцій ({ }).

1. Створення бази даних

Спрощена конструкція оператора **CREATE DATABASE**:

CREATE {**DATABASE** | **SCHEMA**} *'ім'я_БД'*
USER *'ім'я_користувача'* **PASSWORD** *'пароль'*
[**PAGE_SIZE** [=] *число*]
[**DEFAULT CHARACTER SET** *код*]**;**

Слово **SHEMA** є синонімом слова **DATABASE**.

'ім'я_БД' – задає шлях і ім'я бази даних. Якщо в назві присутні пропуски, то використовуються подвійні лапки.

USER *'ім'я_користувача'* – задає ім'я користувача. Даний користувач є власником бази даних. Йому надається право повністю керувати нею, тобто створювати, змінювати і знищувати різні об'єкти, виконувати запити на вибірку, маніпулювати даними. Типовим ім'ям є **SYSDBA**. Цей користувач реалізує функції системного адміністратора бази даних.

PASSWORD *'пароль'* – задає пароль. Типовим є пароль **masterkey**. В цілях безпеки його рекомендується змінити.

В команді **CREATE DATABASE** лапки навколо шляху до файлу, імені користувача і пароля є обов'язковими.

PAGE_SIZE [=] *число* – задає розмір сторінки бази даних. Це може бути 1 024, 2 048, 4 096, 8 192, 16 384. За замовчуванням використовується 1 024. Бажано, щоб розмір сторінки співпадав з розміром кластера жорсткого диска.

DEFAULT CHARACTER SET *код* – встановлення кодування (набору символів), яке використовується за замовчуванням. Для Східної Європи необхідно задавати **WIN1251**. Якщо даний параметр не задано, то кодування встановлюється в **NONE**.

Приклад:

```
CREATE DATABASE 'D:\Students\univer.gdb'
USER 'SYSDBA' PASSWORD 'masterkey'
PAGE_SIZE 8192
DEFAULT CHARACTER SET WIN1251;
```

2. Знищення активної бази даних

```
DROP DATABASE;
```

3. Приєднання до бази даних

CONNECT *'ім'я_БД'* **[USER** *'ім'я_користувача'***]** **[PASSWORD** *'пароль'***];**

Приклад:

```
CONNECT 'D:\Students\univer.fdb'
USER 'SYSDBA' PASSWORD 'masterkey';
```

4. Від'єднання від бази даних

DISCONNECT {*ім'я_БД***| ALL | DEFAULT} ;**

ALL або **DEFAULT** від'єднує усі відкриті бази даних**.**

Контрольні запитання:

1) На які категорії можна поділити команди мови SQL?
2) Який синтаксис команди створення бази даних?
3) Який стандартний користувач і пароль задаються при створенні бази даних?
4) Як знищити базу даних?

6. Таблиці

6.1. Команди створення і знищення таблиць

Таблиці використовуються для збереження і перегляду даних.

Для створення таблиці виконується команда **Database ⇒ New Table**. Після цього задається структура таблиці. Бажано зразу ж правильно задавати ім'я таблиці, оскільки пізніше змінити його можна тільки копіюванням усіх даних у нову таблицю за допомогою команди **Copy object** … з контекстного меню таблиці у ***Database Explorer***.

Після створення у вікні **Table** на вкладці **Fields** можна переглядати і коригувати структуру таблиці. Потрібно знати, що *IBExpert* зберігає усю інформацію про таблиці (метадані) в спеціальних таблицях, які називаються ***системними*** (*system tables*). Імена системних таблиць починаються з „RDB$". І тому, для коригування будь-якого імені поля чи його типу або властивостей необхідно змінити домен або створити новий при виконанні команди **Edit Fields** з контекстного меню назви поля.

На вкладці **Constraints** задаються обмеження у вигляді первинних, зовнішніх, унікальних ключів і правил на ввід даних (**Checks**). Вкладка **Indices** використовується для створення ключів сортування. На вкладці **Data** вводяться дані. Вкладка **Master / Detals View** відображає головну таблицю і їй підпорядковану, що зручно для перегляду даних з двох таблиць.

На вкладці **DDL**, яка наявна при відображенні будь-якого об'єкта бази даних, відображається скрипт *SQL*-команд створення таблиці і пов'язаних з нею об'єктів.

5. Створення таблиці

CREATE TABLE *ім'я_таблиці*
(*ім'я_поля тип_даних* **[(***розмір_поля* **[**, *точність***])] [***обмеження***],**
ім'я_поля тип_даних **[(***розмір_поля* **[**, *точність***])] [***обмеження***],…,**
[PRIMARY KEY (*ім'я_поля***)],**
[UNIQUE (*ім'я_поля***)],**
[FOREIGN KEY (*ім'я_поля***) REFERENCES** *ім'я_таблиці1* **(***ім'я_поля_первинного_ключа_таблиці1***)**
[ON DELETE {NO ACTION|CASCADE|SET DEFAULT|SET NULL}]
[ON UPDATE {NO ACTION|CASCADE|SET DEFAULT|SET NULL}]],
[CHECK (*умова_обмежень***)]**
);

При створенні таблиці задається перелік полів, а також

тип_даних – задається зарезервований тип даних, або назва домену, або розрахункове поле (**COMPUTED BY** *вираз*). Розрахункове поле створюється на основі вже описаних полів. Хоча можливість створення такого поля існує, але нею користуватись не рекомендується, оскільки це суперечить теорії нормалізації баз даних. Для розрахунків необхідно використовувати представлення або процедури;

обмеження – задаються після типу даних для конкретного поля, або після опису всіх полів.

Знищувати таблицю можна в тому випадку, якщо вона не містить даних, не має зв'язку з іншими таблицями і не використовується в представленнях і процедурах.

6. Знищення таблиці

DROP TABLE *ім'я_таблиці***;**

Приклад:

Знищення таблиці **VYKLAD**.

DROP TABLE VYKLAD**;**

6.2. Типи даних

Тип даних вказується для кожного поля при створенні таблиці.

Дані в *Interbase* можуть бути наступних типів:

Числові

SMALLINT – цілі числа від -128 до +127.

INTEGER – цілі числа від -32 768 до +32 767.

BIGINT – цілі числа від -2 147 483 648 до +2 147 483 647.

FLOAT – дійсні числа від $3.4*10^{-38}$ до $3.4*10^{38}$ з 7 значущими цифрами, як додатні, так і від'ємні.

DOUBLE PRECISION – дійсні числа від $1.7*10^{-308}$ до $1.7*10^{308}$ з 15 значущими цифрами, як додатні, так і від'ємні.

NUMERIC[(*розмірність* [, *точність*])] – фіксований формат дійсних чисел, де *розмірність* – загальне число знаків (максимальне 18 знаків); *точність* – число знаків після коми.

DECIMAL[(*розмір* [, *точність*])] – фіксований формат дійсних чисел.

Дата, час

DATE – поле дати, задається у форматі dd.mm.yyyy

TIME – час, задається у форматі hh:mm:ss

TIMESTAMP – дата і час, задається у форматі dd.mm.yyyy hh:mm.

Текстові

CHAR(*розмір***) [CHARACTER SET** *код***] [(COLLATE** *код1***)]** – рядок символів фіксованої довжини, що містить будь-які друковані символи розмірністю від 0 до 32767.

VARCHAR (*розмір*) **[CHARACTER SET** *код***] [(COLLATE** *код1***)]** – рядок символів змінної довжини, що містить будь-які друковані символи розмірністю від 0 до 32 767.

Набір символів **CHARACTER SET** задається при створенні бази даних і використовується за замовчуванням для всіх текстових полів. Для підтримки кирилиці необхідно вказати код WIN1251. За допомогою даної опції можна перевизначити даний набір для конкретного поля.

За допомогою опції **COLLATE** *код1* можна задати порядок сортування символів. Існує два варіанти сортування WIN1251 і PXW_CYRL. При WIN1251 сортування відбувається відповідно до алфавіту спочатку по великим літерам, потім – по малим. В порядку сортування PXW_CYRL великі і малі літери мають однакову вагу, сортування відбувається в такому порядку: а А б Б в В … я Я.

Тип даних BLOB

Тип даних **BLOB** (*Binary Large OBject*) призначений для збереження великої кількості даних змінного розміру у двійковому вигляді, які не можна розміщувати в полях визначених типів, наприклад, графіка, цифровий звук, відеофрагменти, тексти великого обсягу.

Реалізація даних типу **BLOB** у базі даних дещо відрізняється від інших полів. На сторінці даних розміщується тільки ідентифікатор **BLOB,** а для самих об'єктів виділяється окреме місце.

Загальний опис типу:

BLOB [SUB_TYPE *число***] [SEGMENT SIZE** *розмір***]**

Якщо *число* дорівнює **1** або вказана константа **TEXT**, то це текст. Якщо *число* дорівнює **0** , то це дані невизначеного типу.

Опис типу даних для стовпця типу **CHAR, VARCHAR** або **BLOB-text** може включати пропозицію **CHARACTER SET** визначаючи специфічне кодування для вибраного стовпця. Інакше стовпець використовує визначене за замовчуванням для бази даних кодування. Якщо кодування бази даних змінене, всі стовпці згодом визначені мають нове кодування, але існуючі стовпці не змінюються.

За допомогою опції **COLLATE** вказується вибраний порядок сортування. Для кодування **WIN1251** допустимі порядки сортування **WIN1251, WIN1251_UA, PXW_CYRL.**

SEGMENT SIZE *розмір* – вказується розмір поля.

Домени

Домен – це набір усіх допустимих значень, які може приймати певне поле таблиці. Після створення домен може використовуватись при визначенні стовпців таблиць як додатковий користувацький тип даних.

7. Створення домену

CREATE DOMAIN *назва_домену* **[AS]** *тип_даних*
[DEFAULT { значення | NULL | USER}]
[NOT NULL] [CHECK (*умова_обмежень***)]**
[CHARACTER SET *код* **[COLLATE** *сортування***]];**

Для типу даних команда створення домену може включати:

- значення за замовчуванням **DEFAULT;**
- умову недопустимості невизначених даних **NOT NULL**;
- обмеження **CHECK.** При заданні умов замість імені поля вказується службове слово **VALUE**;
- кодування **CHARACTER SET** і порядок сортування **COLLATE** для типу **CHAR, VARCHAR** або **BLOB-текст**.

Приклад:

Створення домену **GOD**, який може приймати цілі значення, більші від 10, і за замовчуванням дорівнює 50.

CREATE DOMAIN GOD
AS INTEGER
DEFAULT 50
CHECK (**VALUE** > 10)**;**

Створення домену **DESCRIPT** з типом BLOB-текст і кодуванням

CREATE DOMAIN DESCRIPT
AS BLOB SUB_TYPE TEXT **SEGMENT SIZE** 80
CHARACTER SET WIN1251**;**

8. Зміна домену

ALTER DOMAIN *назва_домену*
{[SET DEFAULT { значення | NULL | USER}]
| [DROP DEFAULT]
| [ADD [CONSTRAINT] CHECK (*умова_обмежень***)]**
| [DROP CONSTRAINT]};

Дана команда може змінювати будь-які властивості існуючого домену, крім типу даних і опції **NOT NULL,** тобто дозволяє перевизначати або знищувати значення за замовчуванням, додавати або знищувати обмеження **CHECK.**

9. Знищення домену

DROP DOMAIN *назва_домену***;**

Домен можна знищити в тому випадку, якщо він не використовується в таблицях.

6.3. Обмеження

Обмеження (*Constraints*) – це сукупність правил, які встановлюють зв'язки між таблицями, а також, можуть обмежувати значення, що допустимі до вводу в поля таблиці.

Розрізняють обмеження поля і обмеження таблиці. ***Обмеження поля*** відноситься до одного конкретного поля і розміщується після типу даних. ***Обмеження таблиці*** відноситься до одного чи групи полів і розміщується в кінці таблиці після визначення усіх полів.

Типи обмежень:

1) задання за замовчуванням певного значення або **NULL**-значення або для текстових полів імені користувача бази даних

DEFAULT *значення* **| NULL | USER**

2) умова недопустимості невизначених даних (NOT NULL);

3) обмеження CHECK

[CONSTRAINT *ім'я_обмеження***] CHECK** *умова*

Ім'я обмежень бажано задавати, оскільки вони використовуються при пошуку помилок і знищенні. Якщо ім'я обмеження не задано, то воно генерується системою.

Приклади обмежень для числових полів

CHECK (*ім'я_поля* **[NOT] BETWEEN** 0 **AND** 6**)**
CHECK (*ім'я_поля* **>** 10000 **AND** *ім'я_поля* **<=** 2000000**)**
CHECK (*ім'я_поля* =10 **OR (***ім'я_поля* **>** 20 **AND** *ім'я_поля* <= 100**) OR** *ім'я_поля* **IS [NOT] NULL)**

Приклади обмежень для текстових полів

CHECK (*ім'я_поля* **[NOT] IN (**′software′**,** ′hardware′**,** ′other′**,** ′N/A′**))**
CHECK (*ім'я_поля* **[NOT] LIKE** ′**%**ware**%**′**)**
CHECK (*ім'я_поля* **[NOT] CONTAINING** *значення***)**
CHECK (*ім'я_поля* **[NOT] STARTING WITH** ′V′**)**
CHECK (*ім'я_поля* **= UPPER (***ім'я_поля***))**

4) визначення первинного ключа

[CONSTRAINT *ім'я_обмеження***] PRIMARY KEY,**

5) визначення унікального ключа

[CONSTRAINT *ім'я_обмеження***] UNIQUE,**

Поля, значення в яких не можуть бути невизначеними або повторюватись, і, які не є первинними ключами називаються ***унікальними***.

При визначенні первинного або унікального ключів таблиці обов'язково потрібно задавати обмеження **NOT NULL** для поля.

6) визначення зовнішнього ключа і зв'язок з іншою таблицею за визначеним полем. При зв'язку з іншою таблицею задаються умови цілісності для знищення записів ON DELETE,

або зміни ключового поля ON UPDATE. За замовчуванням підтримується опція NO ACTION.

[CONSTRAINT *ім'я_обмеження***] FOREIGN KEY (***ім'я_поля***) REFERENCES** *ім'я_таблиці1* **(***ім'я_поля_первин._ключа_таблиці1***)**

[ON DELETE { NO ACTION | CASCADE | SET DEFAULT|SET NULL }]

[ON UPDATE {NO ACTION | CASCADE | SET DEFAULT | SET NULL }]];

Варіанти дій:

NO ACTION (за замовчуванням) – неможливо змінити або видалити значення в головній таблиці (первинний ключ), поки існують відповідні значення в підпорядкованій (зовнішній ключ);

CASCADE – при зміні значень ключового поля (первинного ключа) в головній таблиці змінюються відповідно значення в зв'язаному стовпці підпорядкованої таблиці (зовнішній ключ). При видаленні записів з головній таблиці видаляються відповідні записи із підпорядкованої таблиці;

SET NULL – при зміні значень первинного ключа головної таблиці чи видаленні записів з головної таблиці значення стовпця, який є зовнішнім ключем у підпорядкованій таблиці, встановлюються в NULL;

SET DEFAULT – значення стовпця, який є зовнішнім ключем у підпорядкованій таблиці, встановлюються в значення, яке в списку значень первинного ключа головної таблиці записано за замовчуванням.

Приклад:

Створюється таблиця T1 з первинним ключем P1. Створюється таблиця T2 з зовнішнім ключем F2, яка зв'язується з таблицею T1 з первинним ключем P1. Зміни значень первинного ключа P1 таблиці T1 автоматично приводять до зміни ключа F2 таблиці T2. Видалення записів таблиці T1 приводить до встановлення NULL-значень зовнішнього ключа відповідних записів таблиці T2.

CREATE TABLE T1 (P1 **INTEGER NOT NULL PRIMARY KEY);**

CREATE TABLE T2
(F2 **INTEGER REFERENCES** T1(P1)
ON UPDATE CASCADE
ON DELETE SET NULL);

Визначення ключів або обмеження **CHECK**, якщо вони включають декілька полів, вказуються після визначення усіх полів таблиці. Перед обмеженнями варто задавати параметр **[CONSTRAINT** *ім'я_обмеження***]** для визначення імені обмеження.

Приклад:

Створення таблиці **PREDMET** і зв'язок її з таблицею **VYKLAD** по полю **VNOM**.

CREATE TABLE PREDMET
(PNOM **INTEGER NOT NULL PRIMARY KEY**,
PNAME **VARCHAR**(15),
VNOM **INTEGER** ,
CONSTRAINT FK_PREDMET_1 **FOREIGN KEY** (VNOM) **REFERENCES** VYKLAD (VNOM),
GOD **INTEGER CHECK**(GOD>0))**;**

6.4. Модифікація структури таблиці

Команда **ALTER TABLE** дає можливість змінити структуру таблиці. Потрібно зауважити, що дана команда може містити декілька операторів **ADD, DROP,** які розділяються комами.

10. Модифікація структури таблиці

За допомогою команди **ALTER TABLE** можна:

- додати поле в таблицю

ALTER TABLE *ім'я_таблиці* **ADD** *ім'я_поля тип_даних* **[(***розмір_поля* **[,** *точність***])] [***обмеження_поля***];**

• знищити поле з таблиці

ALTER TABLE *ім'я_таблиці* **DROP** *ім'я_поля***;**

• змінити назву поля, його тип або порядок в таблиці

ALTER TABLE *ім'я_таблиці*
ALTER [COLUMN] *ім'я_поля* **{ TO** *нове_ім'я_поля*
| TYPE *новий_тип_даних* **[(** *розмір_поля* **[,** *точність* **]**
| POSITION *номер_нової_позиції* **} ;**

• додати обмеження в таблиці

ALTER TABLE *ім'я_таблиці* **ADD [CONSTRAINT** *ім'я_обмеження***]** *обмеження***;**

Обмеження можуть включати створення первинного, унікального або зовнішнього ключа, задання умов за допомогою параметру **CHECK.** Синтаксис такий самий, як і при створенні таблиць. За допомогою опції **CONSTRAINT** *ім'я_обмеження* вказується ім'я обмеження. Якщо вона не задана, то ім'я обмеження задається системою.

• знищити обмеження з таблиці

ALTER TABLE *ім'я_таблиці* **DROP CONSTRAINT** *ім'я_обмеження***;**

Приклади:

1) ***Додавання нового поля в таблицю з використанням домену***

ALTER TABLE VYKLAD
ADD VNAME1 NAM **NOT NULL**,
ADD VNAME2 NAM**;**

2) ***Додавання нового поля в таблицю з вказанням типу поля***

ALTER TABLE VYKLAD
ADD ZARPL **DECIMAL** (7,2)**;**

3) Перейменування поля

ALTER TABLE VYKLAD **ALTER** ZARPL **TO** ZARPLATA;

4) Модифікація параметрів поля

ALTER TABLE VYKLAD **ALTER** VNAME **TYPE VARCHAR** (10)**;**

5) Знищення поля

ALTER TABLE VYKLAD **DROP** ZARPLATA**;**

6) Задання первинного ключа

ALTER TABLE VYKLAD
ADD CONSTRAINT PK_VYKLAD **PRIMARY KEY** (VNOM)**;**

7) Задання зовнішнього ключа

ALTER TABLE PREDMET
ADD CONSTRAINT FK_PREDMET_1 **FOREIGN KEY** (VNOM)
REFERENCES VYKLAD(VNOM)
ON DELETE CASCADE
ON UPDATE CASCADE;

8) Знищення ключа

ALTER TABLE VYKLAD
DROP CONSTRAINT PK_VYKLAD**;**

9) Створення розрахункового поля

ALTER TABLE VYKLAD
ADD NOVA_ZARP **COMPUTED BY** (ZARPLATA*2)**;**

6.5. Робота з записами

Дані в таблиці вводяться після задання структур таблиць і встановлення зв'язків між ними. При роботі із записами таблиці використовуються команди мови ***DML***, які передбачають ввід даних, їх модифікацію, знищення.

Інформація про кожен запис вводиться окремою командою **INSERT**.

11.Додавання записів у таблицю

INSERT INTO *ім'я_таблиці* **[(***ім'я_поля* **[,***ім'я_поля* **...])] VALUES (***значення* **[,***значення***...]);**

Приклад:

Ввід даних у таблицю VYKLAD

INSERT INTO VYKLAD (VNOM, VFAM, DATA, OKLAD, POSADA) **VALUES** (1001, ′Кириченко′, ′1.09.2003′, 560.80, ′Ст. викладач′**);**

12.Знищення записів із таблиці

DELETE FROM *ім'я_таблиці* **[WHERE** *умова_відбору***];**

Якщо не вказана умова відбору, то знищуються всі записи з таблиці. Структура таблиці зберігається.

Приклад:

Знищити дані про викладача, табельний номер якого рівний 105

DELETE FROM VYKLAD **WHERE** VNOM=105**;**

13. Модифікація записів у таблиці

UPDATE *ім'я_таблиці* **SET** *ім'я_поля = нове_значення* **[WHERE** *умова_відбору***];**

Якщо умова відбору не задана, тоді операція модифікації буде застосована до всіх записів.

Приклад:

Збільшити зарплату старшим викладачам на 12%

UPDATE VYKLAD **SET** OKLAD=OKLAD*1.12
WHERE POSADA=′Ст. викладач′**;**

6.6. Індекси

Індекс – це впорядкований вказівник на записи в таблиці. Індекс складається з пар значень: „значення поля” – „фізичне розташування цього поля”. Значення поля є впорядковані. Таким чином, по значенню поля, що входить в індекс, можна швидко знайти місце в таблиці, де розташований запис, що містить дане значення.

Індекси використовуються для:

1) прискорення виконання запитів. Індекси створюються для таблиць за тим полем, яке часто використовується при сортуванні або умовах відбору **WHERE**.

2) забезпечення унікальності значень для первинних і унікальних ключів. При додаванні нового запису відбувається швидкий пошук на наявність значення ключового поля в таблиці.

3) забезпечення цілісності бази даних. Використовуються для швидкої перевірки умов, які накладаються при створенні зовнішніх ключів.

При створенні первинних, унікальних і зовнішніх ключів індекси створюються автоматично.

14. Створення індексу

CREATE [UNIQUE] [ASC|DESC] INDEX *ім’я_індексу*
ON *ім’я_таблиці* **(***ім’я_поля* **[,***ім’я_поля* **...]);**

де

UNIQUE – індекс дозволяє вносити в таблицю тільки унікальні значення;

ASC|DESC – задає порядок сортування: **ASC** – в порядку зростання, **DESC** – в порядку спадання. Якщо дана опція не задана, то сортування відбувається в порядку зростання.

Приклад:

Створення індексу I1 для таблиці STUDENTS за полем SFAM

CREATE ASC INDEX I1

ON STUDENTS (SFAM)**;**

Індекс може включати декілька полів сортування. Наприклад, сортування по прізвищу, імені, по батькові.

Індекси рекомендуються створювати після внесення даних в таблицю.

15. Модифікація індексу

ALTER INDEX *ім'я_індексу* **{ACTIVE|INACTIVE};**

де

ACTIVE|INACTIVE – два стани індексу (активний, тобто може використовуватись в запитах, і неактивний – відключений).

16. Знищення індексу

DROP INDEX *ім'я_індексу* **;**

Якщо дані в таблиці часто змінюються, доповнюються або видаляються, то виникає необхідність оптимізації (оновлення) індексу. Для цього потрібно відключити його, а потім підключити заново, або знищити і створити новий з тими ж самими властивостями.

Використання індексу можна відстежити у повідомленні програми після виконання запиту – **PLAN (STUDENTS INDEX (I1))**. Якщо в запитах поле, за яким створений індекс, не використовується в умовах відбору або сортуванні, то відбувається ***прямий*** перебір всіх записів – **PLAN (STUDENTS NATURAL)**.

Контрольні запитання:

1) Де зберігаються дані?
2) Які типи даних ви знаєте?
3) Що таке домен?
4) Що означає NULL-значення?
5) Якою командою створюється таблиця?
6) Що таке обмеження?
7) Які є типи обмежень?
8) Для чого використовується первинний ключ?

9) Які особливості унікального ключа?
10) Як задаються обмеження на поле?
11) Які правила можуть задаватись при створенні зв'язку між таблицями?
12) Які команди використовуються для модифікації структури таблиці?
13) Які дії можна виконати за допомогою команди **ALTER TABLE... ALTER**?
14) За допомогою якої команди записи додаються у таблицю?
15) Якою командою можна видалити дані з таблиці?
16) Як додати розрахункове поле в таблицю?
17) Для чого використовуються індекси?

7. Запити

7.1. Виконання запитів в IBExpert

Запити використовуються для вибірки з таблиць даних, які задовольняють певним умовам.

З цією метою в *IBExert* використовується ***SQL builder*** (рис. 7.1), який викликається командою ***Tools ⇒ Query builder***.

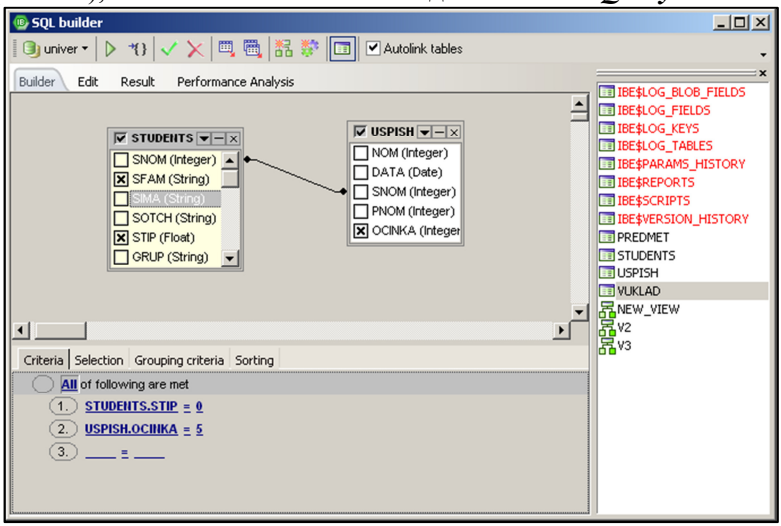


Рис. 7.1. Вікно ***SQL builder***

Зразу ж рекомендується відзначити опцію **Autolink tables** для відображення встановлених зв'язків між таблицями. Кнопка **Show tables**  відображає список таблиць. Вибрану таблицю необхідно перетягнути за допомогою мишки у бланк запиту вкладки **Builder**. Потім вибираються поля для виводу.

У нижній частині вікна розміщені чотири вкладки.

На вкладці **Criteria** задаються умови відбору. За допомогою кнопки **work area**  **⇒Add condition** створюється форма для задання умови. Поля у форму можна перетягувати мишкою з таблиць. Клацнувши на значку рівності, із списку можна вибрати інший знак операції. Умов можна задавати декілька. Опція **All** забезпечує виконання всіх умов. Її також можна змінити: **Any** – виконання будь-якої умови, **None** – не виконується жодна з умов, **Not all** – не виконується якась з умов.

На вкладці **Sorting** вибираються поля для сортування записів запиту. Запит готовий до виконання.

Вкладці **Selection** використовується для групових запитів. Вибирається поле, яке є назвою групи, і поле для розрахунків, для якого задається функція **Aggregate** для знаходження **MIN**, **MAX**, **SUM**, **AVG**, **COUNT**. Якщо поле для назви групи не вибрано, то розрахунки проводяться по всій таблиці.

На вкладці **Grouping criteria** подібно до першої вкладки задаються умови, які мають виконуватись у груповому запиті.

На вкладці **Edit**, яка розташована праворуч від вкладки **Builder**, відображається згенерований програмою *SQL*-запит.

На вкладці **Resalt**, яка з'являється після виконання запиту, відображаються результати .

7.2. Основні конструкції команди SELECT

17. Формування запитів

SELECT [DISTINCT] *перелік_полів*
FROM *перелік_таблиць*

[WHERE *умова***]**
[GROUP BY *ім'я_поля* **[HAVING** *умова_для_групи***]]**
[ORDER BY *перелік_полів***];**

де **DISTINCT** – для виключення появи однакових рядків (дублікатів);

SELECT *перелік_полів* – перелік полів і допустимих виразів над полями, які включаються у результуючу таблицю. Елементи списку розділяються комами. Для заміни списку всіх полів використовують *. Якщо використовується декілька таблиць, тоді перед назвою поля необхідно вказувати назву або псевдонім таблиці. Назва або псевдонім таблиці і назва поля розділяються крапкою. Назву поля у результуючій таблиці можна змінити за допомогою ключового слова **AS**

ім'я_таблиці.поле **AS** *нове_ім'я_поля*

Після службового слова **FROM** через кому вказується перелік таблиць. Для будь-якої таблиці через пропуск можна задати її друге ім'я (псевдонім).

Приклад:

Вивести прізвища студентів з таблиці **STUDENTS** і їх стипендію, збільшену у два рази. Вивід нової стипендії відбувається у стовпець з назвою **NOVSTIP**. В запиті для таблиці **STUDENTS** задано псевдонім **ST**.

SELECT ST.SFAM, ST.STIP*2 **AS** NOVSTIP
FROM STUDENTS ST**;**

ORDER BY *перелік_полів* – задає сортування записів в результуючій таблиці на основі одного чи декількох полів. Поле можна задавати числом, що вказує на положення цього поля у результуючій таблиці. Для кожного поля можна вказати ключ **DESC** для сортування записів у порядку спадання або ключ **ASC** для сортування у порядку зростання. За замовчуванням встановлюється сортування по зростанню.

Приклад:

Вивести номера, прізвища і імена студентів з таблиці **STUDENTS**. Дані відсортувати у зворотному алфавітному порядку по прізвищу.

SELECT SNOM, SFAM, SIMA
FROM STUDENTS
ORDER BY 2 **DESC**

GROUP BY *ім'я_поля* – групує запити по вказаному полю. Якщо необхідно вивести записи у групі, які задовольняють певній умові, використовують оператор **HAVING**.

WHERE *умова* – задає умови для вибору записів.

В умовах розділу **WHERE** можуть використовуватись оператори

1) порівняння: =, <> (!=), >, <, >=, <=
2) заперечення).
3) Оператор **IN (**множина**)** визначає набір допустимих значень, розділених комами.

Приклад:

Вивести прізвища студентів, які отримують стипендію 25 і 17.

SELECT SFAM
FROM STUDENTS
WHERE STIP **IN** (17, 25).

Символьні значення в наборі беруться в одиночні лапки.

4) Оператор **BETWEEN** A **AND** B визначає діапазон значень, в який має входити шукане значення.

Приклад:

Вивести інформацію про студентів, які складали іспити у період з 09.06.2005 по 12.06.2005.

SELECT SNOM
FROM USPISH

WHERE DATA **BETWEEN** '09.06.2005' **AND** '12.06.2005'

5) Оператор **LIKE** використовується для порівняння символьних даних відповідно до шаблону. Шаблон задається за допомогою одинарних лапок. Дозволяється використання наступних групових символів:

- знак підкреслення (_) замінює один символ;
- знак відсотків (%) замінює будь-яку послідовність символів;

Приклад:

Вивести список дисциплін, які у назві містять слово „програмування".

```
SELECT PNAME
FROM PREDMET
WHERE PNAME LIKE '%ПРОГРАМУВАННЯ%';
```

6) Оператор **CONTAINING** *значення* використовується для пошуку даних, що містять певне значення у вибраному полі.

Приклад:

Вивести список дисциплін, які у назві містять слово „програмування".

```
SELECT PNAME
FROM PREDMET
WHERE PNAME CONTAINING 'ПРОГРАМУВАННЯ';
```

7) Оператор **STARTING WITH** *значення* використовується для пошуку даних, що починаються з певного значення у вибраному полі.

Приклад:

Вивести список прізвищ студентів, які починаються на літеру 'О'

```
SELECT SFAM
FROM STUDENTS
```

WHERE SFAM **STARTING WITH** ′O′**;**

8) Оператор **IS [NOT] NULL** дозволяє вивести поля, які мають або не мають (**NOT**) невизначені значення.

Приклад:

Вивести номери залікових книжок студентів, які по різним причинам не здавали іспит.

SELECT SNOM
FROM USPISH
WHERE OCINKA **IS NULL;**

7.3. Функції у запитах

1) **COUNT(***ім'я_поля***)** – підрахунок кількості рядків або не-NULL значень поля.

Допускає використання **DICTINCT, ALL** або *****.

COUNT(DICTINCT *ім'я_поля***)** – підрахунок не-NULL значень поля, виключаючи дублікати.

COUNT([ALL] *ім'я_поля***)** – підрахунок не-NULL значень поля, включаючи дублікати. **ALL** використовується за замовчуванням.

COUNT(*) – підрахунок кількості рядків, включаючи NULL значення полів.

Приклад:

Підрахувати кількість записів у таблиці **STUDENTS:**

SELECT COUNT(*) **FROM** STUDENTS**;**

Приклад:

Підрахувати кількість студентів у таблиці **USPISH** без повторень.

SELECT COUNT(**DISTINCT** SNOM) **FROM** USPISH**;**

2) **SUM(***ім'я_поля***)** – розраховує суму усіх значень вибраного поля;

3) **AVG(***ім'я_поля***)** – знаходить середнє арифметичне значень вибраного поля;

4) **MAX(***ім'я_поля***)** – знаходить максимальне значення серед значень вибраного поля;

Приклад:

Знайти максимальну стипендію.

SELECT MAX(STIP) **FROM** STUDENTS**;**

5) **MIN(***ім'я_поля***)** – знаходить мінімальне значення серед значень вибраного поля:

Функції **COUNT**, **SUM**, **MIN**, **MAX**, **AVG** називаються ***агрегатними***.

6) **UPPER(***ім'я_поля***)** – перетворює символьну стрічку до верхнього регістра;

Приклад:

Вивести ПІБ студентів, причому прізвища великими літерами.

SELECT UPPER(SFAM, SIMA, SOTCH) **FROM** STUDENTS**;**

7) **LOWER(***ім'я_поля***)** – перетворює символьну стрічку до нижнього регістра;

8) **CAST(***ім'я_поля* **AS** *тип_даних***)** – перетворює дані поля до певного типу даних. Використовується в умовах пошуку для порівняння даних.

Приклад:

Вивести середній бал усіх студентів по таблиці **USPISH**.

SELECT AVG(CAST(OCINKA **AS FLOAT))** **FROM** USPISH**;**

Оскільки, поле OCINKA має цілий тип даних, то без використання функції **CAST** в результаті отримаємо ціле число. Щоб уникнути такої помилки, дані поля OCINKA приводимо до дійсного типу.

9) **EXTRACT(***частина* **FROM** *ім'я_поля***)** – вибирає інформацію про дату і час з даних типу **DATE**, **TIME**, **TIMESTAMP**.

Частина – може бути **YEAR** (значення – 0-5400), **MONTH** (1-12), **DAY** (1-31), **HOUR** (1-23), **MINUTE** (1-59), **SECOND** (0-59.9999), **WEEKDAY** (0-6, 0 – неділя , 1 – понеділок і т.д.), **YEARDAY** (1-366).

Тип даних, які є результатом функції – **SMALLINT**, для секунд – **DECIMAL**(6,4).

Приклад:

Вивести інформацію про всіх студентів, які здали іспити у місяці червні.

```
SELECT SNOM  FROM USPISH
WHERE EXTRACT(MONTH FROM DATA)=6;
```

7.4. Групові запити

Фраза **GROUP BY** дає можливість поділити множину рядків після застосування фільтрації (фраза **WHERE**) на групи за ознакою однакових значень в одному чи кількох стовпцях.

Найчастіше, групування відбувається по тому полю, для якого в умові вказується слово „кожен". Наприклад, для кожного студента, з кожного предмета, кожного дня.

Приклад:

Вивести середній бал для кожного студента по таблиці **USPISH**.

```
SELECT SNOM, AVG(CAST(OCINKA AS FLOAT))
FROM USPISH
GROUP BY SNOM;
```

Фраза **HAVING** дає змогу накладати умови на групи. Зрозуміло, що вона використовується за наявності фрази **GROUP BY**. В умовах фрази **HAVING** можна використовувати агрегатні функції.

Приклад:

Вивести номера студентів, середній бал яких більше 4.

```
SELECT SNOM, AVG(CAST(OCINKA AS FLOAT))
FROM USPISH
GROUP BY SNOM
HAVING AVG(CAST(OCINKA AS FLOAT))>4;
```

7.5. Об'єднання таблиць однакової структури

Оператор **UNION** використовується для об'єднання записів таблиць, які мають однакові структури, виключаючи дублікати. Для виведення усіх даних разом з **UNION** використовується оператор **ALL**.

Приклад:

Вивести прізвища, імена і по батькові усіх студентів і викладачів.

```
SELECT SFAM, SIMA, SOTCH FROM STUDENTS
UNION [ALL]
SELECT VFAM, VIMA, VOTCH FROM VYKLAD;
```

7.6. Встановлення зв'язків між таблицями

При використанні декількох таблиць у запиті, їх потрібно об'єднати по ключових полях.

Перший спосіб полягає у заданні рівності ключових полів в умові **WHERE**:

Приклад:

Вивести прізвища студентів і їх оцінки.

```
SELECT STUDENTS.SFAM, USPISH.OCINKA
FROM STUDENTS, USPISH
WHERE STUDENTS.SNOM=USPISH.SNOM;
```

Другий спосіб полягає у встановленні зв'язків між таблицями по ключових полях:

INNER JOIN – встановлює внутрішнє об'єднання, в якому вибираються тільки ті записи, які містять співпадаючі значення в полях зв'язку;

LEFT [OUTER] JOIN – встановлює лівостороннє зовнішнє об'єднання, тобто таке, в якому вибираються всі записи з лівої таблиці і записи з правої таблиці, для яких значення поля зв'язку співпадає із значенням поля зв'язку лівої таблиці. Якщо таких записів не існує у правій таблиці, то в результаті виконання запиту поля, які виводяться, приймають значення **NULL**.

RIGHT [OUTER] JOIN – встановлює правостороннє зовнішнє об'єднання, тобто таке, в якому вибираються всі записи з правої таблиці і записи з лівої таблиці, для яких значення поля зв'язку співпадає із значенням поля зв'язку правої таблиці;

FULL [OUTER] JOIN – створює повне зовнішнє об'єднання, в якому вибираються всі значення із лівої і правої таблиць.

Приклад:

Вивести прізвища студентів і їх оцінки.

```
SELECT STUDENTS.SFAM, USPISH.OCINKA
FROM STUDENTS INNER JOIN USPISH
ON STUDENTS.SNOM=USPISH.SNOM;
```

Приклад:

Вивести прізвища всіх студентів і їх оцінки.

```
SELECT STUDENTS.SFAM, USPISH.OCINKA
FROM STUDENTS LEFT JOIN USPISH
ON STUDENTS.SNOM=USPISH.SNOM;
```

Якщо записів у таблиці **USPISH** для студента з деяким номером немає, то поле **OCINKA** приймає значення **NULL.**

Якщо зв'язки між таблицями не встановлювати, а просто вказати поля, які виводяться, то в результаті отримаємо „декартовий добуток", тобто усі можливі комбінації поєднання полів.

Приклад: Використання трьох таблиць в запиті.

Вивести прізвища викладачів, дату іспиту і назву предмета.

SELECT DISTINCT VYKLAD.VFAM**,** USPISH.DATA**,** PREDMET.PNAME
FROM VYKLAD **INNER JOIN** PREDMET **INNER JOIN** USPISH
ON USPISH.PNOM=PREDMET.PNOM
ON PREDMET.VNOM=VYKLAD.VNOM**;**

7.7. Вкладені запити

Вкладений запит (підзапит) – це запит, результат виконання якого використовується в іншому запиті. Вкладений запит розміщується в середині іншого запиту у конструкціях **WHERE**, **HAVING**.

Підзапити бувають двох типів: ***прості*** і ***корельовані***.

Простий (незалежний) підзапит – це підзапит, обчислення якого здійснюється один раз, а множина значень, отримана в результаті виконання, використовується під час обробки зовнішнього запиту.

Корельований (залежний) запит – це підзапит, обчислення якого залежить від основного запиту, тобто у під запиті використовуються значення полів зовнішнього запиту.

Приклад:

Вивести номери дисциплін і оцінки, вищі від середньої.

SELECT PNOM, OCINKA
FROM USPISH
WHERE OCINKA > (**SELECT AVG**(OCINKA) **FROM** USPISH)**;**

Даний підзапит є простим.

При використанні в умовах порівняння підзапит повинен повертати лише одне значення.

Якщо підзапит повертає декілька значень, то він повинен використовуватись як аргумент операторів **EXISTS**, **ANY**, **ALL**, **SOME, IN.**

Результатом виконання оператора **EXISTS** є значення „істинно", якщо він здійснює будь-яке виведення записів, і „хибно" в протилежному випадку. Підзапит, який використовується в якості аргументу оператора **EXISTS,** може повертати декілька значень або жодного.

Приклад:

Вивести список студентів з таблиці **STUDENTS**, якщо куратором призначено викладача Петренка.

SELECT SFAM **FROM** STUDENTS
WHERE EXISTS (**SELECT** VFAM **FROM** VYKLAD **WHERE** VFAM='Петренко'**);**

Вивід прізвищ студентів залежить від результату підзапиту. Тобто, якщо викладач Петренко існує, то вивід прізвищ відбувається, в іншому випадку – ні.

Приклад:

Визначити викладачів, які читають хоча б один курс лекцій.

SELECT VFAM **FROM** VYKLAD
WHERE EXISTS (**SELECT * FROM** PREDMET **WHERE** VYKLAD.VNOM=PREDMET.VNOM**);**

Даний підзапит є корельованим (залежним). Для кожного рядка з основного запиту вибирається поле **VNOM** з таблиці **VYKLAD**, потім виконується підзапит, враховуючи значення цього поля, і по результату цього підзапиту відбувається вивід прізвища викладача, або – ні.

Приклад:

Визначити викладачів, які не читають жодного курсу лекцій.

SELECT VFAM **FROM** VYKLAD

WHERE NOT **EXISTS** (**SELECT * FROM** PREDMET **WHERE** VYKLAD.VNOM=PREDMET.VNOM)**;**

Існує ще три оператори, що орієнтовані на підзапити – це **ANY**, **ALL** та **SOME**. Але вони відрізняються від **EXISTS** тим, що в підзапитах використовуються разом з реляційними операторами. Якщо запит повертає множину значень, то використання **ANY** означає порівняння з хоча б одним значенням з даної множини (логічний оператор **OR**), а використання **ALL** – з усіма значеннями (логічний оператор **AND**). Оператори **ANY** та **SOME** взаємозамінні, тому в поданих прикладах будуть працювати однаково.

Якщо підзапит не повертає жодного значення, то оператор **ANY** автоматично повертає хибне значення, а оператор **ALL** – істинне. В першому випадку виконання основного запиту не відбувається, в другому випадку це приводить до виведення результатів основного запиту без будь-яких умов.

Таблиця 7.1.

Використання операторів ANY, ALL з арифметичними операціями

=ANY	Дорівнює будь-якому із множини значень, отриманих підзапитом.
<>ANY	Використання не має змісту. Умова ігнорується.
=ALL	Використання не допустиме.
<>ALL	Не дорівнює ні одному значенню з результату підзапиту.
>ANY, (>=ANY)	Більше (більше або рівне) від будь-якого отриманого числа. Еквівалентно > (>=) від найменшого отриманого числа.
<ANY, <=ANY	Менше (менше або рівне) від будь-якого отриманого числа. Еквівалентно < (<=) від найбільшого отриманого числа.
>ALL, >=ALL	Більше (більше або рівне) всіх отриманих чисел. Еквівалентно > (>=) від найбільшого отриманого числа.

<ALL, <=ALL	Менше (менше або рівне) від всіх отриманих чисел. Еквівалентно < (<=) від найменшого отриманого числа.

Приклад:

Вивести прізвища студентів, які складали іспити (тобто, про них є інформація у таблиці **USPISH**).

SELECT SFAM **FROM** STUDENTS **WHERE** SNOM = **ANY** (**SELECT** SNOM **FROM** USPISH)**;**

Даний запит можна виконати за допомогою оператора **IN**.

SELECT SFAM **FROM** STUDENTS **WHERE** SNOM **IN** (**SELECT** SNOM **FROM** USPISH)**;**

Приклад:

Вивести дані про тих студентів, чиї оцінки вище від оцінок студента з номером 200107.

SELECT * **FROM** USPISH **WHERE** OCINKA > **ALL** (**SELECT** OCINKA **FROM** USPISH **WHERE** SNOM=200107)

7.8. Представлення

Представлення – це віртуальна таблиця, створена на основі запиту до реальних таблиць.

Представлення реалізується як *SQL*-запит, який можна зберегти. Представлення виконується кожен раз, коли відбувається до нього звертання.

18. Створення представлення

CREATE VIEW *ім'я_представлення* **[(***ім'я_поля* **[,** *ім'я_поля* **...])]**
AS
<select>
[WITH CHECK OPTION];

ім'я_поля – задає ім'я поля в представленні. Кількість імен стовпців, визначені в представленні, має відповідати кількості і порядку стовпців, перерахованих в **<select>.**

<select> – команда **SELECT**, яка вибирає дані з таблиць або інших представлень відповідно заданим умовам.

В *IBExpert* представлення **Views** створюється як будь-який об'єкт бази даних. В результаті формується *SQL*-команда, яка коригується користувачем. Крім того, у вікні ***SQL builder*** будь-який *SQL*-запит можна зберегти як представлення, якщо натиснути кнопку **Create View from SELECT**  .

Представлення можуть бути **модифіковані**, тобто до них можна застосувати команди **INSERT**, **DELETE** або **UPDATE**. Але це можливо в тому випадку, якщо усі поля представлення допускають наявність NULL-значень, а **<select>-**команда, що використовується в представленні, не містить підзапитів, агрегатних функцій, виразів **DISTINCT** і **HAVING.**

WITH CHECK OPTION – забороняє модифікацію представлень (застосування команд **INSERT**, **DELETE** або **UDATE** відповідно до представлень), якщо порушуються умови, задані в конструкції **WHERE**.

Представлення не можуть включати конструкцію **ORDER BY**.

Приклад:

Створити представлення, яке містить інформацію про студентів, які отримують стипендію.

CREATE VIEW STIPEN (NAME, STIPENDIA) AS
SELECT SFAM, STIP
FROM STUDENTS
WHERE STIP<>0**;**

19. Знищення представлення

DROP VIEW *ім'я_представлення***;**

Представлення можна видалити в тому випадку, якщо воно не використовується в інших представленнях.

Контрольні запитання:

1) Яка команда використовується для формування запитів?
2) Як задати сортування даних в запитах?
3) Як задаються умови відбору для запитів?
4) Для чого використовується опція **GROUP BY**?
5) Які агрегатні функції ви знаєте?
6) Яка функція використовується для роботи з даними типу **DATE**?
7) Коли використовується функція **CAST**?
8) Як накладається умова при групуванні?
9) Яке призначення оператора **UNION**?
10) Яким чином встановлюється зв'язок між двома таблицями в запитах?
11) Які є види зв'язків між таблицями?
12) Що таке вкладені запити?
13) Які оператори використовуються з вкладеними запитами?
14) Якого типу даних є значення, отримане у результаті виконання оператора **EXISTS**?
15) Що таке представлення?
16) Чим представлення відрізняються від запитів?
17) Які представлення називаються модифікованими?
18) Чи може бути створене представлення на основі іншого представлення?
19) Чи зберігаються дані, отримані в представленнях, у базі даних?

8. Генератори і тригери

8.1. Генератори

Генератор (*Generator*) – це механізм, який створює послідовний унікальний номер, який автоматично вставляється в стовпець під час таких операцій, як **INSERT** або **UPDATE**. Генератори, зазвичай, використовуються для створення унікальних значень для полів, які є первинними ключами.

20. Створення генератора

CREATE GENERATOR *ім'я_генератора***;**

При створенні генератора за замовчуванням його початкове значення дорівнює нулю.

21. Ініціалізація генератора

SET GENERATOR *ім'я_генератора* **TO** *ціле_число***;**

Генератору присвоюється початкове значення.

22. Функція GEN_ID

GEN_ID (*ім'я_генератора* **,** *ціле_число***);**

Дана функція збільшує поточне значення генератора на ціле число.

Приклад:

Створити генератор, який використовується для створення унікальних значень поля **SNOM**, який є первинним ключем таблиці **STUDENTS**, починаючи з 101.

```
CREATE GENERATOR GEN_STUDENTS;
SET GENERATOR GEN_STUDENTS TO 100;
INSERT INTO STUDENTS (SNOM, SFAM)
VALUES(GEN_ID(GEN_STUDENTS, 1), 'Іванчук');
```

В результаті додавання запису в таблицю **STUDENTS**, полю **SNOM** автоматично присвоюється значення 101.

Для автоматичного генерування значень полів функцію **GEN_ID** використовують в тригерах.

23. Знищення генератора

DROP GENERATOR *ім'я_генератора***;**

Генератор можна видалити, якщо він не використовується.

В *IBExpert* генератори **Generators** створюються як будь-який інший об'єкт бази даних командою **Database ⇒ New**

Generator. Після задання імені і початкового значення генеруються *SQL*-команди

CREATE SEQUENCE *ім'я_генератора***;**
ALTER SEQUENCE *ім'я_генератора* **RESTART WITH** *ціле_число***;**

SEQUENCE – означає „послідовність", офіційний термін *SQL*. Для більшої відповідності стандартам *SQL*, рекомендується використовувати даний термін як синонім **GENERATOR**. За замовчуванням початкове значення рівне нулю.

Аналогом функції **GEN_ID** є

NEXT VALUE FOR *ім'я_генератора***;**

Для знищення послідовності використовується команда

DROP SEQUENCE *ім'я_генератора***;**

8.2. Тригери

Тригер (*Trigger*) – це частина програмного коду, що асоціюється з таблицею або виглядом, яка автоматично виконує дії при додаванні, зміні або видаленні запису в таблиці або представленні.

24. Створення тригера

CREATE TRIGGER *ім'я_тригера* **FOR** *ім'я_таблиці*
[ACTIVE | INACTIVE]
{BEFORE | AFTER}
{DELETE | INSERT | UPDATE}
[POSITION *номер***]**
AS
[DECLARE VARIABLE *ім'я_змінної тип_змінної1***;]**
…
[DECLARE VARIABLE *ім'я_змінної тип_змінноїN***;]**
BEGIN
команди_InterBase
END

[ACTIVE | INACTIVE] – необов'язковий параметр, який визначає дію тригера після завершення транзакції. **ACTIVE** – тригер використовується (за замовчуванням). **INACTIVE** – тригер не використовується.

{BEFORE | AFTER} – обов'язковий параметр, який визначає коли відбудеться активізація тригера до події (**BEFORE**) чи після (**AFTER**).

{DELETE | INSERT | UPDATE} – визначає операцію над таблицею, яка викликає тригер для виконання.

[POSITION *номер***]** – визначає порядок виклику тригера для обробки однієї події, наприклад при додаванні запису в таблицю. Номер має бути цілим від 0 до 32 767, за замовчуванням дорівнює нулю. Тригер, який має менший номер, виконується раніше.

Синтаксис тригера складається з двох частин: заголовка і тіла. Заголовок включає команду **CREATE TRIGGER**, ім'я тригера, і вище наведені параметри.

Ім'я тригера пропонується формувати по такому правилу: ім'я таблиці_початкові літери параметрів тригера. Наприклад, **STUDENTS_BI0** – ім'я тригера для таблиці **STUDENTS,** який виконується перед (**BEFORE**) вставкою (**INSERT)** даних з початковим номером 0.

Тіло тригера починається після слова **AS**, містить перелік описів локальних змінних і блок операторів.

Кожна змінна описується окремо за допомогою команди

DECLARE VARIABLE *ім'я_змінної тип_змінної***;**

де *тип_змінної* – стандартні типи даних *InterBase*.

В кінці кожної команди ставиться крапка з комою (**;**). Опис локальних змінних може бути відсутній.

Блок операторів береться в операторні дужки **BEGIN … END**. Він може містити *DML-SQL*-конструкції, а також, програмні конструкції **FOR SELECT…DO, IF-THEN, WHILE...DO** та інші. Кожна команда в блоці завершується крапкою з комою (**;**). Але крапка з комою є стандартним розділювачом команд в *SQL* при написанні скриптів. Отже, щоб уникнути помилок при виконанні, необхідно перед командою

створення тригерів, змінювати роздільник команд скрипту на інший символ, а після завершення процедури відновлювати крапку з комою назад.

Це може бути будь-який символ, який не використовується в даній команді, наприклад ^ або **!!** .

Терм визначається перед створенням тригера командою

SET TERM *терм* **;**

А після завершення тригера – командою

SET TERM ; *терм*

Потрібно зазначити, що зміна терма використовується при роботі в *isql*-режимі або при написанні скриптів.

Тобто, в скрипті тригер повинний мати такий вигляд

```
SET TERM ^ ;
CREATE TRIGGER Pr1
…
END
^
SET TERM ; ^
```


Після **END** крапка з комою не ставиться. Завершенням команди є символ ^.

У блоці програмного коду тригера можуть використовуватись дві контекстні змінні: **NEW** і **OLD**. Змінна **OLD**.*ім'я_стовпця* відповідає за старі значення стовпця, змінна **NEW**.*ім'я_стовпця* – за нові значення.

Дана таблиця ілюструє використання контекстних змінних **NEW** і **OLD** в залежності від типу тригера і можливої дії зі змінною.

Таблиця 8.1.

Використання контекстних змінних NEW і OLD в тригерах

Тип тригера	Контекстні змінні			
	NEW		OLD	
	Читання	Зміна	Читання	Зміна
BEFORE INSERT	Y	Y	N/A	N/A
ALTER INSERT	Y	N/A	N/A	N/A
BEFORE UPDATE	Y	Y	Y	N/A
ALTER UPDATE	Y	N/A	Y	N/A
BEFORE DELETE	N/A	N/A	Y	N/A
AFTER DELETE	N/A	N/A	Y	N/A

Y – означає, що використання змінної в даному типі тригера можливе, **N/A** – означає, що ні.

Приклад:

Використати генератор **GEN_STUDENTS** як лічильник для поля **SNOM** при додаванні записів у таблицю **STUDENTS**.

```
CREATE TRIGGER STUDENTS_BI0 FOR STUDENTS
ACTIVE BEFORE INSERT POSITION 0
AS
BEGIN
  IF (NEW.SNOM IS NULL) THEN
    NEW.SNOM = GEN_ID(GEN_STUDENTS, 1);
END
```

25. Модифікація тригера

ALTER TRIGGER *ім'я_тригера*
[ACTIVE | INACTIVE]

[{BEFORE | AFTER} {DELETE | INSERT | UPDATE}]
[POSITION *номер***]**
[AS *тіло_тригера* **]**
[*терм***]**

При модифікації можна змінити статус тригера, порядок і спосіб його виконання, внести зміни у тіло тригера. Терм вказується, якщо модифікується тіло тригера.

Приклад:

Змінити статус активного тригера *TR1* на пасивний.

ALTER TRIGGER TR1 **INACTIVE;**

26. Видалення тригера

DROP TRIGGER *ім'я_тригера***;**

В *IBExpert* після виконання команди створення тригера відкривається вікно ***Trigger*** (рис. 8.1), в якому задаються усі параметри, описані вище.

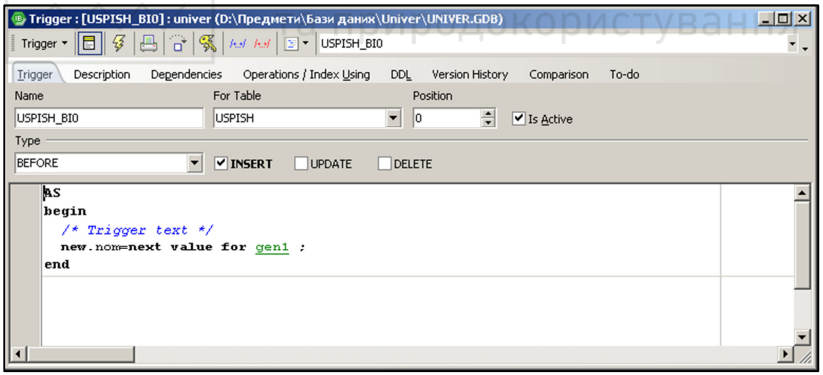


Рис. 8.1. Вікно створення тригера

Рис. 8.1 ілюструє приклад використання функції **NEXT VALUE FOR** замість **GEN_ID** при створенні тригера.

8.3. Виключення

Виключення (*Exeption*) – це повідомлення про помилку, яке задається програмістом.

Виключення використовуються в тригерах і процедурах при виникненні помилкових ситуацій.

27. Створення виключення

CREATE EXCEPTION *ім'я_виключення 'текст повідомлення'***;**

28. Модифікація виключення

ALTER EXCEPTION *ім'я_виключення 'текст повідомлення'***;**

29. Видалення виключення

DROP EXCEPTION *ім'я_виключення***;**

30. Використання виключення

EXCEPTION *ім'я_виключення***;**

Якщо дану команду використати у тригері чи процедурі, то при виникненні помилкової ситуації текст повідомлення буде відображатись на екрані. Це корисно, наприклад, при введенні даних у таблицю.

31. Опрацювання виключення

WHEN EXCEPTION *ім'я_виключення* **DO**
BEGIN
  /* обробка виключення ***/**
END;

Приклад:

Дана команда використовується при опрацюванні помилкової ситуації програмно.

8.4. Конструкції мови *SQL*

Дані конструкції використовуються в тригерах і процедурах.

1. Коментар

/* *коментар* ***/**

2. Конкатенація

′симв_рядок1′ || ′симв_рядок2′

3. Оператор умови IF ... THEN ... ELSE ...

IF (*умова***)**
THEN *оператор_1***;**
[ELSE *оператор_2***];**

умова – логічний вираз, який має значення *TRUE* або *FALSE*;

оператор_1 – виконується, якщо умова приймає істинне значення. Замість *оператор_1* може бути блок операторів, обмежений конструкцією **BEGIN … END**;

оператор_2 – виконується, якщо умова хибна. Замість *оператор_2* може бути блок операторів, обмежений конструкцією **BEGIN … END**.

В ***DSQL*** використовується конструкція

IF (*умова***)**
THEN *оператор_1***;**

4. Оператор циклу WHILE … DO …

WHILE (*умова***) DO** *оператор***;**

умова – логічний вираз, значення якого перевіряється перед кожним виконанням циклу;

оператор – оператор чи блок операторів **BEGIN … END**, який виконується, якщо умова приймає істинне значення.

5. Оператор SELECT

SELECT_команда INTO *список_змінних***;**

Синтаксис даного оператора подібний до синтаксису команди **SELECT**. На відміну від звичайної команди **SELECT** результатом даної команди є один рядок даних. Отримані дані записуються у змінні, вказані після ключового слова **INTO**. Перед змінною ставиться двокрапка (**:**) для того, щоб відрізнити ім’я змінної таблиці від імені стовпця.

6. Оператор циклу FOR … DO …

FOR SELECT_команда INTO *список_змінних* **DO** *оператор***;**

Дана команда використовується, якщо запит повертає більш, ніж один рядок даних і організовує цикл для обробки кожного рядка.

Перед змінною, яка використовується в **SELECT**-командах, ставиться двокрапка (**:**) для того, щоб відрізнити ім'я змінної таблиці від імені стовпця.

оператор – оператор чи блок операторів **BEGIN … END**, який виконується для кожного рядка, отриманого командою **SELECT**.

7. Оператор обробки помилок WHEN … DO …

WHEN
{ [EXCEPTION *ім'я_виключення***]**
| [SQLCODE *код_помилки***]**
| [GDSCODE *код_помилки***]**
| ANY }
DO
BEGIN
/* обробка помилки ***/**
END;

За допомогою даного оператора відбувається обробка помилок в тригерах і процедурах. Помилки можуть бути 3 типів: 1) виключення; 2) *SQL*-помилки, код яких розміщується у змінній *SQLCODE*; 3) *InterBase*-помилки, код яких ідентифікує змінну *GDSCODE*.

За допомогою конструкції **ANY** можна забезпечити обробку помилок будь-якого типу.

Приклад:

Приклади обробки помилок:

а) **WHEN SQLCODE** -803
DO

```
BEGIN
        /* обробка помилки */
END;
```

б)
```
WHEN  ANY
DO
BEGIN
        /* обробка помилки */
END;
```

8. Оператор обробки *SQL*-помилок WHENEVER …

WHENEVER {NOT FOUND | SQLERROR | QLWARNING}
{GOTO *мітка* **| CONTINUE}**

Будь-яка команда ***SQL*** після виконання формує код завершення, що розміщується у змінній **SQLCODE.** В залежності від виконання за допомогою даної команди може бути здійснений перехід до виконання підпрограми обробки помилок (**GOTO** *мітка*) або їх ігнорування (**CONTINUE**). Дана команда використовується в статистичному ***SQL*** і рекомендується для застосування перед будь-якою інструкцією ***SQL***, яка може завершитись помилкою.

Таблиця 8.2.

Значення змінної SQLCODE

SQLCODE	Повідомлення	Опис
0	SUCCESS	Успішне виконання
1 – 99	SQLWARNING	Попередження або інформаційне повідомлення
100	NOT FOUND	Кінець списку даних
<0	SQLERROR	Помилка. Команда не виконана.

Контрольні запитання:

1) Що таке генератор?
2) Якими командами задати створення і використання генератора?
3) Де може використовуватись функція **GEN_ID**?
4) Що таке тригер?
5) Які є види тригерів?
6) Чи може тригер використовуватись як окрема програма?
7) До яких дій по відношенню до таблиць приводить виконання тригерів?
8) Які оператори використовуються в тригерах?
9) Що означає слово **POSITION** 0 в описовій частині тригера?
10) Чи буде виконуватись тригер, в описовій частині якого вказане слово **INACTIVE**?
11) Який зміст мають змінні **NEW** та **OLD** в тілі тригера?
12) Для чого використовуються виключення?

9. Збережені процедури

9.1. Створення процедур

Збережена процедура (*Stored procedure*) – програма, що містить *SQL*-конструкції, яка зберігається у базі даних і виконується на стороні сервера.

32. Створення процедури

CREATE PROCEDURE *ім'я_процедури*
[(*вхідний_параметр тип* **[,** *вхідний_параметр тип ...***])]**
[RETURNS (*вихідний_параметр тип* **[,** *вихідний_параметр тип...***])]**
AS
[DECLARE VARIABLE *ім'я_змінної тип_змінної***;]**
…
[DECLARE VARIABLE *ім'я_змінної тип_змінної***;]**
BEGIN

команди_InterBase
END

Синтаксис збереженої процедури складається з двох частин: заголовка і тіла. Заголовок включає команду **CREATE PROCEDURE**, ім'я процедури, список вхідних параметрів і список параметрів, які повертаються з процедури. Дані списки можуть бути відсутні. В процедурах використовуються стандартні типи даних *InterBase*.

Тіло процедури починається після слова **AS**, містить перелік описів локальних змінних і блок операторів.

Кожна змінна описується окремо за допомогою команди **DECLARE VARIABLE**. В кінці кожної команди ставиться крапка з комою (**;**). Опис локальних змінних може бути відсутній.

Блок операторів береться в операторні дужки **BEGIN … END**. Він може містити *DML-SQL*-конструкції, а також, програмні конструкції **FOR SELECT…DO, IF-THEN, WHILE...DO** та інші.

При використанні процедур, як і тригерів, у скриптах або в *ISQL*-режимі використовуються терми.

Тобто, процедура повинна мати такий вигляд

SET TERM ^ ;
CREATE PROCEDURE Pr1
…
END
^
SET TERM ; ^

Після **END** крапка з комою не ставиться. Завершенням команди є символ ^.

33. Модифікація процедури

ALTER PROCEDURE *ім'я_процедури*
[(*вхідний_параметр тип* **[**, *вхідний_параметр тип ...***])]**
[**RETURNS (***вихідний_параметр тип* [, *вихідний_параметр тип...***])]**
AS

[DECLARE VARIABLE *ім'я_змінної тип_змінної*;
[DECLARE VARIABLE *ім'я_змінної тип_змінної*; …**]]**
BEGIN
команди_InterBase
END

За допомогою цієї команди можна змінити список вхідних і вихідних параметрів і тіло процедури. Заголовок і тіло процедури повинні бути включені в дану команду повністю. Синтаксис такий самий, як і в команді **CREATE PROCEDURE.**

34. Знищення процедури

DROP PROCEDURE *ім'я_процедури*;

Процедуру не можна знищити, якщо вона виконується або використовується в інших процедурах, представленнях, тригерах.

9.2. Виконання процедур

Збережені процедури можна викликати з клієнтських додатків, тригерів або інших процедур.

Існує два види збережених процедур: **SELECT** і **EXECUTE**.

SELECT-процедури (*selectable procedures*) або процедури-вибірки обов'язково повертають один або декілька наборів значень (якщо вони існують), які згруповані по рядках, і можуть використовуватись при створенні запитів разом з таблицями і представленнями.

EXECUTE-процедури (*executable procedures*) або виконані процедури можуть повертати тільки один набір значень, який перерахований у вихідних параметрах, або не повертати жодного значення у викликаючу програму.

SELECT-процедури і EXECUTE-процедури описуються однаково, але викликаються по різному.

SELECT-процедури викликаються за допомогою конструкції:

SELECT * FROM *ім'я_процедури* **[(** *факт_параметр* **[**, *факт_параметр*…**])];**

EXECUTE-процедури викликаються за допомогою конструкції:

EXECUTE PROCEDURE *ім'я_процедури* **[(** *факт_параметр* **[**, *факти_параметр, …***])]**

9.3. Додаткові конструкції мови SQL

35. Оператор виходу EXIT

EXIT;

Виконується вихід з циклу і перехід на останній **END** в процедурі. Використовується тільки в збережених процедурах.

36. Оператор SUSPEND

SUSPEND;

Використовується тільки в збережених процедурах для виведення результатів. Призупиняє роботу збереженої процедури до тих пір, поки додаток не відновить запит, повертає значення вихідних параметрів.

9.4. Приклади створення процедур

Приклад:

Приклад збереженої процедури, який ілюструє використання операторів циклу і перевірки умови.

```
CREATE PROCEDURE P1 RETURNS (R INTEGER)
AS
BEGIN
  R = 0;
  WHILE (R < 5) DO
  BEGIN
    R = R + 1;
    SUSPEND;
    IF (R = 3) THEN
   EXIT;
  END
END
```

Після виклику **SELECT * FROM** P**;** процедура поверне значення 1, 2 и 3 у викликаючий додаток.

Приклад:

Визначити максимальну стипендію.

```
CREATE PROCEDURE  PR2 RETURNS (Max_stip DECIMAL (8,2))
AS
BEGIN
 SELECT MAX(STIP) FROM STUDENTS INTO :Max_stip;
 SUSPEND;
END
```

Виклик

```
SELECT * FROM PR2;
```

або

```
EXECUTE PROCEDURE PR2;
```


Приклад:

Вивести прізвища студентів, які отримують вказану стипендію.

```
CREATE PROCEDURE  PR3  (ST DECIMAL (8,2))
RETURNS (Fam VARCHAR(15), Stip DECIMAL (8,2))
AS
BEGIN
 FOR SELECT SFAM, STIP FROM STUDENTS WHERE STIP=:ST  INTO :Fam, :Stip
 DO
 SUSPEND;
END
```

Виклик

```
SELECT * FROM PR3(40);
```

В *IBExpert* після виконання команди створення процедури ***Database ⇒ New Procedure*** відкривається вікно ***Procedure*** (рис. 9.1), яке можна використовувати для створення і виконання процедур.

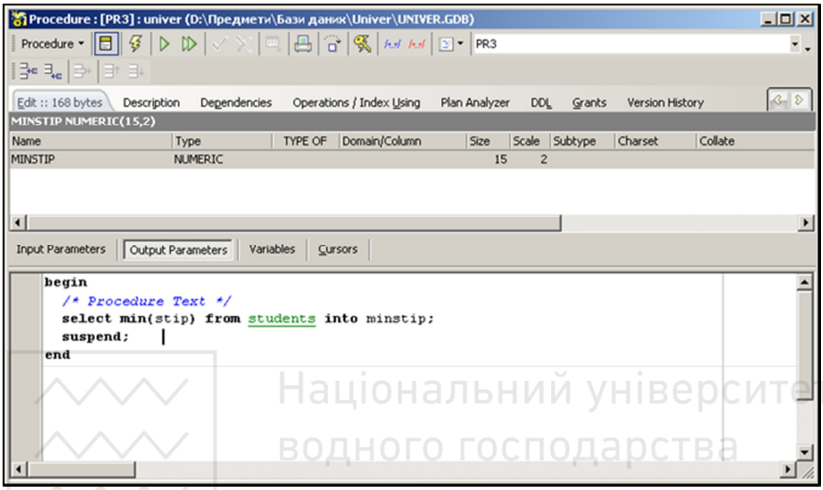


Рис. 9.1. Вікно створення тригера

Контрольні запитання:

1) Що таке збережені процедури?
2) Якою командою створюються збережені процедури?
3) Які команди і оператори може включати тіло процедури?
4) Які є види збережених процедур?
5) Яким чином викликаються збережені процедури?
6) Чим відрізняються SELECT-процедури від EXECUTE-процедур?

10. Транзакції у InterBase

10.1. Поняття про транзакції

Транзакція – це логічний блок команд з базою даних, який може бути успішно виконаний, якщо при цьому не

порушується цілісність бази даних, або відмінений – в протилежному випадку або за рішенням користувача.

Такими командами є INSERT/UPDATE/DELETE або SELECT.

Транзакція характеризується властивостями, які скорочено можна назвати ***ACID*** (*Atomicity, Consistency, Isolation, Durability* – атомарність, узгодженість, ізольованість, довговічність):

1) ***Атомарність.*** Виконуються усі команди транзакції або не виконується жодна з даного блоку. Якщо може виконатись тільки частина транзакції, то вона відміняється повністю.
2) ***Узгодженість.*** База даних знаходиться у стабільному, несуперечливому стані. Транзакція переводить базу даних у інший стан, який є стабільним і несуперечливим, тобто не повинна порушуватись цілісність бази даних.
3) ***Ізольованість.*** На роботу транзакції не повинні впливати інші транзакції. Результати виконання транзакції будуть доступні іншим транзакціям тільки після її підтвердження.
4) ***Довговічність.*** Після успішного завершення транзакції, результати її виконання повинні зберігатись у базі даних. Ніяка дія не може повернути базу даних у початковий стан, який був до виконання транзакції.

Дані функції виконує сервер. Програміст вирішує, які команди повинні виконуватись в транзакції. Бажано, щоб кожна транзакція включала мінімальну кількість команд і змінювала щонайменше даних. Виконання таких правил дозволить ефективно забезпечити одночасну роботу багатьох користувачів. Для підтвердження транзакції використовується команда **COMMIT**, для відміни – команда **ROLLBACK**. У першому випадку усі зміни вносяться у базу даних. У другому – база даних повертається до попереднього цілісного стану, і така ситуація називається ***відкатом*** транзакції. Крім того, система повинна забезпечити кожному користувачу такий режим

роботи, при якому не відчувалося втручання інших користувачів. Таке завдання СКБД називається ***паралелізмом транзакцій***.

Але при такій роботі виникають такі проблеми доступу до даних:

1) ***Проблема останньої зміни.*** Виникає в тому випадку, коли декілька користувачів коригують одні і ті ж самі дані, використовуючи початкові значення. Тоді частина змін може бути втрачена, оскільки наступна транзакція перепише зміни, зроблені попередньою, і в результаті будуть у таблицю внесені зміни, виконані останньою транзакцією. Розв'язуванням даної проблеми є послідовне внесення змін.
2) ***Проблема неправильного читання.*** Транзакція може зчитувати для роботи дані, які в даний момент змінюються або знищуються іншою транзакцією, тобто які не є остаточними. Тобто, користувач зразу ж отримує заздалегідь невірну інформацію. Для виключення даної проблеми виконується зчитування даних після внесення всіх змін.
3) ***Проблема неузгодженого читання.*** Виникає в тому випадку, якщо одна транзакція неодноразово зчитує одні і ті ж дані. А під час виконання першої транзакції друга змінює їх. Тому при повторному читанні перша транзакція отримує зовсім інший набір, що може привести до порушення їх цілісності або неузгодженості.
4) ***Проблема читання „фантомів".*** Транзакція вибирає дані з таблиці, а інша додає або знищує дані до завершення першої. Вибрані дані і робота з ними будуть некоректні.

Для вирішення таких проблем використовується ***блокування (ізоляція)*** – тимчасове обмеження на виконання деяких операцій обробки даних для забезпечення паралельної роботи користувачів.

Кожна транзакція має свій рівень ізоляції, який встановлюється під час її запуску і зберігається до її закінчення.

Визначені 4 рівні ізоляції:

1) ***0 рівень. Читання непідтверджених даних.*** (*Read Uncommitted).* Змінювати дані може тільки одна транзакція. Якщо другій транзакції необхідно змінити ті ж самі дані, то вона повинна дочекатись завершення першої.
2) ***1 рівень. Читання підтверджених даних.*** (*Read Committed*). Якщо перша транзакція почала зміну даних, то ніяка інша транзакція не може зчитати їх до завершення першої.
3) ***2 рівень. Повторюване читання.*** (*Repeatable Read*, *Snapshot*). Якщо перша транзакція зчитує даних, то ніяка інша транзакція не може їх змінити до завершення першої.
4) ***3 рівень. Впорядкований.*** (*Serializable*). Якщо перша транзакція звертається до даних, то друга транзакція не може додавати або знищувати дані, які можуть використовуватись під час роботи першої.

Кожен рівень блокування включає попередній і накладає свої умови. Реалізація кожного наступного рівня дедалі складніша.

Таблиця 10.1.

Вирішення проблем відповідно до рівня блокування

Рівень блокування	Проблеми			
	Фантомне читання	Неузгоджене читання	Неправильне читання	Остання зміна
Serializable	+	+	+	+
Repeatable Read	-	+	+	+
Read Committed	-	-	+	+
Read Uncommitted	-	-	-	+

10.2. Механізм реалізації транзакцій у Firebird

У сервері *Firebird*, як і в *InterBase*, реалізована багатоверсійна архітектура баз даних (*Multi Generation Architecture – MGA*).

Її суть полягає в тому, що всі зміни, що проводяться над певним записом таблиці, відбуваються насправді над копією цього запису, яка називається ***версією***. Для одного запису допустимо існування декількох версій відповідно до кількості транзакцій.

Завдяки *MGA* робота транзакцій реалізована з використанням механізму *MVCC* – управління паралельними доступом за допомогою багатоверсійності (*MVCC – MultiVersion Concurrency Control*).

Кожна транзакція має свій унікальний ідентифікатор (*transaction ID – TID#*). Кожна нова транзакція має номер більший, чим та, що почалася раніше.

Для реєстрації транзакцій використовуються ***сторінки обліку транзакцій*** (*Transaction Inventory Page – TIP*). Після початку виконання транзакції на даних сторінках з'являється повідомлення про номер транзакції, початок запуску і статус, що та є *активною – active*).

Якщо транзакція завершилась успішно і є *підтвердженою*, то вона отримала статус *committed*.

Відмінені транзакції мають статус *rolled back*.

Є ще один вид транзакцій з невизначеним статусом *limbo*, які так і називаються транзакції *Лімбо*.

Коли транзакція з певним ідентифікатором *TID#* намагається змінити запис, то з'являється версія цього запису з номером даного ідентифікатора в заголовку. Але після успішного завершення даної транзакції і отримання статусу *committed* ніяких змін із версією не відбувається.

Якщо наступна транзакція зчитує запис, то вона зчитує усі версії і вибирає із заголовків номера транзакцій,

При цьому відбувається додаткова перевірка статусів відповідних транзакцій на сторінках обліку.

Якщо транзакція, що створила дану версію, була відмінена, то така версія є сміттям (*garbage*), і її необхідно

знищити. Підтверджена версія з найбільшим номером є актуальною.

Процес видалення непотрібних версій записів називають *збиранням сміття*. Дану роль виконує транзакція, що зчитує запис. Після вибору активної версії запису усі інші версії, що залишилися, видаляються.

10.3. Параметри транзакцій

Роботою транзакцій можливо керувати шляхом задання їм певних параметрів. Налаштування параметрів транзакції визначається заданими константами.

Усі параметри можна поділити на групи:

1) **Режим доступу**:
   а) ***read*** – дозволяється тільки операції читання;
   б) ***write*** – дозволяються операції запису;

   За замовчуванням встановлюється режим ***read write***.
2) **Режим блокування**:
   а) ***wait*** – встановлює режим відтермінування рішення конфлікту (за замовчуванням);
   б) ***nowait*** – виникнення конфлікту призводить миттєво до встановлення помилки;
3) **Рівень ізоляції**:
   а) ***read_commited***, ***rec_version*** – надає можливість читати підтверджені дані інших транзакцій. Додатковий параметр ***rec_version*** дозволяє читати записи, що мають непідтверджені версії;
   б) ***read_commited***, ***no_rec_version*** – надає можливість читати підтверджені дані інших транзакцій. Додатковий параметр ***no_rec_version*** не дозволяє читати записи, що мають непідтверджені версії;
   в) ***concurrency*** – під час запуску транзакції створюється миттєвий „знімок” стану бази даних (тобто, запуск транзакцій на даний момент), тому зміни, зроблені в інших транзакціях, не доступні;
   в) ***consistency*** – аналогічно параметру ***concurrency***, але ще додається блокування таблиці на запис.

Контрольні запитання:

1) Що таке транзакція?
2) Що означає абревіатура ***ACID***?
3) Якими властивостями володіє транзакція?
4) Які проблеми виникають при роботі з транзакціями?
5) Які є рівні ізоляції транзакцій?
6) Назвіть статуси роботи транзакцій.
7) Як може завершуватись транзакція?
8) Які параметри можна задавати транзакціям?

11. Безпека даних та привілеї

11.1. Користувачі та їх права

Із використанням комп'ютерної техніки гостро постає проблема захисту, збереження і конфіденційності інформації.

Серед усіх користувачів бази даних головним є системний адміністратор, який зареєстрований під іменем SYSDBA з паролем „masterkey". Ім'я SYSDBA не може змінюватись. В цілях безпеки бази даних пароль необхідно зразу ж поміняти.

За замовчуванням системний адміністратор створює базу даних, володіє всіма правами над будь-яким її об'єктом, реєструє користувачів і надає їм права доступу до цих об'єктів. При реєстрації користувача вказується його ім'я і пароль. Ім'я не може перевищувати 31 символ і не є чутливим до регістра клавіатури. Пароль може складатись до 32 символів включно і є регістрочутливим, хоча для ідентифікації користувача використовуються тільки перші 8 символів.

Дані про користувачів бази даних зберігаються в особливій базі даних користувачів ISC4.gdb. Всі перевірки прав доступу здійснюються сервером *InterBase*. Про це необхідно пам'ятати, якщо відбувається копіювання бази даних з одного комп'ютера на інший.

Права – це дозвіл користувачу виконувати якусь операцію над певним об'єктом бази даних. До даних об'єктів відносяться таблиці, їх поля, представлення і збережені процедури.

Користувачам можуть надаватись

- Для таблиць і їх полів – права на виконання операцій **INSERT**, **DELETE**, **UPDATE**, **SELECT**, **REFERENCE** (для додавання записів, знищення, коригування, виконання запитів і встановлення обмежень для зовнішнього ключа таблиці).
- Для представлень і їх полів – права на виконання операцій **INSERT**, **DELETE**, **UPDATE**, **SELECT**.
- Для збережених процедур – право на виконання **EXECUTE**. Потрібно пам'ятати, що збережена процедура, яка після виконання має на меті модифікувати таблицю, буде виконана тільки у тому випадку, якщо її виконавець має права на модифікацію цієї таблиці.

37. Надання прав

GRANT *права* **ON [TABLE]** *ім'я_таблиці* [(*поля*)]
TO {*користувачі*| **PUBLIC}**
[WITH GRANT OPTION];

де *права* – список з одного або декількох прав, розділених комами. Права можуть бути наступні:

- **ALL [PRIVILEGES]** – повний дозвіл;
- **SELECT** – дозвіл на виконання запитів до таблиці;
- **INSERT** – дозвіл на додавання записів в таблицю;
- **UPDATE [(***ім'я_поля***[,** *ім'я_поля***...])]** – дозвіл на коригування даних таблиці;
- **DELETE** – дозвіл на видалення даних з таблиці.
- **REFERENCES [(***ім'я_поля***[,** *ім'я_поля***...])]** – дозвіл на встановлення зовнішніх ключів.

При наданні дозволу на коригування даних таблиці **UPDATE** чи встановлення зовнішніх ключів **REFERENCES** можна додатково вказувати імена полів, які можна коригувати чи визначати як зовнішній ключ.

користувачі – список, який включає одне або декілька імен користувачів, розділених комами.

поля – список полів таблиці, для яких встановлюється доступ.

PUBLIC означає передачу прав усім зареєстрованим користувачам.

WITH GRANT OPTION означає, що вказані користувачі можуть надавати надані їм права іншим користувачам.

Для представлень синтаксис команди подібний, тільки замість імені таблиці вказується ім'я представлення.

Права на виконання збережених процедур надаються командою

GRANT EXECUTE ON PROCEDURE *ім'я_процедури*
TO {*користувачі*| **PUBLIC**}
[WITH GRANT OPTION];

38. Відміна прав

REVOKE [GRANT OPTION FOR] *права*
ON [TABLE] *ім'я_таблиці*
FROM {*користувачі*| **PUBLIC**};

GRANT OPTION FOR – використовується для відміни уповноважень, наданих раніше користувачу за допомогою опції **WITH GRANT OPTION.**

Приклад:

Надати користувачу **STUD** права на виконання запитів до таблиці **STUDENTS** і модифікацію полів **SFAM** і **SIMA** цієї таблиці.

GRANT SELECT, UPDATE (SFAM, SIMA**)**
ON STUDENTS
TO STUD**;**

Приклад:

Надати користувачам **STUD** і **USER** права на виконання запитів до таблиці **STUDENTS** і модифікацію полів **SFAM** і **SIMA** цієї таблиці з можливістю передачі вказаних прав іншим користувачам.

GRANT SELECT, UPDATE (SFAM, SIMA**)**
ON STUDENTS
TO STUD, USER
WITH GRANT OPTION;

У програмі *IB Expert* для реєстрації нових користувачів використовується ***User Manager*** (***Tools*** ⇒ ***User Manager*** або піктограма  панелі інструментів ***Tools***), для надання прав – ***Grant Manager*** (***Tools*** ⇒ ***Grant Manager*** або піктограма  панелі інструментів ***Tools***).

У вікні *Grant Manager* для вибраного користувача можна задавати права доступу до об'єктів бази даних (рис. 11.1):

 – Право доступу для конкретного користувача на вибраний об'єкт (*Grant*).

 – Право доступу для конкретного користувача на вибраний об'єкт з можливістю надання цього права іншим користувачам (*WITH GRANT OPTION*).

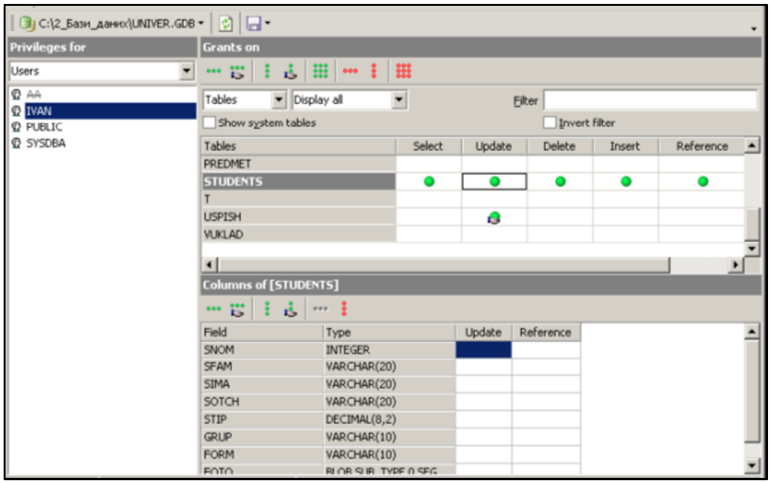


Рис. 11.1. Вікно *Grant Manager*

Користувачу Ivan надано повний доступ до таблиці до таблиці **STUDENTS** і право (з можливістю надання цього права іншим користувачам) на коригування даних таблиці **USPISH**.

11.2. Ролі

Дуже часто декілька користувачів виконують одну і ту ж роботу з об'єктами бази даних. Їм присвоюються однакові права. Але по певним причинам їх імена можуть змінюватись. Тому для зручності використовують механізм ролей.

Роль – це ім'я абстрактного користувача, який наділяється певними правами доступу до певних об'єктів бази даних. Після створення і визначення прав, роль можна присвоювати конкретному користувачу при час його реєстрації, а потім доступу до бази даних. Фактично, деякий користувач може мати більше, ніж одну роль, і використовувати їх в залежності від виду роботи з базою даних. Крім того, роль набагато легше коригувати, ніж змінювати права групи користувачів.

Ролі є об'єктами бази даних.

Спрощено порядок дії ролі можна записати таким чином:

39. Створення ролі

CREATE ROLE *ім'я_ролі*;

40. Надання прав ролі

GRANT *права* **ON [TABLE]** *ім'я_таблиці* [(*поля*)]
TO *ім'я_ролі*
[WITH GRANT OPTION];

41. Призначення конкретним користувачам даної ролі

GRANT *ім'я_ролі* **TO** *ім'я_користувачів*;

При приєднанні до певної бази даних користувач, крім імені і пароля, повинен вказувати надану йому роль. Потрібно пам'ятати, що під час одного сеансу може працювати тільки один користувач з даною роллю.

42. Знищення ролі

DROP *ім'я_ролі*;

Приклад:

Створити роль **WRITER** і надати їй право коригування таблиці **STUDENTS** з можливістю передачі вказаних прав іншим користувачам. Користувачам **STUD** і **USER** присвоїти дану роль.

CREATE ROLE WRITER;

GRANT INSERT, UPDATE, DELETE ON STUDENTS
TO WRITER
WITH GRANT OPTION;

GRANT WRITER **TO** STUD, USER;

У програмі *IB Expert* реєстрація нових ролей відбувається на вкладці ***Roles*** інструмента ***User Manager*** (***Tools ⇒ User Manager***). На вкладці *Membership* користувачам можна задавати певні ролі (*Users –> Roles*) або до обраної ролі підключати вибраних користувачів (*Roles –> Users*). ***AO*** – надання прав системного адміністратора (рис. 11.2).

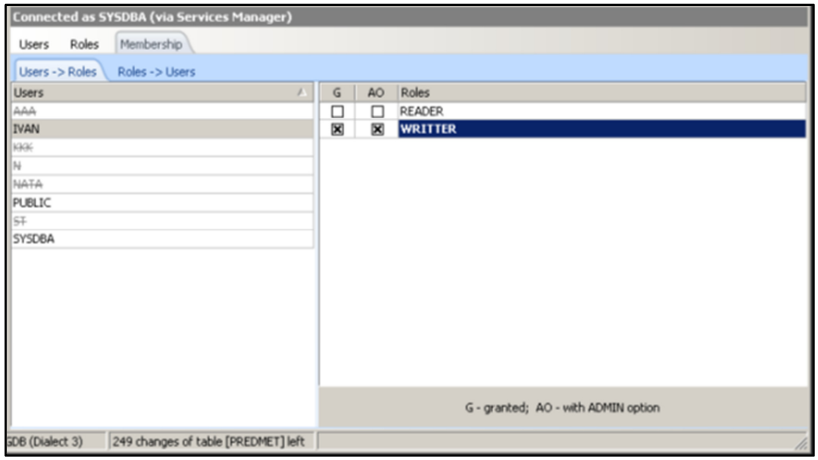


Рис.11. 2. Вкладка *Membership* вікна *User Manager*

Для надання прав ролям використовується ***Grant Manager*** (***Tools ⇒ Grant Manager***).

Контрольні запитання:

1) Яку інформацію містить зареєстрований запис про користувача?
2) Який користувач має повний доступ до бази даних?
3) Як зареєструвати користувача у програмі *IB Expert*?
4) Які права мають користувачі?
5) Яка команда використовується для надання прав користувачам?
6) Що таке ролі?
7) Для чого створюються ролі?
8) Який механізм дії ролі?
9) Як відмінити права, надані користувачу?
10) Як знищити роль?

Розділ ІІІ. Лабораторні роботи

Лабораторна робота № 1

Тема: Створення бази даних.

Завдання до виконання:

1. Завантажте програму *IBExpert.*

2. За допомогою команди **Database ⇒ New Database** створіть на диску **D:\Students** у своїй папці базу даних **Univer.gdb** для користувача **SYSDBA** з паролем **masterkey**. Розмір сторінки задайте 4096, кодування WIN1251, **Server** – Remote, **Server name** – localhost. Після чого відбувається перенаправлення на сторінку реєстрації, де вказуються **Server Version** – Firebird 2.5 і псевдонім бази даних. (Для використання створеної бази даних, наприклад на іншому комп'ютері, її необхідно зареєструвати за допомогою команди **Database ⇒ Register Database**).

3. За допомогою команди **Database ⇒ Connect to Database** відкрийте базу даних **UNIVER**.

4. Створіть домен **NAM** для текстових даних, що містять не більше 20 символів, задавши кодування і порядок сортування **WIN1251**.

5. За допомогою команди **Database ⇒ New Table** створіть таблицю **STUDENTS** з полями

Таблиця **STUDENTS**

SNOM	SFAM	SIMA	SOTCH	STIP	GRUP	FORM	FOTO
200101	Іванчук	Сергій	Петрович	75,00	ПМ-32	платна	<null>

SNOM – номер залікової книжки студента є первинним ключем, задаються значення Not Null і PK. Поля **SFAM** – прізвище, **SIMA** – ім'я, **SOTCH** – по батькові студента, для яких задайте тип даних у вигляді домену **NAM**. Поле **GRUP** – група, в якій навчається студент, для нього задайте значенням за замовчуванням назву своєї групи в одинарних лапках. Поле **STIP** – стипендія – має тип даних **NUMERIC** або **DECIMAL** з 2

знаками після коми. Для поля **FOTO** – фото студента – вибирається тип даних **BLOB SUB_TYPE** BYNARY **SIZE** 2048. Для поля **FORM** – форма навчання – задайте обмеження, що допускають ввід тільки двох значень ’платна’ або ’бюджет’.

6. За допомогою команди **Database ⇒ New Table** створіть таблицю **USPISH** з полями

Таблиця **USPISH**

NOM	DATA	SNOM	PNOM	OCINKA
1	13.06.2005	200101	301	5

NOM є первинним ключем, для поля **DATA** – дата здачі іспиту – тип даних **DATE**. **OCINKA** – оцінка за іспит є цілим. Для поля **OCINKA** допускається ввід цілих значень від 1 до 5 включно або NULL. За замовчуванням встановлюється значення **NULL** (не визначено).

7. За допомогою **SQL Editor** в командному режимі створіть таблицю **VYKLAD** з полями

Таблиця **VYKLAD**

VNOM	VFAM	VIMA	VOTCH	KAF	POSADA	OKLAD
10001	Бачишина	Лариса	Дмитрівна	ПМ	Ст. викладач	710,40

VNOM – табельний номер викладача, є цілим, не допускає невизначених значень, задається як первинний ключ таблиці. **VFAM** – прізвище, **VIMA** – ім’я, **VOTCH** – по батькові викладача. Для прізвища, імені, по батькові задайте тип даних у вигляді домену **NAM**, **KAF** – місце роботи. **POSADA** – посада викладача, **OKLAD** – оклад викладача. Для окладу викладача виберіть тип даних **NUMERIC** або **DECIMAL**, вказавши 2 знаки після коми.

8. Створіть таблицю **PREDMET** з полями

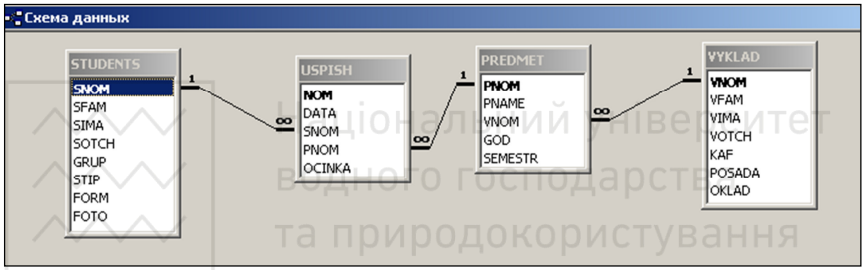
Таблиця **PREDMET**

PNOM	PNAME	VNOM	GOD	SEMESTR
301	Бази даних	101	100	5

PNOM – номер предмета в навчальному плані є цілим і є первинним ключем**. PNAME** – назва предмета. **GOD** – кількість годин по даному предмету. **SEMESTR** – семестр, в якому читається предмет. Для семестру задайте обмеження від 1 до 10.

9. На вкладці **Constraints** вікна **Table** для таблиці **USPISH** задайте зовнішній ключ для зв'язку з таблицею **STUDENTS** по полю **SNOM.** Задайте умови цілісності, що задовольняють умовам каскадування. При створенні таблиці **PREDMET** задайте зв'язок у вигляді зовнішнього ключа з таблицею **VYKLAD** по полю **VNOM**. Задайте умови цілісності: при знищенні – **SET NULL**, при зміні – **CASCADE.**

10. В зошит запишіть структури таблиць і схему даних.



11. Завершіть роботу.

Лабораторна робота № 2

Тема: Коригування бази даних.

Завдання до виконання:

1. Завантажте програму *IBExpert.*
2. Відкрийте базу даних **Univer.**
3. За допомогою команди **Tools ⇒ SQL Editor** відкрийте вікно для вводу SQL-команд і виконайте наступні дії:
   1) задайте обмеження для поля **SEMESTR** таблиці **PREDMET** так, щоб значення данного поля були >0 і <=10;
   2) встановіть зв'язок таблиці **USPISH** з таблицею **PREDMET** по полю **PNOM**. Задайте умови цілісності, що задовольняють умовам

каскадування;

3) додайте у таблицю **PREDMET** поле **POLE** типу **FLOAT**;
4) змініть назву поля **POLE** на **POLE1**;
5) змініть тип даних на **VARCHAR(15)**;
6) розташуйте дане поле третім по порядку;
7) видаліть поле **POLE1**;
8) введіть дані у таблиці **VYKLAD** і **PREDMET** (не менше 6 записів);
9) видаліть запис про викладача, прізвище якого **Бачишина**;
10) модифікуйте у таблиці **PREDMET** запис про предмет Бази даних, задавши семестр 6 і кількість годин 108;
11) створіть розрахункове поле **OPLATA**, в якому вираховується податок 13% від зарплати;
12) видаліть поле **OPLATA**;
13) закрийте *SQL Editor*.

4. В *IBExpert* відкрийте таблицю **STUDENTS**, введіть 7 значень у дану таблицю. Закрийте таблицю.

5. Відкрийте таблицю **USPISH**, введіть 15 значень у дану таблицю. Закрийте таблицю.

6. Завершіть роботу.

Лабораторна робота № 3

Тема: Прості запити. Групові операції. Використання агрегатних функцій.

Завдання до виконання:

1. Завантажте програму *IBExpert*.
2. Відкрийте базу даних **Univer**.
3. Використовуючи команду **SELECT** у командному вікні *SQL Editor*, виконайте наступні запити:
   1) Вивести на екран прізвища, імена та розмір стипендії всіх студентів. Дані відсортувати в алфавітному порядку по прізвищах.
   2) Вивести розрахунок стипендії після підвищення її

у два рази для кожного студента. Задати назву для нового стовпця **NSTIP**.

3) Вивести на екран прізвища та імена студентів, стипендія яких дорівнює *75*.
4) Вивести на екран номери залікових книжок студентів, які не мають трійок по певному предмету.
5) Вивести інформацію про студентів, які мають *4* і *5* по певному предмету.
6) За допомогою оператора **IN** вивести прізвища студентів, імена яких *Анатолій* або *Володимир*.
7) Вивести інформацію про студентів, прізвища яких починаються з літер із діапазону „*А*” – „*М*”.
8) Вивести номери залікових книжок студентів, які складали іспити від *1.06.2005* до *15.06.2005*.
9) Вивести номери залікових книжок студентів, які складали іспити в червні.
10) Вивести номери залікових книжок студентів, які не принесли фото.
11) Вивести номери залікових книжок студентів, які не здали іспити (отримали незадовільні оцінки). Дані не повинні повторюватись.
12) Вивести список викладачів, прізвища яких починаються на літеру „*Б*”.
13) Вивести список дисциплін, в назві яких є слово „*бази*”.
14) Підрахувати кількість записів у таблиці **STUDENTS**.
15) Підрахувати суму стипендій, яка виплачується на групу
16) Підрахувати кількість студентів, які отримують стипендію.
17) Вивести середній бал успішності по таблиці **USPISH**.
18) Вивести мінімальну і максимальну оцінки для кожного студента по результатам сесії.
19) Вивести мінімальну позитивну оцінку для

кожного студента.

20) Підрахувати кількість студентів, які складали іспит з кожного предмета (на будь-яку оцінку).
21) Підрахувати кількість студентів, які склали іспит з кожного предмета позитивно.
22) Вивести середній бал кожного студента.
23) Вивести список студентів (**SNOM**), середній бал яких <4,3.
24) Вивести кількість студентів які склали 3 іспити.
25) Підрахувати кількість студентів, які навчаються на бюджеті і платній формі.

4. Завершіть роботу.

Лабораторна робота № 4

Тема: Багатотабличні запити. Вкладені запити. Представлення.

Завдання до виконання:

1. Завантажте програму *IBExpert*.
2. Відкрийте базу даних **Univer**.
3. Використовуючи команду **SELECT** у командному вікні ***SQL Editor***, виконайте наступні запити:

1) Вивести прізвища, імена, по батькові студентів і викладачів, відсортувати їх в алфавітному порядку по прізвищу.
2) До запиту 1 додати вивід стовпця *POSADA*, який включає значення *'студент'* і *'викладач'* відповідно для кожного запису.
3) Вивести прізвища викладачів і назви предметів, які вони викладають.
4) Вивести прізвища викладачів, назви предметів, які вони викладають і кількість годин по кожному предмету, якщо кількість годин >100.
5) За допомогою команди **INSERT** доповнити таблицю **STUDENTS** записами про двох нових студентів. Вивести прізвища усіх студентів і номера предметів, з яких вони склали іспити.

(лівостороннє об’єднання).

6) Записати попередній запит з використанням правостороннього об’єднання.
7) Вивести прізвища студентів і назви предметів, іспити з яких вони повинні скласти, тобто розклад іспитів (операція декартового добутку).
8) Вивести прізвища студентів і назви предметів, іспити з яких вони склали (задати зв’язок між таблицями).
9) Використовуючи вкладені запити, вивести інформацію про предмети, які викладає викладач Бачишина.
10) Використовуючи вкладені запити, вивести прізвища студентів, які отримують максимальну стипендію (розмір максимальної стипендії наперед невідомий).
11) Використовуючи оператори **EXISTS**, **ANY**, **ALL**, **SOME**, сформувати запити, що реалізують наступні завдання:

a) Вивести дані про студентів, що мають незадовільні оцінки

b) Вивести прізвища тих студентів, що мають лише одну „2” .

c) Вивести дані про студентів, що здали всі іспити.

d) Вивести прізвища тих викладачів, що читають більше ніж один предмет.

e) Вивести прізвища всіх студентів, що здавали іспит у вказаний день.

f) Вивести назви тих дисциплін, для вивчення яких відведена однакова кількість годин.

g) Вивести прізвища тих студентів, що мають ім’я, яке співпадає із заданим.

h) Вивести прізвища всіх студентів, які здавали іспити (без повторень).

4. Завершіть роботу.

Лабораторна робота № 5

Тема: Генератори. Тригери. Конструкції мови SQL.

Завдання до виконання:

1. Завантажте програму *IBExpert.*
2. Відкрийте базу даних **Univer**.
3. В *SQL Editor* створіть генератор **GEN1** з початковим значенням, рівним 20. За допомогою команди **INSERT** додайте запис у таблицю **VYKLAD**, використовуючи функцію **GEN_ID.**
4. За допомогою засобів *IBExpert* створіть генератор **GEN2.** Задайте за допомогою тригера лічильник для поля **NOM** для додавання нових записів у таблицю **USPISH,** починаючи з 30.
5. Створіть тригер, який перетворює назви предметів таблиці **PREDMET** у великі прописні літери при модифікації чи додаванні нових записів у дану таблицю.
6. Створіть тригер, який при додаванні нового запису у таблицю **STUDENTS** нараховує стипендію 180 грн. студентам державної форми навчання.
7. Створіть виключення **Forma**, задавши текст повідомлення: „Введіть правильно форму навчання: платна або бюджет”. Створіть тригер, який при вводі неправильної форми навчання видає дане повідомлення.
8. Перевірте виконання тригерів.

Лабораторна робота № 6

Тема: Збережені процедури.

Завдання до виконання:

1. Завантажте програму *IBExpert.*
2. Відкрийте базу даних **Univer**.
3. Створіть процедуру, яка визначає мінімальну стипендію групи. Виведіть результат у вихідний параметр Min_stip.
4. Створіть процедуру, що визначає прізвища студентів, які отримують максимальну стипендію в групі.

5. Створіть процедуру, яка індексує стипендію на 0,25. (Коефіцієнт індексації може змінюватись). Задайте коефіцієнт індексації як вхідний параметр.

6. Створіть процедуру, що виводить прізвища студентів, які отримують вказану стипендію. Задайте розмір стипендії в програмі як локальну змінну.

7. Створіть процедуру, яка обчислює сумарну кількість годин, прочитану кожним викладачем.

8. Створіть процедуру, яка нараховує стипендію студента за результатами сесії. Якщо середній бал рівний 5 , то стипендія =200, якщо більше або рівний 4 і менший 5, то стипендія =100. В іншому випадку стипендія рівна 0.

9. Створіть процедуру, яка знищує з таблиці **STUDENTS** запис по прізвищу *Іванчук*.

Лабораторна робота № 7

Тема: Безпека бази даних. Користувачі, ролі, права.

Завдання до виконання:

1. Завантажте програму *IBExpert.*
2. Відкрийте базу даних **Univer**.
3. Використовуючи *User Manager*  , створити користувача *ST* з паролем *ST.*
4. Використовуючи *Grant Manager*  , надайте йому права на модифікацію полів **SEMESTR**, **GOD** таблиці **PREDMET**.
5. Закрийте базу даних **Univer**. Відкрийте її заново як користувач ST. Перевірте, чи можна коригувати поля **SEMESTR**, **GOD** таблиці **PREDMET**, інші поля даної таблиці.
6. Відкрийте базу даних **Univer** як користувач SYSDBA.
7. Створіть роль READER, що дозволяє виконувати запити до таблиці **STUDENTS**.

8. Відкрийте базу даних **Univer** заново як користувач ST з роллю READER. Переконайтесь у дії даної ролі, створивши простий запит до таблиці **STUDENTS.**

9. Відмініть права, надані користувачу ST, і роль READER.

10. Видаліть їх.

11. Самостійно створіть користувачів і ролі, надавши їм певні права. Переконайтесь у правильній роботі даних користувачів по відношенню до різних об'єктів бази даних.

12. Виконайте повторно п.п. 3-10 у командному режимі, використовуючи *SQL Editor*.

13. Команди запишіть у зошит.

14. Завершіть роботу.

КОМАНДИ SQL

СЛОВНИК

A – Z

ACID – набір вимог до транзакції: неподільність, узгодженість, ізольованість, довговічність. *ATOMICITY*, *CONSISTENCY*, *ISOLATION*, *DURABILITY*.

DCL (*Data Control Language*) – мова керування даними – складається із засобів, які визначають чи дозволити користувачеві виконувати певні чи дії ні (***GRANT***, ***REVOKE***).

DDL (*Data Definition Language*) – мова визначення даних – складається з команд, які створюють, змінюють і знищують об'єкти (таблиці, індекси, представлення і так далі) у базі даних (***CREATE***, ***ALTER***, ***DROP***).

DML (*Data Manipulation Language*) – мова маніпулювання даними – це набір команд, що забезпечують додавання, модифікацію чи видалення даних в об'єктах бази даних, а також організацію запитів даних (***INSERT***, ***UPDATE***, ***DELETE***, ***SELECT***).

DSQL (*Dynamic SQL*) – **динамічний *SQL*** – код, що генерується додатком під час його виконання. Динамічний ***SQL*** замінює статистичний в тих випадках, якщо необхідний код не може бути написаний під час створення додатку, а залежить від дій користувача під час роботи цього додатку.

ESQL (*Embedded SQL*) – **вбудований *SQL*** – це код статистичного ***SQL***, який вмонтований в текст програми, написаної на тій чи іншій алгоритмічній мові.

Firebird SQL Server – сервер баз даних, оснований на відкритому коді *Interbase* 6.0.

IBExpert – оболонка з графічним інтерфейсом користувача, призначена для розробки і адміністрування баз даних *Interbase* і *Firebird*.

ISQL (*Interactive SQL*) – **інтерактивний *SQL*** – дає можливість користувачам виконувати *SQL*-команди і

отримувати результати за допомогою відповідних утиліт або спеціальних стандартних засобів для роботи з базами даних.

PSQL (*Procedural SQL*) – **процедурний *SQL*** – це код *SQL*, що включає конструкції ***FOR, WHILE, IF***, що використовуються при написанні процедур, тригерів, скриптів.

SQL (*Structured Query Language*) – структурована мова запитів, яка надає засоби створення і обробки даних в реляційних базах даних і є основною базовою мовою в різних СУБД.

-А-

АГРЕГАТНІ ФУНКЦІЇ – це функції *COUNT*, *SUM*, *MIN*, *MAX*, *AVG*, які використовуються у запитах.

АДМІНІСТРАТОР БАЗИ ДАНИХ – людина або група людей, що контролюють проектування, роботу і використання однієї або кількох баз даних. *DATA-BASE ADMINISTRATOR*

АТРИБУТ (ПОЛЕ) – стовпець таблиці в реляційних базах даних. *ATTRIBUTE*

АТРИБУТ ДАНИХ – характеристика даних, наприклад: довжина запису. *DATA ATTRIBUTE*

-Б-

БАЗА ДАНИХ – іменована сукупність даних, яка відображає стан об’єктів і їх відношень у предметній області, організованих за певними правилами. База даних передбачає загальні принципи опису, зберігання та маніпулювання даними. *DATA BASE*

БЕЗПЕКА ДАНИХ – захист даних від несанкціонованого доступу до них. *DATA SECURITY*

БЛОКУВАННЯ ДАНИХ – тимчасове обмеження на виконання деяких операцій обробки даних для забезпечення паралельної роботи користувачів. *LOCK*

-В-

ВИБІРКА ДАНИХ – процес пошуку і зчитування даних з бази даних. *DATA RETRIEVAL*

ВИКЛЮЧЕННЯ – це повідомлення про помилку, яке задається програмістом. *EXEPTION*

ВИКОНАНА ПРОЦЕДУРА – збережена процедура, яка викликається командою *EXECUTE PROCEDURE* і повертає набір вихідних параметрів або не повертає зовсім. *EXECUTABLE PROCEDURE*

ВІДНОВЛЕННЯ БАЗИ ДАНИХ – процес, який у базі даних відновлює дані, ушкоджені внаслідок помилок персоналу, неправильної роботи обладнання або операційної системи. *DATABASE RECOVERY*

ВІДНОШЕННЯ – 1) таблиця в реляційних базах даних; 2) зв'язок між об'єктами, елементами бази даних. *RELATION*

ВКЛАДЕНИЙ ЗАПИТ – це запит, результат виконання якого використовується в іншому запиті.

-Г-

ГЕНЕРАТОР – це механізм, який створює послідовний унікальний номер, який автоматично вставляється в стовпець під час таких операцій, як ***INSERT*** або ***UPDATE***. *GENERATOR*

-Д-

ДАНІ – інформація, подана у вигляді, придатному для обробки автоматичними засобами за можливої участі людини. *DATA*

ДОМЕН – набір допустимих елементарних значень, з якого можуть вибиратися значення атрибута даних. *DOMAIN*

ДОСТУП ДО ДАНИХ – надання даних користувачеві в процесі роботи або прийняття від нього даних за допомогою послідовності операцій пошуку, читання чи записування. *DATA ACCESS*

ДАТАЛОГІЧНА МОДЕЛЬ – опис структури в термінах конкретної СУБД чи технології, які вибираються для реалізації бази даних на основі інфологічної моделі.

-З-

ЗАГОЛОВОК ВІДНОШЕННЯ – перший рядок таблиці.

ЗАПИС – рядок таблиці в базах даних. *RECORD*

ЗАПИТ – звернення до бази даних, що містить завдання на пошук, читання в базі даних відповідно до деякої умови і видачу інформації користувачеві в потрібному вигляді. *QUERY*

ЗБЕРЕЖЕНА ПРОЦЕДУРА – програма, що містить SQL-конструкції, яка зберігається у базі даних і виконується на стороні сервера. *STORED PROCEDURE*

ЗОВНІШНІЙ КЛЮЧ – це спеціальний (не унікальний) ключ таблиці, який використовується для встановлення зв'язків між таблицями з метою забезпечення цілісності бази даних. *FOREIGN KEY*

-І-

ІЄРАРХІЧНА МОДЕЛЬ ДАНИХ – модель даних типу дерево, в якій елементи одного рівня підпорядковані елементам вищого рівня. *HIERARCHICAL DATA MODEL*

ІНФОЛОГІЧНА МОДЕЛЬ – це опис майбутньої бази даних, що відображає інформацію про предметну область у вигляді, незалежному від засобів і технологій реалізації проекту за допомогою природної мови, формул, графіків, діаграм, таблиць та інших засобів, зрозумілих як розробникам БД, так і звичайним користувачам.

ІНФОРМАЦІЙНА СИСТЕМА – сукупність організаційних і технічних засобів для збереження та обробки інформації з метою забезпечення інформаційних потреб користувачів. *INFORMATION SYSTEM*

-К-

КАРДИНАЛЬНЕ ЧИСЛО – кількість кортежів таблиці-відношення. *DIRECTORY*

КОРЕЛЬОВАНИЙ (ЗАЛЕЖНИЙ) ЗАПИТ – це підзапит, обчислення якого залежить від основного запиту, тобто у під запиті використовуються значення полів зовнішнього запиту.

КОРТЕЖ (ЗАПИС) – рядок таблиці в базах даних. *TUPLE*

-М-

МЕРЕЖЕВА МОДЕЛЬ ДАНИХ – модель даних, в якій дані подані у вигляді орієнтованого графа, до будь-якого вузла якого може входити більше одного зв'язку. *NETWORK DATA MODEL*

МОВА ВИЗНАЧЕННЯ ДАНИХ – набір команд, які створюють об'єкти (таблиці, індекси, представлення і так далі) у базі даних. *DATA DEFINITION LANGUAGE (DDL)*

МОВА МАНІПУЛЮВАННЯ ДАНИМИ – це набір команд, що забезпечують додавання, модифікацію чи видалення даних в об'єктах бази даних. *DATA MANIPULATION LANGUAGE (DML)*

МОВА КЕРУВАННЯ ДАНИМИ – набір засобів, які визначають, чи дозволити користувачеві виконувати певні чи дії ні. *DATA CONTROL LANGUAGE (DCL)*

МОДЕЛЬ ДАНИХ – спосіб подання даних у вигляді певної структури та засоби маніпулювання ними. *DATA MODEL*

-О-

ОБМЕЖЕННЯ ТАБЛИЦІ – це частина визначень таблиці, яка обмежує значення, що допустимі до вводу в поля таблиці. *CONSTRAINTS*

-П-

ПАРАЛЕЛІЗМ ТРАНЗАКЦІЙ – забезпечення кожному користувачу такого режиму роботи, при якому не відчувалося втручання інших користувачів.

ПЕРВИННИЙ КЛЮЧ ТАБЛИЦІ – поле або сукупність полів таблиці, які не допускають повторення значень і однозначно визначають кожен запис таблиці. *PRIMARY KEY*

ПОЛЕ – частина запису, що містить дані конкретного типу. *FIELD*

ПРАВА – це дозвіл якому-небудь користувачу виконувати якусь операцію над певним об'єктом бази даних. *GRANT*

ПРЕДСТАВЛЕННЯ – це віртуальна таблиця, створена на основі запиту до реальних таблиць. *VIEW*

ПРОЕКТУВАННЯ БАЗИ ДАНИХ – розробка схеми даних для деякої предметної області. *DATA BASE DESING*

ПРОСТИЙ (НЕЗАЛЕЖНИЙ) ПІДЗАПИТ – це підзапит, обчислення якого здійснюється один раз, а множина значень, отримана в результаті виконання, використовується під час обробки зовнішнього запиту.

ПРОЦЕДУРА-ВИБІРКА – збережена процедура, яка викликається командою SELECT і повертає набір вихідних параметрів у вигляді таблиці. *SELECTABLE PROCEDURE*

-Р-

РЕЛЯЦІЙНА АЛГЕБРА – математичний апарат для маніпулювання відношеннями реляційної моделі даних, що включає операції об'єднання, перетину, віднімання, декартового добутку, селекції, проекції, поєднання і ділення відношень. *RELATION ALGEBRA*

РЕЛЯЦІЙНА БАЗА ДАНИХ – база даних, реалізована відповідно до реляційної моделі даних. *RELATION DATA BASE*

РЕЛЯЦІЙНА МОДЕЛЬ ДАНИХ – модель даних, в котрій дані організовані у двомірні відношення (таблиці), до яких можна застосувати операції реляційної алгебри. *RELATION DATA MODEL*

РОЛЬ – це ім'я абстрактного користувача, який наділяється певними правами доступу до певних об'єктів бази даних. *ROLE*

-C-

СИСТЕМА УПРАВЛІННЯ (КЕРУВАННЯ) БАЗАМИ ДАНИХ (СУБД) – комплекс програм і мовних засобів, призначених для створення, введення та використання баз даних. *DATABASE MANAGEMENT SYSTEM (DBMS)*

СИСТЕМНІ ТАБЛИЦІ – таблиці, де зберігаються метадані сервером *Interbase* в спеціальних таблицях. *SYSTEM TABLES*

СОРТУВАННЯ – упорядкування сукупності об'єктів до заданих відношень порядку. *SORT*, *SORTING*

СТЕПІНЬ ВІДНОШЕННЯ – кількість атрибутів таблиці-відношення.

СТРУКТУРОВАНА МОВА ЗАПИТІВ – сукупність мовних засобів створення і обробки даних в реляційних базах даних, що дозволяють задовольнити інформаційні потреби користувачів. *STRUCTURED QUERY LANGUAGE (SQL)*

СХЕМА БАЗИ ДАНИХ – опис бази даних, поданий у вигляді відношень із встановленими зв'язками без кортежів. *DATABASE SCHEMA*

-T-

ТАБЛИЦЯ (від лат. Tabula – дошка, таблиця) – спосіб формалізованого подання даних у вигляді двовимірного масиву. *TABLE*

ТЕРМ – знак, який використовується для завершення тригера або процедури, відмінний від крапки з комою (;). *TERM*

ТРАНЗАКЦІЯ – це логічний блок команд з базою даних, який може бути успішно виконаний, якщо при цьому не порушується цілісність бази даних, або відмінений – в протилежному випадку або за рішенням користувача. *TRANSACTION*

ТРИГЕР – це частина програмного коду, що асоціюється з таблицею або виглядом, яка автоматично виконує дії при додаванні, зміні або видаленні запису в таблиці або представленні. *TRIGGER*

-У-

УНІКАЛЬНИЙ КЛЮЧ – поля, значення в якому не може бути невизначеними або повторюватись, і який не є первинним ключем. *UNIQUE*

-Ф-

ФАЙЛ ДАНИХ – файл, що містить дані, а не команди (на відміну від командного чи програмного файлів). *DATA FILE*

ЛІТЕРАТУРА

1. Ковязин А., Востриков С. Мир InterBase. Архитектура, администрирование и разработка приложений баз данных в InterBase/Firebird/Yaffil. Издание 3-е, дополненное. – М.: КУДИЦ-ОБРАЗ; СПб.: Питер, 2005. – 496 с.

2. Глушаков С.В., Ломотько Д.В. Базы данных: Учебный курс.—Харьков: Фолио; М.: ООО «Издательство АСТ», 2001. – 504 с.

3. Скляр А. Я. Введение в Interbase – М.: Горячая линия – Телеком, 2002. – 517 с: ил.

4. Пасічник В. В., Резніченко В. А. Організація баз даних та знань. – К.: Видавнича група BHV, 2006. – 384 с.: іл.

5. Астахова И. Ф. SQL в примерах и задачах; Учеб. пособие / И. Ф. Астахова, А. П. Толстобров, В. М. Мельников. – Мн.: Новое знание, 2002. – 176 с.

6. http://ibase.ru – сайт СУБД InterBase, Firebird, Yaffil SQL.

7. www.ibexpert.net – офіційний сайт розробника IBExpert KG.

8. www.firebirdsql.org – офіційний сайт розробника Firebird.