Міністерство освіти і науки України
Тернопільський національний економічний університет
Факультет комп'ютерних інформаційних технологій
Кафедра спеціалізованих комп'ютерних систем

Методичні рекомендації до виконання
лабораторних робіт

з дисципліни

"СИСТЕМИ УПРАВЛІННЯ БАЗАМИ ДАНИХ"

для студентів освітнього ступеня «бакалавр»
спеціальності 151 – Автоматизація та комп'ютерно-інтегровані технології
освітньо-професійна програма - Автоматизація та комп'ютерно-інтегровані технології»

ТЕРНОПІЛЬ
ТНЕУ
2019

Методичні рекомендації до виконання лабораторних робіт з дисципліни “Системи управління базами даних” для студентів ОС «бакалавр», 151 – Автоматизація та комп’ютерно-інтегровані технології / Укл.: Пітух І.Р.– Тернопіль, 2019. – 34 с.

Методичні рекомендації до виконання лабораторних робіт складаються з частин, що рекомендовані програмою на основі галузевого стандарту вищої освіти України з спеціальності 151 – Автоматизація та комп’ютерно-інтегровані технології

Укладач: **Пітух Ігор Романович**, к.т.н., доцент

Рецензенти: **Франко Ю. П**. к.т.н., доцент кафедри комп’ютерних технологій Тернопільського національного педагогічного університету

Івасьєв С. В. к.т.н., ст. викладач кафедри кібербезпеки Тернопільського національного економічного університету

Затверджено на засіданні кафедри спеціалізованих комп’ютерних систем протокол №8 від 28.03.2019 р.

Розглянуто та схвалено науково-методичною комісією з автоматизації та комп’ютерно-інтегрованих технологій протокол №3 від 28.03.2019 р.

Розглянуто та схвалено науково-методичною радою факультету комп’ютерних інформаційних технологій, протокол № 6 від 28.03.2019 р.

ЗМІСТ

Лабораторна робота №1
Тема: Створення бази даних.

Теоретичні відомості:

В даних методичних вказівках розглядаються ключові моменти використання структурованої мови запитів **SQL** (*Structured Query Language*), яка надає засоби створення і обробки даних в реляційних базах даних і є основною базовою мовою в різних СУБД.

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

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

DML (*Data Manipulation Language*) – мова маніпулювання даними – це набір команд, що визначають, які значення представлені в об'єктах бази даних у будь-який момент часу.

DCL (*Data Control Language*) – Мова Керування Даними – складається із засобів, які визначають чи дозволити користувачеві виконувати певні чи дії ні.

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 варіанти використання;
- фігурні дужки визначають обов'язкове включення однієї з конструкцій ({ })

Інсталяція програми *Firebird 1.5*

Firebird 1.5 є клоном *Interbase*, програми, яка призначена для роботи з базами даних. Інсталяція програми *Firebird 1.5* по замовчуванню відбувається у папку **C:\Program Files\Firebird\Firebird_1_5**. Під час інсталяції необхідно 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, інсталяцію менеджера сервера (*Install Server Control Applet*) на панель управління, копіювання необхідних бібліотек. Менеджер сервера використовується для зміни названих вище опцій і для запуску сервера. В іншому випадку для запуску сервера необхідно завантажити файл **fbserver.exe** з папки **BIN**, яка розташована у папці *Firebird_1_5*.

Як клієнтська система, використовується програма *IBExpert*, ярлик якої

знаходиться на робочому столі IBExpert .

Створення бази даних

Для створення бази даних з використанням утиліти *isql* (*isql* означає *interactive SQL* – інтерактивний SQL) в інтерактивному режимі, в командному

рядку необхідно перейти в директорію C:\Program Files\ Firebird\ Firebird_1_5\ BIN і набрати ***ISQL***.

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

Сукупність дій відносно бази даних, в результаті яких дотримується цілісність бази даних, називається **транзакцією**.

Для підтвердження транзакції використовується команда **COMMIT;**. Якщо транзакція завершується успішно, то усі зміни фіксуються в базі даних.

Якщо сукупність дій, направлених на базу даних, порушують її цілісність, то відповідно їх потрібно відмінити за допомогою команди **ROLLBACK;**.

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

1. Створення бази даних

Розглянемо спрощену конструкцію 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. Типовим ім'ям є **SYSDBA**.

PASSWORD *'пароль'* – задає пароль. Типовим є пароль **masterkey**.

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 1024, 2048, 4096, 8192, 16384. По замовчуванню використовується 1024. Бажано, щоб розмір сторінки співпадав з розміром кластера жорсткого диска.

DEFAULT CHARACTER SET *код* – встановлення кодування, яке використовується по замовчуванню. Для Східної Європи необхідно задавати **WIN1251**. Якщо даний параметр не задано, то кодування встановлюється в **NONE**.

Приклад:

CREATE DATABASE 'D:\Students\univer.gdb'
PAGE_SIZE 8192
USER 'SYSDBA' **PASSWORD** 'masterkey'
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’єдання від бази даних

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

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

5. Створення таблиць

Розглянемо спрощений синтаксис команди

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 (стовпців).

тип_даних – задається зарезервований тип даних або назва домену або розрахункове поле (**COMPUTED BY** *вираз*).

обмеження:

- задання за замовчуванням певного значення або **NULL**-значення або для текстових полів імені користувача бази даних
  DEFAULT *значення* **| NULL | USER**
- умову нерівності нулю (**NOT NULL**);
- обмеження **CHECK**
  [CONSTRAINT *ім’я_обмеження***] CHECK** *умова***,**
- визначення первинного ключа
  [CONSTRAINT *ім’я_обмеження***] PRIMARY KEY,**
- визначення унікального ключа
  [CONSTRAINT *ім’я_обмеження***] UNIQUE,**
- 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);
```

При визначенні первинного або унікального ключів таблиці обов'язково потрібно задавати обмеження **NOT NULL**. Ім'я обмежень бажано задавати, оскільки вони використовуються при пошуку помилок і знищенні обмежень. Якщо ім'я обмежень не задано, то вони генеруються системою.

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 обмежень.

Типи даних :

Числові:

SMALLINT – цілі числа від -128 до +127

INTEGER – цілі числа від -32768 до +32767

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 п

DOUBLE PRECISION – дійсні числа від $1.7*10^{-308}$ до $1.7*10^{308}$ з 15 значущими цифрами

NUMERIC[(*розмірність* [, *точність*])] – фіксований формат.

розмірність – загальне число знаків (максимальне 18 знаків);

точність – число знаків після коми.

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

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 32767.

BLOB [SUB_TYPE *число***] [SEGMENT SIZE [***розмір***]]** – поле, що містить дані великого об'єму такі як графіка, текст, цифровий звук, у двійковому вигляді. Якщо *число* дорівнює **1**, то це текст. Замість числа **1** можна вказувати константу **TEXT**.

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 визначену 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. PXW_CYRL** встановлює порядок сортування для баз даних PARADOX.

Приклади обмежень для числових полів

CHECK (*ім'я_поля* **[NOT] BETWEEN 0 AND 6)**

CHECK (*ім'я_поля* > 10000 **AND** *ім'я_поля* <= 2000000**)**

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* **=10 OR (***ім'я_поля* **> 20 AND** *ім'я_поля* **<= 100) OR** *ім'я_поля* **IS [NOT] NULL)**

CHECK (*ім'я_поля* **[NOT] CONTAINING** *значення***)**

CHECK (*ім'я_поля* **[NOT] STARTING WITH 'V')**

CHECK (*ім'я_поля* **= UPPER (***ім'я_поля***))**

Розрахункове поле

COMPUTED BY (STIP*1,2)

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. Знищення таблиці

DROP TABLE *ім'я_таблиці***;**

Приклад:

Знищення таблиці VYKLAD.

DROP TABLE VYKLAD;

7. Створення домену

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

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 по замовчуванню **DEFAULT**;
- умову нерівності нулю **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 по замовчуванню 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. Знищення домену

DROP DOMAIN *назва_домену*;

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

Завдання до виконання:

1) В командному рядку перейдіть у папку **C:\Program Files\Firebird\Firebird_1_5\bin** і завантажте утиліту **isql.**
2) Створіть на диску **D:** у папці **Students** базу даних **Univer.gdb** для користувача **SYSDBA** з паролем **masterkey**. Розмір сторінки задайте 4096, кодування WIN1251.
3) Під'єднайтесь до бази даних.
4) Створити домен **NAM** для текстових даних, що містять не більше 20 символів, задавши кодування і порядок сортування **WIN1251.**
5) Створіть таблицю **VYKLAD** з полями

Таблиця **VUKLAD**

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

VNOM – табельний номер викладача, є цілим, не допускає нульових значень, вибирається 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.

6) 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. При створенні таблиці **PREDMET** задайте зв'язок з таблицею **VYKLAD** по полю **VNOM**. Задайте умови цілісності: при знищенні – **SET NULL**, при зміні – **UPDATE.**

7) Від'єднайтесь від бази даних і вийдіть з режиму *isql***.**
8) За допомогою ярлика на робочому столі завантажте клієнт-додаток *IBExpert*.
9) За допомогою команди **Database ⇒ Register Database** зареєструйте створену базу даних.
10) За допомогою команди **Database ⇒ Connect to Database** відкрийте базу даних **UNIVER**.
11) За допомогою команди **Database ⇒ New Table** створіть таблицю **STUDENTS** з полями

Таблиця **STUDENTS**

SNOM	SFAM	SIMA	SOTCH	STIP	GRUP	FORM	FOTO
200101	Іванов	Сергій	Петрович	75,00	ПМ-32	платна	

SNOM – номер залікової книжки студента є первинним ключем. Поля **SFAM** – прізвище, **SIMA** – ім'я, **SOTCH** – по-батькові студента, Для прізвища, імені, по-батькові задайте тип даних у вигляді домену **NAM. GRUP** – група, в якій навчається студент. Для поля **GRUP** задайте значенням по замовчуванню назву своєї групи Поле **STIP** – стипендія – має тип даних **NUMERIC** або **DECIMAL** з 2 знаками після коми. Для поля **FOTO** – фото студента – вибирається тип даних **BLOB**, **SUBTYPE**=BYNARY, **SIZE**=2048. Для поля **FORM** – форма навчання – задайте обмеження, що допускають ввід тільки двох значень 'платна' або 'бюджет'.

12) 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 значень від 1 до 5 включно або NULL. По замовчуванню встановлюється значення **NULL** (не визначено).

13) На вкладці **Constraints** вікна **Table** задайте первинний і зовнішній ключі. Встановіть зв'язок таблиці **USPISH** з таблицею **STUDENTS** по полю **SNOM.** Задайте умови цілісності, що задовольняють умовам каскадування.

14) В зошит запишіть структури таблиць і схему даних.

15) Завершіть роботу.

Контрольні запитання:

1) Яка програма є сервером баз даних?
2) Яка програма є клієнтом баз даних?
3) Що таке транзакція?
4) Яка мова використовується для роботи в СУБД?
5) Як створити базу даних?
6) Де зберігаються дані?
7) Які типи даних ви знаєте?
8) Що означає NULL-значення?
9) Яким чином задаються обмеження для даних?
10) Як встановлюються зв'язки між таблицями?
11) Як реалізована цілісність даних в БД?

Лабораторна робота №2
Тема: Коригування бази даних.

Теоретичні відомості:

1. 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***])];**
- 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*;

Потрібно зауважити, що команда **ALTER TABLE** може містити декілька операторів **ADD, DROP,** які розділяються комами.

Приклад:

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 зовнішього 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. Знищення поля

ALTER TABLE VYKLAD **DROP** VNAME1;

10. Створення розрахункового поля

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

2. 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, 'Іванова', '1.09.2003', 560.80, 'Ст.викладач');

3. Знищення записів з 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 співробітника, табельний номер якого рівний 105

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

4. 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';

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. додайте у таблицю **PREDMET** поле **POLE** типу **FLOAT**;
   3. змініть назву поля **POLE** на **POLE1**;
   4. змініть тип даних на **INTEGER**;
   5. розташуйте дане поле третім по порядку;
   6. видаліть поле **POLE1**;
   7. введіть дані у таблиці **VYKLAD** і **PREDMET** (не менше 6 записів);
   8. видаліть запис про викладача, прізвище якого **IВАНОВ**;
   9. модифікуйте у таблиці **PREDMET** запис про предмет Бази даних, задавши семестр 6 і кількість годин 108.
   10. створіть розрахункове поле **OPLATA**, в якому вираховується податок 13% від зарплати.
   11. встановіть зв'язок таблиці **USPISH** з таблицею **PREDMET** по полю **PNOM**. Задайте умови цілісності, що задовольняють умовам каскадування;
   12. переконайтесь, як виконуються дані умови при знищенні записів з таблиці **PREDMET** чи зміні ключового поля.
   13. в зошиті доповніть схему даних;
   14. 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.

Контрольні запитання:

1) Які команди використовуються для модифікації структури таблиці?
2) Які дії можна виконати за допомогою команди **ALTER TABLE... ALTER**?
3) Як ввести дані у таблицю?
4) Як змінити дані, які задовольняють певній умові?
5) Якою командою можна видалити дані з таблиці?
6) Як додати розрахункове поле в таблицю?

Лабораторна робота №3
Тема: Прості запити. Групові операції. Використання агрегатних функцій.

Теоретичні відомості:

Для формування запитів використовується команда SELECT.

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;

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. Вивід прізвищ відбувається у стовпець з назвою SURNAME. В запиті для таблиці STUDENTS задано псевдонім ST.

SELECT ST.SFAM **AS** SURNAME
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. По замовчуванню встановлюється сортування по зростанню.

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

WHERE *умова* – задає умови для вибору записів.

В умовах розділу WHERE можуть використовуватись оператори

1) =, <>, >, <, >=, <=

2) **AND** (логічне і), **OR** (логічне або), **NOT** (логічне 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 _ заміняє один символ;
- знак відсотків % заміняє будь-яку послідовність символів;
- [a-m] – означає, що в даній позиції має бути один символ з вказаного діапазону;
- [!safd] – означає, що в даній позиції має бути один символ, який не входить в даний діапазон.

Приклад:

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

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

6) 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;
```

Функції

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.

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;
5) **MIN(***ім'я_поля***)** – знаходить мінімальне значення серед значень вибраного поля:
6) **UPPER(***ім'я_поля***)** – перетворює символьну стрічку до верхнього регістра;
7) **CAST(***ім'я_поля* **AS** *тип_даних***)** – перетворює дані поля до певного типу даних. Використовується в умовах пошуку для порівняння даних.
8) **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;
```

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 середній бал кожного студента.
19. Вивести мінімальну і максимальну оцінки для кожного студента по результатам сесії.
20. Вивести мінімальну позитивну оцінку для кожного студента.
21. Підрахувати кількість студентів, які складали іспит з кожного предмета (на будь-яку оцінку).
22. Підрахувати кількість студентів, які склали іспит з кожного предмета позитивно.
23. Підрахувати кількість студентів, які навчаються на бюджеті і платній формі.

4) Завершіть роботу.

Контрольні запитання:

1) Яка команда використовується для формування запитів?
2) Як задати сортування записів 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**?

Лабораторна робота №4-5
Тема: Багатотабличні запити. Вкладені запити. Представлення.

Теоретичні відомості:
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;
```

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 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;

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;
```

Використання трьох таблиць в запиті

Приклад:

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;

Використання вкладених запитів

Вкладені запити створюються шляхом розміщення одного запиту (підзапиту) в середині другого. Підзапит повертає завжди одне значення.

Приклад:

Вивести номера дисциплін і оцінки, вищі від середньої.

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

У вкладених запитах можуть використовуватись оператори **EXIST**, **ANY**, **ALL**, **SOME**, які в якості аргументу використовують підзапит.

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

Приклад:

Вивести дані з таблиці USPISH, якщо в ній є відмінні оцінки.

SELECT * **FROM** USPISH **WHERE** USPISH.OCINKA=5 **AND EXIST**
(**SELECT** * **FROM** USPISH **WHERE** USPISH.OCINKA=5);

Існує ще три оператор, що орієнтовані на підзапити – це ANY, ALL та SOME. Але вони відрізняються від EXIST тим, що використовуються разом з реляційними операторами в підзапитах. Оператори ANY та SOME взаємозамінні, тому в поданих прикладах будуть працювати однаково. Оператор ALL вибирає всі значення, що виведенні підзапитом.

Приклад:

Вивести інформацію про студентів, що отримали оцінки з різних навчальних предметів.

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

Даний запит можна виконати за допомогою оператора IN.

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

Приклад:

Вивести дані про тих студентів, чиї оцінки вище або рівні отриманим 13.06.2005

SELECT * **FROM** USPISH **WHERE** OCINKA>= **ALL** (**SELECT** OCINKA **FROM** USPISH **WHERE** DATA='13.06.2005')

Представлення

Представлення – це віртуальна таблиця, створена на основі запиту до реальних таблиць. Представлення виконується кожен раз, коли відбувається до нього звертання.

1. 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. Якщо дане ім'я відсутнє, то вибирається по замовчуванню ім'я стовпця з основної таблиці. Кількість імен стовпців, визначені в представлені, має відповідати кількості і порядку стовпців, перерахованих в **<select>.**

<select> - команда select, яка задає критерії вибору записів.

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

За допомогою модифікації представлень можна змінювати вміст основних таблиць. Але це можливо в тому випадку, якщо усі поля таблиці допускають наявність NULL-значень.

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;
```

2. Знищення представлення

DROP VIEW *ім'я_представлення***;**

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

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, відсотувати 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 здали 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 EXIST, ANY, ALL, SOME, сформувати запити, що реалізують наступні завдання:
    1) Вивести дані про студентів, що мають незадовільні оцінки
    2) Вивести прізвища тих студентів, що мають лише одну „2” .
    3) Вивести дані про студентів, що здали всі іспити.
    4) Вивести прізвища тих викладачів, що читають більше ніж один предмет.
    5) Вивести прізвища всіх студентів, що здавали іспит у вказаний день.
    6) Вивести назви тих дисциплін, для вивчення яких відведена однакова кількість годин.
    7) Вивести прізвища тих студентів, що мають ім’я, яке співпадає із заданим.
    8) Вивести прізвища всіх студентів, які здавали іспити (без повторень).

Контрольні запитання:

1) Яке призначення оператора **UNION**?
2) Яким чином встановлюється зв’язок між двома таблицями в запитах?
3) Які є види зв’язків між таблицями?
4) Що таке вкладені запити?
5) Які оператори використовуються в вкладених запитах?
6) Що таке представлення?
7) Чим представлення відрізняються від запитів?
8) Чи може бути створене представлення на основі іншого представлення?
9) Чи зберігаються дані, отримані в представленнях, у базі даних?

Лабораторна робота №6
Тема: Генератори. Тригери. Конструкції мови SQL.

Теоретичні відомості:

Генератор – 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, які можуть бути вставлені в стовпець, який використовується як первинний ключ.

1. Створення генератора

CREATE GENERATOR *ім'я_генератора*;

При створенні генератора за замовчуваннчм його початкове значення дорівнює нулю.

2. Ініціалізація генератора

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

3. 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, починаючи з 101.

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

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

4. Створення тригера

```
CREATE TRIGGER ім'я_тригера FOR ім'я_таблиці
[ACTIVE | INACTIVE]
{BEFORE | AFTER}
{DELETE | INSERT | UPDATE}
[POSITION номер]
AS тіло_тригера
термінатор
```

[ACTIVE | INACTIVE] – необов'язковий параметр, який визначає дію тригера після завершення транзакції. **ACTIVE –** тригер використовується (по замовчуванню). **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, по замовчуванню дорівнює нулю. Тригер, який має менший номер, виконується раніше.

тіло_тригера – складається з двох частин:

1) блок опису локальних змінних, які використовуються в тригері. Кожна змінна описується конструкцією

DECLARE VARIABLE *ім'я_змінної тип_змінної* ;

і закінчується крапкою з комою (;);

2) блок програмного коду, який починається оператором **BEGIN** і завершується оператором **END**

BEGIN
команди_InterBase
END

Кожна команда завершується крапкою з комою (;).

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

Оскільки, символ крапка з комою, який використовується зазвичай для завершення SQL-команди, використовується також і всередині тригера, то для завершення команди **CREATE TRIGGER** слугує певний символ – термінатор**.** Це може бути будь-який символ, який не використовується в даній команді, наприклад **^** або **!! .**

Термінатор визначається перед створенням тригера командою

SET TERM *термінатор* ;

Після команди **CREATE TRIGGER** дію крапки з комою потрібно відновити командою

SET TERM ; *термінатор*

Приклад:

Створити генератор **GEN_STUDENTS** і створити тригер **STUDENTS_BI**, який використовує даний генератор як лічильник для поля **SNOM** при додаванні записів у таблицю **STUDENTS**.

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

5. Модифікація тригера

ALTER TRIGGER *ім'я_тригера*
[ACTIVE | INACTIVE]
[{BEFORE | AFTER} {DELETE | INSERT | UPDATE}]
[POSITION *номер***]**

[AS *тіло_тригера* **]**
[*термінатор***]**

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

Приклад:

Змінити статус активного тригера *TR1* на пасивний.

ALTER TRIGGER TR1 **INACTIVE;**

6. Видалення тригера

DROP TRIGGER *ім'я_тригера***;**

Конструкції мови SQL

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**.

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, отримані дані записуються у змінні, вказані після ключового слова **INTO.**

6. Оператор циклу FOR … DO …

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

SELECT_команда отримує дані з бази даних. Дана команда отримує за один

раз тільки один рядок, і результат записується у відповідні змінні, вказані після ключового слова **INTO.** Перед кожною змінною після **INTO** ставиться двокрапка (**:**) для того, щоб відрізнити ім'я стовпця від імені змінної;

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

Завдання до виконання:

1) Завантажте програму *IBExpert.*
2) Відкрийте базу даних **Univer**.
3) Створіть генератор **GEN1** з початковим значенням, рівним 20.
4) Створіть за допомогою тригера лічильник для поля **NOM** для додавання нових записів у таблицю **USPISH**.
5) Створіть тригер, який перетворює назви предметів таблиці **PREDMET** у прописні літери при модифікації чи додаванні нових записів у дану таблицю.
6) Створити тригер, який при знищенні запису з таблиці **STUDENTS** буде знищувати пов'язані з ним записи з таблиці **USPISH**.
7) Створити тригер, який при додаванні нових записів до таблиці **USPISH**, перевіряє чи є студент з відповідним номером в таблиці **STUDENTS**.
8) Створити тригер, який при видалені записів з таблиці **VUKLAD** заміняє значення зовнішнього ключа відповідних записів таблиці **PREDMET** на NULL.

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?

Лабораторна робота №7
Тема: Збережені процедури

Теоретичні відомості:

Збережені процедури – частини програмного коду, які зберігаються у базі даних і виконуються на стороні сервера. Збережену процедуру можна викликати з клієнтських додатків, тригерів та інших збережених процедур.

Основою могутніх можливостей, які закладені в збережені процедури є не лише модифіковані команди звичайного SQL (SELECT, UPDATE, INSERT) та засоби організації розгалужень та циклів (IF, WHILE, FOR), але й засоби обробки помилок у виключних ситуаціях. Мова збережених процедур дозволяє реалізувати складні процедури роботи з даними та завдяки орієнтованості на роботу з реляційними даними збережені процедури виходять більш компактними ніж процедури на традиційних мовах програмування.

Синтаксис збережених процедур описується наступним чином:

CREATE PROCEDURE *ім'я_процедури*
[(*вхідний_параметр тип* [, *вхідний_параметр тип* ...])]
[**RETURNS** (*вихідний_параметр тип* [, *вихідний_параметр тип*...])]
AS
тіло_процедури;
[*термінатор*]

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.

Тіло процедури може включати DML-SQL-конструкції, а також, програмні конструкції FOR SELECT…DO, IF-THEN, WHILE...DO та інші.

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

SELECT-процедури обов'язково повертають одне або декілька значень і можуть використовуватись при створенні запитів разом з таблицями і представленнями.

EXECUTE-процедури можуть повертати або не повертати 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* [(*фактичний_параметр* [, *фактичний_параметр*…])];

EXECUTE -процедури викликаються за допомогою конструкції:

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

Конструкції мови SQL

1. Оператор виходу EXIT

EXIT*;*

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

2. Оператор SUSPEND

SUSPEND*;*

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

Приклад:

Приклад збереженої процедури, який ілюструє використання даних операторів.

```
SET TERM !!;
CREATE PROCEDURE P RETURNS (R INTEGER)
AS
BEGIN
R = 0;
WHILE (R < 5) DO
BEGIN
R = R + 1;
SUSPEND;
IF (R = 3) THEN
EXIT;
END
END;
Set Term ;!!
```

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

Приклад:

Визначити максимальну стипендію.

```
SET TERM ^ ;

CREATE PROCEDURE  PR1 RETURNS (Max_stip DECIMAL (8,2))
AS
  BEGIN
SELECT MAX(STIP) FROM STUDENTS
INTO :Max_stip
END
^
SET TERM  ; ^
```

Виклик

```
SELECT * FROM PR1;
```

Приклад:

Вивести прізвища студентів, які отримують вказану стипендію.

```
SET TERM ^ ;

CREATE PROCEDURE  PR2  (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
^
SET TERM ; ^
```

Виклик

```
SELECT * FROM PR1;
```

Завдання до виконання:

1) Завантажте програму *IBExpert.*
2) Відкрийте базу даних **Univer**.
3) Створіть процедуру, яка визначає мінімальну стипендію групи.
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).
6) Створіть процедуру, яка обчислює сумарну кількість годин, прочитаних кожним викладачем.
7) Створіть процедуру, яка нараховує стипендію за результатами сесії.

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?

Лабораторна робота №8
Тема: Безпека бази даних. Користувачі, ролі, права.

Теоретичні відомості:

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.

По замовчуванню системний адміністратор самостійно може створювати бази даних, володіє всіма правами над будь-яким об’єктом бази даних, реєструє користувачів і надає їм права доступу. При реєстрації користувача вказується його ім’я і пароль.

У програмі *IB Expert* для реєстрації нових користувачів використовується *User Manager*  , для надання прав – *Grant Manager*  .

1. Надання прав

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

де *привілеї* – 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 вказані права іншим користувачам.

Права можна встановлювати відносно не тільки таблиці, а й представлення і збережених процедур.

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];

2. 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;

Якщо декілька користувачів виконують одну і ту ж роботу і мають однакові права, то їх об'єднують в групи. Для цього використовується механізм ролей. Ролі дозволяють задавати певні права доступу до об'єктів бази абстрактно, незалежно від імені користувача, а потім конкретному користувачу при реєстрації присвоювати вибрану роль.

Спрощено порядок дії ролі можна записати таким чином:

1. **Створення ролі**

CREATE ROLE *ім'я_ролі*;

2. **Надання прав ролі**

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

3. **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.

Знищення ролі

DROP *ім'я_ролі*;

Завдання до виконання:

1) Завантажте програму *IBExpert.*
2) Відкрийте базу даних **Univer**.
3) Використовуючи *User Manager*, створити користувача *ST* з паролем *ST.*
4) Надати 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) Самостійно створіть користувачів і ролі, надавши їм певні права. Переконайтесь у правильній роботі даних користувачів по відношенню до різних об’єктів бази даних. Результати запишіть у зошит.
10) Завершіть роботу.

Контрольні запитання:

1) Яку інформацію містить зареєстрований запис про користувача?
2) Який користувач має повний доступ до бази даних?
3) Як реєструються користувачі 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?

Рекомендована 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. Д. Крекне. Теория и практика построения баз данных. 8-е издание. Питер, 2003. – 798 с.
4. А.Я. Архангельский. Программирование в Delphi 7. Методика разработки программ для Windows. Базы данных и распределенные приложения. Новые технологии и инструментарии. Москва. «Издательство Бином», 2005 – 1153
5. Методичні вказівки до виконання курсової роботи з дисципліни „Бази даних та інформаційні системи” для студентів денної форми навчання спеціальності 6.080200 „Прикладна математика” /Л.Д.Бачишина, Н.О.Харів - Рівне: НУВГП, 2005.-24 с
6. Документація InterBase 6.
7. Л.М. Дибкова «Економічна інформатика». Посібник, 2002р.
8. В.Д. Руденко,... . „Курс інформатики. 1-2 частини. Київ 2002р.
9. В.Д. Руденко, ... „Практичний курс інформатики” Київ 2001р.
10. Авторський колектив. Браткевич В.В. ... „Інформатика. Комп’ютерна техніка. Комп’ютерні технології”. Київ 2001р.
11. І.Л. Тхір, ... „Посібник користувача ПК”. „Астон”- Тернопіль 2002р.
12. М.В. Макарова «Економічна інформатика». Суми – 2003р.
13. В.М. Беспалов. „Інформатика для економістів”. Київ 2003р.
14. Microsoft Word 2000. Справочник. 2-е издание. СПб.: Питер, 2000 - 383 с.
15. А. Колесников. Excel 2000 (русифицированная версия). - К.:Изд. группа BHV, 1999. - 496 с.
16. Александр Левин. Самоучитель работы на компьютере 6-е издание, исправленное и дополненное. - М.: Изд. «Нолидж", 2000.-656 с.
17. Гетц К., Джилберт М. Программирование в Microsoft Office. - К.: Ирина; Изд-во. Группа BHV, 2000. - 384 с.
18. Гетц К., Литвин П., Гильберт М. Access 2000. Руководство разработчика. - К.: Ирина; BHV. 2000. - 1264 с.
19. Фигурнов В.З. IBM PC для пользователя. Изд. 6-е, перераб.и дополн.-а. Москва, ИНФРА, 1995.
20. Бэрри Мане. Компьютерные сети: Пер. с англ. - М.: БИНОМ, 1995.- 400 с.
21. В.Ф. Ляхович. Основы информатики. - Ростов на /Д: Из-во «Феникс".- 2000.-608 с.
22. Вейскас Д. Эффективная работа с Microsoft Access 7.0 для Windows 95 : Перев. с англ. - СПб.: Питер, 1997. - 848 с.

Навчально-метдичне видання

Пітух Ігор Романович

Методичні рекомендації до виконання лабораторних робіт

з дисципліни

“СИСТЕМИ УПРАВЛІННЯ БАЗАМИ ДАНИХ”

для студентів освітнього ступеня «бакалавр» спеціальності 151 – Автоматизація та комп’ютерно-інтегровані технології освітньо-професійна програма - Автоматизація та комп’ютерно-інтегровані технології»

Підписано до друку 03.04.2019 р.
Формат 60х84/16. Папір офсетний.
Друк офсетний. Зам. № 3-579
Умов.-друк. арк. 1,9. Обл.-вид. арк. 2,3.
Тираж 30 прим.

Віддруковано ФО-П Шпак В. Б.
Свідоцтво про державну реєстрацію В02 № 924434 від 11.12.2006 р.
м. Тернопіль, бульвар Просвіти, 6/4. тел. 8 097 299 38 99.
E-mail: tooums@ukr.net