**Лабораторна робота№6 Використання команд SQL для генерації тестових даних**

**Мета:** Набуття практичних навичок використання операторів DDL та DМL

***Методичні рекомендації***

*При вивченні теми слід звернути увагу на створення та збереження структури бази даних(БД), редагування структури, визначення типів полів, зв’язування таблиць у БД.*

1. Повторити лекції №7, 8, теоретичну частину ЛР № 4,5. Ознайомитися з теоретичною частиною.
2. Для своєї БД виконати описаний в методичних вказівках сценарій.
3. Разробити аналогічний сценарій раніш створених таблиць Викладач, Студент, Група, Предмети, Розклад та Аудиторії.
4. Виконати цей сценарій Надати всі SQL-запити цього сценарія. Написати використані SQL-запити. По кожній таблиці виконати запит: SELECT \* FROM <ім.’я таблиці>

Зробити скрінщот отриманих результатів, який разом з SQL-запитами включити до звіту по ЛР.

1. Сформувати звіт з наданням виконаних команд та відповідними поясненнями.
2. Оформити результати відповідно до стандарту подання лабораторних/ практичних робіт.
3. Результати надсилати на електронну адресу викладача [t.i.lumpova@gmail.com](mailto:t.i.lumpova@gmail.com)

Файл повинен мати назву в такому форматі:

**DB<Номер групи><Номер лекції / практичної / лабораторної [літера позначення типу роботи L – лекція, P – практична, R – лабораторна]<Прізвище англійською>**. Наприклад, **DB3101R**buts.doc.

Не копіюйте фрагментів з різних інформаційних джерел, подумайте і викладіть свою точку зору. При наявності робіт -"близнюків" відповідь буде зараховуватися першому за часом надсилання.

**Строк виконання цієї роботи ІПЗ-31 - 05.11.2023**

**ІПЗ-32 - 06.11.2023**

***Контрольні запитання***

1. Які команди мови SQL використовуються для створення таблиць?
2. Якою командою мови SQL можна додати нові дані до таблиці?
3. В чому різниця форматів команди INSERT: INSERT VALUES та INSERT SELECT?
4. З якою метою в БД створюються тимчасові таблиці?
5. Як створити тимчасову таблицю в БД?
6. Якою командою можна змінювати поля в таблиці в БД?
7. Як переглянути дані таблиці?

**МЕТОДИЧНІ ВКАЗІВКИ**

Сценарій створення тестового набору.

1. Створити таблицю groups – список груп.

CREATE TABLE groups

(

gr\_id SMALLINT NOT NULL,

gr\_name VARCHAR(10) NOT NULL,

gr\_comm VARCHAR(15) NOT NULL,

PRIMARY KEY (gr\_id)

);

2. Вести дані до таблиці

INSERT INTO groups VALUES (10, "ІПЗ-11", "INSERT");

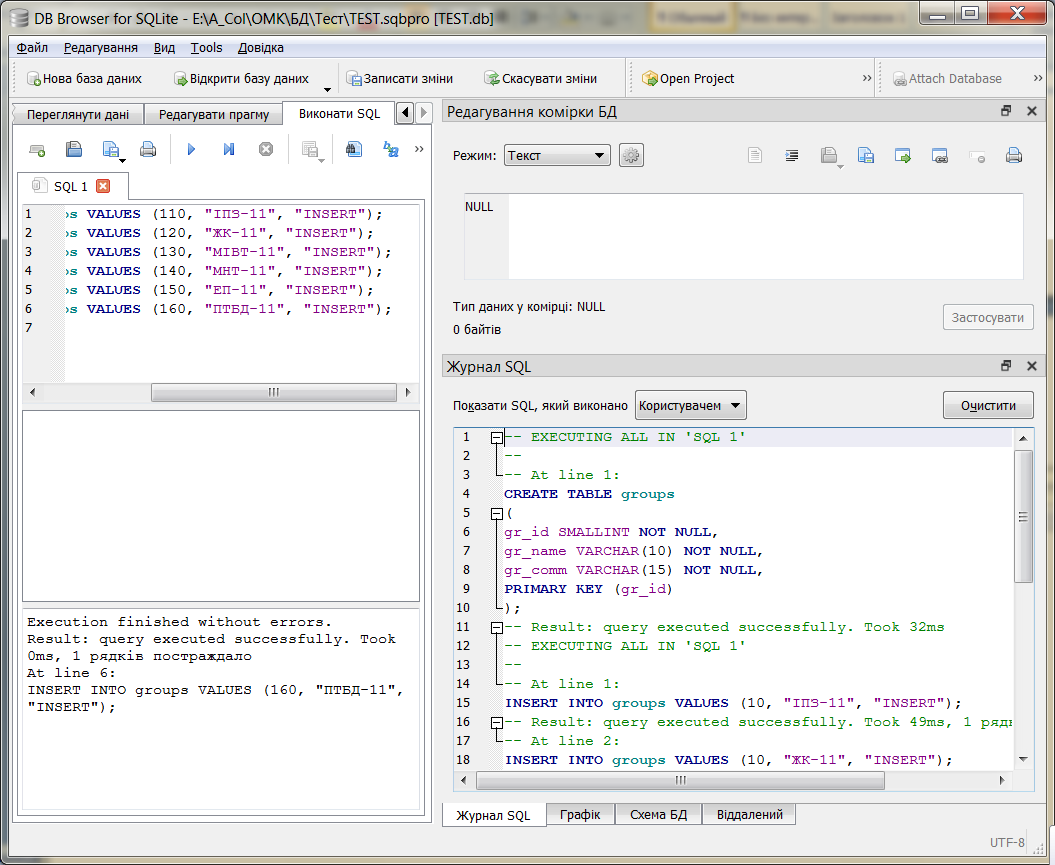
INSERT INTO groups VALUES (10, "ЖК-11", "INSERT");

INSERT INTO groups VALUES (10, "МІВТ-11", "INSERT");

INSERT INTO groups VALUES (10, "МНТ-11", "INSERT");

INSERT INTO groups VALUES (10, "ЕП-11", "INSERT");

INSERT INTO groups VALUES (10, "ПТБД-11", "INSERT");



3. Створити тимчасову таблицю groups\_temp для копіювання таблиці groups та скопіювати туди всю інформацію таблиці groups.

CREATE TEMPORARY TABLE groups\_temp

(

gr\_id SMALLINT NOT NULL,

gr\_name VARCHAR(10) NOT NULL,

gr\_comm VARCHAR(15) NOT NULL,

PRIMARY KEY (gr\_id)

);

INSERT INTO groups\_temp SELECT \* FROM groups where 1=1;

4. Змінити в тимчасовій таблиці groups\_temp первинні ключи та інші поля

UPDATE groups\_temp

SET gr\_id = gr\_id +1,

gr\_name = SUBSTRING(gr\_name,LENGTH(gr\_name),1)+"2",

gr\_comm = "TEST1";

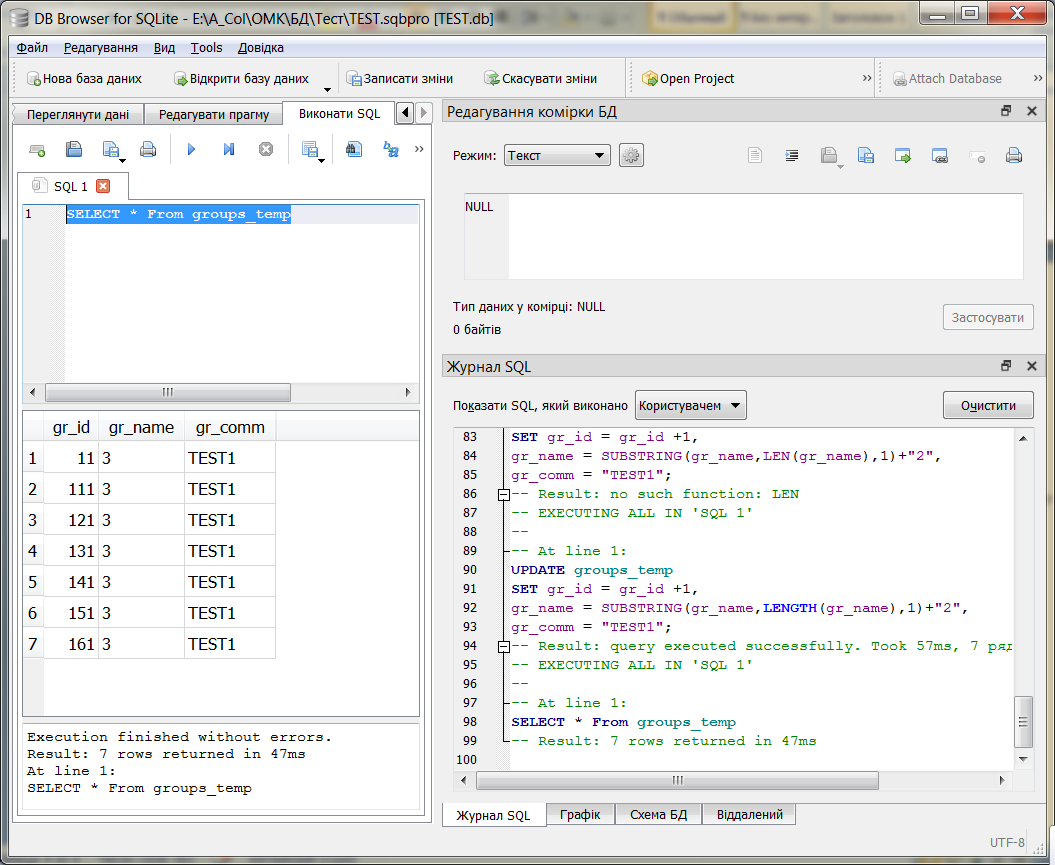
Тут використані функції:

Функція **SUBSTRING**(вираз, початкова позиція, довжина) дозволяє вибрати з выразу його частину заданої довжини, починаючи віт заданої початкової позиції

Функція **LENGTH** (текстовий рядок) повертає число символів в рядку, який задається текстовим рядком.

5. Переглянути дані тимчасової таблиці

SELECT \* From groups\_temp

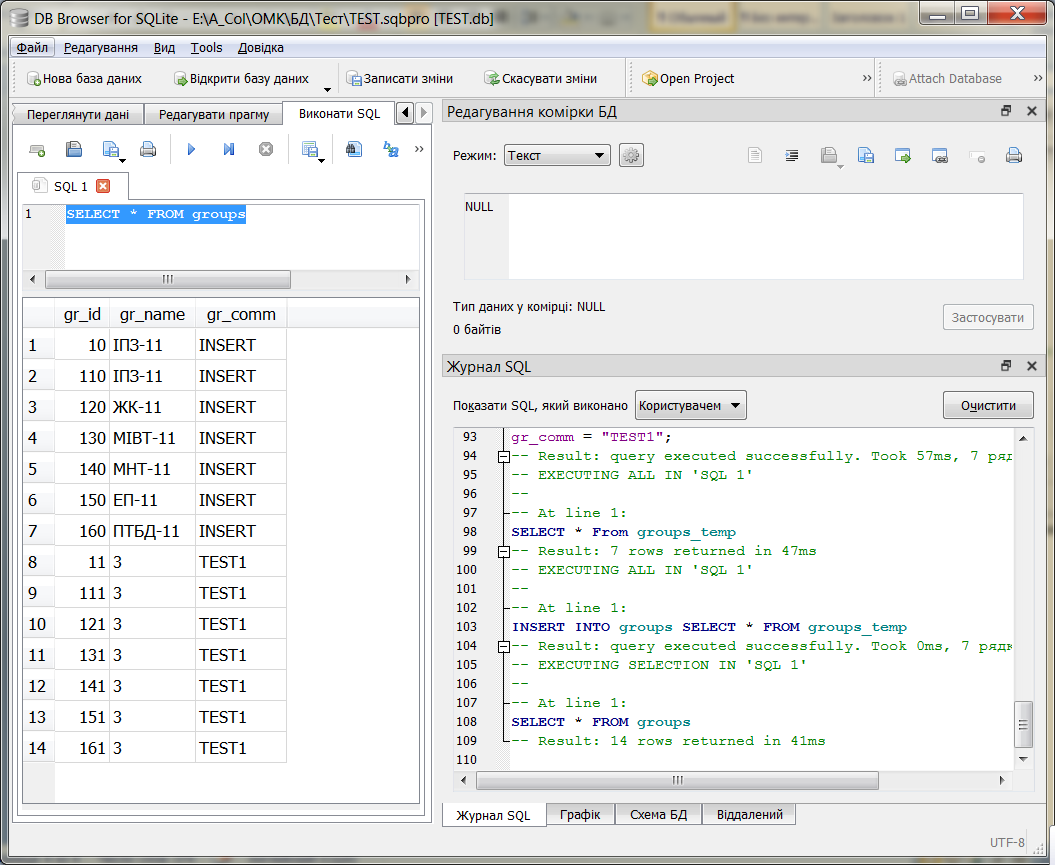


6. Вставити змінені дані з таблиці groups\_temp в таблицю groups

INSERT INTO groups SELECT \* FROM groups\_temp

7. Переглянути отриманий результат

SELECT \* FROM groups



**ТЕОРЕТИЧНА ЧАСТИНА**

**Классификация функций**

За механизмом роботы функцій вони поділяються на **детерміновані та не детерміновані.** *Детермінована функція* повертає один та той же результат при виклику з однаковим значеннями входних параметров.*Недетермінована функція* може повертати різні результати при кожному виклику, навіть, якщо на вхід подаються одні та тіж самі значення.

Правила роботи з функціями, які діють на конкретній платформі, повинні бути описані в документації до неї.

**Агрегатні та скалярні функції**

*Агрегатні функції* застосовуються к набору рядків та повертають одне підсумкове значення. *Скалярні функції* також повертають одне значення, яке залежить від переданих скалярних аргументів. Деякі скалярні функції, наприклад *CURRENT\_TIME*, викликається без аргументів.

**Агрегатні функції в ANSI SQL**

Агрегатна функція повертає одно значення, залежить від множени вхідних даних. Якщо вона входить до списку виразів в команді *SELECT*, то ця команда повинна містити фразу *GROUP BY*, можливо, у поєднанні з *HAVING*.

Наявність *GROUP BY* або *HAVING* необов’язкова, якщо агрегатна функція єдиний вираз, яке вибирається командой *SELECT*.

*Таблица 1. Агрегатні функції в ANSI SQL*

|  |  |
| --- | --- |
| **Функція** | **Призначення** |
| AVG(*expression*) | Обчислює середнє для стовпчика, вказаного в *expression* |
| COUNT(*expression*) | Обчислює вираз *expression* в групі і повертає значення відмінних від NULL |
| COUNT(\*) | Обчислює загальну кількість рядків у вказаній таблиці та представленні |
| MIN(*expression*) | Знаходить мінімальне значення в стовпчику, визначеного виразом *expression* |
| MAX | Знаходить максимальне значення в стовпчику, визначеного виразом *expression* |
| *SUM(стовпець таблиці БД)* | SUM повертає суму значень стовпця таблиці БД. Вона може застосовуватися тільки до стовпців, значення яких числа. |

Функції *ALL*, *ANY* и *SOME* також вважаються агрегатними функціями. Кількість значень, які оброблюються агрегатною функціей, залежить від того, скільки рядків повернув запит. Ці агрегатні функції різняться від скалярних, які використовуются тільки до одного рядка за один виклик.

Загальний синтаксис виклику агрегатної функції:

*aggregate\_function\_name*( [ALL | DISTINCT] *expression* )

Тут *aggregate\_function\_name* може бути *AVG, COUNT, MAX, MIN* або *SUM.*

Ключеве слово *ALL*, мається на увазі по замовчуванню, означає, що в процесі агрегування враховуються всі рядкі. Якщо задано ключеве слово *DISTINCT*, то враховується лише значення, що різняться.

При обчисленні будь-якої агрегатної функції, окрім *COUNT(\*)*, значения NULL игнорируются.

Приклади.

Для кожного запиту для кожного типу обчислюється середній обсяг продажів з початку року.

SELECT type, AVG( ytd\_sales ) AS "average\_ytd\_sales" FROM titles GROUP BY type;

Для кожного запиту для кожного типу обчислюється загальний обсяг продажів з початку року.

SELECT type, SUM( ytd\_sales ) FROM titles GROUP BY type;

**Скалярні функції в стандарті ANSI SQL**

Скалярна функція повертає одиночне значення при кожному виклику.

*Таблица 2. Категорії скалярних функцій*

|  |  |
| --- | --- |
| **Категория** | **Назначение** |
| *CURRENT\_DATE* | Повертає поточну дату |
| *CURRENT\_TIMESTAMP* | Повертає поточну дату та час |

**Приклади**

Різні СУБД повертає результат в різних форматах:

/\* MySQL \*/

**SELECT CURRENT\_TIMESTAMP;**

'2001\_12\_15 23:50:26'

/\* Microsoft SQL Server \*/

**SELECT CURRENT\_TIMESTAMP**

'Dec 15,2001 23:50:26'

**Функции CASE и CAST**

В стандарті ANSI SQL описана функція *CASE*, яка дозволяє наделити SQL\_команды *SELECT* и *UPDATE* логикою *IF/THEN*.

**CASE**

Функція *CASE* позволяє використовує аналог конструкції *IF/THEN/ELSE*

в командах *SELECT* и *UPDATE*. Вона обчислює вказані усмови та повертає одно зі списку можливих значень.

Функція *CASE* має дві форми: *просту* та *пошукову*. В простом выразі *CASE* одно вхідне значення *input\_value* порівнюється з декількома іншими значеннями та повертається результат, ассоційований з першими співпаданням. Пошуковий вираз *CASE* дозволяє проаналізуває декілько логічних умов та повертає результат, ассоційоований з першою логічною умововою, обчислення якого дало істину.

Проста операція порівняння

CASE *input\_value*

WHEN *when\_condition* THEN *resulting\_value*

[*... n*]

[ELSE *else\_result\_value*]

END

Булева пошукова операція

CASE

WHEN *Boolean\_condition* THEN *resulting\_value*

[*... n*]

[ELSE *else\_result\_expression*]

END

В простій формі функції *CASE* значення *input\_value* порівнюється з виразом в кожній гілці *WHEN*. Повертається значення *resulting\_value*, вказане в першій в гілці, що зустрілась, для якої *input\_value = when\_condition*. Якщо жодний вираз *when\_condition* не співпалозс *input\_value*, то повертається значення *else\_result\_value*. Якщо гілка *ELSE пр*опущена, то повертається NULL.

Структура більш складною пошуковою булевскою операцією аналогічна, але в кожній гілці *WHEN з*находиться окрема операція порівняння. Допускається декілька гілок *WHEN*, але не більше однієї гілки *ELSE*.

**Приклади**

Проста операція порівняння, де функція *CASE* застосовується для представлення стовпчика **contract** в зрозумілому вигляді:

SELECT au\_fname,

au\_lname,

CASE contract

WHEN 1 THEN 'Yes'

ELSE 'No'

END 'contract'

FROM authors

WHERE state = 'CA'

Функция *CASE* застосовується в команді *SELECT* для обчислення кількості різних видань в різні діапазони обсягу продаж з початку року.

SELECT CASE

WHEN ytd\_sales IS NULL THEN 'Неизвестно'

WHEN ytd\_sales <= 200 THEN 'Не более 200'

WHEN ytd\_sales <= 1000 THEN 'От 201 до 1000'

WHEN ytd\_sales <= 5000 THEN 'От 1001 до 5000'

WHEN ytd\_sales <= 10000 THEN 'От 5001 до 10000'

ELSE 'Свыше 10000'

END 'Продажи с начала года',

COUNT(\*) 'Количество изданий'

FROM titles

GROUP BY CASE

WHEN ytd\_sales IS NULL THEN 'Неизвестно'

WHEN ytd\_sales <= 200 THEN 'Не более 200'

WHEN ytd\_sales <= 1000 THEN 'От 201 до 1000'

WHEN ytd\_sales <= 5000 THEN 'От 1001 до 5000'

WHEN ytd\_sales <= 10000 THEN 'От 5001 до 10000'

ELSE 'Свыше 10000'

END

ORDER BY MIN( ytd\_sales )

Цей запит повертає такі результати:

Продажи с начала года Количество изданий

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Неизвестно 2

Не более 200 1

От 201 до 1000 2

От 1001 до 5000 9

От 5001 до 10000 1

Свыше 10000 3

Команда *UPDATE*, яка застосовує снижки до всіх книг. К книгам на тему персональных компьютеров применяется скидка

UPDATE titles

SET price = price \*

CASE

WHEN ytd\_sales > 10000 THEN 0.95 \_\_ скидка 5%

WHEN type = 'popular\_comp' THEN 0.75 \_\_ скидка 25%

ELSE 0.9 \_\_ скидка 10%

END

WHERE pub\_date IS NOT NULL

**CAST**

Функція *CAST* застосовується для явного перетворення типу даних виразу

CAST(*expression* AS *data\_type* [(length)])

Функція *CAST* перетворює довільний вираз(*expression*) до іншого типу даних. Для типов, що характеризується довжиною (наприклад, *CHAR* или *VARCHAR*), додатково задається довжина *length*. Деякі перетворення, наприклад з *DECIMAL* в *INTEGER*, виконується із заокругленням. Операція перетворення може завершиться з помилкою, якщо кінцевий тип недостатньо широкий для збереження перетвореного значення.

**Приклад**

Числове значення обсягу продажу з початку року перетворюється до типу *CHAR* та конкатенується з рядковим літералом та частиною назви книги. Стовпчик **ytd\_sales** перетворюється в *CHAR(5)*, а рядок в стовпчику **title** усекається до 30 знаків:

SELECT CAST(ytd\_sales AS CHAR(5)) + ' экземпляров ' +

CAST(title AS VARCHAR(30))

FROM titles

WHERE ytd\_sales IS NOT NULL

AND ytd\_sales > 10000

ORDER BY ytd\_sales DESC

Этот запрос возвращает такие результаты:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

22246 экземпляров The Gourmet Microwave

18722 экземпляров You Can Combat Computer Stress

15096 экземпляров Fifty Years in Buckingham Pala

**Числові скалярні функції**

*Таблица 3. Числові функції в ANSI SQL*

|  |  |
| --- | --- |
| **Функция** | **Призначення** |
| *ABS* | Повертає абсолютну величину числа |
| *CEIL* або *CEILING* | Заокруглює дробне число до найближчого більшого цілого. Цілі числа повертаються без змін |
| *EXP* | підносить константу *e* до вказаного ступеню |
| *FLOOR* | Заокруглює дробне число до найближчого меньшого цілого. Цілі числа повертаються без змін |
| *LN* | Повертає натуральний логарифм |
| *MOD* | Повертає залишок від ділення одного числа на інший |
| *POSITION* | Повертає початкову позицію підрядка в рядку |
| *POWER* | Повертає ступень числа |
| *SQRT* | Повертає квадратний корень |

Приклади використання в різних СУБД функції, яка повертає довжину рядку.

/\* MySQL и PostgreSQL \*/

**SELECT CHAR\_LENGTH('hello');**

**SELECT OCTET\_LENGTH(book\_title) FROM titles;**

/\* Microsoft SQL Server \*/

**SELECT DATALENGTH(title) FROM titles**

**WHERE type = 'popular\_comp'**

/\* Oracle \*/

**SELECT LENGTH('HORATIO') "Length of characters"**

**FROM dual;**

**Рядкові функції та оператори**

*Таблица 5. Рядкові функції та оператори SQL*

|  |  |
| --- | --- |
| **Функція або оператор** | **Призначення** |
| *CONVERT* | Перетворює рядок в інше представлення, не змінюючи кодування |
| *CONCAT*, | Конкатенація двох рядків |
| *LOWER* | Перетворює всі символи в рядку до нижнього регістру |
| *OVERLAY* | Повертає результат заміни одного підрядка до іншого |
| *SUBSTRING* | Повертає частину рядка |
| *TRANSLATE* | Перетворює рядок з одного кодування до іншого |
| *TRIM* | Видаляє начальні та/або кінцеві символи |
| *UPPER* | Перетворює всі символи в рядку до верхнього регістру |