Лабораторна робота 5. Мова маніпулювання даними: прості запити, оператори порівняння, булеві та спеціальні оператори

Мета:

Навчити студентів виконувати прості запити до таблиць бази даних, а також виконувати вибірку з застосуванням операторів порівняння, булевих та спеціальних операторів.

Завдання:

1. Реалізація операції реляційної алгебри «Проекція»:

Виконати прості запити до таблиць бази даних, які повертають всі значення певних стовпців (на вибір студента).

2. Реалізація операції реляційної алгебри «Вибірка»:

Виконати запити, які повертають рядки певної(их) таблиці(ь) згідно вказаних умов з використанням операторів порівняння, булевих та спеціальних операторів.

3. Реалізація операцій зміни схеми відношення:

Виконати запити на додавання стовпця до схеми відношення; видалення стовпця зі схеми відношення; переіменування стовпця будь-якого відношення; додавання обмеження на значення; зміна значення за замовчуванням; видалення обмеження зовнішнього ключа.

Результат:

Студенти повинні подати SQL-скрипти, що відображають запити на вибірку з таблиць та запити на модифікацію схеми відношення згідно завдання та предметної області, їх опис, а також звіт з результатами тестування.

Теоретичні відомості до виконання лабораторної роботи

Команда вибірки SELECT

В даній лабораторній роботі вивчимо команду SELECT (*вибірка*). Цю команду можна назвати **основною у ММД** *SQL*, тому що вона використовується не тільки сама по собі, але і дозволяє значною мірою розширити інші команди маніпулювання даними. І, крім того, з**а її основою будується** практично половина елементів *SQL* і, відповідно, запитів **МВД** *SQL*.

Незважаючи на назву цієї команди, **вона не є**, у чистому виді, **реалізацією** реляційної операції **вибірки**. Її найпростіший формат виконує операцію **проекції**:

SELECT список_стовпців FROM таблиця;

Наприклад:

SELECT Kod, DKod, Mark FROM Rating;

Якщо необхідно вивести вміст *усіх полів* таблиці, то в команді замість *список_стовпців* указується шаблон *. Наприклад:

SELECT * FROM Student;

При цьому необхідно враховувати, що стовпці будуть виведені в тому порядку, в якому вони фізично зберігаються в таблиці чи перелічені в представленні.

Якщо ж стовпці при виводі, наприклад в інтерактивному режимі, *необхідно переставити*, то їх знов-таки *перелічують* у команді.

Необхідно відзначити, що за замовчуванням команда SELECT операцію **проекції** реалізує *не цілком*, не до кінця. Справа в тому, що *проекція* на підмножину схеми відношення має на увазі **виключення** з реляційної таблиці **кортежів, що збігаються**. Команда ж SELECT, наприклад:

SELECT SecondName FROM Student;

виведе прізвища *всіх студентів* (із усіх кортежів) у тому числі і ті, що *повторюються*. Це приклад, у якому, як і у всіх попередніх, за замовчуванням використовується *параметр* ALL. У загальному вигляді формат команди з даного прикладу такий:

SELECT ALL список_полів FROM таблиця;

Для того, щоб *виключити однакові кортежі* з результату, тобто цілком виконати операцію проекції, необхідно ключове слово ALL замінити на DISTINCT. Наприклад:

SELECT DISTINCT SecondName, FirstName FROM Student;

Необхідно відзначити, що за допомогою інструкції DISTINCT з результату проекції видаляються кортежі, які *цілком збігаються*. Відповідно в останньому прикладі повторюваними будуть вважатися рядки, у яких збігаються *й ім'я, і прізвище*.

Наступне *розширення* команди SELECT *аналогічно* команді відновлення (UPDATE) — це **вибірка** і **проекція**. І також, як у UPDATE, для цього використовується оператор WHERE *умова*. Наприклад:

SELECT * FROM Rating WHERE DKod = 2 AND Mark >= 60;

з наступним результатом:

KOD	DKOD	MARK	MDATE
2	2	76	12.10.2023
3	2	85	12.10.2023
4	2	85	12.10.2023

Причому, в *умові* команд SELECT, UPDATE і DELETE можуть використовуватися як *оператори порівняння* (=, >, <, >=, <=, <>), так і *булеві оператори* (AND, OR, NOT), що видно з останнього прикладу. Крім того, в *умові* цих команд, так само, як в *обмеженнях* таблиць і доменів, можуть використовуватися *спеціальні оператори* LIKE, BETWEEN, IN, що згадувалися в лабораторній роботі 4, плюс оператор IS NULL.

Спеціальні оператори: IN, BETWEEN, LIKE, IS NULL.

Оператор IN цілком визначає *деяку множину значень*. Наприклад:

... spec char(2) CHECK(spec IN('АП', 'ОС', 'АМ', 'АС')) ...

Оператор BETWEEN разом з оператором AND задає *діапазон значень*. Причому границі діапазону *входять* у число припустимих значень і можуть бути як числами, так і рядками ASCII-символів. В останньому випадку при порівнянні рядків різної довжини більш короткий рядок *доповнюється* пробілами, що мають найменший ASCII-код серед символів алфавіту.

Оператор IS NULL використовується в командах ММД для визначення кортежів, у яких *відсутні значення* тих чи інших атрибутів. Наприклад:

SELECT * FROM Student WHERE Patronymic IS NULL;

відповідає кортежам, у яких відсутнє по-батькові. Зворотний йому оператор IS NOT NULL дозволяє *відсіяти відсутні значення*.

Наведемо кілька прикладів:

Запит виведе всі дані, які є в таблиці Student, про студентів спеціальностей ОІ („Економічна кібернетика“) та ОС („Прикладна математика“).

SELECT * FROM Student WHERE Spec IN('ОІ', 'ОС');

Запит відобразить коди та рейтинг з певної (тут — другої) дисципліни студентів, які мають допуск до підсумкового контролю з цієї дисципліни, тобто мають рейтинг у межах 30 та 60 балів,

SELECT Kod, DKod, Mark, MDate FROM Rating
WHERE DKod = 2 AND Mark BETWEEN 30 AND 60;

Оператор LIKE застосовуваний *тільки до символьних полів* типу CHAR і VARCHAR і використовується для накладення *шаблонів* на рядки.

Для цього в операторі використовуються спеціальні **символи-шаблони**: символ „підкреслення“ ('_'), що заміняє *один будь-який* символ, і символ „відсоток“ ('%'), що заміняє символьний *рядок довільної довжини*. Наприклад, шаблону „*д_м*“ відповідають рядки „*дам*“, „*дим*“, „*дум*“.

Запит вибере студентів, прізвища яких закінчуються літерою В, а в середині чи на початку мають літеру М, та виведе всі дані про них.

SELECT * FROM Student WHERE SecondName LIKE '%М%В';

Для того, щоб у шаблоні оператора LIKE використовувати і самі символи '_' та '%'

необхідно будь-який символ, наприклад слеш ‘/’, визначити як Escape-символ і випереджати їм кожний з керуючих, у тому числі і самого себе:

… SpName char(60) CHECK(SpName LIKE ‘%/_%/%%//%’ ESCAPE ‘/’), …

У цьому прикладі команда CHECK пропустить будь-які символьні рядки, в яких у будь-якому місці, але послідовно зустрічаються символи ‘_’, ‘%’ і ‘/’. Наприклад:

„У спец_фонд виділити 15% доходу, але не більш 1000 грн/місяць“.

В діалекті SQL СУБД PostgreSQL, згідно з розширенням, введеним в стандарті SQL’99, були додані *регулярні вирази*.

Регулярні вирази застосовують для:

— пошуку у рядку підрядка, який задовольняє шаблону регулярного виразу;

— пошуку та заміни у рядку підрядка, який задовольняє шаблону регулярного виразу;

— перевірки на відповідність заданого рядка шаблону;

— добування з рядка підрядка, який задовольняє шаблону регулярного виразу.

Синтаксис регулярних виразів визначений стандартом POSIX (Portable Operating System Interface for Unix - переносимий інтерфейс операційних систем Unix). В стандарт SQL’99 було додано можливість використання регулярних виразів через оператор SIMILAR TO як розвиток оператора LIKE:

— рядок SIMILAR TO шаблон;

— рядок NOT SIMILAR TO шаблон;

Як і оператор LIKE, оператор SIMILAR TO повертає істину, якщо вміст всього рядку відповідає вмісту шаблону.

Шаблон оператора SIMILAR TO крім символів ‘%’, ‘_’ використовує додаткові символи:

— символ ‘|’ позначає альтернативи елементів;

— символ ‘?’ позначає повторення попереднього елемента 0 або 1 раз;

— символ ‘*’ позначає повторення попереднього елемента 0 або більше раз;

— символ ‘+’ позначає повторення попереднього елемента 1 або більше раз;

— символи ‘{m}’ позначають повторення попереднього елемента рівно m раз;

— символи ‘{m,}’ позначають повторення попереднього елемента m або більше раз;

— символи ‘{m, n}’ позначають повторення попереднього елемента від m до n раз;

— символи ‘( )’ групують елементи в один логічний блок;

— символи ‘[ ]’ визначають клас символів через перерахування припустимих символів або з використанням символу діапазону ‘-’;

Якщо вказані символи необхідно використовувати в шаблоні як звичайні, перед ними використовують символ ‘\’. Також потрібно пам’ятати, що в шаблоні всі символи ‘\’ потрібно також дублювати, тобто писати як ‘\\’.

Розглянемо кілька прикладів демонстрації можливостей оператору SIMILAR TO у порівнянні з оператором LIKE.

Для пошуку рядків, в яких назва вулиці з адреси студента не містить цифр, можна виконати запит:

SELECT * FROM Student WHERE (Address).Street NOT SIMILAR TO ‘%[0-9]+%’;

В шаблоні регулярного виразу визначено новий клас символів як діапазон припустимих цифр, кількість яких від 1 та більше. Якщо виключити оператор NOT, буде отримано рядки, в яких присутня будь-яка кількість цифр. Шаблон ‘[0-9]+’ забезпечить отримання рядків, в яких всі символи — цифри.

Для пошуку рядків, в яких адреса електронної поштової скриньки студента містить 3 або 4 латинські букви, після яких іде 5 цифр, можна виконати запит:

SELECT * FROM Student WHERE Email SIMILAR TO ‘%[a-z]{3,4}[0-9]{5}%’

Команда зміни визначення таблиці ALTER TABLE

Синтаксис команди *ALTER TABLE*:

Додати стовпець в таблицю

```
ALTER TABLE [ IF EXISTS ] [ ONLY ] name [ * ]
        ADD [ COLUMN ] [ IF NOT EXISTS ] column_name data_type [
column_constraint [ ... ] ]
```

В якості ***column_constraint*** можуть бути наступні альтернативи:

```
[ CONSTRAINT constraint_name ]
{ NOT NULL |
  NULL |
  CHECK ( expression ) [ NO INHERIT ] |
  DEFAULT default_expr |
  UNIQUE [ NULLS [ NOT ] DISTINCT ] index_parameters |
  PRIMARY KEY index_parameters |
  REFERENCES reftable [ ( refcolumn ) ]
    [ ON DELETE referential_action ] [ ON UPDATE referential_action ] }
```

Видалити стовпець таблиці

```
ALTER TABLE [ IF EXISTS ] [ ONLY ] name [ * ]
    DROP [ COLUMN ] [ IF EXISTS ] column_name [ RESTRICT | CASCADE ]
```

Змінити тип даних стовпця

```
ALTER TABLE [ IF EXISTS ] [ ONLY ] name [ * ]
    ALTER [ COLUMN ] column_name [ SET DATA ] TYPE data_type
```

Встановити значення за замовчуванням

```
ALTER TABLE [ IF EXISTS ] [ ONLY ] name [ * ]
    ALTER [ COLUMN ] column_name SET DEFAULT expression
```

Видалити значення за замовчуванням

```
ALTER TABLE [ IF EXISTS ] [ ONLY ] name [ * ]
    ALTER [ COLUMN ] column_name DROP DEFAULT
```

Встановити/видалити дозвіл на порожнє значення

```
ALTER TABLE [ IF EXISTS ] [ ONLY ] name [ * ]
    ALTER [ COLUMN ] column_name { SET | DROP } NOT NULL
```

Додати обмеження в таблицю

```
ALTER TABLE [ IF EXISTS ] [ ONLY ] name [ * ]
    ADD table_constraint
```

В якості ***table_constraint*** можуть бути такі альтернативи:

```
[ CONSTRAINT constraint_name ]
{ CHECK ( expression ) [ NO INHERIT ] |
  UNIQUE [ NULLS [ NOT ] DISTINCT ] ( column_name [, ... ] )
index_parameters |
  PRIMARY KEY ( column_name [, ... ] ) index_parameters |
  FOREIGN KEY ( column_name [, ... ] ) REFERENCES reftable [ ( refcolumn
[, ... ] ) ]
```

Змінити обмеження

```
ALTER TABLE [ IF EXISTS ] [ ONLY ] name [ * ]
    ALTER CONSTRAINT constraint_name
```

Видалити обмеження

```
ALTER TABLE [ IF EXISTS ] [ ONLY ] name [ * ]
    DROP CONSTRAINT [ IF EXISTS ]  constraint_name [ RESTRICT | CASCADE]
```

Додати супертип до підтипу (вказати батьківську таблицю)

```
ALTER TABLE [ IF EXISTS ] [ ONLY ] name [ * ]
    INHERIT parent_table
```

Видалити зв'язок супертип-підтип

```
ALTER TABLE [ IF EXISTS ] [ ONLY ] name [ * ]
    NO INHERIT parent_table
```

Переіменувати атрибут

```
ALTER TABLE [ IF EXISTS ] [ ONLY ] name [ * ]
    RENAME [ COLUMN ] column_name TO new_column_name
```

Переіменувати обмеження

```
ALTER TABLE [ IF EXISTS ] [ ONLY ] name [ * ]
    RENAME CONSTRAINT constraint_name TO new_constraint_name
```

Періменувати таблицю

```
ALTER TABLE [ IF EXISTS ] name
    RENAME TO new_name
```

Де

name – ім'я існуючої таблиці;
column_name – ім'я стовпця, існуючого або нового;
new_column_name – нове ім'я для існуючого стовпця;
new_name – нова назва таблиці;
data_type – тип даних нового стовпця або новий тип даних для існуючого стовпця;
table_constraint – нове обмеження для таблиці;
constraint_name – ім'я нового або існуючого обмеження;
CASCADE – автоматично видаляти об'єкти, що залежать від видаленого стовпця або обмеження;
RESTRICT – відмовити у видалення стовпця або обмеження, якщо є будь-які залежні об'єкти. Це опція за замовчуванням.

Наведемо декілька прикладів використання цієї команди:

Додати в таблицю стовпець типу VARCHAR:

```
ALTER TABLE distributors ADD COLUMN address VARCHAR(30);
```

Це призведе до того, що всі існуючі рядки в таблиці будуть заповнені NULL-значеннями для нового стовпця.

Видалити стовпець з таблиці:

```
ALTER TABLE distributors DROP COLUMN address RESTRICT;
```

Змінити типи двох існуючих стовпців за одну операцію:

```
ALTER TABLE distributors
ALTER COLUMN address TYPE varchar(80),
ALTER COLUMN name TYPE varchar(100);
```

Перейменувати існуючий стовпець:

```
ALTER TABLE distributors RENAME COLUMN address TO city;
```

Перейменувати існуючу таблицю:
```
ALTER TABLE distributors RENAME TO suppliers;
```

Перейменувати існуюче обмеження:
```
ALTER TABLE distributors RENAME CONSTRAINT zipchk TO zip_check;
```

Додати обмеження на NULL-значення до стовпця:
```
ALTER TABLE distributors ALTER COLUMN street SET NOT NULL;
```

Видалити обмеження на NULL-значення зі стовпця:
```
ALTER TABLE distributors ALTER COLUMN street DROP NOT NULL;
```

Додати перевірочне обмеження до таблиці та всіх її дочірніх елементів:
```
ALTER TABLE distributors
ADD CONSTRAINT zipchk CHECK (char_length(zipcode) = 5);
```

Додати перевірочне обмеження лише до таблиці, а не до її дочірніх елементів:
```
ALTER TABLE distributors
ADD CONSTRAINT zipchk CHECK (char_length(zipcode) = 5) NO INHERIT;
```
Обмеження перевірки також не буде успадковане майбутніми нащадками.

Видалити перевірочне обмеження з таблиці та всіх її дочірніх елементів:
```
ALTER TABLE distributors DROP CONSTRAINT zipchk;
```

Видалити обмеження перевірки лише з однієї таблиці:
```
ALTER TABLE ONLY distributors DROP CONSTRAINT zipchk;
```
Обмеження перевірки залишається на місці для будь-яких дочірніх таблиць.

Додати обмеження зовнішнього ключа до таблиці:
```
ALTER TABLE distributors
ADD CONSTRAINT distfk FOREIGN KEY (address) REFERENCES addresses (address);
```

Додати обмеження складеного потенційного ключа (з кількома стовпцями) до таблиці:
```
ALTER TABLE distributors ADD CONSTRAINT dist_id_zipcode_key UNIQUE (dist_id, zipcode);
```

Додати обмеження первинного ключа з автоматичною назвою до таблиці, пам’ятаючи, що таблиця може мати лише один первинний ключ:
```
ALTER TABLE distributors ADD PRIMARY KEY (dist_id);
```

Структура звіту до лабораторної роботи

Для кожного з запитів представити:

1) словесна постановка задачі, що вирішується;
2) SQL-код рішення;
3) скриншот отриманого результату.

1. Простий запит до таблиці бази даних, який повертає всі значення певних стовпців (на вибір студента).
2. П’ять запитів, які повертають рядки певної(их) таблиці(ь) згідно вказаних умов:

— значення атрибуту цілочисельного типу даних більше заданого цілого та значення

атрибуту символьного типу недорівнює заданому значенню;

— значення атрибуту типу дата менше ніж системна дата та значення атрибуту символьного типу містить порожні значення;

— значення атрибуту символьного типу дорівнює заданому значенню або цей же атрибут дорівнює іншому заданому значенню;

— значення атрибуту типу дата не є системною датою (NOT) та значення атрибуту символьного типу відповідає одному з перелічених в умові;

— значення атрибуту цілочисельного типу входить в заданий діапазон значень та значення атрибуту символьного типу відповідає заданому шаблону;

3. Шість запитів, які реалізують:

— додавання стовпця типу «час» до схеми відношення;

— видалення стовпця символьного типу зі схеми відношення;

— переіменування стовпця будь-якого відношення;

— додавання обмеження на значення – значення числового стовпця більше 0;

— зміна значення за замовчуванням;

— видалення обмеження зовнішнього ключа.

Звіт до лабораторної роботи 5 можна здати онлайн на сайті ДО edu.op.edu.ua до початку вашого заняття.