Лабораторна робота 4. Маніпулювання даними. Запити.

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	12.10.2011
3	2	85	12.10.2011
4	2	85	12.10.2011

Причому, в *умові* команд SELECT, UPDATE і DELETE можуть використовуватися як *оператори порівняння*, так і *булеві оператори*, 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 і плюс ще оператор IS NULL, що згадувалися в лабораторній роботі 2.

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:

SELECT * FROM Student WHERE Spec IN('OI', 'OC');

Запит виведе всі дані, які є в таблиці Student, про студентів спеціальностей OI („Економічна кібернетика") та OC („Прикладна математика").

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

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 * FROM Student WHERE SecondName LIKE '%M%B';

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

Крім умовних операторів, у командах ММД, конкретно у SELECT, використовуються так звані **агрегатні функції**, що **повертають деяке *єдине* значення поля *для підмножини* кортежів таблиці**.

Агрегатні функції

У стандарті *SQL* визначено 5 таких функцій, хоча будь-яка реалізація мови містить у 4 - 6 разів більше функцій. Особливістю агрегатних функцій є те, що, використовуючи імена полів як аргументи, самі вони вказуються в команді SELECT *замість* або *разом* з полями. Це функції AVG (*average*) і SUM (*summa*), які використовуються *тільки для числових полів*, і функції MAX, MIN і COUNT, що можуть застосовуватися і *для числових, і для символьних атрибутів*. Так функції AVG і SUM дозволяють обчислити відповідно середнє значення і суму значень певного атрибуту у **всіх кортежах** таблиці. Наприклад:

SELECT AVG(Mark) FROM Rating [WHERE DKod = 8];

Цей запит повертає середній рейтинг, підрахований, якщо умова відсутня, по всіх кортежах таблиці або, при наявності умови, тільки по кортежах, що відповідають восьмій дисципліні.

Назви функцій MAX і MIN говорять самі за себе. Функція ж COUNT підраховує *кількість значень* у стовпці чи *кількість рядків* у таблиці.

Наприклад:

SELECT COUNT(*) FROM Student;

Використання * як атрибута забезпечить підрахунок кількості *всіх* рядків у таблиці, *включаючи повторювані і NULL-еві* (яких, за ідеєю, не має бути).

Для підрахунку кількості *не-NULL-евих* значень якого-небудь атрибута використовується формат:

SELECT COUNT([ALL] *поле*) FROM *таблиця*;

Ключове слово ALL використовується *за замовчуванням*. Наприклад:

SELECT COUNT(Patronymic) FROM Student;

До речі, всі інші агрегатні функції в будь-якому випадку *ігнорують* NULL-значення.

Якщо ж необхідно підрахувати число *різних* значень якогось поля і *виключити* при цьому NULL-значення, то в аргументах функції необхідно використовувати оператор

DISTINCT. Наприклад:

```
SELECT  COUNT(DISTINCT Patronymic)  FROM  Student;
```

До речі, оператор DISTINCT може бути вказаний *у будь-якій* агрегатній функції, але в MAX і MIN він *марний*, а в SUM і AVG — не має сенсу, тому що вплине на результат.

Ще одна особливість агрегатних функцій — можливість використання *скалярних виразів* у якості їхніх аргументів, у яких, щоправда, не має бути самих агрегатних функцій. Наприклад:

```
SELECT  AVG(LongevityInc + DegreeInc + TitleInc + SpecialInc)
    FROM  SalaryIncrements;
```

У деяких розширеннях *SQL*, зокрема *PostgreSQL* останнє обмеження зняте.

Скалярні вирази можуть бути аргументами і самої команди SELECT, також як і команди UPDATE. У цьому випадку в них можуть входити і поля, і числові константи, і агрегатні функції.

Наприклад:

```
SELECT (MAX(LongevityInc)+MAX(DegreeInc)+MAX(TitleInc)+MAX(SpecialInc))/4
    FROM  SalaryIncrements;
```

Символьні константи у виразах використовуватися *не можуть*. Але зате їх можна просто включити у вихідну таблицю результату вибірки *в інтерактивному режимі*.

Наприклад:

```
SELECT  Kod, DKod, Mark, 'на 14.10.11'  FROM  Rating;
```

Результат — всі рядки таблиці з додатковим полем:

Kod	DKod	Mark	
1	1	82	на 14.10.11
1	2	10	на 14.10.11
…	…	…	…
8	1	0	на 14.10.11

Чи наприклад:

```
SELECT  AVG(Mark), 'до 10-го тижня'  FROM  Rating;
```

З визначення агрегатних функцій видно, що вони *не можуть* використовуватися в одному SELECT-запиті разом з полями, тому що повертають *одне єдине значення*. Наприклад (***неправильно***):

```
SELECT  Kod, MAX(Mark)  FROM  Rating;
```

Однак на практиці може виникнути така задача: необхідно *отримати середнє значення Mark у межах певної дисципліни*.

Розв'язати цю задачу допоможе оператор GROUP BY, що **забезпечує виділення підгрупи з конкретним значенням атрибута чи підмножини атрибутів** і застосування агрегатних функцій до кожної підгрупи. Наприклад:

```
SELECT  DKod, AVG (Mark)  FROM  Rating
    GROUP BY  DKod;
```

Результат може бути такий:

DKod	
1	60
2	64
3	30
4	97
5	65

Тепер розширимо цей приклад. Припустимо, що необхідно з відношення (таблиці), отриманої в попередньому прикладі, виконати **вибірку** кортежів, для яких *середнє значення* більше ніж, наприклад, 75. Ми вже знаємо, що операція вибірки реалізується за

допомогою оператора WHERE. Однак *результуюча таблиця* цього прикладу будується *на основі груп* з використанням агрегатної функції, тому тут існує низка обмежень.

Для розв'язання подібних задач, де в умові запиту використовується агрегатна функція чи поле, на яке виконується проекція, у *SQL* був уведений ще один оператор, що реалізує операцію **вибірки** — HAVING.

Відтак запит, що відповідає поставленій задачі буде виглядати таким чином:

```
SELECT    DKod, AVG(Mark)      FROM      Rating
      GROUP BY     DKod
      HAVING       AVG(Mark) > 75;
```

DKod	
4	97

з таким результатом:

У вислові HAVING можна використовувати і просто імена полів. Єдина умова при цьому — таке поле чи підмножина полів повинна мати одне і теж значення в межах групи.

Наприклад:

```
SELECT          DKod, AVG(Mark)      FROM      Rating
GROUP BY   DKod          HAVING          DKod <> 4;
```

DKod	
1	60
2	64
5	65

для виключення з результату дисципліни з кодом 4.

Чи, наприклад:

```
… HAVING     NOT      DKod  IN(3, 4);
```

для виключення з результату дисциплін з кодом 3 та 4:

Таким чином, оператор HAVING аналогічний WHERE за винятком того, що рядки відбираються не за значеннями стовпців, а будуються зі значень стовпців зазначених в GROUP BY і значень агрегатних функцій, обчислених для кожної групи, утвореної GROUP BY.

Якщо ж необхідно додати в запит з GROUP BY умову, що містить поле, яке не використовується для групування, то така умова, як і раніше, задається за допомогою WHERE і навіть одночасно з HAVING. Наприклад:

```
SELECT    Kod, AVG(Mark)       FROM      Rating
      WHERE     DKod = 5
      GROUP BY          Kod      HAVING   NOT    Kod     IN(3, 4);
```

Цей запит виключить з підрахунку середнього рейтингу кортежі, що стосуються студентів, що мають код 3 або 4, і підрахує його для п'ятої дисципліни.

Наступний момент, який ми розглянемо, зв'язаний із ***сортуванням значень***, отриманих *у результаті* запиту SELECT. Справа в тому, що кортежі в таблиці зберігаються *в порядку їхнього надходження* (введення). Відповідно при введенні (за замовчуванням) цей порядок буде збережений. Виняток становлять проіндексовані поля. Очевидно, більш зручним для використання є введення множини кортежів таблиці, *відсортованих* відповідно до значень множини символьних та/або числових полів *у прямому чи зворотному порядку*. Для забезпечення такої можливості в *SQL* введений оператор ORDER BY. Наприклад:

```
SELECT    *      FROM      Student     ORDER BY SecondName;
```

Однак у цьому прикладі кортежі, що мають однакове значення прізвища, можуть виявитися невпорядкованими за ім'ям, по-батькові тощо, що також незручно. Поправимо цей приклад:

```
SELECT    *      FROM      Student     ORDER BY SecondName, FirstName, Patronymic;
```

Для сортування кортежів у зворотному алфавітному порядку значень чи атрибута за убуванням використовується ключове слово DESC. Наприклад:

SELECT DKod, Mark FROM Rating ORDER BY DKod, Mark DESC;

(за *DKod — прямий*, за *Mark — зворотний* порядок).

Причому поле, за яким виконується сортування, *не обов'язково* має бути присутнім у підмножині, на яку виконується проекція.

Це особливо актуально у вбудованому режимі, коли результати різних операцій проекції того ж самого відношення виводяться в різні вікна. Наприклад:

1) SELECT DKod FROM Rating ORDER BY DKod;

2) SELECT Kod, Mark FROM Rating ORDER BY DKod, Mark DESC;

Аналогічні підходи можуть використовуватися і для таблиць, отриманих у результаті угруповання. Наприклад:

SELECT DKod, AVG(Mark) FROM Rating GROUP BY DKod ORDER BY DKod;

Однак відсортувати подібну таблицю за результатом агрегатної операції не є можливим, тому що таке поле не пойменоване і не існує ні в базовій таблиці, ані в представленні. На допомогу тут приходить внутрішня (автоматична) нумерація вихідних стовпців відповідно до порядку перерахування в команді SELECT. Наприклад:

SELECT DKod, AVG(Mark) FROM Rating GROUP BY DKod ORDER BY 2 DESC;

DKod	
4	97
5	65
2	64
1	60
3	30

Запит дасть один з отриманих вище результатів, але в іншому порядку:

На жаль, це єдиний спосіб звернутися до таких стовпців, незважаючи на те, що їх можна пойменувати, задавши їм псевдоніми чи, використовуючи вже прийнятий термін, **аліаси**. Наприклад:

SELECT DKod Discipline, AVG(Mark) Averrage_Rating
FROM Rating GROUP BY DKod;

Раніше ми відмітили, що будь-який діалект *SQL* містить набагато більше агрегатних функцій чи подібних агрегатним. Так в ***PostgreSQL*** були введені ***аналітичні*** чи ***віконні*** функції, які окрім задач агрегування, виконують ще й частку операцій над багатовимірними структурами.

Представлення

Представлення (VIEW) — об'єкт, що не містить власних даних. Це іменована *похідна віртуальна таблиця*, що *не може існувати сама по собі*, а визначається в термінах однієї або декількох іменованих таблиць (*базових таблиць* або інших *представлень*).

У загальному випадку термін **похідна таблиця** позначає таблицю, що визначається в термінах інших таблиць і, в остаточному підсумку, у термінах *базових таблиць*, тобто є результатом виконання яких-небудь реляційних виразів над ними. **Базова таблиця** — це така таблиця, що не є похідною.

Більш того, будь-які зміни в основній таблиці будуть *автоматично і негайно* видані через таке „вікно". І навпаки, зміни в представленні будуть *автоматично і негайно* застосовані до його базової таблиці.

У дійсності ж **представлення — це запити**, які виконуються щоразу, коли представлення є об'єктом команди *SQL*.

Формат команди **знищення** досить простий:

DROP VIEW *представлення*;

Команда **створення** представлення має формат:

CREATE VIEW *представлення*[(*імена_стовпців*)] AS *запит*;

Наприклад:

CREATE VIEW Rating_D AS

```
SELECT    Kod, Mark, MDate  FROM   Rating
WHERE     DKod=1 AND Mark >= 30;
```

Це представлення буде містити коди, рейтинг та дати проведення модулю по першій дисципліні тих студентів, у яких цей рейтинг не менший 30 балів:

KOD	MARK	MDATE
1	82	2011-10-10
2	90	2011-10-10
3	46	2011-10-11
4	54	2011-10-10
5	86	2011-10-10

Значення фрази про те, що представлення — це запит, найбільш видний при звертанні до представлень, як до таблиць. Таким чином, команда

```
SELECT    *       FROM   Rating_D WHERE Mark < 60;
```

на практиці конвертується і оптимізується в команду:

```
SELECT    Kod, Mark, MDate       FROM       Rating
      WHERE     DKod=1 AND Mark >= 30 AND Mark < 60;
```

У цьому досить простому представленні як імена полів використовувалися безпосередньо імена полів таблиці, що лежить в основі представлення. Іноді, як у наведеному прикладі, цього досить. Але іноді бажано дати нові імена стовпцям представлення. А іноді — це просто необхідно. Наприклад, у тих випадках коли:

а) деякі стовпці є *вихідними* і, отже, *не поіменовані*. Така ситуація часто виникає при *об'єднанні* відношень з *різними іменами* однотипних атрибутів;

б) два або більше стовпців мають *однакові імена* в таблицях, що беруть участь у *з'єднанні*.

Імена, що стануть іменами стовпців представлення, вказуються в круглих дужках після його імені. Типи даних і розміри *автоматично виводяться* з полів запиту. Наприклад:

```
CREATE VIEW  Rating_D(Student_Number, Discipline_Number,  Rating, Date_of_Mark)
     AS
   SELECT  Kod, DKod, Mark, MDate  FROM Rating       WHERE  Mark >= 30;
```

При цьому приклад з вибіркою значень трансформується в такий:

```
SELECT    *       FROM       Rating_D          WHERE         Rating < 60;
```

Приклад рішення завдання до лабораторної роботи 4.

П.1. Виберіть напрямки польотів певного типу літака (за вибором студента).

Розв'язання.

```
SELECT destination FROM voyage WHERE type_aircraft='Boeing 747';
```

Результуюча таблиця.

destination
Одеса – Нью Йорк
Одеса – Пекін

П.2. Створіть представлення (*view*), що містить інформацію про напрямки рейсів, кількість та загальну суму проданих квитків на кожен напрямок. Результат відсортуйте за напрямком рейсу.

Розв’язання.

```
CREATE VIEW sale AS
SELECT destination, COUNT(place), SUM(C.price) FROM ticket T, voyage V, class C
    WHERE T.class=C.id_class AND C.voyage=V.id_voyage
    GROUP BY destination ORDER BY destination;
```

Результуюча таблиця.

destination	count	SUM
Одеса – Лондон	3	2100.00
Одеса – Нью Йорк	4	3200.00

П.3. Виберіть з бази даних ПІБ пасажирів та рейси, на які заброньовано квитки.

Розв’язання.

```
SELECT P.full_name, V.number FROM passenger P, voyage V, ticket T, class C
    WHERE P.id_passenger=T.passenger AND C.id_class=T.class AND
                                C.voyage=V.id_voyage AND T.operation='бронь';
```

Результуюча таблиця.

full_name	number
Малахов Є.В.	RK 3467
Лінгур Л.М.	RK 4578

П.4. Виберіть з бази даних пункт призначення рейсу, назву класу та кількість місць проданих на кожен з напрямків. Дані відсортуйте за пунктом призначення.

Розв’язання.

```
SELECT V.destination, C.name, COUNT (T.place) FROM ticket T, voyage V, class C
    WHERE T.class=C.id_class AND C.voyage=V.id_voyage AND T.class=C.id_class
    GROUP BY V.destination, C.name;
```

Результуюча таблиця.

Destination	name	count
Одеса – Лондон	економ	1
Одеса – Лондон	економ покращений	1
Одесса-Лондон	бізнес	1
Одесса-Нью Йорк	економ покращений	4

Завдання до лабораторної роботи 4

Створіть **мінімум 10 запитів** з використанням наступних операторів та об’єктів:
1) операторів порівняння;
2) агрегатних функції (AVG, SUM, MIN, MAX, COUNT);
3) спеціальних операторів (IN, BETWEEN, LIKE, IS NULL);
4) сортування;
5) групування;
6) оператора HAVING;
7) представлення;
8) вибірку з декількох таблиць, використовуючи аліаси.