Лабораторна робота 6. Агрегатні функції, представлення

Мета:

Навчити студентів виконувати запити до таблиць бази даних з використанням агрегатних функцій, виконувати сортування та групування даних, а також створювати представлення.

Завдання:

1. Вибірка з використанням агрегатних функцій:

Виконати запити на вибірку даних, які повертають єдине значення для підмножини кортежів.

2. Вибірка з використанням групових операцій:

Виконати запити на вибірку даних, які повертають результат обробки в рамках певних підгруп.

3. Вибірка з використанням операції сортування:

Виконати запити на вибірку даних, які повертають кортежі результату вибірки у впорядкованому вигляді.

4. Вибірка з результатів роботи групових операцій:

Виконати запити на вибірку даних, які повертають результат обробки тільки в заданих підгрупах.

5. Створення представлень:

Виконати запити на створення представлення (view) згідно завдання та предметної області.

Результат:

Студенти повинні подати SQL-скрипти, що відображають запити на вибірку з таблиць бази даних з використанням агрегатних функцій, операцій сортування та групування, запити на створення представлень згідно завдання та предметної області, їх опис, а також звіт з результатами тестування.

Теоретичні відомості до виконання лабораторної роботи

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   DKodHAVING          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;

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

DKod	
4	97
5	65
2	64
1	60
3	30

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;
```

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. Виконати запити на вибірку даних, які повертають:
1.1) максимальне значення атрибуту типу дата;
1.2) мінімальне значення атрибуту символьного типу;
1.3) суму значень атрибуту десяткового типу;
1.4) середнє значення атрибуту цілого типу;
1.5) кількість рядків будь-якої сутності.
2. Виконати запити на вибірку даних, які повертають:
2.1) середнє значення атрибуту цілого типу для кожної групи за значенням атрибуту символьного типу;
2.2) кількість рядків в кожній групі (на вибір студента).
3. Виконати запити на вибірку даних, які повертають:
3.1) всі дані таблиці, впорядковані за зростанням за будь-яким символьним атрибутом;
3.2) згруповані та відсортовані в зворотньому порядку за значенням символьного поля кортежі таблиці, в яких значення цілого атрибуту більше заданого значення (тобто відібрати кортежі за умовою, згрупувати за символьним полем та відсортувати за цим же символьним полем).
4. Виконати запити на вибірку даних, які повертають:
4.1) середнє значення атрибуту цілого типу для підгруп за значенням атрибуту символьного типу, які недорівнюють заданому значенню (тобто виділити підгрупи за атрибутом символьного типу, взяти середнє значення атрибуту цілого типу в кожній підгрупі, вивести тільки ті підгрупи, які недорівнюють заданому значенню).
5. Виконати запити на:

5.1) створення представлення для завдання пункту 3.2.

5.2) виконати запит до представлення створеного в пункті 5.1, який повертає значення тільки однієї підгрупи.

Звіт до лабораторної роботи 6 можна здати онлайн на сайті ДО edu.op.edu.ua до початку вашого заняття.