Індекси

Досить часто виникає необхідність знайти кортеж або групу кортежів з конкретним значенням певної підмножини атрибутів. Такий пошук частіше всього потребує повного перебору рядків таблиці.

Для прискорення розв'язання таких задач в *SQL* підтримуються *індекси*.

Індекс — це упорядкований (в алфавітному або числовому порядку) список вмісту стовпців або групи стовпців у таблиці.

Управління індексами

— *істотно сповільнює* виконання операцій відновлення (INSERT, DELETE),

— індекс займає *додаткове місце в пам'яті*.

Тому цей засіб *SQL* не було внесено в стандарт ANSI, хоча підтримується всіма СУБД.

Синтаксис команди створення індексу:

CREATE INDEX *індекс* ON *таблиця* (*стовпець*[, *стовпець*[, …]]);

Якщо вказано *більш одного атрибута* для створення одного індексу, то дані упорядковуються за значенням *першого поля*, *усередині груп, що* вийшли, здійснюється упорядкування за *значенням другого* поля*, усередині отриманих* груп — за *значенням* третього і т.д.

Суттєвий нюанс: будучи один раз створеним, індекс є *невидимим* для користувача.

Інструкція

CREATE UNIQUE INDEX *індекс* ON *таблиця*(*поле*[, *поле*[, …]]);

фактично дублює первинний ключ. При цьому, якщо таким чином індексується таблиця з *вже існуючими* кортежами, що містять *однакові значення* атрибутів указаних в інструкції, то ця команда буде *відхилена*.

Залежно від СУБД для таблиці може бути створено *до 250* індексів. Знищити індекс можна командою

DROP INDEX *індекс*;

яка *не змінює* вмісту полів.

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*).

Розходження ж між базовою таблицею і представленням характеризується так:

— базові таблиці „*реально існують*" у тім змісті, що вони представляють дані, дійсно збережені в БД;

— представлення ж „*реально не існують*", а просто представляють різні схеми *перегляду* „реальних" даних. Іншими словами, представлення подібні *вікнам*, через які проглядається інформація, що міститься в базових таблицях.

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*.

Створення і знищення представлення здійснюється командами CREATE VIEW і DROP VIEW.

Формат команди **знищення**:

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

Приклад

```
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;
```

Випадки, коли необхідно дати нові імена стовпцям представлення:

а) деякі стовпці є *вихідними* і, отже, *не пойменовані*. 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*.

Приклад:

```
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;
```

В таких випадках, коли інтенсивність модифікації даних незрівнянно менша за інтенсивність вибірки або періодичність модифікації значно більше за вибірку, доцільним стає зберігання даних, які відображаються через представлення. Для розв'язання зазначеної проблеми в деяких СУБД (зокрема, *Oracle* та *PostgreSQL*) введено додатковий елемент МВД *SQL* — **матеріалізоване представлення**.

В СУБД ***PostgreSQL*** матеріалізоване представлення — це поєднання таблиці і звичайного (віртуального) представлення:

```
CREATE TABLE    таблиця [(імена_стовпців)]     AS      запит;
```

Тобто в *PostgreSQL* матеріалізоване представлення — це *реальна* таблиця, але яка створюється на підставі запиту.

Мова маніпулювання даними (ММД, DML) SQL

Основні команди ММД SQL

Основні оператори ММД — це SELECT **(оператор вибірки)**, INSERT, UPDATE і DELETE **(оператори відновлення)**.

Усі *нові кортежі* в *SQL* **вводяться** в таблиці за допомогою команди INSERT:

INSERT INTO *таблиця* VALUES(*значення, значення,* …);

Значення при цьому мають *позиційне значення* і не можуть містити *вирази*. Наприклад:

INSERT INTO Rating VALUES(111, 5, 85, '19.11.2011');

Якщо ж необхідно „пропустити" значення якого-небудь поля при введенні, можна:

— вставити *NULL-значення* у відповідну позицію:

```
INSERT INTO Student
    VALUES(111, 'Іванов', 'Іван', NULL, 12, 'AM', 823, NULL, NULL);
```

Це можливо тільки в тому випадку, якщо в даному полі *припустимі* такі значення;

— *явно вказати імена* стовпців, до котрих будуть уведені нові значення. Пропущеним атрибутам будуть надані або NULL-значення, якщо вони припустимі, або значення за замовчуванням, якщо вони існують (задані). Приклад:

INSERT INTO Rating(MDate, Kod, DKod) VALUES ('21.11.2011', 222, 3);

Рядки з таблиці можна **виключити** за допомогою команди

DELETE FROM *таблиця* [*умова*];

Якщо *не* вказана *умова*, то команда виконує „очищення" таблиці:

DELETE FROM Rating;

Використання умови дозволяє видаляти *кілька кортежів* таблиці:

DELETE FROM Student WHERE GNum < 941;

Якщо в умові вказане ім'я *первинного ключа*, то це забезпечить видалення *одного єдиного* кортежу.

Наступна команда МВД *SQL* дозволяє **змінювати** значення **вже існуючих** полів таблиці:

UPDATE *таблиця* SET *поле = значення*[, *поле = значення*[, …]] [*умова*];

Це найпростіший формат команди, причому ключове слово SET забезпечує реалізацію реляційної операції **проекції** на підмножину схеми відношення, представленої таблицею. Наприклад:

```
UPDATE  Rating    SET  Mark = 30;
```

Ця команда забезпечує запис значення 30 до поля *Mark* **усіх кортежів** таблиці. В інструкції SET *припустиме використання скалярних виразів*. Наприклад:

```
UPDATE  Rating    SET  Mark = Mark +(70 - Mark)*0.2;
```

Для модифікації неодиничної підмножини стовпців їх необхідно перелічити у вислові SET:

```
UPDATE  Rating    SET  Mark = NULL, DKod = 2;
```

Операція **вибірки** з **проекції** дозволяє змінити значення полів лише частини кортежів таблиці. Для цього в команду UPDATE додається *умова*. Наприклад:

```
UPDATE  Student  SET  patronymic = 'Валерійович', GNum = 842
    WHERE        Kod = 111;
```

Команда SELECT (*вибірка*) **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;
```

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

```
SELECT DISTINCT  SecondName, FirstName    FROM    Student;
```

Наступне *розширення* команди SELECT це **вибірка** і **проекція**. 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 згадувався раніше. Наведемо кілька прикладів:

```
SELECT * FROM Student
    WHERE Spec IN('AI', 'AC');
```

Запит виведе всі дані, які є в таблиці Student, про студентів спеціальностей AI (122 «Комп'ютерні науки») та AC (121 «Комп'ютерна інженерія»).

```
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.

```
SELECT * FROM Student
    WHERE Patronymic IS NULL;
```

Запит вибере та виведе всі дані про студентів, які не мають по-батькові.

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

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*,

функції MAX, MIN і COUNT, що можуть застосовуватися і *для числових, і для символьних атрибутів*.

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 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;
```

Всі інші агрегатні функції в будь-якому випадку *ігнорують* NULL-значення.

Якщо 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;
```

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;
```

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 14.10.20
1	2	10	на 14.10.20
…	…	…	…
8	1	0	на 14.10.20

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

Для розв'язання 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.

Наприклад:

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

DKod	
4	97

У вислові HAVING можна використовувати і просто імена полів, 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;
```

для виключення з результату дисципліни з кодом 4.

Чи, наприклад:

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

DKod	
1	60
2	64
5	65

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.

Для забезпечення можливості ***сортування значень*** 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.

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;
```

В ***PostgreSQL*** були введені ***аналітичні*** чи ***віконні*** функції, які окрім задач агрегування, виконують ще й частку операцій над багатовимірними структурами.