Мова управління даними SQL (МУД, DCL)

Управління доступом

Поняття управління доступом має сенс, у першу чергу, у багатокористувальницькому режимі роботи СУБД.

На рівні БД управління доступом полягає в створенні і видаленні користувачів шляхом призначення їхніх імен, паролів і прав доступу до конкретних БД і є прерогативою адміністратора БД (DBA або SYSDBA). Кожен користувач є власником таблиць, пов'язаних з тією або іншою БД. Відповідно, власник таблиці може керувати доступом до даних на рівні таблиць. Таке управління полягає в призначенні **привілеїв** іншим користувачам, тобто визначення того, може чи ні конкретний користувач виконувати ту або іншу команду.

Привілеї *SQL* є **об'єктними**. Тобто користувач має привілей (право) виконувати *конкретну команду* для *конкретного об'єкта* БД: таблиці, представлення, стовпця тощо.

Привілеї, призначені власником таблиці, збігаються за мнемонікою та за змістом з командами МВД *SQL*: SELECT, INSERT, DELETE, UPDATE та REFERENCES.

Для надання того чи іншого доступу до таблиці (таблиць) їхній власник повинен виконати команду GRANT. Її формат:

GRANT *список_операцій* ON *таблиця* TO *список_користувачів*;

Наприклад:

```
GRANT SELECT, INSERT ON Student TO User1, User2;
```

З наведеної множини привілеїв необхідно виділити UPDATE і REFERENCES, тому що вони, крім загального формату команди GRANT, *дозволяють обмежити привілеї множиною стовпців*. Наприклад:

```
GRANT UPDATE(Mark) ON Rating TO Lecturer10;
```

для відновлення полів.

Або, наприклад:

```
GRANT REFERENCES (Spec) ON Speciality TO DeptHead;
```

Ця інструкція дозволяє користувачу *DeptHead* створювати зовнішні ключі, які будуть посилатися на поле *Spec* таблиці *Speciality* як на батьківський ключ. Якщо в цьому привілеї не вказані стовпці, то в якості батьківських можуть

використовуватися будь-які поля цієї таблиці.

Наступний момент. Якщо в *списку_операцій* команди GRANT перераховані **всі привілеї**, тобто якому-небудь користувачеві відкритий **повний доступ** до таблиці, то замість такого списку в *SQL* можна скористатися інструкцією ALL PRIVILEGES. Наприклад:

```
GRANT    ALL [PRIVILEGES]    ON    Student    TO    DeptHead;
```

Аналогічно, якщо **який-небудь** привілей чи список привілеїв необхідно відкрити **для кожного користувача** системи, навіть тих, котрі тільки **будуть створені**, то замість *списку_користувачів* досить вказати оператор PUBLIC. Наприклад:

```
GRANT    SELECT    ON    Rating    TO    PUBLIC;
```

Інструкція в даному прикладі відкриває всім користувачам доступ для читання **всіх даних** таблиці Rating: така дія привілею SELECT.

Однак часто виникає необхідність обмежити перегляд таблиці конкретними полями. Тому що указати **стовпці** можна *тільки в привілеї* UPDATE і REFERENCES, у цьому випадку необхідно скористатися представленнями. Наприклад, створити представлення:

```
CREATE VIEW    StudEmail    AS
      SELECT    SecondName, FirstName, Email    FROM    Student;
```

і надати іншому користувачеві права на перегляд вже не таблиці *Student*, а представлення *StudEmail*:

```
GRANT    SELECT    ON    StudEmail    TO    PUBLIC;
```

На відміну від застосування привілеїв до базових таблиць, привілеї на представлення дозволяють обмежити доступ не тільки до стовпців, але і до кортежів. Наприклад:

```
CREATE VIEW    OSStudents
AS    SELECT    *    FROM    Student
            WHERE    Spec = 'OI'
            WITH CHECK OPTION;

GRANT    UPDATE    ON    OSStudent    TO    DeptHead;
```

Інструкція WITH CHECK OPTION не дозволяє змінювати шифр спеціальності.

Ще одна інструкція, зовні схожа на цю —— WITH GRANT OPTION — дозволяє **транслювати привілеї** по

„ланцюжку“ користувачів. Наприклад, якщо в групі розроблювачів таблиці створюють одні користувачі, додатки – інші, а використовувати дані будуть треті, то власники таблиць можуть *передати повноваження з призначення привілеїв* іншим користувачам:

```
GRANT UPDATE, SELECT ON OSStudents TO DeptHead
    WITH GRANT OPTION;
```

Тоді користувач *DeptHead* може виконати запит:

```
GRANT SELECT ON User1.OSStudents TO ViceDeptHead;
```

де *User*1 — власник таблиці Student і представлення *OSStudents*.

Скасування привілеїв здійснюється командою REVOKE, що має формат абсолютно ідентичний команді GRANT.

В ***PostgreSQL*** існує можливість управляти базами даних, доступом до них і розв’язувати безліч задач над ними, використовуючи **командну** консоль *PostgreSQL*. Для виклику командної консолі призначена команда *psql*, яка підтримує низку опцій.

Наприклад, для отримання інформації про поточні привілеї, які надані ролям при організації доступу до таблиць, використовується опція \dp *Ім’я_таблиці* або \z *Ім’я_таблиці*.

Виклик командної консолі для розв’язання цієї задачі для таблиці *Lecturer* виглядає таким чином:

```
psql /z Lecturer
```

Опція \du [*Шаблон*] надає список *всіх* ролей бази даних або тільки ролей, які відповідають шаблону.

Привілеї доступу описуються списком доступу, елементи якого описуються наступною структурою:

Ім’я_ролі = Список_прав_доступу / Ім’я_власника_таблиці

Список прав доступу може містити наступні символи:

a, r, w, d, x, t — права доступу, відповідно, на *внесення* нового запису, *читання* записів, *зміну* записів, *видалення* записів, *контроль* посилань, *виконання* тригерів.

Особливості управління доступом у СУБД PostgreSQL

Ролі та управління ролями

У версіях СУБД *PostgreSQL* до 8.1 при управлінні користувачами і групами користувачів використовувалися відповідні команди створення CREATE USER, CREATE GROUP та зміни параметрів користувача або групи ALTER USER і ALTER GROUP, відповідно. У версіях СУБД *PostgreSQL* починаючи з 8.1 з'явився більш гнучкий механізм управління — **ролі**.

Для ***створення ролі*** використовується команда CREATE ROLE.

CREATE ROLE *Ім'я_ролі* [[WITH] *Опція* [*Опція*, ...]]

Опціями є:

SUPERUSER | NOSUPERUSER	роль із правами адміністратора / без прав (за замовчуванням — NOSUPERUSER)
CREATEDB | NOCREATEDB	роль може створювати свої БД / не може (за замовчуванням — NOCREATEDB)
CREATEROLE | NOCREATEROLE	роль може створювати інші ролі / не може
INHERIT | NOINHERIT	роль автоматично буде / не буде успадковувати привілеї наслідуваних ними ролей (за замовчуванням — NOINHERIT)
LOGIN | NOLOGIN	роль може встановлювати з'єднання / не може (для групи) (за замовчуванням — LOGIN)
CONNECTION LIMIT connlimit	кількість одночасних підключень ролі до СУБД (за замовчуванням — немає обмежень)
[ENCRYPTED | UNENCRYPTED] PASSWORD *Пароль*	пароль для авторизованого встановлення з'єднання ролі із СУБД, може зберігатися в зашифрованому виді або відкрито (за замовчуванням — ENCRYPTED)
VALID UNTIL *дата*	крайній термін дії ролі (за замовчуванням — без строку дії)

Наприклад, для створення ролі *Director*, яка буде користувачем СУБД (має право підключатися до СУБД через процедуру авторизації), необхідно виконати команду:

CREATE ROLE Director LOGIN;

Наприклад, для створення ролі *Director*, яка буде користувачем СУБД до 31-10-2007, необхідно виконати

команду:

CREATE ROLE Director LOGIN WITH PASSWORD ‘dbms’ VALID UNTIL ‘2007-10-31’;

Наприклад, для створення ролі *Programmers*, яка надалі буде відповідати групі користувачів, необхідно виконати команду

CREATE ROLE Programmers NOLOGIN;

Для **зміни параметрів ролі** використовується команда ALTER ROLE:

ALTER ROLE ім’я_ролі [[WITH] *Опція* [*Опція, …*]]

Опції цієї команди збігаються з опціями команди CREATE ROLE.

Наприклад, для зміни пароля користувача необхідно виконати команду:

ALTER ROLE postgres WITH PASSWORD ‘dbms2000’;

Для **управління спадкуванням ролей** також, як у стандартному *SQL* для управління привілеями, використовуються команди GRANT і REVOKE.

Команда GRANT дозволяє одній ролі успадковувати привілеї іншої ролі та має синтаксис:

GRANT *Ім’я_ролі_яка_надає_привілеї*
TO *Ім’я_ролі_яка_одержує_привілеї*;

Наприклад, для надання ролі *Director* привілеїв ролі *Programmers* необхідно виконати команду:

GRANT Programmers TO Director;

Команда REVOKE, яка дозволяє з однієї ролі зняти привілеї іншої ролі, має такий синтаксис:

REVOKE *Ім’я_ролі_яка_надає_привілеї*
FROM *Ім’я_ролі_яка_одержує_привілеї*;

Наприклад, для зняття з ролі *Director* привілеїв ролі *Programmers* необхідно виконати команду:

REVOKE Programmers FROM Director;

Якщо роль створена без спадкування ролей за замовчуванням (опція INHERIT), при роботі із цією роллю необхідно примусово встановлювати ролі-спадкоємці за допомогою команди SET ROLE. Наприклад:

SET ROLE *Ім’я_успадкованої_ролі*;

для встановлення ролі властивості автоматичного спадкування

Скидання ролей може виконуватися двома способами:

```
SET ROLE NONE;
```

або

```
RESET ROLE;
```

При роботі з ролями можна використовувати дві системні змінні

SESSION_USER ім’я користувача , який відкрив сесію;

CURRENT_USER ім’я користувача, який працює під заданою роллю.

Для отримання значень цих змінних необхідно виконати інструкцію:

```
SELECT SESSION_USER, CURRENT_USER;
```

Після створення ролі, яка може встановлювати з’єднання з СУБД, це з’єднання можна встановити.

Наприклад, для встановлення з’єднання із СУБД користувачеві *Director*:

psql postgres -U Director

Управління схемами даних

У більшості СУБД для логічного поділу даних на окремі іменні простори використовується поняття **схем** (*schema*). Схема дозволяє створювати об’єкти БД (таблиці, представлення, функції) *в окремому просторі імен*, доступ до якого здійснюється еквівалентно об’єктному поданню: *ім’я_схеми.ім’я_об’єкта*.

Для **створення схеми даних** використовується операція

```
CREATE SCHEMA Ім’я_Схеми;
```

Приклад створення схеми *Director*:

```
CREATE SCHEMA Director;
```

Для **надання користувачу прав доступу до схеми** використовується команда:

```
GRANT USAGE ON SCHEMA назва_схеми TO назва_користувача;
```

Приклад надання користувачу прав доступу до схеми

```
GRANT USAGE ON SCHEMA Director TO Director;
```

Для **призначення користувача власником схеми** використовується команда:

ALTER SCHEMA *Ім'я_схеми* OWNER TO *Ім'я_користувача*;

Приклад включення користувача *Director* у схему *Director*:

ALTER SCHEMA Director OWNER TO Director;

Незважаючи на належність схеми відповідному користувачу, всі його запити будуть виконуватися до схеми *public*. Тому при виконанні запитів до конкретної схеми необхідно в якості префікса відповідного об'єкта вказувати її ім'я. Наприклад:

SELECT * FROM Director.Lecturer;

Після встановлення з`єднання з БД СУБД автоматично виконує пошук всіх таблиць з запитів в схемі *public*.

Для **встановлення порядку доступу до схем**, тобто порядку пошуку таблиць в схемах, використовується команда:

SET SEARCH_PATH TO *Ім'я_схеми*;

Наприклад:

SET SEARCH_PATH TO PUBLIC;

Коли для користувача визначено декілька схем, назви схем до імен таблиць в запитах додаються у вказаному порядку. Якщо таблиця не існує в схемі з назвою *ім'я_схеми*1, СУБД буде використовувати наступну назву схеми — *ім'я_схеми*2, доки таблиця не буде знайдена в схемі.

Приклад встановлення порядку пошуку:

SET SEARCH_PATH TO Director, PUBLIC;

Для того, щоб користувач після встановлення з`єднання з СУБД автоматично визначав порядок пошуку в схемах, використовується команда:

ALTER ROLE *Ім'я_користувача* SET SEARCH_PATH = *Ім'я_схеми*1 [, *Ім'я_схеми*2,…];

Наприклад, для того, щоб користувачу *director* автоматично встановлювався порядок доступу до схем *Director*, *PUBLIC* можна виконати команду:

ALTER ROLE Director SET SEARCH_PATH TO Director, PUBLIC;

Схеми доцільно використовувати з метою:

1) логічного розподілу таблиць між групами в одній фізичній БД;
2) обмеження доступу до таблиць з боку різних користувачів на основі представлень.

Наведемо приклад другої мети використання схем.

Нехай в БД існує таблиця:

```
CREATE TABLE Persons
        (Person_Id      INTEGER       PRIMARY KEY,
         Name           VARCHAR(30),
         Sex            CHAR(1),
         Birthday       DATE);
```

до якої внесемо тестову інформацію:

```
INSERT INTO Persons VALUES(1,'Иванов','M','01/01/2000');

INSERT INTO Persons VALUES(2,'Петров','M','01/01/1990');
```

Необхідно користувачу *Director*, коли він виконує запит

```
SELECT * FROM Persons;
```

заборонити отримувати інформацію про людей, яким на поточний момент ще не виповнилося 18 років.

Для цього створимо представлення:

```
CREATE OR REPLACE VIEW Director.Persons AS
        SELECT * FROM Persons
                WHERE EXTRACT(Year FROM (Age(Birthday))) >= 18;
```

Знімемо з користувача *Director* привілеї доступу до таблиці *Persons* зі схеми PUBLIC

```
REVOKE SELECT ON Persons FROM Director;
```

Встановимо привілеї доступу до представлення *Persons* зі схеми *Director*

```
GRANT SELECT ON Director.Persons TO Director;
```

Тепер користувач, виконуючи запит

```
SELECT * FROM Persons;
```

отримає лише записи про людей, яким на поточний момент вже виповнилося 18 років.