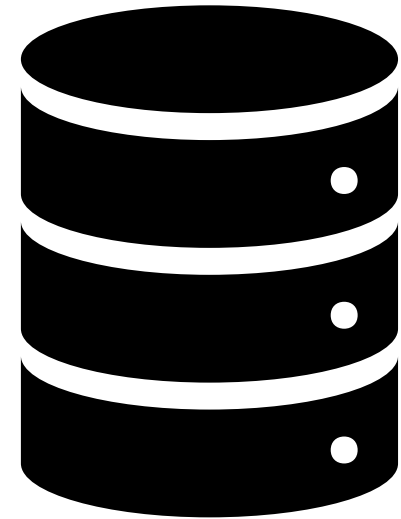


Базы данных

Лекция 7

Дополнительные возможности SQL



Меркурьева Надежда

✉ merkurievanad@gmail.com

✈ @merkurievanad

Data Control Language (DCL)

- Позволяет настраивать доступы к объектам
- Поддерживает 2 типа действий:
 - GRANT – выдача доступа к объекту
 - REVOKE – отмена доступа к объекту
- Права, которые можно выдать на объект:
 - SELECT, INSERT, UPDATE, DELETE, TRUNCATE, REFERENCES, TRIGGER, CREATE, CONNECT, TEMPORARY, EXECUTE, USAGE

DCL: GRANT

```
GRANT { { SELECT | INSERT | UPDATE | DELETE |  
TRUNCATE | REFERENCES | TRIGGER }  
      [, ...] | ALL [ PRIVILEGES ] }  
ON { [ TABLE ] table_name [, ...]  
     | ALL TABLES IN SCHEMA schema_name [,  
... ] }  
   TO role_specification [, ...] [ WITH GRANT  
OPTION ]
```

DCL: REVOKE

```
REVOKE [ GRANT OPTION FOR ]
      { { SELECT | INSERT | UPDATE | DELETE |
TRUNCATE | REFERENCES | TRIGGER }
      [, ...] | ALL [ PRIVILEGES ] }
ON { [ TABLE ] table_name [, ...]
    | ALL TABLES IN SCHEMA schema_name [,
... ] }
FROM { [ GROUP ] role_name | PUBLIC } [,
... ]
[ CASCADE | RESTRICT ]
```

DCL: примеры использования

```
GRANT ALL PRIVILEGES ON kinds TO manuel;
```

```
REVOKE ALL PRIVILEGES ON kinds FROM manuel;
```

```
GRANT SELECT ON kinds TO manuel WITH GRANT  
OPTION;
```

- Теперь manuel может выдавать права на SELECT на табличку kinds
- Если мы хотим забрать grant option у manuel, то нужно использовать CASCADE, чтобы забрать права у всех, кому он выдавал права. Иначе при попытке отозвать права у manuel, запрос упадет с ошибкой

Представление (View)

– это виртуальная (логическая) таблица, представляющая собой поименованный запрос (синоним к запросу), который будет подставлен как подзапрос при использовании представления.

- Не является самостоятельной частью набора данных
- Вычисляется динамически на основании данных, хранящихся в реальных таблицах
- Изменение данных в таблицах немедленно отражается в содержимом представлений

Представление (View)

```
CREATE [OR REPLACE] [TEMP | TEMPORARY]  
VIEW view_name [(column_name [, ...])]  
    AS query;
```

Представление (View)

- **CREATE VIEW** – создание нового представления
- **CREATE OR REPLACE VIEW** – создание или замена уже существующего представления
 - В случае замены в новом представлении должны присутствовать все поля старого представления (имена, порядок, тип данных). Допустимо только добавление новых полей
- **TEMPORARY | TEMP** – временное представление, будет существовать до конца сессии
- *view_name* – название представления
- *column_name* – список полей представления. Если не указан, используются поля запроса
- *query* – **SELECT** или **VALUES** команды

Представление (View)

```
CREATE VIEW v_test  
  AS SELECT 'Hello World';
```

Как делать не надо

```
CREATE VIEW v_test  
  AS SELECT 'Hello World'::text AS hello;
```

Как делать
правильно

Зафиксировали тип

Зафиксировали
название поля

Представление (View)

```
CREATE VIEW comedies AS  
SELECT *  
  FROM films  
  WHERE kind = 'Comedy';
```

Представление (View)

- Горизонтальное представление:

- Ограничение данных по строкам

```
CREATE VIEW V_IT_EMPLOYEE AS
SELECT *
FROM EMPLOYEE
WHERE DEPARTMENT_NM = 'IT';
```

- Вертикальное представление

- Ограничение данных по столбцам

```
CREATE VIEW V_EMP AS
SELECT EMP_NM, DEPARTMENT_NM
FROM EMPLOYEE
WHERE DEPARTMENT_NM = 'IT';
```

EMP_ID	EMP_NM	DEPARTMENT_NM	SALARY_AMT
1	Иванов И.И.	IT	100000
135	Николаев С.Т.	IT	123000
16	Терентьев А.П.	IT	56000

EMP_NM	DEPARTMENT_NM
Иванов И.И.	IT
Степанов Р.В.	R&D
Николаев С.Т.	IT
Медведев И.А.	SALES
Терентьев А.П.	IT

Представление (View)

```
CREATE VIEW name [ ( column_name [, ...] ) ]  
    AS query  
[ WITH [ CASCADED | LOCAL ] CHECK OPTION ]
```

- CHECK OPTION
- LOCAL
- CASCADED

Представление (View)

- Представление называется *обновляемым*, если к нему применимы операции UPDATE и DELETE для изменения данных в таблицах, на которых построено это представление.
- Для того, чтобы представление было обновляемым, должно быть выполнено 2 условия:
 - Соответствие 1-1 между строками представления и таблиц, на которых основано представление
 - Поля представления должны быть простым перечислением полей таблиц

Представление (View)

- Обновляемое представление, основанное на нескольких таблицах, может обновлять только одну таблицу за запрос.
- Как же быть?
 - Явно указывать, значения в каких столбцах вы хотите обновить
 - За одну UPDATE операцию указывать только те столбцы, которые принадлежат одной таблице-источнику
 - DELETE для таких представлений не поддерживается
 - INSERT работает, только если вставка происходит в единственную реальную таблицу

Представления (View)

Для обновляемых
представлений

```
CREATE VIEW name [ ( column_name [, ...] ) ]  
    AS query  
[ WITH [ CASCADED | LOCAL ] CHECK OPTION ]
```

- CHECK OPTION – дополнительная проверка всех UPDATE и INSERT операций
 - CASCADED – проверка целостности этого представления и зависимых представлений
 - LOCAL – проверка целостности только этого представления

VIEW: CASCADED CHECK OPTION

```
CREATE VIEW v_city_a AS
SELECT city_id,
       city,
       country_id
FROM city
WHERE city LIKE 'A%';
```

```
CREATE VIEW v_city_a_usa AS
SELECT city_id,
       city,
       country_id
FROM v_city_a
WHERE country_id = 103
WITH CASCADED CHECK OPTION;
```

При вставке в `v_city_a_usa` будет проверяться только условие из `WHERE`, т.е. `country_id = 103`

`INSERT INTO city_a_usa (city, country_id) VALUES ('Houston', 103);` упадет с ошибкой именно на этапе вставки в `v_city_a`, т.к. не выполняется `city LIKE 'A%'`;

Если заменить `CASCADED CHECK OPTION` на `LOCAL CHECK OPTION`, то вставка отработает!

Изменение представлений

```
ALTER VIEW [IF EXISTS] name ALTER [COLUMN] column_name SET DEFAULT  
expression  
ALTER VIEW [IF EXISTS] name ALTER [COLUMN] column_name DROP DEFAULT  
ALTER VIEW [IF EXISTS] name OWNER TO new_owner  
ALTER VIEW [IF EXISTS] name RENAME TO new_name  
ALTER VIEW [IF EXISTS] name SET SCHEMA new_schema  
ALTER VIEW [IF EXISTS] name SET ( view_option_name [=  
view_option_value] [, ... ] )  
ALTER VIEW [IF EXISTS] name RESET ( view_option_name [, ... ] )  
  
DROP VIEW [IF EXISTS] name [, ...] [ CASCADE | RESTRICT ]
```

Представление (View)

Что может содержать в себе представление?

- Подмножество записей из таблицы БД, отвечающее определённым условиям
- Подмножество столбцов таблицы БД, требуемое программой
- Результат обработки данных таблицы определёнными операциями
- Результат соединения (join) нескольких таблиц
- Результат слияния нескольких таблиц с одинаковыми именами и типами полей, когда в представлении попадают все записи каждой из сливаемых таблиц
- Результат группировки записей в таблице
- Практически любую комбинацию вышеперечисленных возможностей

Представление (View)

Зачем это вообще кому-то нужно?

- Представления скрывают от прикладной программы сложность запросов и саму структуру таблиц БД
- Использование представлений позволяет отделить прикладную схему представления данных от схемы хранения
- С помощью представлений обеспечивается ещё один уровень защиты данных
- Выигрыш во времени за счет оптимизации

Преимущества представлений

- **Безопасность:** можно искусственно ограничивать информацию, к которой у пользователя есть доступ
- **Простота запросов:** при написании запросов обращаемся к вью, как и к обычной таблице
- **Защита от изменений:** пользователю не обязательно знать, что структуры / имена таблиц поменялись. Достаточно обновить представление

Недостатки представлений

- **Производительность:** кажущийся простым запрос с использованием вью на деле может оказаться очень сложным из-за логики, защитой во вью
- **Управляемость:** вью может быть основана на вью, которая в свою очередь тоже основана на другой вью и т.д.
- **Ограничение на обновление:** не любую вью можно обновить, что не всегда очевидно пользователю

CTE (Common Table Expression)

– именованный временный набор данных, используемый в запросе.

```
WITH cte_query_name  
      AS (cte_query)  
main_query;
```

CTE (Common Table Expression)

```
WITH regional_sales AS (  
    SELECT region,  
           sum(amount) AS total_sales  
    FROM orders  
    GROUP BY region  
) , top_regions AS (  
    SELECT region  
    FROM regional_sales  
    WHERE total_sales > (SELECT sum(total_sales)/10  
                          FROM regional_sales))  
SELECT region      AS region,  
       product     AS product,  
       sum(quantity) AS product_units,  
       sum(amount)  AS product_sales  
FROM orders  
WHERE region IN (SELECT region  
                 FROM top_regions)  
GROUP BY region,  
         product;
```

CTE (Common Table Expression)

```
WITH RECURSIVE t(n) AS (  
    VALUES (1)  
    UNION ALL  
    SELECT n+1 FROM t WHERE n < 100  
)  
SELECT sum(n) FROM t;
```

Что делает этот запрос?

Необязательный модификатор RECURSIVE превращает WITH из просто удобной конструкции в уникальную функцию для получения сложных результатов. Используя RECURSIVE, запрос WITH может ссылаться на собственный вывод.

CTE (Common Table Expression)

```
WITH RECURSIVE t(n) AS (  
    VALUES (1)  
    UNION ALL  
    SELECT n+1 FROM t WHERE n < 100  
)  
SELECT sum(n) FROM t;
```

Сначала данные, для
которых рекурсия не
нужна

Через UNION (ALL)
рекурсивная часть

Запрос на получение суммы целых чисел от 1 до 100

CTE (Common Table Expression)

```
WITH RECURSIVE included_parts (sub_part, part, quantity)
  AS (
    SELECT sub_part, part, quantity
      FROM parts
     WHERE part = 'our_product'
    UNION ALL
    SELECT p.sub_part, p.part, p.quantity
      FROM included_parts pr, parts p
     WHERE p.part = pr.sub_part
  )
SELECT sub_part, sum(quantity) as total_quantity
  FROM included_parts
 GROUP BY sub_part
```

CTE (Common Table Expression)

```
WITH moved_rows AS (  
    DELETE FROM products  
        WHERE "date" >= '2010-10-01'  
            AND "date" < '2010-11-01'  
    RETURNING * )  
INSERT INTO products_log  
SELECT *  
    FROM moved_rows;
```

Удаляем данные из
таблички products

Те же самые данные, что
удалили ранее, записываем в
products_log

```
WITH t AS (  
    UPDATE products  
        SET price = price * 1.05  
    RETURNING * )  
SELECT *  
    FROM products;
```

Проиндексировали все
цены в табличке products

Сразу вывели
обновленные данные

Хранимые процедуры

– объект базы данных, представляющий собой набор SQL-инструкций, который компилируется один раз и хранится на сервере

- Похожи на обыкновенные процедуры языков высокого уровня:
 - входные параметры
 - выходные параметры
 - локальные переменные
 - числовые вычисления и операции над символьными данными
- Могут выполняться стандартные операции с базами данных (как DDL, так и DML)
- Возможны циклы и ветвления

Хранимые процедуры

- позволяют повысить производительность
 - расширяют возможности программирования
 - поддерживают функции безопасности данных
-
- Вместо хранения часто запроса, достаточно сослаться на соответствующую хранимую процедуру
-
- Рассматриваем на примере PostgreSQL

Хранимые процедуры

PostgreSQL

```
CREATE [ OR REPLACE ] FUNCTION
name ( [ [ argmode ] [ argname ] argtype [ { DEFAULT | = } default_expr ] [, ...] ] )
[ RETURNS rettype |
  RETURNS TABLE ( column_name column_type [, ...] ) ]
{ LANGUAGE lang_name
  | TRANSFORM { FOR TYPE type_name } [, ... ]
  | WINDOW
  | IMMUTABLE
  | STABLE
  | VOLATILE
  | [ NOT ] LEAKPROOF
  | CALLED ON NULL INPUT
  | RETURNS NULL ON NULL INPUT
  | STRICT
  | [ EXTERNAL ] SECURITY INVOKER
  | [ EXTERNAL ] SECURITY DEFINER
  | COST execution_cost
  | ROWS result_rows
  | SET configuration_parameter { TO value | = value | FROM CURRENT }
  | AS 'definition'
  | AS 'obj_file', 'link_symbol'
} ...
[ WITH ( attribute [, ...] ) ]
```

Хранимые процедуры

```
CREATE FUNCTION add(integer, integer) RETURNS integer  
  AS 'select $1 + $2;'  
  LANGUAGE SQL  
  IMMUTABLE  
  RETURNS NULL ON NULL INPUT;
```

```
SELECT add(20, 22) AS answer;
```

```
answer
```

```
-----
```

```
42
```

Хранимые процедуры

```
CREATE OR REPLACE FUNCTION increment(i integer)
RETURNS integer AS $$
    BEGIN
        RETURN i + 1;
    END;
$$ LANGUAGE plpgsql;
```

```
SELECT increment(41) AS answer;
```

```
answer
```

```
-----
```

```
42
```