PostgreSQL

AIT
Hi-tech team

НАШИ ПРАВИЛА

Включенная камера

Вопросы по поднятой руке

Не перебиваем друг друга

Все вопросы, не связанные с тематикой курса (орг-вопросы и т. д.), должны быть направлены куратору

Подготовьте свое рабочее окружение для возможной демонстрации экрана (закройте лишние соцсети и прочие приложения)

ЦЕЛЬ

Изучить, какие существуют ограничения (constraints), разобрать примеры агрегации данных, изучить группировку

ПЛАН ЗАНЯТИЯ

Установка сервера и Beekeeper Studio

БД и СУБД

Таблица, строки, столбцы, primary key

Нормализация (общее описание)

SQL запросы - практика

Основные операторы

=	равенство
<	меньше
>	больше
<=	меньше или равно
>=	больше или равно
<>	не равно
!=	не равно

Пример использования:

```
SELECT * FROM student WHERE age < 40;
```

Основные операторы

LIKE проверяет соответствует ли значение определенному паттерну (чувствительно к регистру)

ILIKE проверяет соответствует ли значение определенному паттерну(без учета регистра)

AND логическое и

OR логическое или

Пример использования:

```
SELECT * FROM farmer WHERE name LIKE '%s';
```

```
EXPLAIN SELECT * FROM farmer WHERE name LIKE 'Han_';
```

Основные операторы

`IN` проверяет находится ли значение внутури диапазона значений (равнозначно нескольким условиям)

`BETWEEN` проверяет находится ли значение внтури диапазона значений

`IS NULL` проверяет является ли значение NULL-ом

`NOT` в комбинации с другими операторам позволяет формулировать отрицательные условия `NOT LIKE`, `NOT IN`, `NOT BETWEEN`

Пример использования:

```
SELECT * FROM farmer WHERE age IS NULL;
```

Базовые операторы агрегации

COUNT()	расчет количества строк
AVG()	расчет сред/арифметического значения
MAX()	расчет максимального значения
MIN()	расчет минимального значения
SUM()	расчет суммы

Заполним бд, чтобы можно было рассмотреть ряд примеров

Создали таблицу для фермеров.

Добавили в нее значения.

```
CREATE TABLE farmer
 (
   id serial PRIMARY KEY,
   height int,
   age int,
   name varchar,
   number_of_children int,
   country varchar
 );
INSERT INTO
 farmer (name, height, age, number_of_children, country)
VALUES
 ('Johanes', 180, 47, 5, 'Germany'),
 ('Pierre', 175, 40, 2, 'France'),
 ('Gerard', 184, 60, 4, 'Germany'),
 ('Marek', 168, 51, 0, 'Poland'),
 ('Hans', 177, 30, 1, 'Germany');
```

Пример использования оператора <

Получим фермеров моложе 40 лет.

```
SELECT * FROM farmer WHERE age < 40;
```

Результат:

id	height	age	name	number_of_children	country
5	177	30	Hans	1	Germany

Пример использования оператора LIKE

Выберем фермеров, чьи имена заканчиваются на “s”

```
SELECT * FROM farmer WHERE name LIKE '%s';
```

Результат:

id	height	age	name	number_of_children	country
1	180	47	Johanes	5	Germany
5	177	30	Hans	1	Germany

Wild cards

В PostgreSQL существует два подстановочных знака:

- знак процента (%)
- нижнее подчеркивание (_)

Знак процента (%) представляет ноль, один или много символов (или чисел).

Нижнее подчеркивание (_) репрезентирует один единственный знак или символ.

Эти подстановочные знаки можно комбинировать в рамках одного выражения.

Пример использования оператора ILIKE

Выберем фермеров, чьи имена начинаются на “han” без учета регистра

```
SELECT * FROM farmer WHERE name ILIKE 'han_';
```

Результат:

id	height	age	name	number_of_children	country
5	177	30	Hans	1	Germany

Пример использования оператора IN

Получим имена фермеров

```
SELECT name FROM farmer WHERE country IN ('France',
'Poland');
```

Результат:

name
Pierre
Marek

Пример использования оператора BETWEEN

Получим фермеров с указанным ростом

```
SELECT * FROM farmer WHERE height NOT BETWEEN 177
AND 180;
```

Результат:

id	height	age	name	number_of_children	country
2	175	40	Pierre	2	France
3	184	60	Gerard	4	Germany
4	168	51	Marek	0	Poland

Пример использования оператора COUNT()

Получим информацию о том, сколько есть фермеров с указанным ростом

```
SELECT COUNT(*) FROM farmer WHERE height BETWEEN 177
AND 180;
```

Результат:

count
2

Пример использования оператора AVG()

Получим информацию о среднем значении роста AS avarage_height - это псевдоним под которым выводится результат

```
SELECT AVG(height) AS avarage_height FROM farmer;
```

Результат:

avarage_height
176.8000000000000000

Пример использования оператора MIN()

Получим информацию о минимальном возрасте среди фермеров

```
SELECT MIN(age) AS min_age FROM farmer;
```

Результат:

min_age
30

30

GROUP BY

Выводит данные сгруппированные по определенному значению

Получим информацию о среднем возрасте по странам

```
SELECT
 country,
 AVG(height) AS average_height
FROM
 farmer
GROUP BY
 country;
```

Результат:

country	average_height
France	175.0000000000000000
Germany	180.3333333333333333
Poland	168.0000000000000000

168.0000000000000000

GROUP BY

Получим информацию о самых молодых по странам

```
SELECT
 country,
 AVG(height) AS average_height
FROM
 farmer
GROUP BY
 country;
```

Результат:

country	min_age
France	40
Germany	30
Poland	51

GROUP BY

Получим страны у которых фермеров больше двух, с указанием количества

```
SELECT
 country,
 COUNT(id)
FROM
 farmer
GROUP BY
 country
HAVING COUNT(id) >= 2;
```

Результат:

country	count
Germany	3

Ограничения: constraints

SQL позволяет вам определять ограничения для столбцов и таблиц. Ограничения дают вам столько контроля над данными в ваших таблицах, сколько вы пожелаете. Если пользователь пытается сохранить данные в столбце, который нарушает ограничение, возникает ошибка. Это применимо, даже если значение получено из определения значения по умолчанию.

CHECK ограничение

— это наиболее общий тип ограничения. Он позволяет указать, что значение в определенном столбце должно удовлетворять какому-то условию.

Иными словами, с помощью check мы можем проверить удовлетворяет ли значение условию.

В этом примере мы создадим таблицу, где цена может быть только положительной.

```
CREATE TABLE product (
    id serial,
    name text,
    price numeric CHECK (price > 0)
);
```

CHECK ограничение

Из-за того что -15 отрицательное код упадет с ошибкой

```
insert into product (name, price) values
('toy', 10),
('rotten cabbage', -15);
```

CHECK ограничение (с названием)

Ограничению можно задать имя с помощью ключевого слова CONSTRAINT.

Тогда при ошибке будет выводится это название

(не забудьте удалить таблицу product, если она уже создана)

```
CREATE TABLE product (
    id serial,
    name text,
    price numeric CONSTRAINT positive_price
CHECK (price > 0)
);
```

NOT NULL ограничение

(не забудьте удалить таблицу product, если она уже создана)

Создали таблицу продукты.

Добавили ограничения для id и name, чтобы они не могли быть NULL

Попытались создать продукт без указания имени и получили ошибку

```
CREATE TABLE product (
    id serial NOT NULL,
    name text NOT NULL,
    price numeric
);

INSERT INTO product (price) VALUES (20);
```

There was a problem

null value in column "name" of relation "product" violates not-null constraint

UNIQUE ограничение

Ограничение уникальности гарантирует, что данные, содержащиеся в столбце или группе столбцов, уникальны среди всех строк таблицы.

В данном примере мы указали, что title должен быть уникальным.

```
CREATE TABLE dress (
    id serial NOT NULL,
    title text NOT NULL,
    price numeric,
    CONSTRAINT dress_must_be_unique UNIQUE(title)
);
-- здесь будет ошибка
INSERT INTO dress (title, price) VALUES
('little black', 400),
('yellow stripes', 120),
('little black', 180);
```

Click to copy

There was a problem

duplicate key value violates unique constraint "dress_must_be_unique"

PRIMARY KEY ограничение

Ограничение первичного ключа указывает, что столбец или группа столбцов могут использоваться в качестве уникального идентификатора строк в таблице. Для этого необходимо, чтобы значения были уникальными и не были нулевыми. Итак, следующие два определения таблицы принимают одни и те же данные

```
CREATE TABLE products (
    product_no integer PRIMARY KEY,
    name text,
    price numeric
);
```

PRIMARY KEY ограничение

Если несколько столбцов являются первичными ключами - синтаксис будет такой же как в случае с unique:

```
CREATE TABLE example (
    a integer,
    b integer,
    c integer,
    PRIMARY KEY (a, c)
);
```

FOREIGN KEY ограничение

Ограничение внешнего ключа указывает, что значения в столбце (или группе столбцов) должны соответствовать значениям, появляющимся в некоторой строке другой таблицы. Мы говорим, что это поддерживает ссылочную целостность между двумя связанными таблицами.

Допустим, у вас есть таблица продуктов, которую мы уже использовали несколько раз:

```
CREATE TABLE products (
    product_no integer PRIMARY KEY,
    name text,
    price numeric
);
```

FOREIGN KEY ограничение

Предположим также, что у вас есть таблица, в которой хранятся заказы на эти продукты. Мы хотим, чтобы таблица заказов содержала только заказы на действительно существующие товары. Поэтому мы определяем ограничение внешнего ключа в таблице заказов, которое ссылается на таблицу продуктов:

```
CREATE TABLE orders (
    order_id integer PRIMARY KEY,
    product_no integer REFERENCES products (product_no),
    quantity integer
);
```

FOREIGN KEY ограничение

Теперь невозможно создавать заказы с non-NULL product_no, которого бы не было ранее в таблице продуктов.

Мы говорим, что в этой ситуации таблица заказов является ссылаюшейся таблицей, а таблица продуктов — та на которую ссылаются. Аналогичным образом существуют ссылающиеся и ссылочные столбцы.

ON DELETE CASCADE

— это опция, которую можно использовать при определении ограничения внешнего ключа.

Если указано ON DELETE CASCADE, Postgres автоматически удаляет любую строку в дочерней таблице, которая ссылается на удаленную строку из родительской таблицы.

```
CREATE TABLE products (
    product_no integer PRIMARY KEY,
    name text,
    price numeric
);

CREATE TABLE orders (
    order_id integer PRIMARY KEY,
    shipping_address text,
    ...
);

CREATE TABLE order_items (
    product_no integer REFERENCES products ON DELETE RESTRICT,
    order_id integer REFERENCES orders ON DELETE CASCADE,
    quantity integer,
    PRIMARY KEY (product_no, order_id)
);
```

ON DELETE CASCADE

```
CREATE TABLE
 truck (
   id serial PRIMARY KEY,
   farmer_id integer REFERENCES farmer (id) ON DELETE CASCADE,
   brand varchar
 );

INSERT INTO
 truck (farmer_id, brand)
values
 (2, 'WildCat'),
 (3, 'Mercedes'),
 (3, 'Belarus');
 DELETE FROM farmer WHERE id = 3;
-- удалили фермера, его трактора удалились вместе с ним
```

Index

— это отдельная структура данных, которая ускоряет извлечение данных из таблицы за счет дополнительных операций записи и хранения для ее обслуживания.

Используйте оператор CREATE INDEX для создания индекса.

```
CREATE INDEX farmer_index ON farmer (height);
```

Лучший способ углубить свои знания PostgreSQL – это изучение документации.

Ссылка на документацию: https://www.postgresql.org/docs/current/index.html

Ваша новая IT-профессия – Ваш новый уровень жизни

Программирование с нуля в
немецкой школе AIT TR GmbH

AIT
Hi-tech team