НЕТОЛОГИЯ
групп

ЗАНЯТИЕ 1.4

Углубление в SQL

Алексей Кузьмин

Директор разработки; Data Scientist

ДомКлик.ру

aleksej.kyzmin@gmail.com

О ЧЁМ ПОГОВОРИМ И ЧТО СДЕЛАЕМ

- Работа с таблицами
- Работа с данными
- Внешние ключи
- Сложные типы данных

Таблицы

Создание таблиц

Для создания новой таблицы в PostgreSQL Вы можете использовать команду *CREATE TABLE*:

```
CREATE TABLE table_name (
 column_name TYPE column_constraint,
 table_constraint table_constraint
);
```

- table_name - имя таблицы
- column_name - имя колонки
- TYPE - тип колонки
- column constraint - ограничение колонки
- table_constraint - ограничение таблицы

Ограничения колонок:

- NOT NULL
- UNIQUE – каждое значение (кроме NULL) должно быть уникально.
- PRIMARY KEY - комбинация NOT NULL + UNIQUE. На таблицу разрешен только один PRIMARY KEY

Ограничения таблиц:

- UNIQUE (column_list)– уникальность на группу колонок.
- PRIMARY KEY(column_list) – первичный ключ по группе колонок.

Пример создания таблицы с пользователями:

```
CREATE TABLE account(
 user_id serial PRIMARY KEY,
 username VARCHAR (50) UNIQUE NOT NULL,
 password VARCHAR (50) NOT NULL,
 email VARCHAR (355) UNIQUE NOT NULL,
 created_on TIMESTAMP NOT NULL,
 last_login TIMESTAMP
 );
```

Изменение таблицы

Используется команда *ALTER TABLE*:

ALTER TABLE table_name action;

с ее помощью можно:

- добавить, удалить, переименовать или изменить тип у колонки
- установить значение по умолчанию
- переименовать таблицу

ALTER TABLE table_name ADD COLUMN new_column_name **TYPE**;

ALTER TABLE table_name DROP COLUMN column_name;

ALTER TABLE table_name RENAME COLUMN column_name TO new_column_name;

ALTER TABLE table_name ALTER COLUMN column_name [**SET DEFAULT** value | **DROP DEFAULT**]

ALTER TABLE table_name ALTER COLUMN column_name [**SET NOT NULL**| **DROP NOT NULL**]

ALTER TABLE table_name ADD CONSTRAINT constraint_name constraint_definition

ALTER TABLE table_name RENAME TO new_table_name;

Удаление таблицы

Команда - *DROP TABLE*

DROP TABLE [**IF EXISTS**] **table_name** [**CASCADE** | **RESTRICT**];

RESTRICT/CASCADE - если какие-то таблицы ссылаются на эту, то режим RESTRICT не даст удалить таблицу. Режим CASCADE - каскадно удалить зависимости.

ВРЕМЯ ПРАКТИКИ

Практика 1

Создайте таблицу "автор" с полями:
- id
- full name
- псевдоним (может не быть)
- дата рождения

Данные

Вставка данных

Пример:

```
INSERT INTO table(column1, column2, ...)
VALUES
 (value1, value2, ...);
```

- table - название таблицы
- column1, ... - названия колонок
- value1, ... - значения для вставки

```
CREATE TABLE link (
 ID serial PRIMARY KEY,
 url VARCHAR (255) NOT NULL,
 name VARCHAR (255) NOT NULL,
 description VARCHAR (255),
 rel VARCHAR (50)
);
INSERT INTO link (url, name)
VALUES
   ('http://www.google.com','Google');
```

Можно вставлять несколько строк одновременно:

```
INSERT INTO table (column1, column2, ...)
VALUES
 (value1, value2, ...),
 (value1, value2, ...) ,...;
```

```
INSERT INTO link (url, name)
VALUES
 ('http://www.yahoo.com','Yahoo'),
   ('http://www.bing.com','Bing');
```

Модификация данных

Команда:

```
UPDATE table
SET column1 = value1,
    column2 = value2 ,...
WHERE
 condition;
```

Пример:

```
UPDATE link
SET description = 'no description'
WHERE
   description IS NULL;
```

Обновление всех строк:

```
UPDATE link
SET rel = 'nofollow';
```

Все ряды основываясь на другой колонке:

```
UPDATE link
  SET description = name;
```

Удаление данных

Команда:

```
DELETE FROM table
WHERE condition;
```

Пример:

```
DELETE FROM link
WHERE id = 1;
```

ВРЕМЯ ПРАКТИКИ

Практика 2

- Вставьте данные по 3-м любым писателям в указанную таблицу
- Добавьте поле "место рождения" в таблицу
- Обновите данные, проставив корректное место рождения писателю

надо использовать только sql команды

Внешние ключи

Внешний ключ - это поле, которое ссылается на строку в другой таблице. Таблица, содержащая внешний ключ обычно называется "дочерней", а таблица на которую указывают - "родительской".

Таблица может содержать несколько внешних ключей.

таблица адресов доставки:

```
CREATE TABLE so_headers (
 id SERIAL PRIMARY KEY,
 customer_id INTEGER,
 ship_to VARCHAR (255)
 );
```

заказы:

```
CREATE TABLE so_items (
    item_id INTEGER NOT NULL,
    so_id INTEGER,
    product_id INTEGER,
    qty INTEGER,
    net_price NUMBER,
    PRIMARY KEY (item_id,so_id)
);
```

Если мы хотим показать, что so_id ссылается на строку в so_headers (и требует, чтобы она была, то можно передать скрипт создания таблицы:

```
CREATE TABLE so_items (
  item_id INTEGER NOT NULL,
  so_id INTEGER REFERENCES so_headers(id),
  product_id INTEGER,
  qty INTEGER,
  net_price numeric,
  PRIMARY KEY (item_id,so_id)
);
```

Другой способ записи:

```
CREATE TABLE so_items (
 item_id INTEGER NOT NULL,
 so_id INTEGER,
 product_id INTEGER,
 qty INTEGER,
 net_price NUMERIC,
 PRIMARY KEY (item_id, so_id),
 FOREIGN KEY (so_id) REFERENCES so_headers (id)
);
```

PostgreSQL создает констраинт, который будет проверять наличие строки в so_headers при любом изменении таблицы so_items. Кроме того, из so_headers удалять можно будет только строки, на которые никто не ссылается из so_items,

Проверяющие ограничения

CHECK ограничение позволяет проверять допустимое множество значений для поля. Пример:

```
CREATE TABLE employees (
 id serial PRIMARY KEY,
 first_name VARCHAR (50),
 last_name VARCHAR (50),
 birth_date DATE CHECK (birth_date > '1900-01-01'),
 joined_date DATE CHECK (joined_date > birth_date),
 salary numeric CHECK(salary > 0)
);
```

ВРЕМЯ ПРАКТИКИ

Практика 3

- Создайте таблицу "Произведения" с полями: год, название, ссылка на автора. Установите foreign key contsraint
- Вставьте в таблицу пару значений
- Попробуйте удалить автора

Сложные типы данных

JSON

JSON - JavaScript Object Notation. JSON - де-факто стандарт для хранения данных в виде key-value пар.

Рассмотрим простой пример:

```
CREATE TABLE orders (
 ID serial NOT NULL PRIMARY KEY,
 info json NOT NULL
);
```

```
INSERT INTO orders (info)
VALUES
 (
 '{ "customer": "John Doe", "items": {"product": "Beer","qty": 6}}'
 ),
 (
 '{ "customer": "Lily Bush", "items": {"product": "Diaper","qty": 24}}'
 ),
 (
 '{ "customer": "Josh William", "items": {"product": "Toy Car","qty": 1}}'
 ),
 (
 '{ "customer": "Mary Clark", "items": {"product": "Toy Train","qty": 2}}'
   );
```

Запрос данных:

```
SELECT
 info
FROM
 orders;
```

Postgres возвращает ответ в виде типа JSON. Для работы с ним есть 2 специальных оператора -> и ->>

-> возвращает результат в виде JSON-объекта ->> возвращает результат в виде текста

Получить имена всех покупателей:

```
SELECT
 info ->> 'customer' AS customer
FROM
 orders;
```

т.к. оператор -> возвращает JSON-объект, то к его результату можно снова применять оператор -> и ->>. Пример:

```
SELECT
 info -> 'items' ->> 'product' as product
FROM
 orders
ORDER BY
   product;
```

Еще пример. Поиск людей, которые купили 2 продукта:

```
SELECT
 info ->> 'customer' AS customer,
 info -> 'items' ->> 'product' AS product
FROM
 orders
WHERE
 CAST (
 info -> 'items' ->> 'qty' AS INTEGER
  ) = 2
```

ВРЕМЯ ПРАКТИКИ

Практика 4

Создайте таблицу orders (скрипт выше в лекции). Выведите общее количество заказов

Массивы

Массив - это коллекция элементов. В одной колонке Вы можете хранить несколько атрибутов одного типа. Пример:

```
CREATE TABLE contacts (
 id serial PRIMARY KEY,
 name VARCHAR (100),
 phones TEXT []
);
```

Вставка данных:

```
INSERT INTO contacts (name, phones)
VALUES
(
'John Doe',
ARRAY [ '(408)-589-5846',
'(408)-589-5555' ]
);
```

или

```
INSERT INTO contacts (name, phones)
VALUES
(
'John Doe',
'{ "(408)-589-5846", "(408)-589-5555" }'
);
```

Выборка данных:

```
SELECT
 name,
 phones
FROM
 contacts;
```

Запрос конкретного элемента массива:

```
SELECT
 name,
 phones [1]
FROM
 contacts;
```

Индексы начинаются с 1.

Модификация данных

Обновление элемента

```
UPDATE contacts
SET phones [ 2 ] = '(408)-589-5843'
WHERE
 ID = 3;
```

Обновление всего массива

```
UPDATE contacts
SET phones = '{"(408)-589-5843"}'
WHERE
ID = 3;
```

Поиск элемента

```
SELECT
 name,
 phones
FROM
 contacts
WHERE
 '(408)-589-5555' = ANY (phones);
```

Expand

unnest - превращает массив в набор строк

```
SELECT
 name,
 unnest(phones)
FROM
 contacts;
```

ВРЕМЯ ПРАКТИКИ

Практика 5

Выведите сколько раз встречается специлаьный атрибут (special_feature) у фильма

ИТОГИ ЗАНЯТИЯ

Мы сегодня изучили:

- Создание и модификация таблиц
- Вставка и модификация данных
- Constraints
- Сложные типы данных

ВОПРОСЫ

Домашнее задание

Домашнее задание

Спроектируйте базу данных для следующих сущностей:

- Язык (в смысле английский, французский и тп)
- Народность (в смысле славяне, англосаксы и тп)
- Страны (в смысле Россия, Германия и тп)

Правила следующие:

- На одном языке может говорить несколько народностей
- Одна народность может входить в несколько стран
- Каждая страна может состоять из нескольких народностей

Пришлите скрипты создания таблиц и заполнения их 5-ю строками данных

НЕТОЛОГИЯ
групп

Спасибо за внимание!

Алексей Кузьмин

aleksej.kyzmin@gmail.com