Язык SQL

Лекция 1
Введение в язык SQL

Е. П. Моргунов
Сибирский государственный университет науки и технологий
имени академика М. Ф. Решетнева
г. Красноярск
Институт информатики и телекоммуникаций
emorgunov@mail.ru

Компания Postgres Professional
г. Москва

Версия от 18.10.2018

Подготовка к работе

На вашем компьютере уже должна быть развернута база данных demo.

- Войдите в систему как пользователь postgres:

```
su - postgres
```

- Должен быть запущен сервер баз данных PostgreSQL.

```
pg_ctl  start  -D /usr/local/pgsql/data
```

- Для проверки запуска сервера выполните команду

```
pg_ctl  status  -D /usr/local/pgsql/data
```

- или

```
ps -ax | grep postgres | grep -v grep
```

- Запустите утилиту psql и подключитесь к базе данных demo

```
psql -d demo -U postgres
```
(для ОС Debian)

```
psql -d demo
```
(для ОС Xubuntu)

Создание таблиц

Упрощенный синтаксис таков:

```
CREATE TABLE "имя_таблицы"
( имя_поля тип_данных [ограничения_целостности],
  имя_поля тип_данных [ограничения_целостности],
  ...
  имя_поля тип_данных [ограничения_целостности],
  [ограничение_целостности],
  [первичный_ключ],
  [внешний_ключ]
);
```

Для получения справки о синтаксисе SQL-команды:

```
\h CREATE TABLE
```

Таблица «Самолеты»

Описание атрибута	Имя атрибута	Тип данных	Тип PostgreSQL	Ограничения
Код самолета, IATA	aircraft_code	Символьный	char(3)	NOT NULL
Модель самолета	model	Символьный	text	NOT NULL
Максимальная дальность полета, км	range	Числовой	integer	NOT NULL range > 0

```
CREATE TABLE aircrafts
( aircraft_code char( 3 ) NOT NULL,
  model text NOT NULL,
  range integer NOT NULL,
  CHECK ( range > 0 ),
  PRIMARY KEY ( aircraft_code )
);
```

Способы ввода команд

- Способ 1

```
demo=# CREATE TABLE aircrafts ( aircraft_code char( 3 ) NOT NULL,
model text NOT NULL, range integer NOT NULL, CHECK ( range > 0 ),
PRIMARY KEY ( aircraft_code ) );
```

- Способ 2

```
demo=# CREATE TABLE aircrafts
demo-# ( aircraft_code char( 3 ) NOT NULL,
demo(# model text NOT NULL,
demo(# range integer NOT NULL,
demo(# CHECK ( range > 0 ),
demo(# PRIMARY KEY ( aircraft_code )
demo(# );
CREATE TABLE
```

Вместо ввода символа «;» команду можно завершить символами «\g»:

```
demo=# CREATE TABLE aircrafts ... \g
```

Прервать ввод команды можно клавишами Ctrl-C:

```
demo=# CREATE TABLE aircrafts
( aircraft_code char( 3 ) NOT NULL,
demo(# ^C
demo=#
```

Получение описания таблицы

```
\d aircrafts
Таблица "public.aircrafts"
     Колонка     |     Тип       | Модификаторы
-----------------+---------------+--------------
 aircraft_code   | character(3)  | NOT NULL
 model           | text          | NOT NULL
 range           | integer       | NOT NULL
Индексы:
 "aircrafts_pkey" PRIMARY KEY, btree (aircraft_code)
Ограничения-проверки:
 "aircrafts_range_check" CHECK (range > 0)
```

- public означает имя так называемой **схемы**.
- Для реализации **первичного ключа** (PRIMARY KEY) всегда автоматически создается **индекс**. В данном случае тип индекса — btree, т. е. B-дерево.
- Можно задать свои собственные имена для всех ограничений.

Удаление таблицы

Упрощенный синтаксис:

```
DROP TABLE имя_таблицы;
```

Например:

```
DROP TABLE aircrafts;
```

Вставка строк в таблицу (1)

Упрощенный синтаксис:

```
INSERT INTO имя_таблицы
            [( имя_атрибута, имя_атрибута, ... )]
VALUES ( значение_атрибута, значение_атрибута, ... );
```

- В начале команды перечисляются атрибуты таблицы. При этом можно указывать их не в том порядке, в котором они были указаны при ее создании.
- Если вы не привели список атрибутов, тогда вы обязаны в предложении VALUES задавать значения атрибутов с учетом того порядка, в котором они следуют в определении таблицы.

Вставка строк в таблицу (2)

```
INSERT INTO aircrafts ( aircraft_code, model, range )
VALUES ( 'SU9', 'Sukhoi SuperJet-100', 3000 );
```

В ответ мы получим сообщение об успешном добавлении этой строки:

```
INSERT 0 1
```

количество добавленных строк

имеет отношение к внутреннему устройству PostgreSQL

Выборка строк из таблицы

Синтаксис, упрощенный до предела, таков:

```
SELECT имя_атрибута, имя_атрибута, ...
FROM имя_таблицы;
```

Часто бывает так, что требуется вывести значения из всех столбцов таблицы. В таком случае можно не перечислять имена атрибутов, а просто ввести символ «*». Давайте выберем всю информацию из таблицы aircrafts:

```
SELECT * FROM aircrafts;
```

СУБД ответит таким образом:

```
 aircraft_code |          model          | range
---------------+-------------------------+-------
 SU9           | Sukhoi SuperJet-100     | 3000
(1 строка)
```

Вставка нескольких строк в таблицу

```
INSERT INTO aircrafts ( aircraft_code, model, range )
VALUES ( '773', 'Boeing 777-300', 11100 ),
       ( '763', 'Boeing 767-300', 7900 ),
       ( '733', 'Boeing 737-300', 4200 ),
       ( '320', 'Airbus A320-200', 5700 ),
       ( '321', 'Airbus A321-200', 5600 ),
       ( '319', 'Airbus A319-100', 6700 ),
       ( 'CN1', 'Cessna 208 Caravan', 1200 ),
       ( 'CR2', 'Bombardier CRJ-200', 2700 );
```

СУБД сообщит об успешном вводе 8 строк в таблицу aircrafts:

```
INSERT 0 8
```

Что получилось в таблице?

```
SELECT * FROM aircrafts;
```

Теперь в ней уже 9 строк.

```
 aircraft_code |          model          | range
---------------+-------------------------+-------
 SU9           | Sukhoi SuperJet-100     |  3000
 773           | Boeing 777-300          | 11100
 763           | Boeing 767-300          |  7900
 733           | Boeing 737-300          |  4200
 320           | Airbus A320-200         |  5700
 321           | Airbus A321-200         |  5600
 319           | Airbus A319-100         |  6700
 CN1           | Cessna 208 Caravan      |  1200
 CR2           | Bombardier CRJ-200      |  2700
(9 строк)
```

ВАЖНО! При выполнении простой выборки из таблицы СУБД не гарантирует *никакого конкретного порядка* вывода строк.

Упорядочивание строк в выборке

Упорядочим строки по значению атрибута model, а заодно изменим порядок расположения столбцов в выводе информации.

```
SELECT model, aircraft_code, range
FROM aircrafts
ORDER BY model;
```

```
         model          | aircraft_code | range
------------------------+---------------+-------
 Airbus A319-100        | 319           |  6700
 Airbus A320-200        | 320           |  5700
 Airbus A321-200        | 321           |  5600
 Boeing 737-300         | 733           |  4200
 Boeing 767-300         | 763           |  7900
 Boeing 777-300         | 773           | 11100
 Bombardier CRJ-200     | CR2           |  2700
 Cessna 208 Caravan     | CN1           |  1200
 Sukhoi SuperJet-100    | SU9           |  3000
(9 строк)
```

Ограничение множества строк

Выберем модели самолетов, у которых максимальная дальность полета находится в пределах от 4 до 6 тысяч км включительно.

```
SELECT model, aircraft_code, range
FROM aircrafts
WHERE range >= 4000 AND range <= 6000;
```

```
      model       | aircraft_code | range
------------------+---------------+-------
 Boeing 737-300   | 733           | 4200
 Airbus A320-200  | 320           | 5700
 Airbus A321-200  | 321           | 5600
(3 строки)
```

Обновление строк в таблице (1)

Команда UPDATE предназначена для обновления данных в таблицах.

Ее упрощенный синтаксис таков:

```
UPDATE имя_таблицы
SET имя_атрибута1 = значение_атрибута1,
    имя_атрибута2 = значение_атрибута2, ...
WHERE условие;
```

ВАЖНО! Если это условие не задать, то будут обновлены ВСЕ строки в таблице.

Обновление строк в таблице (2)

Предположим, что дальность полета самолета Sukhoi SuperJet стала на 500 км больше.

```
UPDATE aircrafts SET range = 3500
WHERE aircraft_code = 'SU9';
```

СУБД выведет сообщение, подтверждающее успешное обновление одной строки:

```
UPDATE 1
```

```
SELECT * FROM aircrafts WHERE aircraft_code = 'SU9';
```

```
 aircraft_code |         model         | range
---------------+-----------------------+-------
 SU9           | Sukhoi SuperJet-100 | 3500
(1 строка)
```

Удаление строк из таблицы

Для этого используется команда DELETE. Она походит на команду SELECT:

```
DELETE FROM имя_таблицы WHERE условие;
```

Удалим одну строку из таблицы «Самолеты» (aircrafts):

```
DELETE FROM aircrafts WHERE aircraft_code = 'CN1';
```

СУБД сообщит об успешном удалении одной строки:

```
DELETE 1
```

Можно указать и какое-нибудь более сложное условие. Например, удалим информацию о самолетах с дальностью полета более 10 000 км, а также с дальностью полета менее 3000 км:

```
DELETE FROM aircrafts WHERE range > 10000 OR range < 3000;
```

При необходимости удаления ВСЕХ строк из таблицы, команда будет совсем простой:

```
DELETE FROM aircrafts;
```

Восстановление данных в таблице

Теперь в таблице «Самолеты» (aircrafts) нет ни одной строки. Для продолжения работы необходимо эти данные восстановить. Можно использовать несколько способов.

1. Ввести заново команды INSERT из текста пособия, которые вы ранее уже вводили.

2. Используя клавиши «стрелка вверх» и «стрелка вниз», найти команды INSERT в списке истории команд и повторно их выполнить.

3. С помощью специальной команды, предусмотренной в утилите psql, сохранить всю историю выполненных вами команд в текстовом файле:

```
\s имя_файла_для_сохранения_истории_команд
```

Затем нужно открыть его в текстовом редакторе, найти в файле нужные вам команды INSERT и, копируя команды в буфер обмена, вставить их в командную строку утилиты psql и выполнить.

Таблица «Места» (1)

Описание атрибута	Имя атрибута	Тип данных	Тип PostgreSQL	Ограничения
Код самолета, IATA	aircraft_code	Символьный	char(3)	NOT NULL
Номер места	seat_no	Символьный	varchar(4)	NOT NULL
Класс обслуживания	fare_conditions	Символьный	varchar(10)	NOT NULL Значения из списка: Economy, Comfort, Business

Таблица «Места» (2)

```
CREATE TABLE seats
( aircraft_code  char( 3 ) NOT NULL,
  seat_no  varchar( 4 ) NOT NULL,
  fare_conditions varchar( 10 ) NOT NULL,
  CHECK ( fare_conditions IN ( 'Economy', 'Comfort',
                               'Business' ) ),
  PRIMARY KEY ( aircraft_code, seat_no ),
  FOREIGN KEY ( aircraft_code )
  REFERENCES aircrafts (aircraft_code )
    ON DELETE CASCADE
);
```

- Первичный ключ – **составной:** комбинация атрибутов «Код самолета, IATA» и «Номер места». Этот первичный ключ будет **естественным**.
- Предложение FOREIGN KEY создает ограничение ссылочной целостности – **внешний ключ**. В качестве внешнего ключа служит атрибут «Код самолета» (aircraft_code). Он ссылается на одноименный атрибут в таблице «Самолеты» (aircrafts).
- Таблица «Места» называется ссылающейся (referencing), а таблица «Самолеты» – ссылочной (referenced).
- Каскадное удаление – **ON DELETE CASCADE.**

Что получилось?

\d seats

```
Таблица "public.seats"
     Колонка      |           Тип            | Модификаторы
------------------+--------------------------+--------------
 aircraft_code    | character(3)             | NOT NULL
 seat_no          | character varying(4)     | NOT NULL
 fare_conditions | character varying(10)  | NOT NULL
Индексы:
"seats_pkey" PRIMARY KEY, btree (aircraft_code, seat_no)
Ограничения-проверки:
"seats_fare_conditions_check" CHECK (fare_conditions::text
= ANY (ARRAY['Economy'::character varying,
'Comfort'::character varying,
 'Business'::character varying]::text[]))
Ограничения внешнего ключа:
"seats_aircraft_code_fkey" FOREIGN KEY (aircraft_code)
REFERENCES aircrafts(aircraft_code) ON DELETE CASCADE
```

Какие таблицы есть в базе данных?

```
\d

Список отношений
  Схема |     Имя     |   Тип    | Владелец
--------+-------------+----------+----------
 public | aircrafts | таблица | postgres
 public | seats     | таблица | postgres
(2 строки)
```

Как работает внешний ключ

Выполните следующую команду для ввода данных в таблицу «Места»:

```
INSERT INTO seats VALUES ( '123', '1A', 'Business' );
```

СУБД ответит так:

```
ОШИБКА: INSERT или UPDATE в таблице "seats" нарушает
ограничение внешнего ключа "seats_aircraft_code_fkey"
ПОДРОБНОСТИ: Ключ (aircraft_code)=(123) отсутствует в
таблице "aircrafts".
```

Это совершенно логично: если в таблице «Самолеты», на которую ссылается таблица «Места», нет описания самолета с кодом самолета, равным «123», то добавлять информацию о номерах кресел для такого — несуществующего — самолета не имеет смысла. Так действует поддержка правил ссылочной целостности со стороны СУБД.

Заполнение данными таблицы «Места»

Для каждой модели самолетов введите только несколько строк, при этом предусмотрите записи для классов обслуживания «Business» и «Economy». С помощью одной команды INSERT можно ввести сразу несколько строк:

```
INSERT INTO seats
VALUES ( 'SU9','1A', 'Business' ),
       ( 'SU9', '1B', 'Business' ),
       ( 'SU9', '10A', 'Economy' ),
       ( 'SU9', '10B', 'Economy' ),
       ( 'SU9', '10F', 'Economy' ),
       ( 'SU9', '20F', 'Economy' );
```

Затем измените значение атрибута aircraft_code на другое, например, «773», и повторите команду INSERT. Так придется поступить со всеми моделями самолетов.

Группирование строк

Предположим, что нам нужно получить информацию о количестве мест в салонах для всех типов самолетов.

Нерациональное решение:

```
SELECT count( * ) FROM seats WHERE aircraft_code = 'SU9';
SELECT count( * ) FROM seats WHERE aircraft_code = 'CN1';
... ... ... ...
```

Рациональное решение:

```
SELECT aircraft_code, count( * ) FROM seats
GROUP BY aircraft_code;
```

```
 aircraft_code | count
---------------+-------
 773           | 402
 733           | 130
 CN1           |  12
 CR2           |  50
 319           | 116
 SU9           |  97
 321           | 170
 763           | 222
 320           | 140
(9 строк)
```

Сортировка полученной выборки

Если мы захотим отсортировать выборку по числу мест в самолетах, то нужно будет дополнить команду предложением ORDER BY:

```
SELECT aircraft_code, count( * ) FROM seats
GROUP BY aircraft_code
ORDER BY count;
```

```
 aircraft_code | count
---------------+-------
 CN1           |   12
 CR2           |   50
 SU9           |   97
 319           |  116
 733           |  130
 320           |  140
 321           |  170
 763           |  222
 773           |  402
(9 строк)
```

Более сложная задача

Подсчитать количество мест в салонах для всех моделей самолетов, но теперь уже с учетом класса обслуживания (бизнес-класс и экономический класс). В этом случае группировка выполняется уже по двум атрибутам: aircraft_code и fare_conditions.

```
SELECT aircraft_code, fare_conditions, count( * )
FROM seats
GROUP BY aircraft_code, fare_conditions
ORDER BY aircraft_code, fare_conditions;
```

```
 aircraft_code | fare_conditions | count
---------------+-----------------+-------
 319           | Business        |    20
 319           | Economy         |    96
 320           | Business        |    20
 320           | Economy         |   120
 ...
(17 строк)
```

Литература

1. Лузанов, П. PostgreSQL для начинающих / П. Лузанов, Е. Рогов, И. Лёвшин ; Postgres Professional. – М., 2017. – 146 с.
2. Моргунов, Е. П. Язык SQL. Базовый курс : учеб.-практ. пособие. / Е. П. Моргунов ; под ред. Е. В. Рогова, П. В. Лузанова ; Postgres Professional. – М., 2017. – 257 с.
3. PostgreSQL [Электронный ресурс] : официальный сайт / The PostgreSQL Global Development Group. – http://www.postgresql.org.
4. Postgres Professional [Электронный ресурс] : российский производитель СУБД Postgres Pro : официальный сайт / Postgres Professional. – http://postgrespro.ru.

Задание

Для выполнения практических заданий необходимо использовать книгу:

Моргунов, Е. П. Язык SQL. Базовый курс : учеб.-практ. пособие / Под ред. Е. В. Рогова, П. В. Лузанова ; Postgres Professional. – М., 2017. – 257 с.
https://postgrespro.ru/education/books/sqlprimer

1. Прочитать введение и главу 1.
2. Установить ОС Linux (Debian или другой). Указания по установке СУБД PostgreSQL приведены в главе 2, параграф 2.1. Можно воспользоваться виртуальной машиной VirtualBox или аналогичной. Можно использовать уже настроенную ОС Debian (в виде виртуальной машины), полученную у преподавателя.
3. Развернуть учебную базу данных «Авиаперевозки» https://postgrespro.ru/education/demodb (см. главу 2, параграф 2.3). Использовать версию БД от 13.10.2016.
4. Изучить материал главы 3. Запросы к базе данных выполнять с помощью утилиты psql, описанной в главе 2, параграф 2.2.