Язык SQL

Лекция 5
Изменение данных

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 (1)

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
```

- Если у вас база данных demo была модифицирована, то для ее восстановления выполните команду

```
psql -f demo_small.sql -U postgres
```
(для ОС Debian)

```
psql -f demo_small.sql
```
(для ОС Xubuntu)

Подготовка к работе (2)

- Запустите утилиту psql и подключитесь к базе данных demo

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

```
psql -d demo
```
(для ОС Xubuntu)

- Назначьте схему bookings в качестве текущей

```
demo=# set search_path = bookings;
```

5.1. Вставка строк в таблицы

Подготовительные операции (1)

- Создадим две копии таблицы «Самолеты» (aircrafts).
- Это будут *временные* таблицы, которые автоматически удаляются при завершении работы.
- Первая будет предназначена для хранения данных, взятых из таблицы-прототипа, а вторая будет использоваться в качестве журнальной таблицы.

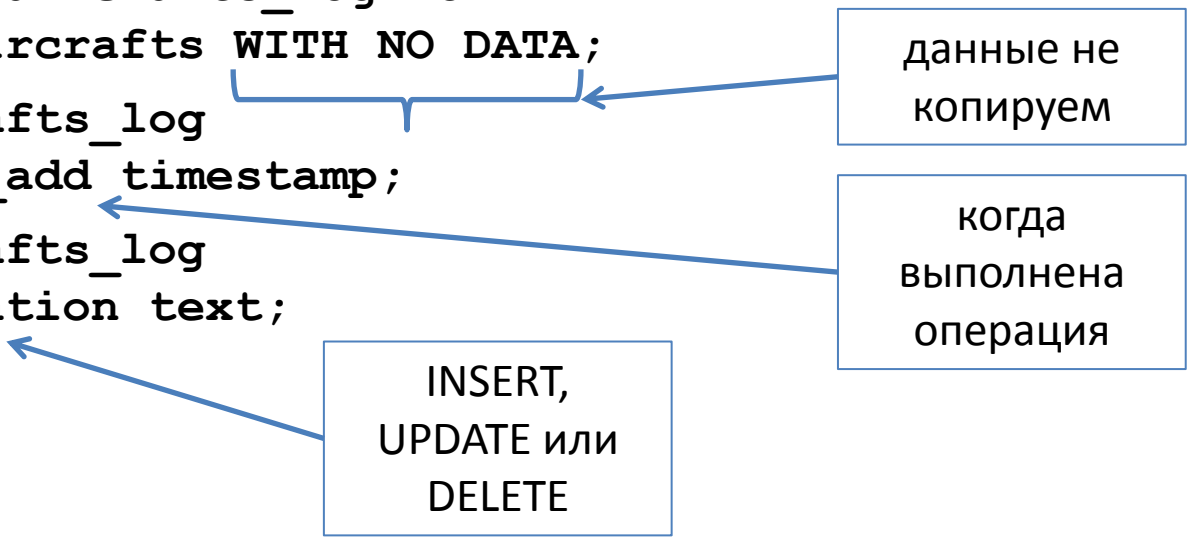
```
CREATE TEMP TABLE aircrafts_tmp AS
  SELECT * FROM aircrafts WITH NO DATA;
ALTER TABLE aircrafts_tmp
  ADD PRIMARY KEY ( aircraft_code );
ALTER TABLE aircrafts_tmp
  ADD UNIQUE ( model );
```

данные не копируем

Ограничения не создаются при копировании таблицы

Подготовительные операции (2)

```
CREATE TEMP TABLE aircrafts_log AS
  SELECT * FROM aircrafts WITH NO DATA;
ALTER TABLE aircrafts_log
  ADD COLUMN when_add timestamp;
ALTER TABLE aircrafts_log
  ADD COLUMN operation text;
```



- Альтернативный способ:

```
CREATE TEMP TABLE aircrafts_tmp
  ( LIKE aircrafts INCLUDING CONSTRAINTS INCLUDING INDEXES
);
```

Команда INSERT с общим табличным выражением

Задача: скопировать в таблицу aircrafts_tmp все данные из таблицы aircrafts, фиксируя все изменения в журнале изменений.

ВАЖНО! При классическом подходе для учета изменений, внесенных в таблицы, используют *триггеры*. Наш пример – *иллюстрация* возможностей общих табличных выражений (CTE), а не единственно верный подход.

```
WITH add_row AS
( INSERT INTO aircrafts_tmp
  SELECT * FROM aircrafts
  RETURNING *
)
INSERT INTO aircrafts_log
SELECT add_row.aircraft_code, add_row.model,
       add_row.range, CURRENT_TIMESTAMP, 'INSERT'
FROM add_row;
INSERT 0 9
```

возвращает внешнему запросу все строки, успешно добавленные в таблицу

* – будут возвращены все столбцы

заполнение журнала

временная таблица

значения для дополнительных столбцов

Что получилось в основной таблице?

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

А что получилось в журнальной таблице?

```
SELECT * FROM aircrafts_log ORDER BY model;
--[ RECORD 1 ]-+--------------------------
 aircraft_code | 319
 model         | Airbus A319-100
 range         | 6700
 when_add      | 2017-01-31 18:28:49.230179
 operation     | INSERT
--[ RECORD 2 ]-+--------------------------
 aircraft_code | 320
 model         | Airbus A320-200
 range         | 5700
 when_add      | 2017-01-31 18:28:49.230179
 operation     | INSERT
...
```

Предложение ON CONFLICT

Ситуация: при вставке новых строк могут возникать ситуации, когда нарушается ограничение первичного или уникального ключей, поскольку вставляемые строки могут иметь значения ключевых атрибутов, совпадающие с теми, что уже имеются в таблице.

Решение: использовать предложение **ON CONFLICT**.

Оно предусматривает два варианта действий:

- отменять добавление новой строки, для которой имеет место конфликт значений ключевых атрибутов, и при этом не порождать сообщения об ошибке – DO NOTHING;
- заменять операцию добавления новой строки операцией обновления существующей строки, с которой конфликтует добавляемая строка – DO UPDATE.

В том случае, когда в предложении ON CONFLICT не указана дополнительная информация об именах столбцов или ограничений, по которым предполагается возможный конфликт, проверка выполняется по *первичному* ключу и по *уникальным* ключам.

Предложение DO NOTHING (1)

- Попробуем добавить строку, которая гарантированно будет конфликтовать с уже существующей строкой, причем, как по первичному ключу aircraft_code, так и по уникальному ключу model.

```
WITH add_row AS
( INSERT INTO aircrafts_tmp
  VALUES ( 'SU9', 'Sukhoi SuperJet-100', 3000 )
  ON CONFLICT DO NOTHING
  RETURNING *
)
INSERT INTO aircrafts_log
SELECT add_row.aircraft_code, add_row.model,
       add_row.range, CURRENT_TIMESTAMP, 'INSERT'
FROM add_row;
```

```
INSERT 0 0
```

это сообщение относится к таблице aircrafts_log, т. е. к команде в главном запросе, а не в общем табличном выражении

ВАЖНО! Не будет выведено никаких сообщений об ошибках.

Предложение DO NOTHING (2)

- Добавляемая строка будет иметь конфликт с существующей строкой как по столбцу aircraft_code, так и по столбцу model.

```
INSERT INTO aircrafts_tmp
VALUES ( 'SU9', 'Sukhoi SuperJet-100', 3000 )
ON CONFLICT ( aircraft_code ) DO NOTHING
RETURNING *;
```

указан конкретный столбец для проверки наличия конфликта – первичный ключ

Получим только такое сообщение:

```
 aircraft_code | model | range
---------------+-------+-------
(0 строк)
INSERT 0 0
```

- Это сообщение было выведено потому, что в команду включено предложение RETURNING *. Сообщение о дублировании значений столбца model не выводится. Как думаете, почему?

Предложение DO NOTHING (3)

- Давайте в предыдущей команде INSERT изменим значение столбца aircraft_code, чтобы оно стало уникальным:

```
INSERT INTO aircrafts_tmp
VALUES ( 'S99', 'Sukhoi SuperJet-100', 3000 )
ON CONFLICT ( aircraft_code ) DO NOTHING
RETURNING *;
```

- Поскольку конфликта по столбцу aircraft_code теперь нет, то далее проверяется выполнение требования уникальности по столбцу model. В результате получим традиционное сообщение об ошибке, относящееся к столбцу model:

```
ОШИБКА: повторяющееся значение ключа нарушает ограничение
уникальности "aircrafts_tmp_model_key"
ПОДРОБНОСТИ: Ключ "(model)=(Sukhoi SuperJet-100)" уже
существует.
```

Предложение DO UPDATE (1)

- Если при вставке новой строки имеет место дублирование по атрибутам первичного ключа со строкой, находящейся в таблице, тогда мы будем обновлять значения всех остальных атрибутов в этой строке, независимо от того, совпадают ли они со значениями в новой строке или нет.

```
INSERT INTO aircrafts_tmp
VALUES ( 'SU9', 'Sukhoi SuperJet', 3000 )
ON CONFLICT ON CONSTRAINT aircrafts_tmp_pkey
  DO UPDATE SET model = excluded.model,
                range = excluded.range
RETURNING *;
```

имя таблицы не указывается

наименование ограничения первичного ключа

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

Таблица **excluded** поддерживается самой СУБД. В этой таблице хранятся все строки, предлагаемые для вставки в рамках текущей команды INSERT.

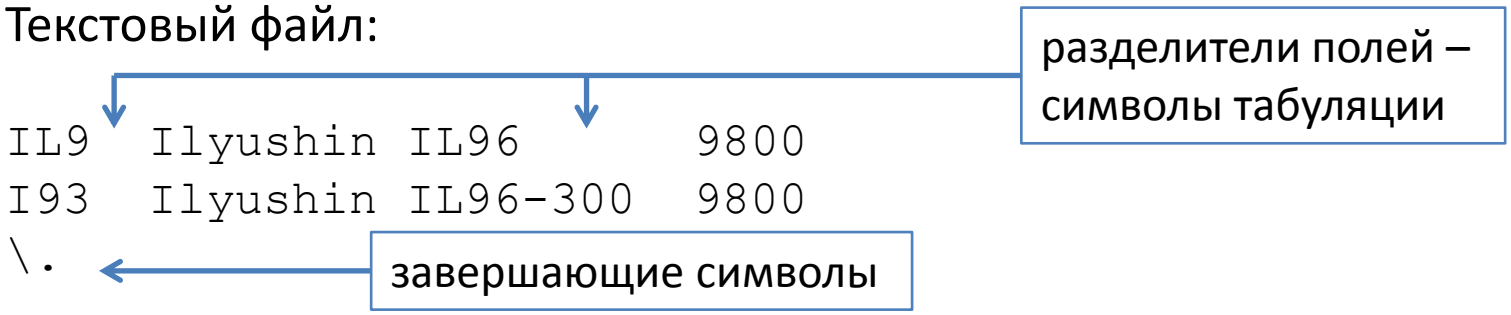
Предложение DO UPDATE (2)

- Предложение ON CONFLICT DO UPDATE гарантирует *атомарное* выполнение операции вставки или обновления строк.
- Атомарность означает, что проверка наличия конфликта и последующее обновление выполняются как *неделимая операция*, т. е. другие транзакции не могут изменить значение столбца, вызывающее конфликт, так, чтобы в результате конфликт исчез и уже стало возможным выполнить операцию INSERT, а не UPDATE, или, наоборот, в случае отсутствия конфликта он вдруг появился, и уже операция INSERT стала бы невозможной.
- Такая атомарная операция даже имеет название **UPSERT** — «UPDATE или INSERT».

Команда COPY

Для массового ввода строк в таблицы используется команда COPY.

Текстовый файл:



Ввод данных из файла в таблицу:

```
COPY aircrafts_tmp FROM '/home/postgres/aircrafts.txt';
COPY 2
```

Выполняются проверки всех ограничений, наложенных на таблицу.

Вывод данных из таблицы в файл:

```
COPY aircrafts_tmp TO '/home/postgres/aircrafts_tmp.txt'
  WITH ( FORMAT csv );
```

CSV — Comma Separated Values

Получим файл такого вида:

```
773,Boeing 777-300,11100
763,Boeing 767-300,7900
SU9,Sukhoi SuperJet-100,3000
...
```

5.2. Обновление строк в таблицах

Запись изменений в журнальную таблицу

```
WITH update_row AS
( UPDATE aircrafts_tmp
  SET range = range * 1.2 WHERE model ~ '^Bom'
  RETURNING *
)
INSERT INTO aircrafts_log
SELECT ur.aircraft_code, ur.model, ur.range,
       CURRENT_TIMESTAMP, 'UPDATE'
FROM update_row  ur;
```

«полезная» работа

запись в журнал

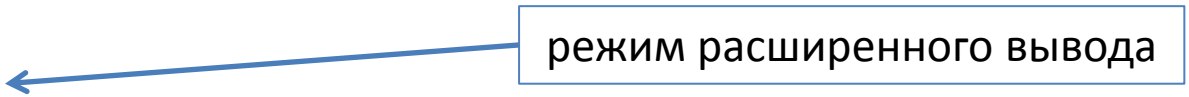
```
INSERT 0 1
```

сообщение относится непосредственно к внешнему запросу

ВАЖНО! Главный запрос может получить доступ к обновленным данным только через временную таблицу, которую формирует предложение RETURNING:

```
...
FROM update_row ur;
```

Что получилось в журнальной таблице?




```
\x
SELECT * FROM aircrafts_log
WHERE model ~ '^Bom'
ORDER BY when_add;
--[ RECORD 1 ]-+---------------------------
 aircraft_code | CR2
 model         | Bombardier CRJ-200
 range         | 2700
 when_add      | 2017-02-05 00:27:38.591958
 operation     | INSERT
--[ RECORD 2 ]-+---------------------------
 aircraft_code | CR2
 model         | Bombardier CRJ-200
 range         | 3240
 when_add      | 2017-02-05 00:27:56.688933
 operation     | UPDATE
```

Правдоподобная ситуация (1)

Представим, что руководство компании хочет иметь возможность видеть динамику продаж билетов по всем направлениям, а именно: общее число проданных билетов и дату/время последнего увеличения их числа для конкретного направления.

Создадим временную таблицу tickets_directions. В ней будет четыре столбца:

- город отправления – departure_city;
- город прибытия – arrival_city;
- дата/время последнего увеличения числа проданных билетов – last_ticket_time;
- число проданных билетов на этот момент времени по данному направлению – tickets_num.

Правдоподобная ситуация (2)

```
CREATE TEMP TABLE tickets_directions AS
SELECT DISTINCT departure_city, arrival_city
FROM routes;
```

нужны только уникальные пары городов отправления и прибытия

```
ALTER TABLE tickets_directions
ADD COLUMN last_ticket_time timestamp;

ALTER TABLE tickets_directions
ADD COLUMN tickets_num integer DEFAULT 0;
```

заполним столбец-счетчик нулями

Первичный ключ не создаем, что не помешает нам однозначно идентифицировать строки в таблице tickets_directions

Правдоподобная ситуация (3)

- Для того чтобы не усложнять изложение материала, создадим временную таблицу, являющуюся аналогом таблицы «Перелеты» (ticket_flights), однако *без внешних ключей*.
- Поэтому мы сможем добавлять в нее строки, не заботясь о добавлении строк в таблицы «Билеты» (tickets) и «Бронирования» (bookings).

```
CREATE TEMP TABLE ticket_flights_tmp AS
SELECT * FROM ticket_flights WITH NO DATA;

ALTER TABLE ticket_flights_tmp
ADD PRIMARY KEY ( ticket_no, flight_id );
```

Основная команда

Теперь представим команду, которая и будет добавлять новую запись о продаже билета и увеличивать значение счетчика проданных билетов в таблице tickets_directions.

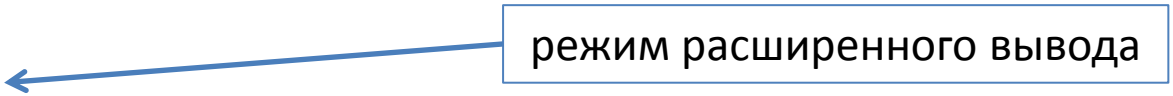
```
WITH sell_ticket AS
( INSERT INTO ticket_flights_tmp
    ( ticket_no, flight_id, fare_conditions, amount )
  VALUES ( '1234567890123', 30829, 'Economy', 12800 )
  RETURNING *
)
UPDATE tickets_directions td
SET last_ticket_time = CURRENT_TIMESTAMP,
    tickets_num = tickets_num + 1
WHERE ( td.departure_city, td.arrival_city ) =
      ( SELECT departure_city, arrival_city
        FROM flights_v
        WHERE flight_id = ( SELECT flight_id
                            FROM sell_ticket )
      );
UPDATE 1
```

продажа билета

сравнение сразу с двумя столбцами

обновление счетчика

Что получилось?




```
\x

SELECT * FROM tickets_directions WHERE tickets_num > 0;
--[ RECORD 1 ]----+---------------------------
 departure_city   | Сочи
 arrival_city     | Красноярск
 last_ticket_time | 2017-02-04 21:15:32.903687
 tickets_num      | 1
```

Другое решение задачи

- Принципиальное отличие от первого варианта: для определения обновляемой строки в таблице tickets_directions используется операция *соединения таблиц*.

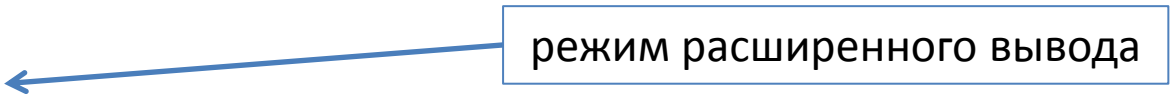
```
WITH sell_ticket AS
( INSERT INTO ticket_flights_tmp
    (ticket_no, flight_id, fare_conditions, amount )
  VALUES ( '1234567890123', 7757, 'Economy', 3400 )
  RETURNING *
)
UPDATE tickets_directions td
SET last_ticket_time = CURRENT_TIMESTAMP,
    tickets_num = tickets_num + 1
FROM flights_v f
WHERE td.departure_city = f.departure_city AND
      td.arrival_city = f.arrival_city AND
      f.flight_id = ( SELECT flight_id FROM sell_ticket );
```

не указывается таблица tickets_directions

условие определения обновляемой строки

UPDATE 1

Что получилось?




```
\x

SELECT * FROM tickets_directions WHERE tickets_num > 0;
--[ RECORD 1 ]----+---------------------------
 departure_city   | Сочи
 arrival_city     | Красноярск
 last_ticket_time | 2017-02-04 21:15:32.903687
 tickets_num      | 1
--[ RECORD 2 ]----+---------------------------
 departure_city   | Москва
 arrival_city     | Сочи
 last_ticket_time | 2017-02-04 21:18:40.353408
 tickets_num      | 1
```

5.3. Удаление строк из таблиц

Запись изменений в журнальную таблицу

```
WITH delete_row AS
( DELETE FROM aircrafts_tmp
  WHERE model ~ '^Bom'
  RETURNING *
)
INSERT INTO aircrafts_log
SELECT dr.aircraft_code, dr.model, dr.range,
       CURRENT_TIMESTAMP, 'DELETE'
FROM delete_row dr;

INSERT 0 1
```

«полезная» работа

запись в журнал

сообщение относится непосредственно к внешнему запросу

История изменений строки

Посмотрим историю изменений строки с описанием самолета Bombardier CRJ-200:

```
SELECT * FROM aircrafts_log
WHERE model ~ '^Bom' ORDER BY when_add;
```

```
--[ RECORD 1 ]-+--------------------------
 aircraft_code | CR2
 model         | Bombardier CRJ-200
 range         | 2700
 when_add      | 2017-02-05 00:27:38.591958
 operation     | INSERT
--[ RECORD 2 ]-+--------------------------
 aircraft_code | CR2
 model         | Bombardier CRJ-200
 range         | 3240
 when_add      | 2017-02-05 00:27:56.688933
 operation     | UPDATE
--[ RECORD 3 ]-+--------------------------
 aircraft_code | CR2
 model         | Bombardier CRJ-200
 range         | 3240
 when_add      | 2017-02-05 00:34:59.510911
 operation     | DELETE
```

Предложение USING

Для удаления конкретных строк из данной таблицы можно использовать информацию не только из нее, но также и из других таблиц.

Ситуация: руководство авиакомпании решило удалить из парка самолетов машины компаний Boeing и Airbus, имеющие наименьшую дальность полета.

```
WITH min_ranges AS
( SELECT aircraft_code,
         rank() OVER
           ( PARTITION BY left( model, 6 )
             ORDER BY range )
         AS rank
  FROM aircrafts_tmp
  WHERE model ~ '^Airbus' OR model ~ '^Boeing'
)
DELETE FROM aircrafts_tmp a
USING min_ranges mr
WHERE a.aircraft_code = mr.aircraft_code AND
      mr.rank = 1
RETURNING *;
```

ранжируем модели каждой компании по возрастанию дальности полета

а почему здесь нет **RETURNING * ?**

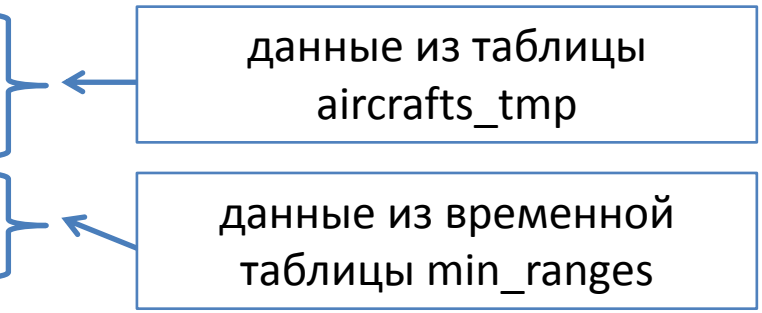
ранг 1 соответствует наименьшей дальности полета

Что мы удаляем?

- Мы включили в команду DELETE предложение RETURNING *, чтобы показать, как выглядят комбинированные строки, сформированные с помощью предложения USING.
- Конечно, удаляются не комбинированные строки, а только оригинальные строки из таблицы aircrafts_tmp.

```
--[ RECORD 1 ]-+----------------
 aircraft_code | 321
 model         | Airbus A321-200
 range         | 5600
 aircraft_code | 321
 rank          | 1
--[ RECORD 2 ]-+----------------
 aircraft_code | 733
 model         | Boeing 737-300
 range         | 4200
 aircraft_code | 733
 rank          | 1

DELETE 2
```



Команда TRUNCATE

- Обе следующие две команды позволяют удалить все строки из таблицы aircrafts_tmp:

```
DELETE FROM aircrafts_tmp;
```

```
TRUNCATE aircrafts_tmp;
```

- Однако команда TRUNCATE работает быстрее.

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. Изучить материал главы 7. Запросы к базе данных выполнять с помощью утилиты psql, описанной в главе 2, параграф 2.2.