

***Федеральное агентство по рыболовству***

***Федеральное государственное бюджетное образовательное***

***учреждение высшего образования***

***«Астраханский государственный технический университет»***

**Система менеджмента качества в области образования, воспитания, науки и инноваций сертифицирована**

**ООО «ДКС РУС» по международному стандарту ISO 9001:2015**

|  |  |
| --- | --- |
| Институт | Информационных технологий и коммуникаций |
| Направление | 09.03.01 Информатика и вычислительная техника |
| Профиль | «Автоматизированные системы обработки информации и управления» |
| Кафедра | «Автоматизированные системы обработки информации и управления» |

**Лабораторная работа № 12**

**«PostgreSQL. Вставка, изменение и удаление строк**»

по дисциплине «СУБД PostgreSQL»

|  |  |  |
| --- | --- | --- |
|  |  | Работа выполнена студентом группы ДИНРБ-31  Кузургалиев Р.А. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (Фамилия И.О.) подпись |
|  |  | Проверил работу:  ст. преподаватель Мамлеева А.Р,\_\_\_\_\_\_\_\_\_\_\_\_  (ученая степень, ученое звание, Фамилия И.О.) |

Работа защищена

«\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_ 2024 г.

**АСТРАХАНЬ – 2024**

**СОДЕРЖАНИЕ ОТЧЕТА**

**Цель работы**: Получить практические навыки в формировании SQL-запросов на вставку, изменение и удаление строк в таблицах СУБД PostgreSQL.

**Литература**: 1. Лекция № 5 «Вставка, изменение и удаление строк».

2. Моргунов Е.П. PostgreSQL. Основы языка SQL: учеб. пособие. – СПб.:

БХВ-Петербург, 2018.- 336 с.: ил.

**Задание 1.**

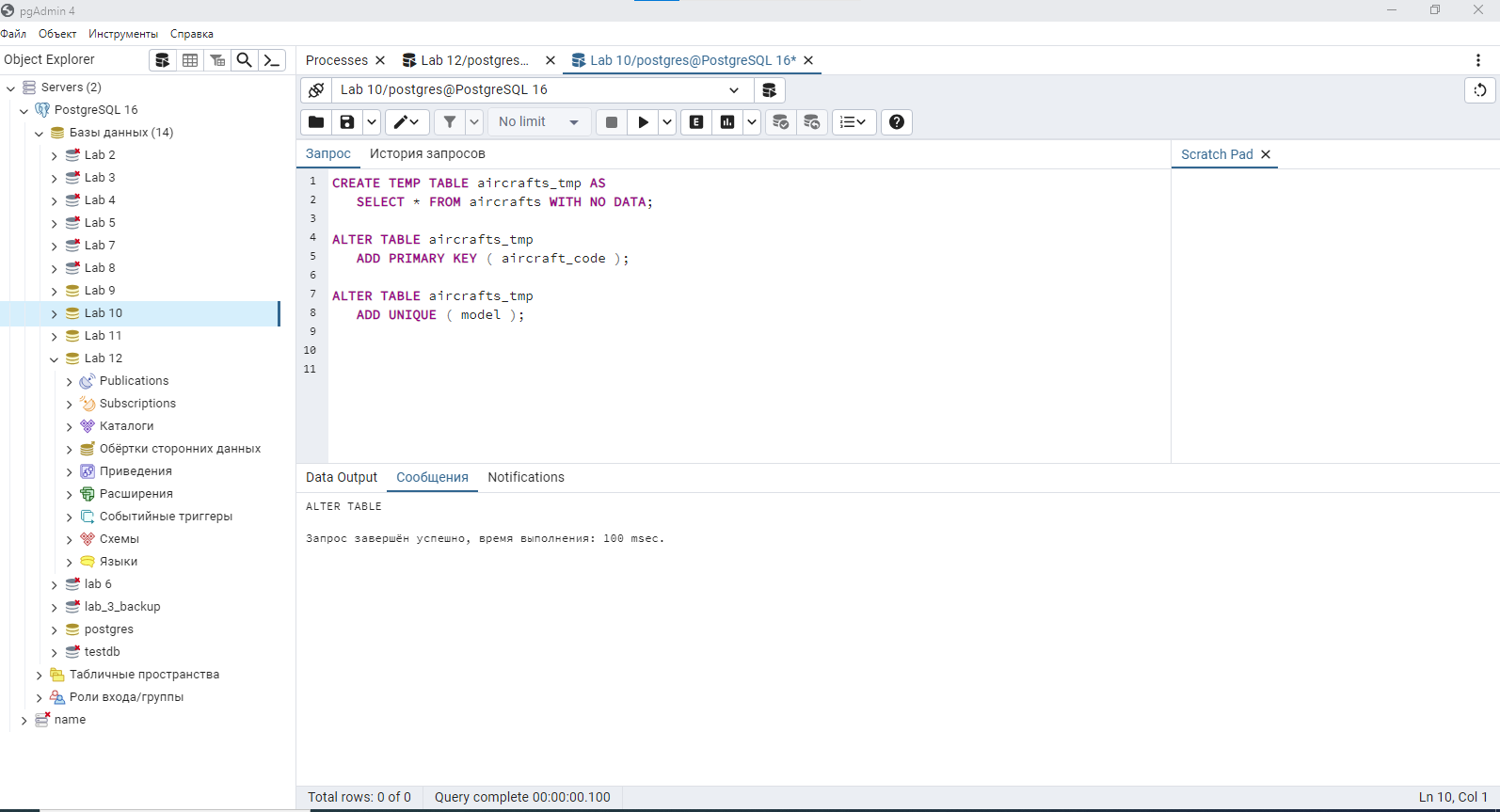
Добавьте в определение таблицы **aircrafts\_log** значение по умолчанию  
**current\_timestamp** и соответствующим образом измените команды **INSERT**,  
приведенные в тексте лекции:

Создание временной таблицы **aircrafts\_tmp:**

**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 );**

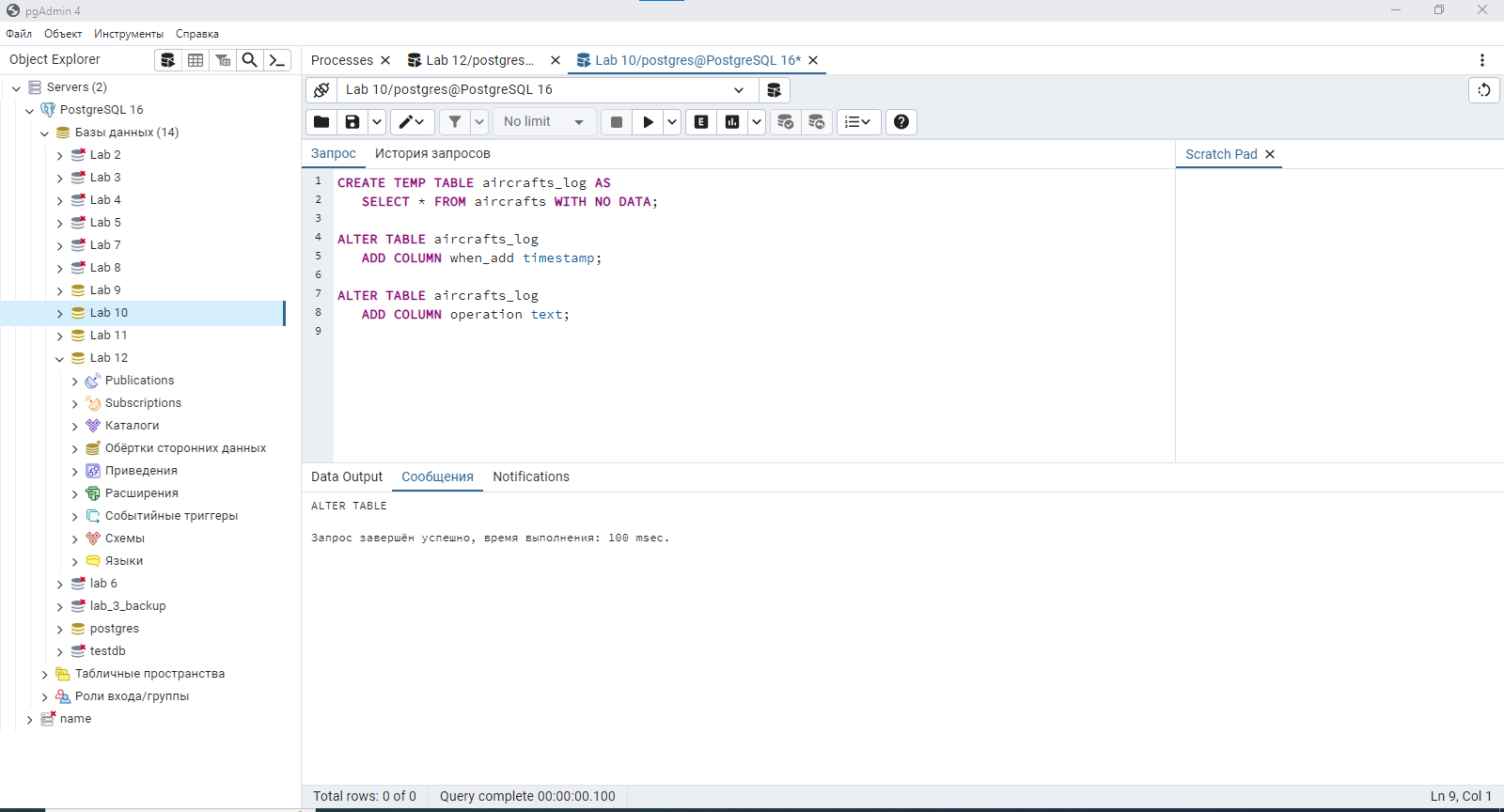


Создание временной таблицы **aircrafts\_log:**

**CREATE TEMP TABLE aircrafts\_log AS  
 SELECT \* FROM aircrafts WITH NO DATA;**

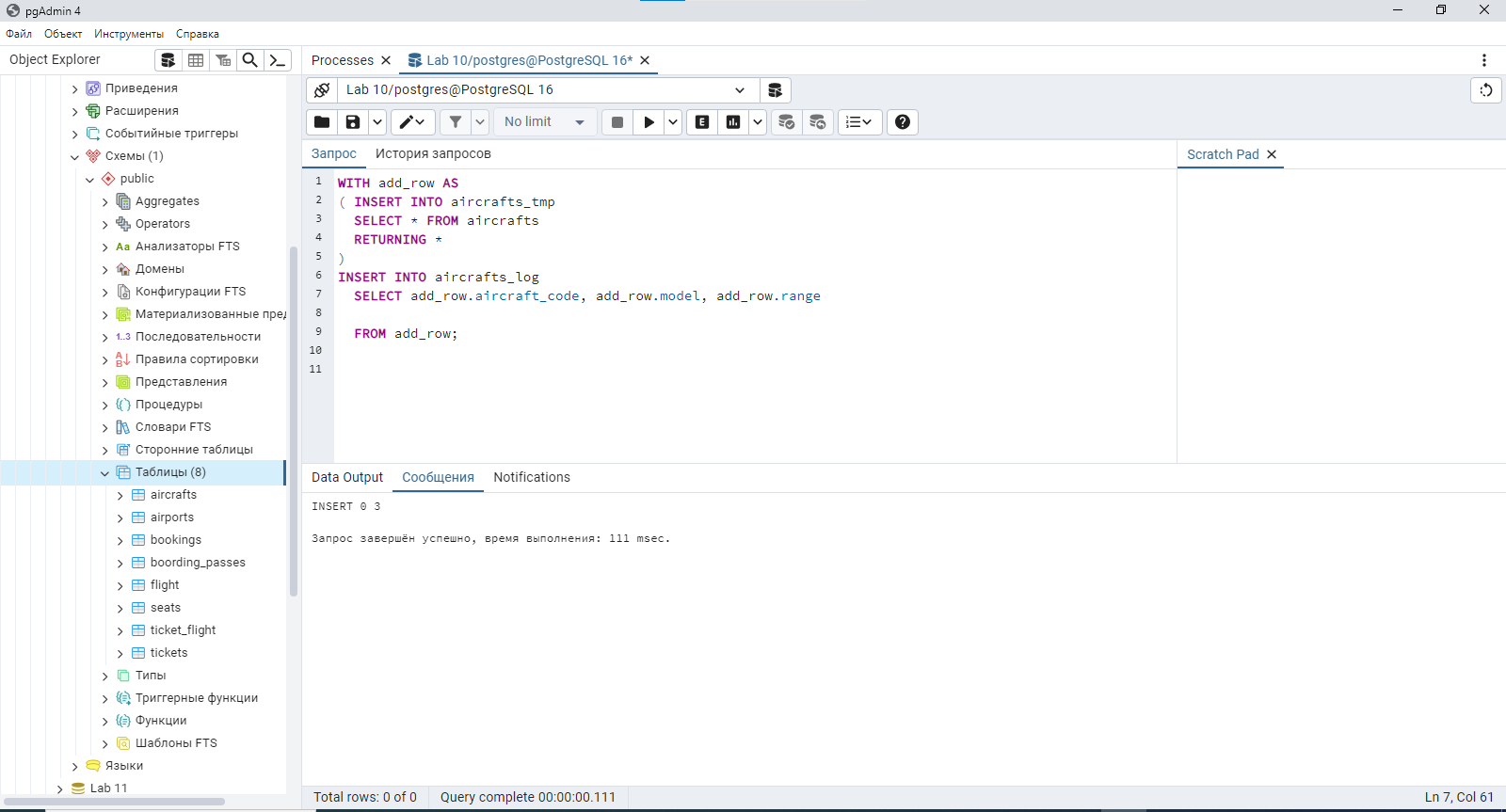
**ALTER TABLE aircrafts\_log  
 ADD COLUMN when\_add timestamp;**

**ALTER TABLE aircrafts\_log  
 ADD COLUMN operation text;**

****

Вставка строк во временные таблицы:

**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;**

****

**Задание 2.**

В предложении **RETURNING** можно указывать не только символ «**∗**», означающий  
выбор всех столбцов таблицы, но и более сложные выражения, сформированные  
на основе этих столбцов.

В тексте лекции мы копировали содержимое таблицы  
«Самолеты» в таблицу **aircrafts\_tmp**, используя в предложении RETURNING  
именно «**∗**».

Однако возможен и другой вариант запроса:

WITH add\_row AS( INSERT INTO aircrafts\_tmp SELECT \* FROM aircrafts RETURNING aircraft\_code, model, range, 'INSERT')INSERT INTO aircrafts\_log SELECT ? FROM add\_row;

Что нужно написать в этом запросе вместо вопросительного знака?

aircraft\_code, model, range

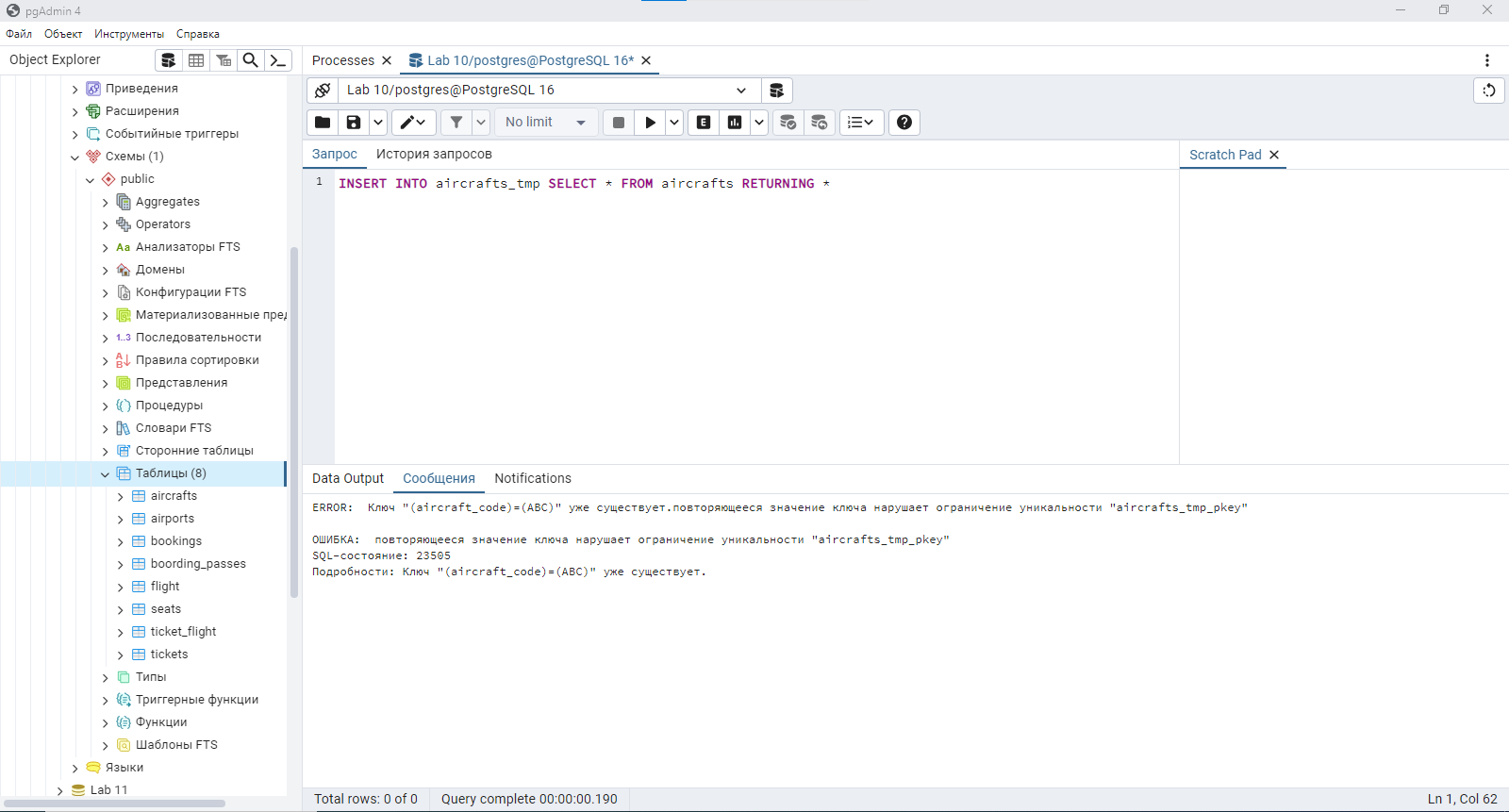
**Задание 3.**

Если бы мы для копирования данных в таблицу **aircrafts\_tmp** использовали команду INSERT без общего табличного выражения **INSERT INTO aircrafts\_tmp SELECT \* FROM aircrafts;** то в качестве выходного результата мы увидели бы сообщение:

INSERT 0 9

Как вы думаете, что будет выведено, если дополнить команду предложением  
**RETURNING \***?

**INSERT INTO aircrafts\_tmp SELECT \* FROM aircrafts RETURNING \*;**

****Проверьте ваши предположения на практике.

Подумайте, каким образом можно использовать выведенный результат?

**Задание 4.**

В тексте лекции в предложениях **ON CONFLICT** команды **INSERT** мы использовали только выражения, состоящие из имени одного столбца.

Однако в таблице «**Места**» (**seats**) первичный ключ является составным и включает два столбца.

Напишите команду **INSERT** для вставки новой строки в эту таблицу и предусмотрите возможный конфликт добавляемой строки со строкой, уже имеющейся в таблице.

Сделайте два варианта предложения **ON CONFLICT**:

* первый — с использованием перечисления имен столбцов для проверки наличия дублирования,
* второй — с использованием предложения **ON CONSTRAINT**.

Для того чтобы не изменить содержимое таблицы «**Места**», создайте ее копию  
и выполняйте все эти эксперименты с таблицей-копией.

**Задание 5.**

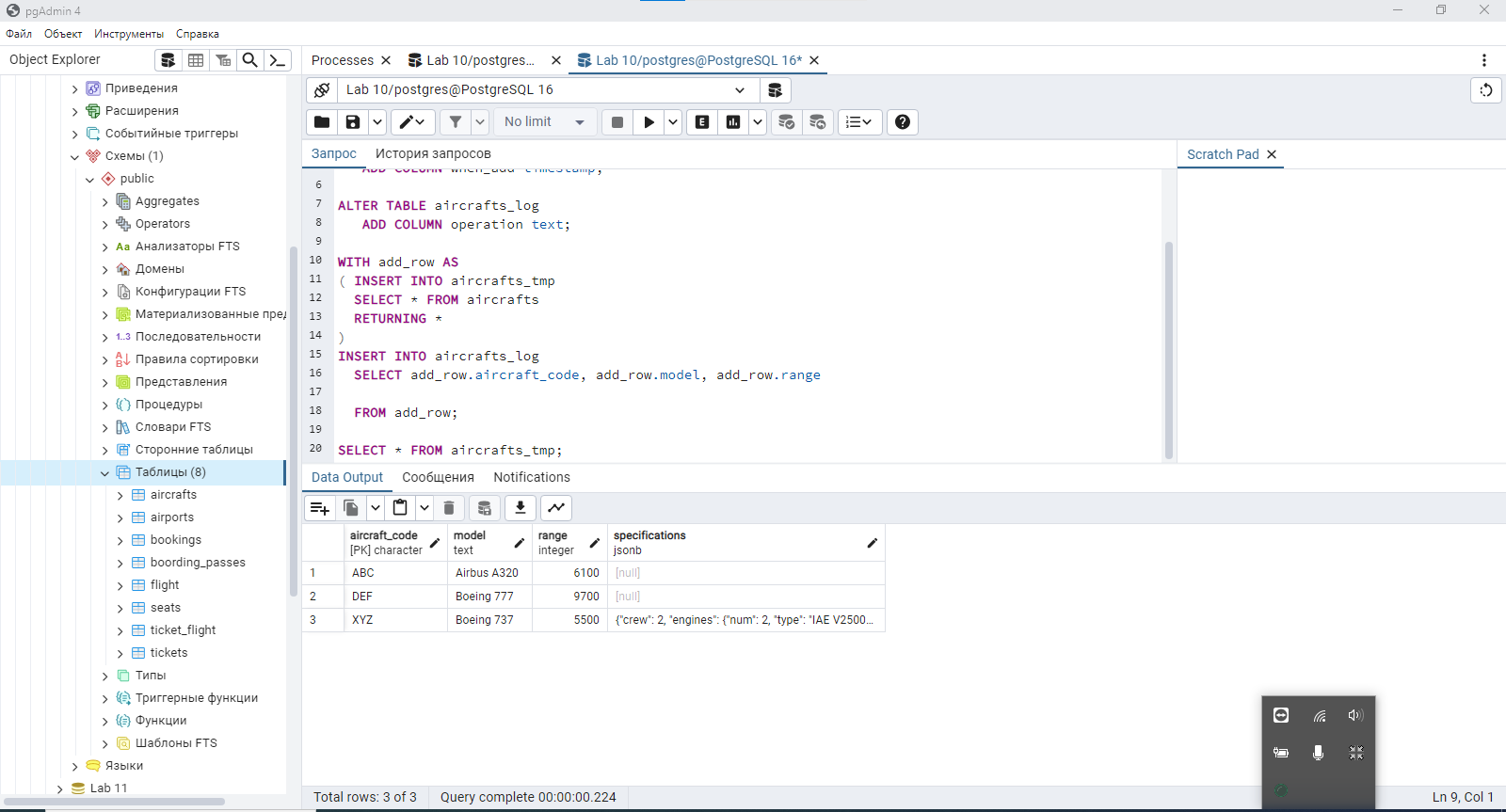
Команда **COPY** по умолчанию ожидает получения вводимых данных в формате  
**text**, когда значения данных разделяются символами табуляции. Однако можно представлять входные данные в формате **CSV** (Comma Separated Values), т.е.  
использовать в качестве разделителя **запятую**.

**COPY aircrafts\_tmp FROM STDIN WITH (FORMAT csv);**

Вводите данные для копирования, разделяя строки переводом строки.  
Закончите ввод строкой '\.'.

**IL9, Ilyushin IL96, 9800  
I93, Ilyushin IL96-300, 9800  
\.**COPY 2  
**SELECT \* FROM aircrafts\_tmp;**

Пример результата:

aircraft\_code | model | range  
--------------+----------------------+-------  
...  
CN1 | Cessna 208 Caravan | 1200  
CR2 | Bombardier CRJ-200 | 2700  
IL9 | Ilyushin IL96 | 9800  
I93 | Ilyushin IL96-300 | 9800  
(11 строк)  


Как вы думаете, почему при выводе данных из таблицы вновь введенные значения в столбце **model** оказались смещены вправо?

**Задание 6.**

Команда **COPY** позволяет получить входные данные из файла и поместить их  
в таблицу. Этот файл должен быть доступен тому пользователю операционной системы, от имени которого запущен серверный процесс, как правило, это  
пользователь **postgres**.

Подготовьте файл, например, **/home/postgres/aircrafts\_tmp.csv**, имеющий такую  
структуру:

– каждая строка файла соответствует одной строке таблицы **aircrafts\_tmp**;  
– значения данных в строке файла разделяются **запятыми**.

Например:

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

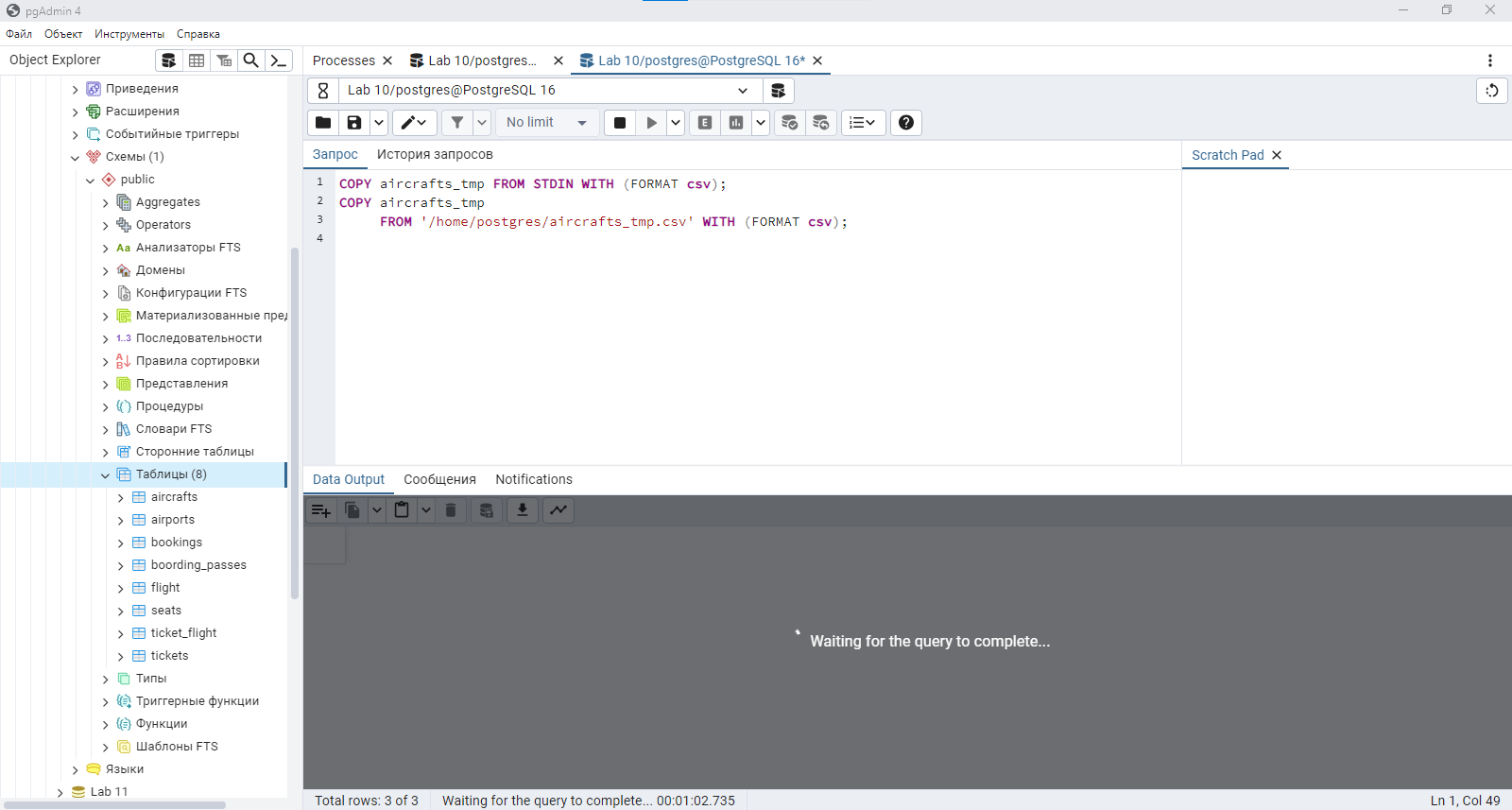
Введите в этот файл данные о нескольких самолетах, причем часть из них уже должна быть представлена в таблице, а часть — нет.

Поскольку при выполнении команды **COPY** проверяются все ограничения целостности, наложенные на таблицу, то дублирующие строки добавлены, конечно же, не будут.

***Ответьте на вопрос: будут добавлены или нет строки, содержащиеся в этом же файле, но отсутствующие в таблице?***

Проверьте свою гипотезу, выполнив вставку строк в таблицу из этого файла:  
**COPY aircrafts\_tmp**

**FROM '/home/postgres/aircrafts\_tmp.csv' WITH (FORMAT csv);**



**Задание 7.**

В тексте лекции был приведен запрос, предназначенный для учета числа билетов, проданных по всем направлениям на текущую дату.

Однако тот запрос был рассчитан на одновременное добавление только одной записи в таблицу «**Перелеты**» (**ticket\_flights\_tmp**).

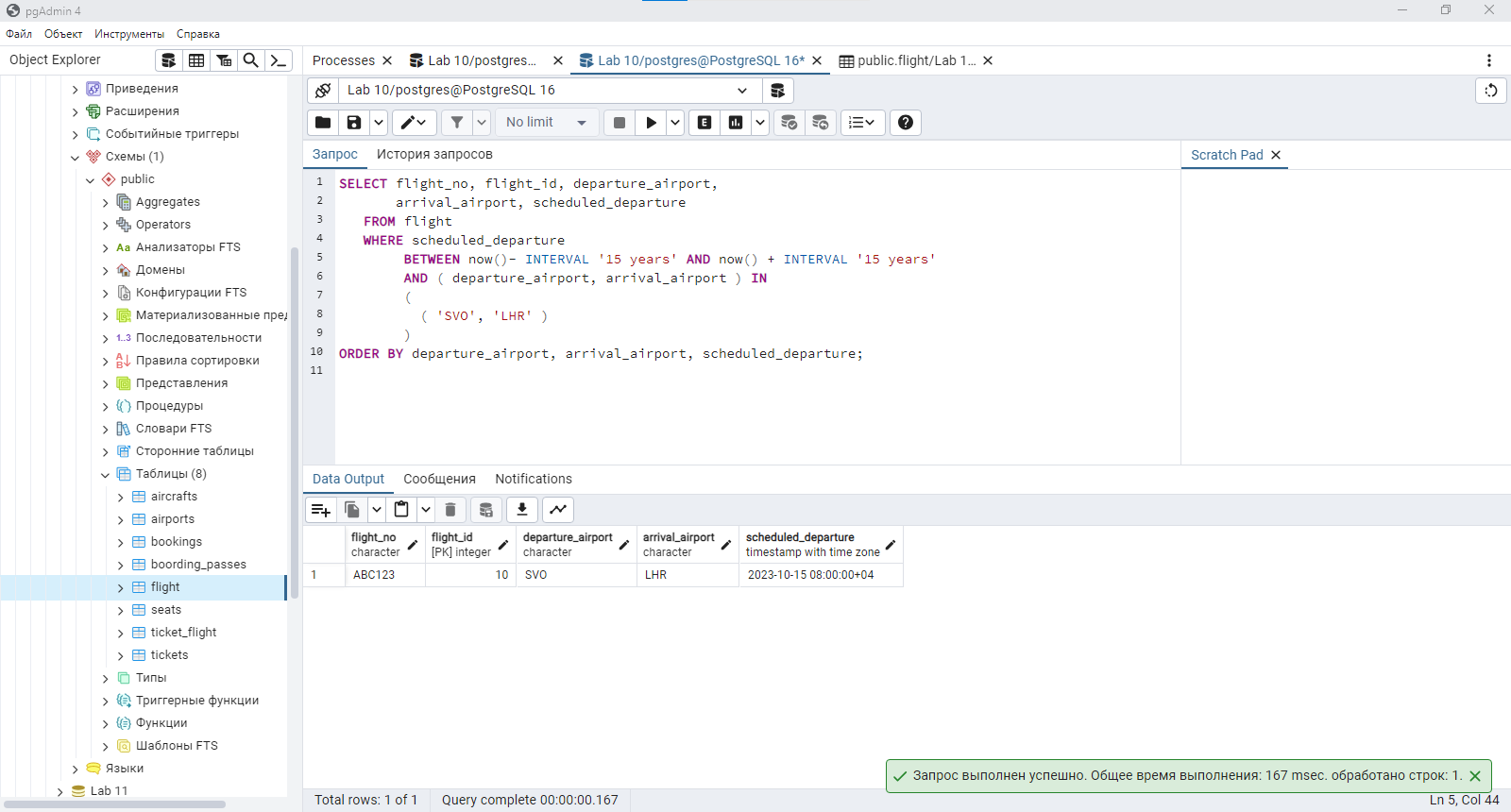
Ниже будет предложен более универсальный запрос, который предусматривает возможность единовременного ввода нескольких записей о перелетах, выполняемых на различных рейсах.

Для проверки работоспособности предлагаемого запроса выберем несколько  
рейсов, например, по маршрутам: Красноярск — Москва, Москва — Сочи, Сочи — Москва,  
Сочи — Красноярск.

Для определения идентификаторов рейсов сформируем вспомогательный запрос, в котором даты начала и конца рассматриваемого периода времени зададим с помощью функции **now**.

Использование этой функции необходимо, поскольку в будущих версиях базы данных могут быть представлены другие диапазоны дат.

**SELECT flight\_no, flight\_id, departure\_city,  
 arrival\_city, scheduled\_departure  
 FROM flights\_v  
 WHERE scheduled\_departure  
 BETWEEN bookings.now() AND bookings.now() + INTERVAL '15 days'  
 AND ( departure\_city, arrival\_city ) IN  
 ( ( 'Красноярск', 'Москва' ),  
 ( 'Москва', 'Сочи'),  
 ( 'Сочи', 'Москва' ),  
 ( 'Сочи', 'Красноярск' )  
 )  
ORDER BY departure\_city, arrival\_city, scheduled\_departure;**

****

Обратите внимание на предикат **IN**: в нем используются не индивидуальные  
значения, а пары значений.

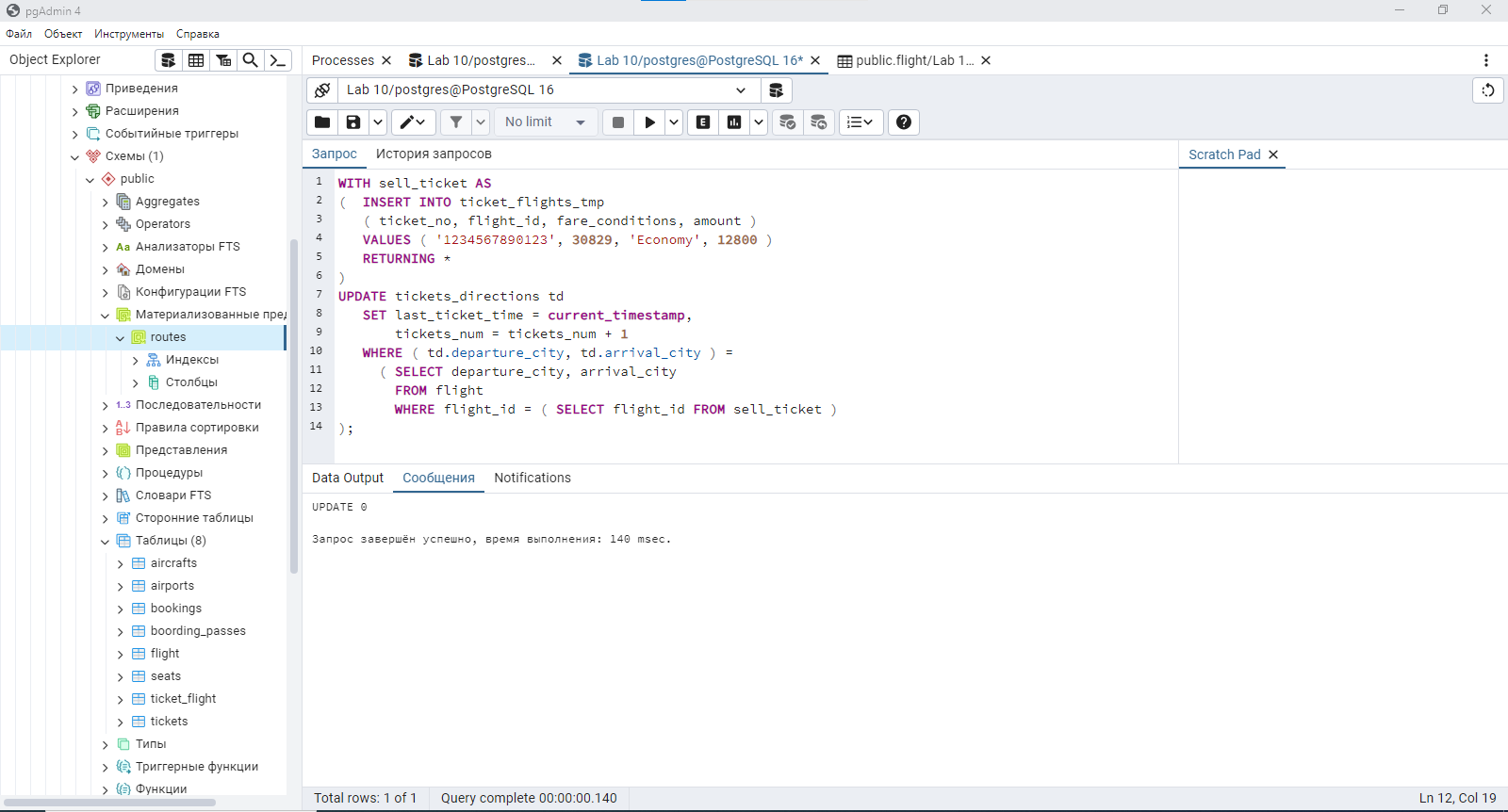
Предположим, что в течение указанного интервала времени пассажир планирует совершить перелеты по маршруту: Красноярск — Москва, Москва — Сочи,  
Сочи — Москва, Москва — Сочи, Сочи — Красноярск.

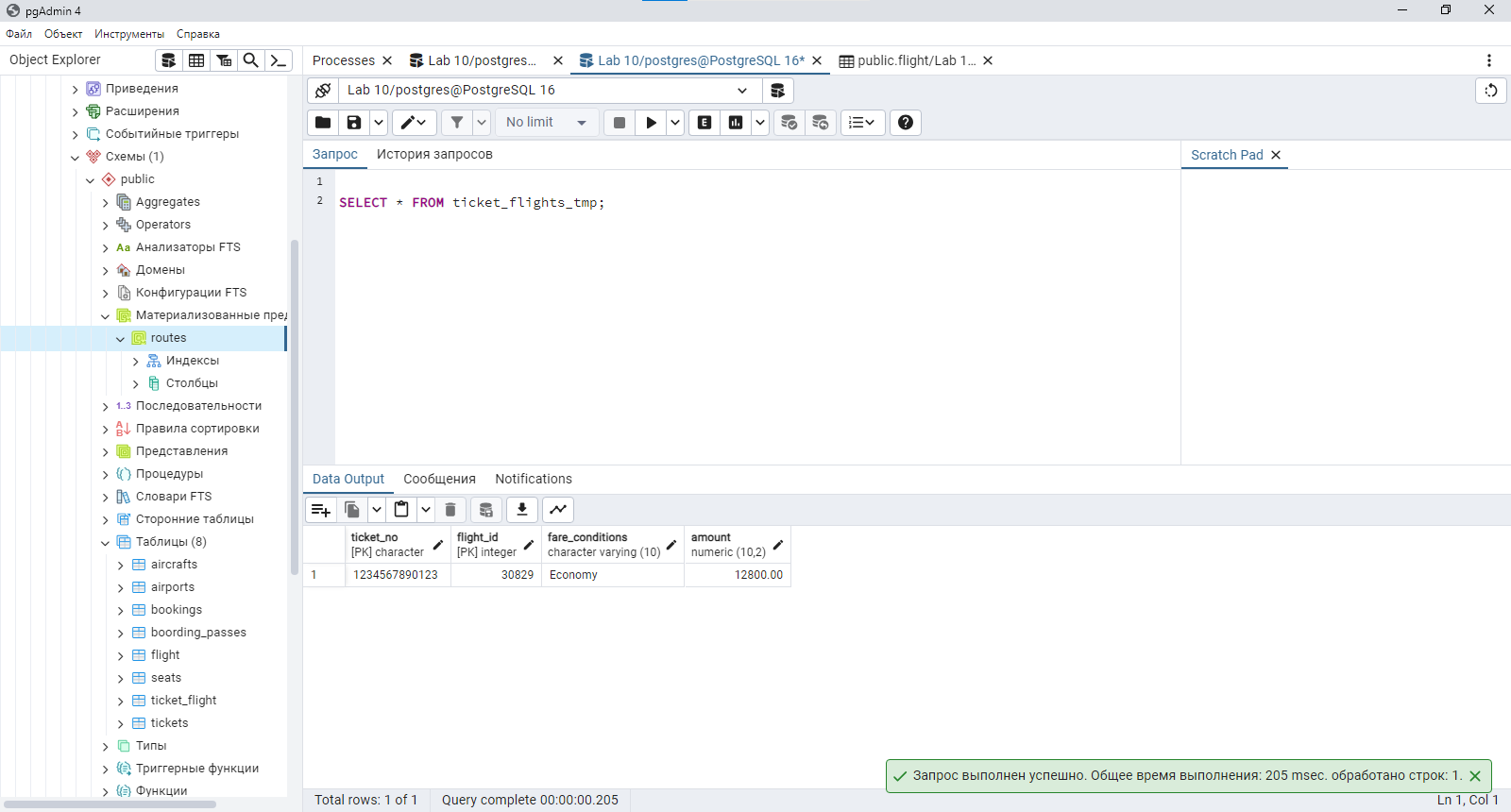
Выполнив вспомогательный запрос, выберем следующие идентификаторы рейсов (в этом же порядке):

13829, 4728, 30523, 7757, 30829.

**WITH sell\_tickets AS  
( INSERT INTO ticket\_flights\_tmp  
 ( ticket\_no, flight\_id, fare\_conditions, amount )  
 VALUES ( '1234567890123', 13829, 'Economy', 10500 ),  
 ( '1234567890123', 4728, 'Economy', 3400 ),  
 ( '1234567890123', 30523, 'Economy', 3400 ),  
 ( '1234567890123', 7757, 'Economy', 3400 ),  
 ( '1234567890123', 30829, 'Economy', 12800 )  
 RETURNING \*  
)  
UPDATE tickets\_directions td  
 SET last\_ticket\_time = current\_timestamp,  
 tickets\_num = tickets\_num +  
 ( SELECT count( \* )  
 FROM sell\_tickets st, flights\_v f  
 WHERE st.flight\_id = f.flight\_id  
 AND f.departure\_city = td.departure\_city  
 AND f.arrival\_city = td.arrival\_city  
 )  
 WHERE ( td.departure\_city, td.arrival\_city ) IN  
 ( SELECT departure\_city, arrival\_city  
 FROM flights\_v  
 WHERE flight\_id IN ( SELECT flight\_id FROM sell\_tickets )  
);**

UPDATE 4





В этой версии запроса предусмотрен единовременный ввод нескольких строк  
в таблицу **ticket\_flights\_tmp**, причем перелеты могут выполняться на различных рейсах.

Поэтому необходимо преобразовать список идентификаторов этих рейсов в множество пар «**город отправления — город прибытия**», поскольку именно для таких пар и ведется подсчет числа забронированных перелетов.

Эта задача решается в предложении **WHERE**, где вложенный подзапрос формирует список идентификаторов рейсов, а внешний подзапрос преобразует этот  
список в множество пар «город отправления — город прибытия».

Затем с помощью предиката **IN** производится отбор строк таблицы **tickets\_directions**  
для обновления.

Теперь обратимся к предложению **SET**.

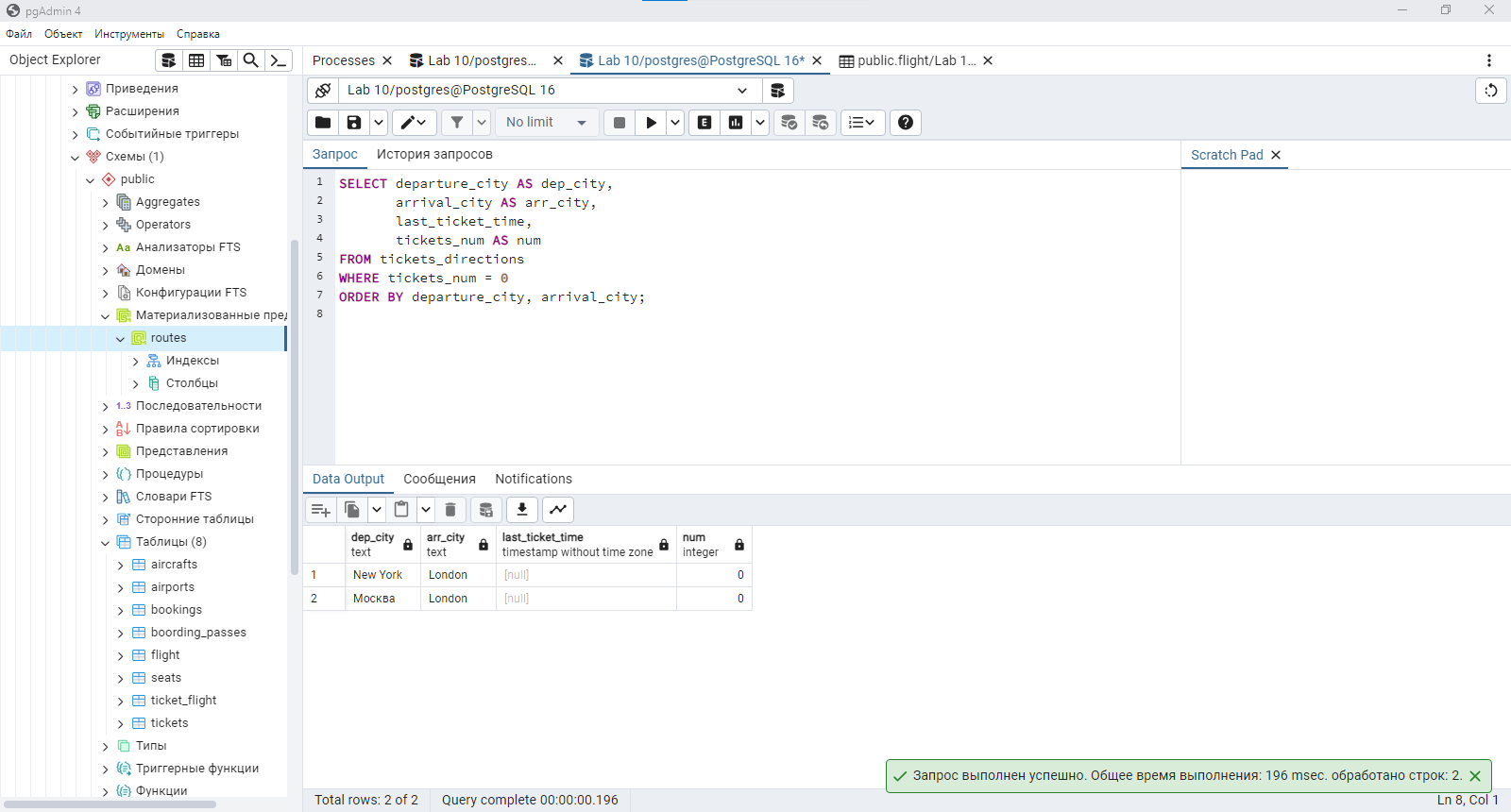
Подзапрос с функцией count вычисляет количество перелетов по *каждому* направлению.

Это коррелированный подзапрос: он выполняется для каждой строки, отобранной в предложении **WHERE**.

В нем используется соединение временной таблицы **sell\_tickets** с представлением **flights\_v**. Это нужно для того, чтобы подсчитать все перелеты, соответствующие паре атрибутов «город отправления — город прибытия», взятых из текущей обновляемой строки таблицы **tickets\_directions**. Этот подзапрос позволяет учесть такой факт: рейсы могут иметь различные идентификаторы **flight\_id**, но при этом соответствовать одному и тому же направлению, а в таблице **tickets\_directions** учитываются именно направления.

В случае попытки повторного бронирования одного и того же перелета для данного пассажира, т.е. ввода строки с дубликатом первичного ключа, такая строка  
будет отвергнута, и будет сгенерировано сообщение об ошибке. В таком случае и таблица **tickets\_directions** не будет обновлена.

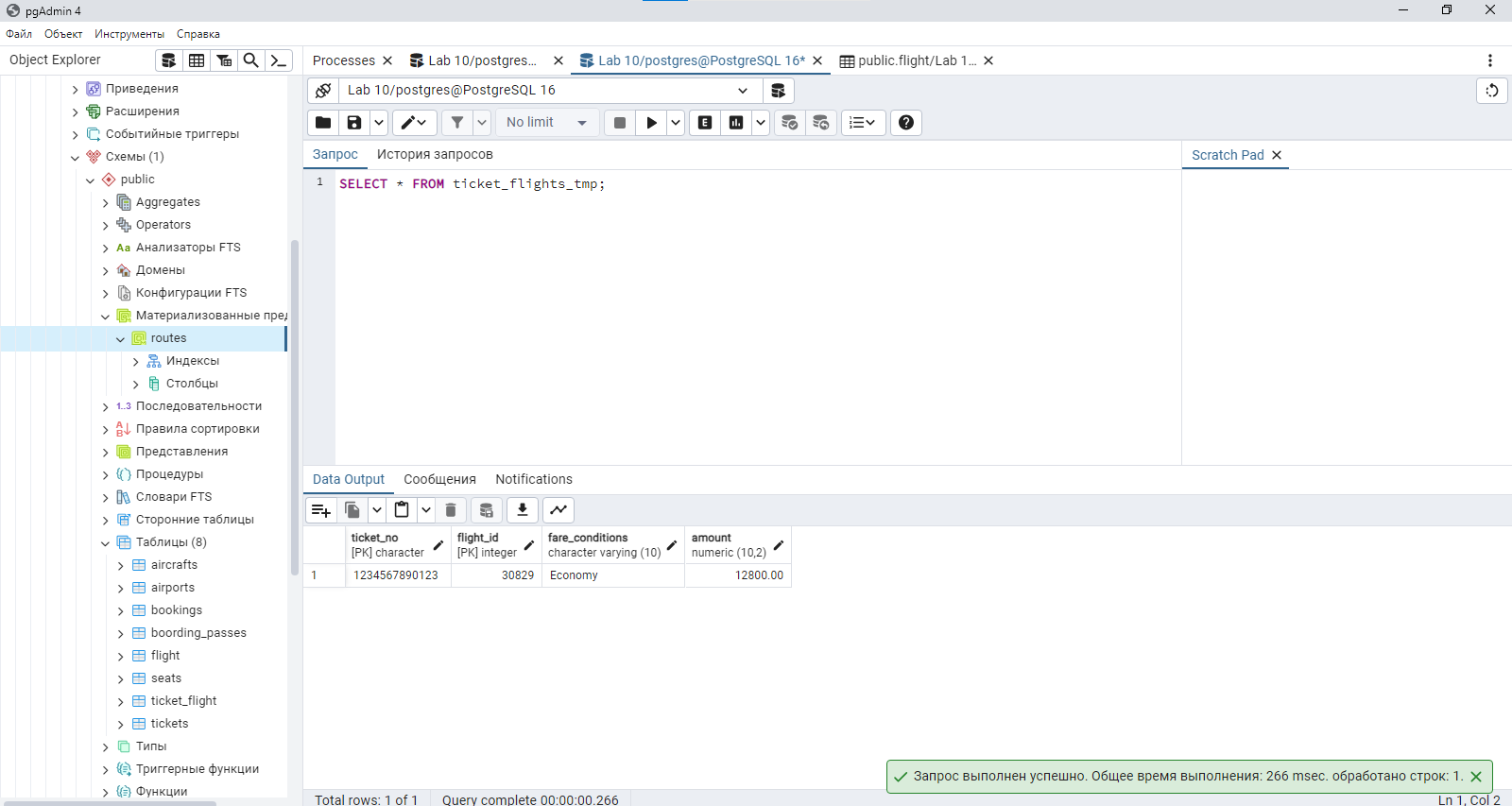
Давайте посмотрим, что изменилось в таблице tickets\_directions.

**SELECT departure\_city AS dep\_city,  
 arrival\_city AS arr\_city,  
 last\_ticket\_time,  
 tickets\_num AS num  
FROM tickets\_directions  
WHERE tickets\_num > 0  
ORDER BY departure\_city, arrival\_city;**

По маршруту Москва — Сочи наш пассажир приобретал два билета, что и отражено в выборке (*пример из тестовой базы данных*).

dep\_city | arr\_city | last\_ticket\_time | num  
------------+------------+----------------------------+-----  
Красноярск | Москва | 2017-02-04 14:02:23.769443 | 1  
Москва | Сочи | 2017-02-04 14:02:23.769443 | 2  
Сочи | Красноярск | 2017-02-04 14:02:23.769443 | 1  
Сочи | Москва | 2017-02-04 14:02:23.769443 | 1  
(4 строки)

А это информация о каждом перелете, забронированном нашим пассажиром:

**SELECT \* FROM ticket\_flights\_tmp;**

ticket\_no | flight\_id | fare\_conditions | amount  
--------------+-----------+-----------------+----------  
1234567890123 | 13829 | Economy | 10500.00  
1234567890123 | 4728 | Economy | 3400.00  
1234567890123 | 30523 | Economy | 3400.00  
1234567890123 | 7757 | Economy | 3400.00  
1234567890123 | 30829 | Economy | 12800.00  
(5 строк)

**Задание 8.**

Предположим, что руководство авиакомпании решило отказаться от использования самолетов компаний **Boeing** и **Airbus**, имеющих наименьшее количество пассажирских мест в салонах.

Мы должны соответствующим образом откорректировать таблицу «**Самолеты**» (**aircrafts\_tmp**).

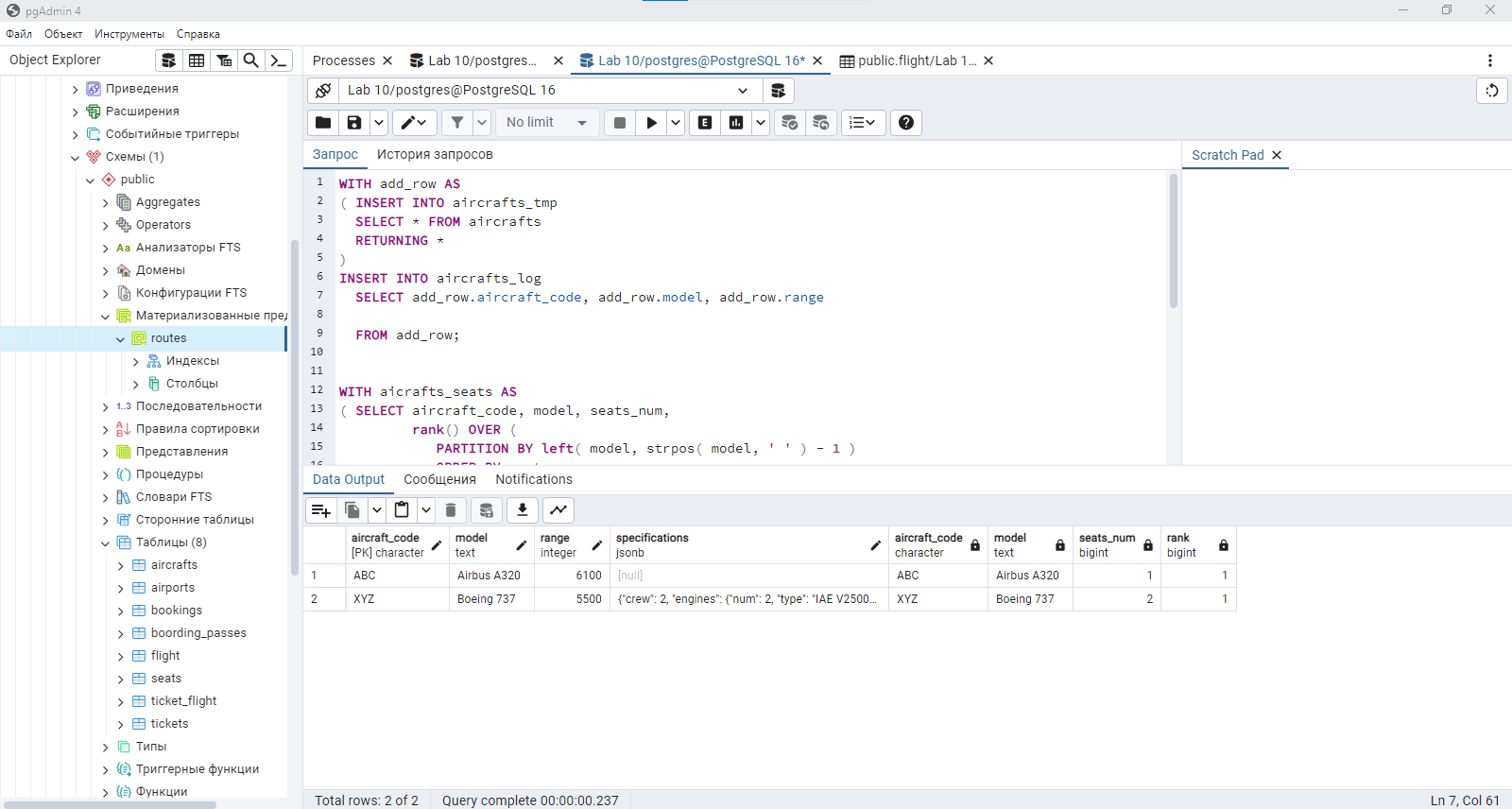
Предлагается следующий алгоритм.

**Шаг 1**. Для каждой модели вычислить общее число мест в салоне.

**Шаг 2**. Используя оконную функцию rank, присвоить моделям ранги на основе  
числа мест (упорядочив их по возрастанию числа мест). Ранжирование выполняется *в пределах каждой компании-производителя*, т.е. для **Boeing** и для **Airbus** — отдельно. Ранг, равный 1, соответствует наименьшему числу мест.

**Шаг 3**. Выполнить удаление тех строк из таблицы **aircrafts\_tmp**, которые удовлетворяют следующим требованиям: модель — **Boeing** или **Airbus**, а число мест  
в салоне — минимальное из всех моделей данной компании-производителя, т.е. модель имеет ранг, равный 1.

**WITH aicrafts\_seats AS  
( SELECT aircraft\_code, model, seats\_num,  
 rank() OVER (  
 PARTITION BY left( model, strpos( model, ' ' ) - 1 )  
 ORDER BY seats\_num  
)  
 FROM  
 ( SELECT a.aircraft\_code, a.model, count( \* ) AS seats\_num  
 FROM aircrafts\_tmp a, seats s  
 WHERE a.aircraft\_code = s.aircraft\_code  
 GROUP BY 1, 2  
 ) AS seats\_numbers  
)  
DELETE FROM aircrafts\_tmp a  
 USING aicrafts\_seats a\_s  
 WHERE a.aircraft\_code = a\_s.aircraft\_code  
 AND left( a.model, strpos( a.model, ' ' ) - 1 )  
 IN ( 'Boeing', 'Airbus' )  
 AND a\_s.rank = 1  
RETURNING \*;**



**Шаг 1** выполняется в подзапросе в предложении WITH.

**Шаг 2** — в главном запросе в предложении WITH.

**Шаг 3** реализуется командой DELETE.

Обратите внимание, что название компании-производителя мы определяем  
путем взятия подстроки от значения атрибута **mode**l: от начала строки до пробельного символа (используем функции **left** и **strpos**).

Мы включили предложение **RETURNING \*,** чтобы увидеть, какие именно модели были удалены.

Предложение **WITH** выдает такой результат (*пример из тестовой таблицы*):

aircraft\_code | model | seats\_num | rank  
--------------+---------------------+-----------+------  
319 | Airbus A319-100 | 116 | 1  
320 | Airbus A320-200 | 140 | 2  
321 | Airbus A321-200 | 170 | 3  
733 | Boeing 737-300 | 130 | 1  
763 | Boeing 767-300 | 222 | 2  
773 | Boeing 777-300 | 402 | 3  
CR2 | Bombardier CRJ-200 | 50 | 1  
CN1 | Cessna 208 Caravan | 12 | 1  
SU9 | Sukhoi SuperJet-100| 97 | 1  
(9 строк)

Очевидно, что должны быть удалены модели с кодами **319** и **733**.

После выполнения запроса получим (это работает предложение **RETURNING \*):**

-[ RECORD 1 ]--+----------------  
aircraft\_code | 319  
model | Airbus A319-100  
range | 6700  
aircraft\_code | 319  
model | Airbus A319-100  
seats\_num | 116  
rank | 1  
-[ RECORD 2 ]--+----------------  
aircraft\_code | 733  
model | Boeing 737-300  
range | 4200  
aircraft\_code | 733  
model | Boeing 737-300  
seats\_num | 130  
rank | 1  
DELETE 2

Обратите внимание, что в результате были выведены комбинированные строки, полученные при соединении таблицы **aircrafts\_tmp** с временной таблицей **aicrafts\_seats**, указанной в предложении **USING**. Но удалены были, конечно, строки из таблицы **aircrafts\_tmp.**

**Задание 9.**

В реальной работе иногда возникают ситуации, когда требуется быстро заполнить таблицу тестовыми данными. В таком случае удобно воспользоваться  
командой **INSERT** с подзапросом. Конечно, число атрибутов и их типы данных  
в подзапросе **SELECT** должны быть такими, какие ожидает получить команда  
**INSERT**.

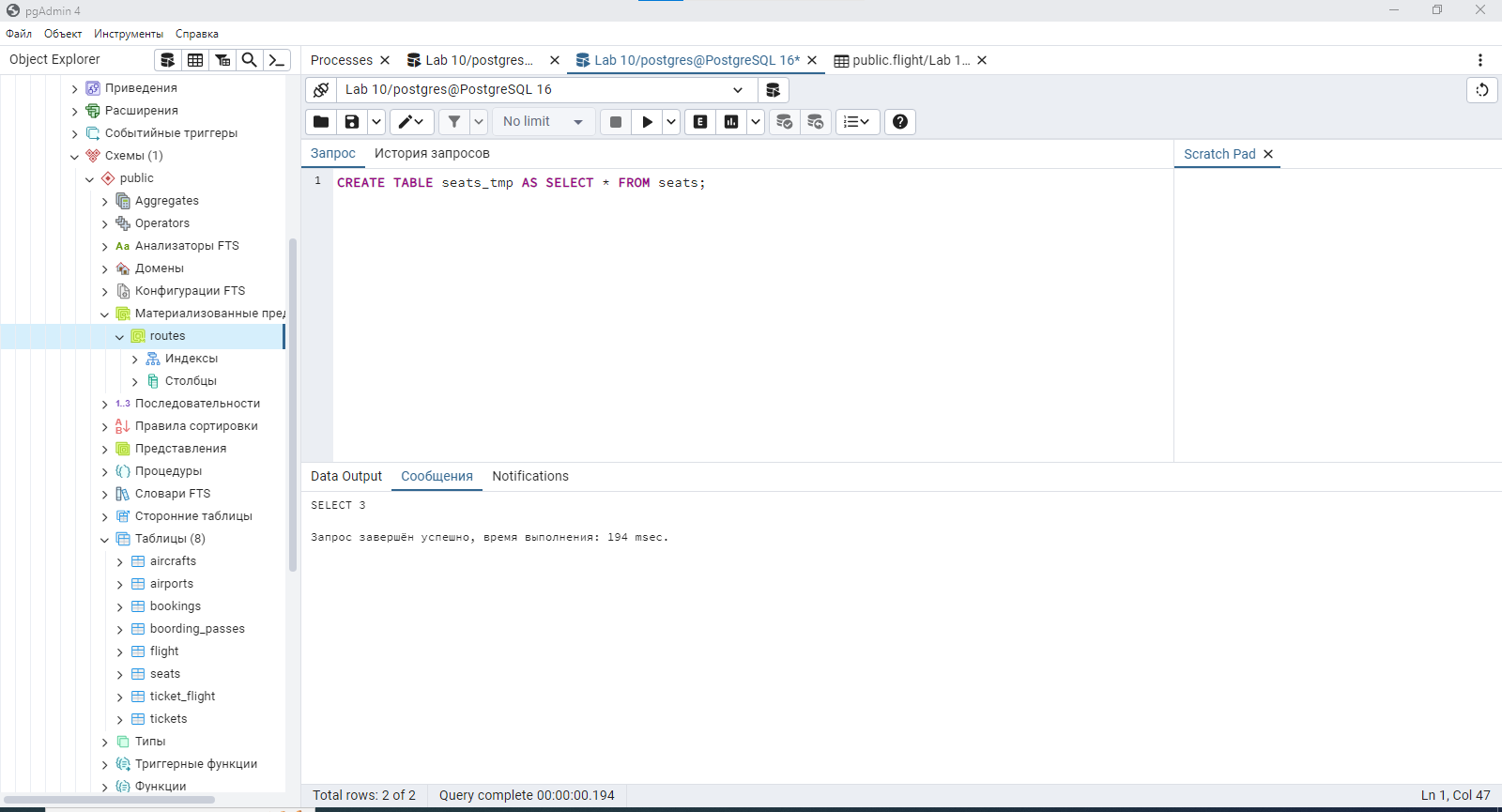
Продемонстрируем такой прием на примере таблицы «**Места**» (**seats**).

Для того чтобы выполнить команду, приведенную в этом упражнении, нужно либо сначала удалить все строки из таблицы seats, чтобы можно было добавлять строки в эту таблицу:

**DELETE FROM seats;**

либо создать копию этой таблицы

**CREATE TABLE seats\_tmp AS SELECT \* FROM seats;**

****чтобы работать с копией.

Как сформировать тестовые данные автоматическим способом? Для этого  
сначала нужно подготовить исходные данные, на основе которых и будут формироваться результирующие значения для вставки в таблицу «**Места**».

В рамках реляционной модели наиболее естественным будет представление исходных данных в виде таблиц. Для формирования каждой строки таблицы «**Места**» нужно задать код модели самолета, класс обслуживания и номер места, который состоит из двух компонентов: номера ряда и буквенного идентификатора позиции в ряду.

Поскольку размеры и компоновки салонов различаются, необходимо для каждой модели указать предельное число рядов кресел в салонах бизнес-класса и  
экономического класса, а также число кресел в каждом ряду. Это число можно  
задать с помощью указания буквенного идентификатора для самого последнего кресла в ряду.

Например, если в ряду всего шесть кресел, тогда их буквенные обозначения будут такими: A, B, C, D, E, F. Таким образом, последней будет буква F. В салоне бизнес-класса число мест в ряду меньше, чем в салоне экономического класса, но для упрощения задачи примем эти числа одинаковыми.

В результате получим первую исходную таблицу с атрибутами:

– код модели самолета;– номер последнего ряда кресел в салоне бизнес-класса;

– номер последнего ряда кресел в салоне экономического класса;

– буква, обозначающая позицию последнего кресла в ряду.

Классы обслуживания также поместим в отдельную таблицу. В ней будет всего один атрибут — класс обслуживания.

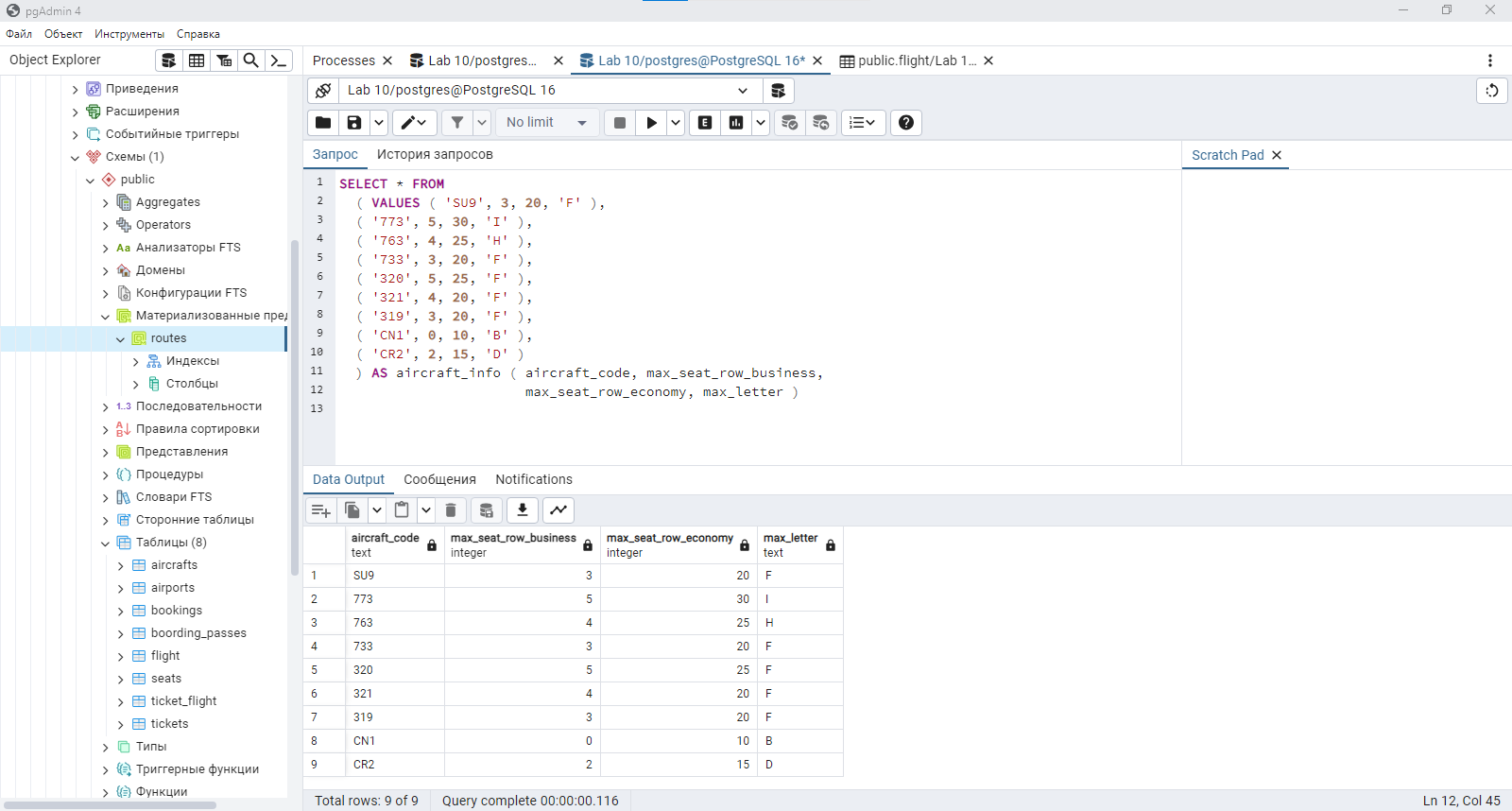
Список номеров рядов также поместим в отдельную таблицу. В ней будет также  
всего один атрибут — номер ряда.

Так же поступим и с буквенными обозначениями кресел в ряду. В этой таблице  
будет один атрибут — латинская буква, обозначающая позицию кресла.

В принципе можно было бы создать все четыре таблицы с помощью команды  
**CREATE TABLE** и ввести в них исходные данные, а затем использовать эти таблицы в команде **SELECT**. Но команда *SELECT* позволяет использовать в предложении **FROM** виртуальные таблицы, которые можно создавать с помощью предложения **VALUES**. Для этого непосредственно в текст команды записываются группы значений, представляющие собой строки такой виртуальной таблицы.

Каждая такая строка заключается в круглые скобки. Вся эта конструкция получает имя таблицы, и к ней прилагается список атрибутов. Это выглядит, например,  
следующим образом:

**FROM  
 ( VALUES ( 'SU9', 3, 20, 'F' ),  
 ( '773', 5, 30, 'I' ),  
 ( '763', 4, 25, 'H' ),  
 ( '733', 3, 20, 'F' ),  
 ( '320', 5, 25, 'F' ),  
 ( '321', 4, 20, 'F' ),  
 ( '319', 3, 20, 'F' ),  
 ( 'CN1', 0, 10, 'B' ),  
 ( 'CR2', 2, 15, 'D' )  
 ) AS aircraft\_info ( aircraft\_code, max\_seat\_row\_business,  
 max\_seat\_row\_economy, max\_letter )**



Здесь **aircraft\_info** определяет имя виртуальной таблицы, а список идентификаторов — имена ее атрибутов (aircraft\_code, max\_seat\_row\_business, max\_seat\_row\_economy, max\_letter).

Эти атрибуты можно использовать во всех частях команды **SELECT**, как если бы это были атрибуты обычной таблицы.

Остальные виртуальные таблицы создаются аналогичным способом.

Для соединения таблиц используется ключевое слово **CROSS JOIN**, хотя в данном случае вместо этого можно было просто поставить запятые.

Как это и бывает всегда, четыре таблицы образуют декартово произведение из своих строк, а затем на основе условия **WHERE** «лишние» строки отбрасываются. В этом условии используется условный оператор **CASE**. Он позволяет нам поставить допустимый номер ряда в зависимость от класса обслуживания:

**WHERE  
 CASE WHEN fare\_condition = 'Business'  
 THEN seat\_row::integer <= max\_seat\_row\_business  
 WHEN fare\_condition = 'Economy'  
 THEN seat\_row::integer > max\_seat\_row\_business  
 AND seat\_row::integer <= max\_seat\_row\_economy**

В этом выражении используется приведение типов: **seat\_row::integer**.

Эта операция необходима, т. к. в виртуальной таблице номера рядов представлены  
в виде символьных строк, а для выполнения сравнения числовых значений в  
данной ситуации нужен целый тип. При написании условного оператора нужно  
учесть, что в виртуальной таблице мы указали не количество рядов в бизнес-классе и экономическом классе, а номера *последних* рядов в этих классах. Поэтому возникает конструкция

**THEN seat\_row::integer > max\_seat\_row\_business  
 AND seat\_row::integer <= max\_seat\_row\_economy**

Также проверяем еще одно условие, сравнивая символьные строки:

**AND letter <= max\_letter;**

Последний этап в работе оператора **SELECT** — это формирование списка выражений, которые будут выведены в качестве итоговых данных. Для формирования номера места используется операция конкатенации ||, которая соединяет номер ряда с буквенным обозначением позиции в ряду.

**SELECT aircraft\_code, seat\_row || letter, fare\_condition**

Итак, SQL-команда, которая позволит за одну операцию ввести в таблицу «**Места**» сразу необходимое число строк, выглядит так:

**INSERT INTO seats ( aircraft\_code, seat\_no, fare\_conditions )  
 SELECT aircraft\_code, seat\_row || letter, fare\_condition  
 FROM  
 -- компоновки салонов  
 ( VALUES ( 'SU9', 3, 20, 'F' ),  
 ( '773', 5, 30, 'I' ),  
 ( '763', 4, 25, 'H' ),  
 ( '733', 3, 20, 'F' ),  
 ( '320', 5, 25, 'F' ),  
 ( '321', 4, 20, 'F' ),  
 ( '319', 3, 20, 'F' ),  
 ( 'CN1', 0, 10, 'B' ),  
 ( 'CR2', 2, 15, 'D' )  
 ) AS aircraft\_info ( aircraft\_code, max\_seat\_row\_business,  
 max\_seat\_row\_economy, max\_letter )  
CROSS JOIN  
 -- классы обслуживания  
 ( VALUES ( 'Business' ), ( 'Economy' )  
 ) AS fare\_conditions (  
 fare\_condition )  
CROSS JOIN  
 -- список номеров рядов кресел  
 ( VALUES ( '1' ), ( '2' ), ( '3' ), ( '4' ), ( '5' ),  
 ( '6' ), ( '7' ), ( '8' ), ( '9' ), ( '10' ),  
 ( '11' ), ( '12' ), ( '13' ), ( '14' ), ( '15' ),  
 ( '16' ), ( '17' ), ( '18' ), ( '19' ), ( '20' ),  
 ( '21' ), ( '22' ), ( '23' ), ( '24' ), ( '25' ),  
 ( '26' ), ( '27' ), ( '28' ), ( '29' ), ( '30' )  
 ) AS seat\_rows ( seat\_row )  
CROSS JOIN  
 -- список номеров (позиций) кресел в ряду  
 ( VALUES ( 'A' ), ( 'B' ), ( 'C' ), ( 'D' ), ( 'E' ),  
 ( 'F' ), ( 'G' ), ( 'H' ), ( 'I' )  
 ) AS letters ( letter )  
WHERE  
 CASE WHEN fare\_condition = 'Business'  
 THEN seat\_row::integer <= max\_seat\_row\_business  
 WHEN fare\_condition = 'Economy'  
 THEN seat\_row::integer > max\_seat\_row\_business  
 AND seat\_row::integer <= max\_seat\_row\_economy  
 END  
AND letter <= max\_letter;**

Выводы: в ходе лабораторной работы я получить практические навыки в формировании SQL-запросов на вставку, изменение и удаление строк в таблицах СУБД PostgreSQL.