|  |  |
| --- | --- |
| Gerb-BMSTU_01 | **Министерство науки и высшего образования Российской Федерации**  **Федеральное государственное бюджетное образовательное учреждение**  **высшего образования**  **«Московский государственный технический университет**  **имени Н.Э. Баумана**  **(национальный исследовательский университет)»**  **(МГТУ им. Н.Э. Баумана)** |

ФАКУЛЬТЕТ **Информатика и системы управления**

КАФЕДРА **Компьютерные системы и сети (ИУ6)**

НАПРАВЛЕНИЕ ПОДГОТОВКИ **09.04.01 Информатика и вычислительная техника**

МАГИСТЕРСКАЯ ПРОГРАММА **09.04.01/07 Интеллектуальные системы анализа,**

**обработки и интерпретации больших данных**

**Отчет**

**по лабораторной работе № 3**

**Название:** Запросы PostgreSQL

**Дисциплина:** Технологии параллельных систем баз данных

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Студент | ИУ6-12М |  |  | Д.С. Каткова |
|  | (Группа) |  | (Подпись, дата) | (И.О. Фамилия) |
|  |  |  |  |  |
| Преподаватель |  |  |  | А.Д. Пономарев |
|  |  |  | (Подпись, дата) | (И.О. Фамилия) |

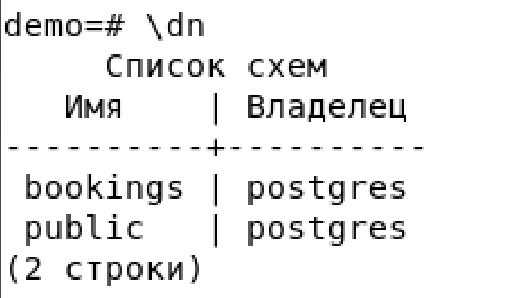
Москва, 2023

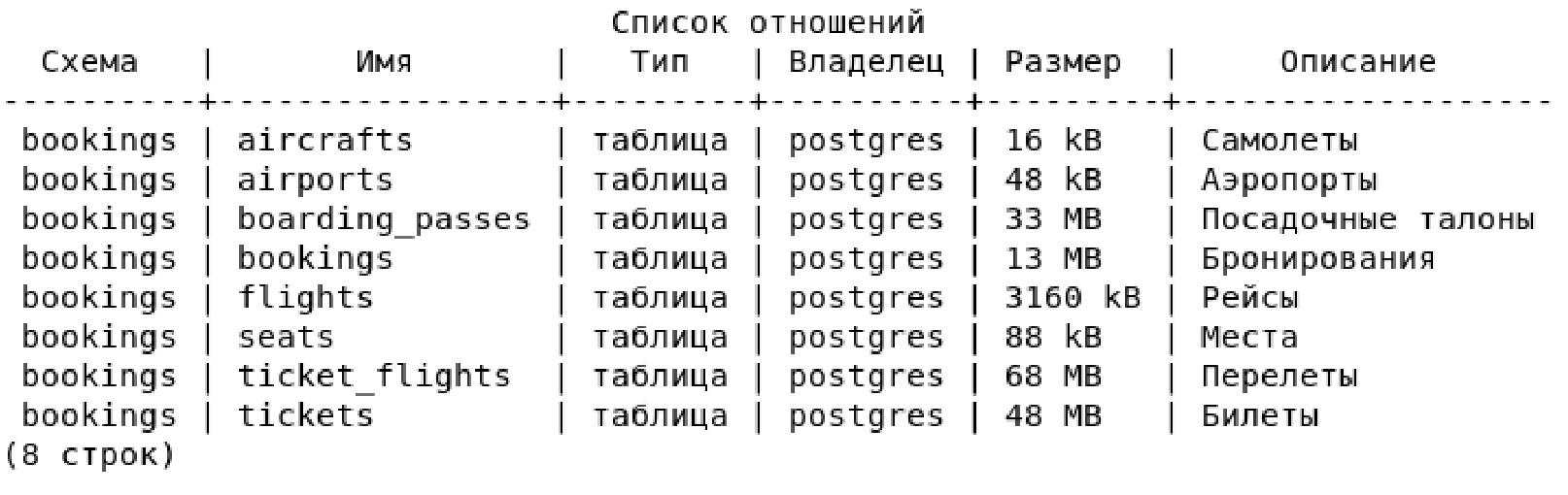
**1 Цель лабораторной работы**

Целью лабораторной работы является формирование следующей компетенции: студент должен научиться разрабатывать сложные запросы SELECT.

**2 Подключение к виртуальной машине и создание БД**

После подключения к виртуальной машине и создания БД можно посмотреть список всех схем и список всех таблиц.

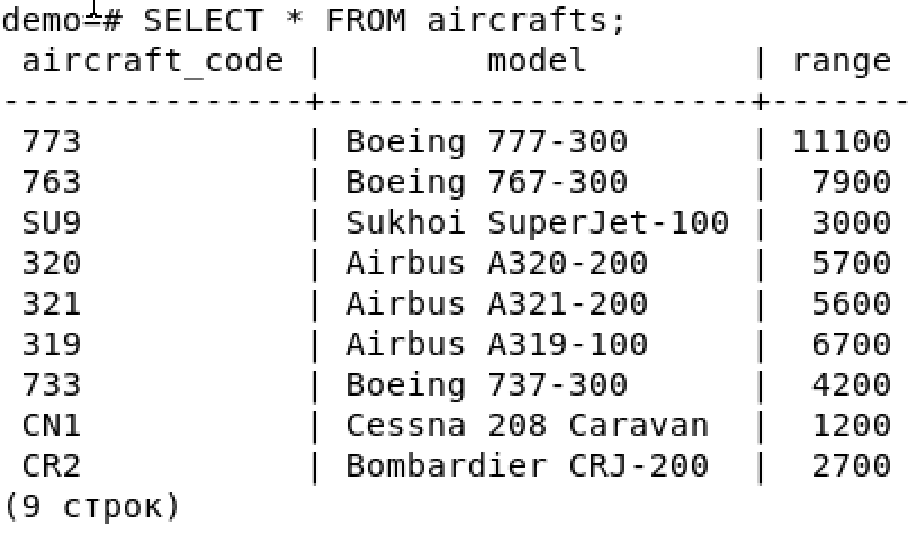




**3 Запросы**

Прежде чем перейти к конкретным запросам, необходимо ознакомиться с содержимым двух таблиц – «Самолеты» и «Аэропорты». Таблица «Самолеты» совсем маленькая, а таблица «Аэропорты» содержит чуть больше ста строк.

*SELECT \* FROM aircrafts;*

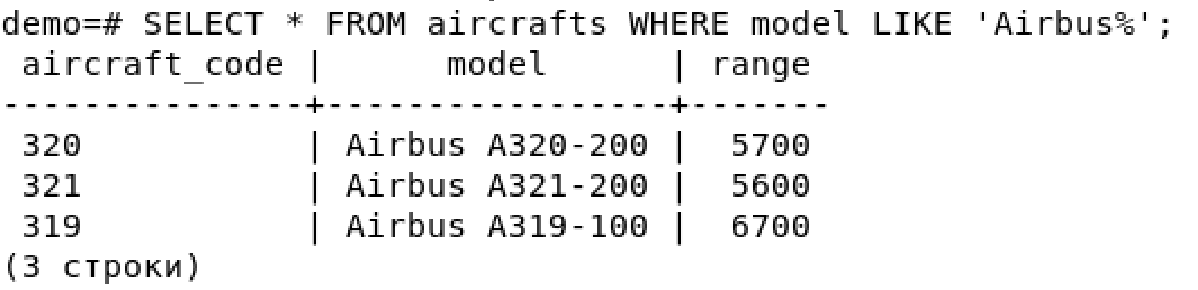


*SELECT \* FROM airports;*



Начнем с различных условий отбора строк в предложении WHERE. Для начала поставим перед собой такую задачу: выбрать все самолеты компании Airbus. В этом нам поможет оператор поиска шаблонов LIKE.

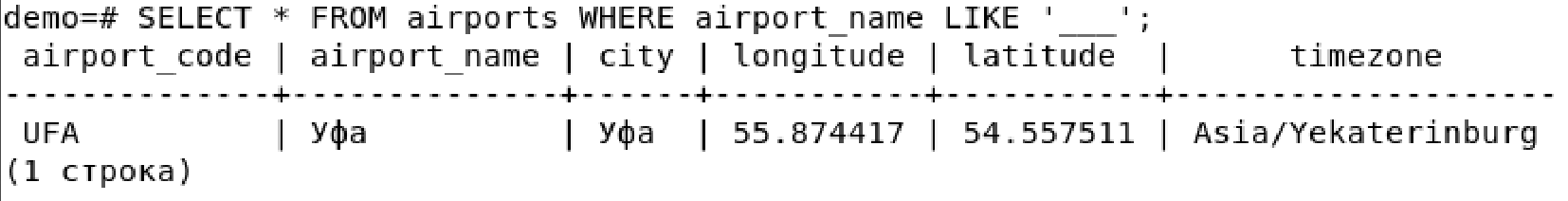
*SELECT \* FROM aircrafts WHERE model LIKE 'Airbus%';*

**

Вместо символа «%» могут быть подставлены любые символы в любом количестве, а может и не быть подставлено ни одного символа.

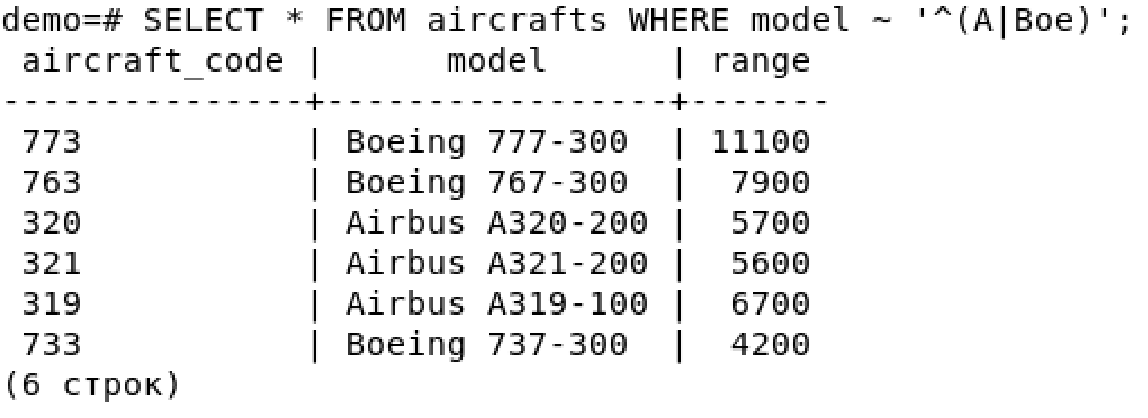
Кроме символа «%» в шаблоне может использоваться и символ подчеркивания \_ «\_»,который соответствует в точности одному любому символу. В качестве примера найдем в таблице «Аэропорты» те из них, которые имеют названия длиной три символа (буквы). С этой целью зададим в качестве шаблона строку, состоящую из трех символов «\_».

*SELECT \* FROM airports WHERE airport\_name LIKE '\_\_\_';*

**

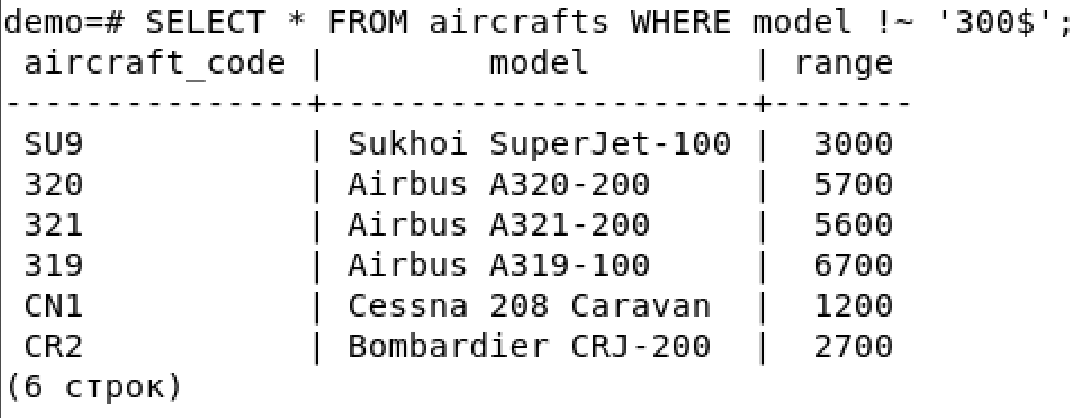
Существует ряд операторов для работы с регулярными выражениями POSIX. Эти операторы имеют больше возможностей, чем оператор LIKE. Для того чтобы выбрать, например, самолеты компаний Airbus и Boeing, можно сделать так:

*SELECT \* FROM aircrafts WHERE model ~ '^(A|Boe)';*

**

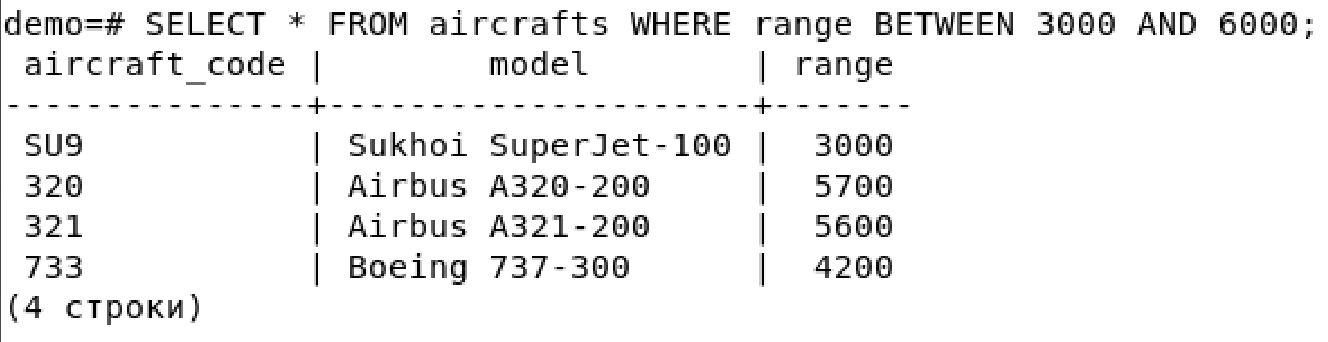
Для инвертирования смысла оператора ~ нужно перед ним добавить знак «!». В качестве примера отыщем модели самолетов, которые не завершаются числом 300.

*SELECT \* FROM aircrafts WHERE model !~ '300$';*



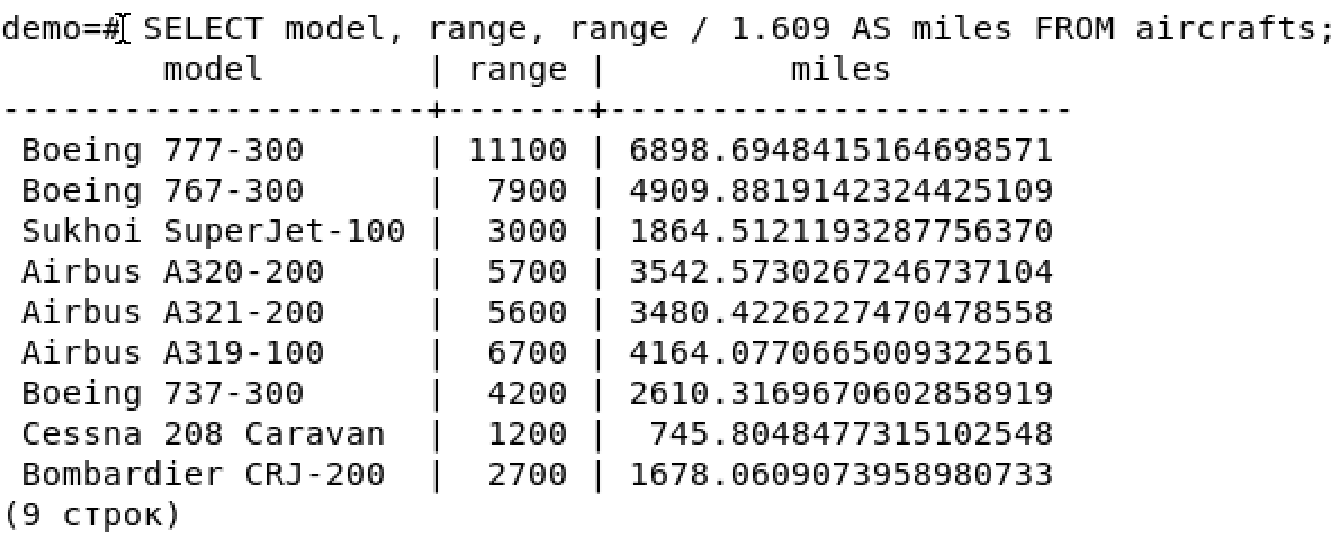
В качестве замены традиционных операторов сравнения могут использоваться предикаты сравнения, которые ведут себя так же, как и операторы, но имеют другой синтаксис. Давайте ответим на вопрос: какие самолеты имеют дальность полета в диапазоне от 3 000 км до 6 000 км? Ответ получим с помощью предиката BETWEEN.

*SELECT \* FROM aircrafts WHERE range BETWEEN 3000 AND 6000;*



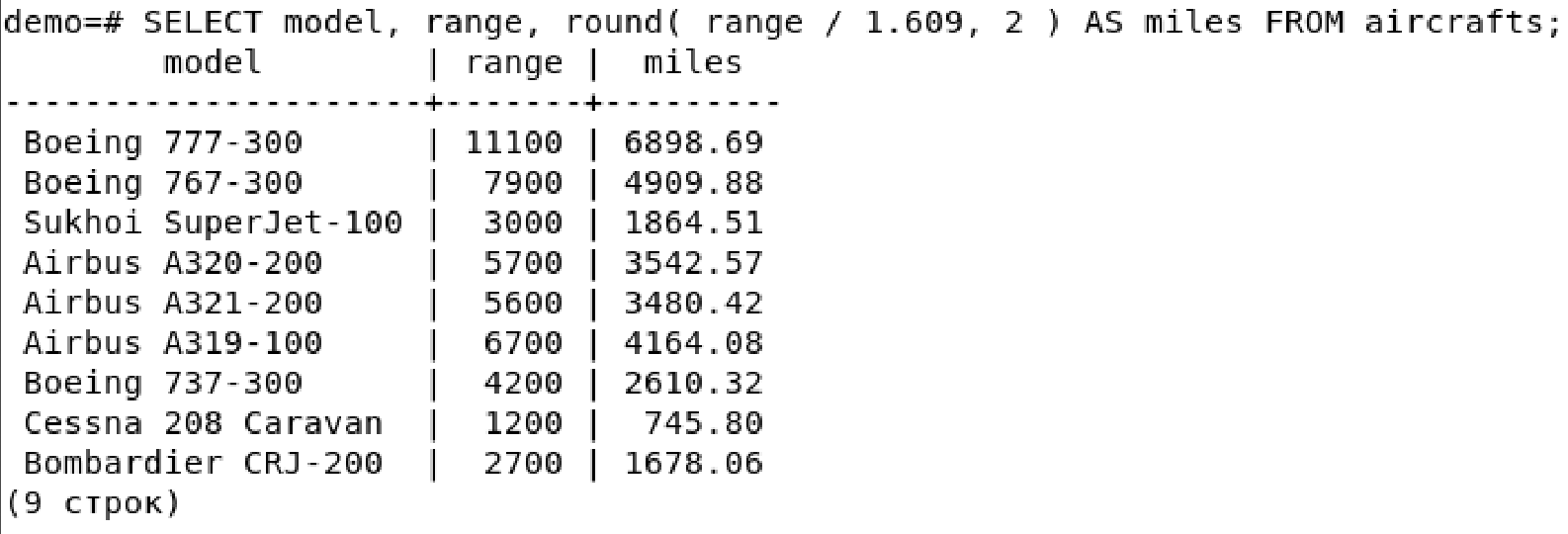
При выборке данных можно проводить вычисления и получать в результирующей таблице вычисляемые столбцы. Если мы захотим представить дальность полета не только в километрах, но и в милях, то нужно вычислить это выражение и для удобства присвоить новому столбцу псевдоним с помощью ключевого слова AS.

*SELECT model, range, range / 1.609 AS miles FROM aircrafts;*



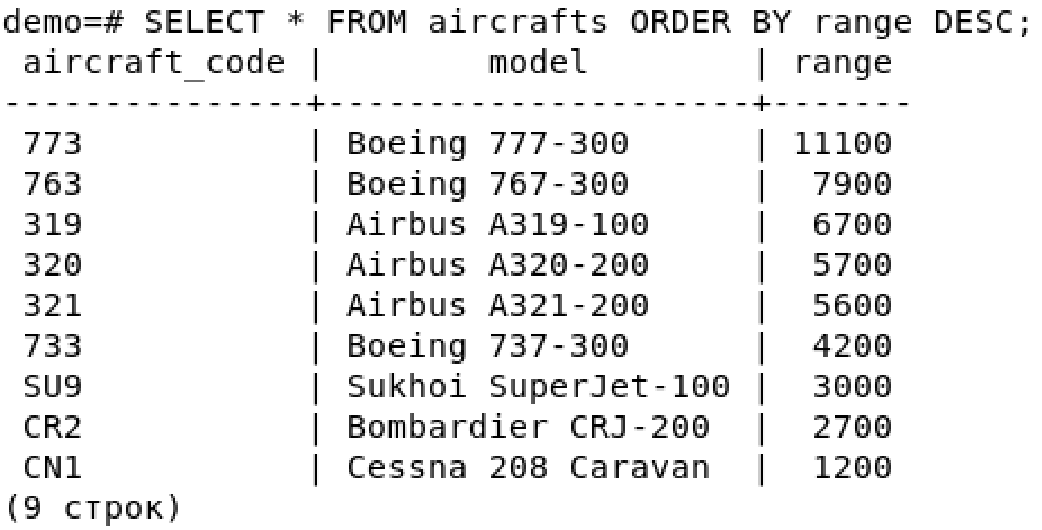
По всей вероятности, такая высокая точность представления значений в милях не требуется, поэтому мы можем уменьшить ее до разумного предела в два десятичных знака:

*SELECT model, range, round( range / 1.609, 2 ) AS miles FROM aircrafts;*



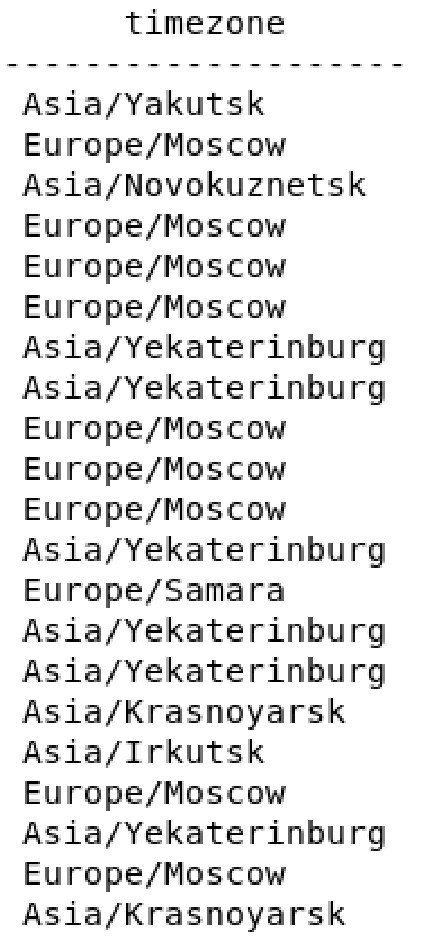
Теперь обратимся к такому вопросу, как упорядочение строк при выводе. Если не принять специальных мер, то СУБД не гарантирует никакого конкретного порядка строк в результирующей выборке. Для упорядочения строк служит предложение ORDER BY. Например, если мы захотим разместить самолеты в порядке убывания дальности их полета, то нужно сделать так:

*SELECT \* FROM aircrafts ORDER BY range DESC;*



Если сделать традиционную выборку, то мы получим список значений, среди которых будет много повторяющихся.

*SELECT timezone FROM airports;*

**

Конечно, это неудобно. Для того чтобы оставить в выборке только неповторяющиеся значения, служит ключевое слово DISTINCT:

*SELECT DISTINCT timezone FROM airports ORDER BY 1;*

**

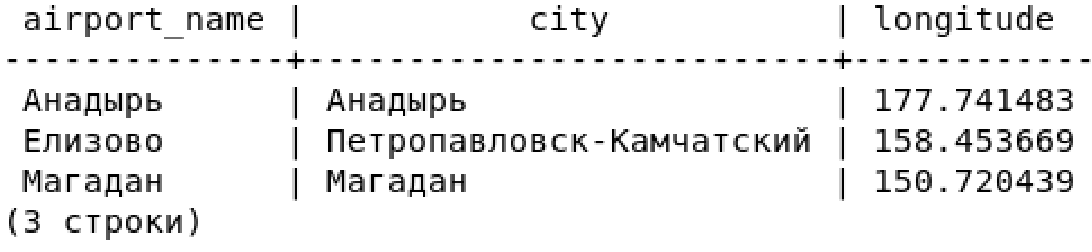
В таблице «Аэропорты» более ста записей. Если мы поставим задачу найти три самых восточных аэропорта, то для ее решения подошел бы такой запрос

*SELECT airport\_name, city, longitude*

*FROM airports*

*ORDER BY longitude DESC*

*LIMIT 3;*

**

А как найти еще три аэропорта, которые находятся немного западнее первой тройки, т. е. занимают места с четвертого по шестое? Для пропуска строк служит предложение OFFSET:

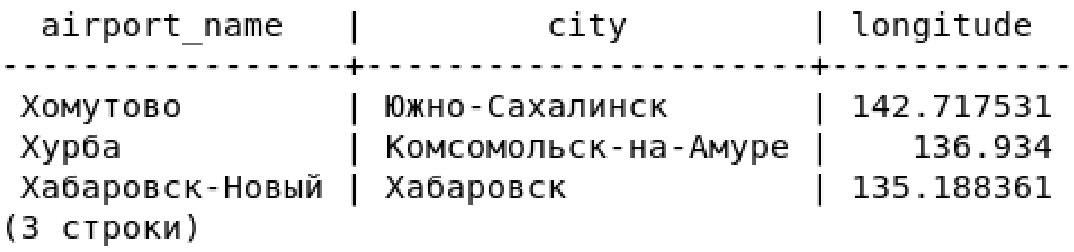
*SELECT airport\_name, city, longitude*

*FROM airports*

*ORDER BY longitude DESC*

*LIMIT 3*

*OFFSET 3;*

**

В таблице «Самолеты» есть столбец «Максимальная дальность полета» (range). Мы можем дополнить вывод данных из этой таблицы столбцом «Класс самолета», имея в виду принадлежность каждого самолета к классу дальнемагистральных, среднемагистральных или ближнемагистральных судов. Воспользовавшись приведенной в указаниях к работе конструкцией в предложении SELECT и назначив новому столбцу имя с помощью ключевого слова AS, получим следующий запрос:

*SELECT model, range,*

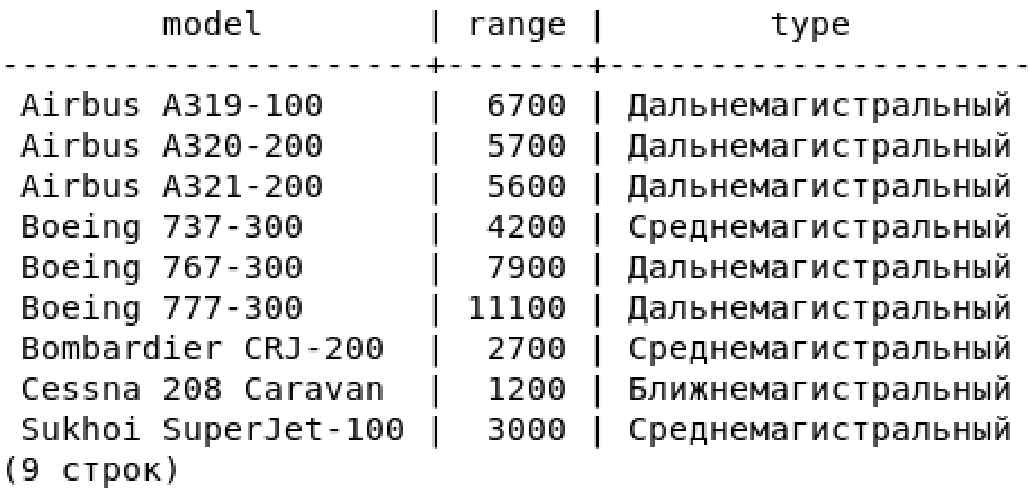
*CASE WHEN range < 2000 THEN 'Ближнемагистральный'*

*WHEN range < 5000 THEN 'Среднемагистральный'*

*ELSE 'Дальнемагистральный' END AS type*

*FROM aircrafts*

*ORDER BY model;*



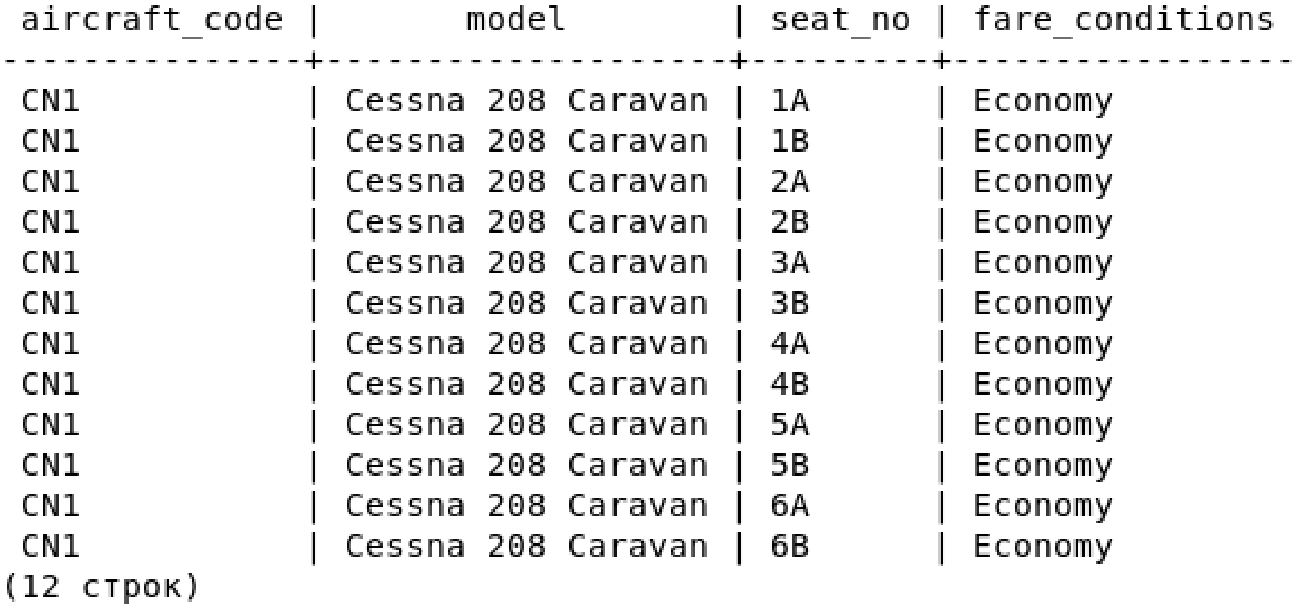
В тех случаях, когда информации, содержащейся в одной таблице, недостаточно для получения требуемого результата, используют соединение (join) таблиц. Покажем способ выполнения соединения на примере следующего запроса: выбрать все места, предусмотренные компоновкой салона самолета Cessna 208 Caravan:

*SELECT a.aircraft\_code, a.model, s.seat\_no, s.fare\_conditions*

*FROM seats AS s*

*JOIN aircrafts AS a ON s.aircraft\_code = a.aircraft\_code*

*WHERE a.model ~ '^Cessna' ORDER BY s.seat\_no;*



Теперь обратимся к так называемым внешним соединениям. Зададимся вопросом: сколько маршрутов обслуживают самолеты каждого типа? Если не требовать вывода наименований моделей самолетов, тогда всю необходимую информацию можно получить из материализованного представления «Маршруты» (routes). Но мы все же будем выводить и наименования моделей, поэтому обратимся также к таблице «Самолеты» (aircrafts). Соединим эти таблицы на основе атрибута aircraft\_code, сгруппируем строки и просто воспользуемся функцией count. В этом запросе внешнее соединение еще не используется.

*SELECT r.aircraft\_code, a.model, count( \* ) AS num\_routes*

*FROM routes r*

*JOIN aircrafts a ON r.aircraft\_code = a.aircraft\_code*

*GROUP BY 1, 2*

*ORDER BY 3 DESC;*

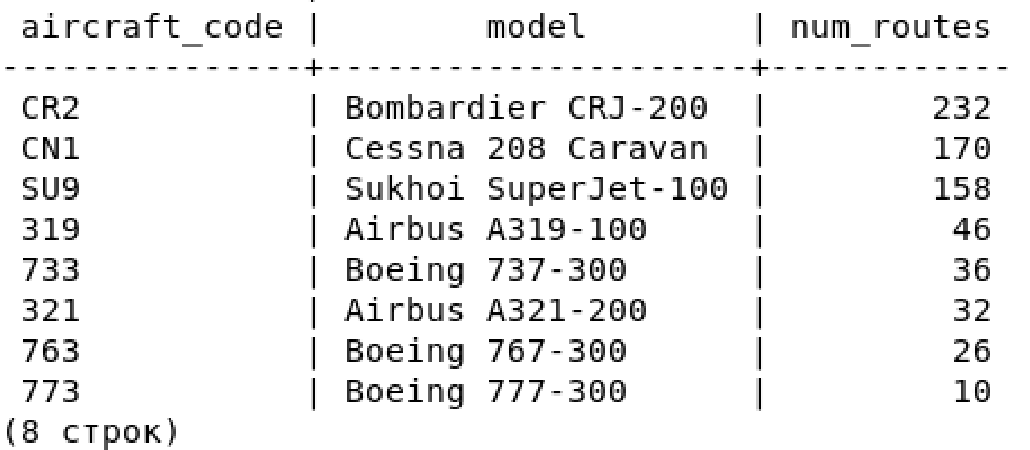
**

Таблица «Самолеты» содержит 9 моделей, а в этой выборке лишь 8 строк. Значит, какая-то модель самолета не участвует в выполнении рейсов. Как ее выявить? С помощью такого запроса:

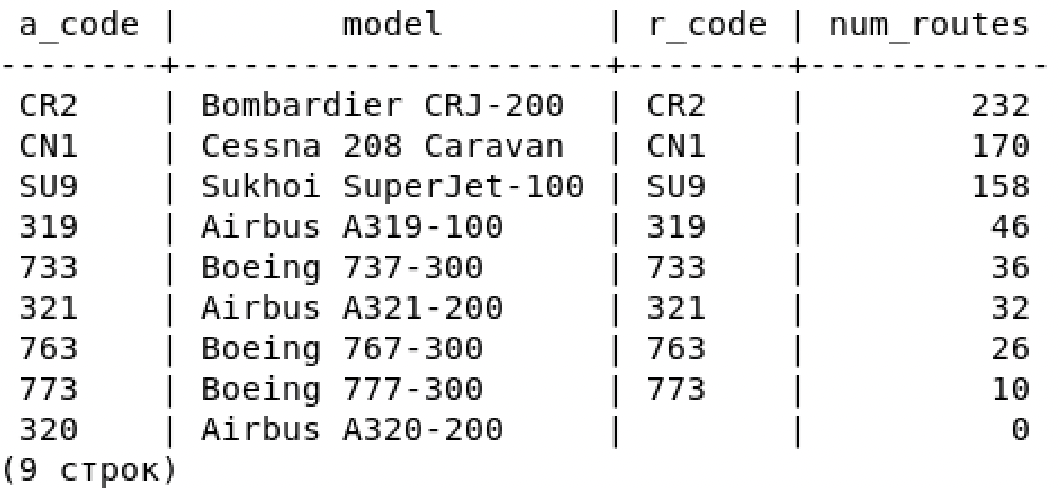
*SELECT a.aircraft\_code AS a\_code, a.model, r.aircraft\_code AS r\_code, count( r.aircraft\_code ) AS num\_routes*

*FROM aircrafts a*

*LEFT OUTER JOIN routes r ON r.aircraft\_code = a.aircraft\_code*

*GROUP BY 1, 2, 3*

*ORDER BY 4 DESC;*

**

В практической работе при выполнении выборок зачастую выполняются многотабличные запросы, включающие три таблицы и более. В качестве примера рассмотрим такую задачу: определить число пассажиров, не пришедших на регистрацию билетов и, следовательно, не вылетевших в пункт назначения. Будем учитывать только рейсы, у которых фактическое время вылета не пустое, т. е. рейсы, имеющие статус Departed или Arrived.

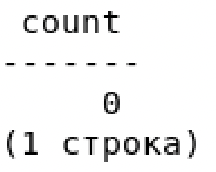
*SELECT count( \* )*

*FROM ( ticket\_flights t*

*JOIN flights f ON t.flight\_id = f.flight\_id )*

*LEFT OUTER JOIN boarding\_passes b ON t.ticket\_no = b.ticket\_no AND t.flight\_id = b.flight\_id*

*WHERE f.actual\_departure IS NOT NULL AND b.flight\_id IS NULL;*



Оказывается, таких пассажиров нет. Вопрос: в какие города можно улететь либо из Москвы, либо из СанктПетербурга?

*SELECT arrival\_city*

*FROM routes WHERE departure\_city = 'Москва'*

*UNION*

*SELECT arrival\_city*

*FROM routes*

*WHERE departure\_city = 'Санкт-Петербург'*

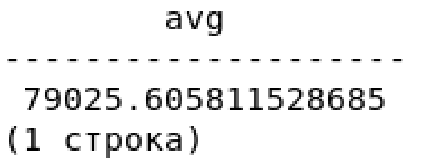
*ORDER BY arrival\_city;*

**

Строки-дубликаты в результирующее множество не включаются. Для их включения нужно использовать UNION ALL.

Для расчета среднего значения по столбцу используется функция avg (от слова average).

*SELECT avg( total\_amount ) FROM bookings;*



Рассмотрим два примера. Первый будет таким: давайте подсчитаем, сколько маршрутов предусмотрено из Москвы в другие города. При формировании запроса не будем учитывать частоту рейсов в неделю, т. е. независимо от того, выполняется какой-то рейс один раз в неделю или семь раз, он учитывается только однократно. Воспользуемся материализованным представлением «Маршруты».

*SELECT arrival\_city, count( \* )*

*FROM routes*

*WHERE departure\_city = 'Москва'*

*GROUP BY arrival\_city*

*ORDER BY count DESC;*

**

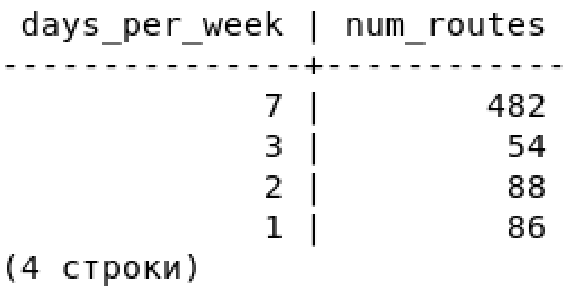
В качестве второго примера рассмотрим ситуацию, когда руководству компании потребовалась обобщенная информация по частоте выполнения рейсов, а именно: сколько рейсов выполняется ежедневно, сколько рейсов — шесть дней в неделю, пять и т. д. Опять обратимся к материализованному представлению «Маршруты». Но теперь при формировании запроса, в отличие от первого примера, воспользуемся столбцом days\_of\_week, в котором содержатся массивы номеров дней недели, когда выполняется данный рейс.

*SELECT array\_length( days\_of\_week, 1 ) AS days\_per\_week, count( \* ) AS num\_routes*

*FROM routes*

*GROUP BY days\_per\_week*

*ORDER BY 1 desc;*



В качестве примера приведем такой запрос: определить, сколько существует маршрутов из каждого города в другие города, и вывести названия городов, из которых в другие города существует не менее 15 маршрутов.

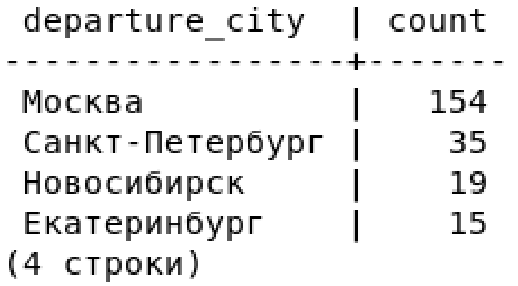
*SELECT departure\_city, count( \* )*

*FROM routes*

*GROUP BY departure\_city*

*HAVING count( \* ) >= 15*

*ORDER BY count DESC;*

**

Предположим, что руководство нашей компании хочет усовершенствовать тарифную политику и с этой целью просит нас предоставить сведения о распределении количества проданных билетов на некоторые рейсы во времени. Количество проданных билетов должно выводиться в виде накопленного показателя, суммирование должно производиться в пределах каждого календарного месяца.

Более детально, в столбцах book\_ref и book\_date выборки должны приводиться номер и время бронирования соответственно. В столбцах month и day должны указываться порядковый номер месяца и день этого месяца. Столбец count должен содержать суммарные (накопленные) количества билетов, проданных на каждый момент времени. С первого дня нового месяца подсчет числа проданных билетов начинается сначала.

*SELECT b.book\_ref, b.book\_date, extract( 'month' from b.book\_date ) AS month, extract( 'day' from b.book\_date ) AS day, count( \* ) OVER (*

*PARTITION BY date\_trunc( 'month', b.book\_date )*

*ORDER BY b.book\_date ) AS count*

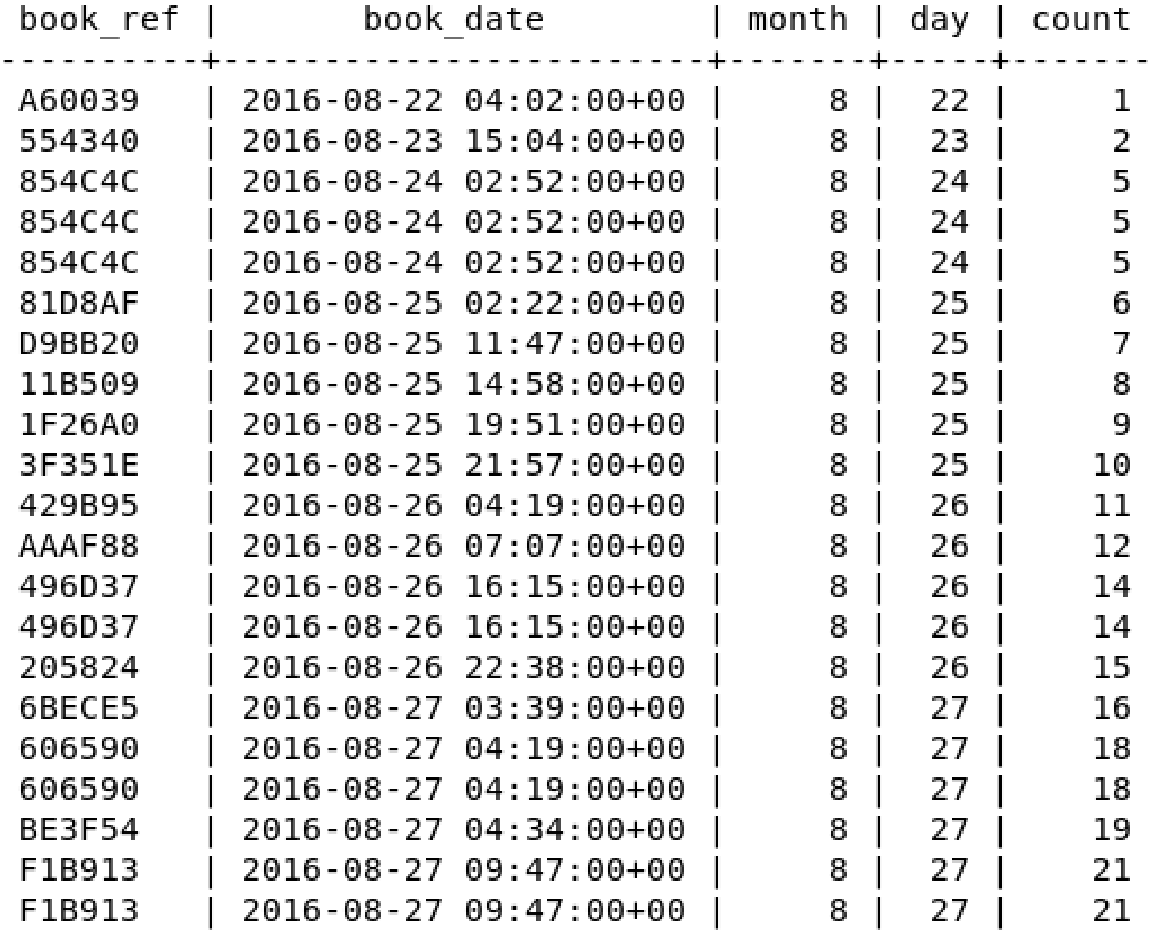
*FROM ticket\_flights tf*

*JOIN tickets t ON tf.ticket\_no = t.ticket\_no*

*JOIN bookings b ON t.book\_ref = b.book\_ref*

*WHERE tf.flight\_id = 1*

*ORDER BY b.book\_date;*

**

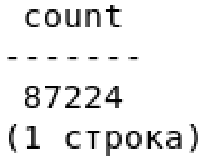
Предположим, что сотрудникам аналитического отдела потребовалось провести статистическое исследование финансовых результатов работы авиакомпании. В качестве первого шага они решили подсчитать количество операций бронирования, в которых общая сумма превышает среднюю величину по всей выборке.

*SELECT count( \* )*

*FROM bookings*

*WHERE total\_amount >*

*( SELECT avg( total\_amount ) FROM bookings );*



В качестве примера давайте выясним, какие маршруты существуют между городами часового пояса Asia/Krasnoyarsk. Подзапрос будет выдавать список городов из этого часового пояса, а в предложении WHERE главного запроса с помощью предиката IN будет выполняться проверка на принадлежность города этому списку. При этом подзапрос выполняется только один раз для всего внешнего запроса, а не при обработке каждой строки из таблицы routes во внешнем запросе. Повторного выполнения подзапроса не требуется, т. к. его результат не зависит от значений, хранящихся в таблице routes. Такие подзапросы называются некоррелированными.

*SELECT flight\_no, departure\_city, arrival\_city*

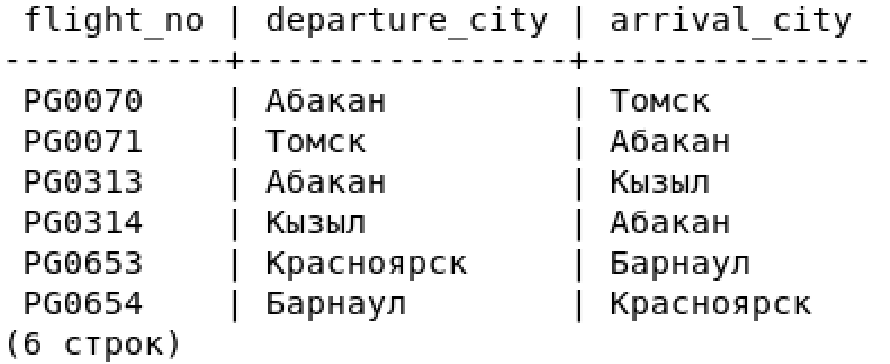
*FROM routes*

*WHERE departure\_city IN*

*( SELECT city FROM airports WHERE timezone ~ 'Krasnoyarsk' )*

*AND arrival\_city IN*

*( SELECT city FROM airports WHERE timezone ~ 'Krasnoyarsk' );*

**

Иногда возникают ситуации, когда от подзапроса требуется лишь установить сам факт наличия или отсутствия строк в конкретной таблице, удовлетворяющих определенному условию, а непосредственные значения атрибутов в этих строках интереса не представляют. В подобных случаях используют предикат EXISTS (или NOT EXISTS). В качестве примера выясним, в какие города нет рейсов из Москвы.

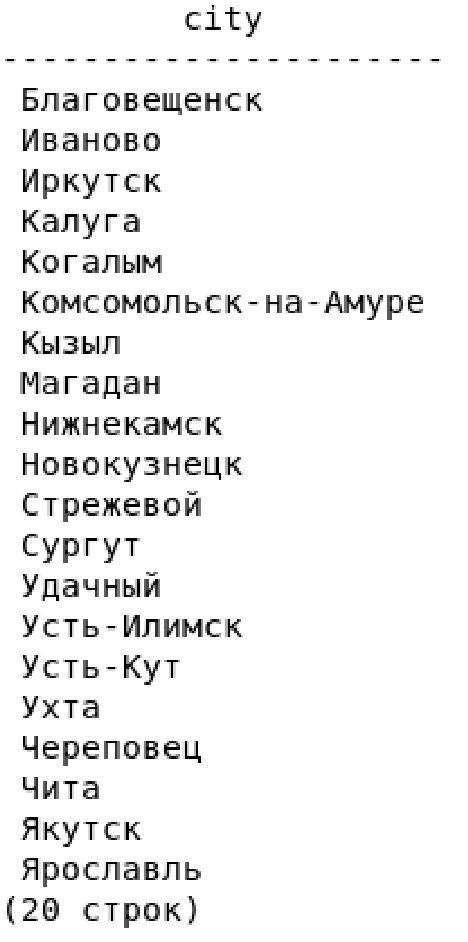
*SELECT DISTINCT a.city*

*FROM airports a*

*WHERE NOT EXISTS*

*( SELECT \* FROM routes r WHERE r.departure\_city = 'Москва' AND r.arrival\_city = a.city ) AND a.city <> 'Москва'*

*ORDER BY city;*

**

В качестве еще одного примера использования подзапроса в предложении FROM решим такую задачу: получить перечень аэропортов в тех городах, в которых больше одного аэропорта.

*SELECT aa.city, aa.airport\_code, aa.airport\_name*

*FROM ( SELECT city, count( \* )*

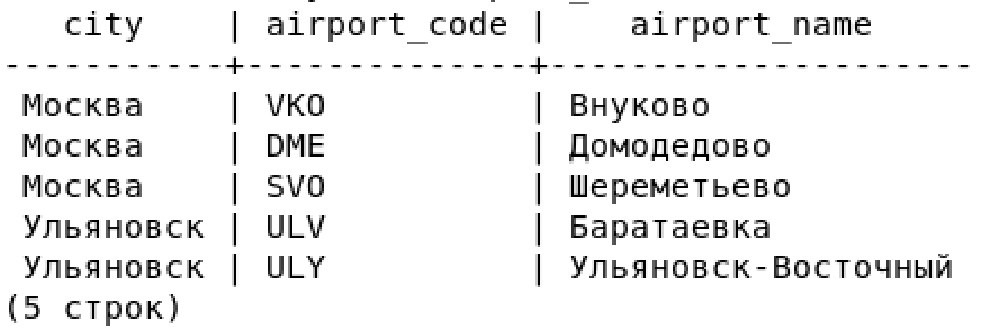
*FROM airports*

*GROUP BY city*

*HAVING count( \* ) > 1 ) AS a*

*JOIN airports AS aa ON a.city = aa.city*

*ORDER BY aa.city, aa.airport\_name;*

**

Для иллюстрации использования подзапросов в предложении HAVING решим такую задачу: определить число маршрутов, исходящих из тех аэропортов, которые расположены восточнее географической долготы 150◦.

*SELECT departure\_airport, departure\_city, count( \* )*

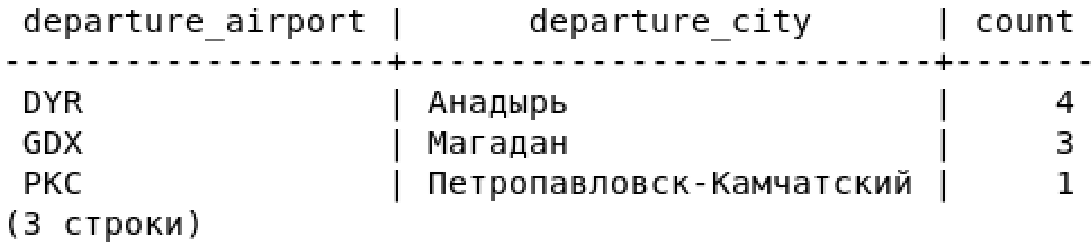
*FROM routes*

*GROUP BY departure\_airport, departure\_city*

*HAVING departure\_airport IN*

*( SELECT airport\_code FROM airports WHERE longitude > 150 )*

*ORDER BY count DESC;*

**

В сложных запросах могут использоваться вложенные подзапросы. Это означает, что один подзапрос находится внутри другого. Давайте в качестве примера рассмотрим такую ситуацию: руководство авиакомпании хочет выяснить степень заполнения самолетов на всех рейсах, ведь отправлять полупустые самолеты не очень выгодно. Таким образом, запрос должен не только выдавать число билетов, проданных на данный рейс, и общее число мест в самолете, но должен также вычислять отношение этих двух показателей. Вот какой запрос получился:

*SELECT ts.flight\_id, ts.flight\_no, ts.scheduled\_departure\_local, ts.departure\_city, ts.arrival\_city, a.model, ts.fact\_passengers, ts.total\_seats, round( ts.fact\_passengers::numeric / ts.total\_seats::numeric, 2 ) AS fraction*

*FROM ( SELECT f.flight\_id, f.flight\_no, f.scheduled\_departure\_local, f.departure\_city, f.arrival\_city, f.aircraft\_code, count( tf.ticket\_no ) AS fact\_passengers, ( SELECT count( s.seat\_no )*

*FROM seats s WHERE s.aircraft\_code = f.aircraft\_code ) AS total\_seats*

*FROM flights\_v f*

*JOIN ticket\_flights tf ON f.flight\_id = tf.flight\_id*

*WHERE f.status = 'Arrived' GROUP BY 1, 2, 3, 4, 5, 6 ) AS ts*

*JOIN aircrafts AS a ON ts.aircraft\_code = a.aircraft\_code*

*ORDER BY ts.scheduled\_departure\_local;*

**

Самый внутренний подзапрос — total\_seats — выдает общее число мест в самолете. Этот подзапрос — коррелированный, т. к. он выполняется для каждой строки, обрабатываемой во внешнем подзапросе, т. е. для каждой модели самолета. Для подсчета числа проданных билетов мы использовали соединение представления «Рейсы» (flights\_v) с таблицей «Перелеты» (ticket\_flights) с последующей группировкой строк и вызовом функции count.

Рассмотренный сложный запрос можно сделать более наглядным за счет выделения подзапроса в отдельную конструкцию, которая называется общее табличное выражение (Common Table Expression — CTE).

*WITH ts AS*

*(SELECT f.flight\_id, f.flight\_no, f.scheduled\_departure\_local, f.departure\_city, f.arrival\_city, f.aircraft\_code, count( tf.ticket\_no ) AS fact\_passengers,*

*( SELECT count( s.seat\_no )*

*FROM seats s*

*WHERE s.aircraft\_code = f.aircraft\_code ) AS total\_seats FROM flights\_v f*

*JOIN ticket\_flights tf ON f.flight\_id = tf.flight\_id*

*WHERE f.status = 'Arrived'*

*GROUP BY 1, 2, 3, 4, 5, 6 )*

*SELECT ts.flight\_id, ts.flight\_no, ts.scheduled\_departure\_local, ts.departure\_city, ts.arrival\_city, a.model, ts.fact\_passengers, ts.total\_seats, round( ts.fact\_passengers::numeric / ts.total\_seats::numeric, 2 ) AS fraction*

*FROM ts*

*JOIN aircrafts AS a ON ts.aircraft\_code = a.aircraft\_code*

*ORDER BY ts.scheduled\_departure\_local;*



Конструкция WITH ts AS (...) и представляет собой общее табличное выражение (CTE). Такие конструкции удобны тем, что позволяют упростить основной запрос, сделать его менее громоздким.

**4 Контрольные вопросы и задания**

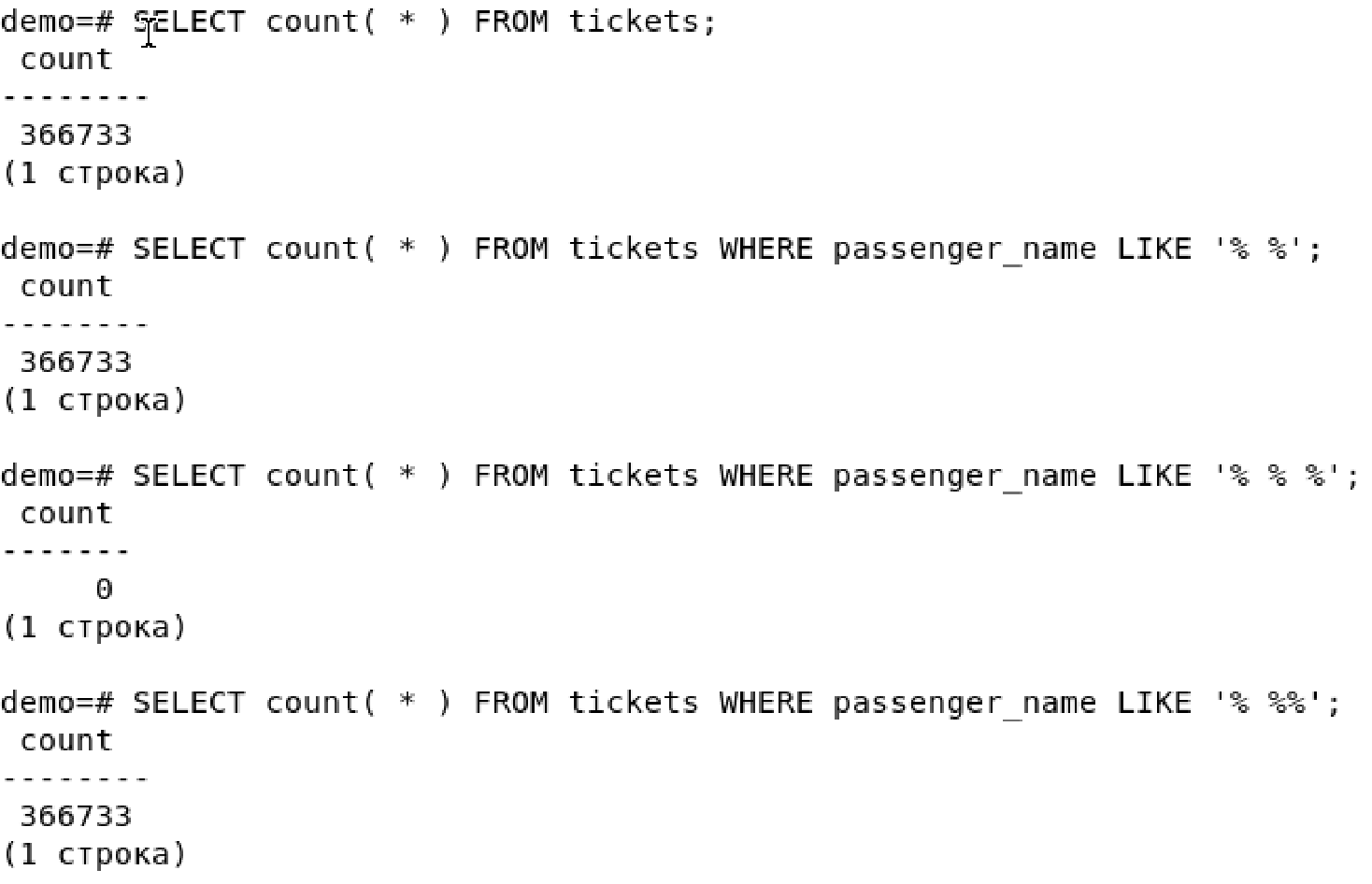
1. В документации сказано, что служебный символ «%» в шаблоне оператора LIKE соответствует любой последовательности символов, в том числе и пустой последовательности, однако ничего не сказано насчет правил обработки пробелов. В таблице «Билеты» (tickets) столбец passenger\_name содержит имя и фамилию пассажира, записанные заглавными латинскими буквами и разделенные одним пробелом. Выясните правила обработки пробелов самостоятельно, выполнив следующие команды и сравнив полученные результаты:

*SELECT count( \* ) FROM tickets;*

*SELECT count( \* ) FROM tickets WHERE passenger\_name LIKE '% %';*

*SELECT count( \* ) FROM tickets WHERE passenger\_name LIKE '% % %';*

*SELECT count( \* ) FROM tickets WHERE passenger\_name LIKE '% %%';*

**

Проанализировав полученные результаты, можно сказать, что пробелы обрабатываются точно так же, как и любые другие символы вместе с маской – 26 мы знаем, что данные пассажиров разделены одним пробелом, соответственно наличие группы «% % %» требует присутствия в строке трех групп слов, разделенных пробелом, чего в данной таблице не представлено.

2. Предложить шаблон поиска в операторе LIKE для выбора из таблицы «Билеты» всех пассажиров с фамилиями, состоящими из пяти букв.

*SELECT passenger\_name FROM tickets WHERE passenger\_name LIKE '\_\_\_\_\_ %';*



3. В разделе документации 9.7.2 «Регулярные выражения SIMILAR TO» рассматривается оператор SIMILAR TO. Он работает аналогично оператору LIKE, но использует шаблоны, соответствующие определению регулярных выражений, приведенному в стандарте SQL. Регулярные выражения SQL представляют собой комбинацию синтаксиса LIKE с синтаксисом обычных регулярных выражений. Самостоятельно ознакомьтесь с оператором SIMILAR TO.

SIMILAR TO сопоставляет строку с шаблоном регулярного выражения SQL. В отличие от некоторых других языков, шаблон должен соответствовать всей строке для успешного выполнения - совпадения подстроки недостаточно. Если какой-либо операнд равен NULL, результатом будет NULL. В противном случае результатом будет TRUE или FALSE.

'abc' SIMILAR TO 'abc' *true*

'abc' SIMILAR TO 'a' *false*

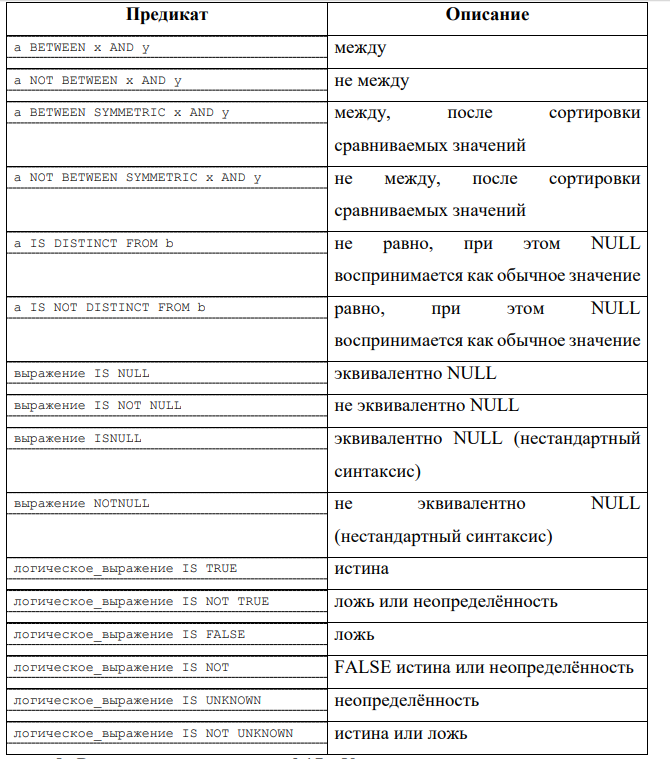
'abc' SIMILAR TO '%(b|d)%' *true*

'abc' SIMILAR TO '(b|c)%' *false*

'-abc-' SIMILAR TO '%\mabc\M%' *true*

'xabcy' SIMILAR TO '%\mabc\M%' *false*

4. В разделе документации 9.2 «Функция и операторы сравнения» представлены различные предикаты сравнения, кроме предиката BETWEEN, рассмотренного в этой главе. Самостоятельно ознакомьтесь с ними.



5. В разделе документации 9.17 «Условные выражения» представлены условные выражения, которые поддерживаются в PostgreSQL. В тексте главы была рассмотрена конструкция CASE. Самостоятельно ознакомьтесь с функциями COALESCE, NULLIF, GREATEST и LEAST.

Oracle/PLSQL функция COALESCE возвращает первое не NULL выражение из списка. Если все выражения определены как Null, то функция COALESCE вернет Null.

*SELECT COALESCE(description, short\_description, '(none)')*

Этот запрос вернёт значение description, если оно не равно NULL, либо short\_description, если оно не NULL, и строку (none), если оба эти значения равны NULL. Аргументы должны быть приводимыми к одному общему типу, который и будет типом результата.

Функция NULLIF сравнивает значение1 и значение2, если они равны - выдаёт NULL, если нет - возвращает значение1. Это может быть полезно для реализации обратной операции к COALESCE.

Функция GREATEST возвращает наибольшее значение в списке выражений. Функция LEAST возвращает наибольшее значение в списке выражений. Выражения должны приводиться к общему типу данных, который станет типом результата. Значения NULL в этом списке игнорируются. Результат выражения будет равен NULL, только если все его аргументы равны NULL.

*GREATEST( expr1, expr2, ... expr\_n )*

*LEAST( expr1, expr2, ... expr\_n )*

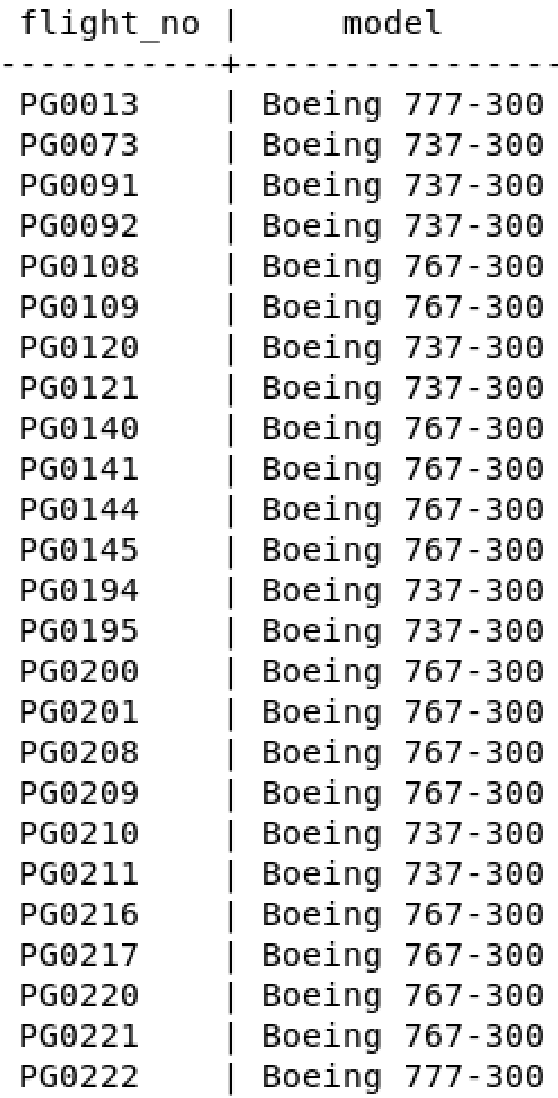
6. Выясните, на каких маршрутах используются самолеты компании Boeing. В выборке вместо кода модели должно выводиться ее наименование, например, вместо кода 733 должно быть Boeing 737-300.

*SELECT flight\_no, model*

*FROM routes*

*JOIN aircrafts ON aircrafts.aircraft\_code = routes.aircraft\_code*

*WHERE model LIKE 'Boeing%';*

**

7. Самые крупные самолеты в нашей авиакомпании — это Boeing 777-300. Выяснить, между какими парами городов они летают. Каждая пара городов была выведена только один раз.

*SELECT DISTINCT GREATEST(departure\_city, arrival\_city), LEAST(departure\_city, arrival\_city)*

*FROM routes*

*JOIN aircrafts ON aircrafts.aircraft\_code = routes.aircraft\_code*

*WHERE aircrafts.model = 'Boeing 777-300' ;*

**

8. Cколько рейсов выполняется из Москвы в Санкт-Петербург? Получить результат в следующем виде: departure\_city, arrival\_city, count.

*SELECT count (\*)*

*FROM routes*

*WHERE departure\_city = 'Москва' AND arrival\_city = 'Санкт-Петербург';*

**

9. Выяснить, сколько различных рейсов выполняется из каждого города, без учета частоты рейсов в неделю, можно с помощью обращения к представлению routes (маршруты).

*SELECT departure\_city, count( \* )*

*FROM routes*

*GROUP BY departure\_city*

*ORDER BY count DESC;*



10. Модифицируйте этот запрос так, чтобы он выводил число направлений, по которым летают самолеты из каждого города. Например, из Москвы в Санкт-Петербург летает несколько различных рейсов, но все эти рейсы относятся к одному направлению.

*SELECT departure\_city, count(DISTINCT arrival\_city)*

*FROM routes*

*GROUP BY departure\_city*

*ORDER BY count DESC;*

**

11. В материализованном представлении «Маршруты» (routes) имеется столбец days\_of\_week, который содержит списки (массивы) номеров дней недели, когда выполняется каждый рейс. Для оптимизации расписания вылетов из Москвы нужно выявить пять городов, в которые из столицы отправляется наибольшее число ежедневных рейсов (маршрутов). Строки в выборке следует расположить в убывающем порядке числа выполняемых рейсов.

*SELECT arrival\_city, array\_length(days\_of\_week, 1) as len, count(\*)*

*FROM routes*

*WHERE departure\_city = 'Москва'*

*GROUP BY arrival\_city, days*

*ORDER BY len DESC, count DESC*

*LIMIT 5;*

**

12.\* Предположим, что служба материального снабжения нашей авиакомпании запросила информацию о числе рейсов, выполняющихся из Москвы в каждый день недели.

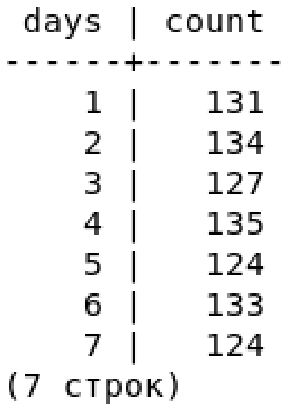
*SELECT unnest(days\_of\_week) AS days, count(\*)*

*FROM routes*

*WHERE departure\_city = 'Москва'*

*GROUP BY days*

*ORDER BY days;*

**

13. Каковы максимальные и минимальные цены билетов на все направления. Оператор SELECT должен возвращать departure\_city, arrival\_city, max(amount ), min(amount )

*SELECT departure\_city, arrival\_city, max(amount), min(amount)*

*FROM flights\_v*

*JOIN ticket\_flights as t ON ticket\_flights .flight\_id = flights\_v.flight\_id*

*GROUP BY departure\_city, arrival\_city*

*ORDER BY departure\_city;*

**

**5 Вывод**

По итогам выполнения данной лабораторной работы были изучены различные подходы к разработке сложных запросов SELECT. Были рассмотрены разнообразные примеры запросов с широким спектром операторов, предикатов и функций. После этого были проработаны практические задачи, связанные с разработкой запросов.