

Язык SQL

Лекция 4 Запросы

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

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)
```

- 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;
```

4.1. Дополнительные возможности команды SELECT

- Основой для экспериментов в этом разделе будут самые маленькие (по числу строк) таблицы базы данных «Авиаперевозки»: «Самолеты» (aircrafts) и «Аэропорты» (airports).
- Сначала посмотрим содержимое этих двух таблиц. Можно включить расширенный режим вывода данных \x.

```
SELECT * FROM aircrafts;
```

```
SELECT * FROM airports;
```

- Условия отбора строк в предложении WHERE могут конструироваться с использованием следующих **операторов сравнения**: =, <>, >, >=, <, <=.
- В предыдущих главах мы уже использовали ряд таких операторов, поэтому сейчас рассмотрим некоторые **другие способы** осуществления отбора строк.

Задача: выбрать все самолеты компании Airbus.

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

← спецсимвол

- Символ «%» имеет специальное значение. Он соответствует *любой последовательности* символов, т. е. вместо него могут быть подставлены:
 - любые символы в любом количестве,
 - а может и не быть подставлено ни одного символа.
- В результате будут выбраны строки, в которых значения атрибута `model` начинаются с символов «Airbus»:

aircraft_code	model	range
320	Airbus A320-200	5700
321	Airbus A321-200	5600
319	Airbus A319-100	6700

(3 строки)

- Шаблон в операторе LIKE всегда покрывает всю анализируемую строку. Поэтому если требуется отыскать некоторую последовательность символов где-то внутри строки, то шаблон должен начинаться и завершаться символом «%».

```
SELECT * FROM aircrafts WHERE model LIKE '%A32%';
```

aircraft_code	model	range
320	Airbus A320-200	5700
321	Airbus A321-200	5600

(2 строки)

- Если по тому столбцу, к которому применяется оператор LIKE, создан индекс для ускорения доступа к данным, то при наличии символа «%» в начале шаблона этот индекс использоваться не будет. В результате может ухудшиться производительность.

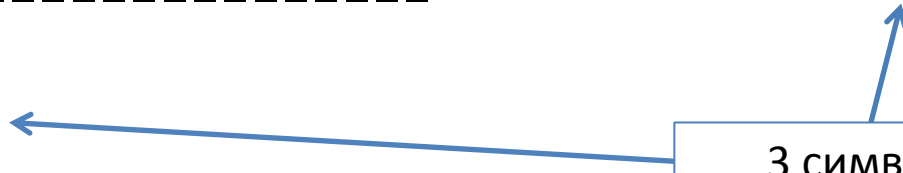
- Кроме символа «%» в шаблоне может использоваться и символ «_», который соответствует в точности *одному любому символу*.
- В качестве примера найдем в таблице «Аэропорты» (airports) те из них, которые имеют названия длиной три символа (буквы).
- С этой целью зададим в качестве шаблона строку, состоящую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

```
SELECT * FROM airports WHERE airport_name LIKE '___';
```

```
--[ RECORD 1 ]+-----
```

airport_code		UFA
airport_name		Уфа
city		Уфа
longitude		55.874417
latitude		54.557511
timezone		Asia/Yekaterinburg

3 символа



Если мы захотим узнать, какими самолетами, кроме машин компаний Airbus и Boeing, располагает наша авиакомпания, то придется усложнить условие:

```
SELECT * FROM aircrafts
WHERE model NOT LIKE 'Airbus%' AND
       model NOT LIKE 'Boeing%';
```

aircraft_code	model	range
SU9	Sukhoi SuperJet-100	3000
CN1	Cessna 208 Caravan	1200
CR2	Bombardier CRJ-200	2700

(3 строки)

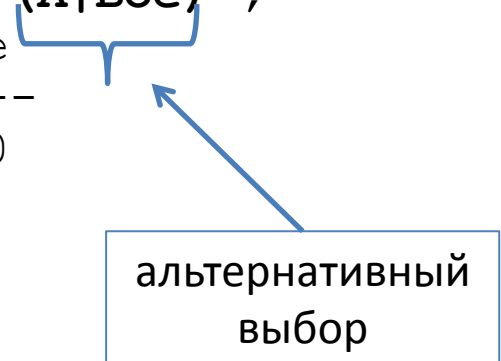
- Эти операторы имеют больше возможностей, чем оператор LIKE.
- Для того чтобы выбрать, например, самолеты компаний Airbus и Boeing, можно сделать так:

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

aircraft_code	model	range
773	Boeing 777-300	11100
763	Boeing 767-300	7900
320	Airbus A320-200	5700
321	Airbus A321-200	5600
319	Airbus A319-100	6700
733	Boeing 737-300	4200

(6 строк)

- Оператор ~ ищет совпадение с шаблоном с учетом регистра символов.
- Символ «^» означает, что поиск совпадения будет привязан к началу строки.
- Если же требуется проверить наличие такого символа в составе строки, то перед ним нужно поставить символ обратной косой черты: «\^».
- Выражение в круглых скобках означает альтернативный выбор между значениями, разделяемыми символом «|». Поэтому в выборку попадут значения, начинающиеся либо на «А», либо на «Вое».



альтернативный
выбор

- Для инвертирования смысла оператора `~` нужно перед ним добавить знак «!»..
- Пример: найти модели самолетов, которые не завершаются числом 300.

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

В этом регулярном выражении символ «\$» означает привязку поискового шаблона к концу строки. Если же требуется проверить наличие такого символа в составе строки, то надо сделать так: «\».

aircraft_code	model	range
SU9	Sukhoi SuperJet-100	3000
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

(6 строк)

- Предикаты сравнения могут использоваться в качестве замены традиционных операторов сравнения.
- Вопрос: какие самолеты имеют дальность полета в диапазоне от 3000 км до 6000 км? Ответ получим с помощью предиката BETWEEN.

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

aircraft_code	model	range
SU9	Sukhoi SuperJet-100	3000
320	Airbus A320-200	5700
321	Airbus A321-200	5600
733	Boeing 737-300	4200

(4 строки)

- Обратите внимание, что граничное значение 3000 было включено в полученную выборку.

Если мы захотим представить дальность полета лайнеров 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;
```

model	range	miles
Boeing 777-300	11100	6898.6948415164698571
Boeing 767-300	7900	4909.8819142324425109

...

(9 строк)

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;
```

model	range	miles
Boeing 777-300	11100	6898.69
Boeing 767-300	7900	4909.88

...

(9 строк)

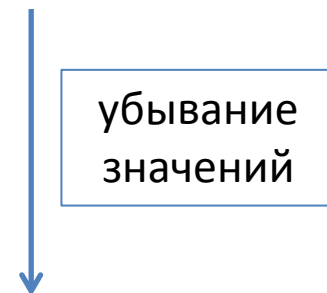
ВАЖНО! Если не принять специальных мер, то СУБД не гарантирует никакого конкретного порядка строк в результирующей выборке.

Можно задать не только возрастающий, но также и *убывающий* порядок сортировки (DESC – DESCENDANT).

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

aircraft_code	model	range
773	Boeing 777-300	11100
763	Boeing 767-300	7900
...		
CR2	Bombardier CRJ-200	2700
CN1	Cessna 208 Caravan	1200

(9 строк)



Вопрос: в каких различных часовых поясах располагаются аэропорты.

Если сделать традиционную выборку

```
SELECT timezone FROM airports;
```

то мы получим список значений, среди которых будет много повторяющихся.

Оставим в выборке только неповторяющиеся значения:

```
SELECT DISTINCT timezone FROM airports ORDER BY 1;
```

Обратите внимание, что столбец, по значениям которого будут упорядочены строки, указан с помощью его порядкового номера в предложении SELECT.


```
timezone
```

```
-----
```

```
Asia/Anadyr
```

```
Asia/Chita
```

```
Asia/Irkutsk
```

```
...
```

```
Europe/Samara
```

```
Europe/Volgograd
```

а в таблице более 100 строк

(17 строк)




Часовые пояса поддерживаются международной организацией IANA (Internet Assigned Numbers Authority), и отличаются от традиционных географических и административных часовых поясов, число которых в России равно одиннадцати.

Задача: найти три самых восточных аэропорта.

Алгоритм: отсортировать строки в таблице по убыванию значений столбца «Долгота» (longitude) и включить в выборку только первые три строки.

```
SELECT airport_name, city, longitude
FROM airports
ORDER BY longitude DESC
LIMIT 3;
```

airport_name	city	longitude
Анадырь	Анадырь	177.741483
Елизово	Петропавловск-Камчатский	158.453669
Магадан	Магадан	150.720439



(3 строки)

Задача: найти еще три аэропорта, которые находятся немного западнее первой тройки, т. е. занимают места с четвертого по шестое.

Алгоритм, использованный в первой задаче, будет дополнен еще одним шагом: нужно *пропустить три первые строки*, прежде чем начать вывод.

```
SELECT airport_name, city, longitude
FROM airports
ORDER BY longitude DESC
LIMIT 3 OFFSET 3;
```

airport_name	city	longitude
Хомутово	Южно-Сахалинск	142.717531
Хурба	Комсомольск-на-Амуре	136.934
Хабаровск-Новый	Хабаровск	135.188361

(3 строки)



Они позволяют вывести то или иное значение в зависимости от условий.

```
CASE WHEN condition THEN result
      [WHEN ...]
      [ELSE result]
END
```

```
SELECT model, range,
       CASE WHEN range < 2000 THEN 'Ближнемагистральный'
            WHEN range < 5000 THEN 'Среднемагистральный'
            ELSE 'Дальнемагистральный'
       END AS type
FROM aircrafts
ORDER BY model;
```

model	range	type
Airbus A319-100	6700	Дальнемагистральный
Airbus A320-200	5700	Дальнемагистральный
Airbus A321-200	5600	Дальнемагистральный
Boeing 737-300	4200	Среднемагистральный
Boeing 767-300	7900	Дальнемагистральный
Boeing 777-300	11100	Дальнемагистральный
Bombardier CRJ-200	2700	Среднемагистральный
Cessna 208 Caravan	1200	Ближнемагистральный
Sukhoi SuperJet-100	3000	Среднемагистральный

(9 строк)

4.2. Соединения

Соединение двух таблиц на основе равенства значений атрибутов (1)

В тех случаях, когда информации, содержащейся в одной таблице, недостаточно для получения требуемого результата, используют **соединение (join)** таблиц.

Задача: 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           -- s -- псевдоним  
     JOIN aircrafts AS a  -- a -- псевдоним  
     ON s.aircraft_code = a.aircraft_code  
WHERE a.model ~ '^Cessna'  
ORDER BY s.seat_no;
```

комментарии



Соединение двух таблиц на основе равенства значений атрибутов (2)

```
aircraft_code |          model          | seat_no | fare_conds
-----+-----+-----+-----
CN1           | Cessna 208 Caravan     | 1A      | Economy
CN1           | Cessna 208 Caravan     | 1B      | Economy
...
CN1           | Cessna 208 Caravan     | 6A      | Economy
CN1           | Cessna 208 Caravan     | 6B      | Economy
(12 строк)
```


- Если бы в качестве исходных сведений мы получили сразу код самолета — «CN1», то запрос свелся бы к выборке из одной таблицы «Места» (seats). Он был бы таким:

```
SELECT * FROM seats WHERE aircraft_code = 'CN1';
```

- Но нам дано название модели, а не ее код, поэтому придется подключить к работе и таблицу «Самолеты» (aircrafts), в которой хранятся наименования моделей.
- Для того чтобы решить, удовлетворяет ли строка таблицы seats поставленному условию, нужно узнать, какой модели самолета соответствует эта строка.
- Как это можно узнать?

Как мы рассуждали? (2)

aircrafts

aircraft_code	model	range	aircraft_code	seat_no	fare_conditions
...			...		
319	Airbus A319-100	6700	CN1	1A	Economy
733	Boeing 737-300	4200	CN1	1B	Economy
CN1	Cessna 208 Caravan	1200	CN1	2A	Economy
			CN1	2B	Economy
			733	1A	Business
			733	1C	Business
			...		

aircrafts

aircraft_code	model	range	aircraft_code	seat_no	fare_con
...					
CN1	Cessna 208 Caravan	1200	CN1	1A	Economy
CN1	Cessna 208 Caravan	1200	CN1	1B	Economy
CN1	Cessna 208 Caravan	1200	CN1	2A	Economy
CN1	Cessna 208 Caravan	1200	CN1	2B	Economy
733	Boeing 737-300	4200	733	1A	Business
733	Boeing 737-300	4200	733	1C	Business
...					

- В каждой строке таблицы `seats` есть атрибут `aircraft_code`, такой же атрибут есть и в каждой строке таблицы `aircrafts`.
- Если с каждой строкой таблицы `seats` *соединить* такую строку таблицы `aircrafts`, в которой значение атрибута `aircraft_code` такое же, как и в строке таблицы `seats`, то сформированная *комбинированная строка*, составленная из атрибутов обеих таблиц, будет содержать не только номер места, класс обслуживания и код модели, но — что важно — и *наименование модели*.
- Поэтому с помощью условия `WHERE` можно будет отобразить только те результирующие строки, в которых значение атрибута `model` будет «Cessna 208 Caravan».

- А какие столбцы оставлять в списке столбцов предложения SELECT, решать нам.
- Даже если мы соединяем две таблицы (или более), то совершенно не обязательно в результирующий список столбцов включать столбцы всех таблиц, перечисленных в предложении FROM. Мы могли бы оставить только атрибуты таблицы seats:

```
SELECT s.seat_no, s.fare_conditions
FROM seats s JOIN aircrafts a
     ON s.aircraft_code = a.aircraft_code
WHERE a.model ~ '^Cessna'
ORDER BY s.seat_no;
```

seat_no	fare_conditions
1A	Economy
1B	Economy
...	

(12 строк)

условие соединения
таблиц

- Сначала формируются *все попарные комбинации* строк из обеих таблиц, т. е. декартово произведение множеств строк этих таблиц. Эти комбинированные строки включают в себя *все атрибуты обеих таблиц*.
- Затем в дело вступает условие `s.aircraft_code = a.aircraft_code`. Это означает, что в результирующем множестве строк останутся только те из них, в которых значения атрибута `aircraft_code`, взятые из таблицы `aircrafts` и из таблицы `seats`, одинаковые. Строки, не удовлетворяющие этому критерию, отфильтровываются. Это означает на практике, что *каждой строке* из таблицы «Места» мы сопоставили *только одну конкретную строку* из таблицы «Самолеты», из которой мы теперь можем взять значение атрибута «Модель самолета», чтобы включить ее в итоговый вывод данных.
- На практике описанный механизм не реализуется буквально. Специальная подсистема PostgreSQL, называемая **планировщиком**, строит план выполнения запроса, который является гораздо более эффективным, чем упрощенный план, представленный здесь.

Тот же запрос, но без использования JOIN:

```
SELECT a.aircraft_code, a.model, s.seat_no,  
       s.fare_conditions  
FROM seats s, aircrafts a  
WHERE s.aircraft_code = a.aircraft_code AND  
       a.model ~ '^Cessna'  
ORDER BY s.seat_no;
```

Условие соединения таблиц `s.aircraft_code = a.aircraft_code` перешло из предложения FROM в предложение WHERE.

ОЧЕНЬ ВАЖНО! Результатом любых реляционных операций над отношениями (таблицами, представлениями) также является *отношение*. Поэтому такие операции можно комбинировать друг с другом.

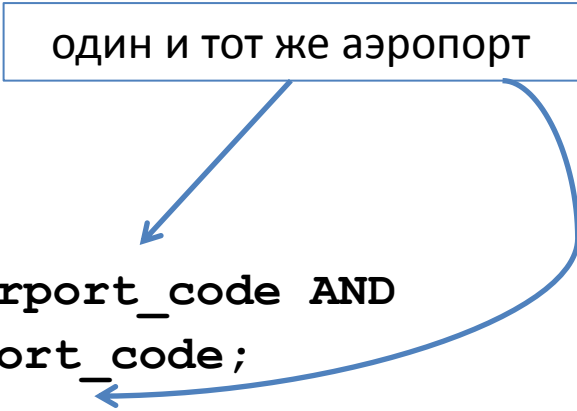
В качестве примера рассмотрим запрос для создания представления «Рейсы» (flights_v), о котором шла речь в предыдущей лекции.

```
CREATE OR REPLACE VIEW flights_v AS
SELECT f.flight_id, f.flight_no, f.scheduled_departure,
       timezone( dep.timezone, f.scheduled_departure )
       AS scheduled_departure_local,
       f.scheduled_arrival,
       timezone( arr.timezone, f.scheduled_arrival )
       AS scheduled_arrival_local,
       f.scheduled_arrival - f.scheduled_departure
       AS scheduled_duration,
       f.departure_airport,
       dep.airport_name AS departure_airport_name,
       dep.city AS departure_city,
       f.arrival_airport,
       arr.airport_name AS arrival_airport_name,
       arr.city AS arrival_city,
       f.status, f.aircraft_code, f.actual_departure,
       timezone( dep.timezone, f.actual_departure )
       AS actual_departure_local,
       f.actual_arrival,
       timezone( arr.timezone, f.actual_arrival )
       AS actual_arrival_local,
       f.actual_arrival - f.actual_departure AS actual_duration
FROM flights f, airports dep, airports arr
WHERE f.departure_airport = dep.airport_code AND
       f.arrival_airport = arr.airport_code;
```

- Будем рассуждать от противного. Пусть в предложении FROM таблица «Аэропорты» (airports) будет указана только один раз:

```
...  
FROM flights f, airports a  
WHERE f.departure_airport = a.airport_code AND  
f.arrival_airport = a.airport_code;
```

один и тот же аэропорт



разные аэропорты



Получается, что данное условие будет выполнено, если только аэропорт вылета и аэропорт назначения будет одним и тем же. Но так не бывает.

- Это означает, что при соединении таблиц `flights` и `airports` PostgreSQL будет пытаться для каждой строки из таблицы `flights` найти такую строку в таблице `airports`, в которой значение атрибута `airport_code` будет равно не только значению атрибута `departure_airport`, но также и значению атрибута `arrival_airport` в таблице `flights`.
- Получается, что данное условие будет выполнено, если только аэропорт вылета и аэропорт назначения будет одним и тем же.
- Таким образом каждую строку из таблицы «Рейсы» (`flights`) необходимо соединять с *двумя различными* строками из таблицы «Аэропорты».
- Однако при однократном включении таблицы «Аэропорты» (`airports`) в предложение FROM сделать это невозможно, поэтому поступают так: к таблице `airports` в предложении FROM обращаются дважды, *как будто* это две копии одной и той же таблицы. Но используют разные псевдонимы.

- Покажем три способа выполнения соединения таблицы с самой собой, отличающиеся синтаксически, но являющиеся функционально эквивалентными.
- Запрос-иллюстрация должен выяснить: сколько всего маршрутов нужно было бы сформировать, если бы требовалось соединить каждый город со всеми остальными городами.
- Если в городе имеется более одного аэропорта, то договоримся рейсы из каждого из них (в каждый из них) считать отдельными маршрутами.
- Поэтому правильнее было бы говорить не о маршрутах из каждого *города*, а о маршрутах из каждого *аэропорта* во все другие аэропорты.
- Ограничение. Конечно, рейсов из любого города *в тот же самый город* быть не должно.

Перечисление имен таблиц в предложении FROM

Вариант 1. Обычное перечисление имен таблиц в предложении FROM.

- Поскольку имена таблиц совпадают, используются псевдонимы. В таком случае СУБД обращается к таблице дважды, как если бы это были различные таблицы.

```
SELECT count( * )  
FROM airports a1, airports a2  
WHERE a1.city <> a2.city;
```

- СУБД соединяет *каждую* строку первой таблицы с *каждой* строкой второй таблицы, т. е. формирует декартово произведение таблиц — все попарные комбинации строк из двух таблиц.
- Затем СУБД отбрасывает те комбинированные строки, которые не удовлетворяют условию, приведенному в предложении WHERE. В нашем примере условие как раз и отражает требование о том, что рейсов из одного города в тот же самый город быть не должно.

```
count  
-----  
10704  
(1 строка)
```

Соединение таблиц на основе неравенства значений атрибутов

Вариант 2. Используем соединение таблиц на основе *неравенства* значений атрибутов.

- Тем самым мы перенесли условие отбора результирующих строк из предложения WHERE в предложение FROM.

```
SELECT count( * )  
FROM airports a1 JOIN airports a2  
ON a1.city <> a2.city;
```

```
count  
-----  
10704  
(1 строка)
```

условие отбора строк



Вариант 3. Явное использование декартова произведения таблиц.

- Для этого служит предложение CROSS JOIN. Лишние строки, как и в первом варианте, отсеиваем с помощью предложения WHERE:

```
SELECT count( * )  
FROM airports a1 CROSS JOIN airports a2  
WHERE a1.city <> a2.city;
```

```
count  
-----  
10704  
(1 строка)
```

ВАЖНО! С точки зрения СУБД эти три варианта эквивалентны, они отличаются лишь синтаксисом. Для них PostgreSQL выберет один и тот же план (порядок) выполнения запроса.

Вопрос: сколько маршрутов обслуживают самолеты каждого типа?

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;
```

aircraft_code	model	num_routes
CR2	Bombardier CRJ-200	232
CN1	Cessna 208 Caravan	170
SU9	Sukhoi SuperJet-100	158
319	Airbus A319-100	46
733	Boeing 737-300	36
321	Airbus A321-200	32
763	Boeing 767-300	26
773	Boeing 777-300	10

(8 строк)



А в таблице «Самолеты» (aircrafts) представлено **9 моделей**. Значит, какая-то модель самолета не участвует в выполнении рейсов. Как ее выявить?

```
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;
```

Столбец включен
лишь для
наглядности.

a_code	model	r_code	num_routes
CR2	Bombardier CRJ-200	CR2	232
CN1	Cessna 208 Caravan	CN1	170
SU9	Sukhoi SuperJet-100	SU9	158
319	Airbus A319-100	319	46
733	Boeing 737-300	733	36
321	Airbus A321-200	321	32
763	Boeing 767-300	763	26
773	Boeing 777-300	773	10
320	Airbus A320-200		0

(9 строк)

NULL

- В качестве базовой таблицы выбирается таблица `aircrafts`, указанная в запросе слева от предложения `LEFT OUTER JOIN`, и для каждой строки, находящейся в ней, из таблицы `routes` подбираются строки, в которых значение атрибута `aircraft_code` такое же, как и в текущей строке таблицы `aircrafts`.
- Если в таблице `routes` нет ни одной соответствующей строки, то при отсутствии ключевых слов `LEFT OUTER` результирующая комбинированная строка просто *не будет сформирована и не попадет в выборку*.
- Но при наличии ключевых слов `LEFT OUTER` результирующая строка все равно будет сформирована. Но тогда в нее вместо значений столбцов правой таблицы будут помещены значения `NULL`.
- **Обратите внимание, что параметром функции `count` является столбец из таблицы `routes`, поэтому `count` и выдает число 0 для самолета с кодом 320.**
- Если заменить его на одноименный столбец из таблицы `aircrafts`, тогда `count` выдаст 1, что будет противоречить цели нашей задачи — подсчитать число рейсов, выполняемых на самолетах каждого типа.

- Кроме левого внешнего соединения существует также и правое внешнее соединение — **RIGHT OUTER JOIN**. В этом случае в качестве базовой выбирается таблица, имя которой указано справа от предложения **RIGHT OUTER JOIN**, а механизм получения результирующих строк в случае, когда для строки базовой таблицы не находится пары во второй таблице, точно такой же, как и для левого внешнего соединения.
- Как сказано в документации, правое внешнее соединение является лишь *синтаксическим приемом*, поскольку всегда можно заменить его левым внешним соединением, поменяв при этом имена таблиц местами.
- Важно учитывать, что порядок следования таблиц в предложениях **LEFT (RIGHT) OUTER JOIN** никак не влияет на порядок столбцов в предложении **SELECT**. В вышеприведенном запросе мы могли бы поменять столбцы местами:

```
SELECT r.aircraft_code AS r_code,  
        a.model,  
        a.aircraft_code AS a_code,  
        count( r.aircraft_code ) AS num_routes
```

...

- Комбинацией этих двух видов внешних соединений является полное внешнее соединение — **FULL OUTER JOIN**.
- В этом случае в выборку включаются строки из левой таблицы, для которых не нашлось соответствующих строк в правой таблице, и строки из правой таблицы, для которых не нашлось соответствующих строк в левой таблице.

Задача: 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.

```
count
-----
      0
(1 строка)
```

- Поскольку нас интересуют только рейсы с непустым временем вылета, нам придется обратиться к таблице «Рейсы» (flights) и соединить ее с таблицей ticket_flights по атрибуту flight_id.
- А затем для подключения таблицы boarding_passes мы используем левое внешнее соединение, т. к. в этой таблице может не оказаться строки, соответствующей строке из таблицы ticket_flights.
- Таблица «Посадочные талоны» (boarding_passes) связана с таблицей «Перелеты» (ticket_flights) по внешнему ключу, а тип связи — 1:1, т. е. каждой строке из таблицы ticket_flights соответствует *не более одной* строки в таблице boarding_passes: ведь строка в таблицу boarding_passes добавляется только тогда, когда пассажир прошел регистрацию на рейс. Однако пассажир может на регистрацию не явиться, тогда строка в таблицу boarding_passes добавлена не будет.

- В предложении WHERE второе условие — `b.flight_id IS NULL`. Оно как раз и позволяет выявить те комбинированные строки, в которых столбцам таблицы `boarding_passes` были назначены значения `NULL` из-за того, что в ней не нашлось строки, для которой выполнялось бы условие `t.ticket_no = b.ticket_no AND t.flight_id = b.flight_id`.
- Конечно, мы могли использовать любой столбец таблицы `boarding_passes`, а не только `b.flight_id`, для проверки на `NULL`.

- Предположим, что возможна такая ситуация: при бронировании билета пассажир выбрал один класс обслуживания, например, «Business», а при регистрации на рейс ему выдали посадочный талон на то место в салоне самолета, где класс обслуживания — «Economy». Задача: выявить все случаи несовпадения классов обслуживания.
- Сведения о классе обслуживания, который пассажир выбрал при бронировании билета, содержатся в таблице «Перелеты» (ticket_flights).
- Однако в таблице «Посадочные талоны» (boarding_passes), которая «отвечает» за посадку на рейс, сведений о классе обслуживания, который пассажир получил при регистрации, нет.
- Эти сведения можно получить только из таблицы «Места» (seats). Причем, сделать это можно, зная код модели самолета, выполняющего рейс, и номер места в салоне самолета.

- Номер места можно взять из таблицы `boarding_passes`, а код модели самолета можно получить из таблицы «Рейсы» (`flights`), связав ее с таблицей `boarding_passes`.
- Для полноты информационной картины необходимо получить еще фамилию и имя пассажира из таблицы «Билеты» (`tickets`), связав ее с таблицей `ticket_flights` по атрибуту «Номер билета» (`ticket_no`).
- При формировании запроса выберем в качестве, условно говоря, базовой таблицы таблицу `boarding_passes`, а затем будем поэтапно подключать остальные таблицы.
- В предложении `WHERE` будет только одно условие: несовпадение требуемого и фактического классов обслуживания.

В результате получим запрос, включающий пять таблиц.

```
SELECT f.flight_no, f.scheduled_departure, f.flight_id,
       f.departure_airport, f.arrival_airport,
       f.aircraft_code, t.passenger_name,
       tf.fare_conditions AS fc_to_be,
       s.fare_conditions AS fc_fact, b.seat_no
FROM boarding_passes b
     JOIN ticket_flights tf
       ON b.ticket_no = tf.ticket_no
          AND b.flight_id = tf.flight_id
     JOIN tickets t ON tf.ticket_no = t.ticket_no
     JOIN flights f ON tf.flight_id = f.flight_id
     JOIN seats s ON b.seat_no = s.seat_no
          AND f.aircraft_code = s.aircraft_code
WHERE tf.fare_conditions <> s.fare_conditions
ORDER BY f.flight_no, f.scheduled_departure;
```

Этот запрос не выдаст ни одной строки, значит, пассажиров, получивших при регистрации неправильный класс обслуживания, не было.

- Чтобы все же удостовериться в работоспособности этого запроса, можно в таблице `boarding_passes` изменить в одной строке номер места таким образом, чтобы этот пассажир переместился из салона экономического класса в салон бизнес-класса.

```
UPDATE boarding_passes
SET seat_no = '1A'
WHERE flight_id = 1 AND seat_no = '17A';
UPDATE 1
```


- Выполним запрос еще раз. Теперь он выдаст одну строку.

```
--[ RECORD 1 ]-----+-----
flight_no      | PG0405
scheduled_departure | 2016-09-13 13:35:00+08
flight_id      | 1
departure_airport | DME
arrival_airport | LED
aircraft_code   | 321
passenger_name  | PAVEL AFANASEV
fc_to_be       | Economy
fc_fact        | Business
seat_no        | 1A
```

- В предложении FROM можно использовать виртуальные таблицы, сформированные с помощью ключевого слова **VALUES**.
- Задача: предположим, что для выработки финансовой стратегии нашей авиакомпания требуется следующая информация: распределение количества бронирований по диапазонам сумм с шагом в сто тысяч рублей. Максимальная сумма в одном бронировании составляет 1 204 500 рублей.

```
SELECT r.min_sum, r.max_sum, count( b.* )
FROM bookings b RIGHT OUTER JOIN
  ( VALUES ( 0, 100000 ),      ( 100000, 200000 ),
            ( 200000, 300000 ), ( 300000, 400000 ),
            ( 400000, 500000 ), ( 500000, 600000 ),
            ( 600000, 700000 ), ( 700000, 800000 ),
            ( 800000, 900000 ), ( 900000, 1000000 ),
            ( 1000000, 1100000 ), ( 1100000, 1200000 ),
            ( 1200000, 1300000 )
  ) AS r ( min_sum, max_sum )
ON b.total_amount >= r.min_sum AND
   b.total_amount < r.max_sum
GROUP BY r.min_sum, r.max_sum
ORDER BY r.min_sum;
```

имя таблицы и
столбцы



min_sum	max_sum	count
0	100000	198314
100000	200000	46943
200000	300000	11916
300000	400000	3260
400000	500000	1357
500000	600000	681
600000	700000	222
700000	800000	55
800000	900000	24
900000	1000000	11
1000000	1100000	4
1100000	1200000	0
1200000	1300000	1

(13 строк)

работает внешнее соединение



В команде SELECT предусмотрены средства для выполнения операций с выборками, как с множествами, а именно:

- предложение **UNION** предназначено для вычисления объединения множеств строк из двух выборок;
- предложение **INTERSECT** предназначено для вычисления пересечения множеств строк из двух выборок;
- предложение **EXCEPT** предназначено для вычисления разности множеств строк из двух выборок.

ВАЖНО! Запросы должны возвращать одинаковое число столбцов, типы данных у столбцов также должны совпадать.

Рассмотрим эти операции, используя материализованное представление «Маршруты» (routes).

- Строка включается в итоговое множество (выборку), если она присутствует *хотя бы в одном* из них.
- Строки-дубликаты в результирующее множество не включаются.
- Для их включения нужно использовать UNION ALL.

Вопрос: в какие города можно улететь *либо* из Москвы, *либо* из Санкт-Петербурга?

```
SELECT arrival_city FROM routes
WHERE departure_city = 'Москва'
UNION
SELECT arrival_city FROM routes
WHERE departure_city = 'Санкт-Петербург'
ORDER BY arrival_city;
```

```
arrival_city
```

```
-----
```

```
Абакан
```

```
Анадырь
```

```
...
```

```
Южно-Сахалинск
```

```
Якутск
```

```
Ярославль
```

```
(87 строк)
```

- Строка включается в итоговое множество (выборку), если она присутствует *в каждом* из них.
- Строки-дубликаты в результирующее множество не включаются.
- Для их включения нужно использовать INTERSECT ALL.

Вопрос: в какие города можно улететь *как* из Москвы, *так и* из Санкт-Петербурга?

```
SELECT arrival_city FROM routes  
WHERE departure_city = 'Москва'
```

INTERSECT

```
SELECT arrival_city FROM routes  
WHERE departure_city = 'Санкт-Петербург'  
ORDER BY arrival_city;
```



```
arrival_city
```

```
-----
```

```
Воркута
```

```
Воронеж
```

```
Казань
```

```
...
```

```
Чебоксары
```

```
Элиста
```

```
(15 строк)
```

- Строка включается в итоговое множество (выборку), если она *присутствует в первом* множестве (выборке), но *отсутствует во втором*.
- Строки-дубликаты в результирующее множество не включаются.
- Для их включения нужно использовать EXCEPT ALL.

Вопрос: в какие города *можно* улететь из Санкт-Петербурга, но *нельзя* из Москвы?

```
SELECT arrival_city FROM routes
WHERE departure_city = 'Санкт-Петербург'
```

EXCEPT

```
SELECT arrival_city FROM routes
WHERE departure_city = 'Москва'
ORDER BY arrival_city;
```

```
arrival_city
```

```
-----
```

```
Иркутск
```

```
Калуга
```

```
Москва
```

```
Удачный
```

```
Череповец
```

```
Якутск
```

```
Ярославль
```

```
(7 строк)
```

4.3. Агрегирование и группировка

Рассмотрим их на примере таблицы «Бронирования» (bookings).

- Расчет среднего значения по столбцу:

```
SELECT avg( total_amount ) FROM bookings;
```

```
avg
```

```
-----
```

```
79025.605811528685
```

```
(1 строка)
```

- Получение максимального значения по столбцу:

```
SELECT max( total_amount ) FROM bookings;
```

```
max
```

```
-----
```

```
1204500.00
```

```
(1 строка)
```

- Получение минимального значения по столбцу:

```
SELECT min( total_amount ) FROM bookings;
```

```
min
```

```
-----
```

```
3400.00
```

```
(1 строка)
```

Пример 1. 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» (routes).

```
SELECT arrival_city, count( * )  
FROM routes  
WHERE departure_city = 'Москва'  
GROUP BY arrival_city  
ORDER BY count DESC;
```

arrival_city	count
Санкт-Петербург	12
Брянск	9
Ульяновск	5
Йошкар-Ола	4
Петрозаводск	4
...	

Пример 2. Предположим, 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.

```
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;
```

длина массива

1 — первое
измерение массива

Воспользуемся столбцом days_of_week, в котором содержатся массивы номеров дней недели, когда выполняется данный рейс.


```
days_per_week | num_routes
-----+-----
              |
              7 |          482
              3 |          54
              2 |          88
              1 |          86
```

(4 строки)

С помощью предложения **HAVING** можно включить в результирующее множество не все *сгруппированные* строки, а лишь те, которые удовлетворяют некоторому условию.

Пример 1. 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;
```

departure_city	count
Москва	154
Санкт-Петербург	35
Новосибирск	19
Екатеринбург	15

(4 строки)

Пример 2. В подавляющем большинстве городов только один аэропорт, но есть и такие города, в которых более одного аэропорта. Давайте их ВЫЯВИМ.

```
SELECT city, count( * )  
FROM airports  
GROUP BY city  
HAVING count( * ) > 1;
```

city	count
Ульяновск	2
Москва	3

(2 строки)

Введение (1)

- Эти функции предоставляют возможность производить вычисления на множестве строк, *логически связанных с текущей строкой*, т. е. имеющих то или иное отношение к ней.
- При работе с оконными функциями используются концепции **раздела (partition)** и **оконного кадра (window frame)**.
- Раздел включает в себя все строки выборки, имеющие в некотором смысле *одинаковые свойства*, например, одинаковые значения определенных выражений, задаваемых с помощью предложения **PARTITION BY**.
- Оконный кадр состоит из *подмножества* строк данного раздела и привязан к текущей строке.
- Для определения границ кадра важным является наличие предложения ORDER BY при формировании раздела.

Введение (2)

Пример. 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.

book_ref	book_date	month	day	count
A60039	2016-08-22 12:02:00+08	8	22	1
554340	2016-08-23 23:04:00+08	8	23	2
854C4C	2016-08-24 10:52:00+08	8	24	5
854C4C	2016-08-24 10:52:00+08	8	24	5
81D8AF	2016-08-25 10:22:00+08	8	25	6
...				
8D6873	2016-08-31 17:09:00+08	8	31	59
E82829	2016-08-31 20:56:00+08	8	31	60
ECA0D7	2016-09-01 00:48:00+08	9	1	1
E3BD32	2016-09-01 04:44:00+08	9	1	2
...				
EB11BB	2016-09-03 12:02:00+08	9	3	14
19FE38	2016-09-03 17:42:00+08	9	3	16
19FE38	2016-09-03 17:42:00+08	9	3	16
536A3D	2016-09-03 19:19:00+08	9	3	18
536A3D	2016-09-03 19:19:00+08	9	3	18
02E6B6	2016-09-04 01:39:00+08	9	4	19

(79 строк)

- Для примера был выбран рейс с идентификатором 1.
- Подсчет числа проданных билетов выполняется в пределах оконного кадра.
- Строки в выборке упорядочены по значениям столбца book_date.

```
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;
```

- OVER – обязательное ключевое слово. Функция count становится оконной функцией.
- Предложение PARTITION BY задает *правило разбиения* строк выборки на разделы.
- Предложение ORDER BY предписывает *порядок сортировки* строк в разделах.

- В рассмотренном примере границы оконного кадра определялись по умолчанию.
- Для указания границ оконного кадра предусмотрены различные способы, приведенные в разделе документации 4.2.8 «Вызовы оконных функций».
- Не только функция `count`, но и другие агрегатные функции (например, `sum`, `avg`) тоже могут применяться в качестве оконных функций. Полный перечень собственно оконных функций приведен в документации в разделе 9.21 «Оконные функции».
- **Оконные функции, в отличие от обычных агрегатных функций, не требуют группировки строк, а работают на уровне отдельных (несгруппированных) строк.**
- Однако если в запросе присутствуют предложения `GROUP BY` и `HAVING`, тогда оконные функции вызываются уже *после* них. В таком случае оконные функции будут работать со строками, являющимися результатом группировки.

Покажем, как с помощью оконной функции `rank` можно проранжировать аэропорты в пределах каждого часового пояса на основе их географической широты.

- Причем будем присваивать более высокий ранг тому аэропорту, который находится севернее.

```
SELECT airport_name, city,  
       round( latitude::numeric, 2 ) AS ltd,  
       timezone,  
       rank() OVER (  
         PARTITION BY timezone  
         ORDER BY latitude DESC  
       )  
FROM airports  
WHERE timezone IN ( 'Asia/Irkutsk', 'Asia/Krasnoyarsk' )  
ORDER BY timezone, rank;
```

сортировка в пределах каждого окна

сортировка на уровне всего запроса

airport_name	city	ltd	timezone	rank
Усть-Илимск	Усть-Илимск	58.14	Asia/Irkutsk	1
Усть-Кут	Усть-Кут	56.85	Asia/Irkutsk	2
Братск	Братск	56.37	Asia/Irkutsk	3
Иркутск	Иркутск	52.27	Asia/Irkutsk	4
Байкал	Улан-Удэ	51.81	Asia/Irkutsk	5
Норильск	Норильск	69.31	Asia/Krasnoyarsk	1
Стрежевой	Стрежевой	60.72	Asia/Krasnoyarsk	2
Богашёво	Томск	56.38	Asia/Krasnoyarsk	3
Емельяново	Красноярск	56.18	Asia/Krasnoyarsk	4
Абакан	Абакан	53.74	Asia/Krasnoyarsk	5
Барнаул	Барнаул	53.36	Asia/Krasnoyarsk	6
Горно-Алтайск	Горно-Алтайск	51.97	Asia/Krasnoyarsk	7
Кызыл	Кызыл	51.67	Asia/Krasnoyarsk	8

(13 строк)

- Усложним запрос — для каждого аэропорта будем вычислять разницу между его географической широтой и широтой, на которой находится самый северный аэропорт в этом же часовом поясе.
- Самый северный аэропорт в каждом часовом поясе, т. е. самая первая строка в каждом разделе, выбирается с помощью оконной функции `first_value`.
- Строго говоря, эта функция получает доступ к первой строке *оконного кадра*, а не раздела.
- Однако когда используются правила формирования оконного кадра по умолчанию, тогда его начало совпадает с началом раздела.

```
SELECT airport_name, city, timezone, latitude,  
       first_value( latitude ) OVER tz  
       AS first_in_timezone,  
       latitude - first_value( latitude ) OVER tz  
       AS delta,  
       rank() OVER tz  
FROM airports  
WHERE timezone IN ( 'Asia/Irkutsk', 'Asia/Krasnoyarsk' )  
WINDOW tz AS ( PARTITION BY timezone  
               ORDER BY latitude DESC )  
ORDER BY timezone, rank;
```

Если используется один и тот же способ формирования разделов и порядок сортировки строк в разделах – вводится предложение **WINDOW**.

...

```
--[ RECORD 4 ]-----+-----  
airport_name      | Иркутск  
city              | Иркутск  
timezone         | Asia/Irkutsk  
latitude         | 52.268028  
first_in_timezone | 58.135  
delta            | -5.866972  
rank             | 4  
--[ RECORD 5 ]-----+-----  
airport_name      | Байкал  
city              | Улан-Удэ  
timezone         | Asia/Irkutsk  
latitude         | 51.807764  
first_in_timezone | 58.135  
delta            | -6.327236  
rank             | 5
```

```
--[ RECORD 6 ]-----+-----  
airport_name      | Норильск  
city              | Норильск  
timezone          | Asia/Krasnoyarsk  
latitude          | 69.311053  
first_in_timezone| 69.311053  
delta             | 0  
rank              | 1  
--[ RECORD 7 ]-----+-----  
airport_name      | Стрежевой  
city              | Стрежевой  
timezone          | Asia/Krasnoyarsk  
latitude          | 60.716667  
first_in_timezone| 69.311053  
delta             | -8.594386  
rank              | 2  
...
```

4.4. Подзапросы

Как в общем случае работает команда SELECT (1)

1. Сначала вычисляются все элементы, приведенные в списке после ключевого слова FROM. Если таблиц больше одной, то формируется декартово произведение из множеств их строк. При этом в комбинированных строках сохраняются все атрибуты из каждой исходной таблицы.
2. Если в команде присутствует условие WHERE, то из полученного декартова произведения *исключаются* строки, которые этому условию не соответствуют.
3. Если присутствует предложение GROUP BY, то результирующие строки группируются на основе совпадения значений одного или нескольких атрибутов, а затем вычисляются значения агрегатных функций.
4. Если присутствует предложение HAVING, то оно отфильтровывает результирующие строки (группы), не удовлетворяющие критерию.

5. Ключевое слово SELECT присутствует всегда.

- Но в списке выражений, идущих после него, могут быть не только простые имена атрибутов, но и их комбинации, созданные с использованием арифметических и других операций, а также вызовы функций.
- Причем эти функции могут быть не только встроенные, но и созданные пользователем.
- В списке выражений не обязаны присутствовать все атрибуты, представленные в строках используемых таблиц.
- Например, атрибуты, на основе которых формируются условия в предложении WHERE, могут отсутствовать в списке выражений после ключевого слова SELECT.
- Предложение SELECT DISTINCT удаляет дубликаты строк.

6. Если присутствует предложение ORDER BY, то результирующие строки сортируются на основе значений одного или нескольких атрибутов. По умолчанию сортировка производится по возрастанию значений.
7. Если присутствует предложение LIMIT или OFFSET, то возвращается только подмножество строк из выборки.

Приведенная схема описывает работу команды SELECT *на логическом уровне*, а на уровне реализации запросов в дело вступает **планировщик**, который и формирует план выполнения запроса.

- 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 );
```

```
count
-----
```

```
87224
```

```
(1 строка)
```



подзапрос

- Подзапрос является частью более общего запроса. Подзапросы могут присутствовать в предложениях SELECT, FROM, WHERE и HAVING, а также в предложении WITH.
- Здесь используется так называемый **скалярный подзапрос**. В результате его выполнения возвращается *только одно скалярное значение* (один столбец и одна строка).

Вопрос: какие маршруты существуют между городами часового пояса Asia/Krasnoyarsk.

```
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'
                        );
```

Такие подзапросы называются
некоррелированными

- 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, т. к. результат подзапроса не зависит от значений, хранящихся в таблице routes.
- Такие подзапросы называются **некоррелированными**.

```
flight_no | departure_city | arrival_city
-----+-----+-----
PG0070   | Абакан        | Томск
PG0071   | Томск         | Абакан
PG0313   | Абакан        | Кызыл
PG0314   | Кызыл         | Абакан
PG0653   | Красноярск    | Барнаул
PG0654   | Барнаул       | Красноярск
```

(6 строк)

Задача: найти самый западный и самый восточный аэропорты и представить полученные сведения в наглядной форме.

```
SELECT airport_name, city, longitude
FROM airports
WHERE longitude IN (
    ( SELECT max( longitude ) FROM airports ),
    ( SELECT min( longitude ) FROM airports )
)
ORDER BY longitude;
```

airport_name	city	longitude
Храброво	Калининград	20.592633
Анадырь	Анадырь	177.741483

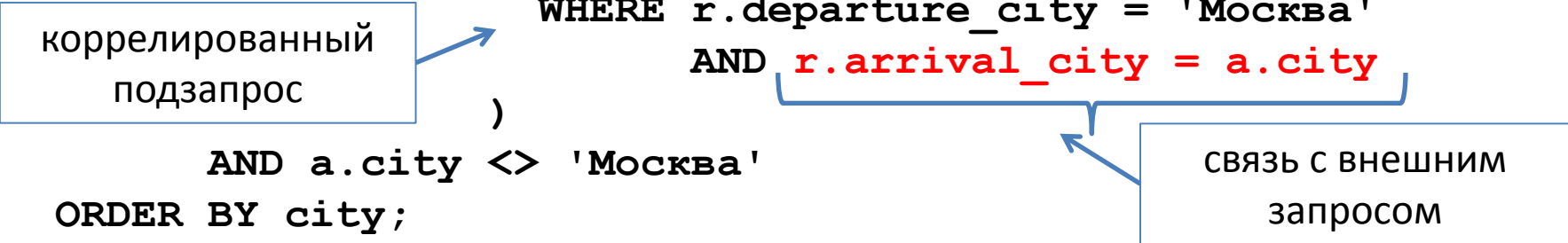
(2 строки)

Когда необходимо, наоборот, исключить какие-либо значения из рассмотрения, можно использовать конструкцию **NOT IN**.

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**).

Вопрос: в какие города нет рейсов из Москвы.

```
SELECT DISTINCT a.city      -- требуется DISTINCT
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;
```



Для каждой строки (каждого города) из таблицы `airports` выполняется поиск строки в представлении `routes`, в которой значение атрибута `arrival_city` такое же, как в *текущей строке* таблицы `airports`. Если такой строки не найдено, значит, в этот город маршрута из Москвы нет.

city

Благовещенск

Иваново

...

Якутск

Ярославль

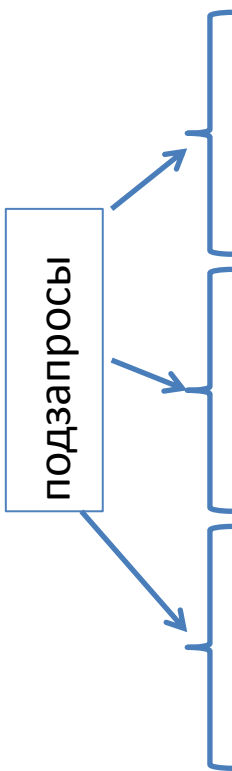
(20 строк)

- Поскольку от подзапроса в предикате EXISTS требуется только установить *факт наличия или отсутствия строк*, соответствующих критерию отбора, то в документации рекомендуется делать так:
`WHERE NOT EXISTS (SELECT 1 FROM routes r ...`
- В представленном запросе мы использовали так называемый **коррелированный (связанный) подзапрос**.
- В теории это означает, что подзапрос выполняется не один раз для всего внешнего запроса, а *для каждой строки*, обрабатываемой во внешнем запросе.
- Однако на практике важную роль играет умение планировщика (это специальная подсистема в СУБД) оптимизировать подобные запросы с тем, чтобы, по возможности, избегать выполнения подзапроса для каждой строки из внешнего запроса.

Предположим, что для выработки ценовой политики авиакомпании необходимо знать, как распределяются места разных классов в самолетах всех типов.

```
SELECT a.model,
  ( SELECT count( * )
    FROM seats s
    WHERE s.aircraft_code = a.aircraft_code
          AND s.fare_conditions = 'Business'
  ) AS business,
  ( SELECT count( * )
    FROM seats s
    WHERE s.aircraft_code = a.aircraft_code
          AND s.fare_conditions = 'Comfort'
  ) AS comfort,
  ( SELECT count( * )
    FROM seats s
    WHERE s.aircraft_code = a.aircraft_code
          AND s.fare_conditions = 'Economy'
  ) AS economy
FROM aircrafts a
ORDER BY 1;
```

Подзапросы коррелированные



model	business	comfort	economy
Airbus A319-100	20	0	96
Airbus A320-200	20	0	120
Airbus A321-200	28	0	142
Boeing 737-300	12	0	118
Boeing 767-300	30	0	192
Boeing 777-300	30	48	324
Bombardier CRJ-200	0	0	50
Cessna 208 Caravan	0	0	12
Sukhoi SuperJet-100	12	0	85

(9 строк)

Решим ту же **задачу**: для выработки ценовой политики авиакомпании необходимо знать, как распределяются места разных классов в самолетах всех типов.

```
SELECT s2.model,  
       string_agg( s2.fare_conditions || ' (' || s2.num ||  
                  ')', ' ', ' ' )  
FROM ( SELECT a.model,  
            s.fare_conditions,  
            count( * ) AS num  
      FROM aircrafts a JOIN seats s  
            ON a.aircraft_code = s.aircraft_code  
      GROUP BY 1, 2  
      ORDER BY 1, 2  
    ) AS s2  
GROUP BY s2.model  
ORDER BY s2.model;
```

агрегатная функция

подзапрос

Подзапрос формирует временную таблицу в таком виде:

model	fare_conditions	num
Airbus A319-100	Business	20
Airbus A319-100	Economy	96
Airbus A320-200	Business	20
Airbus A320-200	Economy	120
...		
Sukhoi SuperJet-100	Business	12
Sukhoi SuperJet-100	Economy	85

(17 строк)

Агрегатная функция `string_agg` отличается от агрегатных функций `avg`, `min`, `max`, `sum` и `count` тем, что она возвращает не числовое значение, а строку символов, составленную из значений атрибутов, указанных в качестве ее параметров. Эти значения берутся из сгруппированных строк.

```
      model      | string_agg
-----+-----
Airbus A319-100   | Business (20), Economy (96)
Airbus A320-200   | Business (20), Economy (120)
Airbus A321-200   | Business (28), Economy (142)
Boeing 737-300    | Business (12), Economy (118)
Boeing 767-300    | Business (30), Economy (192)
Boeing 777-300    | Business (30), Comfort (48),
                  Economy (324)
Bombardier CRJ-200 | Economy (50)
Cessna 208 Caravan | Economy (12)
Sukhoi SuperJet-100 | Business (12), Economy (85)
```

(9 строк)

Задача: 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;
```

- Благодаря использованию предложения **HAVING**, подзапрос формирует временную таблицу в таком виде:

```
city | count
-----+-----
Ульяновск | 2
Москва | 3
```

(2 строки)

- А в главном запросе выполняется соединение временной таблицы с таблицей «Аэропорты» (airports).

```
city | airport_code | airport_name
-----+-----+-----
Москва | VKO | Внуково
Москва | DME | Домодедово
Москва | SVO | Шереметьево
Ульяновск | ULV | Баратаевка
Ульяновск | ULY | Ульяновск-Восточный
```

(5 строк)

Задача: 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;
```

- Подзапрос формирует список аэропортов, которые и будут отобраны с помощью предложения **HAVING** после выполнения группировки.

departure_airport	departure_city	count
DYR	Анадырь	4
GDX	Магадан	3
PKC	Петропавловск-Камчатский	1

(3 строки)

- Это означает, что один подзапрос находится внутри другого.

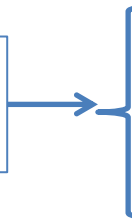
Ситуация: руководство авиакомпании хочет выяснить степень заполнения самолетов на всех рейсах, ведь отправлять полупустые самолеты не очень выгодно. Таким образом, запрос должен не только выдавать число билетов, проданных на данный рейс, и общее число мест в самолете, но должен также вычислять отношение этих двух показателей.

```
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,  
             ...
```

...

```
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;
```

ВЛОЖЕННЫЙ
ПОДЗАПРОС



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`. Конечно, можно было бы вместо такого решения использовать еще один коррелированный подзапрос:

```
( SELECT count( tf.ticket_no )  
  FROM ticket_flights tf  
  WHERE tf.flight_id = f.flight_id  
 ) AS fact_passengers
```

В таком случае уже не потребовалось бы выполнять соединение представления `flights_v` с таблицей `ticket_flights` и группировать строки, достаточно было бы сделать так:

```
FROM flights_v  
WHERE f.status = 'Arrived'  
 ) AS ts JOIN aircrafts AS a
```

...

Внешний запрос вместо кода самолета выводит наименование модели, выбирает остальные столбцы из подзапроса без изменений и дополнительно производит вычисление степени заполнения самолета пассажирами, а также сортирует результирующие строки.

```
--[ RECORD 1 ]-----+-----  
flight_id      | 28205  
flight_no     | PG0032  
scheduled_departure_local | 2016-09-13 08:00:00  
departure_city | Пенза  
arrival_city  | Москва  
model         | Cessna 208 Caravan  
fact_passengers | 2  
total_seats   | 12  
fraction      | 0.17  
--[ RECORD 2 ]-----+-----  
flight_id      | 9467  
flight_no     | PG0360  
scheduled_departure_local | 2016-09-13 08:00:00  
departure_city | Санкт-Петербург  
arrival_city  | Оренбург  
model         | Bombardier CRJ-200  
fact_passengers | 6  
total_seats   | 50  
fraction      | 0.12  
...
```

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.
- В общем табличном выражении может присутствовать больше одного подзапроса.
- Каждый подзапрос формирует временную таблицу с указанным именем.
- Если имена столбцов этой таблицы не заданы явным образом в виде списка, тогда они определяются на основе списка столбцов в предложении SELECT. В нашем примере это было именно так.
- В главном запросе можно обращаться к временной таблице так, как если бы она существовала постоянно.
- Но важно учитывать, что временная таблица, создаваемая в общем табличном выражении, существует только во время выполнения запроса.

- Мы уже решали задачу распределения сумм бронирований по диапазонам с шагом в сто тысяч рублей. Тогда мы использовали предложение **VALUES** для формирования виртуальной таблицы.
- Сначала сформируем диапазоны сумм бронирований с помощью рекурсивного CTE.

```
WITH RECURSIVE ranges ( min_sum, max_sum ) AS
( VALUES ( 0, 100000 )
  UNION ALL
  SELECT min_sum + 100000, max_sum + 100000
  FROM ranges
  WHERE max_sum < ( SELECT max( total_amount )
                    FROM bookings )
)
SELECT * FROM ranges;
```

- сначала выполняется предложение `VALUES (0, 100000)` и результат записывается во временную область памяти;
- затем к этой временной области памяти применяется запрос `SELECT min_sum + 100000, max_sum + 100000`
...
в результате его выполнения формируется только одна строка, поскольку в исходном предложении `VALUES` была сформирована только одна строка и только одна строка была помещена во временную область памяти;
- вновь сформированная строка вместе с исходной строкой помещаются в другую временную область, в которой происходит накапливание результирующих строк;
- к той строке, которая была на предыдущем шаге сформирована с помощью команды `SELECT`, опять применяется эта же команда и т. д.;
- работа завершится, когда перестанет выполняться условие `max_sum < (SELECT max(total_amount) FROM bookings)`

Важную роль в этом процессе играет предложение `UNION ALL`, благодаря которому происходит объединение сформированных строк в единую таблицу. Поскольку строк-дубликатов не возникает, поэтому мы используем не `UNION`, а `UNION ALL`.

```
min_sum | max_sum
-----+-----
          0 | 100000
100000   | 200000
200000   | 300000
300000   | 400000
400000   | 500000
500000   | 600000
600000   | 700000
700000   | 800000
800000   | 900000
900000   | 1000000
1000000  | 1100000
1100000  | 1200000
1200000  | 1300000
(13 строк)
```

Теперь давайте скомбинируем рекурсивное общее табличное выражение с выборкой из таблицы bookings:

```
WITH RECURSIVE ranges ( min_sum, max_sum ) AS
( VALUES ( 0, 100000 )
  UNION ALL
  SELECT min_sum + 100000, max_sum + 100000
  FROM ranges
  WHERE max_sum < ( SELECT max( total_amount )
                    FROM bookings )
)
SELECT r.min_sum, r.max_sum, count( b.* )
FROM bookings b RIGHT OUTER JOIN ranges r
  ON b.total_amount >= r.min_sum AND
     b.total_amount < r.max_sum
GROUP BY r.min_sum, r.max_sum
ORDER BY r.min_sum;
```



имена столбцов

min_sum	max_sum	count
0	100000	198314
100000	200000	46943
200000	300000	11916
300000	400000	3260
400000	500000	1357
500000	600000	681
600000	700000	222
700000	800000	55
800000	900000	24
900000	1000000	11
1000000	1100000	4
1100000	1200000	0
1200000	1300000	1

(13 строк)

работает внешнее
соединение



Описание материализованного представления «Маршруты» (routes)

Описание атрибута	Имя атрибута	Тип PostgreSQL
Номер рейса	flight_no	char(6)
Код аэропорта отправления	departure_airport	char(3)
Название аэропорта отправления	departure_airport_name	text
Город отправления	departure_city	text
Код аэропорта прибытия	arrival_airport	char(3)
Название аэропорта прибытия	arrival_airport_name	text
Город прибытия	arrival_city	text
Код самолета, IATA	aircraft_code	char(3)
Продолжительность полета	duration	interval
Дни недели, когда выполняются рейсы	days_of_week	integer[]

Создание материализованного представления «Маршруты» (routes) (1)

```
CREATE MATERIALIZED VIEW routes AS WITH f3 AS
( SELECT f2.flight_no, f2.departure_airport, f2.arrival_airport,
  f2.aircraft_code, f2.duration,
  array_agg( f2.days_of_week ) AS days_of_week
FROM ( SELECT f1.flight_no, f1.departure_airport,
  f1.arrival_airport,
  f1.aircraft_code, f1.duration, f1.days_of_week
FROM ( SELECT flights.flight_no, flights.departure_airport,
  flights.arrival_airport, flights.aircraft_code,
  ( flights.scheduled_arrival -
  flights.scheduled_departure
  ) AS duration,
  ( to_char( flights.scheduled_departure,
  'ID'::text )
  )::integer AS days_of_week
FROM flights
) f1
GROUP BY f1.flight_no, f1.departure_airport,
  f1.arrival_airport,
  f1.aircraft_code, f1.duration, f1.days_of_week
ORDER BY f1.flight_no, f1.departure_airport,
  f1.arrival_airport,
  f1.aircraft_code, f1.duration, f1.days_of_week
) f2
GROUP BY f2.flight_no, f2.departure_airport, f2.arrival_airport,
  f2.aircraft_code, f2.duration
)
...
```

Создание материализованного представления «Маршруты» (routes) (2)

```
SELECT f3.flight_no,
       f3.departure_airport,
       dep.airport_name AS departure_airport_name,
       dep.city AS departure_city,
       f3.arrival_airport,
       arr.airport_name AS arrival_airport_name,
       arr.city AS arrival_city,
       f3.aircraft_code,
       f3.duration,
       f3.days_of_week
FROM f3,
     airports dep,
     airports arr
WHERE f3.departure_airport = dep.airport_code AND
      f3.arrival_airport = arr.airport_code;
```


Материализованное представление «Маршруты» (routes). Объяснение (1)

- Конструкция WITH f3 AS (...), т. е. общее табличное выражение.
- Во вложенном подзапросе используется функция to_char. Второй ее параметр — «ID» — указывает на то, что из значения даты/времени вылета будет извлечен номер дня недели. Поскольку номер дня недели представлен в виде символьной строки, он преобразуется в тип данных integer.
- Таким образом, вложенный подзапрос вычисляет плановую длительность полета (столбец duration) и извлекает номер дня недели из даты/времени вылета по расписанию (столбец days_of_week).
- Подзапрос следующего уровня просто группирует строки, готовя столбец days_of_week к объединению отдельных номеров дней недели в массивы целых чисел. При этом в предложение GROUP BY включен столбец days_of_week, чтобы заменить дубликаты дней недели одним значением.

Материализованное представление «Маршруты» (routes). Объяснение (2)

- Столбец `days_of_week` включен и в предложение `ORDER BY`, чтобы агрегатная функция `array_agg` собрала номера дней недели в массив в возрастающем порядке этих номеров.
- Во внешнем запросе вызывается функция `array_agg`, которая агрегирует номера дней недели, содержащиеся в сгруппированных строках, в массивы целых чисел.
- На этом работа конструкции `WITH f3 AS (...)` завершается.
- Главный запрос выполняет соединение временной таблицы `f3` с таблицей «Аэропорты» (`airports`), причем, дважды. Это нужно потому, что в таблице `f3` есть столбец `f3.departure_airport` (аэропорт отправления) и столбец `f3.arrival_airport` (аэропорт прибытия), для каждого из них нужно выбрать наименование аэропорта и наименование города из таблицы `airports`

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>.

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 6. Запросы к базе данных выполнять с помощью утилиты `psql`, описанной в главе 2, параграф 2.2.