Министерство науки и 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)



**УЧЕБНО-МЕТОДИЧЕСКОЕ ПОСОБИЕ К ЛАБОРАТОРНЫМ РАБОТАМ**

**ПО КУРСУ «Базы данных (ЮР)»**

**Лабораторная работа № 3** **«****Введение в PostgreSQL. Основы языка SQL»**

Авторы:

Кудрявцев А.П., [kudryavtsevap@bmstu.ru](mailto:kudryavtsevap@bmstu.ru)

Фомин М.М.

Ланцберг А.В.

Москва, 2024

## Основы языка SQL

1.1. Оператор SELECT

Начнем с базовой формы оператора SELECT, который извлекает данные из одной таблицы. Ниже показан синтаксис этого оператора:

SELECT select\_list

FROM table\_name;

В этом синтаксисе:

* Сначала указывается список выбора, который может быть одним столбцом или списком столбцов в таблице, из которой извлекаются данные. Если указывается список столбцов, то необходимо поместить запятую между двумя столбцами, чтобы разделить их. Если требуется выбрать данные из всех столбцов таблицы, можно использовать сокращение – звездочка (\*) вместо указания всех имен столбцов. Список выбора select\_list может также содержать выражения или литеральные значения.
* После ключевого слова FROM следует имя таблицы table\_name, из которой вы хотите запросить данные.

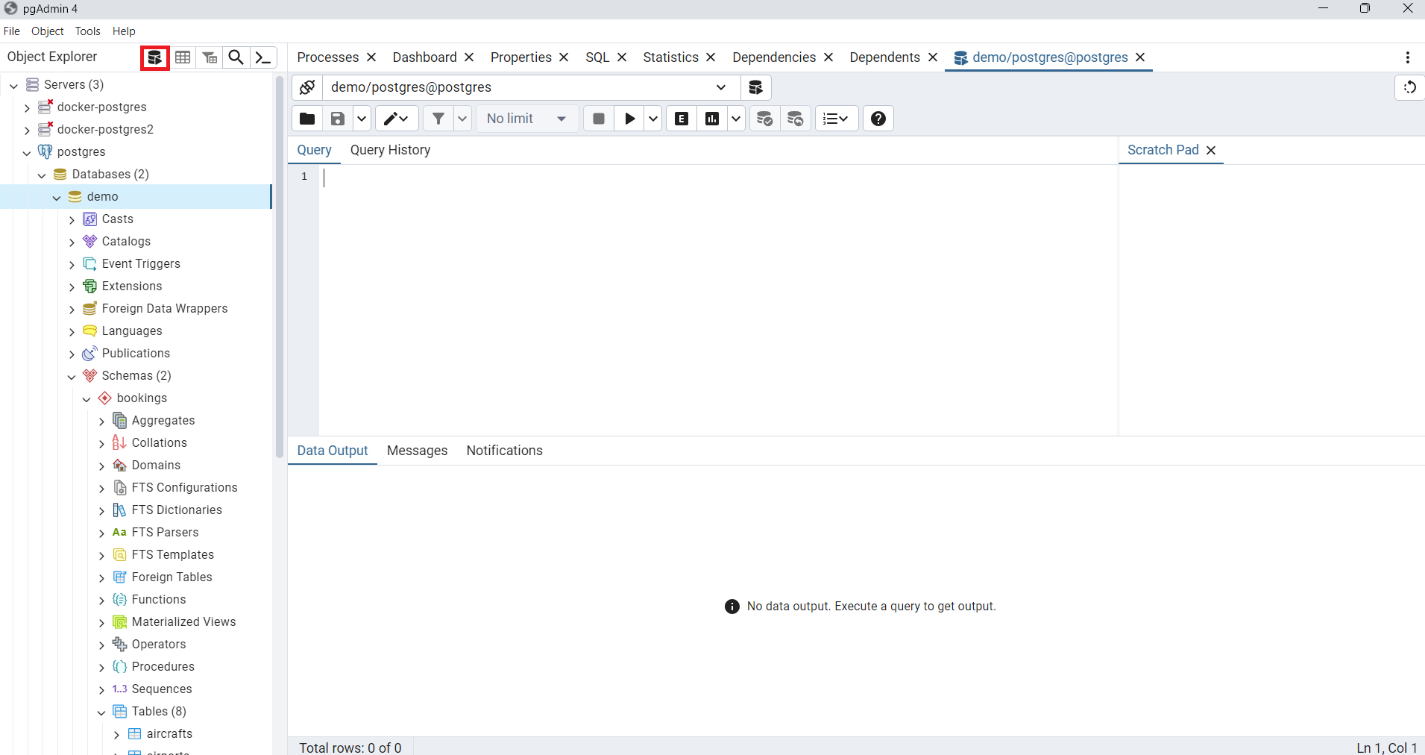
Обратите внимание, что ключевые слова SQL нечувствительны к регистру. Это означает, что SELECT — это эквивалентно select или Select. По соглашению, мы будем использовать все ключевые слова SQL в верхнем регистре, чтобы сделать запросы более удобными для чтения.

**Примеры SELECT**

Рассмотрим несколько примеров использования оператора SELECT. Для демонстрации мы будем использовать таблицу aircrafts из базы данных demo, содержащей столбцы:

* aircraft\_code – код самолета, состоящий из трех символов;
* model – модель самолета, текстовое поле;
* range – дальность полета, целое число.

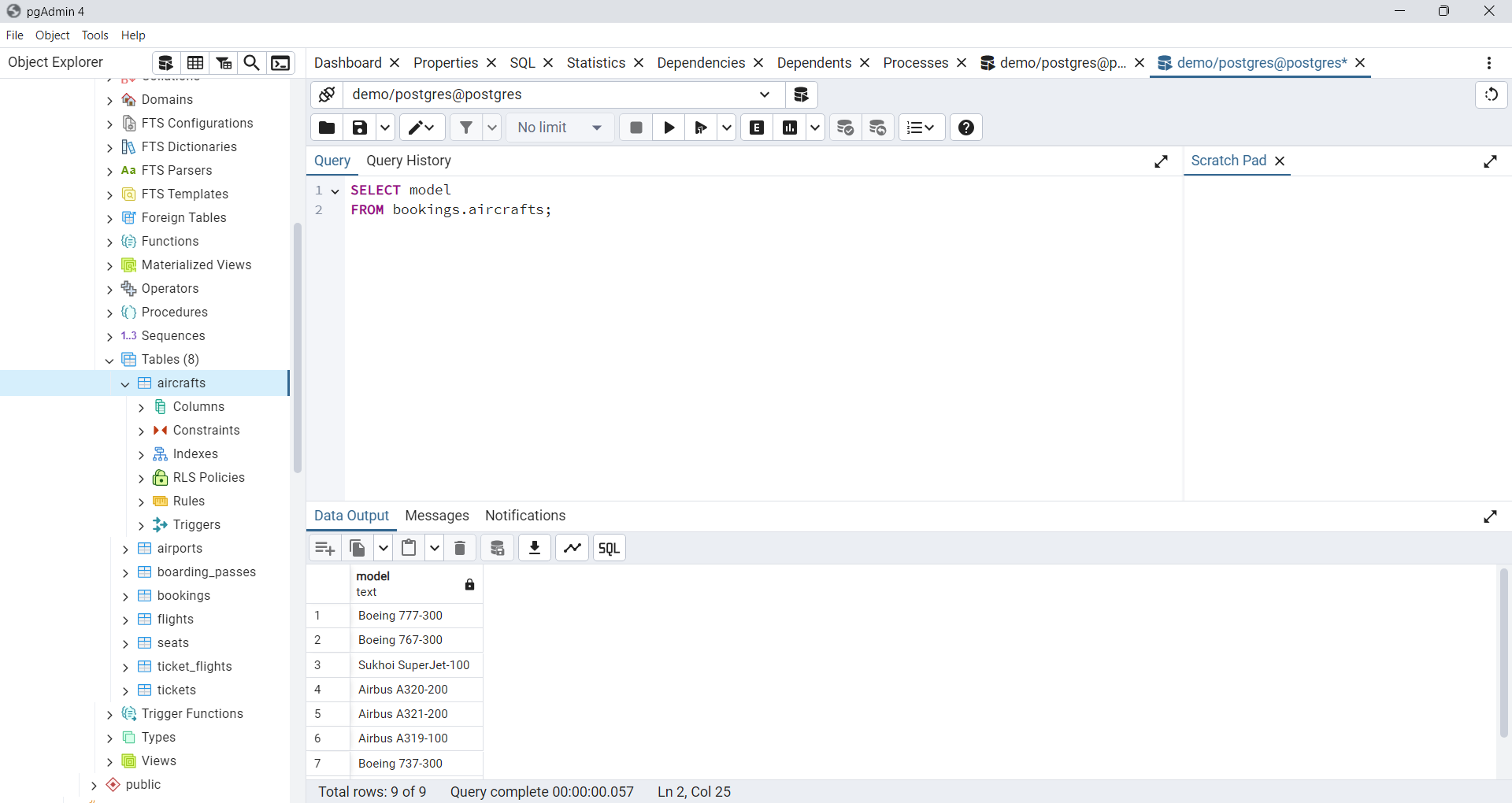
Запустим pgAdmin, подсоединимся к серверу и выделим БД demo. Все приведенные ниже примеры можно выполнять в терминале PSQL Tool, но мы воспользуемся другим инструментом Query Tool, значок которого расположен слева на панели инструментов.



В этом примере используется оператор SELECT вывода моделей самолетов из таблицы aircrafts. В правой панели вводим команду:

SELECT model

FROM bookings.aircrafts;

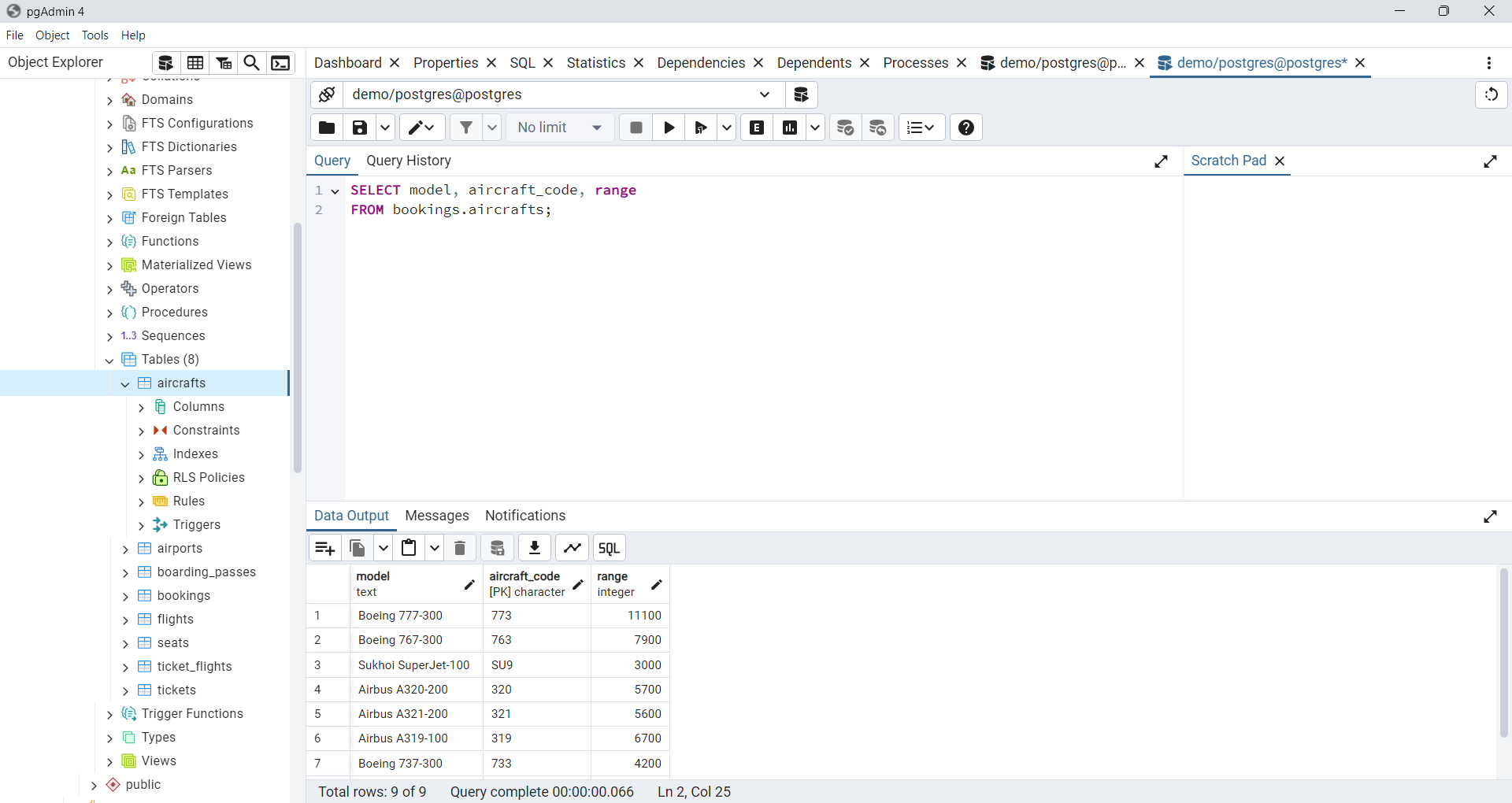


Обратите внимание, что мы добавили точку с запятой в конце оператора SELECT. Точка с запятой не является частью оператора SQL, скорее, она служит сигналом PostgreSQL, указывающим на завершение оператора SQL. Кроме того, точки с запятой используются для разделения двух или более операторов SQL.

Следующий запрос использует оператор SELECT для извлечения кода самолета, модели и дальности полета таблицы aircrafts:

SELECT model, aircraft\_code, range

FROM bookings.aircrafts;



Вместо того чтобы перечислять все столбцы в предложении SELECT по отдельности, мы можем использовать звездочку, чтобы сделать запрос короче.

Следующий запрос использует оператор SELECT \* для извлечения данных из всех столбцов таблицы aircrafts:

SELECT \*

FROM bookings.aircrafts;

Однако использование звездочки в SELECT считается плохой практикой при внедрении операторов SQL в код приложения, например Python, Java или PHP, по следующим причинам.

* *Производительность базы данных*. Предположим, у вас есть таблица со множеством столбцов и существенными данными, оператор SELECT со звездочкой выберет данные из всех столбцов таблицы, потенциально извлекая больше данных, чем требуется для приложения.
* *Производительность приложений*. Извлечение ненужных данных из базы данных увеличивает трафик между сервером PostgreSQL и сервером приложений. Следовательно, это может привести к замедлению времени отклика и снижению масштабируемости ваших приложений.

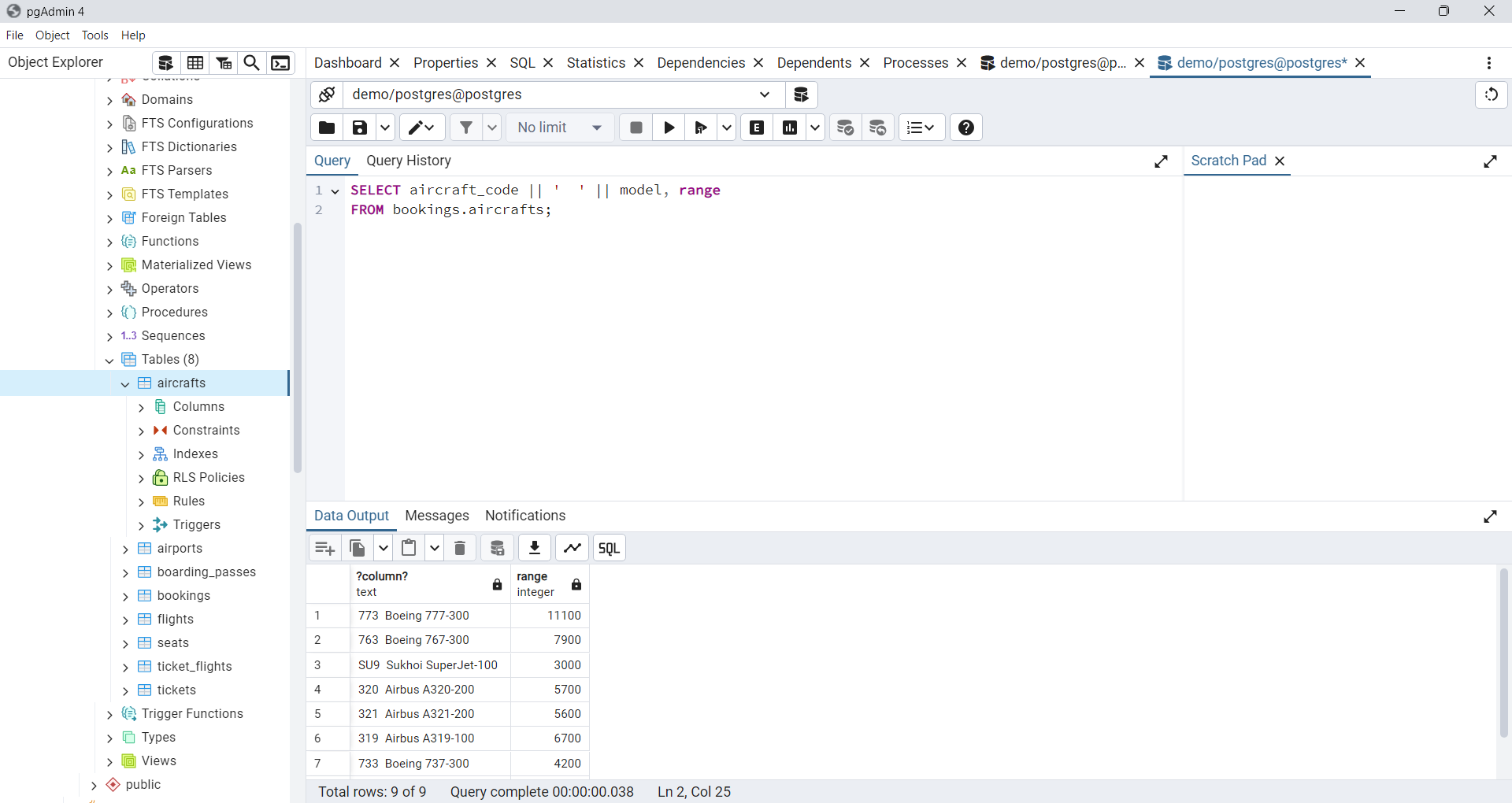
По этим причинам рекомендуется явно указывать имена столбцов в предложении SELECT, когда это возможно. Это гарантирует, что из базы данных будут извлечены только необходимые данные, что способствует более эффективным и оптимизированным запросам.

**Пример использования оператора SELECT с выражениями**

В следующем примере оператор SELECT используется для возврата кода с моделью самолета и дальностью полета:

SELECT aircraft\_code || ' ' || model, range

FROM bookings.aircrafts;



В этом примере мы использовали оператор конкатенации || для объединения кода, пробелов и модели самолета.

Обратите внимание, что первый столбец вывода не имеет имени, а ?column?. Чтобы присвоить имя столбцу в запросе, можно использовать псевдоним столбца:

выражение AS column\_alias

Ключевое слово AS необязательно. Поэтому можно использовать более короткий синтаксис:

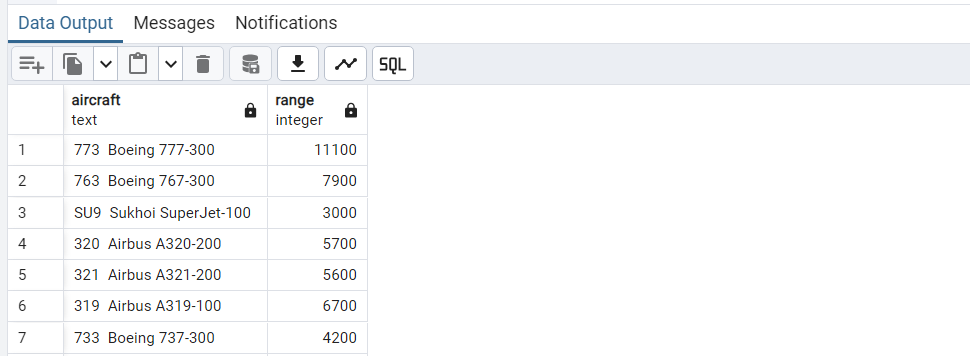
выражение column\_alias

Например, мы можем назначить псевдоним столбцу aircraft первому столбцу запроса следующим образом:

SELECT aircraft\_code || ' ' || model AS aircraft,

range

FROM bookings.aircrafts;



**Использование оператора SELECT без предложения FROM**

Предложение FROM оператора SELECT является необязательным. Поэтому его можно опустить в операторе SELECT.

Обычно предложение SELECT используете с функцией для получения результата функции. Например:

SELECT now();

В этом примере мы используем функцию now() в операторе SELECT. Она вернет текущую дату и время сервера PostgreSQL.

1.1.1. Предложение ORDER BY

При запросе данных из таблицы оператор SELECT возвращает строки в неопределенном порядке. Чтобы отсортировать строки результирующего набора, используйте ORDER BY предложение в операторе SELECT.

Предложение ORDER BY позволяет сортировать строки, возвращаемые SELECT в порядке возрастания или убывания на основе выражения сортировки. Ниже представлен синтаксис ORDER BY предложения:

SELECT select\_list

FROM table\_name

ORDER BY

sort\_expression1 [ASC | DESC],

sort\_expression2 [ASC | DESC],

...;

В этом синтаксисе.

* Сначала указываем выражение сортировки, которое может быть столбцом или выражением, которое вы хотите отсортировать после ключевых слов ORDER BY. Если хотим отсортировать набор результатов на основе нескольких столбцов или выражений, вам нужно поставить запятую между двумя столбцами или выражениями, чтобы разделить их.
* Можно использовать ASC опцию для сортировки строк в порядке возрастания и опцию DESC для сортировки строк в порядке убывания. Если опустить опцию ASC или DESC, то по умолчанию ORDER BY будет использоваться ASC.

Если в предложении SELECT есть псевдоним столбца, можно использовать его в предложении ORDER BY.

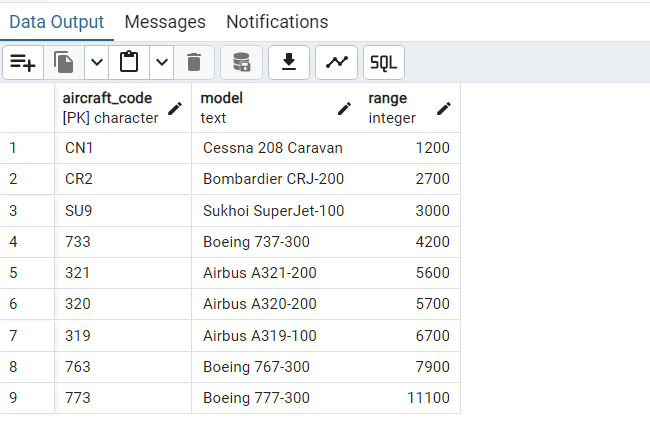
**Примеры использования ORDER BY**

Для демонстрации мы будем использовать таблицу aircrafts из базы данных demo. Следующий запрос использует предложение ORDER BY для сортировки самолетов по дальности полета в порядке возрастания:

SELECT \*

FROM bookings.aircrafts

ORDER BY range;



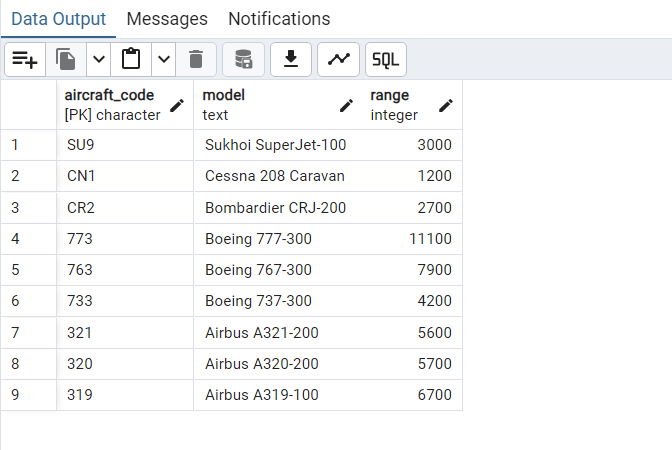
Поскольку опция ASC используется по умолчанию, то мы опустили ее в ORDER BY.

Выполним сортировку таблицы aircrafts по model в порядке убывания:

SELECT \*

FROM bookings.aircrafts

ORDER BY model DESC;



**Используем предложения ORDER BY для сортировки по выражениям**

Функция length() принимает строку и возвращает длину этой строки.

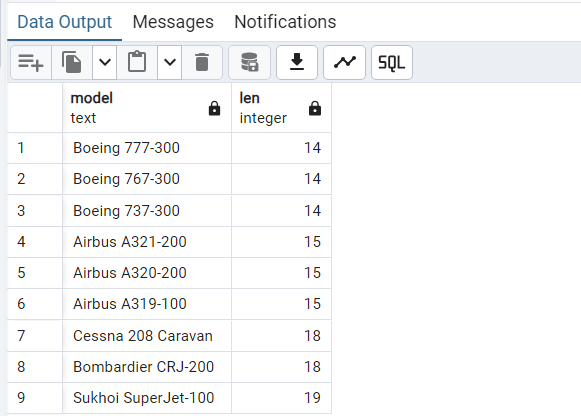
Следующий оператор выбирает модели самолётов и их длину наименования модели. Он сортирует строки по возрастанию длины наименования и убыванию для наименования модели:

SELECT model,

length(model) AS len

FROM bookings.aircrafts

ORDER BY len ASC, model DESC;



1.1.2. Предложение WHERE

Оператор SELECT возвращает все строки из одного или нескольких столбцов таблицы. Чтобы получить строки, удовлетворяющие указанному условию, используйте предложение WHERE.

Синтаксис предложения WHERE PostgreSQL следующий:

SELECT select\_list

FROM table\_name

WHERE condition

ORDER BY sort\_expression;

В этом синтаксисе предложение WHERE размещается сразу после предложения FROM оператора SELECT. В предложении WHERE используется condition для фильтрации строк, возвращаемых из SELECT предложения.

Это condition логическое выражение, которое может принимать значения «истина» (true), «ложь» (false) или «неизвестно» (NULL) Запрос возвращает только строки, которые удовлетворяют условию conditionв предложении WHERE. Другими словами, запрос будет включать только строки, которые приводят к тому, что оценочные condition значения становятся истинными в результирующем наборе.

PostgreSQL оценивает предложение WHERE после предложения FROM, но перед предложением SELECT и ORDER BY.

Если вы используете псевдонимы столбцов в предложении SELECT, вы не можете использовать их в предложении WHERE.

Для формирования условия в предложении WHERE используются операторы сравнения и логические операторы:

| **Оператор** | **Описание** |
| --- | --- |
| = | | Равный |
| > | | Больше, чем |
| < | | Меньше, чем |
| >= | | Больше или равно |
| <= | | Меньше или равно |
| <> или != | | Не равный |
| AND | | Логический оператор И |
| OR | | Логический оператор ИЛИ |
| IN | | Возвращает true, если значение соответствует любому значению в списке. |
| BETWEEN | | Возвращает значение true, если значение находится в диапазоне значений. |
| LIKE | | Возвращает true, если значение соответствует шаблону. |
| IS NULL | | Возвращает true, если значение равно NULL |
| NOT | | Отрицает результат других операторов |

**Примеры предложений WHERE**

Для демонстрации мы будем использовать таблицу рейсов flights из базы данных demo, содержащей столбцы:

* flight\_id – id записи, суррогатный ключ;
* flight\_no – номер рейса, состоящий из 6 символов;
* scheduled\_departure – запланированная дата и время вылета;
* scheduled\_arrival – запланированная дата и время прибытия;
* departure\_airport – аэропорт вылета, состоящий из трех символов;
* arrival\_airport – аэропорт прибытия, состоящий из трех символов;
* status – статус рейса, состоящий из 20 символов, может принимать   
   значения:
  + - Scheduled – рейс доступен для бронирования;
    - On Time – рейс доступен для регистрации;
    - Delayed – рейс доступен для регистрации, но задержан;
    - Departed – самолет уже вылетел и находится в воздухе;
    - Arrived – самолет прибыл в пункт назначения;
    - Cancelled – рейс отменён.
* aircraft\_code – код самолета, состоящий из трех символов;
* actual\_departure – реальная дата и время вылета;
* actual\_arrival – реальная дата и время прибытия.

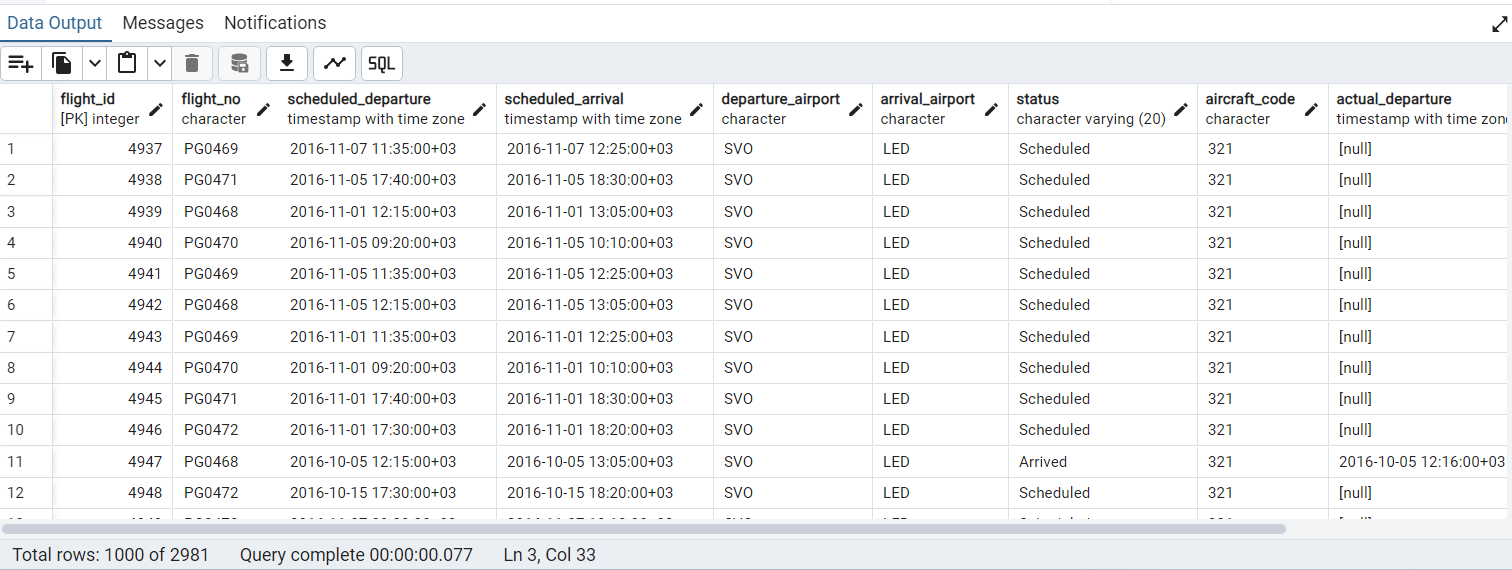
В следующем операторе используется предложение WHERE для поиска рейсов с аэродромов вылета ‘SVO’:

SELECT \*

FROM bookings.flights

WHERE departure\_airport = 'SVO';

Результат (2981 записей):



**Оператор AND**

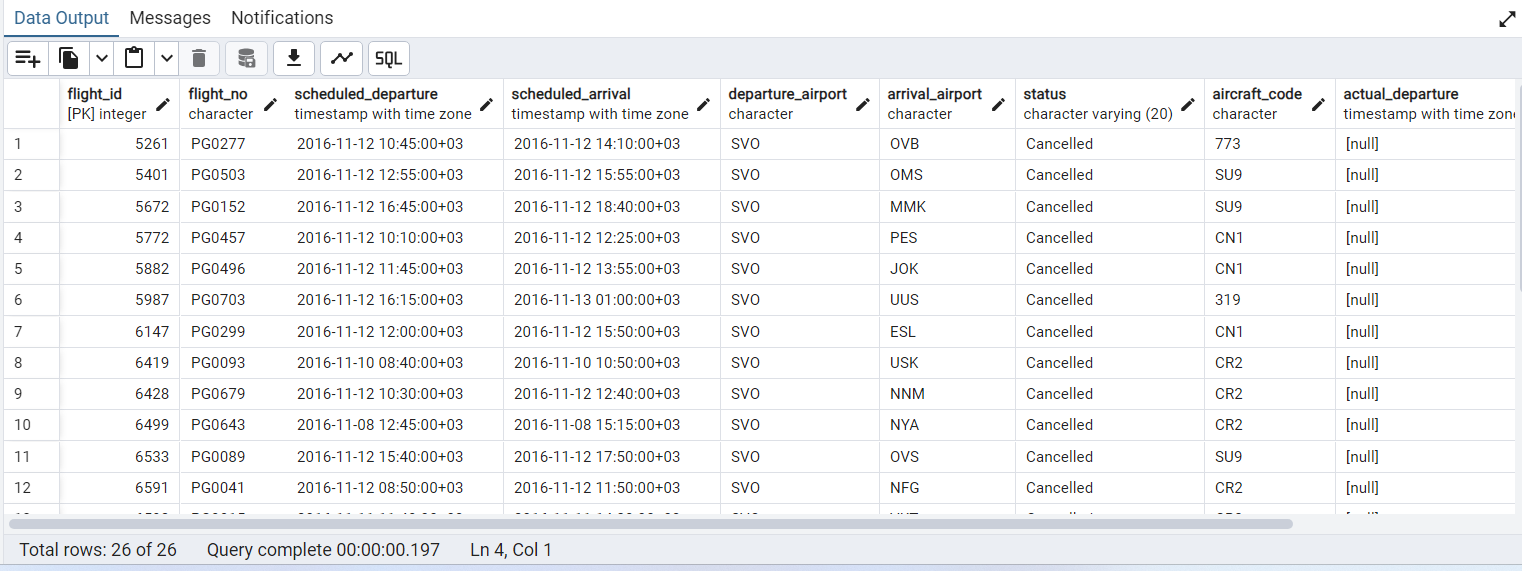
В следующем примере используется предложение WHERE с логическим оператором AND для поиска отмененных рейсов из Шереметьево:

SELECT \*

FROM bookings.flights

WHERE departure\_airport = 'SVO' AND status = 'Cancelled';

Результат (26 записей):



**Оператор IN**

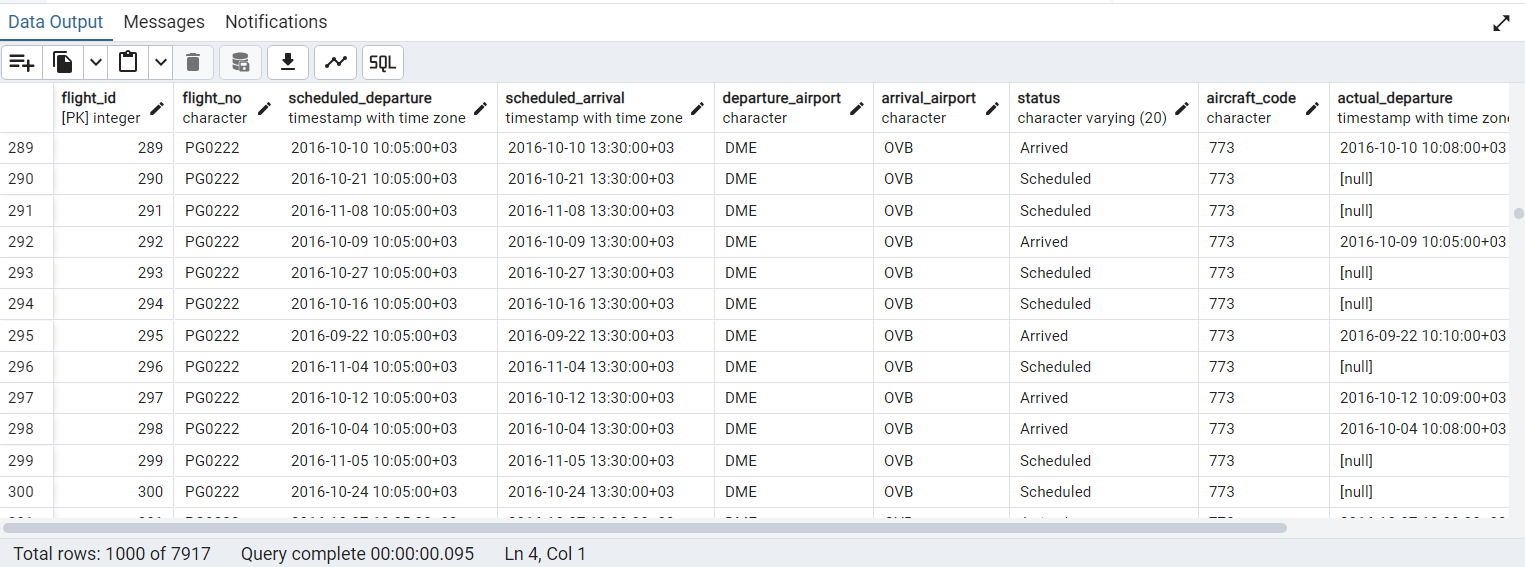
Если вы хотите найти значение в списке значений, вы можете использовать оператор IN. В следующем примере предложение WHERE используется с оператором IN для поиска рейсов из Шереметьево, Внуково и Домодедово:

SELECT \*

FROM bookings.flights

WHERE departure\_airport IN ('SVO', 'VKO', 'DME');

Результат:



**Оператор BETWEEN**

В следующем примере выполняется поиск рейсов доступных для бронирования из Шереметьево в Астрахань с 900 по 1030, используя оператор BETWEEN:

SELECT \*

FROM bookings.flights

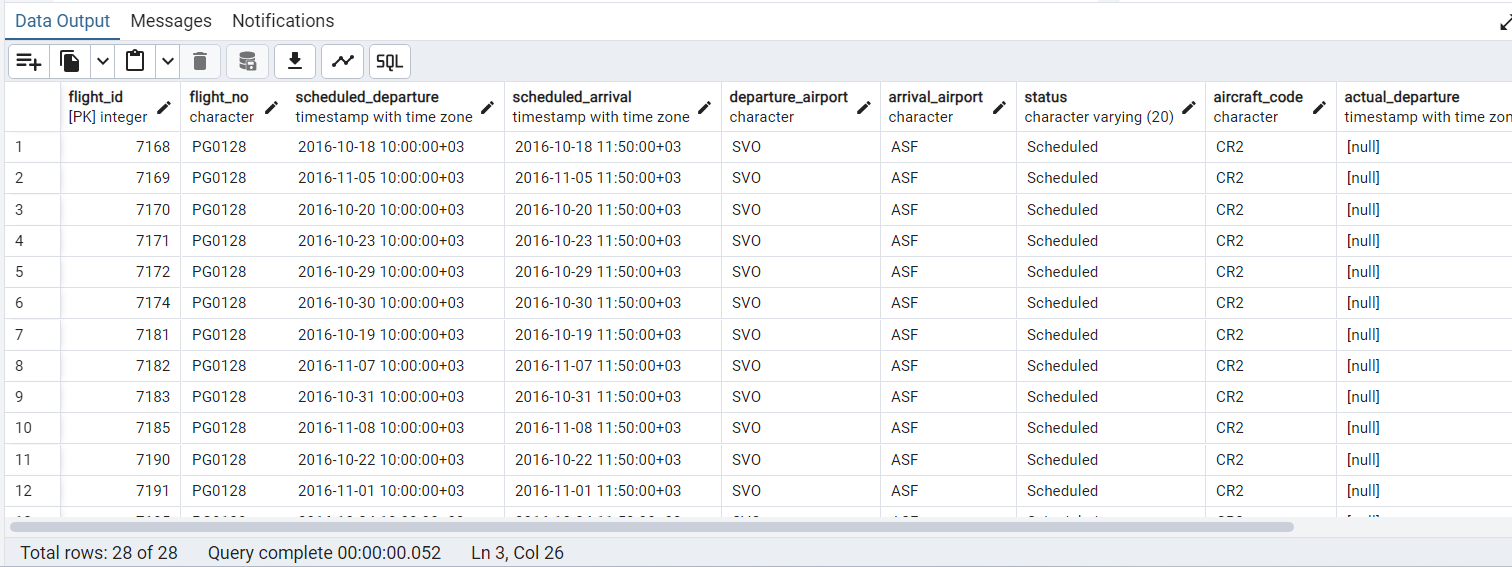
WHERE departure\_airport = 'SVO'

AND arrival\_airport = 'ASF'

AND status = 'Scheduled'

AND scheduled\_departure::time BETWEEN '09:00' AND '10:30';

Результат:



В этом запросе мы использовали приведение даты и времени scheduled\_departure запланированного вылета к типу данных время[[1]](#footnote-1) (time), чтобы сравнивать только время вылета – scheduled\_departure::time.

**Оператор LIKE**

Оператор поиска LIKE использует шаблоны, в котором используются символы подчеркивания «\_» и проценты «%»:

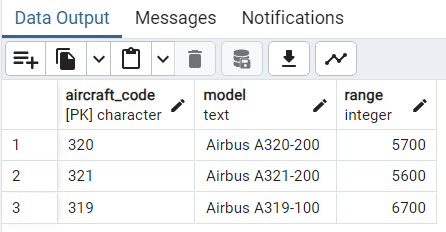
* символ подчеркивания «\_» – соответствует любому одному символу,
* символ проценты «%» – соответствует любому количеству символов.

Например, выведем те самолеты, модель которых начинается с «Airbus»:

SELECT \*

FROM bookings.aircrafts

WHERE model LIKE 'Airbus%';



1.2. Типы данных

Ниже рассмотрены наиболее часто используемые типы данных. Остальные типы данных остаются для самостоятельного изучения.

1.1.1. Логический тип

PostgreSQL поддерживает один логический тип данных, boolean который может иметь три значения: true, false и NULL.

PostgreSQL использует один байт для хранения логического значения в базе данных. Его boolean можно сократить до bool.

В стандартном SQL логическое значение может быть TRUE, FALSE, или NULL. Однако PostgreSQL достаточно гибок при работе со значениями TRUE и FALSE.

В следующей таблице показаны допустимые литеральные значения для TRUE и FALSE в PostgreSQL.

| **True** | **False** |
| --- | --- |
| true | false |
| ‘t’ | ‘f ‘ |
| ‘true’ | ‘false’ |
| ‘y’ | ‘n’ |
| ‘yes’ | ‘no’ |
| ‘1’ | ‘0’ |

Обратите внимание, что начальные или конечные пробелы не имеют значения, и все константные значения, за исключением true и false, должны быть заключены в одинарные кавычки.

Рассмотрим примеры использования логического типа. Сначала создадим новую таблицу[[2]](#footnote-2) journal\_lab\_work для регистрации выполнения лабораторной работ.

CREATE TABLE journal\_lab\_work

(

student\_id int PRIMARY KEY,

num\_lab\_work int NOT NULL,

done bool NOT NULL

);

Вставим некоторые образцы данных в таблицу journal\_lab\_work. Мы используем различные литеральные значения для булевых полей.

INSERT INTO journal\_lab\_work (student\_id, num\_lab\_work, done)

VALUES

(100, 1, TRUE),

(200, 1, FALSE),

(300, 1, 't'),

(400, 1, '1'),

(500, 1, 'y'),

(600, 1, 'yes'),

(700, 1, 'no'),

(800, 1, '0');

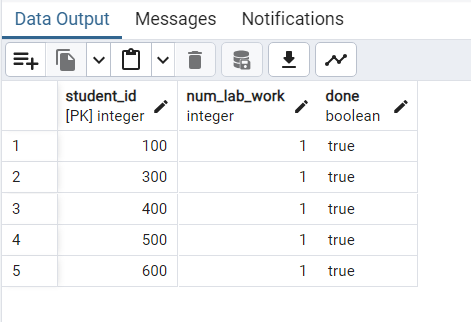
Проверим выполнение лабораторных работ[[3]](#footnote-3):

SELECT \*

FROM journal\_lab\_work

WHERE done = 'yes';

Выполняется выборка строк из таблицы journal\_lab\_work, удовлетворяющих условию done = 'yes'.



Мы можем подразумевать истинное значение, используя столбец Boolean без оператора. Например, следующий запрос возвращает все выполненные лабораторные работы студентами:

SELECT \*

FROM journal\_lab\_work

WHERE done;

1.1.2. Символьные типы

PostgreSQL предоставляет три основных типа символов:

* character(n) или char(n)
* character varying(n) или varchar(n)
* text

В этом синтаксисе n — положительное целое число, указывающее количество символов.

В следующей таблице показаны типы символов в PostgreSQL:

| **Типы персонажей** | **Описание** |
| --- | --- |
| character varying(n),  varchar(n) | переменная длина с ограничением длины |
| character(n), char(n) | фиксированной длины, с пустым заполнением |
| text, varchar | переменная неограниченной длины |

Оба char(n) и varchar(n) могут хранить до n символов. Если вы попытаетесь сохранить строку, которая содержит более n символов, PostgreSQL выдаст ошибку. Однако есть одно исключение: если все лишние символы являются пробелами, PostgreSQL обрезает пробелы до максимальной длины (n) и сохраняет обрезанные символы.

char(n) отличается от varchar(n) тем, что char(n) хранит строки фиксированной длины и, если количество символов меньше n, то строка дополняется пробелами, делая длину равную n. Дополненные пробелами значения хранятся в таблице и отображаются в этом виде. Однако эти дополнительные пробелы считаются семантически не важными. При сравнении двух значений типа char конечные пробелы игнорируются, и они будут удалены при преобразовании значений типа char в любые другие строковые типы.

Тип text данных может хранить строку неограниченной длины. Если не указать целое число n для varchar типа данных, он ведет себя как text тип данных. Производительность varchar(без размера n) и text одинакова.

Преимущество указания спецификатора длины для varchar типа данных заключается в том, что PostgreSQL выдаст ошибку, если вы попытаетесь вставить в varchar(n) столбец строку, содержащую более n символов.

В отличие от varchar, оператор character или char без указания длины (n) аналогичен оператору character(1) или char(1).

В отличие от других систем баз данных, в PostgreSQL нет разницы в производительности между тремя типами символов.

В большинстве случаев следует использовать text или varchar, а использовать varchar(n) только тогда, когда вы хотите, чтобы PostgreSQL проверил длину.

Рассмотрим пример, чтобы понять, как работают типы данных, char и varchar, text.

Сначала создайте новую таблицу с именем char\_test:

CREATE TABLE char\_test

(

id serial PRIMARY KEY,

ch char(1),

vc varchar(10),

t text

);

Затем вставим новую строку в таблицу char\_test:

INSERT INTO char\_test (ch, vc, t)

VALUES

(

'Символ',

'Тестируем varchar',

'Сохраняем очень длинную строку в поле text'

);

PostgreSQL выдал ошибку:

ERROR: value too long for type character(1)

SQL state: 22001

Это связано с тем, что тип данных столбца ch — char(1), и мы попытались вставить в этот столбец строку из трех символов.

Давайте исправим это:

INSERT INTO char\_test (ch, vc, t)

VALUES

(

'С',

'Тестируем varchar',

'Сохраняем очень длинную строку в поле text'

);

PostgreSQL выдает другую ошибку:

ERROR: value too long for type character varying(10)

SQL state: 22001

Это связано с тем, что мы попытались вставить строку, содержащую более 10 символов, в столбец vc с указанным varchar(10) типом данных.

Следующий оператор успешно вставляет новую строку в таблицу[[4]](#footnote-4).

INSERT INTO char\_test (ch, vc, t)

VALUES

(

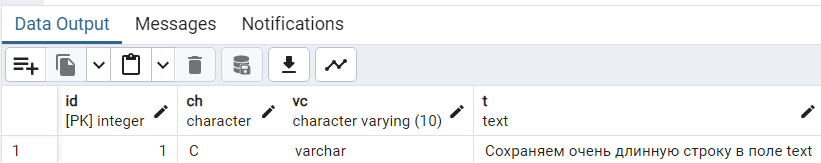
'С',

'varchar',

'Сохраняем очень длинную строку в поле text'

) RETURNING \*;

Вывод:



1.1.3. Тип данных numeric

Тип numeric может хранить числа с большим количеством цифр. Обычно используется тип numeric для хранения чисел, требующих точности, таких как денежные суммы или количества.

Вот синтаксис объявления столбца с numeric типом:

column\_name numeric(precision, scale)

В этом синтаксисе:

* precision – общее количество цифр,
* scale – количество цифр в дробной части.

Тип numeric может содержать значение, содержащее до 131072 цифр до десятичной запятой и 16383 цифр после десятичной запятой. scale типа numeric может быть нулевым, положительным или отрицательным[[5]](#footnote-5).

Ниже объявляется столбец price (цена) с числовым типом, который может хранить итоговые числа с 7 цифрами, 5 цифрами до десятичной точки и 2 цифрами после десятичной точки:

price numeric(7,2)

В следующем примере показано, как объявить столбец числового типа с нулевым scale:

quantity numeric(5, 0)

Это эквивалентно следующему объявлению, в котором явно не указана scale:

quantity numeric(5)

**Типы numeric, decimal и dec**

В PostgreSQL типы numeric и decimal являются синонимами, поэтому они взаимозаменяемы:

decimal(p,s)

Если вы предпочитаете более короткое имя, вы можете использовать имя dec, поскольку dec и decimal — это один и тот же тип:

dec(p,s)

Если точность не требуется, не следует использовать этот numeric, поскольку вычисления со numeric значениями обычно выполняются медленнее, чем с целыми числами, числами с плавающей точкой и числами двойной точности.

Рассмотрим пример использования numeric. Если вы сохраняете значение со scale большим, чем объявленный scale в numeric колонке, то PostgreSQL округлит значение до указанного количества дробных цифр. Создадим таблицу products:

CREATE TABLE products

(

id serial PRIMARY KEY,

name varchar(100) NOT NULL,

price numeric(5,2)

);

Вставим некоторые продукты, цены которых превышают точность, заявленный в столбце price:

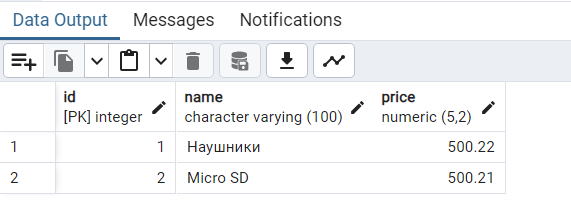
INSERT INTO products (name, price)

VALUES ('Наушники',500.215),

('Micro SD',500.214);

Следующий запрос возвращает все строки таблицы products:

SELECT \* FROM products;



Как видим, произошло округление 500.215 округлили до 500.22, а 500.214 – до 500.21.

1.1.4. Вещественное число с двойной точностью

В PostgreSQL — double precision — это тип переменной точности. Это означает, что PostgreSQL не может точно преобразовать некоторые значения во внутренний формат и может хранить их только как приближенные значения.

Ниже показано, как определить столбец с double precision типом:

column\_name double precision

В качестве альтернативы вы можете использовать тип данных float8 или float, который аналогичен double precision:

colum\_name float

Столбец типа double precision может хранить значения в диапазоне от 1E-307 до 1E+308 с точностью не менее 15 цифр.

Если вы сохраните значение, выходящее за пределы диапазона, PostgreSQL не сможет его сохранить и выдаст ошибку.

Если вы храните числа с очень высокой точностью, PostgreSQL может округлить их, чтобы соответствовать ограничению двойной точности. Это может привести к потере некоторой точности в вычислениях.

Если вы храните очень маленькие числа, близкие к нулю, PostgreSQL может выдать ошибку потери значимости из-за ограничений типа данных двойной точности, который может оказаться неспособным точно представлять такие малые значения, отличные от нуля.

На практике вы будете использовать тип двойной точности для хранения научных измерений.

1.1.5. Тип данных real

Тип real данных позволяет хранить в БД числа с плавающей запятой одинарной точности. Значение типа real занимает 4 байта дискового пространства. Его допустимый диапазон — от -3,40282347 × 1038 и 3,40282347 × 1038 (точность в пределах 6 десятичных цифр).

Обычно этот тип данных используется real для хранения чисел с плавающей точкой с относительно большими диапазонами, когда точность не имеет решающего значения, или когда вас беспокоит объем дискового пространства. Однако вы можете использовать тип данных двойной точности, если вам нужна более высокая точность.

1.1.6. Целочисленные типы данных

Для хранения целых чисел в PostgreSQL можно использовать один из следующих целочисленных типов:

* smallint
* integer
* bigint

В следующей таблице представлена ​​спецификация каждого целочисленного типа:

| **Имя** | **Размер** | **Мин.** | **Макс** |
| --- | --- | --- | --- |
| smallint | 2 байта | -32768 | +32767 |
| integer | 4 байта | -2147483648 | +2147483647 |
| bigint | 8 байт | -9223372036854775808 | +9223372036854775807 |

Если вы попытаетесь сохранить значение за пределами допустимых диапазонов, PostgreSQL выдаст ошибку. PostgreSQL не предоставляет беззнаковые целочисленные типы.

1.1.7. Псевдотип serial

В PostgreSQL последовательность — это особый вид объекта базы данных, который генерирует последовательность целых чисел. Последовательность часто используется как столбец первичного ключа (суррогатный первичный ключ) в таблице.

При создании новой таблицы последовательность можно создать через псевдотип serial следующим образом:

CREATE TABLE serial\_test

(

id serial

);

Присвоив псевдотип serial столбцу id, PostgreSQL выполняет следующее:

* сначала создайте *объект последовательность,* отвечающий за формирование значения для id и устанавливающий следующее значение для столбца в качестве значения по умолчанию;
* во-вторых, добавляет NOT NULL ограничение к id столбцу, поскольку последовательность всегда генерирует целое число, которое является известным значением;
* в-третьих, назначает владельца последовательности столбцу id; в результате чего *объект последовательность* удаляется при удалении столбца id или таблицы.

PostgreSQL предоставляет три последовательных псевдотипа smallserial, serial и bigserial со следующими характеристиками:

| **Имя** | **Размер** | **Диапазон** |
| --- | --- | --- |
| smallserial | 2 bytes | 1 – 32767 |
| serial | 4 bytes | 1 – 2147483647 |
| bigserial | 8 bytes | 1 – 9223372036854775807 |

Давайте рассмотрим пример использования столбцов serial.

Важно отметить, что serial не создает неявно индекс по столбцу и не делает столбец первичным ключом. Однако это можно легко сделать, указав PRIMARY KEY ограничение для serial столбца. Следующий оператор создает таблицу fruits с id столбцом в качестве serial столбца:

CREATE TABLE fruits

(

id serial PRIMARY KEY,

name text NOT NULL

);

Чтобы присвоить значение по умолчанию последовательному столбцу при вставке строки в таблицу, вы игнорируете имя столбца или используете DEFAULT ключевое слово в INSERT операторе. Например:

INSERT INTO fruits(name)

VALUES('Апельсин');

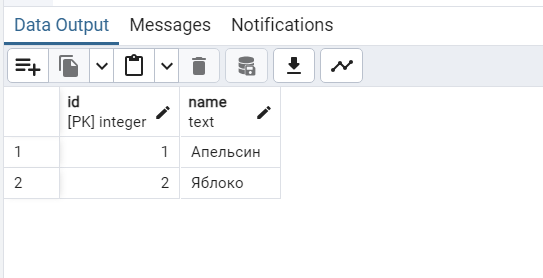
или

INSERT INTO fruits(id, name)

VALUES(DEFAULT,'Яблоко');

PostgreSQL вставил в таблицу две строки fruits со значениями столбцов id 1 и 2.

SELECT \* FROM fruits;



1.1.8. Тип данных date

PostgreSQL предлагает date тип данных, позволяющий хранить даты.

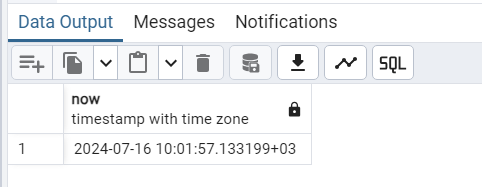
PostgreSQL использует 4 байта для хранения значения даты. Наименьшее и наибольшее значения типа date — 4713 до н.э. и 5874897 н.э. соответственно. При сохранении значения даты PostgreSQL использует формат yyyy-mm-dd, например, 2024-12-31.

**Получить текущую дату**

Чтобы получить текущую дату и время, используйте встроенную функцию now():

SELECT now();

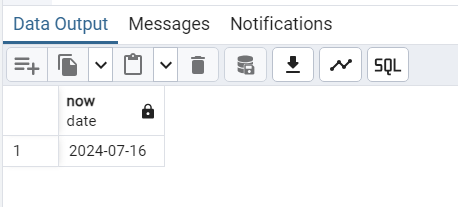
Результат:



Чтобы получить только часть даты (без части времени), используйте оператор приведения (::) для приведения значения datetime к значению date:

SELECT now()::date;

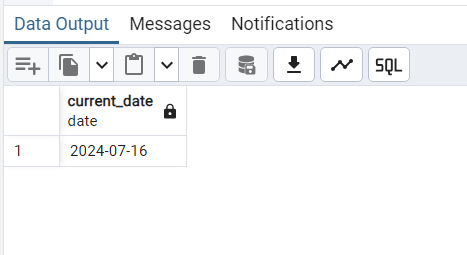
Результат:



Быстрый способ получить текущую дату — использовать функцию CURRENT\_DATE:

SELECT CURRENT\_DATE;

Результат:



**Извлечь год, месяц и день из даты**

Чтобы получить год, квартал, месяц, неделю и день из значения даты, используйте функцию extract().

В следующем выражении извлекаются год, месяц и день из текущей даты:

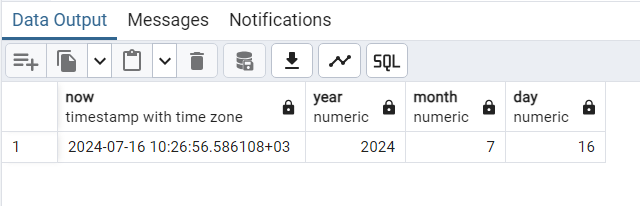
SELECT now(),

extract(YEAR FROM now()) AS year,

extract(MONTH FROM now()) AS month,

extract(DAY FROM now()) AS day;

Результат:



Функции extract позволяет извлечь квартал, декаду, день недели, часы и т.д.

1.1.9. Тип данных time

PostgreSQL предоставляет тип данных time, позволяющий хранить время в базе данных.

Вот синтаксис объявления столбца с типом time данных:

column\_name time(precision);

В этом синтаксисе precision указывает точность в десятых долях секунд для значения времени, которая находится в диапазоне от 1 до 6 знаков.

Тип time требует 8 байтов, а его допустимый диапазон — от 00:00:00 до 24:00:00.

Ниже приведены общие форматы значений time:

HH:MI

HH:MI:SS

HHMISS

Чтобы использовать значение времени с precision, можно использовать следующие форматы:

MI:SS.pppppp

HH:MI:SS.pppppp

HHMISS.pppppp

В этом синтаксисе p указывает точность.

На практике часто используется тип данных time для столбцов, которые хранят только время суток, например время события или смены. Например:

Создадим таблицу рабочих смен на некотором предприятии, назвав ее test\_time с помощью оператора CREATE TABLE:

CREATE TABLE time\_test

(

id serial PRIMARY KEY,

shift\_num int NOT NULL,

start\_at time NOT NULL,

end\_at time NOT NULL

);

Вставим несколько строк в таблицу:

INSERT INTO time\_test(shift\_num, start\_at, end\_at)

VALUES(1, '08:00:00', '12:00:00'),

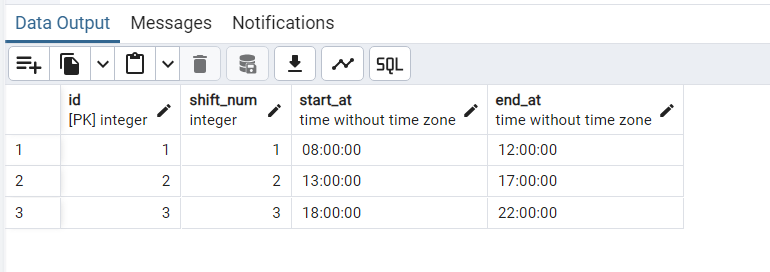
(2, '13:00:00', '17:00:00'),

(3, '18:00:00', '22:00:00');

Запросим данные:

SELECT \* FROM time\_test;

Результат:



**Время с часовым поясом**

Помимо типа данных time, PostgreSQL предоставляет тип данных время с часовым поясом time with time zone (или timetz), позволяющий хранить и управлять временем суток с учетом часового пояса.

Следующий оператор иллюстрирует, как объявить столбец, тип данных которого время с часовым поясом:

column\_name time with time zone

или

column\_name timetz

Размер хранилища типа данных time with time zone составляет 12 байт, что позволяет хранить значение времени в часовом поясе от 00:00:00+1459 до 24:00:00-1459.

Ниже приведен пример time with time zone:

2024-07-18 10:18:38.134767+03

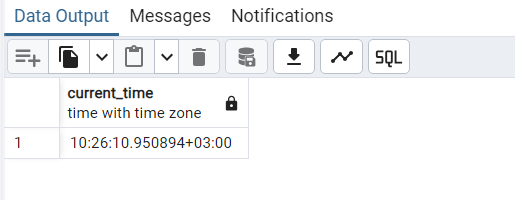
При работе с часовым поясом рекомендуется использовать timestamp вместо типа time with time zone. Это связано с тем, что часовой пояс имеет очень мало смысла, если он не связан и с датой и со временем.

**Получение текущего времени**

Чтобы получить текущее время с учетом часового пояса, используйте функцию CURRENT\_TIME:

SELECT CURRENT\_TIME;

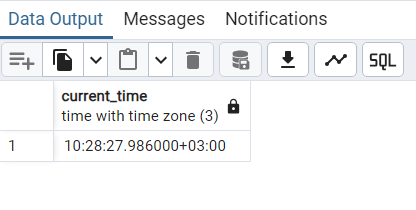
Результат:



Чтобы получить текущее время с определенной точностью, используйте функцию CURRENT\_TIME(precision):

SELECT CURRENT\_TIME(3);

Результат:

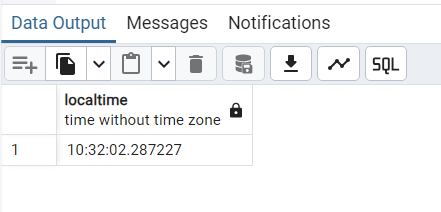


Обратите внимание, что без указания точности функция CURRENT\_TIME возвращает значение времени с максимально возможной точностью.

Чтобы узнать местное время, используйте функцию LOCALTIME:

SELECT LOCALTIME;

Результат:



Аналогично, чтобы получить местное время с определенной точностью, используйте функцию LOCALTIME(precision).

**Извлечение часов, минут и секунд из значения времени**

Чтобы извлечь часы, минуты и секунды из значения времени, используйте функцию extract следующим образом:

extract(part FROM time\_value);

part может быть часом, минутой, секундой или миллисекундой. Например:

SELECT

LOCALTIME,

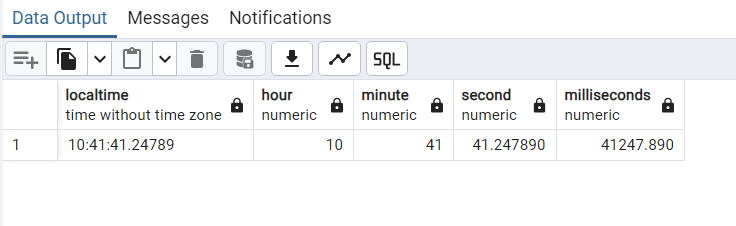
extract (HOUR FROM LOCALTIME) as hour,

extract (MINUTE FROM LOCALTIME) as minute,

extract (SECOND FROM LOCALTIME) as second,

extract (milliseconds FROM LOCALTIME) as milliseconds;

Результат:



1.1.10. Тип данных timestamp

PostgreSQL предоставляет вам два типа временных данных для обработки даты и времени:

* timestamp – дата и время без указания часового пояса;
* timestamptz – дата и время с часовым поясом.

Тип timestamp данных позволяет хранить как дату, так и время. Однако он не содержит данных о часовом поясе. Это означает, что при изменении часового пояса сервера базы данных значение timestamp, хранящееся в базе данных, не изменится автоматически.

Тип timestamptz данных — это дата и время с часовым поясом.

Обратите внимание, что timestamp и timestamptz используют 8 байт для хранения значений в базе данных.

Рассмотрим пример использования timestamp и timestamptz.

Создадим таблицу, состоящую из обоих столбцов timestamp и timestamptz.

CREATE TABLE timestamp\_test

(

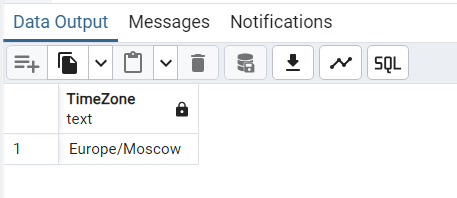
ts timestamp,

tstz timestamptz

);

Узнаем текущий часовой пояс с помощью команды:

SHOW TIMEZONE;



Вставим в таблицу timstamp\_test строку, содержащую дату и время с часовым поясом Свердловской области:

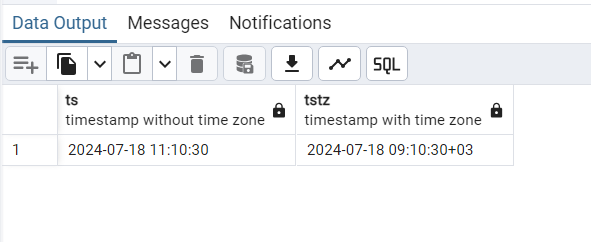
INSERT INTO timestamp\_test (ts, tstz)

VALUES('2024-07-18 11:10:30+05','2024-07-18 11:10:30+05');

После этого запросим данные из столбцов timestamp и timestamptz:

SELECT ts, tstz

FROM timestamp\_test;



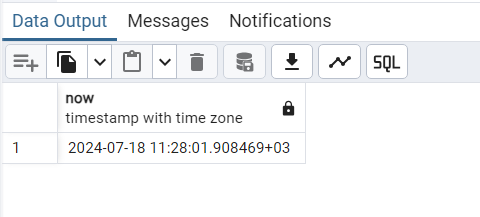
Значение в столбце timestamp не изменяется, тогда как значение в столбце timestamptz корректируется в соответствии с часовым поясом сервера 'Europe/Moscow'.

**Получение текущей даты и времени**

Чтобы получить текущую временную метку, используйте функцию now() следующим образом:

SELECT now();

Результат:



В качестве альтернативы вы можете использовать функцию CURRENT\_TIMESTAMP:

SELECT CURRENT\_TIMESTAMP;

## Практическое задание

1. Выше был рассмотрен пример использования оператора IN:  
   SELECT \*   
   FROM bookings.flights  
   WHERE departure\_airport IN ('SVO', 'VKO', 'DME');  
     
   Получить тот же результат, но не используя оператор IN.
2. Вывести строки таблицы flights, но вместо вывода поля scheduled\_departure вывести столбцы: год, месяц, день, час, минуты.
3. Выведите строки таблицы рейсов flights доступных для бронирования из Шереметьево, вылетающих ранее 900 и позже 1230, используя оператор BETWEEN.
4. В таблице tickets базы данных demo имена пассажиров passenger\_name занесены в виде «имя фамилия» латинскими буквами. Выберите всех пассажиров с именами, состоящими из пяти букв.
5. Объяснить результат запроса.  
   SELECT 'abc '::varchar(5) || '!', 'abc '::char(5) || '!';  
   Результат:  
   "abc !" "abc!"

## 3. Список использованных источников

1. Postgres Pro Standard Документация.

<https://postgrespro.ru/docs/postgrespro>

1. PostgreSQL. Основы языка SQL: учеб. пособие / Е. П. Моргунов; под ред. Е. В. Рогова, П. В. Лузанова. - СПб.: БХВ-Петербург, 2018.

1. Типы данных, в том числе time, будут рассмотрены ниже. [↑](#footnote-ref-1)
2. Создание таблиц и операции с ними рассмотрим в другой лабораторной работе. [↑](#footnote-ref-2)
3. Команда SELECT рассмотрена ниже. [↑](#footnote-ref-3)
4. Ключевое слово RETURNING \* возвращает вставленную строку. [↑](#footnote-ref-4)
5. PostgreSQL 15 и более поздние версии позволяют объявлять scale отрицательным. [↑](#footnote-ref-5)