

# Типы данных



PostgreSQL

# PostgreSQL и типы данных

- SQL — язык со строгой типизацией
  - каждый элемент данных имеет некоторый тип, определяющий его поведение и допустимое использование
- PostgreSQL наделён расширяемой системой типов, более универсальной и гибкой по сравнению с другими реализациями SQL



PostgreSQL

# Типы данных

- Символьные
- Числовые
- Дата и время
- Логические
- Двоичные
- Специальные




PostgreSQL

# Символьные данные

# Символьные данные

- **varchar(n) , char(n) , text**
- Константные значения
  - Последовательность символов, заключённая в апострофы (') - 'PostgreSQL'
    - Две строковые константы, разделённые **пробельными символами** и **минимум одним переводом строки**, объединяются в одну

`SELECT 'привет'`  
`'мир';`



	?column?
1	приветмир

`SELECT 'привет'` `'мир';`



SQL Error [42601]: ERROR: syntax error at or near "'мир'"  
Позиция: 18

- Строковая константы со спецпоследовательностями в стиле **C**
- Строковые константы со спецпоследовательностями **Unicode**
- Строковые константы, заключённые в доллары



# Константы со спецпоследовательностями в стиле C

- Начинаются с буквы **E** (заглавной или строчной)

```
SELECT 'привет\n', E'привет\nмир';
```



	?column?	?column?
1	привет\n	приветмир

Спецпоследовательность	Интерпретация
\b	Символ «збой»
\f	Разрыв страницы
\n	новая строка
\r	возврат каретки
\t	табуляция
\'	апостроф

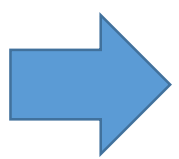


PostgreSQL

# Строковые константы со спецпоследовательностями Unicode

- Позволяют включать в строки символы Unicode по их кодам
- Начинается с **U&** (строчная или заглавная **U** и амперсанд)
- Символы Unicode можно записывать двумя способами:
  - \ и код символа из четырёх шестнадцатеричных цифр (**\043B**)
  - \+ и код символа из шести шестнадцатеричных цифр (**\+00043B**)

```
SELECT U&' \0441\043B\043E\043D' ;  
SELECT u&' \+000441\+00043B\+00043E\+00043D' ;
```



	?column?
1	слон

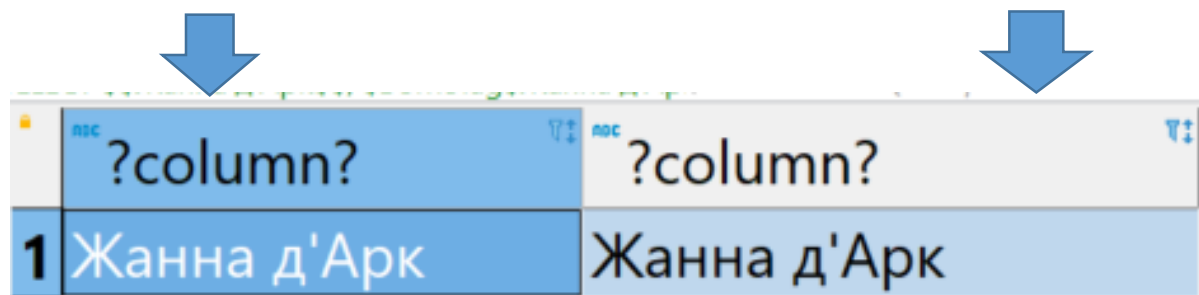
<https://unicode-table.com/>, <http://foxtools.ru/Unicode>

# Строковые константы, заклѳчѳнные в доллары

- Используются для работы со строками, содержащими много апострофов или обратных косых черт (\)
  - Позволяют избежать необходимости «зеркалирования» служебных символов
  - Делают строки более читабельными
  - Обрамляются **\$[тѳг]\$**

ТѳГ

```
SELECT $$Жанна д'Арк$$, $SomeTag$Жанна д'Арк$SomeTag$;
```



	?column?	?column?
1	Жанна д'Арк	Жанна д'Арк





PostgreSQL

# Числовые данные

# Точные числовые данные

- Целочисленные типы – **smallint** (int2), **integer** (int4), **bigint** (int8)
  - **integer** обычно оптимален с точки зрения компромисса между диапазоном допустимых значений и затратами памяти
- Числа фиксированной точности – **numeric**(precision, scale) и **decimal**(precision, scale)
  - **precision** (точность) – общее число цифр в числе
  - **scale** (масштаб) – число значащих цифр, в дробной части числа
  - Могут хранить очень большое количество цифр:  
**131072** цифры — до десятичной точки, **16383** — после точки



# Числовые данные с плавающей точкой

- **real, double precision** и **float( p )**
- Поддерживают специальные значения '**Infinity**' (бесконечность), '**-Infinity**' (отрицательная бесконечность) и '**NaN**' (не число)

Тип данных	Диапазон значений	Точность
real	от $1E^{-37}$ до $1E^{+37}$	не меньше 6 десятичных цифр
double precision	от $1E^{-307}$ до $1E^{+308}$	не меньше 15 десятичных цифр
float( p )	p = 1 до 24 → real; p = 25 до 53 → double precision	

- Если точность вводимого числа выше допустимой - будет выполняться округление значения
- При вводе слишком большого или очень маленького значения будет генерироваться ошибка

# Числовые данные с плавающей точкой

- **ВНИМАНИЕ!**

Сравнение двух чисел с плавающей точкой на предмет равенства их значений может привести к неожиданным результатам:

```
select 0.1::real * 10 = 1.0::real,  
0.1::real = 0.1::real,  
'Infinity'::real > '-Infinity'::real;
```

Результат 1			
select 0.1::real * 10 = 1.0::real, 0.1::real = 0.1::real, 'Infinity'::real > '-Infinity'::real;			
Таблица	<input checked="" type="checkbox"/> ?column?	<input checked="" type="checkbox"/> ?column?	<input checked="" type="checkbox"/> ?column?
1	[ ]	[v]	[v]



# Последовательные типы

- **serial** (int4), **bigserial**(int8) и **smallserial**(int2)
- Реализованы как удобная замена целой группы SQL-команд:
  - Создание объекта **SEQUENCE** – генератор уникальных целых чисел
  - Генерация и получение значений последовательности
- Часто используются в качестве значений суррогатного первичного ключа (Primary Key)
  - Нет необходимости указывать явное значение для вставки в поле PK

```
SELECT * FROM pg_catalog.pg_sequences ps ;
```

# Функции для работы с последовательностями

Функция	Тип результата	Описание
<code>currval('имя')</code>	<code>bigint</code>	Возвращает последнее сгенерированное значение <u>указанной</u> последовательности (которое было возвращено при последнем вызове функции <code>nextval</code> )
<code>lastval()</code>	<code>bigint</code>	Возвращает последнее сгенерированное значение <u>любой</u> последовательности (которое было возвращено при последнем вызове функции <code>nextval</code> )
<code>nextval('имя')</code>	<code>bigint</code>	Генерит и возвращает новое значение последовательности
<code>setval('имя', bigint)</code>	<code>bigint</code>	Устанавливает текущее значение последовательности
<code>setval('имя', bigint, boolean)</code>	<code>bigint</code>	Устанавливает текущее значение последовательности и флаг <code>is_called</code> , указывающий на то, что это значение уже использовалось

```
select currval('reader_id_seq'),  
       setval('reader_id_seq', 10),  
       nextval('reader_id_seq') ;
```

	123 currval ↑↓	123 setval ↑↓	123 nextval ↑↓
1	9	10	11

# Дата и время



# Дата и время

- **date, time и time with time zone (timetz)**
  - Даты обрабатываются в соответствии с григорианским календарем
  - **time** хранит время внутри суток
  - **time with time zone** хранит время с учетом смещения, соответствующего часовому поясу
  - При вводе значений их нужно заключать в одинарные кавычки, как и текстовые строки

```
SELECT '22:25:35+01'::time with time zone,  
       '22:25:35+02'::timetz,  
       '22:25:35+03'::timetz;
```

	timetz	timetz	timetz
1	00:25:35	23:25:35	22:25:35



# Форматы для ввода значений

Тип данных	Формат ввода	Пример
<b>date</b>	'yyyy-mm-dd' – не зависит от локали! 'dd mmm, yyyy' 'mmm dd, yyyy'	'2022-06-15'::date '15 Jun, 2022'::date 'Jun 15, 2022'::date 'Jun 35, 2022'::date - ERROR
<b>time</b>	'hh:mm:[ss]' 'hh:mm:[ss] am' 'hh:mm:[ss] pm'	'22:15:16'::time '10:15:16 am'::time '10:15:16 pm'::time '25:15:68'::time - ERROR
<b>time with time zone (timetz)</b>	'hh:mm:[ss]+tz' 'hh:mm:[ss] am +tz' 'hh:mm:[ss] pm +tz'	'10:25:35+01'::timetz '10:25:35 am +02'::timetz '10:25:35 pm +03'::timetz

# Временная метка (интегральный тип)

- **timestamp, timestamp with time zone (timestamptz)**
  - Получается в результате объединения типов **даты** и **времени**
  - Оба типа занимают 8 байтов
  - Значения типа **timestamptz** хранятся приведенными к нулевому часовому поясу (UTC), а перед выводом приводятся к часовому поясу пользователя

```
SELECT '2022-09-21 22:25:35'::timestamptz,  
       '2022-09-21 22:25:35'::timestamp;
```

	timestamptz	timestamp
1	2022-09-21 22:25:35	2022-09-21 22:25:35

# Тип interval

- Представляет продолжительность отрезка времени
- Формат: **quantity unit [quantity unit ...] direction**
  - **unit** – единица измерения (microsecond, millisecond, second, minute, hour, day, week, month, year, decade, century, millennium)
  - **quantity** – количество единиц измерения
  - **direction** – может принимать значение **ago** («тому назад») либо **быть пустым**

```
SELECT '1 year 2 month ago'::interval,
       '1 year 2 month'::interval,
       current_date ,
       current_date + '1 year 2 month ago'::interval,
       current_date + '1 year 2 month'::interval;
```

interval	interval	current_date	?column?	?column?
-1 years -2 mons	1 year 2 mons	2022-10-20	2021-08-20 00:00:00	2023-12-20 00:00:00

```
SELECT '200-10 1 12:59:10'::interval
```

```
200 years 10 mons 1 day 12:59:10
```

# Тип interval – альтернативный формат

- Стандарт ISO 8601:

**P**[yyyy-mm-dd][**T**hh:mm:ss]

- **P** – обязательный символ в начале строки
- **T** – разделяет дату и время

```
SELECT 'P00020215'::interval,  
       'P2Y2M15DT102020'::interval,  
       'P-00020215'::interval,  
       'P-2Y-2M15DT102020'::interval;
```

Коды единиц  
временных интервалов




Код	Значение
Y	годы
M	месяцы (в дате)
W	недели
D	дни
H	Часы
M	минуты (во времени)
S	секунды

	interval	interval	interval	interval
1	2 years 2 mons 15 days	2 years 2 mons 15 days 10:20:20	-2 years -2 mons -15 days	-2 years -2 mons +15 days 10:20:20

# Вычитание временных меток

- Значения типа **interval** можно получить при вычитании одной временной метки из другой

```
SELECT ( '20221021'::timestamp -  
         '20220821'::timestamp)::interval;
```

	 interval 
1	61 days



PostgreSQL

# Операторы даты/времени

Оператор	Описание	Пример	Результат	Тип результата
<b>date +\ - integer</b>	Добавляет\вычитает к дате заданное число дней	date '2022-09-28' + 7 '2022-09-28'::date - 7	2022-10-05 2022-09-21	date
<b>date +\ - interval</b>	Добавляет\вычитает к дате интервал	date '2022-09-28' + interval '1 hour' '2022-09-28'::date - '1 hour'::interval	2022-09-28 01:00:00 2022-09-27 23:00:00	timestamp
<b>date +\ - time</b>	Добавляет\вычитает к дате время	date '2022-09-28' + time '03:00'	2022-09-28 03:00:00	timestamp
<b>interval +\ - interval</b>	Складывает\вычитает интервалы	interval '1 day' + interval '1 hour' '1 day'::interval - '1 hour'::interval	1 day 01:00:00 1 day -01:00:00	Interval
<b>timestamp +\ - interval</b>	Добавляет\вычитает к метке времени интервал	timestamp '2022-09-28 01:00' + interval '23 hours' '2022-09-28 01:00'::timestamp - '23 hours'::interval	2022-09-29 00:00:00 2022-09-27 02:00:00	timestamp
<b>date - date</b>	Возвращает разницу между датами в днях	date '2022-10-01' - date '2022-09-28'	3	integer
<b>timestamp - timestamp</b>	Вычитает из одной отметки времени другую (преобразуя 24-часовые интервалы в дни)	'20221021'::timestamp - '20220821'::timestamp	61 days	interval

<https://postgrespro.ru/docs/postgresql/14/functions-datetime>

# Логические и двоичные данные



# Логический тип

- **boolean**
- Может иметь три состояния:
  - «**true**» - TRUE, 't', 'true', 'y', 'yes', 'on', '1'
  - «**false**» - FALSE, 'f', 'false', 'n', 'no', 'off', '0'
  - **NULL**
- Реализует трехзначную логику

```
SELECT (5=5) = TRUE,  
       (5=5) = FALSE,  
       null = 'true',  
       null = 'false',  
       null = null;
```

<input checked="" type="checkbox"/> ?column?	<input checked="" type="checkbox"/> ?column?	<input checked="" type="checkbox"/> ?column?	<input checked="" type="checkbox"/> ?column?	<input checked="" type="checkbox"/> ?column?
[v]	[ ]	[NULL]	[NULL]	[NULL]



# Двоичные типы данных

- **bytea**

- Позволяют хранить байты с кодом 0 и другими «непечатаемыми» значениями (значения вне десятичного диапазона 32..126)
- В операциях с двоичными строками обрабатываются байты в чистом виде
- Поддерживает два формата ввода и вывода (параметр ***bytea\_output***):
  - **hex** (шестнадцатеричный) - '\x коды символов в 16-ой системе'
  - **escape** (спецпоследовательностей) – '\коды символов в 8-ой системе'

```
SELECT  '\x48 45 4C 4C 4F 21 A9'::bytea as "_hex",  
'\110\105\114\114\117\041\251'::bytea as "_escape";
```

	_hex	_escape
1	HELLO!©	HELLO!©



PostgreSQL

# Приведение типов

- Приведение типов в PostgreSQL — это осуществление преобразования одного типа информации в другой.
- Для приведения типов данных в PostgreSQL используется:
  - Функция CAST (выражение AS тип)
  - выражение::тип
  - тип выражение
- Здесь:
  - «выражение» — это данные, которые нужно преобразовать
  - «тип» это тип данных, в который нужно преобразовать «выражение»
- Неявные преобразования, производимые PostgreSQL, могут влиять на результат запроса



# Пример приведения типов

```
Select    date '20220925' as dt1,  
          cast('20220925' as date) as dt2,  
          '20220925'::date as dt2;
```

	dt1	dt2	dt2
1	2022-09-25	2022-09-25	2022-09-25