Урок 2

О типах данных и функциях

В большинстве баз данных существуют следующие основные **типы данных**:

- **числовые;**
- **символьные;**
- **дата-время;**
- **флаги.**

Также бывают дополнительные типы данных, такие как массивы, JSON, UUID, бинарные типы, геоданные.

Типы данных играют важную роль в БД по нескольким причинам:

- помогают обеспечить **целостность данных**, гарантируя, что в колонках хранятся только допустимые значения (например, колонка с числовым типом данных не будет принимать строковые значения);
- **оптимизируют хранение**, позволяя эффективно использовать память. Например, хранение целочисленного типа требует меньше памяти, чем хранение этой же информации в виде строки (набора символов). Кроме того, и для числовых, и для символьных типов также существуют подтипы, ограничивающие размер, который может занимать значение колонки в памяти. Например, это может быть ограничение по максимальному количеству символов для строкового типа `VARCHAR(255)` или ограничение по максимальному значению для целочисленного типа данных (тип данных `int` имеет ограничение в 2147483647);
- **оптимизируют производительность**, позволяя быстрее выполнять нужные нам операции. Например, операции сравнения и сортировки выполняются быстрее для числовых типов данных по сравнению с символьными;
- **упрощают работу** с данными, проверяя корректность данных, которые вносятся в базу, а также делая базу данных более читаемой и понятной для разработчиков и администраторов.

Типы данных могут отличаться в зависимости СУБД, поэтому необходимо сверяться с документацией той базы, с которой вы работаете.

Функция — это подпрограмма, которая принимает на вход аргументы, выполняет определенные операции (например, вычисления или манипуляции с данными) и возвращает результат. Например, функция суммирования с двумя входными параметрами будет брать значения первого и второго аргумента, вычислять и затем возвращать их сумму.

При желании результат работы любой функции можно использовать далее, в следующей функции. Для этого необходимо вложить одну функцию в другую. В качестве примера такой вложенности можно представить функцию, которая вычисляет квадрат суммы, где сначала вычисляется сумма первой функцией, а результат уже возводится в квадрат с помощью второй.

Числа

Как и в математике, числа бывают целые и дробные. В отличие от целых чисел, в дробных есть какое-то количество знаков после запятой, которые могут быть и нулями (в таком случае дробное число будет равно целому). В разных СУБД числовые типы обозначаются по-разному, а также имеют различные диапазоны допустимых значений и размер занимаемой памяти:

- **Целочисленные** (TINYINT, SMALLINT, MEDIUMINT, INT (или INTEGER), BIGINT)
- **Десятичные дроби** (FLOAT, DOUBLE, DECIMAL, NUMERIC)

Операторы

У числовых типов есть операторы и функции. Операторы выполняют математические операции, такие как сложение, вычитание, произведение, деление. Кроме этого, в SQL есть дополнительные операторы — остаток отделения, возведение в степень и модуль числа.

Сложение	+	3 + 2 → 5 3 + 2.0 → 5.0	Остаток от деления	%	3 % 2 → 1 3 % 2.0 → 1.00
Вычитание	-	3 - 2 → 1 3 - 2.0 → 1.0	Возведение в степень	^	3 ^ 2 → 9 3 ^ 2.0 → 9.00
Произведение	*	3 * 2 → 6 3 * 2.0 → 6.0	Модуль числа	@	@ -3 → 3 @ -3.0 → 3.0
Деление	/	3 / 2 → 1 3 / 2.0 → 1.50			

Обратите внимание, что когда в операторах вы используете только целое значение — результат будет целочисленным. Если какой-нибудь из аргументов у нас дробный, то и результат будет дробным.

Функции

Для выполнения различных вычислений и преобразований числовых типов в большинстве СУБД реализованы следующие функции:

Остаток от деления	mod (числитель, знаменатель)	mod (5, 2) → 1 mod (11, 3) → 2
Целочисленное деление	div (числитель, знаменатель)	div (5, 2) → 2 div (11, 3) → 3
Округление	round (число)	round (1.54) → 2 round (27.3) → 27
Ближайшее большее целое	ceil (число)	ceil (1.54) → 2 ceil (27.3) → 28
Ближайшее меньшее целое	floor (число)	floor (1.54) → 1 floor (27.3) → 27
Возведение в степень	power (число, степень)	power (3, 3) → 27 sqrt (1.5, 3) → 3.38
Квадратный корень	sqrt (число)	sqrt (2) → 4 sqrt (1.5) → 1.22

💡 Если вы были внимательны, то могли заметить, что остаток от деления можно выполнить и с помощью оператора `%`, и с помощью функции `DIV`, на результат вычисления это не повлияет. Аналогично несколькими способами можно выполнить возведение в степень.

Если мы хотим использовать нашу СУБД в качестве калькулятора, то нам необходимо использовать оператор `SELECT`, после которого перечислить через запятую необходимые нам выражения и функции. При этом нет необходимости указывать оператор FROM, так как мы не запрашиваем данные из какой-либо таблицы.

Пример использования числовых операторов и функций:

```
SELECT 1 + 2 AS simple_sum,          -- результат 3
       1 + 2.0 AS second_sum,        -- результат 3.00
       4 - 8 AS negative_diff,       -- результат -4
       5 * 6 AS abs,                 -- результат 30
```

```
        round(2.5) as our_round,    -- результат 3.00
        ceil(2.5) AS ceil,          -- результат 3.00
        floor(2.5) AS floor,        -- результат 2.00
        power(3, 5) AS power,       -- результат 243
        sqrt(25) AS square_root     -- результат 5.00
```

Символьные типы

Символьные типы включают в себя любые символы, строки, буквы или даже предложения. Как и числовые типы, символьные в разных СУБД описываются по-разному (TEXT, STRING, CHAR, VARCHAR), а также имеют различные диапазоны длин и размер занимаемой памяти. Поэтому необходимо сверяться с документацией той базы данных, с которой вы работаете.

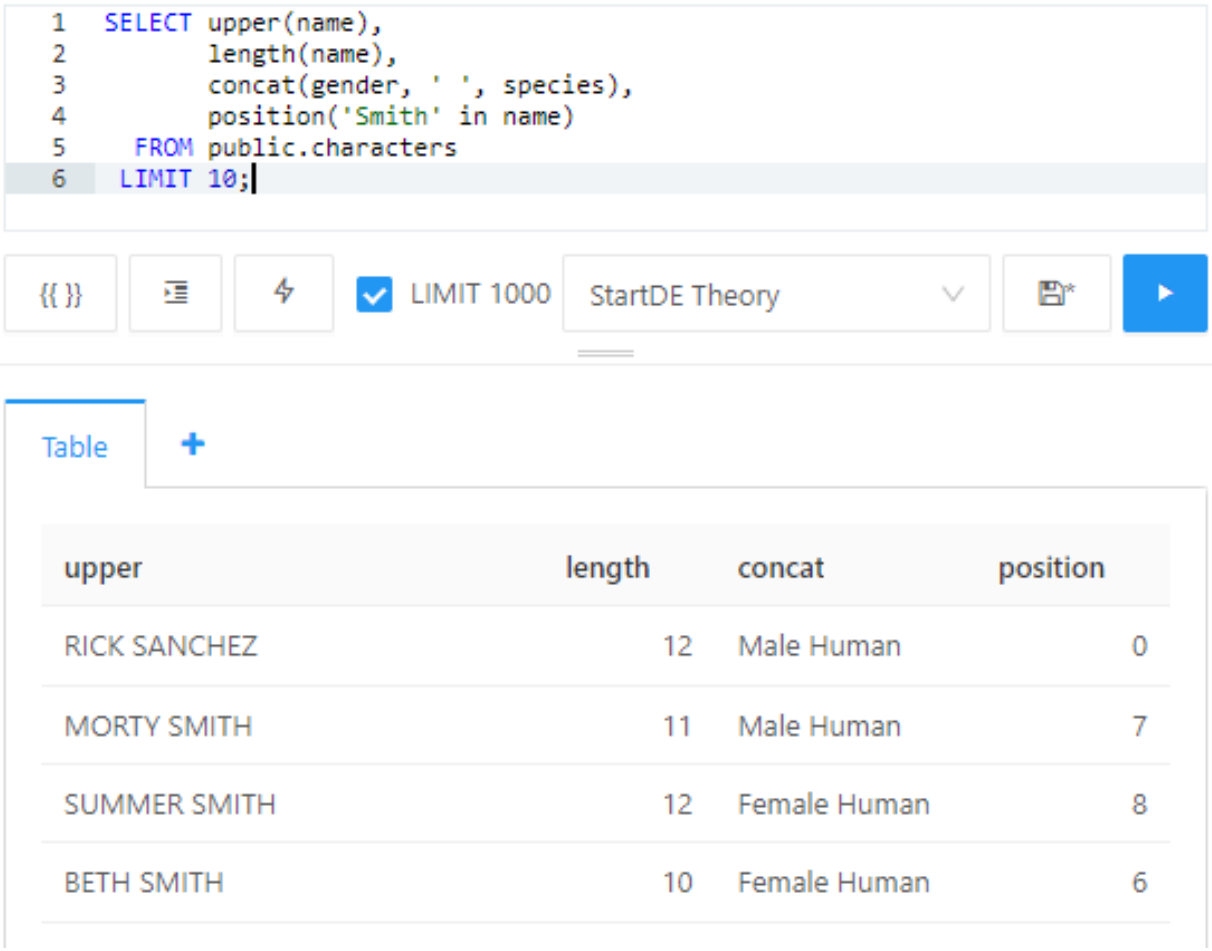
Функции

Объединение	concat (a, b, c)	concat ('У Вали было ', 2, ' апельсина') → 'У Вали было 2 апельсина' concat (1, 2, 3) → '123'
Длина строки	length (строка)	length ('У Вали было 2 апельсина') → 23 length ('123') → 3
Обрезание пробелов	trim (строка)	trim (' текст ') → 'текст' trim (' 123 ') → '123'
Поиск подстроки	position (подстрока in строка)	position ('ель' in '2 апельсина') → 5 position ('4' in '123') → 0
Верхний регистр	upper (текст)	upper ('текст') → 'ТЕКСТ' upper ('123') → '123'
Нижний регистр	lower (текст)	lower ('ТЕКСТ') → 'текст' lower ('123') → '123'

Обратите внимание, что строки необходимо заключать в кавычки, которые могут быть как одинарными, так и двойными. Также следует помнить, что при подсчете длины строки считаются и пробелы.

Пример использования символьных функций:

```
SELECT upper(name),                      -- имя в верхнем регистре
       length(name),                     -- количество символов в
       concat(gender, ' ', species),     -- объединение пола и вид
       position('Smith' in name)         -- позиция подстроки
  FROM public.characters
 LIMIT 10;
```



Дата и время

В SQL есть четыре типа даты и времени:

- Date — просто дата
- DateTime — дата и время

- Time — только время
- Timestamp

Timestamp — это число, которое содержит количество секунд, прошедших с 1 января 1970 года. Этот формат принят в Unix-системах и часто он полезнее, чем дата или время. Как правило, пользователи не задумываются над тем, в каком часовом поясе находится тот или иной сервер. Timestamp всегда отсчитывает секунды по UTC, при каждом запросе мы получаем абсолютное время, которое сервер переводит в текущий для нас часовой пояс. Но под капотом оно не будет отличаться в зависимости от места нахождения сервера.

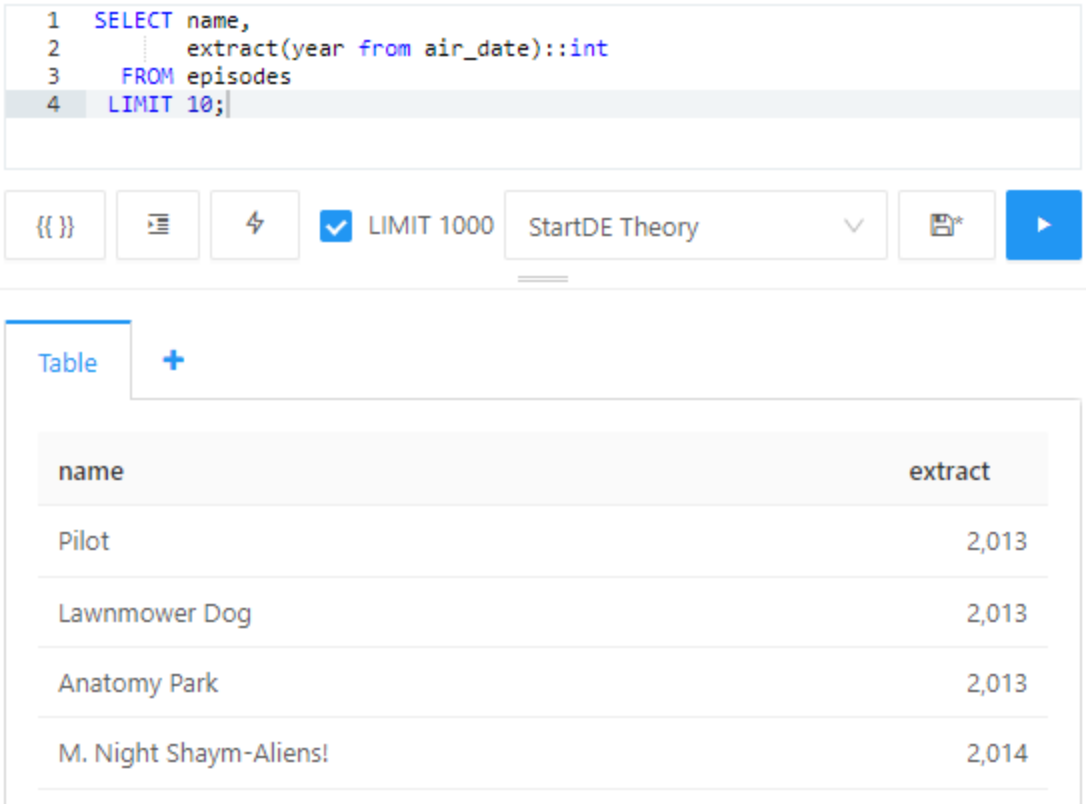
Функции

Текущая дата и время	now ()	now() → date '2026-05-04 12:44:00'
Прибавление	дата + число	date '2001-09-28' + 7 → date '2001-10-05'
Убавление	дата - число	date '2001-09-28' - 7 → date '2001-09-21'
Разница	дата - дата	date '2001-10-01' - date '2001-09-28' → 3
Часть	extract(часть from дата)	extract(year from date '2001-10-01') → 2001

В примерах выше обратите внимание, что `DATE` является конструктором даты, который явно указывает, что строка '2001-09-28' должна интерпретироваться как дата.

Пример использования функций даты и времени:

```
SELECT name,                                -- имя
       extract(year from air_date)::int     -- год
  FROM episodes;
```



Флаг (Boolean)

Данный тип чаще всего описывается как Boolean или Bool. Внутри базы данных он хранится как 1 или 0, а в интерфейсе обычно показывается как True или False. Это флаг, который отвечает на определенный вопрос, например:

- сдал ли студент зачет;
- оплачена ли покупка;
- есть ли пользователю 18 лет;
- пройден ли онбординг.

Функции

Логическое “Не”	NOT	NOT true → false	NOT false → true
Логическое “И”	AND	true AND false → false	true AND true → true
Логическое “Или”	OR	true OR false → true	false OR false → false
Равно	=	5 = 5 → true	false = false → true
Больше	>	5 > 5 → false	7 > 5 → true
Больше или равно	>=	5 >= 5 → true	7 >= 5 → true
Меньше	<	5 < 5 → false	5 < 7 → true
Меньше или равно	<=	5 <= 5 → true	5 <= 7 → true
Между	BETWEEN	5 BETWEEN 3 AND 7 → true	3 BETWEEN 5 AND 7 → false

В SQL приоритетность операторов выполнения логических операций определяет порядок, в котором эти операции будут выполняться. Знание приоритетности операторов помогает правильно формировать запросы и избегать ошибок в логике.

Приоритетность операторов в SQL:

- Операторы сравнения `=`, `!=`, `<`, `>`, `<=`, `>=`, `<>`, `BETWEEN`, `LIKE`, `IN`, `IS NULL`, `IS NOT NULL` имеют самый высокий приоритет среди логических операций и выполняются первыми при оценке выражений
- Логический оператор `NOT` имеет более высокий приоритет, чем `AND` и `OR`, и используется для инверсии логического значения
- Логический оператор `AND` имеет средний приоритет и используется для объединения двух логических выражений, оба из которых должны быть истинными, чтобы результат работы оператора вернул True, иначе он вернет False
- Логический оператор `OR` имеет самый низкий приоритет и используется для объединения двух логических выражений, одно из которых должно быть истинным, чтобы результат работы оператора вернул True, иначе он вернет False

Оператор Case

Оператор `CASE` в SQL является мощным инструментом для создания условных логических выражений, позволяющих выполнять различные действия в зависимости от заданных условий. Он может быть применен в `SELECT`, `UPDATE` и других операторах для выбора данных, их агрегации, обновления и создания вычисляемых полей. Синтаксис оператора похож на условные конструкции `if-else` в языках программирования и может быть применен в различных сценариях.

```
CASE                                    -- оператор
    WHEN condition_1 THEN result_1      -- условие_1
    WHEN condition_2 THEN result_2      -- условие_2
    ...
    ELSE result_N                       -- условие_N
END                                     -- конец логического выра
```

Пример применения логических функций и оператора CASE:

```
SELECT name,
       CASE WHEN dimension = 'Dimension C-137' THEN 'Earth'
              ELSE 'not Earth' END AS Earth,
       type = 'Planet' AS is_planet,
       id > 10 AS not_top10,
         (type = 'Planet')::int AS is_planet_int
  FROM locations
 LIMIT 100;
```

```sql
SELECT name,
       CASE WHEN dimension = 'Dimension C-137' THEN 'Earth'
            ELSE 'not Earth' END AS Earth,
       type = 'Planet' AS is_planet,
       id > 10 AS not_top10,
       (type = 'Planet')::int AS is_planet_int
  FROM locations
 LIMIT 100;
```

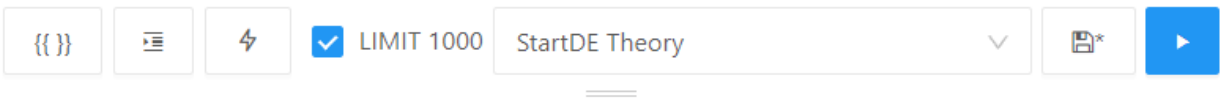


Table +

name	earth	is_planet	not_top10	is_planet_int
Earth (C-137)	Earth	true	false	1
Abadango	not Earth	false	false	0
Citadel of Ricks	not Earth	false	false	0
Worldender's lair	not Earth	true	false	1

Преобразование типов

Мы уже сталкивались с тем, что результатом сложения целого и дробного чисел является дробное число. А также с преобразованием числа в текст при конкатенации текста и числа. Это примеры **неявного преобразования**, когда СУБД понимает, что тип данных одного столбца можно преобразовать в тип данных другого без явного указания.

Также мы можем явно указать, что тип данных должен быть преобразован в другой тип, например, преобразовав текст в число, и работать с ним как с числом. Для этого используется **явное преобразование**, которое осуществляется помощью двух двоеточий `::` или с помощью функции `CAST`.

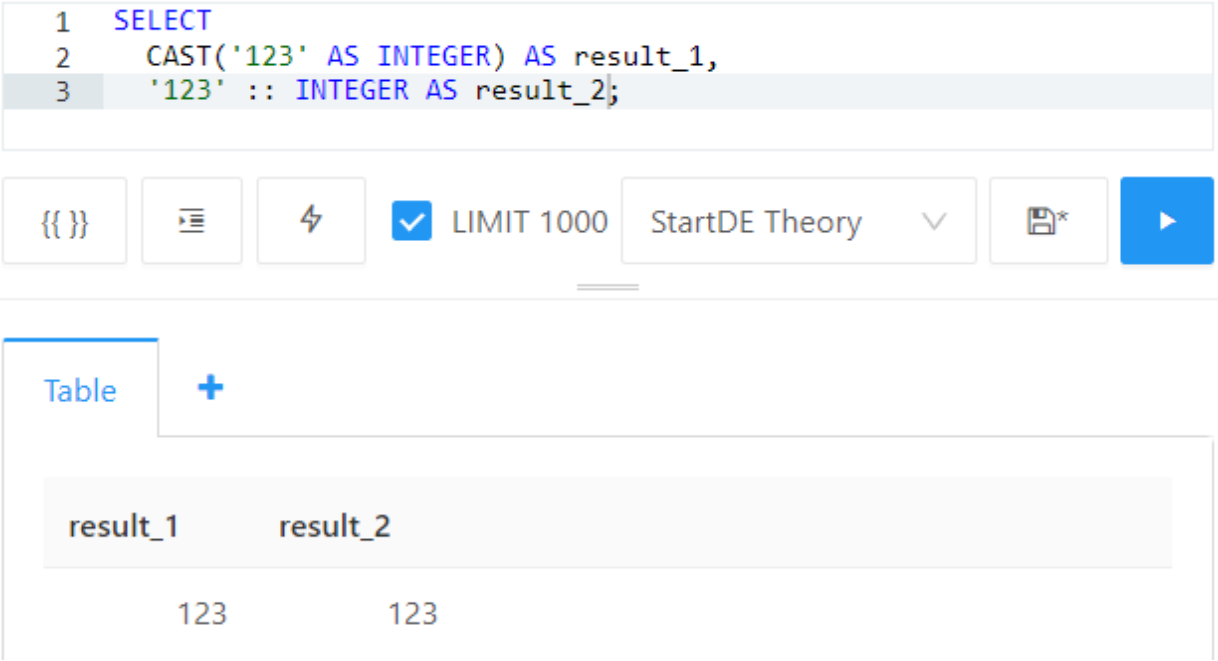
Правилом хорошего разработчика является «явное лучше неявного». Поэтому даже если СУБД преобразовывает все внутри себя, лучше сделать это преобразование явным. Важно помнить это правило, поскольку не все СУБД поддерживают одинаковые правила неявного преобразования типов, что может привести к проблемам при переносе кода между различными

СУБД. Также неявные преобразования могут быть менее производительными, так как СУБД перед преобразованием должна дополнительно определить тип.

> 💡 Явное преобразование делает работу кода более предсказуемой.

Пример преобразования:

```
SELECT
  CAST('123' AS INTEGER),
  '123' :: INTEGER;
```



Дополнительные типы

- **Массивы** — это набор значений одного типа. Например, мы не хотим добавлять 255 колонок и складываем все в единый массив в одну колонку;

- **JSON** — это текстовый формат для работы со структурированными данными. Некоторые СУБД позволяют работать с этим типом не как со строкой, а как с полноценной структурой, быстро находя нужные значения (как в Python);
- **UUID** — это стандарт идентификации. Обычно он генерируется программно, и процент совпадений таких UID-ов очень низок. Это альтернатива ID, который мы заполняем возрастающими значениями;
- **Бинарные типы** позволяют хранить картинки, видео и прочие файлы. С данными типами нет смысла работать с помощью SQL, поэтому на практике файлы хранят в отдельном файловом хранилище, а в таблице хранят ссылку на файл;
- **Геоданные** позволяют указывать, например, точку или область на карте. Некоторые СУБД предоставляют такой функционал для работы.

Разные СУБД представляют разные типы, и это только самые распространенные дополнительные типы.

Алиас

Чтобы дать название тому, что возвращает нам функция или выражение, мы можем использовать псевдоним или алиас (alias). Для этого нам необходимо использовать инструкцию «AS» после нашей функции или нашего выражения. Технически можно её не использовать, это не будет ошибкой. Однако в таком случае ухудшается читаемость больших запросов, поэтому все же рекомендуется использовать псевдонимы.

Посмотрим на один из примеров, приведенных в этом уроке:

```
SELECT name,
       CASE WHEN dimension = 'Dimension C-137' THEN 'Earth'
              ELSE 'not Earth' END AS Earth,
       type = 'Planet' AS is_planet,
       id > 10 AS not_top10,
         (type = 'Planet')::int AS is_planet_int
```

```
 FROM locations
LIMIT 100;
```

В полученной таблице для четырех из пяти столбцов мы явно прописали их названия, то есть алиасы: Earth, is_planet, not_top10, is_planet_int. А что будет выведено, если их не прописать, предлагаем вам проверить самостоятельно 🙂

Резюме

- В большинстве баз данных существует несколько основных **типов данных**:
  - **Числовые**. Среди них выделяют целочисленные (TINYINT, SMALLINT, MEDIUMINT, INTEGER, BIGINT) и десятичные дроби (FLOAT, DOUBLE, DECIMAL, NUMERIC)
  - **Символьные** (TEXT, STRING, CHAR, VARCHAR)
  - **Дата-время** (DATE, DATETIME, TIME, TIMESTAMP)
  - **Флаги** (BOOLEAN или BOOL)
- Разные типы предназначены для разных диапазонов значений (или количества символов) и количества занимаемой памяти
- Преобразование типов в СУБД бывает явное и неявное. Явное преобразование (с помощью оператора `CAST` или `::`) делает работу кода более предсказуемой
- Функция — это подпрограмма, которая принимает на вход аргументы, выполняет определенные операции (например, вычисления или манипуляции с данными) и возвращает результат. При желании результат работы любой функции можно использовать далее, в следующей функции. У числовых типов помимо функций есть операторы, которые выполняют математические операции
- Оператор `CASE` — это мощный инструмент в SQL, который позволяет создавать условия для выполнения различных операций в зависимости от значения столбца или выражения

- Хорошей практикой считается присвоение алиаса (псевдонима) результату вычисления выражения или работы функции