

# Шпаргалка по базам данных / SQL Cheatsheet (XeLaTeX)

Краткий справочник по основным операциям

April 2, 2025

## Contents

1	Базовый Синтаксис и Извлечение Данных	1
2	Фильтрация Данных (WHERE)	1
3	Сортировка Результатов (ORDER BY)	2
4	Агрегатные Функции и Группировка (GROUP BY, HAVING)	2
5	Объединение Таблиц (JOIN)	2
6	Подзапросы (Subqueries)	3
7	Общие Табличные Выражения (CTE - Common Table Expressions)	3
8	Оконные Функции (Window Functions)	3
9	Работа с NULL	4
10	Типы Данных и Преобразования	4

## 1 Базовый Синтаксис и Извлечение Данных

### Основы SELECT и FROM

Извлечение данных – хлеб с маслом аналитика. Начинаем с простого: выбираем нужные столбцы из нужной таблицы.

#### Выбор конкретных столбцов

```
1 SELECT column1, column2 -- Выбираем только нужные колонки
2 FROM table_name;        -- Указываем, из какой таблицы
```

#### Выбор всех столбцов

```
1 SELECT *                -- Выбрать ВСЕ столбцы
2 FROM table_name;
3 -- Внимание: Будь осторожен с '*' на больших таблицах.
4 -- Это может быть медленно и неэффективно.
5 -- Лучше явно перечислять нужные столбцы.
```

## Псевдонимы (AS) и Уникальность (DISTINCT)

Делаем запросы читаемыми и получаем только уникальные записи.

### Псевдонимы (Aliases)

```
1 SELECT
2   user_id AS id, -- Переименовываем user_id в id
3   registration_date AS reg_date -- и
4   registration_date в reg_date
5 FROM
6   users AS u; -- Переименовываем таблицу users в u (удобно для JOIN'ов)
7 -- AS часто можно опустить: SELECT user_id id ...
```

### Уникальные значения

```
1 SELECT DISTINCT city -- Получить список уникальных городов
2 FROM users;
```

## Ограничение Вывода (LIMIT / TOP)

Часто нужно просто взглянуть на первые несколько строк, а не тащить всю таблицу.

### LIMIT (PostgreSQL, MySQL, SQLite)

```
1 SELECT user_id, name
2 FROM users
3 LIMIT 10; -- Показать только первые 10 строк
```

### TOP (SQL Server, MS Access)

```
1 SELECT TOP 10 user_id, name
2 FROM users; -- Аналогично LIMIT 10
```

Примечание: Синтаксис зависит от конкретной СУБД (системы управления базами данных).

## 2 Фильтрация Данных (WHERE)

### Условия отбора строк

WHERE – твой основной инструмент, чтобы отсеять ненужные строки и оставить только то, что соответствует условиям.

### Операторы сравнения

```
1 SELECT product_name, price
2 FROM products
3 WHERE price > 100.0; -- Только товары дороже 100
4
5 SELECT order_id, status
6 FROM orders
7 WHERE status != 'cancelled'; -- Все заказы, кроме отмененных (<> тоже работает)
```

## Логические операторы

```
1 SELECT user_id, city, registration_date
2 FROM users
3 WHERE city = 'Moscow' AND registration_date >=
4   '2024-01-01'; -- Москвичи, зарег. в 2024+
5
6 SELECT product_id, category
7 FROM products
8 WHERE category = 'Electronics' OR category =
9   'Appliances'; -- Электроника ИЛИ Быт. техника
```

## Более сложные условия фильтрации

Часто нужны диапазоны, списки или поиск по шаблону.

### Диапазоны и Списки

```
1 -- BETWEEN: Включает границы диапазона
2 SELECT order_id, order_date
3 FROM orders
4 WHERE order_date BETWEEN '2024-01-01' AND
5   '2024-01-31';
6
7 -- IN: Проверка на вхождение в список
8 SELECT user_id, country
9 FROM users
10 WHERE country IN ('Russia', 'Belarus',
11   'Kazakhstan');
```

### Поиск по шаблону (LIKE)

```
1 -- '%' - любая последовательность символов (0 или больше)
2 -- '_' - ровно один любой символ
3 SELECT name
4 FROM users
5 WHERE name LIKE 'A%'; -- Имена, начинающиеся на 'А'
6
7 SELECT email
8 FROM users
9 WHERE email LIKE '%@example.com'; -- Email на домене example.com
10
11 SELECT product_code
12 FROM products
13 WHERE product_code LIKE 'PRD__X'; -- Коды вида PRD<2 символа>X
```

### Проверка на NULL

```
1 SELECT user_id, phone
2 FROM users
3 WHERE phone IS NULL; -- Пользователи без указанного телефона
4
5 SELECT order_id, delivery_date
6 FROM orders
7 WHERE delivery_date IS NOT NULL; -- Заказы, которые были доставлены (дата есть)
```

### 3 Сортировка Результатов (ORDER BY)

Упорядочивание вывода

Чтобы результаты были представлены в нужном порядке (например, от новых к старым, от дорогих к дешевым).

Примеры сортировки

```
1 -- Сортировка по одному столбцу (по умолчанию ASC
  - возрастание)
2 SELECT product_name, price
3 FROM products
4 ORDER BY price; -- От самых дешевых к дорогим (ASC)
5
6 -- Сортировка по убыванию (DESC)
7 SELECT user_id, registration_date
8 FROM users
9 ORDER BY registration_date DESC; -- От самых новых
  к старым
10
11 -- Сортировка по нескольким столбцам
12 SELECT city, name
13 FROM users
14 ORDER BY city ASC, name ASC; -- Сначала по городу
  (А-Я), потом по имени (А-Я) внутри города
```

Примечание: ORDER BY обычно идет в конце запроса (после WHERE, но до LIMIT).

### 4 Агрегатные Функции и Группировка (GROUP BY, HAVING)

Агрегация: Считаем итоги

Агрегатные функции выполняют вычисления над набором строк и возвращают одно значение.

- COUNT (\*): Общее количество строк.
- COUNT (column): Количество не-NULL значений в столбце.
- SUM (column): Сумма значений.
- AVG (column): Среднее значение.
- MIN (column): Минимальное значение.
- MAX (column): Максимальное значение.

Примеры агрегатных функций

```
1 -- Сколько всего пользователей?
2 SELECT COUNT(*) AS total_users FROM users;
3
4 -- Сколько пользователей указали город?
5 SELECT COUNT(city) AS users_with_city FROM users;
6
7 -- Общая сумма всех заказов
8 SELECT SUM(amount) AS total_revenue FROM orders;
9
10 -- Средняя цена товара
11 SELECT AVG(price) AS avg_product_price FROM
  products;
12
13 -- Самый ранний и самый поздний заказ
14 SELECT MIN(order_date) AS first_order,
  MAX(order_date) AS last_order FROM orders;
```

Группировка (GROUP BY)

Идея: Разделить строки на группы по значениям в одном или нескольких столбцах и применить агрегатные функции к каждой группе отдельно. Представь, что ты сортируешь чеки по магазинам (GROUP BY store), а потом считаешь сумму покупок для каждого магазина (SUM (amount)).

Пример GROUP BY

```
1 -- Количество пользователей в каждом городе
2 SELECT city, COUNT(*) AS user_count
3 FROM users
4 GROUP BY city
5 ORDER BY user_count DESC; -- Сортируем по
  количеству
6
7 -- Средняя сумма заказа для каждого пользователя
8 SELECT user_id, AVG(amount) AS avg_order_amount
9 FROM orders
10 GROUP BY user_id;
```

Важное правило: В части SELECT запроса с GROUP BY могут быть только:

- Столбцы, перечисленные в GROUP BY.
- Агрегатные функции.

Фильтрация После Группировки (HAVING)

Что если нужно отфильтровать результаты *после* агрегации? Например, показать только те города, где больше 100 пользователей? Для этого есть HAVING.

Пример HAVING

```
1 -- Города с более чем 100 пользователями
2 SELECT city, COUNT(*) AS user_count
3 FROM users
4 GROUP BY city
5 HAVING COUNT(*) > 100 -- Фильтруем по результату
  агрегации
6 ORDER BY user_count DESC;
7
8 -- Пользователи, чья средняя сумма заказа больше
  500
9 SELECT user_id, AVG(amount) AS avg_order_amount
10 FROM orders
11 GROUP BY user_id
12 HAVING AVG(amount) > 500;
```

Ключевое отличие (частый вопрос на собеседованиях):

- WHERE фильтрует строки до группировки и агрегации.
- HAVING фильтрует группы после группировки и агрегации (работает с результатами агрегатных функций).

Порядок выполнения (упрощенно): FROM -> WHERE -> GROUP BY -> HAVING -> SELECT -> ORDER BY -> LIMIT.

### 5 Объединение Таблиц (JOIN)

## Соединяем данные из разных таблиц

Данные часто хранятся в разных таблицах (например, пользователи в одной, их заказы в другой). JOIN позволяет их объединить по общему ключу. **Аналогия:** Представь два списка – список сотрудников с их ID и список отделов с ID сотрудников. JOIN позволяет "склеить" эти списки, чтобы увидеть, кто в каком отделе работает.

## Основные типы JOIN'ов

### INNER JOIN (Внутреннее соединение)

```
1 -- Возвращает только те строки, для которых есть
  совпадение ключа в ОБЕИХ таблицах.
2 -- Если у пользователя нет заказов, он НЕ попадет
  в результат.
3 SELECT
4   u.name, -- Имя пользователя из таблицы users
5   o.order_id, -- ID заказа из таблицы orders
6   o.amount -- Сумма заказа
7 FROM
8   users AS u
9 INNER JOIN
10  orders AS o ON u.user_id = o.user_id; --
  Условие соединения (ключи)
```

### LEFT JOIN (Левое внешнее соединение)

```
1 -- Возвращает ВСЕ строки из ЛЕВОЙ таблицы (users)
2 -- и совпадающие строки из ПРАВОЙ таблицы
  (orders).
3 -- Если у пользователя нет заказов, он ВСЕ РАВНО
  попадет в результат,
4 -- но столбцы из orders (order_id, amount) будут
  NULL.
5 -- КРАЙНЕ ВАЖЕН, чтобы не терять данные из
  основной (левой) таблицы!
6 SELECT
7   u.name,
8   o.order_id,
9   o.amount
10 FROM
11   users AS u
12 LEFT JOIN
13   orders AS o ON u.user_id = o.user_id;
```

Другие типы (реже используются):

- RIGHT JOIN: Аналогичен LEFT JOIN, но возвращает все строки из правой таблицы.
- FULL OUTER JOIN: Возвращает все строки из обеих таблиц, подставляя NULL там, где нет совпадений.
- CROSS JOIN: Декартово произведение (каждая строка одной таблицы с каждой строкой другой). Используй с большой осторожностью!

## 6 Подзапросы (Subqueries)

### Запрос внутри запроса

Подзапрос – это SELECT-запрос, вложенный внутрь другого SQL-запроса. Позволяет выполнять более сложные выборки.

#### Подзапрос в WHERE (самое частое)

```
1 -- Найти пользователей, сделавших заказы на сумму
  > 1000
2 SELECT name
3 FROM users
4 WHERE user_id IN ( -- Выбираем user_id из
  подзапроса
5   SELECT user_id
6   FROM orders
7   WHERE amount > 1000
8 );
9
10 -- Найти товары, дороже средней цены всех товаров
11 SELECT product_name, price
12 FROM products
13 WHERE price > ( -- Сравнение со скалярным
  подзапросом (возвращает 1 значение)
14   SELECT AVG(price) FROM products
15 );
```

#### Подзапрос в FROM (Производная таблица)

```
1 -- Сначала считаем среднее в подзапросе, потом
  выбираем из него
2 SELECT t.avg_val
3 FROM (
4   SELECT category, AVG(price) AS avg_val
5   FROM products
6   GROUP BY category
7 ) AS t -- Псевдоним для подзапроса обязателен!
8 WHERE t.avg_val > 50;
```

#### Подзапрос в SELECT (Скалярный подзапрос)

```
1 -- Добавляем столбец с максимальной ценой ко всем
  строкам
2 SELECT
3   product_name,
4   price,
5   (SELECT MAX(price) FROM products) AS
    max_overall_price
6 FROM products;
```

Операторы для подзапросов в WHERE: IN, NOT IN, EXISTS, NOT EXISTS, операторы сравнения (=, >, < и т.д., если подзапрос возвращает одно значение).

## 7 Общие Табличные Выражения (CTE - Common Table Expressions)

### WITH: Улучшаем читаемость сложных запросов

CTE – это именованный временный результирующий набор, на который можно ссылаться в последующем SELECT, INSERT, UPDATE или DELETE. **Преимущества:**

- Читаемость:** Разбивает сложный запрос на логические блоки.
- Структура:** Упрощает понимание логики запроса.
- Переиспользование:** На CTE можно ссылаться несколько раз внутри одного запроса (в большинстве СУБД).
- Рекурсия:** Позволяют писать рекурсивные запросы (редко нужно в DS).

Часто это ЛУЧШЕ, чем вложенные подзапросы!

#### Пример CTE

```
1 -- Найти пользователей, сделавших > 5 заказов на
  сумму > 100
2 WITH UserOrderStats AS ( -- Определяем CTE с
  именем UserOrderStats
3   SELECT
4     user_id,
5     COUNT(*) AS order_count,
6     SUM(amount) AS total_amount
7   FROM orders
8   WHERE amount > 100 -- Фильтр применяется здесь
9   GROUP BY user_id
10 )
11 -- Основной запрос, использующий CTE
12 SELECT
13   u.name,
14   uos.order_count,
15   uos.total_amount
16 FROM
17   users AS u
18 JOIN
19   UserOrderStats AS uos ON u.user_id =
    uos.user_id
20 WHERE
21   uos.order_count > 5 -- Фильтруем по
  результатам CTE
22 ORDER BY
23   uos.total_amount DESC;
```

## 8 Оконные Функции (Window Functions)

## Продвинутая аналитика без схлопывания строк

**Идея:** Оконные функции выполняют вычисления над набором строк ("окном"), которые как-то связаны с текущей строкой, но **не схлопывают строки**, как GROUP BY. Каждая строка сохраняется, и к ней добавляется результат вычисления оконной функции. **Аналогия:** Представь, что ты стоишь в очереди. GROUP BY может сказать тебе только, сколько всего людей в очереди. Оконная функция может сказать тебе твой номер в очереди (ROW\_NUMBER), или какой средний рост у людей вокруг тебя (AVG() OVER(...)), не убирая тебя из очереди.

**Общий синтаксис:** FUNCTION() OVER ( [PARTITION BY ...] [ORDER BY ...] [frame\_clause] )

- PARTITION BY column1, ...: Делит строки на независимые группы (партиции). Функция применяется к каждой партии отдельно. Похоже на GROUP BY, но строки НЕ схлопываются. Если опущено, всё окно - это все строки.
- ORDER BY column2, ...: Определяет порядок строк внутри партии. Важно для ранжирующих функций (RANK, ROW\_NUMBER) и функций смещения (LAG, LEAD).
- frame\_clause (ROWS/RANGE BETWEEN ...): Определяет точное подмножество строк внутри партии для вычисления (например, "текущая строка и 2 предыдущие"). Реже используется новичками.

## Часто используемые оконные функции

### Ранжирующие функции

```
1 -- Ранжирование продуктов по цене ВНУТРИ каждой категории
2 SELECT
3     product_name,
4     category,
5     price,
6     ROW_NUMBER() OVER(PARTITION BY category ORDER
7         BY price DESC) AS rn, -- Уникальный номер 1,
8         2, 3...
9     RANK() OVER(PARTITION BY category ORDER
10        BY price DESC) AS rk, -- Ранг с пропусками (1,
11        2, 2, 4...)
12     DENSE_RANK() OVER(PARTITION BY category ORDER
13        BY price DESC) AS drk -- Ранг без пропусков
14        (1, 2, 2, 3...)
15 FROM products;
```

### Агрегатные функции как оконные

```
1 -- Расчет доли цены продукта от средней цены В ЕГО
2   КАТЕГОРИИ
3 SELECT
4     product_name,
5     category,
6     price,
7     AVG(price) OVER(PARTITION BY category) AS
8     avg_price_in_category,
9     price / AVG(price) OVER(PARTITION BY category)
10    AS price_ratio
11 FROM products;
12 -- Общая сумма продаж нарастающим итогом по дате
13 SELECT
14     order_date,
15     amount,
16     SUM(amount) OVER(ORDER BY order_date ASC) as
17     cumulative_sales
18 FROM orders;
```

### Функции смещения (LAG/LEAD)

```
1 -- Поиск предыдущей даты заказа для каждого
2   пользователя
3 SELECT
4     user_id,
5     order_date,
6     LAG(order_date, 1) OVER(PARTITION BY user_id
7         ORDER BY order_date ASC) AS
8     previous_order_date
9 FROM orders;
10 -- LAG(столбец, смещение_назад,
11   значение_по_умолчанию)
12 -- LEAD(столбец, смещение_вперед,
13   значение_по_умолчанию)
```

**Применение в DS:** Ранжирование (топ N клиентов/товаров в группе), расчет долей (% от тOTALа по группе), поиск временных паттернов (время между событиями с LAG/LEAD), скользящие средние/суммы. **Очень популярная тема на собеседованиях!**

## 9 Работа с NULL

### Обработка отсутствующих значений

NULL – это специальное значение, означающее "отсутствие данных". С ним нужно работать аккуратно.

#### COALESCE: Замена NULL

```
1 -- Возвращает первое не-NULL значение из списка.
2 -- Очень полезно для подстановки значений по
3   умолчанию.
4 SELECT
5     name,
6     COALESCE(phone, 'Телефон не указан') AS
7     phone_display
8 FROM users;
9 -- Если first_name есть, берем его, иначе берем
10  username
11 SELECT COALESCE(first_name, username) AS
12 display_name FROM users;
```

#### NULLIF: NULL, если значения равны

```
1 -- Возвращает NULL, если два выражения равны,
2   иначе возвращает первое выражение.
3 -- Полезно, чтобы избежать деления на ноль или
4   обработать "пустые" строки.
5 SELECT
6     product_name,
7     total_sales,
8     total_quantity,
9     -- Избегаем деления на 0: если quantity=0,
10    NULLIF вернет NULL, и деление даст NULL
11     total_sales / NULLIF(total_quantity, 0) AS
12     average_price
13 FROM product_summary;
```

Помни: Сравнение с NULL через = или != почти всегда дает UNKNOWN (не TRUE). Используй IS NULL или IS NOT NULL.

## 10 Типы Данных и Преобразования

### Основные типы и конвертация

У каждого столбца есть тип данных. Иногда нужно их преобразовывать. **Частые типы данных:**

- Числовые: INTEGER (INT), FLOAT, REAL, NUMERIC, DECIMAL
- Строковые: VARCHAR(n), TEXT, CHAR(n)
- Даты/Время: DATE, TIME, TIMESTAMP, DATETIME
- Логический: BOOLEAN (BOOL)

(Точные имена и доступные типы зависят от СУБД)

## Явное преобразование типов (CAST/CONVERT)

```
1 -- Синтаксис может отличаться!
2
3 -- CAST (стандартный SQL)
4 SELECT CAST('2024-03-15' AS DATE);
5 SELECT CAST(price AS INTEGER) FROM products; --
   Отбросит дробную часть
6 SELECT CAST(user_id AS VARCHAR) FROM users; --
   Число в строку
7
8 -- CONVERT (SQL Server)
9 -- SELECT CONVERT(DATE, '2024-03-15');
10 -- SELECT CONVERT(VARCHAR, user_id);
11
12 -- неявное преобразование тоже бывает, но лучше
   делать явно.
```

Будь внимателен при преобразованиях: возможна потеря данных (например, FLOAT -> INT) или ошибки, если преобразование невозможно ('abc' -> INT).

### На что обратить особое внимание:

- **Практика, практика, практика!** Теория важна, но умение быстро писать рабочие запросы – ключ. Используй онлайн-тренажеры (LeetCode Database, HackerRank SQL, SQL Fiddle) или скачай SQLite базу с Kaggle и экспериментируй.
- **JOIN'ы – твой лучший друг (особенно LEFT):** Четко понимай разницу между INNER и LEFT JOIN. Подумай, когда ты можешь потерять данные с INNER JOIN и почему LEFT JOIN часто безопаснее.
- **Агрегация и Фильтрация Групп:** GROUP BY + Агрегатные функции + HAVING – это классический набор для задач на собеседованиях. Убедись, что ты понимаешь, как они работают вместе и чем WHERE отличается от HAVING.
- **Оконные Функции – покажи свой уровень:** Даже базовое понимание ROW\_NUMBER()/RANK() и агрегатных функций с OVER(PARTITION BY ... ) (например, для расчета доли от итога по группе) произведет хорошее впечатление. LAG/LEAD – бонусные очки.
- **Чистый и Читаемый Код:** Используй псевдонимы (AS), форматируй запросы (отступы), используй CTE (WITH) для сложных запросов вместо нагромождения подзапросов. Это показывает твою аккуратность и профессионализм.

*Удачи на собеседовании! SQL – это мощный инструмент, и уверенное владение им – большой плюс для любого Data Scientist'a.*