Шпаргалка по базам данных / SQL Cheatsheet (XeLaTeX)

Краткий справочник по основным операциям

Содержание

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

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

Выбор всех столбцов

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

Псевдонимы (AS) и Уникальность (DISTINCT)

Делаем запросы читаемыми и получаем только уникальные записи.

Псевдонимы (Aliases)

```sql
SELECT
    user_id AS id, -- Переименовываем user_id в id
    registration_date AS reg_date -- и registration_date в reg_date
FROM
    users AS u; -- Переименовываем таблицу users в u (удобно для JOIN'ов)
-- AS часто можно опустить: SELECT user_id id ...
```

Уникальные значения

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

Ограничение Вывода (LIMIT / TOP)

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

LIMIT (PostgreSQL, MySQL, SQLite)

```sql
SELECT user_id, name
FROM users
LIMIT 10; -- Показать только первые 10 строк
```

TOP (SQL Server, MS Access)

```sql
SELECT TOP 10 user_id, name
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

```sql
SELECT product_name, price
FROM products
WHERE price > 100.0; -- Только товары дороже 100

SELECT order_id, status
FROM orders
WHERE status != 'cancelled'; -- Все заказы, кроме отмененных (<> тоже работает)
```

Логические операторы

```sql
SELECT user_id, city, registration_date
FROM users
WHERE city = 'Moscow' AND registration_date >= '2024-01-01'; -- Москвичи, зарег. в 2024+

SELECT product_id, category
FROM products
WHERE category = 'Electronics' OR category = 'Appliances'; -- Электроника ИЛИ быт. техника
```

Более сложные условия фильтрации

Часто нужны диапазоны, списки или поиск по шаблону.

Диапазоны и Списки

```sql
-- BETWEEN: Включает границы диапазона
SELECT order_id, order_date
FROM orders
WHERE order_date BETWEEN '2024-01-01' AND
'2024-01-31';

-- IN: Проверка на вхождение в список
SELECT user_id, country
FROM users
WHERE country IN ('Russia', 'Belarus',
'Kazakhstan');
```

Поиск по шаблону (LIKE)

```sql
-- '%' - любая последовательность символов (0 или
больше)
-- '_' - ровно один любой символ
SELECT name
FROM users
WHERE name LIKE 'A%'; -- Имена, начинающиеся на
'A'

SELECT email
FROM users
WHERE email LIKE '%@example.com'; -- Email на
домене example.com

SELECT product_code
FROM products
WHERE product_code LIKE 'PRD__X'; -- Коды вида
PRD<2 символа>X
```

Проверка на NULL

```sql
SELECT user_id, phone
FROM users
WHERE phone IS NULL; -- Пользователи без
указанного телефона

SELECT order_id, delivery_date
FROM orders
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

```sql
-- Сортировка по одному столбцу (по умолчанию ASC
- возрастание)
SELECT product_name, price
FROM products
ORDER BY price; -- От самых дешевых к дорогим
(ASC)

-- Сортировка по убыванию (DESC)
SELECT user_id, registration_date
FROM users
ORDER BY registration_date DESC; -- От самых новых
к старым

-- Сортировка по нескольким столбцам
SELECT city, name
FROM users
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

```sql
-- Сколько всего пользователей?
SELECT COUNT(*) AS total_users FROM users;

-- Сколько пользователей указали город?
SELECT COUNT(city) AS users_with_city FROM users;

-- Общая сумма всех заказов
SELECT SUM(amount) AS total_revenue FROM orders;

-- Средняя цена товара
SELECT AVG(price) AS avg_product_price FROM
products;

-- Самый ранний и самый поздний заказ
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

```sql
-- Количество пользователей в каждом городе
SELECT city, COUNT(*) AS user_count
FROM users
GROUP BY city
ORDER BY user_count DESC; -- Сортируем по количеству

-- Средняя сумма заказа для каждого пользователя
SELECT user_id, AVG(amount) AS avg_order_amount
FROM orders
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

```sql
-- Города с более чем 100 пользователями
SELECT city, COUNT(*) AS user_count
FROM users
GROUP BY city
HAVING COUNT(*) > 100 -- Фильтруем по результату агрегации
ORDER BY user_count DESC;

-- Пользователи, чья средняя сумма заказа больше 500
SELECT user_id, AVG(amount) AS avg_order_amount
FROM orders
GROUP BY user_id
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)

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

LEFT JOIN (Левое внешнее соединение)

```sql
-- Возвращает ВСЕ строки из ЛЕВОЙ таблицы (users)
-- и совпадающие строки из ПРАВОЙ таблицы (orders).
-- Если у пользователя нет заказов, он ВСЕ РАВНО попадет в результат,
-- но столбцы из orders (order_id, amount) будут NULL.
-- КРАЙНЕ ВАЖЕН, чтобы не терять данные из основной (левой) таблицы!
SELECT
    u.name,
    o.order_id,
    o.amount
FROM
    users AS u
LEFT JOIN
    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)

```sql
-- Найти пользователей, сделавших заказы на сумму > 1000
SELECT name
FROM users
WHERE user_id IN ( -- Выбираем user_id из подзапроса
    SELECT user_id
    FROM orders
    WHERE amount > 1000
);

-- Найти товары, дороже средней цены всех товаров
SELECT product_name, price
FROM products
WHERE price > ( -- Сравнение со скалярным подзапросом (возвращает 1 значение)
    SELECT AVG(price) FROM products
);
```

Подзапрос в FROM (Производная таблица)

```sql
-- Сначала считаем среднее в подзапросе, потом выбираем из него
SELECT t.avg_val
FROM (
    SELECT category, AVG(price) AS avg_val
    FROM products
    GROUP BY category
) AS t -- Псевдоним для подзапроса обязателен!
WHERE t.avg_val > 50;
```

Подзапрос в SELECT (Скалярный подзапрос)

```sql
-- Добавляем столбец с максимальной ценой ко всем строкам
SELECT
    product_name,
    price,
    (SELECT MAX(price) FROM products) AS max_overall_price
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

```sql
-- Найти пользователей, сделавших > 5 заказов на сумму > 100
WITH UserOrderStats AS ( -- Определяем CTE с именем UserOrderStats
    SELECT
        user_id,
        COUNT(*) AS order_count,
        SUM(amount) AS total_amount
    FROM orders
    WHERE amount > 100 -- Фильтр применяется здесь
    GROUP BY user_id
)
-- Основной запрос, использующий CTE
SELECT
    u.name,
    uos.order_count,
    uos.total_amount
FROM
    users AS u
JOIN
    UserOrderStats AS uos ON u.user_id = uos.user_id
WHERE
    uos.order_count > 5 -- Фильтруем по результатам CTE
ORDER BY
    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 партиции отдельно. Похоже на GROUP BY, но строки НЕ схлопываются. Если опущено, всё окно - это все строки.
- ORDER BY column2, ...: Определяет порядок строк внутри партиции. Важно для ранжирующих функций (RANK, ROW_NUMBER) и функций смещения (LAG, LEAD).
- frame_clause (ROWS/RANGE BETWEEN ...): Определяет точное подмножество строк внутри партиции для вычисления (например, ”текущая строка и 2 предыдущие”). Реже используется новичками.

Часто используемые оконные функции

Ранжирующие функции

```sql
-- Ранжирование продуктов по цене ВНУТРИ каждой категории
SELECT
    product_name,
    category,
    price,
    ROW_NUMBER() OVER(PARTITION BY category ORDER BY price DESC) AS rn, -- Уникальный номер 1, 2, 3...
    RANK()       OVER(PARTITION BY category ORDER BY price DESC) AS rk, -- Ранг с пропусками (1, 2, 2, 4...)
    DENSE_RANK() OVER(PARTITION BY category ORDER BY price DESC) AS drk -- Ранг без пропусков (1, 2, 2, 3...)
FROM products;
```

Агрегатные функции как оконные

```sql
-- Расчет доли цены продукта от средней цены В ЕГО КАТЕГОРИИ
SELECT
    product_name,
    category,
    price,
    AVG(price) OVER(PARTITION BY category) AS avg_price_in_category,
    price / AVG(price) OVER(PARTITION BY category) AS price_ratio
FROM products;

-- Общая сумма продаж нарастающим итогом по дате
SELECT
  order_date,
  amount,
  SUM(amount) OVER(ORDER BY order_date ASC) as cumulative_sales
FROM orders;
```

Функции смещения (LAG/LEAD)

```sql
-- Поиск предыдущей даты заказа для каждого пользователя
SELECT
    user_id,
    order_date,
    LAG(order_date, 1) OVER(PARTITION BY user_id ORDER BY order_date ASC) AS previous_order_date
FROM orders;
-- LAG(столбец, смещение_назад, значение_по_умолчанию)
-- LEAD(столбец, смещение_вперед, значение_по_умолчанию)
```

Применение в DS: Ранжирование (топ N клиентов/товаров в группе), расчет долей (% от тотала 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

```sql
-- Возвращает первое не-NULL значение из списка.
-- Очень полезно для подстановки значений по умолчанию.
SELECT
    name,
    COALESCE(phone, 'Телефон не указан') AS phone_display
FROM users;

-- Если first_name есть, берем его, иначе берем username
SELECT COALESCE(first_name, username) AS display_name FROM users;
```

NULLIF: NULL, если значения равны

```sql
-- Возвращает NULL, если два выражения равны, иначе возвращает первое выражение.
-- Полезно, чтобы избежать деления на ноль или обработать "пустые" строки.
SELECT
    product_name,
    total_sales,
    total_quantity,
    -- Избегаем деления на 0: если quantity=0, NULLIF вернет NULL, и деление даст NULL
    total_sales / NULLIF(total_quantity, 0) AS average_price
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)

```sql
-- Синтаксис может отличаться!

-- CAST (стандартный SQL)
SELECT CAST('2024-03-15' AS DATE);
SELECT CAST(price AS INTEGER) FROM products; -- Отбросит дробную часть
SELECT CAST(user_id AS VARCHAR) FROM users; -- Число в строку

-- CONVERT (SQL Server)
-- SELECT CONVERT(DATE, '2024-03-15');
-- SELECT CONVERT(VARCHAR, user_id);

-- 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).

Фокус для Подготовки к Собеседованию

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'а.