SQL — функции и примеры (MySQL)

Содержание

1 Основы выборок и выражений

1.1 CREATE TABLE — базовый синтаксис

Классическое создание таблицы. Первичный ключ и автоинкремент определяются на целочисленном поле.

```
CREATE TABLE table_name (
    id INT PRIMARY KEY AUTO_INCREMENT,
    name VARCHAR(50)
    -- ... другие столбцы
);
```

1.2 INSERT INTO — добавление строк

Базовая вставка значений в указанные столбцы. Неуказанные столбцы получают `NULL` (если разрешено `NULL`).

```
INSERT INTO таблица (поле1, поле2)
VALUES (значение1, значение2);
```

Замечание: «У незаполненных строк выставляется `NULL`» — это верно для столбцов без `NOT NULL` и значений по умолчанию.

1.3 SELECT и псевдонимы столбцов

Проекция столбцов с переименованием через `AS`.

```
SELECT title AS Название, amount
FROM book;
```

1.4 IF — условное выражение (MySQL)

`IF`(*логическое_выражение*, *выражение_1*, *выражение_2*) вычисляет условие и возвращает одно из двух значений. Все три параметра обязательны.

```
SELECT
    title,
    amount,
    price,
    IF(amount < 4, price * 0.5, price * 0.7) AS sale
FROM book;
```

Примечание (MySQL): `IF` — функция MySQL; в стандартном SQL используется `CASE`.

1.5 WHERE — фильтрация строк

```
SELECT title, price
FROM book
WHERE price < 600;
```

1.6 Математические функции

Функция	Описание	Пример
`CEILING(x)`	возвращает наименьшее целое число, большее или равное x	`CEILING(4.2)=5` `CEILING(-5.8)=-5`
`ROUND(x, k)`	округляет x до k знаков; если k не указано — до целого	`ROUND(4.361)=4` `ROUND(5.86592,1)=5.9`
`FLOOR(x)`	возвращает наибольшее целое $\leq x$	`FLOOR(4.2)=4` `FLOOR(-5.8)=-6`
`POWER(x, y)`	x в степени y	`POWER(3,4)=81.0`
`SQRT(x)`	квадратный корень из x	`SQRT(4)=2.0` `SQRT(2)=1.41...`
`DEGREES(x)`	из радиан в градусы	`DEGREES(3)=171.8...`
`RADIANS(x)`	из градусов в радианы	`RADIANS(180)=3.14...`
`ABS(x)`	модуль x	`ABS(-1)=1` `ABS(1)=1`
`PI()`	число π	`PI()=3.14159...`

1.7 Логические операции AND, OR, NOT

Комбинируем условия через `AND`, `OR` и `NOT`.

```
SELECT title, author, price
FROM book
WHERE (author = 'Булгаков М.А.' OR author = 'Есенин С.А.')
  AND price > 600;
```

1.8 BETWEEN и IN — диапазоны и списки

```
-- BETWEEN включает границы
WHERE amount BETWEEN 5 AND 14; -- эквивалент WHERE amount >= 5 AND amount <= 14

-- IN сопоставляет со списком значений
WHERE author IN ('Булгаков М.А.', 'Достоевский Ф.М.');
```

1.9 LIKE: шаблоны % и _

Оператор LIKE сопоставляет строки с шаблоном. Обычные символы должны совпадать буквально; символы-шаблоны представляют группы символов.

Шаблон	Описание	Пример
%	Любая строка (0 и более символов)	author LIKE '%М.%'
_	Любой одиночный символ	title LIKE 'Поэм_'

1.10 ORDER BY: возрастание/убывание

Сортировка по одному или нескольким столбцам: ASC (по возрастанию) и DESC (по убыванию).

```
-- По убыванию количества внутри автора
SELECT author, Количество
FROM some_table
ORDER BY author, Количество DESC;

-- По возрастанию
SELECT author, Количество
FROM some_table
ORDER BY author, Количество ASC;
```

1.11 DISTINCT и GROUP BY — уникальные значения и группировка

```
-- Уникальные авторы
SELECT DISTINCT author
FROM book;

-- Тот же результат через GROUP BY
SELECT author
FROM book
GROUP BY author;
```

1.12 Агрегатные функции при GROUP BY

```
SELECT
    author,
    SUM(amount)  AS Сумма,
    COUNT(amount) AS Количество,
    MIN(price)   AS min_price
```

```
FROM book
GROUP BY author;
```

1.13 HAVING — условия на агрегаты

```
SELECT
    author,
    MIN(price) AS Минимальная_цена,
    MAX(price) AS Максимальная_цена
FROM book
GROUP BY author
HAVING SUM(price * amount) > 5000
ORDER BY Минимальная_цена DESC;
```

1.14 Порядок выполнения частей запроса

Рекомендуемый порядок обработки в SQL:

1. `FROM`
2. `WHERE`
3. `GROUP BY`
4. `HAVING`
5. `SELECT`
6. `ORDER BY`

Совет: при возможности используйте `WHERE` до `HAVING`; так движок обрабатывает меньше строк.

1.15 Подзапросы (вложенные запросы)

Подзапросы возвращают значения/наборы для внешнего запроса. Удобны с `IN` и сравнительными операторами. *Проверяйте итог подзапроса перед использованием.*

1.16 ANY и ALL — сравнение со множеством

Операторы `ANY` и `ALL` применяются с подзапросами (или списками для иллюстрации).

- `amount > ANY (10, 12)` ⇒ `amount > 10`
- `amount < ANY (10, 12)` ⇒ `amount < 12`
- `amount = ANY (10, 12)` ⇒ `(amount = 10) OR (amount = 12)` ≡ `amount IN (10, 12)`
- `amount <> ANY (10, 12)` — вернёт все записи (в т.ч. 10 и 12)
- `amount > ALL (10, 12)` ⇒ `amount > 12`
- `amount < ALL (10, 12)` ⇒ `amount < 10`
- `amount = ALL (10, 12)` — пусто (эквивалентно `amount = 10 AND amount = 12`)
- `amount <> ALL (10, 12)` — все, кроме 10 и 12

```
-- Пример с подзапросом (структура)
SELECT *
FROM book
WHERE amount > ALL (
  SELECT amount
  FROM thresholds
  WHERE kind = 'limit'
);
```

1.17 Агрегаты и обычные столбцы: GROUP BY или OVER()

Если в запросе есть агрегаты и «обычные» столбцы, все неагрегированные поля должны быть перечислены в `GROUP BY`. Альтернативы: подзапрос или оконная функция `OVER()`.

```
-- Подзапрос
SELECT
    title,
    author,
    amount,
    (SELECT AVG(amount) FROM book) AS Среднее_количество
FROM book
WHERE ABS(amount - (SELECT AVG(amount) FROM book)) > 3;

-- Окно OVER () (MySQL 8+)
SELECT
    title,
    author,
    amount,
    AVG(amount) OVER () AS Среднее_количество
FROM book
WHERE ABS(amount - AVG(amount) OVER ()) > 3;
```

1.18 LIMIT — ограничение числа строк

```
SELECT *
FROM book
ORDER BY price DESC
LIMIT 2; -- не более двух строк в результате
```

1.19 Функции дат и времени: DATEDIFF, DATE_ADD, MONTHNAME

```
-- Кол-во дней между датами
SELECT DATEDIFF('2020-04-01', '2020-03-28'); -- 4
SELECT DATEDIFF('2020-05-09', '2020-05-01'); -- 8

-- Сложение интервала
SELECT DATE_ADD('2020-02-02', INTERVAL 45 DAY);   -- 2020-03-18
SELECT DATE_ADD('2020-02-02', INTERVAL 6 MONTH);  -- 2020-08-02

-- Название месяца (англ.)
SELECT MONTHNAME('2020-04-12'); -- 'April'
```

Совет: группируя по вычисляемому столбцу (например, имени месяца), после `GROUP BY` можно указать как выражение, так и его псевдоним `AS`.

1.20 Компоненты даты: DAY, MONTH, YEAR

```
SELECT DAY('2020-02-01');   -- 1
SELECT MONTH('2020-02-01'); -- 2
SELECT YEAR('2020-02-01');  -- 2020
```

1.21 Сравнение с пустым значением: IS NULL

```
SELECT *
FROM book
WHERE date_payment IS NULL;
```

1.22 RAND и целые диапазоны

```
-- Случайное число [0, 1)
SELECT RAND();

-- Случайное целое 0..365
SELECT FLOOR(RAND() * 365);

-- Добавление случайного числа дней (0..364) к дате
SELECT DATE_ADD('2020-01-01', INTERVAL FLOOR(RAND() * 365) DAY);
```

1.23 UNION и UNION ALL — объединение результатов

```
-- Общая форма
SELECT столбец_1_1, столбец_1_2, ...
```

```
FROM ...
UNION            -- или UNION ALL
SELECT столбец_2_1, столбец_2_2, ...
FROM ...;
```

```
-- Пример
SELECT buy_id, client_id, book_id, date_payment, amount, price
FROM buy_archive
UNION ALL
SELECT b.buy_id, client_id, book_id, bs.date_step_end, bb.amount, price
FROM book AS bk
INNER JOIN buy_book AS bb USING (book_id)
INNER JOIN buy AS b USING (buy_id)
INNER JOIN buy_step AS bs USING (buy_id)
INNER JOIN step AS s USING (step_id)
WHERE bs.date_step_end IS NOT NULL AND s.name_step = 'Оплата'
ORDER BY client_id;
```

Замечание: Оператор WHERE применяется к отдельному подзапросу, а не ко всему результату UNION. Если написать ... UNION ... WHERE ..., то WHERE будет относиться только к последнему SELECT.

```
-- Пример использования WHERE ко всему запросу
SELECT *
FROM (
    SELECT student_id, score FROM exam1
    UNION
    SELECT student_id, score FROM exam2
) AS all_results
WHERE score > 60;
```

1.24 ORDER BY RAND() — случайная сортировка

См. также раздел про RAND выше (повтор).

```
SELECT *
FROM book
ORDER BY RAND();
```

1.25 LEFT, CONCAT, NOW — строковые и дата-функции

```
SELECT LEFT('abcde', 3);         -- 'abc'
SELECT CONCAT('ab', 'cd');       -- 'abcd'
SELECT NOW();                    -- текущая дата-время
```

1.26 Функция SUBSTR (или SUBSTRING)

```
-- Общая форма:
SUBSTR(str, pos, len)
```

Описание:

- Возвращает подстроку из строки `str`, начиная с позиции `pos`.
- Параметр `len` задаёт количество символов для выборки.
- Если `pos > 0` — отсчёт идёт с начала строки (первая буква = 1).
- Если `pos < 0` — отсчёт идёт с конца строки (например, `-1` = последний символ).
- Если `len` опущен, берётся всё до конца строки.

Примеры:

```
SELECT SUBSTR('Привет, мир!', 1, 6);   -- 'Привет'
SELECT SUBSTR('Привет, мир!', 9, 3);   -- 'мир'
SELECT SUBSTR('Привет, мир!', -4, 3);  -- 'мир'
SELECT SUBSTR('Привет, мир!', 8);      -- ' мир!'
```

1.27 FROM_UNIXTIME — из UNIX time в DATETIME

```
SELECT FROM_UNIXTIME(time_unix);
-- Формула: 1970-01-01 + time_unix / 86400
```

1.28 INSTR — позиция подстроки

```
SELECT INSTR('abcdef', 'cd'); -- 3 (позиция первого вхождения), 0 если нет
```

1.29 Функции RIGHT и SUBSTRING_INDEX

```
-- Общая форма:
RIGHT(str, n)
```

Функция RIGHT возвращает последние `n` символов строки `str`. Например: `RIGHT('Привет', 2)` → `'ет'`.

```
-- Общая форма:
SUBSTRING_INDEX(str, delim, count)
```

Функция SUBSTRING_INDEX делит строку `str` по разделителю `delim`.

- Если `count > 0` → возвращаются части слева (до разделителя).
- Если `count < 0` → возвращаются части справа (от конца строки).

Например: `SUBSTRING_INDEX('Иван Петров', ' ', 1)` → `'Иван'`.

```
-- Пример работы вместе:
SELECT name, city, per_diem, date_first, date_last
FROM trip
WHERE RIGHT(SUBSTRING_INDEX(name, " ", 1), 1) = "a"
ORDER BY date_last DESC;
```

Здесь:

- `SUBSTRING_INDEX(name, , 1)` берёт первое слово из поля `name`,
- `RIGHT(..., 1)` возвращает его последнюю букву,
- условие `= "a"` отбирает строки, где первое слово оканчивается на «а».

1.30 Символ \\b обозначает границу слова

Этот символ совпадает с позицией в строке, где один символ является «словесным» (буква, цифра, подчёркивание), а соседний символ — «не-словесный» (пробел, пунктуация, начало/конец строки).

```
-- Пример (использующиеся функции будут разобраны ниже):
INSERT INTO step_keyword
SELECT step_id, keyword_id
FROM step
JOIN keyword
  ON step_name REGEXP CONCAT('\\b', keyword_name, '\\b')
ORDER BY keyword_id;
```

2 Изменение данных и DDL

2.1 INSERT ... SELECT — перенос данных

```
INSERT INTO book (title, author, price, amount)
SELECT title, author, price, amount
FROM supply;

INSERT INTO buy_step (buy_id, step_id, date_step_beg, date_step_end)
SELECT 5, step_id, NULL, NULL
FROM step
```

2.2 UPDATE — обновление значений

```
UPDATE таблица
SET поле = выражение;
```

```
UPDATE book
SET price = 0.7 * price
WHERE amount < 5;
```

Обновление нескольких столбцов:

```
UPDATE таблица
SET поле1 = выражение1,
    поле2 = выражение2;
```

```
UPDATE book
SET amount = amount - buy,
    buy    = 0;
```

2.3 UPDATE с несколькими таблицами (совпадающие столбцы)

```
UPDATE book, supply
SET book.amount = book.amount + supply.amount
WHERE book.title = supply.title
  AND book.author = supply.author;
```

2.4 UPDATE с использованием IF

```
UPDATE book
SET genre_id = IF(
    title = "Стихотворения и поэмы" AND author_id = 5,
    2,
    IF(
        title = "Остров сокровищ" AND author_id = 6,
        3,
        genre_id
    )
);
```

2.5 DELETE — удаление по условию/подзапросу

```
DELETE FROM supply
WHERE title IN (SELECT title FROM book);
```

2.6 CREATE TABLE AS — создание таблицы из запроса

```
CREATE TABLE ordering AS
SELECT author, title, 5 AS amount
FROM book
WHERE amount < 4;
```

```
CREATE TABLE ordering AS
SELECT author,
       title,
       (SELECT ROUND(AVG(amount)) FROM book) AS amount
FROM book
WHERE amount < 4;
```

2.7 DROP TABLE — удаление таблицы

```
DROP TABLE таблица;
```

2.8 Тип DATE — формат и литералы

Тип `DATE` хранит дату в формате `YYYY-MM-DD`. В `INSERT` дата берётся в одинарные кавычки.

2.9 ALTER TABLE — добавление столбцов

```
ALTER TABLE таблица ADD имя_столбца тип;              -- в конец
ALTER TABLE таблица ADD имя_столбца тип FIRST;         -- перед первым
ALTER TABLE таблица ADD имя_столбца тип AFTER имя_столбца_1; -- после указанного
```

2.10 ALTER TABLE — удаление столбцов

```
ALTER TABLE таблица DROP COLUMN имя_столбца;     -- с ключевым словом COLUMN
ALTER TABLE таблица DROP имя_столбца;            -- COLUMN необязателен
ALTER TABLE таблица
  DROP имя_столбца,
  DROP имя_столбца_1;                            -- несколько столбцов
```

2.11 ALTER TABLE — переименование столбца

```
ALTER TABLE таблица CHANGE имя_столбца новое_имя_столбца;
```

Примечание: синтаксис MySQL для `CHANGE` обычно требует и тип столбца; проверить в контексте проекта.

3 Связи и внешние ключи

3.1 Типы связей: 1:M и M:M

Один-ко-многим (книга — автор) и многие-ко-многим (покупатели — книги).

3.2 Многие-ко-многим: связующая таблица

Создаём отдельную таблицу со внешними ключами.

```
CREATE TABLE book (
    book_id   INT PRIMARY KEY AUTO_INCREMENT,
    title     VARCHAR(50),
    author_id INT NOT NULL,
    price     DECIMAL(8,2),
    amount    INT,
    FOREIGN KEY (author_id) REFERENCES author (author_id)
);
```

Отдельно задаём `author_id` (`NOT NULL`), объявляем внешний ключ и указываем `REFERENCES author(author_id)`.

3.3 ON DELETE — варианты поведения

Поведение при удалении родительской строки:

- **CASCADE** — каскадное удаление зависимых строк.
- **SET NULL** — установка `NULL` во внешнем ключе (`NULL` должен быть разрешён).
- **SET DEFAULT** — установка значения по умолчанию (если задано).
- **RESTRICT** — запрет удаления при наличии зависимых строк.

```
CREATE TABLE book (
    book_id   INT PRIMARY KEY AUTO_INCREMENT,
    title     VARCHAR(50),
    author_id INT,
    price     DECIMAL(8,2),
    amount    INT,
    FOREIGN KEY (author_id)
        REFERENCES author (author_id)
        ON DELETE CASCADE
);
```

4 Соединения и операции с JOIN

4.1 USING и ON — способы задания условия

Если имена столбцов совпадают, можно использовать `USING`; в остальных случаях — `ON`.

Также в конструкции `USING` можно указать несколько столбцов одновременно — тогда соединение будет выполняться по совпадению значений во всех этих столбцах.

Важно понимать: соединение с `ON` работает *не только по столбцам*, а вообще по **любому булевому условию**, указанному после `ON`.

```
-- Общая форма соединений USING:
SELECT ...
FROM table1
INNER JOIN table2
USING (col1, col2, col3);

-- Пример использования ON с булевым условием:
-- если название ключевого слова встречается в тексте шага
SELECT s.step_id, k.keyword_id
FROM step s
JOIN keyword k
  ON INSTR(CONCAT(' ', s.step_name, ' '),
           CONCAT(' ', k.keyword_name, ' ')) > 0;
```

4.2 Соединение таблиц по совпадающим столбцам (даже без явных связей)

В SQL таблицы можно объединять (`JOIN`) не только по внешним ключам, но и по *любым* столбцам (или их комбинации), значения которых логически совпадают. Это полезно, когда в схеме БД нет объявленной связи, но по данным связь очевидна (например, по названию книги, автору и цене). Однако такой подход требует аккуратности: возможны дубликаты, ошибки сопоставления из-за разных написаний, колляций и т. п.; по возможности лучше фиксировать связи внешними ключами.

Ниже показан пример:

- `book` ↔ `author` — связь *есть и прописана* по `author_id` (используем `USING(author_id)`).
- `book` ↔ `supply` — физическойл связи *не обвявлено*, но мы можем *логически* соединить по совпадению (`title`, `author`, `price`) с помощью *составного* условия (row value constructor).

```
SELECT
  book.title AS Название,
  name_author AS Автор,
  book.amount + supply.amount AS Количество
```

```
FROM book
  INNER JOIN author USING (author_id)
  INNER JOIN supply
    ON (book.title, author.name_author, book.price)
     = (supply.title, supply.author, supply.price);
```

Замечание:

- Составные сравнения `(a,b,c) = (x,y,z)` поддерживаются в MySQL и удобны для «естественных» ключей.

4.3 INNER JOIN — пересечение

```
SELECT title, name_author
FROM author
INNER JOIN book
    ON author.author_id = book.author_id; -- или USING (author_id)
```

4.4 LEFT/RIGHT JOIN — внешние соединения

LEFT JOIN берёт все записи из левой таблицы; отсутствующие справа заполняются `NULL`. **RIGHT JOIN** — симметрично.

4.5 FULL OUTER JOIN — замечание

Полное соединение концептуально объединяет `LEFT` и `RIGHT`; см. приём через `UNION` в разделе про CTE (ниже). В MySQL самого `FULL OUTER JOIN` нет (см. реализацию через `UNION`, *повтор*).

4.6 CROSS JOIN — декартово произведение (— соединяет две таблицы и создает всевозможные сочетания значений строк в них (условно авторы и жанры):

name_author	name_genre
Булгаков М.А.	Роман
Булгаков М.А.	Поэзия
Булгаков М.А.	Приключения
Достоевский Ф.М.	Роман
Достоевский Ф.М.	Поэзия
Достоевский Ф.М.	Приключения

4.7 Несколько JOIN подряд

```
SELECT title, name_author, name_genre, price, amount
FROM author
INNER JOIN book  ON author.author_id = book.author_id
INNER JOIN genre ON genre.genre_id   = book.genre_id
WHERE price BETWEEN 500 AND 700;
```

4.8 UPDATE с JOIN

```
UPDATE book
INNER JOIN author ON author.author_id = book.author_id
INNER JOIN supply ON book.title = supply.title
                  AND supply.author = author.name_author
SET book.amount = book.amount + supply.amount,
    supply.amount = 0
WHERE book.price = supply.price;
```

4.9 INSERT ... SELECT с JOIN

```
INSERT INTO таблица (список_полей)
SELECT список_полей_из_других_таблиц
FROM таблица_1
JOIN таблица_2 ON ...;
```

4.10 DELETE с USING и JOIN

```
DELETE FROM таблица_1
USING таблица_1
INNER JOIN таблица_2 ON ...
WHERE ...;
```

4.10.1 Пример: удалить авторов с книгами, где amount < 3

```
DELETE FROM author
USING author
INNER JOIN book ON author.author_id = book.author_id
WHERE book.amount < 3;

-- И удалить книги этих авторов (отдельным шагом, при необходимости)
DELETE b
```

```
FROM book AS b
INNER JOIN author AS a ON a.author_id = b.author_id
WHERE b.amount < 3;
```

5 Переменные, REGEXP, CASE, CTE (WITH) и FULL JOIN через UNION

5.1 Переменные пользователя (@var)

Переменные задаются через SET; их можно увеличивать/использовать в выражениях запроса.

```
SET @row_num := 0;
SELECT *,
       (@row_num := @row_num + 1) AS str_num
FROM applicant_order;
```

Нумерация строк с начала выборки.

5.1.1 Нумерация по группе (пример)

```
SET @num_pr := 0;
SET @row_num := 1;

SELECT *,
       IF(program_id = @num_pr,
          @row_num := @row_num + 1,
          @row_num := 1) AS str_num,
       @num_pr := program_id AS add_var
FROM applicant_order;
```

5.2 REGEXP — регулярные выражения

REGEXP фильтрует строки по шаблону; богаче, чем LIKE.

- ^ — начало строки; $ — конец строки
- . — любой одиночный символ
- [...] — любой символ из набора; [a-z] — диапазон
- | — «или» между шаблонами

Примеры:

```
WHERE ProductName REGEXP 'Phone';       -- содержит 'Phone'
WHERE ProductName REGEXP '^Phone';      -- начинается с 'Phone'
WHERE ProductName REGEXP 'Phone$';      -- заканчивается на 'Phone'
```

```
WHERE ProductName REGEXP 'iPhone [78]'; -- 'iPhone 7' или 'iPhone 8'
WHERE ProductName REGEXP 'iPhone [6-8]';-- 'iPhone 6', '7' или '8'
```

```
-- Найти товары, где название содержит 'Phone' или 'Galaxy'
SELECT *
FROM Products
WHERE ProductName REGEXP 'Phone|Galaxy';
```

5.3 CASE — выбор из нескольких условий

```
CASE
  WHEN логическое_выражение_1 THEN выражение_1
  WHEN логическое_выражение_2 THEN выражение_2
  ...
  ELSE выражение_else
END
```

Где можно использовать: SELECT, UPDATE, DELETE, SET, WHERE, ORDER BY, HAVING.

```
SELECT student_name, rate,
       CASE
         WHEN rate <= 10 THEN 'I'
         WHEN rate <= 15 THEN 'II'
         WHEN rate <= 27 THEN 'III'
         ELSE 'IV'
       END AS Группа
FROM (
  SELECT student_name, COUNT(*) AS rate
  FROM (
    SELECT student_name, step_id
    FROM student
    INNER JOIN step_student USING (student_id)
    WHERE result = 'correct'
    GROUP BY student_name, step_id
  ) AS query_in
  GROUP BY student_name
  ORDER BY 2
) AS query_in_1;
```

5.4 CTE (WITH) — табличные выражения

Определяем промежуточные результаты, к которым обращается основной запрос.

```
WITH имя_выражения (имя_1, имя_2, ...)
AS (
  SELECT столбец_1, столбец_2
  FROM ...
)
SELECT ...
FROM имя_выражения;
```

```
WITH get_count_correct (st_n_c, count_correct) AS (
       SELECT step_name, COUNT(*)
       FROM step
       INNER JOIN step_student USING (step_id)
       WHERE result = 'correct'
       GROUP BY step_name
     ),
     get_count_wrong (st_n_w, count_wrong) AS (
       SELECT step_name, COUNT(*)
       FROM step
       INNER JOIN step_student USING (step_id)
       WHERE result = 'wrong'
       GROUP BY step_name
     )
SELECT st_n_c AS Шаг,
       ROUND(count_correct / (count_correct + count_wrong) * 100) AS Успешность
FROM get_count_correct
INNER JOIN get_count_wrong ON st_n_c = st_n_w;
```

5.5 FULL OUTER JOIN в MySQL через UNION

Полное внешнее соединение в MySQL можно реализовать объединением LEFT и RIGHT через UNION.

```
-- Структура
SELECT ...
FROM таблица_1 LEFT JOIN таблица_2 ON ...
UNION
SELECT ...
FROM таблица_1 RIGHT JOIN таблица_2 ON ...;
```

```
WITH get_count_correct (st_n_c, count_correct) AS (
       SELECT step_name, COUNT(*)
       FROM step
       INNER JOIN step_student USING (step_id)
```

```
      WHERE result = 'correct'
      GROUP BY step_name
    ),
    get_count_wrong (st_n_w, count_wrong) AS (
      SELECT step_name, COUNT(*)
      FROM step
      INNER JOIN step_student USING (step_id)
      WHERE result = 'wrong'
      GROUP BY step_name
    )
SELECT st_n_c AS Шаг,
       ROUND(count_correct / (count_correct + count_wrong) * 100) AS Успешность
FROM get_count_correct
LEFT JOIN get_count_wrong ON st_n_c = st_n_w
UNION
SELECT st_n_c AS Шаг,
       ROUND(count_correct / (count_correct + count_wrong) * 100) AS Успешность
FROM get_count_correct
RIGHT JOIN get_count_wrong ON st_n_c = st_n_w
ORDER BY 2;
```

6 Оконные функции

6.1 Общее: синтаксис окна

Оконные функции работают поверх результата запроса (кроме `ORDER BY`/`LIMIT`) и вычисляются в окне:

```
функция(выражение) OVER (
  PARTITION BY столбец_1, столбец_2 -- окно (не обязательно)
  ORDER BY ...                      -- сортировка (не обязательно)
  ROWS BETWEEN ...                  -- рамки (не обязательно)
);
```

Достаточно указать либо окно, либо сортировку.

6.2 ORDER BY в окне: нумерация и ранжирование

```
SELECT student_name,
       COUNT(*) AS Количество,
       ROW_NUMBER() OVER (ORDER BY COUNT(*) DESC) AS Номер
FROM student
INNER JOIN step_student USING (student_id)
```

```
WHERE result = 'correct'
GROUP BY student_name;
```

```
SELECT student_name,
       COUNT(*) AS Количество,
       ROW_NUMBER()  OVER (ORDER BY COUNT(*) DESC) AS Номер,
       RANK()        OVER (ORDER BY COUNT(*) DESC) AS Ранк,
       DENSE_RANK()  OVER (ORDER BY COUNT(*) DESC) AS Рейтинг
FROM student
INNER JOIN step_student USING (student_id)
WHERE result = 'correct'
GROUP BY student_name;
```

6.3 LAG/LEAD и разницы между строками

Функции LAG и LEAD позволяют получить значение из другой строки текущего окна — соответственно, предыдущей или следующей. Это удобно для сравнения соседних записей.

```
-- Общая форма
LAG(expr, offset, default) OVER (PARTITION BY ... ORDER BY ...)
LEAD(expr, offset, default) OVER (PARTITION BY ... ORDER BY ...)
```

Описание:

- `expr` — выражение или столбец, значение которого нужно взять.
- `offset` — сдвиг (по умолчанию 1). Например, `LAG(x,2)` = значение за 2 строки до текущей.
- `default` — значение, возвращаемое, если нужной строки нет (например, у первой строки для `LAG`).
- `PARTITION BY` — делит набор строк на группы (например, по студенту).
- `ORDER BY` — задаёт порядок строк внутри группы.

Примеры:

```
-- Предыдущая попытка студента
SELECT student_id,
       attempt_date,
       result,
       LAG(result) OVER (PARTITION BY student_id ORDER BY attempt_date) AS
   prev_result
FROM attempts;

-- Следующая попытка студента
SELECT student_id,
       attempt_date,
```

```
       result,
       LEAD(result) OVER (PARTITION BY student_id ORDER BY attempt_date) AS
    next_result
FROM attempts;
```

В первом запросе столбец `prev_result` покажет результат предыдущей попытки каждого студента, во втором — `next_result` отразит результат следующей попытки.

```
SELECT student_name,
       COUNT(*) AS Количество,
       IFNULL(LAG(COUNT(*)) OVER (ORDER BY COUNT(*) DESC) - COUNT(*), 0) AS Разница
FROM student
INNER JOIN step_student USING (student_id)
WHERE result = 'correct'
GROUP BY student_name;
```

IFNULL нужна, так как у первой строки нет предшествующей (NULL).

6.4 PARTITION BY — окна по группам

```
WITH get_rate_lesson (mod_id, stud, rate) AS (
  SELECT module_id,
         student_name,
         COUNT(DISTINCT step_id)
  FROM student
  INNER JOIN step_student USING (student_id)
  INNER JOIN step USING (step_id)
  INNER JOIN lesson USING (lesson_id)
  WHERE result = 'correct'
  GROUP BY module_id, student_name
)
SELECT mod_id AS Модуль,
       stud   AS Студент,
       rate   AS Рейтинг,
       ROW_NUMBER() OVER (PARTITION BY mod_id ORDER BY rate DESC)   AS Номер,
       RANK()       OVER (PARTITION BY mod_id ORDER BY rate DESC)   AS Ранк,
       DENSE_RANK() OVER (PARTITION BY mod_id ORDER BY rate DESC)   AS Рейтинг
FROM get_rate_lesson;
```

Если модулей два, нумерация и ранги начинаются заново внутри каждого модуля.

6.5 ROWS BETWEEN — рамки окна

- **UNBOUNDED PRECEDING** — от начала партиции.
- ***n* PRECEDING** — n строк до текущей (включая текущую при агрегациях).

- **CURRENT ROW** — только текущая строка.
- ***n* FOLLOWING** — n строк после текущей.
- **UNBOUNDED FOLLOWING** — до конца партиции.

6.6 Агрегаты как оконные функции

```
WITH get_rate_lesson (mod_id, les, rate) AS (
  SELECT module_id,
         CONCAT(module_id, '.', lesson_position),
         COUNT(DISTINCT step_id)
  FROM step_student
  INNER JOIN step   USING (step_id)
  INNER JOIN lesson USING (lesson_id)
  WHERE result = 'correct'
  GROUP BY module_id, 2
)
SELECT mod_id AS Модуль,
       les    AS Урок,
       rate   AS Пройдено_шагов,
       MAX(rate) OVER (PARTITION BY mod_id) AS Максимум_по_модулю,
       MIN(rate) OVER (PARTITION BY mod_id) AS Минимум_по_модулю
FROM get_rate_lesson;
```

Выводит минимум/максимум по модулю; два модуля — две строки с максимумом/минимумом для каждого.