Оглавление

1. Работа с базами данных

Создание и удаление БД

Чтобы создать новую базу данных, используйте команду `CREATE DATABASE`:

```
CREATE DATABASE shop;  -- Создаёт БД с именем "shop"
```

Для удаления базы данных используйте `DROP DATABASE`:

```
DROP DATABASE shop;  -- Удаляет БД "shop" (внимание: данные исчезнут безвозвратно!)
```

Выбор БД

Перед работой с таблицами нужно выбрать базу данных. В MySQL это делается командой `USE`:

```
USE shop;  -- Выбирает БД "shop" для дальнейших операций
```

2. Таблицы и типы данных

Создание таблиц

Таблица создаётся командой `CREATE TABLE`. Указывается имя таблицы, столбцы и их типы:

```
CREATE TABLE users (
  id INT,
  name VARCHAR(50),
  email VARCHAR(100)
);
```

Изменение таблиц

Добавить или удалить столбец можно через `ALTER TABLE`:

```
ALTER TABLE users ADD age INT;      -- Добавить столбец
ALTER TABLE users DROP COLUMN age; -- Удалить столбец
```

Удаление таблиц

Полное удаление таблицы выполняется командой:

```
DROP TABLE users;
```

Типы данных

- `INT` — целые числа (например, `id`, возраст).
- `VARCHAR(N)` — строка переменной длины до `N` символов (имя, email).
- `TEXT` — длинный текст (статьи, описания), в отличие от `VARCHAR` не требует указания длины.
- `BOOLEAN` — логический тип (`true` / `false`). В MySQL часто используется как `TINYINT(1)`.
- `DATE` — дата в формате `YYYY-MM-DD` (день рождения, дата регистрации).

Выбор типа зависит от хранимых данных и экономии памяти. Например, для коротких строк лучше `VARCHAR`, а для чисел — `INT`.

3. Вставка, обновление и удаление данных

`INSERT` – добавление записей

Чтобы добавить данные в таблицу, используйте `INSERT`. Укажите таблицу, столбцы и значения:

```
INSERT INTO users (name, email) VALUES ('Иван', 'ivan@example.com');
```

Если заполняете все столбцы, перечислять их необязательно:

```
INSERT INTO users VALUES (1, 'Иван', 'ivan@example.com');
```

`UPDATE` – изменение данных

Для обновления существующих записей применяйте `UPDATE`. Укажите таблицу, новые значения и условие для выбора строк:

```
UPDATE users SET email = 'new@example.com' WHERE id = 1;
```

Без условия `WHERE` обновятся **все** записи в таблице!

`DELETE` – удаление записей

Удалить данные можно командой `DELETE`. Как и с `UPDATE`, важно указать условие:

```
DELETE FROM users WHERE id = 1;
```

Если пропустить `WHERE`, удалятся **все** данные из таблицы.

4. Выборка данных (SELECT)

Базовые запросы

```
SELECT * FROM users;
```

Этот запрос выведет **все столбцы и все строки** из таблицы `users`. Звездочка (`*`) означает "все поля".

Выбор конкретных столбцов

```
SELECT name, email FROM users;
```

Запрос вернет только два столбца — `name` и `email` — для **каждой записи** в таблице `users`, остальные поля будут исключены из результата.

Фильтрация (WHERE)

```
SELECT * FROM products WHERE price > 1000;
```

Здесь выбираются **только те товары**, у которых цена больше 1000. Все остальные записи отфильтровываются.

Операторы сравнения

```
SELECT * FROM orders WHERE status != 'completed';
```

Запрос найдет **все заказы**, статус которых НЕ равен `'completed'`. Оператор `!=` означает "не равно".

Сортировка (ORDER BY)

```
SELECT * FROM products ORDER BY price DESC;
```

Результат будет отсортирован **по убыванию цены** (от дорогих к дешевым). `DESC` — это сортировка по убыванию, `ASC` (по умолчанию) — по возрастанию.

Сортировка по нескольким полям

```
SELECT * FROM users ORDER BY last_name ASC, first_name ASC;
```

Сначала записи сортируются **по фамилии** (от А до Я), а если фамилии одинаковые — **по имени** (также от А до Я).

Ограничение выборки (LIMIT)

```
SELECT * FROM news LIMIT 5;
```

Запрос вернет **только первые 5 записей** из таблицы `news`. Это полезно для вывода последних новостей.

LIMIT с OFFSET

```
SELECT * FROM articles LIMIT 10, 5;
```

Пропускает **первые 10 записей** и возвращает **следующие 5**. Используется для постраничного вывода (например, вторая страница при 10 записях на страницу).

Комбинирование условий

```
SELECT name, price FROM products
WHERE category = 'electronics'
ORDER BY price DESC
LIMIT 3;
```

1. Выбирает только **название и цену**
2. Фильтрует **только электронику**
3. Сортирует **от дорогих к дешевым**
4. Возвращает **топ-3 самых дорогих** товара в категории

5. Условия и операторы

Логические операторы

AND - позволяет объединять несколько условий. Все условия должны быть истинными для включения строки в результат.

Пример: Найти всех активных пользователей старше 18 лет

```
SELECT * FROM users WHERE age > 18 AND status = 'active';
```

Запрос вернет только тех пользователей, у которых одновременно возраст больше 18 И статус равен 'active'.

OR - позволяет задавать альтернативные условия. Достаточно выполнения хотя бы одного из условий.

Пример: Найти товары из категории электроники или дешевле 1000 рублей

```
SELECT * FROM products WHERE category = 'electronics' OR price < 1000;
```

Запрос вернет все товары, которые либо относятся к электронике, либо стоят меньше 1000, либо удовлетворяют обоим условиям.

NOT - инвертирует условие, возвращая строки, которые условию не соответствуют.

Пример: Найти все заказы, кроме отмененных

```
SELECT * FROM orders WHERE NOT status = 'cancelled';
```

Запрос вернет все записи о заказах, где статус НЕ равен 'cancelled'.

Операторы для поиска

IN - проверяет, содержится ли значение в указанном списке. Эквивалентен нескольким OR.

Пример: Найти города из России, Беларуси или Казахстана

```
SELECT * FROM cities WHERE country IN ('Russia', 'Belarus', 'Kazakhstan');
```

Запрос вернет все города, где страна соответствует одному из трех указанных значений.

BETWEEN - выбирает значения из заданного диапазона (включительно).

Пример: Найти продажи на сумму от 1000 до 5000 рублей

```
SELECT * FROM sales WHERE amount BETWEEN 1000 AND 5000;
```

Запрос вернет все записи о продажах, где сумма находится в диапазоне от 1000 до 5000 включительно.

LIKE - выполняет поиск по шаблону. % означает любую последовательность символов, _ - один символ.

Пример: Найти сотрудников с именем, начинающимся на "Иван"

```
SELECT * FROM employees WHERE name LIKE 'Иван%';
```

Запрос найдет всех сотрудников с именами типа "Иван", "Иванна", "Иванович" и т.д.

Работа с NULL

IS NULL/IS NOT NULL - специальные операторы для работы с NULL-значениями. NULL означает отсутствие данных.

Пример: Найти клиентов без указанного телефона

```
SELECT * FROM customers WHERE phone IS NULL;
```

Запрос вернет записи клиентов, у которых в поле phone указано NULL (значение отсутствует).

Несколько условий

Пример комбинированного запроса: Найти книги Пушкина или Толстого, изданные в XIX веке

```
SELECT * FROM books
WHERE (author = 'Пушкин' OR author = 'Толстой')
AND year BETWEEN 1800 AND 1900;
```

Запрос вернет книги, у которых автор либо Пушкин, либо Толстой, И при этом год издания между 1800 и 1900.

6. Агрегатные функции и группировка

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

COUNT() — подсчитывает количество строк.

```
SELECT COUNT(*) FROM users;
```

Запрос вернет общее количество записей в таблице `users`.

SUM() — вычисляет сумму значений.

```
SELECT SUM(price) FROM orders;
```

Запрос вернет сумму всех значений в столбце `price` таблицы `orders`.

AVG() — находит среднее значение.

```
SELECT AVG(age) FROM employees;
```

Запрос вычислит средний возраст сотрудников из таблицы `employees`.

MIN() и **MAX()** — находят минимальное и максимальное значения.

```
SELECT MIN(salary), MAX(salary) FROM workers;
```

Запрос покажет наименьшую и наибольшую зарплату в таблице `workers`.

GROUP BY — группирует строки по указанному столбцу.

```
SELECT department, COUNT(*) FROM employees GROUP BY department;
```

Запрос подсчитает количество сотрудников в каждом отделе.

HAVING — фильтрует группы после группировки (аналог `WHERE`, но для групп).

```
SELECT department, AVG(salary)
FROM employees
GROUP BY department
HAVING AVG(salary) > 50000;
```

Запрос выведет отделы, где средняя зарплата больше 50 000.

Примеры использования `GROUP BY` с разными агрегатными функциями

Агрегатные функции часто используют с `GROUP BY`, чтобы получать статистику по группам данных.

1. COUNT() + GROUP BY

Подсчет количества товаров в каждой категории:

```
SELECT category_id, COUNT(*) AS product_count
FROM products
GROUP BY category_id;
```

Запрос вернет количество товаров для каждого `category_id`.

2. SUM() + GROUP BY

Сумма заказов по каждому клиенту:

```
SELECT customer_id, SUM(amount) AS total_spent
FROM orders
GROUP BY customer_id;
```

Результат покажет, сколько всего потратил каждый клиент.

3. AVG() + GROUP BY

Средняя оценка по каждому курсу:

```
SELECT course_id, AVG(rating) AS avg_rating
FROM reviews
GROUP BY course_id;
```

Выводит средний рейтинг для каждого курса.

4. MIN() и MAX() + GROUP BY

Минимальная и максимальная цена в каждой категории:

```sql
SELECT category, MIN(price) AS min_price, MAX(price) AS max_price
FROM products
GROUP BY category;
```

Покажет диапазон цен (`min` и `max`) для каждой категории товаров.

5. GROUP BY + HAVING

Категории с более чем 10 товарами:

```sql
SELECT category_id, COUNT(*) AS product_count
FROM products
GROUP BY category_id
HAVING COUNT(*) > 10;
```

Выведет только те категории, где количество товаров превышает 10.

7. Связи между таблицами (JOIN)

JOIN позволяет объединять данные из нескольких таблиц по общему полю. Например, можно соединить таблицу `users` с таблицей `orders`, чтобы увидеть заказы каждого пользователя.

INNER JOIN возвращает только те строки, где есть совпадения в обеих таблицах. **LEFT JOIN** возвращает все строки из левой таблицы, даже если нет совпадений в правой.

Пример связи "один-ко-многим" — один пользователь может иметь много заказов:

```
SELECT users.name, orders.product
FROM users
INNER JOIN orders ON users.id = orders.user_id;
```

Этот запрос выводит имена пользователей и их заказы, только если у пользователя есть хотя бы один заказ.

Если нужно получить всех пользователей, даже без заказов, используем LEFT JOIN:

```
SELECT users.name, orders.product
FROM users
LEFT JOIN orders ON users.id = orders.user_id;
```

Теперь в результате будут все пользователи, а если заказов нет — вместо `product` будет `NULL`.

Пример с двумя INNER JOIN и WHERE:

```
SELECT users.name, orders.order_date, products.title, products.price
FROM users
INNER JOIN orders ON users.id = orders.user_id
INNER JOIN products ON orders.product_id = products.id
WHERE users.id = 5;
```

Этот запрос:

1. Соединяет таблицы `users`, `orders` и `products`
2. Выбирает имя пользователя, дату заказа, название и цену товара
3. Фильтрует результаты только для пользователя с `id = 5`

Результат покажет все заказы конкретного пользователя с деталями товаров.

Такие запросы часто используются для получения связанных данных из нескольких таблиц с дополнительной фильтрацией.

LEFT JOIN с условием WHERE IS NULL

LEFT JOIN возвращает все записи из левой таблицы, даже если нет совпадений в правой. Это полезно, например, для поиска пользователей **без** заказов.

```
SELECT users.id, users.name
FROM users
LEFT JOIN orders ON users.id = orders.user_id
WHERE orders.user_id IS NULL;
```

Как это работает:

1. `LEFT JOIN` сохраняет всех пользователей, даже если у них нет заказов.
2. `WHERE orders.user_id IS NULL` оставляет только тех, у кого **нет** записей в таблице `orders`.

Результат: список пользователей, которые никогда ничего не заказывали.

Этот приём часто используют для поиска "неактивных" записей в базе данных.