MySQL

Выборка, сортировка и фильтрация данных

Основы

Получение одного значения (столбца)

Главный и единственный оператор, который непосредственно отвечает за выборку данных в MySQL, носит название **SELECT**. Этот оператор может не только извлекать данные из таблиц баз данных, но и отображать результаты математических операций, выводить строки и числа, обработанные специальными функциями MySQL, а также показывать дату и время. **SELECT** всегда возвращает данные в виде таблицы.

```
SELECT 2 + 2 * 2;
```

2 + 2 * 2
6

Получение нескольких значений (столбцов) одновременно

Если нам необходимо узнать несколько результатов выражений сразу, то мы можем сделать запрос, перечислив эти выражения через запятую (функция **SQRT** требует в качестве единственного аргумента какое-либо число и возвращает корень из него, функция **POW** требует в качестве первого аргумента опять некое число, а в качестве второго аргумента степень, в которую мы будем возводить первый аргумент).

```
SELECT SQRT(4), 1 + 2, POW(3, 3);
```

SQRT(4)	1 + 2	POW(3, 3)
2	3	27

Псевдонимы столбцов

Псевдонимы столбцов можно задавать двумя способами - как с использованием ключевого слова **AS**, так и без него .

```
-- 1-й способ
SELECT
   SQRT(4) AS `square root`,
   1 + 2 AS addition,
   POW(3, 3) AS exponentiation;

-- 2-й способ
SELECT
   SQRT(4) `square root`,
   1 + 2 addition,
   POW(3, 3) exponentiation;
```

square root	addition	exponentiation
2	3	27

Выборка из таблиц (SELECT)

Подготовка для выборки из таблиц

Создание нужной нам таблицы:

```
CREATE TABLE products(
	id INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
	name VARCHAR(255),
	ean VARCHAR(255),
	price DECIMAL(20,2),
	created_at DATE NOT NULL
);
```

Заполнение таблицы данными:

```
INSERT INTO products(name, ean, price, created_at)
VALUES
('iPhone', '74748484', 999.99, '2023-01-01'),
('Samsung', '9393983', 1099.99, '2023-02-02'),
('Xiaomi', '35333653', 899.99, '2023-03-03');
```

Использование звездочки * для получения всех столбцов таблицы

С помощью звездочки * мы можем получить все столбцы таблицы в том порядке, который был задан при создании таблицы. **Однако звездочку рекомендуется использовать только в тестовых и образовательных случаях (если мы добавим новые столбцы в таблицу, то приложения, применяющие звездочку, могут сломаться).**

```
SELECT * FROM products;
```

id	name	ean	price	created_at
1	iPhone	74748484	999.99	2023-01-01
2	Samsung	9393983	1099.99	2023-02-03
3	Xiaomi	35333653	899.99	2023-03-03

Использование названий столбцов таблицы для получения значений этих столбцов

Если мы перечислим все столбцы таблицы через запятую в том же порядке, в каком они были перечислены при создании таблицы, то мы получим такой же результат, как при использовании звездочки.

```
SELECT id, name, ean, price, created_at FROM products;
```

id	name	ean	price	created_at
1	iPhone	74748484	999.99	2023-01-01
2	Samsung	9393983	1099.99	2023-02-03
3	Xiaomi	35333653	899.99	2023-03-03

Получение значений столбцов в произвольном порядке

Если нам необходимо получить столбцы не в том порядке, в каком они были объявлены при создании таблицы, то оператор звездочка уже не подходит - нам ничего не остается, как только перечислить их через запятую в том порядке, который нам нужен.

```
SELECT ean, price, created_at  name, id FROM products;
```

ean	price	created_at	name	id
74748484	999.99	2023-01-01	iPhone	1
9393983	1099.99	2023-02-03	Samsung	2
35333653	899.99	2023-03-03	Xiaomi	3

Выборка значений только необходимых столбцов

Нам не всегда понадобятся значения всех столбцов - иногда нужно будет получить только два или три столбца (или какое-то другое число). Чтобы это сделать, надо после **SELECT** указать через запятую только необходимые столбцы (например, в случае нашей таблицы **products** это могут быть **name** и **price**):

```
SELECT name, price FROM products;
```

name	price
iPhone	999.99
Samsung	1099.99
Xiaomi	899.99

Использование псевдонимов для столбцов таблицы

Присвоение псевдонимом может происходить как с ключевым словом AS, так и без него. Заменив в нашей таблице **products** наименование name на **title** (с применением **AS**), **ean** на **code**, а **price** - на **total**, мы получим следующий результат:

```
SELECT name AS title, ean code, price total FROM products;
```

title	code	total
iPhone	74748484	999.99
Samsung	9393983	1099.99
Xiaomi	35333653	899.99

Получение значений столбцов в измененном виде

Важной особенностью оператора **SELECT** является возможность на лету менять значения столбцов (в базе данных они останутся неизменными, а в результате мы отобразим по-другому). Например, предположим, что у нас временно повысились цены на все продукты в два раза - мы не хотим менять их в таблице, но хотим их отображать в увеличенном виде:

```
SELECT id, name, price * 2 FROM products;
```

id	name	price * 2
1	iPhone	1999.98
2	Samsung	2199.98
3	Xiaomi	1799.98

Выборка значений несуществующих столбцов

Еще одной интересной особенностью оператора **SELECT** является способность добавлять при выборке из таблиц несуществующие столбцы. В предыдущем примере мы просто меняли значение столбца **price**, а теперь оставим его без изменения, но затем запросим его вновь, но уже в увеличенном виде:

```
SELECT id, name, price, price * 2 FROM products;
```

id	name	price	price * 2
1	iPhone	999.99	1999.98
2	Samsung	1099.99	2199.98
3	Xiaomi	899.99	1799.98

Применение функций при выборках из таблицы

Кроме математических операций, мы также можем применять к значениям столбцов таблицы специальные функции MySQL. Например, в рекламных целях необходимо узнать, как длина названия продукта влияет на его продажи - для этого следует использовать функцию **CHAR_LENGTH**, которая принимает в качестве аргумента любой текст, а возвращает количество символов в этом тексте:

```
SELECT id, CHAR_LENGTH(name), price FROM products;
```

id	CHAR_LENGTH(name)	price
1	6	999.99
2	7	1099.99
3	6	899.99

Объединение нескольких значений столбцов в одно значение

Еще одной возможностью **SELECT** можно назвать объединение значений нескольких столбцов в один столбец. Например, новое начальство решило, что отныне в название продукта должен обязательно входить код **ean**. Чтобы не ломать уже существующую структуру, просто применим к запросу функцию **CONCAT**, которая принимает сколь угодно аргументов, а возвращает их соединенными вместе (в данном случае применен псевдоним **full_name**, но можно обойтись и без него):

```
SELECT id, CONCAT(name, ' (', ean, ')') AS full_name, price FROM products;
```

id	full_name	price
1	iPhone (74748484)	999.99
2	Samsung (9393983)	1099.99
3	Xiaomi (35333653)	899.99

Проверка на NULL при выборке значений

В таблице **products** столбцы **name**, **ean** и **price** не обязательно должны быть заполнены, т.е. потенциально туда можно этот **NULL** записать. Чтобы отследить такое положение дел для, например, столбца **name**, можно применить специальную функцию **IFNULL**. Если name будет равняться **NULL**, мы будем выводить какое-то значение по умолчанию, если нет - показывать оригинальное имя (применим псевдоним **name_alias**). В нашем случае таблица вернется в неизменном виде, так как у нас нет значений **NULL** в столбце **name**, но если бы были, то вместо них был бы вставлен текст *default name*:

```
SELECT id, IFNULL(name, 'default name') name_alias, price FROM products;
```

id	name_alias	price
1	iPhone	999.99
2	Samsung	1099.99
3	Xiaomi	899.99

Использование условной конструкции IF при выборке значений

Кроме функции **IFNULL**, есть более универсальная функция **IF**, которая принимает три значения - в-первых, некое выражение, которое может быть истинным или ложным, во-вторых, значение, которое будет выведено при истинности первого выражения и, в-третьих, значение, которое будет выведено при ложности первого выражения. Например, мы хотим вывести несуществующий столбец **status**, который показывает, дорогой у нас продукт или дешевый (продукт дороже 1000 будет считаться дешевым) - для этого **IF** подходит идеально:

```
SELECT id, name, price, IF(price > 1000, 'expensive', 'cheap') status FROM products;
```

id	name	price	status
1	iPhone	999.99	cheap
2	Samsung	1099.99	expensive
3	Xiaomi	899.99	cheap

Проверка на несколько условий при помощи CASE … WHEN … THEN … END

Иногда конструкции **IF** бывает недостаточно - представим, что у нас должно быть не два статуса, а три - продукты дороже 900, но дешевле или равные 1000 должны иметь статус **normal**. Для решения этой проблемы подходит несколько необычная конструкция **CASE ... WHEN ... THEN ... END**

```
SELECT id, name, price,
       CASE
       WHEN price > 1000 THEN 'expensive'
       WHEN price <= 1000 AND price > 900 THEN 'normal'
       ELSE 'cheap'
       END AS status
FROM products;
```

id	name	price	status
1	iPhone	999.99	normal
2	Samsung	1099.99	expensive
3	Xiaomi	899.99	cheap

Сортировка данных (ORDER BY)

Основы сортировки в MySQL

- По умолчанию MySQL не гарантирует порядок строк в результате выполнения выборки (хотя в подавляющем большинстве случаев строки будут возвращены в том порядке, в каком они были вставлены в таблицу). Однако если мы укажем явно порядок сортировки строк, то полученная выборка будет гарантированно отсортирована необходимым для нас способом.

- Сортировка осуществляется при помощи оператора **ORDER BY**, после которого указываются столбец (или столбцы), по которым должна идти сортировка. Кроме того, каждому столбцу можно явно указывать порядок, в котором будет происходить сортировка (от большего к меньшему или наоборот).

- При использовании оператора **ORDER BY** без дополнительных опций MySQL выполняет сортировку от меньшего к большему, но это можно указать явно, поставив после столбца оператор **ASC**. Если мы желаем сортировку от большего к меньшему, то после столбца надо поставить оператор **DESC**.

Сортировка от меньшего к большему (ASC)

Отсортируем наши продукты по цене в порядке возрастания (сначала будут идти продукты с меньшей ценой, а потом с большей) - как с использованием оператора **ASC**, так и без:

```
-- 1-й способ (с явным указанием направления сортировки)
SELECT id, name, price FROM products ORDER BY price ASC;

-- 2-й способ (без явного указания направления сортировки)
SELECT id, name, price FROM products ORDER BY price;
```

id	name	price
3	Xiaomi	↓ 899.99
1	iPhone	↓ 999.99
2	Samsung	↓ 1099.99

Сортировка от большего к меньшему (DESC)

Теперь сделаем обратное - отсортируем наши продукты по цене в порядке убывания (сначала будут идти продукты с большей ценой, а потом с меньшей) с использованием оператора **DESC**:

```
SELECT id, name, price FROM products ORDER BY price DESC;
```

id	name	price
2	Samsung	↑ 1099.99
1	iPhone	↑ 999.99
3	Xiaomi	↑ 899.99

Сортировка по столбцам, не участвующим в выборке

MySQL разрешает осуществлять сортировку даже по тем столбцам, которые не участвуют в выборке - в примере попробуем отсортировать по цене (**price**), но не получать ее:

```
SELECT id, name FROM products ORDER BY price;
```

id	name
3	Xiaomi
1	iPhone
2	Samsung

Сортировка по текстовым столбцам (лексикографический порядок)

MySQL также умеет сортировать и по текстовым столбцам, однако делает это в **лексикографическом порядке**. Символ, который идет в рамках своего алфавита после какого-либо другого символа, имеет больший “вес”, чем предыдущий символ. Это означает, что А меньше чем Ж, а '2' меньше, чем '5'. Кроме того, латинские буквы всегда будут идти перед кириллицей. Важный момент - при упомянутой лексикографической сортировке стравнение происходит посимвольно, т.е. в тот момент, когда символ в одной строке меньше символа на этой же самой позиции в другой строке, то первая строка считается имеющей “меньший вес”, а дальнейшее сравнение прекращается.

```
SELECT id, name, price, ean FROM products ORDER BY ean;
```

id	name	price	ean
3	Xiaomi	899.99	↓ 35333653
1	iPhone	999.99	↓ 74748484
2	Samsung	1099.99	↓ 9393983

Решение проблемы лексикографической сортировки

Чтобы отсортировать строки по текстовому столбцу **ean** в “нормальном порядке”, нам надо сначала отсортировать **ean** по его длине, т.е. сначала мы гарантированно получим порядок, в котором строки с меньшим количеством символов будут стоять вначале, а затем произведем стандартную сортировку по самому **ean** (т.е. среди строк с одинаковым количеством символов):

```
SELECT id, name, price, ean FROM products ORDER BY CHAR_LENGTH(ean), ean;
```

id	name	price	ean
2	Samsung	1099.99	9393983
3	Xiaomi	899.99	35333653
1	iPhone	999.99	74748484

Подготовка к сортировке по нескольким столбцам

В прошлом примере мы видели, как сортировка происходила сразу по двум столбцам - по некоему виртуальному столбцу, которого нет в таблице, а также по столбцу **ean**. Рассмотрим подробнее логику сортировки в таких случаях (сортировка по двум или более столбцам). Чтобы это осуществить, сначала занесем в таблице еще две записи, где опять фигурирует продукт Samsung (но уже с большей ценой), а также продукт Xiaomi (с меньшей ценой):

```
INSERT INTO products(name, ean, price, created_at)
VALUES
('Samsung', '5394983', 1199.99, '2023-04-04'),
('Xiaomi', '65322653', 799.99, '2023-05-05');
```

Сортировка по нескольким столбцам

Получим выборку продуктов таким образом, чтобы названия шли в алфавитном порядке (от меньшего к большего), а цены в обратном порядке (от большего к меньшего). Для этого применим сортировку по двум столбцам - названию и цене:

```
SELECT id, name, price, ean FROM products ORDER BY name ASC, price DESC;
```

Сначала были отсортированы имена в алфавитном порядке и были получены три группы - с именами iPhone, Samsung и Xiaomi . Затем в каждой группе произошла сортировка по цене в обратном порядке. Продукты в одной группе не могли “перепрыгнуть” в другую группу, они поменяли свое положение только в своей группе.

id	name	price	ean
1	↓ iPhone	↑↑ 999.99	74748484
4	↓ Samsung	↑↑ 1199.99	5394983
2	↓ Samsung	↑↑ 1099.99	9393983
3	↓ Xiaomi	↑↑ 899.99	35333653
5	↓ Xiaomi	↑↑ 799.99	65322653

Сортировка по порядковому номеру столбца

ORDER BY может сортировать не по имени столбца, а по его номеру в контексте выборки. Например, мы хотим получить выборку с участием трех столбцов - **id**, **price**, **name**. Мы можем указать **name** в качестве того поля, по которому мы будет сортировать, а можем указать номер 3. Понятно, что в данном случае нельзя сортировать по тем полям, которые не запрашиваются - у них нет номера, кроме того, к номерам нельзя применять функции.

```
SELECT id, price, name FROM products ORDER BY 3 ASC;
```

id	price	name
1	999.99	iPhone
2	1099.99	Samsung
4	1199.99	Samsung
3	899.99	Xiaomi
5	799.99	Xiaomi

Особенности символов в верхнем и нижнем регистре при сортировке

Будет учитываться регистр или нет зависит от атрибута **COLLATE** (сопоставление), который всегда явно или неявно следует за объявлением кодировки базы данных, таблицы или столбца. По умолчанию основная кодировка **utf8mb4** имеет **COLLATE** равное **utf8mb4_0900_ai_ci**, которое не учитывает регистр при сравнении строк (т.е. считает 'A' и 'a' одинаковыми). Чтобы регистр учитывался, можно ставить **utf8mb4** сущностям в качестве сопоставления **utf8mb4_0900_as_cs**

```
-- пример создания базы данных с регистрозависимостью
CREATE DATABASE example DEFAULT CHARSET utf8mb4 COLLATE
utf8mb4_0900_as_cs;

-- пример создания таблицы и столбца с регистрозависимостью
CREATE TABLE example (
    name VARCHAR(255) CHARACTER SET utf8mb4 COLLATE utf8mb4_0900_as_cs
) DEFAULT CHARSET utf8mb4 COLLATE utf8mb4_0900_as_cs;
```

Динамическое включение/выключение регистрозависимости при сортировке

При сортировке в случае необходимости регистрозависимость (или наоборот регистронезависимость) можно включить для одного конкретного запроса - это осуществляется при помощи команды **COLLATE**:

```
SELECT id, name FROM products ORDER BY name COLLATE utf8mb4_0900_as_cs;
```

Фильтрация данных встроенными средствами MySQL

Выборка уникальных строк (DISTINCT)

Если мы захотим узнать только имена (**name**) продуктов из нашей таблицы **products**, то мы получим таблицу из пяти строк, но в этой таблице значения Samsung и Xiaomi будут повторяться дважды. Однако иногда могут потребоваться строки без повторений. Это можно сделать с помощью оператора **DISTINCT**, который должен располагаться после оператора **SELECT** и перед названиями столбцов. **DISTINCT** гарантирует, что будут возвращены только те строки, комбинации значений столбцов которых уникальны:

```
SELECT DISTINCT name FROM products;
```

name
iPhone
Samsung
Xiaomi

Выборка уникальных строк (DISTINCT) с учетом регистра

На работу оператора **DISTINCT** влияют уже упоминавшиеся сопоставления (**COLLATE**). Если у нас не учитывается регистр, но если же мы хотим, чтобы он учитывался в рамках запроса на уникальную выборку, то можно сделать это следующим образом (подставим сопоставление после названий тех столбцов, где мы хотим учитывать регистр):

```
SELECT DISTINCT name COLLATE utf8mb4_0900_as_cs AS name FROM products;
```

Ограничение количества строк (LIMIT)

Очень часто нам надо получить не все строки в таблице, а только какую-то часть - например, первые две строки. Для задач такого рода предусмотрен специальный оператор **LIMIT** - воспользуемся им для решения нашей задачи:

```
SELECT id, name, price FROM products LIMIT 2;
```

id	name	price
1	iPhone	999.99
2	Samsung	1099.99

Ограничение количества строк (LIMIT) при сортировке (ORDER BY)

MySQL без явной сортировки потенциально может вернуть строки таблицы в любом порядке. Чтобы это пресечь, воспользуемся оператором **ORDER BY** и отсортируем строки по цене в возрастающем порядке, а уже после этого получим первые две строки. Перефразируя предыдущее предложение, можно сказать, что мы хотим получить два наиболее дешевых продукта. Дополнительно стоит упомянуть, что при таких случаях (когда вместе используются сортировка и лимитирование) сначала будет происходить сортировка, а только потом лимитирование количества строк:

```
SELECT id, name, price FROM products ORDER BY price LIMIT 2;
```

id	name	price
5	Xiaomi	799.99
3	Xiaomi	899.99

Ограничение количества строк (LIMIT) с использованием отступа (OFFSET)

Иногда нам необходимо получить строки не с самого начала, а с отступом - не первые две строки, а две строки, начиная с четвертой (т.е. пропустив первые три строки). Чтобы это реализовать, в MySQL есть оператор **OFFSET**, который работает в связке с **LIMIT**. После **OFFSET** мы указываем количество строк, которые мы хотим пропустить. Также существует альтернативный синтаксис в контексте самого **LIMIT** - после него можно указать через запятую две цифры - первая означает количество строк, которое мы хотим пропустить, а вторая цифра - количество строк, которое мы хотим получить:

```
-- 1-й вариант
SELECT id, name, price FROM products ORDER BY price LIMIT 2 OFFSET 3;

-- 2-й вариант
SELECT id, name, price FROM products ORDER BY price LIMIT 3,2;
```

id	name	price
2	Samsung	1099.99
4	Samsung	1199.99

Фильтрация данных согласно собственным условиям (WHERE)

Основы

- Выборка только уникальных строк и ограничение количества выбираемых строк - замечательные возможности, которые сильно помогают в работе с таблицами баз данных. Однако они не могут покрыть все возможные требования - очень часто нам необходимо применять к выборке очень нестандартные фильтры. Однако, как и всегда, в MySQL существует оператор **WHERE**, который позволяет решить эту проблему. Важно знать, что этот оператор срабатывает еще до того, как будут выполнены **ORDER BY** и **LIMIT** (если они включены в один и тот же запрос).

- Синтаксис **WHERE** крайне прост - сразу после него можно писать условия, согласно которым будут выбираться строки из таблицы. Если условие окажется ложно в рамках какой-то конкретной строки, то эта строка не будет включена в результат. Самые простые условия - сравнение значение какого-то столбца с каким-либо значением при помощи конструкций **>** (больше), **>=** (больше или равно), **<** (меньше), **<=** (меньше или равно), **=** (равно), **<>** или **!=** (не равно).

Простейший пример использования WHERE

Получим все продукты, цена которых больше и равна 900, а затем отсортируем по полученной цене (сортировку применять не обязательно - в данном случае она используется просто для примера):

```
SELECT id, name, price FROM products WHERE price >= 900 ORDER BY price;
```

id	name	price
1	iPhone	999.99
2	Samsung	1099.99
4	Samsung	1199.99

Несколько обязательных условий - оператор AND

WHERE позволяет использовать несколько условий одновременно - например, мы хотим выборку, в которой присутствуют продукты с именем Samsung и ценой более 1100 (эти два условия должны выполняться сразу, а не по отдельности). Чтобы отделять условия, применяется оператор **AND**:

```
SELECT id, name, price FROM products WHERE name = 'Samsung' AND price > 1100;
```

id	name	price
4	Samsung	1199.99

Хотя бы одно истинное условие - оператор OR

Иногда нам необходимо применить несколько условий, но мы согласны на то, чтобы истинным оказалось хотя бы одно из них (первое или второе или третье и т.д.). Для таких задач используется оператор **OR**. Попробуем выбрать все продукты, именем которых является iPhone или которые стоят дешевле 900:

```
SELECT id, name, price, created_at
FROM products
WHERE name = 'iPhone' OR price < 900;
```

id	name	price	created_at
1	iPhone	999.99	2023-01-01
3	Xiaomi	899.99	2023-03-03
5	Xiaomi	799.99	2023-05-05

Комбинация операторов AND и OR

Если после **WHERE** будет установлено несколько условий, которые будут разделены как **OR**, так и **AND**, то сначала будут проверены условия, разделенные **AND**, а уже потом **OR**:

```
SELECT id, name, price, created_at FROM products
WHERE name = 'Samsung' OR price = 999.99 AND created_at > '2023-02-03';
```

Несмотря на то, что под комбинацию двух последних условий не подходит не один продукт, нам будет возвращены два продукта Samsung, т.к. они подходят под первое условие, после которого стоит **OR** (или) и которое было выполнено последним:

id	name	price	created_at
2	Samsung	1099.99	2023-02-02
4	Samsung	1199.99	2023-04-04

Изменение поведения OR и AND при помощи скобок

Стандартное поведение операторов **OR** и **AND** можно изменить, если расставить скобки. Например, следующий запрос идентичен предыдущему во всем, кроме скобок, но он вернет только один продукт Samsung, т.к. скобки поменяли смысл фильтрации:

```
SELECT id, name, price, created_at
FROM products
WHERE (name = 'Samsung' OR price = 999.99) AND created_at > '2023-02-03';
```

id	name	price	created_at
4	Samsung	1199.99	2023-04-04

Применение оператора LIKE (1)

Помимо операций сравнения (больше, меньше и т.д.), в MySQL существует еще ряд других операторов, которые используются в контексте главной директивы **WHERE**. Один из самых интересных таких операторов носит название **LIKE**. Он позволяет искать строки не по конкретным значениям, а по части значения какого-либо текстового столбца. Например, нам надо получить выборку всех строк, которые начинаются с буквы X, после которой могут идти любые другие символы (или вообще ничего не идти). Любые другие символы (или их отсутствие) обозначается символом **%**:

```
SELECT id, name, price FROM products WHERE name LIKE 'X%';
```

id	name	price
3	Xiaomi	899.99
5	Xiaomi	799.99

Применение оператора LIKE (2)

Символ **%** может также стоять в самом начале шаблона поиска - это означает, что в самом начале может находиться что-угодно. Попробуем применить это знание, чтобы найти все продукты, которые заканчиваются на **g**:

```
SELECT id, name, price FROM products WHERE name LIKE '%g';
```

id	name	price
2	Samsung	1099.99
4	Samsung	1199.99

Применение оператора LIKE (3)

Приведем пример символа **%** в нескольких местах одновременно - попробуем найти все продукты, которые содержат букву n (т.е. что угодно может стоять как до этой буквы, так и после). В качестве результата мы получим один продукт iPho**n**e и два продукта Samsu**n**g:

```
SELECT id, name, price FROM products WHERE name LIKE '%n%';
```

id	name	price
1	iPhone	999.99
2	Samsung	1099.99
4	Samsung	1199.99

Применение оператора LIKE (4)

Еще один символ, который используется в контексте оператора **LIKE** - символ нижнего подчеркивания **_**. В отличие от **%** он означает не сколько угодно символов, а любой, но один символ:

```
SELECT id, name, price FROM products WHERE name LIKE 'iPho_';
```

В нашей таблице после символов iPho могут идти только два символа (ne), а не один. Поэтому в данном случает мы получим пустую таблицу. Если же мы хотим получить, что-то осмысленное, то добавим нижнее подчеркивание два раза:

```
SELECT id, name, price FROM products WHERE name LIKE 'iPho__';
```

id	name	price
1	iPhone	999.99

Применение оператора LIKE (5)

Заканчивая часть об операторе **LIKE**, отметим, что он также зависит от сопоставлений (**COLLATE**), т.е. верхний и нижний регистр букв при сравнении будет зависеть от того, какое сопоставление установлено у столбца, таблицы или всей базы данных. Если же мы хотим установить свое сопоставление только на время конкретного запроса, то это делается следующим образом:

```
SELECT id, name, price
FROM products
WHERE name LIKE 'iPho__' COLLATE utf8mb4_0900_as_cs;
```

Применение оператора BETWEEN (численные типы)

Еще одним оператором, использующимся в связке с глобальной директивой **WHERE**, является **BETWEEN**. Как можно понять из названия, он помогает выявить те значения столбцов, которые находятся в определенном интервале (от и до включая). Именно поэтому **BETWEEN** осуществляет сравнение значения не с один другим значением, а с двумя и при этом использует уже упоминавшийся оператор **AND**, но уже в другом контексте. Например, нам надо получить те продукты, цена который между (**BETWEEN**) 900 и (**AND**) 1100, включая оба значения:

```
SELECT id, name, price FROM products WHERE price BETWEEN 900 AND 1100;
```

id	name	price
1	iPhone	999.99
2	Samsung	1099.99

Применение оператора BETWEEN (типы даты и времени)

Кроме чисел, оператор **BETWEEN** можно использовать и для нахождения значений в неком текстовом интервале (с учетом лексикографического сравнения и сопоставлений), а также временных типов данных (**DATETIME**, **DATE** и т.д.). Например, получим те продукты, которые были добавлены в таблицу (**created_at**) в период времени 2 февраля 2023 года и 4 апреля 2023 года (включая оба значения) - для наглядности включим в выборку сам столбец **created_at**:

```
SELECT id, name, price, created_at
FROM products
WHERE created_at BETWEEN '2023-02-02' AND '2023-04-04';
```

id	name	price	created_at
2	Samsung	1099.99	2023-02-02
3	Xiaomi	899.99	2023-03-03
4	Samsung	1199.99	2023-04-04

Применение оператора IN

Следующая конструкция - оператор **IN** - позволяет определить определенный набор разрешенных значений - если значение столбца равно хотя бы одному из этих значений, то вся строка, имеющая столбец с разрешенным значением, попадет в результирующую таблицу. Например, попытаемся получить все продукты, id которых могут иметь следующие значения 2, 20, 1000, 5, '1'. Особо отметим, что в разрешенные значения мы вписали 1 как строку, а столбец id является целочисленным. Это означает, что MySQL пытается преобразовать разрешенные значения в нужный тип - если у него это получается, то подходящие значения будут вставлены в ответ:

```
SELECT id, name, price FROM products WHERE id IN (2, 20, 1000, 5, '1');
```

id	name	price
1	iPhone	999.99
2	Samsung	1099.99
5	Xiaomi	799.99

Применение оператора NOT IN

У оператора **IN** есть конструкция-антипод **NOT IN**, которая делает обратное - определяет список значений, которым значение столбца равняться не должно. Повторим запрос из предыдущего примера, но вместо **IN**, подставим **NOT IN**. Как и ожидалось, в этот раз ни одной строки из предыдущего результата мы не увидим:

```
SELECT id, name, price FROM products WHERE id NOT IN (2, 20, 1000, 5, '1');
```

id	name	price
3	Xiaomi	899.99
4	Samsung	1199.99

Операторы IS NULL и IS NOT NULL (основы)

Есть еще одна пара конструкций, которая используются совместно с директивой **WHERE** - **IS NULL** и **IS NOT NULL**. Они используются для сравнения со столбцом, который может содержать **NULL** в качестве значения. Однако сразу же возникает вопрос - зачем нам такие конструкции, если у нас уже есть **=** и **<>**? Чтобы на этот вопрос ответить, внесем в нашу таблицу products еще один продукт, у которого **ean** будет равен **NULL**:

```
INSERT INTO products(name, ean, price, created_at)  VALUES
('Nokia', NULL, 950.99, '2023-06-06');
```

После успешной вставки, попробуем выбрать все строки у которых ean равен NULL при помощи простого сравнения =:

```
SELECT id, name, price, ean FROM products WHERE ean = NULL;
```

Мы обнаружим, что нам вернулась пустая таблица. Дело в том, что **NULL** - это специальное значение, которое обозначает “отсутствие значение”. Поэтому **NULL** не может быть равен какому-либо другому значению, даже другому **NULL**.

Применение оператора IS NULL

Если если нам надо выбрать те строки, у которых в столбце ean сохранен именно **NULL**, то нам следует применять не равно (=), а конструкцию **IS NULL**:

```
SELECT id, name, price, ean FROM products WHERE ean IS NULL;
```

id	name	price	ean
6	Nokia	950.99	NULL

Применение оператора IS NOT NULL

Конструкция **IS NOT NULL**, как и ожидается, вернет все строки, кроме той, которую мы вставили последней (с **ean** равной **NULL**):

```
SELECT id, name, price, ean FROM products WHERE ean IS NOT NULL;
```

id	name	price	ean
1	iPhone	999.99	74748484
2	Samsung	1099.99	9393983
3	Xiaomi	899.99	35333653
4	Samsung	1199.99	5394983
5	Xiaomi	799.99	65322653

Фильтрация по столбцам, отсутствующим в выборке

Директива **WHERE** и все связанные с ней конструкции могут осуществлять выборку и по значениям тех столбцов, которые не перечислены после **SELECT**. Возможно, для кого-то это очевидно, но на всякий случай подтвердим утверждение примером. Выберем столбцы name и price для тех строк, у которых значение id - четное число (т.е. условие ориентируется на столбец **id**, а выбираются name и **price**):

```
SELECT name, price FROM products WHERE id % 2 = 0;
```

name	price
Samsung	1099.99
Samsung	1199.99

Выборка из столбцов с типом JSON

Подготовка для выборки из таблиц с типом JSON

При выборке данных тип **JSON** по причине своего нестандартного формата должен быть обработан специфичным образом. Чтобы это продемонстрировать, создадим таблицу **documents** с двумя полями - **id** (целочисленный тип) и data (**JSON**):

```
CREATE TABLE documents (
   id INT AUTO_INCREMENT NOT NULL PRIMARY KEY,
   data JSON NOT NULL
);
```

Затем занесем в нашу таблицу два документа:

```
INSERT INTO documents (data)
   VALUES
('{"title": "1st Document", "content": {"en": "Text 1", "ru": "Текст 1"}, "paragraphs": [10, 55, 18]}'),
('{"title": "2nd Document", "content": {"en": "Text 2", "ru": "Текст 2"}, "paragraphs": [16, 47, 98]}');
```

Попытка получить результат при помощи простого сравнения (столбец JSON)

Попробуем получить первую выборку по условию **WHERE** (в следующем запросе значение после **WHERE** идентично тому значению, которое мы вставляли первым):

```
SELECT id, data FROM documents
WHERE data = '{"title": "1st Document", "content": {"en": "Text 1", "ru": "Текст 1"},
"paragraphs": [10, 55, 18]}';
```

Удивительно, но мы получим в ответ пустую таблицу. JSON нельзя получить, если мы пытаемся сравнить значение столбца со строкой - тут требуется дополнительная обработка и синтаксис, который, например, должен быть ориентирован на какое-то значение внутри самой сохраненной JSON строки.

Фильтрация по внутренним полям JSON

Если мы ожидаем, что внутри JSON строки (столбец **data**) есть поле под названием **'title'** и со значением **'1st Document'**, то запрос необходимо сделать следующим образом:

```
SELECT id, data FROM documents WHERE data->'$.title' = '1st Document';
```

id	data
1	{ "title": "1st Document", "content": { "en": "Text 1", "ru": "Текст 1" }, "paragraphs": [10, 55, 18] }

Фильтрация по внутренним вложенным полям JSON

Если у нас значение имеет вложенность, то части пути в запросе разделяются точкой. Попробуем получить документ, к которого в поле **en** поля **content** хранится значение 'Text 2':

```
SELECT id, data FROM documents WHERE data->'$.content.en' = 'Text 2';
```

id	data
2	{ "title": "2nd Document", "content": { "en": "Text 2", "ru": "Текст 2" }, "paragraphs": [16, 47, 98] }

Фильтрация по значениям массивов внутри JSON

Если при поиске по JSON столбцу, искомое значение входит в массив, то надо указать индекс этого значения (в массивах нумерация начинается с нуля):

```
SELECT id, data FROM documents WHERE data->'$.paragraphs[1]' = 55;
```

id	data
1	{ "title": "1st Document", "content": { "en": "Text 1", "ru": "Текст 1" }, "paragraphs": [10, 55, 18] }

Фильтрация по полному значению столбца JSON

Мы уже рассматривали ситуацию, когда мы хотели получить данные по значению всего JSON. Это можно сделать, но сравнивать надо не со строкой, а со специально сконструированным объектом, который отображает весь JSON, хранимый в таблице:

```
SELECT id, data FROM documents
WHERE data = JSON_OBJECT(
      "title", "1st Document",
      "content", JSON_OBJECT("en", "Text 1", "ru", "Текст 1"),
      "paragraphs",  JSON_ARRAY(10, 55, 18)
);
```

Получение значения внутреннего поля JSON

Иногда нам надо получить не весь JSON, а значение какого-либо поля внутри JSON строки. Чтобы этого добиться, необходимо использовать синтаксис, похожий на тот, который уже использовался в директиве **WHERE**, но с использованием функции **JSON_EXTRACT**. Для начала получим значение поля **title**:

```
SELECT id, JSON_EXTRACT(data, '$.title') AS title FROM documents;
```

id	title
1	1st Document
2	2nd Document

Получение вложенного значения внутреннего поля JSON

Получим вложенное значение поля **en** поля **content** (части пути разделяется точкой):

```
SELECT id, JSON_EXTRACT(data, '$.content.en') AS content_en FROM documents;
```

id	content_en
1	Text 1
2	Text 2

Получение значения массива внутри поля JSON

```
SELECT
   id,
   JSON_EXTRACT(data, '$.paragraphs[2]') AS third_paragraph
FROM documents;
```

id	third_paragraph
1	18
2	98