

SQL

- [Тестовые данные и общая информация](#)
- [Типы данных и наборы символов](#)
- [Таблицы](#)
- [Объединения, соединения и группировки](#)
- [Подзапросы](#)
- [Встроенные функции и условная логика](#)
- [Транзакции, индексы и ограничения](#)
- [Представления и метаданные](#)
- [Администрирование](#)
 - [MySQL](#)
 - [Postgresql](#)
- [Для теста](#)

Тестовые данные и общая информация

Перейдите по [ссылке](#) загрузите “sakila database” в разделе Example Databases.

```
wget https://downloads.mysql.com/docs/sakila-db.zip
unzip sakila-db.zip
MariaDB [sakila]> source /home/kali/sql/sakila-db/sakila-schema.sql
MariaDB [sakila]> source /home/kali/sql/sakila-db/sakila-data.sql
```

- К некоторым БД нельзя отправить запрос без from. Поэтому таблица dual со столбцом dummy: `SELECT now() FROM dual;`
- Можно подключить вывод отсета в xml: `mysql -u lrngsql -p --xml bank`
- При соединении с БД возвращается id
Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 11
Server version: 8.0.15 MySQL Community Server - GPL
- В операциях сравнения использовать is null. Значение может быть null, но оно не может быть равно null
- Если столбец autoincrement not null, то `0 = NULL = value+1`

Типы данных и наборы СИМВОЛОВ

Наборы символов:

Список установленных наборов

```
SHOW CHARACTER SET;
```

По умолчанию utf 8mb4

varchar(20) character set latin1 : при создании таблицы другой набор

набор по-умолчанию для всей таблицы

```
create database european_sales character set latin1;
```

Общая информация

- M максимальное количество. Для целых - кол-во цифр, для дробных - общее кол-во цифр, для строк макс. кол-во символов
- D кол-во цифр после запятой, должно быть не больше M-2
- fsp применимо к дате/времени, определяет точность (кол-во дробной части секунды)
- при strict mode перебор значения вызывает исключение, при restrictive - максимально возможное значение

Тип	Доп. информация
char(255 символов) varchar(65535)	char(20) 20 символов char фиксировано, при хранении добавляет справа PAD символы
tinytext(255) text(65535) mediumtext(16 млн) longtext(4 млрд)	если больше, то обрежутся конечные пробелы считаются при сортировке первые 1024 байта, но можно увеличить tinytext, text - можно не использовать

Тип	Доп. информация
tinyint(1) smallint(2) mediumint int bigint(8)	<p>указано кол-во байтов при создании указывается визуальное кол-во цифр+один символ на знак</p> <pre>CREATE TABLE employees (salary INTEGER(5) UNSIGNED);</pre> <p>int максимум 10, поэтому это преобразуется в int(11)</p> <pre>CREATE TABLE employees (id INT(255));</pre> <p>ZEROFILL заполняет 0 слева.</p> <pre>CREATE TABLE documents (document_no INT(5) ZEROFILL);</pre>
DECIMAL = NUMERIC	<p>фиксированное кол-во цифр. Равно int с коэффициентом деления. Первая цифра - общее кол-во цифр (M), вторая - кол-во цифр после запятой. Знак неявно добавляется. Т.е. от - 9.99 до 9.99</p> <pre>CREATE TABLE taxes (tax_rate DECIMAL(3, 2));</pre> <p>M < 65, по-умолчанию 10 при переборе ошибка, при выходе за пределы точности - округление по математическим правилам способ вывода больших чисел с игнором младших цифр</p> <pre>select truncate(tax_rate * 1000, 0) from taxes;</pre> <p>в агрегатных функциях работает общее ограничение на 64 цифры, вычисление точное.</p>

Тип	Доп. информация
float, double	<p>приближенные числа отображаются как были внесены, но в агрегатных функциях могут появиться неточности</p> <div><pre>CREATE TABLE typed_numbers(id TINYINT, float_values FLOAT, decimal_values DECIMAL(3, 2)); INSERT INTO typed_numbers VALUES (1, 1.1, 1.1), (2, 1.1, 1.1), (3, 1.1, 1.1); SELECT * FROM typed_numbers; ┌ id float_values decimal_values ├+-----+-----+-----+ ┌ 1 1.1 1.10 ┌ 2 1.1 1.10 ┌ 3 1.1 1.10 SELECT SUM(float_values), SUM(decimal_values) FROM typed_numbers; ┌ SUM(float_values) SUM(decimal_values) ├+-----+-----+ ┌ 3.3000000715255737 3.30 </pre></div> <p>Нельзя напрямую сравнивать в агрегатах, нужно сравнивать разницу $ABS(v1 - v2) > 0.1$</p> <div><pre>SELECT id, SUM(col1) as v1, SUM(col2) as v2 FROM temp GROUP BY id HAVING v1 <> v2;</pre></div> <p>нельзя, внешне одинаковые числа будут разными</p>

Тип	Доп. информация
битовый тип	<div><div>column_name BIT(M)</div><div><div>CREATE TABLE bit_values (val BIT(7));</div><div>INSERT INTO bit_values VALUES(b'1011'); - равно</div><div>VALUES(b'0001011'), в этой нотации b и B равны.</div><div>INSERT INTO bit_values VALUES(0b1011);</div><div>INSERT INTO bit_values VALUES(2); -</div><div>автоматическое преобразование числа в</div><div>битовую маску</div></div><div><div>select выводит в виде 0x0A</div><div>select bin(...) - без 0 слева</div><div>select lpad(bin(...), 7, '0') - привычный вид с 0 слева</div></div></div>

Тип	Доп. информация
date datetime timestamp year time	<p>00-00-0000 вместо NULL, не отображается</p> <div> <p>%a Краткое имя дня недели - Sun, Mon, ...</p> <p>%b Краткое имя месяца — Jan, Feb, ...</p> <p>%c Числовое значение месяца (0..11)</p> <p>%d Числовое значение дня месяца (00..31)</p> <p>%f Число микросекунд (000000..999999)</p> <p>%H Час дня в 24-часовом формате (00..23)</p> <p>%h Час дня в 12-часовом формате (01..12)</p> <p>%i Минуты в часе (00..59)</p> <p>%j День года (001..366)</p> <p>%M Полное имя месяца (January..December)</p> <p>%m Числовое значение месяца</p> <p>%p AM или PM</p> <p>%s Число секунд (00..59)</p> <p>%W Полное имя дня недели (Sunday..Saturday)</p> <p>%w Числовое значение дня недели (0=Sunday..6=Saturday)</p> <p>%Y Значение года (четыре цифры)</p> </div> <p>timestamp зависит от time_zone</p> <div> <pre> CREATE TABLE datetime_temp(ts TIMESTAMP, dt DATETIME); INSERT INTO datetime_temp VALUES(NOW(), NOW()); SELECT ts, dt FROM datetime_temp; □ 2017-10-14 18:10:25 2017-10-14 18:10:25 SET time_zone = '+03:00'; SELECT ts, dt FROM datetime_temp; □ 2017-10-14 21:10:25 2017-10-14 18:10:25 </pre> </div> <p>Если сервер MAXDB mode, TIMESTAMP = DATETIME column_name TIME;</p> <div> <p>838:59:59 to 838:59:59 + 6 цифр миллисекунд некорректное в 00:00:00, нельзя отличить от корректно добавленного 00:00:00</p> </div>

Таблицы

Типы таблиц:

- Постоянные таблицы (т.е. созданные с помощью инструкции create table)
- Производные таблицы (т.е. строки, возвращаемые подзапросом и хранящиеся в памяти)
- Временные таблицы (т.е. изменяемые данные, хранящиеся в памяти)
- Виртуальные таблицы (т.е. созданные с помощью инструкции create view).

Создание таблицы

```
create table person (  
  person_id smallint unsigned,  
  fname varchar(20),  
  lname varchar(20),  
  eye_color char(2),  
  constraint pk_person primary key (person_id)  
);
```

изменение структуры таблицы

```
ALTER TABLE person MODIFY person_id SMALLINT UNSIGNED AUTO_INCREMENT;
```

какие таблицы ссылаются по внешним ключам на таблицу X

```
SELECT * FROM information_schema.KEY_COLUMN_USAGE WHERE REFERENCED_TABLE_NAME = 'X';
```

```
DROP TABLE favorite_food;
```

Ограничения (constraint)

Первичный ключ, может быть любое кол-во столбцов

```
constraint pk_person primary key (person_id)
```

Создание связи

```
CONSTRAINT fk_fav_food_person_id FOREIGN KEY (person_id) REFERENCES person (person_id)
```


Ограничение набора в столбце

```
enum('ht', 'hy')  
create table person (... eye_color ENUM('BR','BL','GR', ....);
```

AUTO_INCREMENT в описании ключа автоматическое увеличение номера

desc person - описание таблицы

not null при создании столбца запрещает быть пустым

проверка ограничений

```
SELECT * FROM information_schema.check_constraints WHERE table_schema = 'db name' and table_name =  
'table name';
```

просмотр скрипта создания таблицы со всеми внесенными изменениями

```
SHOW CREATE TABLE mytable;
```

Получение/изменение данных

```
INSERT INTO person (person_id, fname, lname, eye_color, birth_date)
```

```
VALUES (null, 'William','Turner', 'BR', '1972-05-27');
```

ORDER BY - группировка в селекте

```
UPDATE person
```

```
SET street = '1225 Tremont St.', city = Boston', state = 'MA', country = 'USA', postal_code = '02138' WHERE  
person_id = 1;
```

```
DELETE FROM person WHERE person_id = 2;
```

Структура select

select	Определяет,какие столбцы следует включить в результирующий набор запроса
from	Определяет таблицы, из которых следует выбирать данные, а также таблицы, которые должны быть соединены
where	Отсеивает ненужные данные
group by	Используется для группировки строк по общим значениям столбцов
having	Отсеивает ненужные данные
order by	Сортирует строки окончательного результирующего набора по одному или нескольким столбцам

Фильтрация

Логические операции в where: and, or, not, для группировки круглые скобки, операторы сравнения, такие как =, !=, <, >, like, in и between; арифметические операторы, такие как +, -, * и /.

Подзапросы в фильтрах:

```
... where film_id = (SELECT film_id FROM film WHERE title = 'RIVER OUTLAW') ...
```

Двухсторонний диапазон. Сначала нижняя граница включительно.

```
WHERE rental date BETWEEN '2005-06-14 AND '2005-06-16';
```

```
WHERE amount BETWEEN 10.0 AND 11.99;
```

```
WHERE last name BETWEEN 'FA' AND 'FR' - FRA не войдет. Можно 'FRB'
```

множество IN, NOT IN - WHERE rating IN ('G', 'PG');

первая буква Q - WHERE left(last name, 1) = 'Q';

регулярки: WHERE last_name REGEXP '^[QY]'

WHERE return_date IS NULL; (is not null) Если WHERE return_date = NULL; - ошибки не будет, но пустая выборка

он результат BETWEEN 180 AND 240;

exist - наличие хотя-бы одного значения в подмножестве

```
SELECT c.first_name, c.last_name  
FROM customer c WHERE EXISTS  
  (SELECT 1 FROM rental r  
   WHERE r.customer_id = c.customer id  
   AND date(r.rental date) < '2005-05-25');
```


Объединения, соединения и группировки

Логическое объединение таблиц

объединяются по толбцам последовательно. Лучше одинаковые псевдонимы.

union (или) объединяет построчно +сортировка +удаление дубликатов, union all без +. Есть предопределенный необязательный столбец typ для ссылки на источник для строки.

intersect (И)

except (не) все что есть в первом исключая повторяющиеся со вторым

order by можно добавить после последнего запроса, относится к первому

выполнение сверху вниз, но intersect высший приоритет, можно использовать скобки

Соединения таблиц

ограничение внешнего ключа не является обязательным условием, чтобы соединить две таблицы. on - условие соединения таблиц. Если во всех таблицах совпадают названия столбцов, то USING (address_id); Using лучше не использовать.

```
SELECT c.first_name, c.last_name, time(r.rental_date) rental_time
FROM customer c
INNER JOIN rental r
ON c.customer_id = r.customer_id WHERE date(r.rental_date) = '2015-04-03';
```

несколько таблиц, порядок перечисления значения не имеет:

```
select c.first_name, c.last_name, ci.city
from city ci
inner join address a on a.address_id = ci.address_id
inner join customer c on a.address_id = c.address_id;
```

с подзапросом:

```
SELECT c.first_name, c.last_name, addr.address, addr.city
FROM customer c
INNER JOIN
  (SELECT a.address_id, a.address, ct.city
   FROM address a
```

```
INNER JOIN city ct ON a.city_id = ct.city_id WHERE a.district = 'California') addr
ON c.address id = addr.address id;
```

если в inner join несколько раз соединение с одной таблицей, то псевдонимы обязательны. возможно ссылаться на саму себя (например если в таблице есть строка, указывающая на родителя)

Типы соединений

- inner join - на каждую строку из левой таблицы строки из правой. (оператор И)
- left/right outer join - все строки левой таблицы, если в правой есть то добавляем иначе null
- cross join - перекрестное (чистое декартовое) соединение. Используется при создании новых данных.
- natural join - типа естественное соединение, предоставляющее выбор типа базе. Лучше не использовать.

Пример cross join

```
SELECT days.dt, COUNT(r.rental_id) num_rentals FROM rental r
RIGHT OUTER JOIN (SELECT DATE_ADD('2005-01-01', INTERVAL (ones.num + tens.num + hundreds.num) DAY)
dt
FROM
(SELECT 0 num UNION ALL
SELECT 1 num UNION ALL
SELECT 2 num UNION ALL
SELECT 3 num UNION ALL
SELECT 4 num UNION ALL
SELECT 5 num UNION ALL
SELECT 6 num UNION ALL
SELECT 7 num UNION ALL
SELECT 8 num UNION ALL
SELECT 9 num) ones
CROSS JOIN
(SELECT 0 num UNION ALL
SELECT 10 num UNION ALL
SELECT 20 num UNION ALL '
SELECT 30 num UNION ALL
SELECT 40 num UNION ALL
SELECT 50 num UNION ALL
SELECT 60 num UNION ALL
SELECT 70 num UNION ALL
```

```
SELECT 80 num UNION ALL
SELECT 90 num) tens
CROSS JOIN
(SELECT 0 num UNION ALL
SELECT 100 num UNION ALL
SELECT 200 num UNION ALL
SELECT 300 num) hundreds
WHERE DATE_ADD('2005-01-01', INTERVAL (ones.num + tens.num + hundreds.num) DAY) < '2006-01-01' days
ON days.dt = date(r.rental_date) GROUP BY days.dt ORDER BY 1
```

Группировка

```
SELECT customer_id, count(*) FROM rental GROUP BY customer id;
```

сортировка по count - либо по номеру, либо определив имя столбца и по имени. Условия для агрегатных функций до создания полного набора, поэтому where count(*) > 40 нельзя. Можно having SELECT customer_id, count(*) FROM rental GROUP BY customer_id HAVING count(*) >= 40; но where можно для условий данных WHERE f.rating IN ('G', 'PG')

Агрегатные функции:

count(*)

```
COUNT(DISTINCT customer_id)
```

```
SELECT
  MAX(amount) max_amt,
  MIN(amount) min_amt,
  AVG(amount) avg_amt,
  SUM(amount) tot_amt,
  COUNT(*) num_payments
FROM payment;
```

агрегаты применяются либо ко всей выборке, либо к группам определенным в group by в аргументе агрегатов любые функции, возвращают число строку дату null: игнорируется при расчетах. Но count(*) отдает кол-во строк, и null считается. То есть count(val) число значений столбца без null, count(*) число строк в столбце val

Способы группировки

Множественная:

```
SELECT fa.actor_id, f.rating, count(*)  
FROM film_actor fa INNER JOIN film f ON fa.film_id = f.film_id  
GROUP BY fa.actor_id, f.rating ORDER BY 1,2;
```

Выражения:

```
SELECT extract(YEAR FROM rental_date) year, COUNT(*) how_many  
FROM rental  
GROUP BY extract(YEAR FROM rental date);
```

Добавка подсумм:

```
GROUP BY fa.actor_id, f.rating WITH ROLLUP
```

Непересекающееся множество DISTINCT (затратная операция)

```
SELECT DISTINCT actor_id FROM film_actor ORDER BY actor_id;
```

В столбцы можно добавлять вычисляемые поля
order by несколько столбцов
desc по убыванию ORDER BY time(r.rental date) desc;
можно использовать номер столбца в перечислении

Подзапросы

подзапрос выполняется до основного запроса, это временная таблица с областью видимости запроса, после выполнения запроса данные удаляются. Если возврат одну строку и один столбец, то в основном запросе используется равенство (<>, >=,...)

```
SELECT customer_id, first_name, last_name FROM customer WHERE customer_id = (SELECT MAX(customer_id)
FROM customer);
```

если множество строк:

in, not in в результирующем множестве

<>=all All сравнивает значение с каждым значением в подзапросе

<>=any - если хотя-бы одно выполняется, то true HAVING sum(amount) > ANY (подзапрос)

если в результате подзапроса одно из значений null, а сравнение not in, то будет пустой вывод: результат сравнения с null - unknown

Типы подзапросов

некоррелированный: может быть выполнен отдельно и не ссылается ни на что из другого

коррелированный: когда в подзапросе используется таблица запроса. выполняется по одному разу для каждой строки-кандидата. Может быть проблема с производительностью. часто вместе с update, delete

```
UPDATE customer c
SET c.last_update = (SELECT max(r.rental_date) FROM rental r WHERE r.customer_id = c.customer_id);
```

проверка наличия

```
UPDATE customer c SET c.last_update =
(SELECT max(r.rental_date) FROM rental r WHERE r.customer_id = c.customer_id)
WHERE EXISTS
(SELECT 1 FROM rental r WHERE r.customer_id = c.customer_id);
```

```
select f.title from film f where exist (
select 1 from film_category fc where category_id = 10 and f.film_id = fc.film_id)
```

создание данных подзапросом. Нужно разделить на группы, когда такой группировки нет в базе


```

SELECT pymnt_grps.name, count(*) num_customers FROM
[(SELECT customer_id, count(*) num_rentals, sum(amount) tot_payments
  FROM payment GROUP BY customer_id) pymnt
INNER JOIN
[(SELECT 'Small Fry' name, 0 low_limit, 74.99 high_limit UNION ALL
[SELECT 'Average Joes' name, 75 low_limit, 149.99 high_limit UNION ALL
[SELECT 'Heavy Hitters' name, 150 low_limit, 9999999.99 high_limit) pymnt_grps
ON pymnt.tot_payments BETWEEN pymnt_grps.low_limit AND pymnt_grps.high_limit
GROUP BY pymnt_grps.name;

```

Обобщенные табличные выражения: создается именованный подзапрос, который потом используется в запросе with actor_s as (запрос) Например общий доход от проката тех фильмов с рейтингом PG, актерский состав которых включает актера, фамилия которого начинается с S

```

WITH actors_s AS (SELECT actor_id, first_name, last_name FROM actor WHERE last_name LIKE 'S%'),
actors_s_pg AS (SELECT s.actor_id, s.first_name, s.last_name, f.film_id, f.title FROM actors_s s
INNER JOIN film_actor fa ON s.actor_id = fa.actor_id
INNER JOIN film f ON f.film_id = fa.film_id
WHERE f.rating = 'PG'),
actors_s_pg_revenue AS (SELECT spg.first_name, spg.last_name, p.amount FROM actors_s_pg spg
INNER JOIN inventory i ON i.film_id = spg.film_id
INNER JOIN rental r ON i.inventory_id = r.inventory_id
INNER JOIN payment p ON r.rental_id = p.rental_id)
SELECT spg_rev.first_name, spg_rev.last_name, sum(spg_rev.amount) tot_revenue
FROM actors_s_pg_revenue spg_rev GROUP BY spg_rev.first_name, spg_rev.last_name ORDER BY 3 desc;

```

скалярные подзапросы можно использовать и в select

```

SELECT (SELECT c.first_name FROM customer c WHERE c.customer_id = p.customer_id) first_name,
[(SELECT c.last_name FROM customer c WHERE c.customer_id = p.customer_id) last_name,
[(SELECT ct.city FROM customer c INNER JOIN address a ON c.address_id = a.address_id
INNER JOIN city ct ON a.city_id = ct.city_id WHERE c.customer_id = p.customer_id) city,
[sum(p.amount) tot_payments, count(*) tot_rentals FROM payment p GROUP BY p.customer_id;

```

создать новую строку в таблице film_actor и у вас имеются следующие данные: имя и фамилия актера; название фильма

```

INSERT INTO film_actor (actor_id, film_id, last_update) VALUES (
[(SELECT actor_id FROM actor WHERE first_name = 'JENNIFER' AND last_name = 'DAVIS'),
[(SELECT film_id FROM film WHERE title = 'ACE GOLDFINGER'),

```

```
now());
```

Встроенные функции и условная логика

Встроенные функции

Строки:

экранирование: дополнительная кавычка " или \
quote() + кавычки при выборке текста:

```
SELECT quote(text_fld) FROM string_tbl; -> 'This string didn\'t work, but it does now'
```

char() берет номера, и объединяет в строку

```
char(97,98,99) -> abc
```

concat() (+ в MS SQL) объединяет строки

```
UPDATE string_tbl SET text_fld = CONCAT(text_fld, 'but now it is longer');  
select concat(first_name, ' ', last_name, ' has been customer since ', date(create_date)) narrative from customer;
```

length() длина строки

position начало подстроки SELECT POSITION('characters' IN vchar_fld) FROM string_tbl; Первый с 1. 0 если не найдено.

locate как position, но 3 аргумент - старт поиска

insert вставка

replace заменяет

substring находит подстроку

объединение данных из группировки в столбец

```
SELECT id, GROUP_CONCAT(data) FROM yourtable GROUP BY id
```

Числа:

ceil(), floor() округление в большую или меньшую сторону к ближайшему целому числу

round() десятичная часть $\geq 0,5$ округлено в большую сторону и наоборот ROUND(72.0909,3) - 3 знака оставить

TRUNCATE(72.0909, 1) - простое усечение, оставит 72.0

аргумент может быть < 0 TRUNCATE(17, -1) = 10, round(17, -1) = 20

sign() знак числа
abs() абсолютное значение

Даты:

cast() строку формата YYYY-MM-DD HH:MM:SS в дату

```
SELECT HOUR(@dt), MINUTE(@dt), SECOND(@dt), DAY(@dt), WEEK(@dt), MONTH(@dt), QUARTER(@dt),  
YEAR(@dt);
```

str_to_date(str, 'format') Формат:

%M	Полное имя месяца (January..December)
%m	Числовое значение месяца
%d	Числовое значение дня месяца (00..31)
%j	День года (001..366)
%W	Полное имя дня недели (Sunday.Saturday)
%Y	Значение года (четыре цифры)
%y	Значение года (две цифры)
%H	Час дня в 24-часовом формате (00..23)
%h	Час дня в 12-часовом формате (01..12)
%i	Минуты в часе (00..59)
%s	Число секунд (00..59)
%f	Число микросекунд (000000..999999)
%p	АМ или PM
%a	Краткое имя дня недели — Sun, Mon,...
%b	Краткое имя месяца — Jan,Feb,...

CURRENT_DATE(), CURRENT_TIME(), CURRENT_TIMESTAMP()

interval

```
SELECT DATE ADD(CURRENT DATE(), INTERVAL 5 DAY)
```

последний день месяца

```
LAST_DAY('2019-09-17')
```

имя месяца

```
DAYNAME('2019-09-18')
```

извлекают элемент даты

```
EXTRACT(YEAR FROM '2019-09-18 22:19:05')
```

кол-во полных дней

```
DATEDIFF('2019-09-03', 2019-06-21)
```

```
SELECT  
CURRENT_TIME() AS 'CUR_TIME',  
ADDTIME(CURRENT_TIME(), 020000) AS 'ADDTIME',  
SUBTIME(CURRENT_TIME(), 020000) AS 'SUBTIME';
```

```
☐ CUR_TIME | ADDTIME | SUBTIME |
```

```
☐ 10:12:34 | 12:12:34 | 08:12:34 |
```

Условная логика

case:

```
SELECT c.first_name, c.last_name,  
CASE  
WHEN active = 0 THEN 0  
ELSE (select count(*) from rental r where r.customer_id = c.customer_id)  
END activity_type  
FROM customer c;
```

преимущество перед if-then в возможности использовать внутри select, insert, update, delete возвращаемые значения одного типа

Использование case для разделения по столбцам

```
SELECT  
☐SUM(CASE WHEN monthname(rental_date) = 'May' THEN 1 ELSE 0 END) May_rentals,  
☐SUM(CASE WHEN monthname(rental_date) = 'June' THEN 1 ELSE 0 END) June_rentals,  
☐SUM(CASE WHEN monthname(rental_date) = 'July' THEN 1 ELSE 0 END) July_rentals  
FROM rental WHERE rental date BETWEEN 2005-05-01' AND '2005-08-01';
```

использование case для получения факта участия актера в фильмах статуса G

```

SELECT a.first_name, a.last_name,
CASE
WHEN EXISTS (SELECT 1 FROM film_actor fa
INNER JOIN film f ON fa.film_id = f.film_id
WHERE fa.actor_id = a.actor_id
AND f.rating = 'G') THEN 'Y' ELSE 'N' END g_actor
FROM actor a WHERE a.last_name LIKE 'S%' OR a.first_name LIKE 'S%';

```

условные обновления

```

UPDATE customer
SET active = CASE
WHEN 90 <= (SELECT datediff(now(), max(rental_date) FROM rental r WHERE r.customer_id =
customer.customer_id))
THEN 0
ELSE 1
END
WHERE active = 1;

```

Аналитические функции

Постобработка сформированных данных. Использование: функция(столбец) over (дополнительное условие группировки)

Функция	Описание
RANK	Аналогична функции DENSE_RANK() за исключением того, что в последовательности ранжированных значений есть пробелы, когда две или более строки имеют одинаковый ранг.
DENSE_RANK	Присваивает ранг каждой строке в своем разделе на основе предложения ORDER BY. Он присваивает одинаковый ранг строкам с одинаковыми значениями. Если две или более строки имеют одинаковый ранг, то в последовательности ранжированных значений не будет пробелов.
ROW_NUMBER	Назначает последовательное целое число каждой строке в своем разделе
CUME_DIST	Вычисляет совокупное распределение значения в наборе значений.
FIRST_VALUE	Возвращает значение указанного выражения относительно первой строки в рамке окна.

Функция	Описание
LAG	Возвращает значение N-й строки перед текущей строкой в разделе. Возвращает NULL, если предшествующей строки не существует.
LAST_VALUE	Возвращает значение указанного выражения относительно последней строки в рамке окна.
LEAD	Возвращает значение N-й строки после текущей строки в разделе. Возвращает NULL, если никакой последующей строки не существует.
NTH_VALUE	Возвращает значение аргумента из N-й строки рамки окна
NTILE	Распределяет строки для каждого раздела окна в указанное количество ранжированных групп.
PERCENT_RANK	Вычисляет процентильный ранг строки в разделе или наборе результатов

Транзакции, индексы и ограничения

Транзакции

Блокировки. Записывающие должны запрашивать и получать блокировку записи для изменения данных, а извлекающие должны запрашивать и получать блокировку чтения. Варианты блокировки:

- запросы на чтение блокируются пока блокировка записи не снята, за один раз для каждой таблицы (или ее части) только одна блокировка записи
- блокировка записи, не нужны блокировки чтения. Но сервер гарантирует, что каждый читатель видит согласованное представление данных до завершения запроса (versioning).

Уровни блокировок

- Блокировка таблиц Не позволяет нескольким пользователям одновременно изменять данные в одной таблице.
- Блокировка страниц Не позволяет нескольким пользователям изменять данные в одной и той же странице (страница — это сегмент памяти, обычно в диапазоне от 2 до 16 Кбайт) таблицы одновременно.
- Блокировка строк Не позволяет нескольким пользователям одновременно изменять одну и ту же строку в таблице.

Есть режим автофиксации и транзакции.

- По умолчанию одиночная инструкция INSERT, UPDATE или DELETE неявно включается в транзакцию и немедленно подтверждается.
- SET AUTOCOMMIT=0 отключает для сессии
- только для транзакционных таблиц
- Некоторые команды заставляют подтвердить транзакцию до их выполнения. Команды (Data Definition Language, DDL) типа ALTER TABLE, LOCK TABLES, ...

start transaction начало транзакции, commit - завершение, rollback - возврат.

Механизмы хранения mysql:

MyISAM	Нетранзакционный, табличная блокировка, скоростной доступ на чтение с малым кол-вом записи
--------	--

MEMORY	Нетранзакционный, табличная блокировка, для скоростного кэша, таблицы в памяти, при нехватке своп на диск, не поддерживает TEXT BLOB
CSV	Транзакционный, данные в файлах с данными с разделением запятыми, обмен данными. Нет индексов.
InnoDB	Транзакционный, блокировка на уровне строки
Merge	Специальный механизм, используемый для создания нескольких идентичных таблиц MyISAM в виде единой таблицы (так называемое разбиение таблицы)
Archive	хранение больших количеств неиндексированных данных, в основном для архивных целей. Не поддерживает DELETE, UPDATE.
Blackhole	для репликации
federated	Одна база на нескольких серверах. Создает клиентское соединение и выполняет еще раз запрос, получает данные. По-умолчанию выключен.
NDB cluster	

статус таблицы

```
show table status like 'customer' \G;
```

изменить механизм хранения определенной таблицы

```
alter table customer engine = InnoDB;
```

Нельзя сочетать разные механизмы в одной транзакции. управление конкурентным доступом с помощью многоверсионности (multiversion concurrency control, MVCC)

Точки сохранения: Возможность частично откатывать транзакцию.

```
SAVEPOINT my_savepoint;
ROLLBACK TO SAVEPOINT my_savepoint;

START TRANSACTION;
UPDATE product
SET date_retired = CURRENT_TIMESTAMP()
WHERE product_cd = 'XYZ';
SAVEPOINT before close accounts;
UPDATE account
```

```
SET status = 'CLOSED', close_date = CURRENT_TIMESTAMP(),
last_activity_date = CURRENT_TIMESTAMP{}
WHERE product_cd = 'XYZ';
ROLLBACK TO SAVEPOINT before_close_accounts;
COMMIT;
```

Индексы

Индексы независимые объекты, хотя и относятся к базе.

Название	Назначение и создание
Простой индекс	ускоряет поиск данных <pre>ALTER TABLE customer ADD INDEX idx_email (email);</pre>
Уникальный индекс	запрещает два элемента с одинаковым значением <pre>ALTER TABLE customer ADD UNIQUE idx_email (email);</pre>
Многостолбцовый индекс	важна последовательность - эффекта не будет при поиске только по последнему столбцу <pre>ALTER TABLE customer ADD INDEX idx_full_name (last_name, first_name);</pre>

Менее 64 индексов на таблицу, 16 столбцов максимум для мультииндекса

Просмотр индексов:

```
SHOW INDEX FROM customer \G;
```

Удаление индекса:

```
ALTER TABLE customer DROP INDEX idx_email;
```

Типы индексов:

- B-Tree Индекс на основе деления отрезка пополам. Хорош для разнородных почти сбалансированных данных

- Битовые маски - когда значений относительно мало. Генерит несколько таблиц, соотв. каждому значению ключа.

```
CREATE BITMAP INDEX idx_active ON customer (active);
```

- Текстовые индексы

Просмотр плана выполнения запроса:

```
EXPLAIN
SELECT customer_id, first_name, last_name
FROM customer
WHERE first_name LIKE 'S%' AND last name LIKE 'P%' \G;
```

Ограничение

Ограничение — условия, накладываемые столбцы таблицы. Типы ограничений:

- Ограничения первичного ключа Определяют столбец или столбцы, для которых гарантируется их уникальность в таблице
- Ограничения внешнего ключа содержать только значения, найденные в столбце первичного ключа другой таблицы (могут также ограничиваться допустимые значения в других таблицах при установке правил update cascade и/или delete cascade)
- Ограничения уникальности Ограничивают один или несколько столбцов таким образом, чтобы они содержали уникальные значения в таблице (ограничение первичного ключа — частный случай ограничения уникальности)
- Проверочные ограничения Ограничивают допустимые значения для столбца

Создание ограничений

при создании таблицы

```
CREATE TABLE customer
customer_id SMALLINT UNSIGNED NOT NULL AUTO_INCREMENT, ... store_id TINYINT UNSIGNED NOT NULL,
address_id SMALLINT UNSIGNED NOT NULL,
PRIMARY KEY (customer_id), CONSTRAINT fk_customer_address FOREIGN KEY (address_id) REFERENCES address
(address_id) ON DELETE RESTRICT ON UPDATE CASCADE,
CONSTRAINT fk_customer_store FOREIGN KEY (store_id) REFERENCES store (store_id) ON DELETE RESTRICT ON
UPDATE CASCADE
```

при изменении таблицы

```
ALTER TABLE customer ADD CONSTRAINT fk_customer_address FOREIGN KEY (address_id) REFERENCES address  
(address id) ON DELETE RESTRICT ON UPDATE CASCADE;
```

удаление ограничений

```
ALTER TABLE customer drop
```

Представления и метаданные

Представления

Создание представления

```
CREATE VIEW customer_vw (customer_id, first_name, last_name, email) AS SELECT ...
```

```
CREATE VIEW film_stats AS
SELECT f.film_id, f.title, f.description, f.rating, (SELECT c.name FROM category c INNER JOIN film_category fc ON
c.category_id = fc.category_id WHERE fc.film_id = f.film_id) category_name,
(SELECT count(*) FROM film_actor fa WHERE fa.film_id = f.film_id) num_actors, (SELECT count(*) FROM inventory
i WHERE i.film_id = f.film_id) inventory_cnt,
(SELECT count(*) FROM inventory i INNER JOIN rental r ON i.inventory_id = r.inventory_id WHERE i.film_id =
f.film_id) num_rentals
FROM film f;
```

Используются для:

- Безопасность данных
- Агрегация данных
- Соккрытие сложности
- Соединение разделенных данных (одна таблица -> две текущая и архив, но редко нужно как одна) объединяются через union all

Представление обновляемое, если:

- не используются агрегатные функции
- нет group by и having.
- В предложениях select и from нет подзапросов, а любой подзапрос в предложении where не ссылается на таблицы в предложении from.
- нет union, union all или distinct.
- Предложение from содержит как минимум одну таблицу или обновляемое представление.
- Если имеется более одной таблицы или представления, предложение from использует только внутренние соединения.

В представлении с одной таблицей (простое представление) не может быть использовано для вставки данных.

В представлении с внутренними join: CREATE VIEW customer details AS SELECT c.customer_id,

- можно обновлять столбцы одной из таблиц
- можно добавлять данные

Метаданные

information_schema - данные о данных

.schemata	базы данных
.tables	информация обо всех таблицах
.views	о представлениях
.columns	о столбцах
.statistics	об индексах
.table_constraints	об ограничениях
.user_privilege	Кто имеет привилегии для различных объектов схемы
.schema_privileges	Кто имеет привилегии для различных баз данных
.table_privileges	Кто имеет привилегии для различных таблиц
.column_privileges	Кто имеет привилегии для различных столбцов таблиц
.routines	Сохраненные подпрограммы (процедуры и функции)
.triggers	Триггеры таблиц
.plugins	Подключаемые модули сервера
.engines	Доступные механизмы хранения
.partitions	Разбиения таблиц
.events	Запланированные события
.processlist	Выполняющиеся процессы
.referential_constraints	Внешние ключи

.parameters	Параметры хранимых процедур и функций
.profiling	Информация о профилях пользователей

```
SET @qry = 'SELECT customer_id, first_name, last_name FROM customer WHERE customer_id = ?';
PREPARE dynsql2 FROM @qry;
SET @custid = 9;
EXECUTE dynsql2 USING @custid;
DEALLOCATE PREPARE dynsql2;
```

Администрирование

MySQL

Базы данных хранятся в /var/lib/mysql

```
sudo grep -R 'datadir' /etc/mysql/
```

Настройка сессии

Уровень изолированности	Черновое чтение	Неповторяющееся чтение	Фантомное чтение	Блокировка чтения
READ UNCOMMITTED	Да	Да	Да	Нет
READ COMMITTED	Нет	Да	Да	Нет
REPEATABLE READ	Нет	Нет	Да	Нет
SERIALIZABLE	Нет	Нет	Нет	Да

Можно установить на уровне сессии

```
SET SESSION TRANSACTION ISOLATION LEVEL READ COMMITTED;
```

Взаимоблокировки: Innodb отказывает самой короткой транзакции цели в трех ключевых областях: задержка, доступность и ошибки

Performance Schema

Это отдельный engine Проверить факт наличия:

```
SHOW ENGINES;
+-----+-----+-----+-----+-----+
| Engine      | Support | Comment                               | Transactions | XA  | Savepoints |
+-----+-----+-----+-----+-----+
| PERFORMANCE_SCHEMA | YES    | Performance Schema                   | NO          | NO  | NO        |
```

включен или нет:

```
SHOW VARIABLES LIKE 'performance_schema';
```

Переменные окружения

Просмотр

```
SHOW VARIABLES LIKE 'performance_schema';
```

Поддерживает %

Настройки

Настройка при запуске

```
shell> mysql --max_allowed_packet=16M
```

Настройка при работе

```
mysql>SET GLOBAL max_connections = 1000; - потеря при перезагрузке  
mysql>SET PERSIST max_connections = 1000; - сохранение после перезагрузки
```

Инструменты (setup_instruments) пишут в потребителя (таблицы данных,).
Схемой sys представления и хранимые подпрограммы над performance_schema.
включение инструментов вызывает дополнительный код

```
statement/sql/select;  
wait/synch/mutex/innodb/autoinc_mutex
```

Крайний левый - тип, далее слева направо подсистемы от общей к частной.
Дайджест — агрегирование запросов путем удаления из них вариаций
Структура таблиц потребителей:

- *_current — события, происходящие на сервере в данный момент;
- *_history — последние 10 завершенных событий на поток;
- *_history_long — последние 10 000 завершенных событий на поток по всему миру

Ограничения:

- Инструментарий должен поддерживаться компонентом MySQL.
- Она собирает данные только после включения конкретного инструмента и потребителя.
- Трудно освободить память. Даже если позже отключите определенные инструменты или потребители, память не будет освобождена, пока вы не перезапустите сервер.

Настройка инструментов мониторинга

Список инструментов: https://mariadb.com/kb/en/performance-schema-setup_instruments-table/

Таблица setup_instruments - включенные инструменты.

Структура:

NAME	Название инструмента
ENABLED	Включен или нет
TIMED	Планировщик. Если disabled, работать не будет

CRUD db, tables, users

```
ALTER USER 'root'@'localhost' IDENTIFIED BY 'newPassword';

CREATE DATABASE IF NOT EXISTS my_timeweb;
DROP DATABASE IF EXISTS my_timeweb;
CREATE USER 'sub_user'@'10.0.%' IDENTIFIED BY 'password';
SELECT * FROM mysql.user;
DROP USER 'user'@'localhost';
```

Привилегии:

ALTER	Менять структуру таблицы или БД
CREATE	Создавать новые БД и таблицы
DELETE	Удалять строки в таблице
INSERT	Добавлять строки в таблицу
SELECT	Читать данные из таблицы
UPDATE	Обновлять данные в таблице
DROP	Удалять БД
ALL PRIVILEGES	все, кроме GRANT;
USAGE PRIVILEGES	никаких привилегий;
FILE	разрешает читать файлы на сервере;
INDEX	создавать индексы для таблиц;
DROP	удалять таблицы;

EVENT	обработка событий;
TRIGGER	создание триггеров.

Привилегия на действия

GRANT	изменять права пользователей;
SUPER	суперпользователь;
PROCESS	получение информации о состоянии MySQL;
RELOAD	позволяет перезагружать таблицы привилегий;
SHUTDOWN	позволяет отключать или перезапускать базу данных;
SHOW DATABASES	просмотр списка баз данных;
LOCK TABLES	блокирование таблиц при использовании SELECT;
REFERENCES	создание внешних ключей для связывания таблиц;
CREATE USER	создание пользователей;

```
GRANT SELECT, INSERT ON my_timeweb.* TO 'user'@'localhost'; из-под root
GRANT ALL PRIVILEGES ON my_timeweb.* TO 'user'@'localhost';
REVOKE SELECT, INSERT ON my_timeweb.* FROM 'user'@'localhost';
```

применение изменений

```
flush privileges;
```

статус, приложения

mysqladmin version

mysqlshow mysql - список таблиц БД mysql

mysql_secure_installation

mysqladmin - административное

mysqlcheck - проверка

mysqldump - сохранение

mysqlimport - импорт текстовых файлов в нужную таблицу

mysqlpump - экспорт базы в sql файл

mysqlslap - статус загрузки сервера

--print-defaults выводит конфиг по умолчанию

Postgresql

Подключение:

```
psql  
[-U имя пользователя  
[-W пароль  
[-d имя базы
```

Сброс пароля пользователя: в файле pg_dba.conf

список пользователей

```
\du  
SELECT username, usesuper, usecreatedb FROM pg_catalog.pg_user;
```

создать пользователя

```
CREATE USER user_name WITH PASSWORD 'myPassword';
```

сменить пароль пользователя

```
ALTER USER user_name WITH PASSWORD 'new_password';
```

список баз данных

```
\l  
SELECT datname FROM pg_database;
```

создать базу данных

```
CREATE DATABASE имя_БД;  
drop database имя_БД
```

установить права пользователя на базу

```
GRANT ALL PRIVILEGES ON DATABASE database1 to dmosk;
```

список таблиц

\dt

Для теста

Пар