|  |  |
| --- | --- |
| **INT Целые числа** | |
| **TINYINT** | 1 байт (8 бит) - 2\*\*8 символов 0-256 UNSIGNED или (-127) - (127) |
| **SMALLINT** | 2 байта 32 676 UNSIGNED 65 535 |
| **MEDIUMINT** | 3 байта 8388607 UNSIGNED 16777215 |
| **INT** | 4 байта 2 x 109 UNSIGNED 4 x 109 |
| **BIGINT** | 8 байт 9 x 1018 UNSIGNED  18 x 1018 |
| **SERIAL** | псевдотип: BIGINT UNSIGNED NOT NULL AUTO\_INCREMENT UNIQUE |
|  |  |
| **FLOAT Вещественные числа** | |
| **FLOAT** | 4 байта |
| **DOUBLE** | 8 байт |
| **DECIMAL(7,4)** | 7 байт под цифры из них 4 байта после запятой - 123.1234 (число хранится в виде строки, обрабатывается сильно медленнее, чем остальные числа, зато не теряется точность. Требуемая точность задается при объявлении столбца данных одного из этих типов) |
|  |  |
| **CHAR Строковый фиксированный тип** | |
| **CHAR** | 40 символов (Если значения похожи по длине - CHAR, иначе — VARCHAR) |
|  |  |
| **VARCHAR(255) Строковый тип переменной длины (максимально 65 536 байт, 21844 для utf-8)**  максимальный объем строки в круглых скобках после запятой | |
| **TINYTEXT** | 1 байт |
| **TEXT** | 2 байта |
| **MEDIUMTEXT** | 3 байта |
| **LONGTEXT** | 4 байта |
|  |  |
| **TEXT хранение объемного текста** | |
|  |  |
| **BLOB хранение бинарных данных** | |
|  |  |
| **NULL — неизвестное значение** | |
| **NOT NULL запрет вставлять в поле NULL значения** | |
|  |  |
| **Календарные типы** | |
| **TIME** | время (3 байта) '00:00:00' hh:mm:ss |
| **YEAR** | год (1 байт) 0000 yyyy-mm-dd hh:mm:ss |
| **DATE** | дата (3 байта) '0000-00-00' yyyy-mm-dd |
| **DATETIME** | дата и время (8) '0000-00-00 00:00:00' |
| **TIMESTAMP** | дата и время (4) '0000-00-00 00:00:00' занимает в 2 раза меньше места, чем  DATETIME, но хранит только интервал от 1970 года до 2038  первый столбец в таблице обновляется автоматически при  создании и обновлении, хранит дату в UTC-формате. |
|  |  |
| **Коллекционные типы** - **допустимые значения задаются списком строк** | |
| **ENUM** | одно значение из списка |
| **SET** | комбинация значений из списка |
|  |  |
| **JSON объект JavaScript, используется для хранения и передачи коллекций** | |
|  | |
| **mysql -u root -p** | Запуск mysql |
| **exit** | Выход из mysql (\q) |
| **mysql -u root shop** | Запуск mysql с назначением базы данных по умолчанию |
|  | Удаление mysql Ubuntu  sudo service mysql stop  sudo apt-get remove --purge mysql-server mysql-client mysql-common  sudo apt-get autoremove  sudo apt-get autoclean |
|  | Установка mysql Ubuntu  wget -c https://dev.mysql.com/get/mysql-apt-config\_0.8.10-1\_all.deb  sudo apt-key adv --keyserver keys.gnupg.net --recv-keys 5072E1F5  sudo dpkg -i mysql-apt-config\_0.8.10-1\_all.deb  sudo apt update  sudo apt-get install mysql-server |
|  | mysql -u root -h 192.168.0.10 -P 3306  (ip & port) удаленное подключение |
| **mysql -V**  **mysqld -V** | Вывод версии mysql в линукс  mysql Ver 8.0.19 for Linux on x86\_64 (MySQL Community Server - GPL)  /usr/sbin/mysqld Ver 8.0.19 for Linux on x86\_64 (MySQL Community Server - GPL) |
| **.my.cnf** | Конфигурационный файл, должен лежать в дом директории (вход без пароля)  [mysql] user=root password=123 |
| **\G** | Вывод результата в вертикальном формате (пишется в конце) |
| **STATUS** | Вывод информации о состоянии сервера (\s) |
| **--** | однострочный комментарий -- сведения о покупателях |
| **/\* … \*/** | Многострочный комментарий /\* комментарий \*/ |
| **COMMENT** | Комментарий для данной таблицы длиной 60 символов.  CREATE TABLE users (id INT, name VARCHAR(255) COMMENT 'Имя покупателя',   birthday\_at DATE COMMENT 'Дата рождения') COMMENT = 'Покупатели'; |
| **SOURCE** | Выполнение SQL-команды из файла SOURCE file.sql |
|  |  |
| **MYSQLDUMP** | Создание резервной копии (из shell из-под root (su) cd /var/lib/mysql)  # mysqldump -u root -p example > example\_1.sql;  mysql> CREATE DATABASE simple;  # mysql -u root -p simple < example\_1.sql;  # mysqldump -u root -p --where="true limit 100" mysql help\_keyword > example\_2.sql;  mysql> CREATE DATABASE sim;  # mysql -u root -p sim < example\_2.sql |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
| **metanit.com/sql/** | Описание mysql на русском языке |
| **www.mysql.ru** | Документация по mysql на русском языке |
| **USE** | Выбор базы данных по умолчанию  USE shop; |
| **SHOW** | Предоставляет информацию по БД, таблицам, столбцам, о состоянии сервера  SHOW DATABASES; # список всех БД  SHOW DATABASES LIKE 'v%'; # список БД начинающихся с v  SHOW TABLES FROM vk; # список таблиц БД vk  SHOW COLUMNS FROM users FROM vk; # список столбцов в таблице  SHOW COLUMNS FROM vk.users; # список столбцов в таблице  SHOW FULL COLUMNS FROM vk.users; # список… + коммент. и привелегии  SHOW COLUMNS FROM vk.users LIKE 'ph%'; # + регулярное выражение  SHOW CREATE TABLE users; # показывает код создающий табл  SHOW TABLE STATUS FROM vk; # инфо о таблицах БД  SHOW VARIABLES LIKE 'datadir'; # путь к каталогу с базами данных  SHOW FULL PROCESSLIST; # выводит запущенные потоки  SHOW INDEX FROM users FROM vk; # инфо по индексу в спецформате |
| **DESCRIBE**  **(DESC)** | Просмотр структуры таблицы, получение инфо по столбцам  Каждая строка в результирующей таблице соответствует отдельному столбцу.  DESCRIBE users; # БД выбрана по USE  DESCRIBE users FROM vk; # БД не по умолчанию  DESCRIBE users 'us%'; # структура столбцов начинающихся на 'us' |
| **SELECT** | Вывод данных на экран (записи из таблицы). Примеры для БД по умолчанию.  SELECT список\_столбцов FROM имя\_таблицы  SELECT 'Hello!';  SELECT 44+56 AS summa; # посчитать и вывести под псевдонимом summa  SELECT \* FROM users LIMIT 10; # вывести 10 записей и все столбцы из таблицы  SELECT id, phone FROM users; # вывести все записи и id и phone столбцы  SELECT \* FROM vk.users; # все записи и все столбцы в табл users БД vk  SELECT \* FROM users LIMIT 1\G; # 1 запись и все ст в табл users вертикально  SELECT phone, metadata FROM users, media LIMIT 1; # записи из разных таблиц  SELECT id, name, phone FROM users ORDER BY name, phone LIMIT 10;  # вывести отсортиров. записи по name и phone в порядке возраст.  SELECT id, name, phone FROM users ORDER BY id DESC LIMIT 10; # по убыванию  SELECT id, name, phone FROM users ORDER BY 2 LIMIT 10;  # вывести отсортиров. записи по 2 столбцу (phone).  SELECT id, name AS имя, phone FROM users ORDER BY имя LIMIT 10;  # вывести отсортиров. записи по 2 столбцу под псевдонимом имя.  SELECT id, name, phone FROM users DISTINT WHERE id > 400 LIMIT 10;  # вывести записи где id > 400 без дублирующих строк  SELECT [STRAIGHT\_JOIN]  [SQL\_SMALL\_RESULT] [SQL\_BIG\_RESULT] [SQL\_BUFFER\_RESULT]  [SQL\_CACHE | SQL\_NO\_CACHE] [SQL\_CALC\_FOUND\_ROWS] [HIGH\_PRIORITY]  [DISTINCT | DISTINCTROW | ALL]  select\_expression,...  [INTO {OUTFILE | DUMPFILE} 'file\_name' export\_options]  [FROM table\_references  [WHERE where\_definition]  [GROUP BY {unsigned\_integer | col\_name | formula} [ASC | DESC], ...]  [HAVING where\_definition]  [ORDER BY {unsigned\_integer | col\_name | formula} [ASC | DESC], ...]  [LIMIT [offset,] rows]  [PROCEDURE procedure\_name]  [FOR UPDATE | LOCK IN SHARE MODE]] |
| **CREATE DATABASE** | Создание базы данных.  CREATE DATABASE shop; # создать БД shop  CREATE DATABASE IF NOT EXISTS shop; # создать БД shopб если её не существ |
| **CREATE TABLE** | Создание таблицы внутри базы данных.  CREATE TABLE имя таблицы (имя столбца параметры столбца, …)  CREATE TABLE IF NOT EXISTS users (id INT(8), ip INT);  CREATE TABLE users (  id INT UNSIGNED NOT NULL PRIMARY KEY AUTO\_INCREMENT,  name VARCHAR(255),  created\_at DATETIME  ); |
| **CREATE INDEX** | Создание индекса в уже существующем столбце  CREATE INDEX index\_of\_catalog\_id ON products (catalog\_id); |
| **RENAME TABLE** | Переименование таблицы  RENAME TABLE customers TO clients; |
| **ALTER TABLE** | Изменение структуры существующей таблицы или её имени.  ALTER TABLE tbl\_name CHANGE id user\_id INT # изменить название столбца  с id на user\_id, обязат указ. параметры столбца  ALTER TABLE tbl\_name CHANGE id id INT UNSIGNED NOT NULL;  # изменить параметры столбца без изменения имени  ALTER TABLE tbl\_name MODIFY id INT UNSIGNED NOT NULL;  # изменить параметры столбца без изменения имени  ALTER TABLE t2 MODIFY id\_1 INT NOT NULL, CHANGE id\_2 id\_3 CHAR(20);  # несколько изменений в 1 команде  ALTER TABLE t1 RENAME t2; # переименовать таблицу из t1 в t2  ALTER TABLE t2 ADD d TIMESTAMP; # добавить новый столбец d  ALTER TABLE t2 ADD INDEX (d), ADD PRIMARY KEY (a);  # добавить индекс к столбцу d и сделать столбец a первичным ключом  ALTER TABLE t2 ADD id INT UNSIGNED NOT NULL AUTO\_INCREMENT, ADD INDEX (id); # добавить новый числовой столбец AUTO\_INCREMENT с именем id  ALTER TABLE t2 DROP COLUMN id\_3; # удалить столбец id\_3  Первый номер последовательности можно установить путем выполнения команды SET INSERT\_ID=# перед ALTER TABLE или использования табличной опции AUTO\_INCREMENT = #.  Существует аргумент DROP PRIMARY KEY |
| **INSERT**  **INSERT INTO** | Вставка новых строк в существующую таблицу  INSERT INTO tbl1 (col1,col2) VALUES(15,col1\*2); # вставить в таблицу tbl1  в колонки col2 и col2 значение 15 и col1\*2  INSERT INTO tbl1 (col1,col2) SELECT …  INSERT INTO tbl1 SET col\_name=expression, col\_name=expression, ...  INSERT INTO meetings (name) SELECT first\_name FROM users limit 20;  INSERT INTO posts (user\_id, header, body) SELECT id, last\_name, first\_name FROM users;  однострочная вставка:  INSERT INTO catalogs VALUES (NULL, 'Процессоры');  многострочная вставка:  INSERT INTO catalogs VALUES (DEFAULT, 'Процессоры'), (DEFAULT, 'Мат.платы'), (DEFAULT, 'Видеокарты');  вставка записи из одной таблицы в другую, в т. ч. и преобразовывая данные.  INSERT INTO cat SELECT \* FROM catalogs; |
| **UPDATE** | Обновляет столбцы в соответствии с их новыми значениями в строках существующей таблицы. В SET указывается, какие именно столбцы модифицируются и какие величины в них установливаются. В WHERE задается, какие строки подлежат обновлению (иначе обновляются все строки). Если задано ORDER BY - строки обновляются в указанном порядке.  UPDATE tbl1 SET name = 'Процессоры (Intel)' WHERE name = 'Процессоры';  UPDATE communities SET id=20 where id=146;  UPDATE communities\_users SET community\_id = FLOOR(1 + (RAND() \* 20)), user\_id = FLOOR(1 + (RAND() \* 100));  UPDATE communities\_users SET community\_id = FLOOR(1 + (RAND() \* 20)) WHERE (community\_id > 75) and (community\_id <101);  UPDATE products SET price = price + 3000; # увелич. у всех товаров цену на 3000  UPDATE products SET productcount= DEFAULT WHERE manufacturer = 'huawei'; |
| **DROP DATABASE** | Удаление базы данных  DROP DATABASE shop;  DROP DATABASE IF EXISTS shop; |
| **DROP TABLE** | Удаление одной или нескольких таблиц.  DROP TABLE tbl1, tbl2;  DROP TABLE IF EXISTS tbl1, tbl2; |
| **DROP INDEX** | Удаление индекса  DROP INDEX index\_name ON tbl\_name; |
| **DELETE** | удаляет строки из таблицы, не обнуляет счетчик нумерации, если использован режим AUTOCOMMIT работает аналогично TRUNCATE  DELETE FROM catalogs; # удаляет все строки  DELETE FROM catalogs LIMIT 2; # удаляет только 2 записи  DELETE FROM catalogs WHERE id > 1 LIMIT 1; # удаляет 1 запись с id > 1  DELETE t1,t2 FROM t1,t2,t3 WHERE t1.id=t2.id AND t2.id=t3.id;  # удалены совпадающие строки из таблиц t1 и t2.  DELETE FROM somelog WHERE user = 'jcole' ORDER BY timestamp LIMIT 1;  # удалит самую старую запись (по timestamp) |
| **TRUNCATE** | Удаляет строки из таблицы и обнуляет счетчики AUTO\_INCREMENT. Работает быстрее DELETE.  TRUNCATE tbl1;  TRUNCATE TABLE tbl1; |
| **SET** | Устанавливает различные опции, влияющие на работу сервера или клиента.  так же используется внутри других команд как 'присвоить' |
| **AUTO\_INCREMENT** | Автоматическое создание уникального индекса только для целочисленных столбцов. Передача столбцу, снабженному этим атрибутом, значения **NULL** или 0, приводит к автоматическому присвоению ему максимального значения столбца, плюс 1.  CREATE TABLE catalogs (id INT UNSIGNED NOT NULL PRIMARY KEY AUTO\_INCREMENT, name VARCHAR(255));  выводит последний вставленный ID:  SELECT LAST\_INSERT\_ID();  Установка нового значения для продолжения нумерации:  ALTER TABLE tablename AUTO\_INCREMENT = тут\_пишем\_нужное\_число; |
| **PRIMARY KEY** | Первичный ключ, допустимо объявление индекса сразу по двум или более столбцам.  CREATE TABLE catalogs (id INT UNSIGNED NOT NULL PRIMARY KEY, name VARCHAR(255))  CREATE TABLE catalogs (id INT UNSIGNED NOT NULL, name VARCHAR(255), PRIMARY KEY(id));  CREATE TABLE catalogs (id INT UNSIGNED NOT NULL, name VARCHAR(255), PRIMARY KEY(id, name(10))); - п. к. по ст. **id** и по первым 10 символам ст. name.  CREATE TABLE catalogs (id SERIAL PRIMARY KEY, name VARCHAR(255));  ALTER TABLE products ADD PRIMARY KEY (id); # добавление первичного ключа  в существующую таблицу)  Удаление первичного ключа:  ALTER TABLE Products DROP PRIMARY KEY; |
| **FOREIGN KEY** | Внешний ключ позволяющий установить связь между таблицами. Внешний ключ устанавливается для столбцов из зависимой таблицы и указывает на один из столбцов из главной таблицы.  [CONSTRAINT имя\_ограничения]  FOREIGN KEY (столбец1, столбец2, ...)  REFERENCES главная\_таблица (столбец\_главной\_таблицы1, ст\_гл\_т2, ...)  [ON DELETE действие]  [ON UPDATE действие]  **CASCADE**: автоматически удаляет или изменяет строки из зависимой таблицы при удалении или изменении связанных строк в главной таблице.  **SET NULL**: при удалении или обновлении связанной строки из главной таблицы устанавливает для столбца внешнего ключа значение **NULL**. (В этом случае столбец внешнего ключа должен поддерживать установку NULL)  **RESTRICT**: отклоняет удаление или изменение строк в главной таблице при наличии связанных строк в зависимой таблице.  **NO ACTION**: то же самое, что и RESTRICT.  **SET DEFAULT**: при удалении связанной строки из главной таблицы устанавливает для столбца внешнего ключа значение по умолчанию, которое задается с помощью атрибуты DEFAULT. (движок InnoDB не поддерживает).  ALTER TABLE profiles  ADD CONSTRAINT profiles\_user\_id\_fk  FOREIGN KEY (user\_id) REFERENCES users(id)  ON DELETE CASCADE,  ADD CONSTRAINT profiles\_photo\_id\_fk  FOREIGN KEY (photo\_id) REFERENCES media(id)  ON DELETE SET NULL;  CREATE TABLE Orders  (id INT PRIMARY KEY AUTO\_INCREMENT,       сustomerId INT,       сreatedAt Date,       CONSTRAINT orders\_custonmers\_fk FOREIGN KEY (CustomerId)  REFERENCES  сustomers (id));  Удаление внешнего ключа:  ALTER TABLE Orders DROP FOREIGN KEY orders\_customers\_fk; |
| **INDEX**  **KEY** | Создание индекса на столбцы с целью оптимизации запросов  (индекс, уникальный индекс (UNIQUE KEY или UNIQUE), составной индекс)  CREATE TABLE drug (  id INT UNSIGNED NOT NULL PRIMARY KEY AUTO\_INCREMENT UNIQUE,  tradename VARCHAR(255) NOT NULL,  active\_substance VARCHAR(255),  drug\_view ENUM ('origin', 'generic', 'combi'),  instruction TEXT,  UNIQUE KEY drug\_tradename\_uk (tradename),  KEY drug\_active\_substance\_idx (active\_substance),  KEY drug\_tradename\_active\_substance\_idx (tradename, active\_substance));  создание индекса в уже существующем столбце  CREATE INDEX index\_of\_catalog\_id ON products (catalog\_id); |
| **DEFAULT** | Значение по умолчанию  CREATE TABLE customers (age INT DEFAULT 18); |
| **CHECK** | Задает ограничение для диапазона значений, хранящихся в столбце.  CREATE TABLE customers (  age INT DEFAULT 18 CHECK(age >0 AND age < 100),  email VARCHAR(30) CHECK(email !=''));  CREATE TABLE customers (  age INT DEFAULT 18,  email VARCHAR(30),  phone VARCHAR(20),  CHECK((age >0 AND age<100) AND (email !='') AND (phone !=''))); |
| **LIMIT** | Лимит  LIMIT 10 # первые 10 строк  LIMIT 5, 10 # с 6 по 15 строки (10 строк начиная после 5) |
| **CONSTRAINT** | Оператор, устанавливающий имя для ограничений, через которые можно управлять ограничениями - удалять или изменять их. Установить имя можно для ограничений PRIMARY KEY, CHECK, UNIQUE, FOREIGN KEY.  CREATE TABLE customers (  id INT AUTO\_INCREMENT,  age INT,  firstname VARCHAR(20) NOT NULL,  lastname VARCHAR(20) NOT NULL,  email VARCHAR(30),  phone VARCHAR(20) NOT NULL,  CONSTRAINT customers\_pk PRIMARY KEY(Id),  CONSTRAINT customer\_phone\_uq UNIQUE(Phone),  CONSTRAINT customer\_age\_chk CHECK(Age >0 AND Age<100)); |
| **DELIMITER** | Назначает признак окончания запроса  DELIMITER //  SELECT 2\*2 // |
| **ZEROFILL** | Дополнить 0 недостающие символы (000123) |
| **UNSIGNED** | Задает беззнаковые числа (не отрицательные значения) |
| **UNIQUE** | Уникальное значение |
| **IGNORE** | Игнорировать ошибку |
| **FROM** | Из |
| **IF NOT EXISTS** | Если не существует |
| **IF EXISTS** | Если существует |
| **EXISTS** | Оператор EXISTS проверяет, возвращает ли подзапрос какое-либо значение, используется для индикации того, что как минимум одна строка в таблице удовлетворяет некоторому условию. Поскольку возвращения набора строк не происходит, то подзапросы с подобным оператором выполняются довольно быстро (быстрее, чем IN).  SELECT \* FROM Products WHERE EXISTS (SELECT \* FROM Orders WHERE Orders.ProductId = Products.Id);  # найдем все товары из табл. Products, на которые есть заказы в табл. Orders  SELECT \* FROM Products WHERE NOT EXISTS (SELECT \* FROM Orders WHERE Products.Id = Orders.ProductId); |
| **WHERE** | Где  = сравнение на равенство  != сравнение на равенство  <> сравнение на неравенство  < меньше чем  > больше чем  <= меньше чем или равно  >= больше чем или равно  SELECT \* FROM products WHERE manufacturer = 'samsung' AND price > 50000;  SELECT \* FROM products WHERE NOT manufacturer = 'samsung';  SELECT \* FROM products WHERE price \* productcount > 100000; |
| **IN** | Определяет набор значений, которые должны иметь столбцы  WHERE manufacturer IN ('Samsung', 'HTC', 'Huawei');  WHERE manufacturer NOT IN ('Samsung', 'HTC', 'Huawei');  WHERE MKB\_code IN (SELECT …) |
| **BETWEEN** | Определяет диапазон значений с помощью начального и конечного значения  WHERE price BETWEEN 20000 AND 50000;  WHERE price NOT BETWEEN 20000 AND 50000; |
| **LIKE**  **NOT LIKE** | Шаблон для сравнения ('выглядит как …')  **\_**  заменяет собой любой 1 символ.  **%**  заменяет собой любое кол-во символов, в том числе и 0.  \ экранирующий символ  SELECT \* FROM ships WHERE class LIKE '%o' ;  SELECT \* FROM ships WHERE class NOT LIKE '%go' AND class LIKE '%o' ; |
| **REGEXP**  **NOT REGEXP** | Регулярное выражение  ^ указывает на начало строки  $ указывает на конец строки  . соответствует любому одиночному символу  [символы] соответствует любому одиночному символу из скобок  [начальный\_символ-конечный\_символ] одиночнsq символ из диапазона  | отделяет шаблоны строки (значение соответствует одному из шаблонов)  WHERE poductname REGEXP 'Phone' # строка должна содержать "Phone",  напр, iPhone X, Nokia Phone N, iPhone  WHERE ProductName REGEXP '^Phone' # строка должна начинаться с "Phone",  напр, Phone 34, PhoneX  WHERE ProductName REGEXP 'Phone$' # строка заканчивается на "Phone",  напр, iPhone, Nokia Phone  WHERE ProductName REGEXP 'iPhone [6-8]' # строка должна содержать  iPhone 6 или iPhone 7 или iPhone 8  WHERE ProductName REGEXP 'Phone|Galaxy' # Phone или Galaxy |
| **IS NULL**  **IS NOT NULL** | Выбрать все строки, столбцы которых имеют значение NULL или не NULL  WHERE productcount IS NULL;  WHERE productcount IS NOT NULL; |
| **DISTINCT** | Выбрать только уникальные значения  SELECT DISTINCT MKB\_code FROM chronic\_illness; |
| **GROUP BY**  **HAVING** | Позволяют сгруппировать данные в рамках команды SELECT.  Если в выражении SELECT производится выборка по одному или нескольким столбцам и также используются агрегатные функции, то необходимо использовать выражение GROUP BY, иначе запрос работать не будет. GROUP BY не работает, если в запросе нет агрегатных функций.  SELECT столбцы  FROM таблица  [WHERE условие\_фильтрации\_строк]  [GROUP BY столбцы\_для\_группировки]  [HAVING условие\_фильтрации\_групп]  [ORDER BY столбцы\_для\_сортировки]  SELECT active\_substance, COUNT(id) FROM drug GROUP BY active\_substance;  SELECT manufacturer, COUNT(\*) AS modelscount  FROM products  WHERE price > 30000  GROUP BY manufacturer  ORDER BY modelscount DESC;  HAVING выполняет фильтрацию групп - определяет, какие группы будут включены в выходной результат. HAVING это 'WHERE' для фильтрации групп.  WHERE - фильтрации строк, HAVING - фильтрации групп.  SELECT active\_substance, COUNT(id) FROM drug GROUP BY active\_substance HAVING COUNT(id) > 1;  SELECT manufacturer, COUNT(\*) AS models, SUM(productcount) AS units  FROM products  WHERE price \* productcount > 80000  GROUP BY manufacturer  HAVING SUM(productcount) > 2  ORDER BY units DESC; |
| **ORDER BY** | Сортирует значения по одному или нескольких столбцам  SELECT \* FROM products ORDER BY price; #отсортировать по возраст (ASC) цены  SELECT \* FROM products ORDER BY price DESC; # отсортировать по убыванию  SELECT \* FROM products ORDER BY name ASC, price DESC; # сорт по 2 столбцам |
| **ANY\_VALUE()** | Возвращает случайное значение из группы  SELECT ANY\_VALUE(name), YEAR(birthday\_at) AS birthday\_year FROM users  GROUP BY birthday\_year  ORDER BY birthday\_year; |
| **WITH ROLLUP** | Добавляет еще одну строку с суммой значений всех предыдущих строк  SELECT SUBSTRING(birthday\_at, 1, 3) AS decade, COUNT(\*) FROM users  GROUP BY decade  WITH ROLLUP; |
| **COUNT** | Вычисляет количество строк в запросе, между COUNT и скобками НЕТ ПРОБЕЛА!!! Агрегатная функция.  SELECT COUNT(\*) FROM products; # всего строк в таблице  SELECT COUNT(manufacturer) FROM products; # всего строк в столбце  (NULL не считается)  SELECT COUNT(DISTINCT manufacturer) FROM products;  # только уникальные названия  комбинация функций:  SELECT COUNT(\*) AS prodcount,  SUM(productcount) AS totalcount,  MIN(price) AS minprice,  MAX(price) AS maxprice,  AVG(price) AS avgprice  FROM products; |
| **SUM** | Вычисляет сумму значений. Агрегатная функция.  SELECT SUM(productcount) FROM products;  SELECT SUM(productcount \* price) FROM products; |
| **AVG** | Вычисляет среднее значение. Агрегатная функция.  SELECT AVG(price) AS average\_price FROM products;  SELECT AVG(price) FROM products WHERE manufacturer='apple';  SELECT AVG(price \* productcount) FROM products; |
| **MAX** | Вычисляет наибольшее значение. Агрегатная функция.  SELECT MIN(price), MAX(price) FROM Products; |
| **MIN** | Вычисляет наименьшее значение. Агрегатная функция.  SELECT MIN(price), MAX(price) FROM Products; |
| **SELECT**  **подзапросы** | INSERT INTO Orders (ProductId, CreatedAt, ProductCount, Price)  VALUES (  (SELECT Id FROM Products WHERE ProductName='Galaxy S8'),  '2018-05-21',  2,  (SELECT Price FROM Products WHERE ProductName='Galaxy S8') ),  SELECT \* FROM Products WHERE Price = (SELECT MIN(Price) FROM Products);  SELECT \* FROM Products WHERE Price > (SELECT AVG(Price) FROM Products);  SELECT \* FROM Products WHERE Id IN (SELECT ProductId FROM Orders);  # если SELECT возвращает больше 1 значения используется **IN** а не =  SELECT \* FROM Products WHERE Id NOT IN (SELECT ProductId FROM Orders);  UPDATE Orders SET Price = (SELECT Price FROM Products WHERE Id=Orders.ProductId) + 3000 WHERE Id=1;  DELETE FROM Orders WHERE ProductId=(SELECT Id FROM Products WHERE ProductName='Galaxy S8');  Чтобы при использовании в операторах сравнения подзапрос мог возвращать набор значений, перед ним необходимо использовать один из операторов: **ALL** или **ANY** (**SOME**).  x > ALL (1, 2) эквивалентно x > 2  x < ALL (1, 2) эквивалентно x < 1  x = ALL (1, 2) эквивалентно (x = 1) AND (x = 2)  x <> ALL (1, 2) эквивалентно x NOT IN (1, 2)  x > ANY (1, 2) эквивалентно x > 1  x < ANY (1, 2) эквивалентно x < 2  x = ANY (1, 2) эквивалентно x NOT (1, 2)  x <> ANY (1, 2) эквивалентно (x <> 1) OR (x <> 2)  SELECT \* FROM Products WHERE Price < ALL(SELECT Price FROM Products WHERE Manufacturer='Apple');  SELECT \* FROM Products WHERE Price < ANY(SELECT Price FROM Products WHERE Manufacturer='Apple');  SELECT \*, (SELECT ProductName FROM Products WHERE Id=Orders.ProductId) AS Product FROM Orders;  # результат подзапроса может представлять отдельный столбец в выборке. Выберем все заказы и добавим к ним информацию о названии товара.  SELECT ProductName,  Manufacturer,  Price,  (SELECT AVG(Price) FROM Products AS SubProds  WHERE SubProds.Manufacturer=Prods.Manufacturer) AS AvgPrice  FROM Products AS Prods  WHERE Price >  (SELECT AVG(Price) FROM Products AS SubProds  WHERE SubProds.Manufacturer=Prods.Manufacturer);  # два коррелирующих подзапроса. Первый подзапрос определяет спецификацию столбца AvgPrice. Он будет выполняться для каждой строки, извлекаемой из таблицы Products. В подзапрос передается производитель товара и на его основе выбирается средняя цена для товаров именно этого производителя. И так как производитель у товаров может отличаться, то и результат подзапроса в каждом случае также может отличаться. Второй подзапрос аналогичен, только он используется для фильтрации извлекаемых из таблицы Products. И также он будет выполняться для каждой строки. |
| **SELECT**  **соединение таблиц** | SELECT \* FROM Orders, Customers;  # соединит таблицы 'неправильно', т.е. каждую строчку с каждой, если в 1 и 2 таблице по 3 строки, получится 9 строк  SELECT \* FROM Orders, Customers WHERE Orders.CustomerId = Customers.Id;  # 3 строки с расширенными данными (столбцы из 2 таблиц)  SELECT Customers.FirstName, Products.ProductName, Orders.CreatedAt  FROM Orders, Customers, Products WHERE Orders.CustomerId = Customers.Id AND Orders.ProductId=Products.Id;  # из 3 таблиц  SELECT C.FirstName, P.ProductName, O.\* FROM Orders AS O, Customers AS C, Products AS P WHERE O.CustomerId = C.Id AND O.ProductId=P.Id;  # из 3 таблиц |
| **JOIN**  **INNER JOIN** | Соединение данных из разных таблиц, по умолчанию INNER JOIN, т.е. строки удовлетворяющие всем запросам. ON - это 'WHERE' относящееся к JOIN  SELECT столбцы  FROM таблица1  [INNER] JOIN таблица2  ON условие1  [[INNER] JOIN таблица3  ON условие2]  WHERE  SELECT O.CreatedAt, O.ProductCount, P.ProductName  FROM Orders AS O  JOIN Products AS P  ON P.Id = O.ProductId;  SELECT Orders.CreatedAt, Customers.FirstName, Products.ProductName  FROM Orders  JOIN Products ON Products.Id = Orders.ProductId  JOIN Customers ON Customers.Id=Orders.CustomerId  WHERE Products.Price > 45000  ORDER BY Customers.FirstName;  SELECT Orders.CreatedAt, Customers.FirstName, Products.ProductName  FROM Orders  JOIN Products ON Products.Id = Orders.ProductId AND Products.Manufacturer='Apple'  JOIN Customers ON Customers.Id=Orders.CustomerId  ORDER BY Customers.FirstName; |
| **LEFT JOIN** | Выборка будет содержать все строки из первой (левой) таблицы, а из второй таблицы только строки, удовлетворяющие условиям  SELECT FirstName, CreatedAt, ProductCount, Price, ProductId  FROM Orders  LEFT JOIN Customers ON Orders.CustomerId = Customers.Id  # включит в выборку те строки из Orders, которых нет в Customers. |
| **RIGHT JOIN** | Выборка будет содержать все строки из второй (правой) таблицы, а из первой таблицы только строки, удовлетворяющие условиям  SELECT FirstName, CreatedAt, ProductCount, Price  FROM Customers  RIGHT JOIN Orders ON Orders.CustomerId = Customers.Id; |
| **UNION** | Объединяет две однотипные выборки с одинаковым количеством столбцов.  SELECT FirstName AS FName, LastName FROM Customers  UNION  SELECT FirstName, LastName FROM Employees ORDER BY FName DESC;  если необходимо вывести повторяющиеся данные - перед вторым SELECT необходимо поставить ALL  SELECT FirstName, LastName FROM Customers  UNION  ALL SELECT FirstName, LastName FROM Employees ORDER BY FirstName;  выборка из одной и той же таблицы:  (SELECT FirstName, LastName, AccountSum + AccountSum \* 0.1 AS TotalSum FROM Customers WHERE AccountSum < 3000)  UNION  (SELECT FirstName, LastName, AccountSum + AccountSum \* 0.3 AS TotalSum FROM Customers WHERE AccountSum >= 3000); |
| **CASE** | Проверяет истинность набора условий и в зависимости от результата проверки может возвращать тот или иной результат.  CASE  WHEN условие\_1 THEN результат\_1  WHEN условие\_2 THEN результат\_2  .................................  WHEN условие\_N THEN условие\_N  [ELSE альтернативный\_результат]  END  SELECT ProductName, ProductCount,  CASE  WHEN ProductCount = 1  THEN 'Товар заканчивается'  WHEN ProductCount = 2  THEN 'Мало товара'  WHEN ProductCount = 3  THEN 'Есть в наличии'  ELSE 'Много товара'  END AS Category  FROM Products; |
| **IF** | В зависимости от результата условного выражения возвращает одно из двух значений. Если условие верно, то возвращается первое значение, иначе возвращается второе значение. IF(условие, значение\_1, значение\_2)  SELECT ProductName, Manufacturer,  IF(ProductCount > 3, 'Много товара', 'Мало товара')  FROM Products; |
| **IFNULL** | Проверяет значение некоторого выражения. Если оно равно NULL, то функция возвращает значение, которое передается в качестве второго параметра.  IFNULL(выражение, значение)  SELECT FirstName, LastName,  IFNULL(Phone, 'не определено') AS Phone,  IFNULL(Email, 'неизвестно') AS Email  FROM Clients; |
| **COALESCE** | Принимает список значений и возвращает первое из них, к-рое не равно NULL  COALESCE(выражение\_1, выражение\_2, выражение\_N)  SELECT FirstName, LastName,  COALESCE(Phone, Email, 'не определено') AS Contacts  FROM Clients;  # Возвращается телефон, если он определен. Если нет, то возвращается эл. адрес. Если и эл. не определен, то возвращается строка "не определено". |
| **TRUE** | 1  SELECT TRUE; # 1  SELECT 2>1; # 1 |
| **FALSE** | 0  SELECT FALSE; # 0  SELECT 2=3; # 0 |
|  |  |
|  |  |
| **Функции работы со строками** | |
| **CONCAT** | Объединяет строки. Если передавать числа, даты - они будут преобразовываться в строки и также объединяться.  SELECT CONCAT('Tom', ' ', 'Smith') # Tom Smith  SELECT UPPER(LEFT(Manufacturer,2)) AS Abbreviation,  CONCAT(ProductName, ' - ', Manufacturer) AS FullProdName  FROM Products  ORDER BY Abbreviation |
| **CONCAT\_WS** | Объединяет строки, но в качестве первого параметра принимает разделитель, который будет соединять строки.  SELECT CONCAT\_WS(' ', 'Tom', 'Smith', 'Age:', 34) # Tom Smith Age: 34 |
| **LENGTH** | Возвращает количество символов в строке  SELECT LENGTH('Tom Smith') # 9 |
| **LTRIM** | Удаляет начальные пробелы из строки  SELECT LTRIM(' Apple') |
| **RTRIM** | удаляет конечные пробелы из строки  SELECT RTRIM(' Apple ') |
| **TRIM** | Удаляет начальные и конечные пробелы из строки  SELECT TRIM(' Tom Smith ') |
| **LOCATE** | Возвращает позицию первого вхождения подстроки find в строку search. Дополнительный параметр start позволяет установить позицию в строке search, с которой начинается поиск подстроки find. Если подстрока search не найдена, то возвращается 0.  LOCATE(find, search [, start])  SELECT LOCATE('om', 'Tom Smith'); # 2  SELECT LOCATE('m', 'Tom Smith'); # 3  SELECT LOCATE('m', 'Tom Smith', 4); # 6  SELECT LOCATE('mig', 'Tom Smith'); # 0 |
| **LEFT** | Вырезает с начала строки определенное количество символов. Первый параметр - строка, а второй - кол-во вырезаемых символов с начала строки:  SELECT LEFT('Apple', 3) # App |
| **RIGHT** | Вырезает с конца строки определенное количество символов. Первый параметр - строка, а второй - кол-во вырезаемых символов с конца строки:  SELECT RIGHT('Apple', 3) # ple |
| **SUBSTRING** | Вырезает из строки str подстроку, начиная с позиции start. Третий необязательный параметр передает количество вырезаемых символов:  SUBSTRING(str, start [, length])  SELECT SUBSTRING('Galaxy S8 Plus', 8), # S8 Plus  (SELECT SUBSTRING('Galaxy S8 Plus', 8, 2) ); # S8 |
| **SUBSTRING\_INDEX** | Вырезает из строки str подстроку. delimiter определяет разделитель внутри строки. count определяет, до какого вхождения разделителя надо вырезать подстроку. Если count положительный, то подстрока вырезается с начала, если count отрицательный, то с конца строки str:  SUBSTRING\_INDEX(str, delimiter, count)  SELECT SUBSTRING\_INDEX('Galaxy S8 Plus', ' ', 1), # Galaxy  (SELECT SUBSTRING\_INDEX('Galaxy S8 Plus', ' ', 2) ), # Galaxy S8  (SELECT SUBSTRING\_INDEX('Galaxy S8 Plus', ' ', -2) ); # S8 Plus |
| **REPLACE** | Заменяет в строке find подстроку search на подстроку replace. Первый параметр функции - строка, второй - подстрока, которую надо заменить, а третий - подстрока, на которую надо заменить  REPLACE(search, find, replace)  SELECT REPLACE('Galaxy S8 Plus', 'S8 Plus', 'Note 8') # Galaxy Note 8 |
| **INSERT** | Вставляет в строку str, заменяя length символов с позиции start подстрокой insert. Первый параметр функции - строка, второй - позиция, с которой надо заменить, третий - сколько символов с позиции start надо заменить вставляемой подстрокой, четвертый параметр - вставляемая подстрока  INSERT(str, start, length, insert)  SELECT INSERT('Galaxy S9', 8, 3, 'Note 9'); # Galaxy Note 9 |
| **REVERSE** | Переворачивает строку наоборот:  SELECT REVERSE('123456789') # 987654321 |
| **LOWER** | Переводит строку в нижний регистр:  SELECT LOWER('Apple') # apple |
| **UPPER** | Переводит строку в верхний регистр  SELECT UPPER('Apple') # APPLE |
| **SPACE** | Возвращает строку, которая содержит определенное количество пробелов |
| **REPEATE** | Возвращает строку, которая содержит определенное количество повторов подстроки str. Количество повторов задается через параметр count.  REPEATE(str, count)  SELECT REPEAT('ab', 5); # ababababab |
| **LPAD** | Добавляет слева от строки str некоторое количество символов, которые определены в параметре pad. Количество добавляемых символов вычисляется по формуле length - LENGTH(str). Если параметр length меньше длины строки str, то эта строка усекается до length символов.  LPAD(str, length, pad)  SELECT LPAD('Tom Smith', 13, '\*'); # \*\*\*\*Tom Smith |
| **RPAD** | Добавляет справа от строки str некоторое количество символов, которые определены в параметре pad. Количество добавляемых символов вычисляется по формуле length - LENGTH(str). Если параметр length меньше длины строки str, то эта строка усекается до length символов.  RPAD(str, length, pad)  SELECT RPAD('Tom Smith', 13, '\*'); # Tom Smith\*\*\*\* |
|  |  |
|  |  |
|  |  |
| **Функции работы с числами** | |
| **ROUND** | Округляет число. Первый параметр - число, второй - длина. Если длина - положит число - до какой цифры после запятой идет округление. Если длина - отриц число - до какой цифры с конца числа до запятой идет округление.  SELECT ROUND(1342.345, 2), # 1342.35  (SELECT ROUND(1342.345, -2)); # 1300;  SELECT ProductName, ROUND(Price \* ProductCount, 2) FROM Products; |
| **TRUNCATE** | оставляет в дробной части определенное количество символов  SELECT TRUNCATE(1342.345, 2); # 1342.34 |
| **ABS** | возвращает абсолютное значение числа  SELECT ABS(-123) # 123 |
| **CEILING** | возвращает наименьшее целое число, которое больше или равно текущему значению.  SELECT CEILING(-123.45), # -123  (SELECT CEILING(123.45)); # 124 |
| **FLOOR** | возвращает наибольшее целое число, которое меньше или равно текущему значению.  SELECT FLOOR(-123.45), # -124  (SELECT FLOOR(123.45)); # 123 |
| **POWER** | возводит число в определенную степень  SELECT POWER(5, 2), # 25  (SELECT POWER(5, 3)); # 125 |
| **SQRT** | получает квадратный корень числа  SELECT SQRT(225); # 15 |
| **SIGN** | возвращает -1, если число меньше 0, и возвращает 1, если число больше 0. Если число равно 0, то возвращает 0.  SELECT SIGN(-5), # -1  (SELECT SIGN(7)); # 1 |
| **DIV** | оператор целочисленного деления  SELECT 5 DIV 2; # 2 |
| **MOD**  **%** | остаток от деления  SELECT 5 MOD 2; # 1  SELECT 5 % 2; # 1 |
| **RAND** | генерирует случайное число с плавающей точкой в диапазоне от 0 до 1  SELECT RAND(); # 0.707365088352935  SELECT RAND(); # 0.173808327956812  Вставка случайных чисел:  UPDATE messages SET  from\_user\_id = FLOOR(1 + (RAND() \* 100)),  to\_user\_id = FLOOR(1 + (RAND() \* 100)); - от 1 до 100  UPDATE messages SET is\_delivered = FLOOR(0.5 + (RAND() \* 1)); - 0 или 1 |
|  |  |
|  |  |
| **Функции работы с датой и временем** | |
| **NOW()** | возвр текущую локальную дату и время на основе системных часов - datetime  SELECT NOW(); # 2018-05-25 21:34:55 |
| **SYSDATE()** | возвр текущую локальную дату и время на основе системных часов - datetime  SELECT SYSDATE(); # 2018-05-25 21:34:55 |
| **CURRENT\_TIMESTAMP()** | возвр текущую локальную дату и время на основе системных часов - datetime  SELECT CURRENT\_TIMESTAMP(); # 2018-05-25 21:32:55 |
| **CURDATE** | возвращают текущую локальную дату в виде объекта date  SELECT CURDATE(); # 2018-05-25 |
| **CURRENT\_DATE** | возвращают текущую локальную дату в виде объекта date  SELECT CURRENT\_DATE(); # 2018-05-25 |
| **CURTIME** | возвращают текущее время в виде объекта time  SELECT CURTIME(); # 20:47:45 |
| **CURRENT\_TIME** | возвращают текущее время в виде объекта time  SELECT CURRENT\_TIME(); # 20:47:45 |
| **UTC\_DATE** | возвращает текущую локальную дату относительно GMT  SELECT UTC\_DATE(); # 2018-05-25 |
| **UTC\_TIME** | возвращает текущее локальное время относительно GMT  SELECT UTC\_TIME(); # 17:47:45 |
| **DAYOFMONTH(date)** | возвращает день месяца в виде числового значения  DAYOFMONTH('2018-05-25') # 25 |
| **DAYOFWEEK(date)** | возвращает день недели в виде числового значения  DAYOFWEEK('2018-05-25') # 6 |
| **DAYOFYEAR(date)** | возвращает номер дня в году  DAYOFYEAR('2018-05-25') # 145 |
| **MONTH(date)** | возвращает месяц даты  MONTH('2018-05-25') # 5 |
| **YEAR(date)** | возвращает год из даты  YEAR('2018-05-25') # 2018 |
| **QUARTER(date)** | возвращает номер квартала года  QUARTER('2018-05-25') # 2 |
| **WEEK(date [, first])** | возвращает номер недели года. Необязательный параметр позволяет задать стартовый день недели. Если этот параметр равен 1, то первым днем считается понедельник, иначе воскресенье.  WEEK('2018-05-25', 1) # 21 |
| **LAST\_DAY(date)** | возвращает последний день месяца в виде даты  LAST\_DAY('2018-05-25') # 2018-05-31 |
| **DAYNAME(date)** | возвращает название дня недели  DAYNAME('2018-05-25') # Friday |
| **MONTHNAME(date)** | возвращает название текущего месяца  MONTHNAME('2018-05-25') # May |
| **HOUR(time)** | возвращает час времени  HOUR('21:25:54') # 21 |
| **MINUTE(time)** | возвращает минуту времени  MINUTE('21:25:54') # 25 |
| **SECOND(time)** | возвращает секунду времени  SECOND('21:25:54') # 54 |
| **EXTRACT** | извлекает из даты и времени какой-то определенный компонент.  EXTRACT(unit FROM datetime)  SECOND (секунды)  MINUTE (минуты)  HOUR (час)  DAY (день)  MONTH (месяц)  YEAR (год)  MINUTE\_SECOND (минуты и секунды)  HOUR\_MINUTE (часы и минуты)  DAY\_HOUR (день и часы)  YEAR\_MONTH (год и месяц)  HOUR\_SECOND (часы, минуты и секунды)  DAY\_MINUTE (день, часы и минуты)  DAY\_SECOND (день, чаы, минуты и секунды)  EXTRACT( MINUTE FROM '2018-05-25 21:25:54') # 25  EXTRACT( DAY\_HOUR FROM '2018-05-25 21:25:54') # 2521  EXTRACT( HOUR\_SECOND FROM '2018-05-25 21:25:54') # 212554  EXTRACT( DAY\_SECOND FROM '2018-05-25 21:25:54') # 25212554 |
| **DATE\_ADD(date, INTERVAL expression unit)** | возвращает объект DATE или DATETIME, который является результатом сложения даты date с определенным временным интервалом. Интервал задается с помощью выражения INTERVAL expression unit, где INTERVAL предоставляет ключевое слово, expression - количество добавляемых к дате единиц, а unit - тип единиц (часы, дни и т.д.) Параметр unit может иметь те же значения, что и в функции EXTRACT, то есть DAY, HOUR и т.д.  DATE\_ADD('2018-05-25', INTERVAL 1 DAY) # 2018-05-26  DATE\_ADD('2018-05-25', INTERVAL 3 MONTH) # 2018-08-25  DATE\_ADD('2018-05-25 21:31:27', INTERVAL 4 HOUR) #2018-05-26 01:31:27 |
| **DATE\_SUB(date, INTERVAL expression unit)** | возвращает объект DATE или DATETIME, который является результатом вычитания из даты date определенного временного интервала  DATE\_SUB('2018-05-25', INTERVAL 4 DAY) # 2018-05-21 |
| **DATEDIFF(date1, date2)** | возвращает разницу в днях между датами date1 и date2  DATEDIFF('2018-05-25', '2018-05-27') # -2  DATEDIFF('2018-05-25', '2018-05-21') # 4  DATEDIFF('2018-05-25', '2018-03-21') # 65  SELECT \* FROM Orders WHERE DATEDIFF(CURDATE(), CreatedAt) = 5; |
| **TO\_DAYS(date)** | возвращает количество дней с 0-го года  TO\_DAYS('2018-05-25') # 737204 |
| **TIME\_TO\_SEC(time)** | возвращает количество секунд, прошедших с момента полуночи  TIME\_TO\_SEC('10:00') # 36000 |
| **DATE\_FORMAT(date, format)**  **TIME\_FORMAT(date, format)** | возвращает объект DATE или DATETIME, отформатированный с помощью шаблона format  возвращает объект TIME или DATETIME, отформатированный с помощью шаблона format  m: месяц в числовом формате 01..12  %с: месяц в числовом формате 1..12  %M: название месяца (January...December)  %b: аббревиатура месяца (Jan...Dec)  %d: день месяца в числовом формате 00..31  %e: день месяца в числовом формате 0..31  %D: номер дня месяца с суффиксом (1st, 2nd, 3rd...)  %y: год в виде двух чисел  %Y: год в виде четырех чисел  %W: название дня недели (Sunday...Saturday)  %a: аббревиатура дня недели (Sun...Sat)  %H: час в формате 00..23  %k: час в формате 0..23  %h: час в формате 01..12  %l: час в формате 1..12  %i: минуты в формате 00..59  %r: время в 12-ти часовом формате (hh:mm:ss AM или PM)  %T: время в 24-ти часовом формате (hh:mm:ss)  %S: секунды в формате 00..59  %p: AM или PM  DATE\_FORMAT('2018-05-25', '%d/%m/%y') # 25/05/18  DATE\_FORMAT('2018-05-25 21:25:54', '%d %M %Y') # 25 May 2018  DATE\_FORMAT('2018-05-25 21:25:54', '%r') # 09:25:54 PM  TIME\_FORMAT('2018-05-25 21:25:54', '%H:%i:%S') # 21:25:24  TIME\_FORMAT('21:25:54', '%k:%i') # 21:25 |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
| **Транзакции** | атомарная группа запросов SQL, т. е. запросы, которые рассматриваются как единое целое. Если база данных может выполнить всю группу запросов, она делает это, но если любой из них не может быть выполнен в результате сбоя или по какой-то другой причине, не будет выполнен ни один запрос группы. Все или ничего.  Нельзя выполнить откат при помощи оператора ROLLBACK для команд (не желательно их вставлять в транзакцию): CREATE INDEX, DROP INDEX, CREATE TABLE, DROP TABLE, TRUNCATE TABLE, ALTER TABLE, RENAME TABLE, CREATE DATABASE, DROP DATABASE, ALTER DATABASE.  ряд операторов, которые неявно завершают транзакцию, как если бы был вызван оператор COMMIT: ALTER TABLE, BEGIN, CREATE INDEX, CREATE TABLE, CREATE DATABASE, DROP DATABASE, DROP INDEX, DROP TABLE, DROP DATABASE, LOAD MASTER DATA, LOCK TABLES, RENAME, SET AUTOCOMMIT=1, START TRANSACTION, TRUNCATE TABLE.  START TRANSACTION; # начало транзакции  SELECT total FROM accounts WHERE user\_id = 4;  UPDATE accounts SET total = total - 2000 WHERE user\_id = 4;  UPDATE accounts SET total = total + 2000 WHERE user\_id IS NULL;  SELECT \* FROM accounts;  COMMIT; # подтверждение вступления в силу транзакции  ROLLBACK; # отмена транзакции  Для работы с точками сохранения предназначены два оператора:  SAVEPOINT и ROLLBACK TO SAVEPOINT  START TRANSACTION;  SELECT total FROM accounts WHERE user\_id = 4;  SAVEPOINT accounts\_4;  UPDATE accounts SET total = total - 2000 WHERE user\_id = 4;  ROLLBACK TO SAVEPOINT accounts\_4; |
| **Переменные** | Переменная доступна только в текущей сессии. Значение присваивается SELECT или SET.  SELECT @total := COUNT(\*) FROM products;  SELECT @total;  если в качестве значения переменной передается имя столбца, содержащего множество значений, то переменная получит последнее значение:  SELECT @id := id FROM products;  SELECT @id;  SET @start := 0;  SELECT @start := @start + 1 AS id, value FROM tbl1;  список всех переменных:  SHOW VARIABLES; |
| **Представления** | Это запрос на выборку (SELECT), которому присваивается уникальное имя и который можно сохранять или удалять из базы данных как обычную хранимую процедуру. Представления позволяют увидеть результаты сохраненного запроса как будто это полноценная таблица базы данных. Представления позволяют гибко управлять правами доступа к таблицам: можно запретить прямое обращение пользователей к таблицам, и разрешить доступ только к представлениям. Представление рассматривается MySQL как полноценная таблица, оно есть в списке таблиц команды SHOW TABLES.  CREATE VIEW <имя> AS <запрос>;  CREATE VIEW cat AS SELECT \* FROM catalogs ORDER BY name;  SELECT \* FROM cat;  CREATE VIEW cat\_reverse (catalog, catalog\_id) AS SELECT name, id FROM catalogs;  SELECT \* FROM cat\_reverse;  # поменяли название столбца name на catalog, а id — на catalog\_id. При этом порядок следования столбцов меняется на обратный.  CREATE OR REPLACE VIEW namecat (id, name, total) AS SELECT id, name, LENGTH(name) FROM catalogs;  SELECT \* FROM namecat ORDER BY total DESC;  # CREATE OR REPLACE VIEW заменяет уже существующее представление  Чтобы в представление можно было вставлять новые записи при помощи INSERT и обновлять при помощи UPDATE, необходимо при создании представления использовать конструкцию WITH CHECK OPTION.  CREATE VIEW v1 AS SELECT \* FROM tbl1 WHERE value < 'fst5'  WITH CHECK OPTION;  INSERT INTO v1 VALUES ('fst4');  Отредактировать представление можно при помощи команды ALTER:  ALTER VIEW v1 AS SELECT \* FROM tbl1 WHERE value > 'fst4'  WITH CHECK OPTION;  или  CREATE OR REPLACE VIEW v1 AS SELECT \* FROM tbl1 WHERE value > 'fst4'  WITH CHECK OPTION;  Для удаления представлений предназначена команда DROP VIEW.  DROP VIEW cat, cat\_reverse, namecat, prod, processors, v1;  DROP VIEW IF EXISTS cat, cat\_reverse, namecat, prod, processors, v1; |
| **Хранимые процедуры** | Хранимые процедуры позволяют сохранить последовательность SQL-операторов и вызывать их по имени. Атрибуты IN, OUT и INOUT доступны лишь для хранимой процедуры, в хранимой функции все параметры всегда имеют атрибут IN. IN - данные передаются строго внутрь хранимой процедуры, если параметру присваивается новое значение, по выходу параметр принимает значение, которое он ранее. OUT - данные передаются строго из хранимой процедуры. Начальное значение параметра внутри хранимой процедуры не принимается во внимание, после вызова параметр имеет значение, присвоенное ему внутри процедуры. INOUT - значение этого параметра как принимается во внимание внутри процедуры, так и сохраняет свое значение по выходу из нее.  DELIMITER //  CREATE PROCEDURE my\_version ()  BEGIN  SELECT VERSION();  END //  CALL my\_version(); # вызов процедуры  просмотр списка процедур:  SHOW PROCEDURE STATUS;  SHOW PROCEDURE STATUS LIKE 'my\_version%'\G;  просмотр содержимого процедуры:  SHOW CREATE PROCEDURE my\_version\G;  удаление процедуры:  DROP PROCEDURE my\_version;  DROP PROCEDURE IF EXISTS my\_version;  Существует два способа назначить переменной новое значение: SET и SELECT (а так же SELECT... INTO ... FROM). Перед  SET @x = value; # входящие или исходящие переманные  SET value = value - 1000;  для объявления локальной переменной без необходимости передавать или возвращать с ее помощью какие-либо значения необходимо объявить такую переменную можно при помощи команды DECLARE  DECLARE id, num INT(11) DEFAULT 0;  DECLARE name, hello, temp TINYTEXT; |
| **Функции** | Хранимые функции позволяют сохранить последовательность SQL-операторов и вызывать их по имени. Разница между процедурой и функцией заключается в том, что функции возвращают значение и их можно встраивать в SQL-запросы, в то время как хранимые процедуры вызываются явно. Ключевое слово DETERMINISTIC (дэтеминистик) сообщает, что при каждом вызове будет возвращаться одно и то же значение, и если его закешировать в рамках запроса, ничего страшного не произойдет. Если значения, которые возвращает функция, каждый раз различны, то используеься NOT DETERMINISTIC. Атрибуты IN, OUT и INOUT доступны лишь для хранимой процедуры, в хранимой функции все параметры всегда имеют атрибут IN.  DELIMITER //  CREATE FUNCTION get\_version ()  RETURNS VARCHAR(255) DETERMINISTIC  BEGIN  RETURN VERSION();  END//  SELECT get\_version(); # вызов процедуры  просмотр списка функций:  SHOW FUNCTION STATUS;  SHOW FUNCTION STATUS LIKE 'my\_version%'\G; |
| **IF** | оператор ветвления  DROP PROCEDURE IF EXISTS format\_now//  CREATE PROCEDURE format\_now (format CHAR(4))  BEGIN  IF(format = 'date') THEN  SELECT DATE\_FORMAT(NOW(), "%d.%m.%Y") AS format\_now;  END IF;  IF(format = 'time') THEN  SELECT DATE\_FORMAT(NOW(), "%H:%i:%s") AS format\_now;  END IF;  END//  или  DROP PROCEDURE IF EXISTS format\_now//  CREATE PROCEDURE format\_now (format CHAR(4))  BEGIN  IF(format = 'date') THEN  SELECT DATE\_FORMAT(NOW(), "%d.%m.%Y") AS format\_now;  ELSEIF (format = 'time') THEN  SELECT DATE\_FORMAT(NOW(), "%H:%i:%s") AS format\_now;  ELSE  SELECT UNIX\_TIMESTAMP(NOW()) AS format\_now;  END IF;  END//  CALL format\_now('date')//  CALL format\_now('time')// |
| **CASE** | оператор ветвления для множественного выбора  DELIMITER //  DROP FUNCTION IF EXISTS hello//  CREATE FUNCTION hello()  RETURNS VARCHAR(255) DETERMINISTIC  BEGIN  DECLARE time INT;  SET time = DATE\_FORMAT(NOW(), '%H');  CASE  WHEN time > -1 AND time < 6 THEN  RETURN "Доброй ночи";  WHEN time > 17 AND time < 24 THEN  RETURN "Добрый вечер";  WHEN time > 11 AND time < 18 THEN  RETURN "Добрый день";  WHEN time > 5 AND time < 12 THEN  RETURN "Доброе утро";  ELSE  RETURN "Доброго здравия!";  END CASE;  END//  SELECT hello()// |
| **WHILE** | CREATE PROCEDURE NOW3 ()  BEGIN  DECLARE i INT DEFAULT 3;  WHILE i > 0 DO  SELECT NOW();  SET i = i - 1;  END WHILE;  END//  Для досрочного выхода из цикла предназначен оператор **LEAVE**.  DROP PROCEDURE IF EXISTS NOWN//  CREATE PROCEDURE NOWN (IN num INT)  BEGIN  DECLARE i INT DEFAULT 0;  IF (num > 0) THEN  cycle: WHILE i < num DO  IF i >= 2 THEN LEAVE cycle;  END IF;  SELECT NOW();  SET i = i + 1;  END WHILE cycle;  ELSE  SELECT 'Ошибочное значение параметра';  END IF;  END//  Для досрочного прекращения итерациипредназначен оператор **ITERATE**.  DROP PROCEDURE IF EXISTS numbers\_string//  CREATE PROCEDURE numbers\_string (IN num INT)  BEGIN  DECLARE i INT DEFAULT 0;  DECLARE bin TINYTEXT DEFAULT '';  IF (num > 0) THEN  cycle : WHILE i < num DO  SET i = i + 1;  SET bin = CONCAT(bin, i);  IF i > CEILING(num / 2) THEN ITERATE cycle;  END IF;  SET bin = CONCAT(bin, i);  END WHILE cycle;  SELECT bin;  ELSE  SELECT 'Ошибочное значение параметра';  END IF;  END//  CALL numbers\_string(9)// |
| **REPEAT** | условие для покидания цикла располагается в конце. Тело цикла в любом случае выполняется хотя бы один раз. В конце цикла после ключевого слова UNTIL располагается условие; если оно истинно, работа цикла прекращается, если ложно, происходит еще одна итерация.  DROP PROCEDURE IF EXISTS NOW3//  CREATE PROCEDURE NOW3 ()  BEGIN  DECLARE i INT DEFAULT 3;  REPEAT  SELECT NOW();  SET i = i - 1;  UNTIL i <= 0  END REPEAT;  END//  CALL NOW3()// |
| **LOOP** | Цикл LOOP, в отличие от операторов WHILE и REPEAT, не имеет условий выхода. Поэтому он должен обязательно иметь в составе оператор LEAVE.  DROP PROCEDURE IF EXISTS NOW3//  CREATE PROCEDURE NOW3 ()  BEGIN  DECLARE i INT DEFAULT 3;  cycle: LOOP  SELECT NOW();  SET i = i - 1;  IF i <= 0 THEN LEAVE cycle;  END IF;  END LOOP cycle;  END//  CALL NOW3()// |
| **Триггер** | специальная хранимая процедура, привязанная к событию изменения содержимого таблицы. Существуют три события изменения таблицы, к которым можно привязать триггер: это изменение содержимого таблицы при помощи команд INSERT, DELETE и UPDATE (BEFORE INSERT, AFTER INSERT и т.д.)  CREATE TRIGGER catalogs\_count AFTER INSERT ON catalogs  FOR EACH ROW  BEGIN  SELECT COUNT(\*) INTO @total FROM catalogs;  END//  DROP TRIGGER IF EXISTS catalogs\_count// # удаление триггера  SHOW TRIGGERS\G // # просмотр списка триггеров  То есть, если в таблице обновляется поле name, то получить доступ к старому значению можно по имени **OLD.name**, а к новому — **NEW.name**  CREATE TRIGGER check\_catalog\_id\_insert BEFORE INSERT ON products  FOR EACH ROW  BEGIN  DECLARE cat\_id INT;  SELECT id INTO cat\_id FROM catalogs ORDER BY id LIMIT 1;  SET NEW.catalog\_id = COALESCE(NEW.catalog\_id, cat\_id);  END//  Функция **COALESCE** возвращает первое не NULL-значение  CREATE TRIGGER check\_catalog\_id\_update BEFORE UPDATE ON products  FOR EACH ROW  BEGIN  DECLARE cat\_id INT;  SELECT id INTO cat\_id FROM catalogs ORDER BY id LIMIT 1;  SET NEW.catalog\_id = COALESCE(NEW.catalog\_id, OLD.catalog\_id, cat\_id);  END//  UPDATE products SET catalog\_id = NULL WHERE name = 'AMD RYZEN 5 1600'//  SELECT id, name, price, catalog\_id FROM products//  UPDATE products SET catalog\_id = 3 WHERE name = 'MSI B250M GAMING PRO'//  SELECT id, name, price, catalog\_id FROM products//  UPDATE products SET catalog\_id = NULL WHERE name = 'MSI B250M GAMING PRO'//  SELECT id, name, price, catalog\_id FROM products//  генерация своей собственной ошибки при помощи команды SIGNAL SQLSTATE. код 45000, который предназначен для пользовательских ошибок. Их невозможно перехватить при помощи обработчиков  CREATE TRIGGER check\_last\_catalogs BEFORE DELETE ON catalogs  FOR EACH ROW BEGIN  DECLARE total INT;  SELECT COUNT(\*) INTO total FROM catalogs;  IF total <= 1 THEN  SIGNAL SQLSTATE '45000' SET MESSAGE\_TEXT = 'DELETE canceled';  END IF;  END// |
| **EXPLAIN** | основной способ узнать, какие решения принимает оптимизатор.  Достаточно добавить слово EXPLAIN перед словом SELECT в запросе. MySQL пометит этот запрос специальным флагом. Во время его обработки этот флаг заставляет сервер сообщать информацию о каждом шаге плана выполнения, а не исполнять его.  EXPLAIN SELECT id, name FROM catalogs UNION ALL SELECT id, name FROM catalogs\G; |
|  |  |
|  |  |
|  |  |
|  |  |
|  | Вставка пола:  CREATE TEMPORARY TABLE sex (sex CHAR(1));  INSERT INTO sex VALUES ('m'), ('f');  UPDATE profiles SET sex = (SELECT sex FROM sex ORDER BY RAND() LIMIT 1); |
|  | Вставка случайных чисел:  UPDATE messages SET  from\_user\_id = FLOOR(1 + (RAND() \* 100)),  to\_user\_id = FLOOR(1 + (RAND() \* 100)); - от 1 до 100  UPDATE messages SET is\_delivered = FLOOR(0.5 + (RAND() \* 1)); - 0 или 1 |
|  | Вставка электронного адреса:  UPDATE media SET filename = CONCAT('https://dropbox/vk/file\_', size); |
|  | Вставка JSON:  UPDATE media SET metadata = CONCAT(  '{"',  'owner',  '":"',  (SELECT CONCAT(first\_name, ' ', last\_name) FROM users WHERE id = user\_id),  '"}'); |
|  |  |