Проектирование реляционной базы данных

Разберем принципы реляционной модели и научимся проектировать базу данных.

[Теория реляционных баз данных, нормальные формы](#_t755nhj7uzcu)

[Первая нормальная форма](#_uvs68jqltolt)

[Пример приведения к первой нормальной форме](#_eh2vwxbhc0hf)

[Вторая нормальная форма](#_rxyway75ine5)

[Пример приведения ко второй нормальной форме](#_bb8nzpazxwkt)

[Третья нормальная форма](#_8m9tdmjffmaa)

[Пример приведения к третьей нормальной форме](#_7ksjz1emul0o)

[Нормальная форма Бойса-Кодда](#_2hc4fscyb3tr)

[Четвёртая нормальная форма](#_4y1f5hujyhps)

[Пятая нормальная форма](#_ty6hh778c7om)

[Доменно-ключевая нормальная форма](#_oo0j2lrgwn7a)

[Шестая нормальная форма](#_kncl0atch8ur)

[Практическое применение](#_3h2gm8h7p1py)

[Типы данных в MySQL](#_wmp5ywlhbook)

[Практическое применение](#_6uuthw753cut)

[Ключи](#_uvruapy1kwcw)

[Первичный ключ](#_olm0mm8fumop)

[Уникальный ключ](#_iqpagjvmbnxr)

[Внешний ключ](#_3snogm9ejz6g)

[Индексы и алгоритмы поиска](#_tr9yntmjynt7)

[Практическая работа с учителем](#_1bvq4p7qxw3o)

Практическое [задание](#_g5fwre2xbar7)

[Дополнительные материалы](#_8n4salcam957)

[Используемые ресурсы](#_fh42ram07c5)

# 

# Теория реляционных баз данных, нормальные формы

Принципы реляционной модели были сформулированы в 1969—1970 годах Э. Ф. Коддом (E. F. Codd). Идеи Кодда были впервые публично изложены в статье «A Relational Model of Data for Large Shared Data Banks», ставшей классической.

Обратимся к википедии для определения реляционной модели данных.

Реляционная модель данных (РМД) – логическая модель данных, прикладная теория построения баз данных, которая является приложением к задачам обработки данных таких разделов математики, как теория множеств и логика первого порядка.

С появлением теории реляционных баз данных на смену хранилищам в файловых системах пришли реляционные системы управления базами данных (РСУБД).

Основа теории реляционной модели данных – это нормальные формы (НФ). Для корректного представления и обработки данных существует шесть нормальных форм.

Основные достоинства данного подхода:

1. Позволяет исключить избыточность.
2. Процесс обновления данных становиться простым.
3. Можно легко расширять и дополнять БД, так как она будет отражать сущности реального мира.
4. Проверка целостности данных будет осуществляться за счет БД.

Работу с РСУБД невозможно представить себе без НФ. Давайте рассмотрим шесть НФ.

## Первая нормальная форма

Переменная отношения находится в первой нормальной форме (1НФ), когда в любом допустимом значении отношения каждый его кортеж содержит только одно значение для каждого из атрибутов.

В реляционной модели отношение всегда находится в первой нормальной форме по определению понятия отношение. Что же касается различных таблиц, они могут не быть правильными представлениями отношений и, соответственно, могут не находиться в 1НФ.

### Пример приведения к первой нормальной форме

|  |  |
| --- | --- |
| **Категория** | **Товары** |
| Строительные материалы | Кирпич, цемент, гвозди |
| Отделочные материалы | Гипсокартон, обои, краска |

В ячейке «Товары» данные присутствуют не в атомарном виде, приведем данные к первой нормальной форме.

|  |  |
| --- | --- |
| **Категория** | **Товары** |
| Строительные материалы | Кирпич |
| Строительные материалы | Цемент |
| Строительные материалы | Гвозди |
| Отделочные материалы | Гипсокартон |
| Отделочные материалы | Обои |
| Отделочные материалы | Краска |

## Вторая нормальная форма

Переменная отношения находится во второй нормальной форме, когда она находится в первой нормальной форме, и каждый неключевой атрибут неприводимо (функционально полно) зависит от её потенциального ключа.

### Пример приведения ко второй нормальной форме

|  |  |  |  |
| --- | --- | --- | --- |
| **Категория** | **Дата акции** | **Скидка** | **Товары** |
| Строительные материалы | 21.06.2016 | 10% | Кирпич |
| Строительные материалы | 21.06.2016 | 10% | Цемент |
| Строительные материалы | 21.06.2016 | 10% | Гвозди |
| Отделочные материалы | 25.06.2016 | 20% | Гипсокартон |
| Отделочные материалы | 25.06.2016 | 20% | Обои |
| Отделочные материалы | 25.06.2016 | 20% | Краска |

В этой таблице первичные ключи представлены в виде двух столбцов – категория и дата акции. Схема функционально зависит от категории и не зависит от даты – т. е. зависимость идет от первичного ключа частично. Приведем таблицу ко второму нормальному виду.

|  |  |  |
| --- | --- | --- |
| **Категория** | **Дата акции** | **Скидка** |
| Строительные материалы | 21.06.2016 | 10% |
| Отделочные материалы | 25.06.2016 | 20% |

Альтернативные СУБД и функциональные надстройки

Знакомство с PostgreSQL. Модули Sphinx, Elastic search. Хранилище Redis.

[Что такое PostgreSQL](#_3znysh7)

[Хранение данных](#_2et92p0)

[Создание нового типа](#_tyjcwt)

[Sphinx](#_17dp8vu)

[ElasticSearch](#_3rdcrjn)

[Redis](#_26in1rg)

Практическое [задание](#_lnxbz9)

[Дополнительные материалы](#_7e0g04nzptat)

[Используемая литература](#_uvp6qax5r1ok)

# 

# Что такое PostgreSQL

Мы уже познакомились с СУБД MySQL и умеем работать с ней на базовом уровне. Но знание только одного инструмента для решения задач – это не очень хорошо. Поэтому стоит рассмотреть в качестве решения СУБД PostgreSQL. Почему именно PostgreSQL? Что может дать эта СУБД в отличие от аналогов (MySQL, Firebird)? Сами создатели PostgreSQL заявляют, что их система «самая продвинутая база данных с открытым исходным кодом в мире». Рассмотрим причины, по которым это утверждение может быть истинным.

Установка на Ubuntu производится следующим образом:

|  |
| --- |
| sudo apt-get install postgresql postgresql-contrib  pg\_createcluster 9.3 main --start |

Теперь можно переключиться на пользователя postgres, из-под которого можно настраивать БД по умолчанию.

|  |
| --- |
| sudo -i -u postgres  psql |

Мы видим, что база доступна. Выйдем из интерфейса postgres и создадим пользователя для работы:

|  |
| --- |
| createuser --interactive |

Создадим пользователя pgadmin со всеми правами. Также надо отредактировать файл /etc/postgresql/<version>/main/pg\_hba.conf, добавив туда строчку:

|  |
| --- |
| host all pgadmin 192.168.230.67/32 md5 |

Теперь можно подключаться к самой БД. Для работы с PostgreSQL вам понадобится соответствующий клиент, например, pgAdmin (<https://www.postgresql.org/)>.

## Хранение данных

MySQL, с которой мы уже знакомы, является реляционной БД. В свою очередь, PostgreSQL не просто реляционная, а объектно-реляционная БД. Это означает, что она поддерживает пользовательские объекты, их поведение, типы данных, функции и многое другое.

Таким образом, уже на уровне БД мы можем определять сущности более конкретно, чем таблицы и поля в них. Т. е. это что-то вроде ООП, но на уровне данных со всеми вытекающими – инкапсуляция, полиморфизм, наследование, переопределение методов и т. п.

Есть очень много типов данных, поддерживаемых в Postgres. Здесь есть не только множество вариаций стандартных типов, но и денежный тип, перечисляемый, геометрический, бинарный; сетевые адреса, битовые строки, текстовый поиск, xml, json, массивы, композитные типы и диапазоны. В различных вариациях и MySQL, и Firebird также знают некоторые из перечисленных типов данных, но лишь Postgres поддерживает их все.

Очень интересным типом хранения в Postgres является массив. Postgres – это объектно-реляционная БД, массивы значений здесь хранятся практически для всех типов данных. Создать их можно, добавляя квадратные скобки к спецификации типа данных или при помощи выражения ARRAY. Указание размерности массива необязательно. Рассмотрим пример применения данного типа через меню пикника:

|  |
| --- |
| -- создаем таблицу, у которой значения являются массивами  CREATE TABLE holiday\_picnic (  holiday varchar(50), -- строковое значение  sandwich text[], -- массив  side text[] [], -- многомерный массив  dessert text ARRAY, -- массив  beverage text ARRAY[4] -- массив из 4-х элементов  );  -- вставляем значения массивов в таблицу  INSERT INTO holiday\_picnic VALUES  ('Labor Day',  '{"roast beef","veggie","turkey"}',  '{  {"potato salad","green salad"},  {"chips","crackers"}  }',  '{"fruit cocktail","berry pie","ice cream"}',  '{"soda","juice","beer","water"}'  ); |

Например, MySQL не умеют совершать такое (JSON type). Чтобы создать нечто подобное, придется придумывать обходной путь и строить отдельную таблицу для каждого из значений массива.

Ещё один интересный тип – геометрические данные. Они требуются в реализациях задач, связанных с геолокацией. Геоданные в настоящее время часто становятся неотъемлемой частью многих приложений. PostgreSQL поддерживает большой набор геометрических типов данных, таких как точки, линии, круги и многоугольники. Например, тип PATH (путь), который состоит из последовательности точек. Путь может быть открытым (начальная и конечная точки не связаны) или закрытым (начальная и конечная точки связаны). Рассмотрим в качестве примера пешеходный маршрут, идущий по петле – т. е. начальная и конечная точки связаны. Круглые скобки вокруг набора координат указывают на закрытый путь, а квадратные – на открытый.

|  |
| --- |
| -- создаем таблицу для хранения троп  CREATE TABLE trails (  trail\_name varchar(250),  trail\_path path  );  -- вставляем тропу в таблицу,  -- для которой маршрут определяется координатами в формате широта-долгота  INSERT INTO trails VALUES  ('Dool Trail - Creeping Forest Trail Loop',  ‘((37.172,-122.22261666667),  (37.171616666667,-122.22385),  (37.1735,-122.2236),  (37.175416666667,-122.223),  (37.1758,-122.22378333333),  (37.179466666667,-122.22866666667),  (37.18395,-122.22675),  (37.180783333333,-122.22466666667),  (37.176116666667,-122.2222),  (37.1753,-122.22293333333),  (37.173116666667,-122.22281666667))’); |

Расширение PostGIS позволяет расширить свойства геометрических данных дополнительными пространственными типами, функциями, операторами и индексами. Это расширение позволяет приложениям поддерживать местоположения, растровые и векторные данные, а также даёт совместимость с многими сторонними гео-API.

В MySQL 5.7.8 добавлены подобные расширения для соответствия стандарту OpenGIS. Предлагается хранение, аналогичное Postgres. Но в MySQL значения данных сначала конвертируются в геометрический формат простыми командами.

Поддержка JSON в PostgreSQL позволяет отказаться от строгих схем хранения данных для некоторых сущностей в SQL БД. Это полезно, когда структура требует гибкости: к примеру, если в процессе разработки структура постоянно меняется или заранее неизвестно, какими полями будет обладать объект данных.

В MySQL 5.7.8 добавлена поддержка объектов JSON. Но они индексируются хуже, чем JSONB в PostgreSQL.

## Создание нового типа

Теперь рассмотрим процесс создания нового типа данных, который вы хотели бы хранить в PostgreSQL. Для этого потребуется команда CREATE TYPE, которая создаёт новые типы данных: составной, перечисляемый, диапазон или базовый.

|  |
| --- |
| -- создаем новый составной тип "wine"  CREATE TYPE wine AS (  wine\_vineyard varchar(50),  wine\_type varchar(50),  wine\_year int  );  -- создаем таблицу, которая использует составной тип "wine"  CREATE TABLE pairings (  menu\_entree varchar(50),  wine\_pairing wine  );  -- вставляем данные в таблицу при помощи выражения ROW  INSERT INTO pairings VALUES  ('Lobster Tail',ROW('Stag''s Leap','Chardonnay', 2012)),  ('Elk Medallions',ROW('Rombauer','Cabernet Sauvignon',2012));  /\*  выборка из таблицы с использованием имени колонки  (используйте скобки, отделяемые точкой от имени поля  в составном типе)  \*/  SELECT (wine\_pairing).wine\_vineyard, (wine\_pairing).wine\_type  FROM pairings  WHERE menu\_entree = 'Elk Medallions'; |

Объёмы данных

Разумеется, ничто в IT-мире не является неограниченным физически. И PostgreSQL не исключение. СУБД имеет следующие ограничения:

* Максимальный размер базы данных не ограничен;
* Максимальный размер таблицы: 32 TB;
* Максимальный размер строки: 1.6 TB;
* Максимальный размер поля: 1 GB;
* Максимальное количество строк в таблице не ограничено;
* Максимальное количество столбцов в таблице: 250–1600;
* Максимальное количество индексов в таблице не ограничено.

Естественно, это не означает, что нужно использовать максимальные настройки для всех возможных параметров. Как и с другими БД, следует с умом создавать таблицы и добавлять индексы.

В MySQL есть ограничение размера строк в 65 535 байт. Как правило, общий объём данных лимитируется максимальным размером файла для выбранной операционной системы. PostgreSQL может сохранять данные в наборе файлов, что позволяет обходить это ограничение. Разумеется, большое количество файлов негативно скажется на скорости работы.

Индексы

Postrges имеет широкие возможности индексирования данных. Например, это частичные индексы, которые создаются при необходимости индексации подмножества данных. Это могут быть строки, отфильтрованные по заданным условиям. Данный функционал позволяет сжимать размеры индексов, необходимых для выполнения узкоспециализированных задач.

|  |
| --- |
| -- создание индекса только для платящих пользователей  CREATE INDEX order\_idx ON orders (order\_id)  WHERE status\_id > 5; |

Также в Postgres есть функциональные индексы, которые создаются при помощи любой функции для предварительного вычисления данных для индексирования. К примеру, у вас есть список наименований, который содержит в себе некие ключевые слова в разных форматах. В такой таблице можно создать индекс, который приводит все ключевые слова к нижнему регистру для нормализации данных

|  |
| --- |
| -- создание индекса для URL в нижнем регистре  CREATE INDEX webhits\_lower\_urls\_idx ON webhits (lower(url)); |

Выводы

СУБД PostgreSQL чрезвычайно богата функционалом, имеющим много встроенных приспособлений, и огромным количеством методов их кастомизации. С учётом признания СУБД со стороны сообщества разработчиков, можно говорить, что данное решение для БД устроит многие команды. Мы рассмотрели небольшое количество возможностей, которые предоставляет Postgres по сравнению с другими решениями. Естественно, их гораздо больше, чем мы перечислили выше.

# Sphinx

Мы подробно рассмотрели СУБД MySQL и познакомились с СуБД PostgreSQL. Однако, базы данных, какими бы производительными они ни были, всё равно остаются одним из наиболее медленных мест в системе. И одной из задач, создающих максимальную нагрузку, является задача поиска по массивам данных. Сам по себе SQL ни в одной реализации из коробки не даёт возможности искать данные быстро. Полнотекстовый поиск – это поиск вхождений в БД на основании содержимого, а также совокупность подходов к оптимизации данного процесса.

Для решения подобной задачи существует поисковый механизм Sphinx. Это ПО для полнотекстового поиска, которое разработано Андреем Аксёновым и распространяется по лицензии GNU GPL. Преимуществом решения является быстрота индексации и процесса поиска. Механизм интегрируется с такими СУБД, как MySQL и PostgreSQL, а также имеет API для популярных языков программирования: PHP, Python, Java.

Одним из самых крупных примеров применения Sphinx является проект бесплатных объявлений Craiglist – ежесуточно он обрабатывает 300 миллионов запросов на поиск.

Для работы Sphinx изначально строит индекс, который описывается в файле конфигурации, затем проводит поиск по выстроенному индексу. Но ведь на многих сайтах данные обновляются ежесекундно. Выходит, чтобы не искать по старым данным, нужно будет каждую секунду перестраивать индекс?

Для решения данной проблемы существует симбиоз двух индексов. Главный (main-index) хранит основные данные, а дополнительный (delta-index) – это небольшой фрагмент данных, хранящий только данные, не вошедшие в главный индекс.

Дополнительный индекс переиндексируется быстро, т. к. работает с меньшим количеством данных. Поэтому его и пересчитывать можно чаще. Раз в сутки происходит переиндексация главного индекса. При этом дополнительный индекс сбрасывается.

# ElasticSearch

Одним из основных конкурентов Sphinx на рынке поисковых движков является ElasticSearch. В отличие от Sphinx решение ElasticSearch является надстройкой над библиотекой Apache Lucene. Lucene без надстроек мало на что годен, т. к. это библиотека для реализации поисковиков. Она может только индексировать и искать. API для ввода дaнных, поисковых запросов, кластеризации и прочее отдается на уровень «обертки».

ElasticSearch хорошо масштабируется. Это означает, что можно на лету подключать новые серверы, а движок сам распределит нагрузку между ними. Данные при этом распределятся так, что при отказе одного из серверов потери не произойдет, а система продолжит работать.

Каждому изменению данных соответствую записи в логах сразу на нескольких серверах кластера, что повышает отказоустойчивость и сохранности данных.

ElasticSearch в отличии от Sphinx даёт возможность загружать обычный JSON-объект, после чего он сам все проиндексирует.

Движок почти полностью управляется через HTTP при помощи запросов в JSON-формате.

У ElasticSearch отсутствуют встроенные системы ограничения доступа. При установке он занимает порт 9200 на все интерфейсы, и это довольно опасная уязвимость. До версии 1.2 было доступно выполнение скриптов в запросах. Таким образом, необходимо учитывать это и закрывать данный порт файрволом.

В список преимуществ Sphinx можно отнести высокую скорость индексации данных. Если вам надо быстро обновить индекс по 10 миллионам документов, это несомненный плюс. Также Sphinx менее требователен к ресурсам сервера. В свою очередь, ElasticSearch имеет удобное и богатое API, которое позволяет сделать поиск более релевантным.

# Redis

Помимо уже знакомых нам реляционным СУБД есть базы данных, которые не используют реляционную модель. Redis сохраняет данные в оперативной памяти, а доступ к ним предоставляет по ключу. По желанию пользователя бэкап данных может сохраняться на жёстком диске. Такой подход обеспечивает производительность, в десятки раз большую, чем у реляционных СУБД.

Redis позволяет хранить целые массивы данных, словари, множества уникальных значений, сортированные множества. СУБД позволяет задать время жизни данных, но по умолчанию они хранятся вечно.

Технологическая особенность Redis в том, что это однопоточный сервер. Этот подход упрощает поддержку кода, разрешает проблему атомарности операций и даёт возможность запускать по одному процессу Redis на каждое ядро процессора. При этом каждый процесс будет работать с выделенным портом.

Redis поддерживает репликацию, но в качестве главного сервера может выступать только один сервер. Однако каждый слейв может выступать в роли главного для других. Реплицирование в Redis не вызывает блокировки на серверах. В случае разрыва связи после восстановлении соединения главный и подчиненный сервер производят полную синхронизацию.

Redis поддерживает механизм publish/subscribe. Этот механизм позволяет создавать каналы, на которые можно подписываться и размещать в них сообщения.

Главное ограничение Redis в том, что объем данных хранимых на физическом сервере, лимитирован объемом RAM. Таким образом, хранение в Redis больших данных стоит дорого, но вполне приемлемы следующие варианты применения:

* хранилище сессий пользователей;
* сервер очередей;
* полноценная замена Memcached;
* кэширующее приложение;
* СУБД для небольших приложений;
* хранилище промежуточных результатов вычислений при обработке больших объемов данных.

# Практическое задание

1. Установить и запустить СУБД PostgreSQL.
2. \* Установить и запустить Sphinx. Проверить его работоспособность.

# Дополнительные материалы

1. «PostgreSQL. Для профессионалов» Дж. Уорсли, Дж. Дрейк.

# Используемая литература

Для подготовки данного методического пособия были использованы следующие ресурсы:

1. <https://xakep.ru/2015/06/11/elasticsearch-tutorial/>
2. <http://chakrygin.ru/2013/03/sphinx-install.html>

Обзор движков MySQL, управление и обслуживание. Подготовка к собеседованию

Что такое движки MySQL и для чего они нужны. Как сделать резервную копию и загрузить схему. Готовимся к поступлению на работу.

[Движки MySQL](#_1fob9te)

[MyISAM](#_3znysh7)

[Static (Fixed-length)](#_2et92p0)

[Dynamic length, формат таблиц](#_tyjcwt)

[Compressed](#_3dy6vkm)

[Советы](#_1t3h5sf)

[InnoDB](#_h396o7hfjk9)

[Советы](#_2s8eyo1)

[MERGE](#_17dp8vu)

[HEAP (MEMORY)](#_3rdcrjn)

[Советы](#_26in1rg)

[ARCHIVE](#_lnxbz9)

[CSV](#_35nkun2)

[FEDERATED](#_1ksv4uv)

[BLACKHOLE](#_44sinio)

[Резервная копия и восстановление из бэкапа](#_2jxsxqh)

[Резервная копия](#_z337ya)

[Восстановление из резервной копии](#_3j2qqm3)

[Управление базой данных](#_1y810tw)

[Права пользователей](#_x9fg1611z0ld)

[Как создать нового пользователя](#_2xcytpi)

[Задачи к собеседованию](#_2bn6wsx)

Практическое [задание](#_3as4poj)

[Дополнительные материалы](#_alnw1yu461wu)

[Используемая литература](#_uvp6qax5r1ok)

# Движки MySQL

## MyISAM

* Наличие поддержки транзакций: отсутствует.
* Максимальный размер таблицы на диске: 256 терабайт.
* Тип блокирования: табличный.
* Наличие полнотекстового поиска: есть.
* Поддержка распределения нагрузки при работе в кластере: отсутствует.
* Поддержка целостности данных: отсутствует.
* Поддержка внешних ключей: отсутствует.
* Поддержка реплицирования: есть.
* Лимит на количество индексов: 64.
* Лимит на количество записей в таблице: 2 в 32 степени.
* Максимально допустимая длина ключа: 1000 байт.
* Максимальное место под ключи на жестком диске: (key\_length+4)/0.67.
* Особенности: движок не поддерживает быстрое восстановление, в случае внезапного отказа сами данные сложно восстанавливать.
* При отсутствии разрывов в рядах данных вставки новых записей не блокируют таблицу. Движок поддерживает хранение данных и индексов на разных разделах жесткого диска или на разных устройствах. Каждое поле в таблице может задавать свою кодировку.
* Максимальная сумма длин для полей типов VARCHAR и CHAR: 64 килобайта.

Теперь рассмотрим типы форматов таблиц, основанных на движке MyISAM.

## Static (Fixed-length)

* Применяется автоматически, когда в таблице нет столбцов с типами VARCHAR, VARBINARY, BLOB, TEXT.
* Работает быстрее, стабильнее, оптимизирован для кэширования, потребуется больше места на HDD в сравнении с другими движками.
* При задании типа *принудительно* столбцы типа VARCHAR и CHAR заполнятся пробелами, VARBINARY – нулями.

## Dynamic length, формат таблиц

* Все строки с длиной до 4 приводятся к типу VARCHAR.
* Значения пустых строк и значение ноль (0) не будут занимать память (заметьте, что NULL – это не 0!).
* Строка фрагментируется в автоматическом режиме при обновлениях таблицы. Для дефрагментации потребуется регулярная оптимизация таблицы (OPTIMIZE TABLE).
* Таблицы этого типа хуже поддаются восстановлению при сбоях.

## Compressed

* Тип создается при использовании утилиты myisampack.
* Тип только для чтения.
* Тип хорош для крайне медленных средств хранения информации.
* Тип может иметь как фиксированный, так и динамический размер.

### Советы

Опыт разработчиков показывает, что таблицы на движке MyISAM рано или поздно сбоят. К этому надо быть готовым.

* Нельзя принудительно отключать сервер во время работы.
* Нельзя изменять одни и те же данные с разных серверов.
* Нельзя изменять таблицы удаленно и с сервера одновременно.

Так или иначе, MyISAM уже устарел, что признается и самими разработчиками MySQL. Упор нужно делать на более современный движок – InnoDB.

## 

## InnoDB

* Максимальный размер данных на жестком диске: 64 терабайта.
* Наличие поддержки транзакций: полная (четыре уровня изоляции).
* Тип блокирования: строчный + 2 типа блокировок транзакций (SHARED, EXCLUSIVE).
* Наличие полнотекстового поиска: есть, начиная с версии MySQL 5.6.4.
* Поддержка целостности данных: есть.
* Поддержка внешних ключей: есть.
* Особенности: кластеризация индексов для часто выполняемых запросов; есть возможность запуска на базе ОС с лимитированным размером файлов; большой набор настроек для тюнинга производительности; есть возможность применения Raw Disk для таблиц в обход файловой системы; из коробки автоматически завершает транзакции коммитом; автоматическое отслеживание deadlock-ов.

### Советы

В старых версиях (до 5.6) запрос типа SELECT (\*) FROM table выполняется значительно дольше, чем в MyISAM. Движок не позволяет перенести данные и таблицы копированием файлов БД. Команда mysqldump выполняется медленно – лучше использовать InnodDb Hot Backup. Движок чувствителен к отсутствию индексов – выигрыш от InnoDB пропадает, если нет индексов.

Начиная с версии 5.6, движок InnoDB является полноценной заменой движку MyISAM.

## MERGE

* Применяется для решения задач слияния нескольких таблиц с одинаковыми структурами в одну.
* Очевидно, таблицы должны иметь идентичную структуру вплоть до порядка столбцов.
* Команда DROP не удаляет исходных таблиц.
* Сливаемые таблицы могут располагаться в другой БД.
* Тип применим для создания алиасов (для одной таблицы).
* Не поддерживает полнотекстовый поиск.
* Запрещено смешивать временные и постоянные таблицы.
* Скорость чтения по ключу низкая.
* Отсутствует команда REPLACE.
* Структурные изменения исходных таблиц не фиксируются – как следствие, итоговая таблица сломается.

## 

## HEAP (MEMORY)

* Наличие поддержки транзакций: отсутствует.
* Тип блокирования: табличный.
* Поддержка репликации: есть.
* Лимит на размер ключа: 500 байт.
* Данные в таблице теряются при остановке сервера.
* Используемый формат хранения: фиксированная длина данных.
* При удалении данных память не высвобождается.

### Советы

Таблицы больших размеров сбрасываются из оперативной памяти на более медленный жесткий диск. Соответственно, выигрыш в скорости от использования данного движка теряется.

Используется для локальных вычислений, хранения временных данных.

## ARCHIVE

* Максимальный размер таблицы на диске: нет лимитов.
* Тип блокирования: строчный.
* Нет команд DELETE, REPLACE, UPDATE, ORDER BY, а также полей типа BLOB.
* Команда INSERT буферизируется и выполняется с большой задержкой.
* Крайне медленный SELECT.

Применение: журналирование.

## CSV

* Движок сохраняет данные в CSV-формате.
* Есть возможность редактирования таблиц внешними приложениями.
* Слабая документация, много открытых ошибок.

## FEDERATED

Прозрачное соединение с другим сервером БД (не путать с репликацией). Имеет много ограничений. Разработчики планируют создать функционал подключения к серверам БД, отличным от MySQL.

## BLACKHOLE

* Данные сохраняются в никуда.
* Двоичные журналы пишутся.

Применение: оптимизация реплицирования (мастер не сохраняет данные на диск).

# Резервная копия и восстановление из бэкапа

## Резервная копия

Для создания резервной копии вашей БД можно воспользоваться утилитой mysqldump.



Чтобы сохранить только структуру БД без данных, нужно выполнить команду:



Если вы хотите сохранить несколько таблиц:



Бэкап может занимать много места. Поэтому его можно сжимать в архив сразу же при создании:



Если вы планируете снимать бэкапы регулярно, удобно указывать, когда они были созданы.



## Восстановление из резервной копии

Теперь рассмотрим процесс восстановления БД из бэкапа:



Если вы используете архив, команда изменится:

## Управление базой данных

Для создания новой БД можно воспользоваться утилитой mysqladmin:



Меняем пароль пользователя-администратора:



При помощи ключей к командам можно сделать бэкап более удобным:



* -Q – при указании ключа все имена полей будут обрамлены обратными кавычками;
* -c – полная вставка, включая имена колонок;
* -e – расширенная вставка. Файл выходит меньше по размеру и создается чуть быстрее.

Если вы хотите увидеть доступные для пользователя базы, выполните команду:



Или же для списка таблиц:



Для таблиц на движке InnoDB нужно добавлять ключ --single-transaction, который гарантирует целостность бекапа.

Для таблиц MyISAM это уточнение неактуально, так как они не поддерживают транзакции.

# Права пользователей

## Как создать нового пользователя

Давайте начнем с создания нового пользователя из консоли MySQL:



Получившийся пользователь «newuser» не может делать с базами данных совершенно ничего. Значит, нам нужно предоставить доступ к информации.



Звездочки в этой команде указывают соответственно базу и таблицу, к которым у пользователя будет доступ. Указанная команда даст пользователю права на чтение, редактирование, выполнение любых действий со всеми БД.

После изменения прав доступа всегда обновляйте состояние прав:



# Задачи к собеседованию

Задача 1. У вас есть социальная сеть, где пользователи (id, имя) могут ставить друг другу лайки. Создайте необходимые таблицы для хранения данной информации. Создайте запрос, который выведет информацию:

* id пользователя;
* имя;
* лайков получено;
* лайков поставлено;
* взаимные лайки.

Задача 2. Для структуры из задачи 1 выведите список всех пользователей, которые поставили лайк пользователям A и B (id задайте произвольно), но при этом не поставили лайк пользователю C.

Задача 3. Добавим сущности «Фотография» и «Комментарии к фотографии». Нужно создать функционал для пользователей, который позволяет ставить лайки не только пользователям, но и фото или комментариям к фото. Учитывайте следующие ограничения:

* пользователь не может дважды лайкнуть одну и ту же сущность;
* пользователь имеет право отозвать лайк;
* необходимо иметь возможность считать число полученных сущностью лайков и выводить список пользователей, поставивших лайки;
* в будущем могут появиться новые виды сущностей, которые можно лайкать.

# Практическое задание

1. Создать нового пользователя и задать ему права доступа на базу данных «Страны и города мира».
2. Сделать резервную копию базы, удалить базу и пересоздать из бекапа.

# Дополнительные материалы

1. «MySQL 5» – Игорь Симдянов, Максим Кузнецов.

# Используемая литература

Для подготовки данного методического пособия были использованы следующие ресурсы:

1. <https://habrahabr.ru/post/64851/>
2. <https://habrahabr.ru/post/105954/>
3. <https://www.digitalocean.com/community/tutorials/mysql-ru>
4. <https://jsehelper.blogspot.com.by/2016/01/sql-1.html>

Масштабирование MySQL и NoSQL

Что такое кластер, что такое репликация данных, где использовать NoSQL.

[Репликация базы данных](#_1fob9te)

[Шаг 1. Настройка Мастера](#_3znysh7)

[Шаг 2. Права на репликацию](#_2et92p0)

[Шаг 3. Дамп базы](#_tyjcwt)

[Шаг 4. Создание базы на Слейве](#_3dy6vkm)

[Шаг 5. Настройка Слейва](#_1t3h5sf)

[Шаг 6. Запуск Слейва](#_4d34og8)

[Статус репликации](#_2s8eyo1)

[Кластеризация БД](#_17dp8vu)

[Немного истории](#_26in1rg)

[Подготовка](#_lnxbz9)

[Настройка кластера](#_35nkun2)

[failover-доступ](#_1ksv4uv)

[NoSQL](#_44sinio)

[История](#_2jxsxqh)

[MongoDB](#_z337ya)

[Архитектура](#_3j2qqm3)

[Создание базы данных](#_1y810tw)

[Выборка данных](#_2xcytpi)

[Запрос к вложенным объектам](#_1ci93xb)

[Условные операторы в MongoDB](#_2bn6wsx)

[Оператор $ne](#_qsh70q)

[Поиск по массивам и операторы $in, $nin, $all](#_3as4poj)

[Оператор $or](#_1pxezwc)

[Оператор $size](#_49x2ik5)

[Оператор $exists](#_2p2csry)

[Оператор $regex](#_147n2zr)

[Обновление данных](#_3o7alnk)

[Обновление отдельного поля](#_23ckvvd)

[Удаление поля](#_ihv636)

[Особенности работы MongoDB](#_eebu9z2yq0sk)

[Практическая работа](#_1tky9gov2g44)

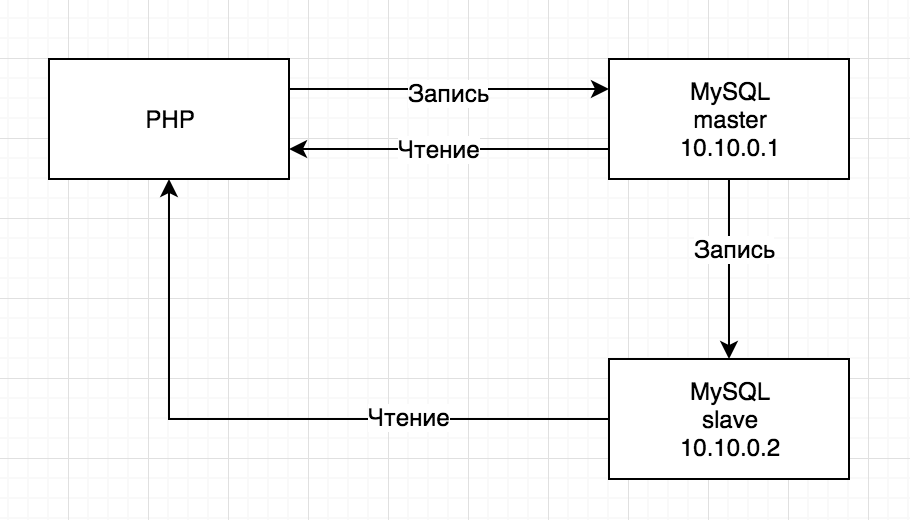
Практическое [задание](#_1v1yuxt)

[Дополнительные материалы](#_y937sk8fclye)

[Используемая литература](#_uvp6qax5r1ok)

# Репликация базы данных

Пассивная MySQL-репликация, построенная по виду Ведущий-Ведомый или Master-Slave, является одним из классических способов построения отказоустойчивой архитектуры приложения. Такой подход позволяет не только зарезервировать данные на зеркальном сервере, но и распределить нагрузку чтения и записи между серверами подобного кластера.



Построение системы реплицирования осуществляется по шагам, которые мы рассмотрим ниже. Для примера будем строить систему из 2 серверов со следующими параметрами:

* Master (ведущий) сервер, 10.10.0.1;
* Slave (ведомый) сервер, 10.10.0.2.

## Шаг 1. Настройка Мастера

Ведущий сервер должен быть обозначен как master. Для этого в файле конфигурации my.cnf вносятся следующие настройки:

|  |
| --- |
| # выбираем ID сервера, произвольное число, лучше начинать с 1 server-id = 1 # путь к бинарному логу log\_bin = /var/log/mysql/mysql-bin.log # название базы данных, которая будет реплицироваться binlog\_do\_db = newdatabase |

После изменения настроек потребуется рестарт ведущего сервера Mysql:

|  |
| --- |
| /etc/init.d/mysql restart |

## Шаг 2. Права на репликацию

Теперь нужно обеспечить доступ к ведущему серверу, чтобы ведомые сервера могли получать обновления:





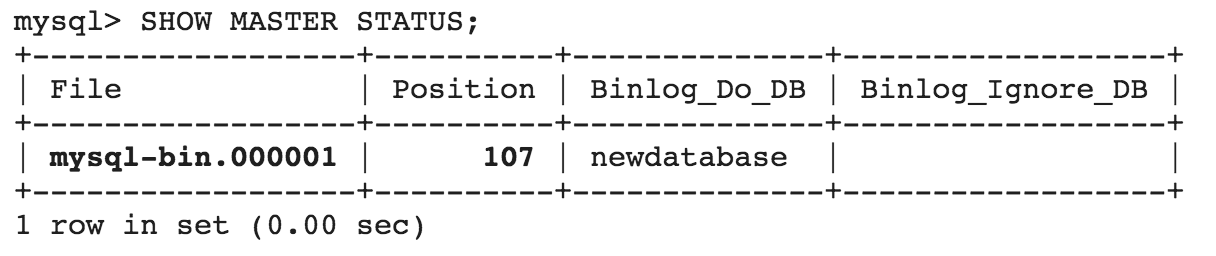
При первом создании репликации нужно заблокировать на изменение все таблицы на ведущем сервере:

|  |
| --- |
| USE newdatabase; FLUSH TABLES WITH READ LOCK; |

После этого проверим статус ведущего сервера:

|  |
| --- |
| SHOW MASTER STATUS; |

Результат будет выглядеть примерно так:



## Шаг 3. Дамп базы

Чтобы система реплицирования данных начала работать, нужно обеспечить ей актуальные данные для старта, сняв дамп данных с заблокированного ведущего сервера:

|  |
| --- |
| mysqldump -u root -p newdatabase > newdatabase.sql |

По завершении снятия дампа можно разблокировать сервер:

|  |
| --- |
| UNLOCK TABLES; |

## Шаг 4. Создание Слейв-сервера

Существует несколько вариантов его создания:

* запустить на выделенном компьютере с IP-адресом, отличным от IP-адреса ведущего сервера;
* воспользоваться виртуализацией и запустить на одном компьютере два контейнера с MySQL-серверами;
* Воспользоваться утилитой mysqld\_multi для запуска на одном компьютере двух mysql-серверов на разных портах, например, 3306 и 3307.

При знакомстве с репликацией, проще всего воспользоваться последним вариантом. Для этого необходимо в конфигурационном файле my.cnf добавить две секции [mysql1] и [mysql2]

|  |
| --- |
| [mysqld]  bind-address = 127.0.0.1  skip-grant-tables  [mysqld1]  socket = /tmp/mysql.sock1  port = 3306  pid-file = /usr/local/var/mysql1/mysqld1.pid  datadir = /usr/local/var/mysql1  [mysqld2]  socket = /tmp/mysql.sock2  port = 3307  pid-file = /usr/local/var/mysql2/mysqld2.pid  datadir = /usr/local/var/mysql2 |

## 

Каждый из серверов будет смотреть в разный каталог данных datadir. Кроме того, мы указываем директиву skip-grant-tables, которая требует, чтобы настройки привилегий пользователей игнорировались и мы могли бы обратиться к серверам не указывая пароль. Дополнительно указываем разные порты и сокеты.

Запустить сервера можно при помощи утилиты mysqld\_multi

|  |
| --- |
| mysqld\_multi start |

Для остановки сервера, можно передать утилите mysqld\_multi

|  |
| --- |
| mysqld\_multi stop |

Теперь один сервер будет работать на 3306 порту, другой — на 3307.

## Шаг 5. Создание базы на Слейве

Теперь поработаем с ведомым сервером. Здесь нам нужно создать копию БД с ведущего сервера:

|  |
| --- |
| CREATE DATABASE newdatabase;  mysql -u root -p newdatabase < newdatabase.sql |

## 

## Шаг 6. Настройка Слейва

Теперь в файле my.cnf на ведомом сервере вам нужно сконфигурировать свойства реплицирования данных:

|  |
| --- |
| # ID Слейва, удобно выбирать следующим числом после Мастера server-id = 2 # Путь к relay-логу relay-log = /var/log/mysql/mysql-relay-bin.log # Путь к bin-логу на Мастере log\_bin = /var/log/mysql/mysql-bin.log # База данных для репликации binlog\_do\_db = newdatabase |

## Шаг 7. Запуск Слейва

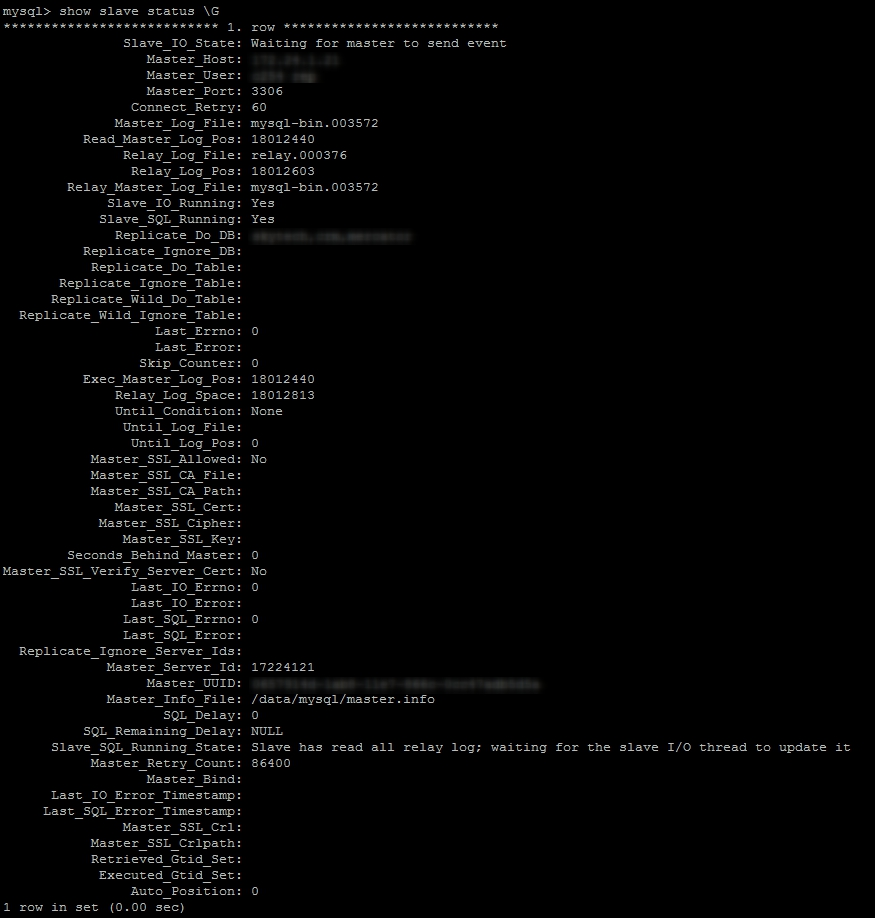
Подготовительные этапы завершены, и мы можем запускать процесс репликации. Для этого осталось только указать, откуда и с какой позиции читать данные ведомому серверу:

|  |
| --- |
| CHANGE MASTER TO MASTER\_HOST='**10.10.0.1**', MASTER\_USER='**slave\_user**',  MASTER\_PASSWORD='**password**', MASTER\_LOG\_FILE = '**mysql-bin.000001**', MASTER\_LOG\_POS = **107**;  START SLAVE; |

Информацию для заполнения полей MASTER\_LOG\_FILE и MASTER\_LOG\_POS извлекается из результата запроса SHOW MASTER STATUS, который был выполнен на втором шаге.

### Статус репликации

Для того, чтобы проверить состояние ведомого сервера, вам надо выполнить на нём вот такой запрос:



# Кластеризация БД

Если возможностей одной ноды MySQL уже недостаточно, на помощь приходит кластеризация.

Технология MySQL Cluster вводит в работу совершенно новый для MySQL способ хранения данных – NDB Cluster. Он позволяет реализовывать выполнение команд по изменению данных на нескольких ведущих серверах, объединенных в кластер.

СУБД MySQL имеет довольно высокие рейтинги использования в веб-проектах, оставаясь при этом простой в использовании и запуске. Все популярные языки программирования для веб поддерживают эту СУБД без дополнительных модулей. Однако это не делает MySQL совершенным решением. MySQL имеет весьма большие проблемы, связанные с быстродействием и надежностью. Тонкая настройка MySQL – это отдельная глава для изучения.

Для создания отказоустойчивого кластера требуется минимум три сервера. Это делается, чтобы два сервера могли проверять корректность своих данных на третьем:

* db1 (IP 10.10.171.2)
* db2 (IP 10.10.171.3)
* db3 (IP 10.10.171.4)

В данной терминологии сервер DB1 – это уже имеющийся у нас сервер со стандартным MySQL-сервером на борту. Данные с него будут переноситься в наш кластер. Сервера DB2 и DB3 для нас являются новыми.

## Немного истории

Для организации кластера мы будем использовать СУБД Percona. На самом деле эта СУБД является тем же MySQL, но при этом – его ответвлением (fork`ом). Для повышения быстродействия MySQL множество системных администраторов и разработчиков постоянно устанавливали пакет исправлений от Google и от Петра Зайцева. Именно он и стал основателем компании Percona. Затем он же создал пакет XtraDB Cluster, включающий в себя библиотеки Gallera Cluster – производительной системы для организации кластеров на базе MySQL.

## Подготовка

Сам по себе XtraDB cluster не содержится в штатных репозиториях Linux ОС. Значит, нам будет необходимо расширить доступные дистрибутивы.

Инструкции по установке для различных систем можно найти здесь  
<https://www.percona.com/doc/percona-server/LATEST/installation.html#installing-from-binaries>

Для того, чтобы узлы (ноды) нашего будущего кластера умели обмениваться данными, каждый сервер должен быть доступен для остальных через порты 4444 и 4567, а также 3306 (штатный порт mysql) и 9199.

## Настройка кластера

В штатном файле настроек нужно ввести строки:

|  |
| --- |
| [mysqld]  datadir=/var/lib/mysql  user=mysql  wsrep\_provider=/usr/lib64/libgalera\_smm.so  wsrep\_cluster\_address=gcomm://10.10.171.2,10.10.171.3,10.10.171.4  binlog\_format=ROW  default\_storage\_engine=InnoDB  innodb\_autoinc\_lock\_mode=2  wsrep\_node\_address=10.10  wsrep\_sst\_method=xtrabackup-v2  wsrep\_cluster\_name=axsystems\_xtradb\_test  wsrep\_sst\_auth="syncuser:some\_pass" |

После этих манипуляций вы можете запускать первую ноду нашего кластера в работу.

|  |
| --- |
| /etc/init.d/mysql bootstrap-pxc |

Как и в случае с репликацией, нам надо создать пользователя, который будет осуществлять синхронизацию:

|  |
| --- |
| mysql@db1> CREATE USER 'syncuser'@'localhost' IDENTIFIED BY 'some\_pass';  mysql@db1> GRANT RELOAD, LOCK TABLES, REPLICATION CLIENT ON \*.\* TO 'syncuser'@'localhost';  mysql@db1> FLUSH PRIVILEGES; |

Теперь полностью копируем конфигурацию на сервер DB2, внося соответствующие коррективы:

|  |
| --- |
| [mysqld]  datadir=/var/lib/mysql  user=mysql  wsrep\_provider=/usr/lib64/libgalera\_smm.so  wsrep\_cluster\_address=gcomm://10.10.171.2,10.10.171.3,10.10.171.4  binlog\_format=ROW  default\_storage\_engine=InnoDB  innodb\_autoinc\_lock\_mode=2  wsrep\_node\_address=10.10  wsrep\_sst\_method=xtrabackup-v2  wsrep\_cluster\_name=axsystems\_xtradb\_test |

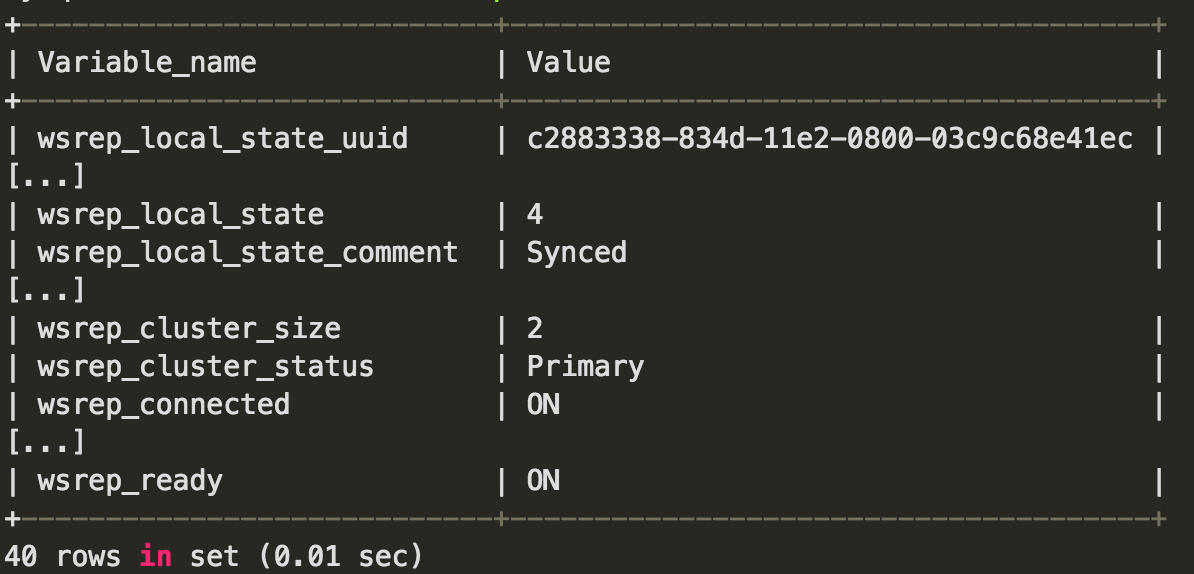
После этого подключаем новый сервер к кластеру:

|  |
| --- |
| /etc/init.d/mysql start |

Процесс запуска довольно долгий в силу того, что второй сервер начинает копировать данные к себе. Ведь вы обратили внимание на то, что мы не снимали бэкап? При этом оба сервера полностью работоспособны и доступны для создания новых данных.

После того, как сервер запустится, вы можете проверять состояние всего кластера, запуская с любого сервера команду:

|  |
| --- |
| mysql> show status like 'wsrep%'; |



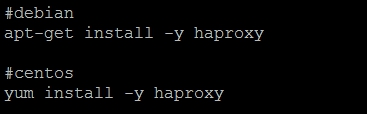
Обратите внимание на следующие параметры:

* status – synced;
* cluster\_size – 2 (узла).

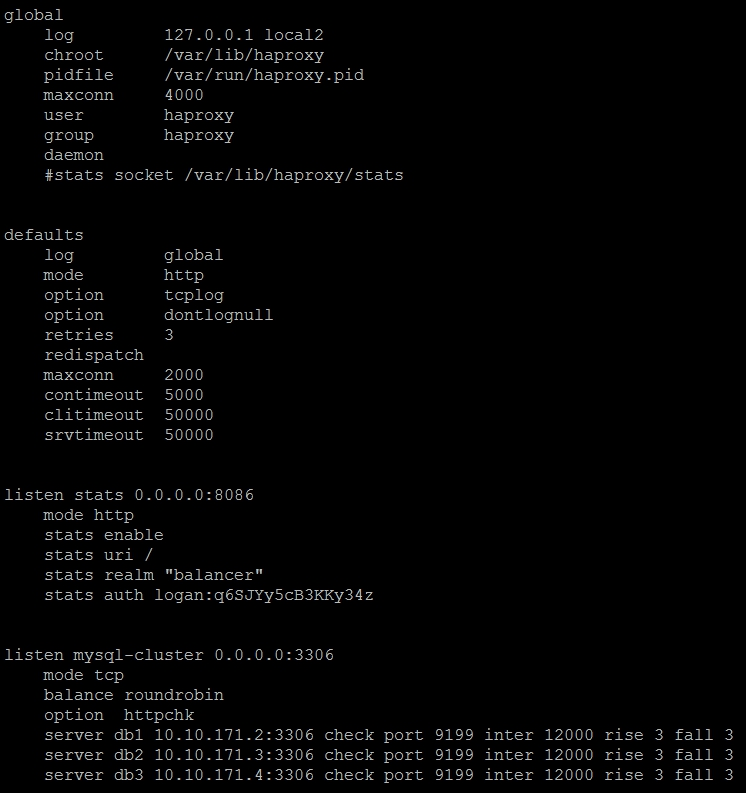
Они говорят, что кластер начал работать в штатном режиме. Теперь мы можем добавлять третий и последующие узлы по образу и подобию.

## failover-доступ

Работа высоконагруженного приложения может быть организована таким образом, чтобы запросы поступали не к одному серверу MySQL, а распределялись между несколькими серверами. Для этого часто применяется балансировщик нагрузки haproxy. Давайте попробуем создать архитектуру, в которой наш балансировщик будет установлен на базе каждой ноды MySQL.

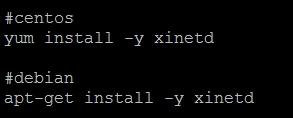


После установки нужно настроить балансировщик, изменив файл /etc/haproxy/haproxy.cfg:

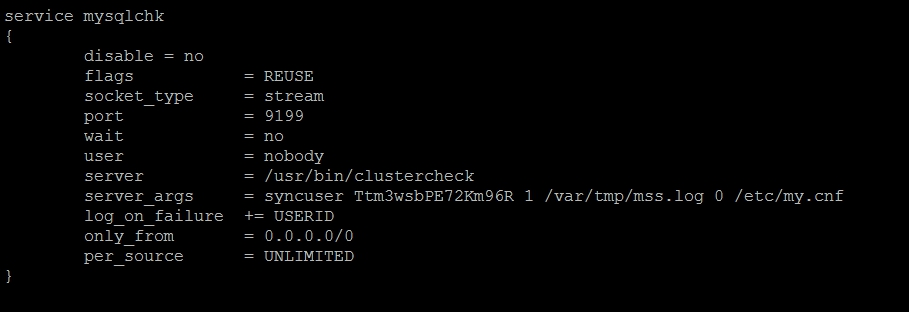


Указанный в настройках сетевой порт 9199 будет применяться балансировщиком для контроля исправности и работоспособности ноды. При этом, разумеется, XtraDB по этому порту не принимает соединения.

Для контроля работы кластера xtradb необходимо создать сервис, который будет управляться через утилиту xinetd.



Теперь напишем скрипт /etc/xinetd.d/mysqlchk для проверки:

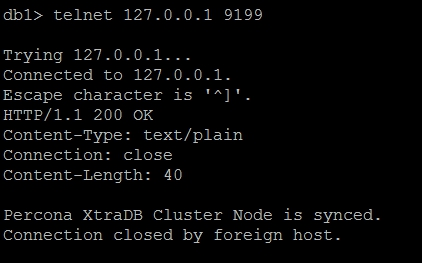


Наиболее важными настройками являются server\_args. Очерёдность в них строго определена: вы указываете имя пользователя, из-под которого производится контроль реплицирования, пароль, файл журналов и файл конфигурации текущей ноды MySQL.

Теперь добавим настройку в /etc/services:



Итак, подготовительная стадия завершена, мы можем перезапустить xinetd и проверить его работу.



Вышеперечисленные действия необходимо произвести со всеми нодами MySQL. По завершении вы перезапускаете балансировщик HAProxy. Вам станет доступен веб-интерфейс балансировки: http://[IP СЕРВЕРА]:8086/). В нём вы можете проверить, что балансировщику доступны все ноды. Отказоустойчивый кластер готов к использованию по адресу localhost!

# NoSQL

Вы наверняка заметили, что технологии NoSQL по популярности обсуждений на технических ресурсах зачастую обходят классические реляционные базы данных. Идёт бурное развитие модульных решений на базе различных языков программирования. Решения NoSQL идут бок о бок с архитектурами, обрабатывающими большие данные (Big Data), легко масштабируемыми и отказоустойчивыми.

## История

Несмотря на то, что понятие NoSQL появилось ещё в девяностых, современный вид подобные решения обрели только в 2009 году. Тогда словом NoSQL именовалось opensource-решение БД, разработанное Карло Строззи, хранившее данные в виде ASCII файлов. Вместо SQL-запросов эта СуБД применяла shell-скрипты. Это решение не имело совершенно ничего общего с тем NoSQL, который развивается в наши дни.

В июне 2009 в городе Сан-Франциско Йохан Оскарссон собрал встречу IT-специалистов, посвященную обзору тенденций среди технологий по хранению и обработке данных. Тому послужили фундаментом такие решения, как BigTable и Dynamo. Для привлечения внимания Эрик Эванс из компании RackSpace предложил простой и ёмкий хэштег – «NoSQL».

Поскольку NoSQL является огромным пластом технологий, охватить его весь нереально. В качестве практической основы мы будем рассматривать MongoDB.

# MongoDB

MongoDB (от англ. humongous – огромный) – open source документоориентированная СУБД, не имеющая описаний схем своих таблиц.

## Архитектура

MongoDB использует для хранения документы в JSON-подобном формате. Сами документы сохраняются в двоичном виде, имея в итоге формат BSON. В Mongo нет транзакций. При этом атомарность гарантируется исключительно на уровне целого документа. Таким образом, понятно, что в СУБД отсутствует частичное обновление документов.

Также в СуБД нет изоляции. Данные, читаемые одним соединением, вполне доступны для изменений другим.

MongoDB имеет асинхронную репликацию master – slave. Она базируется на обмене журналом изменений от мастера к слэйвам. В случае выхода из строя ведущего узла возможно автовосстановление, но для этого серверы mongodb должны быть объединены в т. н. кворум, на основании данных в котором можно гарантированно качественно восстановить данные на мастере.

## Создание базы данных

При установке из коробки БД создаётся в c:/data/db

Давайте запустим mongo.exe и создадим новую БД:

|  |
| --- |
| use users |

После этого нужно установить логин и пароль для администратора:

|  |
| --- |
| db.createUser({user:"admin", pwd:"1234",roles:["readWrite","dbAdmin"]}) |

Для управления используется Mongo-shell.

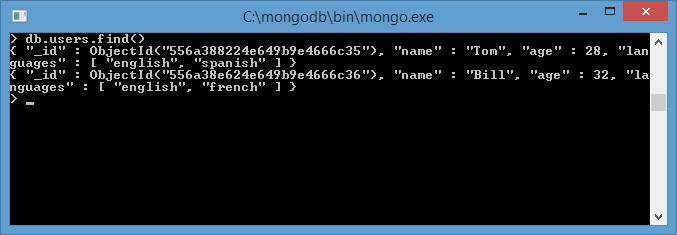
Чтобы добавить данные в коллекцию, нужно вызвать функцию insert.

|  |
| --- |
| db.users.insert({"name": "Tom", "age": 28, languages: ["english", "spanish"]}) |

## Выборка данных

Для выборки данных в отличии от MySQL нужна команда find()

|  |
| --- |
| db.users.find() |



### Запрос к вложенным объектам

Создадим вложенный объект с ключом company.

|  |
| --- |
| db.users.insert({"company":{"name":"Microsoft","age": 25, languages: ["english", "spanish"]}}) |

Теперь для поиска всех документов, имеющих в качестве значения данного ключа name=microsoft, нам необходимо производить вызов через оператор точку:

|  |
| --- |
| db.users.find({"company.name": "Microsoft"}) |

## Условные операторы в MongoDB

Для фильтрации данных в выборках есть возможность указывать условные конструкции при помощи сравнения:

* $gt (больше, чем);
* $lt (меньше, чем);
* $gte (больше или равно);
* $lte (меньше или равно).

Например, найдем все документы, у которых значение ключа age меньше 30:

|  |
| --- |
| db.users.find ({age: {$lt : 30}}) |

В данном случае сравнение производится с типом integer. Когда нужно сравнить строки, конструкция изменится следующим образом:

|  |
| --- |
| db.users.find ({age: {$gt : "30"}}), |

при этом результат тот же. Допускается также комбинация операторов:

|  |
| --- |
| db.users.find ({age: {$gt : 30, $lt: 50}}) |

### Оператор $ne

Также есть оператор отрицания $ne:

|  |
| --- |
| db.users.find ({age: {$ne : 22}}) |

### Поиск по массивам и операторы $in, $nin, $all

Оператор $in по заданному массиву значений выбирает те ключи, значения которых попадают в этот самый заданный массив:

|  |
| --- |
| db.users.find ({age: {$in : [22, 32]}})  db.users.find ({age: {$nin : [22, 32]}}) |

Есть также $all, который очень похож на $in, но он требует, чтобы искомые документы имели весь заданный массив выражений.

|  |
| --- |
| db.users.find ({languages: {$all : [ "english" , "spanish" ]}}) |

### Оператор $or

Оператор задает набор пар ключ-значение для поиска в документе.

|  |
| --- |
| db.users.find ({$or : [{name: "Tom"}, {age: 22}]}) |

Такое выражение найдет все документы, в которых либо name=Tom, либо age=22.

### Оператор $size

Оператор применяется для отыскания документов, в которых массивы данных имеют размерность, равную $size.

|  |
| --- |
| db.users.find ({languages: {$size:2}}) |

### Оператор $exists

Оператор извлекает документы, имеющих заданный ключ в принципе:

|  |
| --- |
| db.users.find ({company: {$exists:true}}) |

### Оператор $regex

Оператор указывает регулярное выражение, которому соответствует искомое поле:

|  |
| --- |
| db.users.find ({name: {$regex:"b"}}) |

## Обновление данных

Наравне с MySQL СУБД MongoDB может менять данные. Самый простой метод – save. Ему передаётся id документа, данные которого надо обновить. В случае, если по данному id документ на найдён, СУБД производит вставку новой записи.

|  |
| --- |
| db.users.save({name: "Eugene", age : 29, languages: ["english", "german", "spanish"]}) |

Данная функция по результатам работы вернёт объект WriteResult:

|  |
| --- |
| WriteResult({"nInserted" : 1 }) |

Но save работает довольно просто. Усложним подход и станем использовать update. Эта функция на вход принимает три параметра:

* query: запрос на поиск;
* objNew: новый документ после обновления;
* options: принимает два аргумента: upsert и multi.

### Обновление отдельного поля

В некоторых ситуациях вам нужно обновить не весь документ, а только один ключ в нём. Специально для этих целей в MongoDB есть функция $set. Если указанный документ не содержит указанное поле, оно будет создано.

|  |
| --- |
| db.users.update({name : "Eugene", age: 29}, {$set: {age : 30}}) |

### Удаление поля

Чтобы удалить поле, используйте $unset:

|  |
| --- |
| db.users.update({name : "Tom"}, {$unset: {salary: 1}}) |

Не волнуйтесь – если ключа в документе нет, оператор просто завершит работу. Помимо этого у вас есть возможность обновить несколько полей сразу:

|  |
| --- |
| db.users.update({name : "Tom"}, {$unset: {salary: 1, age: 1}}) |

Наиболее полный список функций вы всегда сможете найти в [официальной документации MongoDB](https://docs.mongodb.com/).

## Особенности работы MongoDB

При работе с MongoDB непременно нужно знать технические особенности реализации данного решения.

Во-первых, данное хранилище использует достаточно строгие блокировки. И если в базу идёт активная запись, остальные пользователи будут становиться в очередь на чтение. Это нужно иметь в виду при построении приложения.

Во-вторых, MongoDB плохо чистит занимаемое пространство на HDD. Удаляемые логически записи по факту продолжают занимать место, пока вы не запустите сжатие, которое требует, чтобы на диске было доступно как минимум столько же места, сколько уже весит ваша база.

В-третьих, документы в Mongo имеют лимит объёма в 16 мегабайт. Таким образом, для больших агрегаций пригодится механизм map/reduce.

В-четвертых, MongoDB неустойчива по отношению к аварийному выключению. Дело в том, что все используемые данные при работе хранятся в RAM, и лишь спустя какое-то время фиксируются на более медленном HDD. При нештатном отключении данные просто теряются. В последних версиях появилось журналирование, которое позволяет восстанавливать аварийно потерянные данные.

Бэкап в Mongo может быть произведен двумя способами. Можно просто скопировать файлы хранилища. Но, как уже было сказано ранее, есть риск того, что в них не будет храниться вся актуальная информация. Поэтому предпочтительнее применять команду mongodump. Она работает как и mysqldump, собирая дамп данных. Этот дамп можно развернуть при помощи команды mongorestore. Но лучше всего при запуске mongodb настроить репликацию master-slave, снимая резервные копии со slave, чтобы не создавать нагрузку на сервер оригинальных данных. Также slave будет полезен, если умрет master.

# Практическая работа

Практическая часть будет состоять из работы с MongoDB. Для начала нужно познакомиться с утилитой mongodump. Чтобы сделать полный экспорт содержимого и данных схемы, можно воспользоваться командой:

|  |
| --- |
| mongodump |

Данная команда должна быть запущена из новой сессии терминала (при наличии уже запущенного Mongo shell). Данная команда будет использовать стандартный хост и порт, то есть выглядеть это будет вот так:

|  |
| --- |
| mongodump --host localhost --port 27017 |

Вы можете поменять флаги, если вам нужно подключиться к другому серверу.

При запуске данные будут извлекаться в ту же директорию, где и была запущена команда. Чтобы изменить директорию, можно использовать следующий флаг:

|  |
| --- |
| mongodump --out /data/backup/ |

Импорт определенной коллекции:

|  |
| --- |
| mongodump --collection myCollection --db test |

Чтобы сделать экспорт нелокальной базы данных, можно воспользоваться такой командой:

|  |
| --- |
| mongodump --host mongodb1.example.net --port 3017 --username user --password pass --out /opt/backup/mongodump-2013-10-24 |

Чтобы восстановить данные из дампа, нужно воспользоваться командой:

|  |
| --- |
| mongorestore --port <port number> <path to the backup> |

Восстановить данные из дампа:

|  |
| --- |
| mongorestore dump-2013-10-25/ |

Также можно проделать импорт из файла json типа:

|  |
| --- |
| mongoimport --db test --collection docs --file primer-dataset.json |

Давайте проделаем эти действия на примере нашей схемы (схема будет приложена к уроку).

Давайте попрактикуемся с выборками данных. Для начала посмотрим список баз данных и коллекций, которые в них есть.

Показать список схем:

|  |
| --- |
| show dbs; |

Далее используем созданную схему:

|  |
| --- |
| use test; |

Какие коллекции есть в используемой схеме:

|  |
| --- |
| show collections; |

Делаем пробную выборку данных:

|  |
| --- |
| db.docs.find(); |

Выборка с условием:

|  |
| --- |
| db.docs.find({restaurant\_id: "40356731"}); |

Любые запросы далее на выбор учителя.

# Практическое задание

1. Настроить и запустить master-сервер.
2. Установить MongoDB и повторить запросы из методички.

# Дополнительные материалы

1. «MySQL 5» – Игорь Симдянов, Максим Кузнецов.
2. «Обеспечение высокой доступности систем на основе MySQL» – Чарльз Белл, Мэтс Киндал.
3. «MongoDB в действии» – Кайл Бэнкер.

# Используемая литература

Для подготовки данного методического пособия были использованы следующие ресурсы:

1. <http://ruhighload.com/post/%D0%9A%D0%B0%D0%BA+%D0%BD%D0%B0%D1%81%D1%82%D1%80%D0%BE%D0%B8%D1%82%D1%8C+MySQL+Master-Slave+%D1%80%D0%B5%D0%BF%D0%BB%D0%B8%D0%BA%D0%B0%D1%86%D0%B8%D1%8E>
2. <https://prudnitskiy.pro/2015/5/1/xtradb-quickstart/>
3. <https://habrahabr.ru/post/152477/>
4. <https://habrahabr.ru/post/127290/>
5. <http://metanit.com/nosql/mongodb/2.1.php>
6. <https://docs.mongodb.com/getting-started/shell/import-data/>
7. <https://docs.mongodb.com/v3.2/tutorial/install-mongodb-on-os-x/#run-mongodb>

Транзакции и оптимизация запросов

Понятие транзакции, для чего они нужны. Анализ выполнения запросов и повышение производительности.

[Основные понятия транзакции](#_1fob9te)

[Принципы ACID](#_3znysh7)

[Уровни изоляции транзакций](#_2et92p0)

[Транзакции в MySQL](#_tyjcwt)

[Синтаксис команд BEGIN / COMMIT / ROLLBACK](#_1t3h5sf)

[Deadlock](#_3yx9uif188x)

[Практическая работа по транзакциям](#_2s8eyo1)

[EXPLAIN](#_17dp8vu)

[Практическая работа по EXPLAIN](#_3rdcrjn)

Практическое [задание](#_26in1rg)

[Дополнительные материалы](#_y937sk8fclye)

[Используемая литература](#_uvp6qax5r1ok)

# Основные понятия транзакции

Транзакция – это целостная операция, объединяющая под собой несколько запросов в один. Вообще, любой набор действий, который выполняется единой процедурой, является транзакцией.

Отличным примером транзакций из реальной жизни являются банковские операции. Одно действие открытия счёта на самом деле состоит из ряда операций.

С точки зрения баз данных, транзакция – это группа последовательных действий с БД, представляющая собой логическую единицу работы с данными. В итоге транзакция заканчивается либо полным успешным выполнением, либо отменой изменений (и тогда она не должна вызывать последствий). По итогам работы СУБД создает журнал транзакций.

## Принципы ACID

Atomicity – Атомарность.

Этот принцип отвечает за то, что ни одна транзакция не может быть выполнена частично. Как было сказано выше, выполняются либо все шаги, либо ни один из них. Если транзакция не завершена полностью, все действия отменяются, не влияя на систему.

Isolation – Изолированность.

В процессе выполнения параллельные транзакции не должны влиять друг на друга. Это довольно дорогое требование, а потому в MySQL есть не полная блокировка, а различные уровни изолированности.

Consistency – Согласованность/Консистентность.

Транзакция, которая завершилась неким результатом, т. е. зафиксировавшая результаты своей работы, сохраняет согласованность базы данных. Таким образом каждая успешная транзакция сохраняет базу в корректном состоянии.

Консистентность может иметь и более широкое понимание. К примеру, в той же банковской схеме существует требование к равенству суммы списания и суммы зачисления на определенном промежутке времени. Но это не техническое требование, а бизнес-правило. И такие правила гарантируются уже не технической реализацией, а логикой работы транзакций.

Отметим, что в ходе работы транзакции консистентность не требуется. Т. е. то же списание и зачисление будут проходить разными операциями, и между ними будет некий временной промежуток. Но при изоляции транзакций друг от друга другим транзакционным процессам такая несогласованность не будет доступна. При этом атомарность гарантирует, что транзакция будет завершена либо отменена.

Durability – Долговечность/Надежность.

В случае отказа СУБД по какой-либо причине, изменения, сделанные успешной транзакцией, должны оставаться целостными после восстановления работоспособности системы.

## Уровни изоляции транзакций

Для повышения скорости работы в транзакциях реализуется разграничение доступа к данным с разделением по уровням доступа.

0 – Read Uncommitted, Dirty Read – чтение незаконченных изменений своей и других транзакций. При этом, разумеется, не гарантируется, что изменения данных не будут в любой момент отозваны, поэтому подобный тип чтения является источником ошибок.

1 – Read Committed – чтение только изменений, которые были зафиксированы по результатам завершения всех транзакций.

2 – Repeatable Read, Snapshot – чтение всех изменений, но только внутри своей транзакции. Заметьте, что при таком уровне доступа любые изменения, внесённые другими транзакциями после начала своей, недоступны.

3 – Serializable – сериализуемые транзакции. Являются результатом параллельного выполнения сериализуемой транзакции с другими транзакциями. Он должен быть идентичен результату их последовательного выполнения. Проблемы синхронизации не возникают.

Чем выше уровень изоляции, тем больше требуется ресурсов, чтобы его обеспечить. Соответственно, повышение изолированности может приводить к снижению скорости выполнения параллельных транзакций, что является «платой» за повышение надёжности.

# Транзакции в MySQL

|  |
| --- |
| START TRANSACTION  [transaction\_characteristic [, transaction\_characteristic] ...]  transaction\_characteristic:  WITH CONSISTENT SNAPSHOT  | READ WRITE  | READ ONLY  BEGIN [WORK]  COMMIT [WORK] [AND [NO] CHAIN] [[NO] RELEASE]  ROLLBACK [WORK] [AND [NO] CHAIN] [[NO] RELEASE]  SET autocommit = {0 | 1} |

Стоит сказать, что в СУБД MySQL транзакции поддерживаются только в движке InnoDB (мы поговорим о движках подробнее на 7 уроке). Для таблиц, которые не поддерживают транзакции (например, MyISAM), применяется другой принцип поддержания целостности данных – *атомарные операции*. В сравнении с транзакциями они зачастую дают сравнимую или лучшую целостность, имея более высокую производительность. MySQL поддерживает оба принципа, что позволяет выбирать между скоростью и возможностями транзакций. Если код приложения позволяет в случае критических ошибок вызов ROLLBACK, то транзакции явно предпочтительней атомарных операций. Также они гарантируют, что незавершенные обновления не будут сохраняться в БД. Сервер может произвести автоматический откат операций, а БД будет сохранена в консистентном состоянии.

Конечно же, зачастую транзакционные операции можно заменить на атомарные. Но даже при использовании транзакций в системе сохраняется возможность потери данных при непредвиденной остановке сервера. Для MySQL, вне зависимости от наличия транзакций, залогом целостности и безопасности является наличие резервных копий и журнала операций по изменению данных. Эти простые меры позволяют восстановить потерянную информацию.

Атомарность операции состоит в гарантии отсутствия влияния других запросов на выполнение отдельно взятой атомарной операции.

Для обновления данных при помощи одиночной операции можно использовать функции. Применяя следующие приемы, вы создадите весьма эффективную архитектуру:

* Поля модифицируются относительно их текущей величины.
* Обновляются только те поля, которые действительно изменились.

К примеру, при изменении некоторой информации происходит обновление только этой информации и связанных с нею строк. Затем производится проверка, модифицировались эти данные или зависящие от них по сравнению с исходной строкой. Если обновление не удалось, возвращается сообщение, что «некоторые данные, которые вы изменяли, были модифицированы другим пользователем». Затем пользователю выводится предыдущая версия для выбора конечной версии данных.

|  |
| --- |
| *UPDATE tablename SET pay\_back=pay\_back+'relative change';*  *UPDATE customer*  *SET*  *customer\_date='current\_date',*  *address='new address',*  *phone='new phone',*  *money\_he\_owes\_us=money\_he\_owes\_us+'new\_money'*  *WHERE*  *customer\_id=id AND address='old address' AND phone='old phone';* |

Обратите внимание: данный подход эффективно работает даже в случае, если другой пользователь заменит значения в pay\_back или money\_he\_owes\_us.

## Синтаксис команд BEGIN / COMMIT / ROLLBACK

Из коробки СУБД MySQL функционирует с настройкой autocommit. Данная настройка говорит серверу, что обновления будут моментально фиксироваться на хранилище.

Для движков с поддержкой механизма транзакций (InnoDB) режим autocommit вы можете выключить при помощи:

|  |
| --- |
| SET AUTOCOMMIT = 0 |

Затем, чтобы сохранять изменения, вам потребуется применять команду COMMIT каждый раз при необходимости записать обновленные данные.

Также переключение на autocommit возможно и для отдельно взятой операции:

|  |
| --- |
| BEGIN;  SELECT @A:=SUM(salary) FROM table1 WHERE type=1;  UPDATE table2 SET summmary=@A WHERE type=1;  COMMIT; |

Выполняя команды BEGIN или SET AUTOCOMMIT=0, применяйте binary log MySQL, чтобы обеспечить оптимальное хранение резервных копий. Транзакции хранятся в таком журнале в виде единого пакета данных, завершающегося через COMMIT, для обеспечения консистентности.

Также текущую транзакцию автоматически завершают команды:

* ALTER TABLE;
* BEGIN;
* CREATE INDEX;
* DROP DATABASE;
* DROP TABLE;
* RENAME TABLE;
* TRUNCATE.

## Deadlock

В высоконагруженных проектах, которые используют механизм транзакций, рано или поздно появится сообщение об ошибке:

«Deadlock found when trying to get lock; try restarting transaction».

Она говорит, что в процессе транзакции произошла взаимная блокировка двух транзакций. Прежде чем говорить о причинах и способах устранения этой проблемы, стоит разобраться с тем, какие типы блокировок существуют в MySQL.

Официальная документация говорит, что в MySQL имеется два типа блокировок – на чтение (Shared – S) и эксклюзивная (Exclusive – X). Блокировка типа S блокирует выбранные данные на изменение, но позволяет другим запросам читать заблокированные данные. Блокировка типа X более строгая. Она не даёт ни читать, ни писать, ни удалять, ни получать блокировку на чтение.

При более детальном изучении выясняется, что существует два дополнительных вида блокировок. Это intention shared и intention exclusive – блоки таблиц. Они запрещают создание иных блокировок, а также операции LOCK TABLE. Создание подобной блокировки со стороны транзакции – это предупреждение, что она хочет создать соответствующую S или X блокировку.

Итак, если созданная строковая блокировка не дает выполнять команды, транзакция ожидает разблокировки. И когда две транзакции блокируют друг друга, снятия таких блокировок можно ожидать очень долго – это и есть deadlock.

Для получения deadlock-а нужны 2 транзакции, эксклюзивная блокировка и блокировка на чтение, а также строка, которую мы будем блокировать. Последовательность действий:

1. Транзакция 1 создаёт блокировку на чтение и продолжается.
2. Транзакция 2 хочет заблокировать эксклюзивно и ожидает момента, когда Транзакция 1 снимет блокировку на чтение.
3. Транзакция 1 хочет получить эксклюзивную блокировку и ожидает, когда Транзакция 2 получит эксклюзивную блокировку, завершит её и освободит ресурсы.

В момент 3 шага и возникает deadlock, или взаимная блокировка.

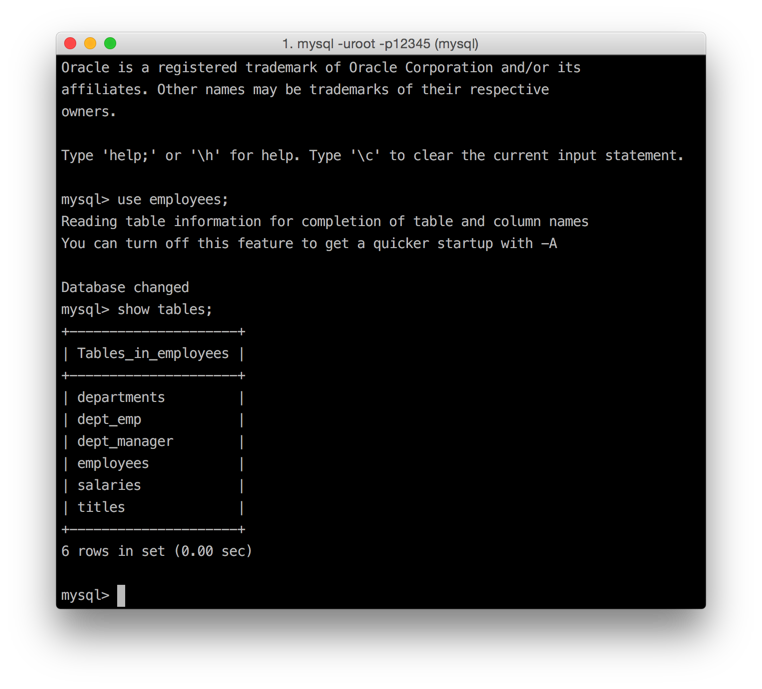
Разработчики СУБД MySQL рекомендуют чаще делать коммиты, контролировать коды ошибок и пытаться перезапустить неудавшуюся транзакцию. Либо можно сразу получать эксклюзивную блокировку. В таком случае на 3 шаге примера выше Транзакция 1 сможет получить блокировку и завершиться.

# Практическая работа по транзакциям

Давайте рассмотрим несколько простых примеров использования транзакций.

Чтобы продемонстрировать работу транзакций, в консоли используем оператор LOCK TABLES, поскольку работу этого оператора проще показать в виде наглядного примера.

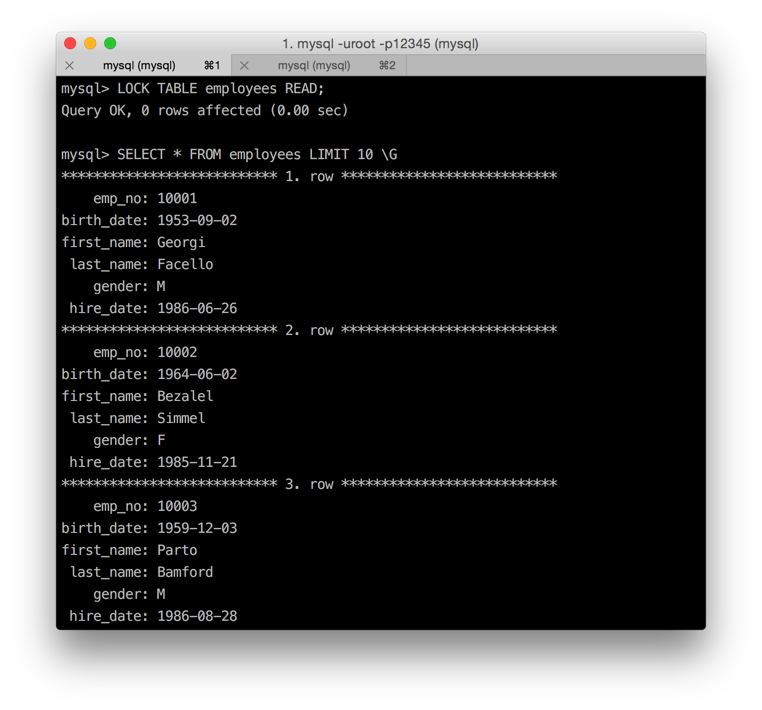
Запустим терминал и авторизуемся в mysql.



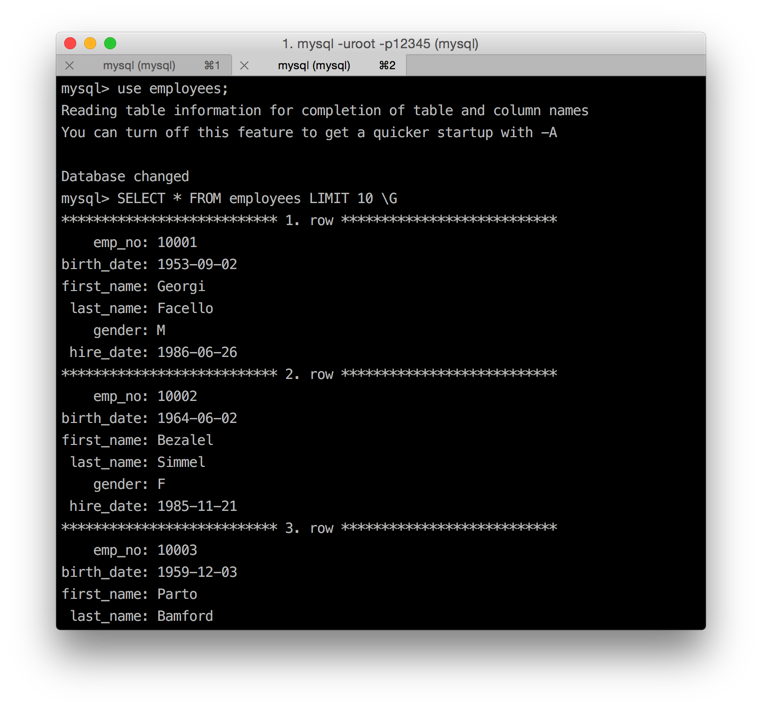
Используем схему employees для демонстрации.

Запустим второе окно и проведем точно такие же действия – авторизуемся и выберем ту же схему.

Пробуем закрыть таблицу из первого окна и выполнить запрос на выборку данных из таблицы.

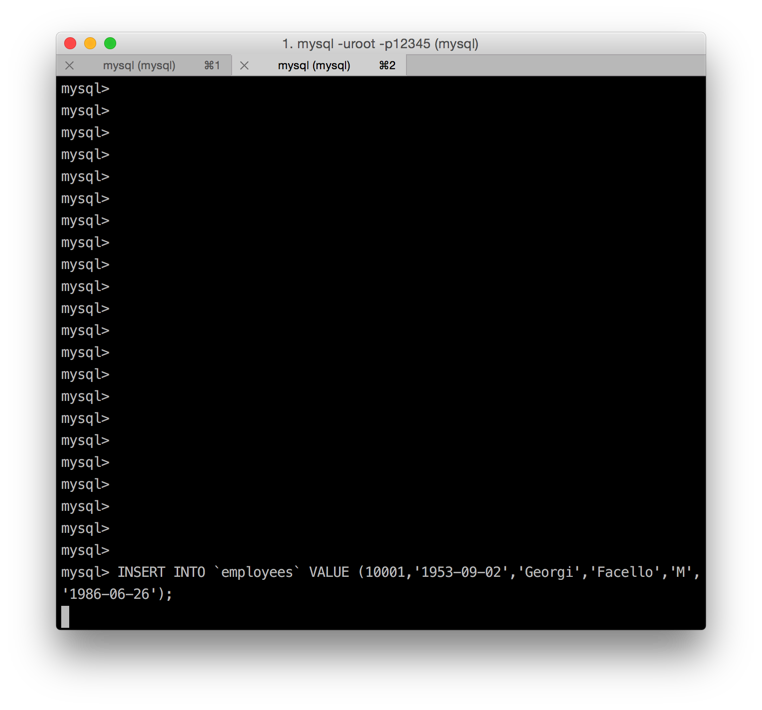


Как мы видим, выборка прошла успешно. Давайте попробуем сделать выборку от второго пользователя.



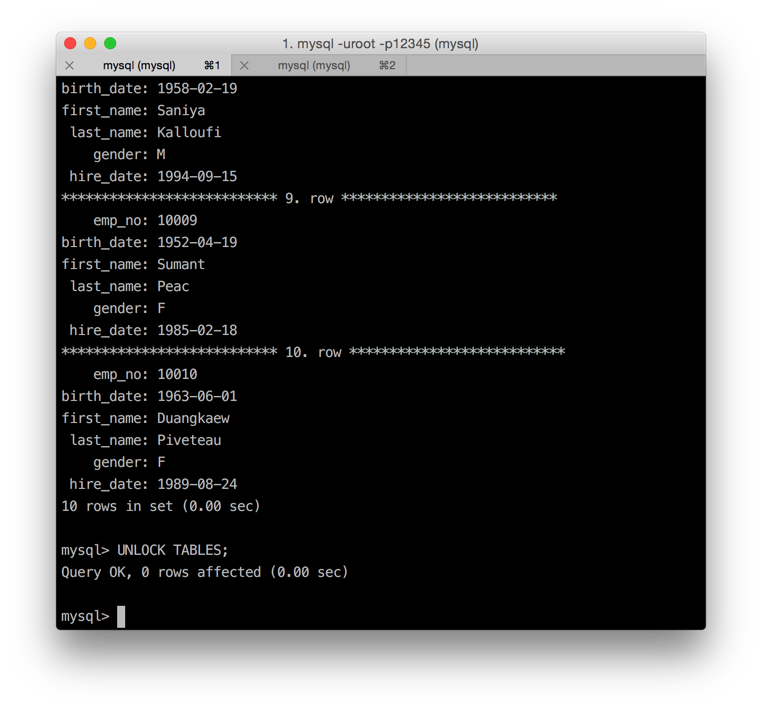
У второго пользователя выборка тоже проходит успешно.

Теперь давайте попробуем добавить нового сотрудника от второго пользователя.



Добавление данных зависло и не продолжается. Это будет продолжаться до тех пор, пока мы не снимем блокировку данных с таблицы.

После снятия блокировки с таблицы запрос сразу выполнится.



С помощью подобного примера легко показать принцип работы транзакций.

# EXPLAIN

Синтаксис оператора:

|  |
| --- |
| EXPLAIN [table\_name] |

Или:

|  |
| --- |
| EXPLAIN SELECT [options] |

Когда SELECT имеет перед собой ключевое слово EXPLAIN, движок MySQL вернет вам план выполнения указанного запроса, а также уведомит о том, в каком порядке и как будут связываться таблицы.

Команда EXPLAIN позволяет проводить анализ, например, когда вам следует добавлять индексы на столбцы. Также есть возможность проанализировать оптимальность шагов выполнения запроса и связывания таблиц. Вы можете изменить порядок связывания через команду STRAIGHT\_JOIN.

Для сложных многоуровневых связей EXPLAIN выдает данные по всем указанным в SQL-запросе таблицам. При выводе они указываются ровно в том же порядке следования, в котором их читает MySQL. Все связи создаются за 1 проход («single-sweep multi-join»). MySQL получает строку из первой таблицы в списке, ищет подходящую строку во 2 таблице, затем в 3, и т. д. При завершении обработки MySQL возвращает полученные столбцы и проходит список в обратную сторону до нахождения таблицы с наибольшим совпадением строк. Следующая строка будет считана из этой таблицы.

Вывод команды:

* table. Имя таблицы, с которой будут производиться действия.
* type. Тип связи в таблице.
* possible\_keys. Здесь указываются индексы, возможные для отыскания подходящих строк в таблице. Столбец не зависит от порядка таблиц, т. е. в реальной ситуации не все указанные ключи подойдут для сгенерированного оптимизатором перечня таблиц. В столбце данные могут отсутствовать, т. е. оптимизатор не смог подобрать индекс. Тогда для ускорения работы запроса надо посмотреть на условие WHERE и попытаться отыскать в нем ссылки на столбец, который можно добавить в индекс.
* key. Здесь содержится имя индекса, который оптимизатор MySQL будет применять во время выполнения запроса.
* key\_len. Длина ключа, примененного оптимизатором. По полученному значению можно увидеть, сколько частей выбранного составного ключа будет применено.
* ref. Показывает столбцы или константы, которые будут использованы с ключом key при выборке.
* rows. Здесь указывается количество строк, выбранных MySQL для анализа перед выполнением запроса.
* Extra. Здесь указывается дополнительная информация о выполнении запроса.
* Distinct. Поиск строк заканчивается после обнаружения первого совпадения строки. MySQL не будет продолжать поиск строк для текущей комбинации.
* Not exists. MySQL произвел анализ LEFT JOIN, нашел присоединяемую строку и не будет искать в этой таблице другие совпадающие строки. Например:

|  |
| --- |
| SELECT \* FROM t1 LEFT JOIN t2 ON t1.id=t2.id WHERE t2.id IS NULL; |

Допустим, t2.id задан в NOT NULL. Тогда система проверит t1 и начнет сканировать t2 в поисках строк, соответствующих t1.id. Когда MySQL обнаруживает в t2 искомую запись, он понимает, что t2.id не принимает значение NULL, и останавливает поиск других строк в t2, имеющих идентичный id. Для всей таблицы t1 сервер производит 1 поиск в t2, вне зависимости от количества совпадений в t2.

* range checked for each record (index map: #). MySQL не удалось отыскать нужного и подходящего запросу индекса для использования. Тогда каждому набору строк в предшествующих таблицах он будет искать подходящий индекс и использовать его. Это медленная операция, но она быстрее, чем поиск без индекса.
* Using filesort. MySQL делает дополнительный поиск, чтобы выяснить метод извлечения строк для сортировки.
* Using index. Для получения данных используется только информация из индекса.
* Using temporary. Для выполнения выборки MySQL должен создать временную таблицу. Классическая ситуация для ORDER BY отличного от GROUP BY.
* Where used. WHERE применяется в процессе выявления строк, сопоставляемых со следующей таблицей, или данных, посылаемых в ответе.

В процессе оптимизации скорости работы ваших SQL-запросов смотрите на наличие Using filesort и Using temporary и старайтесь убирать их.

Нижеуказанные типы связывания таблиц мы рассмотрим в порядке убывания скорости работы:

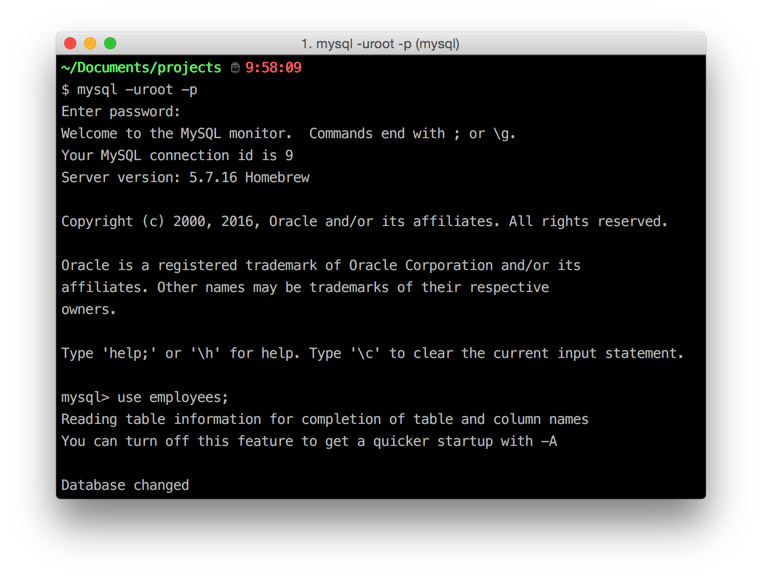
* system. Указанная таблица состоит только из 1 строки.
* const. Таблица содержит до 1 соответствующей строки, считываемой в начале запроса. Таблицы const очень быстрые, т.к. читаются только единожды.
* eq\_ref. Для всех наборов строк MySQL будет сопоставлять 1 строку из текущей таблицы. Это самый лучший тип связи строк в реальных многострочных таблицах.
* ref. Читаются абсолютно все строки с соответствующими им индексами для всех имеющихся комбинаций строк. Работает хорошо, когда ключ сопоставлен не всей таблице, а только нескольким её строкам.
* range. Обрабатываются строки, находящиеся в указанном диапазоне. Выбранный оптимизатором индекс выводится в значении key.
* index. Тот же ALL, но идет поиск только по дереву индексов. Он быстрее ALL, так как индекс почти всегда меньше реальных данных.
* ALL. Для всех комбинаций строк оптимизатор производит полный анализ таблицы. Самый медленный вариант.

Для определения эффективности типа связи необходимо получить произведение значений rows. Это довольно грубая оценка количества строк для анализа при выполнении выборки. То же число применяется для задания лимита на количества запросов в системной переменной max\_join\_size.

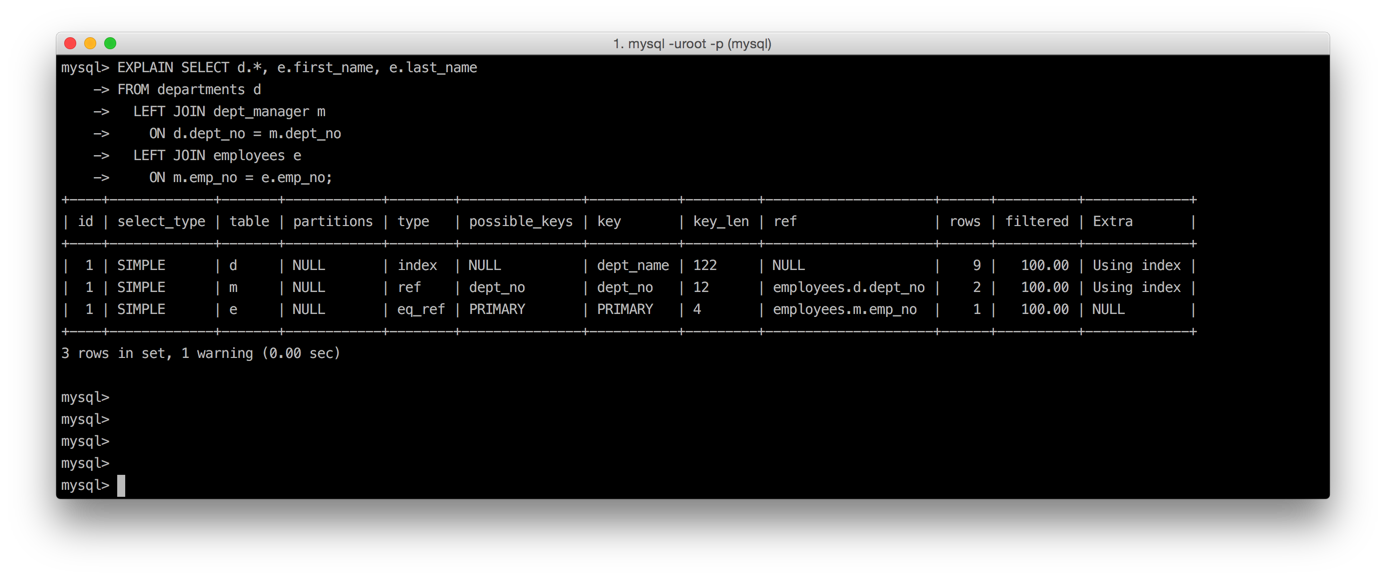
# Практическая работа по EXPLAIN

Давайте попробуем выполнить несколько запросов и проанализируем вывод

Используем нашу служебную схему employees и выполним запрос.



|  |
| --- |
| EXPLAIN SELECT d.\*, e.first\_name, e.last\_name  FROM departments d  LEFT JOIN dept\_manager m  ON d.dept\_no = m.dept\_no  LEFT JOIN employees e  ON m.emp\_no = e.emp\_no |



# Практическое задание

1. Реализовать практические задания на примере других таблиц и запросов.
2. Подумать, какие операции являются транзакционными, и написать несколько примеров с транзакционными запросами.
3. Проанализировать несколько запросов с помощью EXPLAIN.

# Дополнительные материалы

1. «MySQL 5» – Игорь Симдянов, Максим Кузнецов.

# Используемая литература

Для подготовки данного методического пособия были использованы следующие ресурсы:

1. <https://ru.wikipedia.org/wiki/%D0%A2%D1%80%D0%B0%D0%BD%D0%B7%D0%B0%D0%BA%D1%86%D0%B8%D1%8F_(%D0%B8%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0)>
2. <https://ru.wikipedia.org/wiki/ACID>
3. <http://www.mysql.ru/docs/man/ANSI_diff_Transactions.html>
4. <http://www.mysql.ru/docs/man/COMMIT.html>
5. <http://www.mysql.ru/docs/man/EXPLAIN.html>

Объединение запросов, хранимые процедуры, триггеры, функции

Для чего нужны view, хранимые процедуры, триггеры и функции.

[VIEW](#_fddwwjfianmc)

[Примеры](#_jjxjuc5bwm9b)

[Хранимые процедуры и функции](#_bo2q1tvpktnj)

[Примеры](#_ssya59ntl1wd)

[Триггеры](#_mnt8w0ahuyqg)

[Пример создания триггеров](#_eincxhs9ok2i)

[Практическая работа](#_lfio92x9hwsa)

Практическое [задание](#_l91ncnekhns)

[Дополнительные материалы](#_l3wc3oro41n4)

[Используемая литература](#_uvp6qax5r1ok)

# VIEW

Очень полезная возможность – сохранить запрос, который часто выполняется. Делается это с помощью простой команды.

Синтаксис команды:

|  |
| --- |
| CREATE  [OR REPLACE]  [ALGORITHM = {UNDEFINED | MERGE | TEMPTABLE}]  [DEFINER = { user | CURRENT\_USER }]  [SQL SECURITY { DEFINER | INVOKER }]  VIEW view\_name [(column\_list)]  AS select\_statement  [WITH [CASCADED | LOCAL] CHECK OPTION] |

### Примеры

|  |
| --- |
| CREATE VIEW Londonstaff  AS SELECT \*  FROM Salespeople  WHERE city = 'London'; |

Обращение происходит как к простой таблице:

|  |
| --- |
| SELECT \* FROM Londonstaff; |

Можно изменять представление, но изменения будут проходить не в представлении (так как оно не содержит никакой реальной таблицы), а в таблице, на которую это представление указывает.

|  |
| --- |
| CREATE VIEW Salesown  AS SELECT snum, sname, city  FROM Salespeople; |

|  |
| --- |
| UPDATE Salesown  SET city = 'Palo Alto'  WHERE snum = 1004; |

Чтобы удалить представление, используется команда:

|  |
| --- |
| DROP VIEW < view name > |

# Хранимые процедуры и функции

Сперва стоит разобраться, зачем могут понадобиться хранимые процедуры и функции.   
Все запросы, которые выполняет MySQL, проходят анализатор и компилируются на стороне сервера. Каждый раз на это может уходить значительное время. Одинаковые запросы будут автоматически кешироваться, что позволит выполнять их, минуя шаги анализа и компиляции.  
Если в приложении есть очень сложные логические части, которые включают в себя циклические выборки, очень большие аналитические задачи, их можно выполнить намного быстрее с помощью хранимых процедур и функций.

Не стоит делать все простые операции (скажем, CRUD) в приложении на хранимых процедурах и функциях, так как значительного прироста производительности за счет этого не добиться, но управляемость кода, который вы пишете, очень сильно упадет. Подобным способом решаются только сложные аналитические задачи.

Синтаксис создания хранимых процедур и функций.

|  |
| --- |
| CREATE  [DEFINER = { user | CURRENT\_USER }]  PROCEDURE sp\_name ([proc\_parameter[,...]])  [characteristic ...] routine\_body CREATE  [DEFINER = { user | CURRENT\_USER }]  FUNCTION sp\_name ([func\_parameter[,...]])  RETURNS type  [characteristic ...] routine\_body proc\_parameter:  [ IN | OUT | INOUT ] param\_name type func\_parameter:  param\_name type type:  Any valid MySQL data type characteristic:  COMMENT 'string'  | LANGUAGE SQL  | [NOT] DETERMINISTIC  | { CONTAINS SQL | NO SQL | READS SQL DATA | MODIFIES SQL DATA }  | SQL SECURITY { DEFINER | INVOKER } routine\_body:  Valid SQL routine statement |

### Примеры

Для того, чтобы создать процедуру, нужно изменить служебную переменную delimiter, которая служит для обозначения конца команды, т. е. в конце команды можно использовать не «;», а любой другой символ. Это нужно, чтобы внутри хранимой процедуры можно было легко использовать символ «;» и анализатор не думал, что это конец команды.

В процедуре бывает 2 типа параметров – входные и выходные. Во входных параметрах значения передаются в функцию, через выходные параметры значения можно записать, например, в переменную.

Объявим процедуру с выходным параметром.

|  |
| --- |
| mysql> delimiter //  mysql> CREATE PROCEDURE simpleproc (OUT param1 INT)  -> BEGIN  -> SELECT COUNT(\*) INTO param1 FROM t;  -> END// Query OK, 0 rows affected (0.00 sec)  mysql> delimiter ; |

Пример вызова процедуры:

|  |
| --- |
| mysql> CALL simpleproc(@a); Query OK, 0 rows affected (0.00 sec) mysql> SELECT @a; +------+ | @a | +------+ | 3 | +------+ 1 row in set (0.00 sec) |

Давайте теперь разберемся, что происходит в примере выше.

1. Мы объявили переменную @a. В MySQL переменные объявляются с помощью символа @.
2. CALL simpleproc(@a); мы вызвали процедуру и записали результат в переменную, так как параметр является выходным.
3. SELECT @a; вывели результат на экран (вывели значение переменной).

Функция может принимать входящие параметры, а также возвращать значения. Этот подход может быть намного проще, не нужно объявлять переменные, результат сразу передается в SELECT.

|  |
| --- |
| mysql> CREATE FUNCTION hello (s CHAR(20)) mysql> RETURNS CHAR(50) DETERMINISTIC  -> RETURN CONCAT('Hello, ',s,'!'); Query OK, 0 rows affected (0.00 sec) mysql> SELECT hello('world'); +----------------+ | hello('world') | +----------------+ | Hello, world! | +----------------+1 row in set (0.00 sec) |

В функции определяется, какие параметры она будет принимать, а также возвращать.

1. s CHAR(20) принимает параметр размером до 20 символов.
2. RETURNS CHAR(50) DETERMINISTIC возвращает до 50 символов.
3. SELECT hello('world'); – вызываем функцию и выводим результат на экран.

Как вы видите, синтаксис хранимых процедур и функций довольно прост, не отличается от подхода в любых других языках программирования.

# Триггеры

Чтобы в СУБД запускались действия по событиям (например, после удаления данных в определенной таблице), используются триггеры.

Про триггеры можно сказать следующее. Они очень удобны для обработки каких-либо событий в СУБД: будучи установленным один раз, обработчик будет работать и выполнять свою функцию. Но есть и ряд проблем, при которых подобный подход может создать трудности – например, при отладке иногда достаточно сложно понять, что какие-то действия обрабатывает триггер. Поэтому многие считают, что это плохой паттерн, при его использовании отладка очень сложна, особенно если триггеров очень много.

Главный плюс триггеров: они минуют долгую фазу анализа и компиляции запроса.

Из этого можно сделать вывод: для использования триггеров нужно иметь четкую спецификацию ПО, в которой будет описаны места кода, где они используются. Триггеры нельзя использовать бездумно и вешать без предварительного описания, а также не взвесив все за и против данного подхода.

### Пример создания триггеров

|  |
| --- |
| mysql> CREATE TABLE account (acct\_num INT, amount DECIMAL(10,2)); Query OK, 0 rows affected (0.03 sec) mysql> CREATE TRIGGER ins\_sum BEFORE INSERT ON account  -> FOR EACH ROW SET @sum = @sum + NEW.amount; Query OK, 0 rows affected (0.06 sec) |

# Практическая работа

Используя БД «Сотрудники» (которую мы получили на предыдущем уроке), давайте создадим VIEW.

|  |
| --- |
| CREATE VIEW view\_managers  AS  SELECT d.\*, e.first\_name, e.last\_name  FROM departments d  LEFT JOIN dept\_manager m  ON d.dept\_no = m.dept\_no  LEFT JOIN employees e  ON m.emp\_no = e.emp\_no |

Создадим функцию:

|  |
| --- |
| delimiter //  drop function if exists emp\_dept\_id //  --  -- returns the department id of a given employee  --  create function emp\_dept\_id( employee\_id int )  returns char(4)  reads sql data  begin  declare max\_date date;  set max\_date = (  select  max(from\_date)  from  dept\_emp  where  emp\_no = employee\_id  );  set @max\_date=max\_date;  return (  select  dept\_no  from  dept\_emp  where  emp\_no = employee\_id  and  from\_date = max\_date  limit 1  );  end // |

# Практическое задание

1. Создать на основе запросов, которые вы сделали в ДЗ к уроку 3, VIEW.
2. Создать функцию, которая найдет менеджера по имени и фамилии.
3. Создать триггер, который при добавлении нового сотрудника будет выплачивать ему вступительный бонус, занося запись в таблицу salary.

# Дополнительные материалы

1. <https://www.w3schools.com/sql/>

# Используемая литература

Для подготовки данного методического пособия были использованы следующие ресурсы:

1. <http://dev.mysql.com/doc/refman/5.7/en/create-view.html>
2. <http://www.mysql.ru/docs/gruber/mg20.html>
3. <http://dev.mysql.com/doc/refman/5.7/en/create-procedure.html>
4. [http://dev.mysql.com/doc/refman/5.7/en/](http://dev.mysql.com/doc/refman/5.7/en/trigger-syntax.html)
5. [-syntax.html](http://dev.mysql.com/doc/refman/5.7/en/trigger-syntax.html)

SQL-команды DML

Научимся, пользуясь SQL, манипулировать данными из нашей БД.

[Типы команд DML](#_mnhvf1tpgu10)

[SELECT](#_fddwwjfianmc)

[Простые запросы](#_nv9rrbt7a5dj)

[DISTINCT](#_tboy6v9ga8gh)

[WHERE](#_iixsp6w3t25n)

[Операторы сравнения](#_ptwc44rlat53)

[Булевы операторы](#_w7jy6l6s1ga5)

[IN, BETWEEN, LIKE, REGEXP](#_2vtng9qyxswf)

[IN](#_lj1ufegzytnm)

[BETWEEN](#_y3jd4t5b07gs)

[LIKE](#_6ktxss8lkw06)

[REGEXP](#_i2e6z0lg5hju)

[GROUP BY и агрегирующие функции](#_t0u109qgnf95)

[Примеры запросов](#_rc9dcdkq5brz)

[HAVING](#_7heoao5jow8o)

[Примеры запросов](#_ld0bd4e9uv2q)

[ORDER BY](#_boxf0cu0n308)

[Примеры запросов](#_542n6r92bqot)

[LIMIT](#_bmxh5opzi6p)

[Примеры запросов](#_jnac608w0f4q)

[JOIN](#_8acrh72lr91q)

[Примеры запросов](#_uar597tklx4v)

[UNION](#_59xvuyni7hw1)

[Примеры запросов](#_seaws0qmfdj0)

[UPDATE](#_pvs5taz5z7xv)

[Примеры запросов](#_ajfsthupkivz)

[DELETE](#_7xq8xbbxmpac)

[Примеры запросов](#_3yeeo2v84mct)

[INSERT](#_w30dsyd7unc)

[Примеры использования](#_b5isajnkzm)

[INSERT ... SELECT](#_qnzhhwolw0ts)

[Пример запроса](#_6b48fndhmffn)

[Кавычки в MySQL](#_hi0j1uxicjtf)

[Практическая работа](#_yys42jm5cqu0)

Практическое [задание](#_l91ncnekhns)

[Дополнительные материалы](#_l3wc3oro41n4)

[Используемая литература](#_uvp6qax5r1ok)

# Типы команд DML

Data Manipulation Language (DML) (язык управления или манипулирования данными) – это семейство компьютерных языков, используемых в компьютерных программах или пользователями баз данных для получения, вставки, удаления или изменения данных в базах данных.

Функции языков DML определяются первым словом в предложении (часто называемом запросом), которое почти всегда является глаголом. В случае с SQL эти глаголы – «select» («выбрать»), «insert» («вставить»), «update» («обновить») и «delete» («удалить»). Это превращает код на языке в последовательность обязательных утверждений (команд) относительно базы данных.

Рассмотрим каждую из команд подробно.

# SELECT

Одна из самых сложных по структуре команд в MySQL. Позволяет выбрать любой срез данных из БД. Таблицы можно объединять, данные можно сортировать, объединять, группировать.

|  |
| --- |
| **SELECT  [ALL | DISTINCT | DISTINCTROW ]  [HIGH\_PRIORITY]  [MAX\_STATEMENT\_TIME = *N*]  [STRAIGHT\_JOIN]  [SQL\_SMALL\_RESULT] [SQL\_BIG\_RESULT] [SQL\_BUFFER\_RESULT]  [SQL\_CACHE | SQL\_NO\_CACHE] [SQL\_CALC\_FOUND\_ROWS]  [FROM *table\_references*  [WHERE *where\_condition*]  [GROUP BY {*col\_name* | *expr* | *position*}  [ASC | DESC], ... [WITH ROLLUP]]  [HAVING *where\_condition*]  [ORDER BY {*col\_name* | *expr* | *position*}  [ASC | DESC], ...]  [LIMIT {[*offset*,] *row\_count* | *row\_count* OFFSET *offset*}]** |

Рассмотрим теперь отдельно все возможности выборки.

Мы можем проводить вычисления, не используя вообще никаких таблиц.

|  |
| --- |
| **mysql> SELECT 1 + 1;  -> 2** |

Также можно сделать запрос в несуществующую таблицу:

|  |
| --- |
| **mysql> SELECT 1 + 1 FROM DUAL;  -> 2** |

## Простые запросы

Рассмотрим типовые запросы при выборке данных:

|  |
| --- |
| **SELECT \* FROM tbl\_name;** |

В данном контексте мы выбираем все поля (\*) из таблицы tbl\_name. Если мы хотим сделать выборку определенных полей, нам нужно указать их после оператора SELECT.

|  |
| --- |
| **SELECT field1, field2 FROM tbl\_name;** |

Поля можно объединять:

|  |
| --- |
| **SELECT field1, field2, field3 + field4 FROM tbl\_name;** |

Пользуясь оператором CONCAT, можно сцепить значения колонок в одну. Используя оператор AS, можно задать результирующей колонке специфическое имя full\_name.

|  |
| --- |
| **SELECT CONCAT(last\_name, ', ', first\_name) AS full\_name FROM mytable;** |

### DISTINCT

Если нам нужно выбрать только уникальные записи в поле, нужно использовать DISTINCT.

|  |
| --- |
| **SELECT DISTINCT field1 AS distinct\_field1, field2 FROM tbl\_name;** |

## WHERE

Теперь рассмотрим условия выборки, для этого у нас есть клауза WHERE:

|  |
| --- |
| **SELECT field1, field2 FROM tbl\_name WHERE field1 = 20;** |

Мы выбираем из таблицы tbl\_name поля field1, field2 и указываем дополнительно условие, что поле field1 должно быть равно 20.

В WHERE можно указывать любые условия из булевой алгебры, пользуясь следующим синтаксисом выражений.

### Операторы сравнения

|  |
| --- |
| **comparison\_operator: = | >= | > | <= | < | <> | !=** |

Равенство: =

Неравенство: <>, !=

Больше, меньше, больше или равно, меньше или равно: >, <, >=, <=

Приведем примеры:

|  |
| --- |
| **SELECT field1, field2 FROM tbl\_name WHERE field1 > 20;**  **SELECT field1, field2 FROM tbl\_name WHERE field1 != 20;**  **SELECT field1, field2 FROM tbl\_name WHERE field1 <= 20;** |

### Булевы операторы

|  |
| --- |
| **expr:**  **expr OR expr**  **| expr || expr**  **| expr XOR expr**  **| expr AND expr**  **| expr && expr**  **| NOT expr**  **| ! expr**  **| boolean\_primary IS [NOT] {TRUE | FALSE | UNKNOWN}**  **| boolean\_primary**  **boolean\_primary:**  **boolean\_primary IS [NOT] NULL**  **| boolean\_primary <=> predicate**  **| boolean\_primary comparison\_operator predicate**  **| boolean\_primary comparison\_operator {ALL | ANY} (subquery)**  **| predicate** |

Приведем несколько примеров:

|  |
| --- |
| **SELECT field1, field2 FROM tbl\_name WHERE field1 > 20 OR field2 !=30;**  **SELECT field1, field2 FROM tbl\_name WHERE field1 != 20 AND field2 < 10;**  **SELECT field1, field2 FROM tbl\_name WHERE field1 IS NOT 20;** |

### IN, BETWEEN, LIKE, REGEXP

|  |
| --- |
| **predicate:**  **bit\_expr [NOT] IN (subquery)**  **| bit\_expr [NOT] IN (expr [, expr] ...)**  **| bit\_expr [NOT] BETWEEN bit\_expr AND predicate**  **| bit\_expr SOUNDS LIKE bit\_expr**  **| bit\_expr [NOT] LIKE simple\_expr [ESCAPE simple\_expr]**  **| bit\_expr [NOT] REGEXP bit\_expr**  **| bit\_expr** |

### IN

Оператор IN можно использовать для сравнения с определенным множеством:

|  |
| --- |
| **SELECT field1, field2 FROM tbl\_name WHERE field1 IN (118,17,113,23,72);** |

Также можно проводить сравнение с результатом другого запроса:

|  |
| --- |
| **SELECT \* FROM table1 WHERE id IN (SELECT id FROM table2);** |

### BETWEEN

Выбирать из определенного интервала значений можно с помощью оператора BETWEEN:

|  |
| --- |
| **SELECT field1, field2 FROM tbl\_name WHERE field1 BETWEEN 117 AND 220;** |

Также во всех этих выражениях можно использовать оператор NOT, который служит логическим отрицанием:

|  |
| --- |
| **SELECT field1, field2 FROM tbl\_name WHERE field1 NOT IN (118,17,113,23,72);**  **SELECT field1, field2 FROM tbl\_name WHERE field1 NOT BETWEEN 117 AND 220;** |

### LIKE

Оператор LIKE интересен тем, что позволяет осуществлять поиск по строкам, используя примитивный синтаксис регулярных выражений SQL:

|  |  |
| --- | --- |
| **Символ** | **Описание** |
| % | Соответствует любому количеству символов, даже нулевых |
| \_ | Соответствует ровно одному символу |

Давайте приведем пример оператора LIKE в действии.

Имена питомцев, которые начинаются с «b»:

|  |
| --- |
| **mysql> SELECT \* FROM pet WHERE name LIKE 'b%'; +--------+--------+---------+------+------------+------------+ | name | owner | species | sex | birth | death | +--------+--------+---------+------+------------+------------+ | Buffy | Harold | dog | f | 1989-05-13 | NULL | | Bowser | Diane | dog | m | 1989-08-31 | 1995-07-29 | +--------+--------+---------+------+------------+------------+** |

Имена питомцев, которые заканчиваются «fy»:

|  |
| --- |
| **mysql> SELECT \* FROM pet WHERE name LIKE '%fy'; +--------+--------+---------+------+------------+-------+ | name | owner | species | sex | birth | death | +--------+--------+---------+------+------------+-------+ | Fluffy | Harold | cat | f | 1993-02-04 | NULL | | Buffy | Harold | dog | f | 1989-05-13 | NULL | +--------+--------+---------+------+------------+-------+** |

Имена питомцев, которые содержат «w»:

|  |
| --- |
| **mysql> SELECT \* FROM pet WHERE name LIKE '%w%'; +----------+-------+---------+------+------------+------------+ | name | owner | species | sex | birth | death | +----------+-------+---------+------+------------+------------+ | Claws | Gwen | cat | m | 1994-03-17 | NULL | | Bowser | Diane | dog | m | 1989-08-31 | 1995-07-29 | | Whistler | Gwen | bird | NULL | 1997-12-09 | NULL | +----------+-------+---------+------+------------+------------+** |

Ищем имена, которые состоят из пяти символов:

|  |
| --- |
| **mysql> SELECT \* FROM pet WHERE name LIKE '\_\_\_\_\_'; +-------+--------+---------+------+------------+-------+ | name | owner | species | sex | birth | death | +-------+--------+---------+------+------------+-------+ | Claws | Gwen | cat | m | 1994-03-17 | NULL | | Buffy | Harold | dog | f | 1989-05-13 | NULL | +-------+--------+---------+------+------------+-------+** |

### REGEXP

Также поддерживается синтаксис полноценных регулярных выражений.

Если нам нужен реально регистрозависимый поиск в строке, нужно использовать ключевое слово BINARY. Следующий пример будет искать строки, начинающиеся с маленькой буквы:

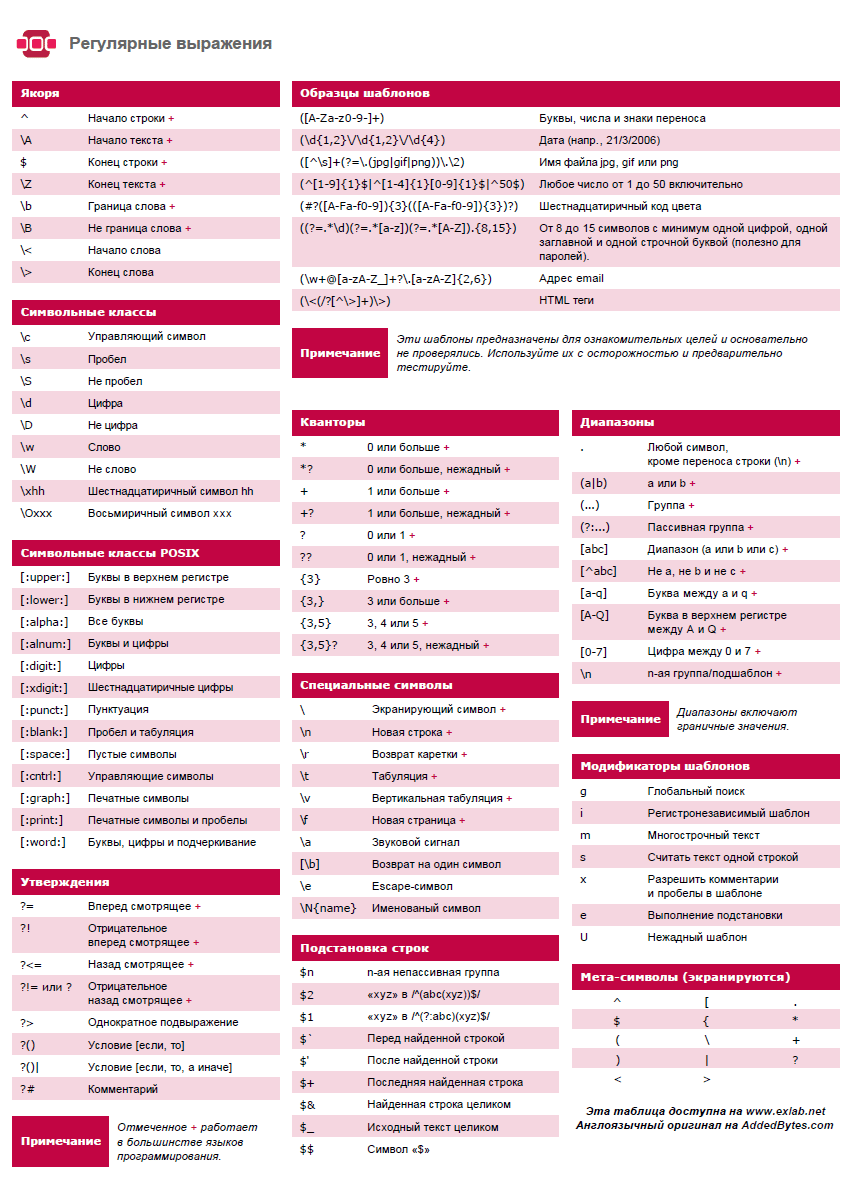
|  |
| --- |
| **mysql> SELECT \* FROM pet WHERE name REGEXP BINARY '^b';** |

В базе нет записей, начинающихся с маленькой буквы.

Поищем питомца, имя которого заканчивается на «fy»:

|  |
| --- |
| **mysql> SELECT \* FROM pet WHERE name REGEXP 'fy$'; +--------+--------+---------+------+------------+-------+ | name | owner | species | sex | birth | death | +--------+--------+---------+------+------------+-------+ | Fluffy | Harold | cat | f | 1993-02-04 | NULL | | Buffy | Harold | dog | f | 1989-05-13 | NULL | +--------+--------+---------+------+------------+-------+** |

Подробнее о синтаксисе регулярных выражений можно прочитать в памятке.



## GROUP BY и агрегирующие функции

Очень часто (например, при формировании отчетности) перед нами встает задача группировки и агрегации данных. Для этого у нас есть GROUP BY и функции-агрегаторы.

|  |  |
| --- | --- |
| **Имя** | **Описание** |
| AVG() | Среднее значение |
| COUNT() | Количество записей |
| MAX() | Максимальное значение |
| MIN() | Минимальное значение |
| SUM() | Сумма значений |
| STD(expr)  STDDEV(expr) | Возвращает среднеквадратичное отклонение значения в аргументе expr. Эта функция является расширением ANSI SQL. Форма STDDEV() обеспечивает совместимость с Oracle. |

Самый большой практический интерес вызывают первые пять функций.

Функции-агрегаторы используются только совместно с оператором GROUP BY.

### Примеры запросов

Вычислить максимальную зарплату у сотрудников:

|  |
| --- |
| **SELECT MAX(salary), CONCAT(last\_name, ', ', first\_name) AS full\_name**  **FROM tbl\_employee**  **GROUP BY full\_name;** |

Вычислить среднюю зарплату у сотрудников:

|  |
| --- |
| **SELECT AVG(salary), CONCAT(last\_name, ', ', first\_name) AS full\_name**  **FROM tbl\_employee**  **GROUP BY full\_name;** |

Вычислить максимальную зарплату у сотрудников из отдела IT:

|  |
| --- |
| **SELECT MAX(salary), CONCAT(last\_name, ', ', first\_name) AS full\_name**  **FROM tbl\_employee**  **WHERE department = "IT"**  **GROUP BY full\_name;** |

## HAVING

Данный оператор очень часто путают с оператором WHERE, ведь если его использовать без GROUP BY, то работают они одинаково. Но смысл этого оператора в том, что он позволяет фильтровать записи после того, как у нас выполнилась выборка и группировка.

Давайте вернемся к спецификации запроса SELECT:

|  |
| --- |
| **SELECT  [ALL | DISTINCT | DISTINCTROW ]  [HIGH\_PRIORITY]  [MAX\_STATEMENT\_TIME = *N*]  [STRAIGHT\_JOIN]  [SQL\_SMALL\_RESULT] [SQL\_BIG\_RESULT] [SQL\_BUFFER\_RESULT]  [SQL\_CACHE | SQL\_NO\_CACHE] [SQL\_CALC\_FOUND\_ROWS]  [FROM *table\_references*  [WHERE *where\_condition*]  [GROUP BY {*col\_name* | *expr* | *position*}  [ASC | DESC], ... [WITH ROLLUP]]  [HAVING *where\_condition*]** |

Как вы видите, оператор HAVING обладает такой же спецификой, что и оператор WHERE (where\_condition), другими словами, эти операторы имеют абсолютно одинаковую семантику, просто выполняются в разные моменты времени.

Не нужно использовать оператор HAVING вместо оператора WHERE, это противоречит стандарту SQL.

### Примеры запросов

Выбрать работников из отдела IT с максимальной зарплатой от 10000.

|  |
| --- |
| **SELECT MAX(salary) AS max\_salary, CONCAT(last\_name, ', ', first\_name) AS full\_name**  **FROM tbl\_employee**  **WHERE department = "IT"**  **GROUP BY full\_name**  **HAVING max\_salary > 10000;** |

Выбрать всех сотрудников, посчитать среднюю зарплату, найти сотрудников у которых зарплата ниже 3000.

|  |
| --- |
| **SELECT AVG(salary) AS avg\_salary, CONCAT(last\_name, ', ', first\_name) AS full\_name**  **FROM tbl\_employee**  **GROUP BY full\_name**  **HAVING avg\_salary < 3000;** |

## 

## ORDER BY

Сортировка полученных данных по определенной колонке. Есть два вида сортировки – ASC и DESC (ascending, descending order – сортировка по возрастанию / убыванию). Если явно не указывать вид сортировки, используется ASC.

### Примеры запросов

Вычислить среднюю зарплату сотрудников, найти зарплаты более 10000 и отсортировать по возрастанию.

|  |
| --- |
| **SELECT AVG(salary) AS avg\_salary, CONCAT(last\_name, ', ', first\_name) AS full\_name**  **FROM tbl\_employee**  **GROUP BY full\_name**  **HAVING avg\_salary > 10000**  **ORDER BY avg\_salary;** |

Вычислить максимальную зарплату у сотрудников из отдела IT, отобрать зарплаты больше 10000, отсортировать по убыванию.

|  |
| --- |
| **SELECT MAX(salary) AS max\_salary, CONCAT(last\_name, ', ', first\_name) AS full\_name**  **FROM tbl\_employee**  **WHERE department = "IT"**  **GROUP BY full\_name**  **HAVING max\_salary > 10000**  **ORDER BY max\_salary DESC;** |

## LIMIT

Оператор используется для ограничения количества выбираемых данных или выборки определенного среза данных со смещением.  
LIMIT примечателен тем, что ограничение выборки происходит в самом конце выполнения запроса. MySQL выполняет сам запрос, вернет его полный результат, и только после этого будут отброшены лишние данные. По этим причинам нужно максимально ограничить диапазон выбираемых данных, чтобы не было падения производительности СУБД.

### Примеры запросов

Выбрать сотрудника из отдела IT с максимальной зарплатой.

|  |
| --- |
| **SELECT MAX(salary) AS max\_salary, CONCAT(last\_name, ', ', first\_name) AS full\_name**  **FROM tbl\_employee**  **WHERE department = "IT"**  **GROUP BY full\_name**  **ORDER BY max\_salary DESC**  **LIMIT 1;** |

Выбрать сотрудников c 5 по 10 из отдела IT с максимальной зарплатой:

|  |
| --- |
| **SELECT MAX(salary) AS max\_salary, CONCAT(last\_name, ', ', first\_name) AS full\_name**  **FROM tbl\_employee**  **WHERE department = "IT"**  **GROUP BY full\_name**  **ORDER BY max\_salary DESC**  **LIMIT 4, 5;** |

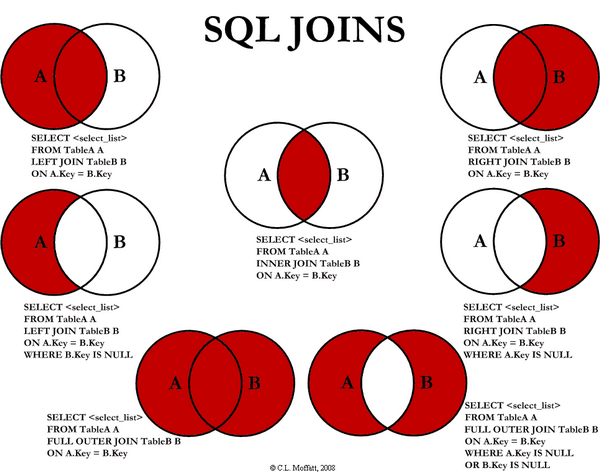
## JOIN

JOIN реализует в себе возможности реляционной алгебры: таблицы можно объединять.

Многие разработчики считают эту тему сложной для понимания, но на самом деле, прежде чем рассматривать синтаксис JOIN, нужно хорошо разобраться, как объединяются таблицы, на диаграммах.

Существуют следующие виды JOIN:

|  |
| --- |
| **join\_table**:  **table\_reference** [INNER | CROSS] JOIN **table\_factor** [**join\_condition**]  | **table\_reference** STRAIGHT\_JOIN **table\_factor**  | **table\_reference** STRAIGHT\_JOIN **table\_factor** ON **conditional\_expr**  | **table\_reference** {LEFT|RIGHT} [OUTER] JOIN **table\_reference** **join\_condition**  | **table\_reference** NATURAL [{LEFT|RIGHT} [OUTER]] JOIN **table\_factor** |

Нагляднее всего можно пояснить это при помощи диаграмм.

Для хорошего понимания JOIN нужно представлять себе два и более множеств и четко понимать, какие таблицы объединяются и в каком порядке.

### Примеры запросов

Объединение трех таблиц с помощью LEFT JOIN:

|  |
| --- |
| **SELECT \* FROM tbl\_table1 LEFT JOIN (tbl\_table2, tbl\_table3, tbl\_table4) ON (tbl\_table2.a=tbl\_table1.a AND tbl\_table3.b=tbl\_table1.b AND tbl\_table4.c=tbl\_table1.c)** |

Можно объединять множества между собой, а не с первой таблицей. При этом объединение происходит с результирующим множеством:

|  |
| --- |
| **SELECT \* FROM Table1 t1 LEFT OUTER JOIN Table2 t2**  **ON t1.fk = t2.id**  **LEFT OUTER JOIN Table3 t3**  **ON t1.fk\_table3 = t3.id;** |

Чтобы понимать, что при этом происходит, давайте снова представим это в виде диаграммы множеств:

# 

При объединении любого количества множеств в любом порядке нужно всегда представлять диаграмму пересечения множеств.

## UNION

В языке SQL ключевое слово UNION применяется для объединения результатов двух SQL-запросов в единую таблицу, состоящую из схожих строк.

Существуют два основных правила, регламентирующие порядок использования оператора UNION:

* Число и порядок извлекаемых столбцов должны совпадать во всех объединяемых запросах.
* Типы данных в соответствующих столбцах должны быть совместимы.

|  |
| --- |
| **SELECT …**  **UNION [ALL | DISTINCT] SELECT …**  **[UNION [ALL | DISTINCT] SELECT ...]** |

### Примеры запросов

Даны две таблицы:

|  |  |
| --- | --- |
| **person** | **amount** |
| Иван | 1000 |
| Алексей | 2000 |
| Сергей | 5000 |

|  |  |
| --- | --- |
| **person** | **amount** |
| Иван | 2000 |
| Алексей | 2000 |
| Петр | 35000 |

При выполнении следующего запроса

|  |
| --- |
| **(SELECT \* FROM sales2005)**  **UNION**  **(SELECT \* FROM sales2006);** |

получается результирующий набор, однако порядок строк может произвольно меняться, поскольку ключевое выражение ORDER BY не было использовано:

|  |  |
| --- | --- |
| **person** | **amount** |
| Иван | 1000 |
| Алексей | 2000 |
| Иван | 2000 |
| Сергей | 5000 |
| Петр | 35000 |

В результате отобразятся две строки с Иваном, так как они различаются значениями в столбцах. Но при этом в результате присутствует лишь одна строка с Алексеем, поскольку значения в столбцах полностью совпадают.

Применение UNION ALL дает другой результат, так как дубликаты не скрываются. Выполнение запроса

|  |
| --- |
| **(SELECT \* FROM sales2005)**  **UNION ALL**  **(SELECT \* FROM sales2006);** |

даст следующий результат, выводимый без упорядочивания ввиду отсутствия выражения ORDER BY:

|  |  |
| --- | --- |
| **person** | **amount** |
| Иван | 1000 |
| Иван | 2000 |
| Алексей | 2000 |
| Алексей | 2000 |
| Сергей | 5000 |
| Петр | 35000 |

# UPDATE

Позволяет обновить данные в таблице:

|  |
| --- |
| **UPDATE [LOW\_PRIORITY] [IGNORE] table\_reference**  **SET col\_name1={expr1|DEFAULT} [, col\_name2={expr2|DEFAULT}] ...**  **[WHERE where\_condition]**  **[ORDER BY ...]**  **[LIMIT row\_count]** |

### Примеры запросов

Увеличить зарплату сотрудникам отдела IT на 100:

|  |
| --- |
| **UPDATE tbl\_employee emp**  **SET emp.salary = emp.salary + 100**  **WHERE department = "IT";** |

Обновить данные из нескольких таблиц:

|  |
| --- |
| **UPDATE items, month SET items.price = month.price**  **WHERE items.id = month.id;** |

# DELETE

Данный оператор используется для удаления записей из таблицы. Синтаксис оператора следующий:

|  |
| --- |
| **DELETE [LOW\_PRIORITY] [QUICK] [IGNORE] FROM tbl\_name**  **[WHERE where\_condition]**  **[ORDER BY ...]**  **[LIMIT row\_count]** |

Работу WHERE, ORDER BY и LIMIT мы подробно рассмотрели в разделе, посвященном SELECT. В этой команде данные операторы работают абсолютно аналогично.

### Примеры запросов

Удаляем всех сотрудников из отдела IT:

|  |
| --- |
| **DELETE FROM tbl\_employee**  **WHERE department = "IT";** |

# INSERT

Позволяет добавить данные в таблицу. Можно добавить одну или несколько записей.

|  |
| --- |
| **INSERT [LOW\_PRIORITY | DELAYED | HIGH\_PRIORITY] [IGNORE]  [INTO] *tbl\_name*  [(*col\_name*,...)]  {VALUES | VALUE} ({*expr* | DEFAULT},...),(...),...** |

Или:

|  |
| --- |
| **INSERT [LOW\_PRIORITY | DELAYED | HIGH\_PRIORITY] [IGNORE]  [INTO] *tbl\_name*  SET *col\_name*={*expr* | DEFAULT}, ...** |

[LOW\_PRIORITY | DELAYED | HIGH\_PRIORITY] приоритет, с которым запись будет добавлена. Конкурентные записи поддерживаются только движком MyISAM (подробнее о движках на уроке 7).

Если вы используете LOW\_PRIORITY, запись может не добавляться значительный промежуток времени, а точнее до тех пор, пока другие клиенты осуществляют операции чтения.

По умолчанию записи добавляются в режиме LOW\_PRIORITY.

[INTO] tbl\_name – в какую таблицу (tbl\_name) будет осуществляться добавление записи.

{VALUES | VALUE} – какое значение или значения будут добавляться в таблицу.

### Примеры использования

|  |
| --- |
| **INSERT INTO *tbl\_name* (a,b,c) VALUES(1,2,3),(4,5,6),(7,8,9);** |

## 

|  |
| --- |
| **INSERT INTO *tbl\_name* (a,b,c) VALUE(1,2,3);** |

## 

## INSERT ... SELECT

INSERT можно делать на основе SELECT – добавить данные в таблицу из выборки.

|  |
| --- |
| **INSERT [LOW\_PRIORITY | HIGH\_PRIORITY] [IGNORE]  [INTO] *tbl\_name*  [PARTITION (*partition\_name*,...)]  [(*col\_name*,...)]  SELECT ...  [ ON DUPLICATE KEY UPDATE  *col\_name*=*expr*  [, *col\_name*=*expr*] ... ]** |

### Пример запроса

|  |
| --- |
| **INSERT INTO tbl\_temp2 (fld\_id)  SELECT tbl\_temp1.fld\_order\_id  FROM tbl\_temp1 WHERE tbl\_temp1.fld\_order\_id > 100;** |

# Кавычки в MySQL

Давайте сначала поговорим о том, что такое строки в MySQL. Строка представляет собой последовательность символов, заключенных либо в одинарные кавычки (‘ ‘) - апострофы, либо в двойные кавычки (“ “). При использовании диалекта ANSI SQL допустимы только одинарные кавычки. Например:

'a string'  
"another string"

Внутри строки некоторые последовательности символов имеют специальное назначение. Каждая из этих последовательностей начинается обратным слешем ('\'), известным как escape-символ или символ перехода. MySQL распознает следующие escape-последовательности:

|  |  |
| --- | --- |
| \0 | Символ 0 (NUL) в ASCII-коде. |
| \' | Символ одиночной кавычки ('). |
| \" | Символ двойной кавычки ("). |
| \b | Возврат на один символ. |
| \n | Символ новой строки (перевода строки). |
| \r | Символ перевода каретки. |
| \t | Символ табуляции. |
| \z | Символ (Control-Z) таблицы ASCII(26). Данный символ можно закодировать, чтобы обойти проблему, заключающуюся в том, что под Windows ASCII(26) означает конец файла (проблемы возникают при использовании ASCII(26) в выражении mysql database < filename). |
| \\ | Символ обратного слеша. |
| \% | Символ процентов '%'. Используется для поиска копий литерала '%' в контекстах, где выражение '%' в противном случае интерпретировалось бы как групповой символ. |
| \\_ | Символ подчеркивания '\_'. Используется для поиска копий литерала '\_' в контекстах, где выражение '\_' в противном случае интерпретировалось бы как групповой символ. |

Обратите внимание, что при использовании '\%' или '\\_' в контекстах некоторых строк будут возвращаться значения строк '\%' и '\\_', а не '%' и '\_'.

Существует несколько способов включить кавычки в строку:

* Одиночная кавычка (апостроф) ' внутри строки, заключенной в кавычки '**'**', может быть записана как '**''**'.
* Двойная кавычка '**"**' внутри строки, заключенной в двойные кавычки '**"**', может быть записана как '**""**'.
* Можно предварить символ кавычки символом экранирования ('\').
* Для символа '**'**' внутри строки, заключенной в двойные кавычки, не требуется специальной обработки; его также не требуется дублировать или предварять обратным слешем. Точно так же не требует специальной обработки двойная кавычка '**"**' внутри строки, заключенной в одинарные кавычки '**'**'.

Ниже показаны возможные варианты применения кавычек и escape-символа на примерах выполнения команды SELECT:

|  |
| --- |
| mysql> SELECT 'hello', '"hello"', '""hello""', 'hel''lo', '\'hello'; +-------+---------+-----------+--------+--------+ | hello | "hello" | ""hello"" | hel'lo | 'hello | +-------+---------+-----------+--------+--------+ mysql> SELECT "hello", "'hello'", "''hello''", "hel""lo", "\"hello"; +-------+---------+-----------+--------+--------+ | hello | 'hello' | ''hello'' | hel"lo | "hello | +-------+---------+-----------+--------+--------+ mysql> SELECT "This\nIs\nFour\nlines"; +--------------------+ | This Is Four lines | +--------------------+ |

Если необходимо вставить в строку двоичные данные (такие как BLOB), следующие символы должны быть представлены как escape-последовательности:

|  |  |
| --- | --- |
| NUL | ASCII 0. Необходимо представлять в виде '\0' (обратный слеш и символ ASCII '0'). |
| \ | ASCII 92, обратный слеш. Представляется как '\\'. |
| ' | ASCII 39, единичная кавычка. Представляется как '**\'**'. |
| " | ASCII 34, двойная кавычка. Представляется как '**\"**'. |

Также нужно отдельно упомянуть, для чего используется символ `.

Можно встретить следующий запрос:

|  |
| --- |
| CREATE TABLE `pracownik`  (  `id` TINYINT auto\_increment,  `name` VARCHAR,  ) ENGINE=InnoDB; |

В данной ситуации, если не использовать символ обратной кавычки, нельзя будет брать ключевые слова mysql в качестве имен (например date, min, max, order и др.).

# Практическая работа

1. Создаем и наполняем базу данных «Сотрудники».
2. Разбираемся с тем, как устроен DUMP и команды на создание таблиц.
3. Практикуем запросы – выбираем данные:
   * добавляем сотрудника в отдел;

|  |
| --- |
| **INSERT INTO `employees`**  **VALUES**  **(10001,'1953-09-02','Georgi','Facello','M','1986-06-26'),**  **(10002,'1964-06-02','Bezalel','Simmel','F','1985-11-21');** |

* + удаляем сотрудника из отдела;

|  |
| --- |
| **DELETE FROM `employees`**  **WHERE id = 10001;** |

* + сколько денег было выплачено за любой год;

|  |
| --- |
| **SELECT \* FROM `salaries` WHERE `from\_date` = '1986-06-26'** |

* + находим менеджеров каждого отдела (SELECT, JOIN);

|  |
| --- |
| **SELECT d.\*, e.first\_name, e.last\_name**  **FROM departments d**  **LEFT JOIN dept\_manager m**  **ON d.dept\_no = m.dept\_no**  **LEFT JOIN employees e**  **ON m.emp\_no = e.emp\_no** |

* + смотрим максимальную зарплату менеджеров;

|  |
| --- |
| **SELECT e.first\_name, e.last\_name, MAX(s.salary) max\_salary**  **FROM dept\_manager m**  **LEFT JOIN employees e ON m.emp\_no = e.emp\_no**  **LEFT JOIN salaries s on m.emp\_no = s.emp\_no**  **GROUP BY e.first\_name, e.last\_name**  **ORDER BY max\_salary DESC;** |

* + на усмотрение преподавателя можно сделать еще несколько выборок в любом интересном разрезе.

# Практическое задание

База данных «Страны и города мира»:

1. Сделать запрос, в котором мы выберем все данные о городе – регион, страна.
2. Выбрать все города из Московской области.

База данных «Сотрудники»:

1. Выбрать среднюю зарплату по отделам.
2. Выбрать максимальную зарплату у сотрудника.
3. Удалить одного сотрудника, у которого максимальная зарплата.
4. Посчитать количество сотрудников во всех отделах.
5. Найти количество сотрудников в отделах и посмотреть, сколько всего денег получает отдел.

# Дополнительные материалы

1. <https://www.w3schools.com/sql/>

# Используемая литература

Для подготовки данного методического пособия были использованы следующие ресурсы:

1. <http://dev.mysql.com/doc/refman/5.7/en/insert.html>
2. <http://dev.mysql.com/doc/refman/5.7/en/select.html>
3. <http://dev.mysql.com/doc/refman/5.7/en/expressions.html>
4. <http://dev.mysql.com/doc/refman/5.7/en/pattern-matching.html>
5. <http://www.exlab.net/tools/sheets/regexp.html>
6. <http://dev.mysql.com/doc/refman/5.7/en/union.html>
7. <https://ru.wikipedia.org/wiki/Union_(SQL)>
8. <http://dev.mysql.com/doc/refman/5.7/en/delete.html>
9. <http://dev.mysql.com/doc/refman/5.7/en/update.html>
10. <https://launchpad.net/test-db/>
11. <http://www.exlab.net/tools/sheets/regexp.html>

SQL - команды DDL

Научимся, пользуясь SQL, создавать, изменять и удалять БД и объекты БД.

[Data Definition Language (DDL)](#_t755nhj7uzcu)

[Создание базы данных](#_sl9dqmqh1k1y)

[Создание таблицы](#_sk7b8g8vf7hy)

[Объявление полей в БД](#_brk5o6dvdqjt)

[Выбор типа данных для колонки](#_b5i5y0l9tq1j)

[Объявление ключей и индексов](#_w6j8rhf942qv)

[Внешний ключ](#_24fyxakoa54s)

[Alter table](#_xkyl14lhd8k5)

[Drop](#_hvzclo73gzoy)

[Практическая работа](#_n78lhk61oy4o)

Практическое [задание](#_g5fwre2xbar7)

[Дополнительные материалы](#_hs9imvnirnvf)

[Используемая литература](#_gymsq8ljtflr)

# Data Definition Language (DDL)

Data Definition Language (DDL) – это язык, с помощью которого можно описать структуру данных в БД. Пользуясь DDL, можно создать и удалить БД, добавить таблицу с любой структурой и связями, изменить и удалить таблицу.

Команды делятся на три вида: create, alter, drop. С помощью этих команд можно провести любые операции с БД и её составляющими.

Далее мы обзорно рассмотрим синтаксис всех трех команд.

# Создание базы данных

Для создания БД используется следующая команда:

|  |
| --- |
| CREATE DATABASE [IF NOT EXISTS] db\_name |

Команда создает базу данных с именем «db\_name».

Ключ [IF NOT EXISTS] используется для проверки на существование базы данных. Если ключ не указан и база данных существует, это приведет к возникновению ошибки.

# Создание таблицы

|  |
| --- |
| CREATE [TEMPORARY] TABLE [IF NOT EXISTS] tbl\_name  (create\_definition,...)  [table\_options]  [partition\_options]  CREATE [TEMPORARY] TABLE [IF NOT EXISTS] tbl\_name  [(create\_definition,...)]  [table\_options]  [partition\_options]  [IGNORE | REPLACE]  [AS] query\_expression |

Команда позволяет создать таблицу с именем «tbl\_name». Также как и в случае создания БД, есть флаг проверки [IF NOT EXISTS], который позволяет проверить на существование в БД таблицы с аналогичным именем.

[TEMPORARY] позволяет создать таблицу только на текущую сессию. Сразу после завершения сессии таблица будет удалена.

## 

## 

## Объявление полей в БД

|  |
| --- |
| column\_definition:  data\_type [NOT NULL | NULL] [DEFAULT default\_value]  [AUTO\_INCREMENT] [UNIQUE [KEY] | [PRIMARY] KEY]  [COMMENT 'string']  [COLUMN\_FORMAT {FIXED|DYNAMIC|DEFAULT}]  [STORAGE {DISK|MEMORY|DEFAULT}]  [reference\_definition]  | data\_type [GENERATED ALWAYS] AS (expression)  [VIRTUAL | STORED] [UNIQUE [KEY]] [COMMENT comment]  [NOT NULL | NULL] [[PRIMARY] KEY]  data\_type:  BIT[(length)]  | TINYINT[(length)] [UNSIGNED] [ZEROFILL]  | SMALLINT[(length)] [UNSIGNED] [ZEROFILL]  | MEDIUMINT[(length)] [UNSIGNED] [ZEROFILL]  | INT[(length)] [UNSIGNED] [ZEROFILL]  | INTEGER[(length)] [UNSIGNED] [ZEROFILL]  | BIGINT[(length)] [UNSIGNED] [ZEROFILL]  | REAL[(length,decimals)] [UNSIGNED] [ZEROFILL]  | DOUBLE[(length,decimals)] [UNSIGNED] [ZEROFILL]  | FLOAT[(length,decimals)] [UNSIGNED] [ZEROFILL]  | DECIMAL[(length[,decimals])] [UNSIGNED] [ZEROFILL]  | NUMERIC[(length[,decimals])] [UNSIGNED] [ZEROFILL]  | DATE  | TIME[(fsp)]  | TIMESTAMP[(fsp)]  | DATETIME[(fsp)]  | YEAR  | CHAR[(length)] [BINARY]  [CHARACTER SET charset\_name] [COLLATE collation\_name]  | VARCHAR(length) [BINARY]  [CHARACTER SET charset\_name] [COLLATE collation\_name]  | BINARY[(length)]  | VARBINARY(length)  | TINYBLOB  | BLOB  | MEDIUMBLOB  | LONGBLOB  | TINYTEXT [BINARY]  [CHARACTER SET charset\_name] [COLLATE collation\_name]  | TEXT [BINARY]  [CHARACTER SET charset\_name] [COLLATE collation\_name]  | MEDIUMTEXT [BINARY]  [CHARACTER SET charset\_name] [COLLATE collation\_name]  | LONGTEXT [BINARY]  [CHARACTER SET charset\_name] [COLLATE collation\_name]  | ENUM(value1,value2,value3,...)  [CHARACTER SET charset\_name] [COLLATE collation\_name]  | SET(value1,value2,value3,...)  [CHARACTER SET charset\_name] [COLLATE collation\_name]  | JSON  | spatial\_type |

При объявлении столбца объявляется тип данных, а также ряд дополнительных опций data\_type [NOT NULL | NULL] [DEFAULT default\_value].

[NOT NULL | NULL] используется для проверки в колонке нулевого/ненулевого значения. При добавлении данных поле может быть обязательным, для этого мы должны использовать флаг [NOT NULL].

[DEFAULT default\_value] используется для задания полю значения по умолчанию.

При объявлении полей в БД мы можем сразу присвоить значение ключевым полям – [UNIQUE [KEY] | [PRIMARY] KEY, также поле можно создать с опцией [AUTO\_INCREMENT], что позволит каждый раз при добавлении новых данных увеличивать значение поля на единицу. Используется для отслеживания уникальности первичного ключа.

[COMMENT 'string'] позволяет написать комментарий к полю в БД. Очень удобно просматривать комментарии при работе с БД. Комментировать необходимо действительно важные значения.

[STORAGE {DISK|MEMORY|DEFAULT}] определяет носитель, куда будут записываться данные, – на диск или в оперативную память.

### Выбор типа данных для колонки

Типы данных мы рассматривали в предыдущем уроке, сегодня надо обратить внимание только на синтаксис DDL.

## Объявление ключей и индексов

|  |
| --- |
| CREATE [TEMPORARY] TABLE [IF NOT EXISTS] tbl\_name  { LIKE old\_tbl\_name | (LIKE old\_tbl\_name) }  create\_definition:  col\_name column\_definition  | [CONSTRAINT [symbol]] PRIMARY KEY [index\_type] (index\_col\_name,...)  [index\_option] ...  | {INDEX|KEY} [index\_name] [index\_type] (index\_col\_name,...)  [index\_option] ...  | [CONSTRAINT [symbol]] UNIQUE [INDEX|KEY]  [index\_name] [index\_type] (index\_col\_name,...)  [index\_option] ...  | {FULLTEXT|SPATIAL} [INDEX|KEY] [index\_name] (index\_col\_name,...)  [index\_option] ...  | [CONSTRAINT [symbol]] FOREIGN KEY  [index\_name] (index\_col\_name,...) reference\_definition  | CHECK (expr)  index\_col\_name:  col\_name [(length)] [ASC | DESC]  index\_type:  USING {BTREE | HASH}  reference\_definition:  REFERENCES tbl\_name (index\_col\_name,...)  [MATCH FULL | MATCH PARTIAL | MATCH SIMPLE]  [ON DELETE reference\_option]  [ON UPDATE reference\_option]  reference\_option:  RESTRICT | CASCADE | SET NULL | NO ACTION |

При объявлении ключа мы должны задать его имя [CONSTRAINT [symbol]], далее объявляется тип ключа – PRIMARY KEY | UNIQUE [INDEX|KEY] | FOREIGN KEY.

Так же можно объявить индекс.

|  |
| --- |
| {INDEX|KEY} [index\_name] [index\_type] (index\_col\_name,...)  index\_type:  USING {BTREE | HASH} |

Индексы используются для быстрого поиска, мы это разбирали в предыдущем уроке.

## Внешний ключ

|  |
| --- |
| [CONSTRAINT [symbol]] FOREIGN KEY  [index\_name] (index\_col\_name,...) reference\_definition  reference\_definition:  REFERENCES tbl\_name (index\_col\_name,...)  [ON DELETE reference\_option]  [ON UPDATE reference\_option]  reference\_option:  RESTRICT | CASCADE | SET NULL | NO ACTION |

Чтобы внешний ключ был создан корректно, необходимо, чтобы тип данных колонки в таблице совпадал с типом данных колонки, на которую этот ключ указывает.

Объявление самого внешнего ключа не отличается от объявления других ключей, только в дополнение к ним имеется ряд опций.

REFERENCES tbl\_name (index\_col\_name, ...) – указывает, на какую таблицу ссылается внешний ключ

ON DELETE, ON UPDATE – операции, которые выполняются при удалении или изменении связанной записи – для этого существуют опции RESTRICT | CASCADE | SET NULL | NO ACTION – выбросить ошибку, удалить связанные данные, установить данные в значение NULL, не выполнять никаких действий.

Благодаря внешнему ключу контролируется целостность связанных данных.

# Alter table

Данная команда позволяет изменить структуру таблицы – добавить, изменить и удалить колонку в таблице.

|  |
| --- |
| ALTER [IGNORE] TABLE ***tbl\_name***  [***alter\_specification*** [, ***alter\_specification***] ...]  [***partition\_options***] ***alter\_specification***:  ***table\_options***  | ADD [COLUMN] ***col\_name*** ***column\_definition***  [FIRST | AFTER ***col\_name*** ]  | ADD [COLUMN] (***col\_name*** ***column\_definition***,...)  | ADD {INDEX|KEY} [***index\_name***]  [***index\_type***] (***index\_col\_name***,...) [***index\_option***] ...  | ADD [CONSTRAINT [***symbol***]] PRIMARY KEY  [***index\_type***] (***index\_col\_name***,...) [***index\_option***] ...  | ADD [CONSTRAINT [***symbol***]]  UNIQUE [INDEX|KEY] [***index\_name***]  [***index\_type***] (***index\_col\_name***,...) [***index\_option***] ...  | ADD FULLTEXT [INDEX|KEY] [***index\_name***]  (***index\_col\_name***,...) [***index\_option***] ...  | ADD SPATIAL [INDEX|KEY] [***index\_name***]  (***index\_col\_name***,...) [***index\_option***] ...  | ADD [CONSTRAINT [***symbol***]]  FOREIGN KEY [***index\_name***] (***index\_col\_name***,...)  ***reference\_definition***  | ALGORITHM [=] {DEFAULT|INPLACE|COPY}  | ALTER [COLUMN] ***col\_name*** {SET DEFAULT ***literal*** | DROP DEFAULT}  | CHANGE [COLUMN] ***old\_col\_name*** ***new\_col\_name*** ***column\_definition***  [FIRST|AFTER ***col\_name***]  | LOCK [=] {DEFAULT|NONE|SHARED|EXCLUSIVE}  | MODIFY [COLUMN] ***col\_name*** ***column\_definitiondrop***  [FIRST | AFTER ***col\_name***]  | DROP [COLUMN] ***col\_name - удаление***  | DROP PRIMARY KEY  | DROP {INDEX|KEY} ***index\_name***  | DROP FOREIGN KEY ***fk\_symbol***  | DISABLE KEYS - включает и выключает внешние ключи  | ENABLE KEYSrename  | RENAME [TO|AS] ***new\_tbl\_name - переименовать***   | RENAME {INDEX|KEY} ***old\_index\_name*** TO ***new\_index\_name***  | ORDER BY ***col\_name*** [, ***col\_name***] … - сортировка по названию  | CONVERT TO CHARACTER SET ***charset\_name*** [COLLATE ***collation\_name***]  | [DEFAULT] CHARACTER SET [=] ***charset\_name*** [COLLATE [=] ***collation\_name***]  | DISCARD TABLESPACE  | IMPORT TABLESPACE  | FORCE  | {WITHOUT|WITH} VALIDATION  | ADD PARTITION (***partition\_definition***)  | DROP PARTITION ***partition\_names***  | DISCARD PARTITION {***partition\_names*** | ALL} TABLESPACE  | IMPORT PARTITION {***partition\_names*** | ALL} TABLESPACE  | TRUNCATE PARTITION {***partition\_names*** | ALL}  | COALESCE PARTITION ***number***  | REORGANIZE PARTITION ***partition\_names*** INTO (***partition\_definitions***)  | EXCHANGE PARTITION ***partition\_name*** WITH TABLE ***tbl\_name*** [{WITH|WITHOUT} VALIDATION]  | ANALYZE PARTITION {***partition\_names*** | ALL}  | CHECK PARTITION {***partition\_names*** | ALL}  | OPTIMIZE PARTITION {***partition\_names*** | ALL}  | REBUILD PARTITION {***partition\_names*** | ALL}  | REPAIR PARTITION {***partition\_names*** | ALL}  | REMOVE PARTITIONING  | UPGRADE PARTITIONING ***index\_col\_name***:  ***col\_name*** [(***length***)] [ASC | DESC] ***index\_type***:  USING {BTREE | HASH} ***index\_option***:  KEY\_BLOCK\_SIZE [=] ***value***  | ***index\_type***  | WITH PARSER ***parser\_name***  | COMMENT '***string***' ***table\_options***:  ***table\_option*** [[,] ***table\_option***] ... (see [CREATE TABLE](http://dev.mysql.com/doc/refman/5.7/en/create-table.html) options) |

Как видно из спецификации команды, мы можем осуществить любые операции по изменению таблицы, пользуясь данной командой.

Команда может выступать в нескольких ипостасях в зависимости от опций:

* ADD – добавить в таблицу колонку, ключ, индекс.
* CHANGE – изменить имя колонки.
* DROP – удалить колонку, ключ, индекс.
* RENAME – сменить имя таблице.
* CONVERT TO CHARACTER SET – сменить кодировку.

Остальные опции используются для манипуляций механизмами, которые мы рассмотрим в других уроках.

# Drop

Синтаксис для удаления БД.

|  |
| --- |
| DROP {DATABASE | SCHEMA} [IF EXISTS] ***db\_name*** |

Синтаксис для удаления таблицы.

|  |
| --- |
| DROP [TEMPORARY] TABLE [IF EXISTS]  ***tbl\_name*** [, ***tbl\_name***] ...  [RESTRICT | CASCADE] |

Оператор DROP используется для удаления БД или таблицы. Синтаксис очень простой.

[IF EXISTS] перед удалением проверяет наличие данной БД или таблицы.

[RESTRICT | CASCADE] удалит связанные таблицы или выдаст ошибку.

# Практическая работа

Создадим БД «Страны и города мира» с помощью готовых скриптов, приложенных к уроку. Научимся делать импорт SQL-файлов.

* Подключаемся к MySQL с помощью команды:

|  |
| --- |
| mysql -u[username] -p[password] |

* Можно использовать пользователя root, которого мы создали при установке MySQL:

|  |
| --- |
| mysql -uroot -p[password] |

* Для того, чтобы создать базу данных используется команда:

|  |
| --- |
| CREATE DATABASE geodata; |

* Выходим из MySQL shell:

|  |
| --- |
| exit; |

* Рассмотрим подробнее файлы для создания БД:
  + Tables.sql – файл, который содержит в себе структуру БД, команды DDL для создания схемы.
  + \_cities.sql, \_countries.sql, \_regions.sql – файлы необходимые для импорта данных.

На этом уроке мы импортируем схему и данные. Для этого используется команда:

|  |
| --- |
| mysql -u[username] -p[password] [database name] < file.sql |

# Практическое задание

1. Имеющаяся у нас схема не очень годится для работы. Нужно привести её в нормальный вид, таблицы должны выглядеть следующим образом:

**Таблица \_countries**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **FIELD** | **TYPE** | **NULL** | **KEY** | **EXTRA** |
| id | INT | NOT | Primary Key | auto\_increment |
| title | VARCHAR(150) | NOT | INDEX |  |

**Таблица \_regions**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **FIELD** | **TYPE** | **NULL** | **KEY** | **EXTRA** |
| id | INT | NO | Primary Key | auto\_increment |
| country\_id | INT | NO | Foreign Key \_countries (id) |  |
| title | VARCHAR(150) | NO | INDEX |  |

**Таблица \_cities**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **FIELD** | **TYPE** | **NULL** | **KEY** | **EXTRA** |
| id | INT | NO | Primary Key | auto\_increment |
| country\_id | INT | NO | Foreign Key \_countries (id) |  |
| important | TINYINT(1) | NO |  |  |
| region\_id | INT | NO | Foreign Key \_regions (id) |  |
| title | VARCHAR(150) | NO | INDEX |  |

Самый простой способ – написать команды ALTER TABLE в отдельном файле с расширением .sql и импортировать его:

|  |
| --- |
| mysql -u[username] -p[password] [database name] < file.sql |

На изменение схемы может уйти очень много времени (до 1 часа), так что не пугайтесь, если команда заставит вас долго ждать.

Для проверки результатов работы могут быть полезны следующие команды:

Показать команду DDL, с помощью которой таблица была создана:

|  |
| --- |
| SHOW CREATE TABLE `<yourtable>`; |

Показать структуру таблицы:

|  |
| --- |
| SHOW COLUMNS FROM `<yourtable>`; |

# Дополнительные материалы

1. <http://www.mysqltutorial.org/>

# Используемая литература

Для подготовки данного методического пособия были использованы следующие ресурсы:

1. <http://dev.mysql.com/doc/refman/5.7/en/alter-table.html>
2. <http://dev.mysql.com/doc/refman/5.7/en/create-table.html>

**NG**

**SELECT u.\* FROM orders o**

**JOIN users u ON u.users\_id = o.users\_id**

**WHERE o.status = 1 AND COUNT(o.status) > 5**

**GROUP BY o.users\_id**

**Чем характерна третья нормальная форма по сравнению с 1 и 2?**

Ваш ответ

**Каждый неключевой атрибут неприводимо зависит от первичного ключа**

**Каким свойством не обладает первичный ключ?**

Ваш ответ

**Минимальность**

**Какие типы данных в MySQL позволяют хранить строковые значения?**

Ваш ответ

String

Ваш ответ

Char

**Есть таблица компьютеров PC (model, speed, price).**

**Какой запрос выведет:**

**- все компьютеры, с тактовой частотой больше 1500 МГц**

**- среднюю стоимость компьютеров с таким же быстродействием?**

Ваш ответ

**SELECT pc.speed, AVG(pc.price) FROM PC WHERE pc.speed > 1500**

**Какие способы помогают избежать взаимных блокировок в транзакциях?**

Ваш ответ

**Использование только блокировок на чтение (S)**

Ваш ответ

**Использование только эксклюзивных блокировок (X)**

Есть таблица с пользовательскими данными об их транспортных средствах. Марка машины вводится вручную, поэтому некоторые пользователи указывают «Mer**c**edes», а некоторые – «Mer**s**edes». Какие запросы выбирают всех владельцев автомобилей Mercedes-Benz, даже если они написали название марки с ошибкой?

Ваш ответ

**SELECT \* FROM cars WHERE brand LIKE “Mer\_edes”**

Столбец name таблицы tbl содержит запись со строкой 'cat'. Укажите запросы, при помощи которых можно извлечь данную запись?

Ваш ответ

SELECT \* FROM tbl WHERE name LIKE '\*\*\*';

Ваш ответ

SELECT \* FROM tbl WHERE name LIKE '%';

### 

### 

|  |  |
| --- | --- |
| **Категория** | **Товары** |
| Строительные материалы | Кирпич |
| Строительные материалы | Цемент |
| Строительные материалы | Гвозди |
| Отделочные материалы | Гипсокартон |
| Отделочные материалы | Обои |
| Отделочные материалы | Краска |

## Третья нормальная форма

Переменная отношения находится в третьей нормальной форме тогда и только тогда, когда она находится во второй нормальной форме, и отсутствуют транзитивные функциональные зависимости неключевых атрибутов от ключевых.

### Пример приведения к третьей нормальной форме

Привести к третьей нормальной форме – это значит выделить из таблицы первичный ключ, который будет уникально идентифицировать запись в таблице.

|  |  |
| --- | --- |
| **ID** | **Категория** |
| 1 | Строительные материалы |
| 2 | Отделочные материалы |

|  |  |  |
| --- | --- | --- |
| **Категория** | **Дата акции** | **Скидка** |
| 1 | 21.06.2016 | 10% |
| 2 | 25.06.2016 | 20% |

### 

### 

### 

|  |  |
| --- | --- |
| **Категория** | **Товары** |
| 1 | Кирпич |
| 1 | Цемент |
| 1 | Гвозди |
| 2 | Гипсокартон |
| 2 | Обои |
| 2 | Краска |

## Нормальная форма Бойса-Кодда

Переменная отношения находится в нормальной форме Бойса-Кодда (иначе – в усиленной третьей нормальной форме), когда каждая её нетривиальная и неприводимая слева функциональная зависимость имеет в качестве своего детерминанта некоторый потенциальный ключ.

Другими словами, в таблице должен быть только один первичный ключ.

## Четвёртая нормальная форма

Переменная отношения находится в четвертой нормальной форме, если она находится в нормальной форме Бойса-Кодда и не содержит нетривиальных многозначных зависимостей.

Предположим, что рестораны производят разные виды пиццы, а службы доставки ресторанов работают только в определенных районах города. Составной первичный ключ соответствующей переменной отношения включает три атрибута: {Ресторан, Вид пиццы, Район доставки}.

Такая переменная отношения не соответствует 4НФ, так как существует следующая многозначная зависимость:

{Ресторан} → {Вид пиццы}

{Ресторан} → {Район доставки}

То есть, например, при добавлении нового вида пиццы придется внести по одному новому кортежу для каждого района доставки. Возможна логическая аномалия, при которой определенному виду пиццы будут соответствовать лишь некоторые районы доставки из обслуживаемых рестораном районов.

Для предотвращения аномалии нужно декомпозировать отношение, разместив независимые факты в разных отношениях. В данном примере следует выполнить декомпозицию на {Ресторан, Вид пиццы} и {Ресторан, Район доставки}.

Однако, если к исходной переменной отношения добавить атрибут, функционально зависящий от потенциального ключа, например, цену с учётом стоимости доставки ({Ресторан, Вид пиццы, Район доставки} → Цена), полученное отношение будет находиться в 4НФ и его уже нельзя подвергнуть декомпозиции без потерь.

## Пятая нормальная форма

Переменная отношения находится в пятой нормальной форме (иначе – в проекционно-соединительной нормальной форме), когда каждая нетривиальная зависимость соединения в ней определяется потенциальным ключом (ключами) этого отношения.

Другими словами, если есть сложная связь между зависимыми атрибутами, она должна выноситься в отдельную таблицу. Не нужно делать зависимость трех и более атрибутов. На практике такое встречается очень редко, но придерживаться данного правила необходимо.

## Доменно-ключевая нормальная форма

Переменная отношения находится в ДКНФ, когда каждое наложенное на неё ограничение является логическим следствием ограничений доменов и ограничений ключей, наложенных на данную переменную отношения.

## Шестая нормальная форма

Переменная отношения находится в шестой нормальной форме, когда она удовлетворяет всем нетривиальным зависимостям соединения. Из определения следует, что переменная находится в 6НФ, когда она неприводима, то есть не может быть подвергнута дальнейшей декомпозиции без потерь. Каждая переменная отношения, которая находится в 6НФ, также находится и в 5НФ.

Введена К. Дейтом в его книге как обобщение пятой нормальной формы для хронологической базы данных.

## Практическое применение

Из всех представленных форм практическое применение обычно имеют первые четыре НФ. Наши схемы мы будем создавать на основе четырех НФ.

# 

# 

# 

# Типы данных в MySQL

Для успешного проектирования РСУБД недостаточно понимания нормальных форм. Как известно, объем данных, которые необходимо сохранить, практически не ограничен (пример – математические числа), но память компьютера имеет предел. Поэтому мы ограничены набором типов данных, каждый из них ограничен своим набором байт.

Также практическое понимание типов данных необходимо для построения быстрого и эффективного поиска по БД.

Разберемся, какие есть типы данных в MySQL. Типы данных соответствуют стандартам ANSI SQL.

**TINYINT[(M)] [UNSIGNED] [ZEROFILL]**

Очень малое целое число. Диапазон со знаком от -128 до 127. Диапазон без знака от 0 до 255.

**BIT, BOOL**

Являются синонимами для TINYINT(1).

**SMALLINT[(M)] [UNSIGNED] [ZEROFILL]**

Малое целое число. Диапазон со знаком от -32768 до 32767. Диапазон без знака от 0 до 65535.

**MEDIUMINT[(M)] [UNSIGNED] [ZEROFILL]**

Целое число среднего размера. Диапазон со знаком от -8388608 до 8388607. Диапазон без знака от 0 до 16777215.

**INT[(M)] [UNSIGNED] [ZEROFILL]**

Целое число нормального размера. Диапазон со знаком от -2147483648 до 2147483647. Диапазон без знака от 0 до 4294967295.

**INTEGER[(M)] [UNSIGNED] [ZEROFILL]**

Синоним для INT.

**BIGINT[(M)] [UNSIGNED] [ZEROFILL]**

Большое целое число. Диапазон со знаком от -9223372036854775808 до 9223372036854775807. Диапазон без знака от 0 до 18446744073709551615. Для столбцов типа BIGINT необходимо учитывать некоторые особенности:

Все арифметические операции выполняются с использованием значений BIGINT или DOUBLE со знаком, так что не следует использовать беззнаковые целые числа больше чем 9223372036854775807 (63 бита), кроме операций, выполняемых логическими функциями. В противном случае несколько последних разрядов результата могут оказаться неверными из-за ошибок округления при преобразовании BIGINT в DOUBLE. MySQL 4.0 может обрабатывать данные типа BIGINT в следующих случаях:

Использование целых чисел для хранения больших беззнаковых величин в столбце с типом BIGINT.

В случаях MIN(big\_int\_column) и MAX(big\_int\_column).

При использовании операторов («+» , «-», «\*» и т.д.), когда оба операнда являются целыми числами.

Точное значение целого числа всегда можно хранить в столбце с типом BIGINT в виде строки. В этом случае MySQL выполнит преобразование строки в число без промежуточного преобразования.

Если оба аргумента являются целочисленными величинами, при выполнении над ними операций «+» , «-» и «\*» будут использоваться правила BIGINT-арифметики. Это означает, что при умножении двух больших целых чисел (или результатов вычислений функций, возвращающих целые числа) результат операции может оказаться непредсказуемым, если он превосходит значение 9223372036854775807.

**FLOAT (точность) [UNSIGNED] [ZEROFILL]**

Число с плавающей точкой. Атрибут точности может иметь значение <=24 для числа с плавающей точкой обычной (одинарной) точности и между 25 и 53 – для числа с плавающей точкой удвоенной точности. Эти типы данных сходны с типами FLOAT и DOUBLE, описанными ниже. FLOAT(X) относится к тому же интервалу, что и соответствующие типы FLOAT и DOUBLE, но диапазон значений и количество десятичных знаков не определены.

В версии MySQL 3.23 это истинная величина числа с плавающей точкой. В более ранних версиях MySQL тип данных FLOAT(точность) всегда имеет два десятичных знака.

Следует отметить, что использование типа данных FLOAT может привести к неожиданным проблемам, так как все вычисления в MySQL выполняются с удвоенной точностью.

**FLOAT[(M,D)] [UNSIGNED] [ZEROFILL]**

Малое число с плавающей точкой обычной точности. Допустимые значения: от -3,402823466E+38 до -1,175494351E-38, 0, и от 1,175494351E-38 до 3,402823466E+38. Если указан атрибут UNSIGNED, отрицательные значения недопустимы. Атрибут M указывает количество выводимых пользователю знаков, а атрибут D – количество разрядов, следующих за десятичной точкой. Обозначение FLOAT без указания аргументов или запись вида FLOAT(X), где X <=24 справедливы для числа с плавающей точкой обычной точности.

**DOUBLE[(M,D)] [UNSIGNED] [ZEROFILL]**

Число с плавающей точкой удвоенной точности нормального размера. Допустимые значения: от -1,7976931348623157E+308 до -2,2250738585072014E-308, 0, и от 2,2250738585072014E-308 до 1,7976931348623157E+308. Если указан атрибут UNSIGNED, отрицательные значения недопустимы. Атрибут M указывает количество выводимых пользователю знаков, а атрибут D – количество разрядов, следующих за десятичной точкой. Обозначение DOUBLE без указания аргументов или запись вида FLOAT(X), где 25 <= X <= 53 справедливы для числа с плавающей точкой двойной точности.

**DOUBLE PRECISION[(M,D)] [UNSIGNED] [ZEROFILL] , REAL[(M,D)] [UNSIGNED] [ZEROFILL]**

Данные обозначения являются синонимами для DOUBLE.

**DECIMAL[(M[,D])] [UNSIGNED] [ZEROFILL]**

«Неупакованное» число с плавающей точкой. Ведет себя подобно столбцу CHAR, содержащему цифровое значение. Термин «неупакованное» означает, что число хранится в виде строки, и при этом для каждого десятичного знака используется один символ. Разделительный знак десятичных разрядов, а также знак «-» для отрицательных чисел, не учитываются в M (но место для них зарезервировано). Если атрибут D равен 0, величины будут представлены без десятичного знака, т. е. без дробной части. Максимальный интервал значений типа DECIMAL тот же, что и для типа DOUBLE, но действительный интервал для конкретного столбца DECIMAL может быть ограничен выбором значений атрибутов M и D. Если указан атрибут UNSIGNED, отрицательные значения недопустимы. Если атрибут D не указан, его значение по умолчанию равно 0. Если не указан M, его значение по умолчанию равно 10. В более ранних, чем MySQL 3.23, версиях аргумент M должен содержать в себе место для знака числа и десятичного знака.

**DEC[(M[,D])] [UNSIGNED] [ZEROFILL] , NUMERIC[(M[,D])] [UNSIGNED] [ZEROFILL]**

Данные обозначения являются синонимами для DECIMAL.

**DATE**

Дата. Поддерживается интервал от '1000-01-01' до '9999-12-31'. MySQL выводит значения DATE в формате 'YYYY-MM-DD', но можно установить значения в столбец DATE, используя как строки, так и числа.

**DATETIME**

Комбинация даты и времени. Поддерживается интервал от '1000-01-01 00:00:00' до '9999-12-31 23:59:59'. MySQL выводит значения DATETIME в формате 'YYYY-MM-DD HH:MM:SS', но можно устанавливать значения в столбце DATETIME, используя как строки, так и числа.

**TIMESTAMP[(M)]**

Временная метка. Интервал от '1970-01-01 00:00:00' до некоторого значения времени в 2037 году. MySQL выводит значения TIMESTAMP в форматах YYYYMMDDHHMMSS, YYMMDDHHMMSS, YYYYMMDD или YYMMDD в зависимости от значений M: 14 (или отсутствующее), 12, 8, или 6; но можно также устанавливать значения в столбце TIMESTAMP, используя как строки, так и числа. Столбец TIMESTAMP полезен для записи даты и времени при выполнении операций INSERT или UPDATE, так как при этом автоматически вносятся значения даты и времени самой последней операции, если эти величины не введены программой. Можно также устанавливать текущее значение даты и времени, задавая значение NULL. Аргумент M влияет только на способ вывода столбца TIMESTAMP; для хранения его значений всегда используется 4 байта. Следует учитывать, что столбцы TIMESTAMP(M), где M равно 8 или 14, представляют собой числа, в то время, как столбцы TIMESTAMP(M) с иным значением аргумента M являются строками. Это убеждает, что можно надежно сделать дамп и восстановить таблицу с этими типами столбцов!

**TIME**

Время. Интервал от '-838:59:59' до '838:59:59'. MySQL выводит значения TIME в формате 'HH:MM:SS', но можно устанавливать значения в столбце TIME, используя как строки, так и числа.

**YEAR[(2|4)]**

Год в двухзначном или четырехзначном форматах (по умолчанию формат четырехзначный). Допустимы следующие значения: с 1901 по 2155, 0000 для четырехзначного формата года и 1970-2069 при использовании двухзначного формата (70-69). MySQL выводит значения YEAR в формате YYYY, но можно задавать значения в столбце YEAR, используя как строки, так и числа (тип данных YEAR недоступен в версиях, предшествующих MySQL 3.22).

**[NATIONAL] CHAR(M) [BINARY]**

Строка фиксированной длины, при хранении всегда дополняется пробелами в конце строки до заданного размера. Диапазон аргумента M составляет от 0 до 255 символов (от 1 до 255 в версиях, предшествующих MySQL 3.23). Концевые пробелы удаляются при выводе значения. Если не задан атрибут чувствительности к регистру BINARY, величины CHAR сортируются и сравниваются как независимые от регистра в соответствии с установленным по умолчанию алфавитом.

Атрибут NATIONAL CHAR (или его эквивалентная краткая форма NCHAR) представляет собой принятый в ANSI SQL способ указания, что в столбце CHAR должен использоваться установленный по умолчанию набор символов (CHARACTER). В MySQL это принято по умолчанию. CHAR является сокращением от CHARACTER. MySQL позволяет создавать столбец типа CHAR(0).

В основном это полезно, если необходимо обеспечить совместимость с некоторыми старыми приложениями, которые зависят от наличия столбца, но реально эту величину не используют. Кроме того, такая возможность может очень пригодиться в случае, если необходим столбец, который может содержать только 2 значения, а именно CHAR(0) (т.е. столбец, который не определен как NOT NULL, занимает только один бит и принимает только 2 значения: NULL или «»).

**CHAR**

Это синоним для CHAR(1).

**[NATIONAL] VARCHAR(M) [BINARY]**

Строка переменной длины. Примечание: концевые пробелы удаляются при сохранении значения (в этом заключается отличие от спецификации ANSI SQL). Диапазон аргумента M составляет от 0 до 255 символов (от 1 до 255 в версиях, предшествующих MySQL Version 4.0.2). Если не задан атрибут чувствительности к регистру BINARY, величины VARCHAR сортируются и сравниваются как независимые от регистра.

**TINYBLOB, TINYTEXT**

Столбец типа BLOB или TEXT с максимальной длиной 255 (2^8 - 1) символов.

**BLOB, TEXT**

Столбец типа BLOB или TEXT с максимальной длиной 65535 (2^16-1).

**MEDIUMBLOB, MEDIUMTEXT**

Столбец типа BLOB или TEXT с максимальной длиной 16777215 (2^24-1).

**LONGBLOB, LONGTEXT**

Столбец типа BLOB или TEXT с максимальной длиной 4294967295 (2^32-1) символов. Следует учитывать, что в настоящее время протокол передачи данных сервер/клиент и таблицы MyISAM имеют ограничение 16 Мб на передаваемый пакет/строку таблицы, поэтому пока нельзя использовать этот тип данных в его полном диапазоне.

**ENUM('значение1','значение2',...)**

Перечисление. Перечисляемый тип данных. Объект строки может иметь только одно значение, выбранное из заданного списка величин 'значение1', 'значение2', ..., NULL или специальная величина ошибки «». Список ENUM может содержать максимум 65535 различных величин.

**SET('значение1','значение2',...)**

Набор. Объект строки может иметь ноль или более значений, каждое из которых должно быть выбрано из заданного списка величин 'значение1', 'значение2', ... Список SET может содержать максимум 64 элемента.

### Практическое применение

На практике чаще всего используются INT, FLOAT, VARCHAR, TEXT, DATETIME, TIMESTAMP. Этими типами данных можно описать любую таблицу. Но знание и понимание типов данных необходимо, потому что встречаются задачи, где их использование необходимо.

# Ключи

## Первичный ключ

В РСУБД существует несколько видов ключей. Самый главный – первичный. Он уникально идентифицирует строку данных, без него приведение к 4НФ было бы невозможно.

Первичный ключ обладает следующими свойствами:

1. Уникальность.
2. Долговечность.
3. Минимальность.

Первичный ключ может быть простым или составным. Если первичный ключ состоит из нескольких атрибутов, он является составным.

## Уникальный ключ

Уникальный ключ обладает всеми свойствами первичного, но первичным не является. Используется для уникальных атрибутов (например номер банковского счета, email-адрес).

## Внешний ключ

Внешний ключ является ключом – ссылкой, связью между таблицами. Внешние ключи используются для проверки РСУБД на целостность. Если атрибут, на который указывает внешний ключ, не будет присутствовать в таблице, на которую он указывает, это приведет к ошибке.

# Индексы и алгоритмы поиска

Давайте представим себе ситуацию: перед нами стоит задача найти в упорядоченном списке число. Допустим, у нас есть список из 1000 элементов, нас интересует элемент под номером 521.

1, 2, 3 . . . 521 . . . 998, 999, 1000

Первый алгоритм, который может прийти нам в голову, это простой перебор через цикл со сравнением с заданным числом. Для того, чтобы найти в этом списке интересующий нас элемент, нам необходимо сделать 521 шаг цикла. Потенциально, это может быть очень медленно и проблематично.

1 2 3 . . . 521 . . . 998 999 1000

Данную проблему может решить алгоритм быстрого поиска по упорядоченному списку – метод деления пополам. Допустим, нам необходимо найти все то же число 521. Давайте разделим наш числовой ряд пополам и проверим, в какой части осталось число 521.

1 2 3 . . . 500 . . . 521 . . . 998 999 1000

Результат будет 500 < 521. Значит интересующее нас число больше. Делим пополам правую часть и сравниваем снова. Продолжаем выполнять эту операцию до тех пор, пока не находим исходное число.

500 . . . 521 . . . 750

При применении данного метода мы можем за 4–5 шагов найти любое интересующее нас значение. Насколько мы видим, данный метод очень эффективен.

Это был простейший пример поиска в отсортированном списке. В современных СУБД реализовано много алгоритмов поиска, по разным типам данных и структурам. Механизм, обеспечивающий возможности ускоренного поиска, называется индексом.

В MySQL есть несколько видов индексов – binary tree (B-Tree) и hash table.

Hash-индекс применяется для сравнения/построения индексов строчных и/или двоичных данных. Каждому значению индексируемого выражения сопоставляется значение определенной хэш-функции, отображающей исходное значение на целое число (иногда на строку).

B-Tree индекс дает скорость выборки порядка log(N), hash дает линейную. В реальной жизни hash и B-Tree применяются совместно, то есть для вычисления значений B-Tree индекса все равно применяются хэши.

Если мы объявляем ключи, индексы на эти поля создаются автоматически, так как ключи проверяют структуру данных на целостность, что является очень трудозатратной вычислительной операцией. Для их оптимизации индексы проставляются механизмом MySQL. Но если нам нужен индекс на неключевом поле, значит нам нужно будет создать его вручную.

Также есть специальный тип индексов – full text. Он используется для построения индекса по текстовым полям с большим объемом символов.

# Практическая работа с учителем

Учимся вместе проектировать базу данных в MySQL Workbench – строить таблицы, выбирать тип данных – на примере интернет-магазина.

Для начала надо установить Workbench. Скачать его можно по ссылке:

<https://dev.mysql.com/downloads/workbench/>

Наша задача состоит в построении простой схемы карточки товаров и корзины в интернет-магазине. Схема должна состоять из следующих таблиц:

Таблица «Товары»:

|  |  |  |
| --- | --- | --- |
| **Поле** | **Описание** | **Тип данных** |
| ID | Первичный ключ таблицы | INT |
| name | Наименование товара | VARCHAR(50) |
| description | Описание товара | VARCHAR(255) |
| price | Цена товара | FLOAT |
| count | Количество товаров на складе | INT |
| category | Категория товара, внешний ключ | INT |

Таблица «Категория»:

|  |  |  |
| --- | --- | --- |
| **Поле** | **Описание** | **Тип данных** |
| ID | Первичный ключ таблицы | INT |
| name | Наименование категории (уникальный ключ) | VARCHAR(50) |

Таблица «Заказ»:

|  |  |  |
| --- | --- | --- |
| **Поле** | **Описание** | **Тип данных** |
| ID | Первичный ключ таблицы | INT |
| date | Дата заказа | DATETIME |
| address | Адрес доставки | VARCHAR(255) |

Таблица «Товары в заказе»:

|  |  |  |
| --- | --- | --- |
| **Поле** | **Описание** | **Тип данных** |
| ID | Первичный ключ таблицы | INT |
| order\_id | Ссылка на заказ | INT |
| goods\_id | Ссылка на товар | INT |

# Практическое задание

1. Спроектировать базу данных «Страны и города мира» с помощью MySQL Workbench. Творческая работа, проектировать на собственное усмотрение. Схема будет такой: страна ➝ область ➝ район (если есть) ➝ город (поселок, деревня). Сдаем в формате .zip: упаковываем готовый скрипт, который сгенерировал Workbench, и схему Workbench.

# Дополнительные материалы

1. <https://habrahabr.ru/post/254773/>

# Используемые ресурсы

Для подготовки данного методического пособия были использованы следующие ресурсы:

1. <https://habrahabr.ru/post/254773/>
2. <https://ru.wikipedia.org/wiki/%D0%A0%D0%B5%D0%BB%D1%8F%D1%86%D0%B8%D0%BE%D0%BD%D0%BD%D0%B0%D1%8F_%D0%B1%D0%B0%D0%B7%D0%B0_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85>
3. <https://ru.wikipedia.org/wiki/%D0%9D%D0%BE%D1%80%D0%BC%D0%B0%D0%BB%D1%8C%D0%BD%D0%B0%D1%8F_%D1%84%D0%BE%D1%80%D0%BC%D0%B0#.D0.9F.D0.B5.D1.80.D0.B2.D0.B0.D1.8F_.D0.BD.D0.BE.D1.80.D0.BC.D0.B0.D0.BB.D1.8C.D0.BD.D0.B0.D1.8F_.D1.84.D0.BE.D1.80.D0.BC.D0.B0_.281NF.29>
4. <http://phpclub.ru/mysql/doc/column-types.html>
5. <http://ru.stackoverflow.com/questions/246659/b-tree-%D0%B8-hash-%D0%B8%D0%BD%D0%B4%D0%B5%D0%BA%D1%81%D1%8B>