GeekBrains

Базы данных. Интерактивный курс

Быстрый старт

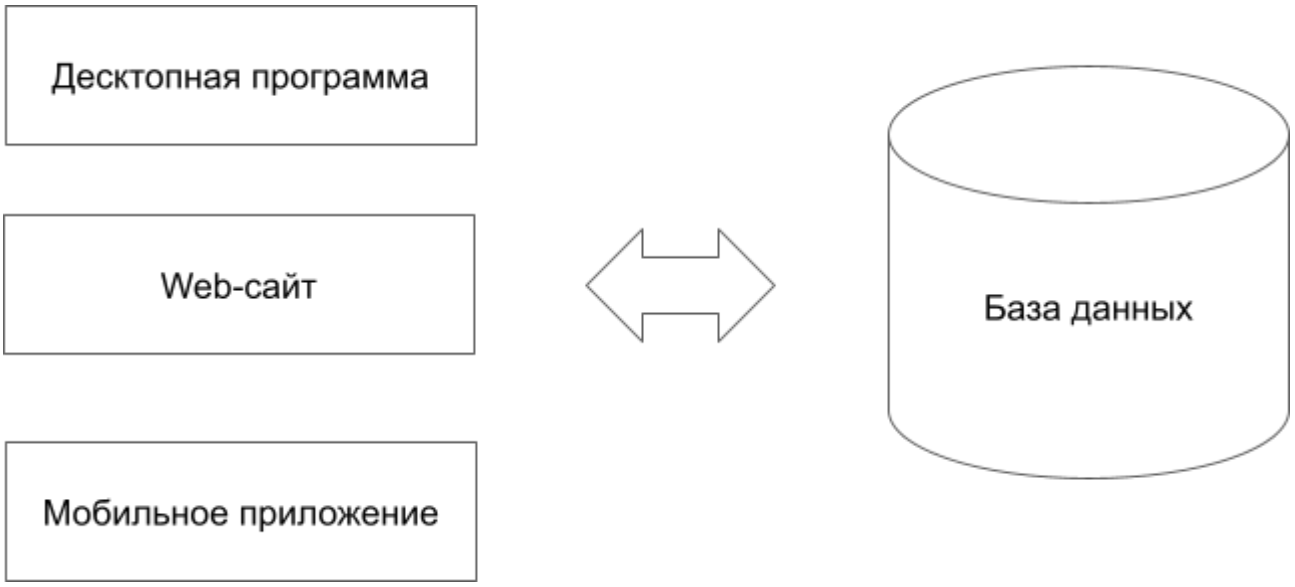
Быстрый старт

Знакомство с реляционными базами данных будет вестись на примере MySQL.

Данные и программы

Данные живут дольше, чем программы. Наши программы пока слишком недолговечны и часто меняются. Во время жизненного цикла данных в разное время их может обслуживать несколько программ. Иногда одни и те же данные обслуживает несколько программ одновременно.

Поэтому в программировании принято отделять данные от кода и держать их в специализированном хранилище — базе данных.



База данных — это совокупность информационных материалов, организованных таким образом, чтобы их можно было найти и обработать при помощи компьютера.

Обычный текстовый файл тоже база данных, пусть и очень примитивная.

Почему недостаточно обычных файлов?

Дело в том, что файлы довольно ограничены по возможностям. При работе с большим объемом данных необходимо обеспечить их компактность. Их лучше записывать в бинарном, а не текстовом формате, возможно, применяя механизмы сжатия. Это не очень наглядно, и для восприятия в любом случае потребуется конвертация данных в формат, доступный человеку. Когда множество параллельных процессов или клиентов обращаются к файлу с целью записать или извлечь информацию из него, очень трудно обеспечить конкурентный доступ. Необходимо либо прибегать к блокировкам файла, либо создавать очередь для запросов. В файлы очень легко записывать информацию, если мы вносим ее в самый конец, однако очень не просто отредактировать запись в середине. Кроме того, чтобы что-то найти в файле, приходится его сканировать от начала до конца. Чтобы кешировать часто используемые данные в оперативной памяти, вам придется писать собственную программу.

Ситуация еще больше усложняется, если наш файл гигантского объема и просто не убирается на одном компьютере. И вот у вас уже несколько файлов, которые хранятся на нескольких компьютерах. Как искать среди них информацию? Придется писать дополнительное программное обеспечение.

А что будем делать, если одновременно два клиента захотят исправить один и тот же документ, внося в него совершенно разную информацию?

Поэтому практически сразу после появления операционных систем и файлов над базами данных стали появляться программные надстройки. Они позволяли управлять, искать, пополнять и редактировать данные внутри базы данных. Решать те все проблемы, которые мы с вами обозначили.

Такая надстройка стала называться системой управления базами данных или сокращенно СУБД. В нашем курсе для краткости мы будем называть базами данных совокупность как самого хранилища, так и этой программной надстройки.

История развития СУБД

Современные базы данных за последние 60 лет прошли длительный путь развития. Давайте кратко пробежимся по истории СУБД, от первых иерархических баз данных до современных NoSQL-решений.

Иерархические базы данных

Первые базы данных были иерархическими. Это, наверное, вообще первое, что приходит в голову программистам.

Иерархия — это дерево, состоящее из узлов, у которых может быть несколько потомков. При помощи такой структуры хорошо описываются иерархические структуры организаций и производств.

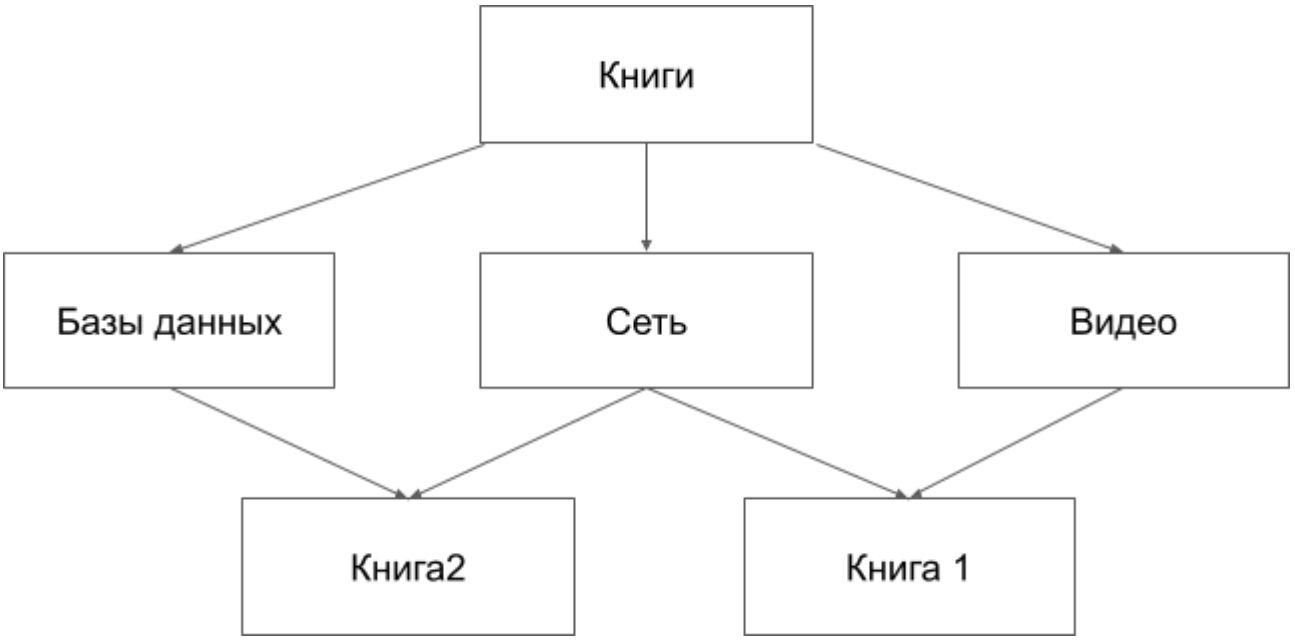
Примеров иерархий очень много и они постоянно находятся у нас перед глазами. На экране вы можете видеть иерархию транспортной системы. Есть вершина — транспорт, от которого расходятся узлы с видами транспорта и последующей детализацией специализации ТС.

Иерархии очень наглядны и хорошо описываются деревьями, у которых прекрасно изученный мат. аппарат.

Главное их достоинство — высокая скорость обработки операций. Первые компьютеры не отличались высокой производительностью: чем проще организована база данных, тем быстрее она работает.

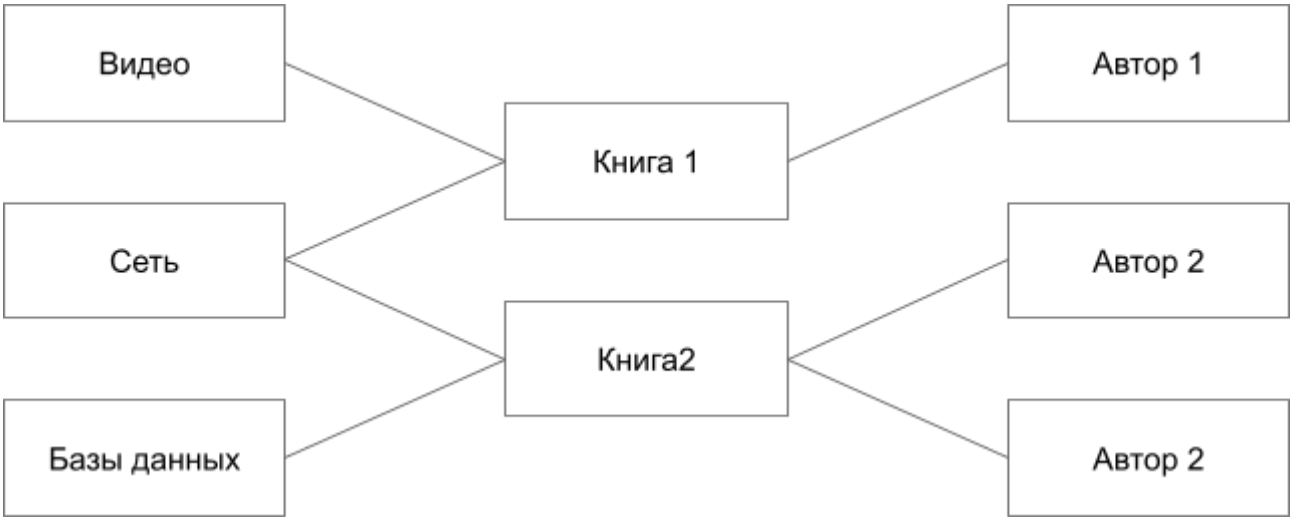
Основной недостаток иерархической структуры базы данных — невозможность реализовать отношения «многие ко многим».

Например, если мы создаём каталог книг, одна книга может относиться сразу к нескольким разделам.



Сетевые базы данных

Была разработана новая модель данных — сетевая. Она расширила иерархическую модель, позволяя одной записи участвовать в нескольких отношениях «предок-потомок».



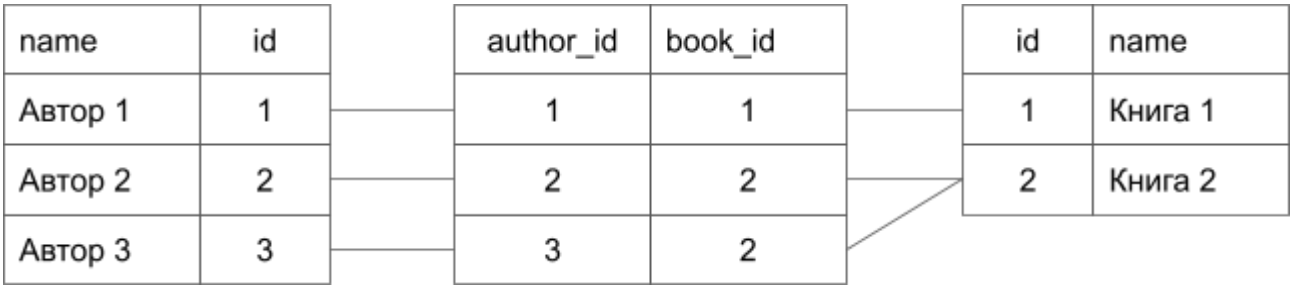
В связи с развитием социальных сетей эти базы данных получили второе дыхание в виде графовых СУБД, которые относятся к современному NoSQL-течению. Только в современной интерпретации ценность приобретают не сами данные, а связи между узлами. Тем не менее, что примечательно, многие идеи, которые появляются в новых популярных NoSQL-базах, были придуманы или опробованы в прошлом.

Просто на тот момент это было либо экономически нецелесообразно, либо мода и влияние больших компаний толкало рынок в сторону других моделей. Конечно, у сетевых баз данных имелись

недостатки: подобно своим иерархическим предкам, сетевые базы данных были очень жесткими. Наборы отношений и структура записей должны были быть заданы наперед. Изменение структуры базы данных обычно означало ее полную перестройку.

Реляционные базы данных

Следующим шагом стало развитие реляционных баз данных. Реляционная модель данных была попыткой упростить структуру базы данных. В ней отсутствовала явная структура «предок-потомок», а все данные были представлены в виде простых таблиц, разбитых на строки и столбцы.



name	id
Автор 1	1
Автор 2	2
Автор 3	3

author_id	book_id
1	1
2	2
3	2

id	name
1	Книга 1
2	Книга 2

Теоретические основы новой реляционной модели данных впервые были описанны доктором Коддом в 1970 году. Поначалу его работа предоставляла лишь академический интерес. Однако, спустя 10 лет, основываясь на его работе, реляционные базы данных создали сначала Oracle, затем IBM, а потом и множество других компаний.

Реляционные СУБД прочно вошли в компьютерный мир и актуальны до сих пор. Это самый распространенный вид баз данных. Львиная доля курса будет посвящена именно ему.

За 40 лет было множество попыток заменить реляционные баз данных чем-то более новым и прогрессивным.

Долгое время на смену СУБД пророчили приход XML и объектно-ориентированных баз данных. Однако эти технологии так и не стали массовыми, хотя продукты существуют и по сей день. XML вышел из моды из-за своей избыточности. ООП-базы данных, которые решают проблему несовместимости реляционных баз данных, основанных на множествах, и ООП-программ, основанных на деревьях, тоже не стали слишком популярны.

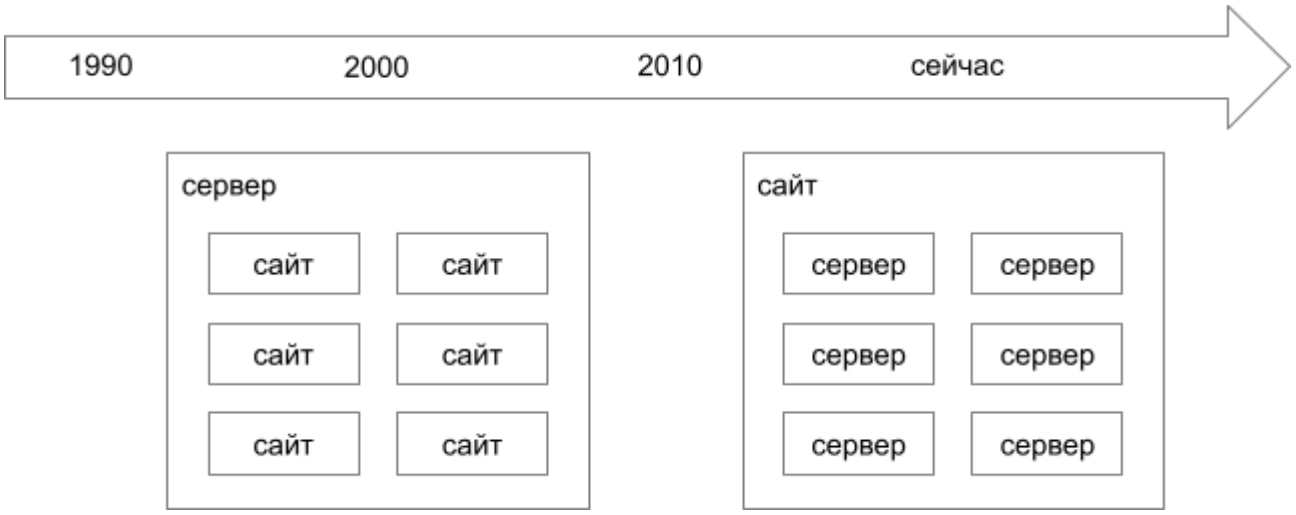
Наиболее популярные СУБД на сегодняшний день — Oracle, MS SQL и DB2 среди коммерческих, MySQL, PostgreSQL и Firebird среди свободных.

Индекс популярности баз данных

Следить за индексом популярности баз данных можно по рейтингу на сайте db-engines.com. Здесь представлены не только реляционные базы данных, но и NoSQL-решения. Тем не менее, по этому ресурсу вы можете отслеживать динамику интереса к базам данных на протяжении длительного времени.

NoSQL базы данных

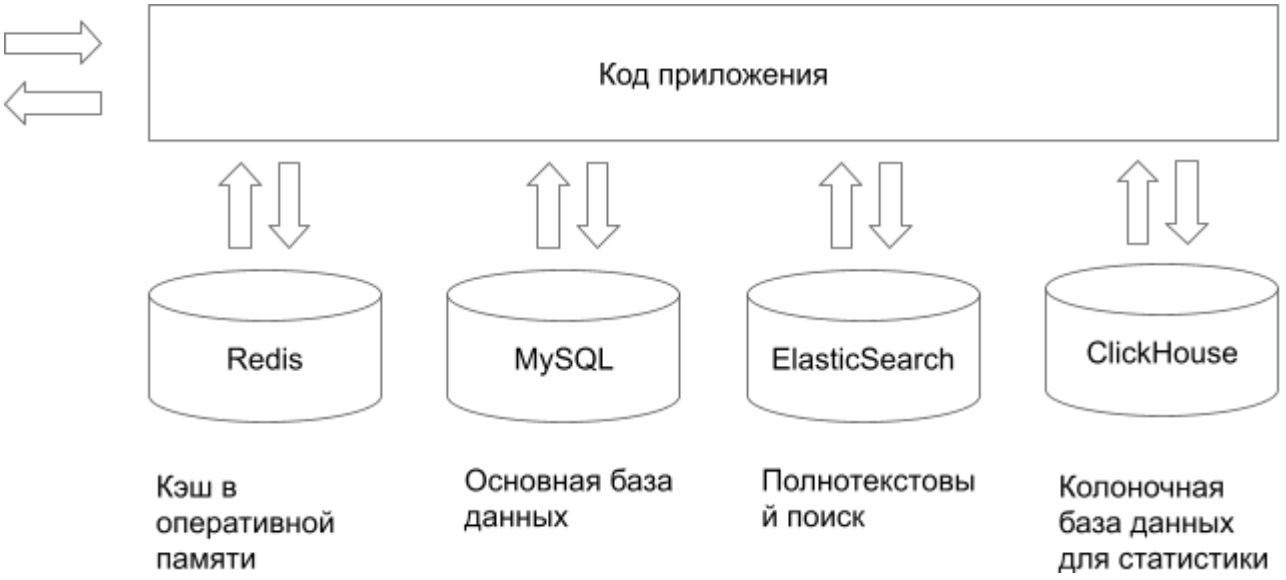
После стремительного развития интернета в принципах построения баз данных произошел переломный момент. Первые веб-сайты и сервисы были очень невелики: зачастую на одном сервере убирались сотни сайтов. Однако по мере вовлечения все новых и новых пользователей проекты начали укрупняться и очень скоро им стало не хватать не то что одного, а десятков, сотен, а затем и тысяч серверов.



Не стали помещаться на одном сервере и базы данных. Поэтому очень скоро сначала реляционные СУБД, а потом и новые игроки стали отказываться от традиционного подхода хранения данных.

Стали строиться распределенные хранилища и интенсивно использоваться гораздо большие объемы оперативной памяти. Новые подходы и распределенная структура баз данных привели к ситуации, когда реализовать стандартный SQL-язык стало либо очень сложно, либо почти невозможно.

В результате на рынке стали появляться специализированные СУБД, ориентированные под решение тех или иных задач, зачастую полностью расположенные в оперативной памяти, предоставляющие свой язык запросов, иногда вообще не следующий многолетней традиции SQL.



На экране представлены типичные представители NoSQL-базы данных. На самом деле их гораздо больше.

Redis — это очень быстрое хранилище построенное по принципу «ключ-значение». Оно полностью расположено в оперативной памяти, сервер реализован в виде однопоточного EventLoop-цикла, когда один поток опрашивает по кругу соединения в неблокирующем режиме. За счет того, что не происходит переключение процессора на другие процессы, достигается гигантская производительность порядка 100 000 RPS (это зачастую в сотни раз выше, чем в лучших реляционных базах данных).

Если на сервере не хватает оперативной памяти, чтобы разместить индекс, можно разбить данные на части — шарды и хранить несколько копий такого шарда на разных компьютерах. В результате образуется кластер, который ведет себя как единый компьютер с огромным количеством оперативной памяти. Так действуют grid-решения в Oracle, MySQL, так могут поступать и NoSQL-базы данных вроде ElasticSearch и MongoDB.

Собирать JSON-документ из нескольких таблиц может быть долго и накладно. В этом случае можно хранить не отдельные значения документа, а готовый, собранный заранее, документ. Для этого используются документо-ориентированные СУБД, примером может служить та же MongoDB.

Часто в реляционных базах данных штатный механизм полнотекстового поиска не предусмотрен или реализован неэффективно. Поэтому можно прибегать к базам данных, специально предназначенных для полнотекстового поиска, позволяющих регулировать любые параметры поискового механизма. Яркий представитель таких баз данных — ElasticSearch.

Традиционные СУБД плохо предназначены для операций в реальном времени, например для обсчета статистики. Гораздо лучше для этих целей подходит колоночная база данных. В ней запрещены операции редактирования и удаления, данные сжаты, что позволяет обеспечивать исключительно быстрый механизм агрегации. Один из представителей таких баз данных — ClickHouse.

Большой проблемой для баз данных, разработанных в прошлом, стала распределенная природа современных приложений. Они не только работают на нескольких серверах, но и зачастую разбросаны по нескольким дата-центрам в разных точках мира. Это грозит тем, что разные части распределенной базы данных могут терять связность и необходимы меры по поддержанию работоспособности и доступности в таких условиях. Как раз для этого предназначена база данных Cassandra.

Это не значит, что у новых NoSQL-баз данных вообще нет недостатков, их много и они настолько существенны, что не позволят вам отказаться от традиционных реляционных баз данных. Просто эксплуатируя NoSQL-базу данных для решения задач, под которые она заточена, можно добиться удивительных успехов. При этом важно не использовать такое решение там, где проявляются слабые стороны того или иного хранилища.

Поэтому современный сайт или приложение может использовать совокупность нескольких хранилищ. Например, мы можем запоминать результат ресурсоемкой операции для ускорения чтения, т. е., использовать кеш. Можем использовать основную базу данных для долговременного хранения. Можем предоставлять пользователям возможность искать данные по ключевому слову или фильтровать их различными способами. Чтобы время от времени перемалывать большие объемы накопленных данных, идеально подходят колоночные базы данных.

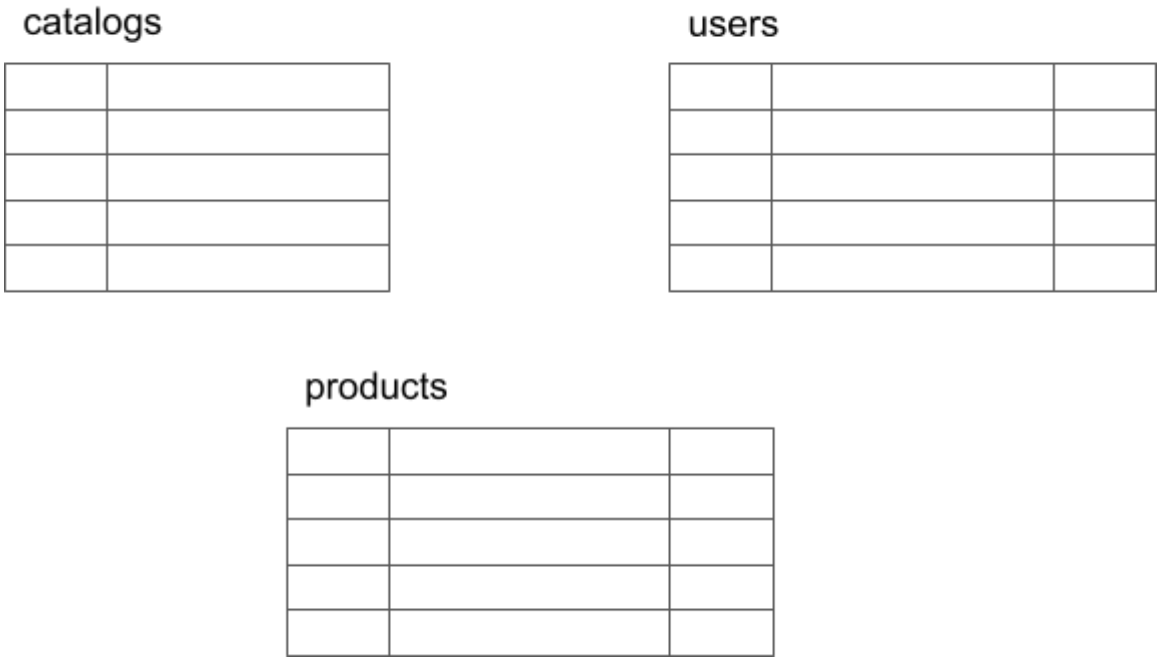
На протяжении всего курса мы будем рассматривать реляционные базы данных на примере MySQL. Последнее занятие нашего курса будет посвящено современным NoSQL-решениям.

Основы реляционных баз данных

Большая часть курса будет посвящена реляционным базам данных. В них информация организована в виде прямоугольных таблиц, разделенных на строки и столбцы, на пересечении которых содержатся значения.

Таблицы

База данных состоит из нескольких таблиц. Каждая таблица имеет уникальное имя, описывающее ее содержимое.



Начнем формировать базу данных, с которой мы будем работать в течение курса. Пусть это будет база данных интернет-магазина компьютерных комплектующих. Ниже представлена таблица catalogs.

id	name	total
1	Процессоры	15
2	Видеокарты	10
3	Материнские платы	24
4	Оперативная память	12

Строки

Каждая горизонтальная строка этой таблицы представляет отдельную физическую сущность — один каталог. Четыре строки таблицы вместе представляют все четыре каталога интернет-магазина. Все данные, содержащиеся в конкретной строке таблицы, относятся к каталогу, который описывается этой строкой.

Столбцы

Каждый вертикальный столбец таблицы catalogs представляет один элемент данных для каждого из каталогов. На пересечении строки и столбца таблицы содержится только одно значение. Например, в

строке, представляющей видеокарты, в столбце name содержится название раздела. В столбце total этой же строки находится значение 10, сообщающее количество доступных для покупки товаров.

Все значения, содержащиеся в одном и том же столбце — данные одного типа. Например, в столбце name содержатся только строки, в столбце total — только числовые значения. У каждого столбца в таблице есть свое имя, которое обычно служит его заголовком. Все столбцы в одной таблице должны иметь уникальные имена, однако разрешается присваивать одинаковые имена столбцам, расположенным в различных таблицах.

На практике такие имена столбцов, как name (имя), id (идентификатор), description (описание) и тому подобные, часто встречаются в различных таблицах одной базы данных. Столбцы таблицы упорядочены слева направо, и их порядок определяется при создании таблицы. В любой таблице всегда есть как минимум один столбец.

В отличие от столбцов, строки таблицы не имеют определенного порядка. Это значит, что если последовательно выполнить два одинаковых запроса для отображения содержимого таблицы, нет гарантии, что оба раза строки будут перечислены в одном и том же порядке. Конечно, можно попросить SQL-запрос отсортировать строки перед выводом, однако порядок сортировки не имеет ничего общего с фактическим расположением строк в таблице.

Пустая таблица

В таблице может содержаться любое количество строк. В том числе и ноль строк, в этом случае таблица называется пустой. Пустая таблица сохраняет структуру, определенную ее столбцами, просто в ней не содержатся данные.

Первичный ключ

Поскольку строки в реляционной таблице не упорядочены, нельзя выбрать строку по ее номеру в таблице. В таблице нет первой, последней или тринадцатой строки.

В правильно построенной реляционной базе данных в каждой таблице есть столбец (или комбинация столбцов), для которого значения во всех строках различны. Этот столбец (или столбцы) называется первичным ключом (primary key) таблицы.

Первичный ключ у каждой строки уникальный. В таблице с первичным ключом нет двух совершенно одинаковых строк.

Таблица, в которой все строки отличаются друг от друга, в математических терминах называется отношением (relation). Именно этому термину реляционные базы данных и обязаны своим названием, поскольку в их основе лежат отношения, т. е., таблицы с отличающимися друг от друга строками.

id	name	total
1	Процессоры	15
2	Видеокарты	10
3	Материнские платы	24
4	Оперативная память	12

Связи между таблицами

В иерархических базах данных довольно легко выстраивать отношения «предок-потомок». В реляционной базе данных происходит отказ от явных связей, однако, отношение «предок-потомок» между категориями и товарными позициями не утеряно.

Оно реализовано в виде одинаковых значений, хранящихся в двух таблицах, а не в виде явного указателя. Таким способом реализуются все отношения, существующие между таблицами реляционной базы данных.

Столбец одной таблицы, значения в котором совпадают со значениями столбца, являющегося первичным ключом другой таблицы, называется внешним ключом (foreign key). Например, здесь на экране внешним ключом выступает столбец category_id. Одним из главных преимуществ реляционных баз данных — возможность извлекать связанные между собой данные, используя эти отношения.

products

id	name	category_id
1	Intel Core i7	1
2	Intel Xeon Silver	1
3	AMD Ryzen 3	1
4	GeForce GTX 1060	2
5	GeForce GTX 1070	2
6	Radeon RX 580	2

categories

id	name
1	Процессоры
2	Видеокарты

Для нас наличие первичного ключа сейчас является чем-то само собой разумеющимся, хотя первые реляционные СУБД его просто не поддерживали.

Транзакции

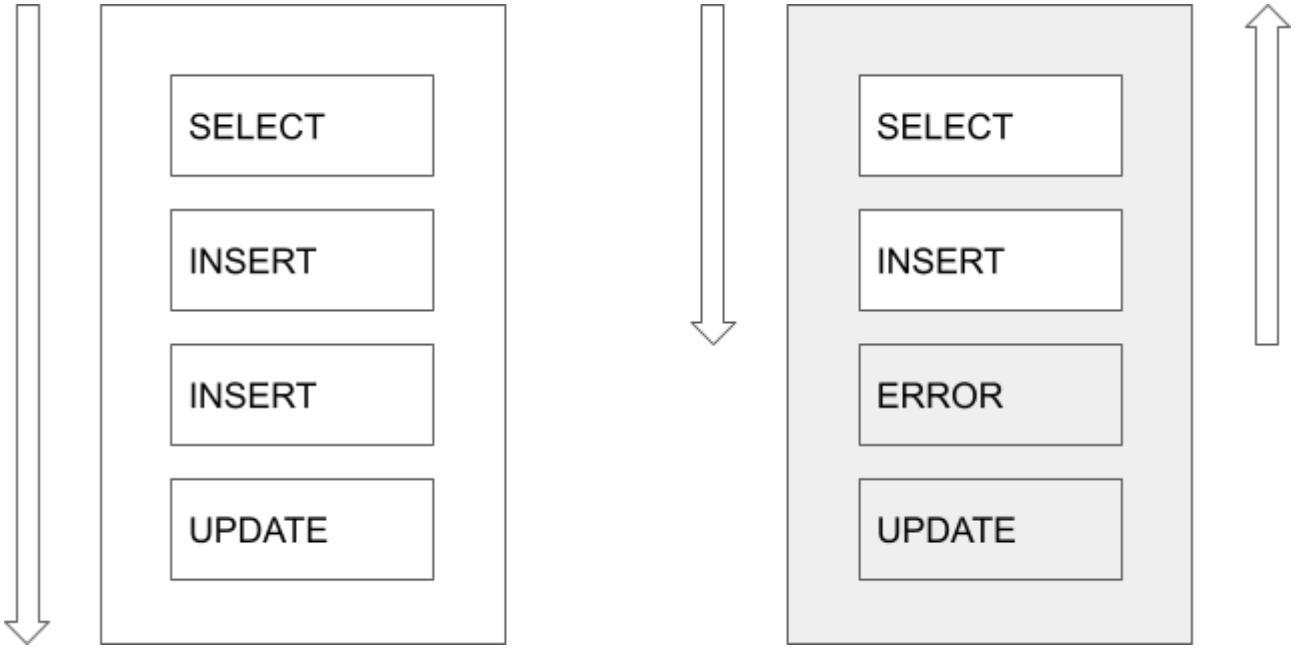
В приложении многое может пойти не так:

- может отказать аппаратное обеспечение;

- может произойти исключительная ситуация внутри приложения, в том числе когда последовательность операций выполнена наполовину;
- разрыв сети может отрезать приложение от базы данных;
- несколько клиентов могут одновременно изменять одни и те же данные.

Все эти проблемы обычно решаются при помощи транзакций.

Транзакции — это способ группировки приложением нескольких операций записи и чтения в одну логическую единицу. По сути все операции записи и чтения в ней выполняются как одна: вся транзакция либо целиком выполняется успешно (с фиксацией изменений) или целиком завершается неудачно (с прерыванием и откатом к исходному состоянию). Транзакции значительно упрощают для приложения обработку ошибок, так как нет нужды заботиться о частичных отказах, когда часть операций завершилась успешно, а часть — нет.



Принцип ACID

Практически все реляционные базы данных поддерживают транзакции, которые остались почти неизменными с самых первых реляционных баз данных.

Появившиеся в конце 2000-х годов нереляционные NoSQL-базы данных ставили целью улучшить положение дел с реляционными базами данных с помощью новых моделей данных, репликации и секционирования. Транзакции стали главной жертвой этого новшества. Многие базы данных нового поколения полностью от них отказались. Или поменяли значение термина: теперь он стал означать намного более слабый набор функциональных гарантий, чем ранее.

Гарантии безопасности, которые обеспечивают транзакции, обычно принято обозначать аббревиатурой ACID — атомарность, согласованность, изоляция и сохраняемость.

Атомарность обозначает реакцию базы данных на сбой, когда в рамках транзакции операции записи выполнены лишь наполовину. Если транзакцию не удается завершить (зафиксировать изменения), то она прерывается и базе данных приходится откатить все уже выполненные в рамках этой транзакции операции записи. Без атомарности сложно было бы понять, какие операции уже выполнены, а какие — нет. Если попробовать выполнить операции повторно, возникает риск, что часть операций будет продублирована.

Согласованность означает выполнение бизнес-правил. Не должно быть отрицательного возраста или записей, которые ссылаются на уже удаленные записи. В системе бухгалтерского учета кредит должен сходиться с дебетом. Если транзакция начинается с состояния базы данных, которое удовлетворяет этим правилам, то после выполнения транзакции база данных должна также удовлетворять им. Таким образом данные переходят из одного корректного состояния в другое, тоже корректное, не оставляя стороннему наблюдателю возможности увидеть недопустимую совокупность промежуточных значений.

Сейчас бизнес-правила очень часто хранятся в приложении, база данных может ничего не знать о сложных правилах бизнес-логики. Она способна проверять лишь некоторые специальные виды правил, например, ограничения внешнего ключа или уникальности. Однако в целом допустимость или недопустимость данных определяется приложением — база данных лишь обеспечивает хранение. Есть мнение, что букву C в эту аббревиатуру добавили для красоты, чтобы получилось красивое и запоминающееся слово acid (кислота).

Изолированность означает, что параллельно выполняемые транзакции изолированы друг от друга — они не могут помешать друг другу. Т. е., каждая транзакция выполняется так, будто она единственная во всей базе данных. База данных гарантирует, что результат фиксации транзакций такой же, как если бы они выполнялись последовательно, одна за другой, хотя в реальности они могут выполняться параллельно.

Задача СУБД — предоставить надежное место для хранения данных.

Сохраняемость — это обязательство базы данных не терять записанных данных, даже в случае сбоя аппаратного обеспечения или фатального сбоя самой базы данных.

Правила довольно неплохие, почему современные NoSQL-базы данных от них отказываются?

С развитием веба мы получили не просто популярные приложения, а приложения, к которым одновременно обращаются тысячи клиентов. Это приводит к проблема масштабируемости баз данных. Даже в небольших реляционных базах данных никуда не деться от операций соединения, которые могут оказаться медленными.

Под транзакциями подразумевают блокировку некоторой части данных, из-за чего они становятся недоступными другим клиентам. При высоких нагрузках это может стать неприемлемым, так как из-за блокировок запросы пользователей выстраиваются в очередь.

Проблема в том, что мы не можем бесконечно долго заниматься вертикальным масштабированием, увеличивая мощность серверов, на которых расположена СУБД. Рано или поздно нам приходится размещать базу данных на нескольких серверах. Делается это при помощи механизма репликации, с которым мы обязательно познакомимся в рамках курса. Таким образом, база данных оказывается на разных машинах или даже в разных дата-центрах.

Как только база данных становится распределенной, нам необходимо обеспечивать ее устойчивость к сетевым сбоям.

CAP-теорема

Противоречия обеспечения согласованности и распределенных систем часто описывает CAP-теорема, которую в 2000 году сформулировал Эрик Брюер. Теорема касается трех взаимосвязанных требований, существующих в большой распределенной системе данных:

Согласованность означает, что все клиенты должны прочитать одно и то же значение в ответ на один и тот же запрос, даже если одновременно с этим производятся обновления.

Доступность соответствует тому, что все клиенты базы данных всегда имеют возможность читать и записывать данные. Мы не можем ждать для этого завершения каких-либо операций.

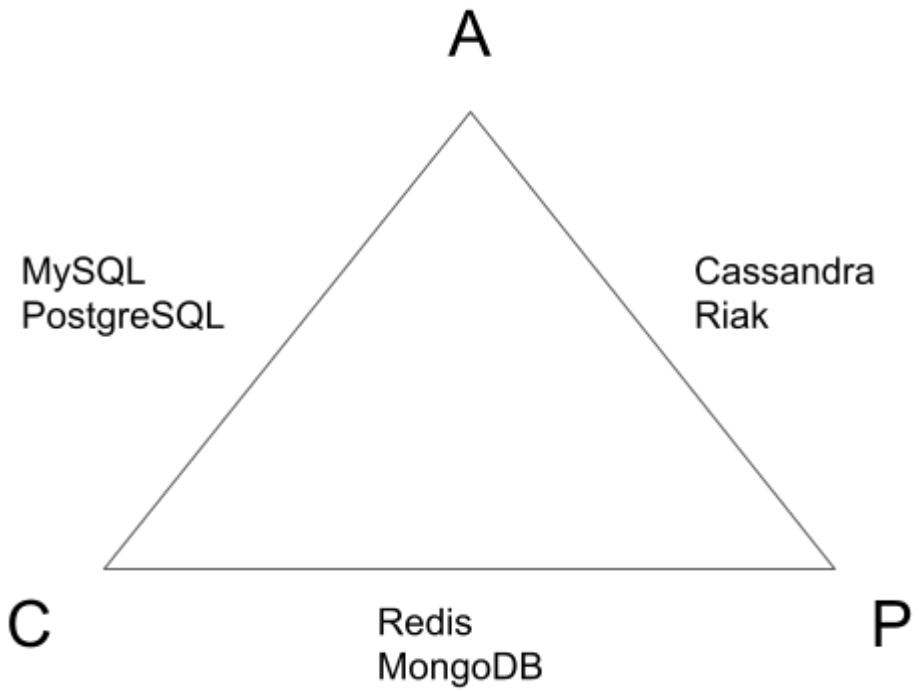
Устойчивость к разделению означает, что базу данных можно разделить между несколькими машинами; более того, она продолжит работу даже в случае отказа сегмента сети.

CAP-теорема утверждает, что в любой системе можно гарантированно обеспечить выполнение только двух из этих трех требований. Это аналог поговорки «Программа может быть хорошей, быстрой и дешевой — выбирай любые два свойства».

Чем большая согласованность требуется от системы, тем меньше будет ее устойчивость к разделению.

Однако в распределенной системе сетевые сбои очень вероятны. Мы не можем избежать задержек и потерь пакетов, обрыва связанности, вплоть до того, что часть распределенной базы данных оказывается изолированной от другой. Поэтому избежать узла устойчивости к разделению в современных веб-приложениях почти невозможно. Таким образом, любая СУБД может располагаться лишь на одной грани CAP-треугольника. Точнее это зависит от режима работы СУБД, так как часто многие СУБД могут располагаться на той или иной грани.

Традиционные реляционные СУБД, спроектированные для работы на одном компьютере, обеспечивают согласованность и доступность. Они с трудом могут обеспечивать устойчивость к разделению. Как видите, MySQL, PostgreSQL, реляционные базы данных находятся на грани CA (согласованность и доступность). Существуют сторонние решения и от Google и от компании Percona, которые позволяют перевести базу данных MySQL на грань CP, т. е., обеспечить согласованность данных и устойчивость к разделению. Однако в этом случае страдает доступность: мы вынуждены дожидаться операции обновления базы данных.



Существует большое количество самых разнообразных NoSQL-решений, которые находятся на разных гранях этого треугольника. Они специально созданы для решения проблемы CAP-теоремы.

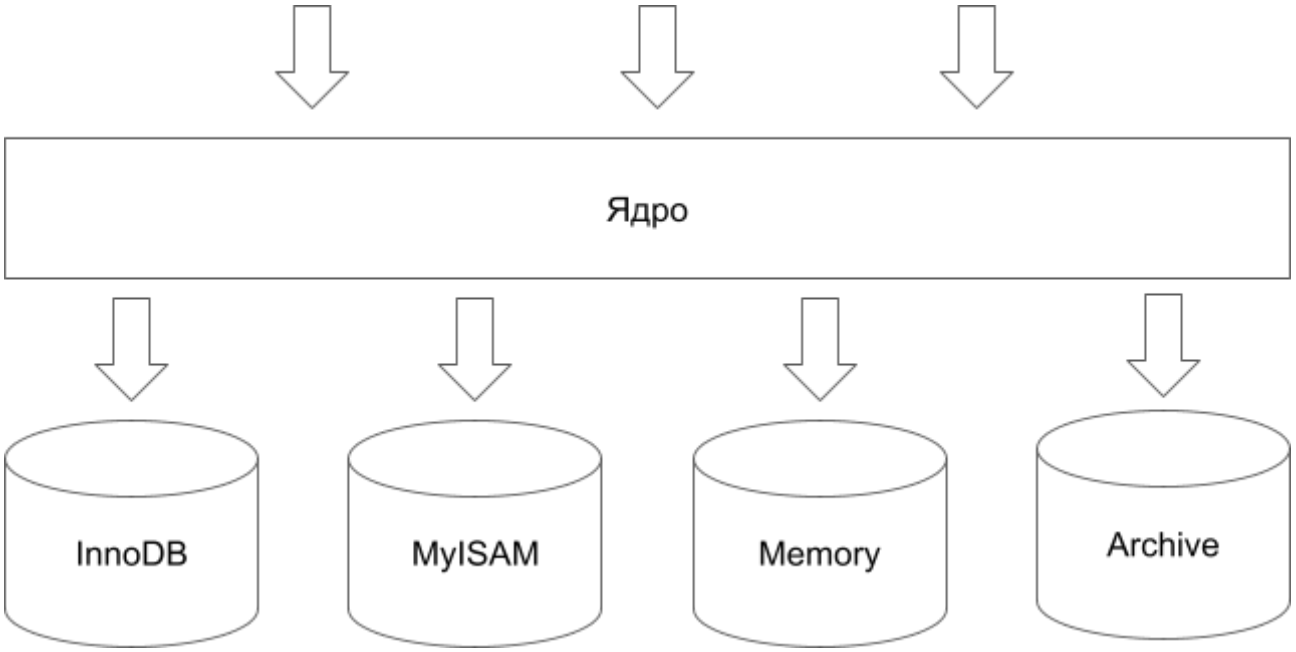
Укрупнение проектов, распределенная структура современных веб-решений приводит к тому, что требуются новые решения, новые базы данных, которые, обеспечивая меньшую согласованность, остаются устойчивыми в разделению.

Именно этим вызван интерес к современным NoSQL-базам данных, которые компенсируют слабую приспособленность реляционных баз данных к распределенной работе.

В том числе поэтому в современных приложениях традиционную реляционную СУБД зачастую сопровождают одна или несколько NoSQL-баз данных. Реляционная СУБД выступает в качестве основного хранилища, NoSQL-базы данных решают специализированные задачи или обеспечивают сетевую связность в условиях нестабильной сети.

Архитектура MySQL

MySQL на сегодняшний день самая популярная база данных с открытым кодом. По популярности ее превосходит только коммерческая СУБД Oracle. MySQL используется во множестве проектов, наверное, самый известный из них — электронная энциклопедия Wikipedia. Существует множество форков этой базы данных: Percona, MariaDB. Мы будем знакомиться с оригинальной версией MySQL.



Архитектуру MySQL можно условно разбить на две части: это ядро, сама база данных и движки, которые реализуют тот или иной механизм баз данных. На экране представлено несколько движков, которые могут использоваться MySQL. По умолчанию используется InnoDB. Более подробно каждый из движков будет рассмотрен на следующих уроках.

Такая архитектура позволяет разрабатывать базу данных усилиями нескольких команд. Одна команда может сосредоточиться на ядре, другая — на каком-либо отдельном движке. Например, движок InnoDB долго разрабатывался отдельной компанией.

MySQL построена по клиент-серверной технологии: и клиент, и сервер являются программами, которые могут быть расположены на разных компьютерах или на том же самом.

Сервер хранит и обслуживает базы данных, клиенты шлют ему запросы на языке SQL и получают в ответ результирующие таблицы с данными.

Клиенты MySQL

Клиентами MySQL-сервера могут выступать консольные утилиты, специализированные программы с графическим интерфейсом и, конечно же, программный код.

Например, перед вами бесплатная программа DBeaver, которая позволяет подключаться к различным базам данных и доступен в основных операционных системах (Windows, Mac OS X, Linux).

Однако в течение курса мы будем использовать в основном консольный клиент MySQL, прибегая к графическому интерфейсу только для удобства просмотра объемных выборок.

Дело в том, что именно с консольными клиентами вам чаще придется иметь дело на удаленных серверах, доступ к которым может быть ограничен, например, подсетью дата-центра или локальной машиной.

Владея командной строкой, вы легко сможете освоить любой графический клиент, однако в обратную сторону это не работает. Навигация по базам данных, таблицам, системным параметрам в консольных утилитах требует ввода команд. В графических интерфейсах эти команды скрыты за выбором раскрывающихся и обычных списков, прокруткой.

Кроме того, консольные утилиты — зачастую самый быстрый способ выполнения больших пакетных операций. Например, у вас имеется база данных на два десятка гигабайт и вам необходимо снять резервную копию или, наоборот, развернуть ее на другом сервере. Зачастую консольные утилиты из дистрибутива **mysql** — это самый быстрый способ выполнения задачи.

Консольный клиент MySQL

Клиент доступен из командной строки при помощи утилиты **mysql**. В Windows для установки необходимо отметить соответствующую галку при установке MySQL или прописать путь к bin-папке MySQL в переменной окружения PATH. Как вариант, просто найти папку **bin** из дистрибутива **mysql**, перейти в нее и запустить MySQL прямо в ней.

Для соединения с сервером мы набираем название клиента MySQL, далее после ключа **-u** указываем имя MySQL-пользователя и ключ **-p**, чтобы клиент запросил у нас пароль. Сразу после установки пользователь, как правило, один — **root**. Поэтому для получения доступа к серверу достаточно набрать команду:

```
mysql -u root -p
```

Итак, у нас выводится приглашение **mysql>**, после которого можно набирать команды, адресованные серверу. Например, выведем фразу Hello world!:

```
SELECT "Hello world!";
```

Команды завершаются точкой с запятой. Если команда не убирается на одной строке, возможен переход на другую строку после нажатия клавиши Enter — запрос отправляется серверу только после того, как консольный клиент MySQL встретит символ точки с запятой.

Обратите внимание на результирующую таблицу. В первой строке таблицы с результатами содержатся заголовки столбцов, а в следующих строках — ответ сервера на запрос. Обычно заголовками столбцов становятся имена, полученные из таблиц базы. Если же извлекается не столбец таблицы, а значение выражения (как это происходит в приведенном выше примере), MySQL дает столбцу имя запрашиваемого выражения. После этого сообщается количество возвращаемых строк (1 row in set — одна строка в результате) и время выполнения запроса.

Для ввода ключевых слов можно использовать любой регистр символов, например, **select** можно набрать маленькими буквами:

```
select "Hello world!";
```

Если запрос не завершен, вид командной строки после ввода меняется. Таким образом, программа MySQL показывает, что завершенного выражения она пока что не получила и ожидает его полного ввода.

```
mysql> SELECT "Hello
    "> wold!";
```

Введенные ранее команды не обязательно вводить снова, для этого достаточно их вызвать клавишами <↑> и <↓> (стрелка вверх и стрелка вниз).

Если сервер установлен на удаленном хосте, соединиться с ним можно, указав IP-адрес или домен удаленного сервера в параметре **-h**, в параметре **-P** мы можем указать порт.

```
mysql -u root -h 192.168.0.10 -P 3306
```

Внутренние команды клиента MySQL

Утилита **mysql** поддерживает различные команды, представленные в таблице ниже.

Команда	Сокращение	Описание
USE	\u	Выбор базы данных
SOURCE	\.	Выполнение SQL-команды из файла
SYSTEM	\!	Выполнение команды операционной системы
STATUS	\s	Вывод информации о состоянии сервера
EXIT	\q	Выход
	\G	Вывод результата в вертикальном формате

Иногда результирующие таблицы содержат слишком большое число столбцов, в результате их структура нарушается и воспринимать информацию практически невозможно.

Да, в этом случае удобно воспользоваться графическим клиентом, но это не всегда возможно. Выход из ситуации — использование так называемого вертикального режима вывода.

```
SELECT * FROM mysql.user LIMIT 1\G
```

В таком режиме каждая строка таблицы выводится при помощи отдельной таблицы, в первом столбце которой перечисляются имена столбцов, а во втором — их значения для данной строки.

Для этого используется внутренняя команда клиента **\G**. Повторимся: это команда клиента — то есть не часть SQL-запроса, сервер о ней ничего не знает. Поэтому отправлять ее из других клиентов или из собственного программного кода не следует — запрос будет выполнен с ошибкой.

Давайте посмотрим, как работают другие команды клиента MySQL. Например, получить текущий статус сервера можно при помощи команды **STATUS**. У этой команды есть сокращенный вариант **\s**.

Выполнить системную команду можно при помощи команды **SYSTEM**. Ниже мы запрашиваем список файлов в текущей директории:

```
mysql> SYSTEM
```

В Windows команда **SYSTEM** в MySQL-клиенте не поддерживается. Для команды **SYSTEM** существует сокращение **\!**.

Точно такая же ситуация с командой выхода **EXIT**: мы можем использовать вместо нее сокращение **\q**.

Некоторые SQL-команды довольно длинные и в них легко ошибиться. Повторный набор может быть довольно утомителен и неудобен.

Вместо этого команды можно набирать в файлах, используя для этого любой текстовый редактор. После того, как SQL-файл сформирован, его можно выполнить при помощи команды **SOURCE**.

Давайте создадим SQL-запрос, выводящий фразу Hello world! (например, в редакторе DBeaver):

Пример: **hello.sql**.

```
SELECT "Hello world!";
```

Давайте сохраним файл с именем **hello.sql** в какой либо папке. После этого в директории с файлом можно запустить MySQL-клиент в диалоговом режиме и выполнить файл при помощи команды **SOURCE**:

```
mysql> SOURCE hello.sql
```

Таким образом, если вы совершите ошибку в запросе, его можно поправить в любом текстовом редакторе и выполнить повторно.

Конфигурационный файл .my.cnf

При входе в диалоговый режим утилиты **mysql** мы каждый раз вынуждены вводить параметры. Если этого не делать, то попытка входа завершится неудачей. Чтобы каждый раз не вводить параметры, можно воспользоваться специальным конфигурационным файлом **.my.cnf**. Этот файл обычно размещается в домашней директории пользователя (это касается их Windows и UNIX-подобных операционных систем).

Файл начинается с точки, в UNIX-системах ведущей точкой помечаются скрытые файлы. Так как MySQL изначально разрабатывалась в UNIX, эта традиция распространяется в том числе на Windows-пользователей.

При запуске любой консольной утилиты из дистрибутива MySQL, если параметры не задаются явно, они будут извлекаться из этого файла.

Пример: **.my.cnf**.

```
[mysql]
user=root
password=
```

Мы указываем параметры в полной форме, а не в сокращенной, как при запуске консольного клиента. Если вы указываете здесь настоящий пароль, важно назначить потом файлу права для чтения только вашему пользователю.

После того, как файл готов, вы получаете возможность запуска MySQL-клиента без указания логина и пароля:

```
mysql
```

Создание дампа

Одна из часто встречающихся задач — перенос баз данных с одного сервера на другой или просто создание резервной копии базы данных.

Основным инструментом для создания SQL-дампов служит утилита **mysqldump**. Утилита создает текстовый файл с SQL-инструкциями, воспроизведя которые на другом сервере, вы сможете полностью воссоздать базу данных.

Конфигурационный файл **.my.cnf** будет действовать и на эту утилиту. Для этого секцию **[mysql]** следует поменять на **[client]**. После этого директивы, которые указаны в файле **.my.cnf**, будут распространяться в том числе и на **mysqldump**.

Как правило, для создания резервной копии в качестве параметра утилите **mysqldump** указывается какая-то база данных.

У нас пока нет своих собственных баз данных — это тема следующего ролика, но мы можем сделать дамп системной базы данных **mysql**, которую **MySQL** использует для хранения собственных настроек, имен пользователей, хранимых процедур и т. д.

```
mysqldump mysql > mysql.sql
```

Если мы захотим развернуть дамп, то должны уже воспользоваться утилитой **mysql**. Мы не будем разворачивать дамп системной базы данных, чтобы ничего не повредить, вместо этого мы воспользуемся безобидным файлом **hello.sql**, который не вносит изменений.

```
mysql < hello.sql
```

Управление базами данных

Создание базы данных средствами SQL осуществляется при помощи оператора **CREATE DATABASE**. Давайте создадим базу данных **shop**.

```
mysql> CREATE DATABASE shop;
```

Чтобы посмотреть список существующих баз данных, мы можем воспользоваться оператором.

```
mysql> SHOW DATABASES;
```

Как видим, возвращается список баз данных. Причем помимо нашей базы данных **shop**, можно видеть уже существующие системные базы данных. С базой данных **mysql** мы уже знакомы. Базы данных **performance_schema** и **sys** предназначены для исследования параметров производительности. Информационная схема **information_schema** предоставляется СУБД для описания структуры данных, мы немного затронем ее в текущем ролике.

База данных в MySQL — это подкаталог на жестком диске, который можно физически обнаружить в каталоге данных. Это каталог операционной системы, где физически располагаются базы данных, такие как таблицы, логи, журналы транзакций и т. п.

Узнать, где он расположен, можно при помощи запроса:

```
SHOW VARIABLES LIKE 'datadir';
```

Давайте выйдем из клиента **mysql** и перейдем внутрь каталога (у вас путь может отличаться).

```
cd /usr/local/var/mysql/
```

Здесь находится множество файлов, со многими из них мы с вами познакомимся во время курса. Сейчас нас будут интересовать директории. Каждая директория здесь — база данных, в том числе и созданная нами директория **shop**.

Здесь нет только одной базы данных, соответствующей информационной схеме **information_schema**, так как это виртуальная база данных, которая формируется на лету.

Давайте заглянем внутрь нашей базы данных **shop**. Для этого переходим внутрь каталога.

```
cd shop
```

Там мы можем обнаружить единственный файл **db.opt**, если мы заглянем внутрь, то увидим, что это текстовый файл, содержащий сведения о кодировке по умолчанию.

В курсе будем работать с единственной кодировкой — UTF8.Именно она по умолчанию выставляется в последних версиях MySQL, поэтому подробно на кодировках останавливаться не будем.

Если база данных — это подкаталог в каталоге данных, то мы можем создать его и руками. Давайте это сделаем. Для этого можно воспользоваться UNIX-командой **cp** и параметром **-r** для рекурсивного копирования всего содержимого директории.

Вы можете воспользоваться любым способом копирования, в том числе и графическим интерфейсом вашей операционной системы.

```
cp -r shop foo
```

Итак, мы создали новую базу данных **foo** путем копирования ее из базы данных **shop**. Давайте вернем в консоль **mysql** и убедимся, что у нас появилась новая база данных. Запускаем команду **SHOW DATABASE**, и видим новую базу данных **foo**.

```
mysql> SHOW DATABASES;
```

Разумеется, не следует прибегать к такому приему на практике, так как в разных СУБД организация баз данных сильно отличается. Прием создания базы данных через формирования директории не сработает у вас в другой базе данных. А вот SQL-запрос **CREATE DATABASE** стандартен и будет работать во всех базах данных. Кроме того, варварский способ копирования директории сработает только в отношении пустой базы данных. Если вы поступите так в отношении таблиц, особенно при работающем сервере, вы почти наверняка их повредите. Для копирования баз данных лучше прибегать либо к SQL-дампам, либо к специализированным утилитам.

Удаление баз данных можно осуществить и штатными средствами при помощи оператора **DROP DATABASE**, за которым следует имя базы данных:

```
DROP DATABASE foo;
SHOW DATABASES;
```

Кстати, так как имя базы данных — это имя каталога, чувствительность к регистру определяется конкретной операционной системой. Так, операционной системе Windows имена с заглавной и прописной буквы будут обозначать одну и ту же базу данных, в то время как в UNIX-подобной операционной системе это будут две разные базы данных.

При попытке создания уже существующей базы данных возвращается ошибка:

```
CREATE DATABASE shop;
ERROR 1007 (HY000): Can't create database 'shop'; database exists
```

Часто такое поведение нежелательно, особенно при работе с объёмными SQL-дампами, которые выполняются в пакетном режиме. Для предотвращения такой ошибки оператор **CREATE DATABASE** можно снабдить конструкцией **IF NOT EXISTS**. Она помогает создать базу данных, если она еще не существует, если же существует — никаких действий не производится.

```
DROP DATABASE shop;
Query OK, 0 rows affected (0.00 sec)
CREATE DATABASE IF NOT EXISTS shop;
Query OK, 1 row affected (0.00 sec)
CREATE DATABASE IF NOT EXISTS shop;
Query OK, 0 rows affected (0.00 sec)
```

Как видно, после удаления базы данных **shop** первый оператор **CREATE DATABASE** возвращает ответ, в котором сообщается, что было произведено одно действие (1 row affected), т. е., база данных создается.

Второй оператор **CREATE DATABASE** возвращает сообщение, что запрос не произвел ни одной операции (0 rows affected), т. е., база данных не создается, однако и ошибка не выдается.

Точно такой же механизм разработан для оператора **DROP DATABASE**: добавление ключевого слова **IF EXISTS** удаляет базу данных, если она существует, и не производит никаких действий, если база данных отсутствует.

```
DROP DATABASE IF EXISTS shop;
Query OK, 1 rows affected (0.03 sec)
DROP DATABASE IF EXISTS shop;
Query OK, 0 rows affected (0.00 sec)
```

Приступая к работе с базами данных, следует выбрать БД по умолчанию, к таблицам которой будет осуществляться обращение. Каждая клиентская программа решает эту задачу по-своему. Например, в консольном клиенте **mysql** выбрать новую базу данных можно при помощи команды **USE**.

```
USE shop
```

Если текущая база данных не выбрана, то обращение по умолчанию будет заканчиваться сообщением об ошибке — **No database selected** (База данных не выбрана).

```
SHOW TABLES;
ERROR 1046 (3D000): No database selected
```

Можно не выбирать текущую базу данных, однако в этом случае во всех операторах придётся явно указывать, какая база данных должна использоваться.

```
SHOW TABLES FROM mysql;
```

При обращении к таблицам потребуется явное указание префикса базы данных. Это правило действует и в отношении таблиц:

```
SELECT mysql.User.User, mysql.User.Host FROM mysql.User;
```

Здесь мы из таблицы **User** базы данных **mysql** извлекаем поля **User** и **Host**. Такие составные имена называются полными квалификационными именами столбцов и таблиц. В результате явного обращения к базам данных, SQL-запросы становятся достаточно громоздкими и менее гибкими, так как смена имени базы данных требует изменения SQL-запроса.

Другой способ выбрать базу данных при использовании консольного клиента **mysql** — указание имени базы данных в конце списка параметров. Давайте при запуске утилиты **mysql** в качестве текущей базы данных выберем **shop**.

```
mysql -u root shop
```

Создание таблиц

Создать таблицу внутри базы данных можно при помощи оператора **CREATE TABLE**. Мы еще будем более подробно рассматривать его в следующих роликах.

```
CREATE [TEMPORARY] TABLE [IF NOT EXISTS] users
[(create_definition,...)]
[table_options] [select_statement]
```

Начинается оператор ключевым словом **CREATE TABLE**, за которым следует имя таблицы, а в круглых скобках — параметры столбцов:

```
CREATE TABLE users (k INT);
Query OK, 0 rows affected (0.11 sec)
CREATE TABLE users (k INT);
ERROR 1050: Table 'users' already exists
```

Повторное создание таблицы приводит к возникновению ошибки «Таблица users уже существует». Однако необязательная конструкция **IF NOT EXISTS** сообщает, что таблицу необходимо создавать только в том случае, если она не существует, в противном случае никаких действий предпринимать не следует.

```
CREATE TABLE IF NOT EXISTS users (k INT);
Query OK, 0 rows affected (0.02 sec)
```

Такое поведение базы данных бывает удобным при выполнении большого количества SQL-инструкций в пакетном режиме, например из файла. Ошибка вызовет остановку выполнения всего пакетного файла, несмотря на то, что дальнейшее его выполнение в такой ситуации не приведет к сбою.

Для просмотра структуры таблицы можно использовать оператор **DESCRIBE**.

```
DESCRIBE users;
```

Каждая строка в результирующей таблице соответствует отдельному столбцу.

Вместо имени столбца можно использовать LIKE-шаблон с применением спецсимволов **_** (заменяет собой любой один символ) и **%** (заменяет собой любое количество символов).

```
DESCRIBE users 'us_r';
```

Ниже приводится запрос, который выводит структура столбцов, имена которых заканчиваются на символ **'e'**. Следует обратить внимание, что для использования шаблона строку необходимо заключать в одинарные кавычки, в противном случае сервер MySQL сообщит об ошибочном SQL-запросе.

```
DESCRIBE users '%e';
```

Однако, следует помнить, что это не стандартный оператор, другие базы данных их не реализуют и его возможности довольно ограничены.

Информационная схема

Операторы **SHOW** и **DESCRIBE** являются нестандартными, другие базы данных, отличные от MySQL, их могут не предоставлять. Более того, возможности этих операторов довольно ограничены.

Каждая СУБД имеет системную базу данных, в которой хранятся учетные записи, таблицы привилегий и другая информация, необходимая для управления СУБД. В MySQL такая системная база данных носит название **mysql**. Структура системной базы данных для разных СУБД отличается: для того, чтобы унифицировать процесс обращения к системной базе данных, вводится специальный набор представлений, оформленных в виде базы данных **INFORMATION_SCHEMA**, которая доступна каждому клиенту MySQL.

База данных **INFORMATION_SCHEMA** является виртуальной и располагается в оперативной памяти — для неё нет физического соответствия на жёстком диске, как для других баз данных. Это означает, что невозможно выбрать базу данных **INFORMATION_SCHEMA** при помощи оператора USE, как и выполнить по отношению к таблицам этой базы данных запросы с участием операторов **INSERT**, **UPDATE** и **DELETE**.

Допускается использование только оператора **SELECT** (более подробно мы будем его изучать в следующих роликах).

```
SELECT * FROM INFORMATION_SCHEMA.SCHEMATA
```

Операторы **SHOW** и **DESCRIBE** короче, но не закреплены в стандарте SQL, они работают только в MySQL. Информационную схему обязаны реализовывать все реляционные СУБД, поддерживающие язык запросов SQL. С таблицами информационной схемы можно работать как с таблицами любой базы данных. Например, чтобы извлечь список таблиц базы данных **shop**, можно воспользоваться следующим запросом:

```
SELECT * FROM INFORMATION_SCHEMA.TABLES WHERE TABLE_SCHEMA = 'shop';
```

Информационной схеме будет посвящен отдельный ролик.

Документация

Клиент **mysql** поддерживает команду **HELP**, при помощи которой можно ознакомиться с документацией прямо в консоли.

```
HELP DESCRIBE;
```

Однако гораздо удобнее воспользоваться онлайн-документацией на официальном сайте MySQL: https://dev.mysql.com/doc/refman/5.7/en/.

Помимо онлайн-варианта, можно загрузить документацию в виде PDF-файла или zip-архива.

Дополнительные материалы

Базам данных вообще и MySQL в частности посвящено большое количество литературы разной степени сложности. Помимо книг, посвященных СУБД MySQL, следует обращать внимание на книги, которые специализируются на языке запросов SQL. Это стандартный специализированный язык для общения с реляционными базами данных. Больше половины курса будет посвящена именно ему, почти все его элементы одинаковы для всех баз данных. Существуют исключения, вроде команд **SHOW** и **DESCRIBE**, которые могут быть реализованы в SQL-диалекте одной базы данных и отсутствовать в другой.

1. Шварц Б., Зайцев П., Ткаченко В., Заводны Дж., Ленц А., Бэллинг Д. MySQL. Оптимизация производительности, 2-е издание. — Пер. с англ. — СПб.: Символ-Плюс, 2010. — 832 с.
2. Чарльз Белл, Мэтс Киндал и Ларс Талманн. Обеспечение высокой доступности систем на основе MySQL / Пер. с англ. — М. : Издательство "Русская редакция"; СПб. : БХВ-Петербург, 2012. — 624 с.
3. Чаллавала Ш., Лакхатария Дж., Мехта Ч., Патель К. MySQL 8 для больших данных. — М.: ДМК Пресс, 2018. — 226с.
4. Поль Дюбуа. MySQL. — Пер. с англ. — М.: ООО "И.Д. Вильямс", 2007. — 1168 с.
5. Поль Дюбуа. MySQL. Сборник рецептов. — Пер. с англ. — М.: Символ-Плюс, 2004. — 1056 с.
6. Кузнецов М.В., Симдянов И.В. MySQL на примерах. — СПб.: БХВ-Петербург, 2007. — 592с.

7. Кузнецов М.В., Симдянов И.В. MySQL 5. — СПб.: БХВ-Петербург, 2006. — 1024с.
8. Дейт К. Дж. Введение в системы баз данных, 8-е издание.: Пер. с англ. — М.: Издательский дом "Вильямс", 2005. — 1328 с.
9. Кляйн К., Кляйн Д., Хант Б. SQL. Справочник, 3-е издание. — Пер. с англ. — СПб: Символ-Плюс, 2010. — 656с.
10. Линн Бейли. Head First. Изучаем SQL. — СПб.: Питер, 2012. — 592 с.
11. Гровф, Джеймс Р., Вайнберг, Пол Н., Оппель, Эндрю Дж. SQL: полное руководство, 3-е изд. : Пер. с англ. — М.: ООО "И.Д. Вильямс", 2015. — 960 с.
12. Дейт К. Дж. SQL и реляционная теория. Как грамотно писать код на SQL. — Пер. с англ. — СПб.: Символ-Плюс, 2010. — 480 с.
13. Карвин Б. Программирование баз данных SQL. Типичные ошибки и их устранение. — Рид Групп, 2011. — 336 с.
14. Клеппман М. Высоконагруженные приложения. Программирование, масштабирование, поддержка. — СПб.: Питер, 2018. — 640 с.: ил.
15. Редмонд Эрик , Уилсон Джим Р. Семь баз данных за семь недель. Введение в современные базы данных и идеологию NoSQL. — М: ДМК Пресс — 384с.
16. Фаулер, Мартин, Садаладж, Прамодкумар Дж. NoSQL: новая методология разработки нереляционных баз данных. — Пер. с англ. — М.: ООО "И.Д. Вильямс", 2013. — 192 с.
17. Робинсон Ян, Вебер Джим, Эифрем Эмиль. Графовые базы данных: новые возможности для работы со связанными данными. — 2-е изд. — М.: ДМК Пресс, 2016. — 256 с.
18. Карпентер Д., Хьюитт Э. Cassandra. Полное руководство. 2-е изд. — М.: ДМК Пресс, 2017. — 400 с.
19. Бэнкер Кайл. MongoDB в действии. — М.: ДМК Пресс, 2017. — 394с.
20. https://redis.io/documentation.