ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ КАЗЕННОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«МОСКОВСКИЙ УНИВЕРСИТЕТ МИНИСТЕРСТВА ВНУТРЕННИХ ДЕЛ
РОССИЙСКОЙ ФЕДЕРАЦИИ ИМЕНИ В.Я. КИКОТЯ»

# БАЗЫ ДАННЫХ

Учебное пособие

Москва
2020

**ББК 32.972.134**
**Б17**

Рецензенты:
старший преподаватель кафедры информационной безопасности Воронежского института МВД России
кандидат технических наук **С. В. Зарубин**; заместитель начальника полиции (по оперативной работе) УВД по СВАО ГУ МВД России по г. Москве **А. С. Благоразумов**

Коллектив авторов:
П. В. Орехов, А. Г. Большунов, Д. В. Галиев, А. О. Тычкова

**Базы данных** : учебное пособие / [П. В. Орехов и др.]. – М. : Московский университет МВД России имени В.Я. Кикотя, 2020. – 104 с.
ISBN 978-5-9694-0874-6

**Б17**

В учебном пособии подробным образом рассмотрены архитектура MySQL, а также вопросы установки, настройки и администрирования системы управления базами данных. Выделены основные параметры управления политикой безопасности СУБД. Описаны технологии резервного копирования и восстановления баз данных. Проработаны аспекты обеспечения безошибочности выполнения транзакций в различных базах данных.

Предназначено для курсантов и слушателей образовательных организаций системы МВД России.

ББК 32.972.134

**ISBN 978-5-9694-0874-6**

# ОГЛАВЛЕНИЕ

# ГЛАВА 1. АРХИТЕКТУРА MySQL

Архитектура MySQL очень отличается от архитектур иных серверов баз данных, что делает эту систему управления базами данных (далее – СУБД) полезной для одних целей, но одновременно неудачным выбором для других. Система MySQL не идеальна, но адаптирована для работы в таких сложных средах, как веб-приложения. Гибкость системы проявляется во многом. Например, в возможности настроить работу на различных аппаратных средствах и поддерживаемых типах данных. Кроме того, MySQL предоставляет встроенные приложения, хранилища данных, программное обеспечение для индексации и распространения контента, надежные системы и управление транзакциями в режиме реального времени.

Чтобы использовать MySQL максимально эффективно, необходимо понять ее структуру. Одной из самых важных особенностей данной системы является ее архитектура подсистемы хранения, которая выполняет обработку поступающих запросов и иных серверных команд отдельно от операций хранения и извлечения данных. Данная система является актуальной, поскольку при решении различных задач необходимо выбирать не только способ хранения данных, но и оптимизировать производительность базы данных, основные характеристики и пр.

В этой главе описаны основные различия между подсистемами хранения, почему они так важны, рассмотрены этапы развития MySQL, а также приведены сравнительные результаты производительности систем различных версий и перечислены эталонные тесты оценки производительности баз данных.

## 1.1. Логическая архитектура MySQL

За многие годы процесс разработки и выпуска MySQL претерпел существенные изменения, но сейчас приобрел определенную стабильность. Oracle (производитель программного обеспечения – прим. авт.) периодически выпускает релизы промежуточных этапов разработки с предварительным просмотром функций, которые в конце концов будут включены в следующий GA-релиз. Они предназначены для тестирования и получения обратной связи, а не для использования на производстве, но Oracle заявляет, что они высокого качества и готовы к выпуску в любое время. Компания также периодически выпускает лабораторные превью, которые представляют собой специальные сборки, содержащие только избранные функции, для оценки заинтересованными сторонами.

MySQL остается программным продуктом с открытым исходным кодом и имеет GPL (General Public License, универсальная общедоступная лицензия), при этом полный исходный код (конечно, за исключением коммерчески лицензированных плагинов) доступен лишь сообществу разработчиков. Разработчики Oracle понимают, что было бы неразумно предоставлять разные версии сервера сообществу. MySQL AB пробовала поступать подобным образом, но получилось, что ее клиенты лишились тестирования и обратной связи от пользователей. Это противоречило интересам корпоративных клиентов, поэтому такая практика была прекращена.

Теперь, когда Oracle выпускает некоторые серверные плагины только для бизнес-клиентов, MySQL целиком и полностью соответствует так называемой модели с открытым ядром. Для тех пользователей, которые действительно нуждаются в коммерческих лицензированных плагинах, они вполне прием-

лемы. Коммерческие плагины не ограничивают функциональность модели разработки – сервер самодостаточен и без них. Ценно то, что Oracle создает большинство функций в виде плагинов. Если бы функции были встроены прямо на сервере без API, выбора бы не было: вы получили бы одну реализацию и были бы ограничены в возможности скомпоновать что-то, что вам подходит лучше. Например, если Oracle все же выпустит функцию полнотекстового поиска InnoDB в качестве плагина, у нее будет возможность использовать тот же API (интерфейс прикладного уровня) для разработки аналогичного плагина для Sphinx или Lucene, который многие пользователи могут посчитать более полезным. Таким образом, MySQL становится одной из самых распространенных систем в мире.

Чтобы понять принципы работы сервера MySQL необходимо понимать, каким образом происходит взаимодействие между его компонентами. Логический вид данного взаимодействия представлен на рисунке.

Логическая архитектура MySQL условно разделена на три уровня. На верхнем уровне содержатся различные службы, которые не являются уникальными для данной системы. Эти службы предназначены для обеспечения идентификации, безопасности, поддержки соединений и т. д. и необходимы для большинства сетевых клиент-серверных или серверных инструментов.

Средний уровень более интересен для изучения. На данном уровне располагается основная часть «интеллекта» MySQL, а именно: математические функции, кеширование, шифрование, анализ запросов, оптимизация. Кроме того, на данном уровне реализуются все независимые от подсистемы хранения данных функции, такие как триггеры, хранимые процедуры и представления.


*Рис. Логическая архитектура MySQL*

На третьем уровне содержатся подсистемы хранения данных. Их функционал заключается в сохранении и извлечении всех данных, хранимых в MySQL. Как и любая другая файловая система GNU/Linux, данные подсистемы хранения данных имеет свои плюсы и минусы. Стоит отметить, что сервер обеспечивает взаимодействие с файловыми системами с помощью API подсистемы хранения данных. Этот интерфейс стирает разницу между различными подсистемами хранения данных и делает их практически идентичными и незаметными на уровне запросов. Кроме того, API содержит большое количество различных низкоуровневых функций, которые выполняют операции типа «начать транзакцию», «извлечь строку с данным ключом». Данные подсистемы не производят анализ SQL-кода, а только отправляют ответ на исходящие от сервера запросы.

*Управление соединениями и безопасность.* Для каждого клиентского соединения выделяется отдельный поток внутри

процесса сервера. Запросы по данному соединению исполняются в пределах этого потока, который, в свою очередь, выполняется одним ядром или процессором. Сервер кеширует потоки, так что их не нужно создавать или уничтожать для каждого нового соединения.

Когда клиенты (приложения) подключаются к серверу MySQL, сервер должен их идентифицировать. Идентификация основывается на имени пользователя, адресе хоста, с которого происходит соединение, и пароле. Также можно использовать сертификаты X.509 при соединении по протоколу Secure Sockets Layer (SSL). После того, как клиент подключился, для каждого запроса сервер проверяет наличие необходимых привилегий (например, разрешено ли клиенту использовать команду SELECT применительно к таблице Country базы данных World).

*Оптимизация и выполнение.* MySQL осуществляет синтаксический разбор запросов для создания внутренней структуры (дерева разбора), а затем выполняет ряд оптимизаций. В их число входят переписывание запроса, определение порядка чтения таблиц, выбор используемых индексов и т. п. Можно повлиять на работу оптимизатора, включив в запрос специальные ключевые слова-подсказки (hints). Также существует возможность попросить сервер объяснить различные аспекты оптимизации. Это позволит вам понять, какие решения принимает сервер, и даст отправную точку для изменения запросов, схем и настроек с целью достижения максимальной эффективности работы.

Оптимизатор не интересуется тем, в какой подсистеме хранения данных находится каждая таблица, но подсистема хранения данных влияет на то, как сервер оптимизирует запрос. Оптимизатор запрашивает подсистему хранения данных

о некоторых ее возможностях и стоимости определенных операций, а также о статистике по содержащимся в таблицах данным. Например, некоторые подсистемы хранения поддерживают типы индексов, которые могут быть полезными для выполнения определенных запросов.

Прежде чем выполнять синтаксический анализ запроса, сервер обращается к кешу запросов, в котором могут храниться только команды SELECT и соответствующие им результирующие наборы. Если поступает запрос, идентичный уже имеющемуся в кеше, серверу вообще не нужно выполнять анализ, оптимизацию или выполнение запроса – он может просто отправить в ответ на запрос сохраненный результирующий набор.

## 1.2. Транзакции

Весь учебный материал, рассмотренный выше, представляет независимое использование баз данных различными пользователями SQL. В определенных ситуациях это можно считать стандартной процедурой, например, когда идет работа по составлению отчетов или работа по определенным сценариям баз данных. Но возникают ситуации, когда для выполнения определенных действий необходимо параллельное участие нескольких пользователей баз данных, которые должны выполнять одновременно определенные действия. Поэтому необходимо рассмотреть вопрос обеспечения информационного взаимодействия, а также необходимой инфраструктуры для работы нескольких пользователей одновременно с одной базой данных.

*Многопользовательские базы данных*. СУБД позволяют не только одному пользователю запрашивать и изменять данные,

но и выполнять аналогичные действия в многопользовательском режиме. При этом СУБД контролирует соблюдение требований непротиворечивости информации, вносимой в базу данных. В случае, когда в многопользовательском режиме каждый клиент выполняет только запросы к базе данных, система управления базами данных работает без ошибок и с высоким быстродействием. Но бывают ситуации, когда множество пользователей выполняют операции внесения изменений в хранимую информацию, в таком случае на СУБД нагрузка значительно увеличивается.

Предположим, что оператору в городской библиотеке необходимо подвести итоги работы всех городских отделений библиотеки за год. При этом в момент формирования годового отчета происходят параллельно несколько различных операций, например:

1) в одном из отделений читатель получает книги;
2) в другом отделении читатель компенсирует утрату книги;
3) в главном отделении библиотеки проходит инвентаризация и добавляются новые книги в базу.

Возникает вопрос, какие именно чи́сла должны отображаться в отчете в момент его формирования и будут ли эти числа отображать перечисленные выше операции.

Рассмотрим другую ситуацию: банковское обслуживание является классическим примером транзакций, который демонстрирует их необходимость. Представим базу данных банковских счетов, которая содержит два счета: текущий (current) и сберегательный (save) счета. Чтобы перевести 500 руб. с одного счета клиента на другой счет, нужно сделать, как минимум, три шага, а именно:

1) оценить, действительно ли на одном счете остаток денежных средств содержит запрашиваемую сумму;

2) списать с одного счета указанную сумму;

3) зачислить на другой счет 500 руб.

Таким образом, для безошибочного выполнения вышеописанных операций в процессе выполнения все ячейки в используемых таблицах должны быть заблокированы.

*Блокировка ячеек. Блокировка* – это механизм, применяемый СУБД для управления одновременно используемыми ресурсами. Когда используемая часть базы данных заблокирована, то остальные пользователи, обращающиеся к этим данным (чтение или запись), должны ожидать, пока блокировка не будет снята. Большинство СУБД, существующих на сегодняшний день, используют одну из следующих стратегий:

1. При выполнении операций записи, СУБД блокирует доступ на выполнение всех операций с данной таблицей. Таким образом, если в этот момент другой пользователь попытается прочитать информацию из данной таблицы, то его запрос будет заблокирован. Блокировка продлится до тех пор, пока операция записи не будет завершена.

2. При выполнении операции записи СУБД будет блокировать только операции записи в данной таблице, а операции чтения будут доступны. Причем при выполнении операции чтения пользователь будет видеть данные, которые были сохранены в таблице до начала внесения в нее изменений. После окончания операции записи просматриваемые данные будут обновлены.

У описанных выше подходов есть свои плюсы и минусы. Так, при использовании первого подхода при наличии большого количества одновременных запросов на выполнение операций чтения и записи, время ожидания доступности данных может быть достаточно велико. При использовании второго подхода могут возникать проблемы при длительных запросах

на изменение данных. Отметим, что на практике оба эти подхода реализуются одинаково часто в зависимости от используемой СУБД и задач, которые она решает. Например, Microsoft SQL Server реализует первую стратегию, Oracle Database использует вторую, а MySQL – обе стратегии.

Со своей стороны СУБД MySQL предлагает такой выбор. Подсистемы хранения данных могут реализовывать собственные политики блокировки и уровни детализации блокировок. Управление блокировками является важным решением при проектировании подсистем хранения данных. Фиксация детализации на определенном уровне может дать в ряде случаев лучшую производительность, но сделать эту подсистему менее подходящей для других целей. В связи с тем, что MySQL поддерживает несколько различных подсистем хранения баз данных, нет необходимости принимать единственное решение для всех подсистем.

Основные уровни блокировки:

1. Табличный уровень – запрет на выполнение одновременных действий несколькими пользователями одновременно в одной таблице. Когда клиент хочет выполнить запись в таблицу (вставку, удаление, обновление и др.), он захватывает блокировку на запись для всей таблицы. Такая блокировка предотвращает все остальные операции чтения и записи. В тот момент, когда никто не производит запись, любой клиент может получить блокировку на чтение и она не будет конфликтовать с другими аналогичными блокировками. Блокировки таблиц имеют различные модификации для обеспечения высокой производительности в разных ситуациях. Например, блокировки таблицы READ LOCAL разрешают некоторые типы одновременных операций записи. Также блокировки записи имеют более высокий приоритет, чем блокировки чтения, поэтому

запрос блокировки записи будет помещен в очередь перед уже имеющимися запросами блокировки чтения (блокировки записи могут отодвигать в очереди блокировки чтения, но блокировки чтения не могут отодвигать блокировки записи). Хотя подсистемы хранения данных могут управлять своими собственными блокировками, сама СУБД MySQL также использует для различных целей разные блокировки, действующие на уровне таблицы. Например, для таких команд, как ALTER TABLE, сервер применяет табличную блокировку вне зависимости от подсистемы хранения данных.

2. Страничный уровень – запрет нескольким пользователям одновременно изменять данные на одной и той же странице идентичной таблицы.

3. Строчная блокировка – запрет на одновременную работу нескольких пользователей в одной строке одновременно. Блокировка на уровне строк доступна среди прочих в подсистемах хранения данных InnoDB и Falcon. Блокировки строк реализуются подсистемами хранения данных, а не сервером. Сервер ничего не знает о блокировках, реализованных подсистемой хранения данных, и все подсистемы хранения данных реализуют блокировки по-своему.

Если бы все СУБД работали безотказно при любых условиях, пользователи выполняли все алгоритмы и требования без ошибок, а завершение работы приложений всегда происходило бы корректно, в таком случае необходимости в реализации процедуры транзакций на практике бы не было. К сожалению, ни одна из существующих систем не имеет абсолютной защиты от вышеназванных ошибок. В таком случае с целью минимизации возможных отказов в обслуживании необходима процедура, контролирующая выполнение всех операций. В системах управления базами данных такую функцию выполняют транзакции.

*Транзакция* – неделимая последовательность действий, состоящая из SQL запросов, рассматриваемая как единое целое. Таким образом, если система управления базами данных может выполнить всю последовательность действий, тогда они выполняются без ошибок. В случае, если в процессе выполнения один из запросов не может быть выполнен, тогда вся последовательность действий будет невыполненной. Следовательно, транзакция может существовать только в двух состояниях: выполнено и не выполнено. Промежуточных состояний нет.

На языке SQL, транзакция начинается с команды START TRANSIACTION, далее оператор либо фиксирует изменения, вносимые транзакцией, вводом команды COMMIT, либо отменяет их вводом команды ROLLBACK.

Если вернуться к примеру с банковским переводом денег со счета current на счет save, то команда на выполнение транзакции будет иметь следующий вид:

```
START TRANSACTION;
SELECT ball FROM current WHERE account_id=10113229;
UPDATE current SET ball=ball – 500
WHERE account_id=10113229;
UPDATE save SET ball=ball + 500
WHERE account_id=10113229;
COMMIT;
```

Стоит отметить, что выполнение транзакций абсолютно не защищает от сбоя во время выполнения. Например, что произойдет в случае сбоя сервера базы данных во время выполнения четвертой строки SQL запроса? Клиент, вероятно, просто потеряет 500 руб. А если другой процесс снимет весь остаток с текущего счета в момент между выполнением строк 3 и 4? Банк предоставит клиенту кредит размером 500 руб., даже не зная об этом.

В связи с этим для предотвращения возникновения таких ошибок, система управления базами данных должна соответствовать требованиям ACID, которые включают в себя требования по атомарности (atomicity), непротиворечивости (consistency), изолированности (isolation) и долговечности (durability). Рассмотрим более подробно каждый из этих критериев.

*Атомарность*. Транзакция должна функционировать как единая неделимая единица работы таким образом, чтобы вся транзакция была либо выполнена, либо отменена. Когда транзакции являются атомарными, не существует такого понятия, как частично выполненная транзакция: все или ничего.

*Непротиворечивость.* База данных должна всегда переходить из одного непротиворечивого состояния в последующее. В нашем примере непротиворечивость гарантирует, что сбой между третьей и четвертой строками не приведет к исчезновению 500 руб. с одного из счетов. Поскольку транзакция не будет зафиксирована, ни одно из изменений в этой транзакции не будет отражено в базе данных.

*Изолированность.* Результаты транзакции обычно невидимы другим транзакциям, пока она не закончена. Это гарантирует, что, если в нашем примере программа суммирования остатков на банковских счетах будет запущена после третьей строки, но перед четвертой, она по-прежнему увидит 500 руб. на текущем счете.

*Долговечность.* Будучи зафиксированы, внесенные в ходе транзакции изменения становятся постоянными. Это означает, что изменения должны быть записаны так, чтобы данные не могли быть потеряны в случае сбоя системы. Долговечность, однако, является несколько расплывчатой концепцией, поскольку на самом деле существует много уровней. Некоторые стратегии обеспечения долговечности предоставляют более

сильные гарантии безопасности, чем другие, и ни одна из них не является надежной на 100 %.

Таким образом, с целью защиты от различных ошибок, возникающих при выполнении различных операций с базами данных, СУБД первоначально запускает транзакцию, далее выдает SQL инструкции на выполнение, и в случае успешного завершения, выдает команду COMMIT. Например, SQL инструкции на перевод денежных средств с одного счета на другой примут следующий вид:

```
START TRANSACTION;
UPDATE current SET ball=ball–500
WHERE account_id=10113229 AND ball>500;
IF <exactly one row was updated by the previous statement> THEN
UPDATE save SET ball=ball+500 WHERE account_id=10113229;
IF <exactly one row was updated by the statement> THEN
COMMIT;
ELSEROLLBACK;
END IF;
ELSE ROLLBACK;
END IF;
```

В данном блоке SQL-кода первоначально запускается транзакция, далее происходит попытка списания денежных средств с одного счета при наличии на нем условленной суммы и зачисление средств на другой счет. В данном случае четко видно, что при выполнении всех условий транзакция фиксируется, если же какое-либо из условий не выполняется или выполняется с ошибкой, происходит откат к первоначальным значениям.

Используя транзакцию, СУБД гарантирует, что денежные средства или останутся на первоначальном счете, или перейдут на другой счет. Вне зависимости от того, была ли выполнена транзакция удачно или завершилась ошибкой, во время ее выполнения все используемые ресурсы заблокированы, а по ее

окончании блокировка снимается. Стоит отметить, что если транзакция полностью не завершается, а во время ее выполнения сервер выключается, то при его включении сразу произойдет откат к первоначальным значениям.

Транзакции ACID гарантируют, что банк не потеряет ваши деньги. Вообще очень трудно или даже невозможно сделать это с помощью логики приложения. Чтобы обеспечить гарантии ACID, ACID – совместимый сервер баз данных должен выполнить множество сложных действий, о которых вы, возможно, даже не догадываетесь.

Как и в случае повышения уровня детализации блокировок, оборотной стороной усиленной безопасности является то, что сервер базы данных должен выполнять больше работы. Сервер базы данных с транзакциями ACID обычно требует большей мощности процессора, объема памяти и дискового пространства, чем без них. Как мы уже упоминали, это тот самый случай, когда архитектура подсистем хранения данных MySQL оказывается благом. Вы можете решить, требует ли ваше приложение использования транзакций. Если они вам на самом деле не нужны, можно добиться большей производительности, выбрав для некоторых типов запросов нетранзакционную подсистему хранения данных. Чтобы установить нужный уровень защиты без использования транзакций, применяется команда LOCK TABLES.

*Уровни изоляции. Изолированность* – более сложное понятие, чем кажется на первый взгляд. Стандарт SQL определяет четыре уровня изоляции с конкретными правилами, устанавливающими, какие изменения видны внутри и вне транзакции, а какие нет. Более низкие уровни изоляции обычно допускают большую степень совместного доступа и вызывают меньше накладных расходов.

Рассмотрим четыре уровня изоляции:

1. *READ UNCOMMITTED.* На этом уровне изоляции транзакции могут видеть результаты незафиксированных транзакций. На этом уровне вы можете столкнуться со множеством проблем, если не знаете абсолютно точно, что делаете. Используйте этот уровень, если у вас есть на то веские причины. На практике READ UNCOMMITTED применяется редко, поскольку его производительность ненамного выше, чем у других уровней, имеющих множество преимуществ. Чтение незафиксированных данных еще называют грязным чтением (dirtyread).

2. *READ COMMITTED.* Уровнем изоляции по умолчанию для большинства СУБД (но не для MySQL) является READ COMMITTED. Он удовлетворяет вышеприведенному простому определению изолированности: транзакция увидит только те изменения, которые были уже зафиксированы другими транзакциями к моменту ее начала, а произведенные ею изменения останутся невидимыми для других транзакций, пока текущая транзакция не будет зафиксирована. На этом уровне возможен феномен невоспроизводимого чтения (non repeatable read). Это означает, что вы можете выполнить одну и ту же команду дважды и получить различный результат.

3. *REPEATABLE READ.* Этот уровень изоляции решает проблемы, которые возникают на уровне READ UNCOMMITTED. Он гарантирует, что любые строки, которые считываются в контексте транзакции, будут выглядеть также при последовательных операциях чтения в пределах одной и той же транзакции, однако теоретически на этом уровне возможен феномен фантомного чтения (phantom reads). Проще говоря, фантомное чтение может происходить в случае, если вы выбираете некоторый диапазон строк, затем другая транзакция вставляет но-

вую строку в этот диапазон, после чего вы выбираете тот же диапазон снова. В результате вы увидите новую «фантомную» строку. В InnoDB и Falcon проблема фантомного чтения решается с помощью MVCC (multi version concurrency control).

REPEATABLE READ является в MySQL уровнем изоляции транзакций по умолчанию. Некоторые другие подсистемы хранения данных поступают так же, но выбор остается за конкретной подсистемой.

4. *SERIALIZABLE*. Самый высокий уровень изоляции. Решает проблему фантомного чтения, заставляя транзакции выполняться в таком порядке, чтобы исключить возможность конфликта. В двух словах, уровень SERIALIZABLE блокирует каждую строку, которую транзакция читает. На этом уровне может возникать множество задержек и конфликтов при блокировках. На практике данный уровень изоляции применяется достаточно редко, но потребности вашего приложения могут заставить вас использовать его, согласившись с меньшей степенью совместного доступа в пользу стабильности данных.

*Взаимоблокировки.* Возникновение взаимоблокировок возможно, когда несколько транзакций запрашивают блокировку одних и тех же ресурсов. Кроме того, возникновение взаимоблокировок возможно, если происходят запросы на блокировку ресурсов от транзакции в различном порядке.

Взаимоблокировки могут происходить, когда несколько транзакций блокируют одни и те же ресурсы. Рассмотрим в качестве примера следующие две транзакции, обращающиеся к таблице money:

1. Транзакция #1 START TRANSACTION.

```
UPDATE money_price SET ball =ball – 45.50
WHERE user_id = 213 and date = ‘2020–10–01’;
UPDATE money_price SET ball = ball + 19.80
WHERE user_id = 218 and date = ‘1990–11–19’;
COMMIT;
```

2. Транзакция #2 START TRANSACTION.

```
UPDATE money_price SET deb = deb + 20
WHERE user_id = 218 and date = ‘1990–19–11’;
UPDATE money_price SET deb = deb + 47.20
WHERE user_id = 213 and date =‘2020–10-01’;
COMMIT;
```

Может возникнуть ситуация, когда каждая транзакция выполнит первый запрос, а потом обновит строку данных, при этом заблокировав ее в процессе обновления. Далее обе транзакции попытаются обновить вторую строку, но обнаружат, что она уже заблокирована. В результате одна транзакция будет до бесконечности ожидать окончания другой, и конфликт не разрешится до тех пор, пока не произойдет какое-то событие, которое снимет взаимную блокировку.

Для разрешения этой проблемы в системах баз данных реализованы различные формы обнаружения взаимоблокировок и тайм-аутов. Такие развитые подсистемы хранения данных, как InnoDB, обнаруживают циклические зависимости и немедленно возвращают ошибку. На самом деле, это хорошо, иначе взаимоблокировки проявлялись бы как очень медленные запросы. Другие системы в подобных ситуациях откатывают транзакцию по истечении тайм-аута, что не очень хорошо. Способом, которым InnoDB обрабатывает взаимоблокировки, является откат той транзакции, которая захватила меньше всего монопольных блокировок строк (приблизительный показатель легкости отката).

Поведение и порядок блокировок зависят от конкретной подсистемы хранения данных, так что в некоторых подсистемах при определенной последовательности команд могут происходить взаимоблокировки, хотя в других подсистемах при таких же условиях они не возникают.

Взаимоблокировки имеют двойственную природу: некоторые действительно неизбежны из-за характера обрабатываемых данных, другие же вызваны тем, как работает конкретная подсистема хранения.

Взаимоблокировку нельзя разрешить без отката одной из транзакций, частичного либо полного. Существование взаимоблокировок в транзакционных системах – непреложный факт, с учетом которого ваше приложение и нужно проектировать. При возникновении такой ситуации многие приложения могут просто попытаться выполнить транзакцию с самого начала.

*Ведение журналов транзакций.* Ведение журналов транзакций обеспечивает снижение вероятности сбоя при выполнении транзакции, а также делает ее более быстрой. Первоначально, вместо обновления таблиц с данными, которые располагаются непосредственно на диске в базе данных, происходит внесение изменений в находящуюся в памяти копию этих данных. Такая операция происходит достаточно быстро, ввиду отсутствия необходимости чтения/записи информации с диска. Далее СУБД вносит запись о выполнении транзакции в соответствующий журнал, который в данном случае хранится на диске и является энергонезависимым. Данная операция также является достаточно быстрой в связи с тем, что она сводится к операции ввода-вывода в пределах ограниченной области на диске, а не в различных его случайных местах. В итоге многие СУБД дважды производят сохранение информации об произведенных изменениях на диске. В случае, если в процессе

выполнения происходит сбой, например, когда обновлены сами данные, но нет записи в журнале, подсистема хранения самостоятельно выполнит восстановление изменений после запуска.

## 1.3. Подсистемы хранения в MySQL

В данном разделе представлен обзор различных подсистем хранения, используемых в MySQL. Настоящее пособие не содержит полного описания всех существующих подсистем хранения в связи с большим разнообразием таковых, поэтому будет проведен обзор самых распространенных, применимых в различных ситуациях подсистем. Таким образом, с целью принятия окончательного решения о применении той или иной подсистемы хранения данных зачастую достаточно изучить рассматриваемые в данном пособии подсистемы.

MySQL сохраняет все схемы (именуемые также базами данных) в виде подкаталога своего каталога в файловой системе. В момент создания новой таблицы в MySQL данная система сохраняет определение таблицы в файле с идентичным именем и в формате frm. Таким образом, например, для таблицы TableName файл с именем TableName.frm сохраняется в каталоге файловой системы. Стоит отметить, что в связи с использованием данной системой управления базами данных различных файловых систем (ФС), чувствительность к регистру зависит от конкретной используемой ФС. Так, при эксплуатации СУБД на платформе Windows имена таблиц и данных не чувствительны к регистру, а в системах, подобных Unix, – чувствительны.

С целью определения, какая именно подсистема хранения используется в конкретной таблице, необходимо воспользо-

ваться командой SHOW TABLE STATUS. Для того, чтобы получить информацию, например о таблице client, необходимо выполнить команду: SHOW TABLE STATUS LIKE ‘client’\G. В итоге СУБД выдаст следующую информацию:

```
Name: client
Engine: MyISAM
Row_format: Dynamic
Rows: 8
Avg_row_lenght: 74
Data_lenght: 789
Max_data_lenght: 4294967295
Index_lenght: 2048
Data_free: 0
Auto_increment: NULL
Create_time: 2019_01_07 14:32:44
Update_time: 2019_01_09 10:44:32
Check_time: NULL
Collation: utf8_bin
Checksum: NULL
Create options:
Comment: user and global privileges 1 row in set (0.00 sec)
```

Как можно увидеть из представленного выше ответа СУБД, в данном случае представлена таблица типа MyISAM. Кроме того, ответ содержит еще некоторую служебную информацию. Рассмотрим каждое из полей более подробно:

1. *Name*. Содержит имя таблицы, к которой происходит обращение.

2. *Engine*. Выводит название подсистемы хранения баз. Стоит отметить, что в более ранних версиях данный столбец назывался Type.

3. *Row_format*. Содержит информацию о формате строки. Для таблицы с подсистемой хранения MyISAM данное поле может иметь значение о динамической (dynamic) длине строки, фиксированной (fixed) или сжатой (compressed). В первом слу-

чае длина строки может меняться в связи с тем, что она содержит информацию переменной длины, например это могут быть данные типа VARCHAR или BLOB. Строки с фиксированной длиной содержат поля, длина которых не меняется, например поля типа CHAR и INTEGER. Сжатый формат строк может существовать только в сжатых таблицах.

4. *Rows*. Содержит информацию о количестве строк в указанной таблице. Стоит учитывать, что для транзакционных таблиц данная величина приблизительная, а вот для нетранзакционных – точная.

5. *Avg_row_lenght.* Информация о среднем размере информации строк в таблице.

6. *Data_lenght.* Общий объем информации, содержащийся в таблице, выраженный в байтах.

7. *Max_data_lenght.* Строка содержит информацию о максимальном объеме данных, которые могут быть в данной таблице.

8. *Index_length*. Объем дискового пространства занимают данные индексов.

9. *Data_free*. В таблицах типа MylSAM показывает объем выделенного пространства, которое на данный момент не используется. В нем хранятся ранее удаленные строки. Оно может быть использовано в будущем при выполнении команд INSERT.

10. *Create_time.* Время создания таблицы.

11. *Update_time.* Время последнего изменения таблицы.

12. *Check_time*. Время последней проверки таблицы командой CHECK TABLE.

13. *Collation.* Содержит информацию о кодировке и схеме упорядочивания столбцов по умолчанию.

14. *Checksum.* В случае, если включена система подсчета контрольной суммы для данной таблицы, то в этой строке будет выведено ее текущее значение.

15. *Create options.* Иные параметры, указанные при создании таблицы.

16. *Comment.* Данное поле содержит различную дополнительную информацию. Для таблиц типа MyISAM в нем хранятся комментарии, добавленные при создании таблицы. Если таблица использует подсистему хранения InnoDB, то приводится объем свободного места в табличном пространстве InnoDB.

*Подсистема хранения MyISAM.* Данная подсистема хранения является подсистемой по умолчанию в СУБД MySQL, а также представляет собой согласование между достаточно высокой производительностью и широким набором различных функций, таких как сжатие, полнотекстное индексирование и некоторые другие пространственные функции, используемые в геоинформационных системах. При этом стоит отметить, что данная подсистема не поддерживает транзакции и блокировки на уровне строк.

Подсистема MyISAM по умолчанию хранит каждую таблицу в индексном файле (расширение .MYI) и файле данных (расширение .MYD). Кроме того, формат MyISAM не зависит от платформы, т. е. можно копировать вышеназванные файлы с одной платформы на другую без каких-либо проблем.

Таблицы MyISAM со строками переменной длины, создаваемые в версии MySQL 5.0, настроены по умолчанию на поддержку 256 Тбайт данных с использованием шестибайтных указателей на записи с данными.

В более ранних версиях MySQL указатели по умолчанию были четырехбайтными с максимальным объемом данных

4 Гбайта. Все версии MySQL могут поддерживать размер указателя до 8 байт. Чтобы изменить размер указателя в таблице MyISAM (уменьшить или увеличить), вы должны задать значения для параметров MAX_ROWS и AVG_ROW_LENGTH, которые представляют приблизительную оценку необходимого пространства:

```
CREATE TABLE mytable (
a INTEGER NOT NULL PRIMARY KEY,
b CHAR{18) NOT NULL
) MAX_R0WS = 1000000000 AVG_ROW_LENGTH = 32;
```

В данном запросе MySQL серверу сообщается, что он должен хранить в таблице по крайней мере 32 Гбайта данных. Чтобы выяснить, какое решение принял MySQL, необходимо запросить статус таблицы командой:

mysql> SHOW TABLE STATUS LIKE ‘mytable’ \G

```
Name: mytable
Engine: MylSAM
Roveformat: Fixed
Rows: 0
Avg_row_lengthi 0
Datajength: 0
MaxjdataJength: 98784247807
Index_length: 1024
Datajree: 0
Auto^increment: NULL
Create_time: 2002–02–24 17:36:57
Update_time: 2002–02–24 17:36:57
Check_time: NULL
Create_options: max_rows=1000000000 avg_row_length=32
Comment:
1 rowinset (0.05 sec)
```

В результате будет получен следующий ответ сервера на запрос:

```
Name: mytable
Engine: MylSAM
Roveformat: Fixed
Rows: 0
Avg_row_lengthi 0
Datajength: 0
MaxjdataJength: 98784247807
Index_length: 1024
Datajree: 0
Auto^increment: NULL
Create_time: 2002–02–24 17:36:57
Update_time: 2002–02–24 17:36:57
Check_time: NULL
Create_options: max_rows=1000000000 avg_row_length=32
Comment:
1 rowinset (0.05 sec)
```

Можно заметить, что MySQL прописал в памяти параметры создания, указанные ранее. Кроме того, отображается возможность хранения 91 ГБ данных. В дальнейшем, с помощью команды ALTER TABLE возможно изменить заданный размер указателя, однако, это может привести к тому, что вся таблица индексов и данных будет переписываться снова, а при этом данная процедура будет занимать достаточно много времени.

*Особенности MylSAM.* Как одна из самых старых подсистем хранения, включенных в MySQL, MyISAM обладает многими функциями, которые были разработаны за годы использования СУБД для решения различных задач:

1. *Блокировка и конкуренция.* MyISAM блокирует целиком таблицы, но не строки. Запросы на чтение получают разделяемые (на чтение) блокировки всех таблиц, к которым они обращаются. Запросы на запись получают монопольные (на запись) блокировки. Однако вы можете вставлять новые строки в таблицу в момент, когда исполняются запросы на выборку

данных из таблицы (конкурентные вставки). Это очень важная и полезная возможность.

2. *Автоматическое исправление*. MySQL поддерживает автоматическую проверку и исправление таблиц типа MyISAM.

3. *Ручное исправление*. Вы можете использовать команды CHECK TABLE mytable и REPAIR TABLE mytable для проверки таблицы на предмет наличия ошибок и их устранения. Вы также можете использовать командную утилиту myisamchk для проверки и исправления таблиц, когда сервер находится в автономном (offline) режиме.

4. *Особенности индексирования*. Вы можете создавать индексы по первым 500 символам столбцов типа BLOB и TEXT в таблицах MylSAM. MylSAM поддерживает полнотекстовые индексы, которые индексируют отдельные слова для сложных операций поиска.

5. *Отложенная запись ключей*. Таблицы MyISAM, помеченные при создании параметром DELAY_KEY_WRITE, не записывают измененные индексные данные на диск в конце запроса. Вместо этого MyISAM сохраняет изменения в буфере памяти. Сброс индексных блоков на диск происходит при заполнении буфера или закрытии таблицы. Это позволяет увеличить производительность работы с часто изменяемыми таблицами. Однако в случае сбоя сервера или системы индексы наверняка будут повреждены и потребуют восстановления. Это можно сделать с использованием скрипта, запускающего утилиту myisamchk перед перезапуском сервера, или при помощи параметров автоматического восстановления (даже если вы не используете параметр DELAY_KEY_WRITE, эти меры предосторожности не помешают). Вы можете сконфигурировать отложенную запись ключей как глобально, так и для отдельных таблиц.

6. *Сжатые таблицы MyISAM.* Некоторые таблицы, например, в приложениях на компакт-дисках или в отдельных встроенных (embedded) средах после их создания и заполнения данными никогда больше не изменяются. Такие таблицы можно сжать средствами MyISAM.

Для сжатия таблиц существует специальная утилита myisampack. Модифицировать сжатые таблицы невозможно (хотя при необходимости их следует распаковать, внести изменения и снова сжать), но при этом они занимают гораздо меньше места на диске. В результате их использования увеличивается производительность, поскольку из-за меньшего размера требуется меньше операций ввода-вывода для поиска на диске необходимых записей. Над сжатыми таблицами MyISAM можно строить индексы, но они также доступны только для чтения.

На современном оборудовании накладные расходы по распаковке данных для чтения в большинстве приложений незначительны. Эти накладные расходы перекрываются значительным выигрышем за счет уменьшения количества операций ввода/вывода. Строки сжимаются по отдельности, так что MySQL не нужно распаковывать всю таблицу (или даже страницу) с целью извлечения единственной строки.

*Подсистема InnoDB.* Подсистема хранения InnoDB была разработана для транзакционной обработки, в частности, для обработки большого количества краткосрочных транзакций, которые чаще благополучно завершаются, чем откатываются. Она остается наиболее популярной транзакционной подсистемой хранения. Высокая производительность и автоматическое восстановление после сбоя делают ее популярной и для нетранзакционных применений.

InnoDB сохраняет данные в наборе из одного или нескольких файлов данных. Этот набор файлов именуется табличным пространством (tablespace). Табличное пространство, в сущности, является черным ящиком, которым полностью управляет сама InnoDB. В MySQL 4.1 и более поздних версиях СУБД InnoDB может хранить данные и индексы каждой таблицы в отдельных файлах. Кроме того, данная подсистема может располагать табличные пространства на «сырых» (неформатированных) разделах диска.

Для обеспечения высокой степени конкурентности InnoDB использует MVCC и реализует все четыре стандартных уровня изоляции SQL. Для этой подсистемы уровнем изоляции по умолчанию является REPEATABLE READ, а стратегия блокировки следующего ключа позволяет предотвратить фантомные чтения: вместо того, чтобы блокировать только строки, затронутые в запросе, InnoDB блокирует также интервалы в индексе, предотвращая вставку фантомных строк.

Таблицы InnoDB строятся на кластерных индексах. Структуры индексов в InnoDB очень сильно отличаются от других подсистем хранения. Результатом является более быстрый поиск по первичному ключу. Однако вторичные индексы (отличные от индекса по первичному ключу) содержат все столбцы, составляющие первичный ключ, так что если первичный ключ длинный, то все прочие индексы будут большими. Если над таблицей построено много индексов, то первичный ключ нужно делать как можно меньшим. InnoDB не сжимает индексы.

InnoDB не может строить индексы путем сортировки в отличие от MyISAM. В результате InnoDB загружает данные и создает индексы медленнее, чем MyISAM. Любая операция, которая изменяет структуру таблицы InnoDB, приводит к полной перестройке таблицы, включая все индексы.

Подсистему InnoDB создавали в те времена, когда у большинства серверов были медленные диски, один процессор и ограниченный объем памяти. Сегодня, когда многоядерные серверы с огромным объемом памяти и быстрыми дисками становятся менее дорогими, InnoDB испытывает некоторые проблемы с масштабируемостью.

Помимо обеспечения высокой конкуренции следующей по популярности особенностью InnoDB являются ограничения целостности типа «внешний ключ», которые в самом сервере MySQL еще не реализованы. InnoDB также обеспечивает очень быстрый поиск по первичному ключу. В подсистеме InnoDB поддерживаются разнообразные внутренние оптимизации. В их число входят упреждающее интеллектуальное считывание данных с диска, адаптивный хеш-индекс (автоматическое построение хеш-индексов в памяти для обеспечения очень быстрого поиска) и буфер вставок для ускорения операций вставки.

Подсистема InnoDB очень сложна, и если вы используете InnoDB, то первоначально рекомендуется ознакомиться с разделом «InnoDB Transaction Model and Locking» (транзакционная модель и блокировки в InnoDB) в руководстве по MySQL. Существует много сюрпризов и исключений, о которых необходимо знать перед началом создания приложений с использованием этой подсистемы.

*Подсистема Memory.* Таблицы типа Memory (раньше называвшиеся таблицами типа HEAP) полезны, когда необходимо осуществить быстрый доступ к данным, которые либо никогда не изменяются, либо нет надобности в их сохранении после перезапуска. Обычно таблицы типа Memory обрабатываются примерно на порядок быстрее, чем таблицы MyISAM. Все их данные хранятся в памяти, поэтому запросам не нужно ждать выполнения операций дискового ввода/вывода. Струк-

тура таблицы Memory сохраняется после перезапуска сервера, но данные теряются.

Вот несколько хороших применений для таблиц Memory:

1. Для «справочных» таблиц или таблиц «соответствия», например для таблицы, в которой почтовым кодам соответствуют названия регионов.

2. Для кеширования результатов периодического агрегирования данных.

3. Для промежуточных результатов при анализе данных.

Таблицы Memory поддерживают индексы типа HASH, обеспечивающие большую скорость выполнения поисковых запросов.

Таблицы Memory работают очень быстро, но не всегда годятся в качестве замены дисковых таблиц. Они используют блокировку на уровне таблицы, что уменьшает конкуренцию при записи, и не поддерживают столбцы типа TEXT и BLOB. Также они допускают использование только строк фиксированного размера, поэтому значения типа VARCHAR сохраняются как значения типа CHAR, что повышает расход памяти.

MySQL внутри себя использует подсистему Memory для хранения промежуточных результатов при обработке запросов, которым требуется временная таблица. Если промежуточный результат становится слишком большим для таблицы Memory или содержит столбцы типа TEXT или BLOB, то MySQL преобразует его в таблицу MyISAM на диске.

Многие часто путают таблицы Memory с временными таблицами, которые создаются командой CREATE TEMPORARY TABLE.

Временные таблицы могут использовать любую подсистему хранения. Это не то же самое, что таблицы типа Memory.

Временные таблицы видны только в одном соединении и полностью исчезают при его закрытии.

*Подсистема Archive.* Подсистема хранения Archive позволяет выполнять только команды INSERT и SELECT и до версии MySQL 5.1 не поддерживала индексирование. Она требует значительно меньше операций дискового ввода/вывода, чем MyISAM, поскольку буферизует записываемые данные и сжимает все вставляемые строки с помощью библиотеки zlib. Кроме того, каждый запрос SELECT требует полного сканирования таблицы. По этим причинам таблицы Archive идеальны для протоколирования и сбора данных, когда анализ чаще всего сводится к сканированию всей таблицы, а также в тех случаях, когда требуется обеспечить быстроту выполнения запросов INSERT на главном сервере репликации. Подчиненные серверы репликации могут использовать для той же таблицы другие подсистемы хранения, следовательно, существует возможность проиндексировать таблицу на подчиненном сервере для большей производительности при анализе.

Подсистема Archive поддерживает блокировку на уровне строк и специальный системный буфер для вставок с высокой степенью конкурентности. Она обеспечивает согласованные чтения, останавливая выполнение команды SELECT после того, как извлечено столько строк, сколько было в таблице на момент начала выполнения запроса. Она также обеспечивает невидимость результатов пакетной вставки, пока процедура не будет завершена. Эти особенности эмулируют некоторые аспекты поведения транзакций и MVCC, но Archive не является транзакционной подсистемой хранения. Она лишь оптимизирована для высокоскоростной вставки и хранения данных в сжатом виде.

*Выбор оптимальной подсистемы хранения данных*. При разработке приложения для MySQL вы должны решить, какую подсистему хранения использовать. Если не задаться этим вопросом на этапе проектирования, то, скорее всего, через какое-то время вы столкнетесь с определенными сложностями. Так, вы можете обнаружить, что используемая по умолчанию подсистема не обеспечивает нужных возможностей, например транзакций, или может оказаться, что для той совокупности запросов на чтение и запись, которую генерирует ваше приложение, необходимы более детальные блокировки, чем блокировки на уровне таблицы в MyISAM.

Поскольку допустимо выбирать способ хранения данных для каждой таблицы в отдельности, вы должны ясно понимать, как будет использоваться каждая таблица и какие данные в ней планируется хранить. Это также поможет получить хорошее представление о приложении в целом и о потенциале его роста. Вооружившись этой информацией, вы сможете осознанно выбирать подсистемы хранения данных.

*Критерии выбора*. Хотя на решение о выборе подсистемы хранения могут влиять многие факторы, обычно все сводится к нескольким базовым критериям.

Вот основные элементы, которые следует принимать во внимание:

1. *Транзакции.* Если вашему приложению требуются транзакции, то InnoDB является наиболее стабильной, хорошо интегрированной, проверенной подсистемой хранения. Однако ожидается, что со временем появятся транзакционные подсистемы, которые станут сильными соперниками для InnoDB.

MyISAM можно назвать хорошим вариантом, если задача не требует транзакций и в основном предъявляет запросы типа SELECT или INSERT.

Иногда отдельные компоненты приложения (например, протоколирование) попадают в эту категорию.

2. *Конкурентный доступ.* Как лучше удовлетворить ваши требования к конкуренции, зависит от рабочей нагрузки. Если вам требуется просто осуществлять в конкурентном режиме операции чтения и вставки, то MyISAM является прекрасным выбором. Если нужно поддержать совокупность одновременных операций так, чтобы они не искажали результатов друг друга, то хорошо подойдет какая-нибудь подсистема с возможностью блокировок на уровне строки. Необходимость регулярно проводить операции резервного копирования также может повлиять на ваш выбор. Если существует возможность периодически останавливать сервер для выполнения данной процедуры, то подойдет любая подсистема хранения данных. Однако, если требуется осуществлять резервное копирование без остановки сервера, выбор уже не так очевиден.

Необходимо помнить, что использование различных подсистем хранения увеличивает сложность резервного копирования и настройки сервера.

3. *Восстановление после сбоя.* Если объем данных велик, то нужно серьезно оценить, сколько времени займет восстановление базы после сбоя. Таблицы MyISAM обычно чаще оказываются поврежденными и требуют значительно больше времени для восстановления, чем, например, таблицы InnoDB. На практике это одна из самых важных причин, по которым многие используют подсистему InnoDB даже при отсутствии необходимости в транзакциях.

Наконец, вы можете обнаружить, что приложению требуются конкретные возможности или оптимизации, которые могут обеспечить только некоторые подсистемы хранения MySQL. Например, многие приложения используют оптимиза-

ции кластерных индексов. В настоящее время эту возможность обеспечивают только InnoDB и SolidDB. С другой стороны, лишь MyISAM поддерживает полнотекстовый поиск. Если подсистема хранения отвечает одному или нескольким критическим требованиям, но не отвечает другим, то нужно либо найти компромисс, либо выбрать разумное проектное решение.

# ГЛАВА 2. РЕЗЕРВНОЕ КОПИРОВАНИЕ И ВОССТАНОВЛЕНИЕ

Первоначально большинство системных администраторов сосредотачивают свои силы на обеспечении быстродействия системы, удобства пользования, а также на дружелюбном интерфейсе. Таким образом, получается условно безотказно работающая СУБД. Но все это существует до тех пор, пока не происходит первый серьезный сбой. Одним из основных механизмов восстановления системы после сбоев является периодическое резервное копирование. Резервное копирование способно восстановить системы после серьезных сбоев, а также имеет важное значение для обеспечения высокого быстродействия. В связи с тем, что любая СУБД полноценно работает и без запуска системы резервного копирования, поэтому про нее вспоминают лишь, когда происходит серьезный сбой. Следовательно, необходимо еще при проектировании базы данных учитывать и прорабатывать то, каким образом система резервного копирования будет функционировать, чтобы оказывать наименьшее влияние на производительность СУБД.

Если на этапе планирования резервное копирование не вносилось в общий проект, то потом его обычно приходится наспех запускать уже на работающей системе. И в этот момент может обнаружиться, что принятые ранее решения исключают самые высокопроизводительные методы подготовки резервных копий. Например, сервер баз данных системы уже сконфигурирован, далее появляется необходимость воспользоваться LVM для снятия мгновенных снимков файловой системы, но уже поздно. Кроме того, может выясниться, что настройка системы для резервного копирования оказывает серьезное влия-

ние на производительность. И если заранее не сформировать план восстановления и не испытать его на практике, то в момент аварии все пойдет совсем не так гладко, как хотелось бы.

## 2.1. Общие сведения

Системы резервного копирования похожи на системы мониторинга и оповещения: большинство системных администраторов периодически применяют все новые, ранее не используемые механизмы. Это нецелесообразно, потому что существуют хорошие, отлично поддерживаемые и гибкие программы для резервного копирования, причем некоторые из них бесплатные и поставляются с открытым исходным кодом.

Прежде всего уточним основные термины, поскольку нет их единого понимания. Рассмотрим понятия «горячее», «теплое» и «холодное» резервное копирование. Обычно эти слова описывают влияние резервного копирования на текущую работу. Так, «горячее» резервное копирование означает, что сервер не нужно останавливать.

Следующие важные понятия – restore (возвращать) и recover (восстанавливать). Под возвратом стоит понимать извлечение данных из резервной копии, либо загрузку их в MySQL, либо размещение файлов в локальном каталоге, к которому СУБД периодически будет обращаться самостоятельно. Под процессом восстановления обычно понимается весь процесс приведения системы или ее части в рабочее состояние после возникновения сбоев различного рода. В это понятие вкладывается смысл не только восстановления данных из резервной копии, но и все остальные шаги, необходимые для возобновления функционирования в полном объеме, в частности, перезапуск MySQL, изменение конфигурации, прогрев кешей сервера и т. д.

Существует общепринятое понимание, что процесс восстановления представляет собой, в основном, только восстановление непосредственно таблиц базы данных после сбоя. Но этот процесс имеет явные отличия от процесса восстановления всего сервера. Процедура восстановления подсистемы хранения должна привести в соответствие файлы данных и журналов. Она также должна гарантировать, что в файлах данных отражены только модификации, произведенные зафиксированными транзакциями, и накатить те транзакции из журналов, которые еще не были применены к файлам данных. При использовании транзакционной подсистемы хранения это может быть частью общего процесса восстановления или даже частью резервного копирования. Однако это не то восстановление, которое может потребоваться после случайного выполнения команды DROP TABLE.

Основные проблемы заключаются в том, что организовать гладкий процесс резервного копирования проще, чем создать хорошие процедуры восстановления. И этому есть несколько причин. Если нет резервной копии, то не с чего восстанавливаться, поэтому при построении системы внимание уделяют, прежде всего, резервному копированию. Более того, вообще не следует строить систему резервного копирования, пока не определены требования к восстановлению.

Резервное копирование – это регулярно выполняемая процедура. Поэтому первоначально при проектировании базы данных стоит учитывать процесс автоматизации процедуры резервного копирования. Если резервное копирование производится за пределы рабочей площадки, то, скорее всего, данные на копии шифруются или защищаются каким-либо другим способом.

Очень легко рассуждать о том, какой ущерб нанесет компрометация данных, и совсем упустить из виду, что произойдет, если никто не сможет прочитать зашифрованный том, чтобы восстановить с него данные, или позабыть о том, чего стоит извлечь один файл из монолитного зашифрованного файла-архива. Один человек может спланировать, спроектировать и реализовать процедуру резервного копирования, особенно, при наличии подходящих инструментов.

Рассмотрим пример: некий заказчик сообщил, что резервное копирование стало выполняться с молниеносной скоростью, стоило указать при запуске mysqldump флаг -d, и поинтересовался, почему никто не сообщил ему, что этот флаг может настолько ускорить процедуру. Если бы он попытался вернуть данные с таких копий, то очень быстро понял бы причину: при наличии флага -d копирование данных не происходит.

При проектировании системы резервного копирования и восстановления стоит принять во внимание:

1. Какой объем данных вы можете потерять без серьезных последствий. Понадобится ли восстановление на конкретный момент времени в прошлом или допустимо потерять всю работу с момента снятия последней резервной копии. Есть ли какие-то законодательные требования.

2. Скорость восстановления информации после сбоя. Допустимы ли простой или остановка сервера при выполнении резервного копирования.

3. Тип данных, которые необходимо восстанавливать, а также содержание: восстановление всей серверной части, базы данных, конкретные таблицы или журналы транзакций и выполняемых команд.

*Рекомендации по резервному копированию.* Прежде чем начинать подробное описание имеющихся вариантов, стоит

учитывать, что большинству пользователей или администраторов баз данных нужны решения по резервному копированию и восстановлению. Перечисленные далее шаги стоит выполнять при первоначальном проектировании баз данных.

1. Для больших баз данных физическое резервное копирование совершенно необходимо: этот механизм работает быстро, что крайне важно. Рекомендуется выполнять резервные копии на основе снимков, но технология InnoDB HotBackup – вполне приемлемая альтернатива, если работа проходит только с таблицами типа InnoDB.

2. Необходимо включать в резервную копию двоичные журналы, чтобы оставалась возможность восстановления на конкретный момент времени в прошлом.

3. Сохранение нескольких поколений резервных копий, а файлы двоичных журналов необходимо хранить достаточно длительное время, чтобы из них можно было вернуть данные.

4. Периодически необходимо проводить проверку процедуры резервного копирования и восстановления, проходя весь процесс восстановления от начала до конца.

5. Периодически необходимо создавать логические резервные копии (возможно, из физических копий для эффективности). Кроме того, необходимо контролировать, чтобы было достаточно двоичных журналов для восстановления из последней логической копии.

6. Если возможно, проводить периодические проверки физических резервных копий с целью проверки и уточнения, можно ли будет восстановиться с них. Старайтесь осуществлять такую проверку еще в процессе резервного копирования, до отправки в место хранения.

7. Необходимо проводить мониторинг процесса резервного копирования и самих резервных копий независимо от тех

инструментов, которые применяются для резервного копирования. Необходимо внешнее подтверждение их пригодности.

*Причины создания резервных копий.* Если вы создаете высокопроизводительную систему на основе MySQL, то без резервных копий не обойтись. Назовем основные причины этого:

1. *Восстановление после аварии.* Это процесс, который следует при попытке восстановления системы в случае, если вышло из строя оборудование, когда из-за какой-то нелепой ошибки оказались поврежденными данные, либо когда по той или иной причине сервер и хранящаяся на нем информация стали недоступны или непригодны к использованию (потенциальных причин множество). Вероятность какой-то одной конкретной аварии довольно мала, но ведь они суммируются. Нужно быть готовым ко всему: случайному подключению не к тому серверу и выполнению команды ALTER TABLE, пожару в здании, злонамеренным действиям хакера, ошибке в коде MySQL.

2. *Аудит*. Иногда нужно знать, как выглядели данные или схема в какой-то момент в прошлом. Например, вы можете оказаться участником судебного процесса или обнаружить в приложении ошибку и задаться вопросом, что когда-то делала программа (иногда версии кода, хранящейся в системе управления версиями, бывает недостаточно).

3. *Тестирование.* Один из самых простых способов протестировать приложение на реальных данных состоит в том, чтобы периодически «заливать» на тестовый сервер свежую информацию с промышленного. Если имеется резервная копия, то достаточно просто восстановить с нее данные.

Проверьте справедливость своих допущений. Например, считаете ли вы, что поставщик совместного хостинга регулярно делает резервные копии сервера MySQL, который предо-

ставляется вам по договору? Хотя разделяемый хостинг имеет мало общего с высокой производительностью, мы хотим подчеркнуть, что подобные непроверенные предположения могут привести к печальным последствиям. Заметим, что многие поставщики хостинга не занимаются резервным копированием MySQL, тогда как другие просто копируют файлы на работающем сервере, поэтому созданная копия, скорее всего, окажется бесполезной (несогласованной).

## 2.2. Компромиссы при резервировании данных

Резервное копирование MySQL сложнее, чем кажется на первый взгляд. Можно сказать, что резервная копия – это просто копия данных, но из-за требований приложения, архитектуры подсистем хранения в MySQL и конфигурации вашей системы получить такую копию может оказаться совсем нелегко.

Если известно, сколько данных можно относительно безболезненно потерять, то к выработке стратегии резервного копирования можно подойти более осмысленно. Нужна ли возможность воссоздания баз на конкретный момент времени или достаточно восстановить данные с копии, снятой прошлой ночью, смирившись с потерей всей проделанной с тех пор работы? Если восстановление на конкретный момент времени необходимо, то, вероятно, придется осуществлять регулярное копирование и включить режим записи в двоичный журнал, чтобы можно было вернуть данные, а затем накатить на них журналы до требуемого момента.

В целом, чем больше данных вы готовы потерять, тем проще процедура резервного копирования. Если же требования очень жесткие, то гарантировать воссоздание всей информации гораздо сложнее. Существуют даже разные варианты восста-

новления на конкретный момент времени. «Мягкие» требования означают допустимость воссоздания данных в состоянии, «достаточно близком» к моменту сбоя. «Жесткие» требования подразумевают необходимость восстановить все зафиксированные транзакции, даже если произошло что-то ужасное (например, сервер в буквальном смысле сгорел). Для этого применяют специальные методы, например, хранят двоичные журналы на отдельном SAN-томе или используют DRBD-репликацию диска.

Если MySQL на время резервного копирования допустим, то это самый безопасный и вообще наилучший способ получить копию данных с минимальными шансами внести искажения или несогласованность. Если сервер остановлен, то информацию можно копировать, не заботясь о таких неприятностях, как «грязные» буферы в пуле буферов InnoDB или в других кешах. Не нужно беспокоиться и о том, что данные модифицируются в процессе резервного копирования, а поскольку сервер не испытывает никакой нагрузки со стороны приложения, то копирование происходит быстрее.

Однако вывод сервера из эксплуатации обходится дороже, чем может показаться. Даже если удается свести к минимуму само время простоя, остановка и запуск MySQL могут занять довольно много времени, если нагрузка и объем данных велики.

Если в пуле буферов InnoDB много «грязных» буферов, т. е. велик объем данных, которые в памяти уже модифицированы, но на диск еще не записаны, то для их сохранения у InnoDB может уйти много времени. Время остановки InnoDB можно контролировать с помощью конфигурационной переменной innodb_fast_shutdown, которая определяет режим работы с пулом буферов и буфером вставки в ходе процедуры остановки, но это лишь переносит работу на другое время, а не

устраняет ее необходимость. Поэтому существенно сократить общее время остановки и запуска таким образом не удастся. Иногда это можно сделать путем настройки других аспектов InnoDB, но такие изменения конфигурации оказывают гораздо более широкое влияние на производительность. Перезапуск тоже может занять много времени. Открытие всех таблиц и прогрев кешей – длительный процесс, особенно если таблиц и данных много. Если присвоить параметру innodb_fast_shutdown значение 2, то остановка InnoDB происходит быстро, зато во время запуска подсистемы хранения инициирует полную процедуру восстановления. И даже после того, как сервер запустился, пройдет еще немало времени, прежде чем он как следует «прогреется» и будет готов к полноценной работе.

Следовательно, стремясь к достижению высокой производительности, вы почти наверняка захотите спроектировать процедуру резервного копирования так, чтобы не переводить промышленный сервер в автономный режим. Но в зависимости от требований, предъявляемых к согласованности данных, резервное копирование в оперативном режиме все равно может означать прерывание обслуживания на заметное время.

Например, один из наиболее часто упоминаемых методов резервного копирования начинается с выполнения команды FLUSH TABLES WITH READ LOCK. Тем самым мы говорим MySQL, чтобы он сбросил и заблокировал все таблицы, а также опустошил кеш запросов. На это требуется время: сколько конкретно, предсказать невозможно. Оно окажется большим, если для получения глобальной блокировки чтения придется ждать завершения выполнения длительной команды или если количество таблиц велико. Пока блокировки не будут освобождены, изменение данных на сервере невозможно. Команда FLUSH TABLES WITH READ LOCK обходится не так дорого,

как остановка сервера, потому что большая часть данных остается кешированной в памяти и сервер «прогрет», но все равно она мешает нормальной работе.

Если это проблема, то придется искать альтернативу. Например, один из применяемых методов состоит в том, чтобы делать резервную копию на подчиненном сервере репликации, который входит в пул серверов, так что его можно останавливать и перезапускать со сравнительно небольшими издержками. Мы еще вернемся к вопросу, касающемуся оперативного и автономного резервного копирования. А пока скажем лишь, что в современных версиях MySQL реализовать оперативное резервное копирование без прерывания обслуживания затруднительно.

*Логическое и физическое резервное копирование.* Существует два основных способа резервного копирования данных в MySQL: логическое (такая копия называется также «дампом») и путем копирования исходных файлов. В логической резервной копии данные представлены в формате, который MySQL может интерпретировать как SQL-команды или как текст с разделителями. Исходные файлы – это просто файлы в том виде, в каком они находятся на диске.

У каждого способа копирования данных есть свои плюсы и минусы.

*Логические резервные копии.* У логических резервных копий есть следующие достоинства:

1) это обычные файлы, которые можно обрабатывать с помощью стандартных текстовых редакторов и таких инструментов командной строки, как grep или sed. Это очень полезно, когда требуется вернуть информацию или просто просмотреть ее;

2) из них легко восстанавливать данные. Достаточно просто подать файл по конвейеру на вход программы mysql или воспользоваться программой mysqlimport;

3) резервное копирование и возврат данных можно выполнять по сети, т. е. не на той же машине, где работает сервер MySQL;

4) процедуру можно очень гибко настраивать, потому что mysqldump – инструмент, которым многие предпочитают пользоваться для снятия логических копий, – принимает множество параметров, например, фразу WHERE, позволяющую указать, какие строки включать в резервную копию;

5) они не зависят от подсистемы хранения. Поскольку для создания логической копии данные запрашиваются у сервера MySQL, то различия между подсистемами хранения нивелируются. Следовательно, очень просто снять копию таблицы типа InnoDB и восстановить ее в таблицу типа MyISAM. С физическими резервными копиями это невозможно;

6) задав подходящие флаги при вызове mysqldump, во многих случаях можно даже импортировать логическую копию в другую СУБД, например, PostgreSQL;

7) они могут помочь избежать повреждения данных. Если диски сбоят, то при копировании физических файлов получится поврежденная резервная копия, и если вы специально не проверите ее по окончании резервного копирования, то узнаете об этом только тогда, когда не сможете воспользоваться ей в целях восстановления. Когда же данные MySQL в памяти не повреждены, то иногда можно получить заслуживающую доверия логическую копию, если снять хорошую физическую не удается.

Но у логических резервных копий есть и недостатки:

1) для их генерации требуется работа сервера, так что процессор загружается сильнее. В некоторых случаях логиче-

ские копии оказываются объемнее исходных физических файлов. ASCII-представление данных не всегда так же эффективно, как внутреннее представление в подсистеме хранения. Например, для хранения целого числа необходимо 4 байта, но для записи его в виде ASCII-текста может понадобиться до 12 символов. Зачастую логические копии хорошо поддаются сжатию, но для этого нужно дополнительное процессорное время;

2) потеря точности в представлении чисел с плавающей точкой может помешать правильному восстановлению данных из файлов дампа (в состав заплаты Google для MySQL входит доработка программы mysqldump, решающая эту проблему);

3) для восстановления данных из логической копии необходимо загружать, интерпретировать команды и перестраивать индексы, что возлагает на сервер еще больше работы.

Но самый главный недостаток – стоимость выгрузки данных из MySQL и обратной их загрузки с помощью команд SQL.

*Физические резервные копии.* У физических резервных копий есть следующие достоинства:

1) для получения физической копии нужно просто скопировать требуемые файлы в другое место. Никакой дополнительной работы для их генерации выполнять не придется;

2) трудоемкость возврата данных из физической копии может быть проще и зависит от подсистемы хранения. В случае с MyISAM достаточно просто скопировать файлы в исходное место, тогда как для InnoDB требуется остановить сервер и, возможно, предпринять еще какие-то действия;

3) физические копии можно переносить между платформами, операционными системами и версиями MySQL;

4) восстановление с физических копий может оказаться быстрее, потому что серверу не нужно выполнять SQL-команды

и строить индексы. При наличии таблиц InnoDB, которые не помещаются целиком в память сервера, восстановление данных из физических файлов может быть гораздо быстрее.

К недостаткам можно отнести следующее.

Объем физических файлов InnoDB, как правило, гораздо больше соответствующих логических копий. Обычно в табличном пространстве InnoDB очень много неиспользуемого места. К тому же какое-то пространство отведено под цели, не связанные с хранением табличных данных (буфер вставки, сегмент отката и т. д.). Не всегда физическую копию можно перенести на другую платформу, операционную систему или версию MySQL. В частности, препятствием может стать чувствительность к регистру букв и формат чисел с плавающей точкой. Перенос файлов в систему с другим форматом чисел с плавающей точкой вообще невозможен (впрочем, в большинстве современных процессоров применяется формат IEEE).

Работать с физическими копиями обычно проще и эффективнее.

Не считайте резервную копию (особенно физическую) пригодной для использования, пока не проверили ее. В случае InnoDB это означает, что нужно запустить экземпляр MySQL, дать InnoDB завершить процедуру восстановления, а затем выполнить команду CHECK TABLES. Можно опустить этот шаг или просто проверить файлы утилитой innochecksum, но мы не советуем так делать. Для MyISAM следует выполнить команду CHECK TABLES или воспользоваться утилитой myisamchk.

Можно также пойти на хитрость и скомбинировать оба подхода: снять физические копии, а потом запустить экземпляр MySQL и с его помощью создать из них логические копии. В результате вы получаете все лучшее от обоих подходов, не создавая на промышленном сервере излишнюю нагрузку по

формированию дампа. Это особенно удобно, когда имеется возможность делать мгновенные снимки файловой системы – вы создаете снимок, копируете его на другой сервер, разворачиваете, а затем проверяете физические файлы и выполняете логическое резервное копирование.

Требования к восстановлению диктуют, что нужно копировать. Простейшая стратегия состоит в том, чтобы скопировать данные и определения таблиц, однако, это лишь необходимый минимум. Обычно для полного восстановления промышленного сервера требуется гораздо больше. Перечислим кое-что из того, что стоит включать в состав резервной копии MySQL.

*Неочевидные данные.* Не забудьте о данных, которые не бросаются в глаза, например, двоичные журналы и журналы транзакций InnoDB.

*Код.* В современном сервере MySQL может содержаться много программного кода, в частности, триггеры и хранимые процедуры. Если вы выполняете резервное копирование базы данных mysql, то большая часть этого кода в нее войдет. Однако тогда становится трудно восстановить единственную базу из всего множества, поскольку часть «данных» в этой базе, к примеру, хранимые процедуры, на самом деле содержится в базе mysql.

*Конфигурация репликации*. Для восстановления сервера, участвующего в репликации, следует включать в резервную копию все необходимые для репликации файлы: двоичные журналы, журналы ретрансляции, индексные файлы журналов и info-файлы. Как минимум, следует добавить результаты выполнения команды SHOW MASTER STATUS и/или SHOW SLAVE STATUS. Полезно также выполнить команду FLUSH LOGS, чтобы MySQL начала новый двоичный журнал.

Восстановление на конкретный момент времени проще делать, когда отсчет ведется от начала, а не от середины журнала.

Если потребуется восстановить данные после настоящей катастрофы, скажем, выстроить сервер с нуля в новом центре обработки данных после землетрясения, то вы оцените полезность хранения конфигурационных файлов сервера в составе резервной копии.

*Отдельные файлы операционной системы.* Как и в случае с конфигурацией сервера, очень важно сохранить все внешние конфигурационные файлы, существенные для работы операционной системы. На UNIX-сервере это могут быть таблицы cron, конфигурация пользователей и групп, административные сценарии и правила sudo.

Подобные рекомендации во многих случаях расцениваются как совет: «Копируйте все». Но, если информации очень много, то это может оказаться слишком дорогим удовольствием, поэтому при выполнении резервного копирования стоит проявить изворотливость. Например, данные, двоичные журналы и конфигурационные файлы операционной системы и сервера можно копировать по отдельности.

*Инкрементное резервное копирование.* При наличии большого объема данных общепринятой стратегией является регулярное инкрементное копирование. Рассмотрим основные особенности данного типа копирования:

1. Делайте резервную копию двоичных журналов. Это самый простой, наиболее распространенный и в общем случае наилучший способ создания инкрементных резервных копий.

2. Не копируйте таблицы, которые не изменялись. Некоторые подсистемы хранения, например, MyISAM, записывают время последней модификации каждой таблицы. Узнать это время можно, заглянув в файл на диске или воспользовавшись

командой SHOW TABLE STATUS. При использовании InnoDB можно написать триггер, который поможет отследить время последнего изменения, занося его в специальную маленькую таблицу. Но это нужно делать только для таблиц, которые изменяются редко, чтобы накладные расходы были минимальны. Специальный сценарий резервного копирования сможет легко определить, какие таблицы были изменены. Справочные таблицы, содержащие названия месяцев на разных языках или сокращенные названия государств или регионов, имеет смысл поместить в отдельную базу данных, которая не копируется каждый раз.

3. Не копируйте строки, которые не изменялись. Если записи в таблицу только добавляются (командой INSERT), как, например, в таблицу, где хранится протокол посещения страниц веб-сайта, то можно включить в нее столбец типа TIMESTAMP и включать в резервную копию только строки, добавленные с момента последнего копирования. Можно также применить подсистему хранения Merge, что позволит хранить старые данные в статических таблицах. Некоторую информацию не копируйте вовсе. Иногда это вполне оправдано – если имеется хранилище, которое создается из других данных и, строго говоря, является избыточным, то можно включать в копию лишь те элементы, которые нужны для воссоздания хранилища, а его же не копировать. Эта идея может оказаться здравой, даже если воссоздавать хранилище из исходных данных очень долго.

Отказ от копирования всех данных может со временем принести многократно большую экономию, чем удалось бы получить при наличии полной копии. Кроме того, можно не копировать некоторые временные значения, например, таблицы, в которых хранятся данные о веб-сеансах.

Включайте в резервную копию только изменения в двоичном журнале. Чтобы вычислить разницу между двоичными журналами, можно воспользоваться программой rdiff и помещать только изменения с момента снятия последней копии (но при этом периодически делайте полные резервные копии). Еще один полезный инструмент, которым мы часто пользуемся, – программа rdiff-backup, которая объединяет функциональность rdiff и rsync для формирования законченного решения по резервному копированию. Или можно просто выполнять после снятия каждой копии команду FLUSH LOGS, чтобы начать новый журнал; тогда вычислять двоичные дельты вообще не понадобится. Включайте в резервную копию только изменения в файлах данных. Этот совет аналогичен предыдущему. Для данной цели в UNIX применяются все те же программы rdiff и rdiff-backup. Такая стратегия особенно полезна при работе с очень большими базами, которые мало изменяются.

Предположим, что из терабайта данных ежедневно изменяется только 50 Гбайт. Тогда имеет смысл копировать каждый день только двоичные дельты, а изредка делать полную копию.

Выигрыш состоит в том, что применить двоичные дельты к полной копии последовательной операцией дискового чтения/записи гораздо быстрее, чем накатывать двоичный журнал. Впрочем, собственно вычисление двоичной дельты может потребовать больше времени, чем снятие полной копии.

Недостаток инкрементного резервного копирования – повышенная сложность восстановления. Если вы вынуждены проводить восстановление в условиях стресса, то оцените, насколько проще вернуть данные из единственной копии по сравнению с последовательным накатыванием инкрементных копий. Так что, если имеется возможность снимать полные резервные копии, то мы рекомендуем ей воспользоваться.

В любом случае время от времени выполнять полное копирование необходимо – мы советуем делать это раз в неделю. Вряд ли стоит рассчитывать на то, что можно будет восстановиться, имея только инкрементные копии за год. Даже неделя – это трудоемко и рискованно.

*Подсистемы хранения и согласованность.* Наличие в MySQL различных подсистем хранения может существенно осложнить процедуру резервного копирования. Проблема заключается в том, как получить согласованную копию базы данных при наличии в ней таблиц произвольных типов.

На самом деле существуют два вида согласованности: согласованность данных и согласованность файлов.

*Согласованность данных.* При снятии резервной копии необходимо гарантировать, что данные в копии согласованы по времени. Например, в базе данных сайта электронной торговли счета-фактуры и платежи должны быть согласованы друг с другом. Восстановление платежа без соответствующего счета или наоборот – прямая дорога к неприятностям.

При оперативном резервном копировании (без остановки сервера) необходимо получить согласованные дубликаты всех взаимосвязанных таблиц. Это означает, что нельзя просто блокировать и копировать таблицы по одной, т. е. процедура резервного копирования должна знать о структуре базы больше, чем хотелось бы. Если используется нетранзакционная подсистема хранения, то не остается другого выбора, как выполнить команду LOCK TABLES для всех таблиц, которые должны копироваться вместе, и освободить блокировки только после того, как процедура будет полностью завершена.

Встроенная в InnoDB технология MVCC способна помочь в этом отношении. Вы можете начать транзакцию, скопировать группу взаимосвязанных таблиц и потом зафиксировать тран-

закцию. Не следует одновременно с этим использовать команду LOCK TABLES, если вы хотите получить согласованную копию, поскольку она неявно фиксирует транзакцию. Если установлен уровень изоляции REPEATABLE READ, то этот метод позволит получить идеально согласованный по времени снимок данных, не блокируя при этом работу сервера на время снятия копии.

Однако такой подход не защитит от неудачно спроектированной логики приложения. Предположим, что интернет-магазин вставляет запись о платеже, фиксирует транзакцию, а затем вставляет счет-фактуру уже в другой транзакции. Процедура резервного копирования может начаться между двумя этими операциями, и тогда платеж войдет в копию, а счет-фактура – нет. Поэтому к использованию транзакций следует подходить со всей ответственностью, объединяя взаимосвязанные операции вместе.

Получить согласованную логическую копию таблиц InnoDB позволяет также программа mysqldump, которая поддерживает флаг --single-transaction, делающий в точности то, что было описано выше. Но при этом могут возникать очень длинные транзакции, что при некоторых характеристиках рабочей нагрузки приводит к неприемлемо высоким издержкам.

Помочь в резервном копировании групп взаимосвязанных таблиц могут инструменты, поддерживающие «групповое резервное копирование», например, ZRM (будет рассмотрен ниже) или mkparallel-dump, входящий в комплект Maatkit.

*Согласованность файлов.* Не менее важна внутренняя согласованность каждого файла. Так, в резервную копию не должен попасть файл, состояние которого отражает частичное выполнение большой команды UPDATE.

Кроме того, необходимо, чтобы все скопированные файлы были согласованы друг с другом. Если внутренняя согласованность нарушена, то при восстановлении вас поджидают неприятные сюрпризы (скорее всего, данные будут повреждены). А если взаимосвязанные объекты копируются в разные моменты времени, то они не будут согласованы между собой.

Примерами могут служить файлы MyISAM с расширениями .MYD и .MYI.

В случае нетранзакционной подсистемы хранения, в частности MyISAM, единственный вариант – заблокировать и сбросить таблицы.

Иными словами, нужно выполнить команды LOCK TABLES и FLUSH TABLES, чтобы сервер сохранил на диск накопившиеся в памяти изменения, или команду FLUSH TABLES WITH READ LOCK. По завершении сброса можно безопасно копировать физические файлы, в которых хранятся таблицы MyISAM.

В случае InnoDB гарантировать согласованность файлов на диске несколько сложнее. Даже после команды FLUSH TABLES WITH READ.

LOCK InnoDB не прекращает работу в фоновом режиме: потоки буфера вставки, журнала и записи продолжают сохранять изменения в журнал и файлы табличного пространства. Эти потоки задуманы асинхронными – именно выполнение работы в фоновых режимах позволяет InnoDB достигать высокого уровня конкуренции, поэтому от команды LOCK TABLES они не зависят. Следовательно, нужно гарантировать не только внутреннюю согласованность каждого файла, но и то, что файлы журналов и табличного пространства копируются в один и тот же момент времени.

Если резервная копия снимается, когда некоторый поток изменяет файл, или файлы журналов копируются не одновре-

менно с файлами табличного пространства, то после восстановления данные могут оказаться поврежденными. Избежать такого развития событий можно двумя способами:

1. Дождаться, пока потоки вытеснения и объединения с буфером вставки завершат работу. Можно следить за тем, что выводит команда SHOW INNODB STATUS, и начинать копирование, когда не останется «грязных» или ожидающих операций записи буферов. Но на это может уйти много времени, и все равно из-за наличия фоновых потоков безопасность не гарантируется. Поэтому данный подход не совсем удобен.

2. Делать согласованный снимок файлов данных и журналов с помощью такой системы, как LVM. Обязательно следует снимать файлы данных и журналов согласованно относительно друг друга; делать снимки по отдельности бессмысленно.

После того как файлы куда-то скопированы, можно снять блокировки и продолжить эксплуатацию сервера MySQL в обычном режиме.

*Репликация.* Анализируя результаты сравнений различных СУБД, стоит отметить, что механизм репликации в MySQL идеален для резервного копирования. У использования репликации в качестве составной части общей стратегии резервного копирования есть свои достоинства.

Основное преимущество резервного копирования на подчиненном сервере состоит в том, что не нужно прерывать работу и дополнительно нагружать главный сервер. Это основная причина для организации подчиненного сервера, даже если он не нужен с целью балансирования нагрузки или обеспечения высокой доступности. Если бюджет ограничен, то всегда можно использовать сервер резервного копирования и в других целях, например для генерации отчетов при условии, что вы не

будете производить на нем никаких операций записи, изменяющих резервируемые данные.

Подчиненный сервер необязательно должен быть целиком выделен только для копирования; важно лишь, чтобы он успевал догнать главный к моменту снятия следующей копии, если другие занятия не позволяют ему реплицировать информацию без задержек постоянно.

Делая резервные копии на подчиненном сервере, не забывайте сохранять всю информацию о процедуре репликации, например, позицию подчиненного сервера относительно главного. Это полезно для клонирования новых подчиненных серверов, для повторного применения двоичных журналов к главному серверу с целью восстановления на конкретный момент времени, для повышения подчиненного сервера до уровня главного и для многих других целей.

Кроме того, убедитесь, что в момент остановки подчиненного сервера нет открытых временных таблиц, поскольку это может помешать возобновлению репликации.

Для восстановления после некоторых сбоев бывает очень полезна намеренная задержка репликации. Предположим, что вы производите репликацию с паузой в один час. Если на главном сервере была выполнена нежелательная команда, то у вас есть час, чтобы это заметить и остановить подчиненный сервер до того, как он воспроизведет это событие из своего журнала ретрансляции. Затем можно назначить подчиненный сервер главным и выполнить несколько событий из журнала, обойдя ненужные команды. Это может оказаться намного быстрее, чем техника восстановления на конкретный момент времени. Сценарий mk-slave-delay из комплекта Maatkit поспособствует в решении этой задачи.

Информация на подчиненном и главном сервере может различаться.

Часто думают, что подчиненные сервера – точные копии главных, но опыт показывает, что расхождения в данных – обычное дело, и у MySQL нет средств для обнаружения этой проблемы. Резервное копирование некорректной или поврежденной информации на подчиненном сервере не приведет ни к чему хорошему.

Наличие реплицированной копии может защитить от таких проблем, как расплавление диска на главном сервере, но никаких гарантий нет.

Репликация – это не резервное копирование.

*Резервное копирование двоичных журналов.* Двоичные журналы сервера – одна из важнейших вещей, которые следует включать в резервную копию. Они совершенно необходимы для восстановления на конкретный момент времени, а поскольку их размер обычно меньше, чем сами данные, то копировать их можно чаще. Если существует резервная копия информации, снятая в какой-то момент времени, и все двоичные журналы, накопившиеся с этого момента, то эти журналы можно воспроизвести, т. е. «накатить» изменения, произведенные с момента последней полной резервной копии.

В MySQL двоичный журнал также используется для репликации. Это означает, что стратегии резервного копирования и восстановления тесно связаны с конфигурацией репликации.

Двоичные журналы требуют особого сохранения. Если вы потеряли данные, то утратить еще и журналы было бы совсем некстати. Чтобы уменьшить риск потери журналов, их можно хранить на отдельном томе. Это нормально, даже если вы хотите делать снимки двоичных журналов средствами LVM. Для большей безопасности журналы можно разместить на SAN-

хранилище или реплицировать на другое устройство с помощью DRBD.

Рекомендуется проводить резервное копирование не реже одного раза в неделю. Если же потеря данных более чем за 30 мин недопустима, то копируйте их, по крайней мере, раз в 30 мин. Можно также использовать подчиненный сервер репликации, работающий в режиме только чтения, задав флаг --log_slave_ updates. Позиции в журналах подчиненного и главного сервера, конечно, не будут совпадать, но обычно найти нужную для восстановления точку не составляет труда.

Ниже приведена рекомендуемая конфигурация для записи в двоичный журнал.

Есть и еще несколько конфигурационных параметров, относящихся к двоичному журналу, например настройки для ограничения размера одного журнала.

*Формат двоичного журнала.* Двоичный журнал содержит последовательность событий. У каждого события имеется заголовок фиксированной длины, в котором хранится разнообразная информация, в частности, текущая временная метка и имя подразумеваемой по умолчанию базы данных. Для просмотра содержимого двоичного журнала можно воспользоваться инструментом mysqlbinlog, который распечатывает часть сведений из заголовка. Ниже приведен пример вывода:

```
1. shell# at 2//
2. #071030 10:47:21 server id 3 end_log_pos 369 Query thread_ld=13
3. SET TIMESTAMP–1193755641/*! */;
4. insert into test(a) values/)/*!*/;
```

В строке 1 указано смещение в байтах от начала файла журнала (в данном случае 277).

В строке 2 приведена следующая информация:

1. Дата и время события, MySQL использует их для генерации команды SET TIMESTAMP.

2. Идентификатор исходного сервера, который необходим для предотвращения зацикливания репликации и других проблем.

3. Величина end_log_pos, т. е. смещение начала следующего события в байтах. Для большинства событий в транзакции из нескольких команд это значение некорректно. Во время выполнения транзакции MySQL копирует события в буфер на главном сервере, но позиция следующего события в журнале в этот момент неизвестна.

4. Тип события. В примере выше это Query, но существует много других типов.

5. Идентификатор потока, обработавшего событие на исходном сервере; важно для аудита и для выполнения функции CONNECTION_ID().

6. Величина exec_time, истинное назначение которой неясно даже отдельным разработчикам MySQL. Обычно здесь хранится время, затраченное на выполнение команды, но при некоторых условиях демонстрируются достаточно странные показатели. Например, вы увидите очень большое значение в журнале ретрансляции на подчиненном сервере, если его поток ввода/вывода далеко отстал от главного сервера, пусть даже на главном сервере команды были выполнены очень быстро. Мы советуем не полагаться на эту величину.

7. Код ошибки, возникшей при обработке события на исходном сервере. Если при воспроизведении на подчиненном сервере возникнет иная ошибка, репликация на всякий случай прекратится.

В остальных строках печатаются SQL-команды, необходимые для воспроизведения события. Здесь же вы найдете

пользовательские переменные и прочие специальные установки, например, временную метку на момент начала обработки команды.

При использовании построчной репликации, появившейся в версии MySQL 5.1, событие не содержит текста SQL-команд, а является «образом» модификаций, произведенных данной командой в таблице. Эти события не предназначены для чтения человеком.

Безопасное удаление старых двоичных журналов. Необходимо продумать стратегию удаления двоичных журналов, чтобы MySQL не заполнила ими весь диск. До какого размера вырастет журнал, зависит от его формата и от рабочей нагрузки (при построчной репликации, появившейся в версии MySQL 5.1, размер одной записи больше). Мы рекомендуем по возможности хранить журналы до тех пор, пока они не станут бесполезны. Двоичные журналы могут пригодиться для настройки подчиненных серверов репликации, для анализа рабочей нагрузки, для аудита и для восстановления на конкретный момент времени с помощью полной резервной копии. Учитывайте все это, когда будете решать, насколько долго хранить журналы.

Обычно используют конфигурационную переменную expire_logs_days, которая говорит MySQL по истечении какого времени следует удалять журналы. До выхода версии MySQL 4.1 этой переменной не существовало, поэтому журналы приходилось удалять вручную.

Значение параметра expire_logs_days считывается на этапе запуска сервера или в момент, когда MySQL ротирует двоичный журнал, поэтому если он никогда не заполняется до конца и не ротируется, то сервер и не будет удалять старые записи. Решение о том, какие файлы уничтожить, сервер принимает, исходя из даты модификации файла, а не из его содержимого.

## 2.3. Резервное копирование данных

Существуют хорошие и плохие способы резервного копирования, причем самые очевидные из них не обязательно лучшие.

Хитрость состоит в том, чтобы по максимуму задействовать пропускную способность сети, диска и процессора и постараться закончить копирование как можно быстрее. Для того, чтобы найти «золотую середину», придется поэкспериментировать.

*Снятие логической резервной копии.* Прежде всего, нужно отчетливо понимать, что существует две разновидности логических копий: SQL-дампы и файлы с разделителями.

*SQL-дампы*. SQL-дамп лучше знаком пользователям, так как это именно то, что по умолчанию создает программа mysqldump. Файл дампа содержит как описание таблицы, так и данные, причем и то и другое представлено в виде SQL-команд. Файл начинается с комментариев, в которых устанавливаются значения различных параметров MySQL.

Они включены для того, чтобы помочь при восстановлении, а также ради совместимости и корректности. Затем идет описание таблицы и данные. В самом конце сценарий воссоздает параметры, которые были изменены в начале дампа.

Созданный дамп может быть выполнен для восстановления данных. Это удобно, но подразумеваемые по умолчанию параметры mysqldump не годятся для снятия очень больших резервных копий.

Программа mysqldump не единственный инструмент, позволяющий создать логическую резервную копию. Это можно сделать, к примеру, с помощью phpMyAdmin. Но мы хотим отметить не столько проблемы, свойственные тому или иному

инструменту, а недостатки монолитной логической копии как таковой. Так, схема и данные хранятся вместе. Хотя это удобно, когда нужно восстановить все из одного файла, но усложняет задачу, если требуется восстановить только одну таблицу или только данные. Эту трудность можно устранить, произведя выгрузку дампа дважды – один раз для данных, другой – для схемы, но следующих проблем все равно не избежать.

*Огромные SQL-команды.* Разбор и выполнение всех SQL-команд в дампе требует от сервера значительных усилий. Поэтому загрузка данных происходит довольно медленно.

*Один гигантский файл.* Большинство текстовых редакторов не умеют обрабатывать файлы с очень длинными строками. Хотя в некоторых случаях для извлечения нужных данных можно воспользоваться потоковыми редакторами, например sed или grep, но лучше, чтобы файлы были поменьше.

Создание логической копии обходится дорого. Существуют более эффективные способы извлечь из MySQL данные помимо передачи их по протоколу взаимодействия между клиентом и сервером в виде результирующих наборов.

Все эти ограничения означают, что SQL-дамп становится непригодным для работы по мере роста таблиц. Однако существует и другая возможность: экспортировать информацию в файлы с разделителями.

*Резервные копии в виде файлов с разделителями.* Для создания логической копии данных в формате файла с разделителями можно воспользоваться командой SELECT INTO OUTFILE SQL. Ту же команду выполнит программа mysqldump, запущенная с флагом --tab. Файлы с разделителями содержат значения, представленные в кодировке ASCII, без SQL-команд, комментариев и имен столбцов.

Следующая команда выводит результат в формате CSV (с разделителями-запятыми) – общепринятом для представления табличных данных.

Результирующий файл компактнее и им проще манипулировать с помощью утилит командной строки, чем SQL-дампом. Но самое главное достоинство кроется не в этом, а в скорости резервного копирования и последующего возврата данных. Для загрузки информации назад в таблицу предназначена команда LOAD DATA INFILE, которой следует указать те же параметры, что при выгрузке.

*Параллельная выгрузка и загрузка.* Зачастую гораздо быстрее выгружать и загружать данные параллельно на машине с несколькими процессорами. Под словом «параллельно» мы здесь понимаем, что производится выгрузка либо загрузка нескольких таблиц сразу, а не синхронную работу нескольких программ над одной и той же таблицей. Когда два приложения одновременно загружают данные в одну таблицу, ничего хорошего, как правило, не получается.

Для параллельной выгрузки или загрузки не нужны какие-то особые инструменты; все можно сделать вручную, запустив несколько экземпляров программы резервного копирования. Тем не менее существуют специальные приложения и сценарии, предназначенные именно для этой цели, например mk-parallel-dump из комплекта Maatkit и mysqlp dump. Эталонные тесты показывают, что mk-parallel-dump выгружает данные в несколько раз быстрее, чем стандартная mysqldump.

В версии MySQL 5.1 программа mysqlimport поддерживает одновременный импорт несколькими потоками. Вариант mysqlimport из версии 5.1 может работать и в более ранних версиях MySQL.

Но если задать слишком высокую степень параллелизма, то параллельная выгрузка и загрузка данных может занять даже больше времени. Кроме того, иногда это приводит к фрагментации информации, что негативно сказывается на производительности.

*Восстановление из резервной копии.* На самом деле, наиболее важно восстановление. В этом разделе мы сосредоточимся на аспектах этого процесса, характерных именно для MySQL, в предположении, что вы знаете, как настраивать прочие части окружения. Регулярно практикуйте «учебные тревоги», чтобы точно знать, как восстанавливать данные, когда случится сбой. И обязательно проверяйте резервные копии.

Порядок восстановления зависит от того, как снималась резервная копия. Вам может понадобиться выполнить все или некоторые из перечисленных ниже шагов:

1. Остановить сервер MySQL.
2. Записать куда-нибудь конфигурацию сервера и права доступа к файлам.
3. Скопировать данные с резервной копии в каталог данных MySQL.
4. Внести изменения в конфигурационные файлы.
5. Изменить права доступа к файлам.
6. Запустить сервер в режиме ограниченного доступа и подождать, пока он перейдет в состояние готовности.
7. Загрузить логические файлы резервной копии.
8. Проверить и воспроизвести двоичные журналы.
9. Убедиться, что все восстановлено.
10. Перезапустить сервер в режиме полного доступа.

Если есть шанс, что текущие версии файлов еще понадобятся, не замещайте их файлами из резервной копии. Например, если копия не содержит двоичных журналов, а они необходимы для восстановления на конкретный момент времени,

то не затирайте текущие журналы устаревшими версиями из копии. При необходимости переименуйте их или скопируйте в какое-то другое место.

*Ограничение доступа к MySQL*. Во время восстановления часто бывает важно, чтобы к MySQL не мог обращаться никто, кроме процесса восстановления. В сложных системах дать такую гарантию трудно. Для уверенности в том, что сервер будет недоступен приложениям, пока мы все не проверим, мы запускаем его с флагами -skip-networking и --socket=/tmp/mysql_recover.sock. Это особенно важно для логических копий, которые загружаются по частям.

*Восстановление из физических файлов.* Процедура возврата данных из физических файлов довольно прямолинейна, иначе говоря, вариантов здесь немного. Хорошо это или плохо, зависит от требований к восстановлению. Обычно все сводится к простому копированию файлов туда, где им надлежит находиться.

Нужно ли останавливать MySQL, определяется подсистемой хранения. Файловые объекты MyISAM обычно не зависят друг от друга, поэтому простого копирования файлов с расширениями .frm, .MYI и .MYD для каждой таблицы достаточно даже на работающем сервере. Сервер найдет таблицу, как только к ней поступит запрос или будет выполнена иная команда, для которой необходима данная таблица (например, SHOW TABLES). Если во время копирования этих файлов таблица была открыта, то вполне возможны неприятности, поэтому предварительно удалите или переименуйте ее либо заблокируйте командами LOCK TABLES и FLUSH TABLES.

С InnoDB дело обстоит иначе. Если восстанавливается традиционно сконфигурированная база InnoDB (когда все таблицы хранятся в одном табличном пространстве), то нужно

будет остановить MySQL, скопировать или переместить файлы на место, а затем перезапустить сервер. Кроме того, нужно убедиться, что файл журнала транзакций соответствует файлам, содержащим табличное пространство. Если эти файлы не соответствуют друг другу, например, табличное пространство вы заменили, а про журнал транзакций забыли, то InnoDB откажется запускаться. Поэтому так важно включать в резервную копию не только файлы данных, но и журнал транзакций.

Если используется недавно появившаяся возможность размещать по одной таблице в каждом файле (режим innodb_file_per_table), то InnoDB хранит данные и индексы для каждой таблицы в файле с расширением .ibd, представляющем собой нечто вроде комбинации MYI и MYD-файлов в подсистеме MyISAM.

Резервное копирование и восстановление отдельных таблиц в этом случае может выполняться простым копированием этих файлов и делать это можно на работающем сервере, но не так просто, как в случае MyISAM. Отдельные файлы не являются независимыми от InnoDB в целом. В каждом IBD-файле записана информация, говорящая InnoDB о том, как этот файл связан с основным (общим) табличным пространством. При возврате такого файла необходимо попросить InnoDB «импортировать» его.

У этой процедуры много ограничений. Самое серьезное заключается в том, что восстановить таблицу можно только на тот сервер, с которого она была скопирована. Не то, чтобы резервное копирование и восстановление таблиц в этой конфигурации было вообще невозможно, но это сложнее, чем могло бы показаться.

Наличие таких сложностей означает, что восстановление физических файлов способно оказаться очень трудоемким

процессом и можно наделать ошибок. Эвристическое правило состоит в том, что чем сложнее и труднее становится процедура восстановления, тем важнее оградить себя от неприятностей, создавая также логические копии. Всегда полезно их иметь на случай, если что-то пойдет не так, и вы не сможете заставить MySQL использовать физическую копию.

*Запуск MySQL после восстановления физических файлов*. Перед тем как запускать восстановленный сервер MySQL, нужно сделать несколько вещей.

Первое и самое важное – проверить конфигурацию сервера и убедиться в том, что для восстановленных файлов задан правильный владелец и права доступа. Это нужно делать до попытки запустить сервер. Если эти атрибуты хоть чем-то отличаются от требуемых, MySQL может не запуститься. В различных системах они задаются по-разному, поэтому посмотрите в своих записях, как они должны быть установлены. Обычно для файлов и каталогов указывается владелец и группа mysql, причем владельцу и группе должны быть разрешены чтение и запись, а всем остальным запрещен всякий доступ.

Советуем также следить за журналом ошибок MySQL в ходе запуска. В UNIX-подобных системах для этого можно выполнить такую команду: $ tail -f /var/log/mysql/mysql.err. Точное местоположение журнала ошибок зависит от системы.

Начав мониторинг этого файла, запускайте сервер MySQL и наблюдайте за тем, что будет появляться в журнале. Если ошибок не возникнет, значит, сервер восстановился нормально и может работать.

Наблюдение за журналом ошибок особенно важно в последних версиях MySQL. В прежних реализациях сервер просто не запускался, если InnoDB обнаруживала ошибку, но теперь он запускается, однако, InnoDB отключается. Даже если

кажется, что сервер стартовал без проблем, следует в каждой базе данных выполнить команду SHOW TABLE STATUS и снова проверить журнал ошибок.

*Восстановление из логической копии.* Если восстановление производится из логической резервной копии, а не из физических файлов, то для загрузки данных в таблицы понадобится сам сервер MySQL. Копирования файлов на уровне операционной системы недостаточно.

Прежде чем загружать файл дампа, посмотрите на его размер и подумайте, сколько времени он будет обрабатываться и не надо ли что-то сделать перед началом процедуры, например, уведомить пользователей или деактивировать часть приложения. Полезно на этот период отключить запись в двоичный журнал, если только вы не хотите реплицировать восстанавливаемые данные на подчиненный сервер: загружать огромный дамп серверу и так-то нелегко, а запись в двоичный журнал только увеличивает издержки (возможно, без всякой необходимости). Кроме того, для некоторых подсистем хранения загрузка очень больших файлов требует особых предосторожностей. Так, загружать файл размером 100 Гбайт в таблицу InnoDB одной транзакцией – безумная затея, поскольку в результате образуется огромный сегмент отката. Необходимо разбить файл на порции и фиксировать транзакцию после загрузки каждой из них.

Существует две разновидности восстановления в соответствии с двумя видами логических копий.

Загрузка SQL-файлов SQL-дамп содержит исполняемые команды SQL. Вам остается только запустить его. В предположении, что демонстрационная база данных Prestupnost выгружена в один файл вместе со схемой, для восстановления дан-

ных обычно используется такая команда: $ mysql<prestupnost-backup.sql.

Можно также загрузить файл из клиента mysql командой SOURCE. Хотя это просто другой способ сделать то же самое, но некоторые вещи при этом оказываются проще. Например, если вы обладаете административными правами в MySQL, то можете отключить запись в двоичный журнал команд, выполняемых в текущем соединении, а затем загрузить файл без перезапуска сервера: mysql> SET SQL_LOG_BIN = 0; mysql> SOURCE prestupnost-backup.sql; mysql> SET SQL_LOG_BIN = 1.

Однако при использовании команды SOURCE имейте в виду, что ошибка не прерывает выполнение пакета команд, как это происходит по умолчанию в случае чтения из стандартного ввода mysql.

Если резервная копия была сжата, то не нужно перед загрузкой отдельно распаковывать ее. Разархивирование и загрузку можно объединить в одной операции, что будет гораздо быстрее: $ gunzip -c prestupnost-backup.sql.gz | mysql. Загрузка файлов с разделителями. При выгрузке данных командой SELECT INTO OUTFILE загружать их обратно следует командой LOAD DATA INFILE с теми же параметрами.

Можно также воспользоваться программой mysqlimport, которая является не чем иным, как оберткой вокруг LOAD DATA INFILE. Куда загружать данные из файла, утилита mysqlimport определяет на основании соглашений об именовании.

Необходимо, чтобы вы выгрузили не только данные, но и схему. В таком случае, это SQL-дамп и для его загрузки можно воспользоваться приемами из предыдущего раздела.

Для команды LOAD DATA INFILE существует замечательная оптимизация. Поскольку эта директива читает данные непосредственно из файла, то может возникнуть мысль пред-

варительно его распаковать, а это очень длительная операция, которая к тому же активно обращается к диску.

Но существует обходной путь, по крайней мере, в системах, поддерживающих «именованные каналы» (или FIFO-очереди), к каковым относится и GNU/Linux. Сначала создайте именованный канал и направьте в него распакованные данные:

```
$ mkfifo /tmp/backup/default/prestupnost/payment.fifo
$ chmod 666 /tmp/backup/default/prestupnost/payment.fifo
$ gunzip -c /tmp/backup/default/prestupnost/payment.txt.gz >
/tmp/backup/default/prestupnost/payment.fifo.
```

Обратите внимание на употребление знака >, для перенаправления распакованного потока в файл payment.fifo нужен именно он, а не знак «‘|’», который создает анонимный канал между программами. Payment.fifo – именованный канал, поэтому анонимный здесь ни к чему.

Канал ждет, пока какая-нибудь программа откроет его и начнет читать данные с другого конца. В этом-то и заключается особенность: сервер MySQL может получать распакованные данные из канала точно так же, как из обычного файла. Не забудьте отключить запись в двоичный журнал, если она не нужна: mysql> SET SQL_LOG_BIN = 0. Необязательно: -> LOAD DATA INFILE /tmp/backup/default/prestupnost/payment.fifo' -> INTO TABLE prestupnost.payment. После того, как MySQL закончит загрузку данных, программа gunzip завершится, и именованный канал можно будет удалить. Такую же технику можно применять для загрузки сжатых файлов из командного клиента mysql директивой SOURCE.

Проверка резервных копий. Если поменять тип столбца с DATETIME на TIMESTAMP, чтобы сэкономить место и ускорить обработку, то определение таблицы станет выглядеть так: CREATE TABLE tbl(colltimestamp NOT NULL, col2 timestamp

NOT NULL default CURRENT_TIMESTAMP onupdate CURRENT_TIMESTAMP, ... прочие столбцы ... ).

Такое определение таблицы приводит к синтаксической ошибке в версии MySQL 5.0.40, на которой оно было создано. Выгрузить данные удастся, а загрузить обратно – уже нет. Вот из-за таких странных, непредвиденных ошибок и надо проверять резервные копии.

*Восстановление на конкретный момент времени.* Самый распространенный способ восстановления на конкретный момент времени в MySQL заключается в том, чтобы вернуть данные с последней полной резервной копии, а затем воспроизвести двоичные журналы, накопившиеся за этот период (иногда это называется «накатом журналов»). Имея двоичный журнал, можно восстановиться на любой момент времени. Можно даже без особого труда восстановить только одну базу данных.

Нередко возникает задача отменить действие необдуманно выполненной команды, например DROP TABLE. Рассмотрим упрощенный пример, на котором покажем, как это сделать, когда имеются только таблицы типа MyISAM.

Предположим, что в полночь задание резервного копирования выполнило следующие команды, которые копируют базу данных в другое место на том же сервере:

mysql> FLUSH TABLES WITH READ LOCK;

-> server1# cp --a /var/lib/mysql/prestupnost/backup/prestupnost;

mysql> FLUSH LOGS; -> server1# mysql -e "SHOW MASTER STATUS" --vertical > /backup/master.info; mysql> UNLOCK TABLES.

Позже в тот же день кто-то по ошибке выполнил такую команду: mysql> USE prestupnost; mysql> DROP TABLE prestupnost.gruppirovka. Для простоты допустим, что мы можем корректно восстановить эту базу данных отдельно от других

(т. е. в этой базе не существует таблиц, которые были вовлечены в запросы, затрагивающие несколько баз данных). Предположим еще, что ошибочная команда была обнаружена спустя некоторое время. Наша задача – восстановить все, что происходило с этой базой данных, за исключением вышеупомянутой команды. Иными словами, мы должны сохранить все обновления в других таблицах, в том числе имевшие место после злополучной команды.

Это не так уж трудно сделать. Прежде всего остановим MySQL, чтобы предотвратить дальнейшие модификации, и вернем из резервной копии только базу данных prestupnost:

server1# /etc/init.d/mysql stop

server1# mv /var/lib/mysql/prestupnost /var/lib/mysql/ prestupnost.tmp server1# cp -a /backup/prestupnost /var/lib/mysql.

Запретим обычные соединения, временно добавив в файл my.cnf такие строчки:

Skip-networking socket=/tmp/mysql_recover.sock.

Теперь можно без опаски запустить сервер:

Server1# /etc/init.d/mysql start.

Следующая задача – найти в двоичном журнале те команды, которые мы хотим воспроизвести, и те, которые следует пропустить.

Как выясняется, после полуночи сервер создал только один двоичный журнал. Мы можем обработать его утилитой grep и найти ошибочно выполненную команду: server1# mysqlbinlog --database=prestupnost /var/log/mysql/mysql- bin.000215 | grep -B 3 -i ‘droptableprestupnost.gruppirovka’ # at 352

#070919 16:11:23 server id 1 end_log_pos 429 Query thread_id=16 exec_time=0 error_code=0 SET TIMESTAMP= 1190232683/*!*/;

DROP TABLE prestupnost.payment/*!*/:

Итак, команда, которую мы хотим пропустить, начинается с позиции 352 в файле журнала, а следующая за ней – с позиции 429.

Воспроизведем журнал до позиции 352 и далее с позиции 429 до конца: server1# mysqlbinlog --database=prestupnost /var/log/mysql/mysql-

bin.000215 --stop-position=352 | mysql -uroot -p

server1# mysqlbinlog --database=prestupnost /var/log/mysql/mysql-

bin.000215 --start-position=429 | mysql -uroot –p.

Теперь осталось только еще раз проверить данные, на всякий случай остановить сервер, вернуть файл my.cnf в исходное состояние и снова запустить сервер.

# ГЛАВА 3. БЕЗОПАСНОСТЬ

Рассматривая построение безопасных серверов баз данных, необходимо отметить, что безопасность возможна лишь при обеспечении полной защиты всего хоста сервера (а не только сервера MySQL) от различных видов атак: подслушивания трафика, изменения команд, воспроизведения и отказа в обслуживании. Однако в данной главе будут рассмотрены основные методы обеспечения безопасности исключительно MySQL сервера и раскрыты особенности построения систем защиты путем настройки MySQL.

MySQL имеет несколько рубежей защиты, но главный среди них основан на списках управления доступом (ACL) для всех соединений, запросов и других операций, которые пользователи могут попытаться выполнить. Кроме того, возможна реализация SSL-зашифрованных соединений между клиентами MySQL и серверами. Аналогично любому защищенному соединению SSH.

В целом, можно сказать, что многие из приведенных в пример концепций обеспечения безопасности сервера вообще не являются специфичными для MySQL; одни и те же системы обеспечения безопасности применимы почти ко всем информационным системам.

Прежде чем рассматривать особенности защиты путем настройки MySQL сервера, необходимо выделить ряд требований, обязательных к соблюдению:

1. Доступ к таблице user в MySQL базе данных должен быть ограничен. Права на чтение, изменение, запись должны быть только у администратора сервера.

2. Запрещено выделять больше привилегий, чем необходимо пользователям. Предоставление привилегий всем учетным записям без необходимости создает угрозу безопасности данных, хранящихся на сервере.

## 3.1. Основы учетных записей

Основное средство обеспечение безопасности учетных записей в MySQL – это паролевая защита. Пароли встречаются в нескольких аспектах в MySQL. В следующих разделах приведены рекомендации, позволяющие конечным пользователям и администраторам сохранять эти пароли в безопасности и избегать их компрометации. Также MySQL поддерживает плагины хеширования, что позволяет выстроить более безопасную паролевую защиту базы данных.

Говоря о системах безопасности, построенных на паролевой защите, необходимо рассматривать данную защиту в трех аспектах:

1. Паролевая защита конечного пользователя.
2. Пароли администратора.
3. Пароли и их хранение.

Пользователи MySQL должны использовать следующие рекомендации для обеспечения безопасности паролей:

1. При запуске клиентской программы для подключения к серверу MySQL, можно указать пароль в различных формах. Для многих систем имеется поддержка ввода пароля через консоль путем отправки строкового значения в открытом виде. Данный вариант недопустим по различным причинам (хранение истории терминала, перехват сообщения и пр.). Самые безопасные методы – это запрос клиентской программы на ввод пароля или указание пароля должным образом в защищенном файле опций.

2. Используйте утилиту mysql_config_editor, которая позволяет хранить учетные данные аутентификации в зашифрованном файле mylogin.cnf. Файл может быть прочитан позже

клиентскими программами MySQL для получения учетных данных аутентификации для подключения к серверу MySQL.

3. Используйте опцию -pyour_pass или --password=your_pass в командной строке:

```
shell>mysql -u username -p db_name

Enter password: ********
```

С практической точки зрения это удобно, однако, влечет за собой весьма большие риски. Так, пароль может быть раскрыт при системном вызове с доступом к шеллу. Большинство клиентских MySQL приложений, как правило, скрывают аргумент пароля командной строки нулями во время их инициализации. Однако существует весьма короткий интервал времени, в который ваш пароль может быть доступен через системный вызов. Кроме того, в некоторых системах эта стратегия перезаписи неэффективна, и пароль остается видимым для ps.

Если операционная среда, в которой находится сервер MySQL, настроена на отображение текущей команды в строке заголовка окна терминала, пароль остается видимым до тех пор, пока команда выполняется, даже если команда прокрутилась вне поля зрения в области содержимого окна.

4. Используйте опцию -p или --password в консоли, не указывая пароль. В этом случае клиентская программа отправит вам запрос на его ввод:

```
shell>mysql -u username -p db_name

Enter password: ********
```

Символы * указывают, где находится поле ввода пароля. Пароль не отображается при его вводе.

Это значительно безопаснее, однако даже такой способ ввода, при котором осуществляется интерактивный ввод (аналогично смене пароля в UNIX системах), несет за собой неко-

торые оговорки. К примеру, не получится использовать MySQL запросы в составе скриптов без предварительной инициализации. Так, некоторые интерпретаторы будут пытаться считать аргумент для интерактивного ввода в самом скрипте, что, разумеется, приведет к ошибке исполнения.

Для того, чтобы избежать этого, достаточно указать пароль в файле конфигурации [client] в разделе.my.cnf файла в вашем домашнем каталоге:

```
[client] password=your_pass
```

Однако при таком решении этой проблемы возникает новая уязвимость – непосредственно файл конфигурации. Для того, чтобы использовать такой способ хранения пароля, необходимо дополнительно разграничить права доступа к данному файлу конфигурации в системе. Для конкретных пользователей и администратора необходимо установить права доступа chmod 400 (chmoda+rwx,u-wx,g-rwx,o-rwx) либо Chmod 600 (chmoda+rwx,u-x,g-rwx,o-rwx):

```
shell>chmod 600.my.cnf
```

Чтобы вызвать из командной строки конкретный файл параметров, содержащий пароль, можно использовать параметр --defaults-file=file_name, где file_name – это полный путь к файлу:

```
shell>mysql --defaults-file=/home/francis/mysql--opts
```

В Unix клиент mysql записывает запись выполненных операторов в файл истории (см. журнал mysqlLogging). По умолчанию этот файл имеет имя.mysql_history и создается в вашем домашнем каталоге. Пароли могут быть записаны в виде обычного текста в SQL-операторах, таких как CREATE USER и ALTER USER, поэтому при использовании этих операторов они регистрируются в файле истории. Чтобы обезопасить дан-

ный файл, необходимо использовать разграничение доступа, аналогично тому, как было описано ранее для.my.CNF файла.

Если интерпретатор команд используемой операционной системы настроен на ведение истории, любой файл, в котором сохраняются команды, будет содержать пароли MySQL, введенные в командной строке не интерактивно. Например, bash использует ~/.bash_history. Любой такой файл должен иметь режим ограниченного доступа.

*Рекомендации администратору базы данных по обеспечению безопасности паролей.* Администраторы баз данных должны использовать следующие рекомендации для обеспечения безопасности паролей:

MySQL хранит пароли для учетных записей пользователей в mysql.usertable. Доступ к этой таблице никогда не должен предоставляться никаким неадминистративным учетным записям.

Должен быть установлен срок действия паролей учетных записей, поэтому пользователи должны их регулярно менять, не используя комбинации предыдущих паролей.

Плагин validate_password можно использовать для применения политики допустимого пароля, его настройка будет рассмотрена в следующих темах.

Пользователь, имеющий доступ для изменения каталога плагинов (значение системной переменной plugin_dir) или my.cnf-файл, указывающий расположение каталога плагинов, может заменять плагины и изменять возможности, предоставляемые плагинами, включая плагины аутентификации.

Такие файлы, как файлы журналов, в которые могут быть записаны пароли, должны также быть защищены.

*Пароли и ведение журнала.* Пароли могут быть записаны в виде обычного текста в SQL-операторах, таких как CREATE USER, GRANT, SET PASSWORD, и операторах, вызывающих

функцию PASSWORD (). Если такие операторы регистрируются сервером MySQL как записанные, пароли в них становятся видимыми любому, кто имеет доступ к журналам.

Ведение журнала операторов позволяет избежать записи паролей в открытом виде для следующих операторов:

```
CREATE USER ...
IDENTIFIED BY ...
ALTER USER ...
IDENTIFIED BY ...
GRANT ...
IDENTIFIED BY ...
SET PASSWORD ...
SLAVE START ...
PASSWORD = ...
CREATE SERVER ...
OPTIONS(... PASSWORD ...)
ALTER SERVER ...
OPTIONS(... PASSWORD ...)
```

Содержимое файла журнала аудита, созданного плагином журнала аудита, не шифруется. По соображениям безопасности этот файл должен быть записан в каталог, доступный только серверу MySQL и пользователям с обоснованной необходимостью для просмотра журнала (как упоминалось ранее, обеспечение безопасности данных журналов – задача администратора не сервера MySQL, а администратора операционной системы, в которой развернут MySQL).

Подразумевается, что операторы, которые не могут быть проанализированы (например, из-за синтаксических ошибок), не записываются в общий журнал запросов. В случаях, требующих протоколирования всех операторов, включая те, которые содержат ошибки, следует использовать параметр --log-raw, имея в виду, что это не касается интерактивного запроса пароля.

Переписывание пароля происходит только тогда, когда ожидаются простые текстовые пароли. Для операторов с синтаксисом, которые ожидают хеш-значение пароля, перезапись не происходит. Если обычный текстовый пароль указан ошибочно для такого синтаксиса, пароль регистрируется как заданный, без перезаписи. Например, следующая инструкция регистрируется, как показано ниже, поскольку ожидается хеш-значение пароля:

```
CREATE USER 'user1'@'localhost'
IDENTIFIED BY PASSWORD 'not-so-secret';
```

Чтобы защитить файлы журналов от несанкционированного доступа, необходимо располагать их в каталоге, доступ к которому имеется у сервера MySQL и администратора операционной системы.

*Хеширование паролей в MySQL.* Учетные записи пользователей MySQL сервера хранятся в таблице user базы данных MySQL. Каждой учетной записи MySQL может быть назначен пароль, хотя таблица user хранит не текстовую версию пароля, а хеш-значение, вычисленное из него. То есть при сбросе пароля, редактировании, создании новой учетной записи MySQL хеширует введенный пароль и только потом передает его в таблицу.

MySQL использует пароли в двух случаях взаимодействия клиент-серверной части:

1. В случае первоначальной аутентификации, при подключении к серверу. Тогда клиент предоставляет пароль, сервер хеширует его по своему алгоритму (в зависимости от версии MySQL это происходит по-разному), затем обращается к таблице с хешами и сверяет хеш введенного пароля и имеющегося в таблице по учетной записи, запрос к которой осуществляет пользователь.

2. После успешного подключения к серверу, в случае необходимости смены пароля либо сброса учетных данных (разумеется, для этого необходимы права), имеющихся в таблице. Тогда для смены пароля можно воспользоваться командой PASSWORD() для генерации хеша пароля либо использовать оператор создания пароля (CREATE USER, GRANT или SET PASSWORD).

Другими словами, сервер проверяет хеш-значения во время аутентификации, когда клиент впервые пытается подключиться. Сервер генерирует хеш-значения, если подключенный клиент вызывает функцию PASSWORD () или использует оператор генерации пароля для установки или изменения пароля.

В MySQL с развитием системы и технологий хеширования изменялись стандарты преобразования паролей, а также методов их хранения. Функция PASSWORD (), которая вычисляет хеш-значения паролей, варьировала длину строки в зависимости от версии MySQL.

*Базовая система хеширования.* Базовая система хеширования преобразовывала пароль в строку длиной 16 байт.

```
mysql>SELECT PASSWORD('mypass');
+——————————————+
| PASSWORD('mypass') |
+——————————————+
| 6f8c114b58f2ce9e     |
+——————————————+
```

MySQL 4.1 изменил систему хеширования паролей. Данное изменение обеспечило лучшую безопасность и снизило риск перехвата паролей. В итоге произошли следующие изменения:

1. Изменился формат значений пароля, создаваемых функцией PASSWORD().

2. Изменилось расширение столбца паролей.

3. Контроль над методом хеширования стал по умолчанию.

4. Контроль над разрешенными методами хеширования для клиентов, пытающихся подключиться к серверу, изменился в MySQL 4.1 в два этапа:

1) MySQL 4.1.0 использовал предварительную версию метода хеширования 4.1. Этот метод не прижился и в дальнейшем был изменен;

2) в MySQL 4.1.1 метод хеширования был изменен для получения более длинного 41-байтового хеш-значения:

```
mysql>SELECT PASSWORD('mypass');
+-------------------------------------------+
| PASSWORD('mypass')                        |
+-------------------------------------------+
| *6C8989366EAF75BB670AD8EA7A7FC1176A95CEF4 |
+-------------------------------------------+
```

Более длинный хеш-формат пароля обладает лучшими криптографическими свойствами, а аутентификация клиента на основе длинных хешей более безопасна, чем на основе старых коротких хешей.

Чтобы вместить более длинные хеши паролей, столбец пароля в пользовательской таблице был изменен в этой версии на 41 байт.

Расширенный столбец паролей может хранить хеши паролей как в форматах pre-4.1, так и в форматах 4.1. Формат любого заданного хеш-значения можно определить двумя способами:

1. Длина: хеши 4.1 и pre-4.1 составляют 41 и 16 байт соответственно.

2. Хеши паролей в формате 4.1 всегда начинаются с символа* в отличии от предыдущих версий.

Чтобы разрешить явную генерацию хешей паролей до версии 4.1, были внесены два дополнительных изменения:

1. Добавлена функция OLD_PASSWORD(), которая возвращает хеш-значения в 16-байтовом формате.

2. В целях совместимости была добавлена системная переменная old_passwords, позволяющая администраторам баз данных и приложениям контролировать метод хеширования. Значение old_passwords по умолчанию, равное 0, заставляет хеширование использовать метод 4.1 (41-байтовые хеш-значения), но установка old_passwords=1 приводит к хешированию по методу pre-4.1. В этом случае функция PASSWORD() выдает 16-байтовые значения и эквивалентна OLD_PASSWORD().

Чтобы разрешить администраторам баз данных управлять методами подключения клиентов, была добавлена системная переменная secure_auth. Запуск сервера с этой переменной отключен или включен, разрешает или запрещает клиентам подключаться с использованием более старого метода хеширования паролей до версии 4.1. До MySQL 5.6.5 secure_auth по умолчанию отключен. Начиная с версии 5.6.5, secure_auth включен по умолчанию для содействия более безопасной конфигурации по умолчанию. Администраторы баз данных могут отключить его по своему усмотрению, но это не рекомендуется, а хеши паролей до версии 4.1 устарели и их следует избегать.

Кроме того, клиент mysql поддерживает опцию --secure-auth, которая аналогична secure_auth, но со стороны клиента. Его можно использовать для предотвращения подключений к менее защищенным учетным записям, использующим хеширование паролей до версии 4.1. Эта опция отключена по умолчанию до MySQL 5.6.7, а затем вновь включена по умолчанию.

*Проблемы совместимости, связанные с методами хеширования.* Расширение столбца пароля в MySQL 4.1 с 16 байт до 41 байта влияет на операции установки или обновления следующим образом:

1) если вы выполняете новую установку MySQL, столбец пароля автоматически становится длиной 41 байт;

2) обновления с MySQL 4.1 или более поздней версии до текущих версий MySQL не должны вызывать никаких проблем в отношении столбца пароля, поскольку обе версии используют одинаковую длину столбца и метод хеширования паролей;

3) для обновления до версии 4.1 или более поздней версии необходимо обновить системные таблицы после обновления.

Метод хеширования 4.1 понятен только серверам и клиентам MySQL 4.1 (и выше), что может привести к некоторым проблемам совместимости. В 4.1 или выше клиент может подключиться к pre-4.1 серверу, потому что клиент понимает, как методы хеширования используются. Однако клиент до версии 4.1, пытающийся подключиться к серверу версии 4.1 или выше, может столкнуться с трудностями. Например, клиент MySQL версии 4.0 может выйти из строя со следующим сообщением об ошибке:

```
shell>mysql -h localhost --u root
Client does not support authentication protocol
requested by server;
consider upgrading MySQL client
```

Различия между короткими и длинными хешами паролей важны по ряду причин. Во-первых, при аутентификации, во-вторых, при создании или редактировании пароля сервер преобразует пароль:

1. Если столбец длинный, он может содержать либо короткие, либо длинные хеши и сервер может использовать любой формат:

1) клиенты pre-4.1 могут подключаться, но поскольку клиентское приложение может использовать только метод хе-

ширования pre-4.1, они могут аутентифицироваться только с использованием учетных записей с короткими хешами;

2) 4.1 и более поздние версии клиенты могут аутентифицироваться с помощью учетных записей с короткими или длинными хешами.

Даже для учетных записей с коротким хешем процесс аутентификации на самом деле немного более безопасен для клиентов 4.1 и более поздних версий, чем для старых клиентов. С точки зрения безопасности в порядке надежности методы подключения выглядят так:

1. Предварительная аутентификация клиента 4.1 с помощью короткого хеша пароля.
2. Аутентификация клиента 4.1 или более поздней версии с помощью короткого хеша пароля.
3. Аутентификация клиента 4.1 или более поздней версии с помощью длинного хеша пароля.

Способ, которым сервер генерирует хеши паролей для подключенных клиентов, зависит от ширины столбца Password и системной переменной old_passwords. Сервер версии 4.1 или более поздней генерирует длинные хеши только при соблюдении определенных условий: столбец пароля должен быть достаточно широким, чтобы содержать длинные значения, а old_passwords не должен иметь значения 1.

Если в таблице поле для ввода хеша имеет достаточную длину, то сервер может сохранять любые значения, как короткие, характерные для версий до 4.1, так и длинные, присущие всем современным версиям MySQL. Тогда по умолчанию функция PASSWORDS() будет генерировать лишь длинные, безопасные значения, однако, если установить значение old_passwords=1, сервер будет воспринимать данные параметра

выше по приоритету и будет генерировать короткие хеши, записывая их в длинные поля таблицы с учетными данными.

Системная переменная old_passwords предназначена для обеспечения обратной совместимости с клиентами до версии 4.1 в тех случаях, когда сервер генерировал бы длинные хеши паролей. Этот параметр не влияет на аутентификацию (клиенты 4.1 и более поздних версий все еще могут использовать учетные записи с длинными хешами паролей), но он предотвращает создание длинного хеша паролей в пользовательской таблице в результате операции смены пароля. Если бы это было разрешено, учетная запись больше не могла бы использоваться клиентами до 4.1.

```
mysql>SET @@session.old_passwords = 0;
Query OK, 0 rows affected (0.00 sec)
mysql>SELECT @@session.old_passwords,
@@global.old_passwords;
+—————————————+———————————————+
| @@session.old_passwords | @@global.old_passwords |
+—————————————+———————————————+
|                       0 |                     1 |
+—————————————+———————————————+
1 row in set (0.00 sec)
mysql>CREATE USER 'newuser'@'localhost' IDENTIFIED BY
'newpass';
Query OK, 0 rows affected (0.03 sec)
mysql>SET PASSWORD FOR 'existinguser'@'localhost' =
PASSWORD('existingpass');
Query OK, 0 rows affected (0.00 sec)
```

В MySQL 4.1 или более поздней версии возможны различные варианты использования хешей. Это зависит от того, является ли столбец пароля коротким или длинным, а если длинным, то запускается ли сервер с включенным или отключенным old_passwords.

Сценарий 1. Столбец короткого пароля в таблице пользователей:

1) в столбце «пароль» можно хранить только короткие хеши;

2) сервер использует только короткие хеши во время аутентификации клиента;

3) для подключенных клиентов операции генерации хеша пароля, включающие функцию PASSWORD() или операторы генерации пароля, используют исключительно короткие хеши. Любое изменение пароля учетной записи приводит к тому, что эта учетная запись имеет короткий хеш пароля;

4) old_passwords не имеет значения, поскольку при использовании столбца с коротким паролем сервер все равно генерирует только короткие хеши паролей.

Этот сценарий возникает, когда установка MySQL до версии 4.1 была обновлена до версии 4.1 или более поздней, но mysql_upgrade не был запущен для обновления системных таблиц в базе данных MySQL. Эта конфигурация не рекомендуется, поскольку она не позволяет использовать более безопасное хеширование паролей 4.1.

Сценарий 2. Длинный столбец паролей; сервер запущен с old_passwords=1:

1) короткие или длинные хеши могут храниться в столбце «пароль»;

2) 4.1 и более поздних версий клиенты могут проходить проверку подлинности для учетных записей, которые имеют короткие или длинные хеш-значения;

3) 4.1 клиенты могут проходить проверку подлинности только для учетных записей, которые имеют короткие хеш-значения;

4) для подключенных клиентов операции генерации хеша пароля, включающие функцию PASSWORD() или операторы генерации пароля, используют исключительно короткие хеши. Любое изменение пароля учетной записи приводит к тому, что эта учетная запись имеет короткий хеш пароля.

В этом случае вновь созданные учетные записи имеют короткие хеши паролей, поскольку old_passwords=1 предотвращает генерацию длинных хешей. Кроме того, если вы создадите учетную запись с длинным хешем до установки old_passwords в 1, изменение пароля учетной записи при old_passwords=1 приведет к тому, что учетной записи будет предоставлен короткий пароль, как следствие, потере преимуществ безопасности более длинного хеша.

Чтобы создать новую учетную запись с длинным хешем пароля или изменить пароль любой существующей учетной записи для использования длинного хеша, сначала необходимо установить значение сеанса old_passwords, равным 0, оставив глобальное значение, равным 1, как описано ранее.

В этом случае сервер имеет обновленный столбец паролей, но работает с методом хеширования паролей по умолчанию, установленным для генерации хеш-значений до версии 4.1. Это не рекомендуемая конфигурация, но может быть полезна в переходный период, когда клиенты и пароли pre-4.1 обновляются до 4.1 или более поздней версии. Когда это будет сделано, предпочтительно запустить сервер с old_passwords=0 isecure_auth=1.

Сценарий 3. Длинный столбец паролей; сервер запущен с old_passwords=0:

1) в столбце пароль могут храниться как длинные, так и короткие хеши;

2) 4.1 и более поздние версии клиенты могут аутентифицироваться с помощью учетных записей с короткими или длинными хешами;

3) клиенты до версии 4.1 могут использовать только короткие хеши;

4) для подключенных клиентов операции генерации хеша пароля, включающие функцию PASSWORD() или операторы генерации пароля, используют исключительно длинные хеши. Изменение пароля учетной записи приводит к тому, что эта учетная запись имеет длинный хеш пароля.

Однако может сложиться ситуация, в которой клиент до 4.1 изменит пароль, но авторизоваться не сможет ввиду того, что имеет право передавать только короткий хеш, а в базе уже лежит длинный. Если возникла эта проблема, вы можете изменить пароль. Например, обычно вы используете SET PASSWORD следующим образом, чтобы изменить пароль учетной записи:

```
SET PASSWORD FOR 'some_user'@'some_host' =
PASSWORD('password');
```

Чтобы изменить пароль, но создать короткий хеш, используйте вместо этого функцию OLD_PASSWORD():

```
SET PASSWORD FOR 'some_user'@'some_host' =
OLD_PASSWORD('password');
```

OLD_PASSWORD() полезен для ситуаций, в которых явно необходимо сгенерировать короткий хеш. Недостатки для каждого из предыдущих сценариев можно суммировать таким образом:

1. В сценарии 1 вы можете довольствоваться лишь короткими хешами, что не является безопасным.

2. В сценарии 2 old_passwords=1 использование любого хеша доступно, однако, если пользователь с 41-байтовым ключом

захочет сменить пароль, система преобразует его в короткий, что несомненно ведет к ослаблению системы безопасности.

3. В сценарии 3 учетные записи не могут использовать короткие хеши, за исключением случаев, когда в явном виде указано OLD_PASSWORD().

Лучший способ избежать проблем совместимости, связанных с короткими хешами паролей, – это не использовать их:

1) обновить все клиентские программы до MySQL 4.1 или более поздней версии;

2) запустить сервер с old_passwords=0;

3) сбросить пароль для любой учетной записи с коротким хешем пароля, чтобы использовать длинный хеш пароля;

4) для обеспечения дополнительной безопасности запустить сервер с параметром secure_auth=1.

## 3.2. Привилегии и производительность

Существует также немало проблем в разрезе безопасности серверов MySQL. Вполне очевидно, что при использовании сервера «как есть» имеется возможность перехватывания трафика с возможностью прослушивания всех данных в открытом виде (разве что за исключением пароля к учетной записи, как было описано выше). В процессе аутентификации пароль шифруется, однако вся остальная информация передается в виде текста и может быть прочитана любым, кто способен «прослушивать» канал передачи данных. Если соединение между клиентом и сервером проходит через ненадежную сеть и этот контур уязвим, можно использовать протоколы шифрования защищенных соединений.

Также можно использовать внутреннюю поддержку SSL MySQL, чтобы сделать соединение еще более безопасным.

Кроме того, для удобства и дополнительной защиты можно использовать SSH, который обеспечивает получение зашифрованного TCP/IP-соединения между сервером MySQL и клиентом MySQL.

Для этого подойдет абсолютно любой клиент, поддерживающий SSH-соединение.

Чтобы сделать систему MySQL безопасной, необходимо выполнить ряд условий:

1. Требовать, чтобы все учетные записи MySQL имели пароль. Клиентская программа не обязательно знает личность человека, который ее запускает. Обычно для клиентских/серверных приложений пользователь может указать любое имя пользователя для клиентской программы. Например, другой человек может использовать программу MySQL для подключения, просто вызвав ее как mysql-u other_userdb_name, если other_user не имеет пароля. Если все учетные записи имеют пароль, подключение с помощью учетной записи другого пользователя становится намного сложнее.

2. Необходимо удостовериться, что единственная учетная запись пользователя Unix с правами чтения или записи в каталогах базы данных – это учетная запись, используемая для запуска mysqld.

Запрещено запускать сервер MySQL от имени пользователя root Unix. Это крайне опасно, поскольку любой пользователь с правами доступа к файлам может заставить сервер создавать файлы от имени root (например, ~root/.bashrc).

Mysqld может (и должен) быть запущен как обычный, непривилегированный пользователь. Можно создать отдельную учетную запись Unix с именем mysql, чтобы сделать систему более безопасной. Чтобы запустить mysqld как другого пользователя Unix, необходимо указать параметр user, который задает

имя пользователя в группе [mysqld] my.cnf, где указываются параметры сервера:

```
[mysqld]
user=mysql
```

Это приводит к тому, что сервер запускается от имени указанного пользователя, независимо от того, был ли он запущен вручную или с помощью mysqld_safe или mysql.server.

Запуск mysqld как пользователя Unix, отличного от root, не означает, что нужно изменить имя пользователя root в таблице user. Имена пользователей для учетных записей MySQL не имеют ничего общего с именами пользователей для учетных записей Unix.

Привилегированная учетная запись может также использоваться для чтения любого файла, доступный для чтения или доступен пользователю Unix, от имени которого работает сервер. С этой привилегией вы можете прочитать любой файл в таблице базы данных. Этим можно злоупотребить, например, используя LOAD DATA для загрузки файла /etc/passwd в таблицу, которая затем может быть отображена с помощью SELECT.

Чтобы ограничить расположение, в котором файлы могут быть прочитаны и записаны, необходимо установить систему secure_file_priv в определенный каталог.

Запрещено также выделять привилегии неадминистративным пользователям. Выходные данные MySQL admin processlist и SHOW PROCESSLIST показывают текст любых выполняемых в данный момент операторов, поэтому любой пользователь, которому разрешено видеть список процессов сервера, может видеть операторов, выпущенных другими пользователями, такие как UPDATE user SET password= PASSWORD ('not_secure').

СУПЕРПРИВИЛЕГИЯ может использоваться для завершения клиентских подключений, изменения работы сервера путем изменения значений системных переменных и управления серверами репликации.

Необходимо ограничить использование символических ссылок на таблицы (эта возможность может быть отключена с помощью опции --skip - symbolic-links). Это особенно важно, если mysqld был запущен от имени root, потому что каждый, кто имеет доступ на запись к каталогу данных сервера, может удалить любой файл в системе!

Сохраненные программы и приложения должны быть написаны с использованием рекомендаций по безопасности.

Если используемый DNS-сервер не относится к доверенным, следует использовать IP-адреса, а не имена хостов в таблицах грантов. В любом случае, нужно быть очень осторожным при создании записей таблицы, используя значения имен хостов, содержащие подстановочные знаки.

Если имеется необходимость в ограничении количества подключений, разрешенных к одной учетной записи, это можно сделать, установив переменную max_user_connections в mysqld. Оператор GRANT также поддерживает параметры управления ресурсами для ограничения объема использования сервера, разрешенного учетной записи.

Если каталог плагинов доступен для записи сервером, пользователь может записать исполняемый код в файл в каталоге с помощью SELECT ... INTO DUMPFILE. Это можно предотвратить, сделав plugin_dir только для чтения или путем установки --secure-file-priv в надежный каталог.

*Как запустить MySQL от имени обычного пользователя*. В Windows можно запустить сервер как службу Windows, используя обычную учетную запись пользователя. В Linux же

при установке через пакетные менеджеры mysqlservermysqld должен быть запущен локальным пользователем операционной системы mysql. Запуск другим пользователем операционной системы не поддерживается сценариями инициализации, включенными в установку.

В Unix (или Linux для установок, выполняемых с использованием tar или tar.GZ packages), MySQL servermysqld может быть запущен любым пользователем. Однако по соображениям безопасности не следует запускать сервер от имени пользователя Unix root. Чтобы запустить mysqldкак обычный непривилегированный пользователь Unix user_name, необходимо выполнить следующие действия:

1. Остановить сервер, если он работает (используйте MySQL admin shutdown).

2. Изменить каталоги и файлы базы данных таким образом, чтобы имя пользователя имело право на чтение и запись файлов в них (возможно, вам придется сделать это с правами суперпользователя Unix):

```
shell>chown --R user_name/path/to/mysql/datadir
```

3. Запустить сервер от имени пользователя user_name. Другой альтернативой является запуск mysqldc правами суперпользователя Unix и использование опции --user=user_name. Mysqld запускается, затем переключается на запуск от имени пользователя Unix user_name, прежде чем принимать какие-либо соединения.

4. Чтобы запустить сервер от заданного пользователя автоматически во время запуска системы, можно указать имя пользователя, добавив опцию user в группу [mysqld] файла /etc/my.cnf или my.cnf в каталоге данных сервера:

```
[mysqld]
user=user_name
```

Если же аппаратная часть сервера не защищена и используется в недоверенном контуре, необходимо назначить пароль для MySQLroot в таблицах грантов. В противном случае любой пользователь с учетной записью на этой машине может запустить клиент mysql с параметром --user=root и выполнить любую операцию. Назначать пароли на учетные записи в MySQL в целом необходимо, однако, когда дело касается учетных записей root это становится критически важным действием.

*Проблемы безопасности при загрузке данных*. Оператор LOAD DATA может загружать файл, расположенный на хосте сервера или, если указано ключевое слово LOCAL, на хосте клиента.

Существует две потенциальные проблемы безопасности с локальной версией LOAD DATA.

Передача файла с клиентского хоста на серверный хост инициируется сервером MySQL. Теоретически можно настроить сервер таким образом, что он сообщал бы клиентской программе передавать файл по выбору сервера, а не файл, названный клиентом в инструкции LOAD DATA. Такой сервер может получить доступ к любому файлу на клиентском хосте, к которому клиентский пользователь имеет доступ на чтение. Он может фактически ответить запросом на передачу файла на любой оператор, а не просто загрузить данные локально, поэтому более фундаментальная проблема заключается в том, что клиенты не должны подключаться к ненадежным серверам.

В веб-среде, где клиенты подключаются с веб-сервера, пользователь может использовать LOAD DATA LOCAL для чтения любых файлов, к которым процесс веб-сервера имеет доступ на чтение (при условии, что пользователь может выполнить любую инструкцию на MySQL сервере). В этой среде клиент по отношению к серверу MySQL фактически является

веб-сервером, а не удаленной программой, запускаемой пользователями, которые подключаются к веб-серверу.

Чтобы избежать проблем с загрузкой данных, клиенты должны избегать использования LOCAL. Чтобы избежать подключения к ненадежным серверам, клиенты могут установить безопасное соединение и проверить удостоверение сервера, подключившись с помощью параметра --ssl - mode=VERIFY_ IDENTITY и соответствующего сертификата безопасности.

Чтобы позволить администраторам и приложениям управлять возможностью загрузки локальных данных, необходимо настроить систему таким образом:

1. На стороне сервера:

Системная переменная local_infile управляет локальными возможностями на стороне сервера. В зависимости от параметра local_infile сервер отказывается или разрешает локальную загрузку данных клиентами, включившими локальную загрузку на стороне клиента. По умолчанию local_infile включен.

Чтобы явно заставить сервер отказать или разрешить локальные операторы загрузки данных (независимо от того, как клиентские программы и библиотеки настроены во время сборки или во время выполнения), необходимо запустить mysqld с отключенным или включенным local_infile соответственно. local_infile также может быть настроен во время работы.

2. На стороне клиента:

По умолчанию клиентская библиотека в двоичных дистрибутивах MySQL компилируется с включенным ENABLED_ LOCAL_INFILE. Если MySQL компилируется из исходного кода, необходимо настроить его с отключенным или включенным ENABLED_LOCAL_INFILE в соответствии с тем, должны ли клиенты иметь возможность локальной загрузки.

Клиентские программы, использующие API C, могут управлять загрузкой данных явно, вызывая mysql_options (), чтобы отключить или включить опцию MYSQL_OPT_ LOCAL_INFILE.

Для клиента MySQL локальная загрузка данных по умолчанию отключена. Чтобы отключить или включить эту возможность, можно использовать опцию local=0 или --local[=1].

Для клиента mysqlimport локальная загрузка данных по умолчанию отключена. Чтобы отключить или включить эту возможность, можно использовать опцию --local=0 или --local[=1].

Во всех случаях, успешное использование клиентом локальной операции загрузки также требует разрешения сервера.

Если локальная возможность отключена, то на стороне сервера или клиента, клиент, пытающийся выполнить локальную инструкцию LOAD DATA, получает следующее сообщение об ошибке:

```
ERROR 1148: The used command is not allowed with this
MySQL version
```

*Рекомендации по безопасности клиентского ПО.* Приложения, которые обращаются к MySQL, не должны доверять никаким данным, введенным пользователями, не относящимся к конкретному запросу. Самый простой пример – переполнение буфера с последующей командой DROP DATABASE MySQL. Это грубый пример, но крупные утечки данных могут произойти в результате деятельности злоумышленников, использующих аналогичные методы.

Распространенной ошибкой является защита только строковых значений данных. Числовые данные также подвержены уязвимостям в запросах. Если приложение генерирует запрос, например, SELECT * FROM table WHERE ID=234, когда пользователь вводит значение 234, пользователь может ввести зна-

чение 234 или 1=1, чтобы заставить приложение генерировать запрос SELECT * FROM table WHERE ID=234 or 1=1. В результате сервер извлекает каждую строку в таблице. Это открывает доступ к каждой строке и вызывает чрезмерную нагрузку на сервер. Самый простой способ защиты от этого типа атаки заключается в использовании одинарных кавычек вокруг числовых констант: SELECT * FROM table WHERE ID='234'. Если пользователь вводит дополнительную информацию, все это становится частью строки. В числовом контексте MySQL автоматически преобразует эту строку в число и удаляет из нее все конечные нечисловые символы.

Иногда люди думают, что если база данных содержит только общедоступные данные, то она не нуждается в защите. Это заблуждение. Даже если допустимо отображать любую строку в базе данных, администратор все равно должен защищать сервер от атак типа «отказ в обслуживании» (например, тех, которые основаны на методе из предыдущего абзаца, который заставляет сервер тратить ресурсы впустую). В противном случае ваш сервер перестанет отвечать на запросы легитимных пользователей.

Подытоживая сказанное, перечислим порядок действий для обеспечения безопасности клиентского ОП:

1. Включить строгий режим SQL, чтобы указать серверу на необходимость проверки вводимых в запросах данных.

2. Попробовать ввести одинарные и двойные кавычки (' и ") во всех ваших веб-формах. Если в результате таких запросов MySQL выдает какую-либо ошибку MySQL, следует немедленно ликвидировать данную уязвимость.

3. Проверить динамические URL-адреса, добавив %22 ("), %23 (#), и %27 ( ' ) к ним.

4. Попробовать изменить типы данных в динамических URL-адресах с числовых на символьные, используя символы, показанные в предыдущих примерах. Приложение должно быть защищено от этих и подобных атак.

5. Использовать по возможности в простых полях не цифры, а символы, пробелы и специальные символы.

6. Проверить размер данных, прежде чем передавать их в MySQL.

7. Попросить ваше приложение подключиться к базе данных, используя имя пользователя, отличное от того, которое вы используете для административных целей. Не предоставляйте своим приложениям никаких привилегий доступа, которые им не нужны.

# ЗАКЛЮЧЕНИЕ

Чрезвычайно важно помнить, что обеспечение безопасности – это комплексная деятельность, направленная на минимизацию возможных угроз.

Начинать обеспечение безопасности серверов с базами данных необходимо от непосредственно физической защиты аппаратной части сервера (ведь множество атак реализуется лишь при непосредственном контакте с устройством). В дальнейшем обеспечение безопасности рассматривается как составная часть процесса администрирования сервера, так как с каждым обновлением и каждым новым пользователем возникают новые векторы атак, противостоять которым можно лишь при постоянном контроле уязвимостей как самого серверного ПО, так и настроек конкретного сервера.

Учебное пособие

**Орехов Павел Васильевич**,
кандидат технических наук

**Большунов Александр Геннадьевич**

**Галиев Денис Вадимович**

**Тычкова Анастасия Олеговна**

# БАЗЫ ДАННЫХ

Редактор *Чеботарева С. О.*
Корректор *Лосева О. С.*
Компьютерная верстка *Лосева О. С.*

Московский университет МВД России имени В.Я. Кикотя
117437, г. Москва, ул. Академика Волгина, д. 12

| | | | |
|---|---|---|---|
| Подписано в печать 23.12.2020 | Формат 60×84 1/16 | Тираж | 60 экз. |
| Заказ № 260 | Цена договорная | Объем | 3,76 уч.-изд. л.<br>5,99 усл. печ. л. |