Лекция: Оптимизация

Оптимизация — это процесс тонкой настройки системы, направленный на повышение скорости ее работы или сокращение объема используемой памяти. В первой части лекции объясняется, когда и как нужно оптимизировать базы данных. Бинарные дистрибутивы, доступные на Web-узле MySQL, оптимизированы для общего применения. Чтобы адаптировать программу к каким-то специфическим требованиям, ее необходимо перекомпилировать. Об этом и пойдет речь в конце лекции.

Предварительные действия

Перед началом проектирования базы данных поставьте себе задачу добиться максимальной ясности спецификации, даже если на это уйдет больше времени. Помните о том, что услуги программистов стоят дорого, особенно если им приходится разбираться с малопонятным проектом. Простое решение обычно является наилучшим.

Перенося базу данных в производственную среду, позаботьтесь о том, чтобы производительность базы данных была адекватной. Если к проекту прилагается формальная спецификация требований, просмотрите, указываются ли в ней какие-либо ограничения производительности. Для приложений, работающих с базами данных, нередко задается максимальное время выполнения запросов. Продолжительность времени между вводом инструкции и получением результатов запроса зависит от многих факторов. Необходимо заранее учесть те факторы, которые впоследствии нельзя будет контролировать.

Если обнаруживается, что система требует оптимизации, в первую очередь подумайте об обновлении аппаратной части. Это может оказаться самым дешевым вариантом. В 1965 г. Гордон Мур (Gordon Moore) установил, что вычислительные мощности удваиваются каждые 18 месяцев. Данное правило называют законом Мура. Но, несмотря на столь стремительный рост производительности, удельная стоимость вычислительных средств неуклонно снижается. Например, центральные процессоры за полтора года удвоят тактовую частоту, хотя стоить будут так же, как и полтора года назад. Таким образом, обновление компьютера может обойтись дешевле, чем оптимизация проекта. Во вторую очередь стоит подумать об обновлении программного обеспечения. Основной программный компонент— это операционная система. Известно, что Linux и BSD UNIX позволяют повысить производительность старых компьютеров, превосходя в этом отношении коммерческие операционные системы, такие, как Windows, особенно если бессбойная работа сервера очень важна.

Обновляется и сама программа MySQL. Когда появится новая версия, ее производительность будет повышена в сравнении с текущей. Но и в текущую версию регулярно вносят мелкие исправления, так что желательно идти в ногу со временем. Основная причина оптимизации — желание сэкономить деньги (оставим в стороне личное удовлетворение и другие причины). Не забывайте об этом в своих попытках повысить производительность программы. Нет смысла затрачивать на оптимизацию больше денег, чем она способна принести. Стоит потрудиться над такой программой, с которой работает множество людей, особенно если это коммерческое приложение. Что касается программ с открытыми кодами, то важность оптимизации здесь трудно определить. Лично я рассматриваю работу над такими проектами как хобби.

Чтобы процесс оптимизации был максимально эффективным, сосредоточьте усилия на самой медленной части программы, улучшение которой обеспечит наибольшую отдачу. Обычно пытаются найти более быстрые альтернативы применяемым алгоритмам. В вычислительной технике относительная эффективность алгоритма записывается в нотации "большого О". Например, запись O(n) означает, что время выполнения алгоритма пропорционально числу обрабатываемых элементов n. Алгоритм типа O(n) является очень медленным. Проанализируйте используемые в программе алгоритмы и подумайте, что можно сделать для их улучшения.

Тесты производительности

Прежде чем приступать к оптимизации, нужно вооружиться средствами измерения производительности. Предусмотрительные разработчики MySQL написали группу Perl сценариев, предназначенных для тестирования производительности MySQL и других СУБД. Эти сценарии расположены в каталоге sql-bench исходного дистрибутива. В подкаталоге Results находятся результаты множества тестов существующих систем, которые можно сравнить с собственными оценками.

В сценариях используется демонстрационная база данных, в которой выполняется восемь различных тестов. Эта база данных называется test и инсталлируется вместе с MySQL. Сценарий run_all_tests запускает все тесты последовательно. При наличии опции -log результаты работы сценария будут сохранены в каталоге output для последующего просмотра. Ниже приведена команда, запускающая тесты из каталога sqlbench.

#./run-all-tests –user=leon –password=secret --leon

Для работы этого сценария необходимо наличие в системе интерпретатора Perl и модуля DBI. Для экспериментов я использую старый компьютер Pentium с частотой 100 МГц, работающий под управлением RedHat Linux. Несмотря на слабую вычислительную мощность, программа MySQL демонстрирует на нем вполне приемлемую производительность. Кроме того, ограниченные возможности системы позволяют быстро выявлять неэффективные программные решения. Результаты тестов, полученные на этом компьютере, показаны в листинге1. Несложно убедиться, что моя система работает примерно в 10 раз медленнее, чем самая медленная из систем, результаты тестов которых имеются в каталоге Results. Если бы такую производительность продемонстрировал рабочий сервер, нужно было бы немедленно обновить его аппаратную часть.

```
================================================================
Operation                                |                1|
                                         |          mysql-1|
----------------------------------------------------------------
Results per test in seconds:             |                 |
----------------------------------------------------------------
ATIS                                     |           549.00|
Alter-table                              |          4836.00|
Big-tables                               |           270.00|
Connect                                  |           607.00|
Create                                   |          2027.00|
Insert                                   |        +88846.00|
Select                                   |        +18339.00|
wisconsin                                |           135.00|
----------------------------------------------------------------
```

Листинг 1.

```
------------------------------------------------------------
alter table add (992)                    |         2670.00|
alter table drop (496)                   |         2066.00|
connect (10000)                          |           95.00|
connect select 1 row (10000)             |          115.00|
connect select simple (10000)            |          106.00|
count (100)                              |          364.00|
count distinct (1000)                    |         +779.00|
count distinct 2 (1000)                  |         +605.00|
count distinct big (120)                 |         1185.00|
count distinct group (1000)              |         +726.00|
count distinct group on key (1000)       |         +716.00|
count distinct group on key parts (1)    |         +708.00|
count distinct key prefix (1000)         |         +671.00|
count group on key parts (1000)          |          706.00|
count on key (50100)                     |        +7005.00|
create + drop (10000)                    |          109.00|
create MANY tables (10000)               |         1196.00|
create index (8)                         |           52.00|
create key drop (10000)                  |          115.00|
create table (31)                        |            0.00|
delete all (12)                          |          219.00|
delete all many keys (1)                 |         8960.00|
delete big (1)                           |            5.00|
delete big many keys (128)               |         8956.00|
delete key (10000)                       |          145.00|
drop index (8)                           |           47.00|
```

```
drop table (28)                          |            0.00|
drop table when MANY tables (10000)      |          105.00|
insert (350768)                          |         1044.00|
insert duplicate (100000)                |          211.00|
insert (100000)                          |        17849.00|
insert many fields (2000)                |           97.00|
insert select 1 key (1)                  |          102.00|
insert select 2 keys (1)                 |          125.00|
min max (60)                             |          301.00|
min max on key (85000)                   |        +2627.00|
multiple value insert (100000)           |          102.00|
order by big (10)                        |          560.00|
order by big key (10)                    |          503.00|
order by big key2 (10)                   |          479.00|
order by big key desc (10)               |          531.00|
order by big key diff (10)               |          586.00|
order by big key prefix (10)             |          487.00|
order by key2 diff (500)                 |           57.00|
order by key prefix (500)                |           31.00|
outer join (10)                          |          989.00|
outer join found (10)                    |          918.00|
outer join not found (500)               |       +36000.00|
outer join on key (10)                   |          810.00|
select 1 row (10000)                     |           17.00|
select 2 rows (10000)                    |           23.00|
select big (10080)                       |          700.00|
select column + column (10000)           |           24.00|
select diff key (500)                    |         +1470.00|
select distinct (800)                    |          145.00|
select group (2925)                      |          +896.00|
select group when MANY tables (10000)    |          502.00|
select join (100)                        |           25.00|
select key (200000)                      |         +1123.00|
select key2 (200000)                     |         +1213.00|
select key2 return key (200000)          |         +1174.00|
select key2 return prim (200000)         |         +1197.00|
select key prefix (200000)               |         +1245.00|
select key prefix join (100)             |          197.00|
select key return key (200000)           |         +1106.00|
select key many fields (2000)            |          172.00|
select query cache (10000)               |         1075.00|
select query cache2 (10000)              |         1075.00|
select range (410)                       |         +1812.00|
select range key2 (25010)                |          185.00|
select range prefix (25010)              |          197.00|
select simple (10000)                    |           11.00|
select simple join (500)                 |           20.00|
update big (10)                          |          440.00|
update of key (40000)                    |          627.00|
update of key big (501)                  |          351.00|
update of primary key many keys(256)     |         3290.00|
update with key (300000)                 |         1011.00|
update with key prefix (100000)          |          290.00|
wisc benchmark (114)                     |           44.00|
---------------------------------------------------------
TOTALS                                   |      +124540.00|
---------------------------------------------------------
```

Листинг 2.

Сценарий compare results суммирует и сравнивает результаты тестов. В листинге 1 приведен лишь

один набор результатов. В первом блоке чисел указано время выполнения каждого из восьми тестов в секундах. Во втором блоке отображается статистика отдельных операций по всем тестам. Числа со знаком "плюс" — это приблизительные оценки, полученные для тестов, время выполнения которых превысило максимум.

Результаты тестов, предоставляемые разработчиками MySQL, можно использовать для выбора аппаратной платформы и операционной системы. На Web узле MySQL (http://www.mysql.com) постоянно публикуются обновляемые результаты и графики сравнения показателей MySQL с показателями других СУБД, работающих на идентичном оборудовании. Приводятся также данные, касающиеся работы MySQL на разных платформах.

Конечно, все эти тесты отражают лишь относительную производительность сервера. С их помощью можно узнать, насколько возрастет скорость его работы при изменении тех или иных настроек, но тесты не могут помочь в оптимизации базы данных. Для оценки производительности запросов необходимо воспользоваться инструкцией EXPLAIN. Эта инструкция, помимо всего прочего, сообщает о том, сколько записей будет прочитано при выполнении заданной инструкции SELECT. Каждая строка результатов соответствует одной исходной таблице, а порядок строк совпадает с порядком обращения к таблицам. Сообщаемое число записей может быть приблизительным, но погрешность очень мала. Произведение счетчиков записей является грубым критерием производительности запроса. Чем меньше это произведение, тем быстрее выполняется запрос.

Представим себе, к примеру, объединение таблицы, содержащей 15000 слов, с таблицей, содержащей 100000 слов. В худшем случае программе MySQL придется просмотреть все записи обеих таблиц. Сначала выбирается первая запись первой таблицы, а затем начинается просмотр записей второй таблицы до тех пор, пока не будет найдено совпадение. Умножив 15000 на 100000, получим 1,5 миллиарда операций чтения. На практике это число оказывается немного меньшим, но и его достаточно, чтобы получить представление о скорости запроса. Далее будет рассказываться о том, как с помощью индексов уменьшить количество записей, читаемых в процессе объединения таблиц.

С помощью журнала медленных запросов можно легко найти наименее эффективные запросы. В дистрибутив MySQL входит сценарий mysqldumpslow предназначенный для упорядочения записей этого журнала по указанному в них времени выполнения запроса.

Оптимизация проекта

Давайте вспомним некоторые теоретические аспекты. Нормализация — это такой метод оптимизации базы данных, при котором избыточность хранящейся в ней информации оказывается минимальной. Следовательно, уменьшается время, затрачиваемое приложением на поддержание целостности базы данных. Нормализация достигается за счет повышения объема работы, выполняемой сервером, так как увеличивается число таблиц и серверу приходится чаще создавать их объединения. В процессе денормализации в базу данных вносят некоторую избыточность, чтобы сократить объем работы по извлечению информации.

Наиболее эффективный тип денормализации включает создание итоговых данных. Под этим может подразумеваться добавление к таблице столбца, хранящего результаты вычислений по другим столбцам. Например, если в таблице накапливаются данные о прохождении грузов, то в ней будут столбцы с указанием времени прибытия и отбытия груза. Чтобы не вычислять каждый раз время стоянки, можно посчитать его один раз и занести результат в отдельный столбец. Управлять подобной избыточностью несложно.

Иногда создают не просто итоговые столбцы, а целые таблицы. Например, можно сохранять результаты ключевых запросов в таблице, которая обновляется раз в день. Это избавит сервер от необходимости все время выполнять одни и те же трудоемкие запросы, хотя и повысит риск получения пользователями неактуальных данных. Если таблицы содержат часто изменяемую информацию, лучше делать их резидентными. Такие таблицы хранятся в памяти и уничтожаются при перезагрузке сервера. Приложение должно быть готово к возможному отсутствию таблицы и должно уметь воссоздавать ее в случае необходимости. Хороший пример — Web-приложение, хранящее параметры сеанса в базе данных.

Реляционные базы данных хорошо работают с типизированными значениями фиксированного размера. В MySQL поддерживаются типы переменной длины, например BLOB и TEXT, но управлять ими сложнее. Такого рода информацию лучше хранить в файлах, а в базе данных достаточно запоминать путевые имена этих файлов в столбцах типа CHAR. Если база данных используется в Web-приложениях, помните о том, что у Web-сервера есть кеш-буфер загружаемых файлов изображений и аудиоклипов, поэтому он будет работать с такими файлами быстрее, чем MySQL.

Еще одна причина избегать столбцов подобного типа заключается в появлении записей переменной длины со всеми вытекающими отсюда последствиями. При внесении изменений такая таблица становится дефрагментированной что приводит к замедлению доступа к ней. Для извлечения динамической строки может потребоваться несколько операций чтения, что также не способствует повышению производительности.

Иногда возникает проблема — определить, когда стоит использовать столбцы типа CHAR, а когда VARCHAR. Если в таблице есть столбцы типа BLOB или TEXT, то предпочтение отдается типу VARCHAR, потому что все записи таблицы будут динамическими. То же самое справедливо для случая, когда средняя размерность значений столбца не превышает половины его размерности. Например, столбец типа VARCHAR (80) средняя размерность которого равна 10 символам, определен правильно. Если же средняя размерность превышает 40 символов, нужно поменять тип столбца на CHAR (80) Данное правило направлено на оптимизацию скорости работы с таблицами. Когда более важным фактором является экономия дискового пространства, то в большинстве случаев следует пользоваться типом VARCHAR. Для таблиц MyISAM поддерживается опция DELAY_KEY_WRITE. Она заставляет программу хранить изменения табличных индексов в памяти, пока таблица не будет закрыта. Это сокращает время записи на диск измененных табличных данных, но также повышает риск повреждения таблицы в случае сбоя сервера. Если используется данная опция, то при каждом перезапуске сервера необходимо проверять таблицы на предмет повреждений.

Процедура analyse() представляет собой удобное средство проверки таблицы после вставки данных, так как она определяет диапазон значений каждого столбца в полученном наборе записей. Ее нужно применять в инструкции SELECT, которая извлекает все записи отдельной таблицы. На основании анализа таблицы процедура analyse() предложит оптимальный тип данных для каждого столбца.

В некоторых случаях процедура analyse() сообщает о том, что вместо типа CHAR должен применяться тип ENUM. Это происходит, когда столбец содержит небольшое число повторяющихся значений. Столбец типа ENUM занимает гораздо меньше места, поскольку в действительности он хранит лишь номера элементов перечисления. Многие типы данных допускают регулирование своей размерности. Например, в столбце типа CHAR может храниться столько уникальных значений, что приводить его к типу ENUM нет никакого смысла, и все равно формальная размерность оказывается избыточной. То же самое касается типа INT, у которого существуют более "короткие" эквиваленты: MEDIUMINT, SMALLINT и TINYINT. Но не забудьте учесть будущее пополнение таблицы. Например, если в таблице 16000 записей, то для первичного ключа вполне подойдет тип SMALLINT. Если же предполагается, что в таблице будет более 65535 записей, следует остановиться на типе INT.

Обратите внимание на столбцы, в которых не могут присутствовать значения NULL. Для экономии места такие столбцы нужно объявлять со спецификатором NOT NULL. Числовые столбцы, в которых не могут храниться отрицательные числа, должны иметь спецификатор UNSIGNED.

Оптимизация приложений

Подключение к базе данных MySQL происходит относительно быстро в сравнении с другими СУБД, но это время можно еще уменьшить за счет кэширования соединений. Требуется лишь прикладная среда, позволяющая хранить идентификаторы соединений в памяти во время работы сервера. Например, модуль PHP непрерывно работает на Web-сервере. Он поддерживает функцию mysql pconnect(), которая создает постоянные соединения. Получив запрос на подключение к серверу, модуль PHP попытается использовать существующее соединение, если это возможно. В протоколах JDBC и ODBC тоже применяется технология кэширования соединений. Она особенно удобна, когда приложение создает большое число соединений за короткий промежуток времени. К примеру, если приложение вставляет данные в таблицу, можно предварительно помещать данные в буфер, с тем чтобы позднее занести их в таблицу в пакетном режиме. В этом случае лучше сразу же заблокировать таблицу, чтобы не пришлось многократно обновлять табличные индексы.

Приложение может дотировать информацию, извлекаемую из базы данных. Это выгодно, если данные меняются нечасто. Когда изменение данных все же происходит, приложение запрашивает принудительную очистку буфера. Предположим, что в Internet магазине имеется каталог продаваемых товаров. Этот каталог пополняется или обновляется раз в неделю, а то и меньше. Когда приложение отправляет клиенту html-страницу с описанием товара, оно вполне может взять информацию из кэша.Если администратор захочет воспользоваться приложением для обновления цены товара, он должен будет очистить кэш. То же самое применимо и к программным блокам. Если нужно узнать название, цену и категорию товара, введите один запрос и сохраните полученные значения в

программных переменных. Основная работа по выборке данных заключается в поиске нужной записи. Не имеет особого значения, 100 или 1000 байтов извлекаются из нее.

Оптимизация запросов

Незаметно для пользователей программа MySQL оптимизирует предложения WHERE инструкции SELECT. Обычно не нужно заботиться о том, сколько скобок указано в выражении или каков порядок таблиц в объединении. Вместо этого сосредоточьтесь на индексах. Они позволяют ускорить операции выборки данных за счет замедления операций записи. Конечно, индексы занимают дополнительное место на диске, но они незаменимы с точки зрения эффективной организации таблиц.

Когда программа MySQL извлекает данные из таблицы, ей достаточно просмотреть один индексный столбец, чтобы найти нужные записи и не сканировать всю таблицу. Если к объединенной таблице применимы два индекса, программа выбирает из них тот который позволит прочесть меньшее число записей.

Разрешается создавать индекс, охватывающий несколько столбцов. Программа MySQL может работать с частями индекса, но они должны просматриваться строго слева направо. Например, если индекс включает столбцы имени и фамилии, то при обращении к первому столбцу индекс будет использован, а ко второму — нет (при условии, что перед этим не было обращения к первому столбцу). Это правило применимо и к символам индексируемого столбца, содержащего текстовые данные (тип CHAR, VARCHAR или BLOB). Когда в предложении WHERE присутствует оператор LIKE, индекс задействуется лишь в том случае, если шаблон сравнения содержит все литеральные символы слева, а метасимволы — справа. Так, шаблон ‘abc %’ разрешает использование индекса, а шаблон ‘abc % xyz’ — нет.

В листинге 2 приведены инструкции, создающие две таблицы. Таблица word будет содержать 14346 записей, а таблица dictionary — 104237. В первую таблицу слова заносятся пользователями, а вторая таблица содержит список известных программе слов. Пользователи часто вводят несуществующие слова. Запрос, анализируемый в листинге 3, предназначен для выяснения количества распознанных слов. Условию отбора соответствуют 911 записей.

```
mysql> EXPLAIN SELECT word.word, dictionary.word
-> FROM word LEFT JOIN dictionary
-> ON word.word=dictionary.word
-> WHERE word.class = '_VERBO' \G
*************************** 1.row ***************************
table: word
type: ALL
possible_keys: NULL
key: NULL
key_len: NULL
ref: NULL
rows: 14346
Extra: where used
table: dictionary
type: ALL
possible_keys: NULL
key: NULL
key_len: NULL
```

Листинг 3.

В запросе участвуют три столбца: столбцы word и class таблицы word и столбец word таблицы dictionary. Известно, что тестовому условию отбора соответствуют 911 записей таблицы word, поэтому наша задача состоит в том, чтобы сократить диапазон сканирования первой таблицы до соответствующего уровня. Для этого необходимо создать индекс по столбцу class. Сначала я планировал включить в индекс только упомянутый столбец, но потом подумал о других запросах, которые приходится направлять таким таблицам. Я, например, часто создаю отчет, в который включается все содержимое таблицы, отсортированное сначала по классам, а затем — по словам. Разумнее будет включить в индекс сразу два столбца (листинг 4).

```
ALTER TABLE word
ADD INDEX (class, word)
```

Листинг 4.

Теперь инструкция EXPLAIN выдает другие результаты (листинг 5). В поле Key len сообщается о том, что индекс охватывает 16 символов столбца class. По оценке программы MySQL, ей придется просмотреть 1517 записей, хотя мы знаем, что их всего 911

```
Mysql > EXPLAIN SELECT word. word, dictionary. Word
->      FROM word LEFT JOIN dictionary
->      ON word. Word = dictionary. Word
->      WHERE word. Class = "_VERBO" \G

      table: word
       type: ref
possible keys: class
          key: class
      key len: 16
     ref: const
    rows: 1517
   Extra: where used; Using index

   table: dictionary
    type: ALL
possible keys: NULL
     key: NULL
 key len: NULL
     ref: NULL
    rows: 104237
       Extra:
2 rows in set (0.00 sec)
```

Листинг 5.

Итак, появление индекса привело к сокращению диапазона сканирования в 15 раз, но инструкция все же вынуждена просматривать 45 миллионов записей. Осталось еще учесть столбцы word в обеих таблицах. Разберемся сначала с таблицей word. В процессе объединения таблиц программа MySQL использует не более одного индекса от каждой таблицы. Если появляются варианты, то выбирается индекс с более узким диапазоном. Созданный нами индекс уже охватывает столбец word, к тому же, как видно из листинга 5, диапазон поиска существенно сузился. Теперь перейдем к таблице dictionary. Пока что инструкция SELECT вынуждена сканировать ее целиком. Добавление индекса к столбцу word позволит программе сразу же находить нужную запись (листинг 6).

```
ALTER TABLE dictionary
ADD INDEX (word)
```

Листинг 16.

Эффект этого действия продемонстрирован в листинге 7. Как видите, количество просматриваемых записей таблицы dictionary сократилось до одной!

```
Mysql > EXPLAIN SELECT word. word, dictionary. Word
->      FROM word LEFT JOIN dictionary
->      ON word. Word = dictionary. Word
->      WHERE word. Class = "_VERBO" \G
********************** 1. row ********************
      table: word
       type: ref
possible keys: class
          key: class
      key len: 16
     ref: const
    rows: 1517
```

```
   Extra: where used; Using index

   table: dictionary
    type: ref
possible keys: word
     key: word
 key len: 64
     ref: word. word
    rows: 1
        Extra: Using index
2 rows in set (0.26 sec)
```

Листинг 7.

Оптимизация инструкций

От оптимизации больше всего выигрывают запросы на выборку, но существуют также методики повышения эффективности других инструкций, в частности INSERT. Можно избежать затрат времени на анализ инструкции, если воспользоваться преимуществами значений по умолчанию. Вместо того чтобы указывать значения всех столбцов, задайте лишь те из них, которые отличаются от стандартных установок, а остальное пусть сделает MySQL. Сказанное иллюстрирует листинг 8, в котором показаны определение таблицы и инструкция INSERT

```
CREATE TABLE address {
   ID int (11) NOT NULL AUTO INCREMENT,
   Name Prefix CHAR (16) default NULL,
   Name First CHAR (32) default NULL,
Name Middle CHAR (32) default NULL,
Name Last CHAR (64) NOT NULL default,
Name Suffix CHAR (16) default NULL,
Company CHAR (64) default NULL,
Street1 CHAR (64) default NULL,
Street2 CHAR (64) default NULL,
Street3 CHAR (64) default NULL,
City CHAR NOT NULL default ‘’,
State Prov CHAR (64) NOT NULL default,
Postal Code CHAR (16) NOT NULL default,
Country Code CHAR (2) default NULL,
Phone1 CHAR (32) default NULL,
Phone2 CHAR (32) default NULL,
Fax CHAR (32) default NULL,
Email CHAR (64) NOT NULL default ‘’,
PRIMARY KEY (ID)
};

INSERT INTO address (Name First, Name Last)
   VALUES (‘Leon’, ‘Atkinson’);
```

Листинг 8.

По умолчанию операции записи имеют приоритет над операциями чтения, но программа MySQL не прервет выполнение инструкции SELECT, если в очереди вдруг появится инструкция INSERT. Последняя окажется заблокированной до тех пор, пока инструкция SELECT не завершится. У инструкции INSERT есть также специальный флаг DELAYED, при наличии которого инструкция помещается в очередь без блокирования клиентского приложения, что повышает его оперативность.

Если есть несколько записей, предназначенных для вставки в таблицу, воспользуйтесь многострочной инструкцией INSERT. Еще быстрее работает инструкция LOAD DATA INFILE. Для полной очистки таблицы лучше вызывать инструкцию TRUNCATE TABLE, а не DELETE. В этом случае программа MySQL удалит и снова создаст табличный файл, вместо того чтобы удалять записи одна за другой.

Если в состав инструкции входит сложное выражение, замените его пользовательской функцией. Естественно, это имеет смысл делать только тогда, когда предполагается многократно вызывать

инструкцию. О создании собственных функций рассказывается в лекции "Расширение возможностей MySQL".

Обслуживание таблиц

Можно ускорить выполнение запросов, если хранить таблицы и индексы в упорядоченном виде. Инструкция OPTIMIZE TABLE улучшает таблицу тремя способами. Во-первых, она устраняет пустые промежутки, оставшиеся после удаления записей. Для таблиц MyISAM это означает возможность одновременного выполнения инструкций INSERT и SELECT. Во-вторых, она соединяет распределенные фрагменты таблиц с динамическими записями. И, наконец, она сортирует индексы.

Инструкция ALTER TABLE позволяет отсортировать записи таблицы. Это тоже способствует ускорению некоторых запросов, хотя и не устраняет потребность в индексах. Если таблица меняется редко, а дисковое пространство ограничено, имеет смысл сжать таблицу с помощью утилиты myismpack. После этого таблица будет доступна только для чтения. Ее индексы необходимо перестроить, вызвав утилиту myismchk. Данная методика позволяет уменьшить размер таблицы на 40-70%, в зависимости от формата ее содержимого. В листинге .9 показан процесс сжатия таблицы, содержащей названия штатов США.

```
# myisampack. exe state
Compressing state. MYD: (50 records)
- Calculating statistic
- Compressing file
32.42%
Remember to run myisamchk – rq on compressing tables
# myisamchk – rq state
- check key delete - chain
- check records delete – chain
- recovering (with sort) MyISAM – table 'state.MYI'
Data records: 50
- Fixing index 1
- Fixing index 2
```

Листинг 9.

Настройка конфигурации сервера

Когда речь заходит об объеме оперативной памяти сервера, совет всегда один: чем больше — тем лучше. Увеличение объема памяти способствует ускорению работы программы MySQL, так как в оперативной памяти она хранит свои временные таблицы и буферы записей. В подкаталоге support files дистрибутива содержатся образцы конфигурационных файлов с различными вариантами настроек, касающихся использования памяти. Выберите тот вариант, который соответствует исходным параметрам сервера. Поработав с сервером какое-то время, можно будет оценить, какие из настроек требуют корректировки.

В листинге 10 показана конфигурация сервера, располагающего как минимум 1 Гбайт ОЗУ, четырьмя жесткими дисками и четырьмя центральными процессорами. Обратите внимание на важность индексного буфера. В данной конфигурации предполагается, что сервер хранит табличные данные на первом диске, а временные файлы — на втором. Таблицы InnoDB находятся на третьем диске, а журналы InnoDB — на четвертом.

```
[mysqld]
set – variable = key buffer = 384M
set – variable = max allowed packet = 1M
set – variable = table cache = 512
set – variable = sort buffer = 2M
set – variable = record buffer = 2M
set – variable = thread cache size = 8
set – variable = thread concurrency = 8
set – variable = myisam sort buffer size = 64M
```

```
log - bin      =
server - id    = 1
tmpdir         = /disk2/tmp/

# Таблицы BDB
set - variable = bdb cache size = 384M
set - variable = bdb max lock = 100000

# Таблицы InnoDB
innodb data home dir = /disk3/
innodb log group home dir = /gisk4/
innodb log arch dir = /disk4/
innodb data file path = ibdata1: 250M; ibdata2: 500M; ibdata3: 1000M
set - variable = innodb mirrored log groups = 1
set - variable = innodb log files in groups = 3
set - variable = innodb log files size = 5M
set - variable = innodb log buffer size = 8M
innodb flush log at trx commit = 1
innodb log archive = 0
set - variable = innodb buffer pool size = 16M
set - variable = innodb additional mem pool size = 2M
set - variable = innodb file io threads = 4
set - variable = innodb lock wait timeout = 50
```

Когда сервер проработает какое-то время, выполните инструкцию SHOW STATUS, чтобы узнать его производительность. Сравните значения показателей Key reads и Key_read_requests. Их соотношение будет очень низким, если программа MySQL часто пользуется индексным буфером. В случае необходимости попытайтесь повысить размер буфера.

Проследите изменение показателя Open tables, сравнивая его со значением серверной переменной table cache, которое можно узнать с помощью инструкции SHOW VARIABLES. Когда табличный буфер заполняется, программа MySQL вынуждена закрывать одни таблицы, чтобы открывать другие. Показатель Opened tables отражает число таблиц, открывавшихся с момента запуска сервера. Сравните его с общим числом запросов (показатель Questions). Чем больше будет размер табличного буфера, тем реже придется открывать и закрывать таблицы.

Серверная переменная thread_cache_size задает размер кэша потоков. Как правило, на каждый процессор должно приходиться два потока. Сравните показатели Threads_created и Connections, чтобы определить, как часто серверу приходилось повторно использовать потоки.

Просмотрите еще раз список переменных демона mysqld. Есть много разных буферов и кэшей, увеличение размера которых способно повысить производительность сервера. После изменения конфигурации обязательно проведите повторные замеры.

Перекомпиляция MySQL

Команда разработчиков MySQL прилагает огромные усилия для оптимизации исполняемых файлов программы. Лучше всего пользоваться бинарными дистрибутивами, которые доступны на Web-узле MySQL. Вряд ли вам удастся получить более качественный исполняемый файл. Например, в дистрибутивы Linux зачастую включаются нестабильные версии компиляторов и библиотек. Разработчики MySQL всегда применяют самые стабильные версии в сочетании с оптимальными опциями компиляции.

Необходимость в компиляции возникает, когда для данной платформы невозможно найти

скомпилированную версию программы, хотя эта ситуация маловероятна. Еще одна причина — желание поэкспериментировать с различными библиотеками. Но подобными экспериментами не стоит слишком увлекаться, так как в результате можно получить нестабильно работающий исполняемый файл.

На Web-узле MySQL приведена информация о том, как компилировать программу на различных платформах. Не поленитесь просмотреть рекомендации специалистов, поскольку здесь есть много "подводных камней", особенно в случае старых операционных систем.

Перед началом компиляции убедитесь в наличии утилит qzip и qnutar Они необходимы для извлечения файлов из tar-архива. Учтите, что версия утилиты tar для Solaris содержит ошибку, которая не позволяет распаковывать некоторые архивы, поэтому желательно иметь GNU-версию утилиты.

Нужен также компилятор языка C++. Вполне подойдет eqcs. Не забудьте и об утилите make.

Те, кто имеют опыт компиляции программ с открытыми кодами, должны быть знакомы со сценариями конфигурации, создаваемыми утилитой autoconf. Саму ее запускать не нужно. Файл Makefile создается сценарием configure. В листинге 11 показан вызов этого сценария с установками, которые рекомендованы разработчиками MySQL. Сценарий configure должен запускаться из каталога, содержащего исходные коды программы.

```
CFLAGS = "-03" \
CXX = gcc \
CXXFLAGS = "-03 –felide–constructors–fno–exceptions–fno–rtti" \
./configure –prefix = /usr/local/mysql \
--enable-assembler \\
--with-mysql-ldflags=-all-atatic
```

Листинг11.

Параметры сценария configure можно получить, вызвав сценарий с опцией –help. Если нужно включить поддержку таблиц Berkeley DB или InnoDB, не забудьте указать соответствующие опции. В исходные дистрибутивы MySQL входят все необходимые для этого файлы, по этому путь к библиотекам Berkeley DB задается только в том случае, когда требуется использовать их альтернативные версии.

Лекция: Каталог данных MySQL

Концептуальные принципы построения большинства систем управления реляционными базами данных одинаковы: все они состоят из набора баз данных, каждая из которых, в свою очередь, включает набор таблиц. Однако каждая система по-своему организует управляемые данные. Не является исключением в этом отношении и MySQL.

По умолчанию вся информация, управляемая сервером mysqld, содержится в так называемом каталоге данных MySQL (MySQL data directory). В нем хранятся все базы данных и файлы состояния с информацией о функционировании сервера. Пользователь, выполняющий функции администратора MySQL, должен знать структуру этого каталога и уметь использовать его в своей повседневной работе.

В этой лекции даются исчерпывающие ответы на следующие вопросы.

- Как определить месторасположение каталога данных. Это необходимо для эффективного управления его содержимым.
- Как организуется и обеспечивается доступ к базам данных и таблицам сервера. Эта информация необходима для создания расписания операций превентивной поддержки и восстановления поврежденных таблиц после сбоя.
- Где размещаются сгенерированные сервером файлы состояния и что они содержат. Эти файлы содержат информацию о работе сервера, которая может быть очень полезна для устранения всевозможных проблем.
- Как изменить месторасположение каталога данных по умолчанию или отдельных баз данных.

Знание этих вопросов важно для управления размещением дискового пространства системы. Это может быть необходимо для распределения данных по физическим дискам или перемещения данных в файловых системах с целью освобождения пространства на одном из дисков. Данная информация пригодится и при планировании размещения новых баз данных.

Значительную пользу от чтения этой лекции получат даже те пользователи, которые не занимаются

администрированием MySQL. Дополнительные знания о принципах работы сервера никогда не будут лишними.

Размещение каталога данных

По умолчанию местоположение для каталога данных устанавливается при компиляции сервера. Обычно при инсталляции с исходной дистрибуции устанавливается каталог /usr/local/var, при инсталляции из двоичной дистрибуции — /usr/local/mysql/data, а при инсталляции из файла RPM -- /var/lib/mysql

Размещение каталога данных можно задать и явным образом при запуске сервера. Для этих целей применяется опция --datadir= /path/to/dir. Она оказывается весьма кстати, если каталог данных необходимо разместить в месте, отличном от того, которое указывается по умолчанию.

Администратор MySQL обязательно должен знать, где находится каталог данных. При запуске нескольких серверов следует записать местоположение всех каталогов данных. Если же размещение каталога неизвестно (например, из-за того, что предыдущий администратор плохо вел свои записи), его можно определить несколькими методами.

- Воспользоваться командой mysqladmin variables для получения пути к

каталогу данных непосредственно с сервера. На компьютере, работающем под управлением ОС UNIX, результат ее ввода будет выглядеть примерно так:

```
% mysqladmin variables
+------------------+-----------------+
| Variable name    | Value           |
+------------------+-----------------+
| back log         | 5               |
| connect timeout  | 5               |
| basedir          | /var/local/     |
| datadir          | /usr/local/var/ |
...
```

Из приведенных выше результатов видно, что каталог данных размещается в каталоге /usr/local/var/ сервера.

На компьютере, работающем под управлением ОС Windows, результаты ввода этой же команды будут выглядеть следующим образом.

```
% mysqladmin variables
+------------------+-----------------+
| Variable name    | Value           |
+------------------+-----------------+
| back log         | 5               |
| connect timeout  | 5               |
| basedir          | c: \mysql\      |
| datadir          | c: \mysql\data\ |
...
```

Если на компьютере запущено несколько серверов, каждый из них использует свой порт TCP/IP и разъем. Чтобы получить информацию о каталоге данных от каждого сервера, достаточно подключиться с помощью опций --port и --socket к соответствующему порту и разъему.

```
% mysqladmin --port=port_num variables
% mysqladmin --socket=/path/to/socket variables
```

Команду mysqladmin можно запускать на любом компьютере, который подключен к серверу. Для подключения к серверу с удаленного компьютера применяется опция --host=host_name.

```
% mysqladmin --host=host_name variables
```

С компьютера, работающего под ОС Windows, можно подключиться к работающему через именованный канал серверу Windows NT с помощью опции --pipe, активизирующей соединение по именованному каналу, и опции --socket=pipe_name, определяющей имя канала.

```
C:\> mysqladmin --pipe --socket=pipe_name variables
```

- Воспользоваться командой ps для вывода командной строки исполняемого процесса mysql. Попробуйте одну из указанных ниже команд (в зависимости от версии ps, поддерживаемой системой) и поищите переменную --datadir в выводимых результатах.

```
% ps axww | grep mysql                    ps BSD—UNIX
% ps -ef | grep mysql                     ps системы System V
```

Команда ps особенно полезна при запуске на одном компьютере нескольких серверов, поскольку позволяет узнать месторасположение сразу всех каталогов данных. Недостаток этого метода заключается в том, что команду ps обязательно нужно запускать на главном компьютере. Кроме того, она будет бесполезна, если переменная --datadir не описана явным образом в командной строке mysql.

- Если MySQL инсталлировалась из исходной дистрибуции, месторасположение каталога данных можно получить из информации о конфигурации. Так, например, месторасположение каталога указывается в элементе верхнего уровня Makefile. Однако будьте осторожны, поскольку позиция каталога является в Makefile значением переменной localstatedir, а не datadir, как ожидают многие. Кроме того, если дистрибуция размещается на смонтированной сетевой файловой системе NFS и используется для установки MySQL на несколько компьютеров, в информации конфигурации отражаются данные только для компьютера, на котором система устанавливалась последней. Вполне возможно, что им окажется не тот компьютер, для которого необходимы данные.
- Если все предыдущие методы вам не подходят, можно воспользоваться командой find **для поиска файлов базы данных.** Приведенная ниже команда ищет все файлы .frm (описания), являющиеся частью инсталляций MySQL:

```
% find / -name "*.frm" -print
```

Во всех примерах этой лекции в качестве каталога данных MySQL определен каталог datadir. Вполне возможно, что на других компьютерах для этих целей может применяться другой каталог.

Структура каталога данных

Каталог данных MySQL содержит все управляемые сервером базы данных и таблицы. Они организованы в структуру простого дерева, что позволяет, в свою очередь, воспользоваться преимуществами иерархической структуры файловых систем ОС UNIX и Windows.

Каждой базе данных соответствует подкаталог, расположенный внутри каталога данных. Таблицам базы данных соответствуют файлы, размещенные внутри каталога базы данных.

Каталог данных содержит также создаваемые сервером файлы состояния, такие, например, как журналы. Они обеспечивают важную информацию о функционировании сервера и просто незаменимы для администратора, особенно когда сервер начинает сбоить и нужно найти источник проблемы. Если неправильно сформированный запрос приводит к сбою в работе сервера, с помощью журналов можно вычислить "виновника".

Как обеспечивается доступ к данным сервера MySQL

Каждый элемент внутри каталога данных находится под управлением MySQL-сервера mysqld. Клиентские программы никогда не обращаются к данным напрямую. Точку взаимодействия, с помощью которой осуществляется доступ к базам данным, обеспечивает сервер, работающий в качестве промежуточного звена между клиентскими программами и данными (рис. 1).

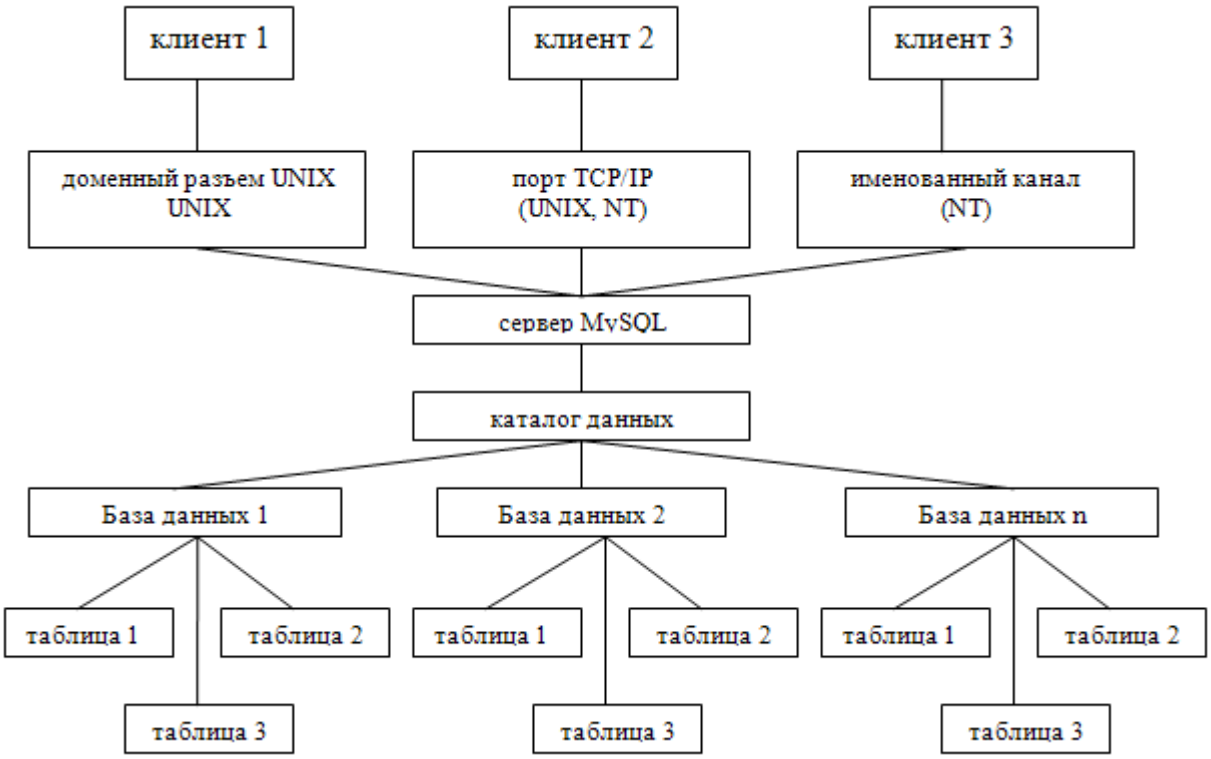

Рис.1. Сервер является промежуточным звеном между клиентскими программами и данными

В процессе запуска сервер открывает регистрационные файлы (если таковые запрашиваются) и предоставляет сетевой интерфейс к каталогу данных, прослушивая сетевые соединения. Для получения доступа к данным клиентская программа устанавливает соединение с сервером и посылает SQL-запрос на выполнение определенных операций (например, создание таблицы, извлечение или обновление записей). Сервер выполняет все операции и результаты отправляет обратно клиенту. Сервер представляет собой мультипотоковый механизм, поэтому может обслуживать несколько клиентских соединений одновременно. Однако так как две и более операции обновления не могут выполняться в один момент, на практике происходит разделение запросов по наборам, чтобы исключить одновременную попытку двух клиентов изменить одну запись.

В обычных условиях использование сервера как единственной точки доступа к базе данных обеспечивает защиту от всякого рода недоразумений, связанных с одновременным доступом нескольких процессов к таблицам базы данных. Администраторы, однако, должны знать, что в некоторых случаях серверы не обладают эксклюзивным контролем над каталогом данных.

В одном каталоге данных запущено несколько серверов. Как правило, для управления всеми базами данных используется один сервер, хотя можно запустить сразу несколько серверов. Если это делается для обеспечения доступа к нескольким отдельным каталогам данных, то проблем с взаимодействием различных серверов не возникает. Можно задать нескольким серверам один каталог данных (хотя удачным такое решение назвать сложно). В таком случае необходимо обязательно убедиться, что системы обеспечивают надежную блокировку файлов друг от друга. Иначе серверы начнут взаимодействовать некорректно. Кроме того, при одновременном запуске нескольких серверов в одном каталоге регистрационные файлы будут содержать беспорядочные данные, а не ценную информацию.

Запущены утилиты isamchk и myisamchk. Эти утилиты используются для поддержки таблиц, устранения проблем и отладки баз данных. Поскольку они обладают возможностью изменения содержимого таблиц, то могут получать доступ к данным одновременно с сервером. Это может стать источником повреждений данных. Администратор должен понимать, как ограничить подобный совместный доступ, чтобы снизить вероятность разрушения таблиц.

Представление баз данных

Каждая управляемая сервером MySQL база данных имеет свой собственный каталог. Он представлен в виде подкаталога каталога данных и имеет такое же название, как и собственно база. Так, например, базе данных my_db будет соответствовать каталог базы данных DATADIR/my_db.

Такое представление значительно упрощает понимание предназначения и принципов работы некоторых операторов обработки баз данных. Так, оператор create database db_name создает пустой подкаталог db_name в каталоге данных, устанавливая права владения и режим, которые обеспечивают доступ только для пользователя сервера MySQL (UNIX-пользователя, работающего на сервере). Аналогичных результатов создания базы данных пользователь сервера может добиться и вручную, введя следующие команды.

% mkdir DATADIR/db_name Создает каталог базы данных
% chmod 700 DATADIR/db_name Делает его доступным только для пользователя сервера MySQL

Такой подход, заключающийся в создании новой базы данных посредством создания пустого каталога, противоречит принципам, принятым в некоторых других СУБД, которые создают большое количество управляющих и системных файлов даже для "пустой" базы данных.

Также легко можно реализовать и команду DROP database. Команда drop database db_name удаляет из каталога данных подкаталог db_name вместе со всеми расположенными в нем файлами. Тех же результатов можно достичь с помощью команды:

% rm -rf DATADIR/db_name

(Разница между этими двумя командами заключается в том, что сервер удалит лишь файлы, расширение которых отвечает табличным файлам. Если же в каталоге базы данных были созданы также какие-либо другие файлы, они останутся нетронутыми и каталог удален не будет.)

Команда show databases на самом деле выводит список названий подкаталогов каталога данных. Некоторые системы управления базами данных поддерживают специальную таблицу со списком всех баз данных. В MySQL такой таблицы нет. Благодаря простоте структуры список баз является списком подкаталогов каталога данных. Следовательно, и необходимость в подобной таблице отсутствует.

Представление таблиц баз данных

Каждая таблица представлена в каталоге базы данных в виде трех файлов: файла формы (описания), файла данных и файла индексов. Основное имя файла соответствует названию таблицы, а его расширение отражает тип файла. Краткое описание расширений представлено в табл.1. По расширениям файлов данных и индексов можно определить, используется ли в таблице старый формат ISAM или новый MyISAM.

Таблица 1. Типы файлов MySQL

Тип файла	Расширение имение файла	Содержимое файла
Файл формы	frm	Описывает структуру таблицы (столбцы, типы столбцов, индексы и т.п.)
Файл данных	ISD (ISAM) или MYD (MyISAM)	Содержит данные таблицы, т.е. его строки
Файл индексов	ISM (ISAM) или MYI (MyISAM)	Содержит дерево индексов для каждого файла данных. Этот файл существует независимо от того, имеются в таблице индексы или нет

При выполнении оператора CREATE TABLE tblname, определяющего структуру таблицы, сервер создает файл tblname.frm с внутренней кодировкой структуры. Кроме того, создаются также файлы данных и индексов с информацией об отсутствии записей и индексов. (Если оператор create table включает спецификации индексов, в файле индексов они отражаются соответствующим образом.) Параметры владельца и режима файлов таблицы устанавливаются такими, чтобы обеспечить доступ

только пользователю сервера MySQL.

При исполнении оператора alter table расшифровывает файл tbl_name.frm и изменяет файлы данных и индексов с учетом определенных оператором структурных изменений. Такие же операции имеют место и при выполнении операторов create index и drop index, поскольку они рассматриваются сервером как эквивалентные оператору ALTER table. В процессе выполнения оператора drop table из каталога базы данных удаляются все три представляющих таблицу файла.

Пользователь не может вручную создать или изменить таблицу, хотя имеется возможность удалить ее — для этого достаточно удалить три соответствующих файла. Так, например, эквивалентом оператора drop table my_tbl для текущей базы данных my_tbl может быть команда:

```
% rm -f DATADIR/my_db/my_tbl. *
```

Вывод оператора SHOW tables mydb представляет собой простой список имен (без расширений) FRM-файлов каталога базы данных my_db. Как уже отмечалось ранее, некоторые СУБД поддерживают специальный реестр со списком всех таблиц баз данных. В MySQL такой реестр не нужен, поскольку список таблиц легко определяется благодаря структуре каталога данных.

Ограничения операционной системы на имена баз данных и таблиц

В MySQL устанавливается несколько основных правил присвоения имен базам данных и таблицам.

- Имена могут включать буквы и цифры текущего набора символов, а также символы подчеркивания и доллара ("_" и "$").
- Длина имен не может превышать 64 символа.

Однако так как именам баз данных и таблиц соответствуют названия каталогов и файлов, операционная система, под управлением которой работает сервер, может накладывать дополнительные ограничения.

Во-первых, в именах баз данных и таблиц можно использовать только разрешенные для имен файлов символы. Так, например, символ "$" разрешается правилами MySQL, однако в некоторых операционных системах его нельзя применять в именах файлов. Такого рода ограничения отсутствует в ОС UNIX и Windows. Наиболее значительные проблемы могут возникнуть в момент ссылки на имена баз данных при выполнении задач администрирования непосредственно из оболочки. Предположим, например, что база данных имеет имя $my_db. В этом случае всякая ссылка на имя базы может интерпретироваться как ссылка на переменную:

% Is $my_db my_db: Undefined variable.

Чтобы избежать возможных недоразумений, нужно либо вообще избегать использования символа "$", либо использовать кавычки, подтверждая его специальное значение:

% Is my_db

% Is '$my_db'

Необходимо использовать одинарные кавычки, поскольку двойные кавычки не отражают специального значения символов имени.

Во-вторых, несмотря на то, что MySQL разрешает задавать для баз данных и таблиц имена длиной до 64 символов, на самом деле их длина ограничивается максимально возможной длиной имен файлов. В большинстве случаев эта проблема как таковая отсутствует, хотя в некоторых UNIX-системах System V все еще существует старое ограничение в 14 символов. В таком случае имя таблицы должно содержать не более 10 символов, поскольку четыре остальных позиции отводится под точку и трехсимвольное расширение.

В-третьих, на присвоение имен базам данных и таблицам оказывает влияние также чувствительность используемой файловой системы к регистру символов. Если буквы нижнего и верхнего регистров операционной системой воспринимаются по-разному (как, например, в ОС UNIX), имена my_tbl и My_tbl будут указывать на разные таблицы. Если же регистр не играет никакой роли (как, например, в Windows), my_tbl и My_tbl окажутся разными названиями одной и той же таблицы. Об этом следует

помнить при разработке базы данных, которую впоследствии планируется перенести на другую платформу.

Влияние структуры каталога данных на производительность системы

Построение структуры каталогов данных на основании иерархической структуры файловой системы делает структуру каталога данных понятной для пользователей. В то же время, эта структура оказывает определенное влияние на производительность системы. Особенно это справедливо в отношении операций открытия файлов таблиц баз данных.

Поскольку в структуре каталога данных таблица представляется несколькими файлами, для открытия каждой таблицы требуется не один, а сразу несколько дескрипторов файлов. Сервер весьма эффективно кэширует дескрипторы, однако загруженному серверу для обслуживания множества одновременных клиентских соединений и обработки сложных запросов к нескольким таблицам потребуется большое количество дескрипторов. Дескрипторы файлов — весьма ограниченный ресурс во многих системах, особенно в тех из них, в которых по умолчанию установлен низкий лимит.

Еще один негативный момент, связанный с представлением одной таблицы в виде нескольких файлов, заключается в том, что с увеличением таблиц увеличивается и время их открытия. Операции открытия таблиц целиком и полностью базируются на системных операциях открытия файлов, а следовательно, зависят от эффективности работы механизмов поиска файлов операционной системы. Как правило, эта проблема несущественна, хотя о ней все же стоит помнить при создании базы данных с большим количеством таблиц.

Так, например, каталог базы данных, включающей 10000 таблиц, содержит 30000 файлов. При открытии большого количества таблиц замедление выполнения операций открытия становится достаточно заметным. (Особенно это относится к файловым системам Linux ext2 и Solaris.) Если же эта проблема приобретает действительно угрожающие масштабы, возможно, имеет смысл пересмотреть структуру своих таблиц в соответствии со спецификой работы приложений и соответствующим образом их реорганизовать. Тщательно подумайте, действительно ли необходимо такое большое число таблиц. Иногда приложения генерируют их безо всякой на то необходимости. Много таблиц с аналогичными структурами могут генерироваться приложениями, которые создают отдельную таблицу для каждого пользователя. Чтобы объединить такие таблицы в одну, достаточно добавить столбец, который идентифицирует пользователя владельца строки. Если в результате таких действий число таблиц значительно уменьшится, производительность приложения возрастет.

Всегда при проектировании базы данных следует анализировать, подходит ли для данного приложения та или иная стратегия. Однако существуют также причины, не позволяющие объединять таблицы описанным выше способом. Основными среди них являются следующие.

- Ограничение дискового пространства. Объединение таблиц приводит к уменьшению их числа (уменьшая, в свою очередь, время открытия), но сопровождается добавлением дополнительных столбцов (занимая дополнительное дисковое пространство). В результате складывается достаточно распространенный компромисс между временем обработки и свободным пространством, и администратор должен решить, какой фактор более важен. Если на первом плане стоит скорость, возможно, придется пожертвовать дополнительным дисковым пространством. Если же объемы дисков сильно ограничены, придется использовать большее количество таблиц и немного дольше ждать при открытии.
- Вопросы безопасности. Эта причина также может в значительной степени ограничить возможности и желание объединить таблицы. Основная цель использования отдельных таблиц для каждого пользователя — открытие доступа к данным таблицы только пользователям, обладающим соответствующими привилегиями. На самом деле права пользователей определяются полномочиями на уровне таблицы. После объединения данные всех пользователей окажутся в одной таблице.

  Возможности MySQL не позволяют ограничить для определенного пользователя доступ к определенным строкам таблиц. Соответственно, нельзя объединить таблицы, не потеряв контроля над доступом. Однако если доступ к данным контролируется приложением (пользователи не подключаются непосредственно к базе данных), можно спокойно объединить таблицы и для определения прав доступа воспользоваться соответствующими средствами приложения.

MySQL накладывает внутреннее ограничение на размеры таблицы, однако, поскольку таблицы представляются в виде файлов, учитывать следует также и максимальный размер файла, разрешаемый операционной системой. Другими словами, максимально возможный размер таблицы определяется наиболее жестким ограничением среди ограничений MySQL и операционной системы.

В последнее время намечается тенденция к ослаблению ограничений на размеры таблиц. Так, например, если в ОС IBM AIX 4.1 существовало ограничение на размер файла 2 Гбайта, то в ОС IBM AIX 4.2 оно увеличилось до приблизительно 64 Гбайт. Внутренний лимит на размер таблицы в MySQL в новых версиях также увеличивается. Так, если во всех предшествующих версии 3.23 системах он составляет 4 Гбайта, то в 3.23 был поднят до приблизительно 9 миллионов терабайт. Данные табл. 2 позволяют оценить, как внутренний лимит на размер таблицы в MySQL сопоставляется с ограничением файловой системы AIX. Подобные сопоставления можно применять и для других операционных систем.

Таблица 2. Сопоставление ограничений MySQL и операционной системы

Версия MySQL	Версия AIX	Максимальный размер таблицы	Ограничивающий фактор
MySQL 3 22 22	AIX 4 1	2 Гбайт	Максимальный размер файла AIX 2 Гбайта
MySQL 3 22 22	AIX 4 2	4 Гбайт	Максимальный размер таблицы MySQL — 4 Гбайта
MySQL 3 23	AIX 4.1	2 Гбайт	Максимальный размер файла AIX 2 Гбайта
MySQL 3 23 11	AIX 4 2	64 Гбайт	Максимальный размер таблицы MySQL — 64 Гбайта

Файлы состояния MySQL

Помимо подкаталогов баз данных, каталог данных MySQL содержит множество файлов состояния. Краткий обзор этих файлов представлен в табл. 3, а детальное их описание следует далее. По умолчанию основная часть имени этих файлов содержит имя главного компьютера. В таблице это имя заменено словом HOSTNAME.

Сервер записывает ID-номер своего процесса (Process ID — PID) в PID- файл при запуске и удаляет этот файл при завершении работы. Именно с помощью PID-файла сервер позволяет находить себя работающим процессам. Так, например, если во время завершения работы системы запустить сценарий mysql.server для завершения работы и сервера MySQL, данный сценарий обратится к PID-файлу. Это обращение позволит определить, какому процессу отправить команду на завершение работы.

Таблица 3.

Тип файла	Имя по умолчанию	Содержимое файла
ID-номер процесса	HOSTNAME, pid	ID-номер процесса сервера
Журнал ошибок	HOSTNAME, err	События запуска и завершения работы, а также записи об ошибках
Общий журнал	HOSTNAME, log	События подключения /отключения и информация о запросах
Журнал обновлений	HOSTNAME, nnn	Текст всех запросов, изменяющих содержимое или структуру таблицы

Журнал ошибок создается сценарием safe_mysqld. Впоследствии именно в этот журнал перенаправляются все стандартные сообщения об ошибках сервера. Другими словами, журнал содержит все сообщения, записываемые сервером в stderr, и существует, только если сервер запускается с помощью вызова сценария safe_mysqld. (Это более предпочтительный метод запуска сервера, поскольку safe_mysqld перезапускает сервер при сбое в работе из-за ошибки.)

Общий журнал и журнал обновления являются необязательными. Используя опции --log и --log-update, можно задать регистрацию только необходимой информации.

Общий журнал предоставляет информацию о функционировании сервера: кто и с какого компьютера подключился и какие запросы присылает. Журнал обновлений также содержит информацию о

запросах, однако только о тех из них, которые связаны с изменением содержимого баз данных. Содержимое же журнала обновлений представляется в виде операторов SQL. Впоследствии их можно выполнить, представив в виде ввода для клиента mysql. Журналы обновлений оказываются особенно полезными в случае сбоя, когда необходимо отследить все изменения, внесенные с момента последнего резервирования базы данных. Их использование позволяет восстановить базы данных до состояния, в котором они находились перед самым сбоем.

Ниже представлены данные, которые заносятся в общий журнал в течение короткой клиентской сессии. На протяжении работы клиент создает таблицу в базе данных test, вставляет в нее строку и затем удаляет всю таблицу.

```
990509  7:34:09     492 Connect                paul@localhost on test
                        492 Query                      show databases
                    492 Query                  show tables
                    492 Field List             tbl_l
                    492 Field List             tbl 2
990509  7:34:22         492 Query              CREATE TABLE my_tbl (val INT)
990509  7:34:34         492 Query              INSERT INTO my_tbl VALUE(A)
990509  7:34:38         492 Query              DROP TABLE my tbl
990509  7:34:40         492 Quit
```

Отдельные столбцы общего журнала отражают дату и время события, ID-номер сервера, тип события и относящуюся к нему специальную информацию Тот же сеанс в журнале обновлений отобразится следующим образом:

```
use test;
CREATE TABLE my_tbl (val INT);
INSERT INTO my_tbl VALUE(A);
DROP TABLE my_tbl;
```

Для получения расширенной формы журнала обновлений используется опция --log-long-format. В расширенном журнале предоставляется информация также об отправителе и времени поступления запроса. Эти данные занимают, конечно, больше места на диске, однако позволяют узнать, кто и что пытался сделать. Сопоставлять информацию о событиях в общем журнале и журнале обновлений для этого не нужно. Для уже описанной выше сессии расширенный журнал обновлений будет выглядеть таким образом:

```
# Time: 9905097:43:42
# User@Host: paul [paul] @ localhost []
use test;
CREATE TABLE my_tbl (val INT);
# User@Host: paul [paul] @ localhost {]
INSERT INTO my_tbl VALUE(A);
# Time: 9905097:43:43
# User@Host: paul [paul] 0 localhost П
DROP TABLE my_tbl;
```

Администратору настоятельно рекомендуется убедиться, что файлы журналов защищены от просмотра случайными пользователями. Как общий журнал, так и журнал обновлений могут содержать такую критически важную информацию, как пароли: ведь они включают текст отправляемых запросов. Ниже представлен пример одной записи журнала, которую вряд ли даже начинающий администратор захочет показать другому пользователю, поскольку она содержит пароль пользователя.

990509 7:47:24 4 Query UPDATE user SET Password=PASSWORD("secret") WHERE user="root"

Для защиты каталога данных можно воспользоваться простой командой:

```
% chmod 700 DATADIR
```

Запустите эту команду, зарегистрировавшись в качестве пользователя-владельца каталога данных. Не забудьте также зарегистрироваться под именем этого пользователя на сервере, иначе команда не только запретит другим пользователям

доступ к каталогу данных (что и требуется), но также закроет базы данных для сервера (чего допустить ни в коем случае нельзя).

Файлы состояния размещаются на верхнем уровне каталога данных вместе с каталогами баз данных. Поэтому иногда пользователи беспокоятся, что имена файлов состояния могут конфликтовать с именами баз данных (например, при выполнении сервером оператора SHOW databases). Этого бояться не стоит. Информация о событиях и состояниях хранится в файлах, а базы данных записаны в каталогах, что позволяет исполняемым программам легко отличить их, вызвав команду stat(). (Именно таким образом их различает сервер.) Просматривая каталог данных с помощью опции ls -1, пользователь может отличить файлы состояния от каталогов баз данных, определив первую букву данных в режиме: ' - ' или ' d':

```
% ls -l DATADIR

total 31

drwxrwx-- 1 mysqladm mysqlgrp  1024 May  8 13:22 blgdb

drwxrwx-- 2 mysqladm mysqlgrp  1024 Dec 15 22:34 mysql

-rw-rw--- 1 mysqladm mysqlgrp    64 May  9 20:11 pit-vlper. 001

-rw-rw-r- 1 mysqladm mysqlgrp 24168 May  9 20:11 pit-vlper. Err

-rw-rw--- 1 mysqladm mysqlgrp  4376 May  9 20:11 pit-vlper. Log

-rw-rw-r- 1 mysqladm mysqlgrp     5 May  9 20:11 pit-vlper. Pld

drwxrwx-- 7 mysqladm mysqlgrp   512 Sep 10  1998 sql-bench

drwxrwx-- 2 mysqladm mysqlgrp   512 May  9 07:34 test
```

Можно также просто просмотреть список имен, ведь имена всех файлов состояния включают точку, а в именах баз данных она не используется (более того, запрещена).

Перемещение содержимого каталога данных

В предыдущем разделе описывалась структура каталога данных, создаваемая сервером по умолчанию. Этот каталог содержит все рабочие базы данных и файлы состояния. Иногда возникает необходимость в определении специального места для хранения содержимого каталога данных. В этой части лекции рассказывается, зачем может потребоваться перемещать отдельные части каталога данных (и даже сам каталог),какие его компоненты можно перемещать и как такое перемещение осуществить.

Возможности MySQL позволяют администратору перемещать каталог данных или его внутренние элементы на другое место. Необходимость в этом может быть вызвана следующими причинами.

- Каталог данных можно разместить на диске большего размера, чем используется в настоящий момент.

- Если каталог данных располагается на часто используемом диске, перемещение его на другой диск позволит уровнять загрузку среди физических дисков. В этом случае можно разместить файлы баз данных и журналов на отдельном диске или распределить их по нескольким дискам сразу.

- Каждый из одновременно запущенных серверов можно разместить в своем каталоге данных. Такой подход является одним из способов обойти ограничения на файловые дескрипторы, особенно если эти ограничения нельзя устранить посредством настройки ядра системы.

- Некоторые операционные системы хранят PID-файлы сервера в отдельном каталоге, например /var/run. Возможно, для большей согласованности работы системы администратор пожелает разместить в этой папке и PID-файлы MySQL.

Методы перемещения

Существует два способа перемещения компонентов каталога данных.

- Определение опции загрузки сервера с помощью командной строки или в группе [mysqld] конфигурационного файла.
- Перемещение элементов и создание в исходном каталоге символической связи (symbolic link), указывающей на новое местоположение.

Ни один из приведенных методов не является универсальным для переноса информации. В табл. 4 отмечается, какие компоненты каталога данных можно перемещать и какой метод следует для этого использовать. Если применяется первый метод, можно задать опции в глобальном конфигурационном файле /etc/my.cnf (C:\my.cnf на компьютерах, работающих под управлением ОС Windows). В последних версиях ОС Windows этот файл может располагаться в системной папке (C:\Windows).

Таблица 4. Обзор методов перемещения

Перемещаемый компонент	Применяемый метод перемещения
Цеый каталог данных	Опция запуска или символическая связь
Каталоги отдельных баз данных	Символическая связь
Отдельные таблицы баз данных	Символическая связь
PID-файл	Опция запуска
Файл общего журнала	Опция запуска
Файл журнала обновлений	Опция запуска

Для перемещения можно также применить файл my.cnf, расположенный в каталоге данных по умолчанию, однако делать это не рекомендуется. Если ваша цель — переместить весь каталог данных, необходимо оставить этот каталог нетронутым на старой позиции, чтобы разместить в нем конфигурационный файл со ссылкой на "реальный" каталог данных. Это может привести к путанице. Для определения опций сервера лучше воспользоваться конфигурационным файлом /etc/my.cnf.

Определение эффекта перемещения

Прежде чем приступать к перемещению каких-либо компонентов, настоятельно рекомендуется убедиться, что эта операция приведет к желаемому эффекту. Для получения информации о пространстве диска некоторые пользователи предпочитают использовать команды du, df и ls -l, хотя этот выбор, в первую очередь, определяется правильным пониманием структуры используемой файловой системы.

В приведенном ниже примере существует едва заметная ловушка, в которую можно попасться при перемещении каталога данных. Предположим, что каталог данных /usr/local/var планируется переместить в каталог /var/mysql, поскольку согласно выводу команды df файловая система /var содержит больше свободного пространства.

```
% df /usr/var
Filesystem   1k-blocks  Used     Avail   Capacity   Mounted on
/dev/wd0s3e  396895     292126   73018   80%        /usr
/dev/wd0s3f  1189359    1111924  162287  15%        /var
```

Сколько же пространства освободится в файловой системе /usr в результате перемещения каталога данных? Чтобы вычислить этот объем, воспользуемся командой du -s и посмотрим, сколько этот каталог занимает:

```
% cd /usr/local/var
% du -s
133426
```

Как видно, этот каталог занимает чуть более 130 Мбайт, которые можно освободить в /usr. Однако можно ли этот прием реализовать на самом деле? Запустите команду df в каталоге данных:

```
% df /usr/local/var
Filesystem   1k-blocks  Used     Avail   Capacity   Mounted on
/dev/wd0s3f  1189359    1111924  162287  15%        /var
```

Что же получается? При запросе объема свободного пространства в файловой системе, содержащей каталог /usr/local/var, команда df отображает свободный объем в /var. Почему так? Ответ на этот вопрос дает команда ls -1:

```
% is -l /usr/local
lrwxrwxr 1 root wheel 10 Dec 11 23:46 var -> /var/mysql
```

Из результатов выполнения этой команды видно, что /usr/local/var является символической связью с /var/mysql. Другими словами, каталог данных уже перемещен в файловую систему /var и включает указывающую на нее символическую связь. Соответственно, никакой выгоды перемещение каталога данных из /usr в /var не принесет.

Суть этого примера состоит в том, что несколько действий по определению эффекта перемещения могут показать нецелесообразность подобного перемещения. Такая предосторожность позволяет вовремя остановиться и не тратить уйму времени на перемещение только для того, чтобы затем понять, что достичь нужной цели невозможно.

Перемещение каталога данных

Для перемещения каталога данных необходимо завершить работу сервера и только после этого перенести каталог на новую позицию. Затем необходимо удалить данные исходного каталога и заменить его символической связью, указывающей на новую позицию, либо перезапустить сервер с опцией, определяющей новое местоположение. Синтаксис командной строки и конфигурационного файла представлен в табл. 5.

Таблица .5. Синтаксис перемещения каталога данных

Метод	Синтаксис
Командная строка	--datadir=/path/to/dir
Конфигурационный файл опций	[mysqld] datadir=/path/to/dir

Перемещение баз данных

Базы данных можно перемещать только с помощью метода символической связи. Для этого необходимо завершить работу сервера, перенести каталог базы данных, затем удалить этот каталог и заменить его символической связью, указывающей на новую позицию. После этого можно перезапустить сервер.

Так, например, перемещение базы данных bigdb на другое место выполняется с помощью следующих команд.

% mysqladmin -u root -p shutdown

Enter password: ********

% cd DATADIR

% tar cf - bigdb | (cd /var/db; tar xf -)

% mv bigdb bigdb.orig

% ln -s /var/db/bigdb .

% safe_mysqld &

Меры предосторожности при перемещении

Необходимо завершить работу сервера перед выполнением операции перемещения и запустить его снова впоследствии. При перемещении некоторых компонентов (например, каталога базы данных) можно, хотя и не рекомендуется, оставить сервер в рабочем состоянии. В этом случае следует убедиться, что сервер не обращается к перемещаемой базе данных. Не забудьте также выполнить оператор flush tables перед перемещением базы данных, чтобы сервер закрыл все открытые файлы таблиц. Игнорирование этих моментов может привести к повреждению таблиц.

Для выполнения всех этих команд необходимо зарегистрироваться в качестве владельца каталога данных. Как видите, исходный каталог данных переименовывается для безопасности в bigdb.orig. После проверки правильности работы сервера с перемещенной базой данных его можно удалить:

% rm -rf bigdb.orig

Перемещение таблиц баз данных

Перемещение отдельных таблиц баз данных — далеко не самая лучшая идея. В случае необходимости эту операцию можно реализовать посредством переноса файлов таблиц на новую позицию и установки символических связей в исходном каталоге данных, указывающих на новое местоположение. Однако если впоследствии выполнить операторы alter table или optimize table, изменения внесены не будут.

В процессе выполнения каждого такого оператора в каталоге базы данных сначала создается временная таблица, которая и поддается изменению или оптимизации. Сразу после этого исходная таблица удаляется, а ее имя присваивается временной таблице. В результате этой процедуры символические связи будут удалены, а новая измененная таблица окажется записанной в том же каталоге данных, откуда ранее была перемещена исходная таблица. В конечном итоге, старые перемещенные ранее файлы таблиц оказываются на новой позиции. В большинстве случаев пользователи о них забывают, что способствует неэффективному использованию дискового пространства. Кроме того, в процессе изменения удаляются символические связи, из-за чего впоследствии оказывается довольно трудно вспомнить, куда были перемещены файлы таблиц.

Поскольку практически невозможно гарантировать, что обладающие соответствующими полномочиями пользователи не будут пытаться изменить или оптимизировать таблицы (таким образом, сводя на нет все усилия по их перемещению), лучше оставить файлы таблиц размещенными в каталоге базы данных.

Перемещение файлов состояния

Перемещение PlD-файла, общего журнала и журнала обновлений осуществляется с помощью символических связей. Как уже отмечалось ранее, журнал ошибок создается сценарием safemysqld и поэтому не может перемещаться куда-либо (если, конечно, для этого не прибегнуть к редактированию safemysqld).

Для записи файла состояния в новую позицию завершите работу сервера и перезапустите его, точно определив посредством соответствующей опции новое местоположение. Синтаксис командной строки и файла опций для каждого файла состояния представлен в табл. 6.

Таблица 6. Синтаксис перемещения файлов состояния

Метод	Синтаксис

Командная строка	`-pid-file=pidfile`
	`-log=logfile`
	`-log-update=updatefile`

Файл опций	`[mysqld]`
	`pid-file=pidfilee`
	`log=logfile`
	`log-update=updatefile`

Удаление перемещенной базы данных

Удалить базу данных можно с помощью оператора drop database, хотя в старых версиях MySQL с удалением перемещенной базы данных могут возникнуть проблемы. Таблицы такой базы данных будут удалены правильно. Ошибка возникает при попытке сервера удалить каталог базы данных поскольку он является лишь символической связью, а не реальным каталогом. Администратор MySQL должен самостоятельно удалить каталог базы данных и указывающую на него связь. Эта проблема устранена в MySQL версии 3 23 и выше.

Если определить имя файла состояния, указав полный путь, то файл будет создан в определенной этим путем позиции. Во всех остальных случаях файл создается в каталоге данных. Так, например, при определении опции --pid-file=/var/run/mysqld.pid PID-файл mysqld.pid будет создан в каталоге /var/run. Если же определена опция --pid-file=mysqld.pid, этим файлом окажется файл DATADIR/mysqld.pid.

При определении имени журнала обновлений без расширения MySQL будет создавать последовательные имена каждый раз при открытии этого журнала. Эти имена будут дополняться расширением nnn, где nnn — следующий не используемый существующим файлом журнала обновлений номер (например, update.001, update.002 и тп). Чтобы избежать создания подобных имен сервером, достаточно определить имя с явным расширением.

Лекция: Расширение возможностей

Библиотека функций отладки

В MySQL входит библиотека функций отладки, первоначально созданная Фредом Фишем (Fred Fish). Чтобы разрешить ее использование, нужно на этапе компиляции программы вызвать сценарий configure с опцией –with_debug. Если в распоряжении имеется бинарный дистрибутив, проверьте версию какого либо исполняемого файла. Программы, скомпилированные с поддержкой отладки, имеют суффикс –debug. Библиотека функций отладки является частью библиотеки mysqlclient. Макросы библиотеки объявлены в файле dbug.h.

Названия всех макросов начинаются с префикса DBUG_ (табл. .1). Если нужно отключить отладку, определите макроконстанту DBUG_OFF. При ее наличии все остальные макросы игнорируются.

Таблица .1.

Макрос	Описание
DBUG ENTER (функция)	Этот макрос принимает имя функции, в которую входит программа. Его нужно указывать после объявления локальных переменных, но перед вызовом каких либо инструкций. Например: DBUG ENTER ("main")
DBUG EXECUTE (ключевое слово)	Этот макрос помечает указанную инструкцию меткой. Например: DBUG EXECUTE ("where", print where(tmp,"cache"));
DBUG FILE	Эта макроконстанта инкапсулирует дескриптор выходного файла, в который записывается отладочная информация. Например: sprintf (DBUG FILE, "\nWHERE: (%s)" , info);

DBUG LONGJMP (среда, значение)	Если в программе используется функция LONGJMP(), замените ее данным макросом.
DBUG POP ()	Этот макрос восстанавливает предыдущее состояние отладки. DBUG POP ();
DBUG_PRINT (ключевое слово, формат, [аргументы])	Этот макрос записывает отладочную информацию в файловый поток, как если бы была вызвана функция fprintf с константой DBUG_FILE в качестве дескриптора. Первый это ключевое слово которое можно использовать с описанным ниже флагом d. Второй аргумент — это набор параметров, передаваемый функции fprintf. Например: DBUG_PRINT ("mfunkt" , ("name: "%s"" , name));
DBUG PROCESS (имя)	Этот макрос задает имя текущего процесса. Например: DBUG PROCESS (argv [0]);
DBUG PUSH (формат)	Этот макрос задает новые параметры для текущего сеанса отладки. Все они помещаются в стек, поэтому можно восстанавливать предыдущие состояния с помощью макроса DBUG POP (). Например: DBUG PUSH (" d:t ");
DBUG RETURN (значение)	Этот макрос заменяет инструкцию return. Если функция нечего не возвращает, пользуйтесь макросом DBUG VOID RETURN ().
DBUG SET JMP (среда)	Этот макрос заменяет функцию setjmp ().
DBUG VOID RETURN	Этот макрос указывает на то, что функция не возвращает никаких значений

Отладка функции начинается с того, что в ее начало помещается макрос DBUG ENTER(). Затем все вызовы инструкции return заменяются либо макросом DBUG ENTER(), либо DBUG VOID RETURN. Это позволяет отладчику определять, когда управление передается той или иной функции.

Макрос DBUG EXECUTE() помечает отдельную строку кода ключевым словом. Макрос DBUG PRINT записывает сообщение в отладочный файл. Можно напрямую работать с этим файлом благодаря макроконстанте DBUG FILE, в которой хранится его дескриптор.

В листинге показан пример отладки функции, которая вычисляет факториал заданного целого числа.

```
ttinclude <dbug.h>

int factorial (register int value)
{
   DBUG ENTR ("factorial");
   DBUG PRINT ("find", ("find %d factorial", value));
if (value > 1)
{
value *= factorial (value - 1);
}
   DBUG ENTR ("result", ("result %d", value));
   DBUG RETURN (value);
}
```

Листинг 1.

Программа, работающая с библиотекой функций отладки, обычно начинает отладку, вызывая макрос DBUG PUSH(). Утилиты MySQL включают отладку, если получен соответствующий аргумент командной строки или если установлена специальная переменная среды.

В табл. 2 перечислены флаги, понимаемые отладчиком и передаваемые макросу DBUG PUSH (). Они определяют, какая информация должна быть представлена в выходных данных. Строка формата выглядит как последовательность флагов, разделенных двоеточиями. Некоторые флаги требуют наличия параметров. Например, флаг d принимает список ключевых слов, разделенных запятыми.

Таблица 2.

Флаг	Описание
d [, ключевые слова]	Этот флаг разрешает выводить информацию макросам с именами вида DBUG ключевое слово. Если список ключевых слов не указан, то подразумеваются все макросы. Ключевые слова должны задаваться без префикса DBUG.
D [,время]	Этот флаг свидетельствует о том, что вывод отладочной информации должен быть задержан на указанное число десятых долей секунды. Например, флаг D, 25 означает,

	что при выводе каждой строки будет выдерживаться пауза длительностью 2,5 секунды.
f [,функции]	Этот флаг разрешает выводить отладочную информацию только из указанных функций. Например, флаг f main означает, что будут включены макросы, находящиеся в теле функции main ().
F	Этот флаг указывает на то, что каждую строку отладочной информации необходимо сопровождать именем исходного файла.
g	Этот флаг включает режим профилирования. В результате будет создан файл dbugmon.out. В качестве аргумента может быть указан список функций, для которых выполняется профилирование. В противном случае подразумеваются все функции. Более подробную информацию об этом можно найти в файле dbug/readme.prof дистрибутива MySQL
i	Этот флаг указывает на то, что каждую строку отладочной информации необходимо сопровождать идентификатором процесса или потока, в зависимости от контекста.
L	Этот флаг указывает на то, что каждую строку отладочной информации необходимо сопровождать номером строки исходного файла.
n	Этот флаг указывает на то, что каждую строку отладочной информации необходимо сопровождать информацией о глубине вызова текущей функции.
N	Этот флаг включает нумерацию строк в файле отладки.
o [, файл]	Этот флаг говорит о том, что отладочная информация должна направляться в указанный файл. По умолчанию эта информация отображается на экране, но многие клиентские программы изменяют данную установку, создавая в каталоге /tmp файл с именем программы и расширением .Trace.
O [, файл]	Этот флаг аналогичен флагу o, но после каждой записи в файл будет очищаться файловый буфер
P [, процессы]	Этот флаг разрешает выводить отладочную информацию только указанным процессам. Имя процесса должно быть задано с помощью макроса DBUG_PROCESS
P	Этот флаг указывает на то, что каждую строку отладочной информации необходимо сопровождать именем процесса
r	Этот флаг заставляет выравнивать выводимую информацию по левому краю экрана после вызова макроса DBUG PUSH ().
S	При наличии этого флага отладчик будет вызывать функцию sanity(file, line) для каждой отлаживаемой функции, пока первая не вернет значение, отличное от нуля
t [, уровень]	Этот макрос включает вывод строк, помечающих точки вызова и завершения функций. Через запятую может быть указан максимальный уровень трассировки, по достижении которого отладочная и трассировочная информация перестает выводиться.

Исходная документация, написанная Фредом Фишем, находится в файле dbug/user.r в исходном каталоге MySQL. С помощью утилиты nroff этот файл можно преобразовать в формат Postscript или в текстовый формат.

Создание наборов символов

Допускается включать в программу MySQL новые наборы символов. Для простого, однобайтового набора требуется лишь один файл с четырьмя таблицами преобразований. В случае сложного набора необходимо также написать функцию, выполняющую сортировку строк.

Листинг .2

```
# конфигурационный файл для набора символов latin1.
# массив ctype{} (должен содержать 257 элементов).

  00 20 20  20  20  20  20  20  20  20  28  28  28  28  28  28  20
20 20  20  20  20  20  20  20  20  20  20  20  20  20  20  20  20
48  10  10  10  10  10  10  10  10  10  10  10  10  10  10  10  10
... ... ... ...
02  02  02  02  02  02  02  02  02  02  02  02  02  02  02  02  02
02  02  02  02  02  02  02  02  02  02  02  02  02  02  02  02  02
```

```
# Массив to lower (должен содержать 256 элементов).

00  01  02  03  04  05  06  07  08  09  0A  0B  0C  0D  0E  0F
10  11  12  13  14  15  16  17  18  19  1A  1B  1C  1D  1E  1F
... ... ... ...
E0  E1  E2  E3  E4  E5  E6  E7  E8  E9  EA  EB  EC  ED  EE  EF
F0  F1  F2  F3  F4  F5  F6  F7  F8  F9  FA  FB  FC  FD  FE  FF

# Массив to upper (должен содержать 256 элементов).

00  01  02  03  04  05  06  07  08  09  0A  0B  0C  0D  0E  0F
10  11  12  13  14  15  16  17  18  19  1A  1B  1C  1D  1E  1F
... ... ... ...
E0  E1  E2  E3  E4  E5  E6  E7  E8  E9  EA  EB  EC  ED  EE  EF
D0  D1  D2  D3  D4  D5  D6  F7  D8  D9  DA  DB  DC  DD  DE  FF

# Массив sort order (должен содержать 256 элементов).

00  01  02  03  04  05  06  07  08  09  0A  0B  0C  0D  0E  0F
10  11  12  13  14  15  16  17  18  19  1A  1B  1C  1D  1E  1F
... ... ... ...
41  41  41  41  5C  5B  5C  43  45  45  45  45  49  49  49  49
44  4E  4F  4F  4F  4F  5D  F7  D8  55  55  55  59  59  DE  FF
```

Десятичное значение	Шестнадцатеричное значение	Описание
1	0x01	прописная буква
2	0x02	строчная буква
4	0x04	цифра
8	0x08	символ пробела
16	0x10	знак пунктуации
32	0x20	управляющий символ
64	0x40	пусто
128	0x80	шестнадцатеричная

Вторая группа значений представляет собой таблицу ASCII, предназначенную для перевода символов в нижний регистр. Например, символ в позиции 0x41 — это прописная буква "A", но ей соответствует значение 0x61, т.е. строчная "a". Третья группа значений определяет таблицу ASCII для перевода символов в верхний регистр. Последняя таблица задает порядок сортировки и обычно совпадает с третьей таблицей.

Если истинный порядок сортировки невозможно отразить в столь простой таблице, то необходимо написать специальные функции сортировки. Для этого нужно создать файл в каталоге strings дерева MySQL. Здесь же находятся файлы всех остальных наборов символов, например, ctype_big5.c. Программа MySQL ищет в этом файле пять функций и четыре массива. Имена всех функций и массивов включают стандартный префикс и название набора. В листинге 3 показаны прототипы функций и определения массивов для набора символов big5.

Листинг 3.

```
uchar NEAR ctype big5{257}
uchar NEAR to lower big5{}
uchar NEAR to upper{257}
uchar NEAR sort order big5{257}

my bool my like range big5(
        const char *ptr,
```

```
        uint ptr length,
        pchar escape,
        uint ers length,
        char *min str,
        char *max str,
        uint *min length,
        uint *max length)

int my strcoll big5(
        const uchar * s1,
        const uchar * s2)

int my strnncoll big5(
        const uchar * s1,
        int len1,
        const uchar * s2
        int len2)

int my strnxfrm big5(
        uchar * dest,
        uchar * src,
        int len,
        int srclen)

int my strxfrm big5(
        uchar * dest,
        uchar * src,
        int len)
```

Четыре функции осуществляют сравнение строк. Функция с префиксом my like range находит наименьшую и наибольшую строки (с учетом реестра), соответствующие выражению в операторе LIKE. Само выражение передается в аргументе ptr. Аргумент ptr length определяет длину выражения. Содержимое наименьшей и наибольшей строк заносится в аргументы min str и max str соответственно.

Функции с префиксами my strcoll и my strncoll служат аналогами обычных функций strcoll() и strncoll() языка C, которые, в свою очередь, являются версиями функций strcmp и strncmp, учитывающими региональные установки. Функции с префиксами my strxfrn и my strnxfrn эмулируют стандартные функции strxfrn() и strnxfrn(). Получить о них более подробную информацию можно в man файлах UNIX.

В начало файла нужно добавить комментарий, подобный тому, что показан в листинге 13.4. На основании строк, приведенных в комментарии, сценарий configure включает набор символов в клиентскую библиотеку. Переменная strxfrn multiply набор задается в том случае, когда у набора символов есть свои функции сортировки строк. Она определяет максимальный коэффициент удлинения строки при ее прохождении через функцию с префиксом my strxfrm. Как следует из листинга, строки в кодировке big5 не растягиваются.

Если в набор входят многобайтовые символы, потребуется определить переменную mbmaxlen набор. Она задает максимально возможное число байтов в представлении символа. Например, в набор big5 входят двухбайтовые символы.

```
/*
  Эти строки анализируются сценарием configure при создании файла
  ctype.c, поэтому не меняйте их без веских оснований.

  .configure. strxfrm multiply big5=1
  .configure. mbmaxlen big5=2
/*
```

Листинг 4.

Все, что осталось теперь сделать, — это добавить имя набора в списки CHARSETS AVAILABLE и COMPILED CHARSETS в файле configure.ini и перекомпилировать программу. Активизировать доступные наборы символов можно с помощью опций командной строки, конфигурационного файла или SQL инструкций.

Создание функций

В программу MySQL можно добавить новые функции, которые будут использоваться точно так же, как и встроенные функции. Существуют два способа создания таких функций. Первый — это включение функции непосредственно в исходный код MySQL, второй — определение функции в формате UDF (User Definable Function — пользовательская функция). Второй способ подходит, когда функцию требуется хранить и отлаживать отдельно от утилит MySQL. Код функции компилируется в виде библиотечного модуля, который загружается с помощью инструкции CREATE FUNCTION. Первый способ менее удобен, поскольку приходится останавливать сервер и заменять его исполняемый файл. Так обычно поступают с функциями, которые планируется сделать частью проекта MySQL. Ниже будет рассмотрен второй подход.

В дистрибутив MySQL входит пример UDF-функций Он находится в файле sql/udf example.cc. В этом файле содержатся определения шести функций. Я скопировал из него строку, используемую утилитой make для правильного вызова компилятора языка C. В комментариях к файлу рекомендуется выполнить команду make udf example.cc, чтобы посмотреть параметры компиляции статического объектного файла, а затем заменить аргумент – с аргументом – shared – o udf example. so. В листинге 5 показана строка компиляции файла в моей системе RedHat в каталоге sql программы MySQL.

```
c++ \
   -DMYSQL SERVER \
   -DDEFAUT MYSQL HOME="\"/usr/local\"" \
   -DDATADIR="\"/usr/local/share/var\"" \
   -DSHAREDIR="\"/usr/local/share/mysql\"" \
   -DHAVE CONFIG H \
   -DDBUG OFF \
   -I../bdb/build unix \
   -I../innobase/include \
   -I./../include \
   -I./../regex \
   -I. \
   -I../include \
   -I.. \
   -03 \
   -fno-implicit-templates \
   -shared \
   -o udf example.cc
```

Листинг 5.

После компиляции совместно используемой библиотеки нужно скопировать ее в один из каталогов, перечисленных в /etc/ld.so.conf файле. Если каталог будет другим, укажите его имя в переменной среды LD LIBRARY PATH.

Для активизации функции нужно выполнить инструкцию CREATE FUNCTION. В листинге 6 демонстрируется загрузка функций METAPHON и AVGCOST из библиотеки udf example. В результате в таблице mysql.func будут созданы две новые записи, и пока инструкция DROP FUNCTION их не удалит, функции останутся доступны всем пользователям даже в случае перезапуска сервера.

```
CREATE FUNCTION METAPHON RETURNS STRING SONAME "udf example.so";
CREATE AGGREGATE FUNCTION AVGCOST RETURNS REAL SONAME "udf example.so";
```

Листинг 6.

В библиотечном файле может содержаться одна или несколько функций. Язык реализации — C или C++. Каждой SQL-функции в этом файле соответствует как минимум одна функция с аналогичным

именем. Кроме того, могут быть созданы функции с суффиксами init и deinit. Например, в файле udf example.cc содержатся определения функций metaphon (), metaphon init () и metaphon deinit (). Когда вводится инструкция, содержащая вызов, сначала происходит обращение к функции с суффиксом init. Затем для каждой записи выполняется основная функция. В конце вызывается функция с суффиксом deinit. Все три функции должны быть безопасны для потоков. Это означает, что в них нельзя использовать глобальные переменные, меняющие свои значения. Функция с суффиксом _init предназначена для динамического выделения памяти, а функция с суфиксом - deinit освобождает выделенную память.

Основная функция может возвращать значение с плавающей запятой, целое число или строку. В первом случае тип результата должен быть double, во втором — long, а в третьем — char *. В листинге 7 показано несколько прототипов функций.

```
Char *metaphon(
   UDF INIT *initid,
   UDF ARGS *args,
   char *result,
   unsigned long *length,
   char *is null,
   char *error);

long long sequence(
   UDF INIT *initid,
   UDF ARGS *args
   char *is null,
   char *error);
double myfunc double(
   UDF INIT *initid,
   UDF ARGS *args
   char *is null,
   char *error);

my bool metaphon init(
   UDF INIT *initid,
   UDF ARGS *args
   char *message);

void metaphon deinit(
   UDF INIT *initid);
```

Листинг .7

Числовые значения возвращаются непосредственно, а строковые — через указатели. Программа MySQL резервирует 255 — символьный буфер для аргумента result. В аргументе length должен быть указан размер возвращаемого значения. Если размер превышает 255 байтов, нужно создать собственный буфер в инициализирующей функции и передать указатель на него в поле ptr структуры UDF_INIT. Описание полей структуры приведено в табл. 4.

Таблица 4.

Поле	Описание
my bool maybe null	Указывает на то, может ли функция возвращать пустое значение. Если один из аргументов функции может быть пустым, в это поле будет записана единица
unsigned int decimals	Содержит количество цифр после запятой, если функция возвращает числовое значение. Будет указана максимальная точность среди всех аргументов
unsigned int max length	Определяет максимальную длину возвращаемой строки
char *ptr	С помощью этого указателя осуществляется обмен данными между инициализирующей функцией и другими двумя функциями. Например, можно выделить блок памяти и записать сюда адрес этого блока

Если функция возвращает пустое значение, аргумент is_null должен быть равен 1. В случае ошибки в аргумент error записывается значение 1. В результате текущая запись и все последующие станут пустыми.

Описание структуры UDF_ARGS приведено в табл. 5. Через эту структуру программа MySQL передает аргументы функции.

Таблица 5.

unsigned int arg_count	Содержит число аргументов функции. Если это значение фиксировано, проверьте его в инициализирующей функции
enum Item_result *arg_type	Содержит массив типов аргументов. Возможные значения массива таковы: INT_RESULT, REAL RESULT и STRING RESULT. Можно осуществлять проверку типов и в случае несовпадения либо возвращать признак ошибки, либо корректировать содержимое массива, приводя аргументы к нужному типу
char **args	Содержит массив значений аргументов. Если аргумент является строкой, в массиве будет храниться указатель на строку. Длины строковых аргументов приведены в массиве lengths. Если аргумент представляет собой целое число или число с плавающей запятой, приведите значение к типу long или double соответственно
unsigned long *lengths	Содержит массив длин аргументов. В инициализирующей функции эти значения устанавливаются по максимуму на основании определений столбцов. В основной функции длины строковых аргументов являются точными

Создание процедур

Процедуры MySQL выполняют операции над результатами запросов. Процедура активизируется при наличии в конце инструкции SELECT ключевого слова PROCEDURE. В настоящий момент в MySQL входит единственная процедура analyse(). Разрешается создавать собственные процедуры, включая их в программу на этапе компиляции.

Процедуры появились в MySQL версии 3.21, но пока что не вызвали особого энтузиазма. Писать их оказалось слишком сложно для большинства пользователей. Например, для создания процедуры на C++ требуется определить класс, производный от класса Procedure. Код последнего находится в файлах sql / procedure.h и sql / procedure c.c. Процедура analyse реализована в файле sql/sql_analyse_c.c.

С другой стороны можно воспользоваться библиотекой mylua (www.fastflow.it/ mylua) которая позволяет динамически загружать процедуры, написанные на языке LUA (www.lua.org). Для запуска сценария LUA в MySQL нужно вызвать функцию LUA() указав ей имя исходного файла, где содержится определение процедуры.