

Федеральное агентство по образованию  
Томский Государственный Университет Систем Управления и  
Радиоэлектроники  
(ТУСУР)

Кафедра Компьютерных систем в управлении и проектировании (КСУП)

Рыбалка Е. Н., Звонков Д. А.

# *Распределенные базы данных*

Учебно-методическое пособие  
для студентов специальности  
230104 – «Системы автоматизированного проектирования»  
Часть 2

**ТОМСК – 2012**

**Рыбалка Е. Н., Звонков Д. А.**

Распределенные базы данных (часть 2): учеб.-методич. пособие / Е. Н. Рыбалка, Д. А. Звонков, – Томск: Томск. гос. ун-т систем упр. и радиоэлектроники, 2012. – 108 с.

В пособии представлены методические указания по выполнению лабораторных работ по дисциплине «Распределенные базы данных». Кроме того, пособие включает в себя описание СУБД с открытым кодом MySQL и ее диалекта языка SQL,.

Пособие предназначено для студентов высших технических учебных заведений, обучающихся по специальности 230104 – «Системы автоматизированного проектирования».

© Рыбалка Е. Н., Звонков Д. А., 2012

© Том. гос. ун-т систем упр. и  
радиоэлектроники, 2012

## Содержание

1	Задания для лабораторных работ.....	5
1.1	Лабораторная работа №6.....	5
1.2	Лабораторная работа №7.....	5
1.3	Лабораторная работа №8.....	6
1.4	Лабораторная работа №9.....	6
1.5	Лабораторная работа №10.....	6
2	Краткие теоретические сведения по операторам языка MySQL.....	7
2.1	Администрирование баз данных .....	7
2.2	Файлы журналов MySQL .....	25
2.3	Добавление новых функций в MySQL .....	33
2.4	Добавление новых процедур в MySQL .....	47
3	Введение в ODBC.....	54
3.1	Введение в ODBC .....	54
3.2	Что такое ODBC.....	54
3.3	Как ODBC стандартизирует доступ к базе данных.....	55
3.4	Архитектура (My)ODBC .....	56
3.5	Типы ODBC-драйверов MySQL .....	57
3.6	Где взять MyODBC.....	58
3.7	Как установить MyODBC .....	58
3.8	Поддерживаемые платформы.....	65
3.9	Список рассылки MyODBC .....	66
3.10	Поддержка MyODBC.....	66
3.11	Как сообщать об ошибках и проблемах с MyODBC.....	67
3.12	Программы, точно работающие с драйвером MyODBC .....	69
3.13	Базисные шаги прикладной программы MyODBC .....	77
3.14	Настройка MyODBC DSN.....	77
3.15	Параметры подключения .....	81
3.16	Связь с сервером MySQL .....	94



## **1 Задания для лабораторных работ**

Для успешного выполнения лабораторных работ необходимо выполнить все этапы по порядку. Отчет о лабораторной работе должен содержать листинги всех команд, которые вы использовали для реализации каждого этапа.

### **1.1 Лабораторная работа №6. Индексация данных**

1. Создайте индексы для полей, участвующих в запросах из лабораторной работы №5;
2. Повторите запросы, заprotoколируйте время выполнения запросов
3. Сравните результаты, сделайте выводы;

### **1.2 Лабораторная работа №7. Хранимые процедуры**

1. Создайте для каждого запроса хранимые процедуры

### **1.3 Лабораторная работа №8. Разграничение прав пользователей**

1. Выделите пользователей базы данных и их роли (не менее 5 групп);
2. Создайте типовые учетные записи для каждой группы пользователей
3. Создайте хранимые процедуры для создания пользователей каждой из групп

### **1.4 Лабораторная работа №9. Резервирование и восстановление**

1. Зарезервируйте базу данных
2. Очистите Базу данных
3. Восстановите базу данных
4. Создайте хранимую процедуру, для автоматического резервирования базы данных;

### **1.5 Лабораторная работа №10. Репликация**

1. Установите локальную копию сервера Баз Данных
2. Настройте его ведомым
3. Осуществите репликацию баз данных

### **1.6 Лабораторная работа №11. Транзакции**

1. Опишите транзакции, на чтение, запись и изменение данных. Необходимо проверять целостность данных в каждый момент времен

## 2 Краткие теоретические сведения по операторам языка MySQL

### 2.1 Администрирование баз данных

#### 2.1.1 Синтаксис OPTIMIZE TABLE

```
OPTIMIZE TABLE tbl_name[,tbl_name]...
```

Команда OPTIMIZE TABLE должна использоваться после удаления большей части таблицы или если в таблице было внесено много изменений в строки переменной длины (таблицы, в которых есть столбцы VARCHAR, BLOB или TEXT). Удаленные записи поддерживаются при помощи связного списка, и последующие операции INSERT повторно используют позиции старых записей. Чтобы перераспределить неиспользуемое пространство и дефрагментировать файл данных, можно воспользоваться командой OPTIMIZE TABLE.

На данный момент команда OPTIMIZE TABLE работает только с таблицами MyISAM и BDB. Для таблиц BDB команда OPTIMIZE TABLE выполняет ANALYZE TABLE. См. раздел See Раздел 4.5.2, «Синтаксис команды ANALYZE TABLE».

Можно применить OPTIMIZE TABLE к таблицам других типов, запустив mysqld с параметром --skip-new или --safe-mode, но в этом случае OPTIMIZE TABLE лишь только выполняет ALTER TABLE.

Команда OPTIMIZE TABLE работает следующим образом:

Если в таблице есть удаленные или разделенные строки, восстанавливает таблицу.

Если индексные страницы не отсортированы - сортирует их.

Если статистические данные не обновлены (и восстановление нельзя осуществить путем сортировки индексов), обновляет их.

Команда OPTIMIZE TABLE для MyISAM представляет собой эквивалент выполнения `myisamchk --quick --check-only-changed --sort-index --analyze` над таблицей.

Обратите внимание: во время работы `OPTIMIZE TABLE` таблица заблокирована!

### 2.1.2 Синтаксис `ANALYZE TABLE`

```
ANALYZE TABLE tbl_name[,tbl_name...]
```

Анализирует и сохраняет распределение ключей для таблицы. Во время проведения анализа таблица заблокирована для чтения. Эта функция работает для таблиц MyISAM и BDB.

Данная команда является эквивалентом выполнения `myisamchk` -а для таблицы.

Сохраненное распределение ключей в MySQL используется для принятия решения о том, в каком порядке следует связывать таблицы, когда для связывания используются не константы, а другая база.

Эта команда выдает таблицу со следующими столбцами:

Table	Имя таблицы
Op	Всегда ``analyze"
Msg_type	Одно из значений status, error, info или warning.
Msg_text	Сообщение.

Просмотреть сохраненное распределение ключей можно при помощи команды `SHOW INDEX`. See Раздел 4.5.6.1, «Получение информации по базам данных, таблицам, столбцам и индексам».

Если таблица не изменялась с момента предыдущего запуска команды `ANALYZE TABLE`, повторный анализ таблицы проводиться не будет.

### 2.1.3 Синтаксис команды `FLUSH`

```
FLUSH flush_option [,flush_option] ...
```

Команда `FLUSH` применяется для очистки части кэша, используемого MySQL. Для запуска `FLUSH` необходимо обладать привилегиями `RELOAD`.

Параметр `flush_option` может быть одним из следующих:



- **HOSTS** Производится очистка таблиц кэша удаленных компьютеров. Сброс таблиц удаленного компьютера следует производить, если один из удаленных компьютеров изменил IP-адрес или если было получено сообщение об ошибке Host ... is blocked. Если во время соединения с сервером MySQL происходит больше ошибок подряд, чем указано в max\_connect\_errors для определенного удаленного компьютера, то MySQL предполагает, что что-то не в порядке, и блокирует последующие попытки установления соединения со стороны этого удаленного компьютера. Сброс таблиц удаленного компьютера позволяет снова попытаться установить соединение. See Раздел A.2.4, «Ошибка Host '...' is blocked». Чтобы это сообщение об ошибке не появлялось, запустите mysqld с параметром -O max\_connect\_errors=999999999.

- **DES\_KEY\_FILE** Производится перезагрузка ключей DES из файла, указанного параметром --des-key-file, при запуске сервера.

- **LOGS** Закрываются и повторно открывается все файлы журналов. Если файл журнала обновлений или файл бинарного журнала был указан без расширения, номер расширения файла журнала будет увеличен на единицу относительно предыдущего файла. Если в имени файла было указано расширение, MySQL закроет и повторно откроет файл журнала обновлений. See Раздел 4.9.3, «Журнал обновлений (update)». Эти действия аналогичны отправке сигнала SIGHUP на сервер mysqld.

- **PRIVILEGES** Производится перезагрузка привилегий из таблиц привилегий в базе данных mysql.

- **QUERY CACHE** Производится дефрагментация кэша запросов, чтобы эффективнее использовать его память. Эта команда не удаляет запросы из кэша, как команда RESET QUERY CACHE.

- **TABLES** Закрываются все открытые таблицы и принудительно закрываются все используемые таблицы. Также сбрасывается кэш запросов.

- **[TABLE | TABLES] tbl\_name [,tbl\_name...]** Производится сброс только указанных таблиц.

- **TABLES WITH READ LOCK** Закрываются все открытые таблицы и блокируется доступ для чтения всех таблиц для всех баз данных, пока не будет запущена команда **UNLOCK TABLES**. Это очень удобный способ создавать резервные копии, если у вас файловая система наподобие Veritas, которая может обеспечить моментальные снимки данных в режиме реального времени.

**STATUS** Большинство переменных состояния сбрасываются в нуль. Эту команду необходимо использовать при отладке запроса.

**USER\_RESOURCES** Все ресурсы пользователя сбрасываются в нулевое значение. Это позволяет заблокированному пользователю подсоединиться еще раз. See Раздел 4.3.6, «Ограничение ресурсов пользователя».

Ко всем приведенным выше командам можно получить доступ при помощи утилиты **mysqladmin**, используя команды **flush-hosts**, **flush-logs**, **reload** или **flush-tables**.

Рекомендуется также ознакомиться с командой **RESET**, которая применяется с репликацией. See Раздел 4.5.4, «Синтаксис команды **RESET**».

#### 2.1.4 Синтаксис команды **RESET**

```
RESET reset_option [,reset_option] ...
```

Команда **RESET** используется для очистки. Кроме того, она также действует как более сильная версия команды **FLUSH**. See Раздел 4.5.3, «Синтаксис команды **FLUSH**».

Чтобы запустить команду **RESET**, необходимо обладать привилегиями **RELOAD**.

- **MASTER** Удаляет все бинарные журналы, перечисленные в индексном файле, обнуляет значения индексного файла **binlog**. В версиях до 3.23.26 - **FLUSH MASTER (Master)**

- **SLAVE** Сбрасывает положение репликации подчиненного компьютера в журналах головного компьютера. В версиях до 3.23.26 эта команда называлась **FLUSH SLAVE (Slave)**

- **QUERY CACHE** Удаляет все результаты запросов из кэша запросов.

### 2.1.5 Синтаксис KILL

`KILL thread_id`

Каждое соединение с `mysqld` запускается в отдельном потоке. При помощи команды `SHOW PROCESSLIST` можно просмотреть список запущенных потоков, а при помощи команды `KILL thread_id` - удалить поток.

Если у вас есть привилегия `PROCESS`, можно просмотреть все потоки. Обладая привилегией `SUPER`, можно удалять любые потоки. В противном случае можно просматривать и удалять только свои собственные потоки.

Для просмотра и удаления потоков можно также применять команды `mysqladmin processlist` и `mysqladmin kill`.

При использовании команды `KILL` для потока устанавливается специальный флаг `kill flag`.

В большинстве случаев удаление потока занимает некоторое время, поскольку этот флаг проверяется с определенным интервалом.

В циклах `SELECT`, `ORDER BY` и `GROUP BY` флаг проверяется только после считывания блока строк. Если установлен флаг удаления, то выполнение оператора будет отменено.

При выполнении команды `ALTER TABLE` флаг удаления проверяется перед считыванием каждого блока строк из исходной таблицы. Если флаг установлен, то выполнение команды отменяется и временная таблица удаляется. При выполнении команд `UPDATE` и `DELETE` флаг удаления проверяется после каждого считывания блока, а также после каждого обновления или удаления строки. Если флаг удаления установлен, то

выполнение оператора отменяется. Обратите внимание: если не используются транзакции, то отменить изменения будет невозможно!

GET\_LOCK() будет отменен при помощи NULL.

Поток INSERT DELAYED быстро сбросит все строки, которые он содержит в памяти и будет удален.

Если поток находится в заблокированной таблице (состояние: Locked), то блокировка таблицы будет быстро отменена.

Если поток ожидает освобождения дискового пространства в запросе write, запись будет отменена с выдачей сообщения о переполнении диска.

### 2.1.6 Синтаксис SET и SHOW

SET позволяет устанавливать переменные и опции.

SHOW имеет множество различных форм, которые представляют информацию о базах данных, таблицах, столбцах, а также информацию о состоянии сервера. В данном разделе описаны следующие синтаксические формы:

```
SHOW [FULL] COLUMNS FROM имя_таблицы [FROM
имя_базы_данных] [LIKE 'шаблон*']
SHOW CREATE DATABASE имя_базы_данных
SHOW CREATE TABLE имя_таблицы
SHOW DATABASES [LIKE 'шаблон' ]
SHOW [STORAGE] ENGINES
SHOW ERRORS [LIMIT [смещение,] row_count]
SHOW GRANTS FOR пользователь
SHOW INDEX FROM имя_таблицы [FROM имя_базы_данных]
SHOW INNODB STATUS ~*~
SHOW [BDB] LOGS
SHOW PRIVILEGES
SHOW [FULL] PROCESSLIST
```

```

SHOW STATUS [LIKE 'шаблон']
SHOW TABLE STATUS [FROM имя^базы данных] [LIKE
'шаблон']
SHOW [OPEN] TABLES [FROM ша_базы_данных] [LIKE 'шаблон']
SHOW [GLOBAL | SESSION] VARIABLES [LIKE 'шаблон']
SHOW WARNINGS [LIMIT [смещение,] row count]

```

Если синтаксис данного оператора SHOW включает часть LIKE 'шаблон', то 'шаблон' — это строка, которая может содержать символы SQL-шаблонов '\*' и '\*\_'. Шаблон (pattern) применяется для ограничения вывода только соответствующими значениями.

Отметим, что существуют и другие формы этих операторов, описанные в других местах:

- Оператор SET PASSWORD - для присвоения пароля пользовательской учетной записи
- Оператор SHOW имеет формы, предоставляющие информацию о главных и подчиненных серверах в репликации:

```

SHOW BINLOG EVENTS
SHOW MASTER LOGS
SHOW MASTER STATUS
SHOW SLAVE HOSTS
SHOW SLAVE STATUS

```

## Синтаксис SET

```

SET присваивание_переменной [, присваивание_переменной]
...
присваивание_переменной:
имя_пользовательской_переменной = выражение |
[GLOBAL|SESSION] имя_системной_переменной - выражение |
@@[global.|session.] тш_системной_переменной = выражение

```

Оператор SET устанавливает значение различным типам переменных, которые влияют на работу сервера и клиента. Он может применяться для присвоения значений пользовательским и системным переменным.

В MySQL 4.0.3 были добавлены опции GLOBAL и SESSION, а также была обеспечена возможность наиболее важным системным переменным изменяться динамически в процессе работы.

В более старых версиях MySQL использовалась опция SET OPTION вместо SET, но на данный момент она устарела. Просто опускайте слово OPTION.

Следующие примеры показывают различный синтаксис, который используется для установки переменных.

Пользовательская переменная записывается как %имя\_переменной и может быть установлена следующим образом:

SET @имя\_переменной = выражение;

К системным переменным в операторе SET можно обращаться как к имя\_переменной. Имени может предшествовать необязательное слово GLOBAL или @@global. для явного указания, что это глобальная переменная, либо SESSION или @@session. - что это переменная сеанса. LOCAL и @@local. - синонимы SESSION И @@session..

Если не указан ни один модификатор, SET устанавливает переменные сеанса.

Синтаксис @@имя переменной для системных переменных поддерживается MySQL с целью обеспечения совместимости с некоторыми другими системами баз данных.

При установке нескольких системных переменных в одном операторе, последняя опция GLOBAL или SESSION применяется для переменных, для которых модификатор не указан.

```
SET sort_buffer_size=10000;
SET @@local.sort_buffer_size=10000;
SET GLOBAL sort_buffer_size=100000Q, SESSION
sort_buffer_size=1000000;
```

```
SET @@sort Jauffer_size=1000000;
SET @@global.sort_buffer_size=1000000,
@@local.sort_buffer_size=1000000;
```

Если вы устанавливаете системную переменную, используя SESSION (по умолчанию), ее значение остается в силе до конца сеанса либо до того момента, когда ей будет присвоено другое значение. Если вы устанавливаете системную переменную, указав слово GLOBAL, что требует привилегии SUPER, ее значение запоминается и используется в новых соединениях до тех пор, пока сервер не будет перезапущен. Если вы хотите сделать установку переменной неизменяемой, вы должны поместить ее в файл опций.

Чтобы предотвратить неправильное использование, MySQL генерирует ошибку, если вы применяете SET GLOBAL с переменной, которую можно использовать только с SET SESSION, либо если вы не указываете GLOBAL, устанавливая значение глобальной переменной.

Если вы хотите установить переменную сеанса в значение GLOBAL или значение GLOBAL в заранее скомпилированное в MySQL значение по умолчанию, можете устанавливать их в DEFADLT. Например, следующие два оператора идентичны и устанавливают переменную сеанса max\_join\_size в глобальное значение:

```
SET max_join_size=DEFAULT,-
SET @@session.maxjoin_size=@0global.max_join_size;
```

Список большинства системных переменных можно получить оператором SHOW variables. См. раздел 6.5.3.19. Чтобы получить отдельную переменную по имени, или список переменных с именами, соответствующими шаблону, воспользуйтесь конструкцией LIKE:

```
SHOW VARIABLES LIKE 'maxjoin^size';
SHOW GLOBAL VARIABLES LIKE 'maxjoin^size';
```

Вы также можете получить значение специфической переменной с помощью синтаксиса @@ [global. | local. ] имя\_переменной в операторе SELECT:

```
SELECT @@max_join_size, @@global.raax_join_size;
```

Когда вы запрашиваете переменную в операторе SELECT имя\_переменной (то есть, без указания global., session., local.), MySQL возвращает значение SESSION, если оно существует, и global - в противном случае.

В следующем списке представлены переменные, которые имеют нестандартный синтаксис либо не приводятся в списке системных переменных в книге MySQL. Руководство администратора (М. : Издательский дом "Вильямс", 2005, ISBN 5-8459-0805-1). Несмотря на то что эти переменные не отображаются через SHOW VARIABLES, вы можете получить их значения с помощью оператора SELECT (за исключением CHARACTER SET или NAMES). Например:

```
SELECT @@AUTOCOMMIT;
```

- **AUTOCOMMIT** = [0 | 1]. Устанавливает режим автоматического завершения транзакций. Если установлено значение 1, все изменения в данных таблиц вступают в силу немедленно. Если установлено значение 0, вы обязаны завершить транзакцию явно оператором COMMIT либо отменить ее с помощью ROLLBACK. Если вы переключаете AUTOCOMMIT с 0 на 1, MySQL выполняет автоматический COMMIT для всех активных транзакций. Другой способ начать транзакцию - это использовать оператор START TRANSACTION или BEGIN. См. раздел 6.4.1.

- **BIGTABLES** = [0 | 1]. Если установлена в 1, все временные таблицы сохраняются на диске вместо того, чтобы сохраняться в памяти. Это немного медленнее, однако при выполнении оператора SELECT, который требует создания большой временной таблицы, не возникает ошибка The table



имя\_таблицы is full (Таблица имя\_таблицы переполнена). Значением по умолчанию для новых соединений является 0 (размещать временные таблицы в памяти). Начиная с MySQL 4.0, вам, как правило, не придется устанавливать эту переменную, поскольку MySQL сам в случае необходимости преобразует временные таблицы в памяти в таблицы на диске. Ранее эта переменная называлась SQL\_BIGFABLES.

- CHARACTER SET {ишг\_набора\_символов | DEFAULT}. Это передает все строки от клиента и обратно с заданным отображением. До версии MySQL 4.1 единственным допустимым значением для имя\_набора\_^символов было cp1551\_koiS, однако вы можете добавить новые варианты отображения, редактируя файл sql/convert.cc исходного дистрибутива MySQL. Начиная с MySQL 4.1, SET CHARACTER SET устанавливает три системных переменных: character\_set\_client и character\_set\_results устанавливаются в заданный набор символов, а character\_set\_connection - в значение character\_set\_database.

Отображение по умолчанию может быть восстановлено указанием default. Следует отметить, что синтаксис SET CHARACTER SET отличается от принятого для установки других большинства других опций.

- FOREIGN\_KEY\_CHECKS = {0 | 1}. Если установлена в 1 (по умолчанию), ограничения внешних ключей для InnoDB контролируются сервером. Если установлено в 0, игнорируются. Отключение проверок внешних ключей может оказаться удобным для перезагрузки таблиц InnoDB в порядке, отличающемся оттого, который требуют их отношения родительский/дочерний. Эта переменная была добавлена в MySQL 3.23.52.

- IDENTITY = значение. Эта переменная - синоним LAST\_INSERT\_ID. Введена с целью достижения совместимости с другими базами данных. Начиная с MySQL 3.23.25, вы можете получить ее значение с помощью SELECT ^IDENTITY. Начиная с MySQL 4.0.3, вы также можете установить ее значение через SET IDENTITY.

- INSERT\_ID = значение. Устанавливает значение, которое будет использовано следующим оператором INSERT или ALTER TABLE для вставки

в столбец AUTO\_INCREMENT. В основном это используется с бинарным журналом.

- `LAST_INSERT_ID = значение`. Устанавливает значение, которое будет возвращено `LAST_INSERT_ID()` в операторах, обновляющих таблицу. Установка этой переменной не обновляет значения, возвращаемого функцией C API `mysql_insert_id()`.

```
NAMES { 'ишя_набора_символов' | DEFAULT }
```

- `SET NAMES` устанавливает три системные переменные сеанса: `character_set_client`, `character_set_connection` и `character_set_result` в указанный набор символов. Отображение по умолчанию может быть восстановлено, если указать `DEFAULT`. Обратите внимание, что синтаксис `SET NAMES` отличается от принятого для установки большинства других опций. Этот оператор появился в MySQL 4.1.0.

- `SQL_AUTO_IS_NULL = 10 | I`. Если установлена в `I` (по умолчанию), вы можете найти последнюю вставленную в таблицу строку, которая содержит столбец `AUTO_INCREMENT`, используя следующую конструкцию:

```
WHERE столбец_auto_increment IS NULL
```

Это поведение характерно для некоторых ODBC-программ, таких как Access. `SQL_AUTO_IS_NULL` была добавлена в MySQL 3.23.52.

- `SQL_BIG_SELECTS = {0 | 1}`. Если установлена в `0`, MySQL прерывает выполнение операторов `SELECT`, которые предположительно будут выполняться долго (то есть операторов, для которых оптимизатор ожидает, что количество проверяемых строк превысит значение `max_join_size`). Это удобно, когда применяются нерациональные условия в конструкции `WHERE`. Значение по умолчанию этой переменной для новых соединений равно `1`, что разрешает выполнять любые операторы `SELECT`. Если вы устанавливаете значение

системной переменной `max_join_size` в значение, отличное от `DEFAULT`, `SQL_BIG_SELECTS` принимает значение 0.

- `SQL_BUFFER_RESULT = {0 | 1}`

`SQL_BUFFER_RESULT` принуждает операторы `SELECT` помещать результаты во временные таблицы. Это помогает MySQL освобождать табличные блокировки пораньше и может быть выгодно в случаях, когда требуется много времени для отправки результатов клиентам. Переменная была добавлена в MySQL 3.23.13.

- `SQL_LOG_BIN = {0 | 1}`. Если установлена в 0, никакой бинарной регистрации для клиента не выполняется. Клиент должен иметь привилегию `SUPER` для установки этой опции. Переменная была добавлена в MySQL 3.23.16.

- `SQL_LOG_OFF = {0 | 1}`. Если установлена в 1, регистрация запросов клиента в общем журнале не ведется. Клиент должен иметь привилегию `SUPER` для установки этой опции.

- `SQL_LOG_UPDATE = {0 | 1}`. Если установлена в 0, регистрация обновлений клиента в журнале не ведется. Клиент должен иметь привилегию `SUPER` для установки этой опции. Переменная была добавлена в MySQL 3.23.5. Начиная с MySQL 5.0.0, считается устаревшей и отображается на `SQL_LOG_BIN`.

- `SQL_QUOTE_SHOW_CREATE = {0 | 1}`. Если установлена в 1, `SHOW CREATE TABLE` выводит имена таблиц и столбцов в кавычках. Если установлена в 0, кавычки отключаются. По умолчанию эта опция включена, чтобы репликация работала с таблицами и столбцами в кавычках. Переменная появилась в MySQL 3.23.26. См. раздел 6.5.3.6.

- `SQL_SAFE_UPDATES = {0 | 1}`. Если установлена в 1, MySQL прерывает операторы `UPDATE` и `DELETE`, у которых не используются ключевые значения в конструкции `WHERE`. Это позволяет перехватывать операторы `UPDATE` и `DELETE`, у которых неправильно используются ключи и

которые могут непреднамеренно изменить или удалить большое количество строк. Переменная была добавлена в MySQL 3.22.32.

- `SQL_SELECT_LIMIT = {значение | DEFAULT}`. Максимальное количество строк, которые может вернуть оператор `SELECT`. Значение по умолчанию для новых подключений - "неограниченное" (unlimited). Если вы изменяете это ограничение, значение по умолчанию может быть восстановлено присвоением `SQL_SELECT_LIMIT` значения `DEFAULT`.

- `SQL_WARNINGS = {0 | 1}`. Эта переменная устанавливает режим вывода предупреждений однострочными операторами `INSERT`. По умолчанию имеет значение 0. Установите ее равной 1, чтобы генерировать строки предупреждений. Переменная была добавлена в MySQL 3.22.11.

- `TIMESTAMP = 1` (значение\_ timestamp | DEFAULT). Устанавливает текущее время для клиента. Применяется для получения оригинальной временной метки при использовании бинарного журнала для восстановления строк. `timestamp` должно быть временной меткой в формате Unix, а не временной меткой MySQL.

- `UNIQUE_CHECKS = (0 | 1)`. Если установлена в 1 (как по умолчанию), выполняется проверка уникальности вторичных индексов в таблицах InnoDB. Если установлена в 0, никаких проверок уникальности не выполняется. Переменная появилась в MySQL 3.23.52.

```
SHOW CHARACTER SET [LIKE 'шаблон']
```

Оператор `SHOW CHARACTER SET` выводит список всех допустимых символьных наборов. Он принимает необязательную конструкцию `LIKE` для указания шаблона имен наборов символов. Например:

```
SHOW CHARACTER SET LIKE 'latin%';
```

- `SHOW COLLATION [LIKE 'шаблон']`

Вывод `SHOW COLLATION` включает все допустимые порядки сопоставления наборов символов. Принимает необязательную конструкцию

LIKE для указания шаблона наименований порядка сопоставления. Например:

```
SHOT COLLATION LIKE 'latin1%';
```

Столбец Default показывает, является ли данный порядок сопоставления порядком по умолчанию для данного набора символов. Compiled отражает, является ли данный набор символом заранее скомпилированным на сервере. Sortlen имеет отношение к объему памяти, необходимому для сортировки строк, представленных в данном наборе символов.

SHOW COLLATION доступен, начиная с версии MySQL 4.1.0.

- SHOW [FULL] COLUMNS FROM имя^таблицы [FROM имя\_базы\_данных] [LIKE 'шаблон']

- SHOW COLUMNS выводит список столбцов заданной таблицы. Если типы столбцов отличаются от тех, что были заданы оператором CREATE TABLE, имейте в виду, что MySQL иногда изменяет типы столбцов при создании или изменении структуры таблицы. Условия, при которых это происходит, описаны в разделе 6.2,5,2.

Ключевое слово full может использоваться, начиная с MySQL 3.23.32 и выше. Оно включает в вывод информацию о привилегиях, которые вы имеете для доступа к данным столбцам. Начиная с MySQL 4.1, слово FULL также включает вывод всех комментариев о столбцах.

Вы можете использовать имя\_базы\_данных.имя\_тайтл как альтернативу синтаксису имя\_таблиц FROM имя\_базы\_данных. Следующие два оператора эквивалентны:

- SHOW COLUMNS FROM rcytable FROM mydb; mysql> SHOW COLUMNS FROM mydb.mytable;

- SHOW FIELDS — это синоним SHOW COLUMNS. Вы также можете вывести список столбцов с помощью команды mysqlshow имя\_\_базы^данных имя\_таблицы.

Оператор DESCRIBE выводит информацию, подобную SHOW COLUMNS. См. раздел 6.3.1.

- **SHOW CREATE DATABASE** имя\_бэы\_даняьк

Выводит оператор **CREATE DATABASE**, который создаст указанную базу данных. Добавлен в MySQL 4.1.

```
SHOW CREATE DATABASE test\G
```

Выводит оператор **CREATE TABLE**, который создаст указанную таблицу.

```
CREATE TABLE t (
    id INT (11) default NULL auto_increment,
    s char (60) default NULL,
    PRIMARY KEY (id) ) TYPE=MyISAM
```

**SHOW CREATE TABLE** заключает в кавычки имена таблицы и столбцов в соответствии со значением опции **SQL\_QUOTE\_SHOW\_CREATE**. См.

- **SHOW DATABASES** [LIKE •шаблон1]

**SHOW DATABASES** выводит список баз данных на хосте сервера MySQL. Вы также можете получить этот список с помощью команды **mysqlshow**. Начиная с MySQL 4.0.2, вы увидите только те базы данных, для доступа к которым имеете какие-либо привилегии, если только у вас нет глобальной привилегии **SHOW DATABASES**.

Если сервер запущен с опцией **—skip-show-database**, вы не сможете использовать этот оператор вообще, если не имеете привилегии **SHOW DATABASES**.

## **Синтаксис SHOW ENGINES**

```
SHOW [STORAGE] ENGINES
```

**SHOW ENGINES** выводит информацию о состоянии механизмов хранения. Это особенно удобно для проверки того, какие механизмы хранения поддерживаются сервером, или для того, чтобы увидеть, какой механизм

используется по умолчанию. Оператор реализован в MySQL 4,1,2. Существует устаревший синоним-SHOW TABLE TYPES.

```
SHOW ENGINES\G
```

Значение `Support` показывает, поддерживается ли данный механизм сервером, и какой из них является механизмом по умолчанию. Например, если сервер запущен с опцией `—def ault-table-type=InnoDB`, то значением `Support` для `InnoDB` будет `DEFAULT`.

### **Синтаксис SHOW ERRORS**

```
SHOW ERRORS      [LIMIT      [смещение, ]      количество_строк]
SHOW COUNT (*)   ERRORS
```

Этот оператор похож на `SHOW WARNINGS`, с тем отличием, что вместо отображения ошибок, предупреждений и примечаний, он выводит только ошибки. `SHOW ERRORS` доступен, начиная с версии MySQL 4.1.0.

Конструкция `LIMIT` имеет тот же синтаксис, что и в операторе `SELECT`.

Оператор `SHOW COUNT (*) ERRORS` отображает количество ошибок.

Вы также можете получить это количество из переменной `errorcount`:

```
SHOW COUNT!*) ERRORS; SELECT @@error_count;
```

### **Синтаксис SHOW GRANTS**

```
SHOW GRANTS FOR пользователь
```

Этот оператор выводит операторы `GRANT`, которые должны быть выполнены для дублирования набора привилегий пользователя MySQL.

```
SHOW GRMTTS FOR 'root'@ 'localhost';
```

Вернет

```
GRANT ALL PRIVILEGES ON *.* TO 'root'@'localhost' WITH
GRANT OPTION
```

Начиная с версии MySQL 4.1.2, чтобы получить список привилегий текущего сеанса, можно воспользоваться любым из перечисленных ниже операторов:

```
SHOW GRANTS, -
SHOW GRANTS FOR CURRENT_USER;
SHOW GRANTS FOR CURRENT_USER();
```

До MySQL 4.1.2 узнать, какой пользователь был аутентифицирован в текущем сеансе, можно было с помощью функцию CURRENT\_USER() (новая в MySQL 4.0.6). Затем полученное значение нужно было указать в операторе SHOW GRANTS. См. раздел 5.8.3.

SHOW GRANTS доступен, начиная с MySQL 3.23.4.

### **Синтаксис SHOW INDEX**

```
SHOW INDEX FROM имя_таблицы [FROM имя_базы_данных]
```

show index возвращает информацию об индексах таблицы в формате, подобном тому, что выдает вызов SQLStatistics в ODBC. SHOW INDEX возвращает следующие поля:

Table. Имя таблицы.

Non\_unique. 0, если индекс не может иметь дублированных ключей, 1 - если может.

Key\_name. Имя индекса.

Seq\_in\_index. Порядок столбца в ключе индекса, начиная с 1.

Column\_name. Имя столбца.

Collation. Как столбец сортируется в индексе. В MySQL может иметь значения 'A' (ascending-по возрастанию) или NULL (не сортировано).

Cardinality. Количество уникальных значений в индексе. Это



обновляется оператором `ANALYZE TABLE` или командой `myisamchk -a`. Cardinality рассчитывается на базе статистики, хранится в виде целого числа, поэтому нет необходимости в большой точности для маленьких таблиц.

**Sub\_part.** Количество проиндексированных символов столбца, если ключ построен на части столбца. Если вся столбец является ключом индекса, принимает значение `NULL`.

**Packed.** Показывает, упакован ли индекс. Если нет - `NULL`.

**Null.** Содержит `YES`, если столбец допускает значение `NULL`.

**Index\_type.** Используемый метод индексации (`BTREE`, `FULLTEXT`, `HASH`, `RTREE`).

**Comment.** Различные замечания. До MySQL 4.0.2, когда был добавлен столбец `Index type`, `Comment` показывает, является ли индекс `FULLTEXT`.

Столбцы `Packed` и `Comment` появились в версии MySQL 3.23.0. Столбцы `Null` и `Index_type` были добавлены в MySQL 4.0.2.

В качестве альтернативы синтаксиса `имя_таблицы FROM имя_базы_данных` можно использовать `имябазы^ данных. имя_таблицы`. Следующие два оператора эквивалентны:

```
SHOW INDEX FROM mytable FROM mydb;
```

```
SHOW INDEX FROM mydb.mytable;
```

**SHOW KEYS** - синоним для **SHOW INDEX**. Список индексов можно также получить по

## 2.2 Файлы журналов MySQL

В MySQL имеется несколько журналов, позволяющих узнать, что происходит внутри `mysqld`:

Журнал	Описание
--------	----------

**Журнал ошибок** В нем хранятся ошибки запуска, работы или завершения работы `mysqld`.

**Журнал isam** В нем хранится информация обо всех изменениях таблиц ISAM. Используется только при отладке кода `isam`.

Общий журнал запросов В нем хранится информация об установленных соединениях и выполненных запросах.

Журнал обновлений log В нем хранятся все команды, меняющие данные; в скором времени выйдет из употребления

Бинарный журнал обновлений В нем хранятся все меняющие что-либо команды. Используется для репликации

Журнал медленных запросов В нем хранятся все запросы, на выполнение которых ушло больше времени, чем указано в переменной `long_query_time` (или запросы, не использовавшие индексов).

Все файлы журналов хранятся в каталоге с данными `mysqld`. С помощью команды `FLUSH LOGS` можно заставить `mysqld` открыть файлы журналов снова (или - в некоторых случаях - переключиться на новый файл).

#### 4.9.1. Журнал ошибок

Журнал ошибок содержит информацию о том, когда запускается и останавливается `mysqld`, а также все критические ошибки, обнаруженные в процессе работы.

В нем содержится информация о запуске и завершении работы `mysqld`, а также обо всех серьезных ошибках, возникших во время работы. Если произойдет неожиданное аварийное завершение работы, и `safe_mysqld` придется перезапустить `mysqld`, `safe_mysqld` внесет в этот файл соответствующую запись. Кроме того, в этот журнал заносится предупреждение в том случае, если `mysqld` обнаружит таблицу, нуждающуюся в автоматической проверке или исправлении.

Все ошибки `mysqld` записывает в `stderr`, который сценарий `safe_mysqld` перенаправляет в файл с именем `'hostname'.err` (в Windows `mysqld` сохраняет его в каталоге `\mysql\data\mysql.err`).

В некоторых ОС в журнал включается распечатка части стека погибшего `mysqld`. С помощью этой информации можно определить причину сбоя

Начиная с MySQL 4.0.10 можно указать, где именно `mysqld` должен сохранять журнал ошибок, с помощью опции `--log-error[=filename]`. Если имя

файла не задается, то тогда `mysqld` будет использовать `mysql-data-dir/'hostname'.err` на Unix и `\mysql\data\mysql.err` на windows.

Если вы выполняете `FLUSH LOGS` старый файл будет сохранен с префиксом `--old` и `mysqld` создаст новый пустой журнал.

На старых версиях MySQL журнал ошибок велся скриптом `mysqld_safe`, который перенаправлял вывод в файл `'hostname'.err`. В старых версиях можно было изменить имя этого файла опцией `--err-log=filename`.

Если вы не указываете `--log-error` или используете опцию `--console`, то ошибки будут выводиться на `stderr` (на терминал).

На Windows вывод всегда пишется в `.err`-файл если `--console` не была указана.

### 2.2.1 Общий журнал запросов

Если вы хотите знать обо всем, что происходит с `mysqld`, нужно запустить систему с ключом `--log[=file]`. После этого информация обо всех соединениях и запросах будет записываться в файл журнала (по умолчанию ему дается имя `'hostname'.log`). Этот журнал может оказаться полезным, если вы подозреваете наличие ошибки в клиентском ПО и хотите выяснить, что, по мнению `mysqld`, клиент передал базе.

Старые версии скрипта `mysql.server` (с MySQL 3.23.4 по 3.23.8) передавали `safe_mysqld` опцию `--log` (включить общий журнал запросов). Если вам нужна большая производительность при запуске MySQL в промышленной эксплуатации, вы можете удалить опцию `--log` из `mysql.server` или поменять ее на `--log-bin`.

Записи в журнал заносятся по мере получения `mysqld` запросов. Порядок их занесения может отличаться от порядка выполнения команд. В этом и заключается основное отличие данного журнала от журналов обновлений и бинарных журналов, в которые информация заносится по мере выполнения запросов, но до отмены блокировок.

### 2.2.2 Журнал обновлений (update)

Обратите внимание: журнал обновлений (update) заменен бинарным журналом (binary) (see Раздел 4.9.4, «Бинарный журнал обновлений»). С этим журналом можно производить те же операции, что и с журналом обновлений.

При запуске с ключом `--log-update[=file_name]` `mysqld` создает журнал, в который заносятся все команды SQL, обновляющие данные. Если имя файла не задано, по умолчанию ему присваивается имя хоста. Если файлу присвоено имя, не содержащее пути доступа к нему, этот файл сохраняется в каталоге с данными. Если у имени `file_name` нет расширения, `mysqld` даст файлу примерно такое имя: `file_name.###`, где `###` - номер, увеличивающийся при каждом выполнении команд `mysqladmin refresh`, `mysqladmin flush-logs`, `FLUSH LOGS` или при перезапуске сервера.

Обратите внимание: чтобы вышеописанная схема могла работать, нельзя самостоятельно создавать файлы с тем же именем, что и у журнала обновлений, а также с некоторыми расширениями, которые могут быть восприняты как номер, в каталоге, использующемся для хранения этого журнала!

При запуске с ключами `--log` или `-l` `mysqld` создает общий журнал в файле с именем `hostname.log`, причем перезапуски и обновления не приводят к созданию нового файла журнала (хотя существующий при таких операциях закрывается и затем открывается вновь). В таком случае скопировать его (в Unix) можно так:

```
mv hostname.log hostname-old.log
mysqladmin flush-logs
cp hostname-old.log to-backup-directory
rm hostname-old.log
```

Журнал обновлений работает избирательно - в него попадают только те команды, которые действительно обновляют данные. Команда `UPDATE` или `DELETE`, выражение `WHERE` которой не находит совпадающих строк, в журнал не заносится - как и команды `UPDATE`, присваивающие столбцам те же

значения, которые у них были до ``обновления".

Запись в журнал осуществляется сразу по завершении работы запроса, но до того, как будут сняты блокировки. Таким образом обеспечивается уверенность в том, что журнал ведется именно в порядке выполнения запросов.

При желании обновить базу в соответствии с данными журналов обновлений можно воспользоваться следующей командой (при условии, что имена файлов журналов соответствуют форме `file_name.###`):

Эта возможность может пригодиться в случае, если возникнет необходимость (в результате серьезного сбоя) привести базу в соответствие с резервной копией и затем повторить все обновления, произошедшие с момента создания копии и до сбоя.

### 2.2.3 Бинарный журнал обновлений

Бинарный журнал обновлений в скором времени должен полностью заменить журнал обновлений, так что мы рекомендуем вам как можно скорее перейти на его использование!

Бинарный журнал содержит всю информацию, имеющуюся в журнале обновлений, в более эффективном формате. В нем имеется информация и о времени выполнения каждого обновляющего базу запроса. В нем не содержится информации о запросах, которые не изменяют данные. Если вам нужно журналировать все запросы (например для выявления проблемного запроса), вам следует использовать общий журнал запросов. See Раздел 4.9.2, «Общий журнал запросов».

Бинарный журнал используется и при репликации подчиненного сервера (slave) с головного (master) (see Раздел 4.10, «Репликация в MySQL»).

При запуске с ключом `--log-bin[=file_name]` `mysqld` создает файл журнала, в который вносятся данные обо всех обновляющих данные командах SQL. Если имя файла не задано, по умолчанию ему дается имя хоста с окончанием `-bin`. Если файлу присвоено имя, не содержащее пути доступа к нему, этот файл сохраняется в каталоге данных.

При вводе расширения в имя файла (например: `--log-bin=filename.extension`) это расширение удаляется без предупреждения.

К имени файла бинарного журнала программа `mysqld` прибавляет специальное расширение - номер, увеличивающийся при каждом выполнении команд `mysqladmin refresh`, `mysqladmin flush-logs`, `FLUSH LOGS` или перезапуске сервера. При достижении файлом журнала максимального размера, заданного в параметре `max_binlog_size`, автоматически создается новый. Все неактивные файлы бинарных журналов можно удалить командой `RESET MASTER` (see Раздел 4.5.4, «Синтаксис команды `RESET`»).

На выбор данных, записываемых в журнал, влияют следующие настройки `mysqld`:

Опция	Описание
-------	----------

<code>binlog-do-db=database_name</code>	Указывает головному серверу что он должен журналировать обновления в двоичный журнал если текущая (т.е. выбранная) база данных - это <code>'database_name'</code> . Остальные базы данных, особо не отмеченные, игнорируются. Имейте в виду, что если вы используете эту опцию, то вам следует делать обновления только в этой базе данных. (пример: <code>binlog-do-db=some_database</code> )
---	--

<code>binlog-ignore-db=database_name</code>	Заставляет master отказаться от занесения в журнал обновлений определенной базы данных (пример: <code>binlog-ignore-db=some_database</code> )
---	---

Чтобы была возможность определить, какие файлы журналов используются в данный момент, `mysqld` создает и индексный файл, содержащий имена всех находящихся в работе файлов. По умолчанию ему присваивается то же имя, что и файлу журнала, но с расширением `.index`. Имя этого файла можно изменить с помощью параметра `--log-bin-index=[filename]`.

При использовании репликации удалять старые файлы журналов не стоит до тех пор, пока вы не будете уверены в том, что они никогда не понадобятся ни одной зависимой базе. Добиться такого результата можно, запуская команду `mysqladmin flush-logs` раз в день и затем удаляя все журналы, созданные более 3 дней назад.

Работать с файлами бинарного журнала можно с помощью программы `mysqlbinlog`. С помощью программы `mysqlbinlog` можно даже считывать файлы журнала прямо с удаленного сервера MySQL!

При запуске `mysqlbinlog` с ключом `--help` на экран выводится дополнительная информация по работе с этой программой.

При работе с настройками `BEGIN [WORK]` или `SET AUTOCOMMIT=0` для резервного копирования нужно использовать бинарный журнал, а не старый журнал обновлений.

Занесение данных в бинарный журнал происходит сразу по завершении исполнения запроса, но до снятия блокировок. Таким образом обеспечивается уверенность в том, что журнал ведется именно в порядке выполнения запросов.

Updates to non-transactional tables are stored in the binary log immediately after execution.

Обновления нетранзакционных таблиц сохраняются в двоичном журнале немедленно после выполнения. Все обновления (`UPDATE`, `DELETE` или `INSERT`), изменяющие данные в транзакционных таблицах (например, BDB-таблицу) находятся в кэше до вызова `COMMIT`. В этот момент `mysqld` пишет всю транзакцию целиком в двоичный журнал перед тем, как выполнить `COMMIT`. Каждый поток при запуске будет создавать буфер размером `binlog_cache_size` для буферизации запросов. Если запрос превышает этот размер, тогда поток откроет временный файл для сохранения транзакции. Временный файл будет удален при выходе потока.

При запуске каждого потока создается буфер запросов, объем которого соответствует значению параметра `binlog_cache_size`. Если запрос не помещается в буфере, поток создаст временный файл для кэша. Временный файл удаляется по завершении работы потока.

Параметр `max_binlog_cache_size` (по умолчанию 4Гб) позволяет ограничить общий объем памяти, используемой для кэширования мультитранзакционного запроса. Если транзакция больше этого - будет произведен откат.

При использовании журнала обновлений или бинарного журнала параллельные операции вставки будут преобразованы в нормальные операции вставки в командах `CREATE ... SELECT` и `INSERT ... SELECT`. Это сделано специально - для того, чтобы обеспечить возможность создания точной копии таблиц путем объединения резервной копии с журналом.

#### 2.2.4 Журнал медленных запросов

При запуске с параметром `--log-slow-queries[=file_name]` `mysqld` создает файл журнала, в котором сохраняются данные обо всех командах SQL, на выполнение которых ушло больше времени, чем указано в значении параметра `long_query_time`. Время, уходящее на первоначальную блокировку таблиц, не включается во время исполнения запроса.

Занесение данных в журнал происходит сразу по завершении исполнения запроса и снятия блокировок. Таким образом, порядок расположения записей может отличаться от порядка выполнения запросов.

Если имя файла не задано, по умолчанию ему дается имя хоста с окончанием `-slow.log`. Если файлу присвоено имя, не содержащее пути доступа к нему, этот файл сохраняется в каталоге с данными.

Этот журнал позволяет определить запросы, на выполнение которых ушло слишком много времени, а, значит, и обнаружить основных кандидатов на оптимизацию. Конечно, при достижении журналом значительного объема эта задача усложняется. В таком случае журнал можно пропустить через команду `mysqldumpslow` и получить краткий отчет о запросах, попавших в список.

При использовании ключа `--log-long-format` на экран выводятся и запросы, не работающие с индексами (see Раздел 4.1.1, «Параметры командной строки `mysqld`»).

#### 2.2.5 Обслуживание файлов журналов

В MySQL предусмотрено наличие нескольких файлов журналов,



позволяющих следить за всеми аспектами работы системы (see Раздел 4.9, «Файлы журналов MySQL»). Правда, иногда приходится проверять, не занимают ли журналы лишнего места, и удалять ненужные.

При работе с журналами MySQL, вам, вероятнее всего, понадобится удалять их или создавать их резервные копии, и указывать MySQL записывать данные журналов в новые файлы (see Раздел 4.4.1, «Резервное копирование баз данных»).

В системе Linux (Red Hat) для этого можно использовать сценарий `mysql-log-rotate`. При установке MySQL с дистрибутива RPM этот сценарий устанавливается автоматически. Обратите внимание: использовать журнал для репликации необходимо с максимальной аккуратностью!

В других ОС вы можете самостоятельно создать небольшой сценарий для обработки журналов, запускаемый из `cron`.

Заставить MySQL создать новый файл журнала можно с помощью команды `mysqladmin flush-logs` или SQL-команды `FLUSH LOGS`. При работе с MySQL версии 3.21 пользоваться можно только командой `mysqladmin refresh`.

Эта команда выполняет следующие действия:

Если используется стандартный журнал (`--log`) или журнал медленных запросов (`--log-slow-queries`), файл журнала (`mysql.log` и `'hostname'-slow.log` по умолчанию) закрывается и открывается вновь.

Если используется журнал обновлений (`--log-update`), файл журнала закрывается, после чего создается новый файл с большим номером.

При использовании одного журнала обновлений нужно очистить журналы и перенести их старые файлы в резервную копию, а затем сделать резервную копию файла `mysql.old` и удалить его.

## 2.3 Добавление новых функций в MySQL

Есть два способа добавить новую функцию в MySQL:

Вы можете добавить функцию через механизм определяемых пользователем функций (`user-definable function`, UDF). Они добавляются

динамически, используя команды `CREATE FUNCTION` и `DROP FUNCTION`. Подробности в разделе "3.1.1 Синтаксис `CREATE FUNCTION/DROP FUNCTION`".

Вы можете добавить функцию как внутреннюю в MySQL. Такие функции компилируются прямо внутрь сервера `mysqld` и становятся доступными на постоянной основе.

Каждый метод имеет свои проблемы:

Если Вы пишете определяемую пользователем функцию, Вы должны установить объектный файл в дополнение к серверу. Если Вы компилируете Вашу функцию прямо в сервер, Вы не должны делать этого.

Вы можете добавлять UDF к двоичному дистрибутиву MySQL. Встроенные функции требуют, чтобы Вы изменили исходники.

Если Вы обновляете MySQL, Вы можете продолжать использовать Ваш предварительно установленный UDF. Для встроенных функций Вы должны повторить модификации каждый раз, когда Вы делаете апгрейд.

Независимо от метода, который Вы используете, чтобы добавить новые функции, они могут использоваться точно так же как местные функции типа `ABS()` или `SOUNDEX()`.

### 2.3.1 Синтаксис `CREATE FUNCTION/DROP FUNCTION`

```
CREATE [AGGREGATE] FUNCTION function_name RETURNS
{STRING|REAL|INTEGER}
    SONAME shared_library_name
DROP FUNCTION function_name
```

Определяемые пользователем функции (user-definable function, UDF) представляют собой способ расширить MySQL новой функцией, которая работает подобно местным (встроенным) функциям MySQL типа `ABS()` или `CONCAT()`.

`AGGREGATE` новая опция для MySQL Version 3.23. Функция с

AGGREGATE работает точно так же, как и встроенная функция GROUP, подобно SUM или COUNT().

CREATE FUNCTION сохраняет имя функции, тип и общедоступное библиотечное имя в таблице mysql.func системы. Вы должны иметь привилегии insert и delete для базы данных mysql, чтобы создавать и удалять функции.

Все активные функции перезагружаются при каждом запуске сервера, если Вы не запускаете mysqld с опцией --skip-grant-tables. В этом случае инициализация пропущена, и UDF станут недоступны. Активная функция представляет собой такую функцию, которая была загружена с помощью CREATE FUNCTION, но не была удалена через вызов DROP FUNCTION.

По поводу правил написания определяемых пользователем функций отсылаю Вас к разделу "3.1 Добавление новой функции, определяемой пользователем в MySQL". Для работы механизма UDF функции должны быть написаны на C или C++, Ваша операционная система должна поддерживать динамическую загрузку, и mysqld должен быть откомпилирован динамически (не статически).

### 2.3.2 Добавление новой функции, определяемой пользователем

Для работы механизма UDF функции должны быть написаны на C или C++, а Ваша операционная система должна поддерживать динамическую загрузку. Дистрибутив исходников MySQL включает файл sql/udf\_example.cc, который определяет 5 новых функций. Консультируйтесь с этим файлом, чтобы видеть, как работают соглашения о вызовах UDF.

Чтобы mysqld мог использовать UDF, Вы должны конфигурировать MySQL с опцией --with-mysqld-ldflags=-rdynamic. Причина этого в том, что на многих платформах (включая Linux) Вы можете загружать динамическую библиотеку (вызовом dlopen()) из статически скомпонованной программы, которая собрана с опцией --with-mysqld-ldflags=-all-static, но если Вы хотите использовать UDF, который должен обратиться к символам из mysqld (подобно примеру methaphone в sql/udf\_example.cc, который использует

default\_charset\_info), Вы должны компоновать программу с -rdynamic. Подробности на man dlopen.

Для каждой функции, которую Вы хотите использовать в инструкциях SQL, Вы должны определить соответствующую функцию на C или на C++. В обсуждении ниже имя ``xxx" используется для имени функции примера. Здесь XXX() (верхний регистр) указывает SQL-обращение к функции, и xxx() (нижний регистр) указывает C/C++-обращение к функции.

Функции, которые Вы пишете на C/C++ для реализации интерфейса с XXX():

xxx() (обязательна)

Основная функция. Это то место, где функциональный результат вычислен. Соответствие между типом SQL и типом возврата Вашей функции на C/C++ показывается ниже: SQL-тип      C/C++-тип

STRING    char \*

INTEGER long long

REAL      double

xxx\_init() (опциональна)

Функция инициализации для xxx(). Это может использоваться для:

Проверки числа параметров XXX().

Проверки, что параметры имеют требуемый тип или выдачи предписания, чтобы MySQL принудительно привел параметры к типам, которые Вы хотите иметь, когда основная функция вызвана.

Распределения любой память, требуемой для основной функции.

Определения максимальной длины результата.

Указания (для функций типа REAL) максимального количества десятичных чисел.

Указания того, может или нет результат быть NULL.

xxx\_deinit() (опционально)

Функция деинициализации для xxx(). Это должно освободить любую память, распределенную функцией инициализации.

Когда инструкция SQL вызывает XXX(), MySQL вызывает функцию инициализации xxx\_init(), чтобы позволить ей выполнить любую требуемую настройку, типа проверки параметра или распределения памяти. Если xxx\_init() возвращает ошибку, инструкция SQL будет прервана с сообщением об ошибке, причем главная и деинициализационная функции не будут вызваны, что стоит иметь в виду при распределении памяти. Иначе основная функция xxx() будет вызвана один раз для каждой строки. После того, как все строки были обработаны, вызывается функция xxx\_deinit(), так что она может выполнить требуемую очистку.

Все функции должны быть безопасны для потоков (не только основная функция, но и остальные: инициализация и деинициализация идут в поточном режиме!). Это означает, что Вам не позволят распределить любые глобальные или менять статические переменные! Если Вы нуждаетесь в памяти, Вы должны распределить ее в xxx\_init() и непременно освободить в xxx\_deinit().

### 2.3.3 3.1.2.1 Соглашения по вызову UDF

Основная функция должна быть объявлена как показано ниже. Обратите внимание, что тип возврата и параметры отличаются в зависимости от того, объявите ли Вы тип возврата функции SQL XXX() как STRING, INTEGER или REAL в вызове CREATE FUNCTION:

Для функций типа STRING:

```
char *xxx(UDF_INIT *initid, UDF_ARGS *args, char *result,
          unsigned long *length, char *is_null, char *error);
```

Для функций типа INTEGER:

```
long long xxx(UDF_INIT *initid, UDF_ARGS *args, char *is_null, char
*error);
```

Для функций типа REAL:

```
double xxx(UDF_INIT *initid, UDF_ARGS *args, char *is_null, char
*error);
```

Функции инициализации и деинициализации объявлены подобно этому:

```
my_bool xxx_init(UDF_INIT *initid, UDF_ARGS *args, char *message);
void xxx_deinit(UDF_INIT *initid);
```

Параметр `initid` передан всем трем функциям. Он указывает на структуру `UDF_INIT`, которая используется, чтобы передать информацию между функциями. Члены структуры `UDF_INIT` перечислены ниже. Функция инициализации должна заполнить любые члены, которые она желает изменить. Чтобы использовать значение по умолчанию для члена, оставьте его неизменным. Перейдем к описанию:

```
my_bool maybe_null
```

`xxx_init()` должна установить `maybe_null` в 1, если `xxx()` может возвращать `NULL`. Значение по умолчанию 1, если любой из параметров объявлен как `maybe_null`.

```
unsigned int decimals
```

Число десятичных цифр. Значение по умолчанию: максимальное количество десятичных цифр в параметрах, переданных основной функции. Например, если функции переданы 1.34, 1.345 и 1.3, значением по умолчанию будет 3, поскольку 1.345 имеет 3 десятичных цифры.

```
unsigned int max_length
```

Максимальная длина результата-строки. Значение по умолчанию отличается в зависимости от типа результата функции. Для строчных функций значение по умолчанию равно длине самого длинного параметра. Для целочисленных функций значение по умолчанию соответствует 21 цифре. Для реальных функций значение по умолчанию 13+количество десятичных чисел, обозначенных как `initid->decimals`. Для числовых функций длина включает любой знак или десятичные символы отметки.

```
char *ptr
```

Указатель, который функция может использовать для собственных целей. Например, функции могут использовать `initid->ptr`, чтобы передать распределенную память между функциями. В `xxx_init()` как обычно распределите память и назначьте ее этому указателю:

```
initid->ptr=allocated_memory;
```

В `xxx()` и `xxx_deinit()` обратитесь к `initid->ptr`, чтобы использовать или освободить память.

## Обработка параметров

Параметр `args` указывает на структуру `UDF_ARGS`, члены которой приведены ниже:

```
unsigned int arg_count
```

Число параметров. Проверьте это значение в функции инициализации, если Вы хотите, чтобы Ваша функция была вызвана со специфическим числом параметров. Например, таким кодом:

```
if (args->arg_count != 2)
{
    strcpy(message, "XXX() requires two arguments");
    return 1;
}
enum Item_result *arg_type
```

Типы для каждого параметра. Возможные значения типов: `STRING_RESULT`, `INT_RESULT` и `REAL_RESULT`. Чтобы удостовериться, что параметры имеют данный тип и возвращают ошибку, если они к нему не принадлежат, проверьте массив `arg_type` в функции инициализации. Например:

```
if (args->arg_type[0] != STRING_RESULT ||
    args->arg_type[1] != INT_RESULT)
{
    strcpy(message, "XXX() requires a string and an integer");
    return 1;
}
```

Вы можете использовать функцию инициализации, чтобы установить элементы `arg_type` к типам, которые Вы хотите получить. Это заставляет MySQL привести параметры к тем типам для каждого обращения к `xxx()`.

Например, чтобы определить первые два элемента как строку и число, сделайте следующее в `xxx_init()`:

```
args->arg_type[0] = STRING_RESULT;
```

```
args->arg_type[1] = INT_RESULT;
```

```
char **args
```

`args->args` сообщает информацию функции инициализации относительно общего характера параметров, с которыми Ваша функция была вызвана. Для постоянного параметра (константы) `i` `args->args[i]` указывает на значение параметра. Для непостоянного параметра `args->args[i]` равно 0. Постоянный параметр представляет собой выражение, которое использует только константы, типа `3`, `4*7-2` или `SIN(3.14)`. Непостоянный параметр представляет собой выражение, которое обращается к значениям, которые могут изменяться, типа имени столбца или функций, которые вызваны с непостоянными параметрами. Для каждого обращения основной функции `args->args` хранит фактические параметры, которые переданы для в настоящее время обрабатываемой строки. Функции могут обратиться к параметру `i` следующим образом:

Параметр типа `STRING_RESULT`, данный как указатель строки плюс длина, позволяет обработку двоичных данных или данных произвольной длины. Содержание строки доступно как `args->args[i]`, а длина строки как `args->lengths[i]`. Вы не должны считать, что строка завершается нулевым символом.

Для параметра типа `INT_RESULT` Вы должны привести `args->args[i]` к типу `long long`:

```
long long int_val;
```

```
int_val = *((long long*) args->args[i]);
```

Для параметра типа `REAL_RESULT` Вы должны привести `args->args[i]` к типу `double`:

```
double real_val;
```

```
real_val = *((double*) args->args[i]);
```

```
unsigned long *lengths
```



Для функции инициализации, массив `lengths` указывает максимальную длину строки для каждого параметра. Для каждого обращения к основной функции `lengths` хранит фактические длины любых строковых параметров, которые переданы для строки, обрабатываемой в настоящее время. Для параметров типов `INT_RESULT` или `REAL_RESULT` `lengths` хранит максимальную длину параметра (как для функции инициализации).

### **Возвращаемые значения и обработка ошибок**

Функция инициализации возвратит 0, если никакая ошибка не произошла, и 1 в противном случае. Если ошибка происходит, `xxx_init()` должна сохранить сообщение об ошибке с нулевым символом в конце в параметре `message`. Сообщение будет возвращено пользователю. Буфер сообщений имеет длину в `MYSQL_ERRMSG_SIZE` символов, но Вы должны попробовать сохранить сообщение в 80 символах так, чтобы это удовлетворило ширине стандартного экрана терминала.

Значение возврата основной функции `xxx()` зависит от типа. Для функций типов `long long` и `double` оно представляет собой собственно функциональное значение. Строковые функции должны вернуть указатель на результат и сохранить длину строки в параметрах `length`. Здесь `result` представляет собой буфер длиной в 255 байт. Установите их к содержанию и длине значения. Например:

```
memcpy(result, "result string", 13);
*length=13;
```

Если Ваши функции строки должны вернуть строку длиннее, чем 255 байт, распределите память для результата через `malloc()` в функции `xxx_init()` или в `xxx()`, а затем освободите память в `xxx_deinit()`. Вы можете сохранять распределенную память в слоте `ptr` структуры `UDF_INIT` для повторного использования в будущем обращении `xxx()`. Подробности в разделе "3.1.2.1 Соглашения о вызове UDF".

Чтобы указывать значение возврата `NULL` в основной функции,

установите `is_null` в 1:

```
*is_null=1;
```

Чтобы указать возврат ошибки в основной функции, установите параметр ошибки (`error`) в значение 1:

```
*error=1;
```

Если `xxx()` устанавливает `*error` в 1 для любой строки, функциональное значение `NULL` для текущей строки и для любых последующих строк, обработанных инструкцией, в которой вызывалась `XXX()`. Причем, `xxx()` не будет даже запрашиваться для последующих строк. ПРИМЕЧАНИЕ: В MySQL до версии 3.22.10 Вы должны установить `*error` и `*is_null`:

```
*error=1;
```

```
*is_null=1;
```

### **Компиляция и установка определяемых пользователем функций**

Файлы, выполняющие UDF, должны компилироваться и устанавливаться на сервере. Этот процесс описан ниже для примерного UDF-файла `udf_example.cc`, который включен в дистрибутив исходников MySQL. Этот файл содержит следующие функции:

`metaphon()` возвращает мета-строку для строкового параметра. Это похоже на `soundex`, но больше заточено под английский.

`myfunc_double()` возвращает сумму ASCII-значений символов в параметрах, поделенную на сумму длин этих параметров.

`myfunc_int()` возвращает сумму длин параметров.

`sequence([const int])` возвратит последовательность, начинающуюся с заданного числа или с 1, если никакого числа задано не было.

`lookup()` возвращает IP-адрес.

`reverse_lookup()` возвращает `hostname` для IP-адреса. Функция может быть вызвана со строкой `"xxx.xxx.xxx.xxx"` или с 4 числами.

Динамически загружаемый файл должен компилироваться как разделяемый объектный файл, используя команду:

```
shell> gcc -shared -o udf_example.so myfunc.cc
```

Вы можете легко выяснять правильные параметры компилятора для Вашей системы, запуская такую команду в каталоге sql Вашего дерева исходных текстов MySQL:

```
shell> make udf_example.o
```

Вы должны выполнить команду компиляции, подобную одной из тех, что отображает make, за исключением того, что Вы должны удалить опцию -c близко к концу строки и добавить -o udf\_example.so в самый конец строки. На некоторых системах удалять -c не надо, пробуйте.

Как только Вы скомпилируете общедоступный объект, содержащий UDF, Вы должны установить его и сообщить MySQL о расширении функциональности. Компиляция общедоступного объекта из udf\_example.cc производит файл с именем udf\_example.so (точное имя может изменяться от платформы к платформе). Скопируйте этот файл в некоторый каталог, где ищет файлы ld, например, в /usr/lib. На многих системах Вы можете устанавливать системную переменную LD\_LIBRARY или LD\_LIBRARY\_PATH, чтобы указать каталог, где Вы имеете Ваши файлы функции UDF. Руководство на dlopen сообщает Вам, которую переменную Вы должны использовать на Вашей системе. Вы должны установить это в mysql.server или в safe\_mysqld и перезапустить mysqld.

После того, как библиотека установлена, сообщите mysqld относительно новых функций этими командами:

```
mysql> CREATE FUNCTION metaphon RETURNS STRING SONAME
"udf_example.so";
```

```
mysql> CREATE FUNCTION myfunc_double RETURNS REAL SONAME
"udf_example.so";
```

```
mysql> CREATE FUNCTION myfunc_int RETURNS INTEGER SONAME
"udf_example.so";
```

```
mysql> CREATE FUNCTION lookup RETURNS STRING SONAME
"udf_example.so";
```

```
mysql> CREATE FUNCTION reverse_lookup RETURNS STRING  
SONAME  
      "udf_example.so";
```

Функции могут быть удалены, используя DROP FUNCTION:

```
mysql> DROP FUNCTION metaphon;  
mysql> DROP FUNCTION myfunc_double;  
mysql> DROP FUNCTION myfunc_int;  
mysql> DROP FUNCTION lookup;  
mysql> DROP FUNCTION reverse_lookup;
```

Инструкции CREATE FUNCTION и DROP FUNCTION модифицируют системную таблицу func в базе данных mysql. Имя функции, тип и общедоступное библиотечное имя будут сохранено в таблице. Вы должны иметь привилегии insert и delete для базы данных mysql, чтобы создавать и удалять свои функции.

Вы не должны использовать CREATE FUNCTION, чтобы добавить функцию, которая уже была создана. Если Вы должны повторно установить функцию, сначала удалите ее через вызов DROP FUNCTION и затем повторно установите ее с помощью CREATE FUNCTION. Вы должны сделать это, например, если Вы откомпилировали новую версию Вашей функции, чтобы mysqld обновил используемую им версию. Иначе сервер продолжит применять старую версию.

Активные функции будут перезагружены при каждом перезапуске сервера, если Вы не запускаете mysqld с опцией --skip-grant-tables. В этом случае инициализация UDF будет пропущена, а UDF-функции станут недоступными. Активная функция представляет собой функцию, загруженную через CREATE FUNCTION, но не удаленную DROP FUNCTION.

### **Добавление новых встроенных функций**

Процедура для добавления новой встроенной функции описана ниже. Обратите внимание, что Вы не можете добавлять встроенные функции к

двоичному дистрибутиву потому, что процедура включает изменение исходного текста MySQL. Вы должны скомпилировать MySQL самостоятельно из исходников. Также обратите внимание, что, если Вы мигрируете на другую версию MySQL (например, когда новая версия выпущена), Вы будете должны повторить процедуру с новой версией.

Чтобы добавить новую встроенную функцию MySQL, нужно:

Добавьте одну строку в файл `lex.h`, которая определяет имя функции в массиве `sql_functions[]`.

Если функциональный прототип прост (берет не более трех параметров), Вы должны в `lex.h` определить `SYM(FUNC_ARG#)` (здесь `#` является числом параметров) как второй параметр в массиве `sql_functions[]` и добавить функцию, которая создает функциональный объект, в `item_create.cc`. Смотрите "ABS" и `create_funcs_abs()` как пример. Если функциональный прототип усложнен (например, берет переменное число параметров), Вы должны добавить две строки к `sql_yacc.yy`. Каждая указывает символ препроцессора, который yacc должен определить (это должно быть добавлено в начале файла). Затем определите функциональные параметры и добавьте элемент с этими параметрами для правила синтаксического анализа `simple_expr`. Для примера, проверьте все местонахождения `ATAN` в `sql_yacc.yy`, чтобы увидеть, как это выполнено.

В `item_func.h` объявите наследование класса из `Item_num_func` или `Item_str_func`, в зависимости от того, возвращает ли Ваша функция число или строку.

В `item_func.cc` добавьте одно из следующих объявлений в зависимости от того, определяете ли Вы числовую или строковую функцию:

```
double Item_func_newname::val()
longlong Item_func_newname::val_int()
String *Item_func_newname::Str(String *str)
```

Если Вы наследуете Ваш объект от любого из стандартных элементов (подобно `Item_num_func`, Вы, вероятно, должны только определить одну из

вышеупомянутых функций и позволить родительскому объекту заботиться о других функциях. Например, класс `Item_str_func` определяет функцию `val()`, которая выполняет `atof()` на значении, возвращенном `::str()`.

Вы должны, вероятно, также определить следующую объектную функцию:

```
void Item_func_newname::fix_length_and_dec()
```

Эта функция должна по крайней мере вычислить `max_length`, исходя из данных параметров. `max_length` задает максимальное число символов, которое функция может возвращать. Эта функция должна также установить `maybe_null=0`, если основная функция не может возвращать значение `NULL`. Функция может проверить, способен ли любой из параметров возвращать `NULL`, проверяя переменную параметров `maybe_null`. Вы можете изучить `Item_func_mod::fix_length_and_dec` в качестве типичного примера того, как все это сделать.

Все функции должны быть поточно-безопасными (другими словами, не используйте любые глобальные или статические переменные в функциях без того, чтобы защитить их через `mutex`).

Если Вы хотите возвращать `NULL` из `::val()`, `::val_int()` или `::str()` Вы должны установить `null_value` в 1 и вернуть из функции 0.

Для объектной функции `::str()` имеются некоторые дополнительные хитрости, которые надо знать:

Параметр `String *str` обеспечивает буфер строки, который может использоваться, чтобы хранить результат. Для получения большего количества информации относительно типа `String` обратитесь к файлу `sql_string.h`.

Функция `::str()` должна вернуть строку, которая хранит результат, или `(char*) 0`, если результатом является `NULL`.

Все текущие функции строки не должны распределять никакую память, если это не абсолютно необходимо!

## 2.4 Добавление новых процедур в MySQL

В MySQL Вы можете определять процедуру на C++, которая может обращаться и изменять данные в запросе прежде, чем они отправятся к пользователю. Модификация может быть выполнена на уровне строки или GROUP BY.

Авторы пакета создали процедуру примера в MySQL Version 3.23, чтобы показать Вам, что там может быть выполнено.

Дополнительно авторы рекомендуют Вам посмотреть файл `mylua`, который Вы можете найти в каталоге `Contrib`. Вы можете использовать язык LUA, чтобы загрузить процедуру в `mysqld` прямо во время выполнения.

### 2.4.1 Анализ процедур

```
analyse([max elements],[max memory])
```

Эта процедура определена в `sql/sql_analyse.cc`. Она исследует результат, полученный из Вашего запроса, и возвращает анализ результатов:

`max elements` (по умолчанию 256) задает максимальное число разных значений, которые `analyse` заметит в столбце. Это используется, чтобы проверить оптимальность применения типа ENUM.

`max memory` (по умолчанию 8192) задает максимум памяти, которую `analyse` должен распределить на столбец при попытке найти все отличные значения.

```
SELECT ... FROM ... WHERE ... PROCEDURE ANALYSE([max
elements],[max memory])
```

### 2.4.2 Написание процедур

На сегодняшний день единственной документацией для этого является исходный код пакета.

Вы можете найти всю информацию относительно процедур, исследуя файлы:

```
sql/sql_analyse.cc  
sql/procedure.h  
sql/procedure.cc  
sql/sql_select.cc
```

## 2.5 Начинка MySQL

Эта глава описывает много вещей, которые Вы должны знать при работе на коде MySQL. Если Вы планируете способствовать MySQL разработке, иметь доступ к коду отлаживаемых версий или хотите только следить за разработкой, следуйте командам в разделе " 2.3.4 Установка из дерева исходников для разработки". Если Вы заинтересованы внутренней организацией MySQL, Вы должны также подписаться на специальный список рассылки [internals@lists.mysql.com](mailto:internals@lists.mysql.com).

### 2.5.1 Потоки в MySQL

Сервер MySQL создает следующие потоки:

Поток TCP/IP-подключений обрабатывает все подключения, запрашивает и создает новый специализированный поток, чтобы обработать авторизацию и запросы SQL для каждого подключения.

В Windows NT имеется драйвер именованного канала, который делает ту же самую работу, что и поток TCP/IP, но с запросами на именованном канале.

Поток сигнала обрабатывает все сигналы. Он также обычно обрабатывает тревоги и вызывает `process_alarm()`, чтобы завершить подключения, которые были неактивны слишком долго.

Если `mysqld` компилируется с `-DUSE_ALARM_THREAD`, специализированный поток, который обрабатывает тревоги, будет создан. Это используется только на некоторых системах, где имеются проблемы с `sigwait()`, или если есть недостатки в применении кода `thr_alarm()` в прикладной программе без специализированного потока обработки сигнала.

Если использована опция `--flush_time=#`, будет создан еще один



специализированный поток, который сбрасывает таблицы на диск.

Каждое соединение обрабатывается своим потоком.

Каждая таблица, на которой использована инструкция INSERT DELAYED, получает собственный поток.

Если Вы используете --master-host, будет запущен поток репликации, чтобы читать и применять модификации с главного сервера.

mysqladmin processlist показывает только подключения, потоки репликации и INSERT DELAYED.

## 2.5.2 Набор тестов MySQL

До недавнего времени основной набор теста был основан на составляющих собственность данных заказчика и по этой причине не был публично доступен. Единственный публично доступная часть процесса тестирования состояла из теста crash-me, эталонного теста Perl DBI/DBD, находящегося в каталоге sql-bench, и разнообразных тестов, размещенных в каталоге tests. Отсутствие стандартизированного публично доступного набора тестов сделало трудным для пользователей и разработчиков тестирование кода MySQL. Чтобы исправить эту ситуацию, авторы пакета создали совершенно новую систему тестов, которая теперь включена в исходные и двоичные дистрибутивы, начиная с Version 3.23.23.

Текущий набор тестов не проверяет все в MySQL, но должен охватить наиболее очевидные ошибки в обработка кода SQL, OS/library проблемы и тестирование репликации. Конечная цель состоит в том, чтобы иметь тесты, покрывающие 100% кода. Вы можете предоставить тесты, которые исследуют функциональные возможности, критичные для Вашей системы, поскольку это гарантирует, что все будущие выпуски MySQL будут хорошо работать с Вашими прикладными программами.

### Запуск набора тестов MySQL

Система теста состоит из интерпретатора языков тестов (mysqltest),

скрипта оболочки, чтобы выполнить все тесты (`mysql-test-run`), фактических случаев тестов, написанных на специальном языке тестов и их ожидаемых результатов. Чтобы выполнить набор теста на Вашей системе после построения, введите `make test` или `mysql-test/mysql-test-run` из корневого каталога исходных текстов. Если Вы установили двоичный дистрибутив, перейдите в корень установки (например, `/usr/local/mysql`) и скомандуйте `scripts/mysql-test-run`. Все тесты должны выполняться. Если этого не произошло, попробуйте выяснить почему и сообщите о проблеме, если это ошибка в пакете MySQL. Подробности в разделе "3.3.2.3 Как сообщать о проблемах и ошибках в наборе тестов MySQL".

Если Вы имеете копию `mysqld` на машине, где Вы хотите выполнить набор тестов, Вы не должны останавливать ее, если она не использует порты 9306 и 9307. Если один из этих портов применяется, Вы должны отредактировать `mysql-test-run` и изменить значения главного или подчиненного порта к тому, которое является доступным.

Вы можете запустить индивидуально каждый тест командой `mysql-test/mysql-test-run test_name`.

Если один тест свалился, проверьте работу `mysql-test-run` с опцией `--force`, чтобы проверить, сбоят ли любые другие тесты.

### **Расширение набора тестов MySQL**

Вы можете использовать язык `mysqltest`, чтобы писать Ваши собственные случаи теста. К сожалению, авторы пакета еще не написали полную документацию для него. Вы можете, однако, рассматривать текущие случаи теста и использовать их как пример. Следующие пункты должны помочь Вам:

Тесты находятся в каталоге `mysql-test/t/*.test`

Случай теста состоит из завершенной точкой с запятой (;) инструкции и подобен вводу клиента командной строки `mysql`. Инструкция по умолчанию: запрос, который будет послан серверу MySQL, если он не распознан как

внутренняя команда (например, `sleep`).

Все запросы, которые производят результаты, например, `SELECT`, `SHOW`, `EXPLAIN` и прочие, нужно предварить указанием `@/path/to/result/file`. Файл должен содержать ожидаемые результаты. Простой способ генерировать файл результата состоит в том, чтобы выполнить `mysqltest -r < t/test-case-name.test` из каталога `mysql-test`, а затем отредактировать сгенерированные файлы результата, если необходимо скорректировать их к ожидаемому выводу. В этом случае будьте очень осторожны относительно добавления или удаления любых невидимых символов. Если Вы должны вставить строку, удостоверьтесь, что поля отделяются позициями табуляции, и имеется табуляция в конце. Вы можете использовать `od -c`, чтобы удостовериться, что Ваш текстовый редактор не добавляет что-нибудь неожиданное в течение процесса редактирования.

Чтобы все соответствовало установке, Вы должны поместить Ваши файлы результата в каталог `mysql-test/r` и назвать их как `test_name.result`. Если тест производит больше, чем один результат, Вы должны использовать `test_name.a.result`, `test_name.b.result` и так далее.

Если инструкция возвращает ошибку, Вы должны на строке перед ней указать `--error error-number`. Здесь `error-number` может быть списком возможных кодов ошибок, отделяемых запятыми (,).

Если Вы записываете случай теста репликации, Вы должны в первой строке файла теста помещать `source include/master-slave.inc`; Чтобы переключаться между главной и подчиненной системами, используйте `connection master`; и `connection slave`;. Если Вы должны делать что-то на альтернативном подключении, Вы можете сделать подключение `connection master1`; для главной и `connection slave1`; для подчиненной системы.

Если Вы должны делать что-то в цикле, Вы можете использовать:

```
let $1=1000;
while ($1)
{
```

```
# Выполняем здесь запрос.
dec $1;
}
```

Чтобы бездействовать между запросами, используйте команду `sleep`. Она поддерживает доли секунды, так что Вы можете указать `sleep 1.5`;, например, чтобы бездействовать 1.5 секунды.

Чтобы выполнять подчиненного с дополнительными параметрами для Вашего случая теста, поместите их в формате командной строки в `mysql-test/t/test_name-slave.opt`. Для главной системы поместите их в файл `mysql-test/t/test_name-master.opt`.

Если Вы имеете вопрос относительно набора теста или случая теста, который может пригодиться всем, напишите об этом на [internals@lists.mysql.com](mailto:internals@lists.mysql.com). Поскольку список не принимает вложения, Вы должны закачать по ftp все релевантные файлы на <ftp://support.mysql.com/pub/mysql/Incoming>.

### **Как сообщать об ошибках в наборе тестов MySQL**

Если Ваша версия MySQL не выполняет набор тестов, Вы должны сделать так:

Не торопитесь посылать отчет об ошибке! Сначала разберитесь толком, что там у Вас происходит и почему. Если отчет все-таки придется послать, пожалуйста, используйте для его генерации скрипт `mysqlbug`, чтобы разработчики могли получить информацию относительно Вашей системы и версии MySQL.

Удостоверьтесь, что включили вывод `mysql-test-run` и содержание всех `.reject` файлов в каталоге `mysql-test/r`.

Если тест валится в наборе, проверьте, что с ним будет происходить при непосредственном запуске командой:

```
cd mysql-test
mysql-test-run --local test-name
```

Если это терпит неудачу, то сконфигурируйте MySQL с опцией `--with-debug` и выполните `mysql-test-run` с опцией `--debug`. Если это также терпит неудачу, закачайте файл трассировки `var/tmp/master.trace` на <ftp://support.mysql.com/pub/mysql/secret>, чтобы авторы могли исследовать это. Пожалуйста, не забудьте также включить полное описание Вашей системы, версию `mysqld` и параметры компиляции.

Попробуйте также выполнить `mysql-test-run` с опцией `--force`, чтобы увидеть, имеется ли любой другой тест, который тоже терпит неудачу.

Если Вы компилировали MySQL самостоятельно, изучите руководство на предмет того, как компилировать MySQL на Вашей платформе или, что предпочтительно, используйте один из готовых двоичных дистрибутивов, который уже откомпилирован и может быть скачан с <http://www.mysql.com/downloads>. Все стандартные двоичные файлы должны проходить тестирование.

Если Вы получаете ошибку, подобно `Result length mismatch` или `Result content mismatch`, это означает, что вывод теста не соответствовал точно ожидаемому выводу. Это может быть ошибкой в MySQL, или дело в том, что Ваша версия `mysqld` производит малость иные результаты при некоторых обстоятельствах. Неудачные результаты теста будут помещены в файл с тем же самым основным именем, что и файл результата, но с расширением `.reject`. Если Ваш случай теста терпит неудачу, Вы должны сравнить два файла. Если Вы не можете увидеть, чем они отличаются, исследуйте их с помощью `od -c` и проверьте их длины.

Если тест терпит неудачу полностью, Вы должны проверить журналы в каталоге `mysql-test/var/log` для выяснения того, что не так.

Если Вы компилировали MySQL с отладкой, можно попробовать отлаживать тест запуском `mysql-test-run` с опциями `--gdb` и/или `--debug`. Подробности в разделе "6.1.2 Создание файлов трассировки ". Если Вы не компилировали MySQL для отладки, вероятно, стоит сделать это. Только определите параметр `--with-debug` для вызова `configure`!

### **3 Введение в ODBC**

#### **3.1 Введение в ODBC**

Open Database Connectivity (ODBC) представляет собой интерфейс прикладной программы (API) для доступа к базам данных. Это основано на спецификациях Call-Level Interface (CLI) от X/Open и ISO/IEC для API баз данных. Как язык доступа к базам данных применяется Structured Query Language (SQL).

ODBC разработан для максимальной способности к взаимодействию, то есть одна прикладная программа может без изменения своего исходного текста работать через интерфейс с какой угодно СУБД. Прикладные программы вызывают функции интерфейса ODBC, которые выполнены в специфических для базы данных модулях, названных драйверами. Использование драйверов изолирует прикладные программы от специфических для базы данных обращений. Подробнее об этом можно почитать на <http://www.microsoft.com/data>.

#### **3.2 Что такое ODBC**

Первое и главное: ODBC является спецификацией для API базы данных. Этот API независим от любой СУБД, операционной системы или языка программирования.

ODBC API основан на спецификациях CLI от X/Open и ISO/IEC. ODBC 3.x полностью соответствует обоим этим спецификациям, более ранние версии ODBC были основаны на предварительных версиях этих спецификаций, но полностью не выполняли их, зато добавили свойства, нужные только разработчикам оконных приложений, например, прокручиваемые курсоры.

Разработчики драйверов для СУБД выполняют функции ODBC API. Приложения вызывают функции в этих драйверах, чтобы обратиться к данным способом, независимым от базы данных. Администратор драйверов (Driver Manager) управляет связью между прикладными программами и драйверами.

### 3.3 Как ODBC стандартизирует доступ к базе данных

Имеются два архитектурных требования:

Прикладные программы должны быть способны обратиться ко многим СУБД, используя тот же самый исходный текст без того, чтобы его перетранслировать или заново компоновать.

Прикладные программы должны быть способны обратиться ко многим СУБД одновременно (через разные драйверы).

ODBC успешно решает эти проблемы следующим способом:

ODBC является интерфейсом уровня вызовов:

Чтобы решить проблему с тем, как прикладные программы обращаются ко многим СУБД, используя один и тот же исходный текст, существует стандарт CLI. ODBC содержит все функции в спецификации CLI и обеспечивает дополнительные функции, обычно требуемые прикладными программами.

ODBC определяет стандартный синтаксис SQL:

В дополнение к стандартному интерфейсу уровня обращения (вызова), ODBC определяет стандартный синтаксис SQL. Он базируется на спецификации X/Open SQL CAE. Если используемый ODBC синтаксис отличается от того, который применяет конкретная СУБД, производится преобразование на лету. Однако, такие преобразования редки потому, что большинство СУБД уже используют стандартный синтаксис языка SQL.

ODBC предоставляет Driver Manager для управления одновременным доступом к многим СУБД:

Хотя использование драйверов решает проблему одновременного доступа ко многим базам данных, код, необходимый, чтобы сделать это, может быть сложен. Прикладные программы которые разработаны, чтобы работать со всеми драйверами, не могут быть статически связаны с любыми драйверами. Вместо этого они должны загрузить драйверы во время выполнения и вызывать функции в них через таблицу указателей функций. Ситуация становится более

сложной, если прикладная программа использует много драйверов сразу. Чтобы избавить программу от проблем с этим, ODBC обеспечивает Driver Manager. Администратор драйверов (Driver Manager) осуществляет все функции ODBC обычно как вызовы функций ODBC в драйверах и статически связан с прикладной программой или загружен прикладной программой во время выполнения. Таким образом, вызовы из прикладной программы функций ODBC по именам обрабатываются в Driver Manager вместо того, чтобы обращаться по указателю к каждому драйверу. ODBC предоставляет много возможностей СУБД, но не требует, чтобы каждый драйвер поддерживал их все.

### 3.4 Архитектура (My)ODBC

Архитектура MyODBC имеет 5 главных компонентов как показано ниже:

Приложение:

Это программа, которая вызывает ODBC API, чтобы обратиться к данным с сервера (MySQL). Прикладная программа общается с администратором драйверов или драйвером, непосредственно использующим стандарт ODBC-обращения. Прикладная программа не заботится, где данные сохранены, как они сохранены или как система конфигурирована, чтобы обратиться к данным. Единственное, что реально надо знать, имя источника данных (Data Source Name, DSN). Ряд задач является общим для всех программ, независимо от того, как они используют ODBC. Эти задачи:

Выбор сервера (MySQL) и связь с ним.

Передача на рассмотрение инструкции SQL для выполнения.

Получение результатов (если они есть).

Обработка ошибок.

Обработка транзакции или обратная перемотка.

Отсоединение от сервера.

Самыми главными тут являются передача на рассмотрение инструкции



SQL для выполнения и получение результатов запроса (если они есть).

Driver manager:

Driver Manager представляет собой библиотеку, которая управляет связью между прикладной программой и драйвером или драйверами. Это делает:

Обработку Data Source Names (DSN).

Загрузку и выгрузку драйверов.

Обработку ODBC-обращения к функции или передачу вызова драйверу.

Драйвер MyODBC:

Драйвер MyODBC представляет собой библиотеку, которая осуществляет функции в ODBC API. Это обрабатывает ODBC-обращения к функции, представляет на рассмотрение запросы SQL на сервер MySQL и возвращает результаты обратно прикладной программе. В случае необходимости, драйвер изменяет запрос прикладной программы так, чтобы запрос соответствовал синтаксису MySQL.

ODBC.INI:

ODBC.INI является главным файлом настройки ODBC, который хранит драйвер и информацию о базе данных, требуемую, чтобы соединиться с сервером. Это используется Driver Manager, чтобы определить который драйвер надо загрузить, используя Data Source Name. Драйвер использует этот файл, чтобы прочесть параметры подключения, основанные на определенном DSN. Для получения большего количества информации прочтите раздел "3.2 Настройка MyODBC DSN".

MySQL SERVER:

Это источник данных.

### **3.5 Типы ODBC-драйверов MySQL**

Как описано ранее, MySQL AB поддерживает два драйвера ODBC с открытыми исходными текстами, а именно MyODBC и MySQL ODBC 3.51, для работы с MySQL через ODBC API.

### 3.6 Где взять MyODBC

MySQL AB распространяет все свои программы под General Public License (GPL). Самую свежую версию MyODBC или MyODBC 3.51 (двоичные коды и исходные тексты) можно скачать с <http://www.mysql.com>. Подробно о MySQL ODBC рассказано на <http://www.mysql.com/downloads/api-myodbc.html>.

### 3.7 Как установить MyODBC

#### 3.7.1 Двоичная версия на Windows

Для установки MyODBC на Windows Вы должны скачать соответствующий дистрибутивный файл для Вашей операционной системы с <http://www.mysql.com/downloads/api-myodbc.html>, распаковать его и выполнить файл SETUP.EXE.

#### 3.7.2 Установка из исходных текстов под Windows

##### Требования

MDAC, Microsoft Data Access SDK с [www.microsoft.com/data](http://www.microsoft.com/data).

Клиентская библиотека и включаемые файлы MySQL из 3.23.14 или выше. Это требуется потому, что MyODBC использует новые обращения, которые существуют только начиная с этой версии. <http://www.mysql.com/downloads/index.html>.

##### Построение MyODBC 3.51

MyODBC 3.51 поставляется в виде исходников с Makefile, который использует nmake. В дистрибутиве есть WIN\_Makefile для формирования нормальной версии и WIN\_Makefile\_debug для формирования отладочной версии драйвера и DLL-библиотек. Для построения драйвера:

Скачайте исходники и распакуйте их в какой-нибудь каталог, скажем, myodbc3-src. Далее выполните следующие команды для создания нормальной версии:

```
Command> cd myodbc3-src
```

```
Command> nmake -f Win_Makefile
```

```
Command> nmake -f Win_Makefile install
```

nmake -f Win\_Makefile формирует версию драйвера и помещает двоичные файлы в подкаталог release. Команда nmake -f Win\_Makefile install инсталлирует (вообще-то просто копирует) библиотеку драйвера и его DLL (myodbc3.lib и myodbc3.dll) в системный каталог ОС. Аналогично Вы можете сформировать версию для отладки, используя Win\_Makefile\_Debug:

```
Command> nmake -f Win_Makefile_debug
```

```
Command> nmake -f Win_Makefile_debug install
```

Вы можете очищать и восстанавливать драйвер, используя команды:

```
Command> nmake -f Win_Makefile clean
```

```
Command> nmake -f Win_Makefile install
```

**ОБРАТИТЕ ВНИМАНИЕ:** Удостоверитесь, что определили правильные библиотеки пользователей MySQL и путь файлов заголовка в Makefile. Это принимает заданный по умолчанию путь C:\mysql\include и C:\mysql\lib\opt (для обычных DLL) или C:\mysql\lib\debug для отладочной версии.

Тестирование библиотек драйвера: после того, как библиотеки драйвера скопированы в системный каталог, Вы можете проверить качество их построения используя выборки, обеспеченные в подкаталоге samples каталога исходного текста:

```
Command> cd samples
```

```
Command> nmake -f Win_Makefile all
```

## **Построение MyODBC 2.50**

MyODBC распространяется в исходниках как VC Project для Windows. Можно формировать драйвер, используя прямые файлы проекта VC (.dsp и .dsw), имеющиеся в дистрибутиве.

### 3.7.3 Установка из исходных текстов под Unix

Чтобы формировать драйвер самостоятельно под Linux, Вы должны иметь:

#### Требования

Клиентская библиотека и включаемые файлы MySQL из 3.23.14 или выше. Это требуется потому, что MyODBC использует новые обращения, которые существуют только начиная с этой версии.  
<http://www.mysql.com/downloads/index.html>

Библиотека MySQL должна быть конфигурирована с опцией `--with-thread-safe-client`. `libmysqlclient` должна быть установлена в системе как разделяемая библиотека.

Должен быть установлен один из администраторов драйверов unix ODBC:

`iodbc 3.0` или старше (<http://www.iodbc.org>) или  
`unixodbc Alpha 3` или старше (<http://www.unixodbc.org>).

Если Вы используете `character set`, который не компилируется в библиотеку MySQL (по умолчанию там будут: `latin1`, `big5`, `czech`, `euc_kr`, `gb2312`, `gbk`, `sjis`, `tis620` и `ujis`), Вы должны установить символьные определения `mysql` из каталога `charset` в `SHAREDIR` (по умолчанию это `/usr/local/mysql/share/mysql/charsets`). Они должны уже быть на месте, если Вы установили сервер MySQL на той же самой машине.

Как только Вы поймете все требуемые файлы и распакуете исходные файлы в отдельный каталог, начинайте сборку драйвера как показано ниже.

#### Настройка

Единственные требуемые параметры; `--with-mysql-libs=DIR` и `--with-mysql-includes=DIR`. Здесь `DIR` задает каталог, где искать библиотеки и включаемые файлы `mysql`.

При использовании `iodbc`, если `iodbc` не установлен в заданное по умолчанию расположение (`/usr/local`), Вам придется использовать `--with-iodbc=DIR` или, если заголовки `iODBC` не находятся в `DIR/include`, придется использовать опцию `--with-iodbc-includes=INCDIR`. То же самое касается и библиотек: если они не в `DIR/lib`, примените явное указание параметром `--with-iodbc-libs=LIBDIR`.

При использовании `unixODBC` для создания `configure` ищите `unixODBC` вместо `iODBC` и используйте параметр `--with-unixODBC=DIR`. Здесь `DIR` задает то место, где установлен `unixODBC`.

Если заголовки и библиотеки `unixODBC` расположены не там, где надо (а надо в `DIR/include` и `DIR/lib` соответственно), укажите на них параметрами `--with-unixODBC-libs=LIBDIR` и `--with-unixODBC-includes=INCDIR`.

Вы можете определять другой префикс вместо `/usr/local` для установки, например, хранить `MyODBC`-драйверы в `/usr/local/odbc/lib`, для этого надо указать параметр `--prefix=/usr/local/odbc`.

Окончательно пример настройки выглядит так:

```
$ ./configure --prefix=/usr/local --with-iodbc=/usr/local
\
        --with-mysql-
libs=/usr/local/mysql/lib/mysql \
        --with-mysql-
includes=/usr/local/mysql/include/mysql
```

### **Построение библиотек драйвера**

Для построения библиотек драйвера Вы должны только выполнить:

```
$ make
```

Это должно сформировать библиотеки. Если происходят какие-то ошибки, исправьте их и продолжите построение. Если Вы не можете собрать пакет, пошлите детальный отчет по e-mail на [myodbc@lists.mysql.com](mailto:myodbc@lists.mysql.com).

### 2.7.3.4 Установка библиотек драйвера

`$ make install`

Устанавливает следующий набор библиотек:

`libmyodbc3.so`, `libmyodbc3-3.51.01.so`, здесь 3.51.01 является версией драйвера и `libmyodbc3.a` для MyODBC 3.51 или

`libmyodbc.so`, `libmyodbc-2.50.39.so`, здесь 2.50.39 является версией драйвера и `libmyodbc.a` для MyODBC 2.50

Обратите внимание, если Вы пробуете использовать `make` из SUN, Вы закончите с ошибками. С другой стороны, `Make` от GNU должен работать прекрасно на всех платформах.

### Замечания для Mac OS X

Если Вы хотите формировать драйвер на Mac OS (Darwin), то используйте следующий пример выбора конфигурации:

```
$ ./configure --prefix=/usr/local --with-
unixodbc=/usr/local \
    --with-mysql-
libs=/usr/local/mysql/lib/mysql \
    --with-mysql-
includes=/usr/local/mysql/include/mysql \
    --disable-shared --enable-gui=no --
host=powerpc-apple
```

Это предполагает, что `unixodbc` и `mysql` установлены в заданные по умолчанию расположения. Если это не так, сконфигурируйте все соответственно.

### Создание файлов `.so`

На большинстве платформ MySQL не формирует или поддерживает `.so`

файлы, так как формирование с общедоступными библиотеками вызывало проблемы в прошлом.

В таких случаях пользователь должен скачать дистрибутив MySQL и скомпилировать его с помощью:

```
--without-server --enable-shared
```

Если Вы все же хотите формировать общедоступные библиотеки драйвера, Вы должны определить опцию настройки `--enable-shared`.

Если Вы конфигурировали пакет с опцией `--disable-shared`, то Вы можете формировать `.so`-файл, используя следующее:

```
$ cd MyODBC-3.51.01
$ make
$ cc -bundle -undefined error -o .libs/libmyodbc3-
3.51.01.so *.o \
    -lz -lc -lmysqlclient -L/usr/local/mysql/lib/mysql/
\
    -compatibility_version 2 -current_version 2.0
```

Это формирует и помещает файл `libmyodbc3-3.51.01.so` в каталог `.libs`

Скопируйте этот файл в каталог библиотек MyODBC (`/usr/local/lib` или что

Вы там определили в `--prefix`).

```
$ cd .libs
$ cp libmyodbc3-3.51.01.so /usr/local/lib
$ cd /usr/local/lib
$ ln -s libmyodbc3-3.51.01.so libmyodbc3.so
```

### 3.7.4 Установка из дерева исходных текстов для разработки

Предостережение: Вы должны читать этот раздел только, если Вы

заинтересованы в разработке пакета. Если Вы только хотите получить MyODBC 3.51, Вы должны использовать стандартный дистрибутив (исходный текст или двоичные модули).

Чтобы получить самое современное дерево исходных текстов для разработчиков, сделайте следующее:

Скачайте BitKeeper с <http://www.bitmover.com/cgi-bin/download.cgi>. Вам нужен Bitkeeper 2.0 или более новый, чтобы обратиться к архиву.

Установите его согласно прилагаемым инструкциям.

Когда BitKeeper установлен, сначала перейдите в каталог, откуда хотите работать, а затем используйте эту команду, если Вы хотите получить снимок исходных текстов MyODBC 3.51:

```
shell> bk clone bk://work.mysql.com:7002/myodbc-3.51
```

В вышеупомянутых примерах исходное дерево будет установлено в подкаталог myodbc-3.51 Вашего текущего каталога.

Вам понадобятся GNU autoconf 2.13, automake 1.4, libtool и m4, чтобы выполнить следующий набор команд:

```
shell> cd myodbc-3.51
```

```
shell> bk -r edit
```

```
shell> aclocal; autoheader; autoconf; automake;
```

```
shell> ./configure # Здесь напишите параметры
```

```
shell> make
```

Под Windows используйте WIN-Makefile и WIN-Makefile\_debug.

Когда пакет собран, выполните `make install`. Эта команда установит драйвер MyODBC 3.51 на Вашей системе. Если Вы установили последние версии требуемых инструментальных средств GNU, и они разрушаются при попытке обработать файлы конфигурации, пожалуйста, сообщите об этом на [myodbc@lists.mysql.com](mailto:myodbc@lists.mysql.com). Однако, если Вы выполняете `aclocal` и получаете ошибку "command not found" или подобную проблему, не сообщайте об этом. Вместо этого, удостоверьтесь, что все необходимые инструментальные средства установлены, и что Ваша переменная PATH задана правильно, так что



оболочка может найти их.

После того, как команда `bk clone` скачает дерево исходных текстов, Вы должны периодически выполнять `bk pull`, чтобы получить очередные модификации.

Вы можете исследовать хронологию изменения дерева со всеми дифами (diff), используя `bk sccstool`. Если Вы видите некоторый странный диф или непонятный код, спрашивайте по e-mail [myodbc@lists.mysql.com](mailto:myodbc@lists.mysql.com). Также, если Вы думаете, что имеете лучшую идею относительно того, как что-то сделать, пришлите патч на тот же адрес. Патч изготавливается командой `bk diffs` после внесения правок в исходные тексты. Если Вы не имеете времени писать код, пошлите описание своей идеи.

BitKeeper имеет хорошую утилиту справочника, к которой Вы можете обращаться через `bk helptool`.

### 3.8 Поддерживаемые платформы

Драйвер MySQL ODBC может использоваться на всех основных платформах, поддерживаемых MySQL:

Все версии Windows: 95/98/NT/ME/2000/XP

OS/2

Mac OS X Server

Amiga

Все версии Unix:

AIX

BSDI

DEC

FreeBSD

HP-UX

Linux

NetBSD

OpenBSD

SGI Irix

Solaris

SunOS

SCO OpenServer

SCO UnixWare

Tru64 Unix

Вообще, MyODBC (3.51) поддержан на всех платформах, которые MySQL поддерживает. Если двоичный дистрибутив не доступен для специфической платформы, то Вы можете формировать драйвер самостоятельно, скачав его исходники. А чтобы это стало простым для других пользователей, пошлите готовые двоичные модули на [myodbc@lists.mysql.com](mailto:myodbc@lists.mysql.com).

### **3.9 Список рассылки MyODBC**

MySQL обеспечивает семейство пользователей своих пакетов списком рассылки, где новички получают решения от опытных пользователей, посылая запросы на [myodbc@lists.mysql.com](mailto:myodbc@lists.mysql.com).

Чтобы подписаться на список рассылки MyODBC, пошлите сообщение на [myodbc-subscribe@lists.mysql.com](mailto:myodbc-subscribe@lists.mysql.com). Чтобы отписаться от списка рассылки MyODBC, пошлите сообщение на [myodbc-unsubscribe@lists.mysql.com](mailto:myodbc-unsubscribe@lists.mysql.com). Вы можете также просматривать архив списка рассылки на <http://lists.mysql.com>.

### **3.10 Поддержка MyODBC**

Есть возможность получить поддержку пакета. Если Вы хотите, чтобы:  
Ошибки исправлялись быстро.

Решать любые проблемы с MyODBC или MySQL.

Была возможность заказывать свойство в драйвере.

Иметь расширение драйвера.

Иметь патчи, поставленные непосредственно в Ваш почтовый ящик.

Иметь прямое взаимодействие с разработчиками MySQL и MyODBC.

Вы должны получить контракт поддержки от MySQL AB. Фирма

MySQL AB поддерживает различные типы лицензий поддержки, чтобы помочь Вам выбрать именно то, что нужно. Для получения большего количества информации относительно поддержки MySQL, посетите сайт <https://order.mysql.com> или пошлите e-mail на [licensing@mysql.com](mailto:licensing@mysql.com).

Как только Вы купите поддержку, Вы сможете посылать запросы или отчеты об ошибках, используя MySQL support wizard на <http://support.mysql.com>, затем группа разработки MySQL начнет работать над Вашей проблемой немедленно.

### **3.11 Как сообщать об ошибках и проблемах с MyODBC**

Если Вы сталкиваетесь с трудностями с MyODBC или с MyODBC 3.51, Вы должны запустить создание файла протокола из ODBC Manager (протокол Вы получаете при запросе файлов регистрации из ODBC ADMIN) или протокол MyODBC (или MyODBC 3.51).

Для получения трассировки ODBC через Driver Manager, Вы должны сделать следующее:

Открыть ODBC Data Source Administrator:

Кликните на Start, укажите на Settings и запустите Control Panel.

На компьютерах под Microsoft Windows 2000, сделайте двойной щелчок на Administrative Tools и сделайте двойной щелчок на Data Sources (ODBC) . На компьютерах с более ранней версией Microsoft Windows дважды щелкните на 32-bit ODBC или на ODBC. Появится диалоговое окно ODBC Data Source Administrator .

Чтобы включить трассировку, Вы должны сделать следующее:

Вкладка Tracing диалогового окна ODBC Data Source Administrator нужна, чтобы сконфигурировать путь ODBC-обращения к функции трассировки.

Когда Вы активизируете трассировку из вкладки Tracing , Driver Manager регистрирует все ODBC-обращения к функциям для всех впоследствии выполненных программ.

ODBC-обращения к функциям из прикладных программ, запущенных ранее, не регистрируются. ODBC-обращения к функциям будут зарегистрированы в журнале, который Вы определяете.

Трассировка прекращается только после того, как Вы нажимаете Stop Tracing Now. Не забудьте, что при включенной трассировке журнал постоянно растет, а это воздействует на эффективность всех Ваших прикладных программ ODBC. Под Unix Вы должны явно установить опцию TRACE в файле настроек ODBC.INI. Установите трассировку в ON или в OFF, используя параметры TRACE или TRACEFILE в `odbc.ini`. Например, так:

```
TraceFile = /tmp/odbc.trace
```

```
Trace = 1
```

Здесь TRACEFILE определяет имя и полный путь файла протокола, а TRACE установлен в ON или в OFF. Вы можете также использовать 1 или YES для ON и 0 или NO для OFF.

Обратите внимание, если Вы используете ODBCConfig из `unixODBC`, то следуйте командам из HOWTO-ODBCConfig на <http://www.unixodbc.org/config.html> для трассировки вызовов `unixODBC`.

Чтобы получить протокол драйвера `MyODBC` или `MyODBC 3.51`, надо сделать следующее:

Гарантируйте, что Вы используете `myodbc3d.dll`, но не `myodbc3.dll` для `MyODBC 3.51` (или `myodbcd.dll` для `MyODBC`). Самый простой способ сделать это состоит в том, чтобы получить `myodbc3d.dll` (или `myodbcd.dll`) из дистрибутива `MyODBC 3.51` и скопировать поверх `myodbc3.dll` (или `myodbc.dll`), который, скорее всего, расположен в каталоге `C:\windows\system32` или в `C:\winnt\system32`. Обратите внимание, что Вы, вероятно, захотите восстановить старый файл `myodbc.dll` после окончания отладки, поскольку он работает намного быстрее, чем `myodbc3d.dll` (или `myodbcd.dll`), так что сделайте резервную копию оригинальной DLL.

Отметьте опцию Trace `MyODBC` на экране конфигурирования соединения через `MyODBC`. Протокол будет записан в файл `C:\myodbc.log`.

Если опция трассировки отсутствует на вышеупомянутом экране, это означает, что Вы не используете драйвер `myodbc.dll`. Под Linux или если Вы используете связь DSN-Less, Вы должны обеспечить в строке подключения параметр `OPTION=4`.

Запустите Вашу прикладную программу и попробуйте получить то, что вызвало проблемы. Проверьте файл протокола MyODBC, чтобы выяснить, что же могло быть неправильно. Если Вы выясняете, что пошло не так, пожалуйста, пошлите сообщение на [myodbc@lists.mysql.com](mailto:myodbc@lists.mysql.com) (если у Вас контракт о поддержке с MySQL AB, пишите на [support@mysql.com](mailto:support@mysql.com)) с кратким описанием проблемы и со следующей дополнительной информацией:

Версия MyODBC

Версия и тип используемого ODBC Driver Manager

Версия сервера MySQL

ODBC-трассировка из Driver Manager

Файл протокола драйвера MyODBC

Простой воспроизводимый пример

Не забудьте, что большее количество информации, которую Вы можете обеспечивать, повышает шансы на решение проблемы! Также стоит почитать архив списка рассылки MyODBC на <http://lists.mysql.com> перед отправкой сообщения: вдруг уже кто-то имел такую проблему и решил ее?

### **3.12 2.12 Программы, точно работающие с драйвером MyODBC**

Большинство программ должно работать с MyODBC, но для каждой из перечисленных ниже это было проверено непосредственно.

Программа

Комментарий

Access

Чтобы работать с Access:

Если Вы используете Access 2000, Вы должны получить и установить самый новый (версия 2.6 или выше) пакет Microsoft MDAC (Microsoft Data

Access Components) с <http://www.microsoft.com/data>. Это устранит следующую ошибку в Access: когда Вы экспортируете данные в MySQL, имена таблицы и столбца не определены. Другой путь обхода этой ошибки состоит в том, чтобы поставить MyODBC не ниже 2.50.33 и MySQL не ниже 3.23, которые вместе обеспечивают обход этой ошибки. Вы должны также получить и применить Microsoft Jet 4.0 Service Pack 5 (SP5), который может быть найден на <http://support.microsoft.com/default.aspx?scid=kb;EN-US;q239114>. Это исправит ряд ситуаций, в которых столбцы помечаются как удаленные (#deleted#) в Access. Обратите внимание, что, если Вы используете MySQL 3.22, Вы должны применить патч для MDAC и использовать MyODBC 2.50.32 или 2.50.34 и выше для решения проблемы.

Для всех версий Access Вы должны включить опцию MyODBC Return matching rows. Для Access 2.0 Вы должны дополнительно включить опцию Simulate ODBC 1.0.

Вы должны иметь timestamp во всех таблицах, которые Вы хотите модифицировать. Для максимальной мобильности TIMESTAMP(14) или просто TIMESTAMP рекомендуются вместо других вариантов TIMESTAMP(X).

Вы должны иметь первичный ключ в таблице. Если это не так, новые или модифицируемые строки могут обнаруживаться как #DELETED#.

Используйте только поля DOUBLE с плавающей точкой. Access рушится при сравнении с single-полями с плавающей точкой. Это приводит к тому, что новые или модифицируемые строки могут обнаруживаться как #DELETED#, или Вы не можете находить или модифицировать строки.

Если Вы связываете таблицу, которая имеет BIGINT как один из столбца, через MyODBC, то результаты будут отображаться как #DELETED. Обойти это можно так:

Создайте еще один фиктивный столбец с TIMESTAMP как тип данных, предпочтительно TIMESTAMP(14).

Проверьте Change BIGINT columns to INT в диалоге опций соединения в ODBC DSN Administrator.

Удалите связь таблицы из и пересоздайте ее.

Это все еще отображает предыдущие записи как #DELETED#, но новые записи будут отображаться правильно.

Если Вы все еще получаете ошибку Another user has changed your data после добавления столбца TIMESTAMP, следующий прием может помочь: не используйте представление листа данных таблицы. Создайте взамен форму с полями, которые Вы хотите видеть, и используйте вид form листа данных. Вы должны установить реквизит DefaultValue для столбца TIMESTAMP в NOW(). Стоит сделать столбец TIMESTAMP скрытым.

В некоторых случаях Access может генерировать запрещенные запросы SQL, которые MySQL не может понимать. Вы можете исправить это, выбирая Query|SQLSpecific|Pass-Through из меню Access.

Access под NT показывает столбцы BLOB как OLE OBJECTS. Если Вы хотите иметь столбцы MEMO вместо этого, Вы должны изменить тип столбца на TEXT с помощью команды ALTER TABLE.

Access не может всегда обрабатывать столбцы DATE правильно. Если Вы имеете проблему с ними, измените столбцы на DATETIME: с ними путаницы нет.

Если Вы имеете в Access столбец, определенный как BYTE, Access пробует экспортировать это как TINYINT вместо TINYINT UNSIGNED. Это задаст Вам проблем, если Вы имеете значения > 127 в этом столбце!

## ADO

Когда Вы программируете с ADO API и MyODBC Вы должны обратить внимание на некоторые заданные по умолчанию свойства, которые пока не поддерживаны сервером MySQL. Например, использование свойства CursorLocation Property как adUseServer возвратит для реквизита RecordCount Property -1. Чтобы иметь правильное значение, Вы должны установить это свойство в adUseClient, как показано в этом коде на VB:

```
Dim myconn As New ADODB.Connection
```

```

Dim myrs As New Recordset
Dim mySQL As String
Dim myrows As Long
myconn.Open "DSN=MyODBCsample"
mySQL = "SELECT * from user"
myrs.Source = mySQL
Set myrs.ActiveConnection = myconn
myrs.CursorLocation = adUseClient
myrs.Open
myrows = myrs.RecordCount

myrs.Close
myconn.Close

```

Другой обход: использовать инструкцию `SELECT COUNT(*)` для подобного запроса, чтобы получить правильное число строк.

#### Active server pages (ASP)

Вы должны использовать параметр `Return matching rows`.

#### Приложения BDE

Вы должны установить параметры `Don't optimize column widths` и `Return matching rows`. Тогда все должно работать.

#### Borland Builder 4

Когда Вы запускаете запрос, Вы можете использовать свойство `Active` или метод `Open`. Обратите внимание, что `Active` запускается автоматически при выдаче запроса `SELECT * FROM ...`, что медленно, если Ваши таблицы большие.

#### ColdFusion (под Unix)

Следующая информация найдена в документации на ColdFusion: используйте следующую информацию, чтобы конфигурировать ColdFusion Server для Linux, чтобы использовать драйвер `unixODBC` с `MyODBC` для источников данных MySQL. Известно, что `MyODBC Version 2.50.26` работает с



MySQL Version 3.22.27 и с ColdFusion для Linux. Любая более новая версия должна также работать. Вы можете скачать MyODBC с <http://www.mysql.com/downloads/api-myodbc.html>. ColdFusion Version 4.5.1 позволяет использовать ColdFusion Administrator, чтобы добавлять источник данных MySQL. Однако, драйвер не включен в поставку ColdFusion Version 4.5.1. Прежде, чем драйвер MySQL появится в раскрывающемся списке ODBC datasources, Вы должны сформировать и скопировать драйвер MyODBC в /opt/coldfusion/lib/libmyodbc.so. Каталог Contrib хранит программу mydsn-xxx.zip, которая позволяет Вам формировать и удалять DSN-файл системного реестра для драйвера MyODBC в программах Coldfusion.

#### DataJunction

Вы должны обновить пакет, иначе будет выводиться ENUM вместо VARCHAR: проблемы с протоколом.

#### Excel

Работает нормально. Некоторые советы:

Если Вы имеете проблемы с датами, попробуйте выбирать их как строки, использующие функцию CONCAT(). Например:

```
select CONCAT(rise_time), CONCAT(set_time) from sunrise_sunset;
```

Значения, полученные как строки этим способом, должны быть правильно распознаны как значения времени Excel97. Цель CONCAT() в этом примере состоит в том, чтобы ввести в заблуждение ODBC в плане того, что столбец имеет строковый тип. Без CONCAT() ODBC знает, что столбец имеет тип time, а Excel не понимает этого. Обратите внимание, что это ошибка в Excel, поскольку он автоматически преобразовывает строку в time, хотя никто его об этом не просит. Это было бы большим удобством, если источником является текстовый файл, но просто глупо, когда в качестве источника данных выступает ODBC-подключение, которое сообщает точные типы для каждого столбца.

#### Word

Чтобы получать данные из MySQL в документах Word/Excel, надо

использовать драйвер MyODBC и Add-in Microsoft Query help. Например, создайте базу данных с таблицей, содержащей 2 столбца текста:

Вставьте строки, используя инструмент командной строки mysql.

Создайте DSN-файл, использующий MyODBC-драйвер, например, my.

Откройте Word.

Создайте пустой новый документ.

При использовании панели инструментов Database, нажмите кнопку insert database.

Нажмите кнопку Get Data.

С правой стороны окна Get Data нажмите кнопку Ms Query.

В Ms Query создайте New Data Source, используя DSN-файл my.

Выберите новый запрос.

Выберите столбцы, которые Вы хотите.

Создайте фильтр, если он нужен.

Выполните сортировку, если она нужна.

Установите Return Data в Microsoft Word.

Нажмите кнопку Finish.

Нажмите Insert data и выберите записи.

Теперь после нажатия на ОК строки появятся в Вашем документе Word.

Delphi

Вы должны использовать BDE Version 3.2 или более новую. Установите опцию Don't optimize column width при соединении с MySQL. Также имеется некоторый потенциально полезный Delphi-код, который устанавливает ODBC-вход и BDE-вход для MyODBC (BDE-вход требует BDE Alias Editor, который можно свободно скачать с Delphi Super Page.

```
fReg:= TRegistry.Create;
```

```
  fReg.OpenKey(' \Software\ODBC\ODBC.INI\DocumentsFab',  
True);
```

```
  fReg.WriteString('Database', 'Documents');
```

```

fReg.WriteString('Description', ' ');
fReg.WriteString('Driver',
'C:\WINNT\System32\myodbc.dll');
fReg.WriteString('Flag', '1');
fReg.WriteString('Password', '');
fReg.WriteString('Port', ' ');
fReg.WriteString('Server', 'xmark');
fReg.WriteString('User', 'winuser');
fReg.OpenKey('\Software\ODBC\ODBC.INI\ODBC Data
Sources', True);
fReg.WriteString('DocumentsFab', 'MySQL');
fReg.CloseKey;
fReg.Free;

Memol.Lines.Add('DATABASE NAME=');
Memol.Lines.Add('USER NAME=');
Memol.Lines.Add('ODBC DSN=DocumentsFab');
Memol.Lines.Add('OPEN MODE=READ/WRITE');
Memol.Lines.Add('BATCH COUNT=200');
Memol.Lines.Add('LANGDRIVER=');
Memol.Lines.Add('MAX ROWS=-1');
Memol.Lines.Add('SCHEMA CACHE DIR=');
Memol.Lines.Add('SCHEMA CACHE SIZE=8');
Memol.Lines.Add('SCHEMA CACHE TIME=-1');
Memol.Lines.Add('SQLPASSTHRU MODE=SHARED AUTOCOMMIT');
Memol.Lines.Add('SQLQRYMODE=');
Memol.Lines.Add('ENABLE SCHEMA CACHE=FALSE');
Memol.Lines.Add('ENABLE BCD=FALSE');
Memol.Lines.Add('ROWSET SIZE=20');
Memol.Lines.Add('BLOBS TO CACHE=64');

```

```
Memol.Lines.Add('BLOB SIZE=32');
```

```
AliasEditor.Add('DocumentsFab','MySQL',Memol.Lines);
```

### C++ Builder

Тестировался с BDE Version 3.0. Единственная известная проблема состоит в том, что, когда схема таблицы изменилась, поля запроса не модифицируются. BDE, кажется, не распознает первичные ключи, только PRIMARY индекс, хотя это не было проблемой.

### Vision

Вы должны использовать параметр Return matching rows.

### Visual Basic

Чтобы модифицировать таблицу, Вы должны определить первичный ключ для таблицы. Visual Basic с ADO не может обрабатывать большие целые числа. Это означает, что некоторые запросы подобно SHOW PROCESSLIST не будут работать правильно. Установите опцию OPTION=16834 в строке подключения ODBC или задайте параметр Change BIGINT columns to INT на экране соединения MyODBC. Вы можете также устанавливать опцию Return matching rows.

### VisualInterDev

Если Вы получаете ошибку [Microsoft][ODBC Driver Manager] Driver does not support this parameter, причина может быть в том, что Вы имеете BIGINT в Вашем результате. Попробуйте установить параметр Change BIGINT columns to INT на экране соединения MyODBC.

### Visual Objects

Вы должны использовать параметр Don't optimize column widths.

### MS Visio Enterprise 2000

Пользователи сделали диаграмму модели базы данных, подключаясь из MS Vision Enterprise 2000 к MySQL через MyODBC (2.50.37 или более новый) и используя функцию обратной разработки в Visio, чтобы восстановить

информацию относительно базы данных (Visio показывает все определения столбца, первичные ключи, индексы и так далее). Также разрабатывали новые таблицы в Visio и экспортировали их в MySQL через MyODBC. Все работало.

### 3.13 Базисные шаги прикладной программы MyODBC

В общем виде, чтобы работать с сервером MySQL из любой программы через ODBC/MyODBC, надо сделать следующее:

- Настроить MyODBC DSN

- Подключиться к серверу MySQL

- Провести инициализацию

- Выполнить команды SQL

- Получить результаты

- Обработать транзакции

- Отсоединиться от сервера

Большинство прикладных программ использует некоторое изменение этих шагов.

### 3.14 Настройка MyODBC DSN

Источник данных идентифицирует путь для данных, который может включать сетевую библиотеку, сервер, базу данных и другие атрибуты. В нашем случае источник данных представляет собой путь к базе данных MySQL. Чтобы соединиться с источником данных, Driver Manager проверяет системный реестр Windows для получения специфической информации подключения.

ODBC Driver Manager и MyODBC Drivers использует вход системного реестра, созданный ODBC Data Source Administrator . Этот вход содержит информацию относительно каждого источника данных и связанного с ним драйвера. Прежде, чем Вы сможете соединяться с источником данных, информация о подключении должна быть добавлена к системному реестру.

Чтобы добавлять и конфигурировать источники данных, используйте ODBC Data Source Administrator. ODBC Administrator модифицирует

информацию о подключениях к источникам данных. Поскольку Вы добавляете источники данных, ODBC Administrator модифицирует информацию системного реестра для них.

Чтобы открыть ODBC Administrator из Control Panel:

Нажмите Start, укажите на Settings и щелкните Control Panel .

На системах под Microsoft Windows 2000 дважды щелкните по Administrative Tools, а затем дважды щелкните по Data Sources (ODBC). На компьютерах под предыдущими версиями Microsoft Windows дважды щелкните по 32-bit ODBC или по ODBC.

Чтобы добавить источник данных в Windows:

Откройте ODBC Data Source Administrator.

В диалоговом окне ODBC Data Source Administrator нажмите Add. Откроется диалоговое окно Create New Data Source.

Выберите там MySQL ODBC 3.51 Driver и нажмите на Finish. Появится диалоговое окно MySQL ODBC 3.51 Driver - DSN Configuration.

В окне Data Source Name введите имя источника данных, к которому Вы хотите обращаться. Это может быть любое имеющее силу имя, которое понравилось.

В окне Description введите описание необходимое для DSN.

В окне Host or Server Name (or IP) напечатайте имя сервера MySQL, к которому Вы хотите обращаться. По умолчанию это local host.

В окне Database Name укажите имя MySQL базы данных, которая будет применяться как заданная по умолчанию база данных.

В окне User задайте имя пользователя базы данных (user ID).

В окне Password надо задать пароль.

В окне Port напечатайте номер порта, если это не значение по умолчанию 3306.

В окне SQL Command Вы можете вводить факультативную команду SQL, которую серверу надлежит выполнить сразу после установления подключения.

Теперь нажмите ОК, чтобы добавить этот источник данных. Обратите внимание: при щелчке на ОК диалоговое окно Data Sources dialog, и ODBC Administrator модифицирует информацию системного реестра. Имя пользователя и строка подключения станут заданными по умолчанию значениями подключения для этого источника данных. Вы можете также проверить, достаточны ли Ваши параметры настройки, чтобы соединиться с сервером, используя кнопку Test Data Source. Эта возможность появилась только начиная с MyODBC 3.51.

Driver Options: Вы можете также видеть кнопку Options, которая отобразит диалог дополнительных параметров, которые управляют поведением драйвера.

Обратите внимание, что параметры Driver Trace Options будут заблокированы (нарисованы серым цветом) при использовании обычной версии DLL.

Чтобы изменить источник данных в Windows:

Откройте окно ODBC Data Source Administrator . Выберите соответствующую вкладку DSN.

Выберите источник данных MySQL, который Вы хотите изменить, а затем нажмите modify и щелкните по Configure. Откроется диалоговое окно MySQL ODBC 3.51 Driver - DSN Configuration.

Измените соответствующие поля источника данных, а затем нажмите ОК.

Когда Вы закончите изменять информацию в этом диалоговом окне, ODBC Administrator модифицирует информацию системного реестра.

Чтобы настроить источник данных в Unix:

В Unix Вы можете конфигурировать DSN-входы непосредственно в файле ODBC.INI. Имеется пример файла odbcs.ini с myodbc как DSN-имя для MyODBC 2.50 и myodbc3 для MyODBC 3.51 Drivers:

;

```
; odbc.ini configuration for MyODBC and MyODBC 3.51  
Drivers
```

```
;
```

```
[ODBC Data Sources]
```

```
myodbc = MySQL ODBC 2.50 Driver DSN
```

```
myodbc3 = MySQL ODBC 3.51 Driver DSNa
```

```
[myodbc]
```

```
Driver      = /usr/local/lib/libmyodbc.so
```

```
Description = MySQL ODBC 2.50 Driver DSN
```

```
SERVER      = localhost
```

```
PORT        =
```

```
USER        = root
```

```
Password    =
```

```
Database    = test
```

```
OPTION      = 3
```

```
SOCKET      =
```

```
[myodbc3]
```

```
Driver      = /usr/local/lib/libmyodbc3.so
```

```
Description = MySQL ODBC 3.51 Driver DSN
```

```
SERVER      = localhost
```

```
PORT        =
```

```
USER        = root
```

```
Password    =
```

```
Database    = test
```

```
OPTION      = 3
```

```
SOCKET      =
```

```
[Default]
```



```

Driver      = /usr/local/lib/libmyodbc3.so
Description = MySQL ODBC 3.51 Driver DSN
SERVER      = localhost
PORT        =
USER        = root
Password    =
Database    = test
OPTION      = 3
SOCKET      =

```

Обратите внимание: если Вы используете unixODBC, то Вы можете использовать следующие инструментальные средства чтобы настроить DSN:

ODBCConfig GUI tool ( HOWTO : ODBCConfig)

odbcinst

### 3.15 Параметры подключения

Можно определять следующие параметры для MyODBC или для MySQL ODBC 3.51 в секции [Data Source Name] файла ODBC.INI или через параметр InConnectionString в вызове SQLDriverConnect().

Параметр	Значение по умолчанию	Комментарий
user ODBC (в Windows)		Имя пользователя для связи с MySQL.
server	localhost	Имя сервера MySQL.
database		База данных по умолчанию
option	0	Целое число, которым Вы можете определять как должен работать MyODBC 3.51. Описано чуть ниже.
port	3306	Порт TCP/IP, чтобы использовать, если server не равен localhost.
stmt		Инструкция, которая будет выполнена, когда установлено подключение к MySQL.
password		Пароль для комбинации server user.

socket

Сокет или именованный канал Windows для связи.

Параметр OPTION используется, чтобы сообщить MyODBC 3.51, что пользователь не на 100% совместим с ODBC. Следующие параметры перечислены в том же самом порядке, в каком они появляются в MyODBC

3.51:Бит Описание

1 Пользователь не может обрабатывать ситуацию, когда MyODBC возвращает реальную ширину столбца.

2 Пользователь не может обрабатывать ситуацию, когда MySQL возвращает истинное число обработанных строк. Если этот параметр установлен, MySQL вернет число найденных строк. Нужно иметь MySQL 3.21.14 или более новый, чтобы это работало.

4 Создать протокол трассировки в файле c:\myodbc.log (/tmp/myodbc.log). Это аналогично указанию MYSQL\_DEBUG=d:t:O,c::\myodbc.log в AUTOEXEC.BAT.

8 Не устанавливать ограничений пакета для результатов и параметров.

16 -Не запрашивать ничего, даже если драйвер хочет запросить.

32 Включить или отключить поддержку динамического курсора. Это не работает в MyODBC 2.50.

64 Игнорировать использование имени базы данных в формате database.table.column.

128 Использование экспериментальных курсоров ODBC manager.

256 Отключить использование расширенной (экспериментальной) выборки.

512 Дополнить поля типа CHAR до полной длины столбца.

1024SQLDescribeCol() возвратит полностью квалифицированные имена столбцов.

2048Использовать сжатый протокол клиент-сервер.

4096Сервер должен игнорировать пробел между именем функции и '('

(нужно для Power Builder). Это делает все ключевые слова именами функций!

8192 Соединиться через именованный канал с сервером mysqld под NT.

16384 Менять столбцы типа LONGLONG на столбцы INT (некоторые прикладные программы не могут корректно обрабатывать LONGLONG).

32768 Вернуть user как Table\_qualifier и Table\_owner из SQLTables.

65536 Читать параметры из групп client и odbc в файле my.cnf

131072 Добавить некоторые дополнительные проверки безопасности (вроде бы не очень и надо, но...).

262144 Выключить использование транзакций

524288 Включить регистрацию запросов в файле c:\myodbc.sql (/tmp/myodbc.sql). Доступно только в режиме отладки в специальной версии драйвера.

Если Вы хотите иметь много параметров, Вы должны сложить вышеупомянутые числа. Например, установка опции в 12 (4+8) дает Вам отладку без ограничений на размеры пакета.

По умолчанию MYODBC3.DLL компилируется для оптимальной эффективности. Если Вы хотите отладить MyODBC 3.51 (например, чтобы получить трассировку), используйте MYODBCD3.DLL вместо стандартного файла MYODBC3.DLL.

### 3.16 Связь с сервером MySQL

Прикладная программа может быть связана с любым числом источников данных и драйверов. Они могут быть вариантами того же самого драйвера и ряда источников данных или несколькими подключениями с тем же самым драйвером и источником данных. Прикладная программа должна сделать следующее, чтобы соединиться с сервером MySQL через MyODBC:

Распределите дескриптор среды

Установите версию ODBC

Распределите дескриптор подключения

Установите факультативные атрибуты подключения перед подключением

Создайте подключение к серверу

Установите факультативные атрибуты подключения после подключением

### 3.16.1 3.4.1 Распределение дескриптора среды

Прежде, чем прикладная программа сможет использовать любую функцию ODBC, надо инициализировать ODBC-связь с помощью интерфейса и сопоставить с ней дескриптор среды. Он обеспечивает доступ к глобальной информации типа имеющих силу дескрипторов подключения и активных дескрипторов подключения.

Чтобы распределить правильный дескриптор среды, прикладная программа:

Объявляет переменную типа SQLHENV. Например, прикладная программа могла бы использовать объявление:

```
SQLHENV henv;
```

Вызывает SQLAllocHandle (в MyODBC 2.50 называется SQLAllocEnv) и передает адрес этой переменной и опции SQL\_HANDLE\_ENV как:

```
SQLAllocHandle(SQL_HANDLE_ENV, SQL_NULL_HANDLE, &henv)
```

или

```
SQLAllocEnv(&henv)
```

Если прикладная программа связана через Driver Manager, то это обращение загружает Driver Manager. Он не вызывает SQLAllocHandle в драйвере потому, что пока не знает, который драйвер вызвать. Это откладывает вызов SQLAllocHandle в драйвере до получения вызовов из прикладной программы, чтобы соединиться с источником данных: тогда-то будет однозначно ясно, какой драйвер нужен.

Если прикладная программа связана непосредственно с драйвером, то это обращение загружает драйвер, и уже драйвер формирует информацию

среды и возвращает распределенную структуру обратно прикладной программе.

### 3.16.2 3.4.2 Установка версии ODBC

Если Вы используете драйвер MyODBC 2.50, то Вы можете игнорировать этот раздел. Прежде, чем прикладная программа создаст соединение, необходимо установить атрибут SQL\_ATTR\_ODBC\_VERSION среды, используя SQLSetEnvAttr:

```
SQLSetEnvAttr(henv, SQL_ATTR_ODBC_VERSION, (SQLPOINTER)
SQL_OV_ODBC3, 0);
```

Этот атрибут заявляет, что прикладная программа следует спецификациям ODBC 2.x или ODBC 3.x при использовании следующих элементов:

**SQLSTATE:** Много значений SQLSTATE различны в ODBC 2.x и ODBC 3.x. Для получения списка кодов SQLSTATE, возвращаемых драйвером MyODBC 3.51 обратитесь к разделу " Коды ошибок MyODBC".

**Типы Date, Time и Timestamp:** следующая таблица показывает идентификаторы типов для данных date, time и timestamp в ODBC 2.x и в ODBC 3.x.

ODBC 2.X	ODBC 3.X
SQL_DATE	SQL_TYPE_DATE
SQL_TIME	SQL_TYPE_TIME
SQL_TIMESTAMP	SQL_TYPE_TIMESTAMP

#### Идентификаторы типов в SQL

SQL_DATE	SQL_TYPE_DATE
SQL_TIME	SQL_TYPE_TIME
SQL_TIMESTAMP	SQL_TYPE_TIMESTAMP

#### Идентификаторы типов в C

SQL_C_DATE	SQL_C_TYPE_DATE
SQL_C_TIME	SQL_C_TYPE_TIME
SQL_C_TIMESTAMP	SQL_C_TYPE_TIMESTAMP

MyODBC 3.51 контролирует версию спецификации ODBC, для которой

прикладная программа написана и отвечает соответственно. Например, если прикладная программа следует версии ODBC 2.x и вызывает SQLExecute до вызова SQLPrepare, драйвер вернет: SQLSTATE S1010 (Function sequence error). Если прикладная программа поддерживает спецификацию ODBC 3.x, то это возвращает: SQLSTATE HY010 (Function sequence error).

### 3.16.3 Распределение дескриптора подключения

Дескриптор подключения обеспечивает доступ к информации относительно того, является ли подключение открытым или нет, имеют ли силу операторные и дескрипторные маркеры на подключении, и открыта ли сейчас транзакция.

Прежде, чем прикладная программа сможет соединиться с сервером MySQL или с драйвером, она должна распределить дескриптор подключения, следующим образом:

Прикладная программа объявляет переменную типа SQLHDBC.

Она затем вызывает SQLAllocHandle (или SQLAllocConnect для версии MyODBC 2.50) и передает адрес этой переменной, дескриптор среды, чтобы распределить подключение, и опцию SQL\_HANDLE\_DBC. Например:

```
SQLHDBC hdbc;  
SQLAllocHandle(SQL_HANDLE_DBC, henv, &hdbc); или  
SQLAllocConnect(henv, &hdbc);
```

Если прикладная программа связана через Driver Manager, то Driver Manager распределяет память, чтобы сохранить информацию относительно подключения и возвращает дескриптор подключения прикладной программе. С другой стороны, если Вы непосредственно компонуете программу через библиотеку драйверов вместо Driver Manager, то эту работу делает уже драйвер.

### 3.16.4 Установка атрибутов соединения (подключения)

Атрибуты подключения представляют собой характеристики подключения. Например, они определяют, что транзакции происходят в уровне подключения, а уровень изоляции транзакции представляет собой атрибут подключения. Точно так же время ожидания входа в систему или число секунд, которые надо ждать при попытке соединиться перед тайм-аутом, тоже атрибуты подключения.

Атрибуты подключения установлены с помощью `SQLSetConnectAttr`, а их текущие параметры настройки могут быть получены с помощью `SQLGetConnectAttr`. Для прикладных программ драйвера MyODBC 2.50 Вы можете использовать `SQLSetConnectOption` и `SQLGetConnectOption`.

Атрибуты подключения могут быть установлены до или после подключения, в зависимости от типа атрибута. Время ожидания входа в систему `SQL_ATTR_LOGIN_TIMEOUT` применяется только при установлении связи и важно, только если установлено перед соединением.

Атрибуты, которые определяют, использовать ли библиотеку курсоров ODBC (`SQL_ATTR_ODBC_CURSORS`) и сетевой размер пакета (`SQL_ATTR_PACKET_SIZE`), должны быть установлены прежде, чем соединение создано потому что, библиотека курсоров ODBC находится между Driver Manager и драйвером, а следовательно должно быть загружена перед драйвером. Подробный перечень атрибутов подключения, поддерживаемых драйверами MyODBC, есть в разделе "4.5.1 `SQLSetConnectAttr`".

### 3.16.5 Установление подключения, использующего MyODBC

После распределения среды и дескрипторов подключения и установки факультативных атрибутов подключения, прикладная программа готова соединиться с сервером MySQL или драйвером MyODBC (через Driver Manager). Имеются две различных функции для этого:

`SQLConnect` и

## SQLDriverConnect

### Соединение через SQLConnect

SQLConnect самая простая функция подключения. Требуется имя источника данных и принимает факультативные user ID и пароль. Прикладная программа передает следующую информацию драйверу через SQLConnect:

DSN: имя источника данных.

UID: имя пользователя для связи с сервером (опционально).

PWD: соответствующий пароль (опционально).

Обратите внимание, что, если Вы уже определили имя пользователя и пароль в параметрах DSN или непосредственно в файле ODBC.INI, Вы можете только определить имеющий силу DSN, а драйвер внутренне получает другую требуемую информацию из записей в DSN сам.

Когда из прикладной программы вызван SQLConnect, Driver Manager использует имя источника данных, чтобы прочитать имя драйвера DLL из соответствующего раздела файла ODBC.INI или из системного реестра. Это затем загружает драйвер DLL и передает ему параметры SQLConnect. Если драйвер нуждается в дополнительной информации, чтобы соединиться с источником данных, он читает эту информацию из того же самого раздела файла ODBC.INI.

Если прикладная программа определяет имя источника данных, которое не значится в файле ODBC.INI или в системном реестре, или если прикладная программа не определяет имя источника данных, Driver Manager ищет заданную по умолчанию спецификацию источника данных. Если он находит заданный по умолчанию источник данных, то загружает заданный по умолчанию драйвер и передает ему определенное прикладной программой имя источника данных. Если не имеется никакого заданного по умолчанию источника данных, Driver Manager возвращает соответствующую ошибку.



Пример: следующий пример распределяет необходимую среду, дескриптор подключения и соединяется с сервером MySQL, используя DSN myodbc3.

```
SQLHENV      henv;
SQLHDBC      hdbc;
SQLHSTMT     hstmt;
SQLRETURN    retcode;

/* Allocate environment handle */
retcode = SQLAllocHandle(SQL_HANDLE_ENV, SQL_NULL_HANDLE,
&henv);
if (retcode == SQL_SUCCESS || retcode ==
SQL_SUCCESS_WITH_INFO)
{
    /* Set the ODBC version environment attribute to
version 3 */
    retcode = SQLSetEnvAttr(henv, SQL_ATTR_ODBC_VERSION,
                           (SQLPOINTER)SQL_OV_ODBC3, 0);
    if (retcode == SQL_SUCCESS || retcode ==
SQL_SUCCESS_WITH_INFO)
    {
        /* Allocate connection handle */
        retcode = SQLAllocHandle(SQL_HANDLE_DBC, henv,
&hdbc);
        if (retcode == SQL_SUCCESS || retcode ==
SQL_SUCCESS_WITH_INFO)
        {
            /* Connect to data source myodbc3 */
            retcode = SQLConnect(hdbc, (SQLCHAR*) "myodbc3",
SQL_NTS,
```

```

                                (SQLCHAR*) "myuser",
SQL_NTS,
                                (SQLCHAR*) "mypassword",
SQL_NTS);
    if (retcode == SQL_SUCCESS || retcode ==
SQL_SUCCESS_WITH_INFO)
    {
        /* Set auto commit to ON */
        retcode = SQLSetConnectAttr(hdbc,
SQL_ATTR_AUTO_COMMIT,

SQL_AUTOCOMMIT_ON, 0);
        printf("\n autocommit returned :%d",
redcode);

        /* Allocate statement handle */
        retcode = SQLAllocHandle(SQL_HANDLE_STMT,
hdbc, &hstmt);
        if (retcode == SQL_SUCCESS || retcode ==
SQL_SUCCESS_WITH_INFO)
        {
            /* Process data */

            ;

            ;

            ;

            /* Free statemt handle */
            SQLFreeHandle(SQL_HANDLE_STMT, hstmt);
        }
        /* Disconnect from the server */
        SQLDisconnect(hdbc);
    }

```

```

        /* Close the connection handle */
        SQLFreeHandle(SQL_HANDLE_DBC, hdbc);
    }
}
/* Close the environment handle */
SQLFreeHandle(SQL_HANDLE_ENV, henv);

```

### **Связь через SQLDriverConnect**

SQLDriverConnect используется, чтобы соединиться с сервером, используя строку подключения. Можно использовать SQLDriverConnect вместо SQLConnect по следующим причинам:

Позволить прикладной программе использовать специфическую для драйвера информацию подключения.

Чтобы драйвер запрашивал пользователя относительно информации подключения.

Соединяться без определения источника данных (DSN less connection).

Строка подключения может состоять из одного или большего количества параметров MyODBC подключения, отделяемых точкой с запятой (;). Если драйвер должен запрашивать пользователя относительно информации подключения, то он отображает диалог подключения.

### **Строка подключения для SQLDriverConnect**

Используя myodbc3 как MySQL ODBC 3.51 DSN:

```

ConnectionString = "DSN=myodbc3"
DSN Less Connection:
ConnectionString = "DRIVER={MySQL ODBC 3.51 Driver};
SERVER=localhost;\
                        DATABASE=test; USER=monty;
PASSWORD=monty;\

```

OPTION=4 ; "

### 3.16.6 Получение информации о драйвере и источнике данных

Как только подключение установлено, прикладная программа должна получить большее количество информации относительно драйвера и источника данных, с которым он связан. Использование следующего API поможет это устроить:

**SQLGetInfo:** возвращает общую информацию относительно драйвера и источника данных, связанного с подключением. Например, какие инструкции SQL прикладная программа выполнит? Прикладная программа использует скроллируемые курсоры? Транзакции? Процедуры? Длинные данные?

**SQLGetTypeInfo:** возвращает информацию относительно типов данных, поддерживаемых сервером. Драйвер возвращает информацию в форме набора результатов SQL. Типы данных предназначены для использования в инструкциях Data Definition Language (DDL).

**SQLGetFunctions:** возвращает информацию относительно того, поддерживает ли драйвер специфическую функцию ODBC. Прикладная программа может всегда использовать эту функцию, чтобы проверить, поддерживает ли драйвер некий API или нет.

**Пример:** получает имя драйвера и версию, имя и версию сервера и соглашения SQL, поддерживаемые драйвером.

```
SQLHDBC      hdbc;
SQLRETURN    retcode;
SQLCHAR      strValue[50];
SQLINTEGER   nValue;
SQLSMALLINT  pcbValue;

/* Connect to the server */
retcode = SQLConnect (..)
```



```

    if (retcode == SQL_SUCCESS || retcode ==
SQL_SUCCESS_WITH_INFO)
    {
        printf("SQL Conformance:%d",nValue);
    }
}

```

### 3.16.7 3.4.7 Прерывание соединения

Когда прикладная программа закончила использовать сервер MySQL, она должна в обязательном порядке закрыть подключение и освободить все предварительно распределенные дескрипторы. Чтобы завершать подключение из MyODBC нужно:

Вызвать `SQLDisconnect`, чтобы закрыть подключение. Если имеются любые открытые операторные дескрипторы на этом подключении, то драйвер внутренне освобождает все открытые инструкции для этого подключения. Прикладная программа может затем использовать дескриптор подключения, чтобы повторно соединиться с тем же самым источником данных или присоединиться к другому источнику данных, если дескриптор подключения не был освобожден.

Вызвать `SQLFreeHandle` с опцией `SQL_HANDLE_DBC`, чтобы освободить подключение и все ресурсы, связанные с дескриптором.

Вызвать `SQLFreeHandle` с опцией `SQL_HANDLE_ENV`, чтобы освободить среду и все ресурсы, связанные с дескриптором.

Обратите внимание, если Вы используете драйвер MyODBC 2.50, Вы должны использовать `SQLFreeConnect` и `SQLFreeEnv`, чтобы освободить дескрипторы подключения и среды соответственно.

## 3.17 Выполнение команд SQL

Ну ладно, подключение установлено, а дальше-то что? Надо работать с сервером, для этого все и затевалось. Работа эта происходит на базе обмена

командами SQL и их результатами. Вот это самое сложное. Прикладная программа может представлять на рассмотрение любую инструкцию SQL, поддерживаемую сервером MySQL. ODBC-программы выполняют почти весь доступ к базе данных, выполняя инструкции SQL. Общая последовательность событий:

Распределите операторный дескриптор  
 Установите факультативные операторные атрибуты,  
 Выполните инструкцию,  
 Соберите все результаты и наконец  
 Освободите операторный дескриптор.

### 3.17.1 Распределение операторного дескриптора

Операторный дескриптор обеспечивает доступ к операторной информации, типа сообщений об ошибках, имени курсора и информации состояния для обработки инструкции SQL. Прежде, чем прикладная программа сможет представлять на рассмотрение серверу инструкцию SQL, она должна распределить операторный дескриптор, используя SQLAllocHandle (или SQLAllocStmt в MyODBC 2.50):

Прикладная программа объявляет переменную типа HSTMT. Это затем вызывает SQLAllocHandle и передает адрес этой переменной, дескриптор подключения, чтобы распределить инструкцию, и опцию SQL\_HANDLE\_STMT:

```
SQLHSTMT hstmt;
SQLAllocHandle(SQL_HANDLE_STMT, hdbc, &hstmt) или
SQLAllocStmt(hdbc, &hstmt)
```

Driver Manager распределяет структуру, чтобы сохранить информацию относительно инструкции и вызывает SQLAllocHandle в драйвере с опцией SQL\_HANDLE\_STMT.

Драйвер распределяет собственную структуру, чтобы сохранить информацию относительно инструкции и возвращает дескриптор инструкции

драйвера назад в Driver Manager. С другой стороны, если Вы компонуете программу непосредственно с драйвером, то именно сам драйвер распределяет операторную структуру и возвращает ее адрес обратно прикладной программе.

Driver Manager возвращает дескриптор инструкции прикладной программе.

Драйвер идентифицирует, которую инструкцию надо использовать при вызове функций ODBC через дескриптор инструкции.

### **3.17.2 Установка операторных атрибутов**

Операторные атрибуты представляют собой характеристики инструкции. Например, они используются, чтобы установить имя курсора для специфической инструкции или задать максимальное количество строк, которые будут выбраны в одной инструкции выборки.

Операторные атрибуты могут быть установлены с помощью SQLSetStmtAttr, а их актуальные параметры настройки можно узнать через вызов SQLGetStmtAttr (SQLSetStmtOption и SQLGetConnectOption соответственно для MyODBC 2.50). Поскольку решительно все операторные атрибуты имеют значения по умолчанию, прикладная программа не обязана их менять, можно оставить все как есть.

### **3.17.3 Передача на рассмотрение инструкций SQL**

MyODBC позволяет прикладной программе представлять на рассмотрение инструкции SQL двумя различными способами:

Подготовленное выполнение

Прямое выполнение

#### **Подготовленное выполнение**

Подготовленное выполнение представляет собой эффективный способ выполнить инструкцию больше одного раза. Инструкция сначала компилируется в план доступа. План доступа затем будет выполнен столько



раз, сколько понадобится.

Подготовленное выполнение более предпочтительно, если прикладная программа:

Выполняет инструкцию больше одного раза, меняя значения параметра.

Нуждается в информации относительно инструкции SQL или набора результатов до выполнения.

Подготовленное выполнение главным образом достигнуто через MySQL API `SQLPrepare` и `SQLExecute`. Подготовленная инструкция выполняется быстрее, чем неприготовленная инструкция или прямое выполнение потому, что драйвер компилирует инструкцию, строит для нее план доступа и возвращает идентификатор плана доступа обратно прикладной программе. Драйвер минимизирует затраты времени на обработку инструкции, поскольку он не должен каждый раз строить план доступа. Уменьшается и трафик.

Чтобы подготовить и выполнить инструкцию, прикладная программа:

Вызывает `SQLPrepare` и передает строку, содержащую инструкцию SQL.

Устанавливает значения любых операторных параметров.

Вызывает `SQLExecute` и делает любую дополнительную обработку, которая является необходимой, типа выборки данных.

По мере надобности повторяет 2 и 3 шаги.

Когда вызвана `SQLPrepare`, драйвер изменяет инструкцию SQL, чтобы использовать синтаксис MySQL без того, чтобы анализировать инструкцию. Это включает замену управляющих последовательностей. Но драйвер не возвращает никаких синтаксических и семантических ошибок.

При вызове `SQLExecute` драйвер:

Получает текущий параметр, оценивает и преобразует его по мере необходимости.

Посылает идентификатор плана доступа и преобразованные значения параметров на сервер MySQL.

Возвращает любые ошибки. Это ошибки, возникшие во время

выполнения программы, типа SQLSTATE 24000 (Invalid cursor state), а также синтаксические и семантические ошибки, если они есть.

Пример: этот пример объясняет, как прикладная программа может использовать подготовленное выполнение. Выборка готовит инструкцию INSERT и вставляет 100 строк данных, заменяя буферные значения.

```
SQLHSTMT hstmt;
SQLRETURN retcode;
retcode = SQLPrepare(hstmt, "INSERT INTO EMP(ID,NAME)
VALUES(?,?)", SQL_NTS);
if (retcode == SQL_SUCCESS || retcode ==
SQL_SUCCESS_WITH_INFO)
{
    SQLINTEGER id;
    SQLCHAR name[30];
    /* do the binding for parameter 1, id */
    retcode = SQLBindParameter(hstmt,1,SQL_PARAM_INPUT,
SQL_C_LONG,
                                SQL_INTEGER, 0,0, &id, 0,
NULL);
    if (retcode == SQL_SUCCESS || retcode ==
SQL_SUCCESS_WITH_INFO)
    {
        /* Now do the bindings for parameter 2, name */
        retcode = SQLBindParameter(hstmt,1,SQL_PARAM_INPUT,
SQL_C_CHAR,
                                SQL_VARCHAR, 0,0, name,
sizeof(name),NULL);
        if (retcode == SQL_SUCCESS || retcode ==
SQL_SUCCESS_WITH_INFO)
        {
```

```

        /* Now insert data by changing id and name
buffer values */
        for (id=1; id <= 100; id++)
        {
            /* Set name as Cmysql1T, Cmysql2TE */
            sprintf(name, "mysql%d", id);
            retcode = SQLExecute(hstmt);
        }
    }

    /* Free param buffer resources */
    retcode = SQLFreeStmt(hstmt, SQL_REST_PARAMS);
}

```

### **Прямое выполнение**

Прямое выполнение представляет собой самый простой способ выполнить инструкцию. Прямое выполнение обычно используется универсальными прикладными программами, которые формируют и выполняют инструкции во время выполнения. Например, следующий код формирует инструкцию SQL и выполняет ее один раз:

```

SQLCHAR *statement;

// Build an SQL statement.
printf("enter the SQL statement:");
scanf("%s", &statement);

// Execute the statement.
SQLExecDirect (hstmt, statement, SQL_NTS);

```

Прикладная программа должна выполнить инструкции, используя именно прямое выполнение, если:

Инструкция нужна однократно.

Прикладная программа не нуждается в информации относительно набора результатов до выполнения.

Основной недостаток использования прямого выполнения: инструкция SQL анализируется каждый раз, когда выполняется.

Чтобы выполнить инструкцию непосредственно, прикладная программа выполняет следующий набор действий:

Устанавливает значения любых параметров.

Вызывает `SQLExecDirect` и передает строку, содержащую инструкцию SQL.

При вызове `SQLExecDirect` драйвер:

Изменяет инструкцию SQL, чтобы использовать синтаксис MySQL без того, чтобы анализировать инструкцию. Это включает замену всех управляющих последовательностей языка.

Получает актуальный параметр, оценивает его и изменяет инструкцию SQL, меняя маркеры параметра на данные с соответствующими преобразованиями.

Посылает измененную инструкцию SQL MySQL для выполнения.

Возвращает любые ошибки. Они включают диагностику выполнения, например, `SQLSTATE 24000 (Invalid cursor state)`, синтаксические ошибки, типа `SQLSTATE 42000 (Syntax error or access violation)` и семантические ошибки, вроде `SQLSTATE 42S02 (Base table or view not found)`.

Пример:

этот пример объясняет, как прикладная программа может использовать прямое выполнение. Он создает таблицу, вставляет, модифицирует и удаляет

некоторые строки данных, а в заключение удаляет всю таблицу.

```
SQLHSTMT hstmt;
SQLRETURN retcode;

/* create table as "my_test" with integer and text field
*/
retcode = SQLExecDirect(hstmt, "CREATE TABLE my_test(id
int,
                        name text", SQL_NTS);
if (retcode == SQL_SUCCESS || retcode ==
SQL_SUCCESS_WITH_INFO)
{
    printf("table created successfully..");
    /* insert 2 rows of data to the table Cmy_testT */
    retcode = SQLExecDirect(hstmt, "INSERT INTO my_test
VALUES(10,'mysql')",
                        SQL_NTS);
    if (retcode == SQL_SUCCESS || retcode ==
SQL_SUCCESS_WITH_INFO)
    {
        printf("row 1 inserted successfully..");
    }
    retcode = SQLExecDirect(hstmt, "INSERT INTO my_test
VALUES(20,'monty')",
                        SQL_NTS);
    if (retcode == SQL_SUCCESS || retcode ==
SQL_SUCCESS_WITH_INFO)
    {
        printf("row 2 inserted successfully..");
    }
}
```

```

/* Now update the second row by changing id from 20 to
100 */
    retcode = SQLExecDirect(hstmt, "UPDATE my_test SET
id=100
                                WHERE name='monty', SQL_NTS);
    if (retcode == SQL_SUCCESS || retcode ==
SQL_SUCCESS_WITH_INFO)
    {
        SQLINTEGER rowcount;
        printf("row updated successfully..");
        /* Get total number of rows affected by the update
statement */
        retcode=SQLRowCount(hstmt, &rowcount);
        printf("total rows affected by the updated
statement:%d",rowcount);
    }
    /* Now delete the newly updated row */
    retcode = SQLExecDirect(hstmt, "DELETE FROM my_test
WHERE id=100",
                                SQL_NTS);
    if (retcode == SQL_SUCCESS || retcode ==
SQL_SUCCESS_WITH_INFO)
    {
        SQLINTEGER rowcount;
        printf("row deleted successfully..");
        /* Get total number of rows affected by the delete
statement */
        retcode=SQLRowCount(hstmt, &rowcount);
        printf("total rows affected by the delete
statement:%d",rowcount);
    }

```

```

    }
}
/* now drop the table Cmy_testT */
retcode = SQLExecDirect(hstmt, "DROP TABLE my_test",
SQL_NTS);
if (retcode == SQL_SUCCESS || retcode ==
SQL_SUCCESS_WITH_INFO)
{
    printf("table dropped successfully");
}

```

### **Операторные параметры**

Параметром является переменная в инструкции SQL. Инструкция SQL может содержать маркеры параметров ("?"), которые указывают значения, которые драйвер получает из прикладной программы во время выполнения.

Например, прикладная программа могла бы использовать следующую инструкцию, чтобы вставить строку данных в таблицу EMPLOYEE:

```
INSERT INTO EMPLOYEE (NAME.AGE) VALUES (?,?)
```

Прикладная программа может использовать маркеры параметров вместо литеральных или постоянных значений в инструкции SQL по следующим причинам:

Требуется выполнить ту же самую подготовленную инструкцию несколько раз с различными значениями параметра.

Значения параметра неизвестны, когда инструкция готовится.

Значения параметра должны быть явно преобразованы из одного типа данных в другой.

Чтобы устанавливать значение параметра, прикладная программа просто устанавливает значение переменной, привязанной к этому параметру, используя `SQLBindParameter`. Неважно, когда это значение установлено, пока

это сделано прежде, чем инструкция выполнена. Прикладная программа может устанавливать значение в любое время и менять его столько раз, сколько потребуется.

Когда инструкция выполнена, драйвер просто получает актуальное значение переменной. Это особенно полезно, когда подготовленная инструкция выполнена больше, чем однажды: прикладная программа устанавливает новые значения для некоторых или всех переменных, каждый раз, когда инструкция выполнена.

Если буфер длин использован в вызове `SQLBindParameter`, он должен быть установлен в одно из следующих значений прежде, чем инструкция выполнена:

Длина данных в байтах в связанной переменной. Драйвер проверяет эту длину только, если переменная символьная или двоичная (`ValueType` равен `SQL_C_CHAR` или `SQL_C_BINARY`).

`SQL_NTS`. Данные являются строкой с нулевым символом в конце.

`SQL_NULL_DATA`. Значение данных равно `NULL`, и драйвер игнорирует значение связанной переменной.

`SQL_DATA_AT_EXEC` или результат макрокоманды `SQL_LEN_DATA_AT_EXEC`. Значение параметра должно быть послано с `SQLPutData`.

Расположения параметров, заданные через `SQLBindParameter`, останутся привязанными к маркерам параметра до вызова `SQLFreeStmt` из прикладной программы с опцией `SQL_RESET_PARAMS` или `SQL_DROP`. Прикладная программа может связать новое место в памяти с маркером параметра в любое время, вызывая `SQLBindParameter`. Пример:

```
SQLINTEGER    id;
```



```

SQLINTEGER      idInd;

// Prepare a statement to insert id
SQLPrepare(hstmt, "INSERT INTO my_table VALUES(?)",
SQL_NTS);

// Bind id to the parameter for the id column
SQLBindParameter(hstmt, 1, SQL_PARAM_INPUT, SQL_C_ULONG,
                  SQL_LONG, 0, 0, &id, 0, &idInd);

// Repeatedly execute the statement, to insert 100 rows
of data
for (id=1; id <= 100; id++)
{
    SQLExecute(hstmt);
}

```

### **Передача данных Long или Blob**

MySQL определяет данные long как любые символьные или двоичные данные, превышающие некий размер, обычно 254 символа. Не всегда реально сохранить в памяти целиком элемент длинных данных. Пример: здоровенный текстовый документ (например, эта книга в типографском формате PostScript) или растровая картинка. А поскольку такие данные не могут быть сохранены в одиночном буфере, прикладная программа посылает их драйверу по частям через SQLPutData, когда инструкция выполнена.

Обратите внимание, что прикладная программа может фактически посылать любой тип данных во времени выполнения с помощью SQLPutData, хотя только символьные и двоичные данные могут быть представлены частями. Однако, если данные достаточно маленькие, чтобы разместиться в одиночном

буфере, не имеется вообще никакой причины использовать SQLPutData. Намного проще позволять драйверу получать данные из буфера напрямую.

Когда Вы должны ввести большие количества данных в столбец long varchar или в long varbinary, Вы можете использовать ODBC-функции SQLPutData и SQLParamData, чтобы ввести данные в меньших сегментах. Данные обеспечены в сегментах через SQLPutData, а SQLParamData используется, чтобы проверить требуют ли параметры данных.

Чтобы посылать длинные данные во время выполнения в сегментах, прикладная программа выполняет следующие действия:

Готовит SQL-инструкцию с маркерами параметра там, где будут данные long или blob. Используется SQLPrepare.

Устанавливает параметр pcbValue в функции SQLBindParameter в значение SQL\_DATA\_AT\_EXEC или SQL\_LEN\_DATA\_AT\_EXEC. Это позволяет драйверу узнать, что Вы будете обеспечивать значения для этого параметра во время выполнения, используя SQLPutData.

Выполняет команду SQL. Если инструкция уже подготовлена, выполняется подготовленная инструкция, используя SQLExecute или SQLExecDirect. Если имеются любые параметры, которые должны получить данные во времени выполнения, то драйвер возвращает SQL\_NEED\_DATA.

Вызывает SQLParamData в ответ на возврат значения SQL\_NEED\_DATA. Если длинные данные должны быть посланы, SQLParamData вернет SQL\_NEED\_DATA. В буфере, указанном параметром ValuePtrPtr, драйвер возвращает значение, которое идентифицирует параметр ожидания данных при выполнении. Если имеется больше, чем один такой параметр, прикладная программа должна использовать это значение, чтобы определить, который параметр ожидается. Заметьте, что данные могут быть запрошены драйвером в любом порядке.

Вызывает SQLPutData:, чтобы послать данные параметра драйверу. Если

данные параметра не вписываются в одиночный буфер, что часто имеет место с длинными данными, вызовы `SQLPutData` из прикладной программы будут повторяться для передачи последующих порций данных.

Вызывает `SQLParamData`: если код возврата равен `SQL_NEED_DATA`, следующий параметр, который ожидает данные во время выполнения, готов их получить, и Вы должны вернуться к шагу 4. Если код возврата равен `SQL_SUCCESS` или хотя бы `SQL_SUCCESS_WITH_INFO`, все данные для всех параметров, ожидающих данных во время выполнения, посланы, и инструкция SQL завершила выполнение.

### 3.5.4 Освобождение операторного дескриптора

Перед выполнением новой инструкции SQL, прикладная программа должна убедиться, что текущие операторные параметры настройки соответствующие. Они включают операторные атрибуты, связанные параметры и наборы результатов. Вообще, параметры и наборы результатов для старой инструкции SQL должны быть освобождены (вызовом `SQLFreeStmt` с опцией `SQL_RESET_PARAMS` или `SQL_UNBIND`).

Когда прикладная программа закончила использовать инструкцию, она вызывает `SQLFreeHandle` с опцией `SQL_HANDLE_STMT` или `SQLFreeStmt` с опцией `SQL_DROP`, чтобы освободить инструкцию. Вызов `SQLDisconnect` автоматически освобождает все инструкции для данного подключения.

Функция `SQLFreeStmt` имеет четыре опции: Опция Что она делает

`SQL_CLOSE` Закрывает курсор, если он существует, и отбрасывает ждущие обработки результаты. Прикладная программа может использовать операторный дескриптор позже.

`SQL_DROP` Закрывает курсор, если он существует, отбрасывает ждущие обработки результаты и освобождает все ресурсы, связанные с операторным дескриптором.

`SQL_UNBIND` Освобождает все буфера возвратов, связанные `SQLBindCol` с операторным дескриптором.

`SQL_RESET_PARAMS` Освобождает все буфера параметров, запрошенные `SQLBindParameter` для операторного дескриптора.