Типы таблиц в MySQL

Лекция 14

Тип таблицы определяет механизм хранения данных, так как некоторые типы таблиц используют специальные связанные с ними программы для доступа к диску, кэшированию, индексации и блокировки.

База данных MySQL поддерживает два различных вида таблиц: транзакционные таблицы (transaction-safe tables) типа InnoDB и BDB и таблицы без поддержки транзакций (non-transaction-safe tables) типа MEMORY (HEAP), MyISAM (ISAM) и MERGE.

MyISAM - данный_тип таблицы не поддерживает транзакции.
MERGE - объединение нескольких идентичных таблиц типа MyISAM.
MEMORY (HEAP) - таблицы хранятся в памяти для быстрого доступа.
BDB (BerkeleyDB) - транзакционный тип таблиц, начиная с MySQL 5.1 не поддерживается.
InnoDB - транзакционный тип таблиц с поддержкой внешних ключей.
FEDERATED - таблицы предназначены для доступа к данным удаленного MySQL сервера как к локальным таблицам.
ARCHIVE - таблицы без индексов.
CSV - данные хранятся в текстовых файлах, с запятой в качестве разделителя.
BLACKHOLE - таблица не содержит данных.
FALCON –работает с многократными потоками и безопасной средой транзакции.
EXAMPLE – представляет собой заглушку.
NDBCLUSTER – этот тип таблиц предназначен для организации кластеров MySQL.

Транзакция - это последовательность операторов манипулирования данными, выполняющаяся *как единое целое.*

13.05.16

Преимущества транзакционных таблиц:

•Надежность и безопасность транзакций. Существует возможность восстановить данные при сбоях в работе MySQL или при возникновении проблем с оборудованием, .

•Возможность объединить несколько операторов и выполнять все эти операторы одной командой COMMIT.

•Возможность отмены внесенных изменений командой ROLLBACK.

•Возможность восстановления изменений при сбоях во время.

•В транзакционных таблицах лучше обеспечивается параллельное выполнение операций обновления таблицы и ее чтения.

Преимущества таблиц без поддержки транзакций:

•Быстродействие таких таблиц выше, чем у транзакционных.

•Требуется меньше дискового пространства.

•Для обновлений используется меньше памяти.

Таблицы типа MyISAM

Каждая **MyISAM** таблица хранится на диске в трёх файлах (в каталоге *mysql/data*), имена этих файлов совпадают с названием таблицы, а расширение файлов указывает на тип хранимых данных и может принимать одно из следующих значений:

frm — файл содержит информацию о структуре таблицы, об именах и типах столбцов и индексов;
MYD —файл содержит данные (MYData);
MYI —файл содержит индексы таблицы (MYIndex).

Таблицы этого типа обеспечивают очень быстрый механизм хранения данных.
Таблицы MyISAM могут быть одного из трех форматов: динамические, статические или сжатые. Первые два - выбираются автоматически.
Таблицы со строками фиксированной длины будут **статическими**, со строками переменной длины - **динамическими**.
Сжатые таблицы - могут быть созданы только при использовании программы *myisampack*

Статические таблицы

Таблица не содержит столбцов VARCHAR, BLOB или TEXT. Тип CHAR и все числовые типы имеют фиксированную длину.

Преимущества статических таблиц.

1. Самый простой и безопасный механизм хранения данных, обеспечивает наиболее быстрый поиск данных по индексу. В индексированных статических таблицах каждая запись находится на определенном расстоянии от начала файла
2. Таблицы MyISAM фиксированного размера как правило удается легко восстановить после сбоя (кроме поврежденных записей) с помощью программы *myisamchk*, так как записи расположены в фиксированных позициях, и легко восстановить все записи.
3. Все индексы могут быть восстановлены.
4. Таблицы MyISAM легко кэшируются.

Недостатком статических таблиц является нерациональное использование дискового пространства.

Динамические таблицы

Для таблиц, содержащих столбцы VARCHAR, BLOB или TEXT, если таблица была создана с параметром ROW_FORMAT=dynamic. Данный формат таблицы требует более сложного управления.

При увеличении длины строки, часть данных может остаться на старом месте, другая же часть может быть сохранена в виде нового фрагмента в другом месте файла. Это может привести к тому, что сегмент файла, помещенный операционной системой в кэш, будет содержать не всю строку.

Дефрагментацию таблицы можно произвести, воспользовавшись командой OPTIMIZE table или программой myisamchk. Статические данные в некоторых столбцах VARCHAR или BLOB одной и той же таблицы, часто считываемые или изменяемые, во избежание фрагментации лучше переместить в другие таблицы.

Свойства динамических таблиц

•Все столбцы со строками являются динамическими (кроме тех, у которых длина меньше 4).
•Перед каждой записью помещается битовый массив, показывающий, пустые столбцы ('') для строковых столбцов, или имеющие значение ноль(0) для числовых столбцов (не путать со столбцами имеющими значение NULL). Столбец отмечается в битовом массиве и не сохраняется на диск, если длина строкового столбца равна нулю после удаления пробелов в конце строки, или у значение числового столбца есть ноль. Строки, содержащие значения, сохраняются в виде байта длины и строки содержимого.
•Динамические таблицы обычно занимают намного меньше дискового пространства, чем таблицы с фиксированной длиной. При добавлении в строку информации, превышающей имеющуюся длину строки, полученная строка будет фрагментирована
•Восстановление после сбоя для таких таблиц является более сложным процессом, так как запись может быть фрагментированной и состоять из нескольких частей, а ссылка на очередной фрагмент или сам фрагмент могут отсутствовать.

Сжатые таблицы

Таблицы этого формата предназначены только для чтения. Сжатые таблицы занимают очень мало дискового пространства. Программа myisampack выполняет сжатие(по алгоритму Хаффмана), и оптимизацию, например, для хранения чисел со значением 0 отводится 1 бит, для хранения столбцов с целочисленными значениями, столбец сохраняется с использованием минимального по размерам возможного целого типа, возможно преобразование значений в перечисляемый тип (ENUM), если в столбце содержится небольшое множество возможных значений.

При сжатии таблицы, каждая запись сжимается отдельно. Для таблиц этого типа возможна обработка записей с фиксированной или динамической длиной.

Таблицы данного типа могут быть распакованы при помощи команды myisamchk.

В таблицах типа MyISAM поддерживается полнотекстовый поиск начиная с версии 3.23.23. Допускается индексирование текстовых столбцов, в том числе и переменной длины

Полнотекстовый поиск

Обычные индексы для быстрого поиска слов (строк) в больших объемах текстовой информации не подходят.

Нужно создать соответствующий индекс типа FULLTEXT. Они могут быть созданы в столбцах CHAR, VARCHAR и TEXT во время создания таблицы командой CREATE TABLE или ALTER TABLE или CREATE INDEX.

Пример

Создание таблицы с полнотекстовыми индексами FULLTEXT.

CREATE TABLE article1(ArtId INT NOT NULL AUTO_INCREMENT PRIMARY KEY, author1 VARCHAR (150), title1 VARCHAR (250), body1 TEXT, FULLTEXT (title1, body1)) ENGINE=MyISAM;

Замечание. Загрузка данных в таблицу, имеющую индекс FULLTEXT, будет более медленной, чем в таблицу не содержащую такой индекс. Для увеличения скорости загрузки больших массивов данных в таблицу, создать таблицу без индекса FULLTEXT, закгузить данные, а затем создать индекс FULLTEXT команд

Для полнотекстового поиска в таблицах типа MyISAM используется функция ***MATCH()***.

MATCH(filelds) AGAINST(words).

При поиске строка поиска ***words,*** заданная в выражении *AGAINST()*,сравнивается с содержимым текста, хранящегося, в столбцах ***filelds***, включенных в индекс FULLTEXT.
Строка поиска может состоять из одного и более слов.

Словом является любая последовательность символов, состоящая из букв, чисел, знаков.
При анализе текстов, словам присваиваются веса, в зависимости от частоты, с которой они встречаются в тексте. Если слово, присутствует во многих документах, оно будет иметь меньший вес, редко встречающееся слово будет иметь более высокий вес.

Короткие слова, состоящие (менее чем из четырех букв), по умолчанию не индексируются.
При работе с MATCH запрещено использовать в качестве имен полей зарезервированные слова MySql.

Режим поиска в естественном языке (**IN NATURAL LANGUAGE MODE**)

Режим поиска в естественном языке является основном видом поиска, используется по умолчанию.

Поиск строки st1 в полях *Fl1*, *Fl2*, включенных в индекс FULLTEXT, выборка отсортирована по релевантности.

SELECT * FROM Tbl1 WHERE MATCH (Fl1, Fl2) AGAINST (st1);

Извлечь коэффициенты релевантности в явном виде (строки не упорядочены):

SELECT Id,MATCH (Fl1,Fl2) AGAINST (st1) AS RELEVANCE FROM Tbl1;

Получить коэффициенты релевантности и отсортировать строки в порядке убывания релевантности

SELECT MATCH (Fl1,Fl2) AGAINST (st1) AS RELEVANCE FROM Tabl1 WHERE MATCH (Fl1 ,Fl2) AGAINST (st1);

В предложении WHERE нельзя использовать псевдоним.

Для режима IN NATURAL LANGUAGE MODE действует правило *«50% threshold»*, слово, встречающееся более чем в 50% всех просматриваемых полей, не учитывается при поиске.

Пример полнотекстового поиска

1. Создать таблицу с полнотекстовыми индексами FULLTEXT.

CREATE TABLE article1(ArtId INT NOT NULL AUTO_INCREMENT PRIMARY KEY, author1 VARCHAR (150), title1 VARCHAR (250), body1 TEXT, FULLTEXT (title1, body1)) ENGINE=MyISAM;

2. Внести данные в таблицу.

INSERT INTO article1 VALUES (0, ' Krause DS', 'The hematopoietic stem cell niche-home for …', 'The hematopoietic stem cell...'),
(0, ' Manesso E', 'Dynamical modelling of haematopoiesis: ...', 'A very high number of different types of blood cells....'),
(0, ' George K.', 'A two-dimensional mathematical model of non-linear ….', 'Certain drugs, for example scopolamine and timolol,...'),
(0, ' Sigvardsson M.', 'New light on the biology and developmental potential of haematopoietic stem cells and progenitor cells.', 'Even though stem cells have been identified in several tissues,....'),
(0, 'Gobaa S', 'Artificial niche microarrays ... ` To understand the regulatory role of niches in maintaining stem-cell fate...'),
(0, 'Roccio M.', 'High-throughput clonal analysis of neural stem cells ...', 'To better understand the extrinsic signals that control neural stem cell (NSC) fate, here...');

Найдем строку 'hematopoietic stem 'в полях title1 и body1 таблицы article1:

SELECT * FROM article1 WHERE MATCH (title1, body1) AGAINST ('hematopoietic stem');

ArtId	author1	title1	body1
4	Sigvardsson M.	New light on the biology and developmental potential of haematopoietic stem cells…	Even though stem cells have one of the best understood somatic stem cells is the bone marrow residing haematopoietic stem ...
1	Krause DS	The hematopoietic stem cell niche-home for friend and foe?	The hematopoietic stem cell ...

Получить коэффициенты релевантности, строки отсортируем в порядке убывания релевантности, строки с нулевой релевантностью не выводятся.

```
SELECT ArtId,MATCH (title1, body1) AGAINST ('haematopoietic stem cell') AS RELEVANCE FROM article1 WHERE MATCH (title1, body1) AGAINST ('haematopoietic stem cell');
```

В результате получим:

ArtId	RELEVANCE
4	0.81473735553517
1	0.79817111570896

Режим **BOOLEAN** (BOOLEAN MODE)

В логическом (бинарном) режиме BOOLEAN релевантность вычисляется как условная мера совпадения заданного шаблона. Положение искомого шаблона в тексте, количество встретившихся вариантов роли не играют.

Особенности режима BOOLEAN.

- Можно использовать и без создания полнотекстового индекса, но процесс поиска будет протекать очень медленно.
- Есть возможность использовать логические оператора.
- Отсутствует автоматическая сортировка(без предложения ORDER BY) в случае указания условия WHERE, но в предложении ORDER BY можно использовать псевдоним (алиас).

```
SELECT ArtId,MATCH (title1, body1) AGAINST ('+haematopoietic
stem') AS RELEVANCE FROM article1
WHERE MATCH (title1, body1) AGAINST ('+haematopoietic stem' IN
BOOLEAN MODE)  ORDER BY RELEVANCE;
```

В результат поиска включены все записи содержащие слово **haematopoietic**, если в записи присутствует слово **stem**, его релевантность будет выше, записи отсортированы по релевантности.

Операторы полнотекстового поиска в режиме BOOLEAN

Оператор	Значение
+	слово должно присутствовать в каждой строке.
-	слово не должно присутствовать ни в одной строке.
<	Оператор уменьшает вклад слова в величину релевантности (слово является менее важным).
>	Оператор увеличивает вклад слова в величину релевантности, которое приписывается строке(слово является более важным).
~	слово может присутствовать, но отрицательно влияет на оценку релевантности строки, нежелательные слова.
*	Звездочка является оператором усечения, обозначает любые символы.
( )	Скобки группируют слова в подвыражения.
"…"	Фраза, заключенная в двойные кавычки, соответствует только строкам, содержащим эту фразу, написанную буквально.

Примеры использования.

stem cell - находит строки, содержащие по меньшей мере одно из этих слов.

+ stem + cell - находит строки, содержащие оба слова.

+ stem cell - находит строки, содержащие слово **stem**, но ранг строки выше, если она также содержит слово **cell** .

+ stem - cell - находит строки, содержащие слово **stem**, но не **cell**.

+ stem +(> cell < somatic) - находит строки, содержащие **stem** и **cell**, или **stem** и **somatic** (в любом порядке), но ранг **stem cell** выше, чем **stem somatic**.

ste* - находит строки, содержащие **ste, stem**, **stems**, и т.д.

"stem cell" - находит строки, содержащие фразу указанную в кавычках. Например, строки типа «*..* **stem cell***..*». , найдутся но не найдутся **stem… cell**.

В бинарном режиме отсутствует ограничение *«50% threshold»*.

Режим поиска в естественном языке с параметром QUERY EXPANSION (**NATURAL LANGUAGE MODE WITH QUERY EXPANSION**)

В результат поиска будет включены все записи, содержащие не только совпадения с шаблоном, но и возможные логические совпадения.

Сначала выполняется запрос аналогичный NATURAL LANGUAGE MODE и формируется выборка.

По полученной выборке производится попытка вычислить слова, имеющие высокую релевантность для этой выборки. В случае, если такие слова присутствуют, поиск производится по ним, но их влияние на релевантность будет существенно ниже.

В итоге создается смешанная выборка – сначала результаты, где присутствует искомое слово, а потом те, которые были получены в результате «повторного» поиска.

Данный режим не рекомендуется использовать для больших объемов информации.

Таблицы типа MERGE

Тип таблиц MERGE позволяет объединять несколько таблиц типа MyISAM в одну.

Все таблицы объединяемые в одну таблицу MERGE, должны иметь идентичную структуру, т.е. одинаковые столбцы, индексы и порядок их следования. При создании таблицы типа MERGE информация о ней будет записана в двух файлах:

- файл с расширением *.rfm*, содержащий структуры таблицы
- файл с расширением *.MRG*. содержащий список индексных файлов, работа с которыми должна вестись как с единым файлом.

К объединенной таблице можно применять команды SELECT, DELETE и UPDATE. При попытке удалить таблицу MERGE командой DROP TABLE, будет уничтожена только таблица MERGE, исходные таблицы MyISAM не будут затронуты.

Пример. Создание таблицы MERGE:

```
CREATE TABLE Tbl1 (EmpId1 INT NOT NULL AUTO_INCREMENT
PRIMARY KEY,
Emp_Nm CHAR(20));
CREATE TABLE Tbl2 (EmpId1 INT NOT NULL AUTO_INCREMENT
PRIMARY KEY,
Emp_Nm CHAR(20));
INSERT INTO Tbl1 (Emp_Nm) VALUES ('1a'), ('1b'), ('1c');
INSERT INTO Tbl1 (Emp_Nm) VALUES ('2a'), ('2b'), ('2c');
CREATE TABLE Tbl_Sum (EmpId1 INT NOT NULL AUTO_INCREMENT
PRIMARY KEY,Emp_Nm CHAR(20), index(EmpId1)) ENGINE=MERGE
UNION=(Tbl1, Tbl2) INSERT_METHOD=LAST;
```

Результат выборки из таблицы Tbl_Sum: `SELECT * FROM Tbl_Sum`

EmpId 1	Emp_Nm
1	1a
2	1b
3	1c
1	2a
2	2b
3	2c

В операторе UNION задаются таблицы для объединения, значение параметра INSERT_METHOD определяет порядок добавления записей в MyISAM таблицы.

Значения параметра INSERT_METHOD:

FIRST — при вставке новой записи в таблицу MERGE, запись размещается в первой таблице из списка в параметре UNION.

LAST — при вставке новой записи в таблицу MERGE, запись размещается в последней таблице списка.

NO — вставка в MERGE таблицу запрещена, использования оператора INSERT приведёт к ошибке. Выражение INSERT_METHOD=NO аналогично отсутствию параметра.

Таблицы типа MEMORY (HEAP)

Таблицы **MEMORY (HEAP)** являются исключительно быстрыми, целиком хранятся в оперативной памяти, но в случае сбоя работы сервера — полная потеря данных.

Структура таблицы типа **MEMORY** определена в файле с расширением frm. При остановке или перезагрузке сервера вся информация содержащаяся в этой таблице теряется, данные о структуре таблицы остаются.

Ограничения таблицы типа **MEMORY** :

- Индексы используются для поиска строк только в запросах, содержащих операторы сравнения "=" и "<=>".
- Возможно использование только неуникальных индексов.
- Не поддерживают типы стобцов TEXT и BLOB.
- До версии MySQL 4.1 не поддерживался AUTO_INCREMENT.

```
CREATE TABLE tbl1 (Id1 INT NOT NULL PRIMARY KEY, Nm CHAR(80))
ENGINE=MEMORY Max_Rows=100;
```

Max_Rows задает максимальное количество строк в таблице.

Таблицы типа BDB (BerkeleyDB)

Berkeley Database – это самостоятельная встроенная система баз данных с открытым кодом. Berkeley Database поддерживает транзакции, блокировки, резервное копирование и репликацию (синхронизацию нескольких копий таблиц базы данных).

Berkeley Database не является сервером баз данных, SQL-ядром, выполняющим запросы или реляционной базой данных. Данные в ней хранятся в виде байтовых массивов, пары (ключ — значение). Программный продукт распространяется в виде библиотеки, подключаемой к коду приложения.
Первая версия Berkeley DB была разработана в Университете Беркли в 1986 году, впоследствии была создана компания Sleepycat Software , в 2006 году купленная корпорацией Oracle.

Поддержка таблиц BDB была включена в дистрибутив исходного кода MySQL, начиная с версии 3.23.34, и была прекращена в версии 5.1 .

Каждая таблица типа BDB хранится в двух файлах, имена файлов совпадают с названием таблицы, расширение файлов указывает на тип хранимых данных:

- *frm* — файл содержит информацию о структуре таблицы
- *db* — файл содержит информацию о индексах.

Таблицы типа BDB хранятся в виде бинарных деревьев (B-деревья или B-tree).
Метод хранения не является оптимальным (во всех остальных типах таблиц в виде бинарных деревьев хранятся индексы).

Пары «ключ-значение» хранятся в листовых страницах, пары «ключ, адрес дочерней страницы» – на внутренних узлах.
Ключи в дереве хранятся в отсортированном порядке, заданном функцией сравнения, указанной при создании базы данных.
Для упрощения задачи обхода дерева страницы на листовом уровне содержат указатели на соседей.
Страница — это совокупность последовательно расположенных записей таблицы. В BDB таблицы блокируются на уровне страниц.

Для таблиц BDB ведется двоичный журнал, где регистрируются все запросы изменяющие данные.

Все незавершенные транзакции обработчик таблицы BDB хранит в файлах журналов, их наличие требуется при запуске MYSQL.

При создании нового файла журнала BDB, MySQL устанавливает контрольные точки и удаляет все старые файлы журналов. Команда FLUSH LOGS, позволяет установить контрольную точку для таблиц Berkeley DB вручную.

Основные особенности таблиц BDB.

1. Поддерживаются транзакции на уровне страниц. Транзакции реализуются посредством журнальных файлов.
2. Блокировка осуществляется на уровне страниц. При выполнении операций манипулирования данными необходимы страничные блокировки, что может приводить к возникновению тупиков, т.е. взаимоблокировок.
3. Ведение файлов журналов необходимо для обеспечения возможности отката транзакций, при отмене транзакции, из журнальных файлов считываются все запросы, выполненные во время последней транзакции (до контрольной точки), и производятся "обратные" действия

4. Все таблицы BDB должны иметь первичный ключ, при его отсутствии создаётся скрытый первичный ключ с атрибутом AUTO_INCREMENT.
5. Подсчёт количества строк в таблице при помощи функции COUNT() происходит медленнее, из-за того что подсчёт строк для таблиц BDB (в отличие от MyISAM) на стороне сервера не поддерживается и полный пересчёт происходит при каждом обращении.
6. Ключи не являются упакованными, как в таблицах MyISAM и занимают больше места.
7. В случае, когда таблица BDB занимает всё свободное место на диске, происходит откат транзакции и вывод сообщения об ошибке. В отличие от BDB, таблицы MyISAM просто будут ждать появления свободного места, что приведёт к зависанию сервера.
8. Таблицы BDB хранятся в файле `*.db', в том же каталоге, где был создан, вследствие этого таблицы BDB нельзя перемещать между каталогами простым копированием, поскольку при их создании путь к файлу таблицы сохраняется. Для переноса базы

9. Если все столбцы, к которым производится обращение в таблице BDB, являются частью одного индекса или одного первичного ключа, то MySQL может выполнить запрос, не обращаясь к самой строке (для таблиц MyISAM это возможно только, если столбцы являются частью одного индекса).
10. Первичный ключ обеспечивает более быструю обработку, чем любой другой ключ, так как он хранится вместе с данными строки. Поскольку остальные ключи хранятся как данные ключа + PRIMARY KEY, важно иметь как можно более короткие первичные ключи, чтобы сэкономить дисковое пространство и увеличить производительность.

Пример .
CREATE TABLE tab1 ((Ind1 INT NOT NULL PRIMARY KEY, Name1 CHAR(80) engine=BDB;

Начиная с версии MySQL 5.1 таблицы, этого типа не поддерживаются, при попытке создать таблицу с механизмом BDB, можно получить сообщение об ошибке типа:
«ERROR The 'BerkeleyDB' feature is disabled; you need MySQL built with 'BerkeleyDB' to have it working»

Таблицы типа InnoDB

InnoDB является системой управления базой данных в рамках MySQL.

В InnoDB есть свой собственный буферный пул для кэширования данных и индексов в основной памяти.

Таблицы и индексы InnoDB хранятся в специальном пространстве памяти, которое может состоять из нескольких файлов, которые могут находится на нескольких дисках или хостах (каждая таблица MyISAM хранится в отдельном файле).

Таблицы InnoDB могут быть практически любого размера .

Основные отличие механизма InnoDB от MyISAM - это поддержка транзакций и внешних ключей.

В таблицах InnoDB можно задавать ограничения внешнего ключа, для обеспечения целостности данных.

Механизм InnoDB разработан Innobase (финляндия), была поглощена Sun Microsystems, затем компанией Oracle Corporation.

Поддержка InnoDB появилась в MySQL 3.23, начиная с версии 5.5 механизм InnoDB стал основным хранилищем по умолчанию.

Для создания резервных копий открытых баз данных InnoDB можно использовать интерактивный инструмент - InnoDB Hot Backup

Достоинства механизма хранения данных **InnoDB**

1. Все таблицы хранятся в едином табличном пространстве, что позволяет снять ограничение на объём таблиц.
3. Таблицы поддерживают автоматическое восстановление после сбоя.
4. Поддерживаются транзакции.
5. Поддерживаются внешние ключи, это единственный тип таблиц в MySQL поддерживающий каскадное удаление.
6. Блокировка выполняется на уровне отдельных строк таблиц.
7. Согласованное неблокирующее чтение в операторах SELECT.

Недостатки механизма хранения данных InnoDB:

1. Нельзя установить ключ для столбцов типа BLOB или TEXT.
2. Таблица не может содержать больше 1000 столбцов.
3. Большинство сценариев таблицы MyISAM будут работать быстрее (плата за безопасность).
4. Для хранения одного итого же объема полезной информации таблицы InnoDB будут использовать больше дискового пространства, чем таблицы MyISAM (плата за снятие ограничения размера таблицы).

Таблицы FEDERATED

FEDERATED - таблицы предназначены для доступа к данным удаленного MySQL сервера как к локальным таблицам.

При создании таблицы создаётся только файл структуры с расширением frm, поскольку данные хранятся на удалённой машине.
Для создания таблицы FEDERATED необходимо сначала создать таблицу на удалённой машине. На удаленном сервере можно использовать любой тип таблиц.
Затем на рабочей машине с указанием на удалённую, структура таблиц на машинах должна быть одинакова. **FEDERATED**-таблицы на локальной машине. Пример

```
CREATE TABLE TST1(Id int, N int) ENGINE=MyISAM;
CREATE TABLE TST2(Id int, N int) ENGINE=FEDERATED CONNECTION =
'mysql://user_name@remote_host:3306/host_name/TST1'
```

В поле CONNECTION таблицы FEDERATED указывается строка подключения к таблице на удалённой машине.

Таблицы работают только с операторами SELECT, INSERT, UPDATE, DELETE, не работают с ALTER TABLE, DROP TABLE, не поддерживают транзакции. Синтаксис запроса проверяется на локальной машине после чего он отправляется на удаленную машину для выполнения, результаты отправляются обратно на локальную машину.

Удаленным сервером может быть только **MySQL**-сервер.

При создание **FEDERATED**-таблицы, указывающей на другую **FEDERATED**-таблицу возможно образование циклических ссылок.

MySQL в бинарных кодах распространяется без поддержки таблиц данного типа. Необходима перекомпиляция **MySQL** с включенной опцией **—DWITH_FEDERATED_STORAGE_ENGINE**.

По умолчанию данный тип таблиц не задействован, для включения сервер запускают с параметром **—federated.**

Таблицы ARCHIVE

Эти таблицы хранятся без индексов, занимают мало места, применяются для хранения редко используемых исторических, архивированных данных.

Таблицы CSV

Данные хранятся в текстовых файлах, с запятой в качестве разделителя. Обычно используются для обмена данными с другим программным обеспечением и прикладными программами, которые могут импортировать и экспортировать в формате CSV (например Excel).

Таблицы BLACKHOLE

Такие таблицы не содержат данных. При создании таблицы на сервере в каталоге баз данных создается только один файл, имя которого начинается с имени таблицы, имеет расширение .frm. Операции вставки данных в таблицу не сохраняют никакие данные, если ведется файл двоичный регистрации, в нем записываются команды SQL, поиск всегда возвращают пустой набор, может использоваться в распределенном проекте базы данных, где данные автоматически скопируются, но не будут сохранены локально.

Таблицы FALCON

Данный механизм работает с многократными потоками и безопасной средой транзакции, который безопасно хранит данные при обеспечении весьма высокой производительности. В настоящее время Falcon обеспечивается только внутри ветки MySQL 5.1.

Falcon был разработан для систем, которые способны поддерживать большую память и многопоточные или мультиядерные среды CPU (64-битные системы представляют собой идеальные платформы для Falcon, где имеется большое доступное пространство памяти и 2, 4 или 8-ядерные CPU).

.
Главные особенности:
•Гибкая блокировка, включая гибкие уровни блокировки и интеллектуальное обнаружение тупика. Возможность модифицировать записи и таблицы без блокировок уровня строки.
•Многопоточность.
•Transaction-safe (полностью совместим с ACID), возможность обработки многократные параллельные транзакции.
•Последовательный файл регистрации обеспечивает высокую эффективность и возможности восстановления.
•Продвинутые индексы B-Tree.
•Ссылочная целостность и гарантируемая проверка правильности данных.
•Сжатие данных, информация на диске хранится в сжатом формате (сжатие и декомпрессия в процессе работы.
•Интеллектуальное дисковое управление размером файла на диске.
•Индексное кэширование обеспечивают быстрый доступ к данным без требования загрузить индексные данные с диска.
•Неявные точки сохранения гарантируют целостность данных в течение транзакций.

Таблицы FALCON

Данный механизм работает с многократными потоками и безопасной средой транзакции, который безопасно хранит данные при обеспечении весьма высокой производительности. В настоящее время Falcon обеспечивается только внутри ветки MySQL 5.1.

Таблицы EXAMPLE

Таблицы этого типа представляет собой заглушку, таблицу этого типа создать можно, но невозможно записать данные в таблицу или извлечь их из нее, полезен как пример разработчикам.

Таблицы NDBCLUSTER

Таблиц предназначены для организации кластеров MySQL, в случае когда таблицы распределены между несколькими компьютерами, объединенными в сеть.