

STORAGE ENGINES



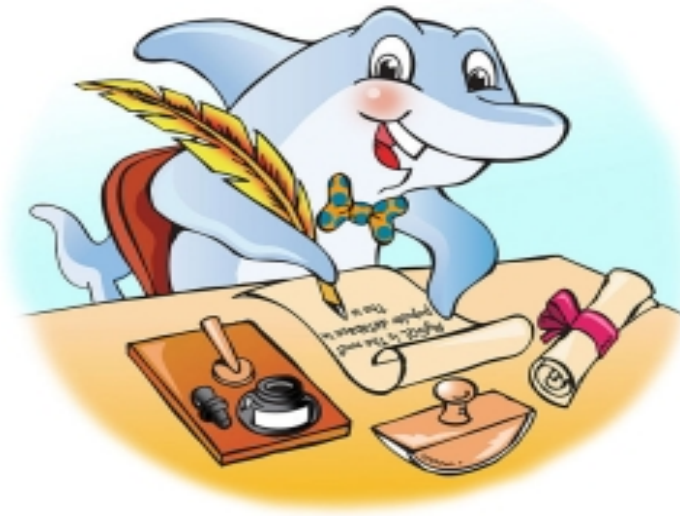
Wagner Bianchi

Certified MySQL 5.0 Developer

Certified MySQL 5.0 Database Administrator

Certified MySQL 5.1 Cluster Database Administrator





Artigos recomendados:

- [http://imasters.uol.com.br/artigo/8065/bancodedados/mysql_innodb - introducao e principais caracteristicas/](http://imasters.uol.com.br/artigo/8065/bancodedados/mysql_innodb_-_introducao_e_principais_caracteristicas/)
- http://imasters.uol.com.br/artigo/7913/mysql/maria_o_novo_storage_engine_do_mysql/

Introdução

- ✎ O MySQL permite a você escolher um entre vários Storage Engines quando inicia a criação de tabelas em seu banco de dados;
- ✎ Os Storage Engines, chamados antes de Table Types (deprecated), também é conhecido como Motor de Armazenamento ou ainda Engenharia de Armazenamento, ao pé da letra;
- ✎ Cada Storage Engine tem suas respectivas características, desde um Engine bastante veloz e enxuto, no caso do MyISAM até um Engine mais robusto que é o INNODB;
- ✎ Neste módulo, trabalharemos com os Storage Engines abordados pela certificação do MySQL, na versão 5.0. Existem vários outros Engines que poderão ser encontrados no Apêndice A;

Introdução

- ✧ O MySQL, por via de sua arquitetura, é tipo como um *Plugable Database*, pois, vários ativistas da comunidade do MySQL desenvolvem seus próprios Engines com características de outros SGBD's, plugam no MySQL e trabalham com aquelas características;

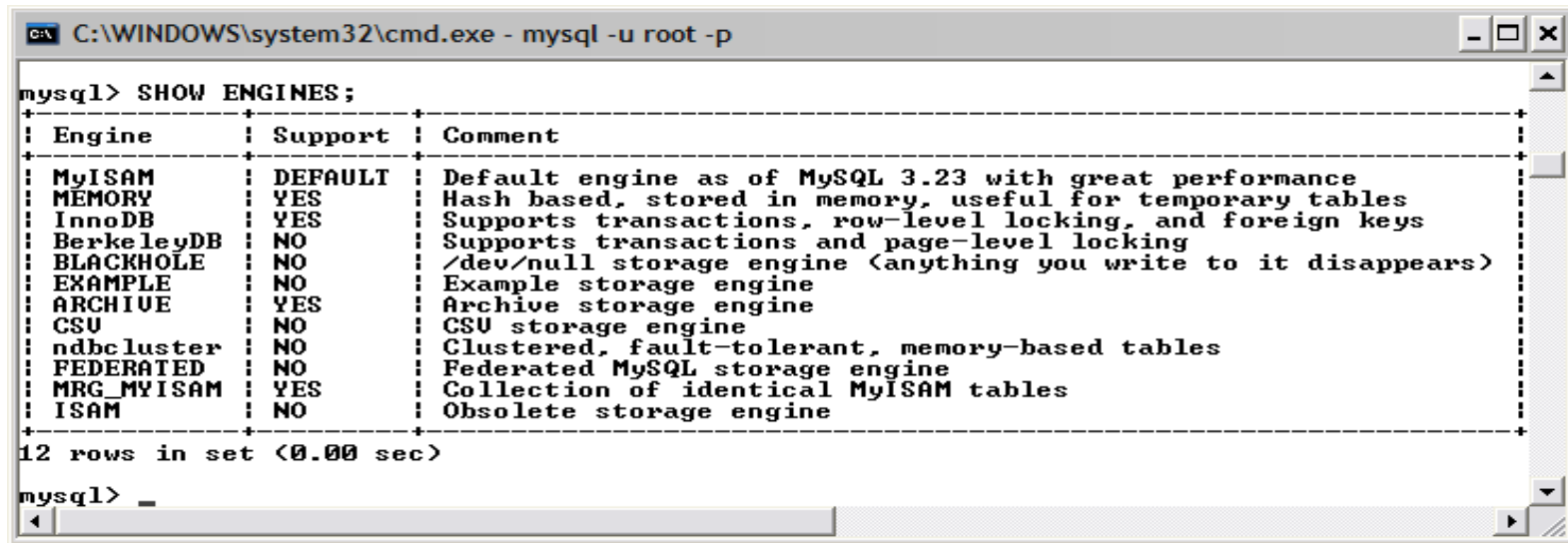


- ✧ Nem todos os Storage Engines listados acima fazem parte de uma instalação default do MySQL. A maioria daqueles que vem desabilitados por padrão tem certa facilidade para sua habilitação;
- ✧ Cada Engine tem suas próprias características que dão muita flexibilidade na utilização do MySQL;

Introdução

Podemos checar quais são os Storage Engines habilitados em nossa instalação de MySQL atual, através do comando:

SHOW ENGINES;



```
C:\WINDOWS\system32\cmd.exe - mysql -u root -p

mysql> SHOW ENGINES;
+-----+-----+-----+
| Engine      | Support | Comment                                     |
+-----+-----+-----+
| MyISAM      | DEFAULT | Default engine as of MySQL 3.23 with great performance |
| MEMORY      | YES     | Hash based, stored in memory, useful for temporary tables |
| InnoDB      | YES     | Supports transactions, row-level locking, and foreign keys |
| BerkeleyDB  | NO      | Supports transactions and page-level locking |
| BLACKHOLE    | NO      | /dev/null storage engine (anything you write to it disappears) |
| EXAMPLE     | NO      | Example storage engine |
| ARCHIVE     | YES     | Archive storage engine |
| CSU         | NO      | CSU storage engine |
| ndbcluster  | NO      | Clustered, fault-tolerant, memory-based tables |
| FEDERATED    | NO      | Federated MySQL storage engine |
| MRG_MYISAM  | YES     | Collection of identical MyISAM tables |
| ISAM        | NO      | Obsolete storage engine |
+-----+-----+-----+

12 rows in set (0.00 sec)

mysql> _
```

Introdução

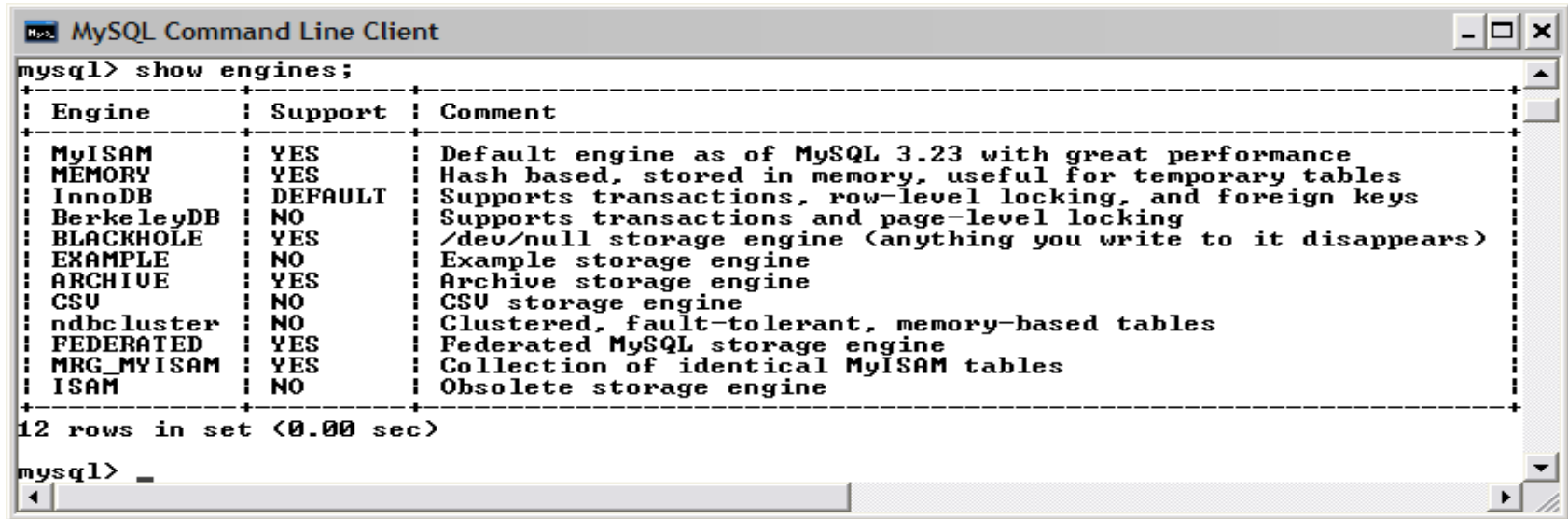
- Podemos perceber que há muitas opções de Storage Engines para se utilizar, cada com a sua característica diferente, sendo que, são eles transacionais ou não-transacionais;
- Atentando-se à figura anterior, percebemos que o Storage Engine padrão do MySQL é o MyISAM, mas isso poderá ser modificado facilmente através da linha de comando – alterando o Engine padrão somente para sua sessão - ou através do arquivo de opções my.ini ou my.cnf, adicionando a seguinte linha ao agrupamento [mysqld]:

```
[mysqld]  
default-storage-engine=INNODB
```

- A sintaxe para se adicionar uma opção não existente ao arquivo de configuração mudou da versão **4.x para a versão 5.x**, Anteriormente era setado com `set-variable=variável=valor`.

Introdução

- Atualmente, nosso servidor de bancos de dados MySQL, tem como Storage Engine padrão o MyISAM. Após adicionarmos a linha no arquivo de opções, o my.ini ou my.cnf e salvá-lo. Reiniciamos o servidor de bancos de dados MySQL e veja o que temos:



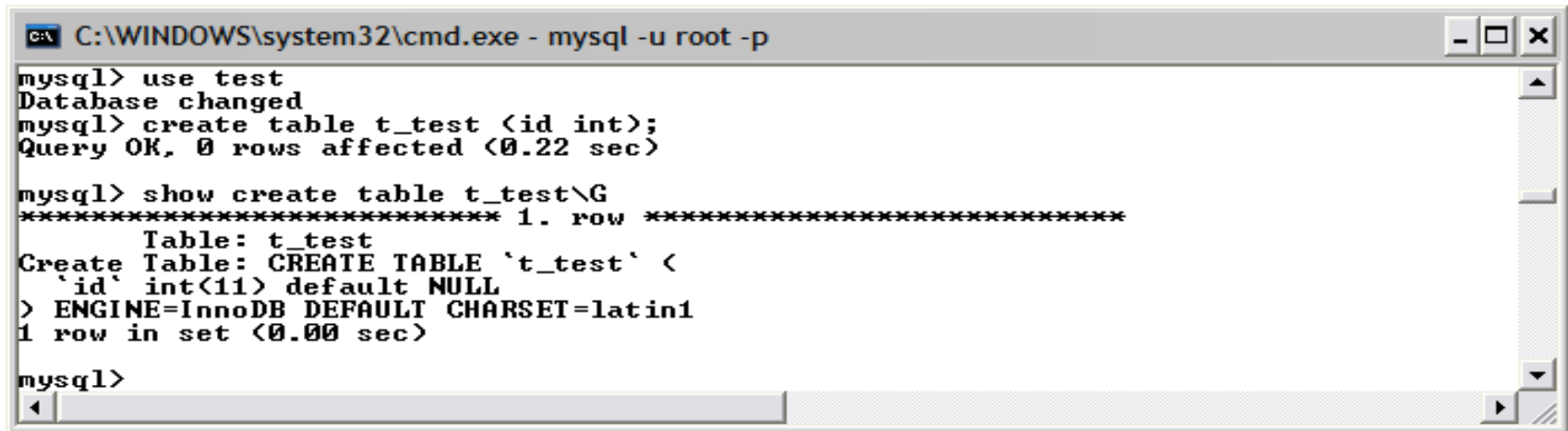
```
mysql> show engines;
```

Engine	Support	Comment
MyISAM	YES	Default engine as of MySQL 3.23 with great performance
MEMORY	YES	Hash based, stored in memory, useful for temporary tables
InnoDB	DEFAULT	Supports transactions, row-level locking, and foreign keys
BerkeleyDB	NO	Supports transactions and page-level locking
BLACKHOLE	YES	/dev/null storage engine (anything you write to it disappears)
EXAMPLE	NO	Example storage engine
ARCHIVE	YES	Archive storage engine
CSU	NO	CSU storage engine
ndbcluster	NO	Clustered, fault-tolerant, memory-based tables
FEDERATED	YES	Federated MySQL storage engine
MRG_MYISAM	YES	Collection of identical MyISAM tables
ISAM	NO	Obsolete storage engine

```
12 rows in set (0.00 sec)  
  
mysql> _
```

Introdução

- Quando criamos uma tabela em um banco de dados do servidor de bancos de dados MySQL sem mencionar a declaração **Engine** em meio ao **CREATE TABLE**, o SGBD elegerá esta tabela como parte como uma tabela controlado pelo Storage Engine padrão atualmente no servidor:



```
C:\WINDOWS\system32\cmd.exe - mysql -u root -p

mysql> use test
Database changed
mysql> create table t_test (id int);
Query OK, 0 rows affected (0.22 sec)

mysql> show create table t_test\G
***** 1. row *****
      Table: t_test
Create Table: CREATE TABLE `t_test` (
  `id` int(11) default NULL
) ENGINE=InnoDB DEFAULT CHARSET=latin1
1 row in set (0.00 sec)

mysql>
```


MyISAM

- ↪ **MyISAM** é o Storage Engine padrão do MySQL, baseado no código do antigo Engine chamado ISAM - já obsoleto na versão 5.0 - só que com muito mais funcionalidades (chamadas de extensões);
- ↪ Quando criamos uma tabela MyISAM, são adicionados três tipos de arquivos dentro do diretório do banco de dados contido dentro do diretório chamado de *DATADIR* ou diretório de dados:
 - Um arquivo “.MYD”: é o arquivo de dados da tabela;
 - Um arquivo “.MYI”: é o arquivo de índices da tabela;
 - Um arquivo “.frm”: é o arquivo que contém o “**CREATE TABLE**”;
- ↪ Normalmente, estes três arquivos são armazenados em um mesmo local, dentro do diretório do banco de dados, sob o diretório de dados do MySQL. Em sistemas que suportam links simbólicos (symlinks), podemos armazenar tais arquivos em outros locais;

MyISAM

- ✎ MyISAM tem o controle mais flexível de colunas auto_increment em relação a outros Storage Engines;
- ✎ Muito rápidas para leitura e bastante lentas para ambientes com muitas escritas;
- ✎ Não tem suporte a transações e nem a integridade referencial;
- ✎ Tem bloqueio em nível de tabela, ou seja, cada comando, seja ele **INSERT**, **DELETE** ou **UPDATE** irão adquirir bloqueio de toda a tabela;
- ✎ **Deadlocks não ocorrem!**

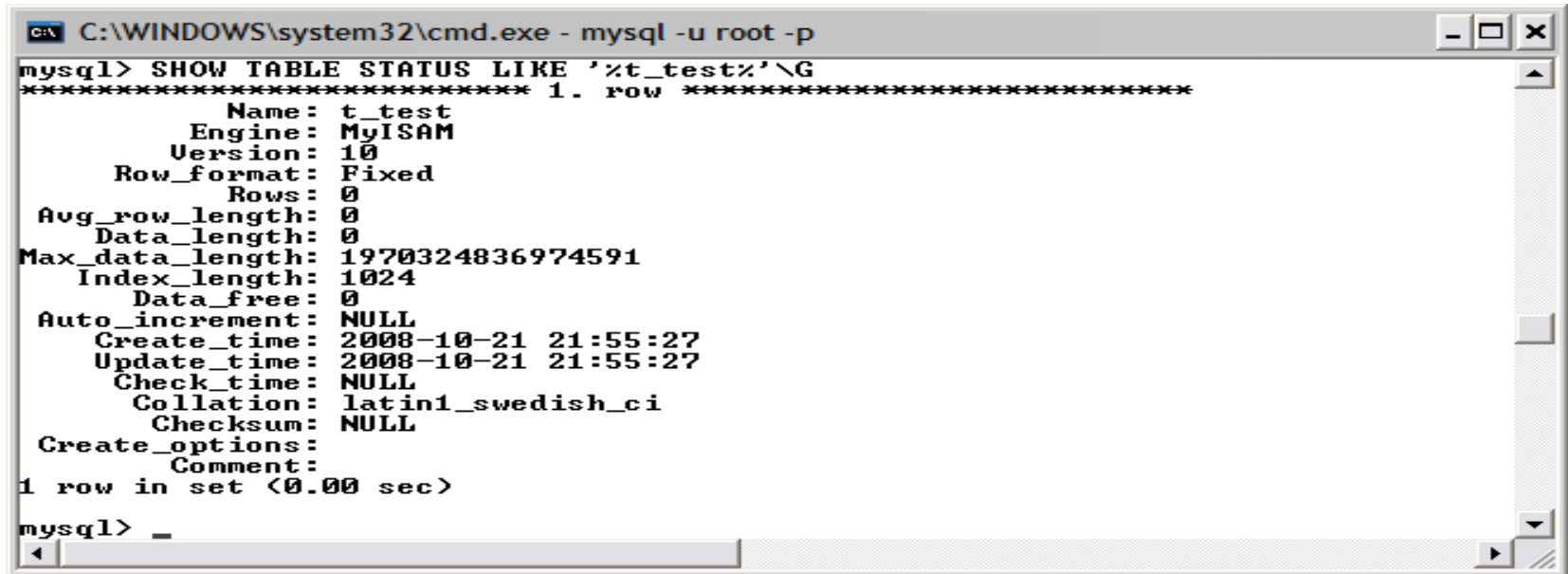
MyISAM

- ↳ Suportam **FULL-TEXT** searching e Tipos de Dados Espaciais para aplicações de Geoprocessamento;
- ↳ Tabelas MyISAM tem seu formatos de linha, podendo ser:
 - **Fixed-Row Format:** todas as linhas deste tipo de tabela MyISAM tem o mesmo tamanho (fácil de recuperar mas necessitam de mais espaço);
 - **Dynamic-Row Format:** o tamanho das linhas varia e não são eficientemente recuperadas. Fragmentações ocorrem com maior facilidade e consomem menos espaço;
 - **Compressed Format:** tais tabelas são comprimidas para liberar espaços, otimizadas para leitura e são read-only.

MyISAM

Podemos checar qual é o formato de linha da tabela através do comando:

SHOW TABLE STATUS LIKE '%t_test%';



```
C:\WINDOWS\system32\cmd.exe - mysql -u root -p
mysql> SHOW TABLE STATUS LIKE '%t_test%'\G
***** 1. row *****
      Name: t_test
      Engine: MyISAM
      Version: 10
      Row_format: Fixed
      Rows: 0
      Avg_row_length: 0
      Data_length: 0
      Max_data_length: 1970324836974591
      Index_length: 1024
      Data_free: 0
      Auto_increment: NULL
      Create_time: 2008-10-21 21:55:27
      Update_time: 2008-10-21 21:55:27
      Check_time: NULL
      Collation: latin1_swedish_ci
      Checksum: NULL
      Create_options:
      Comment:
1 row in set (0.00 sec)

mysql> _
```

MyISAM

- Uma tabela tem formato de linhas fixas quando utiliza tipos de dados para armazenar caracteres diferentes do tipo **VARCHAR**. Se uma tabela contar com um campo com o tipo de dados **VARCHAR**, já passará a ser uma tabela com formato de linha dinâmico;

```
C:\WINDOWS\system32\cmd.exe - mysql -u root -p
mysql> SHOW TABLE STATUS LIKE '%t_test%'G
***** 1. row *****
      Name: t_test
      Engine: MyISAM
      Version: 10
      Row_format: Dynamic
      Rows: 0
      Avg_row_length: 0
      Data_length: 0
      Max_data_length: 281474976710655
      Index_length: 1024
      Data_free: 0
      Auto_increment: NULL
      Create_time: 2008-10-21 21:58:00
      Update_time: 2008-10-21 21:58:00
      Check_time: NULL
      Collation: latin1_swedish_ci
      Checksum: NULL
      Create_options:
      Comment:
1 row in set (0.02 sec)

mysql> _
```

MyISAM

Podemos, na criação de uma tabela MyISAM, dizer o total de linhas que queremos ter na tabela. Caso falte espaço em disco ao adicionar linhas à uma tabela deste tipo, nenhum erro será enviado, as operações serão suspensas até que mais espaço seja liberado;

Para criar uma tabela MyISAM:

```
CREATE TABLE t1 (id int not null) Engine=MyISAM;
```

O MyISAM possui várias opções que podemos adicionar ao arquivo de opções do MySQL, o my.ini ou my.cnf para otimizar sua performance;

Muitos deles requerem determinado equilíbrio em relação à memória disponível no servidor – host – para ser utilizada pelo SGBD;

MyISAM

- MyISAM apresenta graves problemas de crash em tabelas. Sistemas que trabalham em locais com problemas de queda de energia podem ter problemas com tabelas MyISAM, que são mais sensíveis que todas as outras por ter estrutura mais enxuta;
- No **Linux**, podemos facilmente simular o problema através do seguinte comando: `$ kill -9 `pgrep mysqld``
- Quando o banco retorna às operações, o mecanismo de **crash-recovery** do MyISAM não consegue se recuperar;
- Caso você tenha um ambiente com poucas escritas e mais leituras que necessite de boa performance na resposta, considere utilizar o mais novo Engine chamado **MARIA**. O teste do **MARIA** pode ser visto em:

Archive

- 🐛 O **Archive** é um Storage Engine que provê eficiente forma para armazenar grandes quantidades de dados quando não é necessário se ter índices e se precisa liberar espaço em disco;
- 🐛 Esse Engine só suporta declarações **SELECT** e **INSERT**;
- 🐛 Toda tabela Archive é, após ser criada, cria no diretório do banco de dados a que pertence, sob o diretório de dados, um arquivo “.frm”, que é o arquivo que contém o **CREATE TABLE**, os seguintes arquivos:
 - Um arquivo “.ARZ”: esse é o arquivo que contém os dados – datafile;
 - Um arquivo “.ARM”: esse arquivo armazena metadados da tabela;
 - Um arquivo “.frm”: que contém o **CREATE TABLE**;
- 🐛 O Archive vem habilitado a partir da versão MySQL 5.0 max beta;

Archive

- ✎ É uma ótima opção para se formar os históricos ou mesmo armazenar parte dos dados de uma outra grande tabela para aliviar otimizar consultas;
- ✎ Economizar espaços, mas com algumas limitações e com muita segurança, essa é a principal característica de uma tabela que é controlada pelo Storage Engine Archive;
- ✎ Para fazer um teste e já demonstrar como criar tabelas controladas pelo Storage Engine Archive, vamos criar 3 tabelas com o mesmo número de linhas, sendo que uma primeira será uma tabela controlada pelo Storage Engine MyISAM, a segunda utilizaremos o tipo INNODB e ao final, criaremos uma tabela controlada pelo Engine Archive;
- ✎ Ao final, veja o quanto os dados foram comprimidos;

Archive

Tabela 1:

```
mysql> create table test_myisam engine=myisam as  
      > select * from mysql.user;  
Query OK, 112050 rows affected (0.00 sec)  
Records: 112050 Duplicates: 0 Warnings: 0
```

Tabela 2

```
mysql> create table test_innodb engine=innodb as  
      > select * from mysql.user;  
Query OK, 112050 rows affected (0.00 sec)  
Records: 112050 Duplicates: 0 Warnings: 0
```

Tabela 3

```
mysql> create table test_archive engine=archive as  
      > select * from mysql.user;  
Query OK, 112050 rows affected (0.00 sec)  
Records: 112050 Duplicates: 0 Warnings: 0
```

Archive

```
mysql> SELECT table_name table_name,  
-> engine, ROUND(data_length/1024/1024,2) total_size_mb,  
-> table_rows  
-> FROM information_schema.tables  
-> WHERE table_schema = 'gim' and table_name like 'test%'  
-> ORDER BY table_rows ASC;
```

table_name	engine	total_size_mb	table_rows
test_archive	ARCHIVE	1.64	112050
test_myisam	MyISAM	6.46	112050
test_innodb	InnoDB	9.52	112050

Federated

- ✎ O Storage Engine Federated está disponível no MySQL desde a versão 5.0.3;
- ✎ A principal função deste Storage Engine é acessar dados em tabelas de outros servidores MySQL remotos, sem a utilização de replicação ou mesmo qualquer tecnologia de cluster;
- ✎ Quando utilizamos uma tabela controlada pelo Storage Engine Federated em um servidor de bancos de dados MySQL local, as consultas a este banco buscam dados do servidor remoto. Tabelas locais não armazenam dados;
- ✎ Caso você resolva instalar o MySQL a partir do *source* (código fonte), inclua a opção `--with-federated-storage-engine`;

Federated

- Quando criamos uma tabela controlado pelo Storage Engine Federated, é criado um único arquivo, de extensão “.frm” no diretório do banco de dados, sob o diretório de dados do MySQL (*DATADIR*);
- Ao selecionarmos dados de uma tabela do tipo Federated, uma conexão com um servidor remoto é feita e então os dados remotos são retornados. A conexão é realizada utilizando a **MySQL client API**;
- Primeiro, decidimos qual é a tabela do servidor remoto que queremos criar uma tabela “atalho” no servidor local. Essa tabela remota pode ser do tipo MyISAM, INNODB ou controlada por qualquer outro Storage Engine;
- Após definir a tabela remota e os dados, a tabela controlada pelo Storage Engine Federated é criada no servidor local;

Federated

Tabela remota:

```
CREATE TABLE test_table (  
    id INT(20) NOT NULL AUTO_INCREMENT,  
    name VARCHAR(32) NOT NULL DEFAULT '',  
    other INT(20) NOT NULL DEFAULT '0',  
    PRIMARY KEY (id), INDEX name (name),  
    INDEX other_key (other) )  
ENGINE=MyISAM  
DEFAULT CHARSET=latin1;
```

Federated

Tabela local:

```
CREATE TABLE federated_table (  
    id INT(20) NOT NULL AUTO_INCREMENT,  
    name VARCHAR(32) NOT NULL DEFAULT '',  
    other INT(20) NOT NULL DEFAULT '0',  
    PRIMARY KEY (id), INDEX name (name),  
    INDEX other_key (other) )  
ENGINE=FEDERATED DEFAULT CHARSET=latin1  
CONNECTION='mysql://fed_user@remote_host:9306/  
federated/test_table';
```

Federated

Antes da versão 5.0.13, a declaração **COMMENT** era utilizada no lugar de **CONNECTION** ;

A formula geral da string de conexão abordada na declaração **CONNECTION** ou **COMMENT** ao final da tabela é a seguinte:

scheme://user_name[:password]@host_name[:port_num]/db_name/tbl_name

Esse Storage Engine plenitude no caminho do que é chamado de alta-disponibilidade em bancos de dados, pois, com ele, você poderá facilmente obter dados remotos em conjunção com os dados locais;

Uma tabela do tipo Federated não precisa conter exatamente as mesmas colunas da tabela remota;

Federated

- ↳ Tabelas controladas pelo Storage Engine Federated não suportam transações;
- ↳ Suportam **SELECT**, **INSERT**, **DELETE** e **UPDATE**;
- ↳ Não são utilizados nenhum tipo de bloqueio em tabelas controladas por este Storage Engine;
- ↳ Não suportam a criação de índices;
- ↳ Fórum sobre o Storage Engine:
 - <http://forums.mysql.com/list.php?105>

Memory

- Memory Storage Engine controla tabelas previamente criadas, mantendo seus dados em memória. Antes tinham o nome de HEAP;
- Toda tabela controlada pelo Storage Engine MySQL está associada a um arquivo de extensão “.frm”, que é criado no diretório do banco de dados a que ela pertence, sob o diretório de dados do MySQL;
- Para criar uma tabela controlada por este Storage Engine, basta utilizar a declaração Engine, como segue:

```
CREATE TABLE test_table(id int, name CHAR(60)) Engine=MEMORY;
```

- Tem ótima performance mas o conteúdo das tabelas Memory não são mantidos quando o servidor é reiniciado;

Memory

- Após reiniciar o servidor, somente a estrutura da tabela permanecerá no banco de dados;
- Além de índices **BTREE INDEX**, tabelas controladas pelo Storage Engine Memory também podem ter **HASH INDEX**, que são índices baseados em endereços hash de memória: `fx00a26a4d7c2cee8e1af1958caf8365dda4`
- HASH INDEX** são muito velozes para comparações com os operadores = e <=>, enquanto que **BTREE INDEX** são mais usuáris e trabalham bem com qualquer tipo de comparação, por exemplo com o uso de **BETWEEN**;

Merge

- Uma tabela controlada pelo Storage Engine Merge, é uma coleção de tabelas MyISAM de mesma estrutura;
- Uma consulta submetida à uma tabela do tipo Merge, *percorrerá todas as tabelas MyISAM que formam a tabela controlada pelo Storage Engine Merge;
- Cada tabela controlada pelo Storage Engine Merge apresenta dois arquivos no diretório do banco de dados ao qual pertence, sob o diretório de dados do MySQL (*DATADIR*):
 - Um arquivo “.frm”: é o arquivo que contém o **CREATE TABLE**;
 - Um arquivo “.MRG”: arquivo que contém a lista de tabelas que fazem parte da tabela controlada pelo Storage Engine Merge;

Merge

- 🔗 O bloqueios em uma tabela do tipo Merge funcionam igualmente com acontece com tabelas controladas pelo Storage Engine MyISAM, em nível de tabela (*table-level locking*);
- 🔗 O Storage Engine Merge suporta as declaração **SELECT**, **DELETE**, **UPDATE** e **INSERT**. Para a última declaração, a tabela do conjunto que será afetada com a inserção uma nova linha, respeitará a configuração abordada na sintaxe do **CREATE TABLE**, que pode ser:
 - **INSERT_METHOD = LAST**: insere a nova linha da declaração **INSERT** no fim da tabela;
 - **INSERT_METHOD = FIRST**: insere a nova linha da declaração **INSERT** no início da tabela;

Merge

- Uma desvantagem é o aumento de tabelas abertas, que aumenta a quantidade de descriptors, caso precise ler ou escrever dados em uma tabela Merge que coleciona muitas outras tabelas MyISAM;
- Com o comando **ls**, no mysql client, é possível se verificar a quantidade de tabelas estão abertas no momento;
- Pouquíssimo proveito dos índices definidos nas tabelas que fazem parte da coleção, pois, por serem muitos índices, o otimizador precisa primeiro eleger um único para sua busca;
- Tabelas Merge são bastante utilizadas para dados históricos, assim como tabelas controlados pelo Storage Engine Archive;

Merge

Criando as tabelas MyISAM:

```
mysql> create table a (id int) engine=myisam;  
Query OK, 0 rows affected (0.02 sec)
```

```
mysql> create table b (id int) engine=myisam;  
Query OK, 0 rows affected (0.02 sec)
```

Criando a coleção com o Storage Engine Merge:

```
mysql> create table ab (id int)  
      > engine=merge union(a,b) insert_method=last;  
Query OK, 0 rows affected (0.01 sec)
```

Merge

Para alterar o **INSERT_METHOD**:

```
mysql> alter table ab insert_method=first;  
Query OK, 0 rows affected (0.02 sec)  
Records: 0  Duplicates: 0  Warnings: 0
```


INNODB

- ❧ O INNODB é o Storage Engine, de longe, o mais utilizado no mundo por suas características e suporte a vários features importantes;
- ❧ Ao criar uma tabela controlado pelo Storage Engine INNODB, um arquivo “.frm” é criado no diretório do banco de dados, sob o diretório de dados do MySQL (*DATADIR*);
- ❧ Índices e dados são armazenados dentro Tablespace, localizado, em uma instalação padrão, um diretório acima do *DATADIR* (MS Windows) ou em `/var/lib/mysql`, com o nome de *ibdata1*, onde são armazenados também os segmentos de undo ou “*rollback segments*”;
- ❧ INNODB dá suporte às transações, suporte ao modelo ACID (atomicidade, concorrência, isolamento e durabilidade), provê *auto_recovery* muito eficiente, com 4 níveis de utilização e multiversionamento;
- ❧ Suporte à integridade Referencial com Primary Keys e Foreign Keys;

INNODB

- Em 2005, a ORACLE anunciou a compra da empresa finlandesa INNOBase OY, que desenvolveu o INNODB;
- O InnoDB é distribuído sob os termos da GNU GPL (General Public License). O software é embudito ao MySQL através de um acordo contratual;
- Por padrão, em algumas instalações, o INNODB vem habilitado por padrão. Caso se queira retirar o INNODB de uma instalação feita a partir do *source* (código fonte), instale o MySQL com a opção **--skip-innodb**;
- Podemos habilitar e desabilitar o INNODB através do arquivo de opções, my.ini ou my.cnf;

INNODB

- ❏ O INNODB possui uma série de logs que são armazenados em arquivos chamados `ib_logfilex`. Tais logs, em uma instalação padrão, ficam armazenados no mesmo diretório aonde está localizado o Tablespace;
- ❏ O Tablespace do INNODB é um Tablespace compartilhado, ou seja, armazena dados e índices de todas as tabelas INNODB que existirem no servidor MySQL (esse Tablespace é referenciado como sendo uma *single storage area*). Não existe um arquivo específico para índices e outro para dados, assim como acontece com o MyISAM, por exemplo;
- ❏ Podemos ter um Tablespace para cada tabela, adicionando a opção abaixo no arquivo de opções, `my.ini` ou `my.cnf`:

```
[mysqld]  
innodb-file-per-table
```

INNODB

- Quando adicionada a opção citada ao arquivo de opções, devemos reiniciar o MySQL para que, quando o SGBD iniciar o módulo INNODB e ler as opções setadas no arquivo de opções, ele crie um Tablespace para cada tabela;
- Notaremos que agora teremos dois arquivos no diretório do banco de dados sob o diretório de dados do MySQL (*DATADIR*):
 - Um arquivo “.frm”: normalmente criado para cada tabela controlado pelo Storage Engine INNODB;
 - Um arquivo “.ibd”: arquivo que é o Tablespace individual, criado a partir das novas configurações do arquivo de opções;

INNODB

- ↳ Muito cuidado ao configurar o INNODB para atuar com várias Tablespaces. O Tablespace principal continua sendo utilizado para armazenar o dicionário de dados e o segmento de rollback;
- ↳ Com o INNODB, podemos desenvolver os relacionamentos entre as tabelas de nosso banco de dados, utilizando chaves primárias – **PRIMARY KEY** – e chaves estrangeiras – **FOREIGN KEY**;
- ↳ Para atribuir uma constraint ou restrição de relacionamento para garantir a integridade referencial temos que saber o que é a integridade referencial e ter ciências dos cuidados que temos que tomar na hora de firmar o relacionamento entre as tabelas;
- ↳ As chaves estrangeiras, suportadas pelo Storage Engine INNODB tem suas respectivas propriedades de **CASCADE**, **NO ACTION**, **SET NULL** e **SET DEFAULT**;

INNODB

Para criar **FOREIGN KEYS**, temos que nos atentar para as seguintes regras:

- As colunas envolvidas no relacionamento devem ser indexadas. Caso um índice não seja criado explicitamente para a **FOREIGN KEY**, o MySQL criará um automaticamente;
- As colunas envolvidas no relacionamento devem ter o mesmo tipo de dados numérico;
- As colunas devem utilizar ou não a propriedade **UNSIGNED**;

Podemos atribuir os relacionamento na criação das tabelas ou ainda com a declaração **ALTER TABLE**;

Veremos a seguir as duas formas;

INNODB

↗ Declarado a **FOREIGN KEY** na criação da tabela filha:

```
mysql> create table tbl_pai (  
    -> id int not null primary key  
    -> ) engine=innodb;  
Query OK, 0 rows affected (0.02 sec)
```

```
mysql> create table tbl_filha (  
    -> id_pai int not null primary key,  
    -> constraint foreign key (id_pai)  
    -> references tbl_pai(id)  
    -> ) engine=innodb;  
Query OK, 0 rows affected (0.00 sec)
```

INNODB

Declarado a **FOREIGN KEY** com **ALTER TABLE**:

```
mysql> create table tbl_pai (  
    -> id int not null primary key  
    -> ) engine=innodb;
```

Query OK, 0 rows affected (0.03 sec)

```
mysql> create table tbl_filha (  
    -> id_pai int not null primary key  
    -> ) engine=innodb;
```

Query OK, 0 rows affected (0.01 sec)

```
mysql> alter table tbl_filha add constraint  
    -> foreign key (id_pai)references tbl_pai (id);
```

Query OK, 0 rows affected (0.13 sec)

Records: 0 Duplicates: 0 Warnings: 0

INNODB

Podíamos ter utilizado as propriedades **ON DELETE** e **ON UPDATE**, como fazemos logo abaixo, com **CASCADE**:

```
mysql> create table tbl_pai (  
    -> id int not null primary key  
    -> ) engine=innodb;  
Query OK, 0 rows affected (0.00 sec)
```

```
mysql> create table tbl_filha (  
    -> id_pai int not null primary key  
    -> ) engine=innodb;  
Query OK, 0 rows affected (0.00 sec)
```

```
mysql> alter table tbl_filha add constraint foreign key (id_pai)  
    -> references tbl_pai (id) on delete cascade on update  
    cascade;
```

```
Query OK, 0 rows affected (0.05 sec)  
Records: 0  Duplicates: 0  Warnings: 0
```



INNODB

- Colocando a propriedade **CASCADE** na atribuição de uma constraint **FOREIGN KEY**, ao excluir o registro PAI, todos os registros na tabela filha que tenham o mesmo identificador da chave primária da tabela PAI, serão excluídos;
- Colocado a propriedade **NO ACTION** na atribuição de uma constraint **FOREIGN KEY**, ao tentar excluir o registro PAI, este não será excluído por causa da amarração deste registro na tabela filha;
- Colocado a propriedade **SET NULL** na atribuição de uma constraint **FOREIGN KEY**, ao excluir o registro PAI, aonde existir o identificador do PAI – **PRIMARY KEY** – no filho, seu valor será setado para **NULL**. Logicamente, a coluna da **FOREIGN KEY** não poderá receber um valor **NULL** se ela foi declarada como **NOT NULL**;

INNODB

↳ Colocando a propriedade **SET DEFAULT** na atribuição de uma constraint **FOREIGN KEY**, ao excluir o registro PAI, todos os registros na tabela filha que tenham o mesmo identificador da chave primária da tabela PAI, recebe o dado que foi definido na cláusula **DEFAULT** na criação da tabela;

↳ **Demonstrações.**

Exercícios

Com base no conteúdo apresentado, resolva a **Lista 1** de exercícios.



Sakila on the beach!

Referência b

– Manual on-line;