



MySQL™ Technical Stuff

MySQL AB

harrison@mysql.com

<http://www.mysql.com/training>

What Will I Talk About

A: Cool Features

1. Storage Engines
2. Character Sets
3. Fulltext Search
4. Spatial Data
5. Replication

B: API Interfaces

1. C API
2. Other APIs

C: Expanding MySQL

1. New Functions in C
2. User Defined Functions

D: Questions

1. You ask me questions

Cool Features

- So what features make MySQL nifty and cool?
 - Multiple storage engines
 - Internationalization
 - Fulltext searching
 - Spatial Data
 - Replication
 - ...and More!
- Lets discuss each of the above in a bit more detail

Multiple Storage Engines

- What?
 - Able to **choose** method of storage
 - Transactional vs Non-transactional
 - InMemory vs Disk based
 - Able to mix and match as wanted on a **table** level
- Why?
 - Much more **flexible**
 - **Best** of all worlds
 - Not forced into a single solution

Internationalization

www.mysql.com/unicode

- Fully supported on **many** levels in 4.1
- Prior to 4.1, declared on a server level
- Able to declare both **character set** and **collation** (sorting order)
- Many **levels** where they are able to be set (server, database, table, column)
- Can **mix** them even in the same table

Fulltext Search

www.mysql.com/fulltext

- What?
 - **Natural language** matching technology
 - Given a string to match and it determines **relevancy**
 - Results **sorted** by relevancy
 - Allows **boolean** matching
- Why?
 - Very **easy** search engines
 - Much faster than freeform **LIKE** statements
 - Generally better results than **LIKE**

Spatial Data Manipulation

www.mysql.com/gis

- What?
 - Added in MySQL 4.1
 - Enables ability to store **geometry shapes** such as point, line, polygon, etc. in both WKT and WKB formats
 - Fully searchable using optimized **R-Tree** indexes
- Why?
 - Very useful for **GIS** applications
 - **Useful** for other applications as well
 - Find all stores in a set radius

Replication

www.mysql.com/replication

- Details
 - Very **easy** to setup and configure
 - Very **light-weight** on network
 - Implemented as **master-slave**
 - **Asynchronous** logical replication
- Why?
 - Allows for virtually **unlimited** scalability of read queries
 - Useful for hot standby/high availability
 - Load balancing

1. Supported Development Tools

C API based

- PHP
- Perl
- ODBC
- C
- C++
- Python
- Ruby
- Tcl
- Eiffel

Other

- Java (Connector/J)
- .NET

Through ODBC

- VB, VBA
- Word, Excel, Access
- Delphi
- ASP
- ADO
- .NET

MySQL C API Overview

- Through the C API, C and C++ programs can access MySQL databases and servers
- The C API is the fastest API (most other APIs rely on it)
- The C API code is distributed with MySQL
- Included in the `mysqlclient` library
- Many of the MySQL clients use the C API
 - You can use the same environment variables as the client programs

A Simple C API Usage Schema

1. Initialise a connection handler with `mysql_init()`
2. Connect to a MySQL server by calling `mysql_real_connect()`
3. Execute queries with `mysql_query()` or `mysql_real_query()`
4. Process result sets with `mysql_store_result()` or `mysql_use_result()`
5. Free result set with `mysql_free_result()`
Repeat 3 - 5 as many times as desired
6. Terminate the connection with `mysql_close()`

Processing Results

- For non `SELECT` queries find out the number of affected rows with `mysql_affected_rows()`
- For `SELECT`_like queries use `mysql_store_result()` or `mysql_use_result()`
 - with `mysql_store_result()` you retrieve the whole result set at once
 - with `mysql_use_result()` you retrieve the results row by row from the server
- In both cases you access the rows with `mysql_fetch_row()`
- After usage, free the memory with `mysql_free_result()`

Advantages With Result Set Processing Methods

`mysql_store_result()`

- You can move arbitrarily in the result set with `mysql_data_seek()` and `mysql_row_seek()`
- Find out the number of rows with `mysql_num_rows()`
- The server is not tied up for a long time

`mysql_use_result()`

- Retrieves the result set directly from the server, leaving the tables locked for the duration of the retrieval operation
- Client requires less memory for the result set
- Can therefore be faster

A Few MySQL Datatypes

- **MYSQL**
 - Represents a handle to a database connection
- **MYSQL_RES**
 - Represents the result of a query that returns rows
- **MYSQL_ROW**
 - This is a type_safe representation of one row of data
 - Implemented as an array of counted byte strings
- **MYSQL_FIELD**
 - Contains information about a field
 - Consists of the fields name, the name of its table, its type and size
 - Field values are not part of this structure, they are contained in a MYSQL_ROW structure.

Example Of C API Usage

```
#include <stdio.h>
#include <stdlib.h>
#include <mysql.h>

int main(int argc, char **argv) {
    MYSQL *mysql = NULL;
    MYSQL_RES *res;
    MYSQL_ROW row;
    char query[1024];

    mysql = mysql_init(mysql);
    if(!mysql_real_connect(mysql, "localhost", "root", "", "world", 0, NULL, 0)) {
        printf("Couldn't connect to MySQL! (%s)\n", mysql_error(mysql));
        exit(1);
    }

    sprintf(query, "SELECT code, name, population FROM Country WHERE Region like 'Nordic%' AND Population >
1000000");

    if(!mysql_real_query(mysql, query, strlen(query))) {
        res = mysql_store_result(mysql);
        printf("code\tname\tpopulation\n----\t-----\t-----\n");
        if(res) while((row = mysql_fetch_row(res)))
            printf("%s\t%s\t%10i\n", row[0], row[1], atoi(row[2]));
        mysql_free_result(res);
    }

    mysql_close(mysql);
    return(0);
}
```

The Embedded MySQL Server Library

- With the embedded MySQL server library you can run a MySQL server inside the client application

Advantages

- Increased speed
- Simple management
- Full integration with application
- No separate installation of MySQL server

Disadvantages

- You cannot connect to the server from an outside process

How To Use The Embedded MySQL Server Library

- You have to call `mysql_server_init()` before any other MySQL functions
- Each thread should be started with `mysql_thread_init()` and ended with `mysql_thread_end()`
- End the server process with `mysql_server_end()`
- Your code should be linked with `libmysqld.a` instead of `libmysqlclient.a`

Other APIs Available

- Languages that have DB interfaces they are used
 - Perl
 - Java
 - Python
- Otherwise they generally are similar to C API
 - PHP
 - TCL
 - Smalltalk

UDF Example

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <mysql/mysql.h>
#include <uuid/uuid.h>

extern "C" {
    char *uuid(UDF_INIT *initid, UDF_ARGS *args, char *result,
               unsigned long *length, char *is_null, char *error);
}

char * uuid(UDF_INIT *initid, UDF_ARGS *args, char *result,
            unsigned long *length, char *is_null, char *error) {
    uuid_t uu;

    uuid_generate(uu);
    uuid_unparse(uu, result);
    *length = 37;

    return result;
}
```

Extending MySQL

- Two ways to easily extend MySQL to add functions in C/C++
 - User Defined Function (UDF)
 - Follows a set API provided by MySQL
 - Manged with **CREATE/DROP FUNCTION** SQL statements
 - Easy to upgrade between MySQL versions (no changes)
 - Native functions
 - Modify actual MySQL source code to change
 - Generally faster than UDFs
 - Harder to maintain between versions
 - Need to calculate diffs between versions

User Defined Functions

- Code needs to implement at least one function, up to three
- First function is name of SQL function
 - I.e. SQL `FOO()` would need a function named `foo()` in C
- `foo_init()` and `foo_deinit()` are other two optional functions
 - Called once per SQL query
 - Useful to allocate memory only once for extra performance