Introduction To MySQL

Instructor: Dr. Mark Llewellyn
markl@cs.ucf.edu
HEC 236, 407-823-2790

Department of Electrical Engineering and Computer Science
University of Central Florida
MySQL RDBMS

- MySQL is a database server (although it does come with a set of simple client programs). The current stable version is 5.1.50 and can be downloaded from www.mysql.com. (Any of the versions of MySQL 5.1.32 or greater will be fine for our purposes.)

- It is typically used in thin client environments. In other words, it is used in client-server systems where the bulk of the processing and storage takes place on the server, and the client is little more than a dumb terminal.

- MySQL performs multithreaded processing, which means that multiple clients are allowed to connect to it and run queries simultaneously. This makes MySQL extremely fast and well suited to client-server environments such as Web sites and other environments that process numerous transactions for multiple users.
MySQL Community Server

MySQL Community Server is a freely downloadable version of the world's most popular open source database that is supported by an active community of open source developers and enthusiasts.
The MySQL Community Server page.
Scroll down this page and select the proper version for your system and a site to begin download. There will be a registration type form at the top of the page…you can ignore this if you wish and go straight to the download site.
Go back to the main download page and also download MySQL Workbench which contains the Administrator and MySQL Query Browser GUI tools.

The Query Browser and Administrator come in a bundle with some other tools. Scroll down and select the correct option for your machine.
Once again, go back to the main download page and select Connectors.
Download the Connector/J for use with Java applications.
Installing MySQL 5.1.50

- Once you’ve got MySQL downloaded, go through the installation process. It may vary somewhat depending on platform.

- I’ve illustrated the basic install on Windows XP over the next few pages, just to give you an idea of what you should be seeing.
Installing MySQL 5.1.50

• Once you’ve got MySQL downloaded, go through the installation process. It may vary somewhat depending on platform.

• I’ve illustrated the basic install on Windows XP over the next few pages, just to give you an idea of what you should be seeing.

• Once the Window installer is running you should see the following window appear:
Installing MySQL 5.1.50 (cont.)

Your choice here. For this course, a typical set-up will be fine.
Installing MySQL 5.1.50 (cont.)

Select the destination folder for the install.
Installing MySQL 5.1.50 (cont.)

Again, your choice here. If you want to skip the sign-up that’s fine.
If everything has good well up to this point, you should see a window similar to this one. Click the Finish button, cross your fingers, and hang-on while the installer configures your system and gets MySQL up and running as a service.
Installing MySQL 5.1.50 (cont.)

Initial server configuration window
Installing MySQL 5.1.50 (cont.)

MySQL Server Instance Configuration Wizard

MySQL Server Instance Configuration
Configure the MySQL Server 5.1 server instance.

Please select a configuration type.

- **Detailed Configuration**
  Choose this configuration type to create the optimal server setup for this machine.

- **Standard Configuration**
  Use this only on machines that do not already have a MySQL server installation. This will use a general purpose configuration for the server that can be tuned manually.
Installing MySQL 5.1.50 (cont.)

Your choice here. If you are not sure if there is already a MySQL server on your machine, choose the detailed configuration setting.

If you already have an instance of a MySQL server on your machine, you’ll see this screen first, followed by the one above. Select reconfigure instance.
Installing MySQL 5.1.50 (cont.)

Choose the developer machine option.
Installing MySQL 5.1.50 (cont.)

Choose the multifunctional database option:

- **Multifunctional Database**: General purpose databases. This will optimize the server for the use of the fast transactional InnoDB storage engine and the high speed MyISAM storage engine.
- **Transactional Database Only**: Optimized for application servers and transactional web applications. This will make InnoDB the main storage engine. Note that the MyISAM engine can still be used.
- **Non-Transactional Database Only**: Suited for simple web applications, monitoring or logging applications as well as analysis programs. Only the non-transactional MyISAM storage engine will be activated.
Installing MySQL 5.1.50 (cont.)

Choose the installation path to keep InnoDB tables in same area as other MySQL files
Installing MySQL 5.1.50 (cont.)

Select manual setting for this option. The default is 15, I set mine to 10, but you can use any number you would like, but pick something greater than 3 or 4.
Installing MySQL 5.1.50 (cont.)

Accept all defaults in this window.
Installing MySQL 5.1.50 (cont.)

Your choice again
Accept default options

This option is not marked by default, but you can mark and accept it if you want to include MySQL file locations in your PATH statement.
Accept default setting and enter a password for the root (superuser with all privileges by default). Enabling root access from remote machines is only necessary if you will be accessing the DB as the root user from a remote machine – we will not be doing this in this course.

Do not enable this option.
Configuration is about to begin. Now cross your fingers, toes, and anything else you have, take a deep breath, click the Execute button and close your eyes for a few seconds.
You’ve successfully installed MySQL!!
Running MySQL 5.1.50

• If you’ve successfully installed MySQL, it should now be running as a service on your machine. It will start automatically when your machine boots.

• Go into your listing of programs (from the start menu at the bottom: All Programs) and you should see MySQL appear. Since you will be running MySQL clients a lot, it will be easier if you pin the MySQL Client to the start menu.

• To verify that MySQL is running properly as a service you can either check the process window or run a MySQL client.
Running MySQL 5.1.50 (cont.)

Hopefully, you see this output from MySQL. The MySQL server is now awaiting a command from this client.
List all databases managed by this MySQL server which are accessible to this client.

Note: new installations will contain only 3 databases: information_schema, mysql, and test.
List all databases managed by this MySQL server which are accessible to this client.

Terminate client connection.

```plaintext
mysql> show databases;
+-------------------------+
| Database                |
+-------------------------+
| information_schema      |
| hikadb                  |
| colorsurvey             |
| mysql                   |
| prog3                   |
| test                    |
+-------------------------+
6 rows in set (0.00 sec)

mysql> exit;
```
Specifying A Database Within MySQL

• Unless, it is specifically stated, in the following slides we’ll assume that the user has root-level privileges.

• To select a database for use in MySQL the `use` command must be issued. In the example below, we’ll select the `bikedb` database.
Viewing the Schema of a Relation

- To see the schema of a relation within a database, use the `describe <tablename>` command as illustrated below.

```sql
mysql> use bikedb;
Database changed

mysql> create table bikes <
    -> bikename varchar(30) not null,
    -> size int(2),
    -> color varchar(15),
    -> cost int(5),
    -> purchased date,
    -> mileage int(6),
    -> primary key <bikename>
    -> >;
Query OK, 0 rows affected (0.06 sec)

mysql> describe bikes;
+ Field  | Type      | Null | Key | Default | Extra +
|---------|-----------|------|-----|---------|-------+
| bikename| varchar   | NO   | PRI | NULL    |       |
| size    | int(2)    | YES  |     | NULL    |       |
| color   | varchar   | YES  |     | NULL    |       |
| cost    | int(5)    | YES  |     | NULL    |       |
| purchased| date     | YES  |     | NULL    |       |
| mileage | int(6)    | YES  |     | NULL    |       |
|         |           |      |     |         |       |
6 rows in set (0.00 sec)
```

Specify which table’s schema to describe. All information regarding the schema visible to the user is displayed.
Viewing the Relations of a Database

• Once a database has been selected you can see the relations (tables) within that database with the `show tables` command as illustrated below.

```sql
mysql> show tables;
+--------------------------+
| Tables_in_bikedb         |
| bikes                    |
+--------------------------+
1 row in set (0.00 sec)
mysql>
```

Show tables command lists all the relations within a database visible to the user. There are two tables in this database.
Running a Simple Select Query in MySQL

- Within the MySQL monitor, running an SQL query is straightforward. The example below illustrates a simple selection query on the `bikes` table of the `bikedb` database.

```
mysql> select * from bikes;

+---------+--------+--------+--------+----------+----------+
| bikename| size   | color  | cost   | purchased| mileage  |
|----------+--------+--------+--------+----------+----------+
| Colnago Dream Rabobank | 60     | blue/orange | 5500  | 2002-07-07 | 4300     |
| Bianchi Evolution 3   | 58     | celeste | 4800  | 2003-11-12 | 2000     |
| Eddy Merckx Molteni  | 58     | orange  | 5100  | 2004-08-12 | 0        |
| Eddy Merckx Dono     | 58     | blue/black | 5300  | 2004-02-02 | 0        |
| Battaglin Carrera    | 60     | red/white | 4000  | 2001-03-10 | 11200    |
| Gianni Motta Personal| 59     | red/green | 4400  | 2000-05-01 | 8700     |
| Gios Torino Super    | 60     | blue    | 2000  | 1998-11-08 | 9000     |
| Schwinn Paramount P14| 60     | blue    | 1800  | 1992-03-01 | 200      |
| Bianchi Corse Evo 4  | 58     | celeste | 5700  | 2004-12-02 | 300      |
| Colnago Superissimo  | 59     | red     | 3800  | 1996-03-01 | 13000    |
+---------+--------+--------+--------+----------+----------+
10 rows in set (0.00 sec)
```
Creating a Database in MySQL

- From the MySQL monitor enter `create database <db name>`
Dropping a Database in MySQL

- From the MySQL monitor execute the `drop database <dbname>` command.

From within the MySQL monitor, no warning is given when dropping a database. Be very sure that this is what you want to do before you do it.
Manipulating Tables in MySQL

• The creation of a database does not place any relations into the database. Relations must be separately created.

• To create a table within a database, first select the database (or create one if you haven’t already done so), then execute the `create table` command.

```sql
mysql> use sample;
Database changed
mysql> create table articles <
    -> article_id int(9) not null auto_increment,
    -> headline text not null,
    -> data_post datetime not null default '0000-00-00 00:00:00',
    -> text_body text,
    -> who_created int(9) default null,
    -> email_sent int(1) not null default '0',
    -> date_email datetime default null,
    -> who_approved int(9) default null,
    -> pic varchar(255) default null,
    -> primary key (article_id)
    -> );
Query OK, 0 rows affected (0.04 sec)

mysql>
```
Manipulating Tables in MySQL (cont.)

Screen shot that describes the newly created table.

```
mysql> describe articles;
+----------------+----------+--------+----------------+-------+-----------------+-----------+-------------+-----------------+
<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Null</th>
<th>Key</th>
<th>Default</th>
<th>Extra</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>article_id</td>
<td>int(9)</td>
<td>NO</td>
<td>PRI</td>
<td>NULL</td>
<td>auto_increment</td>
<td></td>
</tr>
<tr>
<td>headline</td>
<td>text</td>
<td>NO</td>
<td></td>
<td>NULL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>data_post</td>
<td>datetime</td>
<td>NO</td>
<td></td>
<td>0000-00-00 00:00:00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>text_body</td>
<td>text</td>
<td>YES</td>
<td></td>
<td>NULL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>who_created</td>
<td>int(9)</td>
<td>YES</td>
<td></td>
<td>NULL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>email_sent</td>
<td>int(1)</td>
<td>NO</td>
<td></td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>date_email</td>
<td>datetime</td>
<td>YES</td>
<td></td>
<td>NULL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>who_approved</td>
<td>int(9)</td>
<td>YES</td>
<td></td>
<td>NULL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>pic</td>
<td>varchar&lt;255&gt;</td>
<td>YES</td>
<td></td>
<td>NULL</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
+----------------+----------+--------+----------------+-------+-----------------+-----------+
9 rows in set (0.00 sec)
```
Manipulating Tables in MySQL (cont.)

- The `create table` command has the following general format:

  ```sql
  create [temporary] table
  [if not exists] tablename
  [(create_definition, ...)]
  [table_options] [select_statement];
  ```

- If the `[if not exists]` clause is present, MySQL will produce an error message if a table with the specified name already exists in the database, otherwise the table is created.
Manipulating Tables in MySQL (cont.)

- A temporary table exists only for the life of the current database connection. It is automatically destroyed when the connection is closed or dies.

- Two different connections can use the same name for a temporary table without conflicting with one another.

- Temporary tables are most useful when queries get complex and intermediate results become useful. Also, versions of MySQL earlier than version 4.1 do not have subselect capability and temporary tables are a convenient way to simulate subselect query results.

Note: Non-root users require special permission to be able to create temporary tables. These users must have the Create_tmp_tables privilege set in the user grant table. We’ll see more on this later.
Creating A Temporary Table From A Select Query

A SELECT query produces a result set which has been extracted from one or more tables. A table can be created with the results of this data using the create table command.

Notice that temporary tables do not appear in a table listing.
Manipulating Tables in MySQL (cont.)

- Recall that the `create table` command has the following general format:

  ```sql
  create [temporary] table
  [if not exists] tablename
  [(create_definition, ...)]
  [table_options]
  [select_statement];
  ```

- The table options allow you to specify the MySQL table type. The table type can be anyone of the six types listed in the table on the next slide.
Manipulating Tables in MySQL (cont.)

<table>
<thead>
<tr>
<th>Table Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ISAM</td>
<td>MySQL’s original table handler</td>
</tr>
<tr>
<td>HEAP</td>
<td>The data for this table is only stored in memory</td>
</tr>
<tr>
<td>MyISAM</td>
<td>A binary portable table handler that has replaced ISAM</td>
</tr>
<tr>
<td>MERGE</td>
<td>A collection of MyISAM tables used as one table</td>
</tr>
<tr>
<td>BDB</td>
<td>Transaction-safe tables with page locking</td>
</tr>
<tr>
<td>InnoDB</td>
<td>Transaction-safe tables with row locking</td>
</tr>
</tbody>
</table>

MySQL Table Types

ISAM, HEAP, and MyISAM are available for MySQL versions 3.23.6 or later.
MERGE, BDB, and InnoDB are available for MySQL versions 4.0 and later.

Default table type is InnoDB for MySQL versions 5.1.50.x.
Alter a Table

• After a table has been created, it is possible to change the specifications of its schema. This is done through the `alter table` command:

```
alter table table_name action_list
```

– Note: Changing the schema of a table in a database is not something that is done very often once the database has been created. The time for altering the schema is during the design phase. Altering the schema of an operational database is a very dangerous thing.

• Multiple changes to the table can be made at the same time by separating actions with commas in the `action_list`.

• The possible attribute (column) actions that can be used are shown in the table on the following slide.
## Altering A Table (cont.)

<table>
<thead>
<tr>
<th>Action Syntax</th>
<th>Action Performed</th>
</tr>
</thead>
<tbody>
<tr>
<td>add [column] column_declaration [first</td>
<td>after column_name]</td>
</tr>
<tr>
<td>alter [column] column_name {set default literal</td>
<td>drop default}</td>
</tr>
<tr>
<td>change [column] column_name column_declaration</td>
<td>Modify column declaration with renaming of column</td>
</tr>
<tr>
<td>modify [column] column_declaration</td>
<td>Modify column declaration without renaming column</td>
</tr>
<tr>
<td>drop [column] column_name</td>
<td>Drop a column and all data contained within it.</td>
</tr>
<tr>
<td>rename [as] new_table_name</td>
<td>Rename a table</td>
</tr>
<tr>
<td>table_options</td>
<td>Change the table options</td>
</tr>
</tbody>
</table>

Actions performed by alter table (column related) command

column_name represents the current name of the column, column_declaration represents the new declaration, in the same format as if it were in a create command.
Altering A Table (cont.)

- The screen shot below shows an example of altering a table.

![Schema of bikes before alteration](image1)

There are 10 rows affected because this table currently contains 10 tuples (rows) and the new attribute has been added to all rows.

![Bikes table after the addition of a new column named races_won](image2)
Alterning A Table (cont.)

- The screen shot below shows the tuples currently in the bikes table after the addition of the new attribute illustrating that all of the tuples have assumed the default value on the new attribute.

![Screen shot of MySQL query results](image)

Every tuple in the table has the default value for the new attribute.
Altering A Table (cont.)

- The screen shot below illustrates dropping a column from a table.
- Note that in general, this type of operation may not always be allowed due to constraint violations.

The attribute races_won has been eliminated from the table.
Altering A Table (cont.)

- The screen shot below shows a more complicated example of altering a table.

```
mysql> alter table bikes
   -> add column lastoverhaul datetime after bikename,
   -> modify cost int(8),
   -> add column races_ridden int(3) after mileage;
Query OK, 10 rows affected (0.03 sec)
Records: 10  Duplicates: 0  Warnings: 0

mysql> describe bikes;
+-----------+------------+-------+--------+--------------------+------------+-------------+-----------+-----------+-----------+-----------+-----------+-----------+-----------+-----------+-----------+-----------+-----------+-----------+
<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Null</th>
<th>Key</th>
<th>Default</th>
<th>Extra</th>
</tr>
</thead>
<tbody>
<tr>
<td>bikename</td>
<td>varchar(30)</td>
<td>NO</td>
<td>PRI</td>
<td>NULL</td>
<td></td>
</tr>
<tr>
<td>lastoverhaul</td>
<td>datetime</td>
<td>YES</td>
<td></td>
<td>NULL</td>
<td></td>
</tr>
<tr>
<td>size</td>
<td>int(2)</td>
<td>YES</td>
<td></td>
<td>NULL</td>
<td></td>
</tr>
<tr>
<td>color</td>
<td>varchar(15)</td>
<td>YES</td>
<td>NULL</td>
<td>NULL</td>
<td></td>
</tr>
<tr>
<td>cost</td>
<td>int(8)</td>
<td>YES</td>
<td>NULL</td>
<td>NULL</td>
<td></td>
</tr>
<tr>
<td>purchased</td>
<td>date</td>
<td>YES</td>
<td>NULL</td>
<td>NULL</td>
<td></td>
</tr>
<tr>
<td>mileage</td>
<td>int(6)</td>
<td>YES</td>
<td>NULL</td>
<td>NULL</td>
<td></td>
</tr>
<tr>
<td>races_ridden</td>
<td>int(3)</td>
<td>YES</td>
<td>NULL</td>
<td>NULL</td>
<td></td>
</tr>
</tbody>
</table>
+-----------+------------+-------+--------+---------+---------------+
8 rows in set (0.00 sec)
```

More complicated alter table command.

Bikes table after the alteration
Inserting Data Into A Table

• Data can be entered into a MySQL table using either the insert or replace commands.

• The insert statement is the primary way of getting data into the database and has the following form:

Form 1

```sql
insert [low priority | delayed] [ignore] [into]table_name
(set) column_name1 = expression1,
column_name2 = expression2, ...  
```

Form 2

```sql
insert [low priority | delayed] [ignore] [into]table_name
[(column_name,...)]values (expression,...), (...)...
```

Form 3

```sql
insert [low priority | delayed] [ignore] [into]table_name
[(column_name,...)] select...
```
Inserting Data Into A Table (cont.)

- Form 1 of the insert statement is the most verbose, but also the most common. The `set` clause explicitly names each column and states what value (evaluated from each `expression`) should be put into the table.

- Form 2 (insert values) requires just a comma separated list of the data. For each row inserted, each data value must correspond with a column. In other words, the number of values listed must match the number of columns and the order of the value list must be the same as the columns. (In form 1, the order is not critical since each column is named.)

- Form 3 is used to insert data into a table which is the result set of a `select` statement. This is similar to the temporary table example seen earlier in the notes.

- The following couple of pages give some examples of the different forms of the `insert` command.
Examples: Inserting Data Into A Table

Using Form 1 for insertion – attribute order is not important.
Examples: Inserting Data Into A Table

```
mysql> select * from bikes;
+----------------+-------+-----------+-------+----------+---------+
| bikename       | size  | color      | cost  | purchased | mileage |
|----------------+-------+-----------+-------+----------+---------+
| Colnago Dream Rabobank | 60    | blue/orange | 5500  | 2002-07-07 | 4300    |
| Bianchi Evolution 3 | 58    | celeste   | 4800  | 2003-11-12 | 2000    |
| Eddy Mercckx Molteni     | 58    | orange    | 5100  | 2004-08-12 | 0       |
| Eddy Mercckx Domo         | 58    | blue/black | 5300  | 2004-02-02 | 0       |
| Battaglin Carrera        | 60    | red/white | 4000  | 2001-03-10 | 11200   |
| Gianni Motta Personal    | 59    | red/green | 4400  | 2000-05-01 | 8700    |
| Gios Torino Super        | 60    | blue      | 2000  | 1998-11-08 | 9000    |
| Schwinn Paramount P14    | 60    | blue      | 1800  | 1992-03-01 | 200     |
| Bianchi Corse Evo 4      | 58    | celeste   | 5700  | 2004-12-02 | 300     |
| Colnago Superissimo      | 59    | red       | 3800  | 1996-03-01 | 13000   |
| Ridley Damocles          | 58    | blue/black | 7500  | 2008-06-27 | 0       |
+----------------+-------+-----------+-------+----------+---------+
11 rows in set (0.00 sec)

mysql> insert into bikes
    -> values ('Eddy Mercckx AX5', 58, 'black/red', 8000, '2010-02-02', 150);
Query OK, 1 row affected (0.00 sec)

mysql> select * from bikes;
+----------------+-------+-----------+-------+----------+---------+
| bikename       | size  | color      | cost  | purchased | mileage |
|----------------+-------+-----------+-------+----------+---------+
| Colnago Dream Rabobank | 60    | blue/orange | 5500  | 2002-07-07 | 4300    |
| Bianchi Evolution 3 | 58    | celeste   | 4800  | 2003-11-12 | 2000    |
| Eddy Mercckx Molteni     | 58    | orange    | 5100  | 2004-08-12 | 0       |
| Eddy Mercckx Domo         | 58    | blue/black | 5300  | 2004-02-02 | 0       |
| Battaglin Carrera        | 60    | red/white | 4000  | 2001-03-10 | 11200   |
| Gianni Motta Personal    | 59    | red/green | 4400  | 2000-05-01 | 8700    |
| Gios Torino Super        | 60    | blue      | 2000  | 1998-11-08 | 9000    |
| Schwinn Paramount P14    | 60    | blue      | 1800  | 1992-03-01 | 200     |
| Bianchi Corse Evo 4      | 58    | celeste   | 5700  | 2004-12-02 | 300     |
| Colnago Superissimo      | 59    | red       | 3800  | 1996-03-01 | 13000   |
| Ridley Damocles          | 58    | blue/black | 7500  | 2008-06-27 | 0       |
| Eddy Mercckx AX5         | 58    | black/red | 8000  | 2010-02-02 | 150     |
+----------------+-------+-----------+-------+----------+---------+
12 rows in set (0.00 sec)
```

Using Form 2 for insertion – attribute order is important.
Examples: Inserting Data Into A Table

Creating an initially empty table just like the `bikes` table:

```sql
mysql> create table celestebikes like bikes;
Query OK, 0 rows affected (0.04 sec)
```

Table creation did not place any data into the table:

```sql
mysql> select * from celestebikes;
Empty set (0.00 sec)
```

Using Form 3 for insertion:

```sql
mysql> insert into celestebikes
    -> select *
    -> from bikes
    -> where color = "celeste";
Query OK, 2 rows affected (0.00 sec)
Records: 2 Duplicates: 0 Warnings: 0
```

This table contains the name and cost of those bikes whose color was `celeste` from the source table:

```sql
mysql> select * from celestebikes;
+----------+-----+-------+-----+--------+---------+
| bikename | size| color | cost| purchased| mileage |
|----------+-----+-------+-----+--------+---------+
| Bianchi Evolution 3 | 58  | celeste | 4800| 2003-11-12 | 2000    |
| Bianchi Corse Evo 4  | 58  | celeste | 5700| 2004-12-02 | 300     |
+----------+-----+-------+-----+--------+---------+
2 rows in set (0.00 sec)
```
Examples: Inserting Data Into A Table

Create an initially empty table with a schema different from the base table.

Using Form 3 for insertion

This table contains the those bike tuples whose color was celeste from the source table.
Using Scripts with MySQL

• Entering data to create sample databases using conventional SQL commands is tedious and prone to errors. A much simpler technique is to use scripts. The following illustrates two techniques for invoking scripts in MySQL.

• Create your script file using the text editor of your choice.

• Comments in the SQL script files begin with a # symbol.

• In the script file example shown on the next slide, I drop the database in the first SQL command. Without the if exists clause, this will generate an error if the database does not exist. The first time the script executes (or subsequent executions if the database is dropped independently) the error will be generated…simply ignore the error.
Using Scripts with MySQL (cont.)

Drop the database if it already exists.
Create a new database.
Switch to the new database.
Define schema for the new table.
Insert some tuples
Run a simple selection query on the new table.

```sql
# SQL commands in a script file
drop database if exists testdb;
create database testdb;
use testdb;

create table states (name varchar(15) not null,
abbrev char(2),
capital varchar(25),
population integer,
square_miles integer,
primary key (name));

insert into states values ('Florida', 'FL', 'Tallahassee', 18328240, 54153);
insert into states values ('New York', 'NY', 'Albany', 194909297, 54556);
insert into states values ('Indiana', 'IN', 'Indianapolis', 6376792, 35789);
insert into states values ('Maryland', 'MD', 'Annapolis', 5633597, 9975);

select * from states;
```
Using Scripts with MySQL (cont.)

```
mysql> source c:\state script.sql
Query OK, 0 rows affected, 1 warning (0.00 sec)
Query OK, 1 row affected (0.00 sec)
Database changed
Query OK, 0 rows affected (0.03 sec)
Query OK, 1 row affected (0.00 sec)
Query OK, 1 row affected (0.00 sec)
Query OK, 1 row affected (0.00 sec)
Query OK, 1 row affected (0.00 sec)

<table>
<thead>
<tr>
<th>name</th>
<th>abbrev</th>
<th>capital</th>
<th>population</th>
<th>square_miles</th>
</tr>
</thead>
<tbody>
<tr>
<td>Florida</td>
<td>FL</td>
<td>Tallahassee</td>
<td>18328240</td>
<td>54153</td>
</tr>
<tr>
<td>New York</td>
<td>NY</td>
<td>Albany</td>
<td>194909297</td>
<td>54556</td>
</tr>
<tr>
<td>Indiana</td>
<td>IN</td>
<td>Indianapolis</td>
<td>6376792</td>
<td>35789</td>
</tr>
<tr>
<td>Maryland</td>
<td>MD</td>
<td>Annapolis</td>
<td>5633597</td>
<td>9975</td>
</tr>
</tbody>
</table>
```

Specify which script to execute

Results of select query at end of script.
Importing Data Using the `mysqlimport` Utility

- As with many things in MySQL there are several ways to accomplish a specific task. For getting data into tables, the `mysqlimport` utility is also useful.

- The `mysqlimport` utility reads a range of data formats, including comma- and tab-delimited, and inserts the data into a specified database table. The syntax for `mysqlimport` is:

  ```
  mysqlimport [options] database_name file1 file2 ...
  ```

- This utility is designed to be invoked from the command line.

- The name of the file (excluding the extension) must match the name of the database table into which the data import will occur. Failure to match names will result in an error.
Importing Data Using the `mysqlimportUtility`
(cont.)

- The file shown below was created to import additional data into the states table within the testdb database used in the previous example.

```
1 California CA Sacramento 36756666 155973
2 Texas TX Austin 22118509 261914
3 South Carolina SC Columbia 4147152 30111
4 Georgia GA Atlanta 9685754 47224
```

- In this case, the default field delimiter (tab), default field enclosure (nothing), and the default line delimiter (\n) were used. Many options are available and are illustrated in the table on pages 65-66.
Importing Data Using the `mysqlimport` Utility

Importing a “data file” into a MySQL database table using the `mysqlimport` utility.

```
C:\Program Files\MySQL\MySQL Server 5.1\bin>mysqlimport -u root -vr testdb c:\states.txt
Connecting to localhost
Selecting database testdb
Loading data from SERVER file: c:\states.txt into states
testdb.states:Records: 4 Deleted: 0 Skipped: 0 Warnings: 3
Disconnecting from localhost
```

Table updated

See tables on pages 23-24 for listing of options.
Importing Data Using the `mysqlimport` Utility

**Table before** another client updated the table using the `mysqlimport` utility.

<table>
<thead>
<tr>
<th>name</th>
<th>abbrev</th>
<th>capital</th>
<th>population</th>
<th>square_miles</th>
</tr>
</thead>
<tbody>
<tr>
<td>Florida</td>
<td>FL</td>
<td>Tallahassee</td>
<td>18328240</td>
<td>54153</td>
</tr>
<tr>
<td>New York</td>
<td>NY</td>
<td>Albany</td>
<td>19490297</td>
<td>54556</td>
</tr>
<tr>
<td>Indiana</td>
<td>IN</td>
<td>Indianapolis</td>
<td>6376792</td>
<td>35789</td>
</tr>
<tr>
<td>Maryland</td>
<td>MD</td>
<td>Annapolis</td>
<td>5633597</td>
<td>9975</td>
</tr>
</tbody>
</table>

4 rows in set <0.00 sec>

**Table after** another client updated the table using the `mysqlimport` utility.

```sql
mysql> select * from states;
```

<table>
<thead>
<tr>
<th>name</th>
<th>abbrev</th>
<th>capital</th>
<th>population</th>
<th>square_miles</th>
</tr>
</thead>
<tbody>
<tr>
<td>Florida</td>
<td>FL</td>
<td>Tallahassee</td>
<td>18328240</td>
<td>54153</td>
</tr>
<tr>
<td>New York</td>
<td>NY</td>
<td>Albany</td>
<td>19490297</td>
<td>54556</td>
</tr>
<tr>
<td>Indiana</td>
<td>IN</td>
<td>Indianapolis</td>
<td>6376792</td>
<td>35789</td>
</tr>
<tr>
<td>Maryland</td>
<td>MD</td>
<td>Annapolis</td>
<td>5633597</td>
<td>9975</td>
</tr>
<tr>
<td>California</td>
<td>CA</td>
<td>Sacramento</td>
<td>36756666</td>
<td>155973</td>
</tr>
<tr>
<td>Texas</td>
<td>TX</td>
<td>Austin</td>
<td>22118509</td>
<td>261914</td>
</tr>
<tr>
<td>South Carolina</td>
<td>SC</td>
<td>Columbia</td>
<td>4147152</td>
<td>30111</td>
</tr>
<tr>
<td>Georgia</td>
<td>GA</td>
<td>Atlanta</td>
<td>9685754</td>
<td>47224</td>
</tr>
</tbody>
</table>

8 rows in set <0.00 sec>
### mysqlimport Utility Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>-r</code> or <code>--replace</code></td>
<td>Causes imported rows to overwrite existing rows if they have the same unique key value.</td>
</tr>
<tr>
<td><code>-i</code> or <code>--ignore</code></td>
<td>Ignores rows that have the same unique key value as existing rows.</td>
</tr>
<tr>
<td><code>-f</code> or <code>--force</code></td>
<td>Forces mysqlimport to continue inserting data even if errors are encountered.</td>
</tr>
<tr>
<td><code>-l</code> or <code>--lock</code></td>
<td>Lock each table before importing (a good idea in general and especially on a busy server).</td>
</tr>
<tr>
<td><code>-d</code> or <code>--delete</code></td>
<td>Empty the table before inserting data.</td>
</tr>
<tr>
<td><code>--fields-terminated-by='char'</code></td>
<td>Specify the separator used between values of the same row, default \t (tab).</td>
</tr>
<tr>
<td><code>--fields-contained-by='char'</code></td>
<td>Specify the delimiter that encloses each field, default is none.</td>
</tr>
</tbody>
</table>
### mysqlimport Utility Options (cont.)

<table>
<thead>
<tr>
<th>Option</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>--fields-optionally-enclosed-by=‘char’</td>
<td>Same as --fields-enclosed-by, but delimiter is used only to enclosed string-type columns, default is none.</td>
</tr>
<tr>
<td>--fields-escaped-by=‘char’</td>
<td>Specify the escape character placed before special characters; default is .</td>
</tr>
<tr>
<td>--lines-terminated-by=‘char’</td>
<td>Specify the separator used to terminate each row of data, default is \n (newline).</td>
</tr>
<tr>
<td>-u or --user</td>
<td>Specify your username</td>
</tr>
<tr>
<td>-p or --password</td>
<td>Specify your password</td>
</tr>
<tr>
<td>-h or --host</td>
<td>Import into MySQL on the named host; default is localhost.</td>
</tr>
<tr>
<td>-s or --silent</td>
<td>Silent mode, output appears only when errors occur.</td>
</tr>
<tr>
<td>-v or --verbose</td>
<td>Verbose mode, print more commentary on action.</td>
</tr>
<tr>
<td>-? or --help</td>
<td>Print help message and exit</td>
</tr>
</tbody>
</table>
Importing Data From A File With SQL Statement Load Data Infile

- Using the utility `mysqlimport` to load data into a table from an external file works well if the user has access to a command window or command line.

- If you have access via a connection to only the MySQL database, or you are importing data from within an executing application, you will need to use the SQL statement `Load Data Infile`.

- The `Load Data Infile` statement also provides a bit more flexibility since the file name does not need to match the table name. Other than that, the options are basically the same and the same results are accomplished.

- The example on page 70 illustrates this SQL command which is available in MySQL.
Importing Data From A File With SQL Statement **Load Data Infile** (cont.)

- The basic form of the Load Data Infile statement is:

  ```sql
  LOAD DATA [LOW_PRIORITY | CONCURRENT] [LOCAL] INFILE 'filename'
  [REPLACE | IGNORE]
  INTO TABLE tablename
  [FIELDS]
  [TERMINATED BY 'char']
  [ [OPTIONALLY] ENCLOSED BY 'char' ]
  [ESCAPED BY '\char'] ]
  [LINES]
  [STARTING BY 'char']
  [TERMINATED BY 'char'] ]
  [IGNORE number LINES]
  [ (column_name, ... ) ]
  ```

  - **Either allow concurrent update or block until no other clients are reading from the specified table. See page 75.**
  - **Same as –r and –i options in mysqlimport utility – either replace or ignore rows with duplicate keys.**
  - **Sets the characters that delimit and enclose the fields and lines in the data file. Similar to mysqlimport syntax.**
  - **Ignores lines at the start of the file (miss header info).**
  - **Used to load only certain columns (not entire rows).**
Load Data Infile Example

Text file containing the data to be loaded into the database table.

String fields may be enclosed by double quotes in this file. Numeric values are not enclosed in quotes.

Fields are delimited by commas and lines are terminated by newline characters (an invisible \n)
Load data infile statement indicating all of the parameters which describe the configuration of the input file.

States table before addition of data

States table after addition of data
Load Data Infile Example 2

Text file containing the data to be loaded into the database table.

California already exists in the states table – this one will replace the value of the capital with a different value.
Same basic configuration as in previous example except that we have instructed MySQL to replace duplicate key value rows with new values (in this case replacing California's capital).

States table before addition of data

<table>
<thead>
<tr>
<th>name</th>
<th>abbrev</th>
<th>capital</th>
<th>population</th>
<th>square_miles</th>
</tr>
</thead>
<tbody>
<tr>
<td>Florida</td>
<td>FL</td>
<td>Tallahassee</td>
<td>18328240</td>
<td>54153</td>
</tr>
<tr>
<td>New York</td>
<td>NY</td>
<td>Albany</td>
<td>19490297</td>
<td>54556</td>
</tr>
<tr>
<td>Indiana</td>
<td>IN</td>
<td>Indianapolis</td>
<td>6376792</td>
<td>35789</td>
</tr>
<tr>
<td>Maryland</td>
<td>MD</td>
<td>Annapolis</td>
<td>5633597</td>
<td>9975</td>
</tr>
<tr>
<td>California</td>
<td>CA</td>
<td>Sacramento</td>
<td>36756666</td>
<td>155973</td>
</tr>
<tr>
<td>Texas</td>
<td>TX</td>
<td>Austin</td>
<td>22118507</td>
<td>261914</td>
</tr>
<tr>
<td>South Carolina</td>
<td>SC</td>
<td>Columbia</td>
<td>4147152</td>
<td>1305728</td>
</tr>
<tr>
<td>Georgia</td>
<td>GA</td>
<td>Atlanta</td>
<td>9685754</td>
<td>47224</td>
</tr>
<tr>
<td>Illinois</td>
<td>IL</td>
<td>Springfield</td>
<td>12653544</td>
<td>55593</td>
</tr>
<tr>
<td>Maine</td>
<td>ME</td>
<td>Augusta</td>
<td>1305728</td>
<td>30865</td>
</tr>
<tr>
<td>Michigan</td>
<td>MI</td>
<td>Lansing</td>
<td>10079985</td>
<td>56809</td>
</tr>
<tr>
<td>Oregon</td>
<td>OR</td>
<td>Salem</td>
<td>3559596</td>
<td>96003</td>
</tr>
<tr>
<td>Arizona</td>
<td>AZ</td>
<td>Phoenix</td>
<td>5580811</td>
<td>113642</td>
</tr>
</tbody>
</table>

13 rows in set (0.00 sec)

mysql> load data infile 'c:/states3.txt'
-> replace into table states
-> fields
-> terminated by ','
-> optionally enclosed by '"';
Query OK, 12 rows affected (0.00 sec)
Records: 6 Deleted: 6 Skipped: 0 Warnings: 0

mysql> select * from states;

<table>
<thead>
<tr>
<th>name</th>
<th>abbrev</th>
<th>capital</th>
<th>population</th>
<th>square_miles</th>
</tr>
</thead>
<tbody>
<tr>
<td>Florida</td>
<td>FL</td>
<td>Tallahassee</td>
<td>18328240</td>
<td>54153</td>
</tr>
<tr>
<td>New York</td>
<td>NY</td>
<td>Albany</td>
<td>19490297</td>
<td>54556</td>
</tr>
<tr>
<td>Indiana</td>
<td>IN</td>
<td>Indianapolis</td>
<td>6376792</td>
<td>35789</td>
</tr>
<tr>
<td>Maryland</td>
<td>MD</td>
<td>Annapolis</td>
<td>5633597</td>
<td>9975</td>
</tr>
<tr>
<td>California</td>
<td>CA</td>
<td>Sacramento</td>
<td>36756666</td>
<td>155973</td>
</tr>
<tr>
<td>Texas</td>
<td>TX</td>
<td>Austin</td>
<td>22118507</td>
<td>261914</td>
</tr>
<tr>
<td>South Carolina</td>
<td>SC</td>
<td>Columbia</td>
<td>4147152</td>
<td>1305728</td>
</tr>
<tr>
<td>Georgia</td>
<td>GA</td>
<td>Atlanta</td>
<td>9685754</td>
<td>47224</td>
</tr>
<tr>
<td>Illinois</td>
<td>IL</td>
<td>Springfield</td>
<td>12653544</td>
<td>55593</td>
</tr>
<tr>
<td>Maine</td>
<td>ME</td>
<td>Augusta</td>
<td>1305728</td>
<td>30865</td>
</tr>
<tr>
<td>Michigan</td>
<td>MI</td>
<td>Lansing</td>
<td>10079985</td>
<td>56809</td>
</tr>
<tr>
<td>Oregon</td>
<td>OR</td>
<td>Salem</td>
<td>3559596</td>
<td>96003</td>
</tr>
<tr>
<td>Arizona</td>
<td>AZ</td>
<td>Phoenix</td>
<td>5580811</td>
<td>113642</td>
</tr>
</tbody>
</table>

13 rows in set (0.00 sec)

States table after addition of data. Note that California's capital has been changed!
The **Ignore** Clause of the Insert Command

- While the normal issues of data type compatibility are always of concern, there are other issues to deal with when inserting data into tables.

- There is the possibility that a duplicate of a key may be entered. If so, you will see an error like this:

  `ERROR 1062: Duplicate entry '2' for key 1`

- It is possible to subdue errors by using the keyword `ignore` in the `insert` statement. By using `ignore` any duplicate rows will simply be ignored. They won’t be imported, and the data at the related row of the target table will be left untouched.

  - In your application, you would be wise to check how many rows were affected (imported) whenever using `ignore` because ignoring a record may constitute a failure condition in your application that needs to be handled.
Low Priority and Delayed Inserts

• If you specify `insert low-priority`, the insert waits until all other clients have finished reading from the table before the insert is executed.

• If you specify `insert delayed`, the client performing the action gets an instantaneous acknowledgement that the insert has been performed, although in fact the data will only be inserted when the table is not in use by another thread.

  – This may be useful if you have an application that needs to complete its process in minimum time, or simply where there is no need for it to wait for the effect of an insert to take place. For example, when you’re adding data to a log or audit trail.

  – This feature applies only to ISAM or MyISAM type files.
Inserting/Replacing Data Using `Replace`

- Data can also be entered into a MySQL table using the `replace` command.

- The `replace` statement has forms similar to the `insert` statement:

  **Form 1**
  ```sql
  replace [low priority | delayed] [ignore] [into] table_name
  [set] column_name1 = expression1,
  column_name2 = expression2, ...
  ```

  **Form 2**
  ```sql
  replace [low priority | delayed] [ignore] [into] table_name
  [(column_name,...)] values (expression,...), (...)
  ```

  **Form 3**
  ```sql
  replace [low priority | delayed] [ignore] [into] table_name
  [(column_name,...)] select...
  ```
Using `replace`

- The `replace` statement works similar to `insert`. It always tries to insert the new data, but when it tries to insert a new row with the same primary or unique key as an existing row, it deletes the old row and replaces it with the new values.

- The following examples will illustrate how `replace` operates.

```sql
mysql> use bikedb;
Database changed
mysql> select * from bluebikes;

<table>
<thead>
<tr>
<th>bikename</th>
<th>color</th>
<th>price</th>
<th>total_miles</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gios Torino Super</td>
<td>blue</td>
<td>2000</td>
<td>9000</td>
</tr>
<tr>
<td>Schwinn Paramount P14</td>
<td>blue</td>
<td>1800</td>
<td>200</td>
</tr>
</tbody>
</table>

2 rows in set <0.00 sec>

mysql> replace into bluebikes
   -> values ('Gios Torino Super','blue',4200,11000);
Query OK, 2 rows affected <0.00 sec>

mysql> select * from bluebikes;

<table>
<thead>
<tr>
<th>bikename</th>
<th>color</th>
<th>price</th>
<th>total_miles</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gios Torino Super</td>
<td>blue</td>
<td>4200</td>
<td>11000</td>
</tr>
<tr>
<td>Schwinn Paramount P14</td>
<td>blue</td>
<td>1800</td>
<td>200</td>
</tr>
</tbody>
</table>

2 rows in set <0.00 sec>
```

Changing non-key values. Simplest form of data replacement.
Using Replace (cont.)

Specifying values for a non-existent key. Basically the same as an insert since the key value being replaced does not currently exist.
Performing Updates on Tables

• The `update` command allows you to modify the values of the existing data in a table. The basic format of the statement is:

```sql
update [low priority] [ignore] table_name
set column_name1 = expression1,
    column_name2 = expression2, ...
[where where_definition]
[limit num];
```

• There are basically two parts to the statement: the `set` portion to declare which column to set to what value; and the `where` portion, which defines which rows are to be affected.

• `Limit` restricts the number of rows affected to `num`. 
Using `update` (cont.)

```sql
mysql> select * from bluebikes;
+ bikename      | color | price | total_miles +
+----------------+-------+-------+------------+
| Gios Torino Super | blue  | 4200  | 11000      |
| Schwinn Paramount P14 | blue  | 1800  | 200        |
| Ridley Damocles    | blue  | 8500  | 1000       |
+----------------+-------+-------+------------+
3 rows in set (0.00 sec)

mysql> update bluebikes
    -> set price=price*1.05;
Query OK, 3 rows affected (0.00 sec)
Rows matched: 3  Changed: 3  Warnings: 0

mysql> select * from bluebikes;
+ bikename      | color | price | total_miles +
+----------------+-------+-------+------------+
| Gios Torino Super | blue  | 4410  | 11000      |
| Schwinn Paramount P14 | blue  | 1890  | 200        |
| Ridley Damocles    | blue  | 8925  | 1000       |
+----------------+-------+-------+------------+
3 rows in set (0.00 sec)
```

Global update within the relation. All tuples have their price field increased by 5%.
Using `update` (cont.)

Specific update, only tuples satisfying the select condition (those with price greater than 4500) will have their price field increased by 5%.
Select Queries in MySQL

- The `select` command in MySQL is basically the same as in the standard SQL, however, it does have some additional features. The basic format of the statement is (not all options are shown – for complete details see the SQL Manual):

```
SELECT [ALL | DISTINCT | DISTINCTROW][HIGH_PRIORITY]
    [STRAIGHT_JOIN] [SQL_SMALL_RESULT][SQL_BIG_RESULT]
    [SQL_BUFFER_RESULT][SQ_CACHE | SQL_NO_CACHE]
    select_expression, ...
    [INTO {OUTFILE | DUMPFILE} 'path/to/filename' export_options]
    [FROM table_references
        WHERE where_definition]
    [GROUP BY {col_name | col_alias | col_pos | formula}
        [asc | desc], ...]
    [HAVING where_definition]
    [ORDER BY {col_name | col_alias | col_pos | formula}
        [asc | desc], ...]
    [LIMIT [offset, ] num_rows]
    [PROCEDURE procedure_name];
```
• MySQL features a user permissions system, which allows control over user’s access to the databases under MySQL control.

• There are very few competitors of MySQL (Oracle, Sybase, DB2, and SQL Server) that can match the level of sophistication provided by MySQL’s permissions system in terms of granularity and level of security provided.

Note that I did not include Microsoft Access in the list above. There are a couple of reasons for this; Access concentrates on the client front-end, although available in shareable versions, it lacks the management system that is a key part of any RDBMS. Access provides virtually no user authentication capabilities nor does it have multithreading processing capabilities, in its normal form.
Authorization in MySQL

- *mysql* and the various utility programs such as *mysqladmin*, *mysqlshow*, and *mysqlimport* can only be invoked by a valid MySQL user.

- Permissions for various users are recorded in *grant tables* maintained by MySQL.

- As the root user, you have access to all the databases and tables maintained by the MySQL Server.

- One of these databases is named *mysql* and contains the various information on the users who have access to this installation of MySQL. Some of the tables which comprise this database are shown on the next few pages.
Tables in the mysql Database

The mysql database contains user information

Details on user privileges at the database level. See page 94.

Specific details on privileges at the table level. See page 93

Details on user privileges. See page 91.

Details about the various users. See page 92.
### Contents of the `user` Table

```sql
mysql> use mysql;
Database changed
mysql> describe user;
+-------------+-----------------+----------+---------------+-------+---------------------+
<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Null</th>
<th>Key</th>
<th>Default</th>
<th>Extra</th>
</tr>
</thead>
<tbody>
<tr>
<td>Host</td>
<td>varchar(60)</td>
<td></td>
<td>PRI</td>
<td></td>
<td></td>
</tr>
<tr>
<td>User</td>
<td>varchar(16)</td>
<td></td>
<td>PRI</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Password</td>
<td>varchar(41)</td>
<td></td>
<td></td>
<td>N</td>
<td></td>
</tr>
<tr>
<td>Select_priv</td>
<td>enum('N','Y')</td>
<td></td>
<td></td>
<td>N</td>
<td></td>
</tr>
<tr>
<td>Insert_priv</td>
<td>enum('N','Y')</td>
<td></td>
<td></td>
<td>N</td>
<td></td>
</tr>
<tr>
<td>Update PRIV</td>
<td>enum('N','Y')</td>
<td></td>
<td></td>
<td>N</td>
<td></td>
</tr>
<tr>
<td>Delete_PRIV</td>
<td>enum('N','Y')</td>
<td></td>
<td></td>
<td>N</td>
<td></td>
</tr>
<tr>
<td>Create_PRIV</td>
<td>enum('N','Y')</td>
<td></td>
<td></td>
<td>N</td>
<td></td>
</tr>
<tr>
<td>Drop_PRIV</td>
<td>enum('N','Y')</td>
<td></td>
<td></td>
<td>N</td>
<td></td>
</tr>
<tr>
<td>Reload_PRIV</td>
<td>enum('N','Y')</td>
<td></td>
<td></td>
<td>N</td>
<td></td>
</tr>
<tr>
<td>Shutdown_PRIV</td>
<td>enum('N','Y')</td>
<td></td>
<td></td>
<td>N</td>
<td></td>
</tr>
<tr>
<td>Process_PRIV</td>
<td>enum('N','Y')</td>
<td></td>
<td></td>
<td>N</td>
<td></td>
</tr>
<tr>
<td>File_PRIV</td>
<td>enum('N','Y')</td>
<td></td>
<td></td>
<td>N</td>
<td></td>
</tr>
<tr>
<td>Grant_PRIV</td>
<td>enum('N','Y')</td>
<td></td>
<td></td>
<td>N</td>
<td></td>
</tr>
<tr>
<td>References_PRIV</td>
<td>enum('N','Y')</td>
<td></td>
<td></td>
<td>N</td>
<td></td>
</tr>
<tr>
<td>Index_PRIV</td>
<td>enum('N','Y')</td>
<td></td>
<td></td>
<td>N</td>
<td></td>
</tr>
<tr>
<td>Alter_PRIV</td>
<td>enum('N','Y')</td>
<td></td>
<td></td>
<td>N</td>
<td></td>
</tr>
<tr>
<td>Show_DB_PRIV</td>
<td>enum('N','Y')</td>
<td></td>
<td></td>
<td>N</td>
<td></td>
</tr>
<tr>
<td>Super_PRIV</td>
<td>enum('N','Y')</td>
<td></td>
<td></td>
<td>N</td>
<td></td>
</tr>
<tr>
<td>Create_TMP_TABLE_PRIV</td>
<td>enum('N','Y')</td>
<td></td>
<td></td>
<td>N</td>
<td></td>
</tr>
<tr>
<td>Lock_TABLES_PRIV</td>
<td>enum('N','Y')</td>
<td></td>
<td></td>
<td>N</td>
<td></td>
</tr>
<tr>
<td>Execute_PRIV</td>
<td>enum('N','Y')</td>
<td></td>
<td></td>
<td>N</td>
<td></td>
</tr>
<tr>
<td>Repl_SLAVE_PRIV</td>
<td>enum('N','Y')</td>
<td></td>
<td></td>
<td>N</td>
<td></td>
</tr>
<tr>
<td>Repl_CLIENT_PRIV</td>
<td>enum('N','Y')</td>
<td></td>
<td></td>
<td>N</td>
<td></td>
</tr>
<tr>
<td>ssl_type</td>
<td>enum('ANY','X509','SPECIFIED')</td>
<td>N</td>
<td></td>
<td>N</td>
<td></td>
</tr>
<tr>
<td>ssl_cipher</td>
<td>blob</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>x509_issuer</td>
<td>blob</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>x509_subject</td>
<td>blob</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>max_questions</td>
<td>int(11) unsigned</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>max_updates</td>
<td>int(11) unsigned</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>max_connections</td>
<td>int(11) unsigned</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
+-------------+-----------------+---------+-----+---------+---------------------+
31 rows in set (0.00 sec)
```
## Contents of the `user_info` Table

```sql
mysql> describe user_info;
+-----------------+---------+------+-----+---------+-------------------+
<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Null</th>
<th>Key</th>
<th>Default</th>
<th>Extra</th>
</tr>
</thead>
<tbody>
<tr>
<td>User</td>
<td>varchar</td>
<td>NO</td>
<td>PRI</td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>Full_name</td>
<td>varchar</td>
<td>YES</td>
<td>MUL</td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>Description</td>
<td>varchar</td>
<td>YES</td>
<td></td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>Email</td>
<td>varchar</td>
<td>YES</td>
<td></td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>Contact_information</td>
<td>text</td>
<td>YES</td>
<td></td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>Icon</td>
<td>blob</td>
<td>YES</td>
<td></td>
<td>NULL</td>
<td>NULL</td>
</tr>
</tbody>
</table>
+-----------------+---------+------|-----|---------|-------------------+
6 rows in set (0.02 sec)
mysql>
```
Contents of the `tables_priv` Table

```
mysql> \t;
mysql> describe tables_priv;

+----------------+----------+
| Field          | Type     |
|----------------+----------+
| Host           | char(60) |
| Db             | char(64) |
| User           | char(16) |
| Table_name     | char(64) |
| Grantor        | char(77) |
| Timestamp      | timestamp|
| Table_priv     | set('Select','Insert','Update','Delete','Create','Drop','Grant','References','Index',|
| Column_priv    | set('Select','Insert','Update','References')|
+----------------+----------+
8 rows in set (0.00 sec)
```
## Contents of the `db` Table

```sql
mysql> describe db;
```

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Null</th>
<th>Key</th>
<th>Default</th>
<th>Extra</th>
</tr>
</thead>
<tbody>
<tr>
<td>Host</td>
<td>char(60)</td>
<td>NO</td>
<td>PRI</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Db</td>
<td>char(64)</td>
<td>NO</td>
<td>PRI</td>
<td></td>
<td></td>
</tr>
<tr>
<td>User</td>
<td>char(16)</td>
<td>NO</td>
<td>PRI</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Select_priv</td>
<td>enum('N','Y')</td>
<td>NO</td>
<td>N</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Insert_priv</td>
<td>enum('N','Y')</td>
<td>NO</td>
<td>N</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Update_priv</td>
<td>enum('N','Y')</td>
<td>NO</td>
<td>N</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Delete_priv</td>
<td>enum('N','Y')</td>
<td>NO</td>
<td>N</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Create_priv</td>
<td>enum('N','Y')</td>
<td>NO</td>
<td>N</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Drop_priv</td>
<td>enum('N','Y')</td>
<td>NO</td>
<td>N</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Grant_priv</td>
<td>enum('N','Y')</td>
<td>NO</td>
<td>N</td>
<td></td>
<td></td>
</tr>
<tr>
<td>References_priv</td>
<td>enum('N','Y')</td>
<td>NO</td>
<td>N</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Index_priv</td>
<td>enum('N','Y')</td>
<td>NO</td>
<td>N</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Alter_priv</td>
<td>enum('N','Y')</td>
<td>NO</td>
<td>N</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Create_tmp_table_priv</td>
<td>enum('N','Y')</td>
<td>NO</td>
<td>N</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Lock_tables_priv</td>
<td>enum('N','Y')</td>
<td>NO</td>
<td>N</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Create_view_priv</td>
<td>enum('N','Y')</td>
<td>NO</td>
<td>N</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Show_view_priv</td>
<td>enum('N','Y')</td>
<td>NO</td>
<td>N</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Create_routine_priv</td>
<td>enum('N','Y')</td>
<td>NO</td>
<td>N</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Alter_routine_priv</td>
<td>enum('N','Y')</td>
<td>NO</td>
<td>N</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Execute_priv</td>
<td>enum('N','Y')</td>
<td>NO</td>
<td>N</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Event_priv</td>
<td>enum('N','Y')</td>
<td>NO</td>
<td>N</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Trigger_priv</td>
<td>enum('N','Y')</td>
<td>NO</td>
<td>N</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

22 rows in set (0.00 sec)

```sql
mysql>
```
How The Grant Tables Work

• The various grant tables work together to define access capabilities for the various users of the databases in MySQL. The tables represent a hierarchy which begins at the database level and moves downward to finer and finer granularity in access capabilities.

• To understand how the grant tables work, it is necessary to understand the process that MySQL goes through when considering a request from a client.

Step 1: A user attempts to connect to the MySQL server. The user table is consulted, and on the basis of the username, password, and host from which the connection is occurring, the connection is either refused or accepted. (MySQL actually sorts the user table and looks for the first match.)
How The Grant Tables Work (cont.)

Step 2: If the connection is accepted, any privilege fields in the user table that are set to ‘Y’ will allow the user to perform that action on any database under the server’s control. For administrative actions such as shutdown and reload, the entry in the user table is deemed absolute, and no further grant tables are consulted.

Step 3: Where the user makes a database-related request and the user table does not allow the user to perform that operation (the privilege is set to ‘N’), MySQL consults the db table (see page 84).

Step 4: The db table is consulted to see if there is an entry for the user, database, and host. If there is a match, the db privilege fields determine whether the user can perform the request.
How The Grant Tables Work (cont.)

Step 5: If there is a match on the db table’s Db and User files but Host is blank, the host table is consulted to see whether there is a match on all three fields. If there is, the privilege fields in the host table will determine whether the use can perform the requested operation. Corresponding entries in the db and host tables must both be ‘Y’ for the request to be granted. Thus, an ‘N’ in either table will block the request.

Step 6: If the user’s request is not granted, MySQL checks the tables_priv (see page 83) and columns_priv tables. It looks for a match on the user, host, database, and table to which the request is made (and the column, if there is an entry in the columns_priv table). It adds any privileges it finds in these tables to the privileges already granted. The sum of these privileges determines if the request can be granted.
Managing User Privileges with GRANT and REVOKE

- The basic granting and revocation of privileges in MySQL are accomplished through the `grant` and `revoke` commands.

- The format of the `grant` command is:

  ```
  GRANT privileges [(column_list)]
  ON database_name.table_name
  TO username@hostname [IDENTIFIED BY 'password']
  [REQUIRE [SSL | X509]
   [CIPHER cipher [AND] ]
   [ISSUER issuer [AND] ]
   [SUBJECT subject ] ]
  [WITH GRANT OPTION |
   MAX_QUERIES_PER_HOUR num |
   MAX_UPDATES_PER_HOUR num |
   MAX_CONNECTIONS_PER_HOUR num ]
  ```
Some of the Privileges Assigned with GRANT

<table>
<thead>
<tr>
<th>Privilege</th>
<th>Operations Permitted</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALL or ALL PRIVILEGES</td>
<td>All privileges except for GRANT</td>
</tr>
<tr>
<td>ALTER</td>
<td>Change a table definition using ALTER TABLE excluding the creation and dropping of indices.</td>
</tr>
<tr>
<td>CREATE</td>
<td>Create database or tables within a database.</td>
</tr>
<tr>
<td>CREATE TEMPORARY TABLES</td>
<td>Create temporary tables.</td>
</tr>
<tr>
<td>DELETE</td>
<td>Ability to perform deletions from tables. (Delete DML statements).</td>
</tr>
<tr>
<td>DROP</td>
<td>Ability to drop databases or tables.</td>
</tr>
<tr>
<td>INSERT</td>
<td>Ability to insert data into tables.</td>
</tr>
<tr>
<td>SHUTDOWN</td>
<td>Ability to shutdown the MySQL server.</td>
</tr>
</tbody>
</table>
Displaying Privileges with \texttt{SHOW}

- The SQL command \texttt{SHOW} is used to display the grant privileges for a given user.
- The syntax for the \texttt{SHOW} command is:
  \[
  \texttt{SHOW GRANTS FOR username@hostname}
  \]
- An example is shown below:

```sql
mysql> show grants for mark;
+----------------+
| Grants for mark@%
+----------------+
| GRANT SELECT, INSERT, UPDATE, DELETE, CREATE ON `.*` TO 'mark'@'%' IDENTIFIED BY PASSWORD '*E6ACCE9B249F496B191ED488F598F504239C85E73' WITH GRANT OPTION |
+----------------+
2 rows in set (0.00 sec)
mysql>
```

This user has only SELECT, INSERT, UPDATE, DELETE, and CREATE global privileges.

```sql
mysql> show grants for mark;
+----------------+
| Grants for mark@%
+----------------+
| GRANT ALL PRIVILEGES ON `bikedb`.* TO 'mark'@'%' WITH GRANT OPTION |
+----------------+
2 rows in set (0.00 sec)
mysql>
```

The user has all privileges on the bikedb database.
Revoking User Privileges with `REVOKE`:

- Revocation of privileges in MySQL is accomplished with the `revoke` command.
- The format of the `revoke` command is:

  ```
  REVOKE privileges [(column_list)]
  ON  database_name.table_name
  FROM  username@hostname
  ```

- An example is shown on the next page.
Example - Revoking User Privileges with `REVOKE`

User has SELECT privilege on `testdb.states` table.

Revoking user’s SELECT privilege on `testdb.states`.

User’s grant listing shows that they no longer have SELECT privilege on `testdb.states` table.
The MySQL Administrator Tool

- From MySQL you can download a GUI-based administrator tool to help you administer your MySQL databases.
- This tool implements all of the GRANT, REVOKE, and SHOW functionality available in SQL.
- This tool also contains some system administrator functionality for monitoring system resources and utilization.
- You can download this tool at: http://www.mysql.com/products/.
- A few screen shots of this tool and its capabilities are shown in the next few slides.
The MySQL Administrator Tool – Screen Shots

Initial login screen
Initial screen after successful login.
View of user information screen.

- **Login Information**
  - MySQL User: mark
    - The user has to enter this MySQL User name to connect to the MySQL Server
  - Password:**********
    - Fill out this field if you want to set the user's password
  - Confirm Password:**********
    - Again, enter the user's password to confirm

- **Additional Information**
  - Full Name: Mark Llewellyn
    - The user's full name
  - Description: Tall, dark, and handsome
    - Additional description of the user
  - Email: markl@cs.ucf.edu
    - The user's email address
  - Contact Information:
    - Optional contact information
  - Icon:
    - Icon assigned to the user
View of user privileges for mark user on bikedb
View of user privileges for root user on bikedb. Root user has all privileges by default.
Select a user and a database to grant or revoke privileges.
View of system catalogs which describe the databases maintained by the server.
The MySQL Query Browser Tool

• From MySQL you can also download a GUI-based query browser tool.

• This tool implements all of the basic DML side of SQL with some limitation. For example, editing result sets is possible only if the result set was generated from a single table. Join-based result sets are not editable. This tool also implements many DDL commands.

• This tool is helpful for developing and testing queries.

• A few screen shots of this tool and its capabilities are shown in the next few slides.
The MySQL Query Browser Tool

![MySQL Query Browser 1.2.17](image)

Connect to MySQL Server Instance

- **Stored Connection:**
- **Server Host:** localhost
- **Port:** 3306
- **Username:** root
- **Password:**
- **Default Schema:**

[Details >>] [OK] [Clear] [Cancel]
A First Look At The MySQL Query Browser

- Query input window
- Database selection window
- Result set window
A First Look At The MySQL Query Browser

- **Query input window**

- **Result set window**

**Result set shown for this query.** Note that this query is based on a single table, so the result set is editable.
You can manage multiple result sets simultaneously. Statistics on query execution are always available.