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 ISAM.
Altering 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:

  ```sql
  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>
</table>
| **add** [column] *column_declaration*  
  [first | after *column_name*] | Add a column to the table |
| **alter** [column] *column_name*  
  {set default *literal* | drop default} | Specify new default value for a column or remove old default |
| **change** [column] *column_name*  
  *column_declaration* | Modify column declaration with renaming of column |
| **modify** [column] *column_declaration* | Modify column declaration without renaming column |
| **drop** [column] *column_name* | Drop a column and all data contained within it. |
| **rename** [as] *new_table_name* | Rename a table |
| *table_options* | Change the table options |

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.

```
Command Prompt (2) - mysql -u root -p

mysql> describe bikes;
+ Field + Type + Null + Key + Default + Extra +
| bikename | varchar(30) | YES | PRI | NULL |
| size | int(2) | YES | | NULL |
| color | varchar(15) | YES | | NULL |
| cost | int(5) | YES | | NULL |
| purchased | date | YES | | NULL |
| mileage | int(6) | YES | | NULL |
+---------+---------+-------+-----+--------+
6 rows in set <0.05 sec>

mysql> alter table bikes
  -> add column races_won int(3) default 0;
Query OK, 8 rows affected <0.00 sec>
Records: 8  Duplicates: 0  Warnings: 0

mysql> describe bikes;
+ Field + Type + Null + Key + Default + Extra +
| bikename | varchar(30) | YES | PRI | NULL |
| size | int(2) | YES | | NULL |
| color | varchar(15) | YES | | NULL |
| cost | int(5) | YES | | NULL |
| purchased | date | YES | | NULL |
| mileage | int(6) | YES | | NULL |
| races_won | int(3) | | | 0 |
+---------+---------+-------+-----+--------+
7 rows in set <0.00 sec>
```

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

Bikes table after the addition of a new column named races_won
Altering 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.

![Image of SQL query result]

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.

```
mysql> alter table bikes
-> drop column races_won;
Query OK, 8 rows affected <0.06 sec>
Records: 8 Duplicates: 0 Warnings: 0
mysql> describe bikes;
+--------+---------+-------+-------+-------+----------+--------+
| Field  | Type    | Null | Key   | Default| Extra    |
|--------+---------+-------+-------+-------+----------+--------+
| bikername | varchar<30> | YES  | PRI   | NULL  |          |        |
| size   | int<2>  | YES   | NULL  | NULL  |          |        |
| color  | varchar<15> | YES  | NULL  | NULL  |          |        |
| cost   | int<5>  | YES   | NULL  | NULL  |          |        |
| purchased | date    | YES   | NULL  | NULL  |          |        |
| mileage| int<6>  | YES   | NULL  | NULL  |          |        |
+--------+---------+-------+-------+-------+----------+--------+
6 rows in set <0.00 sec>
mysql> -
```
Altering A Table (cont.)

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

```
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&lt;30&gt;</td>
<td>YES</td>
<td>PRI</td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>color</td>
<td>varchar&lt;15&gt;</td>
<td>YES</td>
<td></td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>cost</td>
<td>int&lt;5&gt;</td>
<td>YES</td>
<td></td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>purchased</td>
<td>date</td>
<td>YES</td>
<td></td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>mileage</td>
<td>int&lt;6&gt;</td>
<td>YES</td>
<td></td>
<td>NULL</td>
<td>NULL</td>
</tr>
</tbody>
</table>
+-------------+----------+-------+---------------+-------+-------+
6 rows in set (0.00 sec)

mysql> alter table bikes
    -> add column lastoverhaul datetime after bikename,
    -> modify cost int<6>,
    -> add column races_ridden int<3> after mileage;
Query OK, 8 rows affected (0.06 sec)
Records: 8 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&lt;30&gt;</td>
<td>YES</td>
<td>PRI</td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>lastoverhaul</td>
<td>datetime</td>
<td>YES</td>
<td></td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>size</td>
<td>int&lt;2&gt;</td>
<td>YES</td>
<td></td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>color</td>
<td>varchar&lt;15&gt;</td>
<td>YES</td>
<td></td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>cost</td>
<td>int&lt;6&gt;</td>
<td>YES</td>
<td></td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>purchased</td>
<td>date</td>
<td>YES</td>
<td></td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>mileage</td>
<td>int&lt;6&gt;</td>
<td>YES</td>
<td></td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>races_ridden</td>
<td>int&lt;6&gt;</td>
<td>YES</td>
<td></td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>-------------</td>
<td>----------</td>
<td>------</td>
<td>-----</td>
<td>---------</td>
<td>-------</td>
</tr>
</tbody>
</table>
8 rows in set (0.00 sec)
```
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
```
insert [low priority | delayed] [ignore] [into] `table_name`

[set] `column_name1 = expression1`,

`column_name2 = expression2`, ...
```

Form 2
```
insert [low priority | delayed] [ignore] [into] `table_name`

[(`column_name`,...)] values (expression,...), (...)...```

Form 3
```
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 from the previous section of 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

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

Create an initially empty table

```
create table bluebikes (  
  -> bikename varchar(30),  
  -> color varchar(15),  
  -> price int(6),  
  -> total_miles int(6),  
  -> primary key (bikename));
```

Using Form 3 for insertion

```
insert into bluebikes  
  -> select bikename,color,cost,mileage  
  -> from bikes  
  -> where color='blue';
```

This table contains the name and cost of those bikes whose color was blue 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.
Using Scripts with MySQL (cont.)

Specify which script to execute

```
mysql> show databases;
+----------------+
| Database       |
+----------------+
| bikedb         |
| mysql          |
| prog3          |
| test           |
+----------------+
4 rows in set <0.00 sec>
mysql> source c:\script.sql
Query OK, 0 rows affected, 1 warning <0.00 sec>
Query OK, 1 row affected <0.01 sec>
Database changed
Query OK, 0 rows affected <0.05 sec>
Query OK, 1 row affected <0.02 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>
```

Results of select query at end of script.

<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>17019068</td>
<td>53997</td>
</tr>
<tr>
<td>Georgia</td>
<td>GA</td>
<td>Atlanta</td>
<td>8684715</td>
<td>57919</td>
</tr>
<tr>
<td>Indiana</td>
<td>IN</td>
<td>Indianapolis</td>
<td>6195643</td>
<td>35870</td>
</tr>
<tr>
<td>Maryland</td>
<td>MD</td>
<td>Annapolis</td>
<td>5508909</td>
<td>9775</td>
</tr>
</tbody>
</table>

4 rows in set <0.00 sec>

mysql>
Piping the script file into MySQL as input. MySQL executes the script. In this case displaying the results of the select query after creating the database and table as well as filling the table. Note that MySQL exits after running the 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 **mysqlimport** Utility (cont.)

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

![Image of Notepad window with states data]

- 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 23-24.
Importing Data Using the `mysqlimport` Utility

```
mysql> show databases;
+----------------+
| Database        |
+----------------+
| bike           |
| mysql          |
| prog3          |
| test           |
| testdb         |
+----------------+
5 rows in set <0.00 sec>
mysql> use testdb;
Database changed
mysql> show tables
+------------------+
| Tables_in_testdb |
| states           |
+------------------+
1 row in set <0.00 sec>
mysql> exit
Bye
C:\Program Files\MySQL 4.1.9\MySQL Server 4.1\bin>mysqlimport -u root -p -vr testdb c:\states.sql
Enter password: ****
Connecting to localhost
Selecting database testdb
Loading data from SERVER file: c:\states.sql into states
testdb.states: Records: 4 Deleted: 0 Skipped: 0 Warnings: 0
Disconnecting from localhost
```

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

See tables on pages 23-24 for listing of options.

Table updated.
Importing Data Using the `mysqlimport` Utility

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

Table after another client updated the table using the `mysqlimport` utility.
### mysqlimport Utility Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>-r or --replace</td>
<td>Causes imported rows to overwrite existing rows if they have the same unique key value.</td>
</tr>
<tr>
<td>-i or --ignore</td>
<td>Ignores rows that have the same unique key value as existing rows.</td>
</tr>
<tr>
<td>-f or --force</td>
<td>Forces mysqlimport to continue inserting data even if errors are encountered.</td>
</tr>
<tr>
<td>-l or --lock</td>
<td>Lock each table before importing (a good idea in general and especially on a busy server).</td>
</tr>
<tr>
<td>-d or --delete</td>
<td>Empty the table before inserting data.</td>
</tr>
<tr>
<td>--fields-terminated-by='char'</td>
<td>Specify the separator used between values of the same row, default \t (tab).</td>
</tr>
<tr>
<td>--fields-enclosed-by='char'</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><code>--fields-optionally-enclosed-by='char'</code></td>
<td>Same as <code>--fields-enclosed-by</code>, but delimiter is used only to enclosed string-type columns, default is none.</td>
</tr>
<tr>
<td><code>--fields-escaped-by='char'</code></td>
<td>Specify the escape character placed before special characters; default is .</td>
</tr>
<tr>
<td><code>--lines-terminated-by='char'</code></td>
<td>Specify the separator used to terminate each row of data, default is \n (newline).</td>
</tr>
<tr>
<td><code>-u or --user</code></td>
<td>Specify your username</td>
</tr>
<tr>
<td><code>-p or --password</code></td>
<td>Specify your password</td>
</tr>
<tr>
<td><code>-h or --host</code></td>
<td>Import into MySQL on the named host; default is localhost.</td>
</tr>
<tr>
<td><code>-s or --silent</code></td>
<td>Silent mode, output appears only when errors occur.</td>
</tr>
<tr>
<td><code>-v or --verbose</code></td>
<td>Verbose mode, print more commentary on action.</td>
</tr>
<tr>
<td><code>-? or --help</code></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 27 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 32.
- 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 are 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)
States table before addition of data

Load data infile statement indicating all of the parameters which describe the configuration of the input file.

States table after addition of data
Load Data Infile Example 2

String fields are 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).

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.

States table after addition of data. Note that California’s capital has been changed!
States table before addition of data

Notice that running the same command on the altered table produced a different set of statistics, since all six key values appear in the infile, their corresponding values in the table are deleted and re-entered using the “new” data.
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 instant 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.

```
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>3800</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)
```
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:

```
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.)

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 pg 711 of SQL Manual):

```sql
SELECT [ALL | DISTINCT | DISTINCTROW][HIGH_PRIORITY]
   [STRAIGHT_JOIN] [SQL_SMALL_RESULT][SQL_BIG_RESULT]
   [SQL_BUFFER_RESULT][SQL_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 RDBMS (cont.)

- 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 46.
- Specific details on privileges at the table level. See page 45.
- Details on user privileges. See page 43.
- Details about the various users. See page 44.
### 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></td>
<td></td>
</tr>
<tr>
<td>Select_priv</td>
<td>enum('N','Y')</td>
<td>N</td>
<td>N</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Insert_priv</td>
<td>enum('N','Y')</td>
<td>N</td>
<td>N</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Update_priv</td>
<td>enum('N','Y')</td>
<td>N</td>
<td>N</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Delete_priv</td>
<td>enum('N','Y')</td>
<td>N</td>
<td>N</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Create_priv</td>
<td>enum('N','Y')</td>
<td>N</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Drop_priv</td>
<td>enum('N','Y')</td>
<td>N</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Reload_priv</td>
<td>enum('N','Y')</td>
<td>N</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Shutdown_priv</td>
<td>enum('N','Y')</td>
<td>N</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Process_priv</td>
<td>enum('N','Y')</td>
<td>N</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>File_priv</td>
<td>enum('N','Y')</td>
<td>N</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Grant_priv</td>
<td>enum('N','Y')</td>
<td>N</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>References_priv</td>
<td>enum('N','Y')</td>
<td>N</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Index_priv</td>
<td>enum('N','Y')</td>
<td>N</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Alter_priv</td>
<td>enum('N','Y')</td>
<td>N</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Show_db_priv</td>
<td>enum('N','Y')</td>
<td>N</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Super_priv</td>
<td>enum('N','Y')</td>
<td>N</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Create_tmp_table_priv</td>
<td>enum('N','Y')</td>
<td>N</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Lock_tables_priv</td>
<td>enum('N','Y')</td>
<td>N</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Execute_priv</td>
<td>enum('N','Y')</td>
<td>N</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Repl_slave_priv</td>
<td>enum('N','Y')</td>
<td>N</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Repl_client_priv</td>
<td>enum('N','Y')</td>
<td>N</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ssl_type</td>
<td>enum(' ', ANY, 'x509', 'SPECIFIED')</td>
<td>N</td>
<td></td>
<td></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>0</td>
<td></td>
</tr>
<tr>
<td>max_updates</td>
<td>int(11) unsigned</td>
<td></td>
<td></td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>max_connections</td>
<td>int(11) unsigned</td>
<td></td>
<td></td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

31 rows in set (0.00 sec)
Contents of the `user_info` Table

```
Command Prompt (2) - mysql -u root -p
16 rows in set <0.00 sec>

mysql> describe user_info;
++ Field        +----------+-------+-------+-----------+---------+
<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>varchar16</td>
<td>YES</td>
<td>PRI</td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>Full_name</td>
<td>varchar60</td>
<td>YES</td>
<td>MUL</td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>Description</td>
<td>varchar255</td>
<td>YES</td>
<td></td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>Email</td>
<td>varchar80</td>
<td>YES</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>Contact_info</td>
<td>text</td>
<td>YES</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>Icon</td>
<td>blob</td>
<td>YES</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
</tr>
</tbody>
</table>
6 rows in set <0.00 sec>
```

### Contents of the `tables_priv` Table

```sql
mysql> 
mysql> describe tables_priv;

+Field        | Type                      
-------------+--------------------------
Host         | char(60)                 
Do           | 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)
```

```
mysql>
```

<table>
<thead>
<tr>
<th>Null</th>
<th>Key</th>
<th>Default</th>
<th>Extra</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>PRI</td>
<td>PRI</td>
<td>CURRENT_TIMESTAMP</td>
</tr>
<tr>
<td></td>
<td>PRI</td>
<td>PRI</td>
<td></td>
</tr>
<tr>
<td></td>
<td>PRI</td>
<td>MUL</td>
<td></td>
</tr>
</tbody>
</table>
```
Contents of the `db` Table

```
Command Prompt (2) - mysql -u root -p

6 rows in set (0.00 sec)
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>Y</td>
<td>PRI</td>
<td>N</td>
<td></td>
</tr>
<tr>
<td>Db</td>
<td>char(64)</td>
<td>Y</td>
<td>PRI</td>
<td>N</td>
<td></td>
</tr>
<tr>
<td>User</td>
<td>char(16)</td>
<td>Y</td>
<td>PRI</td>
<td>N</td>
<td></td>
</tr>
<tr>
<td>Select_priv</td>
<td>enum('N','Y')</td>
<td>Y</td>
<td>PRI</td>
<td>N</td>
<td></td>
</tr>
<tr>
<td>Insert_priv</td>
<td>enum('N','Y')</td>
<td>Y</td>
<td>PRI</td>
<td>N</td>
<td></td>
</tr>
<tr>
<td>Update_priv</td>
<td>enum('N','Y')</td>
<td>Y</td>
<td>PRI</td>
<td>N</td>
<td></td>
</tr>
<tr>
<td>Delete_priv</td>
<td>enum('N','Y')</td>
<td>Y</td>
<td>PRI</td>
<td>N</td>
<td></td>
</tr>
<tr>
<td>Create_priv</td>
<td>enum('N','Y')</td>
<td>Y</td>
<td>PRI</td>
<td>N</td>
<td></td>
</tr>
<tr>
<td>DropPriv</td>
<td>enum('N','Y')</td>
<td>Y</td>
<td>PRI</td>
<td>N</td>
<td></td>
</tr>
<tr>
<td>Grant_priv</td>
<td>enum('N','Y')</td>
<td>Y</td>
<td>PRI</td>
<td>N</td>
<td></td>
</tr>
<tr>
<td>References_priv</td>
<td>enum('N','Y')</td>
<td>Y</td>
<td>PRI</td>
<td>N</td>
<td></td>
</tr>
<tr>
<td>Index_priv</td>
<td>enum('N','Y')</td>
<td>Y</td>
<td>PRI</td>
<td>N</td>
<td></td>
</tr>
<tr>
<td>AlterPriv</td>
<td>enum('N','Y')</td>
<td>Y</td>
<td>PRI</td>
<td>N</td>
<td></td>
</tr>
<tr>
<td>Create_tmp_table_priv</td>
<td>enum('N','Y')</td>
<td>Y</td>
<td>PRI</td>
<td>N</td>
<td></td>
</tr>
<tr>
<td>Lock_tables_priv</td>
<td>enum('N','Y')</td>
<td>Y</td>
<td>PRI</td>
<td>N</td>
<td></td>
</tr>
</tbody>
</table>

15 rows in set (0.00 sec)
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 operations (the privilege is set to ‘N’), MySQL consults the db table (see page 46).

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 user 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 45) 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 ]
  ```
<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 \textit{SHOW}

- The SQL command \textit{SHOW} is used to display the grant privileges for a given user.
- The syntax for the \textit{SHOW} command is:
  \begin{verbatim}
  SHOW GRANTS FOR username@hostname
  \end{verbatim}

- An example is shown below:

```sql
mysql> \
mysql> show grants for mark@localhost;
+ Grants for mark@localhost
+---+---+-----------------------------------------+-------------------+-------------------+
| GRANT USAGE ON `.*` TO 'mark'@'localhost' IDENTIFIED BY PASSWORD '*E6ACCDB2495496B191ED488F598F04239E6ACCDB2495496B191ED488F598F04239' |
| GRANT SELECT ON `testdb`.* TO 'mark'@'localhost' |
| GRANT ALL PRIVILEGES ON `mysql`.'bikes' TO 'mark'@'localhost' |
+---+---+-----------------------------------------+-------------------+-------------------+
3 rows in set (0.00 sec)
```

This user has only \textit{SELECT} privilege on the testdb database.

The user has all privileges on the bikes and mysql databases.
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**

```sql
mysql> show grants for mark@localhost;
+---------------------------------------------+
| Grants for mark@localhost                  |
| GRANT USAGE ON `.*` TO `mark`@`localhost` IDENTIFIED BY PASSWORD `"E5ACED2495496B191ED488F598F04239C85E73` |
| GRANT ALL PRIVILEGES ON `mysql`.`bikes` TO `mark`@`localhost` |
| GRANT SELECT ON `testdb`.`states` TO `mark`@`localhost` |
+---------------------------------------------+ 3 rows in set (0.00 sec)

mysql> revoke select
    > on testdb.states
    > from mark@localhost;
Query OK, 0 rows affected (0.00 sec)

mysql> show grants for mark@localhost;
+---------------------------------------------+
| Grants for mark@localhost                  |
| GRANT USAGE ON `.*` TO `mark`@`localhost` IDENTIFIED BY PASSWORD `"E5ACED2495496B191ED488F598F04239C85E73` |
| GRANT ALL PRIVILEGES ON `mysql`.`bikes` TO `mark`@`localhost` |
+---------------------------------------------+ 2 rows in set (0.00 sec)

mysql> 	;
```

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/administrator/
- 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.
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.

• You can download this tool at: http://dev.mysql.com/downloads/query-browser/1.1.html

• A few screen shots of this tool and its capabilities are shown in the next few slides.
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.