

COP 4710: Database Systems Spring 2006

Introduction To MySQL – Part 2

Instructor : Mark Llewellyn
markl@cs.ucf.edu
CSB 242, 823-2790
<http://www.cs.ucf.edu/courses/cop4710/spr2006>

School of Electrical Engineering and Computer Science
University of Central Florida



Manipulating Tables in MySQL (cont.)

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

```
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 Type	Description
ISAM	MySQL's original table handler
HEAP	The data for this table is only stored in memory
MyISAM	A binary portable table handler that has replaced ISAM
MERGE	A collection of MyISAM tables used as one table
BDB	Transaction-safe tables with page locking
InnoDB	Transaction-safe tables with row locking

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:

```
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.)

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

```
MySQL Command Line Client
mysql> use bikedb;
Database changed
mysql>
mysql>
mysql> describe bikes;
+-----+-----+-----+-----+-----+-----+
| Field | Type | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| bikename | varchar(30) | NO | PRI | NULL |  |
| size | int(2) | YES |  | NULL |  |
| color | varchar(15) | YES |  | NULL |  |
| cost | int(6) | YES |  | NULL |  |
| purchased | date | YES |  | NULL |  |
| mileage | int(6) | YES |  | NULL |  |
+-----+-----+-----+-----+-----+-----+
6 rows in set (0.06 sec)

mysql> alter table bikes
-> add column races_won int(3) default 0;
Query OK, 11 rows affected (0.22 sec)
Records: 11 Duplicates: 0 Warnings: 0

mysql> describe bikes;
+-----+-----+-----+-----+-----+-----+
| Field | Type | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| bikename | varchar(30) | NO | PRI | NULL |  |
| size | int(2) | YES |  | NULL |  |
| color | varchar(15) | YES |  | NULL |  |
| cost | int(6) | YES |  | NULL |  |
| purchased | date | YES |  | NULL |  |
| mileage | int(6) | YES |  | NULL |  |
| races_won | int(3) | YES |  | 0 |  |
+-----+-----+-----+-----+-----+-----+
7 rows in set (0.01 sec)

mysql>
```

Schema of bikes
before alteration

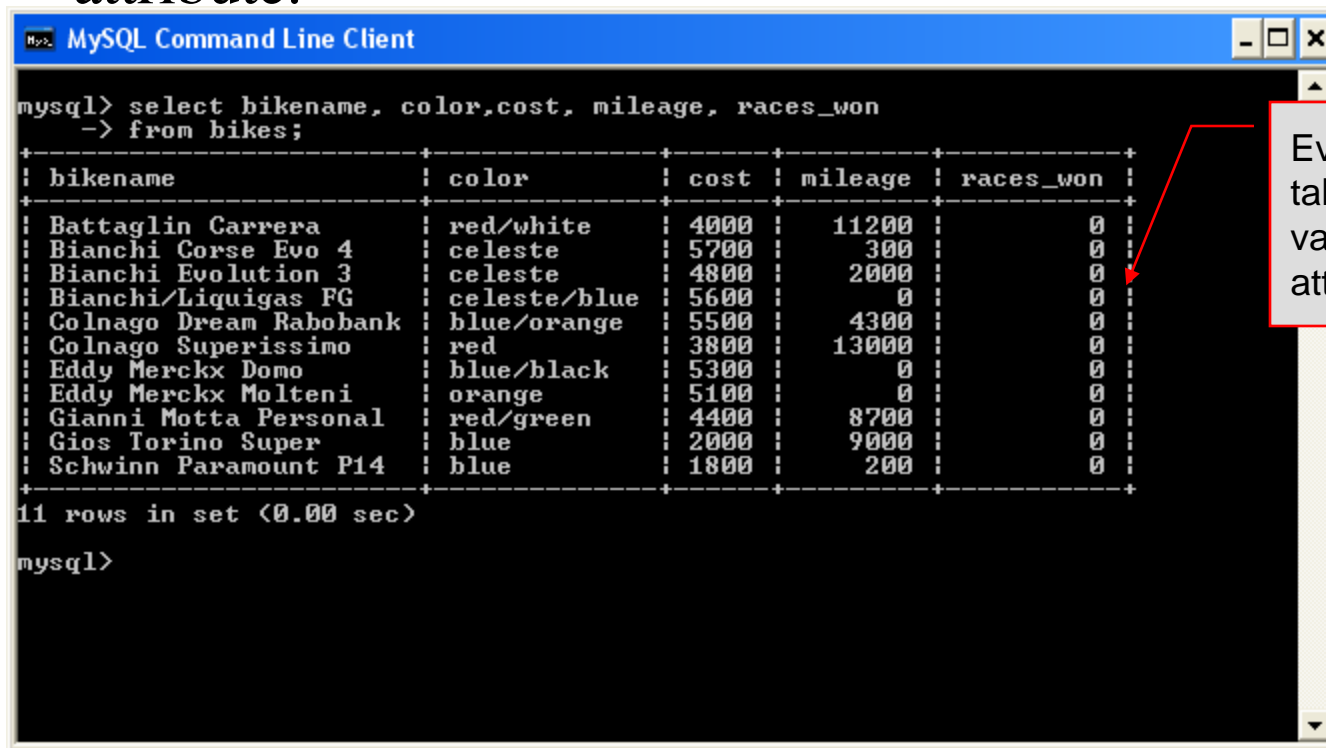
There are 11 rows affected
because this table currently
contains 11 tuples (rows) and
the new attribute has been
added to all rows.

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.



```
mysql> select bikeName, color, cost, mileage, races_won
-> from bikes;
```

bikeName	color	cost	mileage	races_won
Battaglin Carrera	red/white	4000	11200	0
Bianchi Corse Evo 4	celeste	5700	300	0
Bianchi Evolution 3	celeste	4800	2000	0
Bianchi/Liquigas FG	celeste/blue	5600	0	0
Colnago Dream Rabobank	blue/orange	5500	4300	0
Colnago Superissimo	red	3800	13000	0
Eddy Merckx Domo	blue/black	5300	0	0
Eddy Merckx Molteni	orange	5100	0	0
Gianni Motta Personal	red/green	4400	8700	0
Gios Torino Super	blue	2000	9000	0
Schwinn Paramount P14	blue	1800	200	0

11 rows in set (0.00 sec)

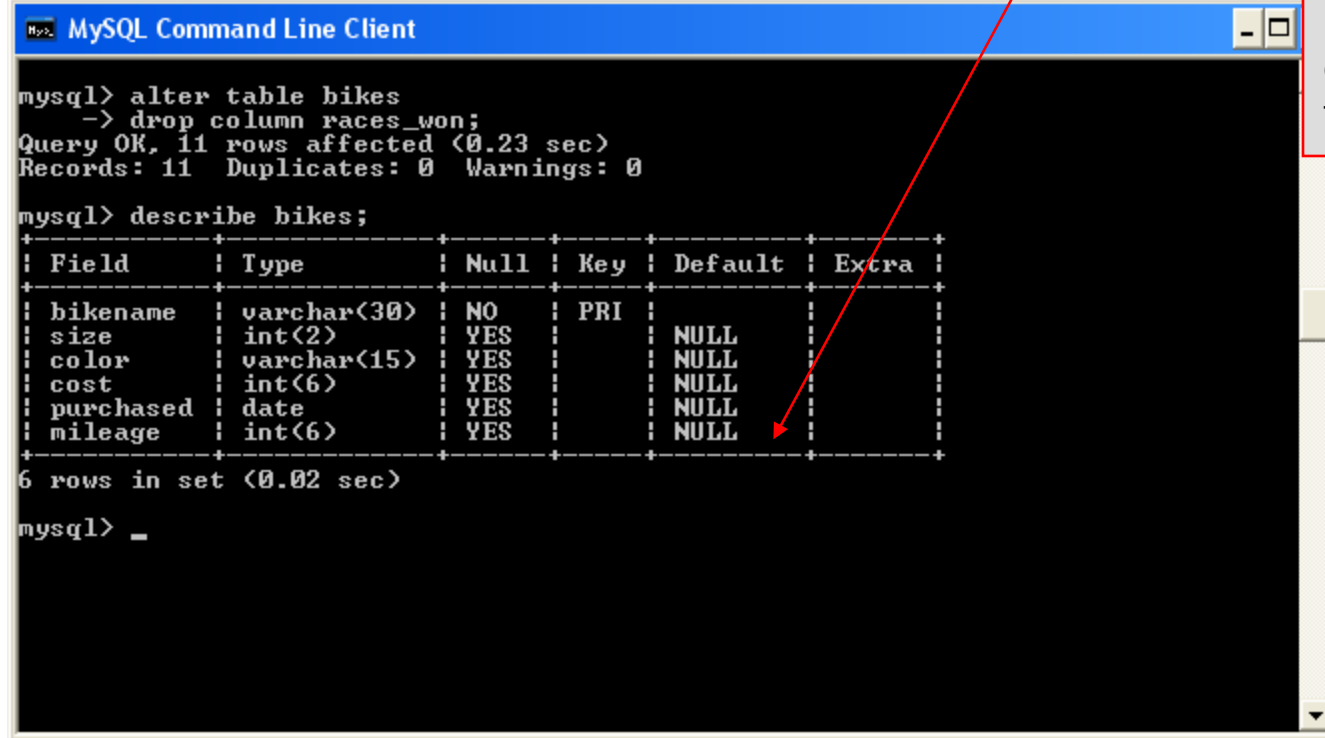
```
mysql>
```

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, 11 rows affected (0.23 sec)
Records: 11 Duplicates: 0 Warnings: 0

mysql> describe bikes;
+-----+-----+-----+-----+-----+-----+
| Field | Type | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| bike  | varchar(30) | NO | PRI | NULL |  |
| size  | int(2) | YES |  | NULL |  |
| color | varchar(15) | YES |  | NULL |  |
| cost  | int(6) | YES |  | NULL |  |
| purchased | date | YES |  | NULL |  |
| mileage | int(6) | YES |  | NULL |  |
+-----+-----+-----+-----+-----+-----+
6 rows in set (0.02 sec)

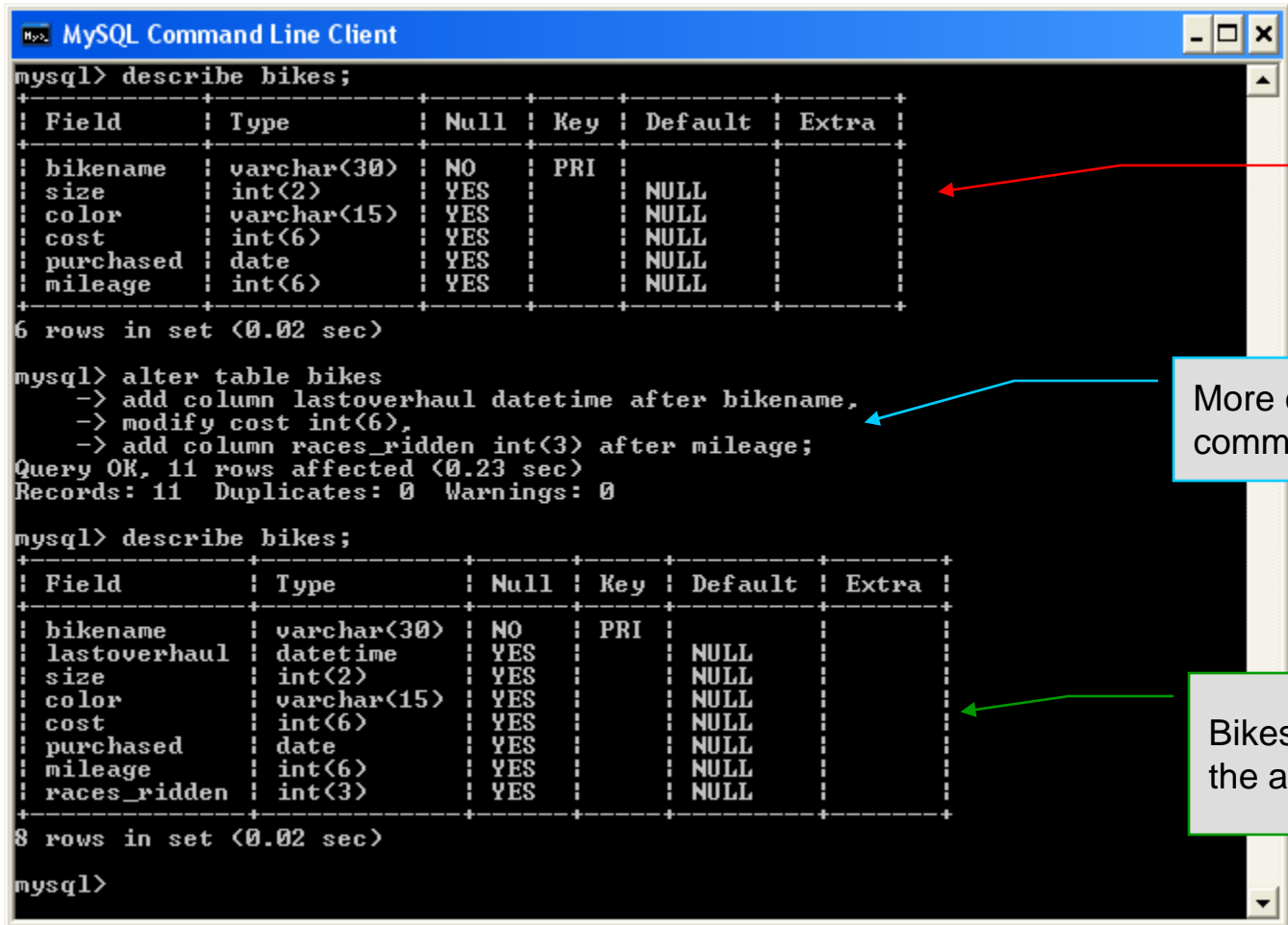
mysql> _
```

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.



The screenshot shows the MySQL Command Line Client interface. It displays the schema of the 'bikes' table before and after an alteration. The initial schema has 6 fields. The alteration command adds 'lastoverhaul' and 'races_ridden' columns and modifies the 'cost' column. The final schema has 8 fields.

Schema of bikes before alteration

Field	Type	Null	Key	Default	Extra
bikename	varchar(30)	NO	PRI		
size	int(2)	YES		NULL	
color	varchar(15)	YES		NULL	
cost	int(6)	YES		NULL	
purchased	date	YES		NULL	
mileage	int(6)	YES		NULL	

6 rows in set (0.02 sec)

More complicated alter table command.

```
mysql> alter table bikes
-> add column lastoverhaul datetime after bikename,
-> modify cost int(6),
-> add column races_ridden int(3) after mileage;
Query OK, 11 rows affected (0.23 sec)
Records: 11 Duplicates: 0 Warnings: 0
```

Bikes table after the alteration

Field	Type	Null	Key	Default	Extra
bikename	varchar(30)	NO	PRI		
lastoverhaul	datetime	YES		NULL	
size	int(2)	YES		NULL	
color	varchar(15)	YES		NULL	
cost	int(6)	YES		NULL	
purchased	date	YES		NULL	
mileage	int(6)	YES		NULL	
races_ridden	int(3)	YES		NULL	

8 rows in set (0.02 sec)

mysql>



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

```
MySQL Command Line Client

mysql> select * from bikes;
+-----+-----+-----+-----+-----+-----+
| bikename      | size | color      | cost  | purchased | mileage |
+-----+-----+-----+-----+-----+-----+
| Battaglin Carrera | 60   | red/white  | 4000  | 2001-03-14 | 11200  |
| Bianchi Corse Evo 4 | 58   | celeste    | 5700  | 2004-12-22 | 300    |
| Bianchi Evolution 3 | 58   | celeste    | 4800  | 2003-11-16 | 2000   |
| Bianchi/Liquigas FG | 58   | celeste/blue | 5600  | 2005-12-02 | 0      |
| Colnago Dream Rabobank | 60   | blue/orange | 5500  | 2002-07-27 | 4300   |
| Colnago Superissimo | 59   | red        | 3800  | 1996-03-01 | 13000  |
| Eddy Merckx Domo | 58   | blue/black | 5300  | 2005-02-02 | 0      |
| Eddy Merckx Molteni | 58   | orange     | 5100  | 2004-08-12 | 0      |
| 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    |
+-----+-----+-----+-----+-----+-----+
11 rows in set (0.00 sec)

mysql> insert into bikes
-> set bikename='Eddy Merckx MXM',
-> cost=8200,
-> mileage=150,
-> purchased='2006-01-14',
-> color='black/red',
-> size=58;
Query OK, 1 row affected (0.03 sec)

mysql> select * from bikes;
+-----+-----+-----+-----+-----+-----+
| bikename      | size | color      | cost  | purchased | mileage |
+-----+-----+-----+-----+-----+-----+
| Battaglin Carrera | 60   | red/white  | 4000  | 2001-03-14 | 11200  |
| Bianchi Corse Evo 4 | 58   | celeste    | 5700  | 2004-12-22 | 300    |
| Bianchi Evolution 3 | 58   | celeste    | 4800  | 2003-11-16 | 2000   |
| Bianchi/Liquigas FG | 58   | celeste/blue | 5600  | 2005-12-02 | 0      |
| Colnago Dream Rabobank | 60   | blue/orange | 5500  | 2002-07-27 | 4300   |
| Colnago Superissimo | 59   | red        | 3800  | 1996-03-01 | 13000  |
| Eddy Merckx Domo | 58   | blue/black | 5300  | 2005-02-02 | 0      |
| Eddy Merckx Molteni | 58   | orange     | 5100  | 2004-08-12 | 0      |
| Eddy Merckx MXM | 58   | black/red  | 8200  | 2006-01-14 | 150    |
| 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    |
+-----+-----+-----+-----+-----+-----+
12 rows in set (0.00 sec)

mysql>
```

Using Form 1
for insertion –
attribute order is
not important.



Examples: Inserting Data Into A Table

```
MySQL Command Line Client
mysql> select * from bikes;
+-----+-----+-----+-----+-----+-----+
| bikename      | size | color      | cost  | purchased | mileage |
+-----+-----+-----+-----+-----+-----+
| Battaglin Carrera | 60   | red/white  | 4000  | 2001-03-14 | 11200  |
| Bianchi Corse Evo 4 | 58   | celeste    | 5700  | 2004-12-22 | 300    |
| Bianchi Evolution 3 | 58   | celeste    | 4800  | 2003-11-16 | 2000   |
| Bianchi/Liquigas FG | 58   | celeste/blue | 5600  | 2005-12-02 | 0      |
| Colnago Dream Rabobank | 60   | blue/orange | 5500  | 2002-07-27 | 4300   |
| Colnago Superissimo | 59   | red        | 3800  | 1996-03-01 | 13000  |
| Eddy Merckx Domo    | 58   | blue/black | 5300  | 2005-02-02 | 0      |
| Eddy Merckx Molteni | 58   | orange     | 5100  | 2004-08-12 | 0      |
| 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    |
+-----+-----+-----+-----+-----+-----+
11 rows in set (0.00 sec)

mysql> insert into bikes
-> values ('Eddy Merckx MXM',58,'black/red',8200,'2006-01-14',150);
Query OK, 1 row affected (0.01 sec)

mysql> select * from bikes;
+-----+-----+-----+-----+-----+-----+
| bikename      | size | color      | cost  | purchased | mileage |
+-----+-----+-----+-----+-----+-----+
| Battaglin Carrera | 60   | red/white  | 4000  | 2001-03-14 | 11200  |
| Bianchi Corse Evo 4 | 58   | celeste    | 5700  | 2004-12-22 | 300    |
| Bianchi Evolution 3 | 58   | celeste    | 4800  | 2003-11-16 | 2000   |
| Bianchi/Liquigas FG | 58   | celeste/blue | 5600  | 2005-12-02 | 0      |
| Colnago Dream Rabobank | 60   | blue/orange | 5500  | 2002-07-27 | 4300   |
| Colnago Superissimo | 59   | red        | 3800  | 1996-03-01 | 13000  |
| Eddy Merckx Domo    | 58   | blue/black | 5300  | 2005-02-02 | 0      |
| Eddy Merckx Molteni | 58   | orange     | 5100  | 2004-08-12 | 0      |
| Eddy Merckx MXM     | 58   | black/red  | 8200  | 2006-01-14 | 150    |
| 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    |
+-----+-----+-----+-----+-----+-----+
12 rows in set (0.00 sec)

mysql>
```

Using Form 2
for insertion –
attribute order
is important.



Examples: Inserting Data Into A Table

```
MySQL Command Line Client
mysql>
mysql>
mysql> create table celestebikes like bikes;
Query OK, 0 rows affected (0.08 sec)

mysql> insert into celestebikes
-> select *
-> from bikes
-> where color = 'celeste';
Query OK, 2 rows affected (0.03 sec)
Records: 2 Duplicates: 0 Warnings: 0

mysql> select * from celestebikes;
+-----+-----+-----+-----+-----+-----+
| bikename      | size | color  | cost  | purchased | mileage |
+-----+-----+-----+-----+-----+-----+
| Bianchi Corse Evo 4 | 58   | celeste | 5700  | 2004-12-22 | 300    |
| Bianchi Evolution 3 | 58   | celeste | 4800  | 2003-11-16 | 2000   |
+-----+-----+-----+-----+-----+-----+
2 rows in set (0.00 sec)

mysql> _
```

Creates an initially empty table just like the bikes table

Using Form 3 for insertion

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



Examples: Inserting Data Into A Table

```
MySQL Command Line Client

mysql>
mysql> create table celestebikes (
  -> name varchar(30),
  -> paint varchar(15),
  -> price int(6),
  -> miles_ridden int(6),
  -> primary key (name)
  -> );
Query OK, 0 rows affected (0.11 sec)

mysql> insert into celestebikes
  -> select bikename, color, cost, mileage
  -> from bikes
  -> where color='celeste';
Query OK, 2 rows affected (0.03 sec)
Records: 2  Duplicates: 0  Warnings: 0

mysql> select * from celestebikes;
+-----+-----+-----+-----+
| name          | paint  | price | miles_ridden |
+-----+-----+-----+-----+
| Bianchi Corse Evo 4 | celeste | 5700  | 300          |
| Bianchi Evolution 3 | celeste | 4800  | 2000         |
+-----+-----+-----+-----+
2 rows in set (0.00 sec)

mysql> _
```

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.

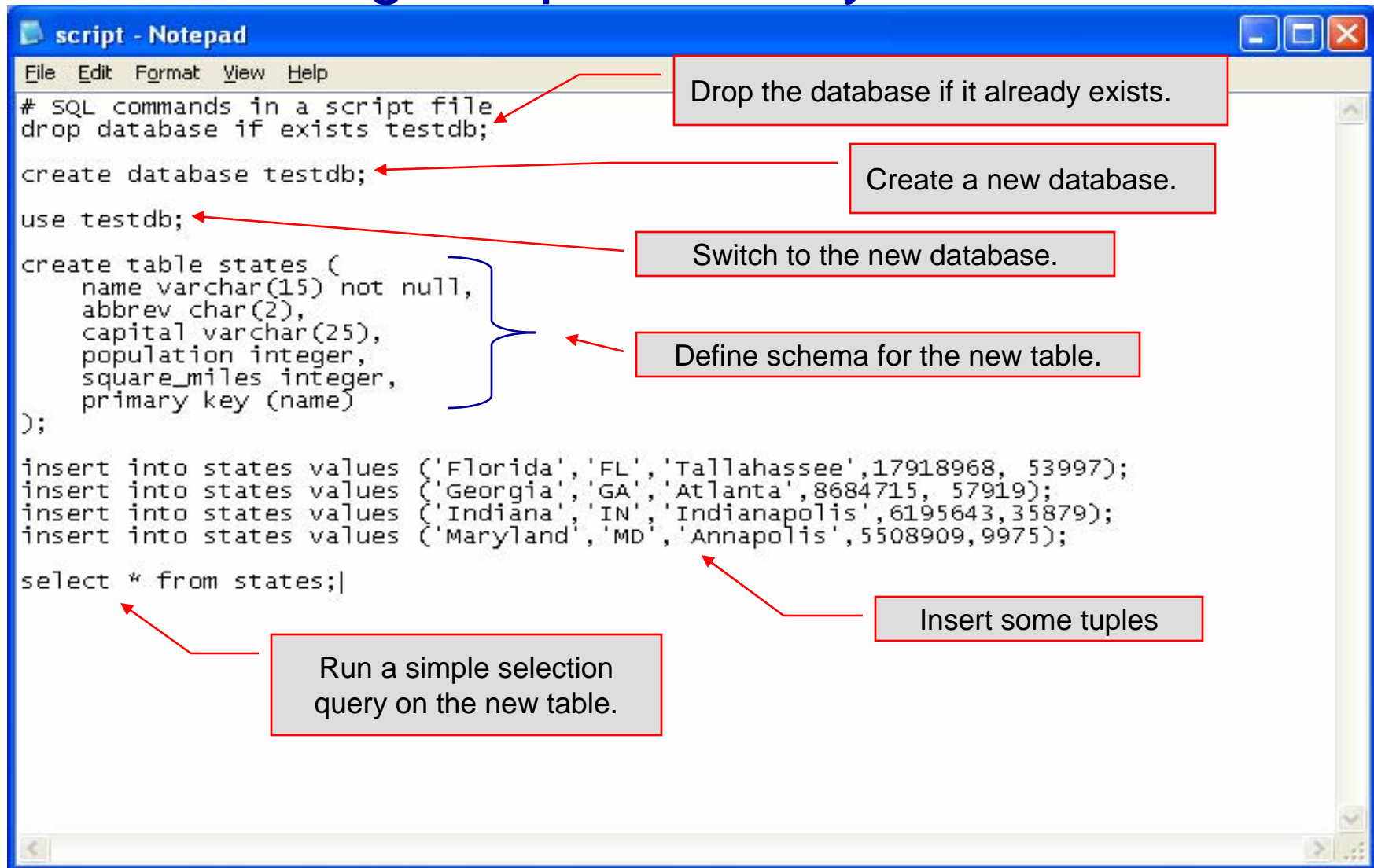


Using Scripts with MySQL

- 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.)



The screenshot shows a Notepad window with a MySQL script. Red arrows point from text boxes to specific lines in the script:

- Drop the database if it already exists.** points to `drop database if exists testdb;`
- Create a new database.** points to `create database testdb;`
- Switch to the new database.** points to `use testdb;`
- Define schema for the new table.** points to the `create table states` block.
- Insert some tuples** points to the `insert into states values` block.
- Run a simple selection query on the new table.** points to `select * from states;`

```
# 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',17918968, 53997);
insert into states values ('Georgia','GA','Atlanta',8684715, 57919);
insert into states values ('Indiana','IN','Indianapolis',6195643,35879);
insert into states values ('Maryland','MD','Annapolis',5508909,9975);

select * from states;
```



Using Scripts with MySQL (cont.)

```
MySQL Command Line Client
mysql> show databases;
+-----+
| Database |
+-----+
| information_schema |
| bikedb      |
| mysql      |
| sample     |
| test       |
+-----+
5 rows in set (0.00 sec)

mysql> source e:\courses\cop 4710 - database systems\spring 2006\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 (0.14 sec)

Query OK, 1 row affected (0.02 sec)

Query OK, 1 row affected (0.03 sec)

Query OK, 1 row affected (0.01 sec)

Query OK, 1 row affected (0.02 sec)

+-----+-----+-----+-----+-----+
| name      | abbrev | capital      | population | square_miles |
+-----+-----+-----+-----+-----+
| Florida   | FL     | Tallahassee | 17918968   | 53997         |
| Georgia   | GA     | Atlanta      | 8684715    | 57919         |
| Indiana   | IN     | Indianapolis  | 6195643    | 35879         |
| Maryland  | MD     | Annapolis    | 5508909    | 9975          |
+-----+-----+-----+-----+-----+
4 rows in set (0.00 sec)

mysql>
```

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:

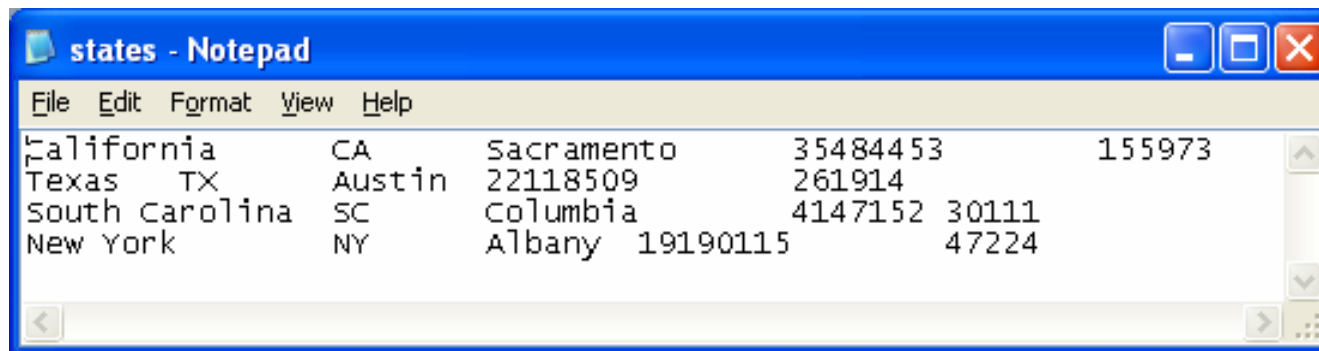
```
mysqlimport [options] database_name file1 file2 ...
```

- 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 `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.



- 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 mysqlimportUtility

```
C:\ Command Prompt (2)

mysql> show databases;
+-----+
| Database |
+-----+
| bikedb   |
| 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 tes
tdb 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

C:\Program Files\MySQL 4.1.9\MySQL Server 4.1\bin>
```

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 mysqlimportUtility

```
C:\ Command Prompt (2) - mysql -u root -p

mysql> select * from states;
+-----+-----+-----+-----+-----+
| name      | abbrev | capital    | population | square_miles |
+-----+-----+-----+-----+-----+
| Florida   | FL     | Tallahassee | 17019068   | 53997         |
| Georgia   | GA     | Atlanta      | 8684715    | 57919         |
| Indiana   | IN     | Indianapolis  | 6195643    | 35870         |
| Maryland  | MD     | Annapolis    | 5508909    | 9775          |
+-----+-----+-----+-----+-----+
4 rows in set (0.00 sec)

mysql> select * from states;
+-----+-----+-----+-----+-----+
| name      | abbrev | capital    | population | square_miles |
+-----+-----+-----+-----+-----+
| Florida   | FL     | Tallahassee | 17019068   | 53997         |
| Georgia   | GA     | Atlanta      | 8684715    | 57919         |
| Indiana   | IN     | Indianapolis  | 6195643    | 35870         |
| Maryland  | MD     | Annapolis    | 5508909    | 9775          |
| South Carolina | SC     | Columbia    | 4147152    | 30111         |
| Texas     | TX     | Austin       | 22118509   | 261914        |
| California | CA     | Sacramento   | 35484453   | 155973        |
| New York  | NY     | Albany       | 19190115   | 47224         |
+-----+-----+-----+-----+-----+
8 rows in set (0.00 sec)

mysql>
```

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

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



mysqlimportUtility Options

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



mysqlimport Utility Options (cont.)

Option	Action
--fields-optionally-enclosed-by='char'	Same as --fields-enclosed-by, but delimiter is used only to enclosed string-type columns, default is none.
--fields-escaped-by='char'	Specify the escape character placed before special characters; default is \.
--lines-terminated-by='char'	Specify the separator used to terminate each row of data, default is \n (newline).
-u or --user	Specify your username
-p or --password	Specify your password
-h or --host	Import into MySQL on the named host; default is localhost.
-s or --silent	Silent mode, output appears only when errors occur.
-v or --verbose	Verbose mode, print more commentary on action.
-? or --help	Print help message and exit



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:

```
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.

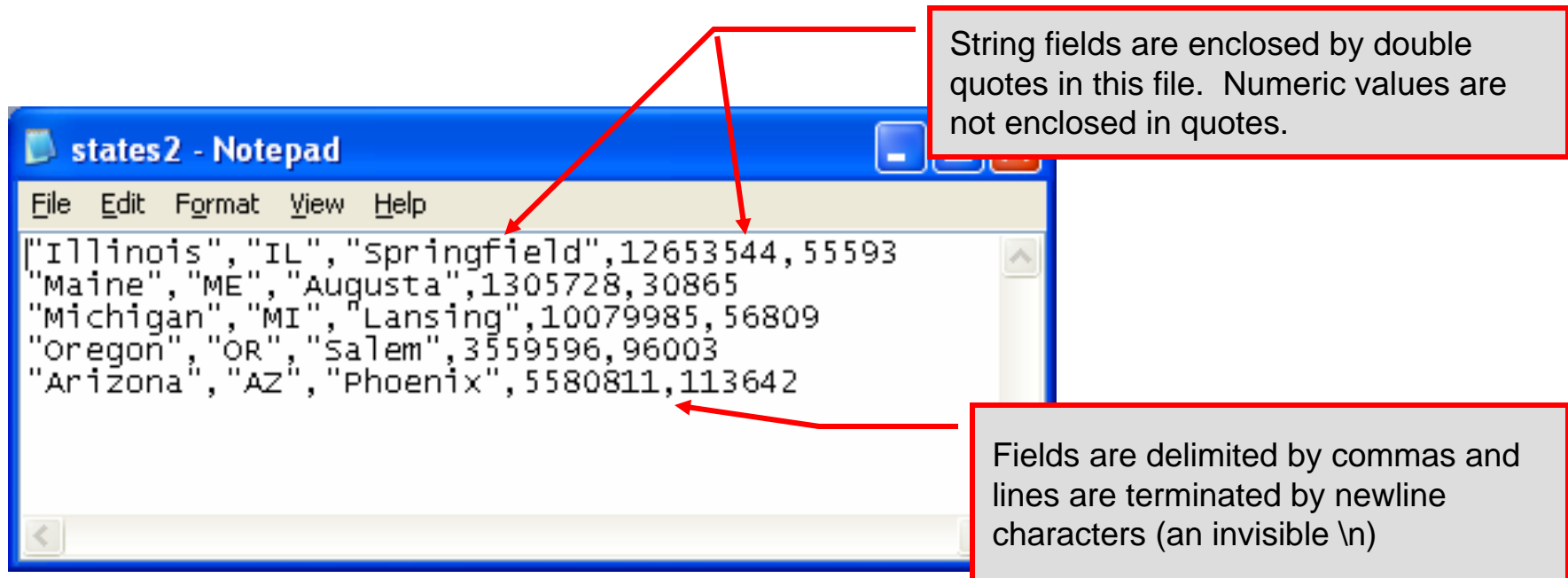
Sets the characters that delimit and enclose the fields and lines in the data file. Similar to `mysqlimport` syntax.

Ignores lines at the start of the file (miss header info)

Used to load only certain columns (not entire rows)



Load Data Infile Example



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

```
"Illinois","IL","Springfield",12653544,55593
"Maine","ME","Augusta",1305728,30865
"Michigan","MI","Lansing",10079985,56809
"Oregon","OR","Salem",3559596,96003
"Arizona","AZ","Phoenix",5580811,113642
```

Fields are delimited by commas and lines are terminated by newline characters (an invisible \n)

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



mysql> select * from states;

name	abbrev	capital	population	square_miles
Florida	FL	Tallahassee	17019068	53997
Georgia	GA	Atlanta	8684715	57919
Indiana	IN	Indianapolis	6195643	35870
Maryland	MD	Annapolis	5508909	9775
California	CA	Sacramento	35484453	155973
Texas	TX	Austin	22118509	261914
South Carolina	SC	Columbia	4147152	30111
New York	NY	Albany	19190115	47224

8 rows in set (0.00 sec)

```
mysql> load data infile 'states2.sql'
-> into table states
-> fields
-> terminated by ','
-> optionally enclosed by '"'
-> ;
```

Query OK, 5 rows affected (0.02 sec)
Records: 5 Deleted: 0 Skipped: 0 Warnings: 0

mysql> select * from states;

name	abbrev	capital	population	square_miles
Florida	FL	Tallahassee	17019068	53997
Georgia	GA	Atlanta	8684715	57919
Indiana	IN	Indianapolis	6195643	35870
Maryland	MD	Annapolis	5508909	9775
California	CA	Sacramento	35484453	155973
Texas	TX	Austin	22118509	261914
South Carolina	SC	Columbia	4147152	30111
New York	NY	Albany	19190115	47224
Illinois	IL	Springfield	12653544	55593
Maine	ME	Augusta	1305728	30865
Michigan	MI	Lansing	10079985	56809
Oregon	OR	Salem	3559596	96003
Arizona	AZ	Phoenix	5580811	113642

13 rows in set (0.00 sec)

mysql>

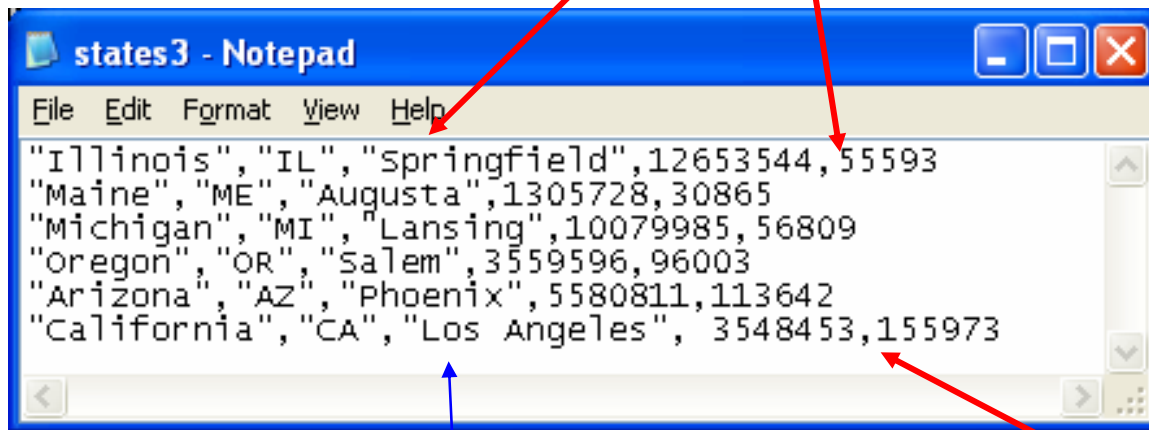
States table **before**
addition of data

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



```
states3 - Notepad
File Edit Format View Help
"Illinois","IL","Springfield",12653544,55593
"Maine","ME","Augusta",1305728,30865
"Michigan","MI","Lansing",10079985,56809
"Oregon","OR","Salem",3559596,96003
"Arizona","AZ","Phoenix",5580811,113642
"California","CA","Los Angeles", 3548453,155973
```

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

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.

Fields are delimited by commas and lines are terminated by newline characters (an invisible \n)



Database changed

mysql> select * from states;

name	abbrev	capital	population	square_miles
Florida	FL	Tallahassee	17019068	53997
Georgia	GA	Atlanta	8684715	57919
Indiana	IN	Indianapolis	6195643	35870
Maryland	MD	Annapolis	5508909	9775
California	CA	Sacramento	35484453	155973
Texas	TX	Austin	22118509	261914
South Carolina	SC	Columbia	4147152	30111
New York	NY	Albany	19190115	47224

8 rows in set (0.00 sec)

```
mysql> load data infile 'states3.sql'
-> replace into table states
-> fields
-> terminated by ','
-> optionally enclosed by '"'
-> ;
```

Query OK, 7 rows affected (0.02 sec)

Records: 6 Deleted: 1 Skipped: 0 Warnings: 0

mysql> select * from states;

name	abbrev	capital	population	square_miles
Florida	FL	Tallahassee	17019068	53997
Georgia	GA	Atlanta	8684715	57919
Indiana	IN	Indianapolis	6195643	35870
Maryland	MD	Annapolis	5508909	9775
California	CA	Los Angeles	3548453	155973
Texas	TX	Austin	22118509	261914
South Carolina	SC	Columbia	4147152	30111
New York	NY	Albany	19190115	47224
Illinois	IL	Springfield	12653544	55593
Maine	ME	Augusta	1305728	30865
Michigan	MI	Lansing	10079985	56809
Oregon	OR	Salem	3559596	96003
Arizona	AZ	Phoenix	5580811	113642

13 rows in set (0.00 sec)

mysql>

States table **before** addition of data

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 **after** addition of data. Note that California's capital has been changed!



States table **before**
addition of data

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

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

name	abbrev	capital	population	square_miles
Florida	FL	Tallahassee	17019068	53997
Georgia	GA	Atlanta	8684715	57919
Indiana	IN	Indianapolis	6195643	35870
Maryland	MD	Annapolis	5508909	9775
California	CA	Los Angeles	3548453	155973
Texas	TX	Austin	22118509	261914
South Carolina	SC	Columbia	4147152	30111
New York	NY	Albany	19190115	47224
Illinois	IL	Springfield	12653544	55593
Maine	ME	Augusta	1305728	30865
Michigan	MI	Lansing	10079985	56809
Oregon	OR	Salem	3559596	96003
Arizona	AZ	Phoenix	5580811	113642

```
13 rows in set (0.00 sec)
```

```
mysql> load data infile 'states3.sql'
```

```
-> 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>
```

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

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

Form 3 `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.

```
C:\> Command Prompt (2) - mysql -u root -p

mysql> select * from bluebikes;
+-----+-----+-----+-----+
| bikename      | color | price | total_miles |
+-----+-----+-----+-----+
| Gios Torino Super | blue  | 3800  | 9000        |
| Schwinn Paramount P14 | blue  | 1800  | 200         |
+-----+-----+-----+-----+
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;
+-----+-----+-----+-----+
| bikename      | color | price | total_miles |
+-----+-----+-----+-----+
| Gios Torino Super | blue  | 4200  | 11000       |
| Schwinn Paramount P14 | blue  | 1800  | 200         |
+-----+-----+-----+-----+
2 rows in set (0.00 sec)

mysql>
```

Changing non-key values. Simplest form of data replacement.



Using Replace (cont.)

```
C:\ Command Prompt (2) - mysql -u root -p
mysql> select * from bluebikes;
+-----+-----+-----+-----+
| bikename          | color | price | total_miles |
+-----+-----+-----+-----+
| Gios Torino Super | blue  | 4200  | 11000       |
| Schwinn Paramount P14 | blue  | 1800  | 200         |
+-----+-----+-----+-----+
2 rows in set (0.00 sec)

mysql> replace into bluebikes
-> values ('Fondriest U107','blue',5300,2200);
Query OK, 1 row affected (0.00 sec)

mysql> select * from bluebikes;
+-----+-----+-----+-----+
| bikename          | color | price | total_miles |
+-----+-----+-----+-----+
| Gios Torino Super | blue  | 4200  | 11000       |
| Schwinn Paramount P14 | blue  | 1800  | 200         |
| Fondriest U107     | blue  | 5300  | 2200        |
+-----+-----+-----+-----+
3 rows in set (0.00 sec)

mysql>
```

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.)

```
CA Command Prompt (2) - mysql -u root -p
mysql> select * from bluebikes;
+-----+-----+-----+-----+
| bikename          | color | price | total_miles |
+-----+-----+-----+-----+
| Gios Torino Super | blue  | 4200  | 11000       |
| Schwinn Paramount P14 | blue  | 1800  | 200         |
| Fondriest U107     | blue  | 5300  | 2200        |
+-----+-----+-----+-----+
3 rows in set (0.01 sec)

mysql> update bluebikes
-> set price=price*1.05;
Query OK, 3 rows affected (0.02 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         |
| Fondriest U107     | blue  | 5565  | 2200        |
+-----+-----+-----+-----+
3 rows in set (0.00 sec)

mysql>
```

Global update within the relation. All tuples have their price field increased by 5%



Using update (cont.)

```
C:\ Command Prompt (2) - mysql -u root -p
mysql> select * from bluebikes;
+-----+-----+-----+-----+
| bikename      | color | price | total_miles |
+-----+-----+-----+-----+
| Gios Torino Super | blue  | 4200  | 11000       |
| Schwinn Paramount P14 | blue  | 1800  | 200         |
| Fondriest U107    | blue  | 5300  | 2200        |
+-----+-----+-----+-----+
3 rows in set (0.00 sec)

mysql> update bluebikes
-> set price = price * 1.05;
-> where price > 4500;
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> select * from bluebikes;
+-----+-----+-----+-----+
| bikename      | color | price | total_miles |
+-----+-----+-----+-----+
| Gios Torino Super | blue  | 4200  | 11000       |
| Schwinn Paramount P14 | blue  | 1800  | 200         |
| Fondriest U107    | blue  | 5565  | 2200        |
+-----+-----+-----+-----+
3 rows in set (0.00 sec)

mysql>
```

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):

```
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

```
C:\ Command Prompt (2) - mysql -u root -p
Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql> show databases;
+-----+
| Database |
+-----+
| bikedb   |
| mysql    |
| prog3    |
| test     |
| testdb   |
+-----+
5 rows in set (0.01 sec)

mysql> use mysql;
Database changed
mysql> show tables;
+-----+
| Tables_in_mysql |
+-----+
| columns_priv     |
| db               |
| func             |
| help_category    |
| help_keyword     |
| help_relation    |
| help_topic       |
| host             |
| tables_priv      |
| time_zone        |
| time_zone_leap_second |
| time_zone_name   |
| time_zone_transition |
| time_zone_transition_type |
| user             |
| user_info        |
+-----+
16 rows in set (0.00 sec)

mysql>
```

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

outt; - Notepad

File Edit Format View Help

```
mysql> use mysql;
Database changed
mysql> describe user;
```

Field	Type	Null	Key	Default	Extra
Host	varchar(60)		PRI		
User	varchar(16)		PRI		
Password	varchar(41)				
Select_priv	enum('N','Y')			N	
Insert_priv	enum('N','Y')			N	
Update_priv	enum('N','Y')			N	
Delete_priv	enum('N','Y')			N	
Create_priv	enum('N','Y')			N	
Drop_priv	enum('N','Y')			N	
Reload_priv	enum('N','Y')			N	
Shutdown_priv	enum('N','Y')			N	
Process_priv	enum('N','Y')			N	
File_priv	enum('N','Y')			N	
Grant_priv	enum('N','Y')			N	
References_priv	enum('N','Y')			N	
Index_priv	enum('N','Y')			N	
Alter_priv	enum('N','Y')			N	
Show_db_priv	enum('N','Y')			N	
Super_priv	enum('N','Y')			N	
Create_tmp_table_priv	enum('N','Y')			N	
Lock_tables_priv	enum('N','Y')			N	
Execute_priv	enum('N','Y')			N	
Repl_slave_priv	enum('N','Y')			N	
Repl_client_priv	enum('N','Y')			N	
ssl_type	enum('', 'ANY', 'X509', 'SPECIFIED')				
ssl_cipher	blob				
x509_issuer	blob				
x509_subject	blob				
max_questions	int(11) unsigned			0	
max_updates	int(11) unsigned			0	
max_connections	int(11) unsigned			0	

31 rows in set (0.00 sec)



Contents of the user_info Table

```
C:\ Command Prompt (2) - mysql -u root -p
16 rows in set (0.00 sec)
mysql> describe user_info;
+-----+-----+-----+-----+-----+-----+
| Field          | Type          | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| User           | varchar(16)   |      | PRI |          |       |
| Full_name      | varchar(60)   | YES  | MUL | NULL     |       |
| Description     | varchar(255)  | YES  |     | NULL     |       |
| Email          | varchar(80)   | YES  |     | NULL     |       |
| Contact_information | text         | YES  |     | NULL     |       |
| Icon           | blob          | YES  |     | NULL     |       |
+-----+-----+-----+-----+-----+-----+
6 rows in set (0.00 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','Alter') |
| Column_priv | set('Select','Insert','Update','References') |
+-----+-----+
8 rows in set (0.00 sec)
```

```
mysql> show tables;
+-----+
| Tables_in_test |
+-----+
| tables_priv     |
+-----+

mysql> describe tables_priv;
+-----+-----+-----+-----+-----+
| Field | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+
| Host  |      |     |          |       |
| Db    |      |     |          |       |
| User  |      |     |          |       |
| Table_name |      |     |          |       |
| Grantor |      |     |          |       |
| Timestamp |      |     |          |       |
| Table_priv |      |     |          |       |
| Column_priv |      |     |          |       |
+-----+-----+-----+-----+-----+
8 rows in set (0.00 sec)
```



Contents of the db Table

```
C:\ Command Prompt (2) - mysql -u root -p
6 rows in set (0.00 sec)
mysql> describe db;
+-----+-----+-----+-----+-----+-----+
| Field                | Type          | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| Host                 | char(60)      |      | PRI |          |       |
| Db                   | char(64)      |      | PRI |          |       |
| User                 | char(16)      |      | PRI |          |       |
| Select_priv          | enum('N','Y') |      |      | Z        |       |
| Insert_priv          | enum('N','Y') |      |      | Z        |       |
| Update_priv          | enum('N','Y') |      |      | Z        |       |
| Delete_priv          | enum('N','Y') |      |      | Z        |       |
| Create_priv          | enum('N','Y') |      |      | Z        |       |
| Drop_priv            | enum('N','Y') |      |      | Z        |       |
| Grant_priv           | enum('N','Y') |      |      | Z        |       |
| References_priv      | enum('N','Y') |      |      | Z        |       |
| Index_priv           | enum('N','Y') |      |      | Z        |       |
| Alter_priv           | enum('N','Y') |      |      | Z        |       |
| Create_tmp_table_priv | enum('N','Y') |      |      | Z        |       |
| Lock_tables_priv     | enum('N','Y') |      |      | Z        |       |
+-----+-----+-----+-----+-----+-----+
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 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 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 ]
```



Some of the Privileges Assigned with GRANT

Privilege	Operations Permitted
ALL or ALL PRIVILEGES	All privileges except for GRANT
ALTER	Change a table definition using ALTER TABLE excluding the creation and dropping of indices.
CREATE	Create database or tables within a database.
CREATE TEMPORARY TABLES	Create temporary tables.
DELETE	Ability to perform deletions from tables. (Delete DML statements).
DROP	Ability to drop databases or tables.
INSERT	Ability to insert data into tables.
SHUTDOWN	Ability to shutdown the MySQL server.

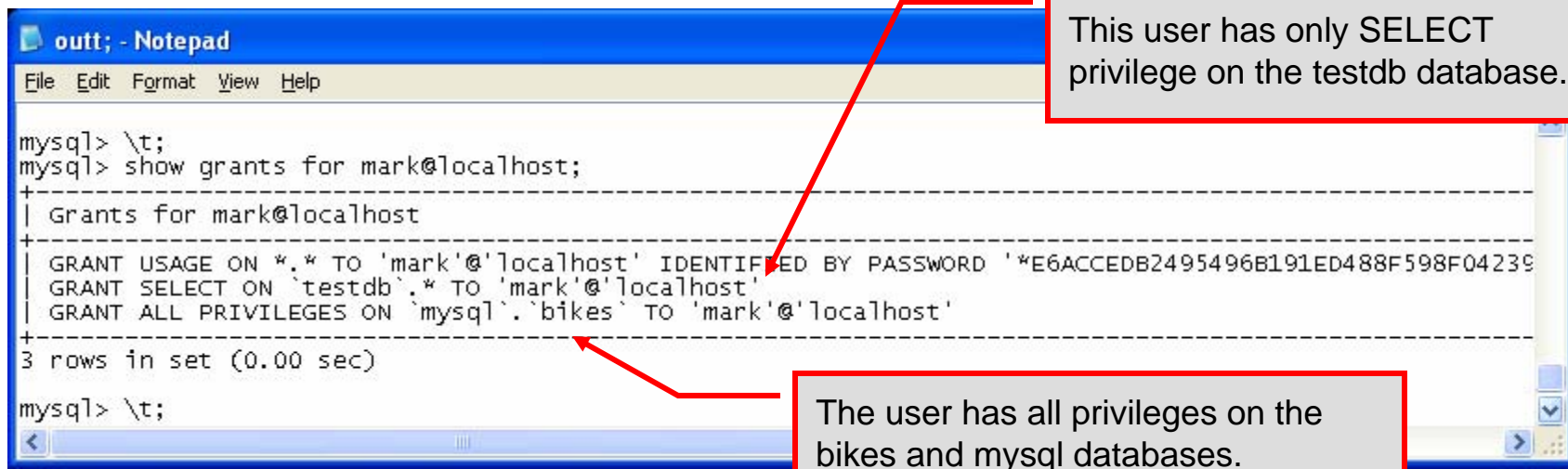


Displaying Privileges with SHOW

- The SQL command SHOW is used to display the grant privileges for a given user.
- The syntax for the SHOW command is:

SHOW GRANTS FOR *username@hostname*

- An example is shown below:



```
outt; - Notepad
File Edit Format View Help

mysql> \t;
mysql> show grants for mark@localhost;
+-----+
| Grants for mark@localhost |
+-----+
| GRANT USAGE ON *.* TO 'mark'@'localhost' IDENTIFIED BY PASSWORD '*E6ACCEDB2495496B191ED488F598F04239' |
| GRANT SELECT ON `testdb`.* TO 'mark'@'localhost' |
| GRANT ALL PRIVILEGES ON `mysql`.* `bikes` TO 'mark'@'localhost' |
+-----+
3 rows in set (0.00 sec)

mysql> \t;
```

This user has only 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

```
outt; - Notepad
File Edit Format View Help
mysql> show grants for mark@localhost;
+-----+
| Grants for mark@localhost |
+-----+
| GRANT USAGE ON *.* TO 'mark'@'localhost' IDENTIFIED BY PASSWORD '*E6ACCEDB2495496B191ED488F598F04239C85E73' |
| 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 '*E6ACCEDB2495496B191ED488F598F04239C85E73' |
| GRANT ALL PRIVILEGES ON `mysql`.`bikes` TO 'mark'@'localhost' |
+-----+
2 rows in set (0.00 sec)

mysql> \t;
```

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.

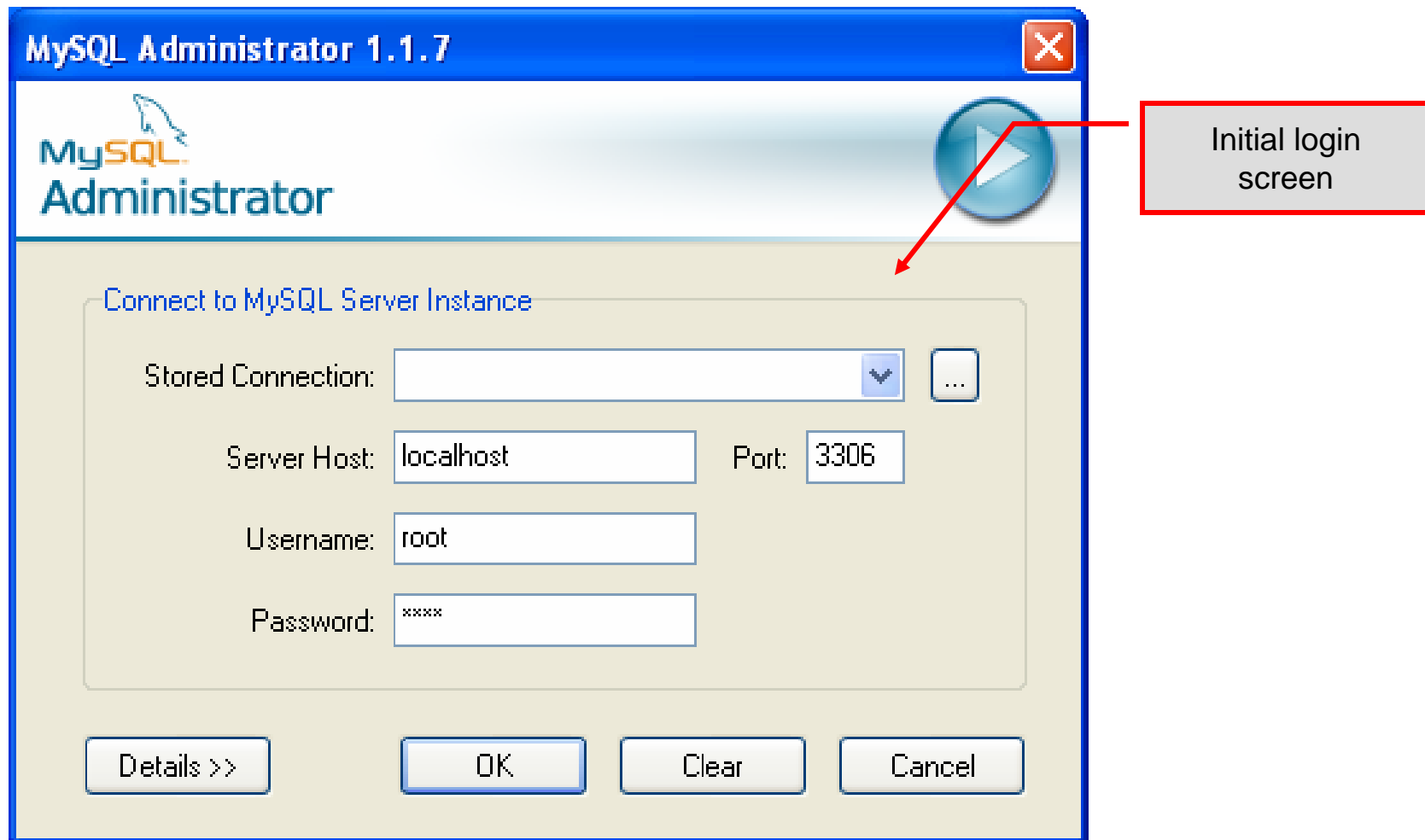


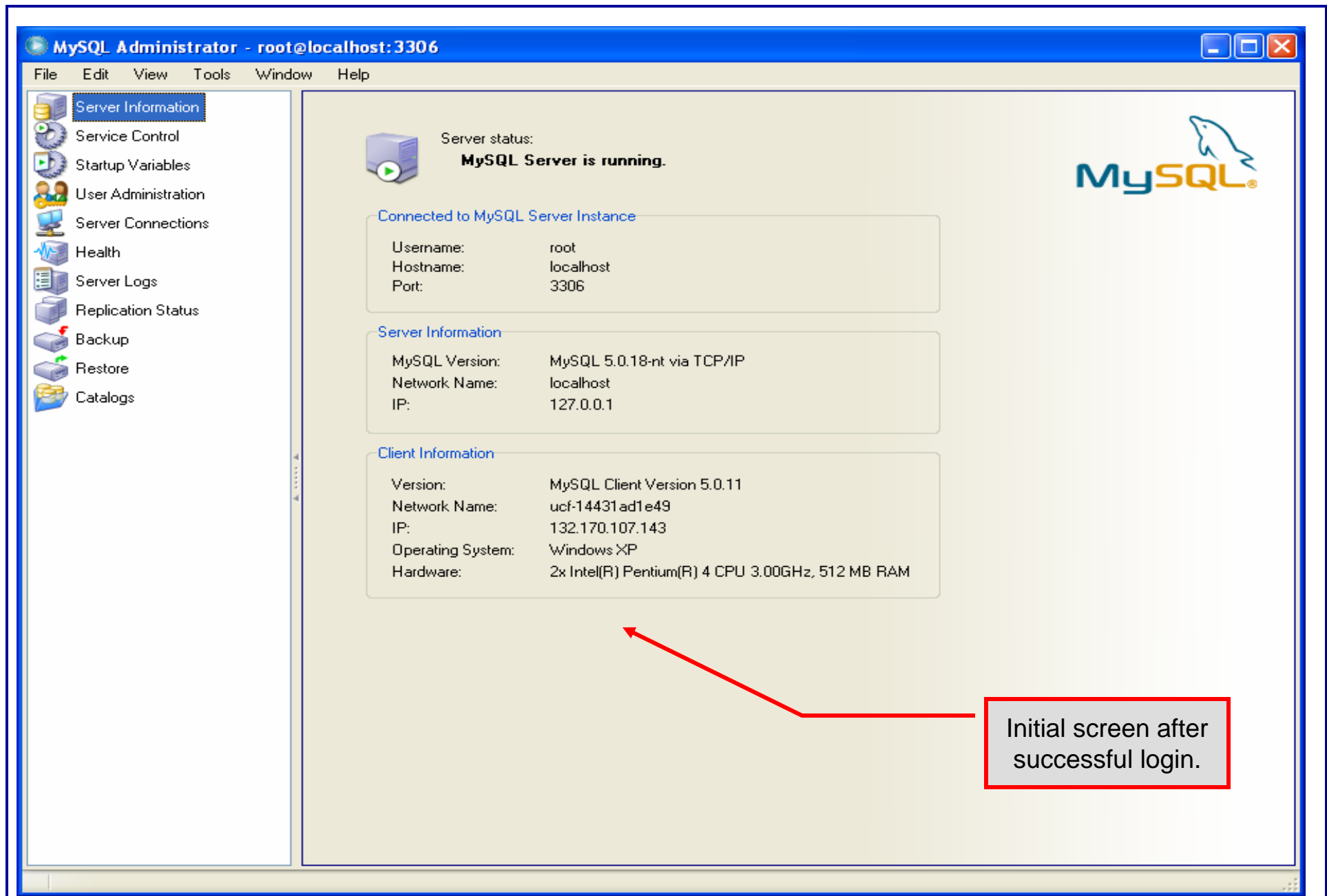
The MySQL Administrator Tool

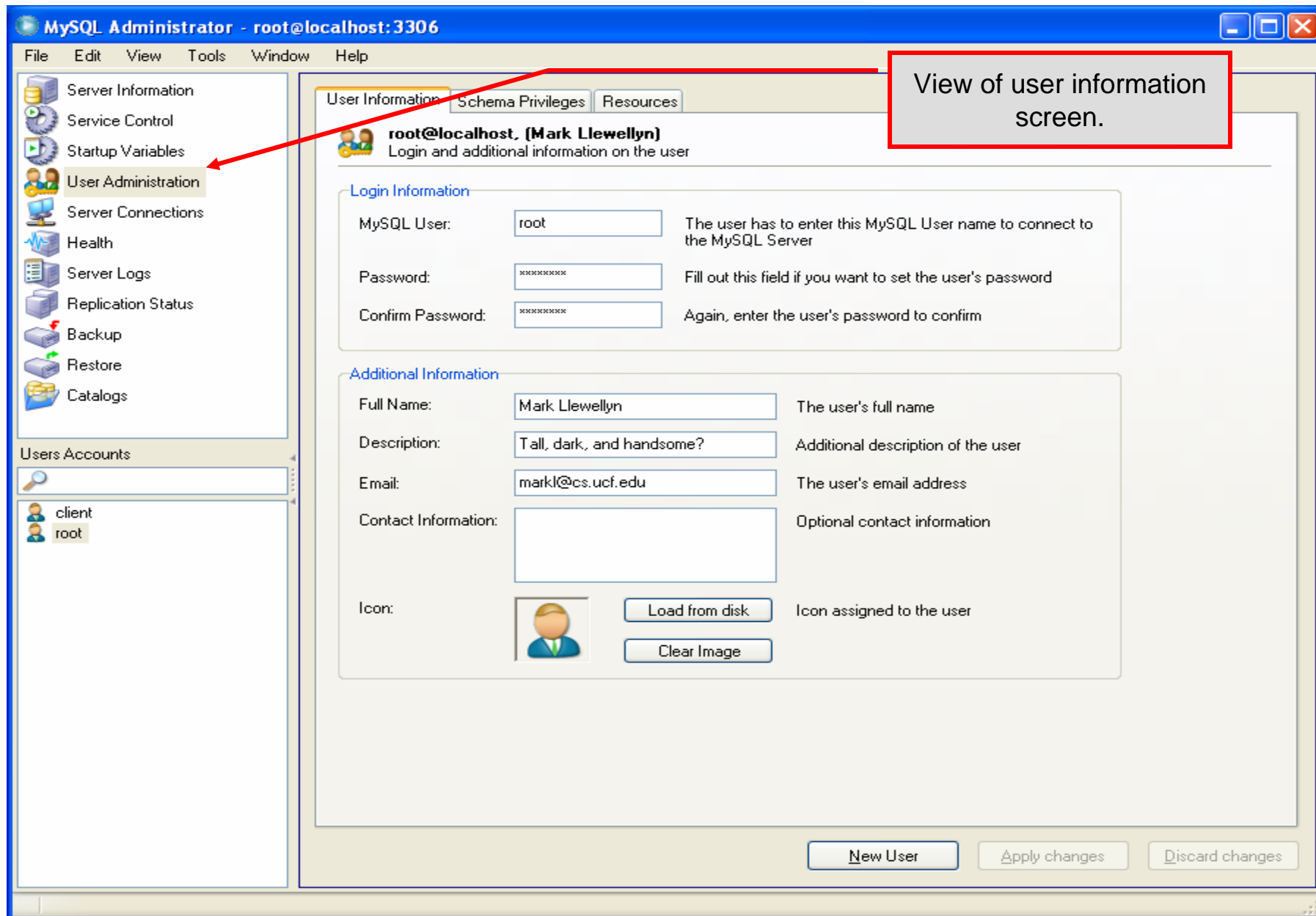
- From MySQL you can download a GUI-based administrator tool to help you administer your MySQL databases.
- This tool implements all of the GRANT, REVOKE, and SHOW functionality available in SQL.
- This tool also contains some system administrator functionality for monitoring system resources and utilization.
- 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.

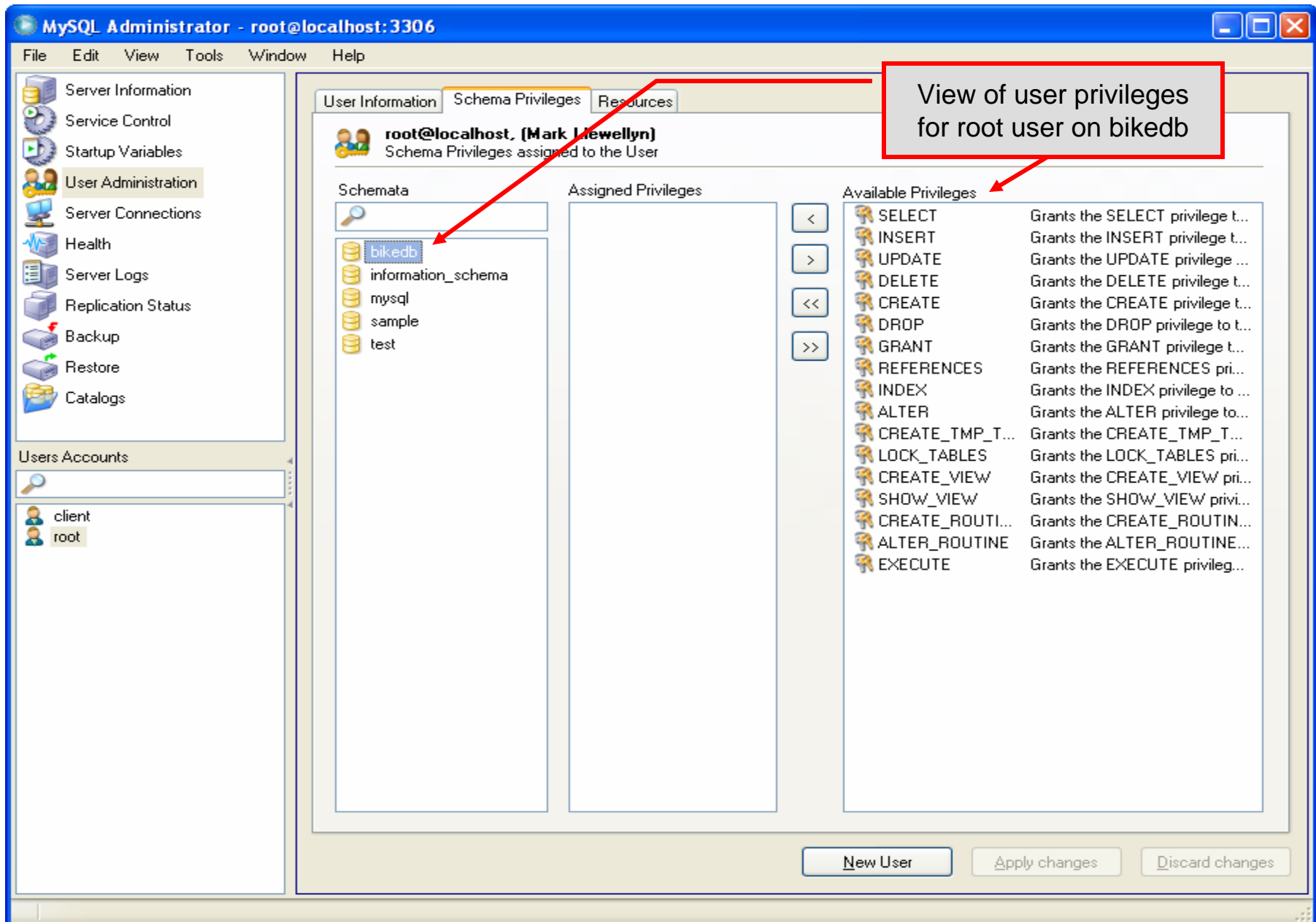


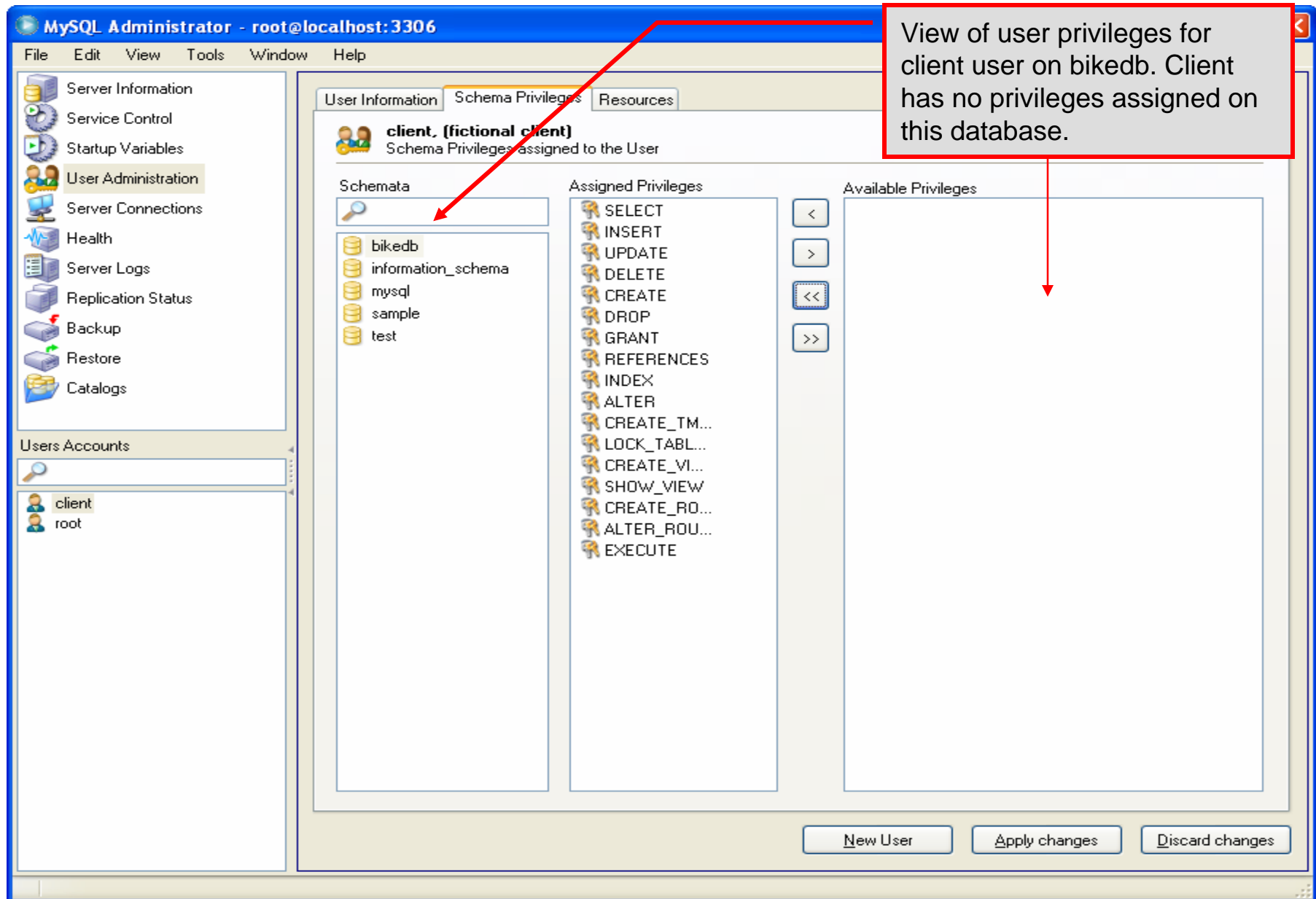
The MySQL Administrator Tool – Screen Shots











MySQL Administrator - root@localhost:3306

File Edit View Tools Help

Server Information
Service Control
Startup Variables
User Administration
Server Connections
Health
Server Logs
Replication Status
Backup
Restore
Catalogs

Users Accounts

mark
root

User Information Schema Privileges Resources

mark@localhost
Schema Privileges assigned to the User

Schemata

- bikedb
- mysql
- prog3
- test
- testdb

Assigned Privileges

- SELECT
- INSERT
- CREATE_TM...

Available Privileges

- UPDATE Grants the UPDATE privilege to the user
- DELETE Grants the DELETE privilege to the user
- CREATE Grants the CREATE privilege to the user
- DROP Grants the DROP privilege to the user
- GRANT Grants the GRANT privilege to the user
- REFERENCES Grants the REFERENCES privilege to the u...
- INDEX Grants the INDEX privilege to the user
- ALTER Grants the ALTER privilege to the user
- LOCK_TABLES Grants the LOCK_TABLES privilege to the ...

New User Apply changes Discard changes

Select a user and a database to grant or revoke privileges.



MySQL Administrator - root@localhost:3306

File Edit View Tools Window Help

Server Information
Service Control
Startup Variables
User Administration
Server Connections
Health
Server Logs
Replication Status
Backup
Restore
Catalogs

Schemata

bikedb
information_schema
mysql
sample
test

Schema Tables Schema Indices Views Stored procedures

bikedb
All tables of the bikedb schema

Table Name	Engine	Rows	Data length	Index length	Update time
bikes	InnoDB	11	16 kB	0 B	
bluebikes	InnoDB	2	16 kB	0 B	

Num. of Tables: 2 Rows: 13 Data Len: 32 kB Index Len: 0 B

Details >> Create Table Edit Table Maintenance Refresh

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.



MySQL Query Browser - root@localhost:3306 / bikedb

File Edit View Query Script Tools Window Help

Go back Next Refresh

SELECT * FROM bikes b

Execute Stop

Resultset 1

bikename	size	color	cost	purchased	mileage
Battaglin Carrera	60	red/white	4000	2001-03-14	11200
Bianchi Corse Evo 4	58	celeste	5700	2004-12-22	300
Bianchi Evolution 3	58	celeste	4800	2003-11-16	2000
Bianchi/Liquigas FG	58	celeste/blue	5600	2005-12-02	0
Colnago Dream Rabobank	60	blue/orange	5500	2002-07-27	4300
Colnago Superissimo	59	red	3800	1996-03-01	13000
Eddy Merckx Domo	58	blue/black	5300	2005-02-02	0
Eddy Merckx Molteni	58	orange	5100	2004-08-12	0
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

11 rows fetched in 0.0050s (0.0006s)

Edit Apply Changes Discard Changes First Last Search

Schemata Bookmarks History

bikedb

- bikes
- bluebikes
- information_schema
- mysql
- sample
- test

Syntax Functions Params Trx

- Data Manipulation
- Data Definition
- MySQL Utility
- Transactional and Locking

Result set shown for this query. Note that this query is based on a single table, so the result set is editable.



MySQL Query Browser - root@localhost:3306 / bikedb

File Edit View Query Script Tools Window Help

Go back Next Refresh

`SELECT * from bikes where cost > 4500`

Execute Stop

Resultset 1 Resultset 2 **Resultset 3 x**

bikename	size	color	cost	purchased	mileage
Bianchi Corse Evo 4	58	celeste	5700	2004-12-22	300
Bianchi Evolution 3	58	celeste	4800	2003-11-16	2000
Bianchi/Liquigas FG	58	celeste/blue	5600	2005-12-02	0
Colnago Dream Rabobank	60	blue/orange	5500	2002-07-27	4300
Eddy Merckx Domo	58	blue/black	5300	2005-02-02	0
Eddy Merckx Molteni	58	orange	5100	2004-08-12	0

Schemata Bookmarks History

bikedb

- bikes
- bluebikes
- information_schema
- mysql
- sample
- test

Syntax Functions Params Trx

- Data Manipulation
- Data Definition
- MySQL Utility
- Transactional and Locking

6 rows fetched in 0.0047s (0.0004s)

Edit Apply Changes Discard Changes First Last Search

1: 1

You can manage multiple result sets simultaneously. Statistics on query execution are always available.

