

Lecture #3

NEWM N510: Web-Database Concepts

MySQL (2)

kharrazi@iupui.edu
<http://www.info510.com>

Review Last Lecture

- Database Overview
- Relational Databases
- Installing MySQL
- Command line MySQL
- MySQL GUI Tools
- SQL Introduction
- SQL: SELECT Statement
- SQL: WHERE (BETWEEN/LIKE/LIMIT) Clause
- SQL: AND & OR
- SQL: IN
- SQL: ORDER BY Clause

Lecture in a Nutshell

1. SQL: CREATE (Database, Table, and Index)
2. SQL: TRUNCATE (Table)
3. SQL: DROP (Database, Table, and Index)
4. SQL: ALTER (Database, Table, and Index)
5. SQL: INSERT
6. SQL: UPDATE
7. SQL: DELETE
8. SQL: Joining and Keys (Inner/Left/Right Join)
9. SQL: GROUP BY & HAVING
10. SQL: Functions

1. SQL: **CREATE** (database, table)

- Syntax:

```
CREATE DATABASE database_name
```

```
CREATE TABLE table_name  
(  
column_name1 data_type,  
column_name2 data_type,  
.....  
)
```

- Examples:

```
CREATE DATABASE my_database_name
```

SQL: CREATE (cont.) (database, table)

- Examples:

```
CREATE TABLE `pat_info` (  
  `P_id` int(8) unsigned not null auto_increment  
    primary key unique,  
  `P_FirstName` varchar(100) default NULL,  
  `P_LastName` varchar(100) default NULL,  
  `City_id` int(11) default NULL,  
  `Phone_id` int(11) default NULL,  
  `Kin_id` int(11) default NULL,  
  `Client_id` int(11) default NULL  
) TYPE=MyISAM;
```

SQL: CREATE (cont.) (database, table)

Column (Data) types:

1. TEXT TYPES

CHAR()	a fixed section from 0 to 255 characters long.
VARCHAR()	a variable section from 0 to 255 characters long.
TINYTEXT	a string with a maximum length of 255 characters.
TEXT	a string with a maximum length of 65535 characters.
BLOB	a string with a maximum length of 65535 characters.
MEDIUMTEXT	a string with a maximum length of 16777215 characters.
MEDIUMBLOB	a string with a maximum length of 16777215 characters.
LONGTEXT	a string with a maximum length of 4294967295 characters.
LOB	a string with a maximum length of 4294967295 characters.

SQL: CREATE (cont.) (database, table)

Column (Data) types:

2. NUMBER TYPES

TINYINT()	-128 to 127 normal (0 to 255 UNSIGNED)
SMALLINT()	-32768 to 32767 normal (0 to 65535 UNSIGNED)
MEDIUMINT()	-8388608 to 8388607 normal (0 to 16777215 UNSIGNED)
INT()	-2147483648 to 2147483647 normal (0 to 4294967295 UNSIGNED)
BIGINT()	-9223372036854775808 to 9223372036854775807 normal (0 to 18446744073709551615 UNSIGNED)
FLOAT(,)	small number with a floating decimal point (approximate)
DOUBLE(,)	a large number with a floating decimal point (approximate)
DECIMAL(,)	a fixed decimal number stored in binary format (exact)

SQL: CREATE (cont.) (database, table)

Column (Data) types:

3. DATE TYPES

DATE	YYYY-MM-DD
TIME	HH:MM:SS
DATETIME	YYYY-MM-DD HH:MM:SS
TIMESTAMP	YYYYMMDDHHMMSS

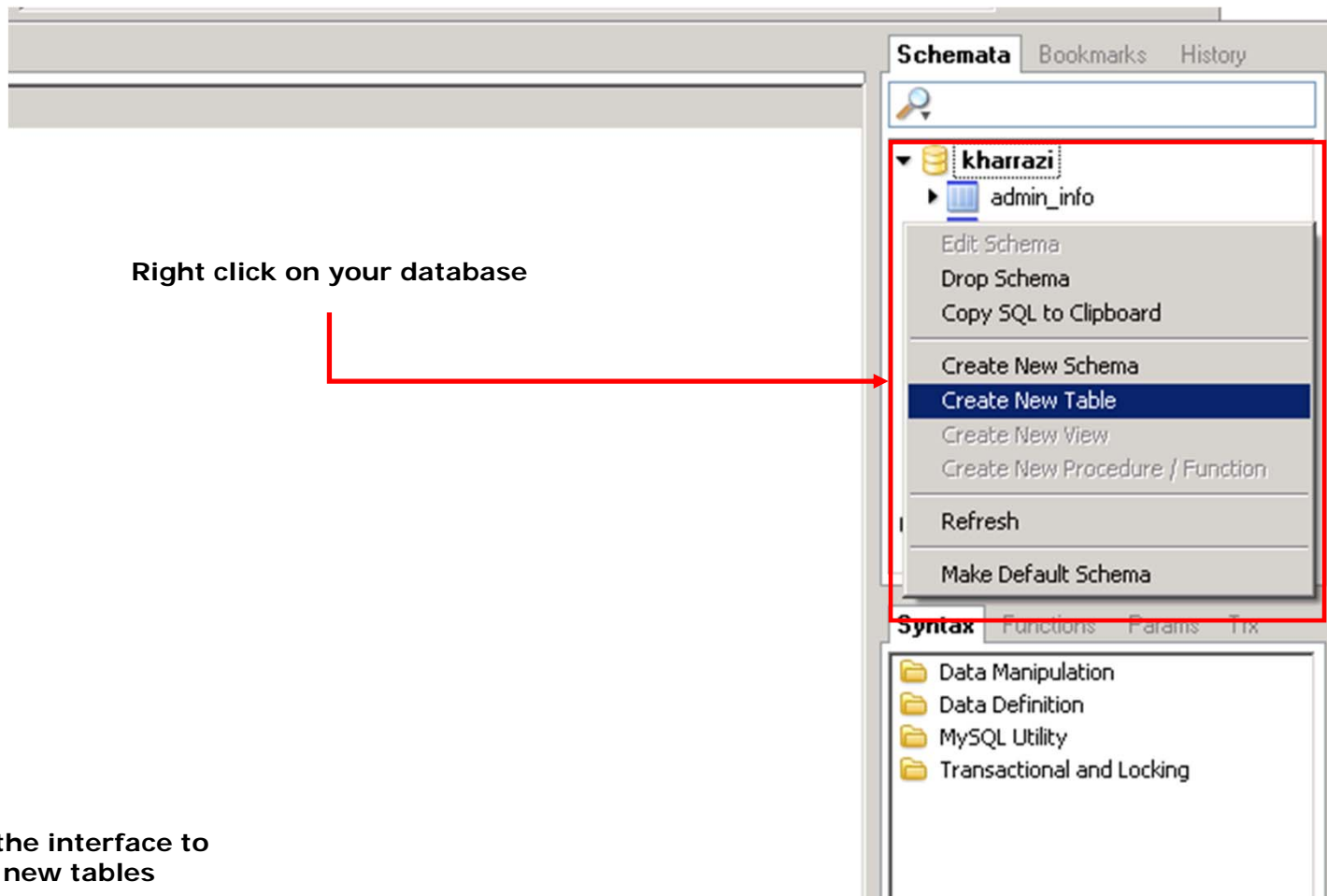
SQL: **CREATE** (cont.) (database, table)

The image shows a screenshot of a database management tool interface. On the left, the 'SQL Query Area' contains the following SQL code:

```
1 CREATE TABLE `pat_info` (  
2   `P_id` int(8) unsigned not null auto_increment primary key unique,  
3   `P_FirstName` varchar(100) default NULL,  
4   `P_LastName` varchar(100) default NULL,  
5   `City_id` int(11) default NULL,  
6   `Phone_id` int(11) default NULL,  
7   `Kin_id` int(11) default NULL,  
8   `Client_id` int(11) default NULL  
9 ) TYPE=MyISAM;
```

On the right, a tree view shows the database structure for 'kharrazi'. The table 'pat_info' is highlighted with a red box. A red arrow points from this box to a message box below. The message box contains the text: 'Table 'pat_info' already exists'.

SQL: **CREATE** (cont.) (database, table)



SQL: CREATE (cont.) (database, table)

Table's Name

MySQL Table Editor

Table Name: Database: kharrazi Comment:

Columns and Indices | Table Options | Advanced Options

Column Name	Datatype	NOT NULL	AUTO INC	Flags	Default Value	Comment
Column Name	Data type				Default value	

Indices | Foreign Keys | Column Details

Index Settings:

Index Name:

Index Kind: INDEX

Index Type: DEFAULT

Index Columns (Use Drag'n'Drop)

Apply Changes | Discard Changes | Close

SQL: **CREATE** (cont.) (database, table)

The screenshot shows the MySQL Table Editor interface for a table named 'test' in the 'kharrazi' database. The 'Columns and Indices' tab is active, showing a table with one column, 'id', which is a primary key. The column's datatype is TINYINT, and it has the UNSIGNED flag checked. The 'Index Settings' section shows a PRIMARY index named 'PRIMARY' on the 'id' column.

Column Name	Datatype	NOT NULL	AUTO INC	Flags	Default Value	Comment
id	TINYINT	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/> UNSIGNED <input type="checkbox"/> ZEROFILL	NULL	

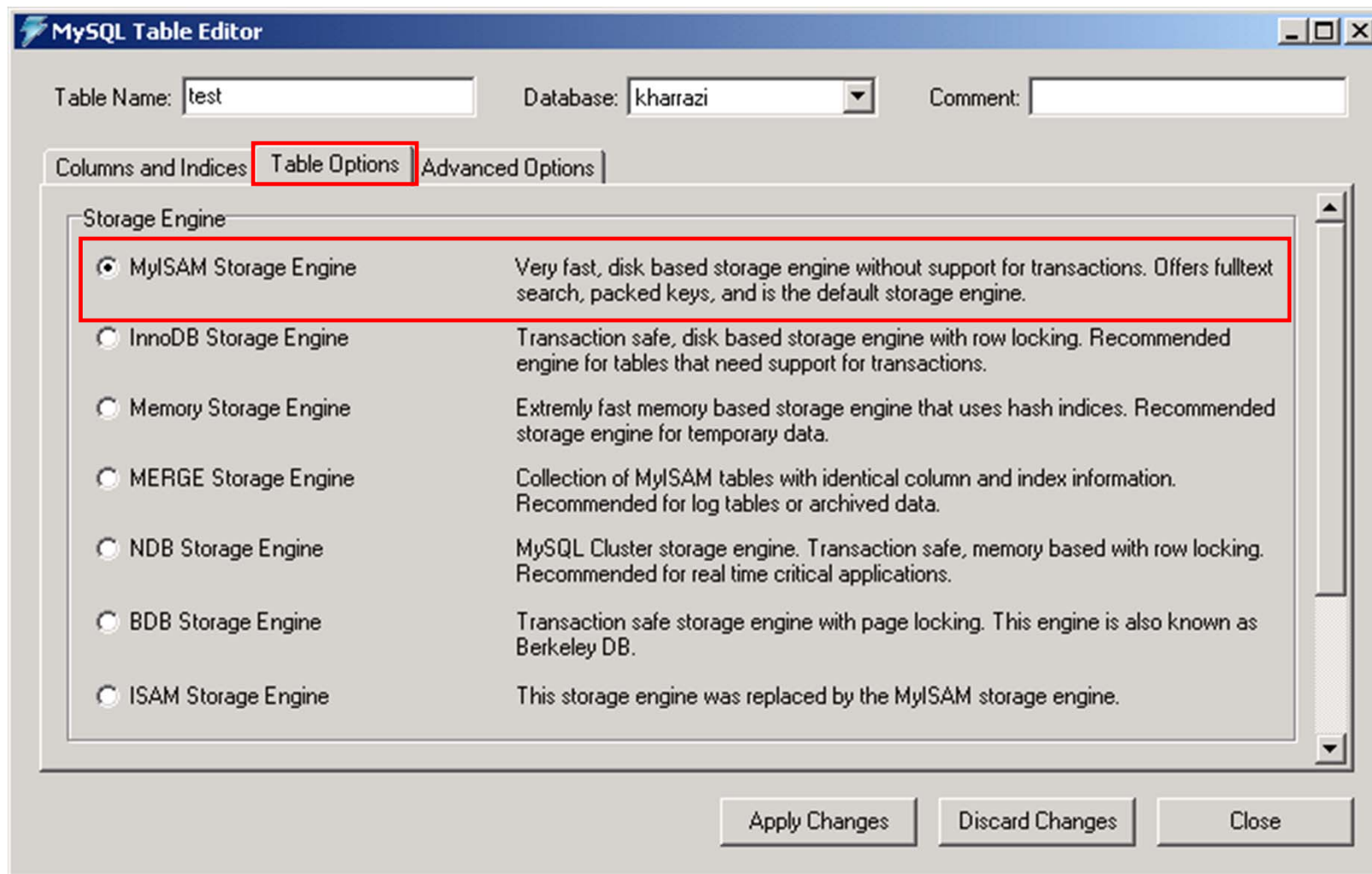
Annotations for the 'id' column:

- Primary Key**: Points to the key icon next to the column name.
- No Negative Numbering**: Points to the **UNSIGNED** flag.
- Add zero before small numbers**: Points to the **ZEROFILL** flag.
- Default value if empty**: Points to the **NULL** default value.

The 'Index Settings' section shows:

- Index Name: PRIMARY
- Index Kind: PRIMARY
- Index Type: DEFAULT
- Index Columns: id

Buttons at the bottom: Apply Changes, Discard Changes, Close.

SQL: CREATE (cont.) (database, table)

SQL: CREATE (cont.) (database, table)

MySQL Table Editor

Table Name: Database: Comment:

Columns and Indices | Table Options | Advanced Options

Column Name	Datatype	NOT NULL	AUTO INC	Flags	Default Value	Comment
id	TINYINT	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/> UNSIGNED <input type="checkbox"/> ZEROFILL	NULL	

Indices | Foreign Keys | Column Details

PRIMARY

Index Settings

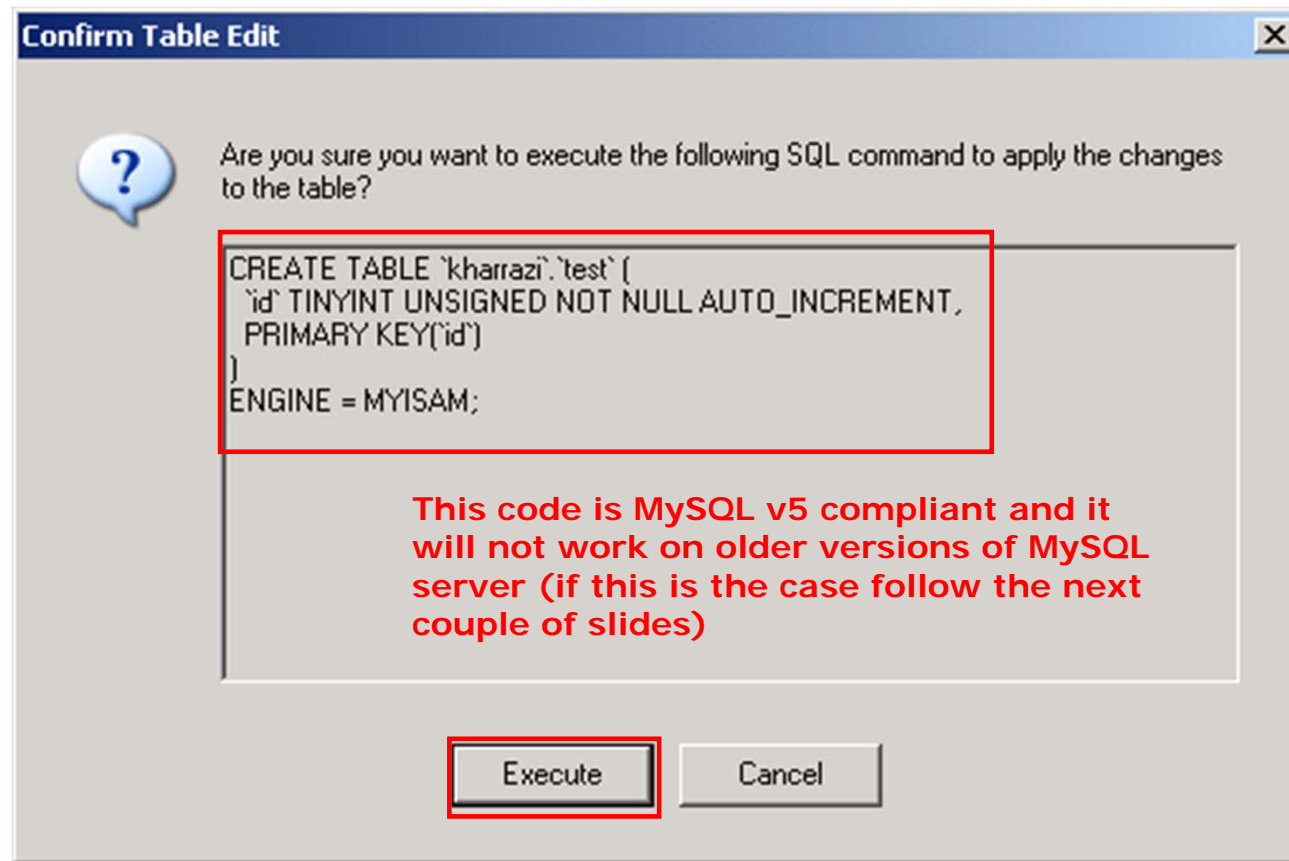
Index Name:

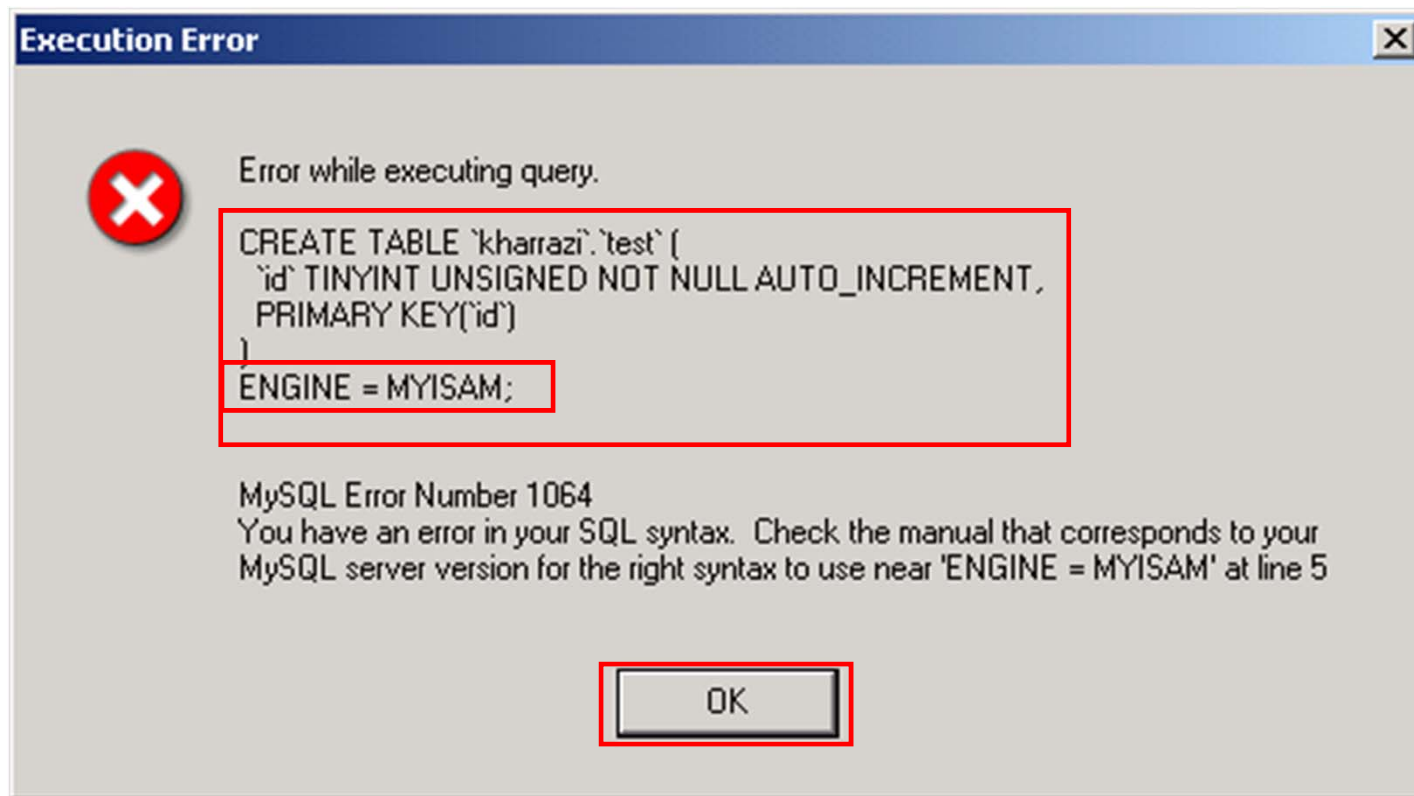
Index Kind:

Index Type:

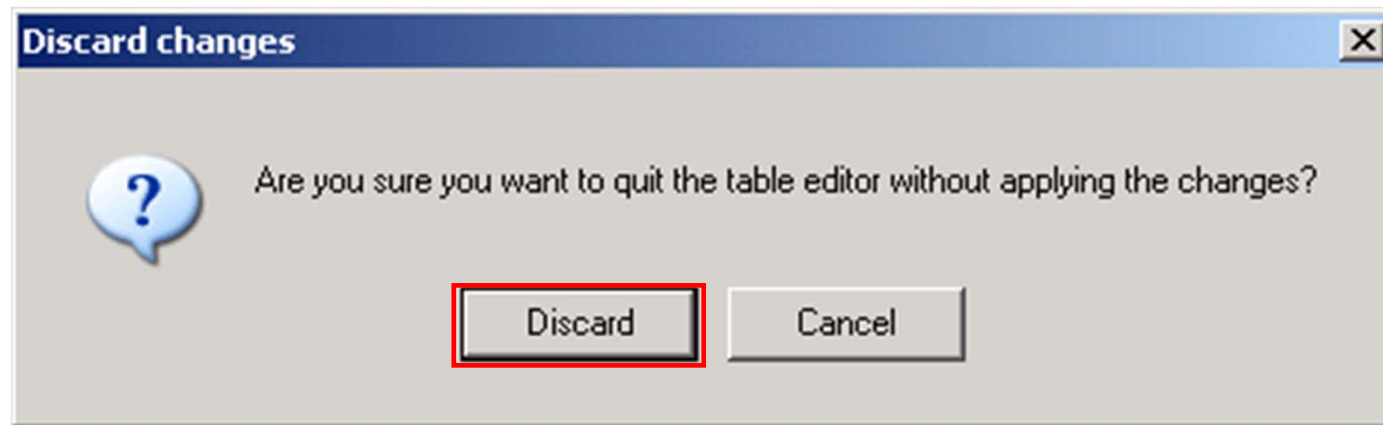
Index Columns (Use Drag'n'Drop)

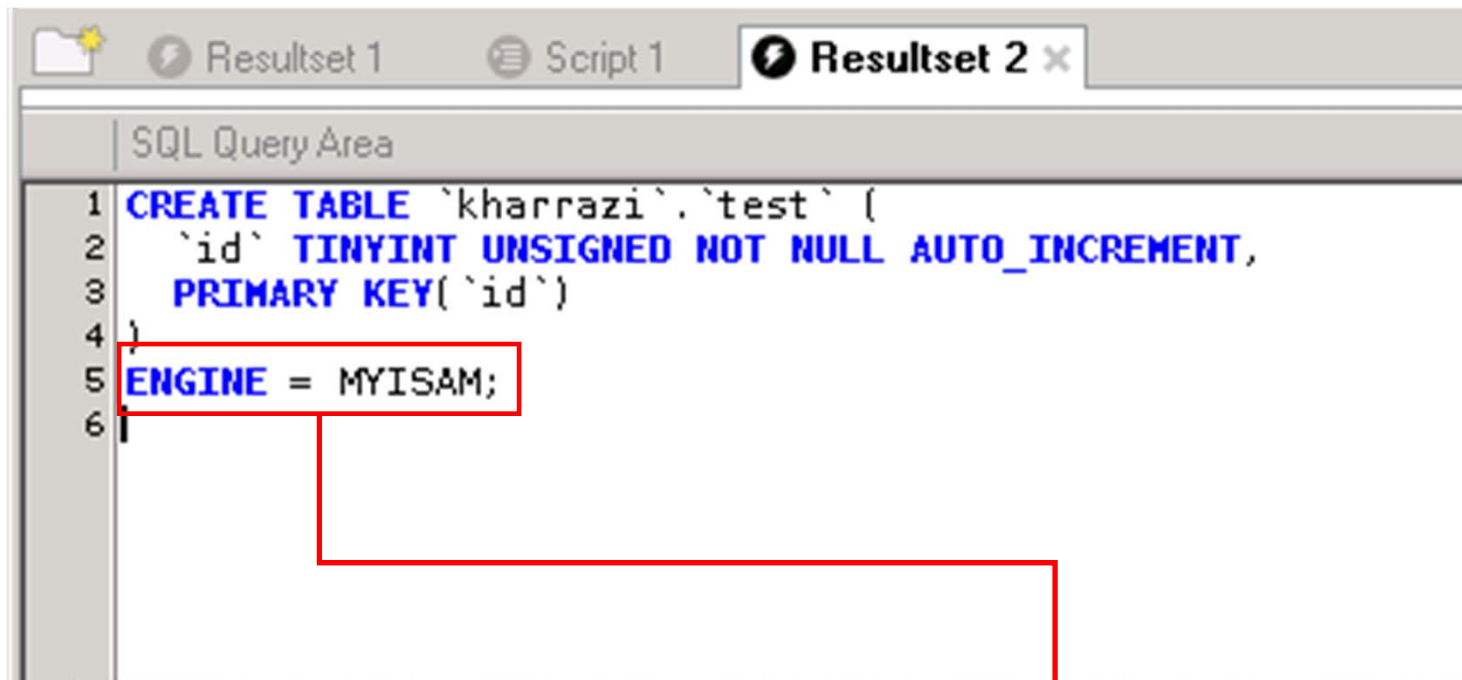
id

SQL: CREATE (cont.) (database, table)

SQL: *CREATE* (cont.) (database, table)

SQL: *CREATE* (cont.) (database, table)

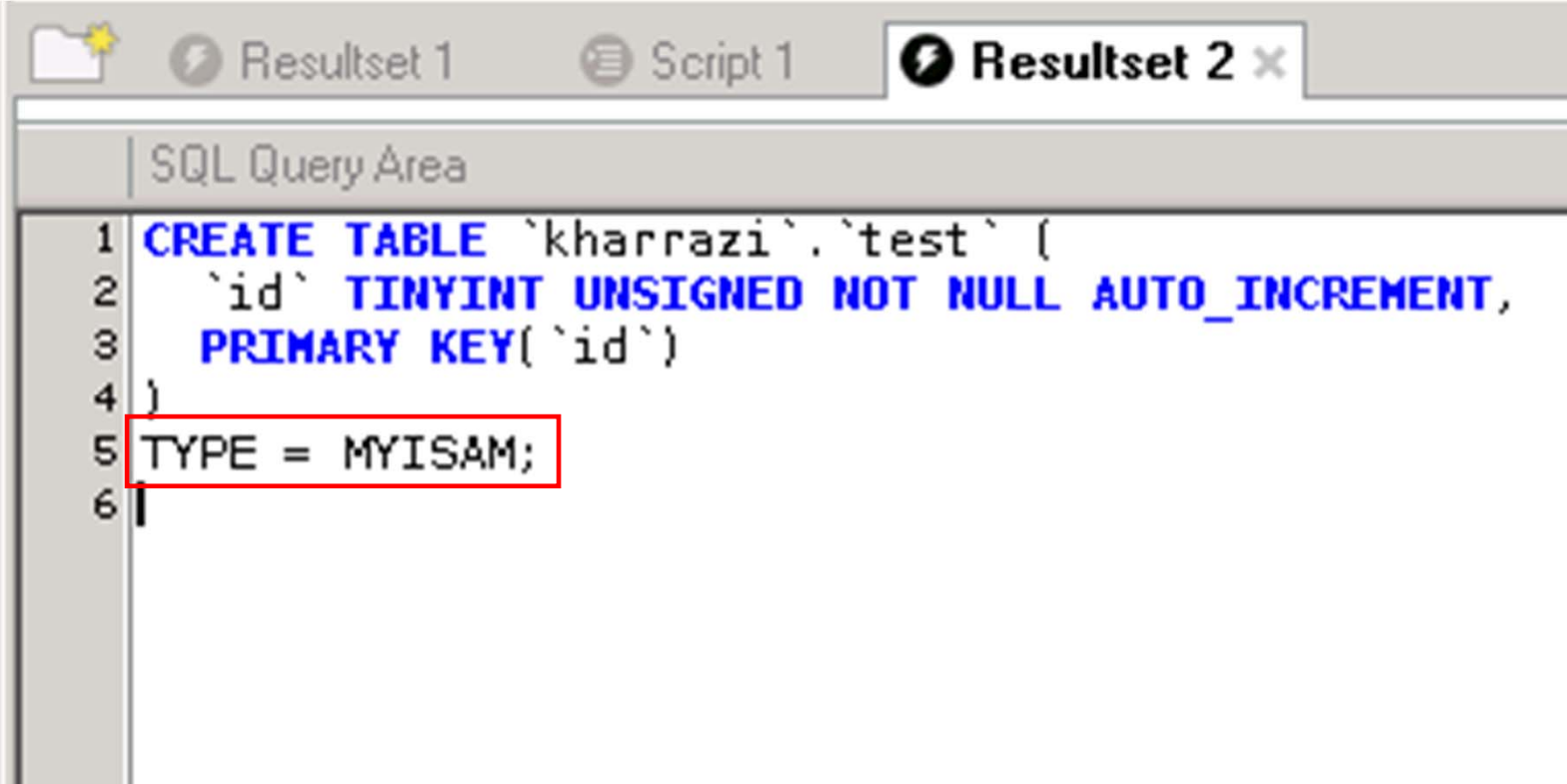


SQL: CREATE (cont.) (database, table)

```
1 CREATE TABLE `kharrazi`.`test` (  
2   `id` TINYINT UNSIGNED NOT NULL AUTO_INCREMENT,  
3   PRIMARY KEY(`id`)  
4 )  
5 ENGINE = MYISAM;  
6
```

Copy and paste the generated code
and then change the 'ENGINE' to
'TYPE'

SQL: *CREATE* (cont.) (database, table)

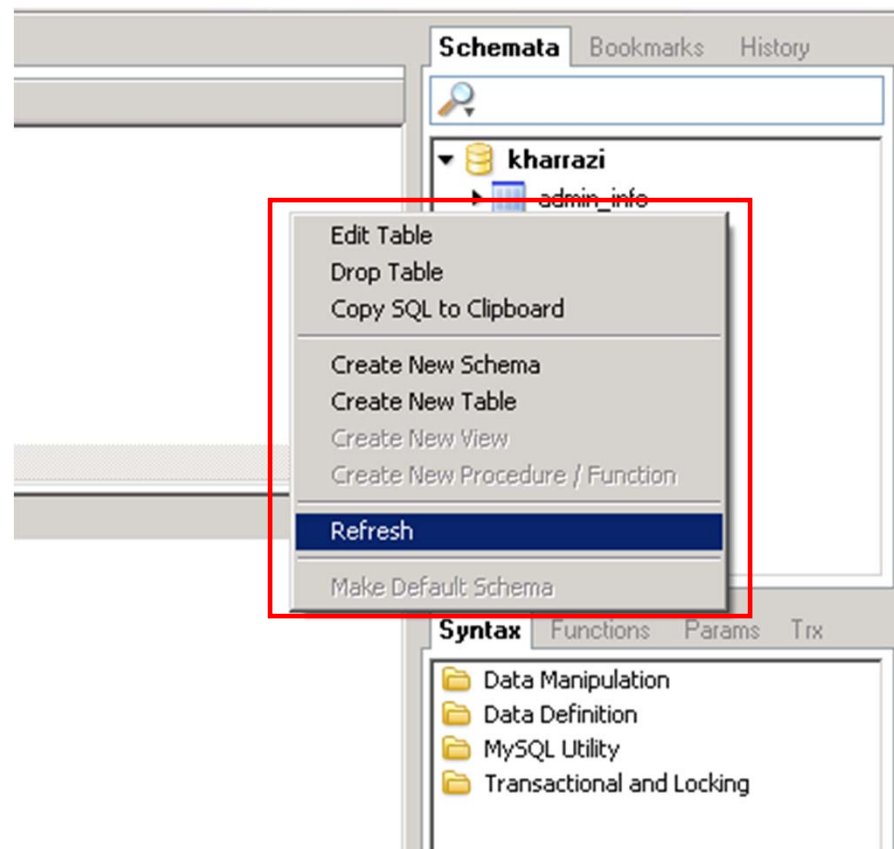


The screenshot shows a SQL query editor window with three tabs: 'Resultset 1', 'Script 1', and 'Resultset 2'. The 'Script 1' tab is active, and the 'SQL Query Area' contains the following SQL code:

```
1 CREATE TABLE `kharrazi`.`test` (  
2   `id` TINYINT UNSIGNED NOT NULL AUTO_INCREMENT,  
3   PRIMARY KEY(`id`)  
4 )  
5 TYPE = MYISAM;  
6
```

The line 'TYPE = MYISAM;' is highlighted with a red rectangular box.

SQL: **CREATE** (cont.) (database, table)



SQL: **CREATE** (cont.) (database, table)

The screenshot displays a MySQL IDE interface. The main window, titled "SQL Query Area", contains the following SQL code:

```
1 CREATE TABLE `kharrazi`.`test` (  
2   `id` TINYINT UNSIGNED NOT NULL AUTO_INCREMENT,  
3   PRIMARY KEY(`id`)  
4 )  
5 TYPE = MYISAM;  
6
```

Below the query area, a "Resultset 2" tab is active, showing an empty result set. To the right, the "Schemata" panel shows the database structure for "kharrazi". The tables listed are: admin_info, city_info, client_info, doc_info, kin_info, lab_info, pat_doc_relate, pat_info, phone_info, test, and test_holding. The "test" table is highlighted with a red rectangle. Below the Schemata panel, the "Syntax" panel is visible, showing categories like Data Manipulation, Data Definition, MySQL Utility, and Transactional and Locking.

SQL: **CREATE** (cont.) (database, table)

The screenshot displays a database management tool interface. The main window is titled "SQL Query Area" and contains the following SQL query:

```
1 SELECT * FROM `kharrazi`.`test`
```

Below the query area, a table structure is visible, with a column named "id" highlighted by a red box. The table structure is as follows:

id

The right-hand side of the interface shows a "Schemata" panel with a tree view of the database structure. The "kharrazi" database is expanded, showing the following tables:

- admin_info
- city_info
- client_info
- doc_info
- kin_info
- lab_info
- pat_doc_relate
- pat_info
- phone_info
- test

Below the "Schemata" panel, there is a "Syntax" panel with the following categories:

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

2. SQL: **TRUNCATE** (table)

- Syntax:

```
TRUNCATE table_name
```

- Examples:

```
TRUNCATE test
```

3. SQL: **DROP** (database, table)

- Syntax:

```
DROP DATABASE database_name  
DROP TABLE table_name
```

- Examples:

```
DROP TABLE test
```


4. SQL: **ALTER** (table)

- Syntax:

```
ALTER TABLE table_name  
ADD column_name datatype
```

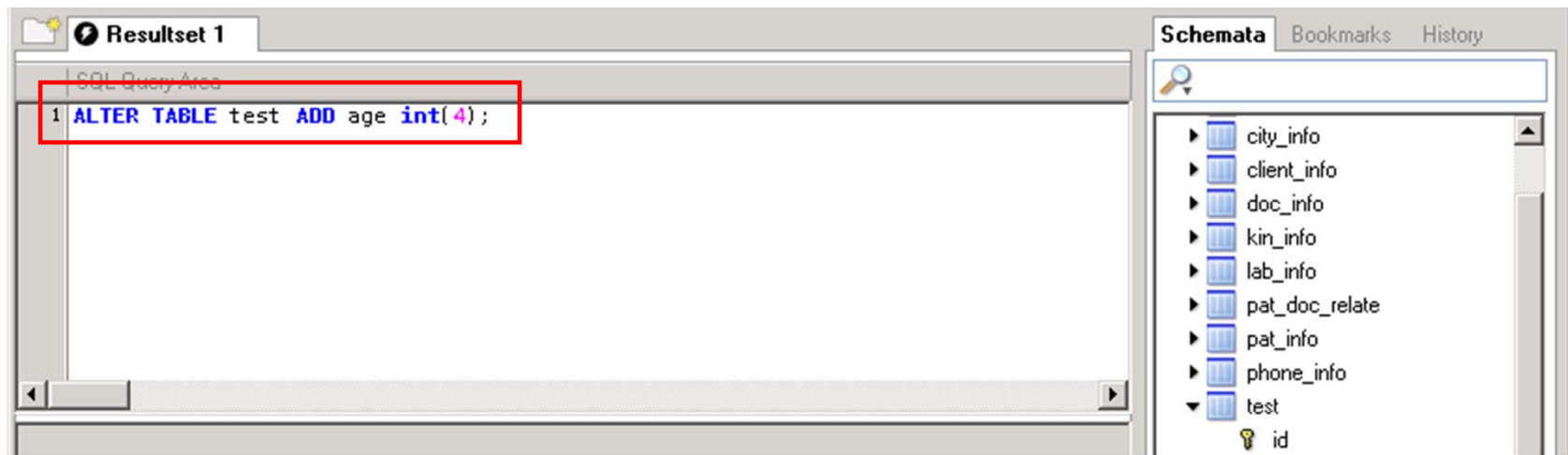
```
ALTER TABLE table_name  
DROP COLUMN column_name
```

- Examples:

```
ALTER TABLE test ADD age int(4);
```

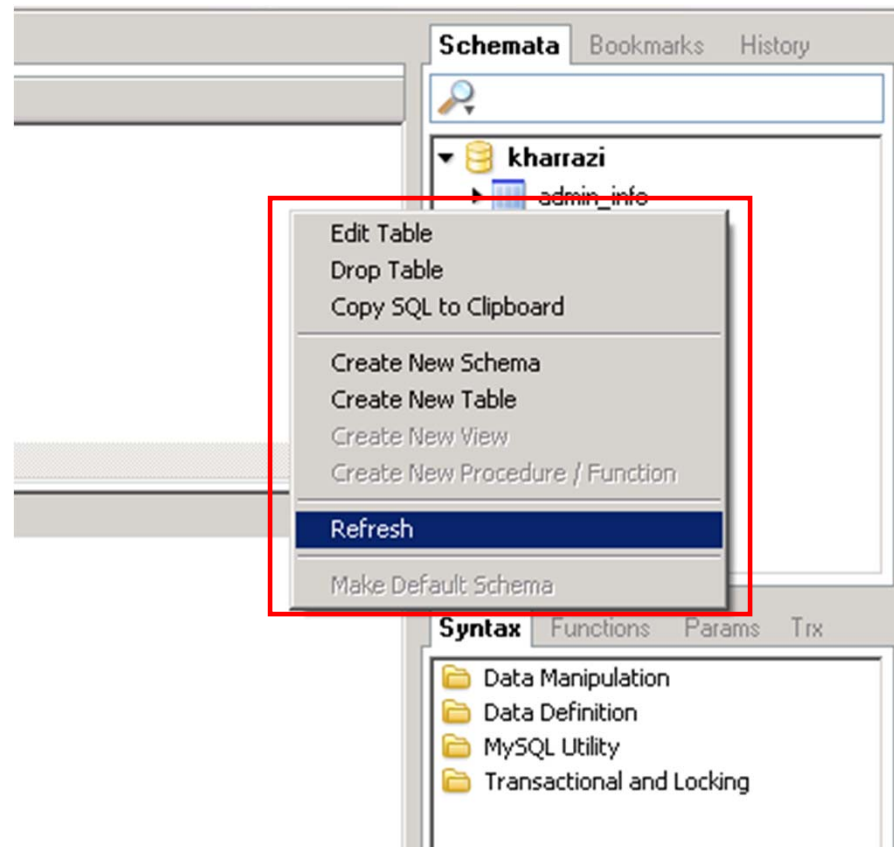
```
ALTER TABLE test DROP COLUMN age;
```

SQL: **ALTER** (cont.) (table)

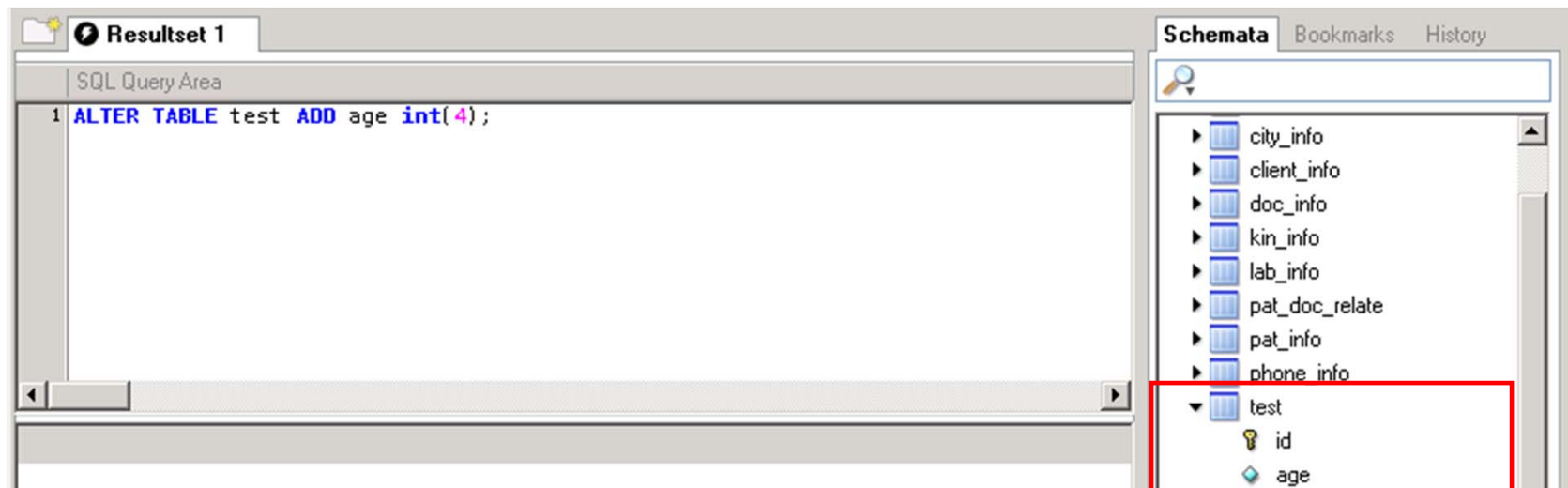


Adding a new column

SQL: **ALTER** (cont.) (table)

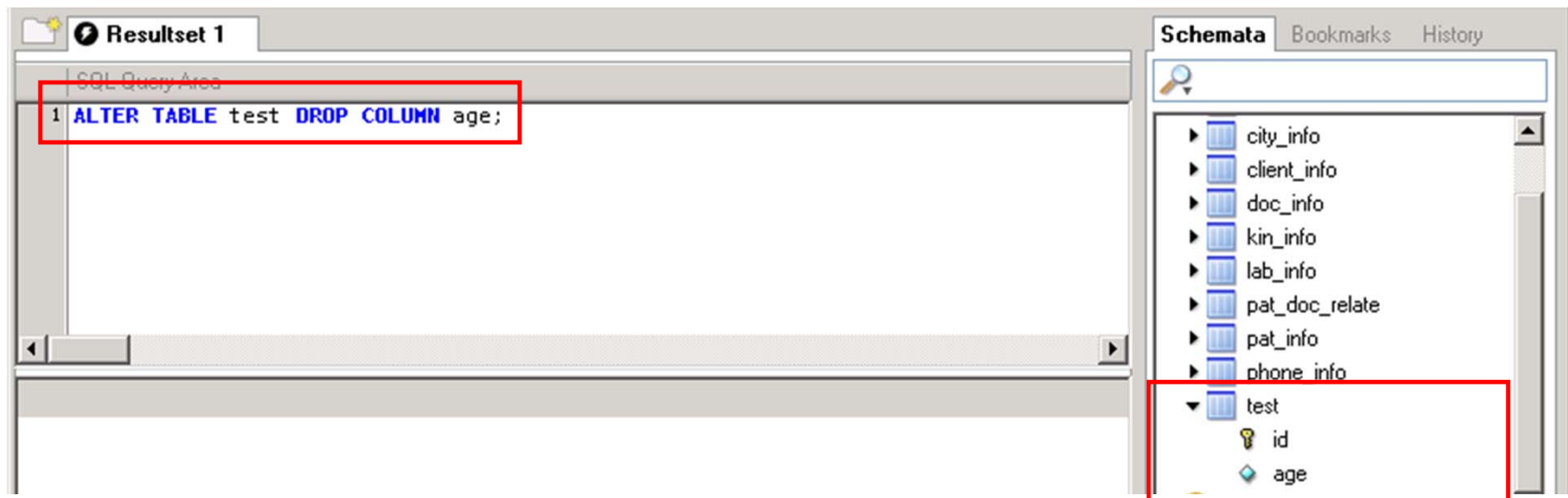


SQL: **ALTER** (cont.) (table)



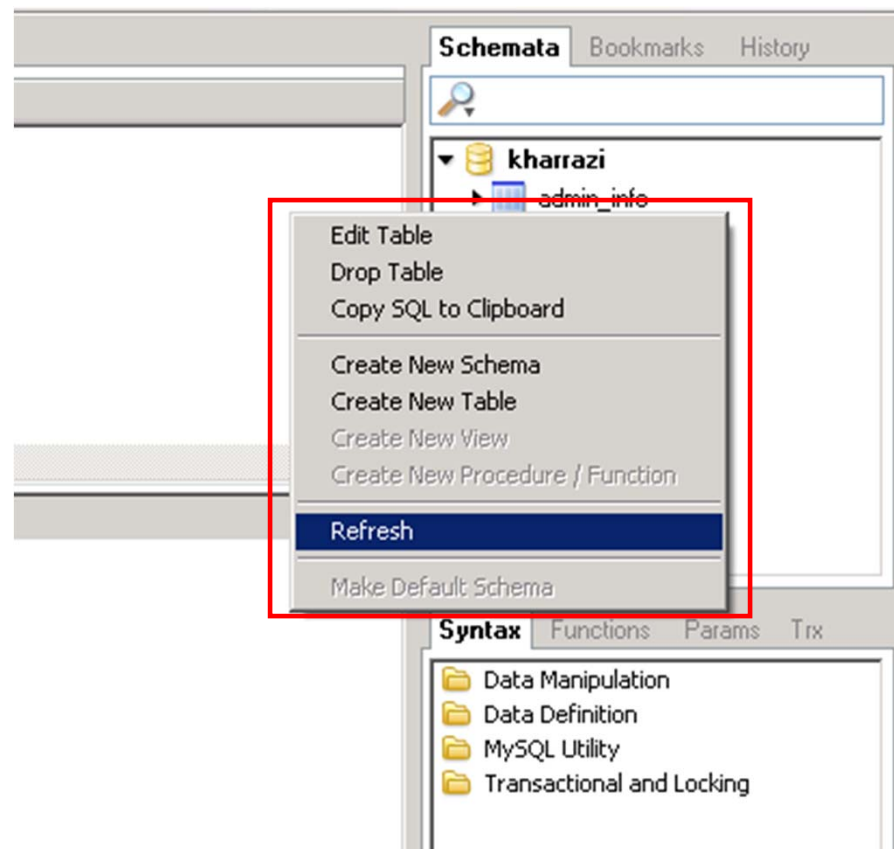
A new column is added to the table

SQL: **ALTER** (cont.) (table)

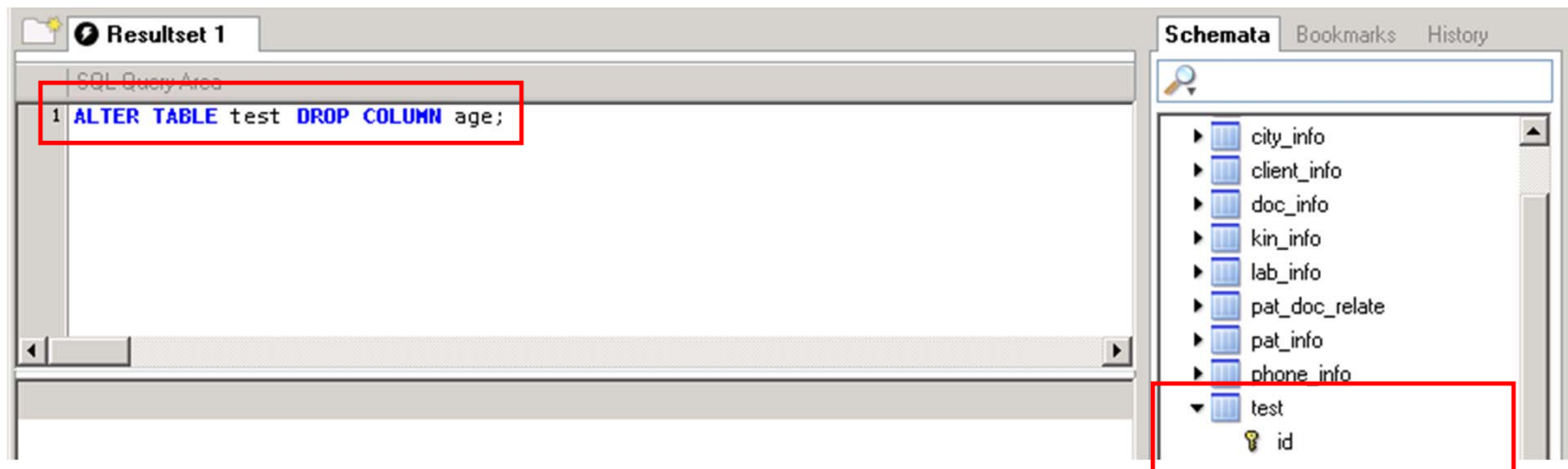


Deleting a column

SQL: **ALTER** (cont.) (table)



SQL: **ALTER** (cont.) (table)



5. SQL: **INSERT INTO** Clause

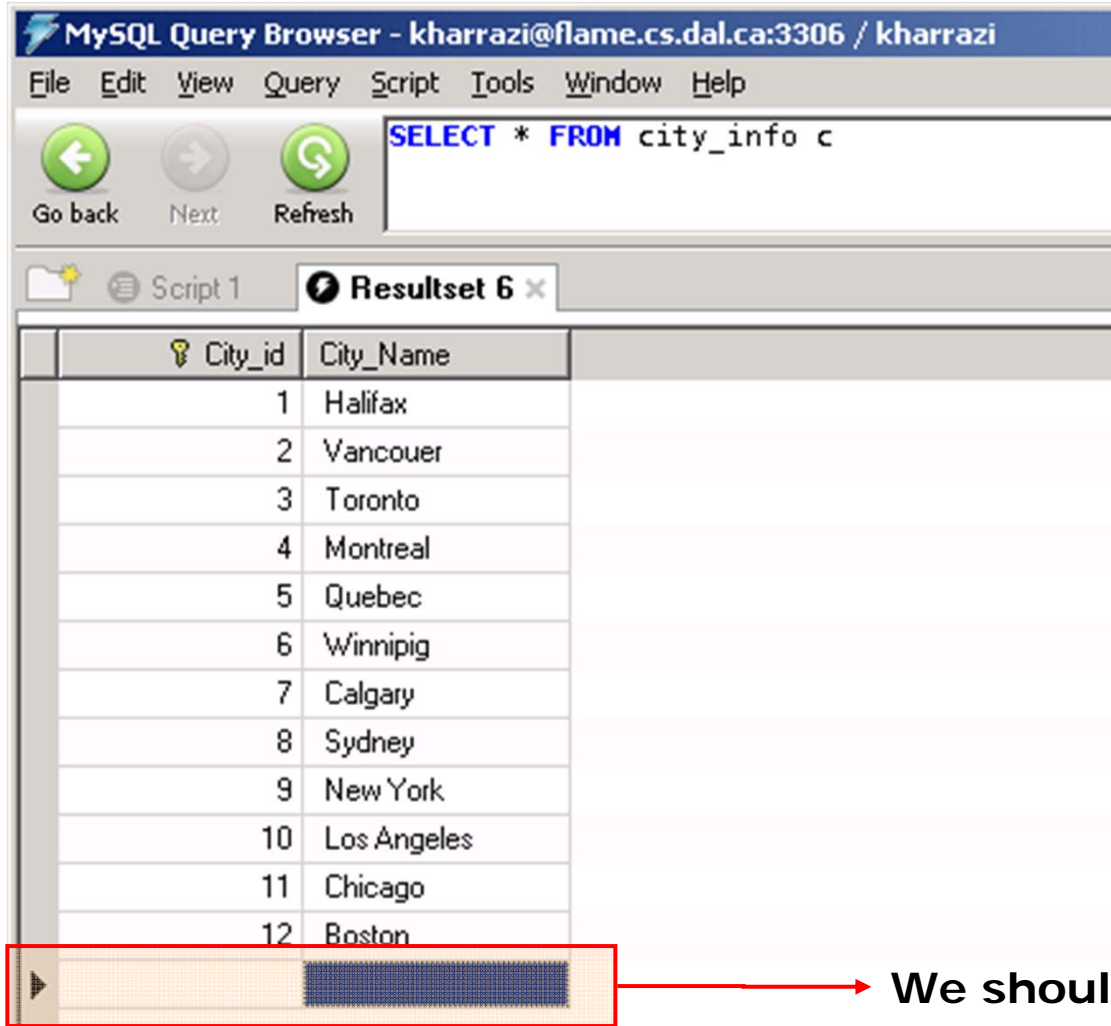
- The **INSERT INTO** statement is used to insert new rows into a table.
- Syntax:

```
INSERT INTO table_name VALUES (value1, value2,....)
```

- Examples:

```
INSERT INTO city_info VALUES (1, 'Berlin')
```


SQL: *INSERT INTO* (cont.)



MySQL Query Browser - kharrazi@flame.cs.dal.ca:3306 / kharrazi

File Edit View Query Script Tools Window Help

Go back Next Refresh

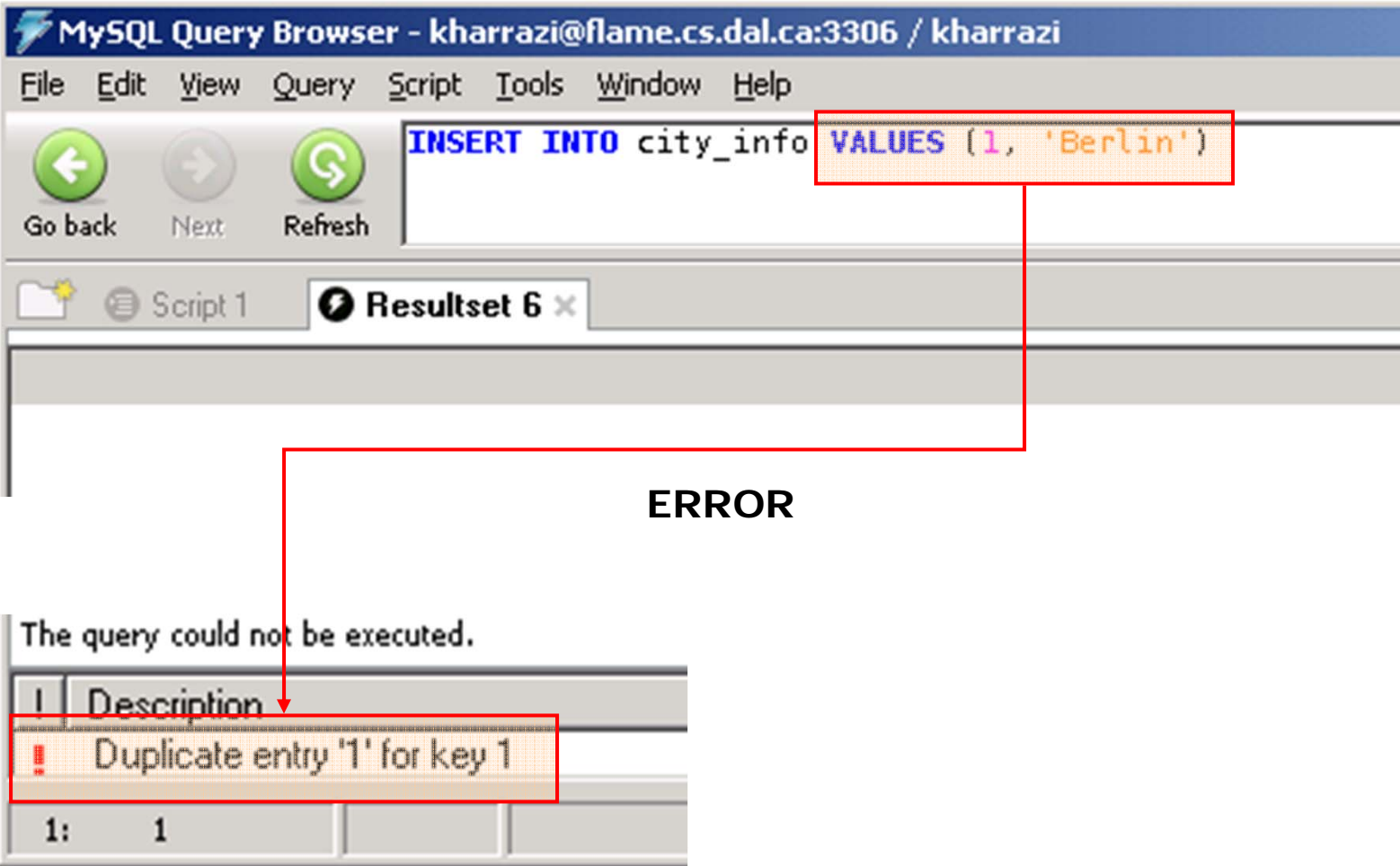
```
SELECT * FROM city_info c
```

Script 1 Resultset 6 x

City_id	City_Name
1	Halifax
2	Vancouver
3	Toronto
4	Montreal
5	Quebec
6	Winnipeg
7	Calgary
8	Sydney
9	New York
10	Los Angeles
11	Chicago
12	Boston

We should insert here

SQL: *INSERT INTO* (cont.)

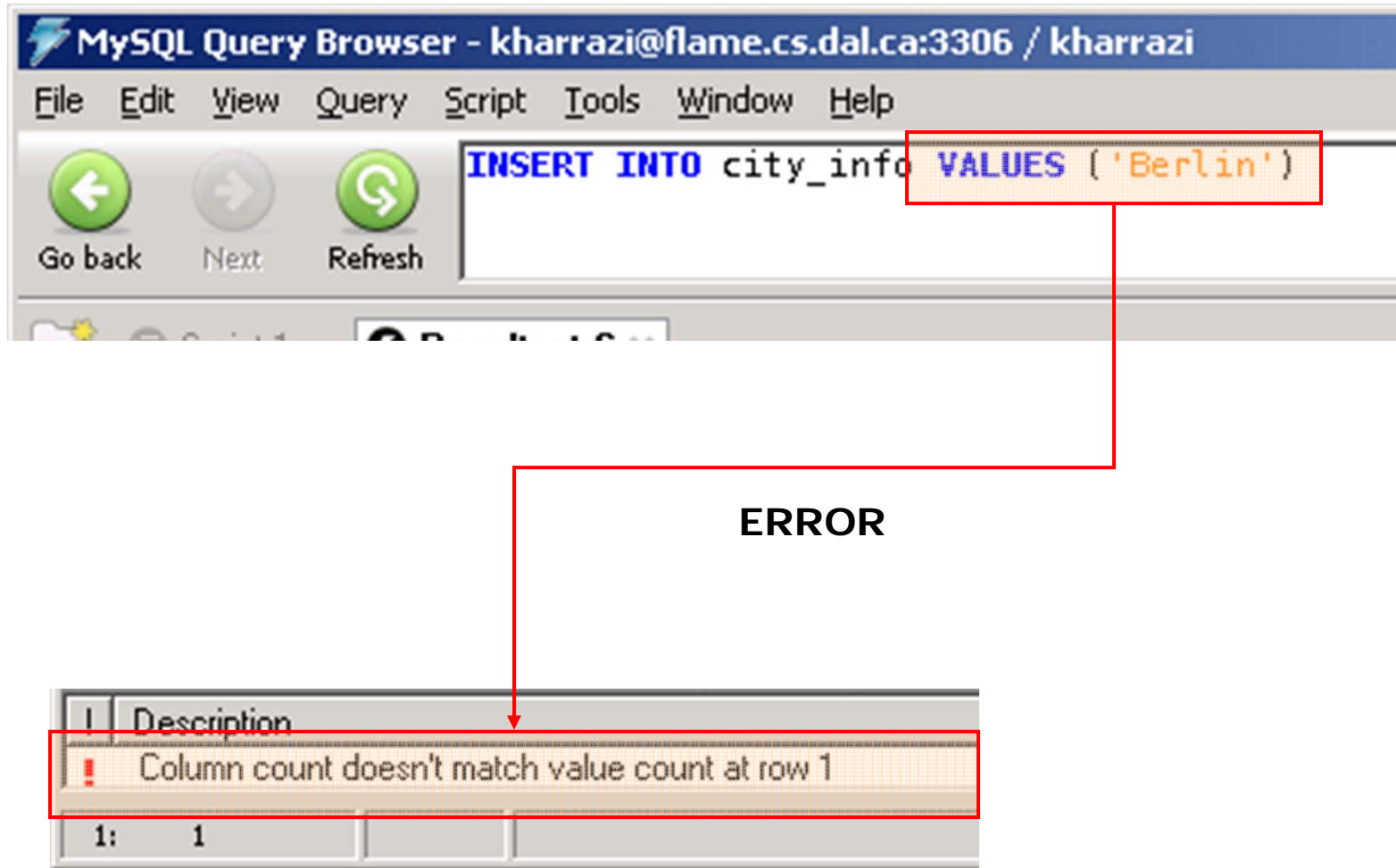


The screenshot shows the MySQL Query Browser interface. The query editor contains the SQL statement: `INSERT INTO city_info VALUES (1, 'Berlin')`. The values `(1, 'Berlin')` are highlighted with a red box. Below the query editor, the error message is displayed: "The query could not be executed." The error details are shown in a table with a red background and a red exclamation mark icon:

Description	
!	Duplicate entry '1' for key 1

The error message is also highlighted with a red box. A red line connects the highlighted values in the query to the error message. The word "ERROR" is written in the center of the image.

SQL: *INSERT INTO* (cont.)



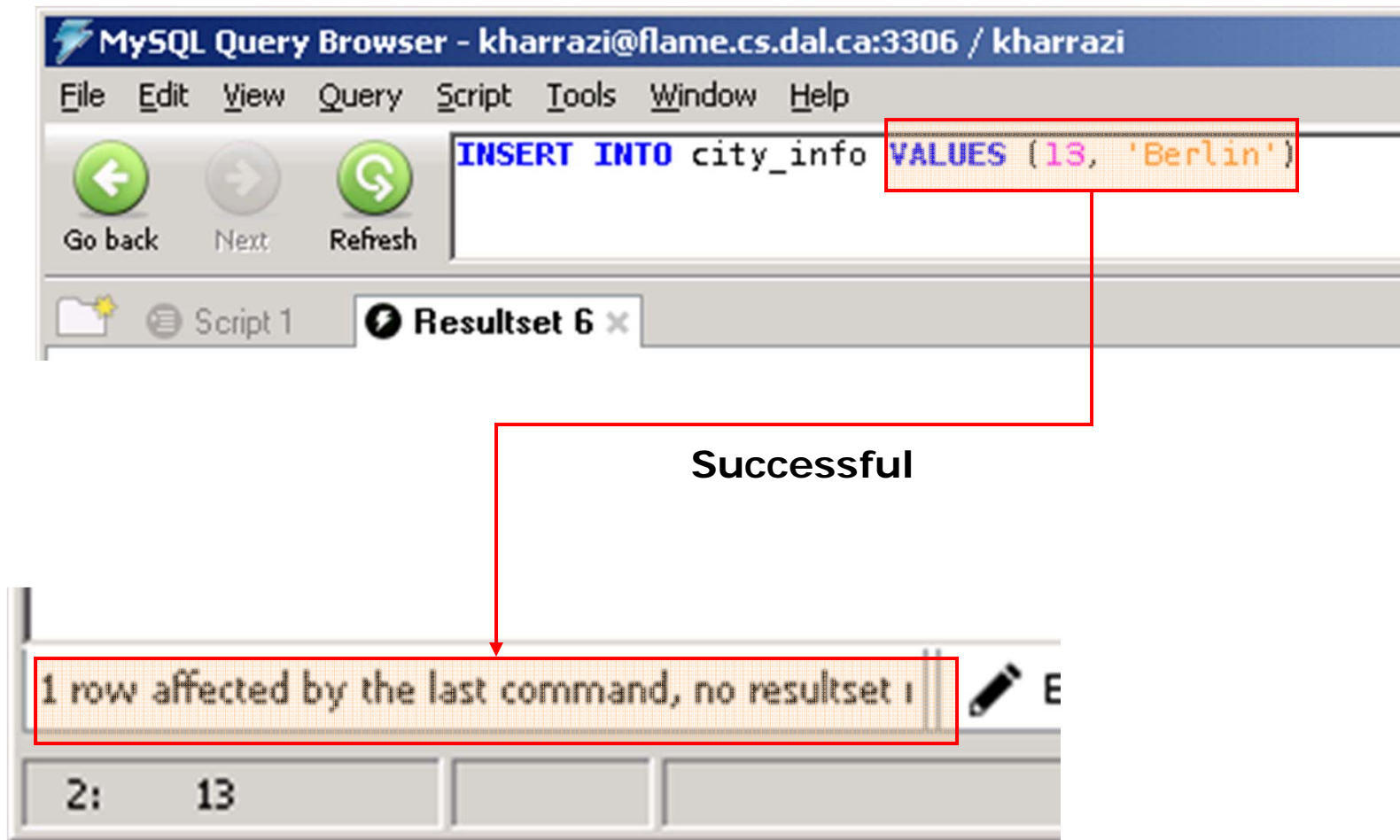
The screenshot shows the MySQL Query Browser interface. The title bar reads "MySQL Query Browser - kharrazi@flame.cs.dal.ca:3306 / kharrazi". The menu bar includes "File", "Edit", "View", "Query", "Script", "Tools", "Window", and "Help". Below the menu bar are three navigation buttons: "Go back", "Next", and "Refresh". The main query area contains the SQL statement: `INSERT INTO city_info VALUES ('Berlin')`. A red box highlights the `VALUES ('Berlin')` portion of the query. A red arrow points from this box to the word "ERROR" in the center of the screen. Below "ERROR", another red arrow points to a red-bordered error message box. The error message box contains a red exclamation mark icon and the text: "Column count doesn't match value count at row 1". Below the error message, a table structure is visible with a header row "Description" and a data row "1: 1".

ERROR

Column count doesn't match value count at row 1

Description
1: 1

SQL: *INSERT INTO* (cont.)



The screenshot shows the MySQL Query Browser interface. The title bar reads "MySQL Query Browser - kharrazi@flame.cs.dal.ca:3306 / kharrazi". The menu bar includes File, Edit, View, Query, Script, Tools, Window, and Help. Below the menu bar are navigation buttons: "Go back", "Next", and "Refresh". The main query area contains the SQL command: `INSERT INTO city_info VALUES (13, 'Berlin')`. The values `(13, 'Berlin')` are highlighted with a red box. Below the query area, there are tabs for "Script 1" and "Resultset 6". The "Resultset 6" tab is active, showing a message: "1 row affected by the last command, no resultset". This message is also highlighted with a red box. A red arrow points from the highlighted values in the query to the message in the resultset tab. The word "Successful" is written in the center of the image, with a red arrow pointing from it to the message in the resultset tab. Below the message, there is a table with two columns and two rows. The first row has the value "2:" in the first column and "13" in the second column. The second row is empty.

MySQL Query Browser - kharrazi@flame.cs.dal.ca:3306 / kharrazi

File Edit View Query Script Tools Window Help

Go back Next Refresh

```
INSERT INTO city_info VALUES (13, 'Berlin')
```

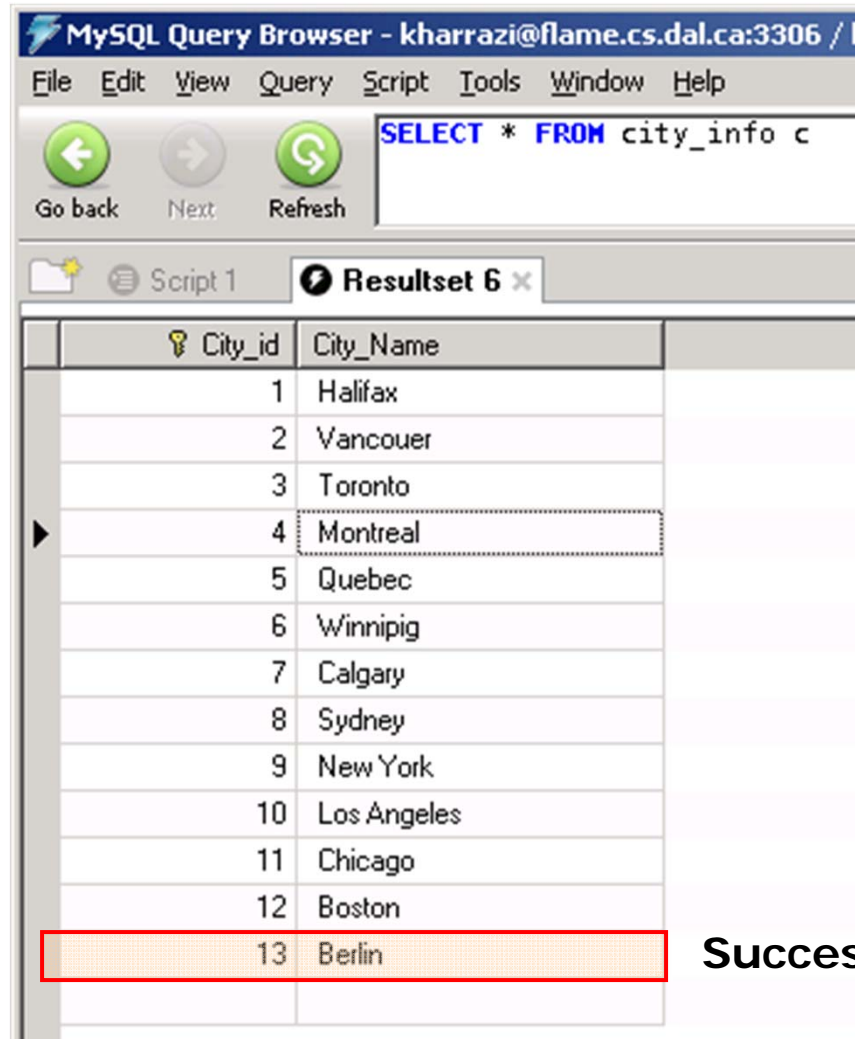
Script 1 Resultset 6 x

Successful

1 row affected by the last command, no resultset

2:	13

SQL: *INSERT INTO* (cont.)



The screenshot shows the MySQL Query Browser interface. The query entered is `SELECT * FROM city_info c`. The result set, titled "Resultset 6", displays a table with two columns: "City_id" and "City_Name". The table contains 13 rows of data. The last row, with "City_id" 13 and "City_Name" Berlin, is highlighted with a red border and labeled "Successfully added".

City_id	City_Name
1	Halifax
2	Vancouver
3	Toronto
4	Montreal
5	Quebec
6	Winnipig
7	Calgary
8	Sydney
9	New York
10	Los Angeles
11	Chicago
12	Boston
13	Berlin

Successfully added

SQL: *INSERT INTO* (cont.)

GUI Insertion (Edit) Mode:

MySQL Query Browser - kharrazi@flame.cs.dal.ca:3306 / kharrazi

File Edit View Query Script Tools Window Help

Go back Next Refresh `SELECT * FROM city_info c` Execute Stop

Resultset 1

City_id	City_Name
1	Halifax
2	Vancouver
3	Toronto
4	Montreal
5	Quebec
6	Winnipeg
7	Calgary
8	Sydney
9	New York
10	Los Angeles
11	Chicago
12	Boston
13	Miami

1 Enter Edit Mode

2 Do the changes (it would be not applied until you confirm)

3 Confirm or reject the changed data

12 rows fetched in 0.0040s (0.0015s) Edit Apply Changes Discard Changes First Last Search

Schemata Bookmarks History

kharrazi

- admin_info
- city_info
- client_info
- doc_info
- kin_info
- pat_doc_relate
- pat_info
- phone_info

Syntax Functions Params Trx

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

6. SQL: **UPDATE/SET** Statement

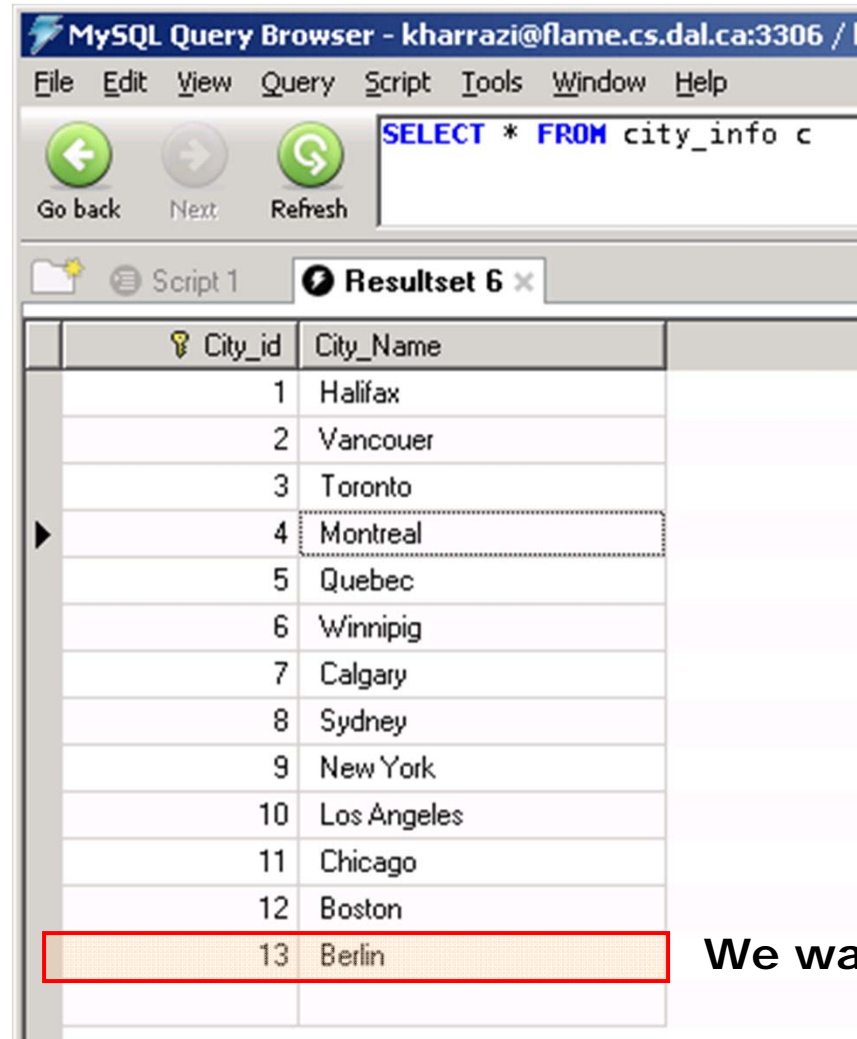
- The **UPDATE** statement is used to modify the data in a table.
- Syntax:

```
UPDATE table_name SET column_name = new_value  
WHERE column_name = old_value
```

- Examples:

```
UPDATE city_info SET City_Name = 'Truro'  
WHERE City_Name = 'Berlin'
```

SQL: *UPDATE/SET* (cont.)

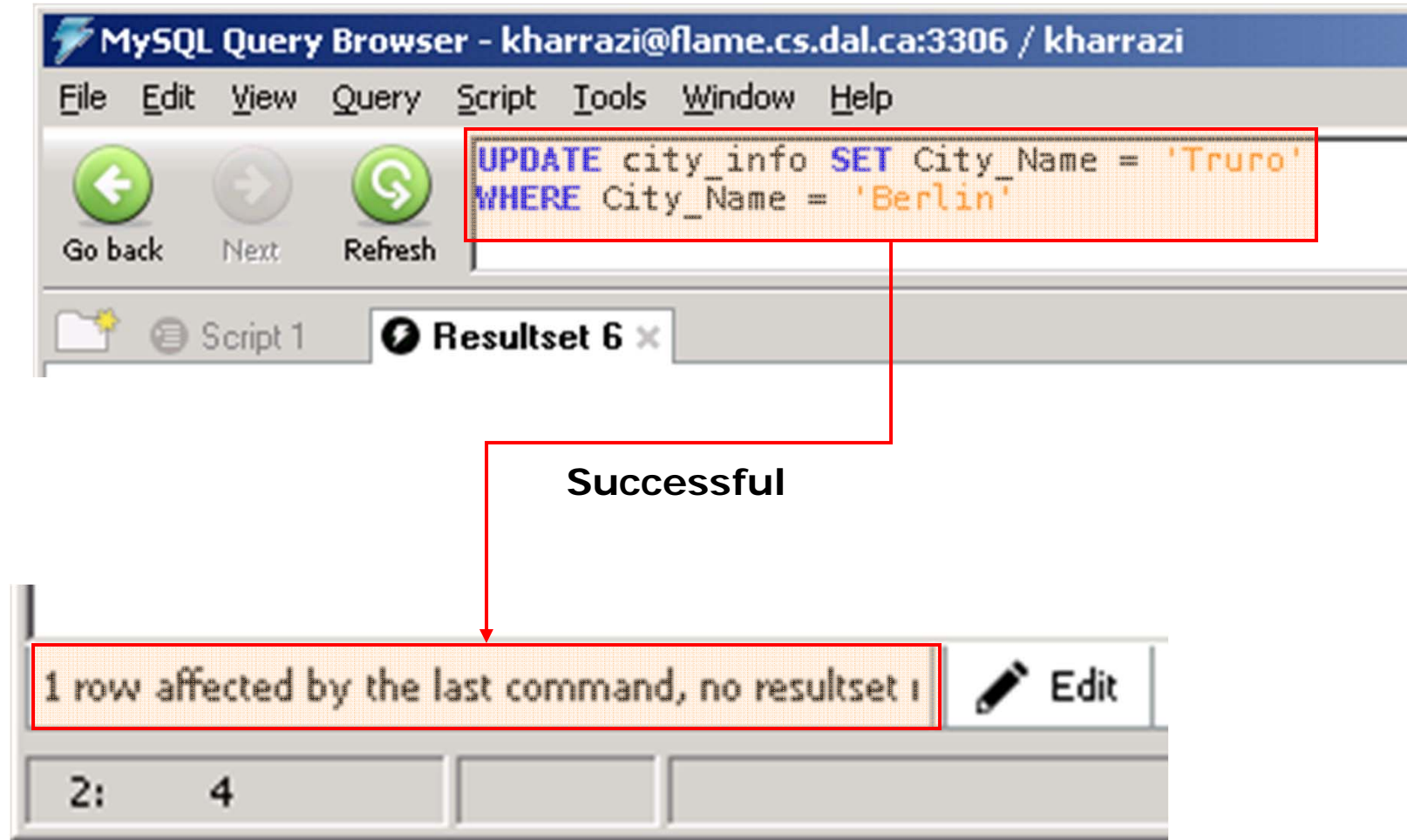


The screenshot shows the MySQL Query Browser interface. The title bar reads "MySQL Query Browser - kharrazi@flame.cs.dal.ca:3306 / k". The menu bar includes "File", "Edit", "View", "Query", "Script", "Tools", "Window", and "Help". Below the menu bar are navigation buttons: "Go back", "Next", and "Refresh". The query editor contains the text "SELECT * FROM city_info c". Below the query editor, there are tabs for "Script 1" and "Resultset 6 x". The main area displays a table with two columns: "City_id" (marked as a primary key) and "City_Name". The table contains 13 rows of data. The row with "City_id" 13 and "City_Name" "Berlin" is highlighted with a red border. To the right of the table, the text "We want to change it" is displayed.

City_id	City_Name
1	Halifax
2	Vancouver
3	Toronto
4	Montreal
5	Quebec
6	Winnipeg
7	Calgary
8	Sydney
9	New York
10	Los Angeles
11	Chicago
12	Boston
13	Berlin

We want to change it

SQL: *UPDATE/SET* (cont.)

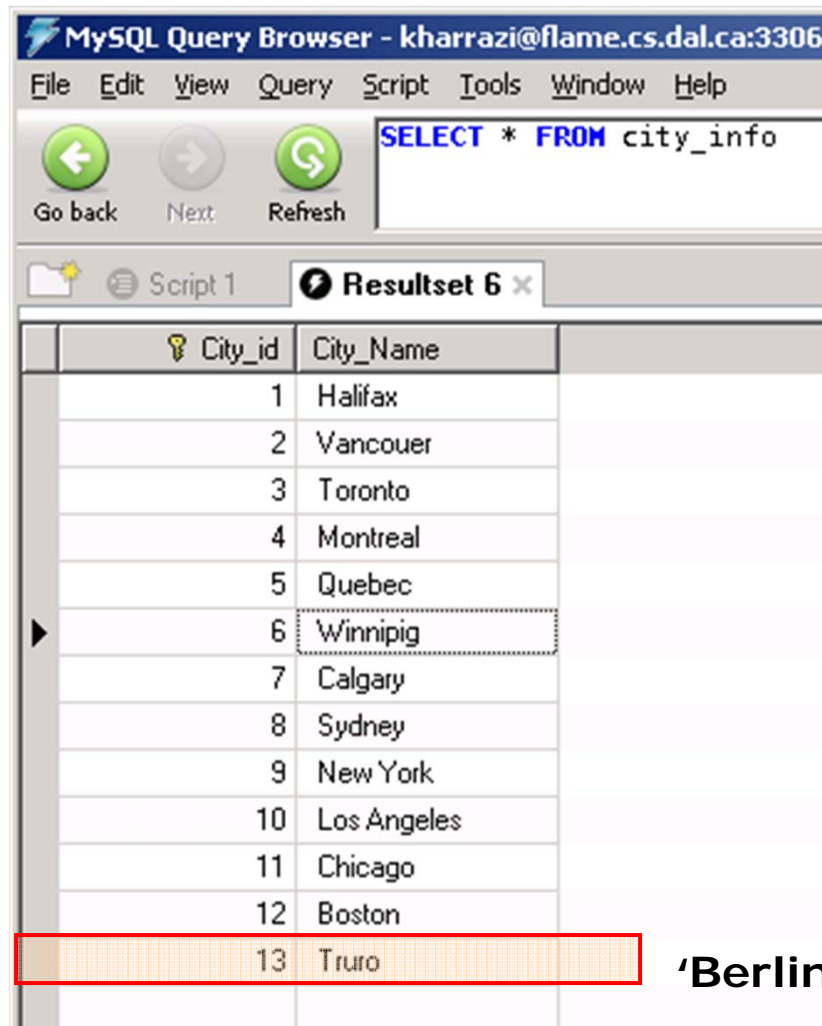


The screenshot shows the MySQL Query Browser interface. The title bar reads "MySQL Query Browser - kharrazi@flame.cs.dal.ca:3306 / kharrazi". The menu bar includes File, Edit, View, Query, Script, Tools, Window, and Help. Below the menu bar are navigation buttons: "Go back", "Next", and "Refresh". The main query area contains the following SQL statement, highlighted with a red box:

```
UPDATE city_info SET City_Name = 'Truro'  
WHERE City_Name = 'Berlin'
```

Below the query area, there are tabs for "Script 1" and "Resultset 6". A red arrow points from the query area to the "Resultset 6" tab, which displays the message "Successful". Below this, another red box highlights the message "1 row affected by the last command, no resultset |", with an "Edit" button to its right. At the bottom of the interface, there are two input fields containing the numbers "2:" and "4".

SQL: *UPDATE/SET* (cont.)



The screenshot shows the MySQL Query Browser interface. The title bar reads "MySQL Query Browser - kharrazi@flame.cs.dal.ca:3306". The menu bar includes "File", "Edit", "View", "Query", "Script", "Tools", "Window", and "Help". Below the menu bar are three navigation buttons: "Go back", "Next", and "Refresh". The query text area contains the SQL statement: `SELECT * FROM city_info`. Below the query area, there are two tabs: "Script 1" and "Resultset 6 x". The "Resultset 6 x" tab is active, displaying a table with the following data:

City_id	City_Name
1	Halifax
2	Vancouver
3	Toronto
4	Montreal
5	Quebec
6	Winnipeg
7	Calgary
8	Sydney
9	New York
10	Los Angeles
11	Chicago
12	Boston
13	Truro

The row with City_id 13 and City_Name 'Truro' is highlighted with a red border. To the right of the table, the text **'Berlin' has changed to 'Truro'** is displayed.

'Berlin' has changed to 'Truro'

7. SQL: **DELETE** Statement

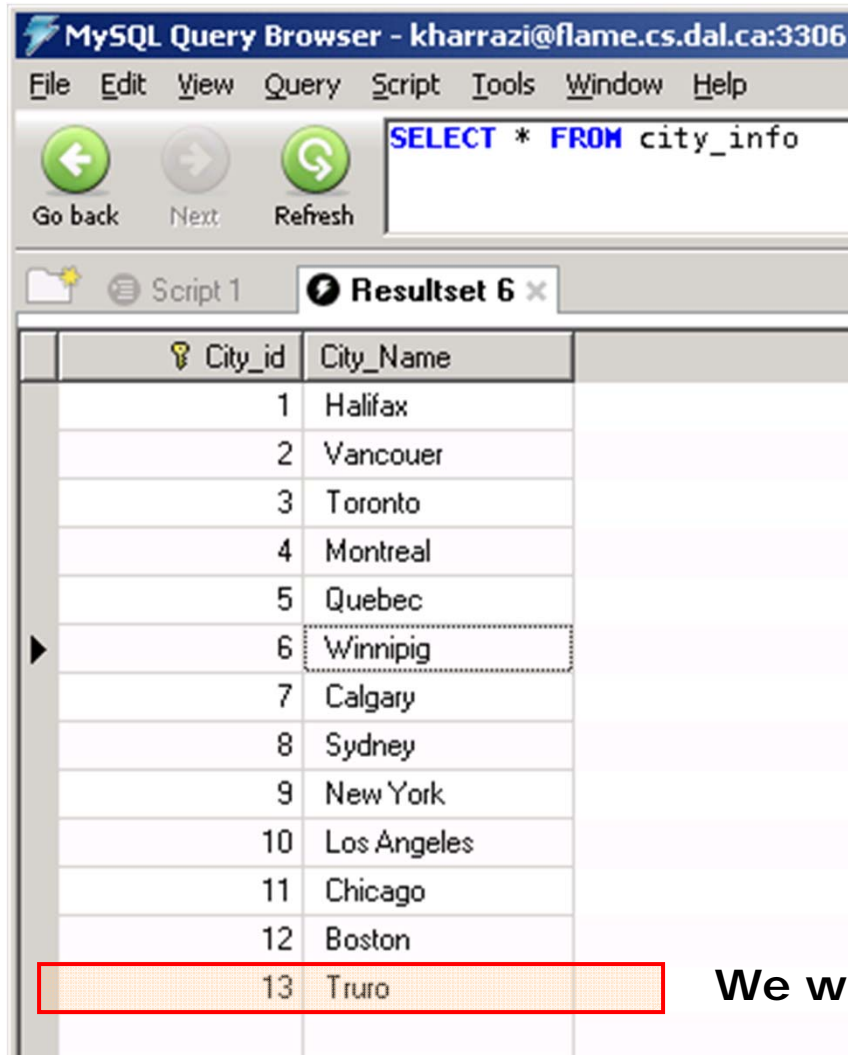
- The **DELETE** statement is used to delete rows in a table.
- Syntax:

```
DELETE FROM table_name WHERE column_name = some_value
```

- Examples:

```
DELETE FROM city_info WHERE City_Name = 'Truro'
```

SQL: *DELETE* (cont.)

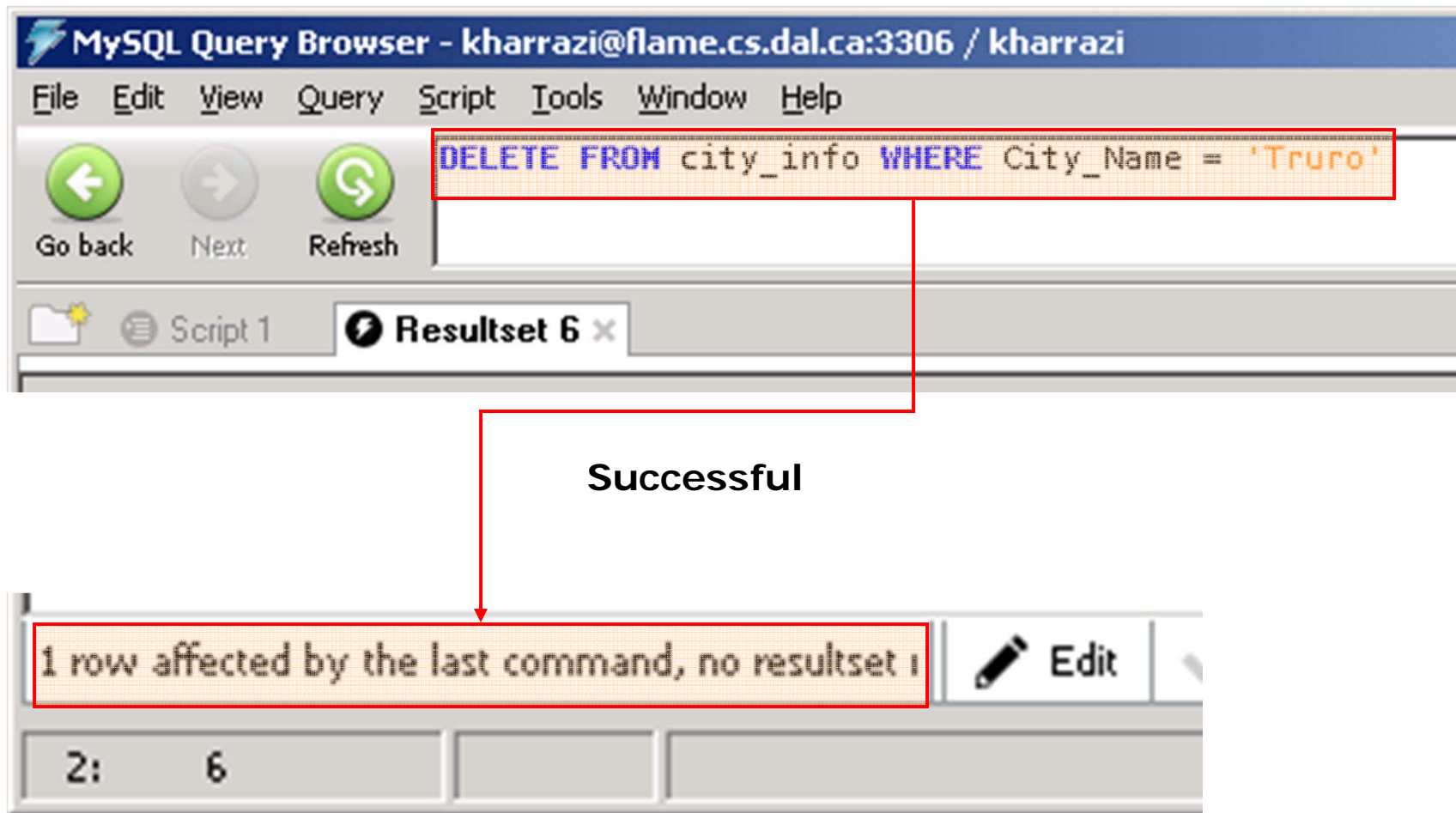


The screenshot shows the MySQL Query Browser interface. The title bar reads "MySQL Query Browser - kharrazi@flame.cs.dal.ca:3306". The menu bar includes "File", "Edit", "View", "Query", "Script", "Tools", "Window", and "Help". Below the menu bar are three navigation buttons: "Go back", "Next", and "Refresh". The query editor contains the text "SELECT * FROM city_info". Below the query editor, there are tabs for "Script 1" and "Resultset 6". The main area displays a table with the following data:

City_id	City_Name
1	Halifax
2	Vancouver
3	Toronto
4	Montreal
5	Quebec
6	Winnipig
7	Calgary
8	Sydney
9	New York
10	Los Angeles
11	Chicago
12	Boston
13	Truro

We want to delete this row

SQL: **DELETE** (cont.)



The screenshot displays the MySQL Query Browser interface. The title bar reads "MySQL Query Browser - kharrazi@flame.cs.dal.ca:3306 / kharrazi". The menu bar includes "File", "Edit", "View", "Query", "Script", "Tools", "Window", and "Help". Below the menu bar are navigation buttons: "Go back", "Next", and "Refresh". The main query area contains the SQL command: `DELETE FROM city_info WHERE City_Name = 'Truro'`. Below the query area, there are tabs for "Script 1" and "Resultset 6". The "Resultset 6" tab is active, showing the message: "1 row affected by the last command, no resultset". An "Edit" button is visible next to the message. A red box highlights the SQL command in the query area, and another red box highlights the execution result message. A red arrow points from the SQL command to the result message, with the word "Successful" written above the arrow.

MySQL Query Browser - kharrazi@flame.cs.dal.ca:3306 / kharrazi

File Edit View Query Script Tools Window Help

Go back Next Refresh

Script 1 Resultset 6 x

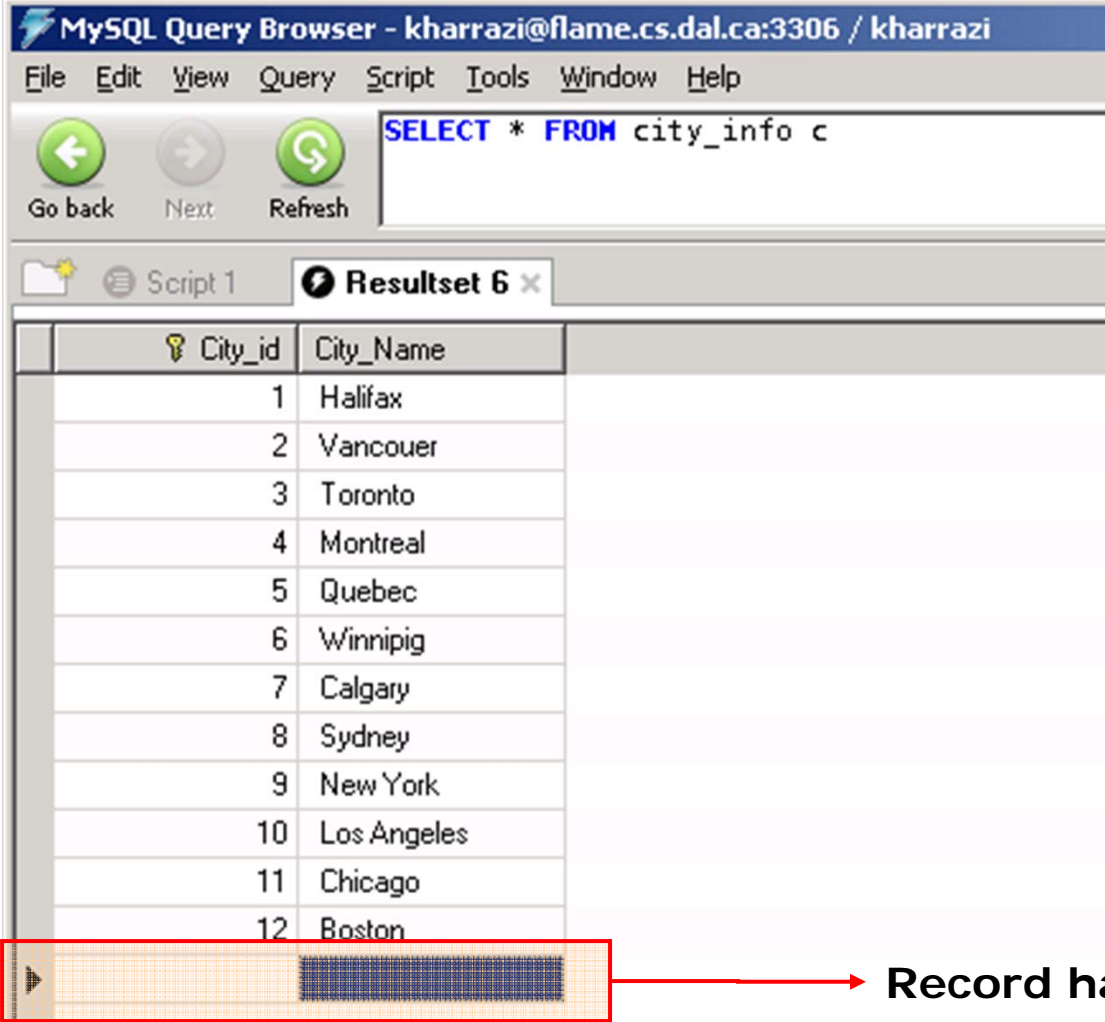
`DELETE FROM city_info WHERE City_Name = 'Truro'`

Successful

1 row affected by the last command, no resultset | Edit

2: 6

SQL: *DELETE* (cont.)



The screenshot shows the MySQL Query Browser interface. The title bar reads "MySQL Query Browser - kharrazi@flame.cs.dal.ca:3306 / kharrazi". The menu bar includes "File", "Edit", "View", "Query", "Script", "Tools", "Window", and "Help". Below the menu bar are three navigation buttons: "Go back", "Next", and "Refresh". The query editor contains the SQL statement: `SELECT * FROM city_info c`. Below the query editor, there are two tabs: "Script 1" and "Resultset 6". The "Resultset 6" tab is active, displaying a table with two columns: "City_id" and "City_Name". The table contains 12 rows of data. A red box highlights the bottom of the table, and a red arrow points from the box to the text "Record has been deleted".

City_id	City_Name
1	Halifax
2	Vancouver
3	Toronto
4	Montreal
5	Quebec
6	Winnipeg
7	Calgary
8	Sydney
9	New York
10	Los Angeles
11	Chicago
12	Boston

Record has been deleted

8. SQL: Joining and Keys

- Elements of the relational database table:

The diagram illustrates the components of a relational database table. A table with 16 rows and 7 columns is shown. The columns are: P_id, P_FirstName, P_LastName, City_id, Phone_id, Min_id, and Client_id. The table is annotated with red arrows and labels:

- Key (auto-increase)**: Points to the P_id column.
- Attribute**: Points to the P_LastName column.
- Column**: Points to the Min_id column.
- Table**: A bracket encompasses the entire table structure.
- Row (Topple)**: Points to the 8th row (Mehrdad Kharrazi).
- Cell**: Points to the cell containing 'Catherin' in the 14th row.
- Foreign Key (referring to another table)**: Points to the value '5' in the City_id column of the 13th row (Brian Adam).

	P_id	P_FirstName	P_LastName	City_id	Phone_id	Min_id	Client_id
▶	1	Peter	Johnsons	1	4	6	14
	2	Mike	Jackson	1	13	6	15
	3	Sara	Henson	3	6	2	16
	4	John	McDonald	5	8	3	17
	5	Michael	Robinson	1	13	6	18
	6	William	Jordan	4	10	4	19
	7	Susan	McKinsy	1	2	5	20
	8	Mehrdad	Kharrazi	2	1	9	21
	9	John	McKinsy	1	9	10	22
	10	John	McDonald	3	18	7	23
	11	Pat	Bentatar	7	25	8	24
	12	Abraham	Lincoln	3	26	27	25
	13	Brian	Adam	5	27	13	26
	14	Catherin	Catholicy	7	28	15	33
	15	Demi	Moore	12	29	23	34
	16	Ebi	Farahanzadeh	11	30	26	42

SQL: Joining and Keys (cont.)

- Foreign Key (1 to Many):

P_id	P_FirstName	P_LastName	City_id	Phone_id	Kin_id	Client_id
1	Peter	Johnsons	1	4	6	14
2	Mike	Jackson	1	13	6	15
3	Sara	Henson	3	6	2	16
4	John	McDonald	5	9	3	17
5	Michael	Robinson	1	13	6	18
6	William	Jordan				19
7	Susan	McKinsy				20
8	Mehdi	Kharrazi				21
9	John	McKinsy				22
10	John	McDonald				23
11	Pat	Bentatar	7	25	8	24
12	Abraham	Lincoln	3	26	27	25
13	Brian	Adam	5	27	13	26
14	Catherin	Catholicy	7	28	15	33
15	Demi	Moore	12	29	23	34
16	Ebi	Farahanzadeh	11	30	26	42

Table: Patient_information (MANY)

City_id	City_Name
1	Halifax
2	Vancouver
3	Toronto
4	Montreal
5	Quebec
6	Winnipeg
7	Calgary
8	Sydney
9	New York
10	Los Angeles
11	Chicago
12	Boston

Table: City_information (ONE)

Foreign Key
(referring to
another table)

SQL: Joining and Keys (cont.)

- Foreign Key (Many to Many):

Patient #1 has doctor #4, #5 and #6.

P_id	P_FirstName	P_LastName	City_id	Phone_id	Kin_id	Client_id
1	Peter	Johnson	1	4	6	14
2	Mike	Jackson	1	13	6	15
3	Sara	Henson	3	6	2	16
4	John	McDonald	5	8	3	17
5	Michael	Robinson	1	13	6	18
6	William	Jordan	4	10	4	19
7	Susan	McKinsy	1	2	5	20
8	Mehdi	Kharrazi	2	1	9	21
9	John	McKinsy	1	9	10	22
10	John	McDonald	3			
11	Pat	Bentatar	7			
12	Abraham	Lincoln	3			
13	Brian	Adam	5			
14	Catherin	Catholicy	7			
15	Demi	Moore	12			
16	Ebi	Ferhonzedch	11			

Table: Patient_information (MANY)

Dr. #1 has patient #2 and #4.

D_id	D_FirstName	D_LastNa...	City_id	Phone_id	Client_id
1	Mary	Kharrazi	1	3	4
2	John	McDonald	4	11	5
3	Robin	Dorby	2	8	6
4	Mary	Johnson	3	1	7
5	Y...	Lee	1	12	8
6	W...	Mclssac	1	4	9
7	John	Peterson	2	16	10
8	Jane	Peterson	2	13	11
9	Susan	Waterloo	3	15	13
10	Samuel	Jackson	2	14	12

Table: Doctor_information (MANY)

P_id	D_id
1	4
1	5
1	6
2	1
3	9
4	2
4	10
5	5
6	2
7	6

Table: Patient_Doctor_relationship

SQL: Joining and Keys (cont.)

- Sometimes we have to select data from two or more tables to make our result complete. We have to perform a join.
- Tables in a database can be related to each other with keys. A **primary key** is a column with a unique value for each row. **The purpose is to bind data together, across tables, without repeating all of the data in every table.**
- We can select data from two tables by referring to two tables using the primary keys relating the tables together.

SQL: Joining and Keys (1 - Many)

- Syntax:

```
SELECT table1.any_column, table2.any_column
FROM table1, table2
WHERE table1.columnX = table2.columnX
```

- Examples:

```
SELECT * FROM pat_info, city_info
WHERE pat_info.City_id = city_info.City_id
```

```
SELECT pat_info.P_FirstName, city_info.City_Name
FROM pat_info, city_info
WHERE pat_info.City_id = city_info.City_id
```

SQL: Joining and Keys (cont.)

- Finding the cities that the patients are living in them?

P_id	P_FirstName	P_LastName	City_id	Phone_id	Kin_id	Client_id
1	Peter	Johnsons	1	4	6	14
2	Mike	Jackson	1	13	6	15
3	Sara	Henson	3	6	2	16
4	John	McDonald	5	9	3	17
5	Michael	Robinson	1	13	6	18
6	William	Jordan				19
7	Susan	McKinsy				20
8	Mehdi	Kharrazi				21
9	John	McKinsy				22
10	John	McDonald				23
11	Pat	Bentatar	7	25	8	24
12	Abraham	Lincoln	3	26	27	25
13	Brian	Adam	5	27	13	26
14	Catherin	Catholicy	7	28	15	33
15	Demi	Moore	12	29	23	34
16	Ebi	Farahanzadeh	11	30	26	42

Table: Patient_information (MANY)

City_id	City_Name
1	Halifax
2	Vancouver
3	Toronto
4	Montreal
5	Quebec
6	Winnipig
7	Calgary
8	Sydney
9	New York
10	Los Angeles
11	Chicago
12	Boston

Table: City_information (ONE)

Foreign Key
(referring to
another table)

SQL: Joining and Keys (cont.)

The screenshot shows a MySQL Query Browser window with the following SQL query:

```
SELECT * FROM pat info, city info
WHERE pat_info.City_id = city_info.City_id
```

The result set, titled "Resultset 1", displays the following data:

P...	P_FirstNa...	P_LastName	City_id	Phone_id	Kin_id	Client_id	Cit...	City_Name
1	Peter	Johnsons	1	4	6	14	1	Halifax
2	Mike	Jackson	1	13	6	15	1	Halifax
3	Sara	Henson	3	6	2	16	3	Toronto
4	John	McDonald	5	8	3	17	5	Quebec
5	Michael	Robinson	1	13	6	18	1	Halifax
6	William	Jordan	4	10	4	19	4	Montreal
7	Susan	McKinsy	1	2	5	20	1	Halifax
8	Mehdi	Kharrazi	2	1	9	21	2	Vancouver
9	John	McKinsy	1	9	10	22	1	Halifax
10	John	McDonald	3	18	7	23	3	Toronto
11	Pat	Bentatar	7	25	8	24	7	Calgary
12	Abraham	Lincoln	3	26	27	25	3	Toronto
13	Brian	Adam	5	27	13	26	5	Quebec
14	Catherin	Catholcy	7	28	15	33	7	Calgary
15	Demi	Moore	12	29	23	34	12	Boston
16	Ebi	Farahanza	11	30	26	42	11	Chicago
17	Fery	Sea	12	31	14	43	12	Boston
18	Graham	Bell	12	32	12	32	12	Boston
19	Hamilton	Green	1	33	4	44	1	Halifax

Red arrows indicate the mapping between the query and the data:

- The `pat info` table in the query corresponds to the `pat info` table on the left, with columns `P_id`, `P_FirstName`, `P_LastName`, `City_id`, `Phone_id`, `Kin_id`, and `Client_id`.
- The `city info` table in the query corresponds to the `City` table at the bottom left, with columns `City_id` and `City_Name`.
- The `WHERE pat_info.City_id = city_info.City_id` clause is highlighted in red, and red arrows show how the `City_id` values in the result set match the `City_id` values in the `City` table.

Table: Patient_information + City_information

SQL: Joining and Keys (cont.)

The screenshot shows a MySQL Query Browser window with the following SQL query:

```
SELECT pat_info.P_FirstName, pat_info.P_LastName, city_info.City_Name
FROM pat_info, city_info
WHERE pat_info.City_id = city_info.City_id
```

The result set shows the following data:

P_FirstName	P_LastName	City_Name
Peter	Johnsons	Halifax
Mike	Jackson	Halifax
Sara	Henson	Toronto
John	McDonald	Quebec
Michael	Robinson	Halifax
William	Jordan	Montreal
Susan	McKinsy	Halifax
Mehdi	Kharrazi	Vancouver
John	McKinsy	Halifax
John	McDonald	Toronto
Pat	Bentatar	Calgary
Abraham	Lincoln	Toronto
Brian	Adam	Quebec
Catherin	Catholicy	Calgary
Demi	Moore	Boston
Ebi	Farahanzadeh	Chicago
Fery	Sea	Boston
Graham	Bell	Boston
Hamilton	Green	Halifax

Table: Patient_information + City_information

SQL: Joining and Keys (1 - Many) (cont.)

- Syntax:

```
SELECT table1.any_column, table2.any_column
FROM table1
INNER/LEFT/RIGHT JOIN table2
ON table1.columnX = table2.columnX
```

- Examples:

```
SELECT pat_info.P_FirstName, pat_info.P_LastName,
city_info.City_Name
```

```
FROM pat_info
```

```
INNER/LEFT/RIGHT JOIN city_info
```

```
ON pat_info.City_id = city_info.City_id
```

SQL: Joining and Keys (1 - Many) (cont.)

P_id	P_FirstName	P_LastName	City_id	Phone_id	Kin_id	Client_id
1	Peter	Johnsons	1	4	6	14
2	Mike	Jackson	1	13	6	15
3	Sara	Henson	3	6	2	16
4	John	McDonald	5	8	3	17
5	Michael	Robinson	1	13	6	18
6	William	Jordan	4	10	4	19
7	Susan	McKinsy	1	2	5	20
8	Mehdi	Kharrazi	2	1	9	21
9	John	McKinsy	1	9	10	22
10	John	McDonald	3	18	7	23
11	Pat	Bentalar	7	25	8	24
12	Abraham	Lincoln	3	26	27	25
13	Brian	Adam	5	27	13	26
14	Catherin	Catholcy	7	28	15	33
15	Demi	Moore	12	29	23	34
16	Ebi	Farahanzadeh	11	30	26	42

City_id	City_Name
1	Halifax
2	Vancouver
3	Toronto
4	Montreal
5	Quebec
6	Winnipeg
7	Calgary
8	Sydney
9	New York
10	Los Angeles
11	Chicago
12	Boston

MySQL Query Browser - kharrazi@flame.cs.dal.ca:3306 / kharrazi

File Edit View Query Script Tools Window Help

Transaction Explain Compare

Resultset 1

```

SQL Query Area
1 SELECT pat_info.P_FirstName, pat_info.P_LastName, city_info.City_Name
2 FROM pat_info
3 INNER JOIN city_info
4 ON pat_info.City_id = city_info.City_id
    
```

P_FirstName	P_LastName	City_Name
Peter	Johnsons	Halifax
Mike	Jackson	Halifax
Sara	Henson	Toronto
John	McDonald	Quebec
Michael	Robinson	Halifax
William	Jordan	Montreal
Susan	McKinsy	Halifax
Mehdi	Kharrazi	Vancouver
John	McKinsy	Halifax
John	McDonald	Toronto

Table: Patient_information + City_information

SQL: Joining and Keys (1 - Many) (cont.)

- Examples:

```
SELECT pat_info.P_FirstName,  
pat_info.P_LastName, city_info.City_Name  
  
FROM pat_info  
  
INNER JOIN city_info  
  
ON pat_info.City_id = city_info.City_id  
  
AND city_info.City_Name = 'Halifax'
```

SQL: Joining and Keys (1 - Many) (cont.)

P_id	P_FirstName	P_LastName	City_id	Phone_id	Kin_id	Client_id
1	Peter	Johnsons	1	4	6	14
2	Mike	Jackson	1	13	6	15
3	Sara	Henson	3	6	2	16
4	John	McDonnald	5	8	3	17
5	Michael	Robinson	1	13	6	18
6	William	Jordan	4	10	4	19
7	Susan	McKinsy	1	2	5	20
8	Mehdi	Kharazi	2	1	9	21
9	John	McKinsy	1	9	10	22
10	John	McDonnald	3	18	7	23
11	Pat	Bentalar	7	25	8	24
12	Abraham	Lincoln	3	26	27	25
13	Brian	Adam	5	27	13	26
14	Catherin	Catholocy	7	28	15	33
15	Demi	Moore	12	29	23	34
16	Ebi	Farahanzadeh	11	30	26	42

City_id	City_Name
1	Halifax
2	Vancouver
3	Toronto
4	Montreal
5	Quebec
6	Winnipig
7	Calgary
8	Sydney
9	New York
10	Los Angeles
11	Chicago
12	Boston

Resultset 1

SQL Query Area

```

1 SELECT pat_info.P_FirstName, pat_info.P_LastName, city_info.City_Name
2 FROM pat_info
3 INNER JOIN city_info
4 ON pat_info.City_id = city_info.City_id
5 AND city_info.City_Name = 'Halifax'
        
```

P_FirstName	P_LastName	City_Name
Peter	Johnsons	Halifax
Mike	Jackson	Halifax
Michael	Robinson	Halifax
Susan	McKinsy	Halifax
John	McKinsy	Halifax
Hamilton	Green	Halifax
Isaac	Killiam	Halifax
Josef	Koopy	Halifax
Lilliam	Toram	Halifax
Sam	Hill	Halifax
Robin	Carrier	Halifax
Peter	Yu	Halifax
Bill	Clincton	Halifax
Woo	Xingho	Halifax
Qu	Minxoush	Halifax

Only patients who reside in Halifax are displayed.

Table: Patient_information + City_information

SQL: Joining and Keys (1 - Many) (cont.)

- Examples:

```
SELECT pat_info.P_FirstName,  
pat_info.P_LastName, city_info.City_Name  
  
FROM pat_info  
  
LEFT JOIN city_info  
  
ON pat_info.City_id = city_info.City_id  
  
AND city_info.City_Name = 'Halifax'
```

SQL: Joining and Keys (1 - Many) (cont.)

P_id	P_FirstName	P_LastName	City_id	Phone_id	Kin_id	Client_id
1	Peter	Johnsons	1	4	6	14
2	Mike	Jackson	1	13	6	15
3	Sara	Henson	3	6	2	16
4	John	McDonnald	5	8	3	17
5	Michael	Robinson	1	13	6	18
6	William	Jordan	4	10	4	19
7	Susan	McKinsy	1	2	5	20
8	Mehdi	Kharrazi	2	1	9	21
9	John	McKinsy	1	9	10	22
10	John	McDonnald	3	18	7	23
11	Pat	Bentalar	7	25	8	24
12	Abraham	Lincoln	3	26	27	25
13	Brian	Adam	5	27	13	26
14	Catherin	Catholcy	7	28	15	33
15	Demi	Moore	12	29	23	34
16	Ebi	Farahanzadeh	11	30	26	42

City_id	City_Name
1	Halifax
2	Vancouver
3	Toronto
4	Montreal
5	Quebec
6	Winnipeg
7	Calgary
8	Sydney
9	New York
10	Los Angeles
11	Chicago
12	Boston

MySQL Query Browser - kharrazi@flame.cs.dal.ca:3306 / kharrazi

File Edit View Query Script Tools Window Help

Transaction Explain Compare

Resultset 1

```

1 SELECT pat_info.P_FirstName, pat_info.P_LastName, city_info.City_Name
2 FROM pat_info
3 LEFT JOIN city_info
4 ON pat_info.City_id = city_info.City_id
5 AND city_info.City_Name = 'Halifax'
    
```

P_FirstName	P_LastName	City_Name
Peter	Johnsons	Halifax
Mike	Jackson	Halifax
Sara	Henson	NULL
John	McDonnald	NULL
Michael	Robinson	Halifax
William	Jordan	NULL
Susan	McKinsy	Halifax
Mehdi	Kharrazi	NULL
John	McKinsy	Halifax
John	McDonnald	NULL

LEFT IS COMPLETE

Patients from all cities are displayed (LEFT SIDE) but the city for those who are not from HALIFAX is shown 'Null' (not displayed).

Table: Patient_information + City_information

SQL: Joining and Keys (1 - Many) (cont.)

- Examples:

```
SELECT pat_info.P_FirstName,  
pat_info.P_LastName, city_info.City_Name  
  
FROM pat_info  
  
RIGHT JOIN city_info  
  
ON pat_info.City_id = city_info.City_id  
  
AND city_info.City_Name = 'Halifax'
```

SQL: Joining and Keys (1 - Many) (cont.)

P_id	P_FirstName	P_LastName	City_id	Phone_id	Kin_id	Client_id
1	Peter	Johnsons	1	4	6	14
2	Mike	Jackson	1	13	6	15
3	Sara	Henson	3	6	2	16
4	John	McDonnald	5	8	3	17
5	Michael	Robinson	1	13	6	18
6	William	Jordan	4	10	4	19
7	Susan	McKinsy	1	2	5	20
8	Mehdi	Kharazi	2	1	9	21
9	John	McKinsy	1	9	10	22
10	John	McDonnald	3	18	7	23
11	Pat	Bentalar	7	25	8	24
12	Abraham	Lincoln	3	26	27	25
13	Brian	Adam	5	27	13	26
14	Catherin	Catholcy	7	28	15	33
15	Demi	Moore	12	29	23	34
16	Ebi	Farahanzadeh	11	30	26	42

City_id	City_Name
1	Halifax
2	Vancouver
3	Toronto
4	Montreal
5	Quebec
6	Winnipig
7	Calgary
8	Sydney
9	New York
10	Los Angeles
11	Chicago
12	Boston

Resultset 1

SQL Query Area

```

1 SELECT pat_info.P_FirstName, pat_info.P_LastName, city_info.City_Name
2 FROM pat_info
3 RIGHT JOIN city_info
4 ON pat_info.City_id = city_info.City_id
5 AND city_info.City_Name = 'Halifax'
        
```

P_FirstName	P_LastName	City_Name
Sam	Hill	Halifax
Robin	Carrier	Halifax
Peter	Yu	Halifax
Bill	Clinton	Halifax
Woo	Xingho	Halifax
Qu	Minxoush	Halifax
NULL	NULL	Vancouver
NULL	NULL	Toronto
NULL	NULL	Montreal
NULL	NULL	Quebec
NULL	NULL	Winnipig
NULL	NULL	Calgary
NULL	NULL	Sydney
NULL	NULL	New York
NULL	NULL	Los Angeles
NULL	NULL	Chicago
NULL	NULL	Boston

All cities (RIGHT SIDE) are shown but only patients from HALIFAX are listed and other patients from other cities are indicated 'Null'

RIGHT IS COMPLETE

Table: Patient_information + City_information

SQL: Joining and Keys (Many - Many) (cont.)

- Syntax:

```
SELECT * FROM table1, table2, table3
WHERE table1.columnX = table2.columnX
AND    table2.columnY = table3.columnY
```

- Examples:

```
SELECT pat_info.P_id, pat_info.P_FirstName,
pat_info.P_LastName, doc_info.D_id,
doc_info.D_FirstName, doc_info.D_LastName

FROM    pat_info, pat_doc_relate, doc_info

WHERE   pat_info.P_id = pat_doc_relate.P_id

AND    doc_info.D_id = pat_doc_relate.D_id
```

SQL: Joining and Keys (Many - Many) (cont.)

P_id	P_FirstName	P_LastName	City_id	Phone_id	Kin_id	Client_id
1	Peter	Johnsons	1	4	6	14
2	Mike	Jackson	1	13	6	15
3	Sara	Henson	3	6	2	16
4	John	McDonald	5	8	3	17
5	Michael	Robinson	1	13	6	18
6	William	Jordan	4	10	4	19
7	Susan	McKinsy	1	2	5	20
8	Mehdi	Kharrazi	2	1	9	21
9	John	McKinsy	1	9	10	22
10	John	McDonald	3	18	7	23
11	Pat	Bentatar	7	25	8	24
12	Abraham	Lincoln	3	26	27	25
13	Brian	Adam	5	27	13	26
14	Catherin	Catholicy	7	28	15	33
15	Demi	Moore				
16	Ebi	Farahanzadeh				

P_id	D_id
1	4
1	5
1	6
2	1
3	9
4	1
4	2
4	10
5	5
6	2
7	6

D_id	D_FirstName	D_LastName
1	Hadi	Kharrazi
2	John	McDonald
3	Robin	Dorby
4	Marry	Johnson
5	Yu	Lee
6	Nancy	McLssac
7	John	Peterson
8	Jane	Peterson
9	Susan	Waterloo
10	Samuel	Jackson

MySQL Query Browser - kharrazi@flame.cs.dal.ca:3306 / kharrazi

File Edit View Query Script Tools Window Help

Transaction Explain Compare

Resultset 1

SQL Query Area

```

1 SELECT pat_info.P_id, pat_info.P_FirstName, pat_info.P_LastName,
2       doc_info.D_id, doc_info.D_FirstName, doc_info.D_LastName
3 FROM pat_info, pat_doc_relate, doc_info
4 WHERE pat_info.P_id = pat_doc_relate.P_id
5 AND doc_info.D_id = pat_doc_relate.D_id
    
```

P_id	P_FirstName	P_LastName	D_id	D_FirstName	D_LastName
1	Peter	Johnsons	4	Marry	Johnson
1	Peter	Johnsons	5	Yu	Lee
1	Peter	Johnsons	6	Nancy	McLssac
2	Mike	Jackson	1	Hadi	Kharrazi
3	Sara	Henson	9	Susan	Waterloo
4	John	McDonald	1	Hadi	Kharrazi
4	John	McDonald	2	John	McDonald
4	John	McDonald	10	Samuel	Jackson
5	Michael	Robinson	5	Yu	Lee
6	William	Jordan	2	John	McDonald
7	Susan	McKinsy	6	Nancy	McLssac
8	Mehdi	Kharrazi	3	Robin	Dorby
8	Mehdi	Kharrazi	7	John	Peterson

SQL: Joining and Keys (Many - Many) (cont.)

- Examples:

```
SELECT pat_info.P_id, pat_info.P_FirstName,  
        pat_info.P_LastName, doc_info.D_id,  
        doc_info.D_FirstName, doc_info.D_LastName  
  
FROM   pat_info, pat_doc_relate, doc_info  
  
WHERE  pat_info.P_id = pat_doc_relate.P_id  
  
AND    doc_info.D_id = pat_doc_relate.D_id  
  
AND    pat_info.P_FirstName = 'Peter'
```

SQL: Joining and Keys (Many - Many) (cont.)

MySQL Query Browser - kharrazi@flame.cs.dal.ca:3306 / kharrazi

File Edit View Query Script Tools Window Help

Transaction Explain

Resultset 1

SQL Query Area

```

1 SELECT pat_info.P_id, pat_info.P_FirstName, pat_info.P_LastName,
2       doc_info.D_id, doc_info.D_FirstName, doc_info.D_LastName
3 FROM pat_info, pat_doc_relate, doc_info
4 WHERE pat_info.P_id = pat_doc_relate.P_id
5 AND doc_info.D_id = pat_doc_relate.D_id
6 AND pat_info.P_FirstName = 'Peter'

```

P_id	P_FirstName	P_LastName	D_id	D_FirstName	D_LastName
1	Peter	Johnsons	4	Marry	Johnson
1	Peter	Johnsons	5	Yu	Lee
1	Peter	Johnsons	6	Nancy	McIssac
47	Peter	Yu	1	Hadi	Kharrazi
47	Peter	Yu	7	John	Peterson

9. SQL: GROUP BY & HAVING

- Syntax:

```
SELECT column, function(column)
FROM table
GROUP BY column
```

- Examples:

```
SELECT    lab_info.P_id,
          AVG (lab_info.RBC)

FROM      lab_info
GROUP BY lab_info.P_id
```

SQL: GROUP BY & HAVING (cont.)

Resultset 1

SQL Query Area

```
1 SELECT * FROM pat_info p
```

P_id	P_FirstNa...	P_LastName	City id
1	Peter	Johnsons	1
2	Mike	Jackson	1
3	Sara	Henson	3
4	John	McDonald	5
5	Michael	Robinson	1
6	William	Jordan	4
7	Susan	McKinsy	1
8	Mehdi	Kharrazi	2
9	John	McKinsy	1
10	John	McDonald	3
11	Pat	Bentatar	7
12	Abraham	Lincoln	3
13	Brian	Adam	5
14	Catherin	Catholicy	7
15	Demi	Moore	12
16	Ebi	Farahanzad...	11

Table: Patient_Information

Resultset 2

SQL Query Area

```
1 SELECT * FROM lab_info l
```

L_id	P_id	RBC	WBC	PLT	Date
1	1	4.8	8.1	248000	2005-05-09
2	1	4.3	10.9	325000	2005-05-19
3	1	5.6	11	129000	2005-06-03
4	1	3.4	6.9	399000	2005-06-24
5	1	7.4	10.2	198000	2005-09-10
6	2	5.2	13	452000	2005-04-10
7	2	4.5	12.5	525000	2005-04-12
8	2	6.7	11.9	540000	2005-04-29
9	2	5.6	12.9	478000	2005-05-12
10	2	5.7	13.8	433000	2005-08-03
11	3	3.4	5.7	320000	2005-07-08
12	3	3.3	5.9	327000	2005-07-16
13	4	5	8.9	195000	2005-08-30
14	5	5.8	9	275000	2005-03-09
15	5	5.9	9.8	287000	2005-06-19
16	5	6	9.9	299000	2005-09-19
17	6	7.2	6.1	324000	2005-02-14
18	6	6.9	6	365000	2005-03-14

Table: LAB_Information

SQL: **GROUP BY** & **HAVING** (cont.)

The left window shows the following SQL query:

```

1 SELECT lab_info.P_id,
2        AVG (lab_info.RBC)
3
4 FROM   lab_info
5 GROUP BY lab_info.P_id

```

The right window shows the following SQL query:

```

1 SELECT * FROM lab_info l

```

The right window also displays a table with the following data:

L_id	P_id	RBC	WBC	PLT	Date
1	1	4.8	8.1	248000	2005-05-09
2	1	4.3	10.9	325000	2005-05-19
3	1	5.6	11	129000	2005-06-03
4	1	3.4	6.9	399000	2005-06-24
5	1	7.4	10.2	198000	2005-09-10
6	2	5.2	13	452000	2005-04-10
7	2	4.5	12.5	525000	2005-04-12
8	2	6.7	11.9	540000	2005-04-29
9	2	5.6	12.9	478000	2005-05-12
10	2	5.7	13.8	433000	2005-08-03
11	3	3.4	5.7	320000	2005-07-08
12	3	3.3	5.9	327000	2005-07-16
13	4	5	8.9	195000	2005-08-30
14	5	5.8	9	275000	2005-03-09
15	5	5.9	9.8	287000	2005-06-19
16	5	6	9.9	299000	2005-09-19
17	6	7.2	6.1	324000	2005-02-14
18	6	6.9	6	365000	2005-03-14

The left window shows the result of the first query:

P_id	AVG (lab_info.RBC)
1	5.1000000953674
2	5.5399998664856
3	3.3500000238419
4	5
5	5.9000000953674
6	6.8499999046326
7	3.6000000238419
8	5.1499998569489
9	4.9500000476837
10	4.4333333969116

The right window shows the result of the second query, with rows 1, 2, 3, 4, 5, 11, 12, and 13 highlighted in green. Colored arrows indicate the mapping between the HAVING clause in the right query and the filtered rows in the right resultset, and from those rows to the corresponding rows in the left resultset.

Table: LAB_Information

SQL: **GROUP BY & HAVING** (cont.)

- Syntax:

```
SELECT column, function(column)
FROM table
GROUP BY column
HAVING function(column) condition value
```

- Examples:

```
SELECT    lab_info.P_id,
           AVG (lab_info.RBC)

FROM      lab_info
GROUP BY  lab_info.P_id
HAVING    AVG (lab_info.RBC)>5
```

SQL: **GROUP BY & HAVING** (cont.)

Resultset 1 x Resultset 2

SQL Query Area

```

1 SELECT lab_info.P_id,
2       AVG (lab_info.RBC)
3
4 FROM lab_info
5 GROUP BY lab_info.P_id

```

P_id	AVG (lab_info.RBC)
1	5.1000000953674
2	5.5399998664856
3	3.3500000238419
4	5
5	5.9000000953674
6	6.8499999046326
7	3.6000000238419
8	5.1499998569489
9	4.9500000476837
10	4.4333333969116

Table: LAB_Information

Resultset 1 x Resultset 2

SQL Query Area

```

1 SELECT lab_info.P_id,
2       AVG (lab_info.RBC)
3
4 FROM lab_info
5 GROUP BY lab_info.P id
6 HAVING AVG (lab_info.RBC)>5

```

P_id	AVG (lab_info...
1	5.1000000953...
2	5.5399998664...
5	5.9000000953...
6	6.8499999046...
8	5.1499998569...

Table: LAB_Information

10. SQL: **Functions**

- There are several basic types and categories of functions in SQL. The basic types of functions are:
 - **Aggregate Functions:** Aggregate functions operate against a collection of values, but return a single value.

(Note: If used among many other expressions in the item list of a SELECT statement, the SELECT must have a GROUP BY clause!)
 - **Scalar functions:** Scalar functions operate against a single value, and return a single value based on the input value.

SQL: *Functions* (cont.)

- **Aggregate functions:**

Function	Description
AVG (column)	Returns the average value of a column
COUNT (column)	Returns the number of rows (without a NULL value) of a column
COUNT (*)	Returns the number of selected rows
COUNT (DISTINCT column)	Returns the number of distinct results
MAX (column)	Returns the highest value of a column
MIN (column)	Returns the lowest value of a column
SUM (column)	Returns the total sum of a column

SQL: *Functions* (cont.)

- **Scalar functions:**

Function	Description
UCASE (c)	Converts a field to upper case
LCASE (c)	Converts a field to lower case
MID (c, start[,end])	Extract characters from a text field
LEN (c)	Returns the length of a text field
INSTR (c)	Returns the numeric position of a named character within a text
LEFT (c,number_of_char)	Return the left part of a text field requested
RIGHT (c,number_of_char)	Return the right part of a text field requested
ROUND (c,decimals)	Rounds a numeric field to the number of decimals specified
MOD (x,y)	Returns the remainder of a division operation
NOW ()	Returns the current system date
FORMAT (c,format)	Changes the way a field is displayed

SQL: *Functions* (cont.)

- Syntax:

```
SELECT function(column) FROM table
```

- Examples:

```
SELECT  AVG (lab_info.RBC)
FROM    lab_info
WHERE   lab_info.P_id = 1;
```

```
SELECT  ROUND (AVG (lab_info.RBC), 2)
FROM    lab_info
WHERE   lab_info.P_id = 1;
```

SQL: Functions (cont.) (AVG, ROUND, SUM)

The screenshot displays a database application interface with a table and three SQL query resultsets. Red boxes and arrows highlight the data and results of the queries.

Table: LAB_Information

L_id	P_id	RBC	WBC	PLT	Date
1	1	4.8	8.1	248000	2005-05-09
2	1	4.3	10.9	325000	2005-05-19
3	1	5.6	11	123000	2005-06-03
4	1	3.4	6.9	399000	2005-06-24
5	1	7.4	10.2	198000	2005-09-10
6	2	5.2	13	452000	2005-04-10
7	2	4.5	12.5	525000	2005-04-12
8	2	6.7	11.9	510000	2005-04-28
9	2	5.6	12.9		
10	2	5.7	13.8		
11	3	3.4	5.7		
12	3	3.3	5.9		
13	4	5	8.9		
14	5	5.8	9		
15	5	5.9	9.8		
16	5	6	9.9		
17	6	7.2	6.1		
18	6	6.9	6		

Query 1: AVG

```
SELECT AVG (lab_info.RBC)
FROM lab_info
WHERE lab_info.P_id = 1;
```

Result: 5.1000000953674

Query 2: SUM

```
SELECT SUM(lab_info.RBC)
FROM lab_info
WHERE lab_info.P_id = 1;
```

Result: 25.500000476...

Query 3: ROUND

```
SELECT ROUND(AVG (lab_info.RBC), 2)
FROM lab_info
WHERE lab_info.P_id = 1;
```

Result: 5.10

SQL: *Functions (cont.)* (COUNT)

Resultset 2 x

SQL Query Area

```
1 SELECT * FROM lab_info l
```

L_id	P_id	RBC	WBC	PLT	Date
1	1	4.8	8.1	248000	2005-05-09
2	1	4.3	10.9	325000	2005-05-19
3	1	5.6	11	123000	2005-08-03
4	1	3.4	6.9	399000	2005-06-24
5	1	7.4	10.2	198000	2005-09-10
6	2	5.2	13	452000	2005-04-10
7	2	4.5	12.5	525000	2005-04-12
8	2	6.7	11.9	540000	2005-04-29
9	2	5.6	12.9	478000	2005-05-12
10	2	5.7	13.8	433000	2005-08-03
11	3	3.4	5.7	320000	2005-07-08
12	3	3.3	5.9	327000	2005-07-16
13	4	5	8.9	195000	2005-08-30
14	5	5.8	9	275000	2005-03-09
15	5	5.9	9.8	287000	2005-06-19
16	5	6	9.9	299000	2005-09-19
17	6	7.2	6.1	324000	2005-02-14
18	6	6.9	6	365000	2005-03-14

Resultset 1 x

SQL Query Area

```
1 SELECT COUNT(*)
2 FROM lab_info
3 WHERE lab_info.P_id = 1;
4
```

COUNT(*)
5

Table: LAB_Information

SQL: Functions (cont.) (MIN, MAX)

L_id	P_id	RBC	WBC	PLT	Date
1	1	4.8	8.1	248000	2005-05-09
2	1	4.3	10.9	325000	2005-05-19
3	1	5.6	11	123000	2005-08-03
4	1	3.4	6.9	399000	2005-06-24
5	1	7.4	10.2	198000	2005-09-10
6	2	5.2	13	452000	2005-04-10
7	2	4.5	12.5	525000	2005-04-12
8	2	6.7	11.9	540000	2005-04-29
9	2	5.6	12.9	478000	2005-05-12
10	2	5.7	13.8	433000	2005-08-03
11	3	3.4	5.7	320000	2005-07-08
12	3	3.3	5.9	327000	2005-07-16
13	4	5	8.9	195000	2005-08-30
14	5	5.8	9	275000	2005-03-09
15	5	5.9	9.8	287000	2005-06-19
16	5	6	9.9	299000	2005-09-19
17	6	7.2	6.1	324000	2005-02-14
18	6	6.9	6	365000	2005-03-14

Table: LAB_Information

SQL Query Area
1 SELECT MAX(lab_info.RBC)
2 FROM lab_info
3 WHERE lab_info.P_id = 1;
4

MAX(lab_info.RBC)

7.4000000953674

SQL Query Area
1 SELECT MIN(lab_info.RBC)
2 FROM lab_info
3 WHERE lab_info.P_id = 1;
4

MIN(lab_info.RBC)

3.4000000953...

SQL: *Functions (cont.)* (UCASE, LCASE)

Resultset 1 x Resultset 2 Resu

SQL Query Area

```

1 SELECT UCASE(pat_info.P_FirstName)
2 FROM pat_info
3

```

UCASE(pat_info.P_FirstName)
PETER
MIKE
SARA
JOHN
MICHAEL
WILLIAM
SUSAN
MEHDI
JOHN
JOHN
PAT
ABRAHAM
BRIAN
CATHERIN

Resultset 1 x Resultset 2 Resultse

SQL Query Area

```

1 SELECT LCASE(pat_info.P_FirstName)
2 FROM pat_info
3

```

LCASE(pat_i..
peter
mike
sara
john
michael
william
susan
mehdi
john
john
pat
abraham
brian
catherin
demi

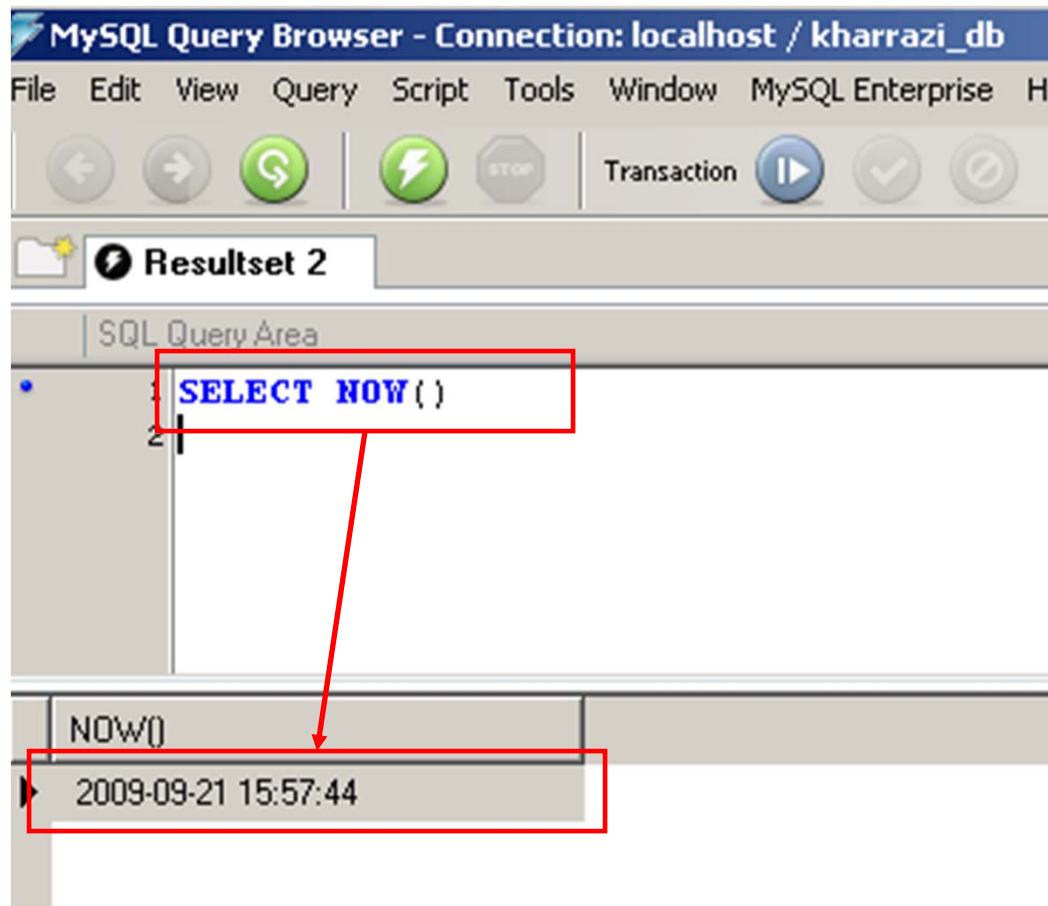
SQL: *Functions (cont.)* (LEFT, RIGHT)

The diagram illustrates the LEFT and RIGHT string functions in SQL. It shows two query windows. The left window shows a query using `LEFT(pat_info.P_FirstName, 3)` which returns 'Pet' for the first row. The right window shows a query using `RIGHT(pat_info.P_FirstName, 3)` which returns 'ter' for the first row. Red arrows connect the 'Pet' and 'ter' results to the word 'Peter'. Below this, the word 'Peter' is shown with indices 1-5 above each letter and 5-1 below each letter, with arrows indicating the direction of the LEFT and RIGHT functions.

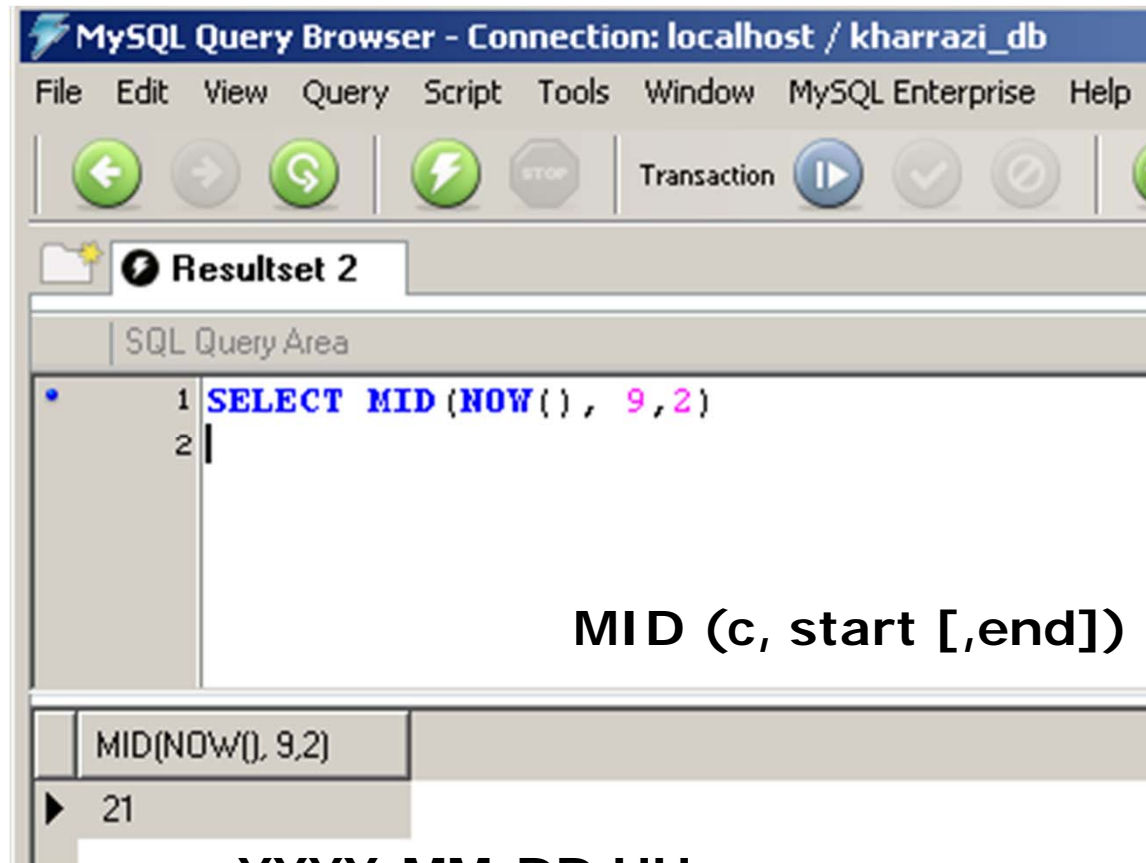
LEFT(pat_info.P_FirstName, 3)	RIGHT(pat_info.P_FirstName, 3)
Pet	ter
Mik	ike
Sar	ara
Joh	ohn
Mic	ael
Wil	iam
Sus	san
Meh	hdi
Joh	ohn
Joh	ohn
Pat	Pat
Abr	ham

Peter
 1 2 3 4 5
 5 4 3 2 1
 Left →
 ← Right

SQL: *Functions* (cont.) (NOW)



YYYY-MM-DD HH:mm:ss

SQL: Functions (cont.) (MID)

YYYY-MM-DD HH:mm:ss
 2009-09-21 ...
 12345678**9**01234..
 →

Summary

- SQL: CREATE (Database, Table, and Index)
- SQL: TRUNCATE (Table)
- SQL: DROP (Database, Table, and Index)
- SQL: ALTER (Database, Table, and Index)
- SQL: INSERT
- SQL: UPDATE
- SQL: DELETE
- SQL: Joining and Keys (Inner/Left/Right Join)
- SQL: GROUP BY & HAVING
- SQL: Functions

Next Session

- Database Design Process
- MySQL Installation
- MySQL Workbench
- MySQL Administration
- MySQL Migration

Exercise

- Please refer to the available text file in the slides section for this session on the course website:
- http://info510.com/core/public_page.php?page_name=slides