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.
`LONGBLOB`	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)`

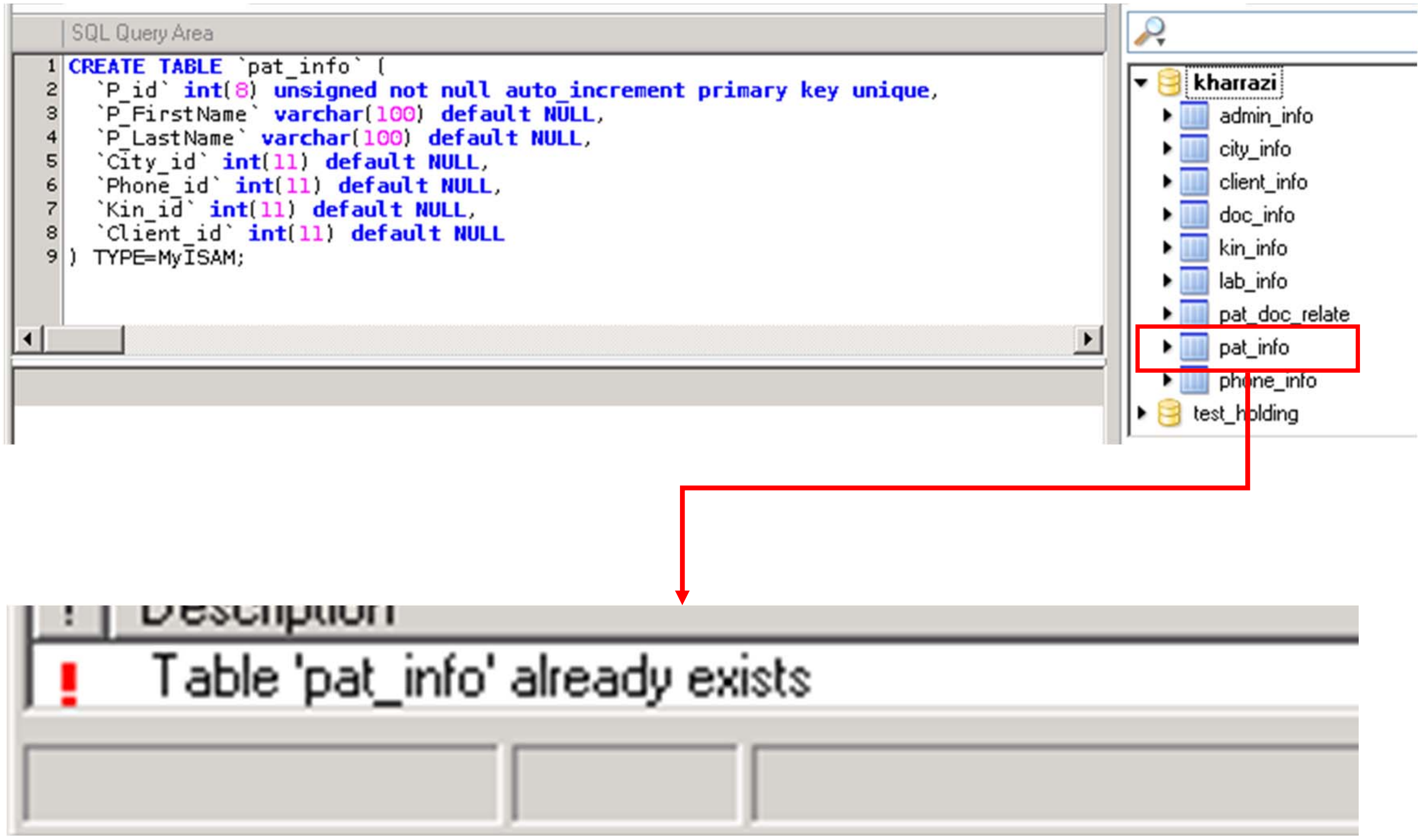
SQL: CREATE (cont.) (database, table)

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)

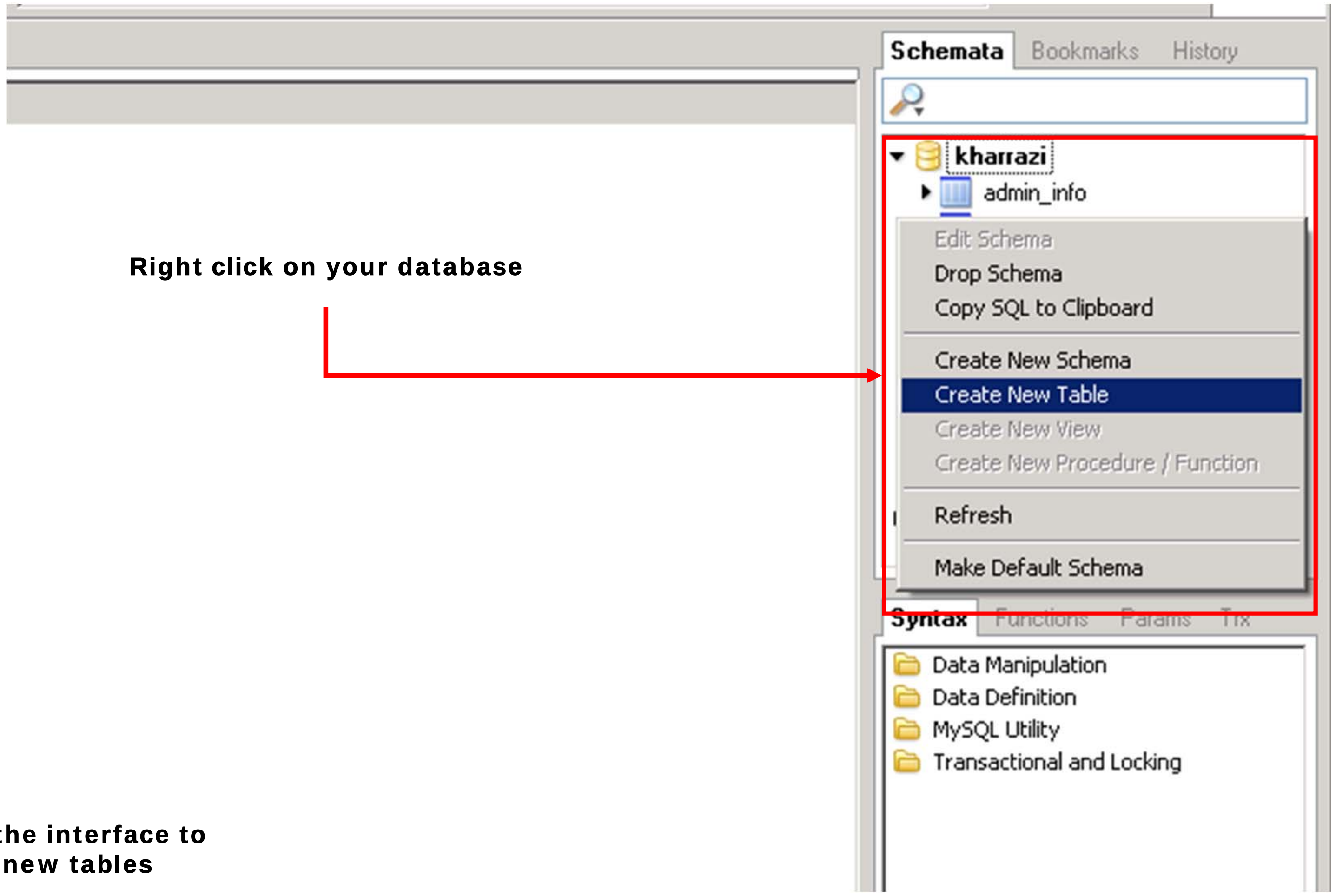


```
CREATE TABLE `pat_info` (
  `P_id` int(8) unsigned not null auto_increment primary key unique,
  `P_FirstName` varchar(100) default NULL,
  `P_LastName` varchar(100) default NULL,
  `City_id` int(11) default NULL,
  `Phone_id` int(11) default NULL,
  `Kin_id` int(11) default NULL,
  `Client_id` int(11)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

```
Table 'pat_info' already exists
```

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

Right click on your database



Using the interface to create new tables

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

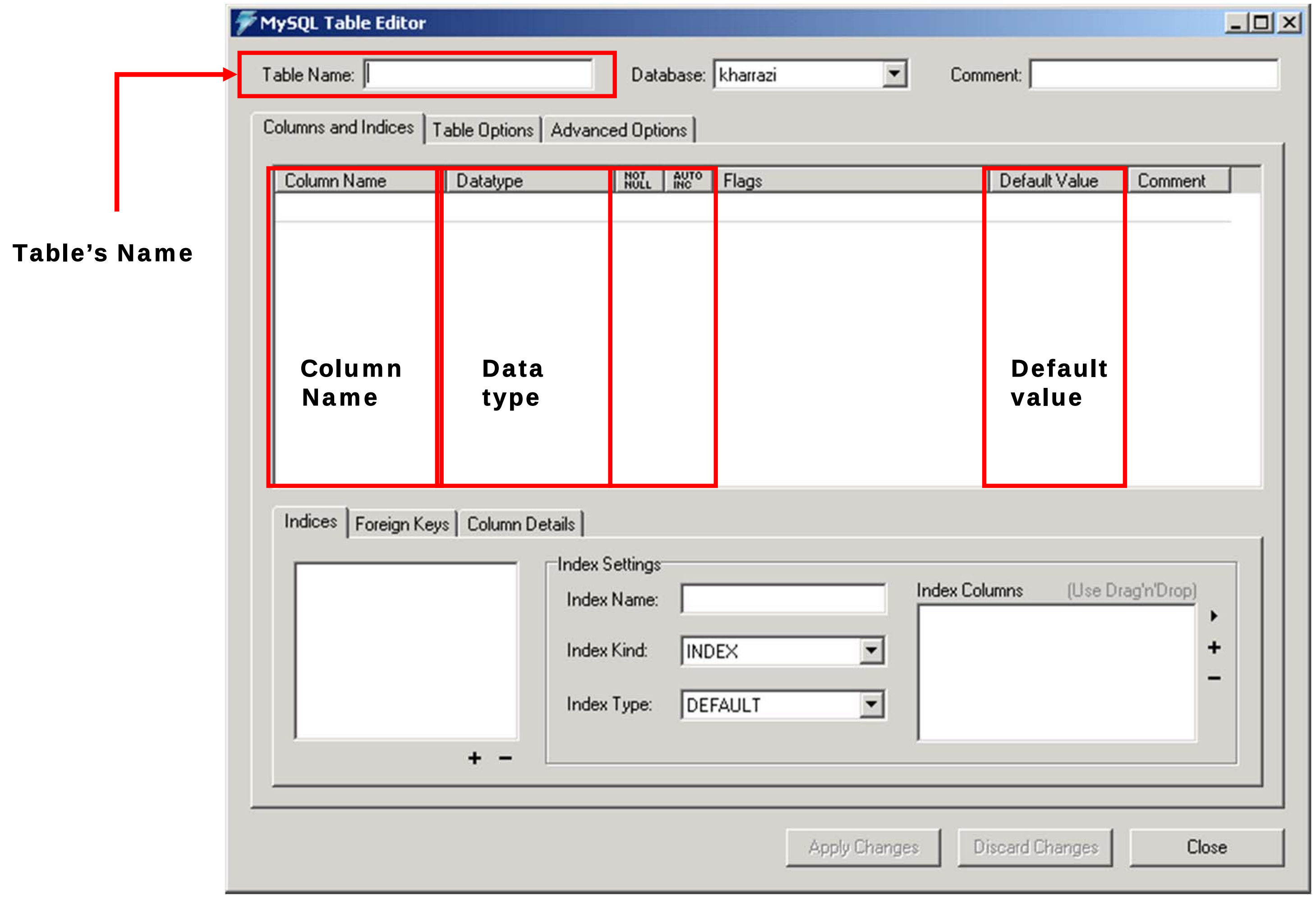


Table Name:

Database: kharrazi

Comment:

Index Name:

Index Kind: INDEX

Index Type: DEFAULT

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

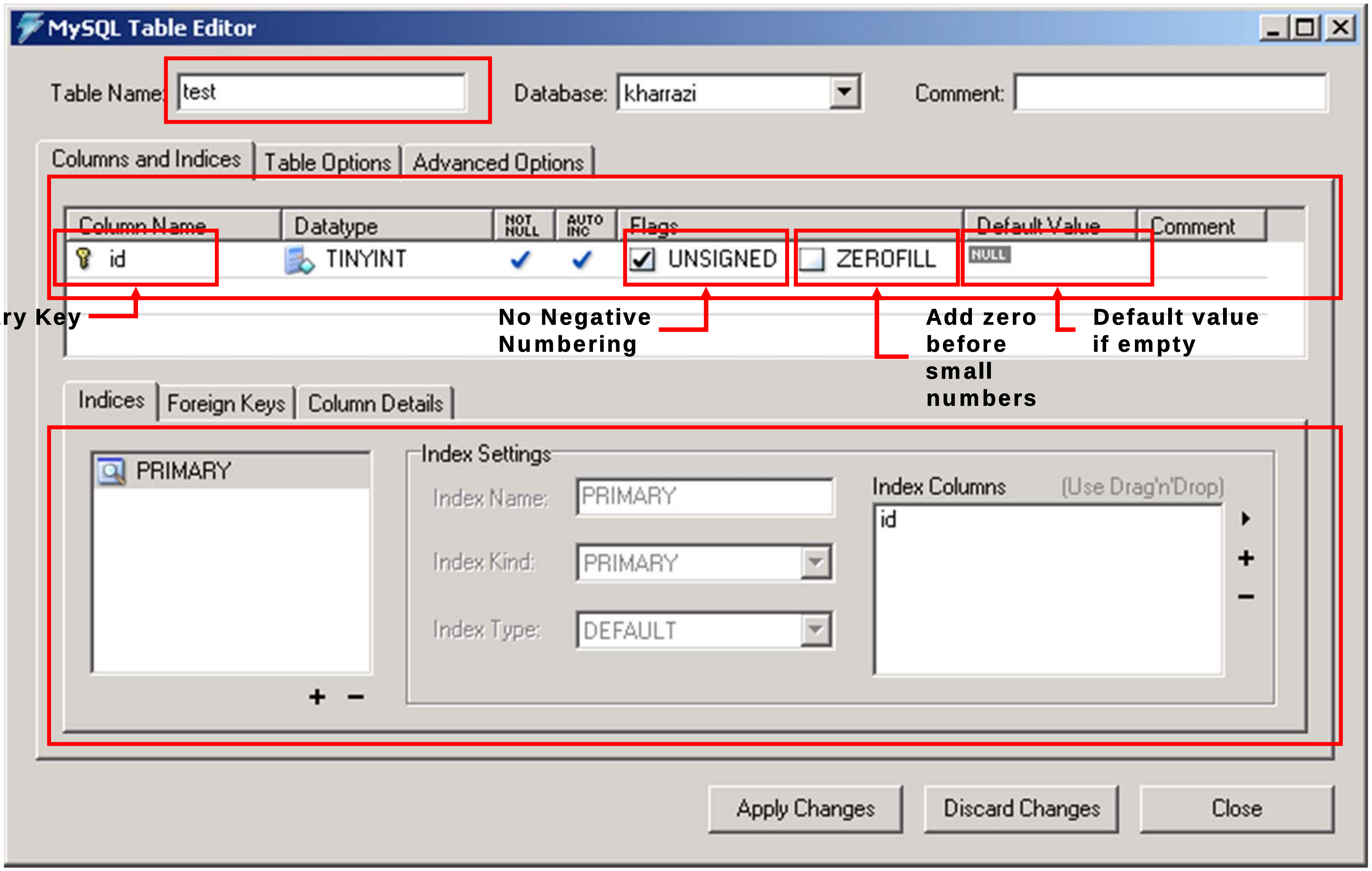


Table Name: test
Database: kharrazi
Comment:

Column Name	Datatype	NOT NULL	AUTO INC	Flags		Default Value	Comment
id	TINYINT	✓	✓	☑ UNSIGNED	☐ ZEROFILL	NULL	

PRIMARY

Index Settings
Index Name: PRIMARY
Index Kind: PRIMARY
Index Type: DEFAULT

Index Columns (Use Drag'n'Drop)
id

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

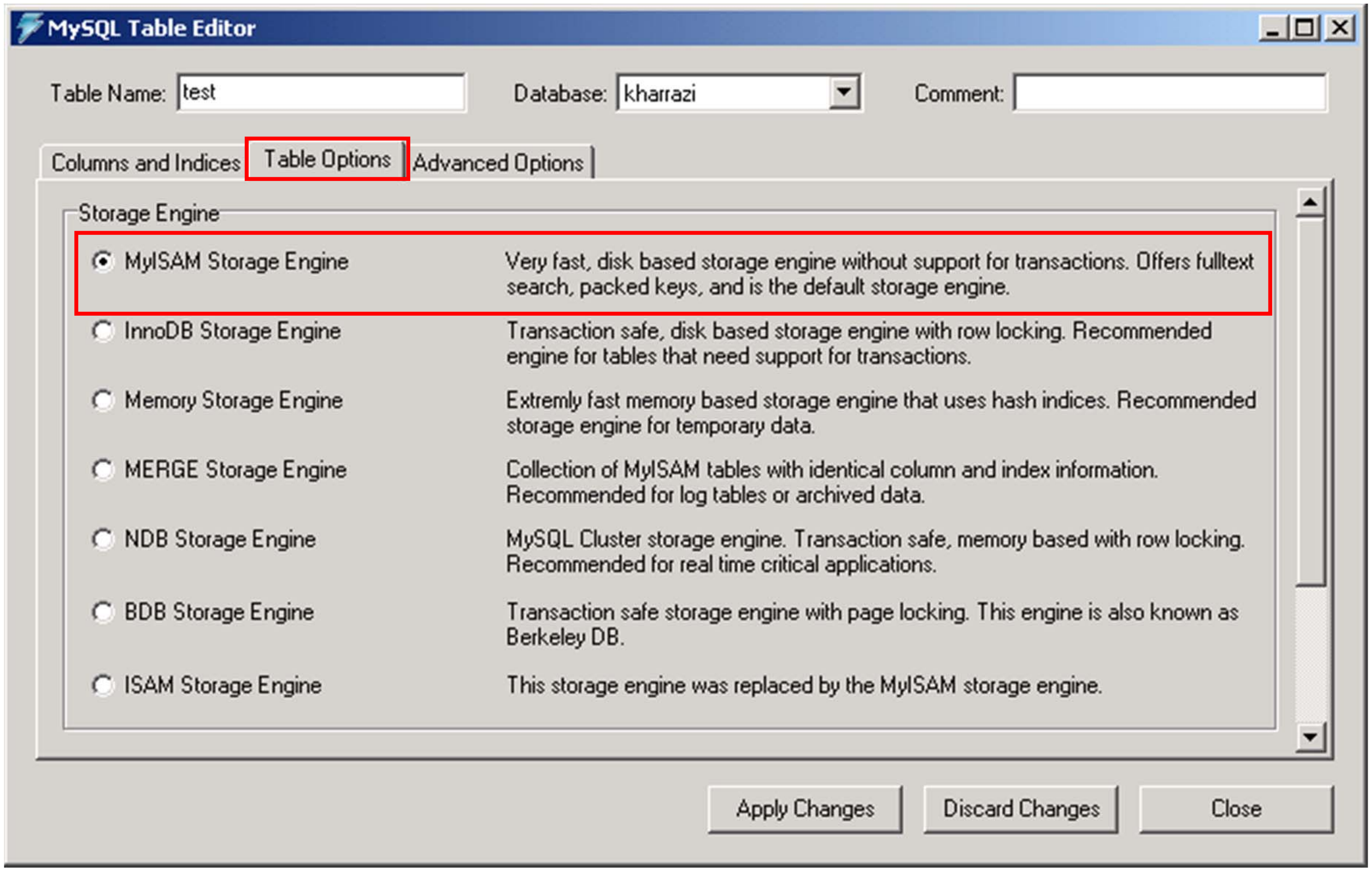


Table Name: test   Database: kharrazi   Comment:

Storage Engine

● MyISAM Storage Engine	Very fast, disk based storage engine without support for transactions. Offers fulltext search, packed keys, and is the default storage engine.
○ InnoDB Storage Engine	Transaction safe, disk based storage engine with row locking. Recommended engine for tables that need support for transactions.
○ Memory Storage Engine	Extremly fast memory based storage engine that uses hash indices. Recommended storage engine for temporary data.
○ MERGE Storage Engine	Collection of MyISAM tables with identical column and index information. Recommended for log tables or archived data.
○ NDB Storage Engine	MySQL Cluster storage engine. Transaction safe, memory based with row locking. Recommended for real time critical applications.
○ BDB Storage Engine	Transaction safe storage engine with page locking. This engine is also known as Berkeley DB.
○ ISAM Storage Engine	This storage engine was replaced by the MyISAM storage engine.

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

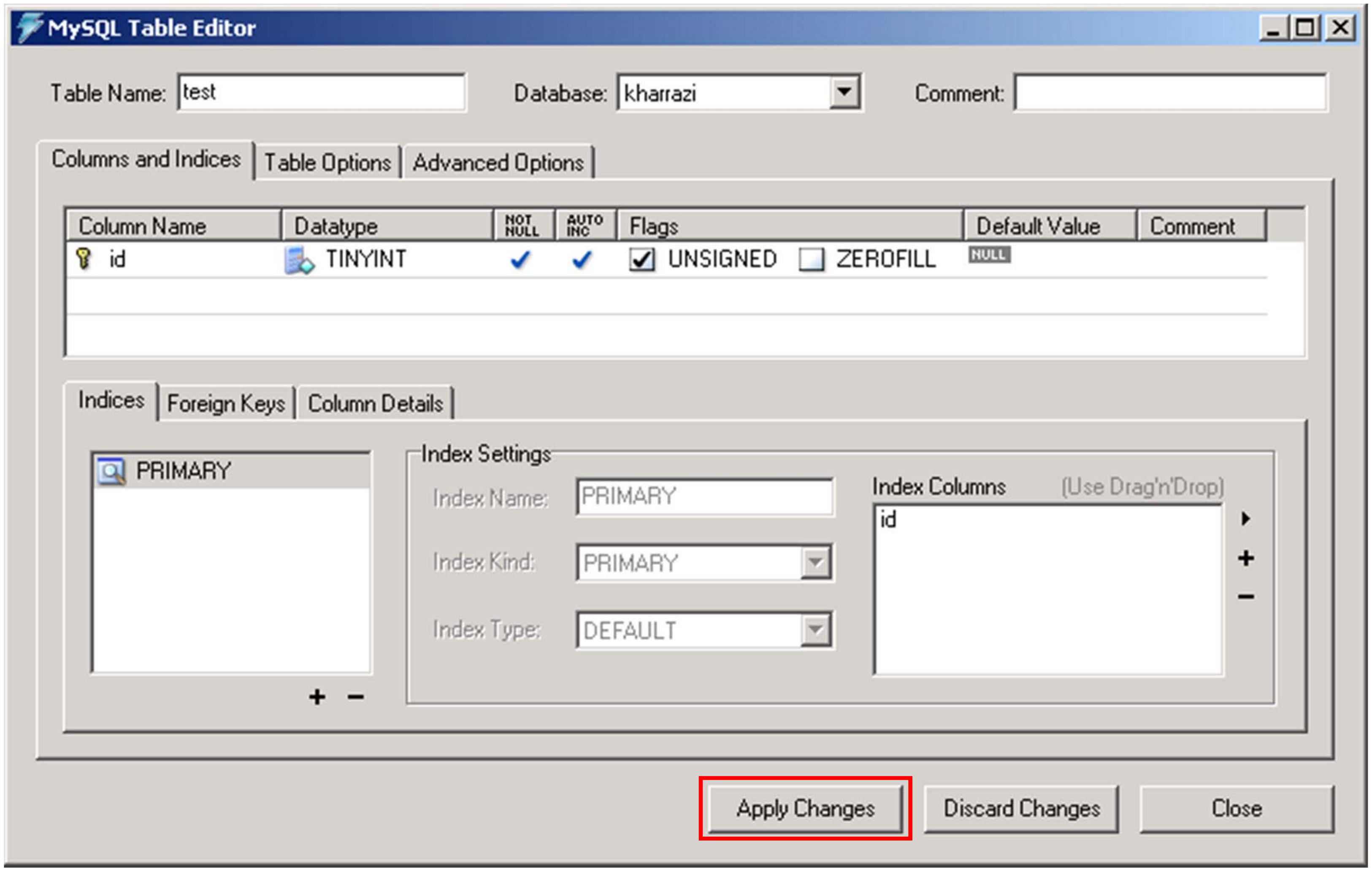

MySQL Table Editor

Table Name: test
Database: kharrazi
Comment:

Columns and Indices | Table Options | Advanced Options

Column Name	Datatype	NOT NULL	AUTO INC	Flags	Default Value	Comment
id	TINYINT	✓	✓	☑ UNSIGNED ☐ ZEROFILL	NULL	

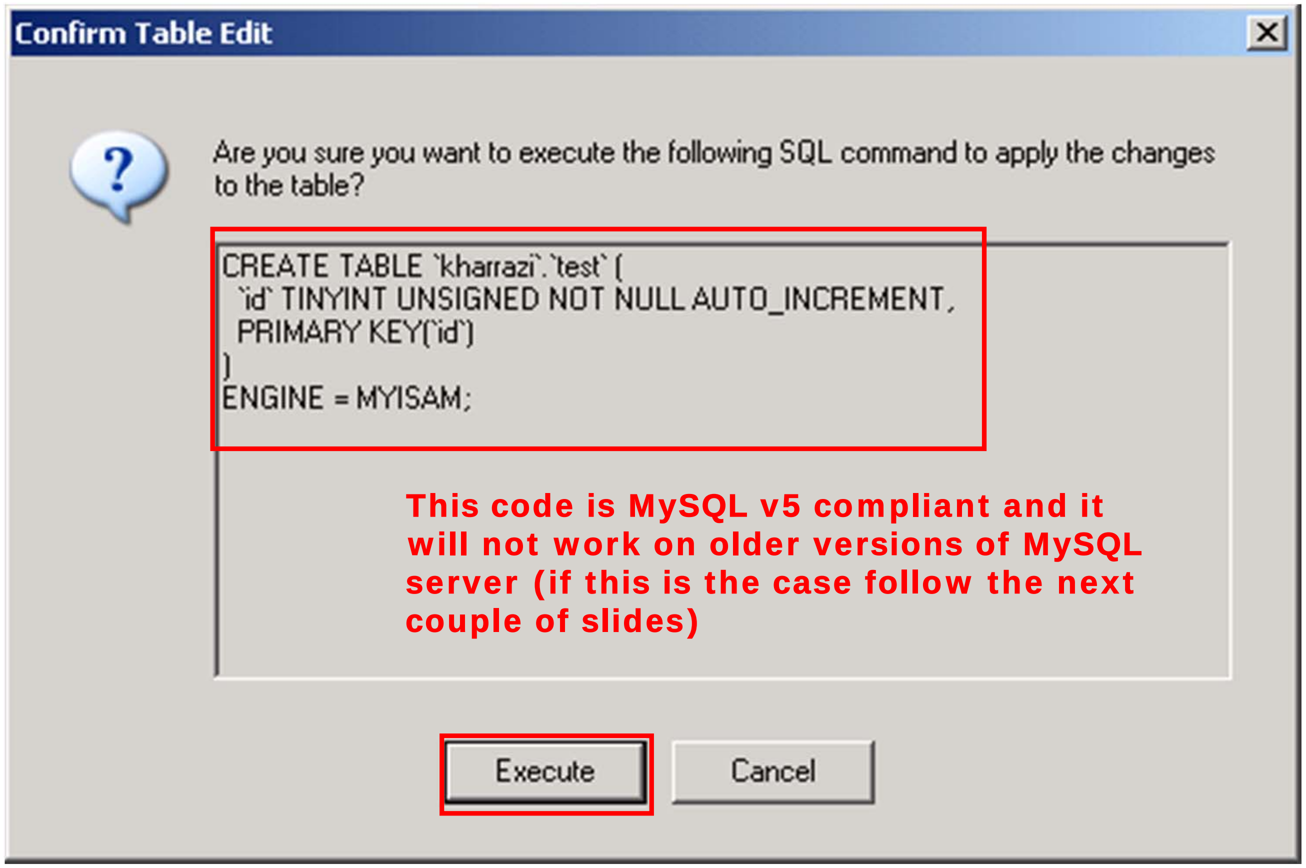
Indices | Foreign Keys | Column Details

PRIMARY

Index Settings
Index Name: PRIMARY
Index Kind: PRIMARY
Index Type: DEFAULT
Index Columns (Use Drag'n'Drop): id

Apply Changes | Discard Changes | Close

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

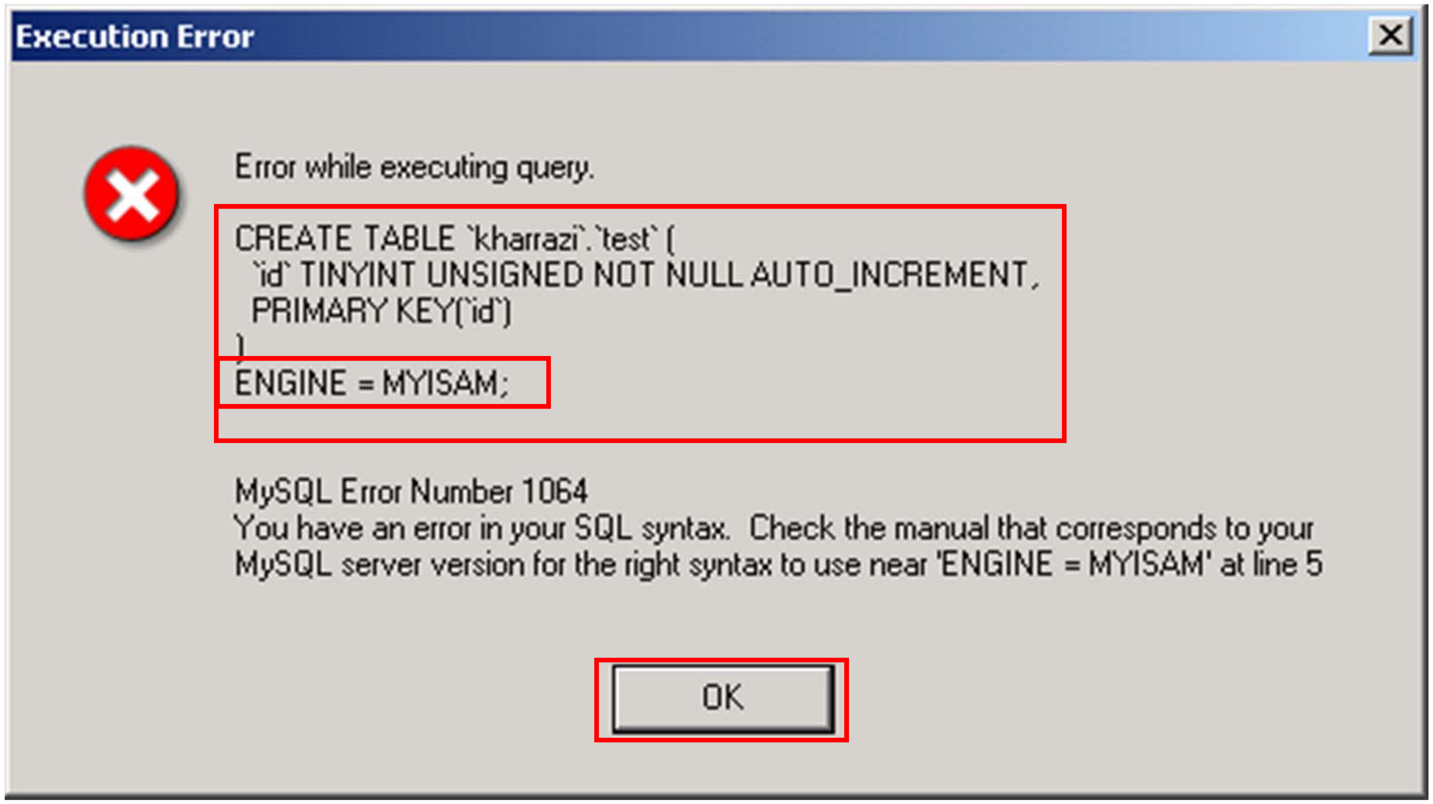


Are you sure you want to execute the following SQL command to apply the changes to the table?

```
CREATE TABLE `kharrazi`.`test` (
  `id` TINYINT UNSIGNED NOT NULL AUTO_INCREMENT,
  PRIMARY KEY(`id`)
)
ENGINE = MYISAM;
```

This code is MySQL v5 compliant and it will not work on older versions of MySQL server (if this is the case follow the next couple of slides)

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

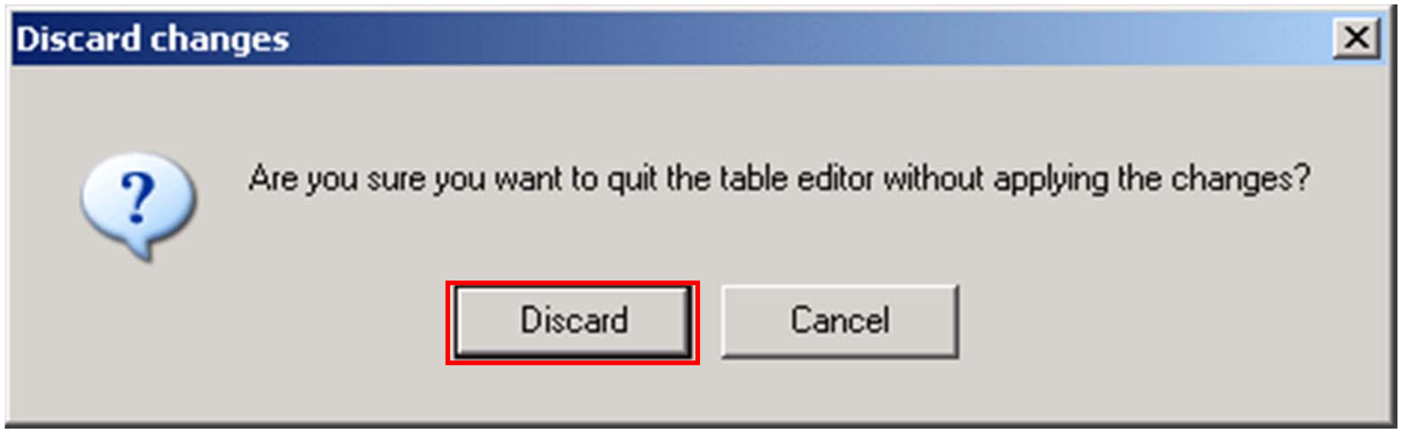


Error while executing query.

```
CREATE TABLE `kharrazi`.`test` (
  `id` TINYINT UNSIGNED NOT NULL AUTO_INCREMENT,
  PRIMARY KEY(`id`)
)
ENGINE = MYISAM;
```

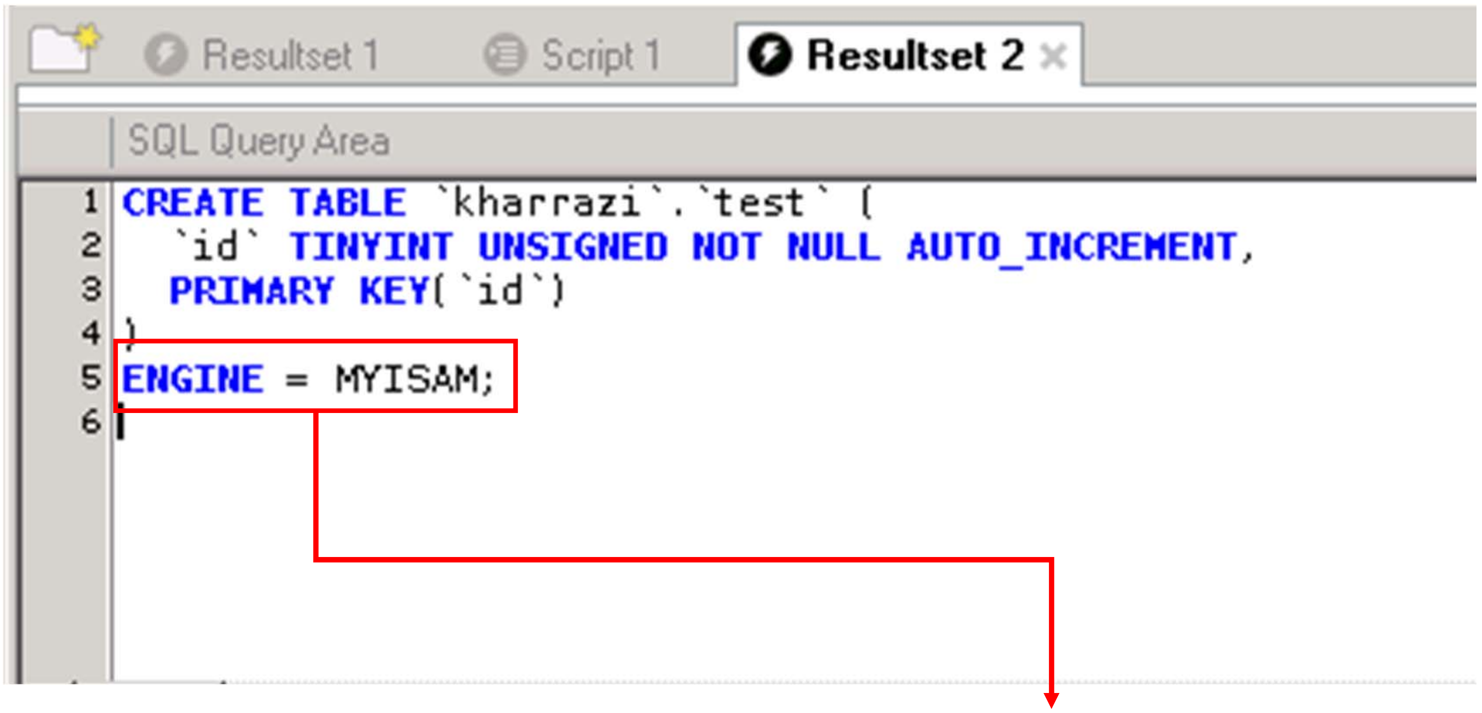
MySQL Error Number 1064
You have an error in your SQL syntax. Check the manual that corresponds to your MySQL server version for the right syntax to use near 'ENGINE = MYISAM' at line 5

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


Discard changes

Are you sure you want to quit the table editor without applying the changes?

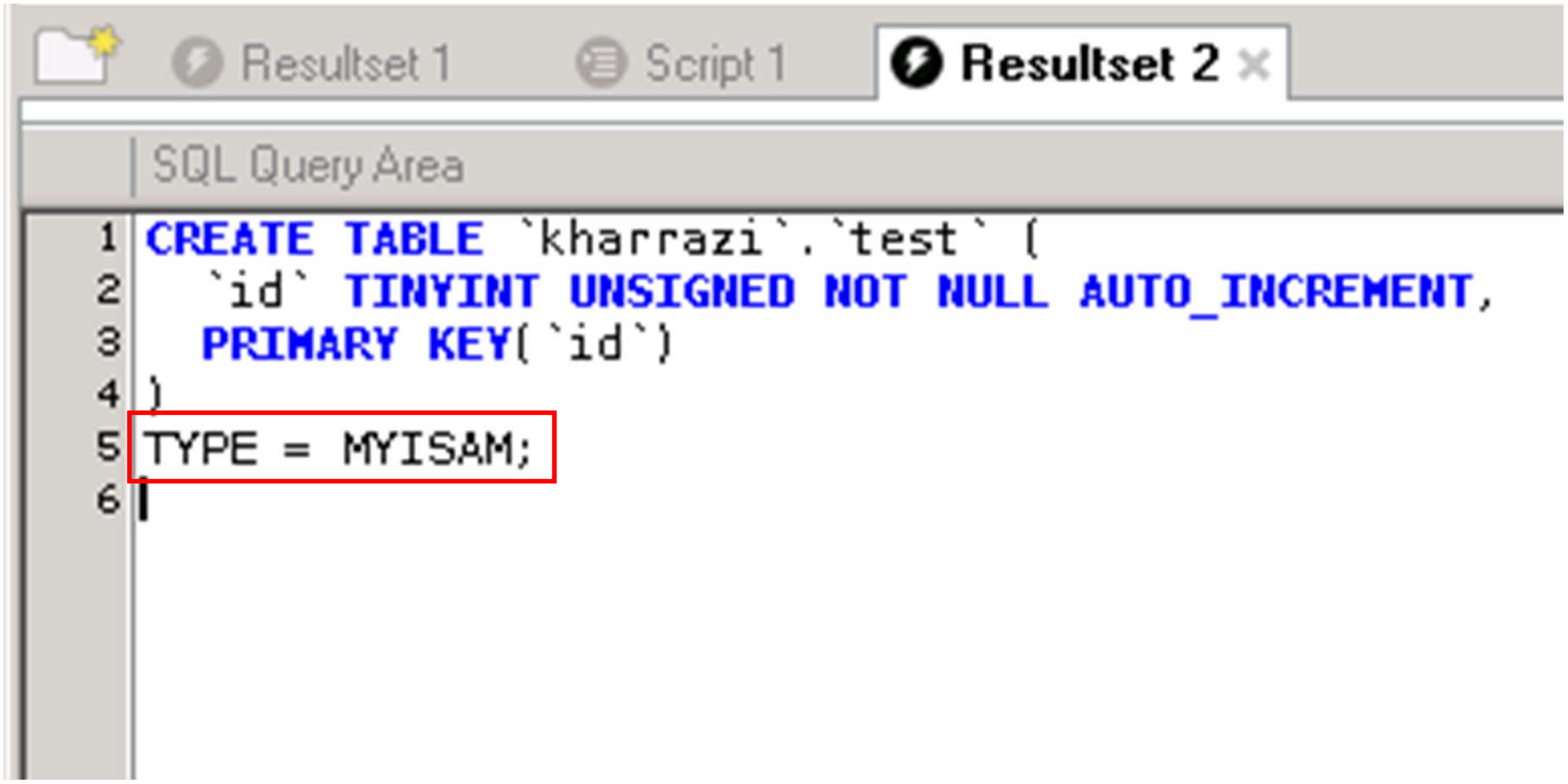
SQL: CREATE (cont.) (database, table)



```
CREATE TABLE `kharrazi`.`test` (
  `id` TINYINT UNSIGNED NOT NULL AUTO_INCREMENT,
  PRIMARY KEY(`id`)
)
ENGINE = MYISAM;
```

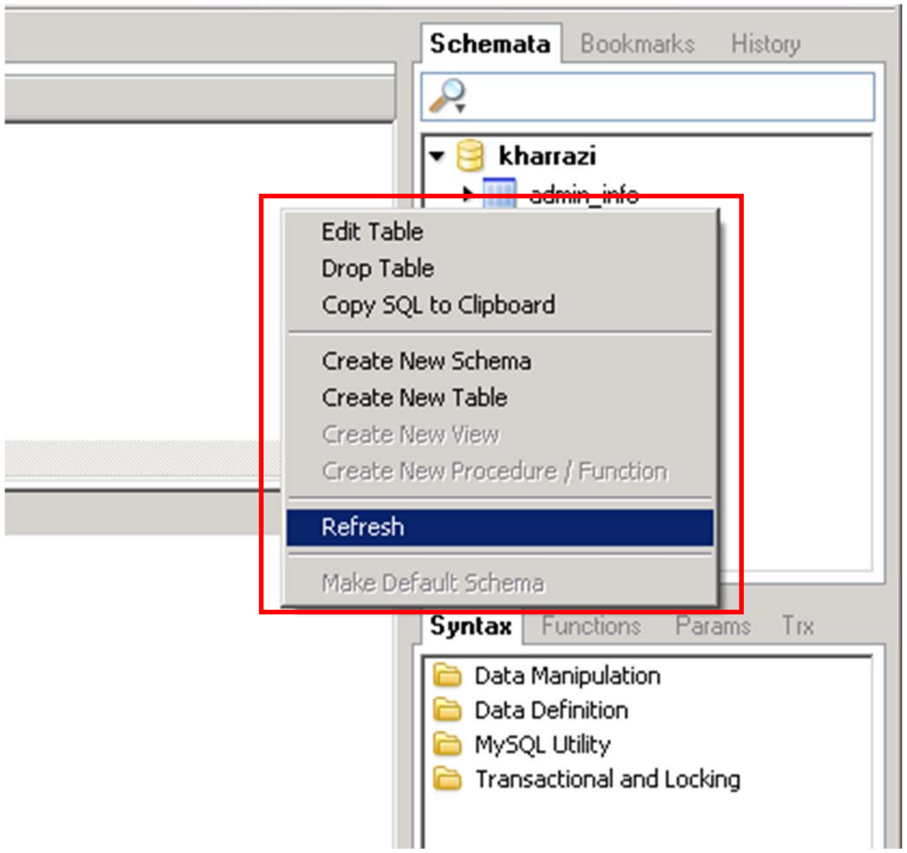
Copy and paste the generated code and then change the 'ENGINE' to 'TYPE'

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

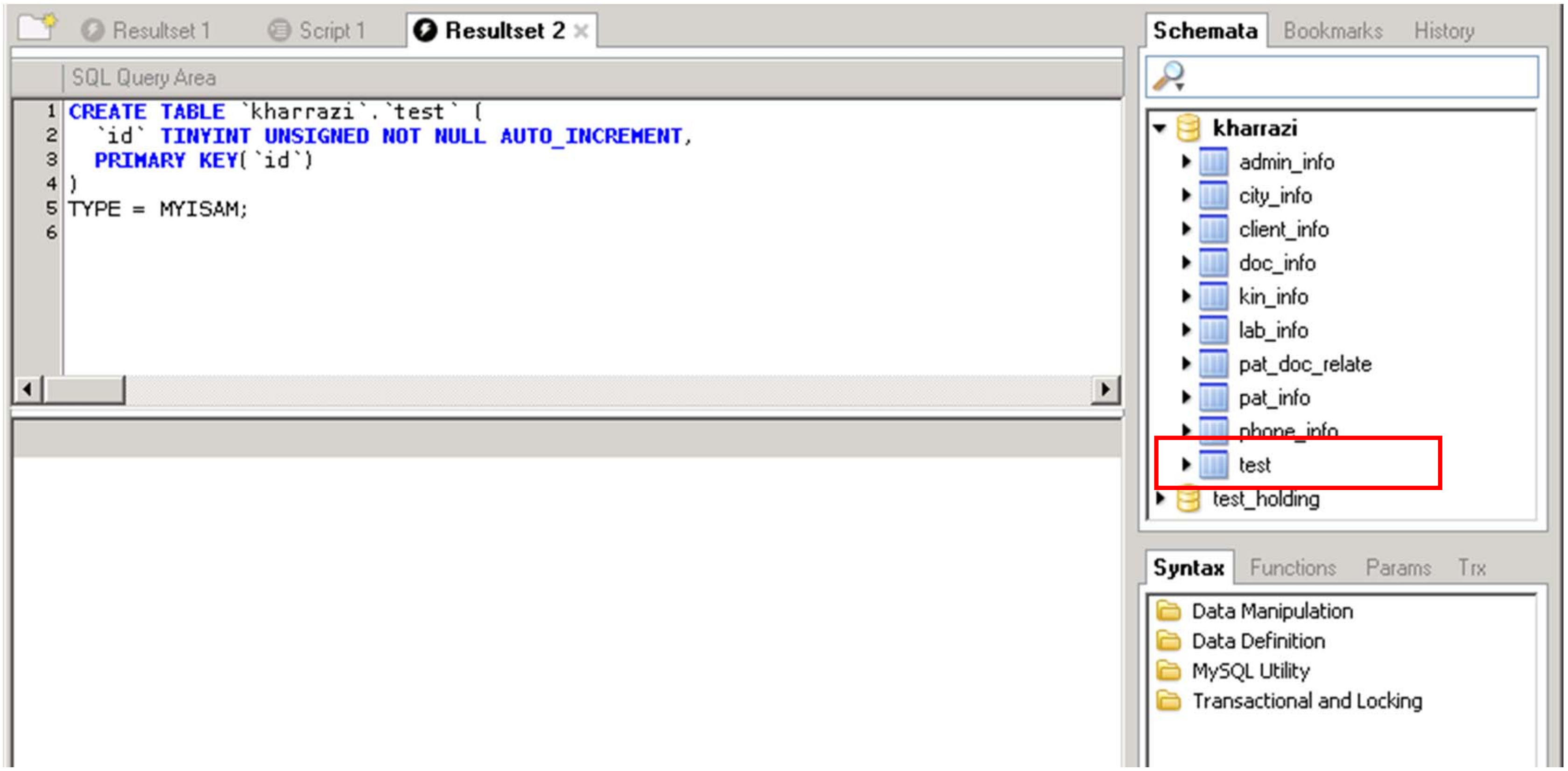


```
CREATE TABLE `kharrazi`.`test` (
  `id` TINYINT UNSIGNED NOT NULL AUTO_INCREMENT,
  PRIMARY KEY(`id`)
)
TYPE = MYISAM;
```

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



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

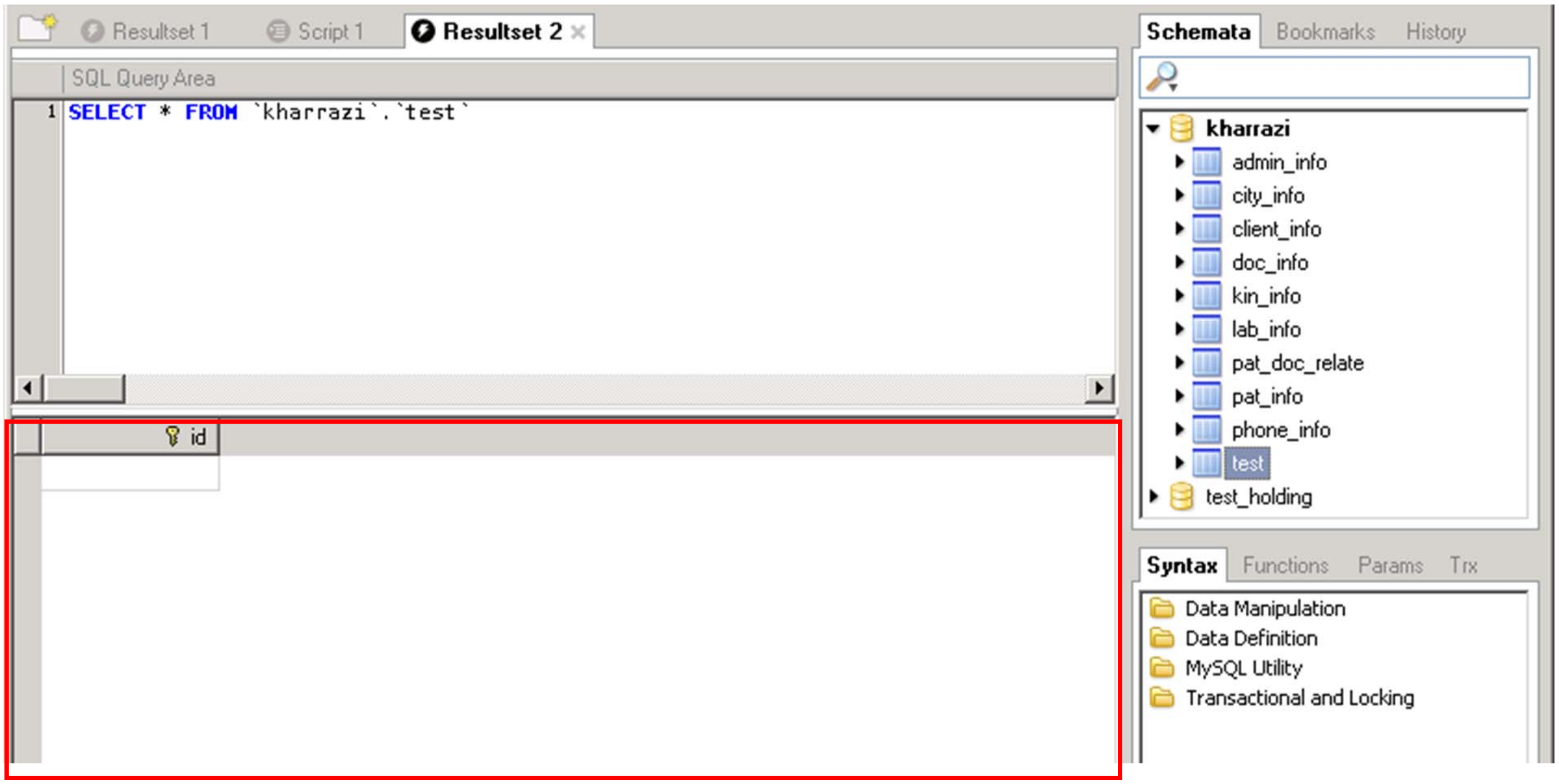


```
CREATE TABLE `kharrazi`.`test` (
  `id` TINYINT UNSIGNED NOT NULL AUTO_INCREMENT,
  PRIMARY KEY(`id`)
)
TYPE = MYISAM;
```

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

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

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



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

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)

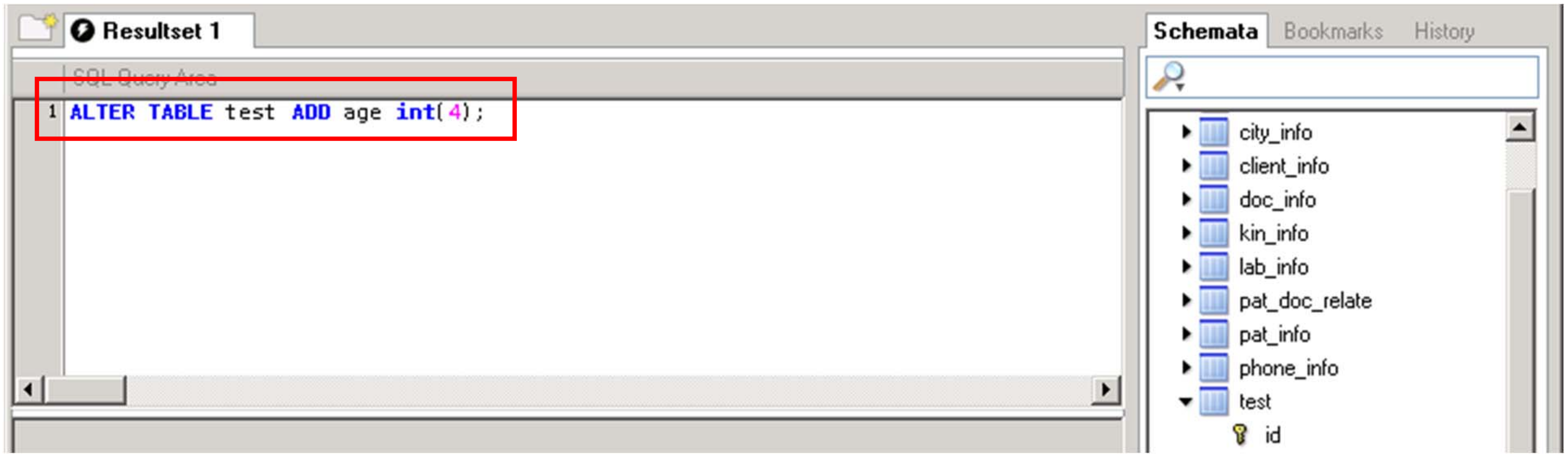
- Syntax:

```
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;
```

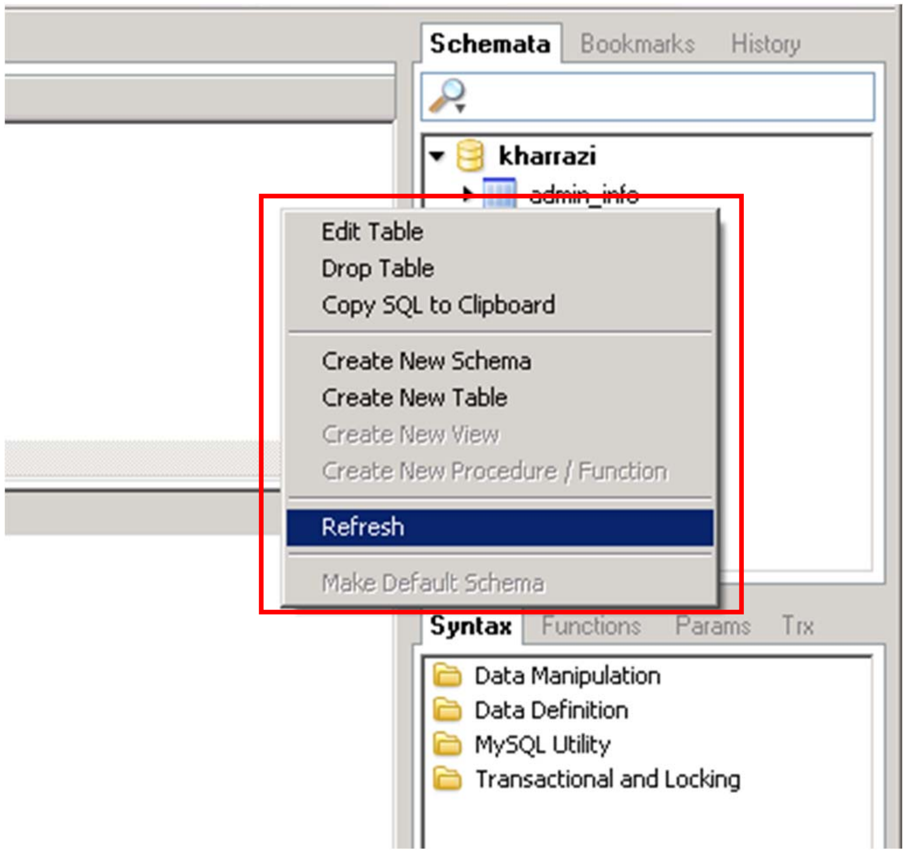
SQL: ALTER (cont.) (table)



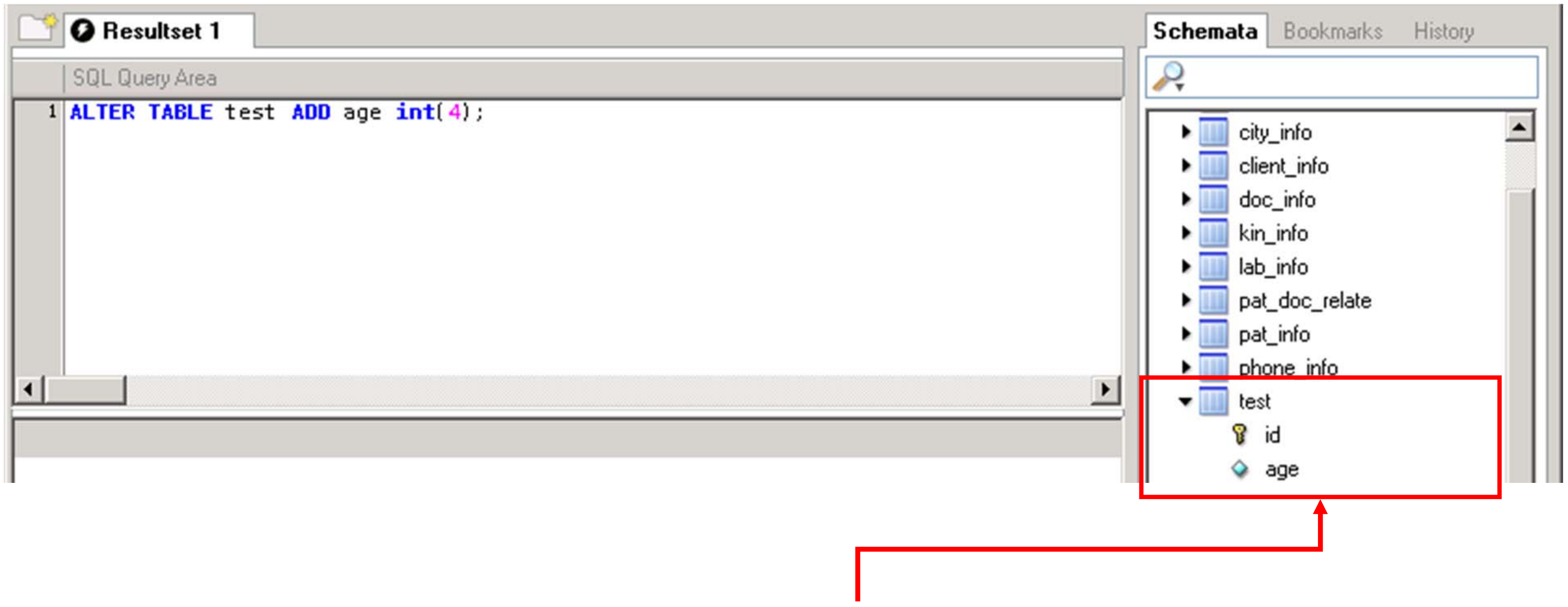
```
ALTER TABLE test ADD age int(4);
```

Adding a new column

SQL: ALTER (cont.) (table)



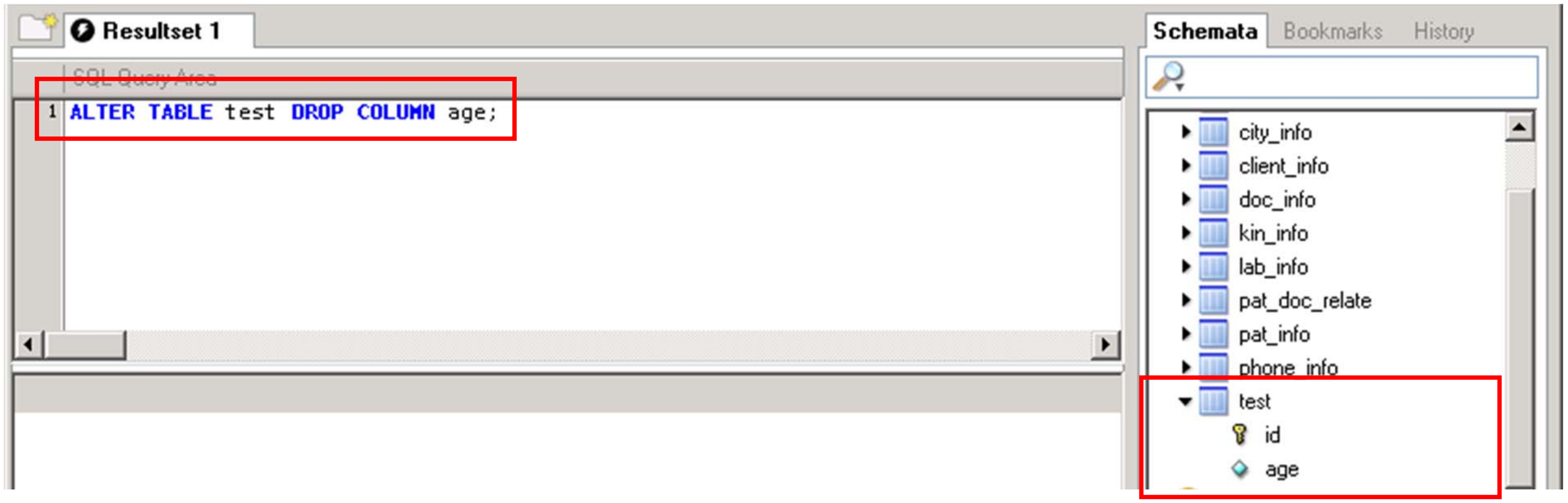
SQL: ALTER (cont.) (table)



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

A new column is added to the table

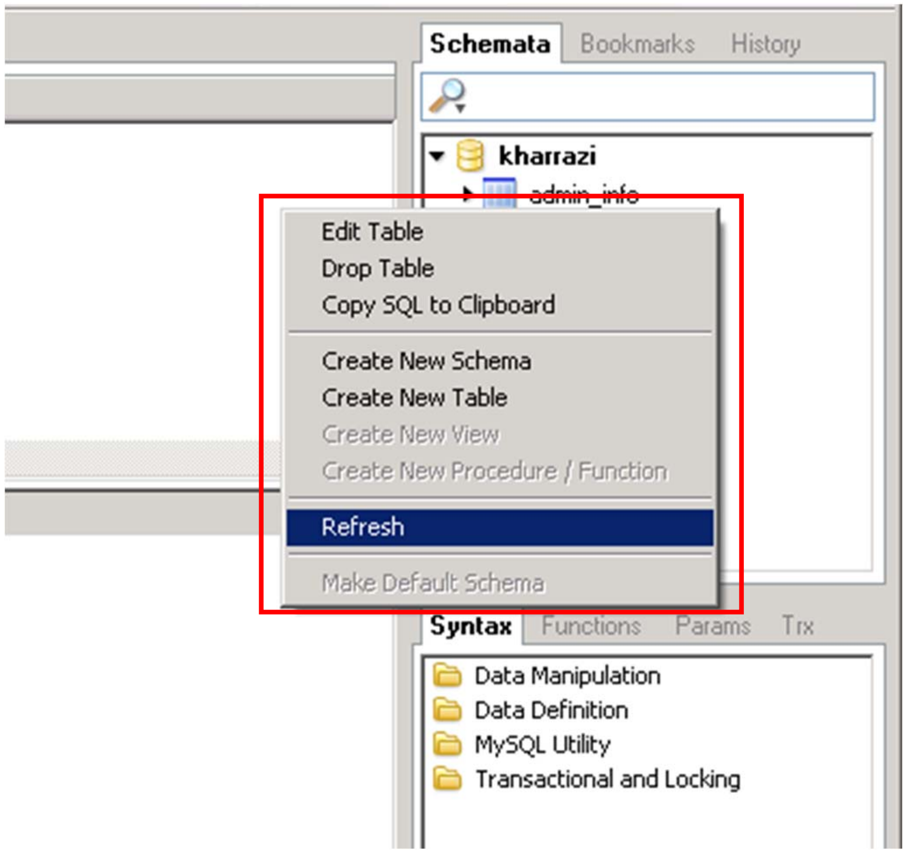
SQL: ALTER (cont.) (table)



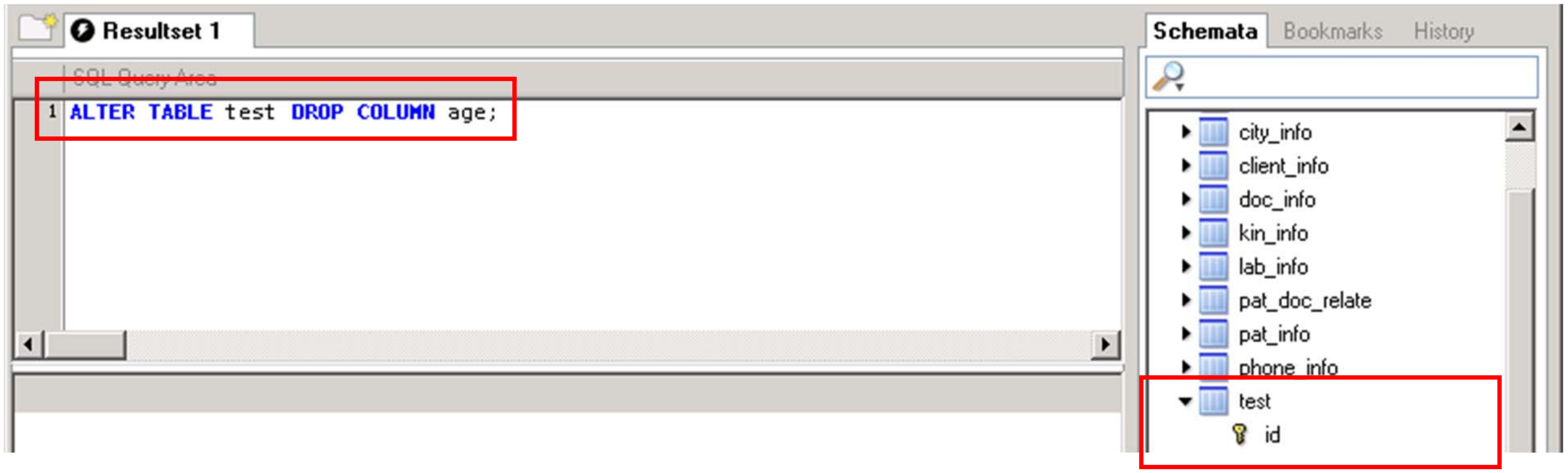
```
ALTER TABLE test DROP COLUMN age;
```

Deleting a column

SQL: ALTER (cont.) (table)



SQL: ALTER (cont.) (table)



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

5. SQL: INSERT INTO Clause

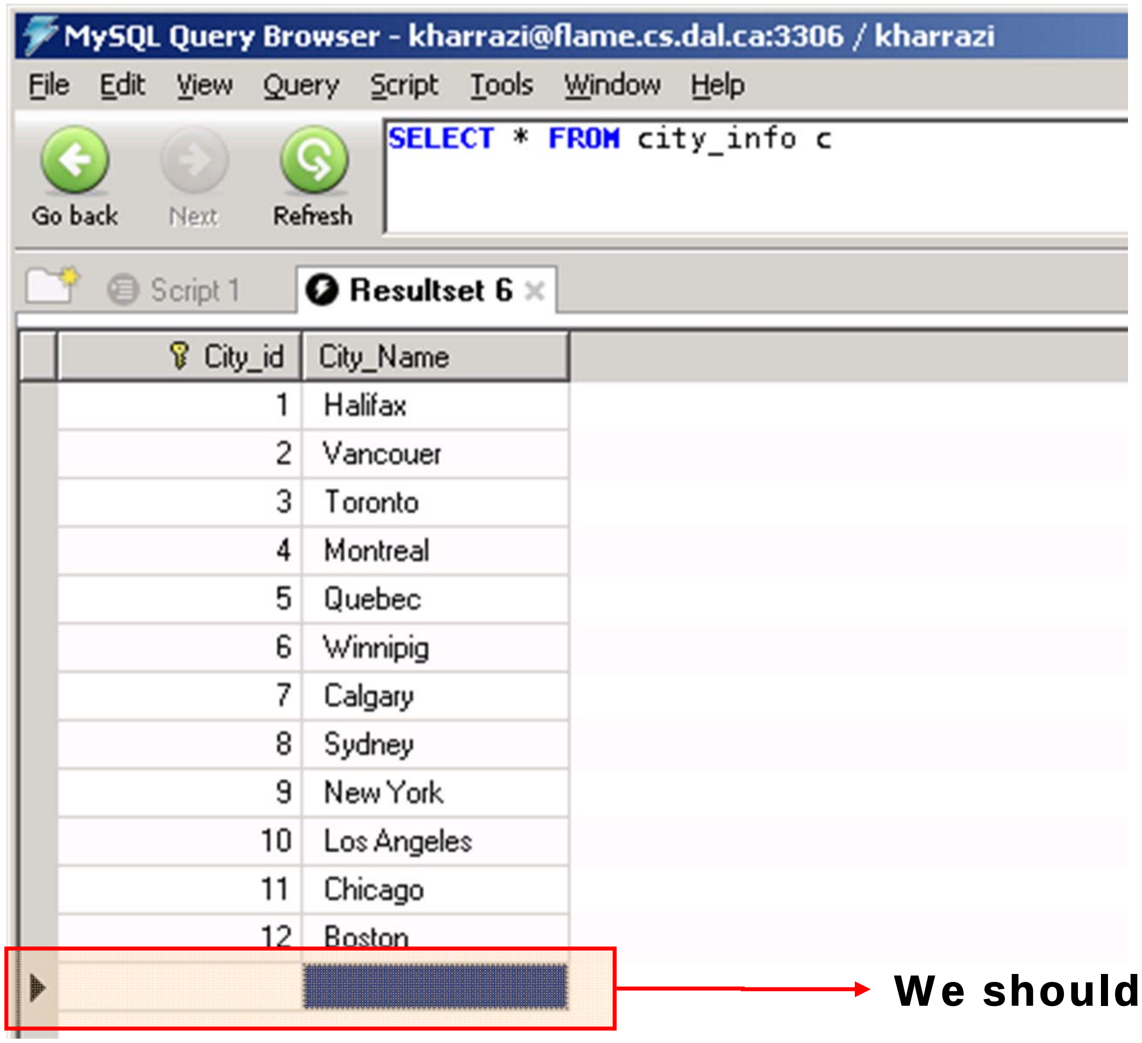
- The **INSERT INTO** statement is used to insert new rows into a table.
- Syntax:

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

- Examples:

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

SQL: *INSERT INTO* (cont.)

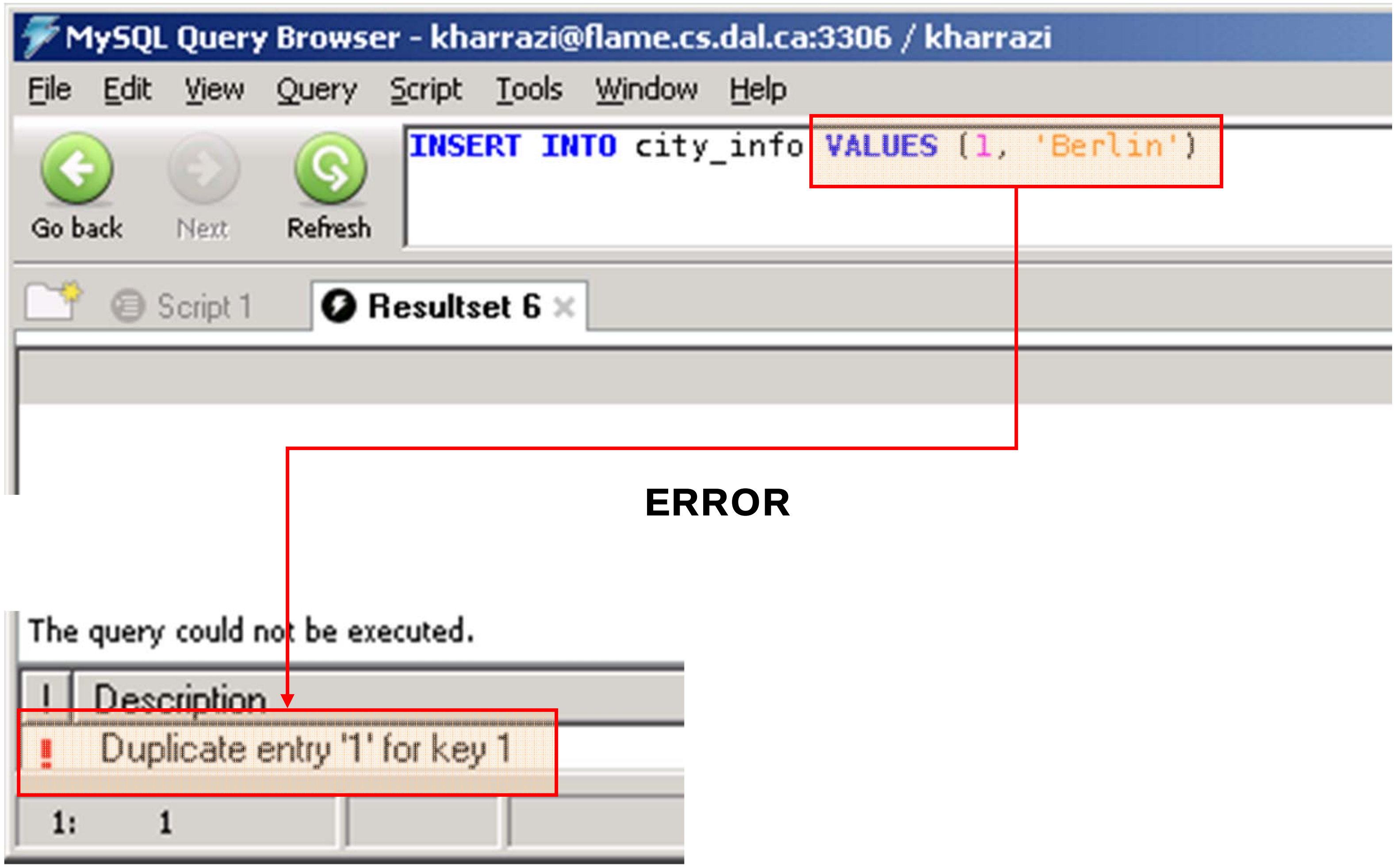

MySQL Query Browser - kharrazi@flame.cs.dal.ca:3306 / kharrazi
File Edit View Query Script Tools Window Help
Go back
Next
Refresh
Script 1
Resultset 6

```
SELECT * FROM city_info c
```

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

We should insert here

SQL: INSERT INTO (cont.)

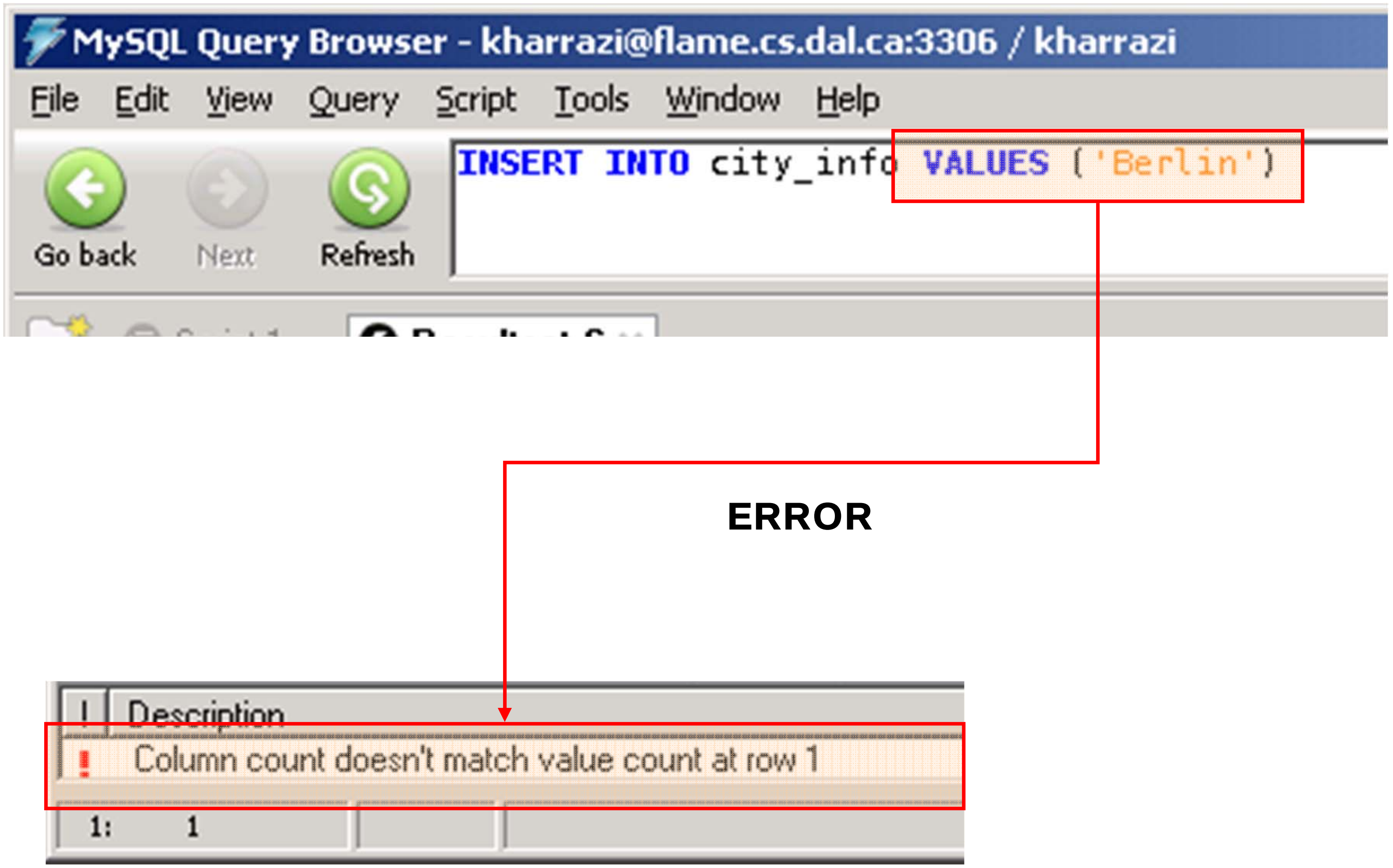

MySQL Query Browser - kharrazi@flame.cs.dal.ca:3306 / kharrazi
File Edit View Query Script Tools Window Help
Go back
Next
Refresh
Script 1
Resultset 6
Description
1: 1

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

ERROR

```
The query could not be executed.
Duplicate entry '1' for key 1
```

SQL: INSERT INTO (cont.)

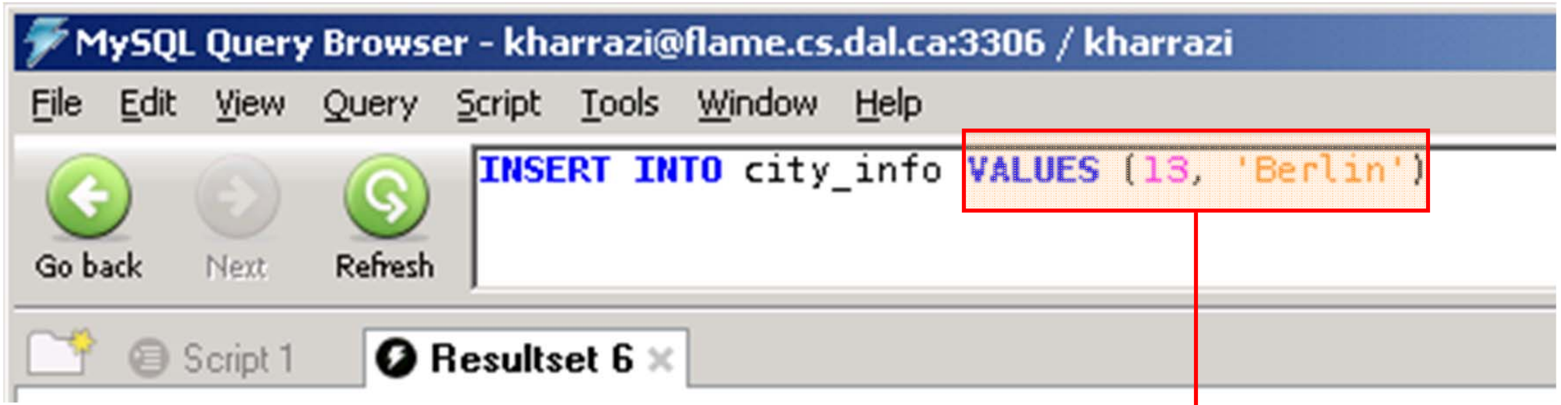

MySQL Query Browser - kharrazi@flame.cs.dal.ca:3306 / kharrazi
File Edit View Query Script Tools Window Help
Go back
Next
Refresh
Description

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

ERROR

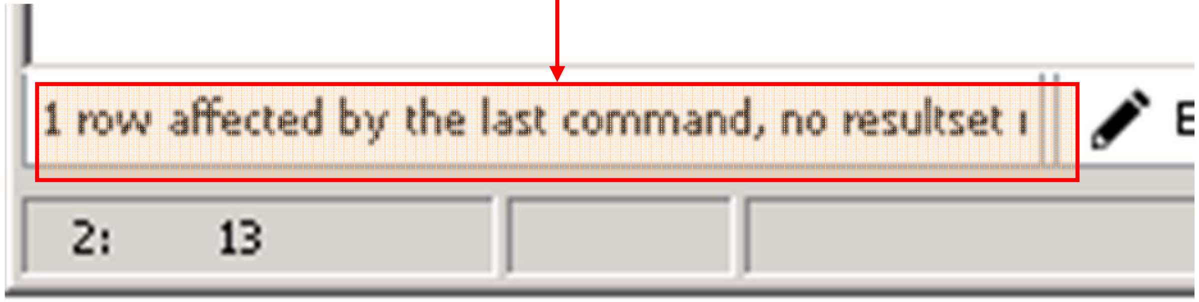
```
Column count doesn't match value count at row 1
```

SQL: INSERT INTO (cont.)



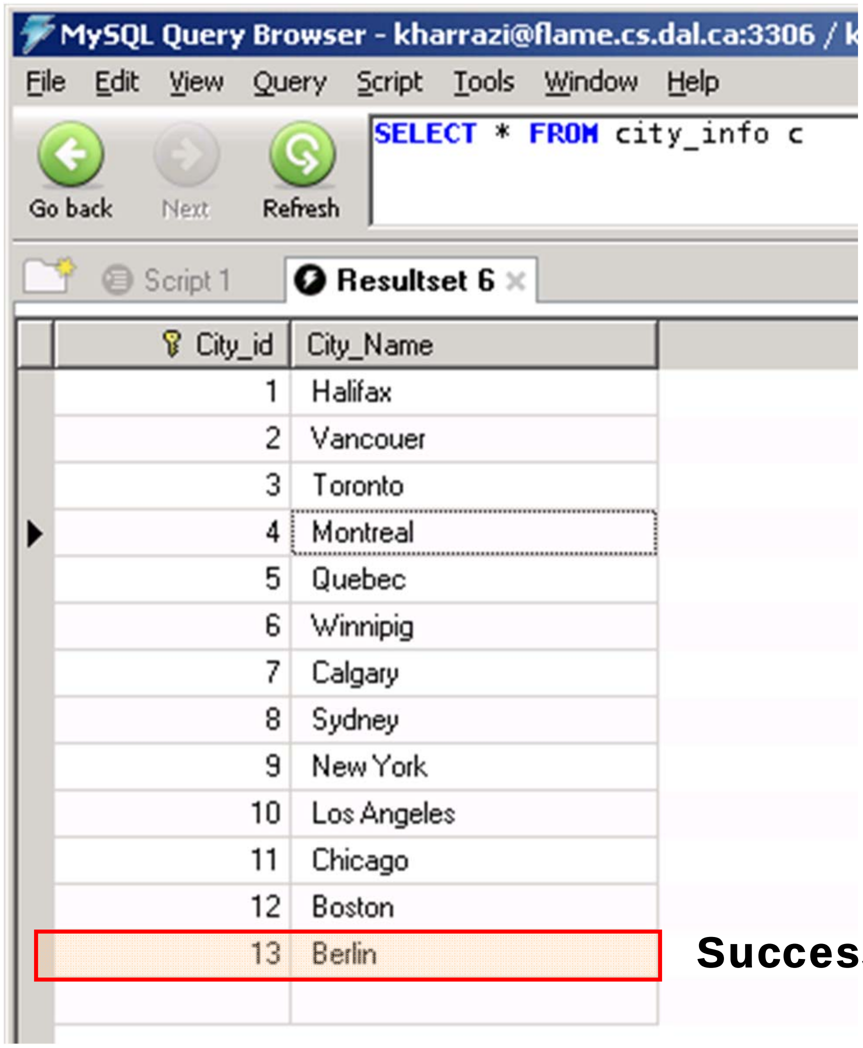
```
INSERT INTO city_info VALUES (13, 'Berlin')
```

Successful



```
1 row affected by the last command, no resultset r
2:        13
```

SQL: INSERT INTO (cont.)


MySQL Query Browser - kharrazi@flame.cs.dal.ca:3306 / k
File Edit View Query Script Tools Window Help
Go back
Next
Refresh
Script 1
Resultset 6

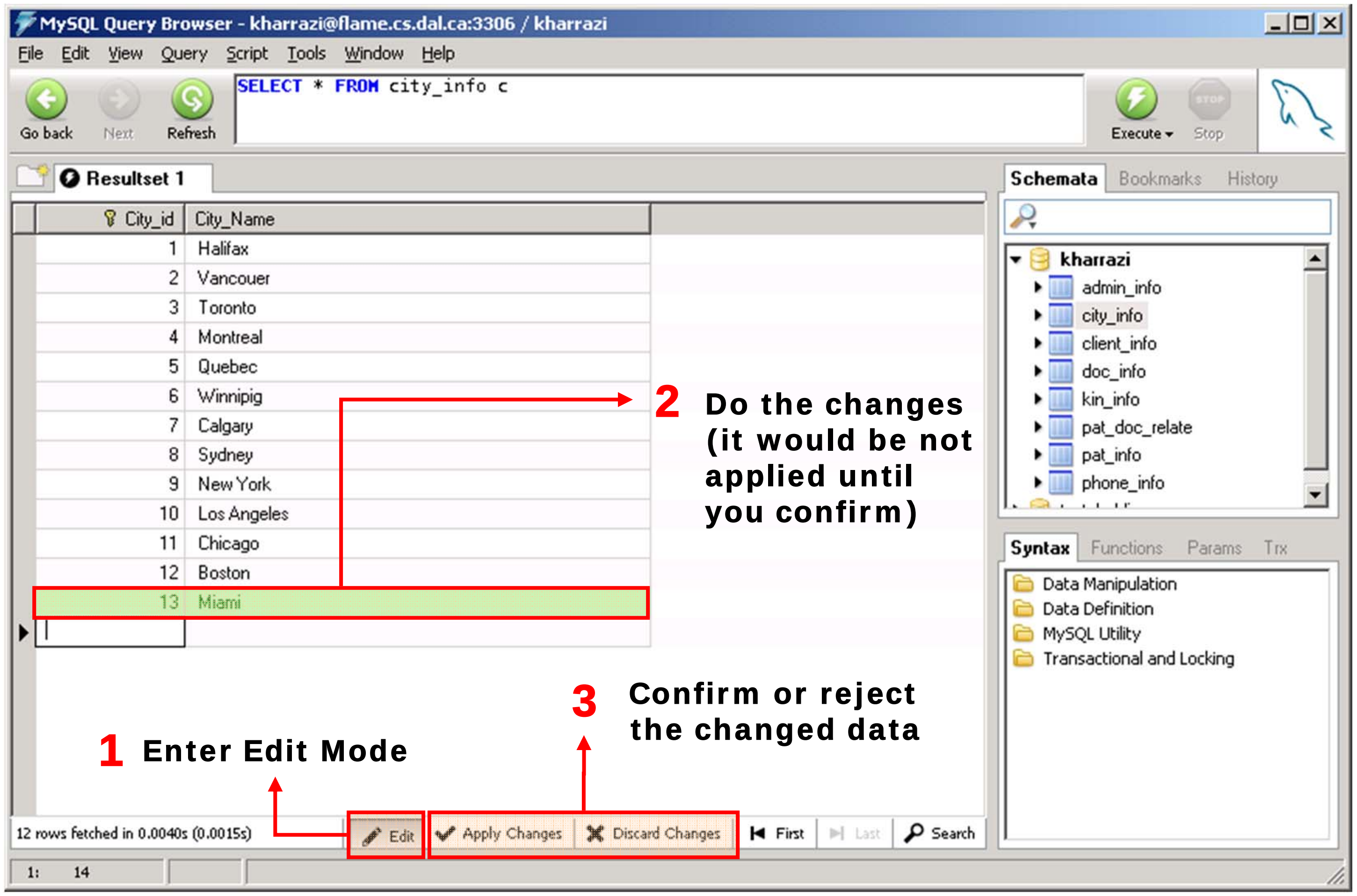
```
SELECT * FROM city_info c
```

City_id	City_Name
1	Halifax
2	Vancouer
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:



```
SELECT * FROM city_info c
```

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

12 rows fetched in 0.0040s (0.0015s)

6. SQL: UPDATE/SET Statement

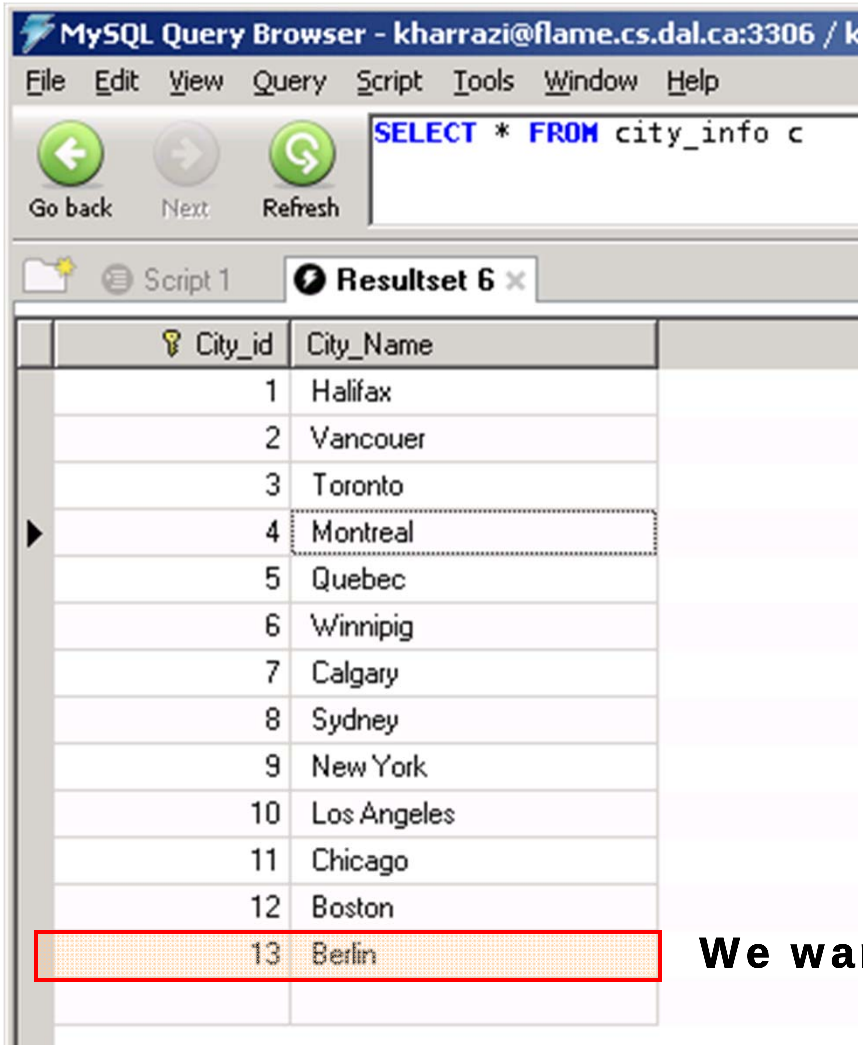
- The **UPDATE** statement is used to modify the data in a table.
- Syntax:

```
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.)

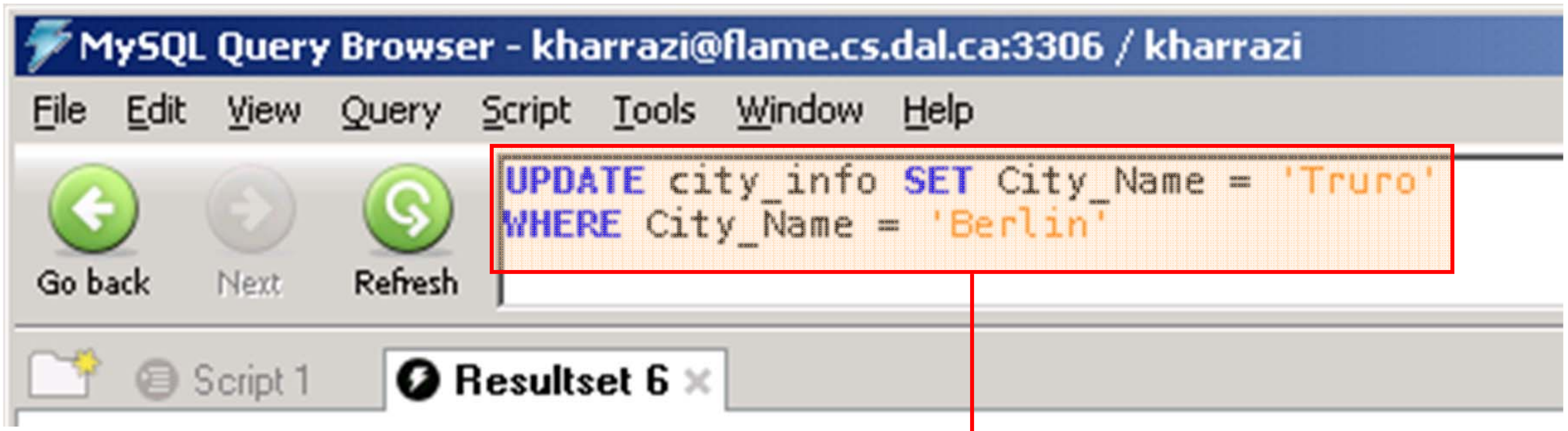

MySQL Query Browser - kharrazi@flame.cs.dal.ca:3306 / k
File Edit View Query Script Tools Window Help
Go back
Next
Refresh
Script 1
Resultset 6

```
SELECT * FROM city_info c
```

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

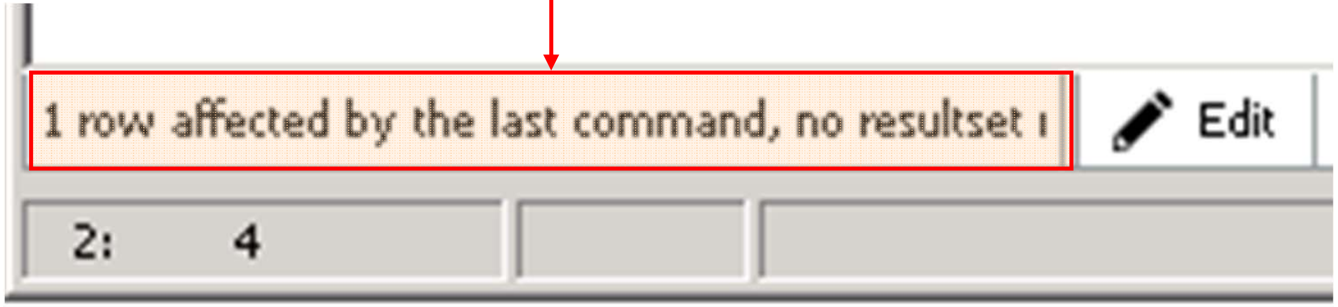
We want to change it

SQL: UPDATE/SET (cont.)



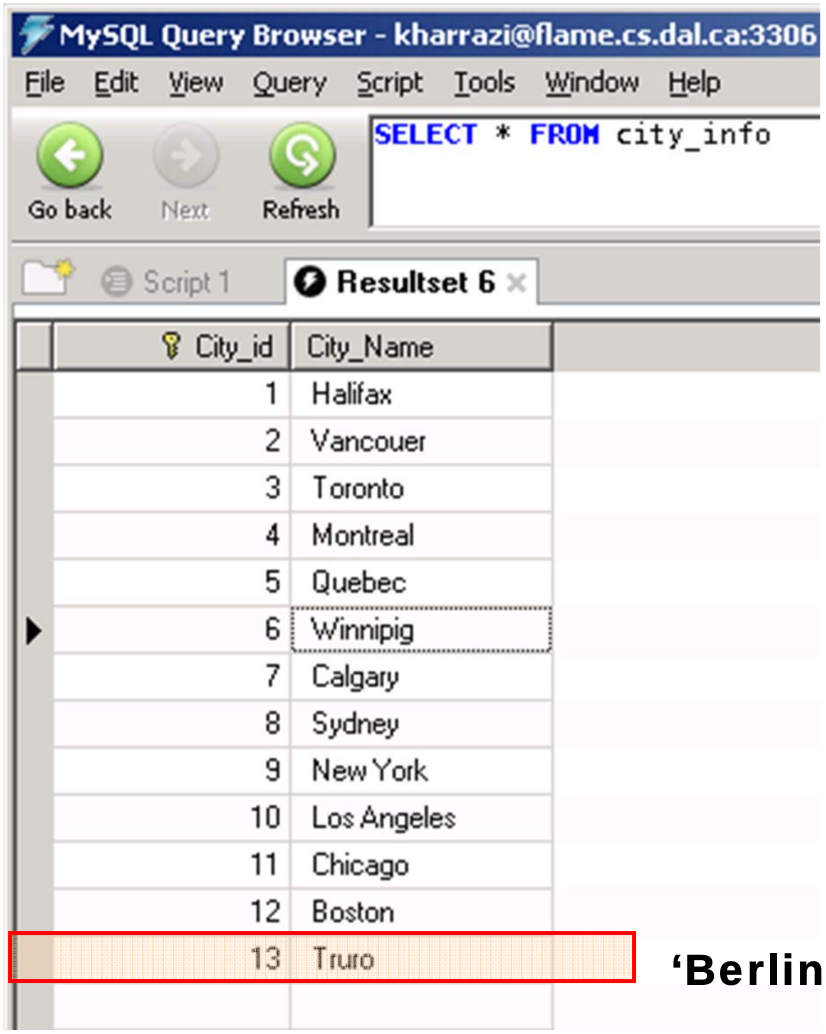
```
UPDATE city_info SET City_Name = 'Truro'
WHERE City_Name = 'Berlin'
```

Successful



```
1 row affected by the last command, no resultset r
```

SQL: UPDATE/SET (cont.)


MySQL Query Browser - kharrazi@flame.cs.dal.ca:3306
File Edit View Query Script Tools Window Help
Go back
Next
Refresh
Script 1
Resultset 6

```
SELECT * FROM city_info
```

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

‘Berlin’ has changed to ‘Truro’

7. SQL: DELETE Statement

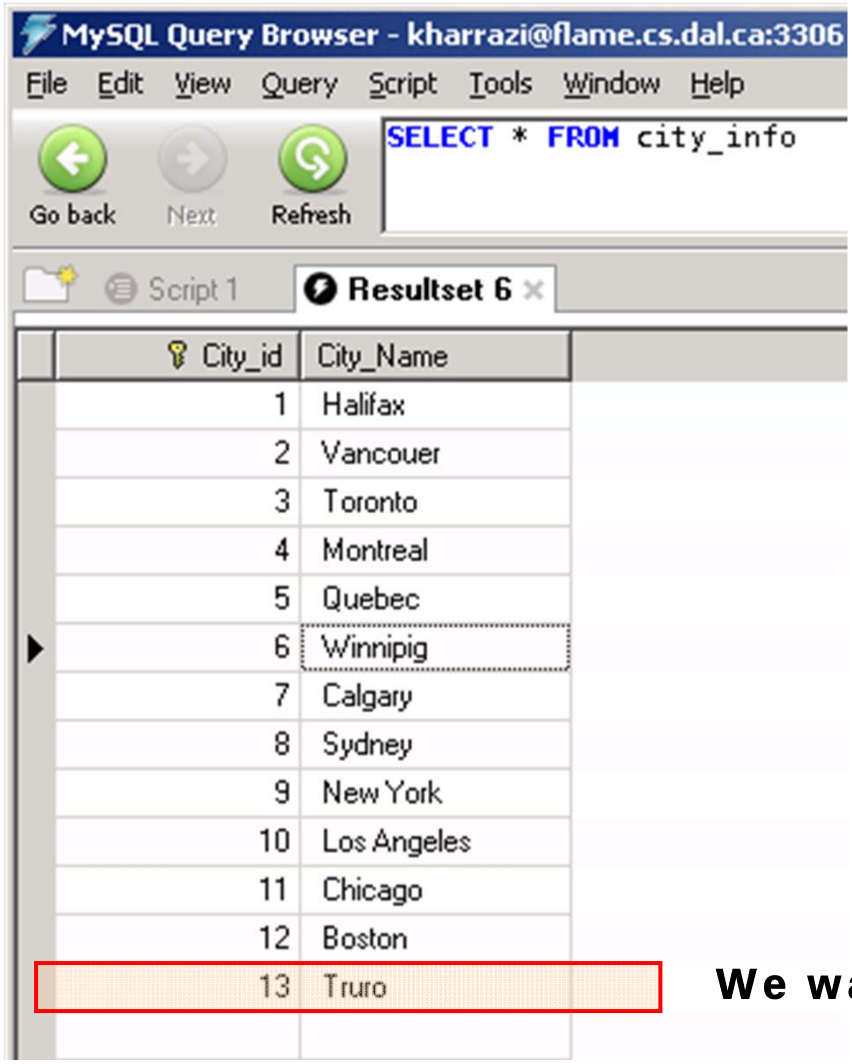
- The **DELETE** statement is used to delete rows in a table.
- Syntax:

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

- Examples:

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

SQL: DELETE (cont.)

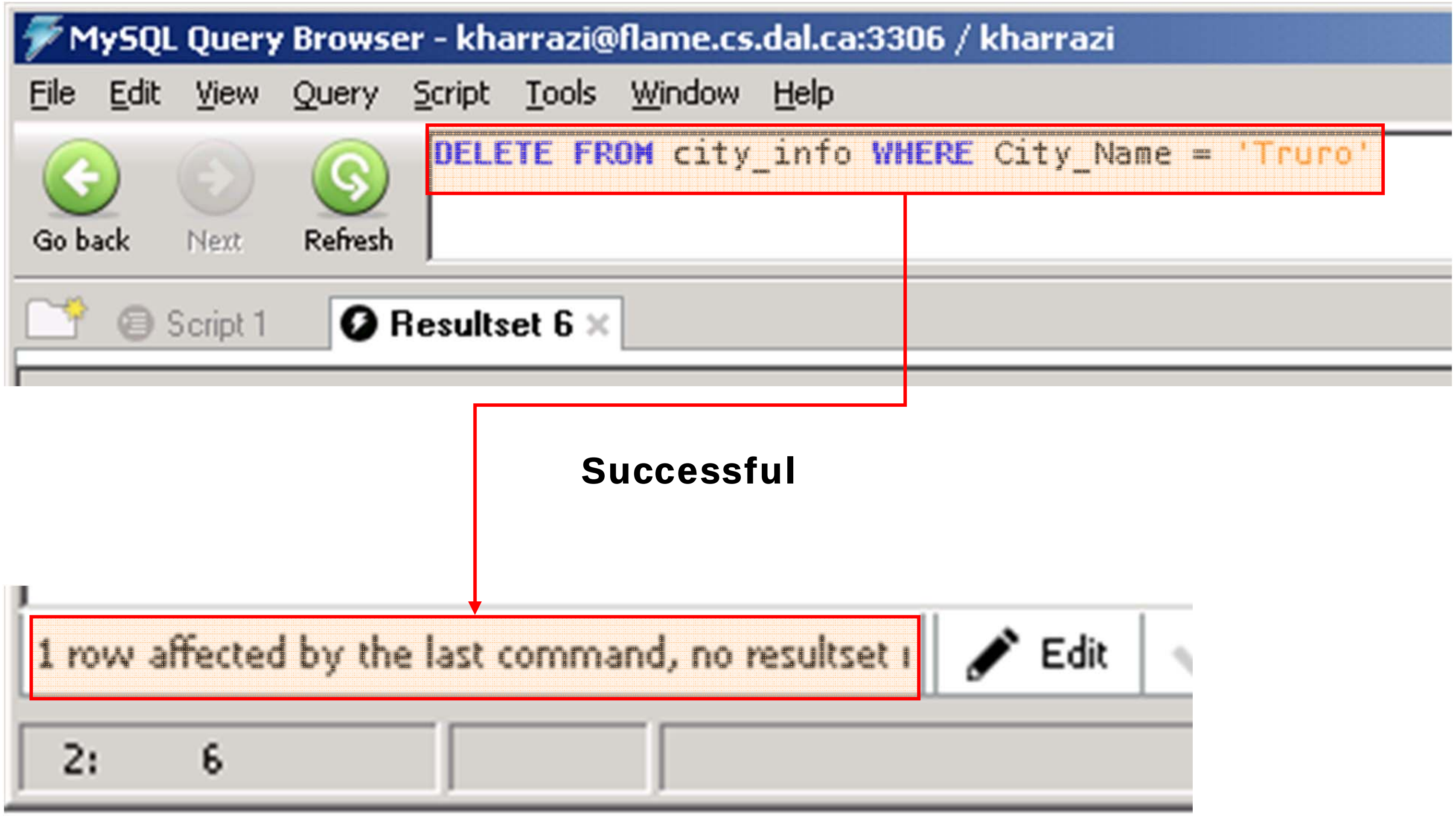

MySQL Query Browser - kharrazi@flame.cs.dal.ca:3306
File Edit View Query Script Tools Window Help
Go back
Next
Refresh
Script 1
Resultset 6

```
SELECT * FROM city_info
```

City_id	City_Name
1	Halifax
2	Vancouer
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.)

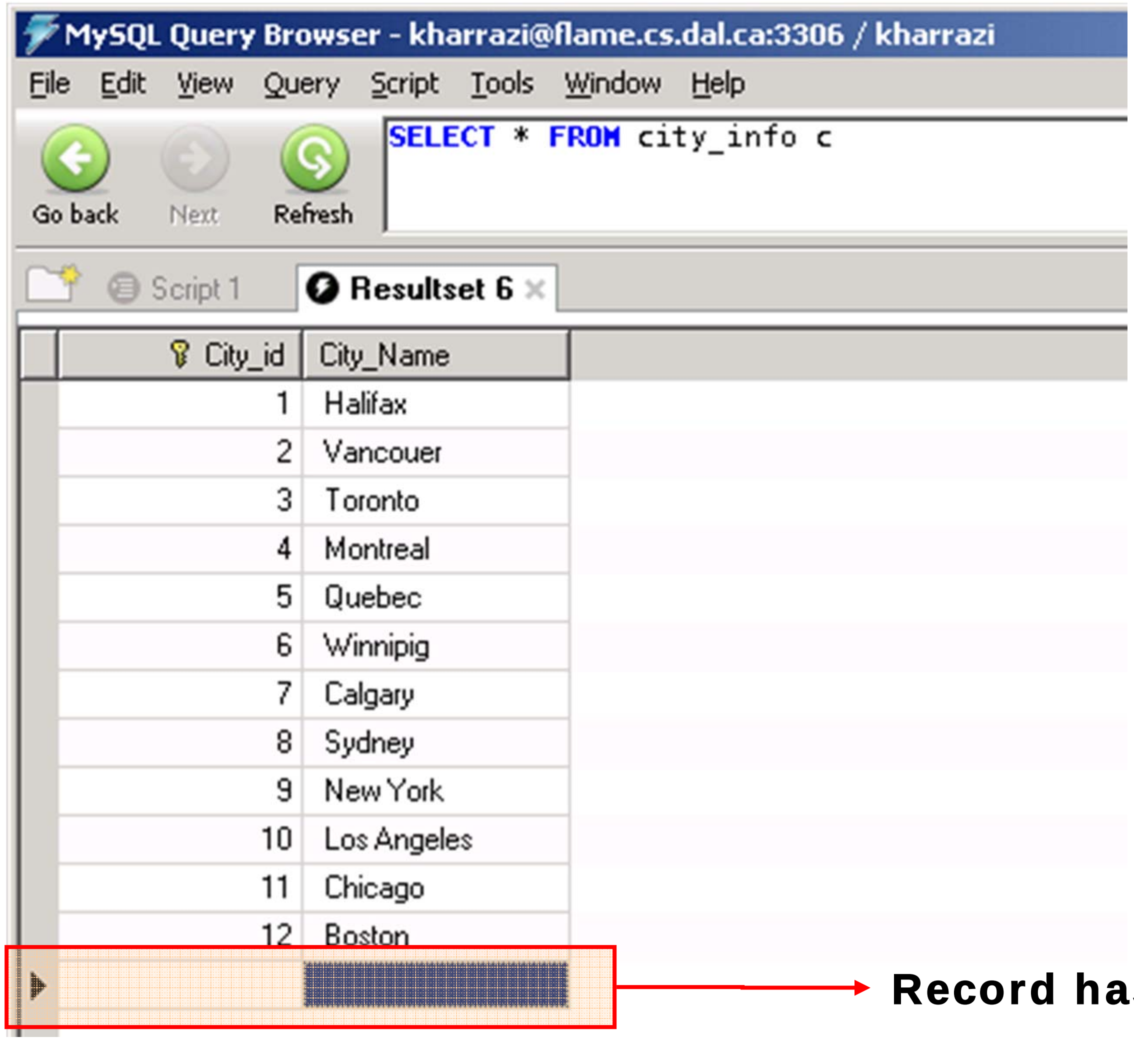

MySQL Query Browser - kharrazi@flame.cs.dal.ca:3306 / kharrazi
File Edit View Query Script Tools Window Help
Go back
Next
Refresh
Script 1
Resultset 6
Edit
2: 6

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

Successful

```
1 row affected by the last command, no resultset r
```

SQL: DELETE (cont.)


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

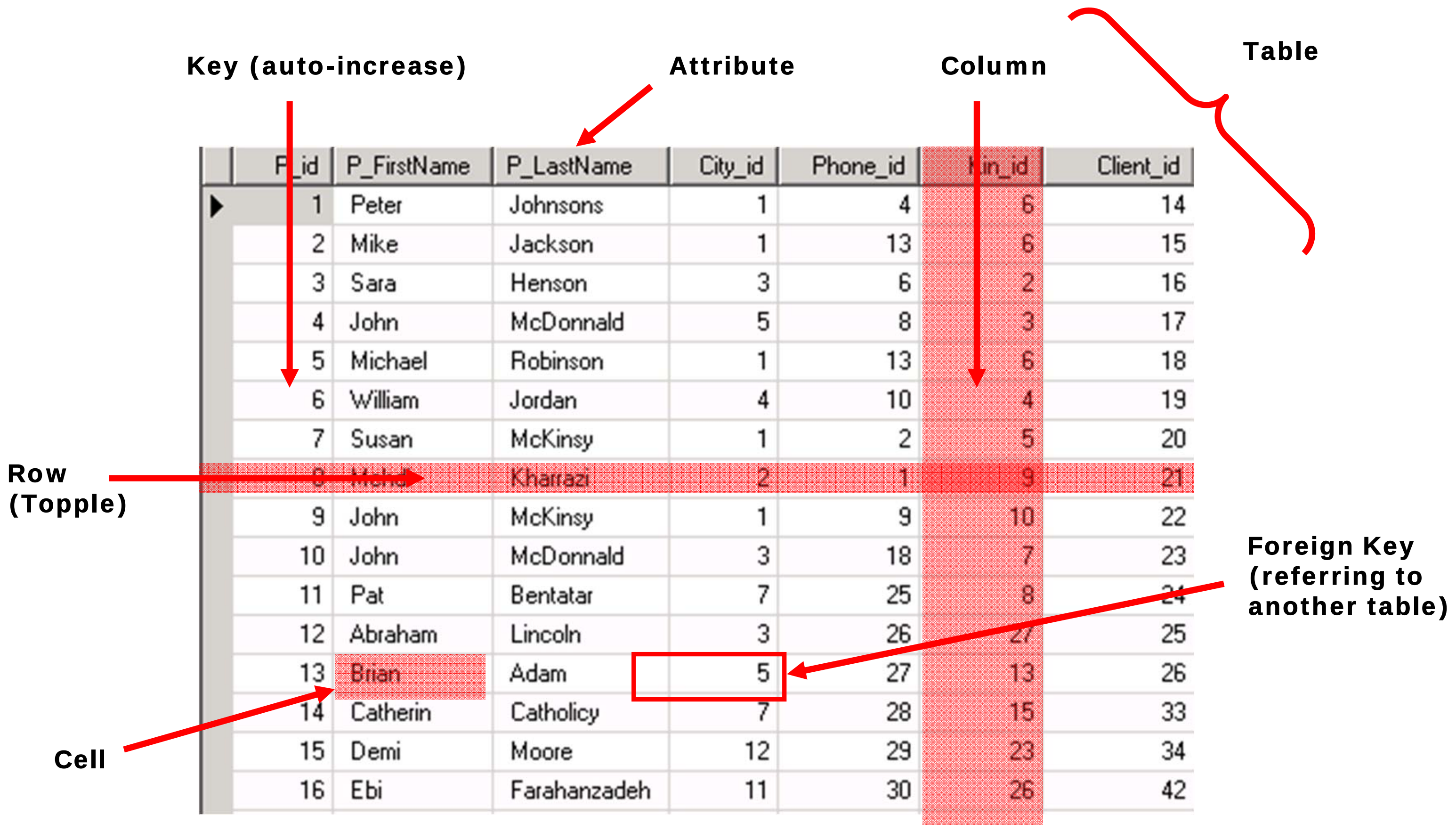
```
SELECT * FROM city_info c
```

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

Record has been deleted

8. SQL: Joining and Keys

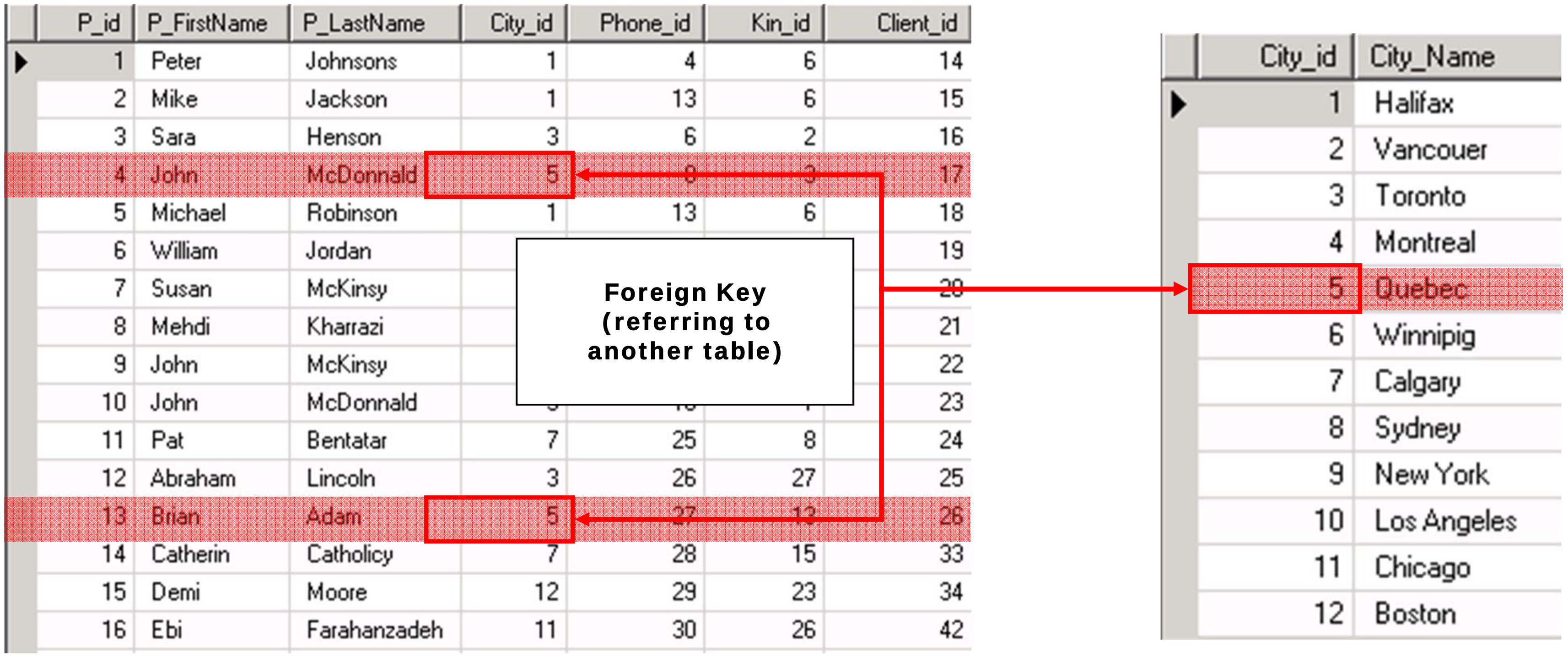
- Elements of the relational database table:



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	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	McDonnald	5	8	3	17
5	Michael	Robinson	1	13	6	18
6	William	Jordan	[illegible]	[illegible]	[illegible]	19
7	Susan	McKinsy	[illegible]	[illegible]	[illegible]	20
8	Mehdi	Kharrazi	[illegible]	[illegible]	[illegible]	21
9	John	McKinsy	[illegible]	[illegible]	[illegible]	22
10	John	McDonnald	[illegible]	[illegible]	[illegible]	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

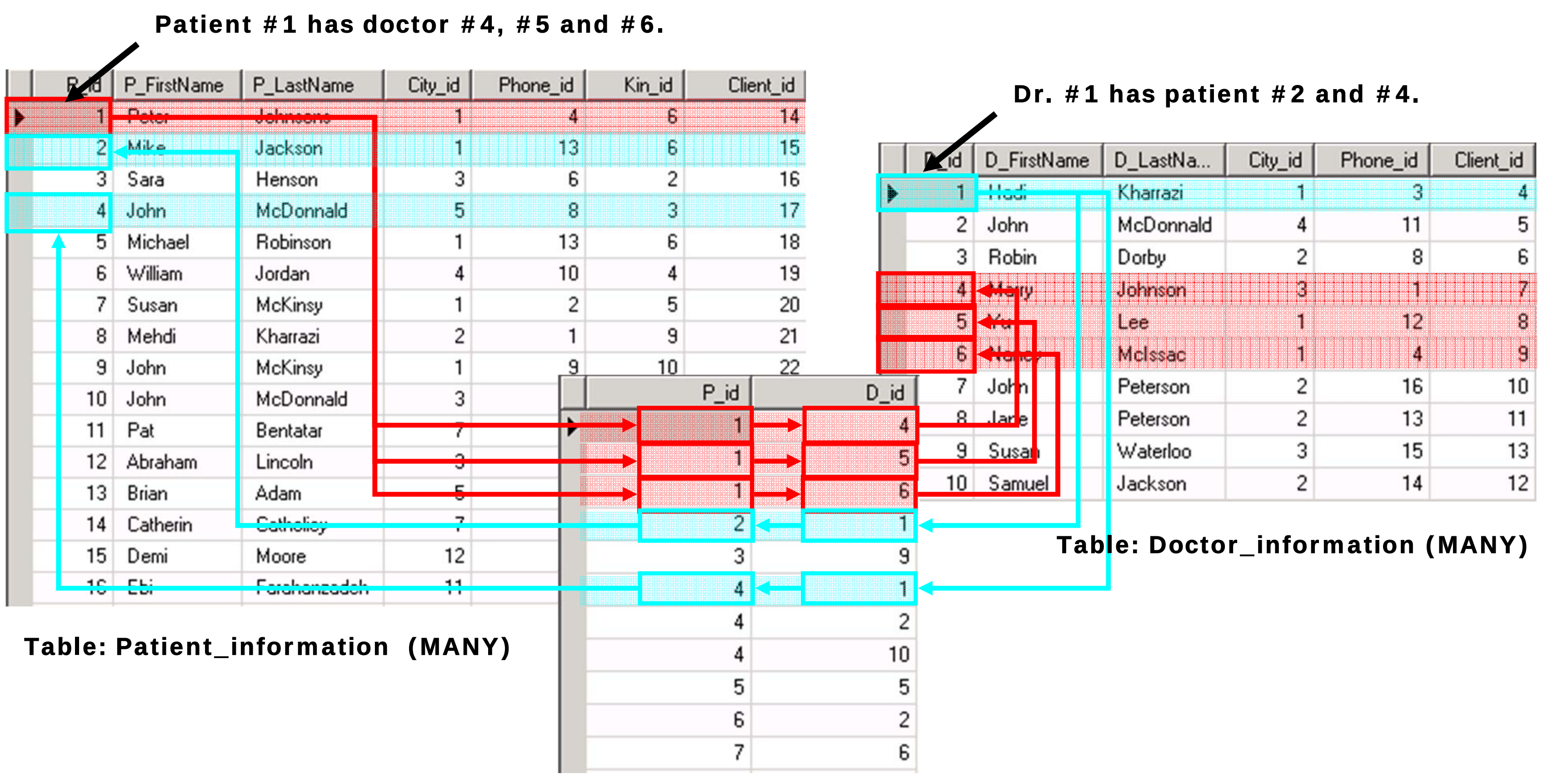
Table: Patient_information (MANY)

City_id	City_Name
1	Halifax
2	Vancouer
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)

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	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			
11	Pat	Bentatar	7			
12	Abraham	Lincoln	3			
13	Brian	Adam	5			
14	Catherin	Catholicy	7			
15	Demi	Moore	12			
16	Ebi	Farahanzadeh	11			

Table: Patient_information (MANY)

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

Table: Patient_Doctor_realationship

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

D_id	D_FirstName	D_LastNa...	City_id	Phone_id	Client_id
1	Hadi	Kharrazi	1	3	4
2	John	McDonnald	4	11	5
3	Robin	Dorby	2	8	6
4	Marry	Johnson	3	1	7
5	Yu	Lee	1	12	8
6	Nancy	McIssac	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)

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:

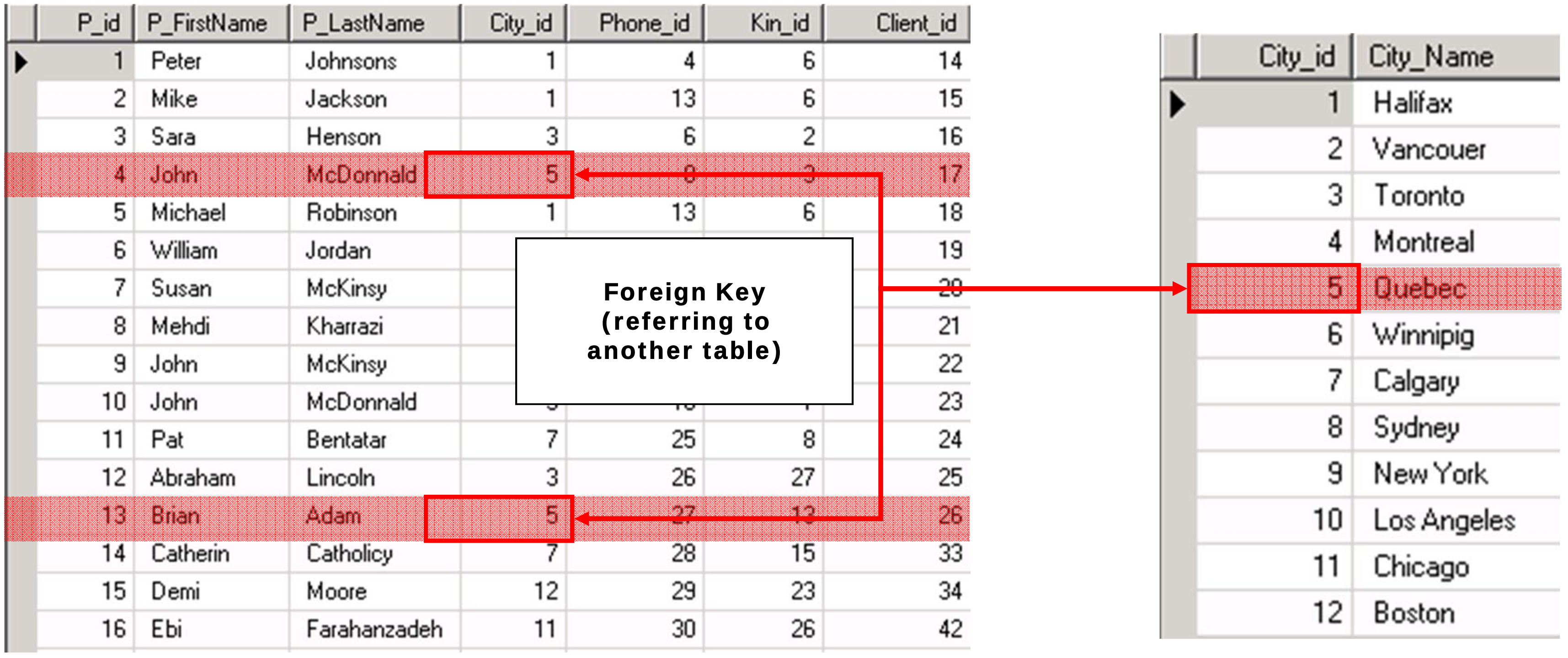
```
SELECT table1.any_column,table2.any_column
FROM table1, table2
WHERE table1.columnX = table2.columnX
```

- 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?


Foreign Key (referring to another table)

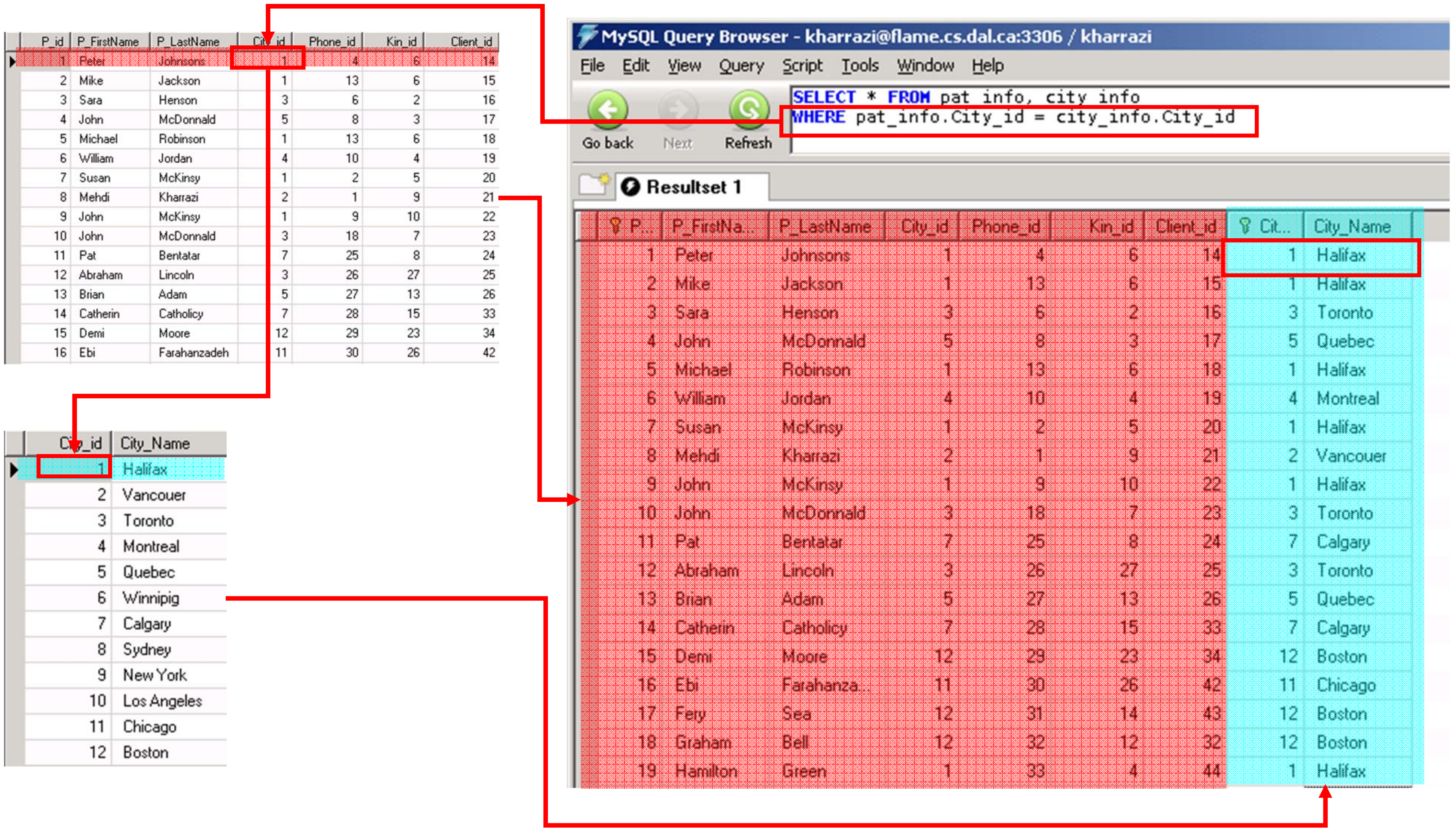
P_id	P_FirstName	P_LastName	City_id	Phone_id	Kin_id	Client_id
1	Peter	Johnsons	1	4	6	14
2	Mike	Jackson	1	13	6	15
3	Sara	Henson	3	6	2	16
4	John	McDonnald	5	8	3	17
5	Michael	Robinson	1	13	6	18
6	William	Jordan	[illegible]	[illegible]	[illegible]	19
7	Susan	McKinsy	[illegible]	[illegible]	[illegible]	20
8	Mehdi	Kharrazi	[illegible]	[illegible]	[illegible]	21
9	John	McKinsy	[illegible]	[illegible]	[illegible]	22
10	John	McDonnald	[illegible]	[illegible]	[illegible]	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	Vancouer
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)

SQL: Joining and Keys (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	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

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

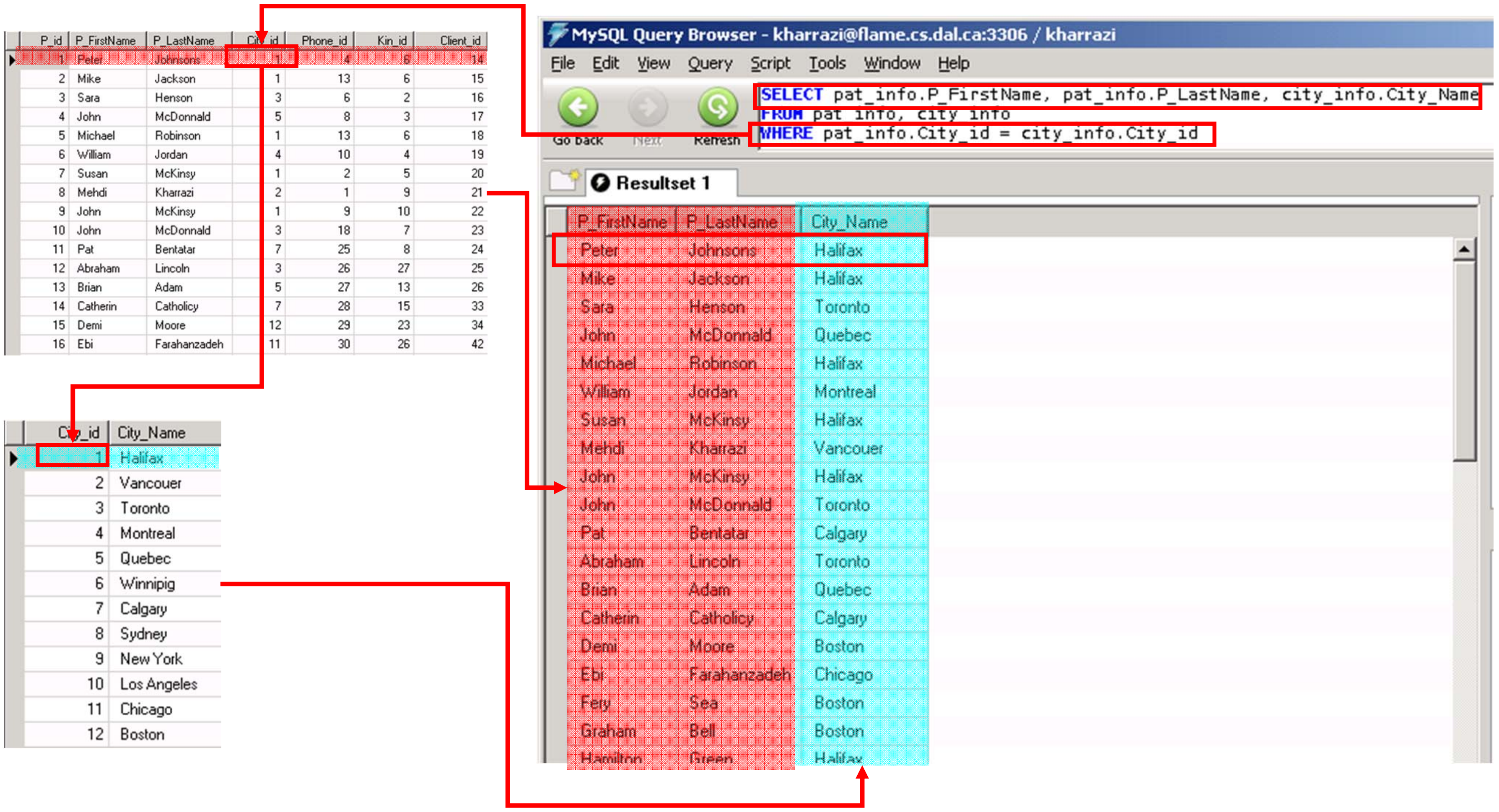
MySQL Query Browser - kharrazi@flame.cs.dal.ca:3306 / kharrazi
File Edit View Query Script Tools Window Help
Go back
Next
Refresh
Resultset 1

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

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	McDonnald	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	Vancouer
9	John	McKinsy	1	9	10	22	1	Halifax
10	John	McDonnald	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	Catholicy	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

Table: Patient_information + City_information

SQL: Joining and Keys (cont.)


MySQL Query Browser - kharrazi@flame.cs.dal.ca:3306 / kharrazi
File Edit View Query Script Tools Window Help
Go back
Next
Refresh
Resultset 1

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	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

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

```
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
```

P_FirstName	P_LastName	City_Name
Peter	Johnsons	Halifax
Mike	Jackson	Halifax
Sara	Henson	Toronto
John	McDonnald	Quebec
Michael	Robinson	Halifax
William	Jordan	Montreal
Susan	McKinsy	Halifax
Mehdi	Kharrazi	Vancouer
John	McKinsy	Halifax
John	McDonnald	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.)

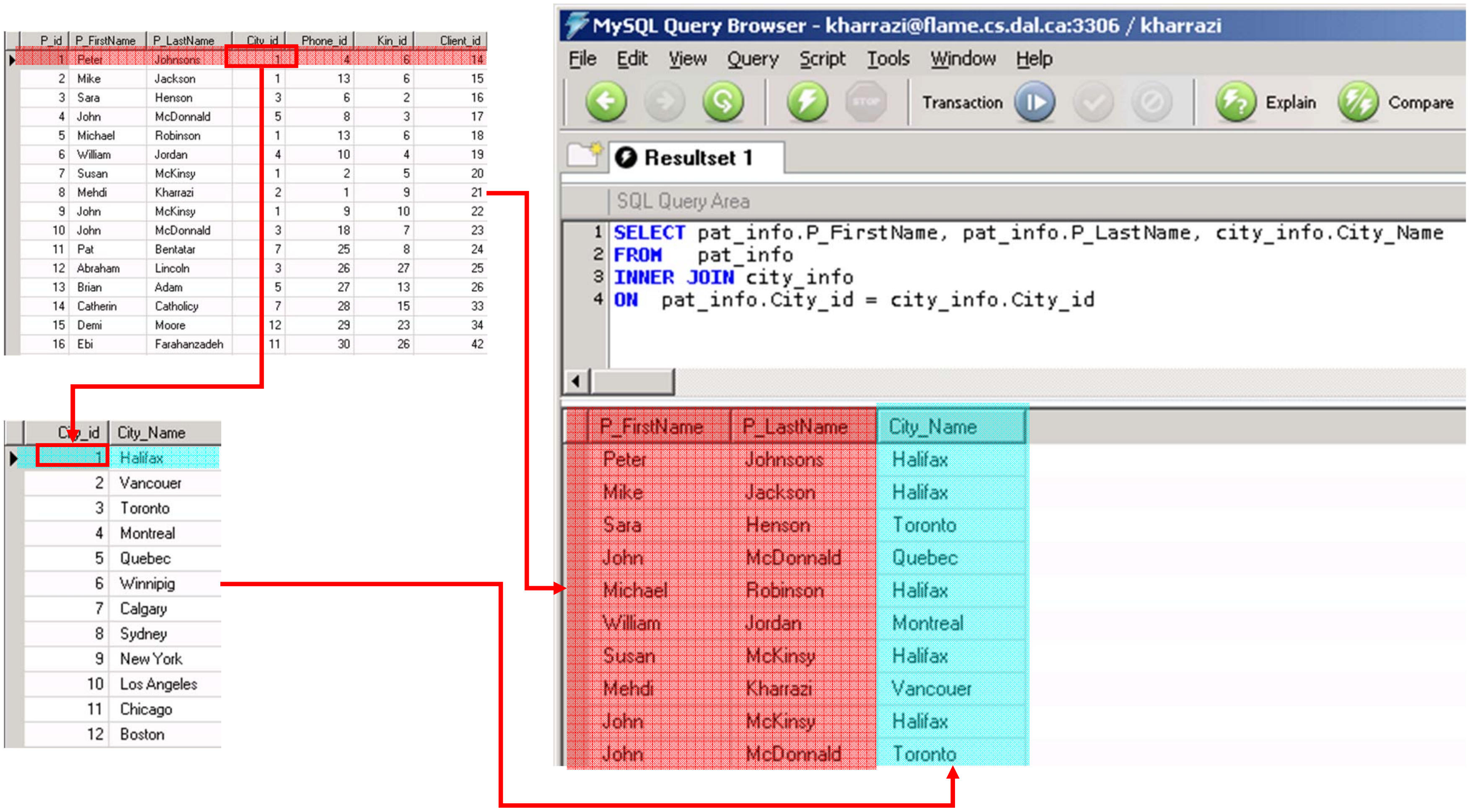
- Syntax:

```
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	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	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

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

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

```sql
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
```

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

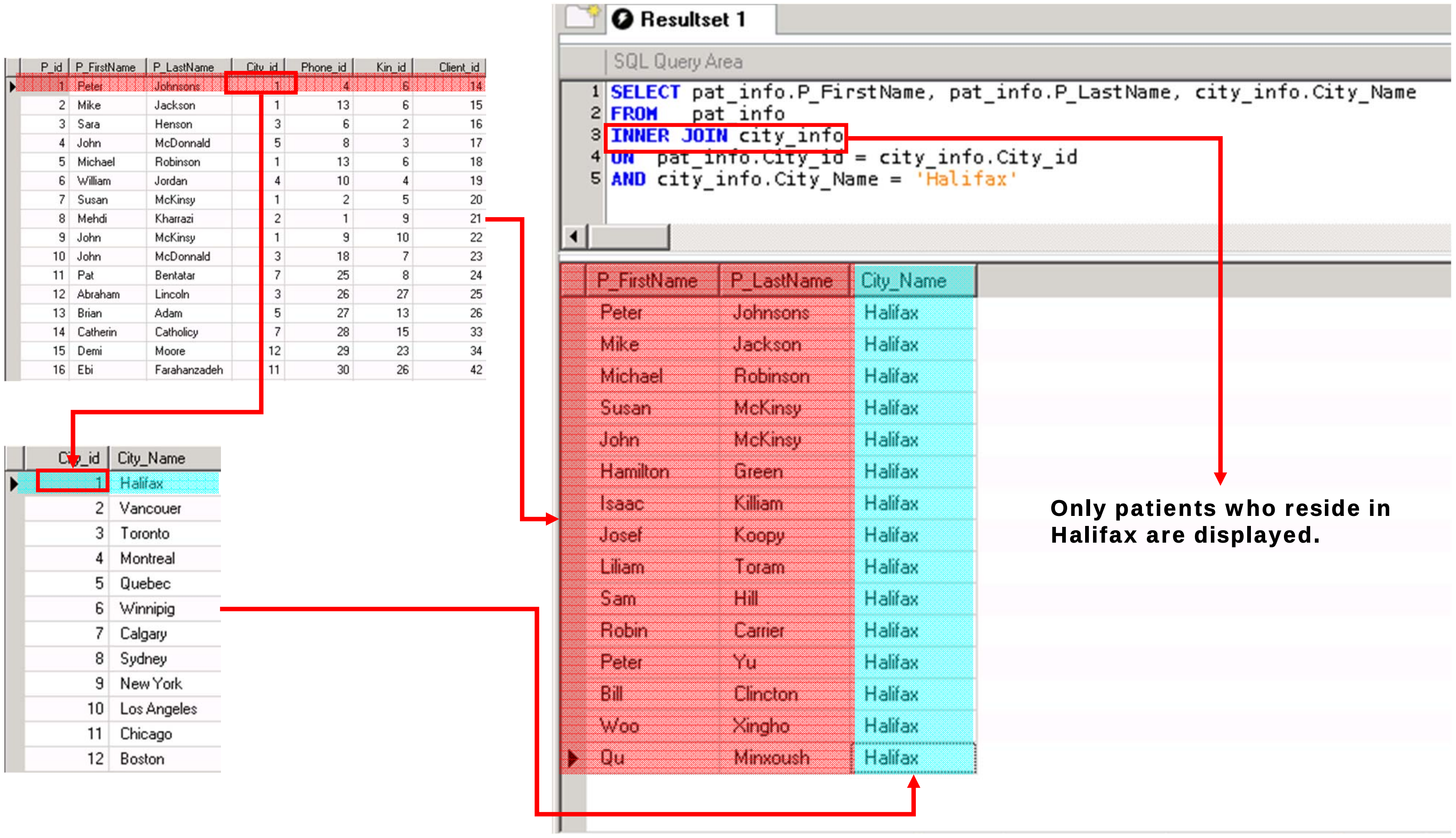
Table: Patient_information + City_information

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	Kharrazi	2	1	9	21
9	John	McKinsy	1	9	10	22
10	John	McDonnald	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

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

```sql
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'
```

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
Liliam	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.

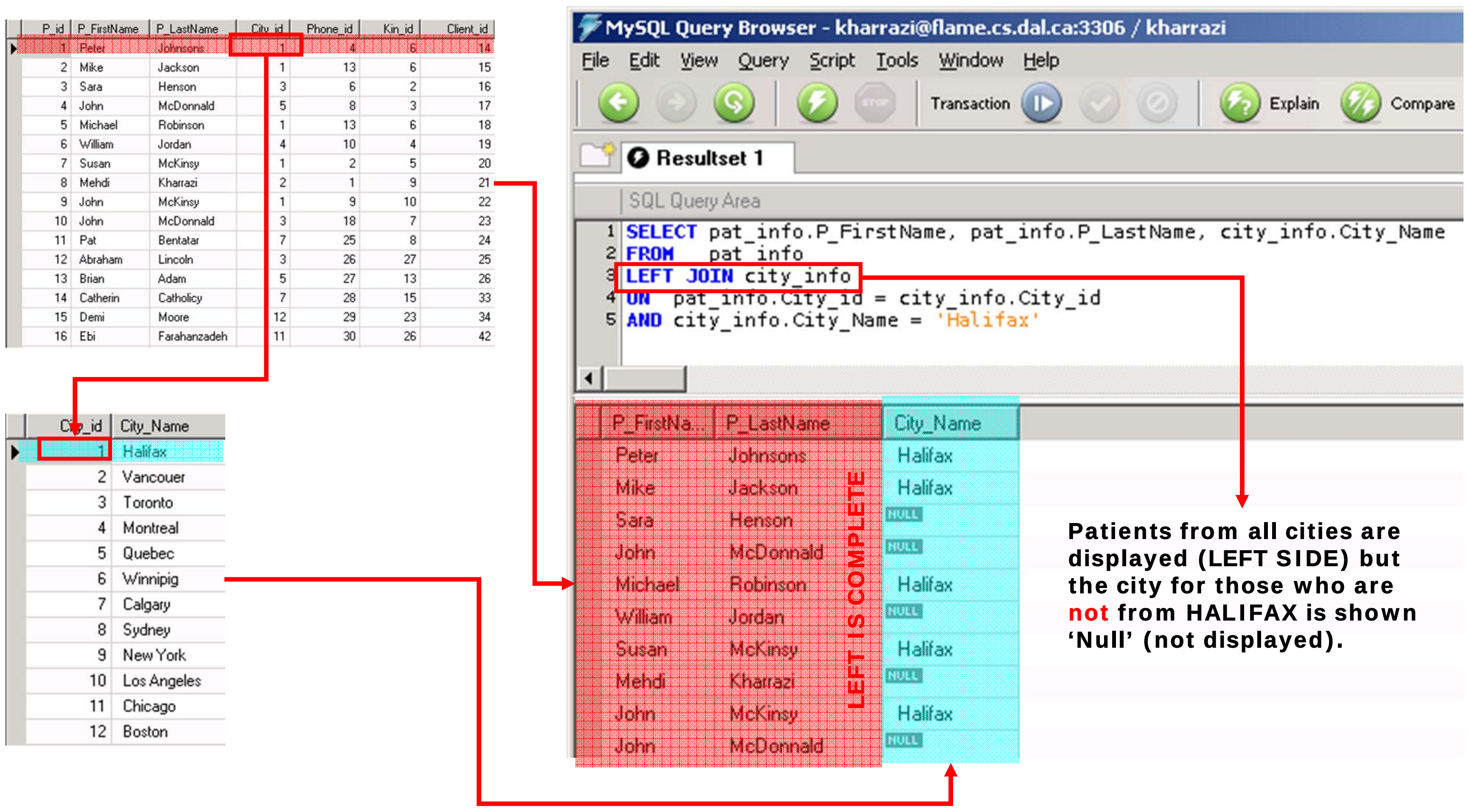
Table: Patient_information + City_information

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	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

City_id	City_Name
1	Halifax
2	Vancouer
3	Toronto
4	Montreal
5	Quebec
6	Winnipig
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

```sql
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'
```

P_FirstNa...	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

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

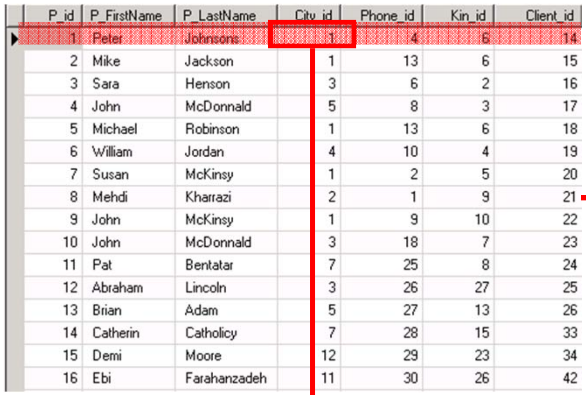
Table: Patient_information + City_information

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

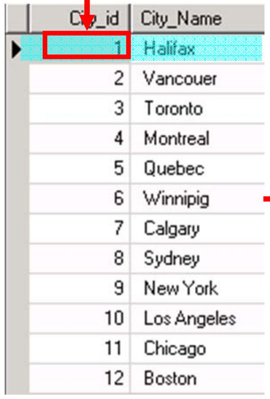
- Examples:

```
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'
```

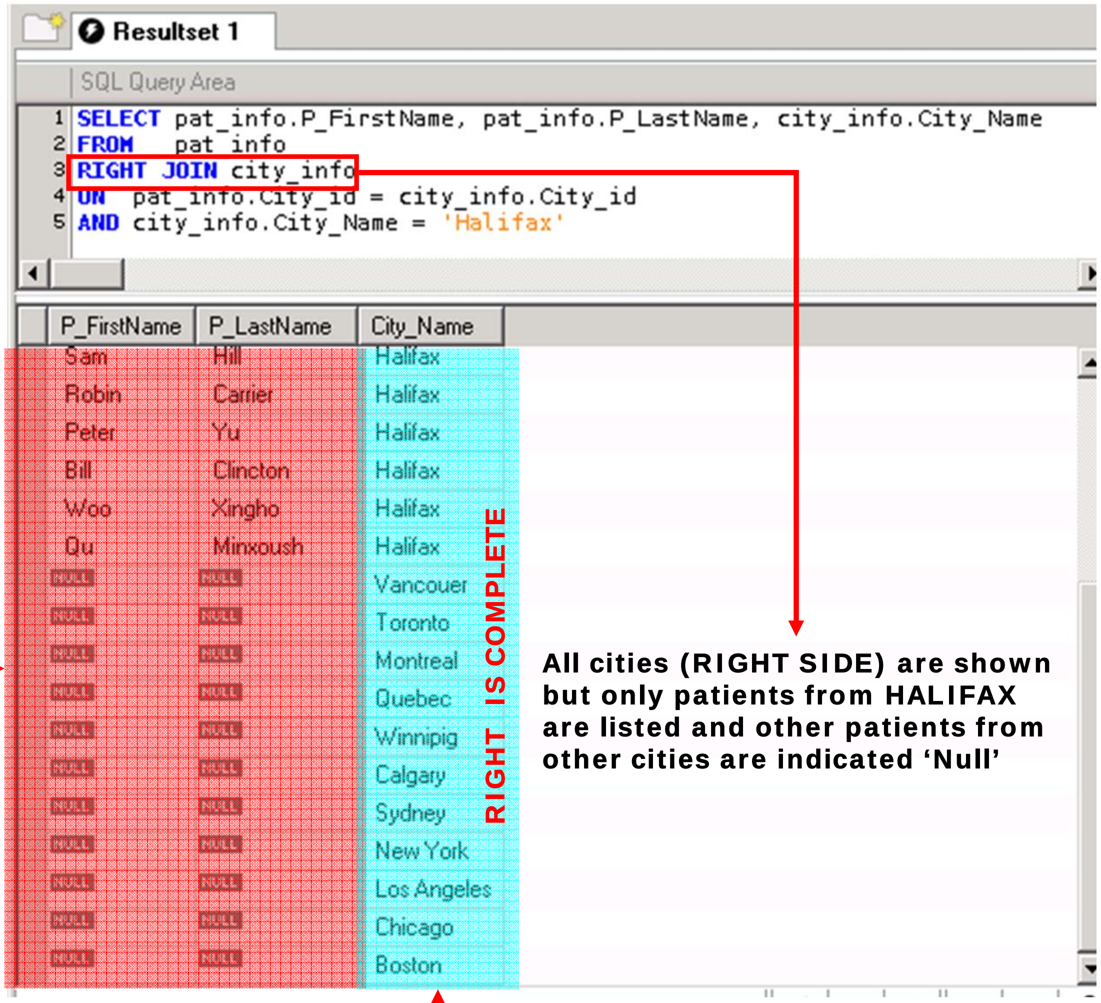
SQL: Joining and Keys (1 - Many) (cont.)



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	Kharrazi	2	1	9	21
9	John	McKinsy	1	9	10	22
10	John	McDonnald	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



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



```sql
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'
```

P_FirstName	P_LastName	City_Name
Sam	Hill	Halifax
Robin	Carrier	Halifax
Peter	Yu	Halifax
Bill	Clincton	Halifax
Woo	Xingho	Halifax
Qu	Minxoush	Halifax
NULL	NULL	Vancouer
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'

Table: Patient_information + City_information

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

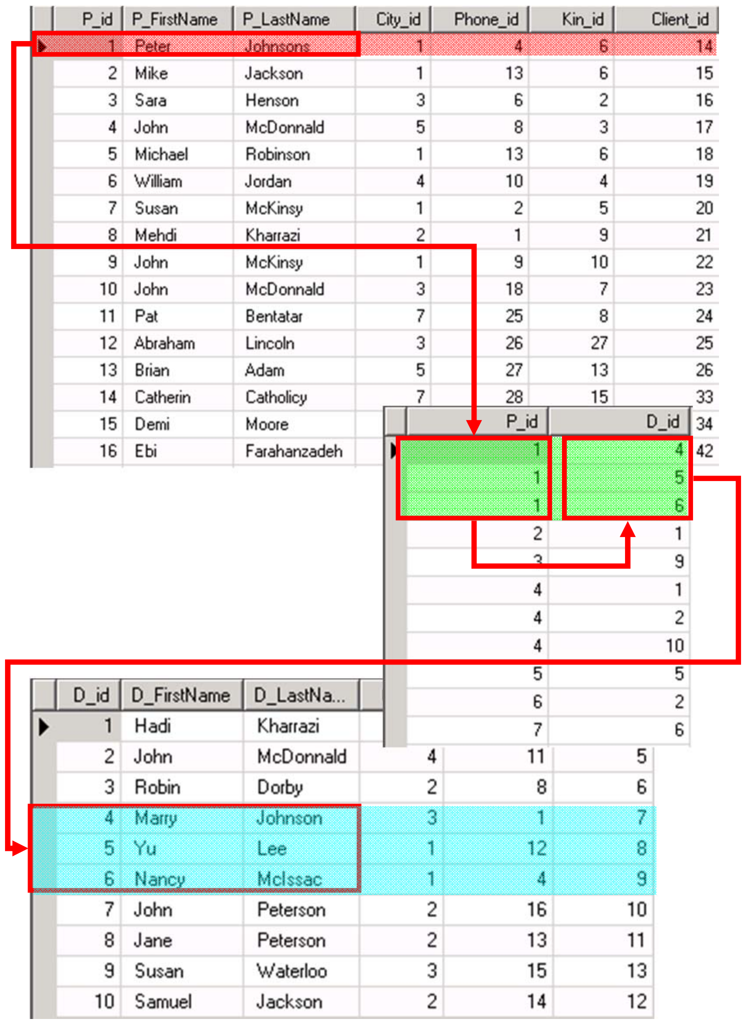
- Syntax:

```
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
```

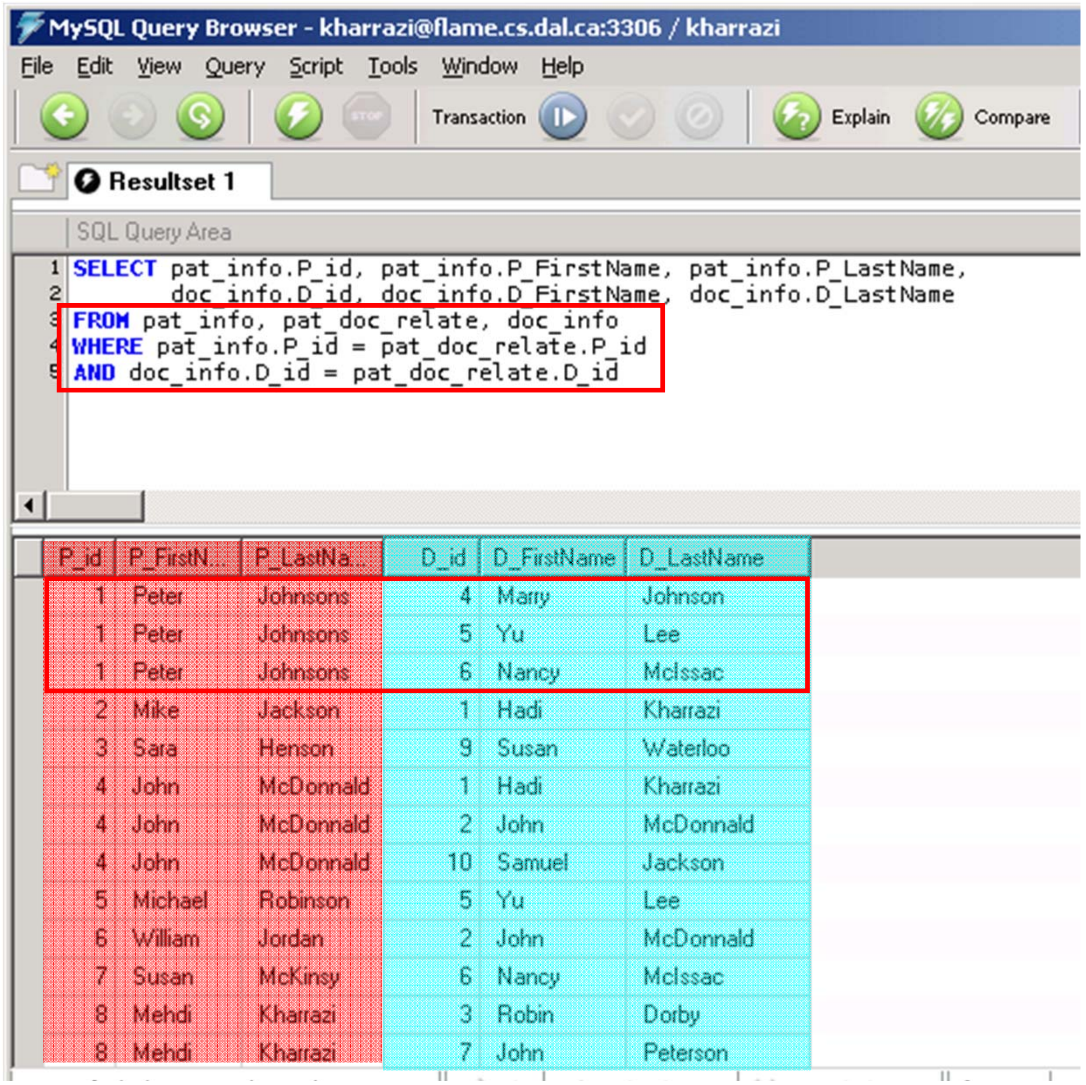
SQL: Joining and Keys (Many - Many) (cont.)



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	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	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				34
16	Ebi	Farahanzadeh				42

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_LastNa...			
1	Hadi	Kharrazi			
2	John	McDonnald	4	11	5
3	Robin	Dorby	2	8	6
4	Marry	Johnson	3	1	7
5	Yu	Lee	1	12	8
6	Nancy	McIssac	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


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

```
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
```

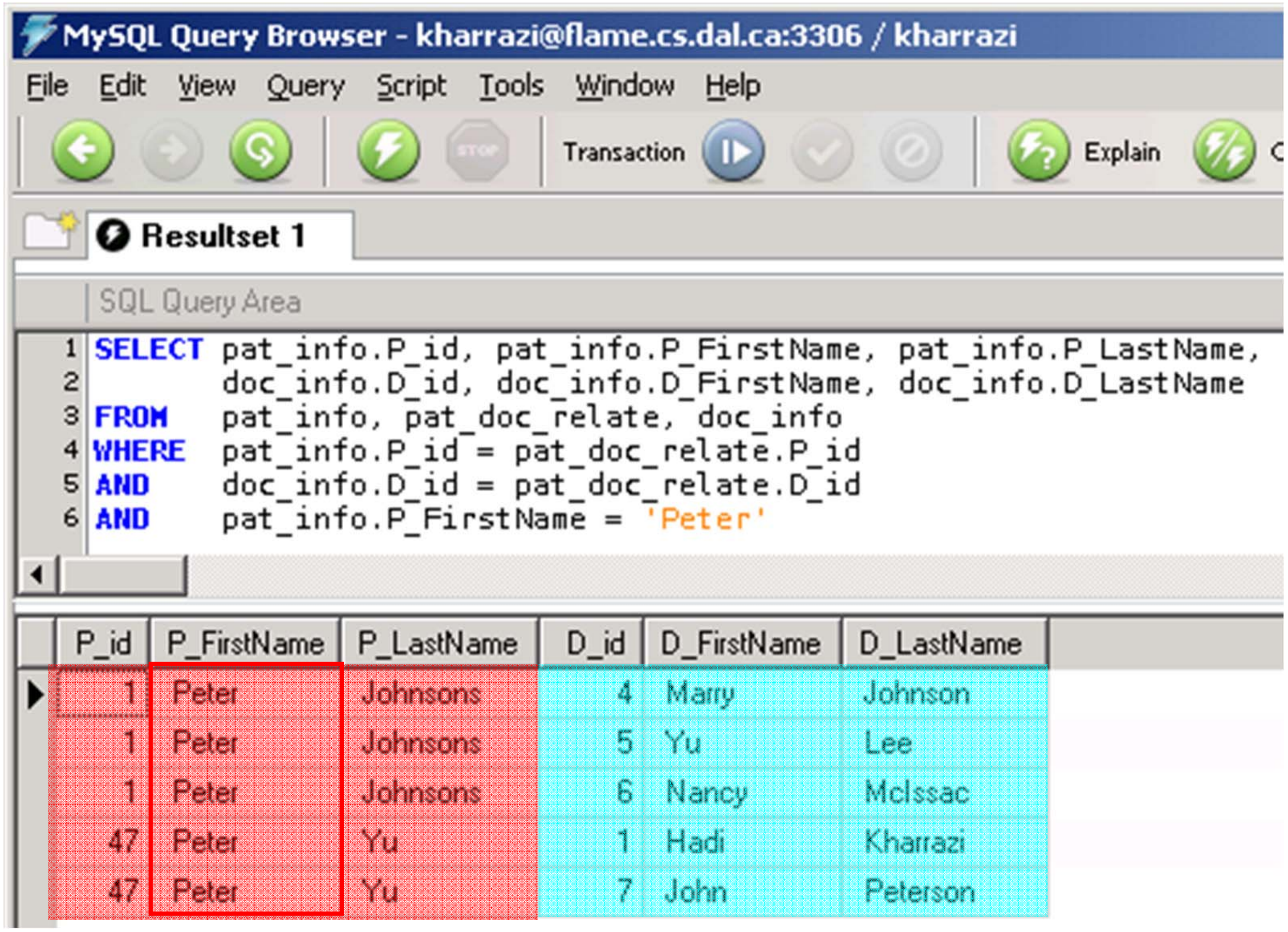
P_id	P_FirstN...	P_LastNa...	D_id	D_FirstName	D_LastName
1	Peter	Johnsons	4	Marry	Johnson
1	Peter	Johnsons	5	Yu	Lee
1	Peter	Johnsons	6	Nancy	McIssac
2	Mike	Jackson	1	Hadi	Kharrazi
3	Sara	Henson	9	Susan	Waterloo
4	John	McDonnald	1	Hadi	Kharrazi
4	John	McDonnald	2	John	McDonnald
4	John	McDonnald	10	Samuel	Jackson
5	Michael	Robinson	5	Yu	Lee
6	William	Jordan	2	John	McDonnald
7	Susan	McKinsy	6	Nancy	McIssac
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

```
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'
```

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

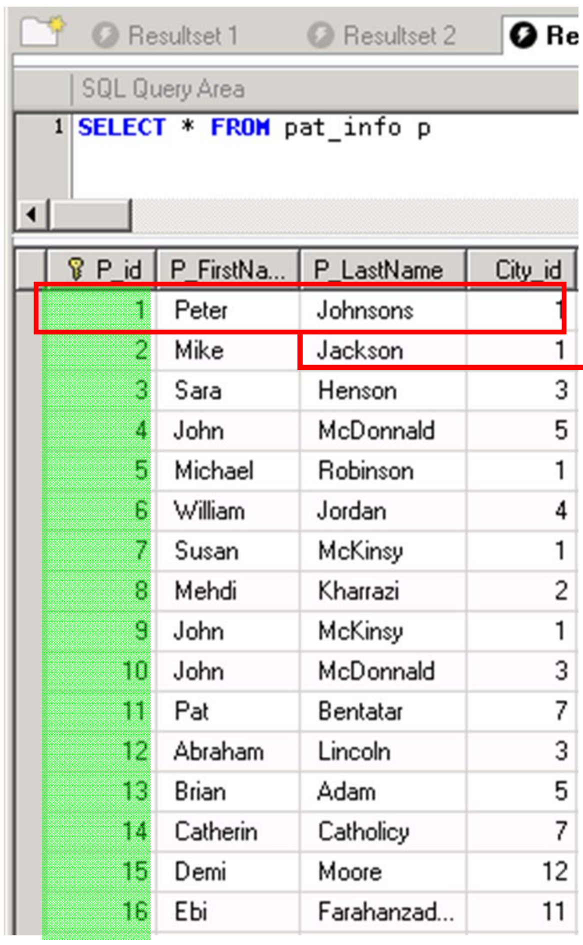
- Syntax:

```
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
```

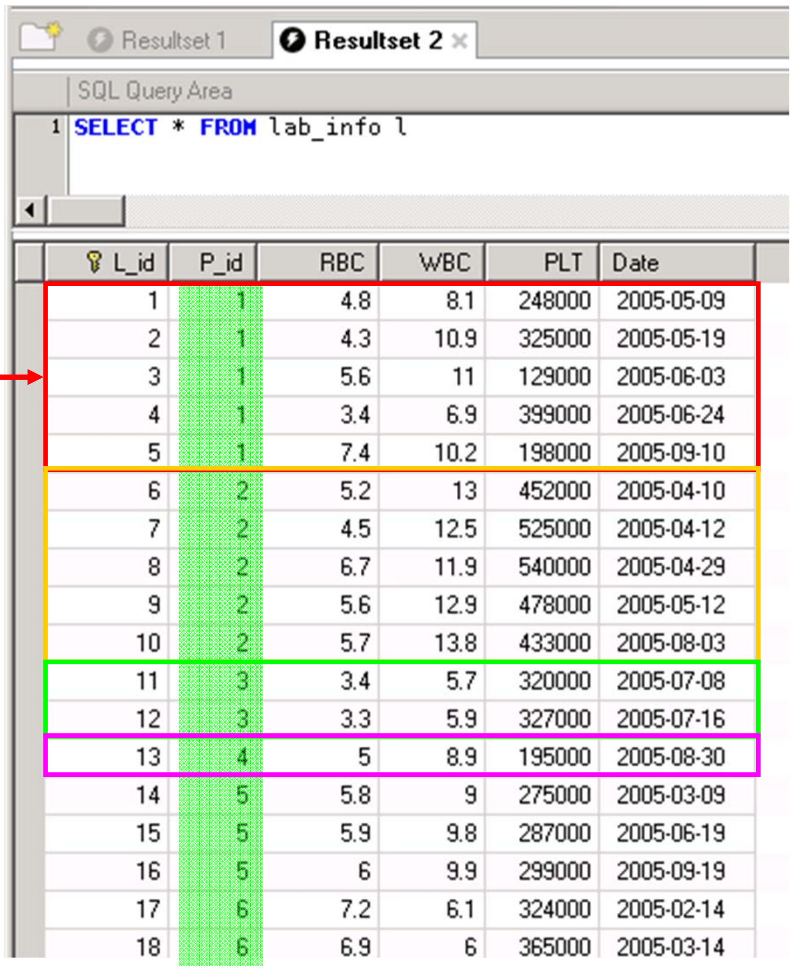
SQL: GROUP BY & HAVING (cont.)



```
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	McDonnald	5
5	Michael	Robinson	1
6	William	Jordan	4
7	Susan	McKinsy	1
8	Mehdi	Kharrazi	2
9	John	McKinsy	1
10	John	McDonnald	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

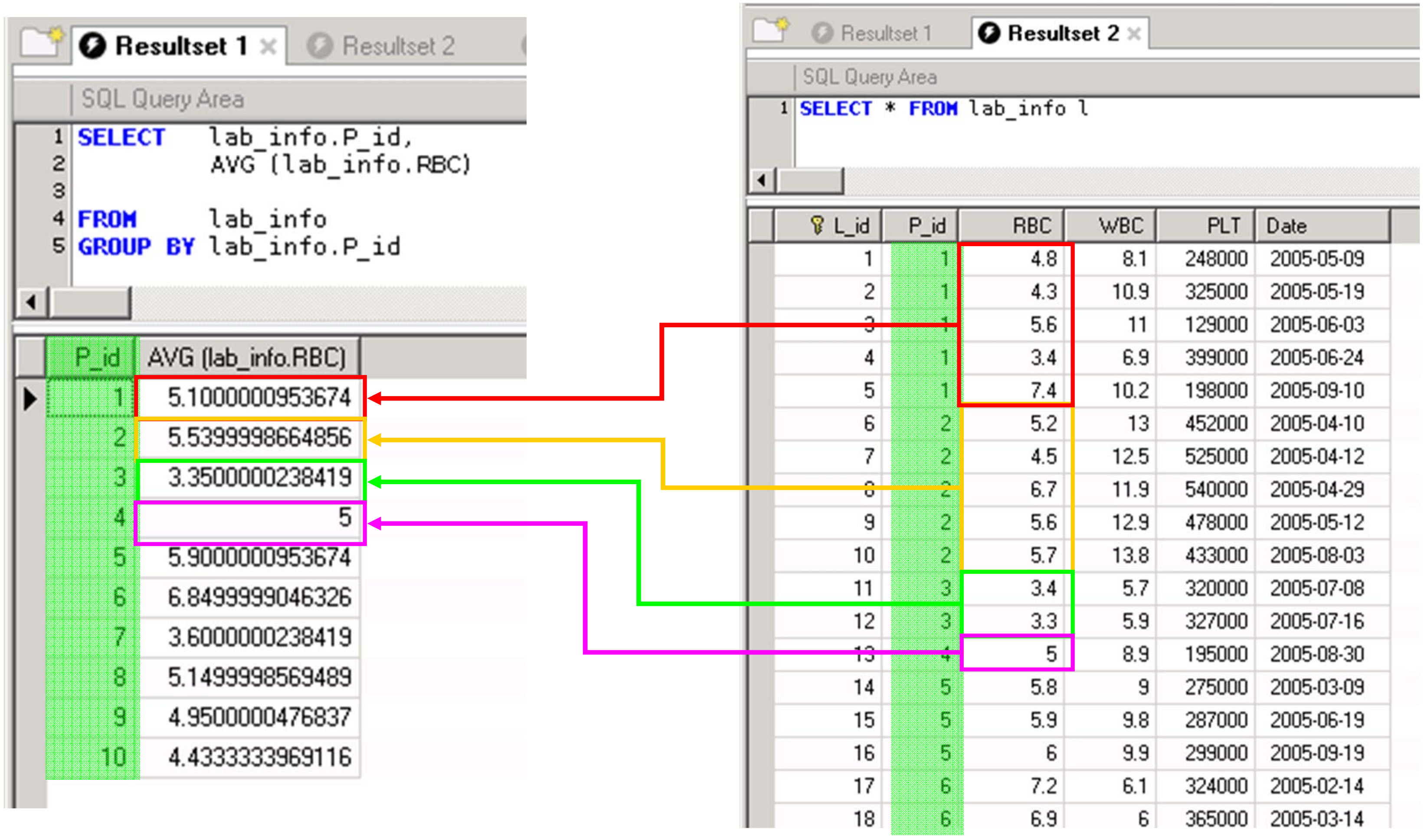


```
SELECT * FROM lab_info l
```

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.)



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

FROM       lab_info
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

```
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.)

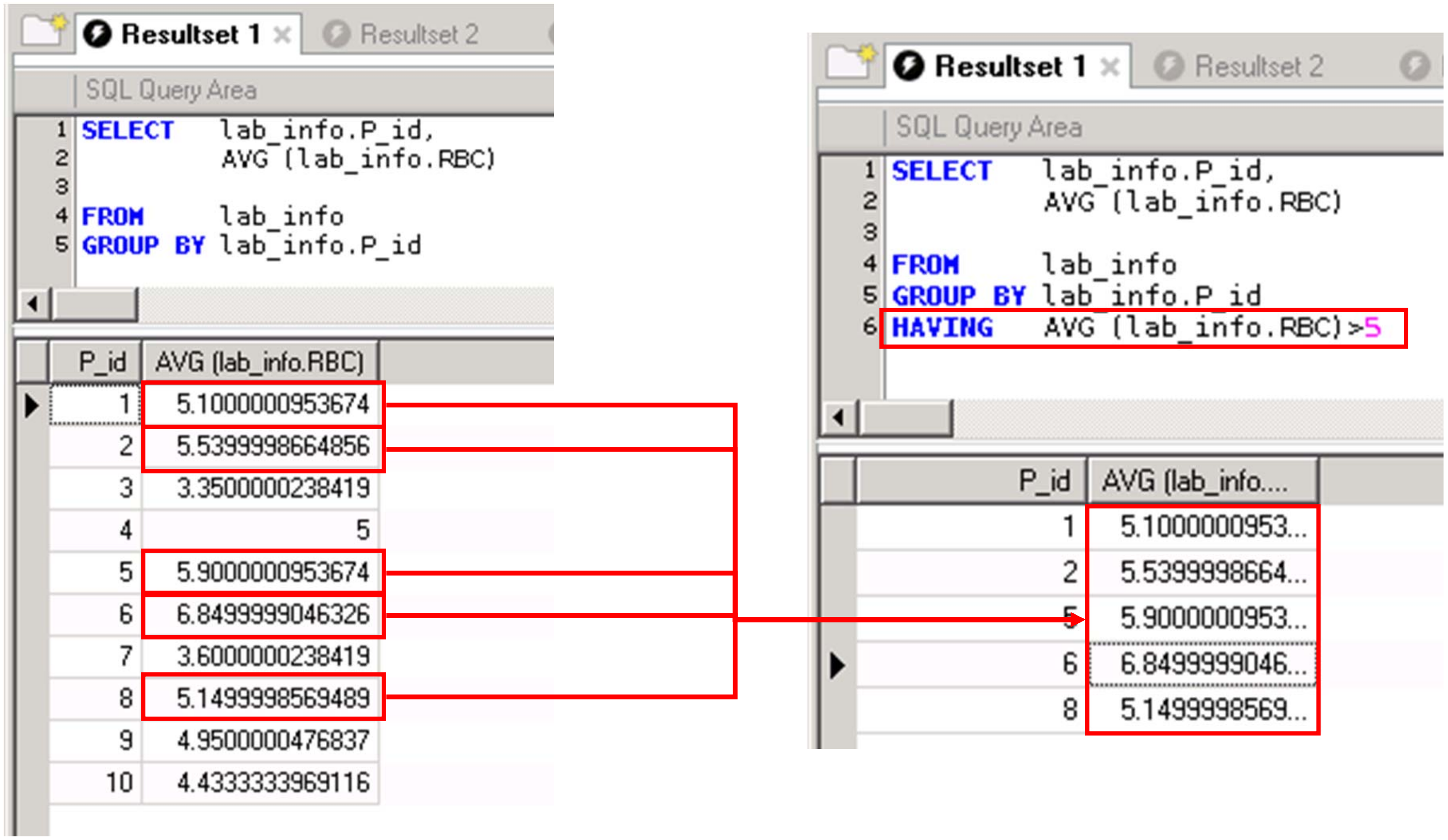
- Syntax:

```
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.)



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

FROM      lab_info
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

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

FROM      lab_info
GROUP BY  lab_info.P_id
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 **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.)

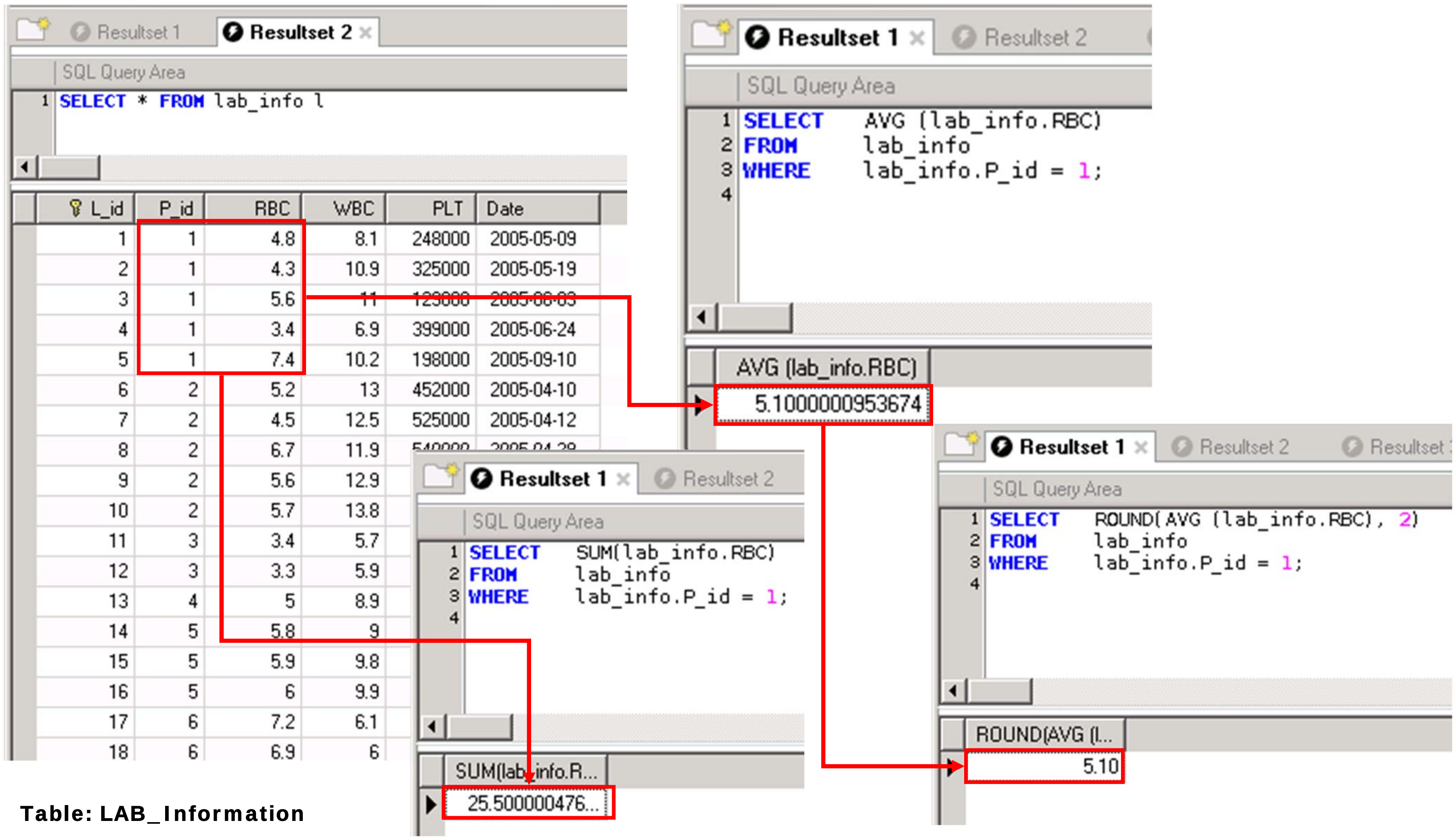
- Syntax:

```
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)*



```
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	2005-06-03
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	2005-04-28
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		

Table: LAB_Information

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

```
AVG (lab_info.RBC)
5.1000000953674
```

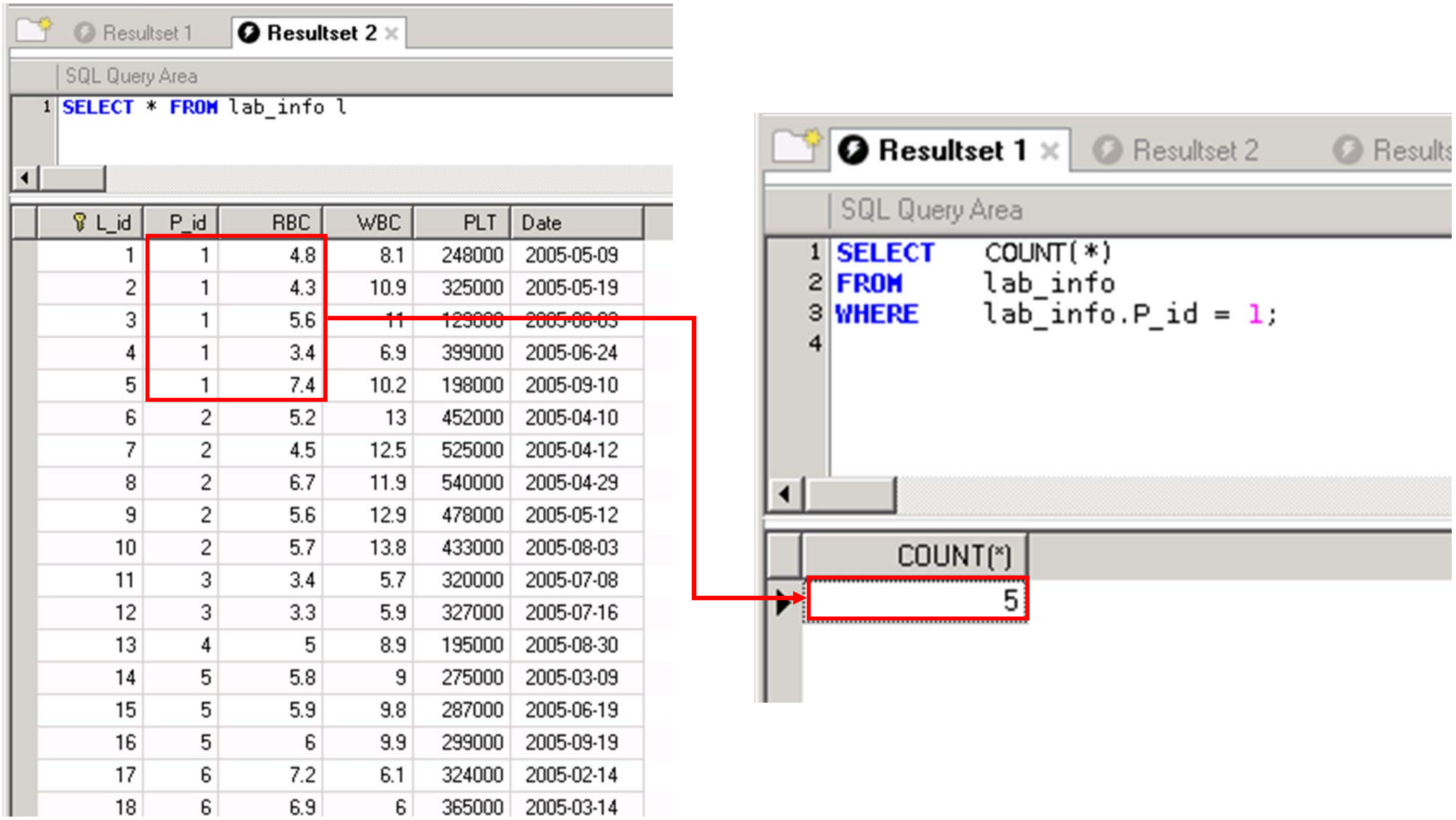
```
SELECT   SUM(lab_info.RBC)
FROM     lab_info
WHERE    lab_info.P_id = 1;
```

```
SUM(lab_info.R...
25.500000476...
```

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

```
ROUND(AVG (l...
5.10
```

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



```
SELECT * FROM lab_info l
```

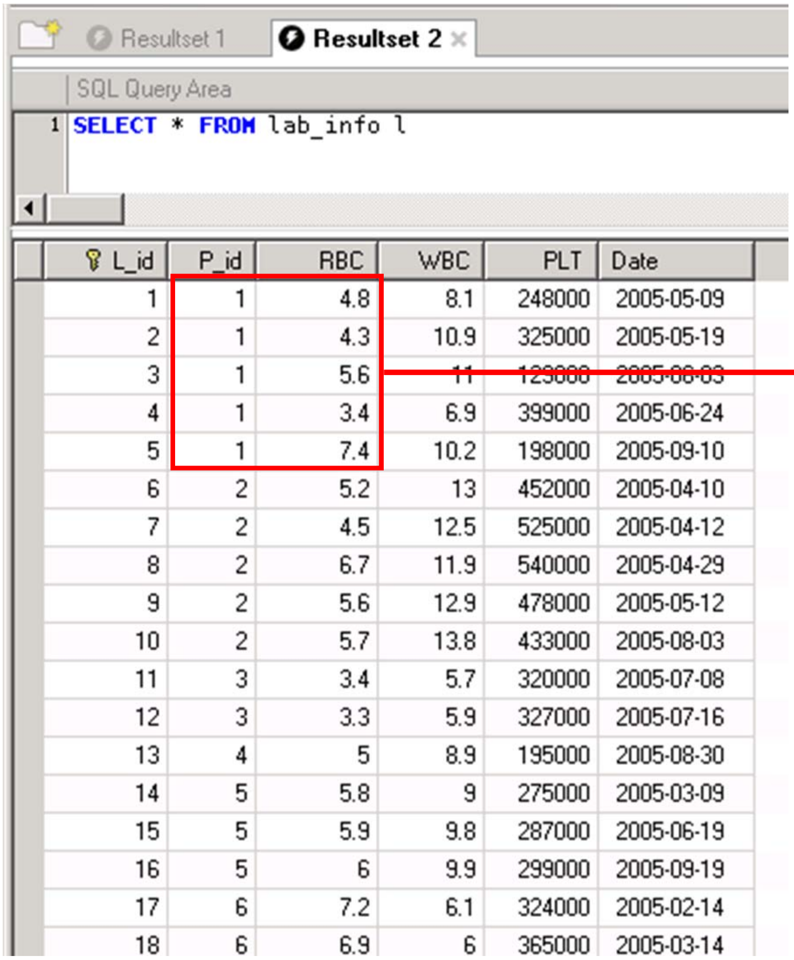
L_id	P_id	RBC	WBC	PLT	Date
1	1	4.8	8.1	248000	2005-05-09
2	1	4.3	10.9	325000	2005-05-19
3	1	5.6	11	123000	2005-06-03
4	1	3.4	6.9	399000	2005-06-24
5	1	7.4	10.2	198000	2005-09-10
6	2	5.2	13	452000	2005-04-10
7	2	4.5	12.5	525000	2005-04-12
8	2	6.7	11.9	540000	2005-04-29
9	2	5.6	12.9	478000	2005-05-12
10	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

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

COUNT(*)
5

Table: LAB_Information

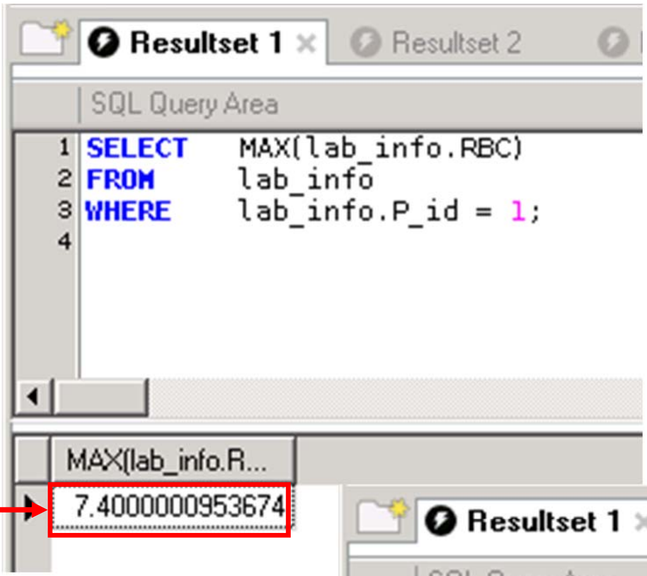
SQL: Functions (cont.) *(MIN, MAX)*



```
SELECT * FROM lab_info l
```

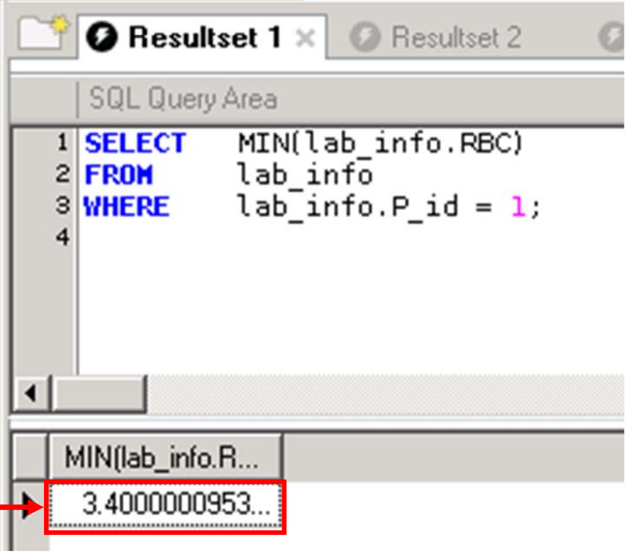
L_id	P_id	RBC	WBC	PLT	Date
1	1	4.8	8.1	248000	2005-05-09
2	1	4.3	10.9	325000	2005-05-19
3	1	5.6	11	123000	2005-08-03
4	1	3.4	6.9	399000	2005-06-24
5	1	7.4	10.2	198000	2005-09-10
6	2	5.2	13	452000	2005-04-10
7	2	4.5	12.5	525000	2005-04-12
8	2	6.7	11.9	540000	2005-04-29
9	2	5.6	12.9	478000	2005-05-12
10	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

Table: LAB_Information



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

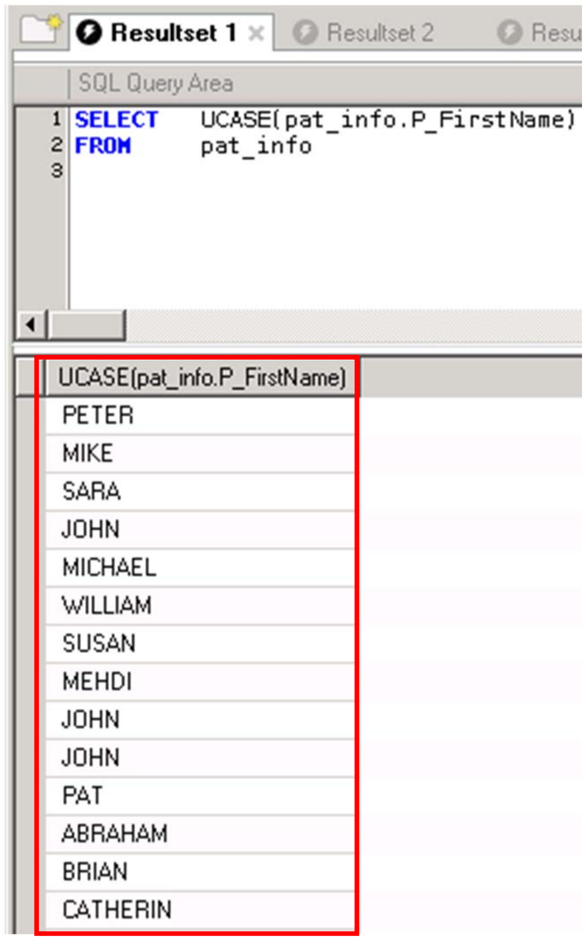
```
MAX(lab_info.R...
7.4000000953674
```



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

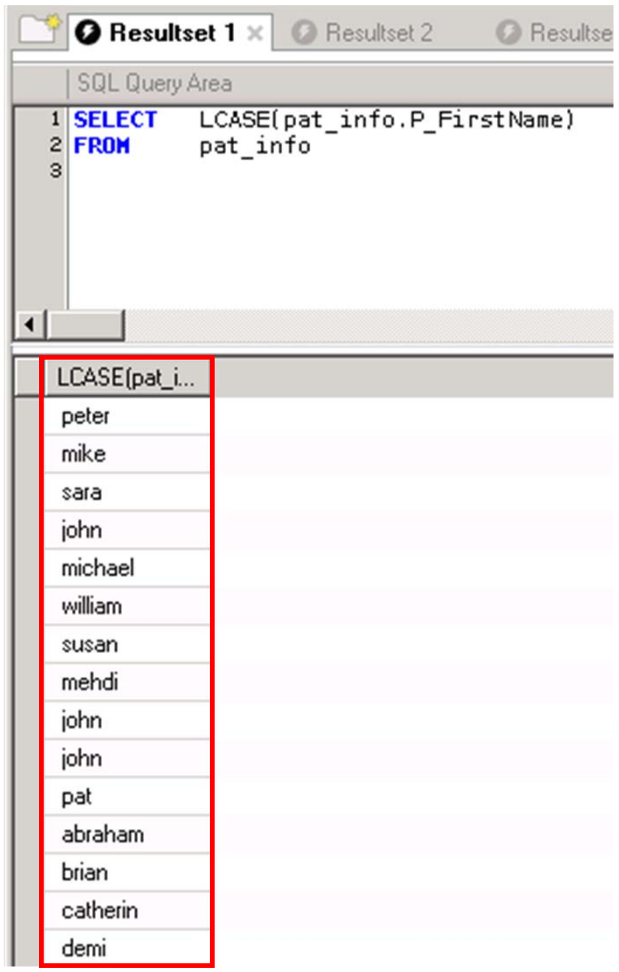
```
MIN(lab_info.R...
3.4000000953...
```

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



```
SELECT    UCASE(pat_info.P_FirstName)
FROM      pat_info
```

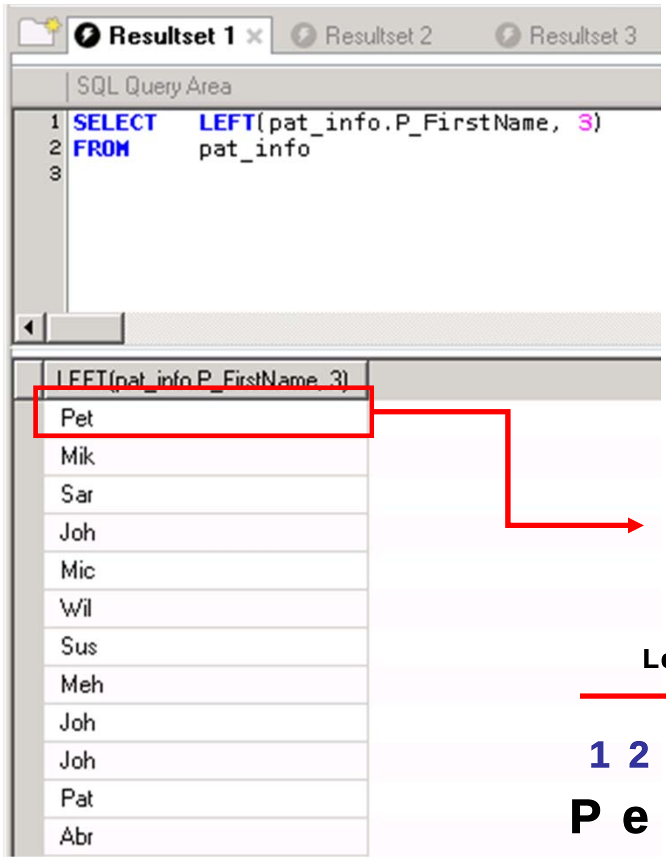
UCASE(pat_info.P_FirstName)
PETER
MIKE
SARA
JOHN
MICHAEL
WILLIAM
SUSAN
MEHDI
JOHN
JOHN
PAT
ABRAHAM
BRIAN
CATHERIN



```
SELECT    LCASE(pat_info.P_FirstName)
FROM      pat_info
```

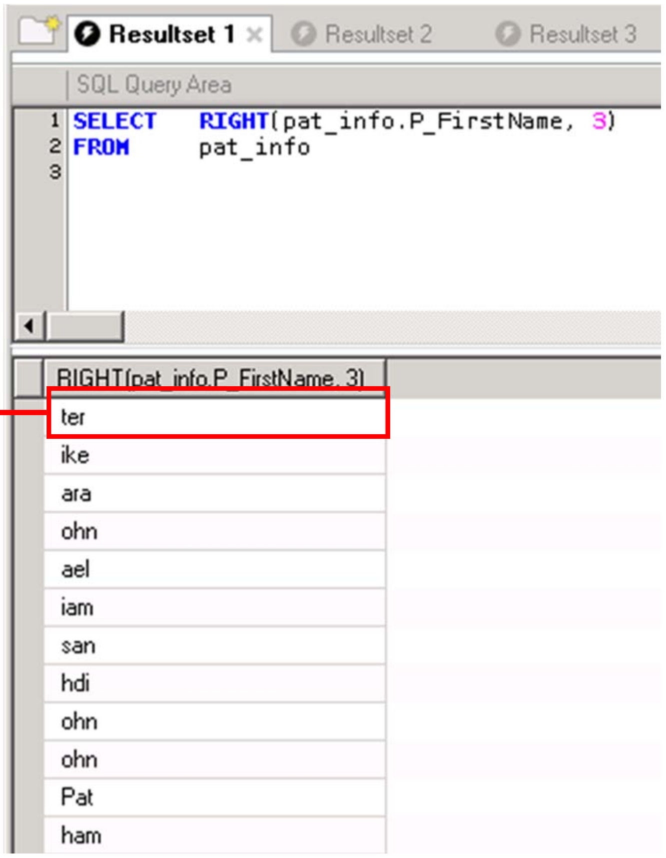
LCASE(pat_i...
peter
mike
sara
john
michael
william
susan
mehdi
john
john
pat
abraham
brian
catherin
demi

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



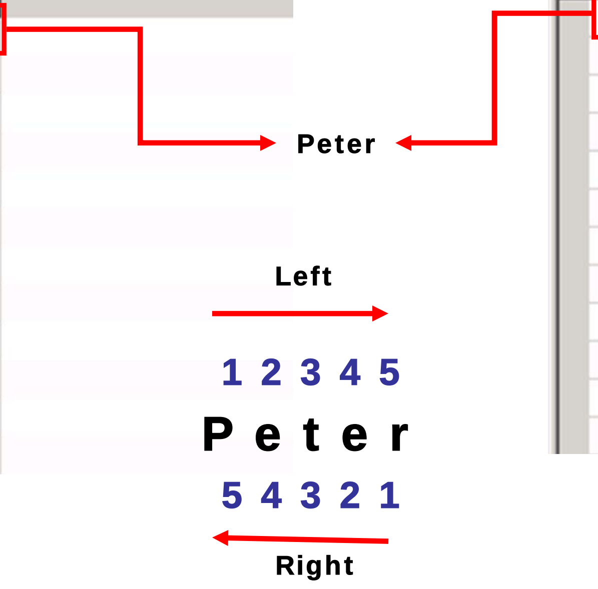
```
SELECT    LEFT(pat_info.P_FirstName, 3)
FROM      pat_info
```

LEFT(pat_info.P_FirstName, 3)
Pet
Mik
Sar
Joh
Mic
Wil
Sus
Meh
Joh
Joh
Pat
Abr

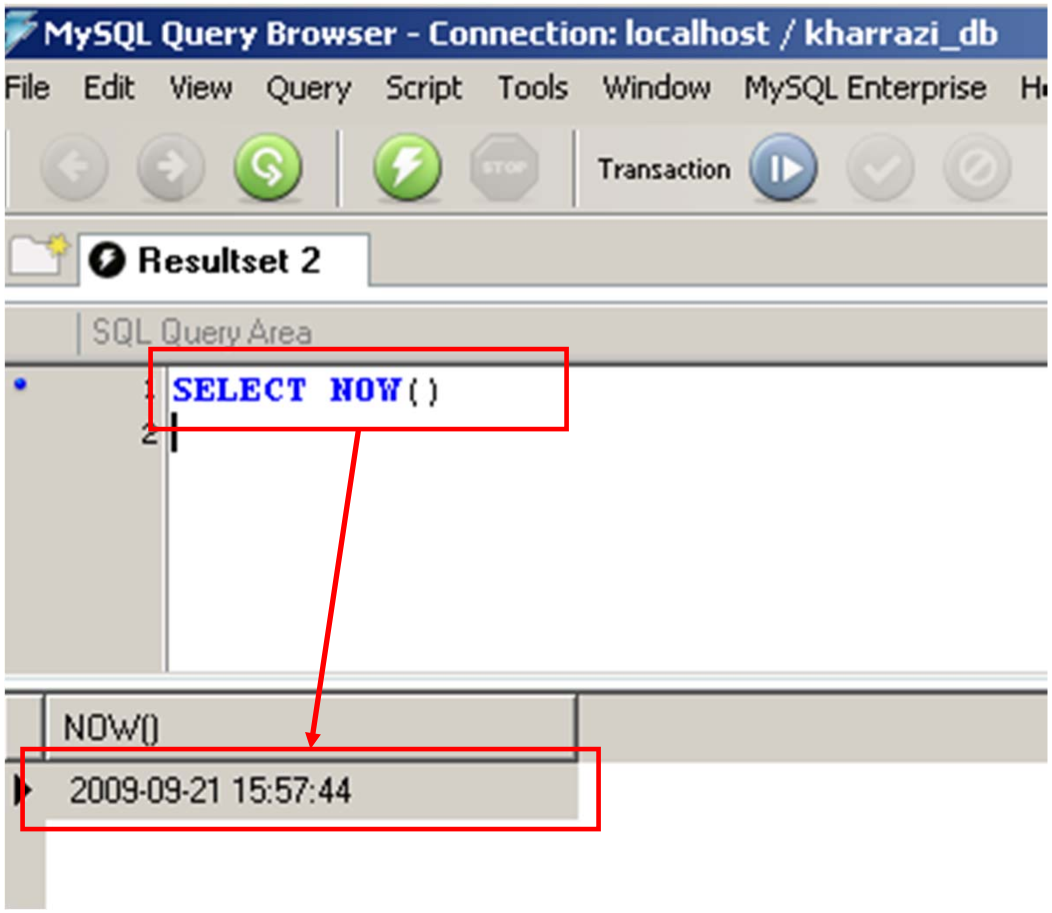


```
SELECT    RIGHT(pat_info.P_FirstName, 3)
FROM      pat_info
```

RIGHT(pat_info.P_FirstName, 3)
ter
ike
ara
ohn
ael
iam
san
hdi
ohn
ohn
Pat
ham



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

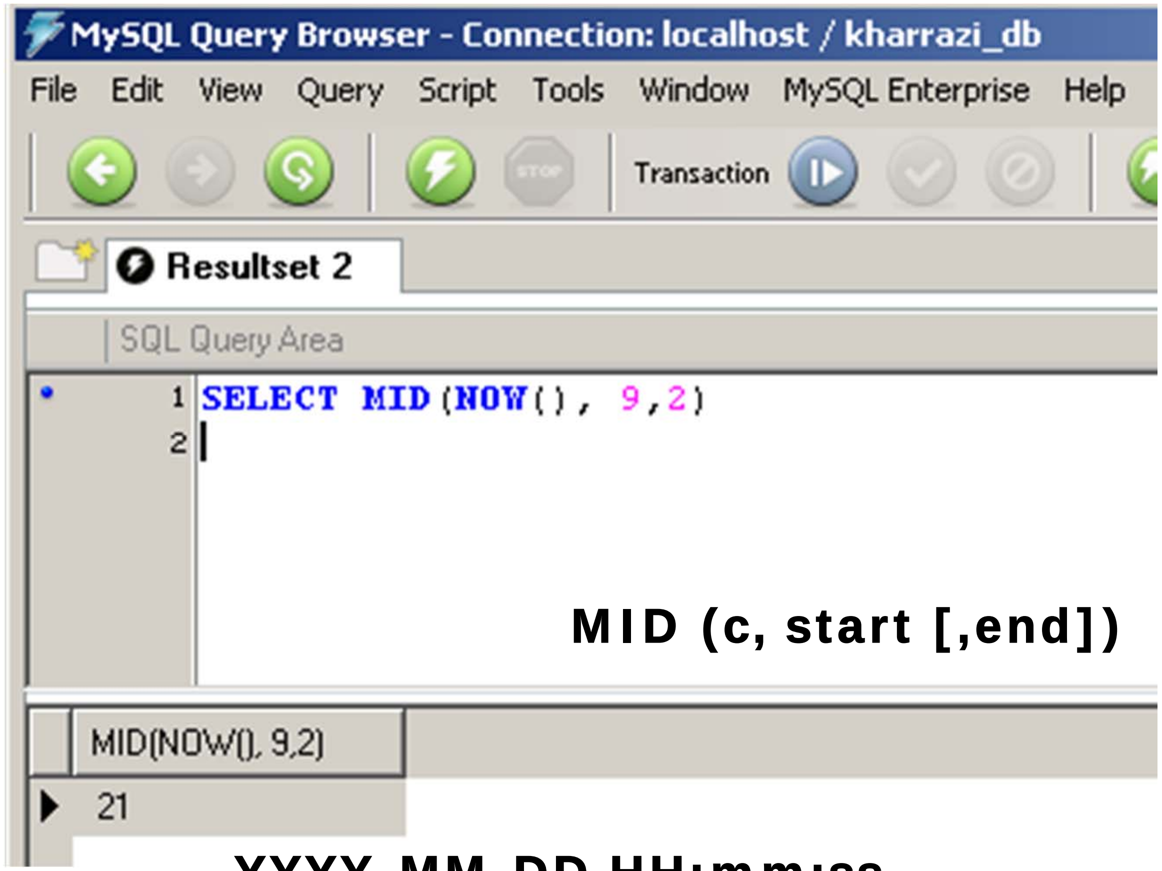

MySQL Query Browser - Connection: localhost / kharrazi_db
File Edit View Query Script Tools Window MySQL Enterprise
Transaction
Resultset 2
SQL Query Area

```
SELECT NOW()
```

```
NOW()
2009-09-21 15:57:44
```

YYYY-MM-DD HH:mm:ss

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



```
SELECT MID(NOW(), 9,2)
```

```
MID(NOW(), 9,2)
21
```

MID (c, start [,end])

YYYY-MM-DD HH:mm:ss
2009-09-21 …
12345678$\underrightarrow{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