



RC: 1337346

GR8TSON TECH. LTD

(think Big, start Small)

ICT SKILL ACQUISITION
ON
DATABASE DESIGN
&
MANAGEMENT

Contents

What is a database?	4
Database Management System (DBMS)	4
Advantages of a DBMS	4
Relational Database Management Systems (RDBMs)	5
Examples of relational Databases	5
Why Use MySQL Database:	5
RDBMS Terminology:	6
Creating a Database	7
Login using MySQL query browser	8
Tables and Data	12
Administrative MySQL Command:	15
a. INSERT INTO	15
1. Direct insert using edit	15
b. SELECT	19
WHERE clause	19
Operators in the WHERE clause	20
AND & OR operators	20
Example with the AND operator	21
Example with the OR operator	21
COMBINING AND & OR	21
THE ORDER BY KEYWORD	21
EXTENDED COMPARISON OPERATORS	21
THE IN OPERATOR	22
THE LIKE OPERATOR	23
INSERT STATEMENT	24
c. UPDATE	25
e. USE Databasename	26
f. SHOW DATABASES	26
g. SHOW TABLES	26
h. SHOW COLUMNS FROM tablename	26
i. SHOW INDEX FROM tablename	26

k. ALTER DATABASE	26
m. ALTER TABLE.....	26
n. DROP TABLE	26
The four main categories of SQL statements are as follows:	26
DML (Data Manipulation Language)	27
DDL (Data Definition Language).....	27
DCL (Data Control Language)	27
GRANT	27
Database Column Types.....	28
Text types:.....	28
MySQL Data Types	29
Number types:	30
TCL (Transaction Control Language)	33
An Introduction to Database Systems	33
Database Design for Mere Mortals.....	34
Design Versus Implementation.....	34
Normalized Design: Pros and Cons	34
Pros of Normalizing.....	35
Cons of Normalizing.....	35
Functional Dependency	36
Normal Forms	37
First Normal Form (1NF)	37
Second Normal Form (2NF).....	38
Third Normal Form (3NF).....	38
Higher Normal Forms.....	39
Database security.....	40
User privileges.....	40

What is a database?

- a. A database is a collection of data that is organized in such a way that it can easily be accessed, managed, and updated.
- b. A Database (DB) is structured such that it can store information about:
 - i. Multiple types of entities;
 - ii. The attributes that describe those entities; and
 - iii. The relationships among the entities
- c. A Database (DB) is collection of related data - with the following properties:
 - i. A DB is designed, built and populated with data for a specific purpose
 - ii. A DB represents some aspect of the real world.

Database Management System (DBMS)

A database management system is a collection of software programs that are used to define, construct, maintain and manipulate data in a database. Database System (DBS) contains:

- a. The Database;
- b. The DBMS; and
- c. Application Programs (what users interact with)

Advantages of a DBMS

A DBMS can provide:

- a. Data Consistency and Integrity - by controlling access and minimizing data duplication
- b. Application program independence - by storing data in a uniform fashion
- c. Data Sharing - by controlling access to data items, many users can access data concurrently
- d. Backup and Recovery
- e. Security and Privacy
- f. Multiple views of data

Relational Database Management Systems (RDBMs)

A database is a separate application that stores a collection of data. Each database has one or more distinct APIs for creating, accessing, managing, searching and replicating the data it holds.

Other kinds of data stores can be used, such as files on the file system or large hash tables in memory, but data fetching and writing would not be so fast and easy with those types of systems.

So nowadays, we use relational database management systems (RDBMS) to store and manage huge volume of data. This is called relational database because all the data is stored into different tables and relations are established using primary keys or other keys known as foreign keys.

A **Relational Database Management System (RDBMS)** is a software that:

- a. Enables you to implement a database with tables, columns and indexes.
- b. Guarantees the Referential Integrity between rows of various tables.
- c. Updates the indexes automatically.
- d. Interprets an SQL query and combines information from various tables.

Examples of relational Databases

Oracle, SQL server, Access, posgreSQL, MySQL

In this tutorial, we shall focus on MySQL which stands for My Structured Query Language.

Why Use MySQL Database:

MySQL is a fast, easy-to-use RDBMS being used for many small and big businesses. MySQL is becoming so popular because of many good reasons:

- a. MySQL is released under an open-source license. So you have nothing to pay to use it.
- b. MySQL is a very powerful program in its own right. It handles a large subset of the functionality of the most expensive and powerful database packages.
- c. MySQL uses a standard form of the well-known SQL data language.

- d. MySQL works on many operating systems and with many languages including PHP, PERL, C, C++, JAVA, etc.
- e. MySQL works very quickly and works well even with large data sets.
- f. MySQL is very friendly to PHP, the most appreciated language for web development.
- g. MySQL supports large databases, up to 50 million rows or more in a table. The default file size limit for a table is 4GB, but you can increase this (if your operating system can handle it) to a theoretical limit of 8 million terabytes (TB).
- h. MySQL is customizable. The open-source GPL license allows programmers to modify the MySQL software to fit their own specific environments.

RDBMS Terminology:

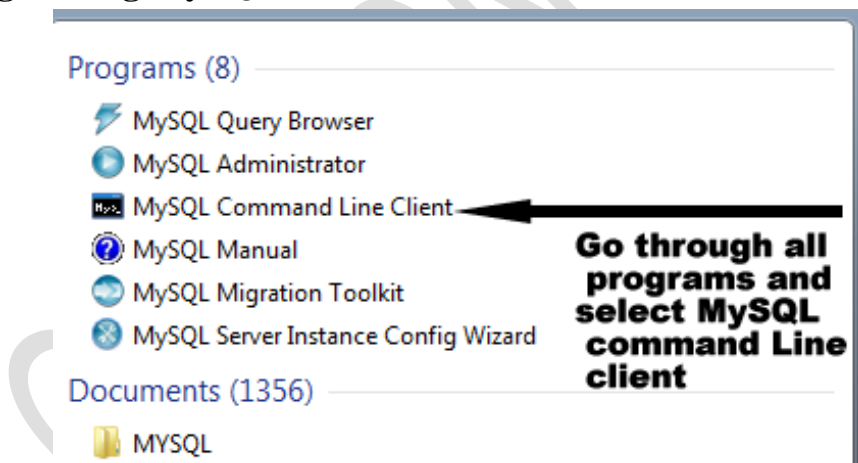
Before we proceed to explain MySQL database system, let's revise few definitions related to database.

1. **Database:** A database is a collection of tables, with related data.
2. **Table:** A table is made up of rows and columns. A table in a database looks like a simple spreadsheet (excel sheet).
3. **Column:** One column (also called *field*) contains data of one and the same kind, for example the column *staff_id*. It is vertical (runs from top to down) in a table.
4. **Row:** A row (also called *record*) is a group of related data, for example the data of one subscription.
5. **Primary Key:** A primary key is unique. A primary key value cannot occur twice in one table. With a primary key, you can find at most one row.
6. **Foreign Key:** A foreign key is the linking pin between two tables. It helps in joining tables together.
7. **Compound Key:** A compound key (composite key) is a key that consists of multiple columns, because one column is not sufficiently unique.
8. **Index:** An index in a database resembles an index at the back of a book.
9. **Referential Integrity:** Referential Integrity makes sure that a foreign key value always points to an existing row.

Creating a Database

In order to create a database, you have to first start up the database by going through the task bar to search for or searching and selecting the MySQL query browser or MySQL command line. Here, we are making use of the MySQL query browser even though we would give a guide on how to use the MySQL command line.

Login using MySQL command Line



When the MySQL command line interface opening the command line, the following interface comes up.

```
C:\Program Files (x86)\MySQL\MySQL Server 4.1\bin\mysql.exe
Enter password: *****
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 1867 to server version: 4.1.15-nt

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql> use demo;
Database changed
mysql> show tables;
+-----+
| Tables_in_demo |
+-----+
| customers      |
| timedemo       |
+-----+
2 rows in set (0.00 sec)

mysql> select * from timedemo;
+-----+-----+-----+-----+-----+
| ts          | dt          | d          | t          | y          |
+-----+-----+-----+-----+-----+
| 2017-01-08 13:13:15 | 2017-01-08 13:13:15 | 2017-01-08 | 13:13:15 | 2017       |
+-----+-----+-----+-----+-----+
1 row in set (0.00 sec)
```

Annotations in the image:

- Enter password**: Points to the password prompt.
- This command selects a particular database**: Points to `mysql> use demo;`
- This command displays all tables in the demo database**: Points to `mysql> show tables;`
- Select command result**: Points to the output of `select * from timedemo;`

This command line environment can be used to insert, select, update, create databases, and create tables, e t c.

Login using MySQL query browser

To login into the MySQL query browser, the following parameters are required:

Server host – Enter **localhost** for server host.

Username – Enter **root** for username

Password – Enter the **password** specified during the installation

Note that most times, server host and username are only specified at the first login. Subsequently, only the password may be specified at login.

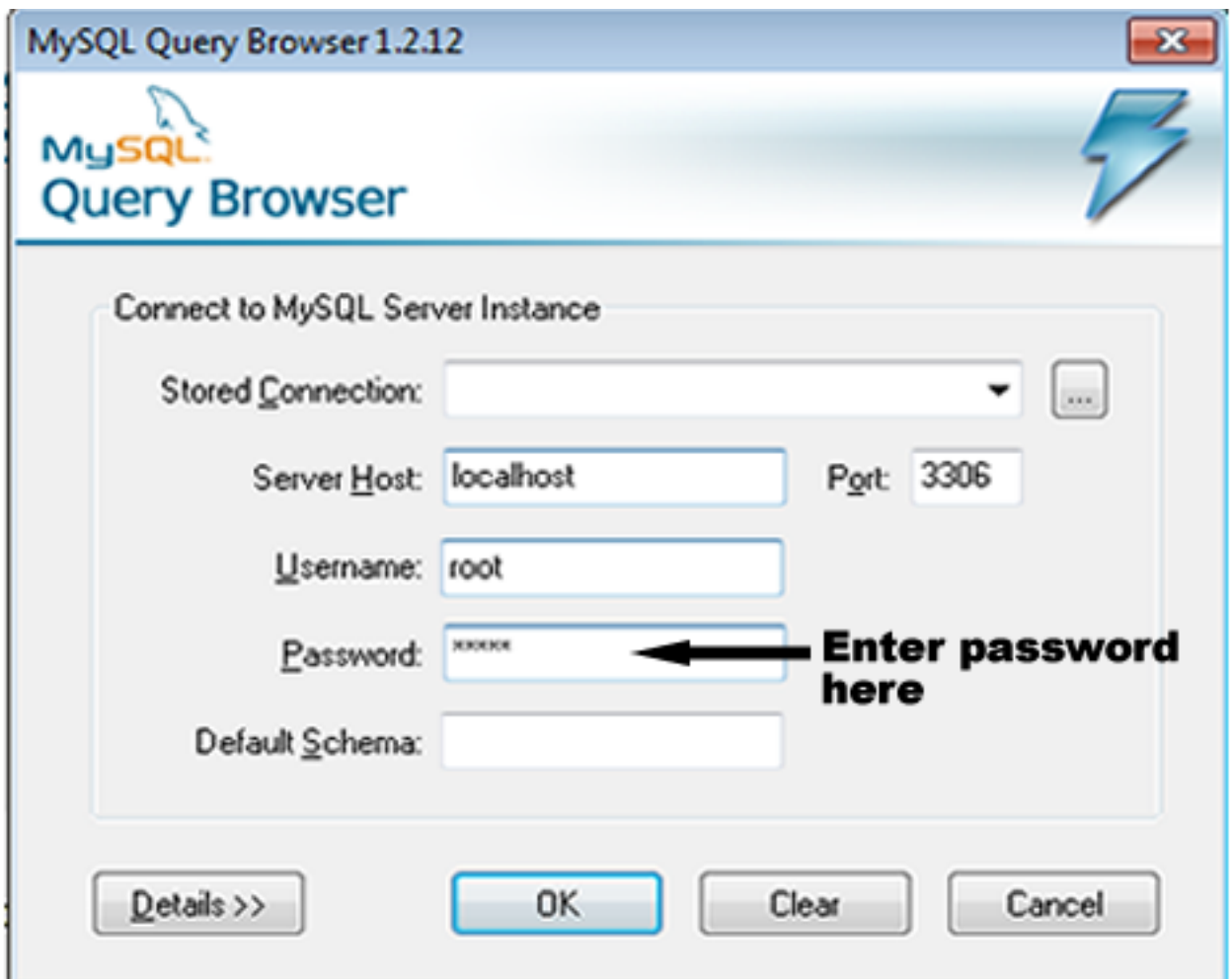


Figure 1. Login

To login, enter the MySQL password in the password field in the query browser and the database environment will open as shown in figure 2 below:

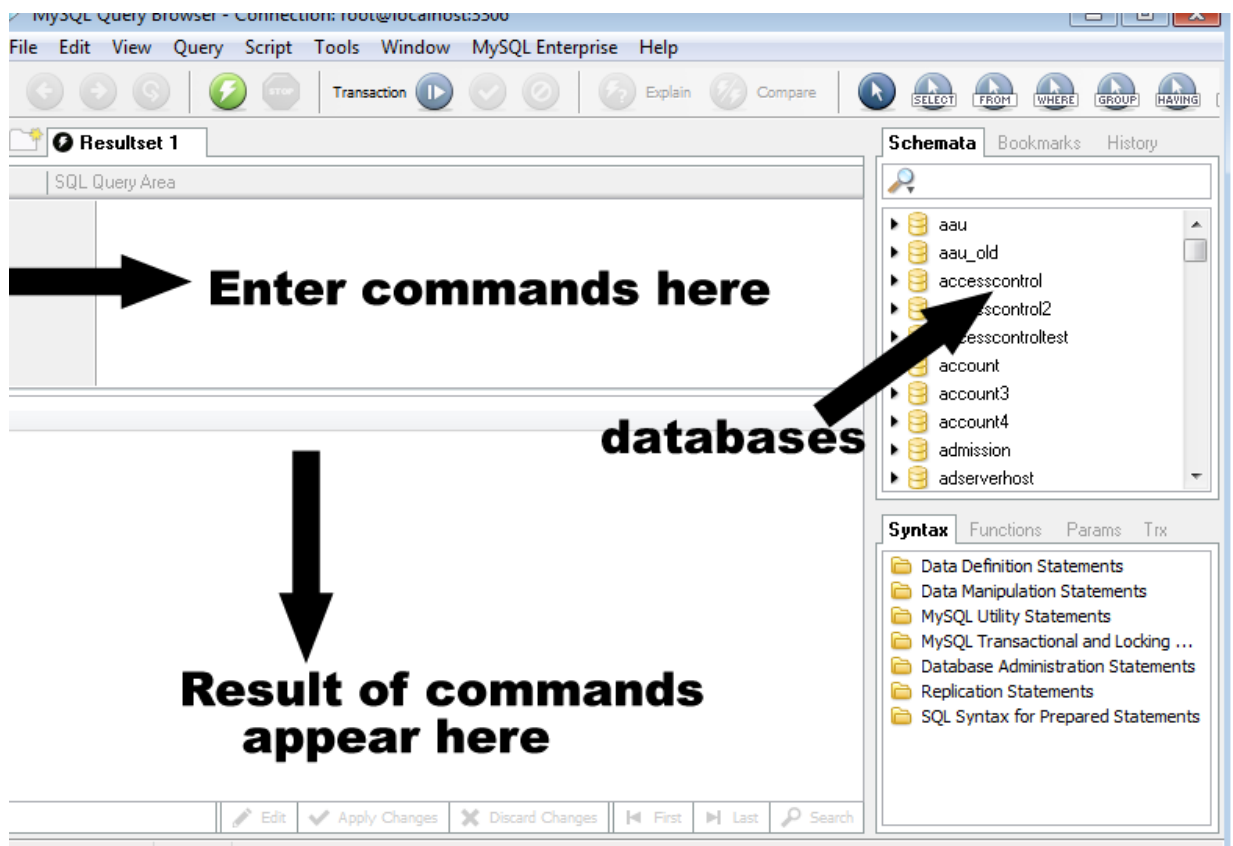


Figure 2: Open query browser

In order to create a database, right click on an existing database name or in the database area as shown in figure 3.

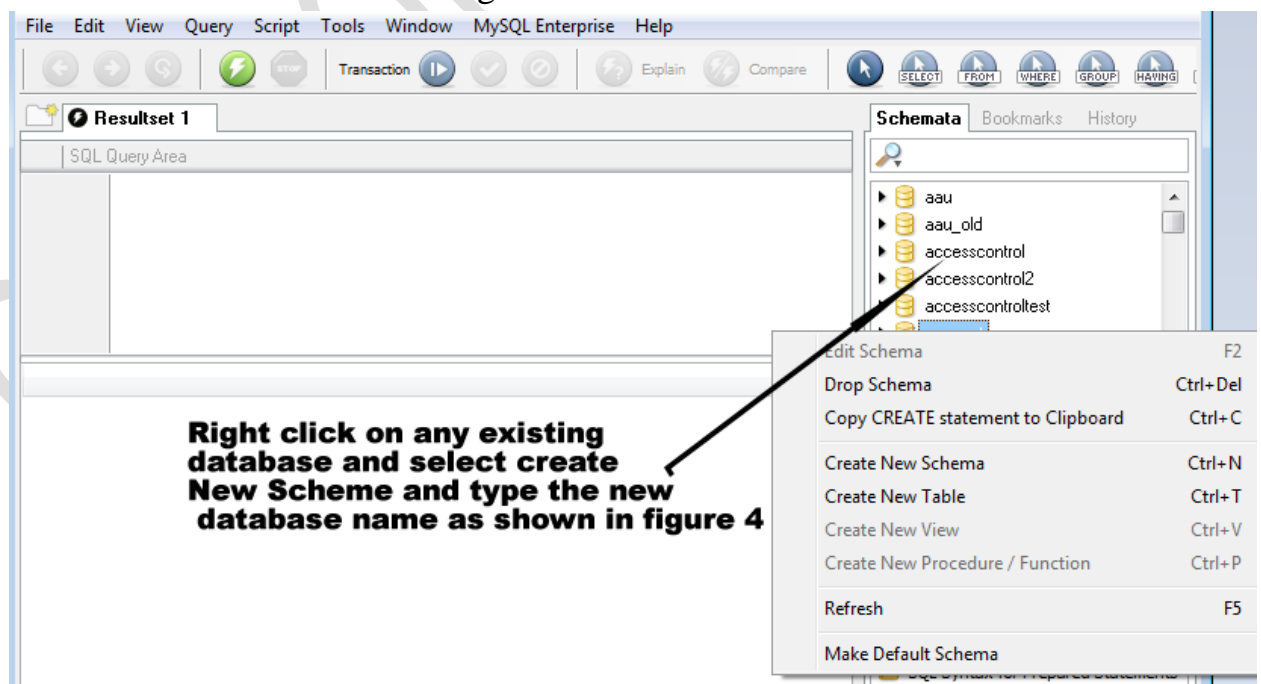


Figure 3: Right click on an existing database name

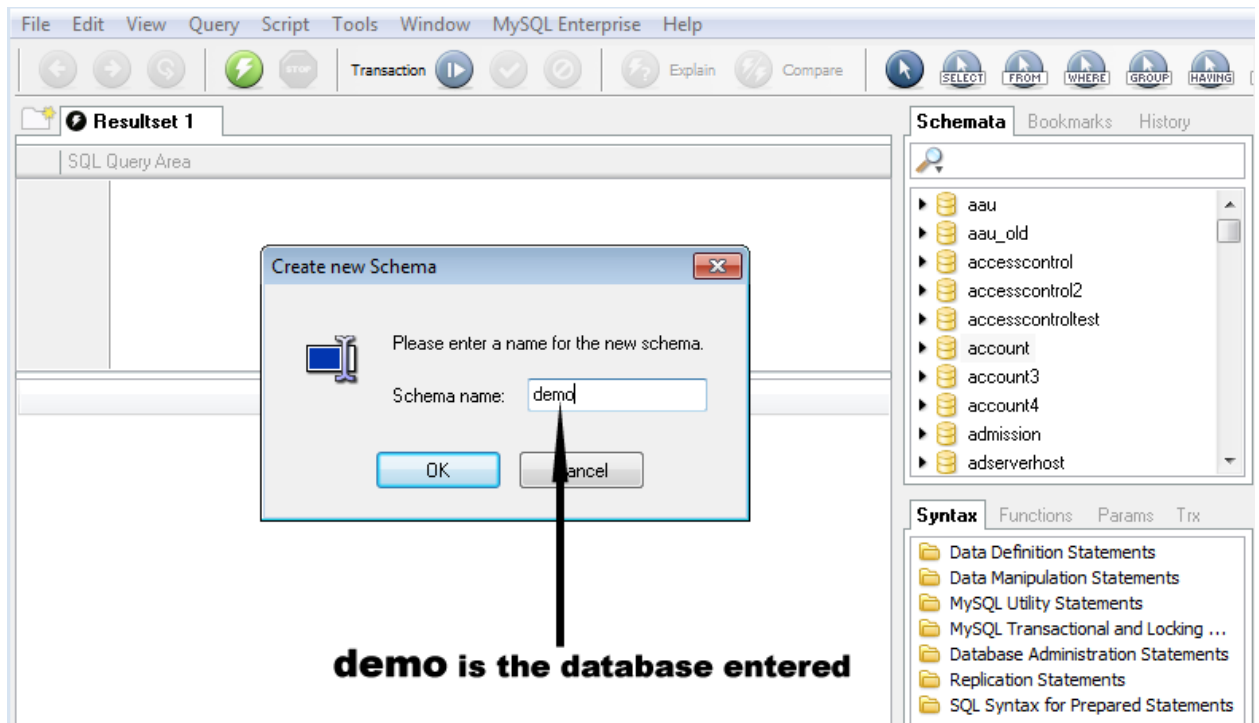


Figure 4: Database named **demo** been created

After typing the database name, click ok and the database name will be created. The name demo will hence be found among the database names in the database names area as shown in figure 5. Note that database names are arranged in ascending order.

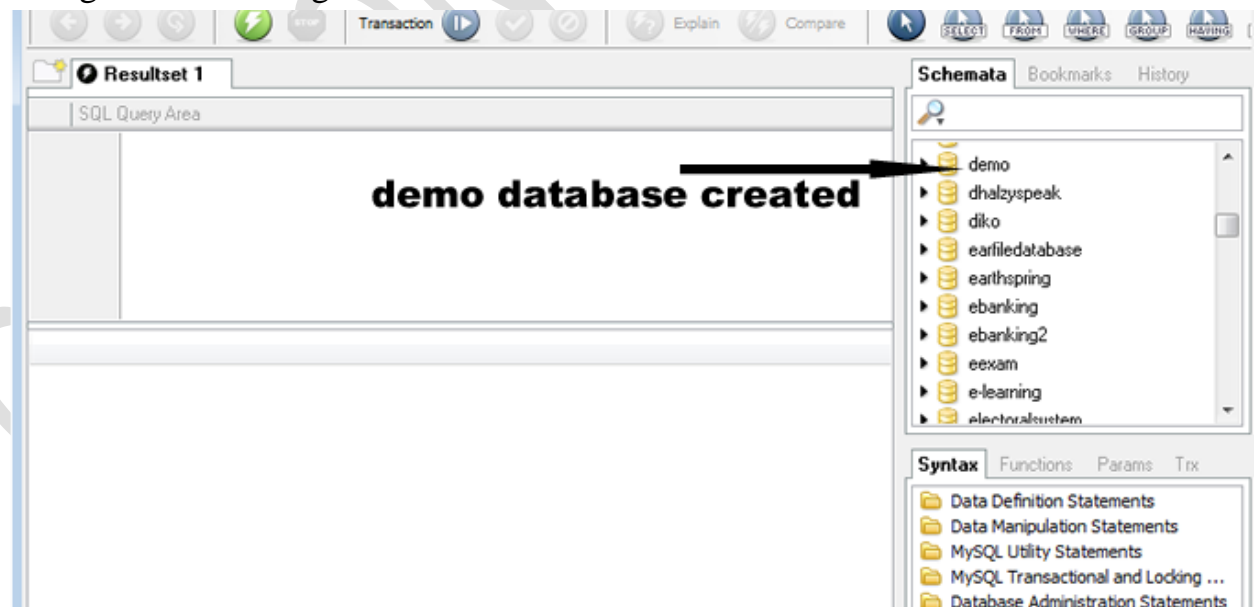


Figure 5: demo database created

After creating a database, tables can now be created inside the database.

Tables and Data

A table is made up of rows (records) and columns (fields). Data is referred to as raw facts or unprocessed facts.

To create a table in the demo, we have to right click on the demo and select *Create New Table*

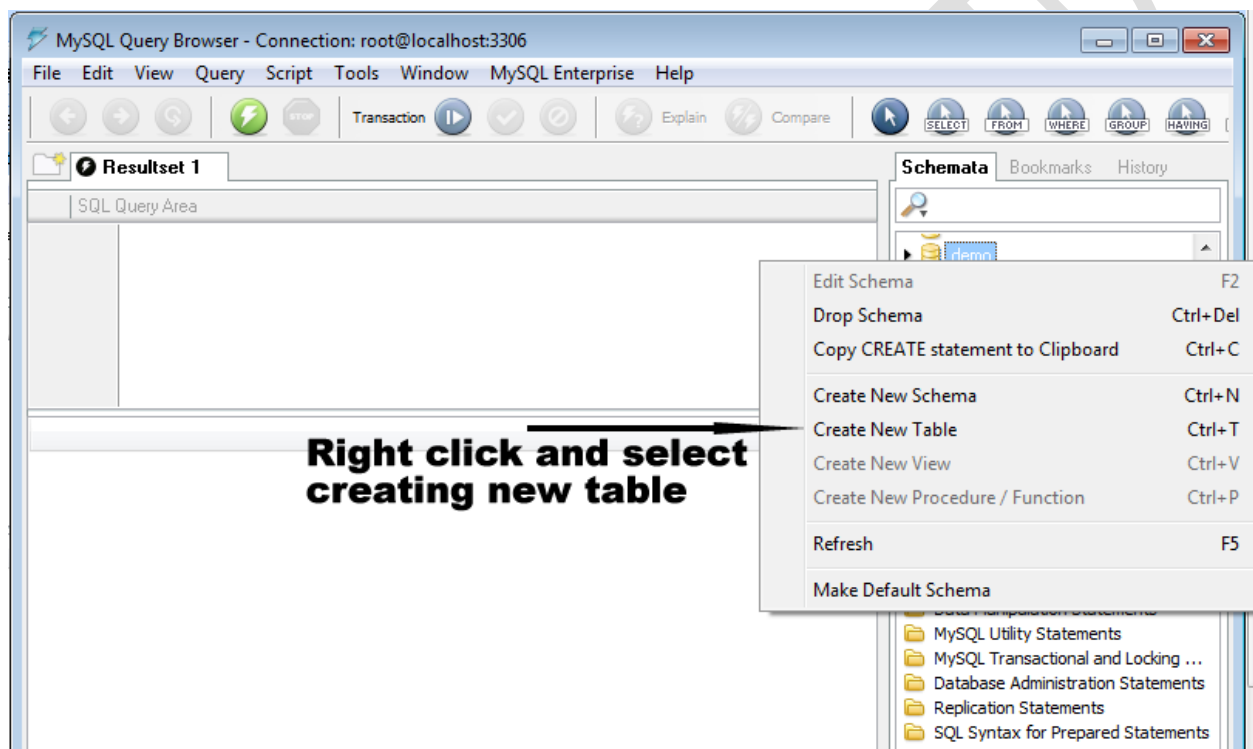


Figure 6: Right Click and select Create new Table

Let us assume that we are creating a table to manage sales of our products in the company. We shall create a table called *customers* as shown in figure 7.

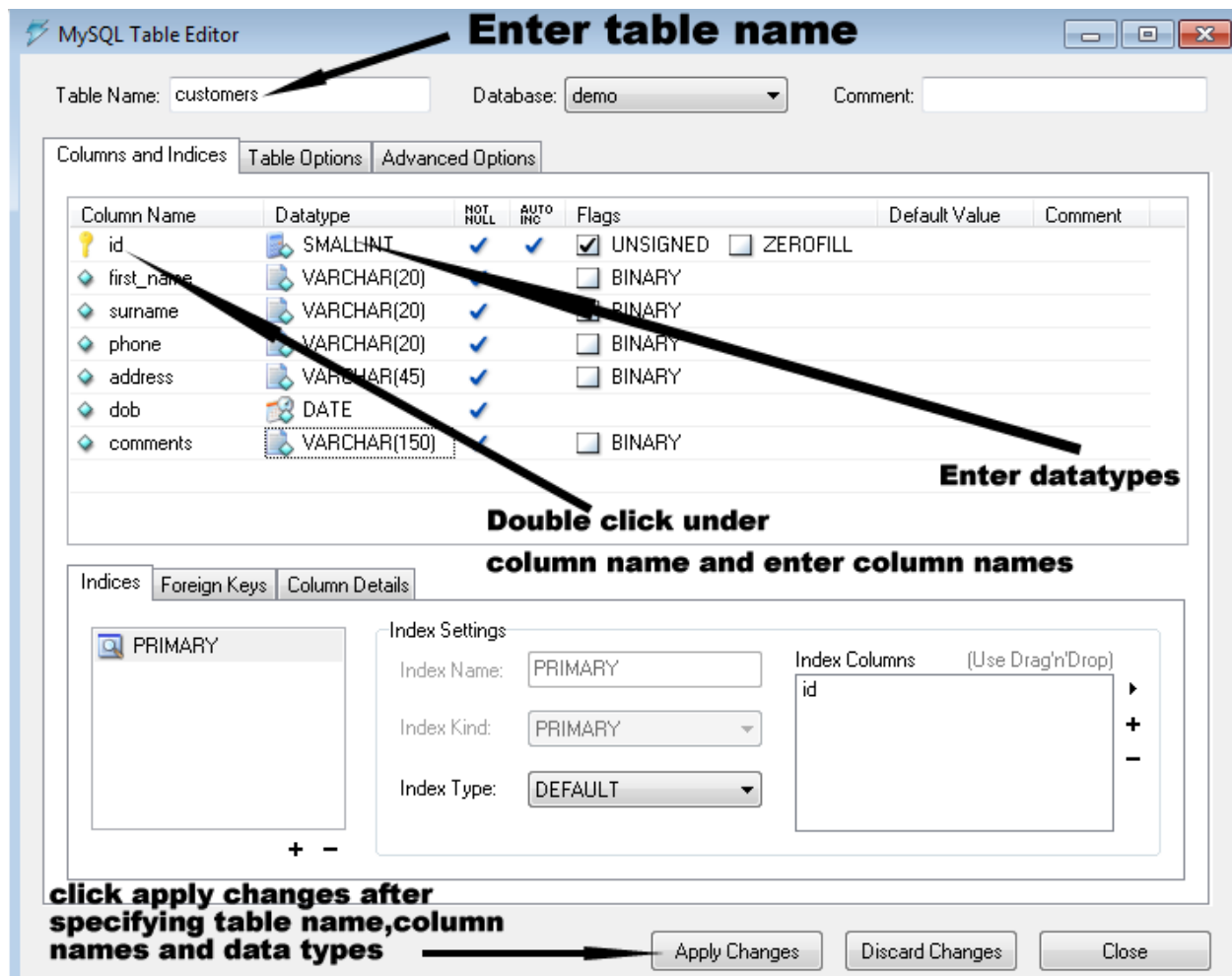


Figure 7: Creating customers table in demo database

Note the following about the customers table:

- A table being created must contain at least one column.
- The id column is the primary key and auto increment. Auto increment means that the values for the id column will be provided by the database automatically during the insertion of data in the database.
- Not null – Any column that is indicated as **not null** must be provided with a value during insertion of data in the database table in which it is found.
- Data type – this specifies the nature of data that a particular column in a database table can support.
- Unsigned – Any column marked as unsigned can only support positive numbers of the integer data type.

After applying changes, you will then click on **Execute** to finally create the table as seen in figure 8.

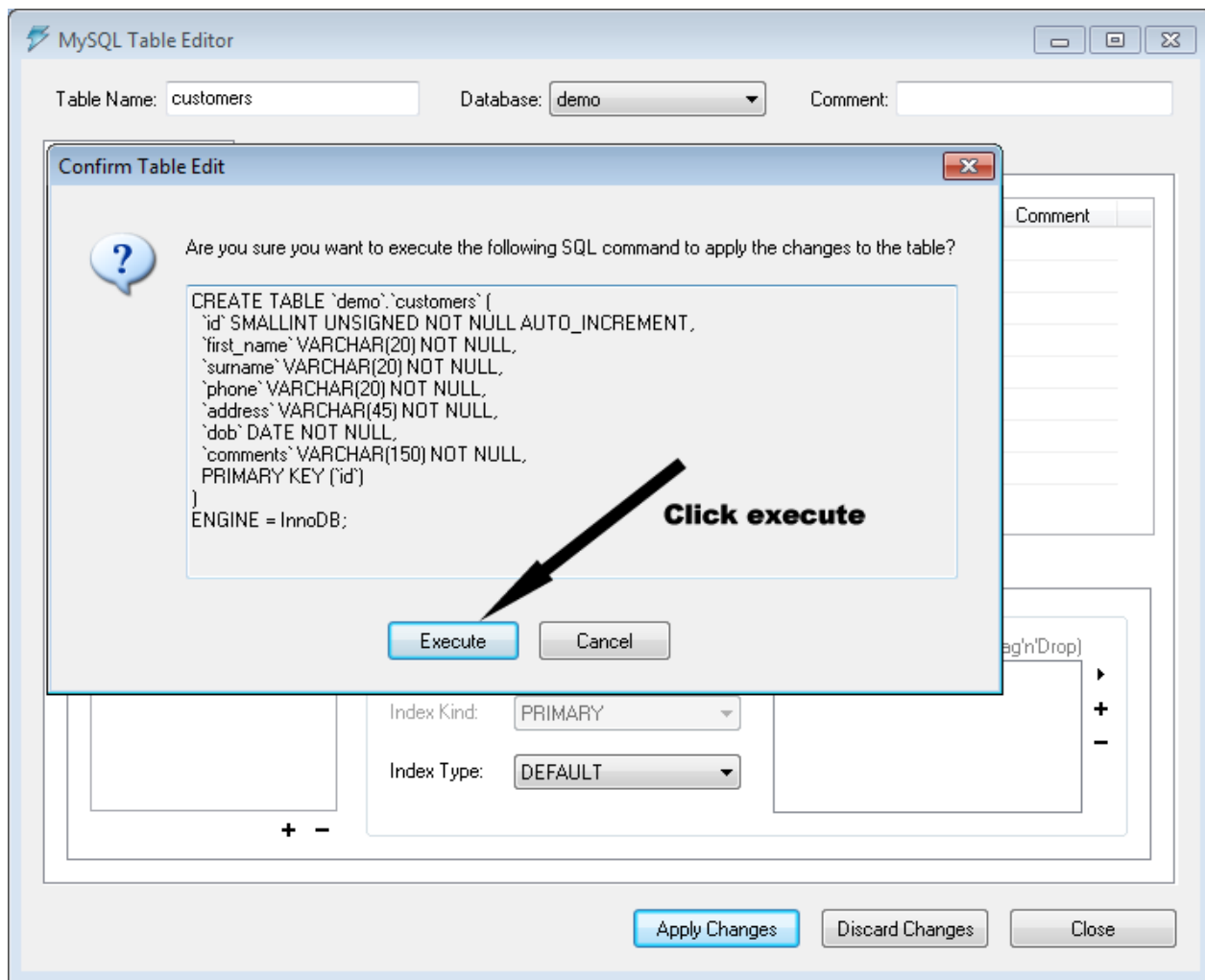


Figure 8: Clicking Execute finally creates the table

Following the same approach for creating, order_items, orders, and product tables have also been created in the demo database.

You can now double click on the demo database to see the created table. To see the table display with the columns, double click on the table as seen in figure 9.

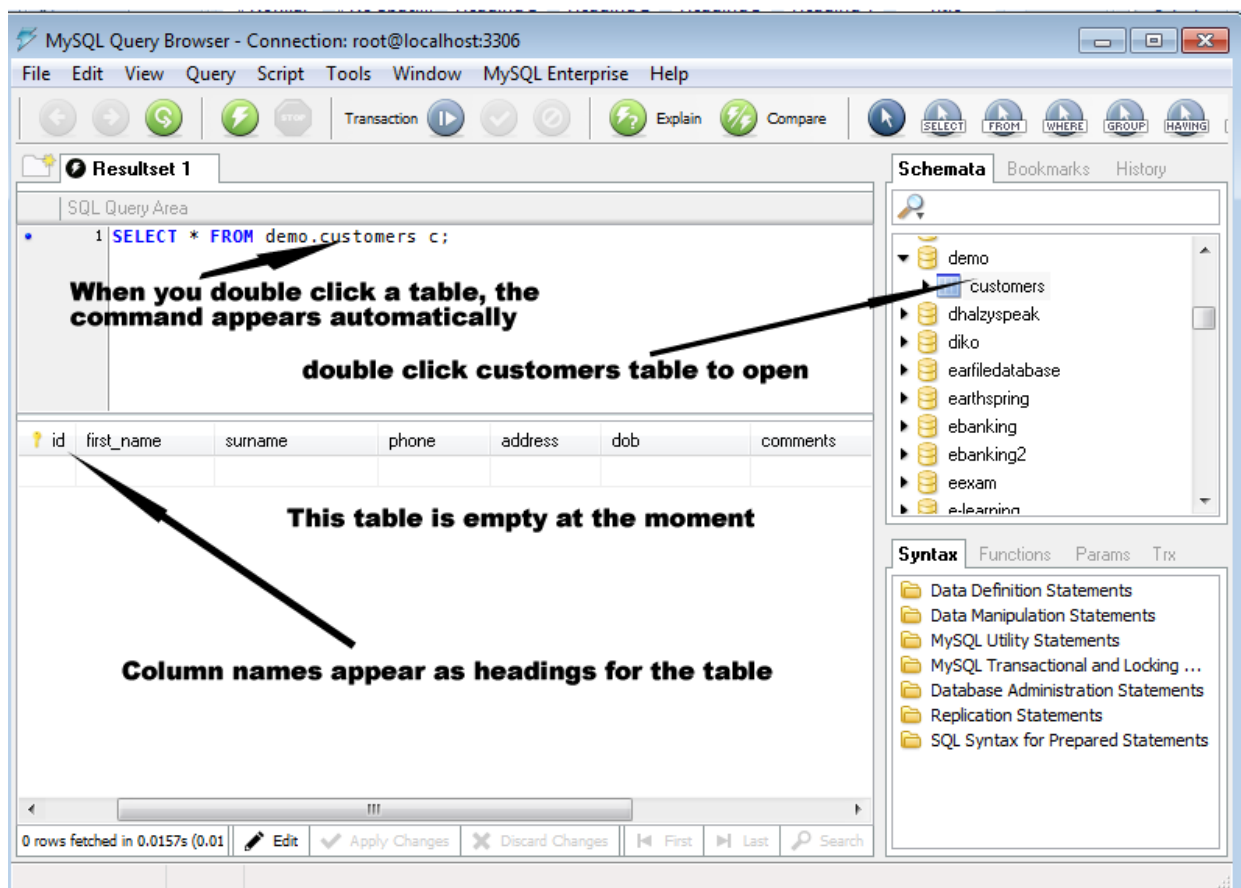


Figure 9: Customers table created.

So many wonderful commands can be carried out on the database we just created.

Administrative MySQL Command:

Here is the list of some important MySQL commands, which can be used with the customers table:

- INSERT INTO** - inserts new data into a database

We shall consider three ways to insert data into the customers table using the query browser.

- Direct insert using edit** as seen in figure 10

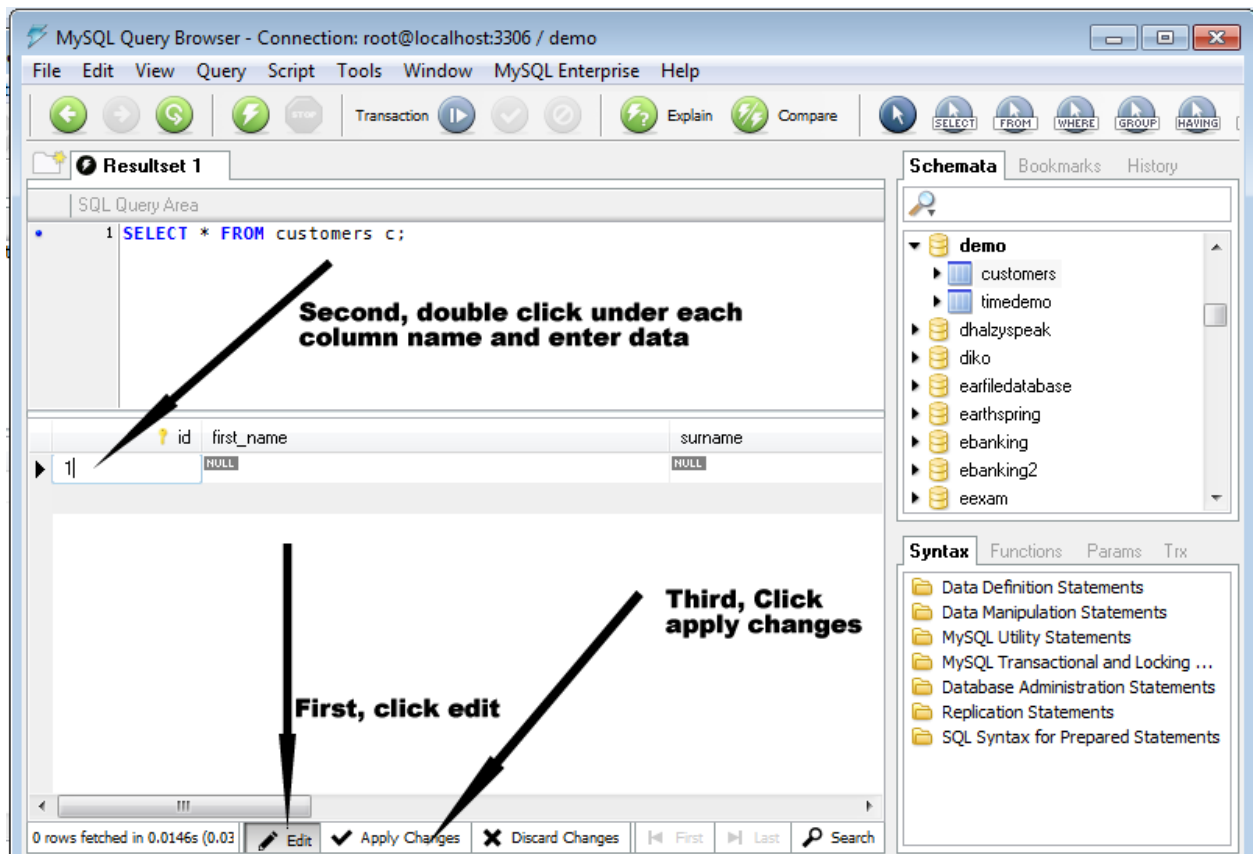


Figure 10 Insert using edit method in the query browser.

Using commands

By using commands in the query browser, we can also insert data into the database as seen in figure 11.

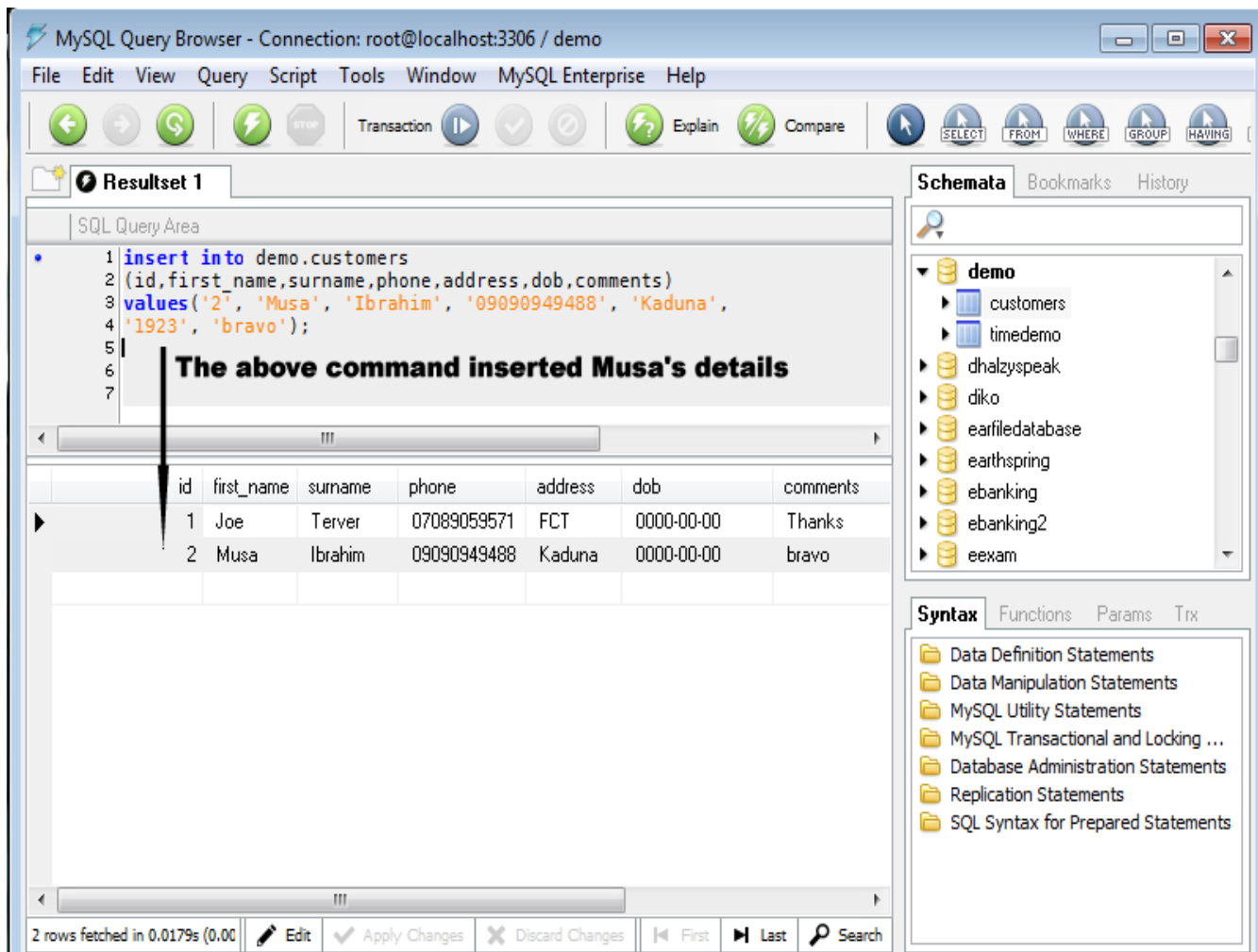


Figure 11 Insert using command in the query browser.

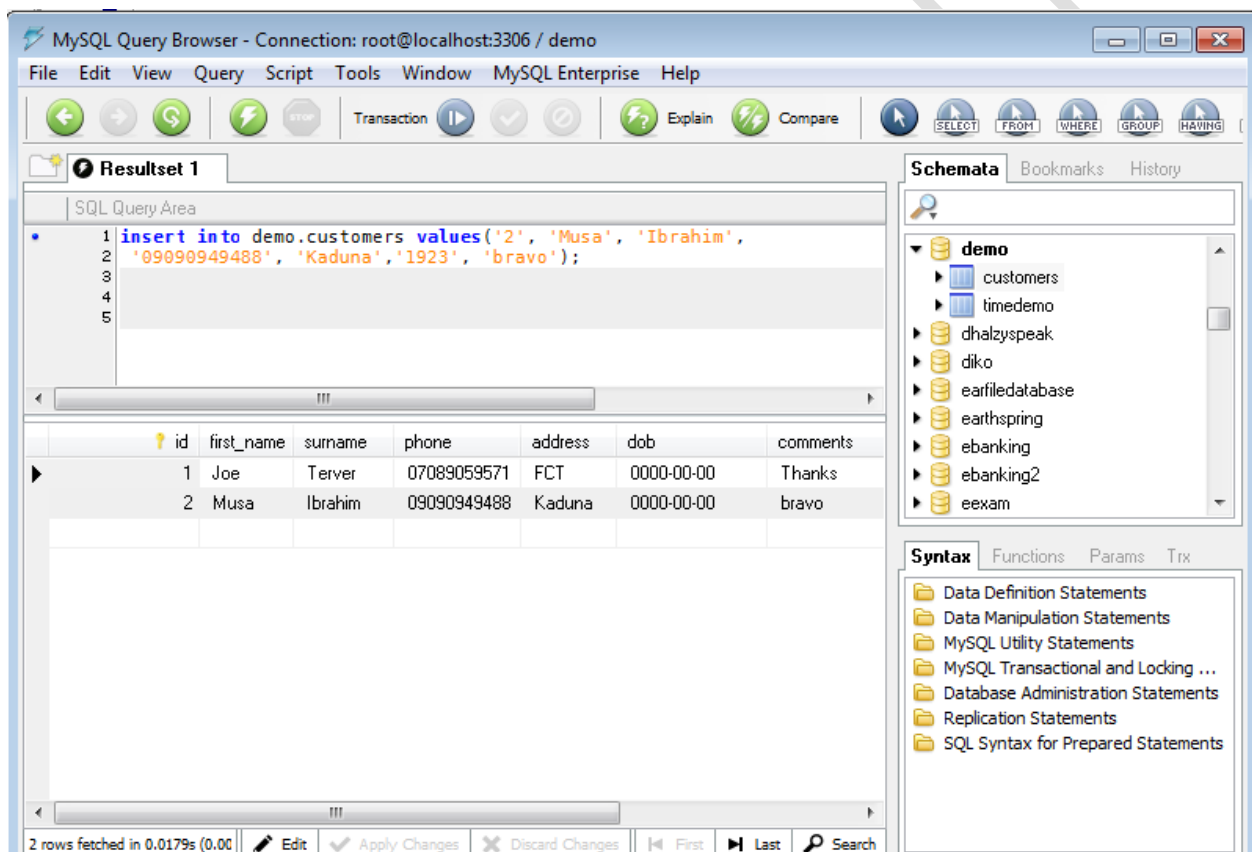
Note the following about the above insert command:

- Insert into – must begin an insert command in MySQL
- demo.customers – is the database name, demo followed by a table name, customers both separated by a period (.). This means that the insert targeted at the customers table in the demo database.
- (id,first_name,surname,phone,address,dob,comments) – is the list of columns in the customers table that are expected to receive data. Any column not included in this list will not received data.
- Values – this precedes the list containing the actual values.

- ('2', 'Musa', 'Ibrahim', '09090949488', 'Kaduna','1923', 'bravo') – is the list of values. Each value is written in single quotes and separated by a comma from the next.
- The semi-colon (;) at the end of the command signifies the end of the that particular command. More than one command can be written and separated by commas.
- The first value, 2 will be inserted into the first column, id, Musa will be inserted into the column first_name and so it applies for all other columns and values respectively.

An insert can be done without specifying the column names. For example, the command can be modified as seen in figure 12.

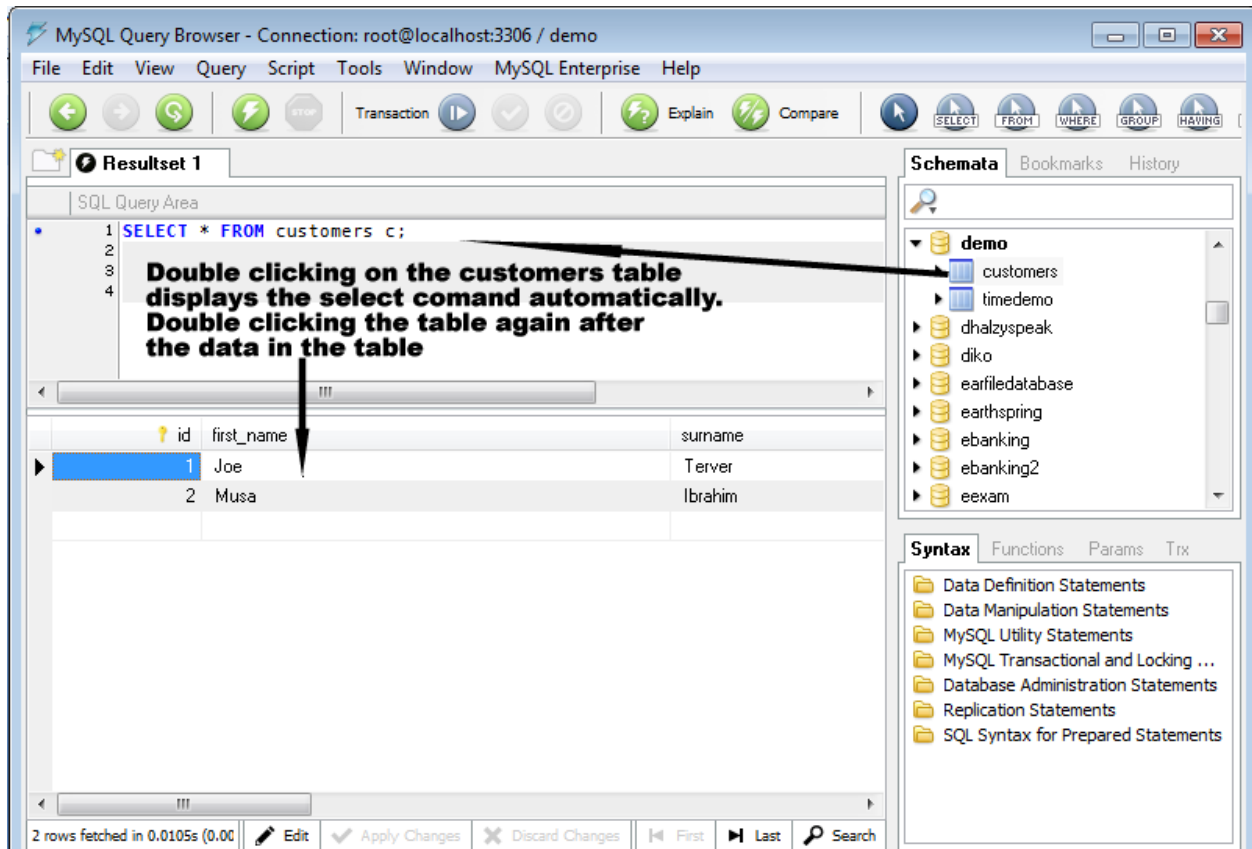
Figure 12. Alternative insert using command



In this approach, observe that the list of columns is removed leaving the list of values to be inserted. In this case, the number of values specified must equal the number of columns in the database and during the insert, the first value, 2 will be inserted into the first column, id, Musa will be inserted into the column first_name and so it applies for all other columns and values respectively. Using this approach, no column can be left out as data must be proved for each column in customers table.

b. **SELECT** - extracts data from a database as seen in figure 13

Figure 13. Select command



Note the following about the above select command, written as *SELECT * FROM customers c;*

- SELECT – must begin a select command
- * means all columns. Therefore, SELECT * means SELECT ALL COLUMNS
- Customers – is the name of the table from which the data will be selected.
- The c after the name of the table is an aka (also known as) for the table, customers
- The select command has the following syntax:

SELECT column_name, column_name FROM table_name;

E.g. Select * FROM customers;

Conditions can also be added to the select command to target particular rows (records) to select by making use of the WHERE clause:

WHERE clause – The WHERE clause is used to filter records. It is used to extract only those records that fulfilled a specific criterion

WHERE clause syntax

SELECT column_name, column_name FROM table_name WHERE
column_name operator value;

To select row number 2, containing Musa's details, we modify the command as such;

```
SELECT * FROM customers c WHERE id = '2';
```

This command will select only the row containing id value of 2.

In the above select command, id is the column name, = is the operator and 2 is the value.

Operators in the WHERE clause

The following operators can be used in the WHERE clause:

Operator	Description
=	Equal
<>	Not equal. Note: In some versions of SQL this operator may be written as !=
>	Greater than
<	Less than
>=	Greater than or equal
<=	Less than or equal
BETWEEN	Between an inclusive range
LIKE	Search for a pattern
IN	To specify multiple possible values for a column

AND & OR operators

The AND & OR operators are used to filter records based on more than one condition. The AND operator displays a record if both the first condition and the

second condition are true while the OR operator displays a record if either the first condition OR the second condition is true.

Example with the AND operator

```
SELECT * FROM customers WHERE id ='2' AND first_name ='Musa';
```

Example with the OR operator

```
SELECT * FROM customers WHERE id ='2' OR first_name ='Musa';
```

COMBINING AND & OR

You can also combine AND and OR (use parentheses to for complex expression).

Example

The following command select all customers whose first_name is Musa and the dob must be 2017-01-15.

```
SELECT * FROM customers WHERE first_name ='Musa' AND (dob =' 2017-01-15'  
OR do =' 2017-01-14');
```

THE ORDER BY KEYWORD

The ORDER BY keyword is used to sort the result-set by one or more columns. It sorts in ascending order by default. To sort the records in a descending order, you can use the DESC keyword.

It has the following syntax

```
SELECT column_name, column_name FROM table_name ORDER BY column  
ASC/DESC, column_name ASC/DESC;
```

Example

```
SELECT * FROM customers ORDER BY dob ASC, first_name DESC;
```

EXTENDED COMPARISON OPERATORS

In addition to the basic comparison operator(e.g. =, <, >, >=, <=) described above, SQL supports extended comparison operators such as BETWEEN, IN, LIKE, and IS NULL.

The BETWEEN operator implements a range comparison, that is, it tests whether a value is between other values. Between comparisons have the following format:

Value-1[NOT] BETWEEN value-2 AND value-3

A comparison which tests if value-1 is greater than or equal to value-2 and less than or equal to value-3 can be written as:

Value-1 >= value-2 AND value-1 <= value-3

Or, if not included,

NOT(Value-1 >= value-2 AND value-1 <= value-3)

Assuming we create a table call products with the following values:

Sn	Product_name	quantity
1	System Unit	200
2	Monitor	1000
3	Laptop	300
4	Mouse	100

Products table

Then, we can check products whose quantity is between 200 and 500 using the following command:

```
SELECT * FROM products WHERE quantity BETWEEN 200 AND 500;
```

THE IN OPERATOR

The IN operator implements comparison to a list of values, that is, it tests whether a value matches any value in the list of values. It has the following format:

Value-1[NOT] IN (value2, [,value-3]...)

Or if NOT is included:

NOT (Value-1=value-2[OR value-1=value-3]...)

Example

```
SELECT first_name from customers WHERE dob IN('2017-01-15', '2017-01-14');
```

THE LIKE OPERATOR

The LIKE operator implements a pattern match comparison, that is, it matches a string value against a pattern string containing wild-card characters.

The wild-card character for LIKE are percent(%) and underscore(_). Underscore matches any single character while percent matches zero or more character.

Example

VALUE	EXPRESSION	RESULT
'abc'	' a '	True
'ab'	' b '	False
'abc'	'%b%'	True
'ab'	'%b%'	True
'abc'	' a'	False
'ab'	'a '	True
'abc'	'a% '	True
'ab'	'a% '	True

The LIKE comparison operator has the following format:

Value-1[NOT] LIKE value-2[ESCAPTE value-3]

All values must be string (characters). The optional ESCAPE sub-clause specifies and escape character for the pattern to use '%' and '_' (and the escape pattern for matching). The value must be a single character string in the pattern, the ESCAPE character precedes any character to be escaped.

For example, to match a string ending with '%' use:

X LIKE '%/%'ESCAPE '/'

A more contrived example that escapes the escape character:

Y LIKE '/%//'ESCAPE '/'

... matches any string with '%/'

The optional NOT reverses the result so that:

Z NOT LIKE 'abc%' is equivalent to:

NOT z LIKE 'abc%'

INSERT STATEMENT

The INSERT statement adds one or more rows to a table. It has two formats:

INSERT INTO table_name [(columns-list)] VALUES(value-list)

and,

INSERT INTO table_name [(column-list)] (query-specification)

The first form inserts a single row into the table and explicitly specifies the column values for the row.

The second form uses the result of query-specification to insert one or more rows into the table. The table rows from the query are the rows added to the insert table.

Note: The query cannot reference the same table been inserted into.

Both forms have an optional columns-list specification. Only the columns listed will be assigned values. Unlisted columns are set to null, so unlisted columns must allow nulls. The values from the VALUES clause (first form) or the column form the query-specification rows (second form) are assigned to the corresponding column in the column-list in order. If the optional column-list is missing, the default column-list is substituted. The default column-list contains all columns in the table in the order they were declared in CREATE or CREATE VIEW.

VALUES

The VALUES clause in the INSERT statement provides a set of values to place in the columns of a new row. It has the following format:

VALUES(value-1[, value-2]...)

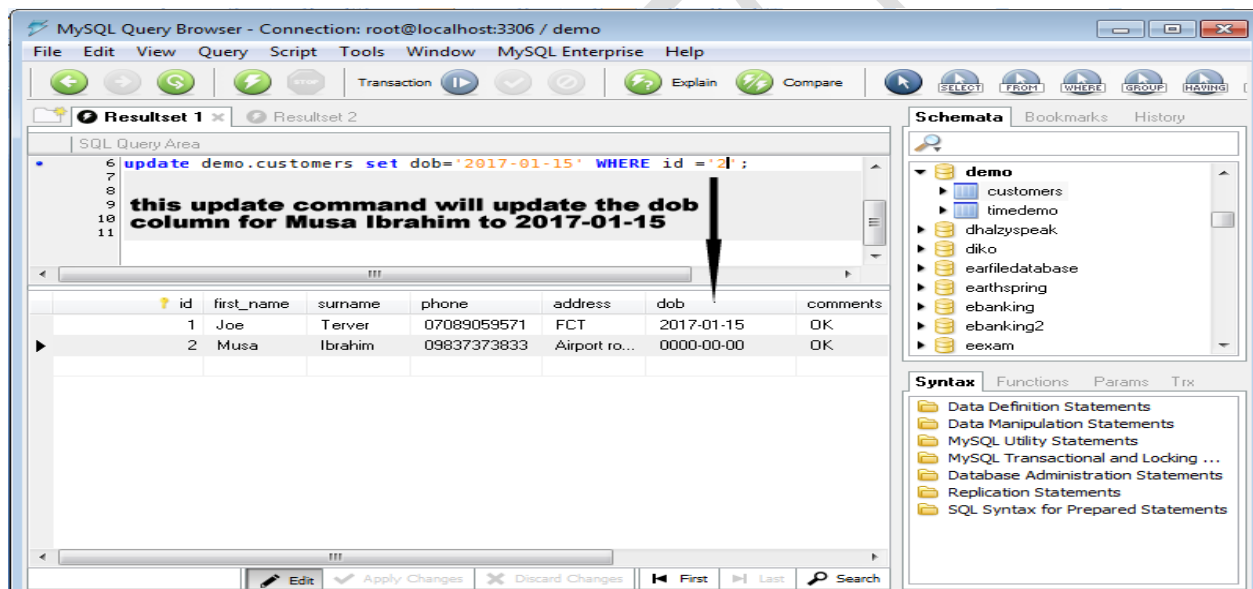
Value-1 and value-2 are literal values or scalar expressions involving literals. They can also specify NULL.

- c. **UPDATE** – Update command can be used to modify data in the database as seen in figure 14 below.

In the customers table, if Joe decides to change his *dob* from 0000-00-00 to 2017-01-15, then an update will be needed to achieve that task using the following command:

UPDATE demo.customers SET dob = '2017-01-15' WHERE id = '1';

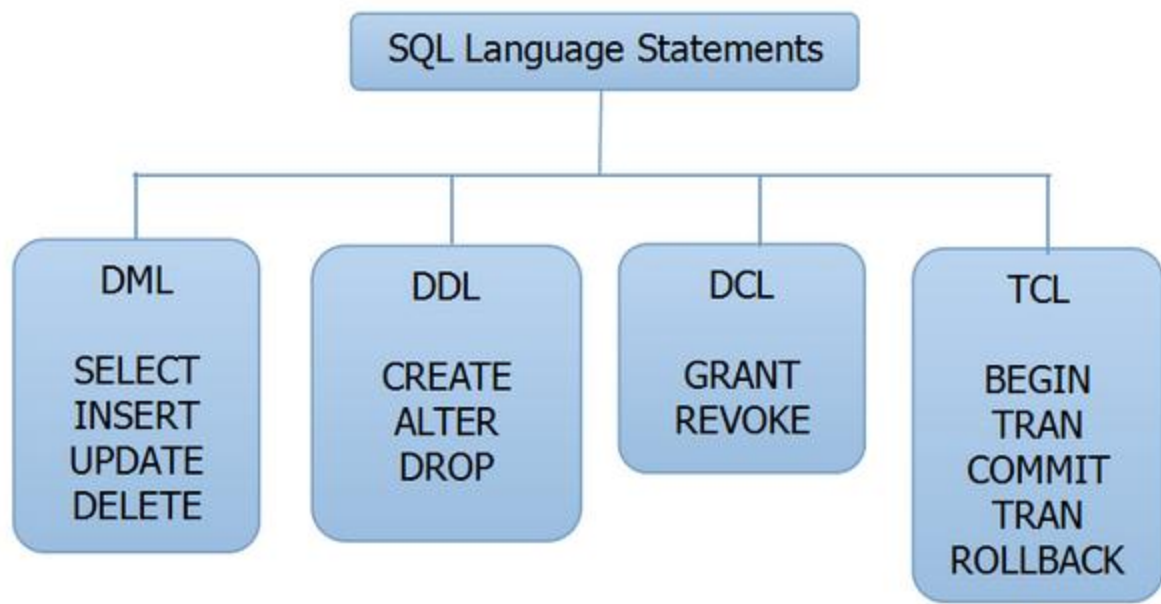
Figure 14 UPDATE command



- d. **DELETE** - deletes data from a database
- e. **USE Databasename**: This will be used to select a particular database in MySQL workarea.
- f. **SHOW DATABASES**: Lists the databases that are accessible by the MySQL DBMS.
- g. **SHOW TABLES**: Shows the tables in the database once a database has been selected with the use command.
- h. **SHOW COLUMNS FROM tablename**: Shows the attributes, types of attributes, key information, whether NULL is permitted, defaults, and other information for a table.
- i. **SHOW INDEX FROM tablename**: Presents the details of all indexes on the table, including the PRIMARY KEY.
- j. **CREATE DATABASE** - creates a new database
- k. **ALTER DATABASE** - modifies a database.
- l. **CREATE TABLE** - creates a new table
- m. **ALTER TABLE** - modifies a table
- n. **DROP TABLE** - deletes a table
- o. **CREATE INDEX** - creates an index (search key)
- p. **DROP INDEX** - deletes an index

The four main categories of SQL statements are as follows:

1. **DML (Data Manipulation Language)**
2. **DDL (Data Definition Language)**
3. **DCL (Data Control Language)**
4. **TCL (Transaction Control Language)**



DML (Data Manipulation Language)

DML statements affect records in a table. These are basic operations we perform on data such as selecting a few records from a table, inserting new records, deleting unnecessary records, and updating/modifying existing records.

DML statements include the following:

SELECT – select records from a table

INSERT – insert new records

UPDATE – update/Modify existing records

DELETE – delete existing records

DDL (Data Definition Language)

DDL statements are used to alter/modify a database or table structure and schema. These statements handle the design and storage of database objects.

CREATE – create a new Table, database, schema

ALTER – alter existing table, column description

DROP – delete existing objects from database

DCL (Data Control Language)

DCL statements control the level of access that users have on database objects.

GRANT – allows users to read/write on certain database objects

REVOKE – keeps users from read/write permission on database objects

Database Column Types

MySQL Data Types

In MySQL there are three main data types :

(1) text (2) number and (3) Date/Time types.

Text types:

Data type	Description
CHAR(size)	Holds a fixed length string (can contain letters, numbers, and special characters). The fixed size is specified in parenthesis. Can store up to 255 characters
VARCHAR(size)	Holds a variable length string (can contain letters, numbers, and special characters). The maximum size is specified in parenthesis. Can store up to 255 characters. Note: If you put a greater value than 255 it will be converted to a TEXT type
TINYTEXT	Holds a string with a maximum length of 255 characters
TEXT	Holds a string with a maximum length of 65,535 characters
BLOB	For BLOBs (Binary Large Objects). Holds up to 65,535 bytes of data
MEDIUMTEXT	Holds a string with a maximum length of 16,777,215 characters
MEDIUMBLOB	For BLOBs (Binary Large Objects). Holds up to 16,777,215 bytes of data
LONGTEXT	Holds a string with a maximum length of 4,294,967,295 characters
LOB	For BLOBs (Binary Large Objects). Holds up to 4,294,967,295 bytes of data
ENUM(x,y,z,etc.)	Let you enter a list of possible values. You can list up to 65535 values in an ENUM list. If a value is inserted that is not in the list, a blank value will be inserted. Note: The values are sorted in the order you enter them. You enter the possible values in this format: ENUM('X','Y','Z')

SET	Similar to ENUM except that SET may contain up to 64 list items and can store more than one choice
-----	--

MySQL Data Types

In MySQL there are three main types : (1) Text (2) Number, and (3) Date/Time types.

1. Text types:

Data type	Description
CHAR(size)	Holds a fixed length string (can contain letters, numbers, and special characters). The fixed size is specified in parenthesis. Can store up to 255 characters
VARCHAR(size)	Holds a variable length string (can contain letters, numbers, and special characters). The maximum size is specified in parenthesis. Can store up to 255 characters. Note: If you put a greater value than 255 it will be converted to a TEXT type
TINYTEXT	Holds a string with a maximum length of 255 characters
TEXT	Holds a string with a maximum length of 65,535 characters
BLOB	For BLOBs (Binary Large OBjects). Holds up to 65,535 bytes of data
MEDIUMTEXT	Holds a string with a maximum length of 16,777,215 characters
MEDIUMBLOB	For BLOBs (Binary Large OBjects). Holds up to 16,777,215 bytes of data

LONGTEXT	Holds a string with a maximum length of 4,294,967,295 characters
LONGBLOB	For BLOBs (Binary Large OBjects). Holds up to 4,294,967,295 bytes of data
ENUM(x,y,z,etc.)	Let you enter a list of possible values. You can list up to 65535 values in an ENUM list. If a value is inserted that is not in the list, a blank value will be inserted. Note: The values are sorted in the order you enter them. You enter the possible values in this format: ENUM('X','Y','Z')
SET	Similar to ENUM except that SET may contain up to 64 list items and can store more than one choice

Number types:

Data type	Description
TINYINT(size)	-128 to 127 normal. 0 to 255 UNSIGNED*. The maximum number of digits may be specified in parenthesis
SMALLINT(size)	-32768 to 32767 normal. 0 to 65535 UNSIGNED*. The maximum number of digits may be specified in parenthesis
MEDIUMINT(size)	-8388608 to 8388607 normal. 0 to 16777215 UNSIGNED*. The maximum number of digits may be specified in parenthesis
INT(size)	-2147483648 to 2147483647 normal. 0 to 4294967295 UNSIGNED*. The maximum number of digits may be specified in parenthesis
BIGINT(size)	-9223372036854775808 to 9223372036854775807 normal. 0 to 18446744073709551615

UNSIGNED*. The maximum number of digits may be specified in parenthesis

FLOAT(size,d) A small number with a floating decimal point. The maximum number of digits may be specified in the size parameter. The maximum number of digits to the right of the decimal point is specified in the d parameter

DOUBLE(size,d) A large number with a floating decimal point. The maximum number of digits may be specified in the size parameter. The maximum number of digits to the right of the decimal point is specified in the d parameter

DECIMAL(size,d) A DOUBLE stored as a string , allowing for a fixed decimal point. The maximum number of digits may be specified in the size parameter. The maximum number of digits to the right of the decimal point is specified in the d parameter.

*The integer types have an extra option called UNSIGNED. Normally, the integer goes from a negative to positive value. Adding the UNSIGNED attribute will move that range up so it starts at zero instead of a negative number.

3. Date types:

Data type	Description
DATE()	A date. Format: YYYY-MM-DD Note: The supported range is from '1000-01-01' to '9999-12-31'
DATETIME()	*A date and time combination. Format: YYYY-MM-DD HH:MI:SS Note: The supported range is from '1000-01-01 00:00:00' to '9999-12-31 23:59:59'
TIMESTAMP()	*A timestamp. TIMESTAMP values are stored as the number of seconds since the Unix epoch ('1970-01-01 00:00:00' UTC). Format: YYYY-MM-DD HH:MI:SS Note: The supported range is from '1970-01-01

00:00:01' UTC to '2038-01-09 03:14:07' UTC

TIME()

A time. Format: HH:MI:SS

Note: The supported range is from '-838:59:59' to '838:59:59'

YEAR()

A year in two-digit or four-digit format.

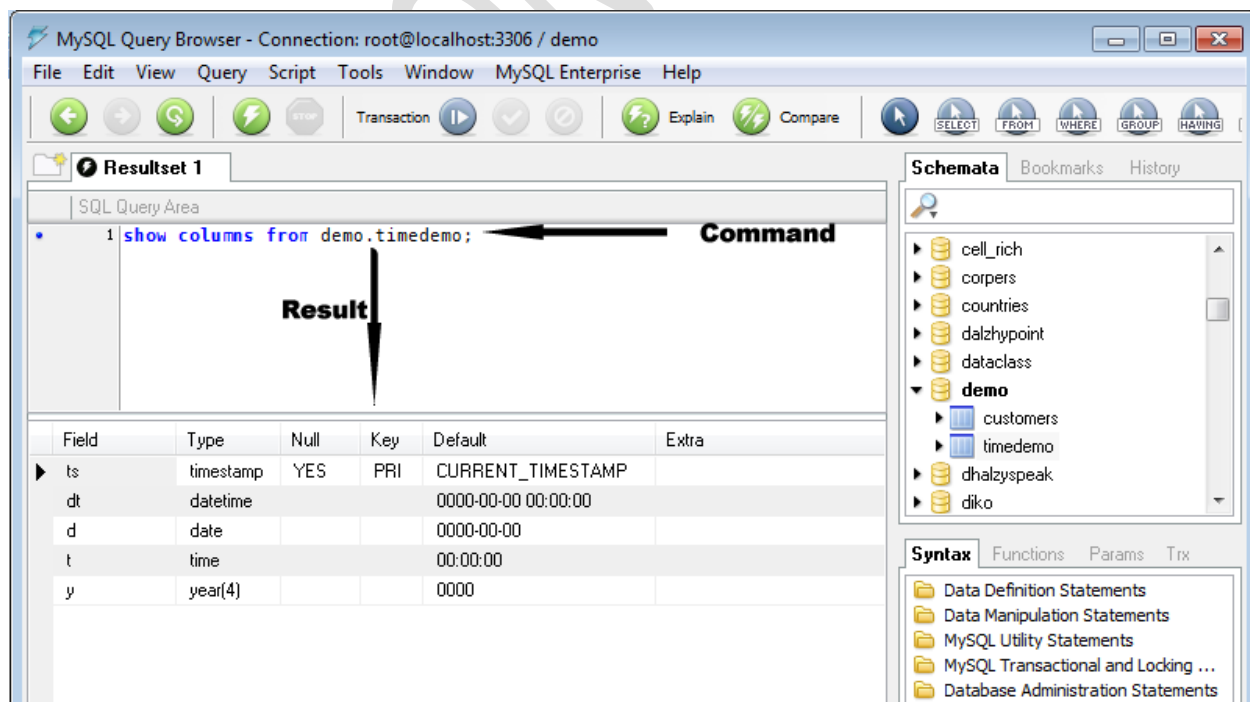
Note: Values allowed in four-digit format: 1901 to 2155. Values allowed in two-digit format: 70 to 69, representing years from 1970 to 2069

To properly demonstrate the date and time column types, we have created the table called `timedemo` using the command below:

Create table `timedemo(ts timestamp, dt datetime, d date, t time, y year)`; This command will create a table called `timedemo` with `ts`, `dt`, `d`, `t`, and `y` as column names and the respective data types.

To see the columns in a particular `timedemo` table, we write:

`show columns from demo.timedemo;`



Insert into `timedemo` values(`now()`,`now()`,`now()`,`now()`,`now()`);

The above command will insert data into the timedemo table based on the column types of the various columns. To see the data inserted, we issue the command

```
Select * from demo.timedemo;
```

TCL (Transaction Control Language)

TCL statements allow you to control and manage transactions to maintain the integrity of data within SQL statements.

BEGIN Transaction – opens a transaction

COMMIT Transaction – commits a transaction

ROLLBACK Transaction – ROLLBACK a transaction in case of any error

What is Normalization?

Simply put, normalization is a formal process for determining which fields belong in which tables in a relational database. Normalization follows a set of rules worked out at the time relational databases were born. A normalized relational database provides several benefits:

1. Elimination of redundant data storage.
2. Close modeling of real world entities, processes, and their relationships.
3. Structuring of data so that the model is flexible.

Normalization ensures that you get the benefits relational databases offer. Time spent learning about normalization will begin paying for itself immediately.

Why Do They Talk Like That?

Some people are intimidated by the *language* of normalization. Here is a quote from a classic text on relational database design:

Normalization Is a Nice Theory

A relation is in third normal form (3NF) if and only if it is in 2NF and every nonkey attribute is nontransitively dependent on the primary key.

C.J. Date

An Introduction to Database Systems

Huh? Relational database theory, and the principles of normalization, were first constructed by people intimately acquainted with set theory and predicate calculus. They wrote about databases for likeminded people. Because of this, people

sometimes think that normalization is “hard”. Nothing could be more untrue. The principles of normalization are simple, common-sense ideas that are easy to apply. Here is another author’s description of the same principle:

A table should have a field that uniquely identifies each of its records, and each field in the table should describe the subject that the table represents.

Michael J. Hernandez

Database Design for Mere Mortals

That sounds pretty sensible. A table should have something that uniquely identifies each record, and each field in the record should be about the same thing. We can summarize the objectives of normalization even more simply:

Eliminate redundant fields.

Avoid merging tables.

You’ve probably intuitively followed many normalization principles all along. The purpose of formal normalization is to ensure that your common sense and intuition are applied consistently to the entire database design.

Design Versus Implementation

Designing a database structure and *implementing* a database structure are different tasks. When you design a structure it should be described without reference to the specific database tool you will use to implement the system, or what concessions you plan to make for performance reasons. These steps come later. After you’ve designed the database structure abstractly, then you implement it in a particular environment—4D in our case. Too often people new to database design combine design and implementation in one step. 4D makes this tempting because the structure editor is so easy to use. Implementing a structure without designing it quickly leads to flawed structures that are difficult and costly to modify. Design first, implement second, and you’ll finish faster and cheaper.

Normalized Design: Pros and Cons

We’ve implied that there are various advantages to producing a properly normalized design before you implement your system. Let’s look at a detailed list of the pros and cons:

We think that the pros outweigh the cons.

Terminology

There are a couple terms that are central to a discussion of normalization: “key” and “dependency”. These are probably familiar concepts to anyone who has built relational database systems, though they may not be using these words. We define and discuss them here as necessary background for the discussion of normal forms that follows.

Primary Key

The primary key is a fundamental concept in relational database design. It’s an easy concept: each record should have something that identifies it uniquely. The primary key can be a single field, or a combination of fields. A table’s primary key also serves as the basis of relationships with other tables. For example, it is typical to relate invoices to a unique customer ID, and employees to a unique department ID.

Note: 4D does not implicitly support multi-field primary keys, though multi-field keys are common in other client-server databases. The simplest way to implement a multi-field key in 4D is by maintaining an additional field that stores the concatenation of the components of your multi-field key into a single field. A concatenated key of this kind is easy to maintain using a 4D V6 trigger.

Pros of Normalizing

1. More efficient database structure.
2. Better understanding of your data.
3. More flexible database structure.
4. Easier to maintain database structure.
5. Few (if any) costly surprises down the road.
6. Validates your common sense and intuition.
7. Avoid redundant fields.
8. Insure that distinct tables exist when necessary.

Cons of Normalizing

You can’t start building the database before you know what the user needs.

A primary key should be unique, mandatory, and permanent. A classic mistake people make when learning to create relational databases is to use a volatile field as the primary key. For example, consider this table:

[Companies]

Company Name

Address

Company Name is an obvious candidate for the primary key. Yet, this is a bad idea, even if the Company Name is unique. What happens when the name changes

after a merger? Not only do you have to change this record, you have to update every single related record since the key has changed.

Another common mistake is to select a field that is *usually* unique and unchanging. Consider this small table:

[People]

Social Security Number

First Name

Last Name

Date of birth

In the United States all workers have a Social Security Number that uniquely identifies them for tax purposes. Or does it? As it turns out, not everyone has a Social Security Number, some people's Social Security Numbers change, and some people have more than one. This is an appealing but untrustworthy key.

The correct way to build a primary key is with a unique and unchanging value. 4D's **Sequence number** function, or a unique ID generating function of your own, is the easiest way to produce synthetic primary key values.

Functional Dependency

Closely tied to the notion of a key is a special normalization concept called *functional dependence* or *functional dependency*. The second and third normal forms verify that your functional dependencies are correct. So what is a "functional dependency"? It describes how one field (or combination of fields) determines another field. Consider an example:

[ZIP Codes]

ZIP Code

City

County

State Abbreviation

State Name

ZIP Code is a unique 5-digit key. What makes it a key? It is a key because it determines the other fields. For each ZIP Code there is a single city, county, and state abbreviation. These fields are functionally dependent on the ZIP Code field. In other words, they belong with this key. Look at the last two fields, State Abbreviation and State Name. State Abbreviation determines State Name, in other words, State Name is functionally dependent on State Abbreviation. State Abbreviation is acting like a key for the State Name field. Ah ha! State Abbreviation is a key, so it belongs in another table. As we'll see, the third normal form tells us to create a new States table and move State Name into it.

Normal Forms

The principles of normalization are described in a series of progressively stricter “normal forms”. First normal form (1NF) is the easiest to satisfy, second normal form (2NF), more difficult, and so on. There are 5 or 6 normal forms, depending on who you read. It is convenient to talk about the normal forms by their traditional names, since this terminology is ubiquitous in the relational database industry. It is, however, possible to approach normalization *without* using this language. For example, Michael Hernandez’s helpful *Database Design for Mere Mortals* uses plain language. Whatever terminology you use, the most important thing is that you go through the process.

Note: We advise learning the traditional terms to simplify communicating with other database designers. 4D and ACI do not use this terminology, but nearly all other database environments and programmers do.

First Normal Form (1NF)

The first normal form is easy to understand and apply:

A table is in first normal form if it contains no repeating groups.

What is a repeating group, and why are they bad? When you have more than one field storing the same kind of information in a single table, you have a repeating group. Repeating groups are the right way to build a spreadsheet, the only way to build a flat-file database, and the *wrong* way to build a relational database. Here is a common example of a repeating group:

[Customers]
Customer ID
Customer Name
Contact Name 1
Contact Name 2
Contact Name 3

What’s wrong with this approach? Well, what happens when you have a fourth contact? You have to add a new field, modify your forms, and rebuild your routines. What happens when you want to query or report based on all contacts across all customers? It takes a lot of custom code, and may prove too difficult in practice. The structure we’ve just shown makes perfect sense in a spreadsheet, but almost no sense in a relational database. All of the difficulties we’ve described are resolved by moving contacts into a related table.

[Customers]

Customer ID

Customer Name

[Contacts]

Customer ID (this field relates [Contacts] and [Customers])

Contact ID

Contact Name

Second Normal Form (2NF)

The second normal form helps identify when you've combined two tables into one. Second normal form depends on the concepts of the primary key, and functional dependency. The second normal form is:

A relation is in second normal form (2NF) if and only if it is in 1NF and every nonkey attribute is fully dependent on the primary key.

C.J. Date

An Introduction to Database Systems

In other words, your table is in 2NF if:

- 1) It doesn't have any repeating groups.
- 2) Each of the fields that isn't a part of the key is functionally dependent on the entire key. If a single-field key is used, a 1NF table is already in 2NF.

Third Normal Form (3NF)

Third normal form performs an additional level of verification that you have not combined tables. Here are two different definitions of the third normal form:

A table should have a field that uniquely identifies each of its records, and each field in the table should describe the subject that the table represents.

Michael J. Hernandez

Database Design for Mere Mortals

To test whether a 2NF table is also in 3NF, we ask, "Are any of the non-key columns dependent on any other non-key columns?"

Chris Gane

Computer Aided Software Engineering

When designing a database it is easy enough to accidentally combine information that belongs in different tables. In the ZIP Code example mentioned above, the ZIP Code table included the State Abbreviation and the State Name. The State Name is determined by the State Abbreviation, so the third normal form reminds you to move this field into a new table. Here's how these tables should be set up:

[ZIP Codes]

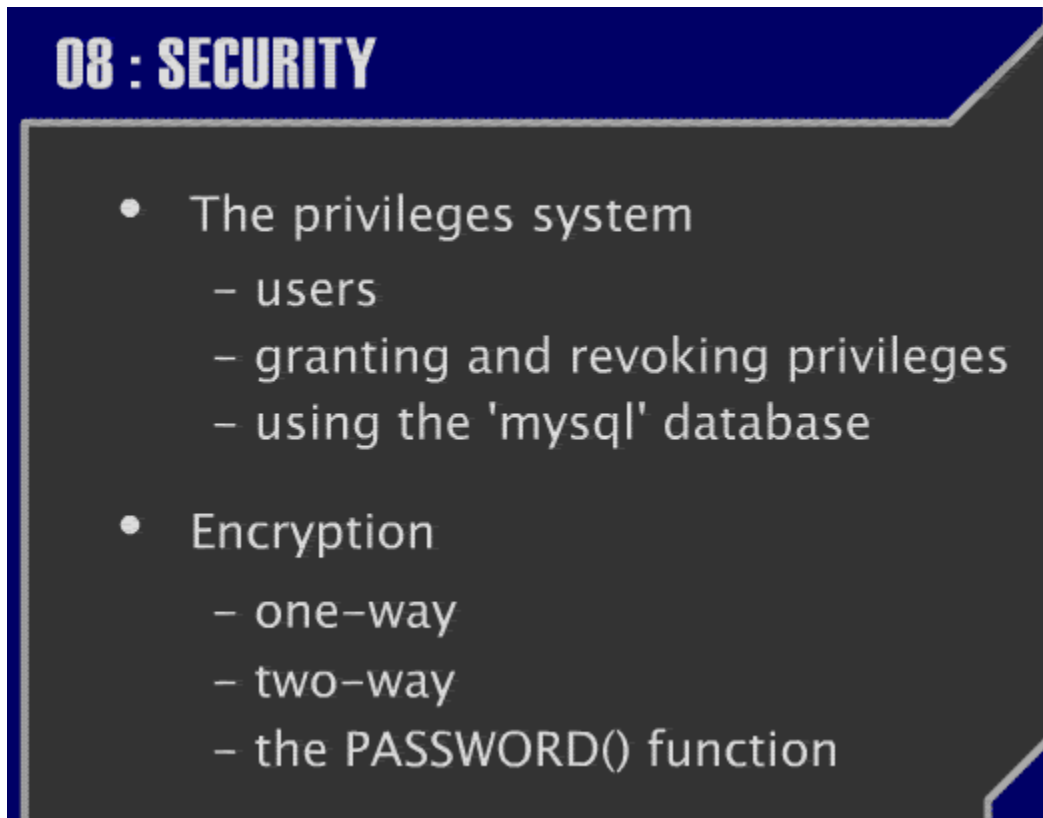
ZIP Code
City
County
State Abbreviation
[States]
State Abbreviation
State Name

Higher Normal Forms

There are several higher normal forms, including 4NF, 5NF, BCNF, and PJ/NF. We'll leave our discussion at 3NF, which is adequate for most practical needs. If you are interested in higher normal forms, consult a book like Date's

Database security

Here we shall look at the MySQL system of privileges that is used to grant user access to in the system.



User privileges

Users of the system can be granted access and assigned their levels of permission as regards what they can do on the system. Such permissions could include all of create, insert, update, alter delete or a combination of any the listed permissions. These user privileges can also be revoked.

The privileges system used in MySQL.

The list of privileges which can be granted to a user include the following:

- i. Select_priv,
- ii. Insert_priv
- iii. Update_priv
- iv. Delete_priv
- v. Create_priv

- vi. Drop_priv
- vii. Grant_priv
- viii. References_priv
- ix. Index_priv
- x. Alter_priv
- xi. Create_tmp_table_priv
- xii. Lock_tables_priv

To assign a privilege to a user, we make use of the following command
Grant privileges_here on database_name. to user's_name_here@localhost identified by 'user's_password';*

*E.g. To grant **select** privileges on a user called **reader** with the password **secret** on a database called **corpers**, we write:*

Grant select on corpers. to reader@localhost identified by 'secret';*

grant select on corpers.* to reader@localhost identified by 'secre';

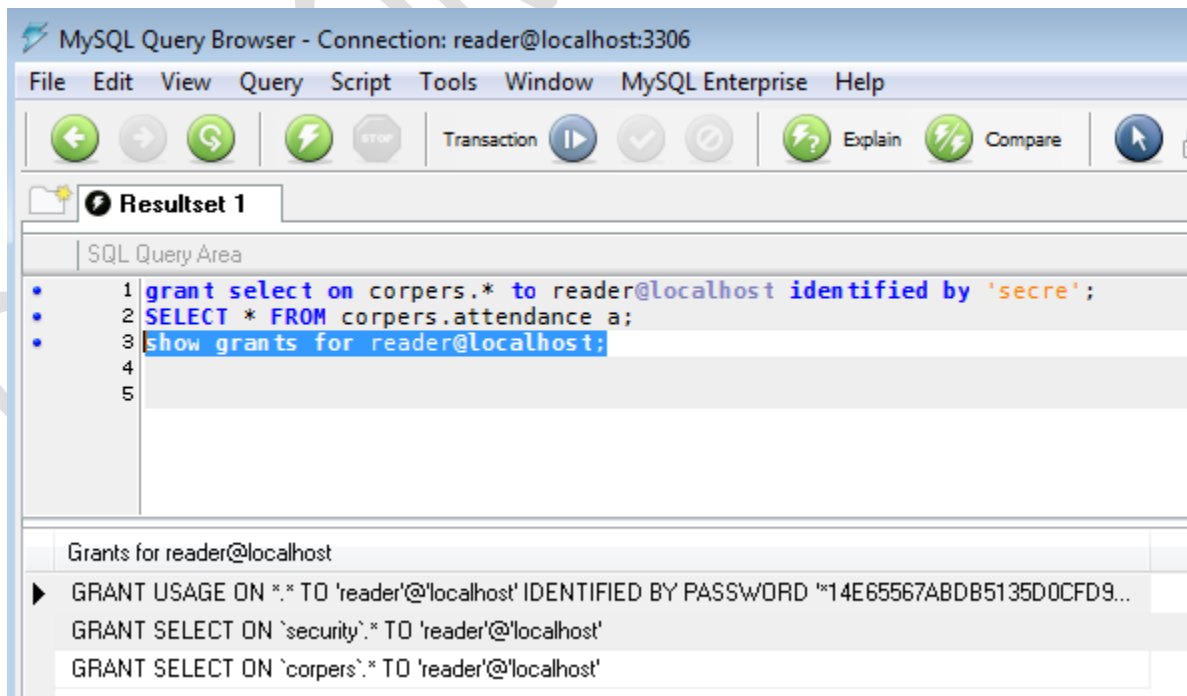
To select from corpers database, we write:

```
SELECT * FROM corpers.attendance a;
```

To see the available privileges on the user reader, we write:

```
show grants for reader@localhost;
```

The result is shown below:



Notice that the password secret for the user is encrypted when the grants are displayed. This is for security reasons.

Note also that the root user has every possible right on the MySQL.

To grant all privileges on all databases and tables to a user, we write:
grant all on *.* to admin@localhost identified by 'admin';

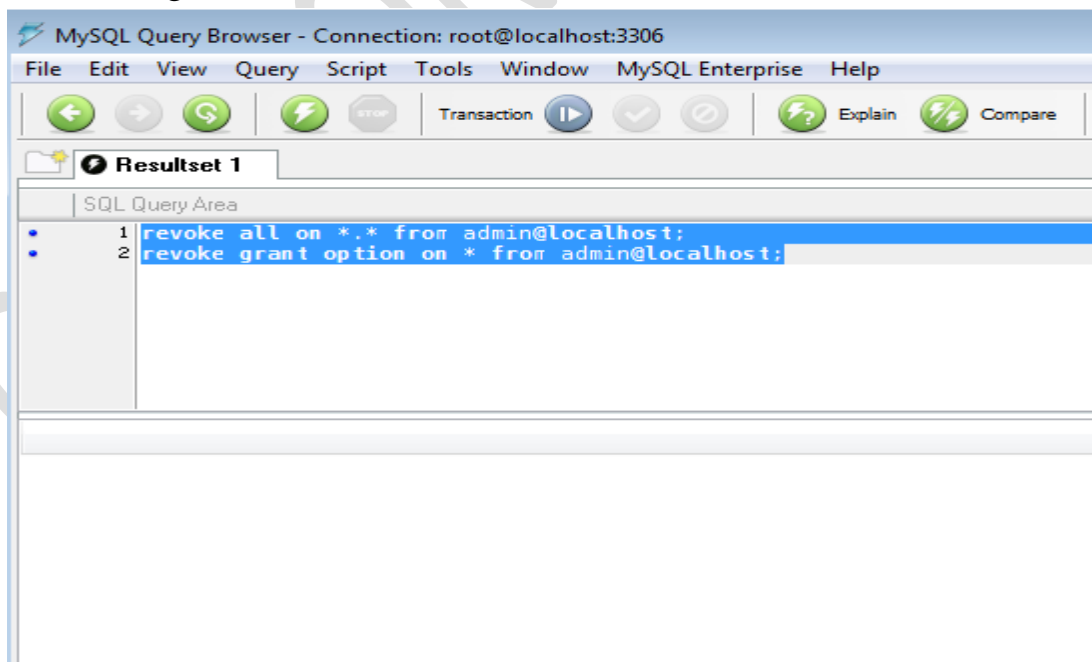
Note: That admin user that is created is not allowed to pass those privileges to someone else.

To make the admin user capable of granting privileges to other users, we add **with grant option** to the above command as follows:

To grant all privileges on all databases and tables to a user, we write:
grant all on *.* to admin@localhost identified by 'admin' with grant option;

If a user is no longer worthy of some privileges, they can also be revoked.
To revoke the **admin** users' privilege, we write:

revoke all on *.* from admin@localhost;
revoke grant option on * from admin@localhost;
See the diagram below:



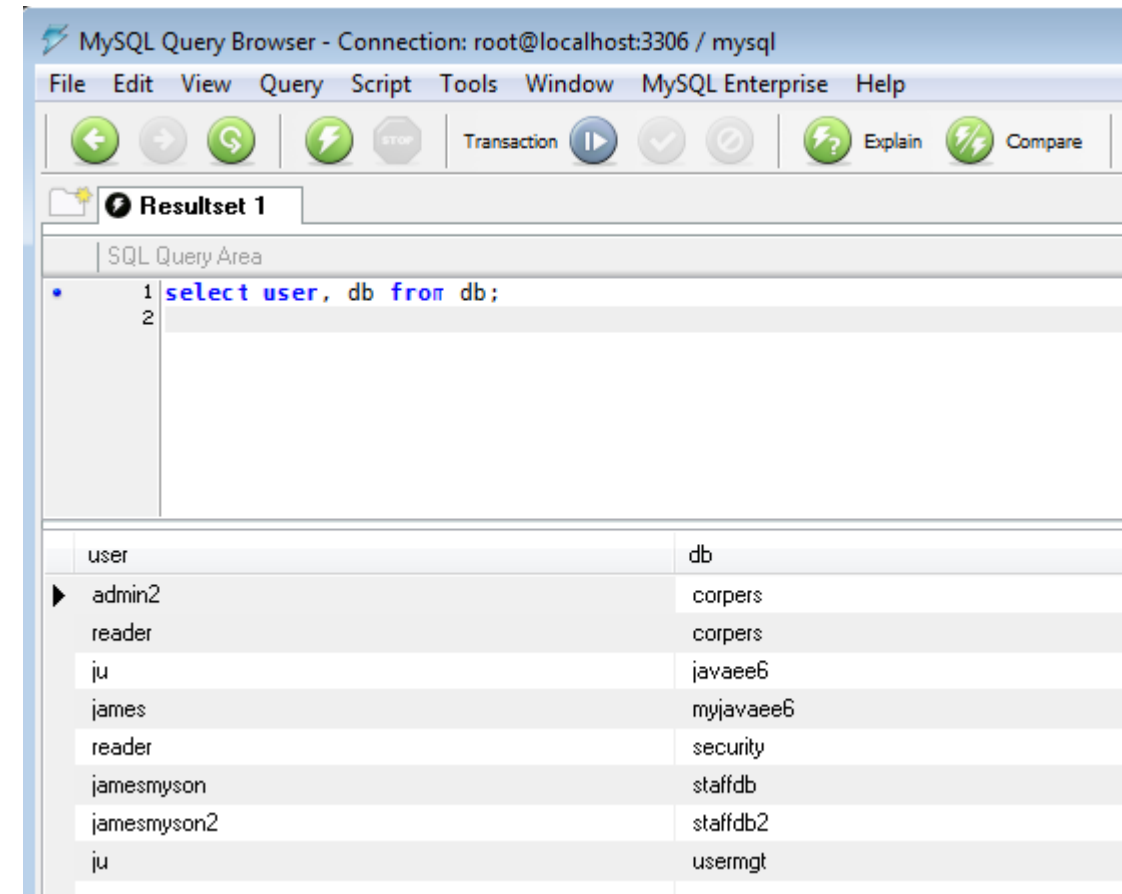
More close look at the mysql privileges system.

There are five tables in the mysql database that keep details about the privileges of the users. They include:

Db – It holds details of users and the privileges they have on each database.

To see exactly which databases users have access to, we write:

Select user, db from db;



The screenshot shows the MySQL Query Browser interface. The title bar indicates the connection is 'root@localhost:3306 / mysql'. The menu bar includes File, Edit, View, Query, Script, Tools, Window, MySQL Enterprise, and Help. Below the menu is a toolbar with icons for navigation and execution. The main area is labeled 'Resultset 1' and 'SQL Query Area'. The query 'select user, db from db;' is entered. Below the query area, a table displays the results of the query.

user	db
admin2	corpers
reader	corpers
ju	javaee6
james	myjavaee6
reader	security
jamesmyson	staffdb
jamesmyson2	staffdb2
ju	usermgt

We can also set privileges manually in the database. But when we do so, it is good for us to use the flush command so that MySQL can re-adjust to any changes made on the system.

References

<https://quizlet.com/159321719/wcmpt308-flash-cards/>
a good url for database studies

GRETSON TECH LTD