INTRODUCTION TO MYSQL

MySQL : It is an Open Source RDBMS Software that uses Structured Query Language . It is available free of cost.
Key Features of MySQL :

1. High Speed.
2. Ease of Use.
3. Available Free of Cost.
4. Supports standards based SQL.
5. Provides portability.
6. High Security.
7. Provides many data types.
8. Handles large database.

MySQL Data Types:

Every column (or data item) should belong to a unique domain (known as data type). These data types help to describe the kind of information a particular column holds. MySQL supports the ANSI SQL data types. Some of the commonly used data types along with their characteristics are as follows:

Class	Data Type	Description	Example
Text	CHAR(size)	A fixed-length string between 1 and 255 characters in length right-padded with spaces to the specified length when stored. Values must be enclosed in single quotes or double quotes.	'Maths' 'TexT'
	VARCHAR(size)	A variable-length string between 1 and 255 characters in length; for example VARCHAR(25). Values must be enclosed in single quotes or double quotes	'Computer' 'Me and u'
NUMERIC	DECIMAL(p,s)	It can represent number with or 17.3 without the fractional part. The size argument has two parts: precision and scale. Precision (p) indicates the number of significant digits and scale (s) maximum number of digits to the right of the decimal point	
	INT	It is used for storing integer values	345
Date	DATE	It represents the date including day, month and year between 1000-01-01 and 9999-12-31	2009-07-02

The Structured Query Language(SQL)

SQL (pronounced SEQUEL for Simple English Query Language) is Non-procedural universal data access language used to access and manipulate data stored in nearly all the data bases available currently. SQL standards are defined by ANSI (American National Standards Institute). SQL statements are used to retrieve and update data in a database. SQL works with database programs like MySQL, MS Access, DB2, Informix, MS SQL Server, Oracle, Sybase, etc. Most of the SQL database programs also have their own proprietary extensions in addition to the SQL standard.

SQL Commands

SQL commands can be classified into the following:

Data Definition Language (DDL): A database scheme is defined by set of definitions, which are expressed, by a special set of commands called Data Definition Language (DDL). They are used to create tables, databases, identify data items, provide unique names to the data items and to define the length and provide the range of values that each data item can assume. They are CREATE TABLE, ALTER TABLE and DROP TABLE commands.

Data Manipulation Language (DML):
The data manipulation language (DML) handles operations such as entering rows into a table, changing data, deleting rows, and extracting data from rows and tables. With DML, one does not change the table's structure, but rather its contents. It contains commands like INSERT, UPDATE and DELETE.

Transaction Control Language (TCL): A transaction is a one complete unit of work. A transaction is successfully completed in and only if all its constituent steps are successfully completed. To manage and control the transactions, the transaction control commands are used. e.g. COMMIT,ROLLBACK,SAVEPOINT.

WORKING WITH SQL

To work on MySQL , you need to open or create the database first:

To Create/Open Database:

mysql> CREATE DATABASE <name of database>;

Now the database with the given name will be created. One must be connected to the database before using it , as below:

mysql> use <name of database>;

Creating Tables
Tables are defined with the CREATE TABLE command. When tables are created its columns are named, data types and sizes supplied for each column. At least one column must be specified.

Syntax:

CREATE TABLE <TableName>(<ColumnName1> <Data Type1>,
<ColumnName2> <Data Type2>,….. ….,<ColumnNameN> <Data Type N>);

Example:

```
mysql> CREATE TABLE Students
(
      RollNo DECIMAL(3),
      Name VARCHAR(25)
);
```

Once the table is created we can insert the record in it, edit or delete existing records, and also we can search for desired record in a very comprehensive way using the SQL Select statement.

Creating tables with SQL Constraints :

- ✓ A Constraint is a condition or check applicable on a field or set of fields.
- ✓ Data constraints are the rules that are defined when a table is created.
- ✓ They can also be defined or modified after creating the tables.
- ✓ When constraints are defined any data entering in the table is first checked to satisfy the condition specified in particular constraint if it is, only then table data can be updated. If data updation/ insertion is violating the defined constraints, database rejects the data (entire record is rejected).
- ✓ When a constraint is applied to a single column, it is called a column level constraint but if a constraint is applied on a combination of columns it is called a table constraint.

Following constraints can be defined on a table in SQL:

Constraints name	Description
PRIMARY KEY	Used to create a primary key
NIQUE	to create a unique key
NOT NULL	to define that column will not accept null values.
FOREIGN KEY/ REFERENCES	to define referential integrity with another table.
DEFAULT	to define the columns default value.
CHECK	to define the custom rule.

Not Null and Default constraints can be applied only at column level rest all constraints can be applied on both column level and table levels.

Use of constraints

CREATE TABLE student (Srollno integer NOT NULL, …);
CREATE TABLE student (Srollno integer UNIQUE, …);
CREATE TABLE student (Srollno integer NOT NULL, Sclass integer, Sname varchar(30), Sclass DEFAULT 12);
CREATE TABLE student (Srollno integer CHECK (Srollno>0), Sclass integer, Sname varchar(30));
CREATE TABLE student (Srollno integer NOT NULL PRIMARY KEY, Sclass integer, Sname varchar(30));
CREATE TABLE teacher (Tid integer NOT NULL, FOREIGN KEY (Studentid) REFRENCES student (Sid));

Inserting the record in existing table

The INSERT INTO command append a new record to an existing table and initializes it to desired values.

Syntax:

```
INSERT INTO table_name (column_name [,column_name])
VALUES (value [,value]);
INSERT INTO Student (RollNo,Name)
VALUES (12333,'Anu');
```

Inserting NULL Values:

```
INSERT INTO Student (RollNo,Name, Class, Grade)
VALUES (12333,'Anu',11, NULL);
```

Inserting Dates:

```
INSERT INTO Student (RollNo,Name, Class, DOB)
VALUES (12333,'Anu',11, '1998-02-24')
```

Inserting Data from another Table:

```
INSERT INTO Marks
SELECT * FROM Student
WHERE Class>10;
```

NOTE: Column names can be omitted if the values are entered in the same order in which they appear in the table. Insert into will give you an error if you omit to enter a mandatory value (non-null).

Deleting Existing records from the table :

The DELETE command deletes one, many, or even all records in a table, depending on the conditions that you specify.

Syntax:

```
DELETE FROM tablename
WHERE   search_conditions;
```

for example

```
DELETE FROM Students
WHERE RollNo>11255;
```

Note: The delete command is VERY dangerous. If run without conditions, it will delete ALL records in a table. In addition, SQL has no undo function. For instance,

```
DELETE FROM Students;
```

Will delete all records from Students table. This is not likely to be what you want.

Modifying the contents of records :

The UPDATE command changes one, many, or even all records in a table, depending on the conditions that you specify

Syntax:

```
UPDATE tablename
SET column_name = expression [,column_name = expression..]
[WHERE     search_conditions];
```

for example(assuming a customer table)

```
UPDATE customer
SET f_name = 'Thomas'
WHERE l_name = 'Smith' and
                date_of_birth = '3/2/1985';
```

An expression can be either a constant value (e.g., 'Thomas') or an operation done on another column or columns (see the example below, assuming a loan table with column rate.).

```
UPDATE TABLE loan
SET rate = rate + 1.5;
```

Because there is no condition (i.e., no WHERE) all records will be updated. All rates will be increased by 1.5.

Selecting data from existing table :

SQL SELECT statement is a comprehensive statement used to search/select records from one or more tables. All the analysis done on a database usually involves some form of select statement.

- **Choosing all fields (columns)** : Use a asterisk (*) to indicate all fields with the select statement:

```
SELECT *
FROM table_name;

SELECT *
FROM customer;
```

- **Choosing a selected list of fields (columns)**

```
SELECT column_name [,column_name]
FROM table_name;

SELECT f_name, l_name, date_of_birth
FROM customer;
```

NOTE: The order in which you list the columns affects their order in the resulting output. Items within [ ] are optional.

- **Temporarily renaming columns in query results**

```
SELECT column_heading AS column_name
[,column_heading AS column_name]
FROM table_name;
```

Example:

```
SELECT f_name as “Name”
FROM customer;
```

- **Including calculated columns in the results**

```
SELECT date_due, rate, principal, rate * principal FROM loan;
```

NOTE: If necessary, use parentheses to clarify order of precedence.

- **Eliminating duplicate query results with distinct**

If you use the keyword distinct after the keyword SELECT, you will only get unique rows. Example:

```
SELECT rate,
FROM loan;
```

(above will display all rate values might be repeated)

```
SELECT distinct rate
FROM loan;
```

(above will display only unique rate values, no repetition)

- **Selecting from all the rows:**

```
SELECT  ALL rate,
 FROM loan;
```

(above query will display all rate values)

- **Selecting rows:** WHERE clause is used to specify the condition for searching. Only those records will be retrieved that satisfy condition given with where clause.

```
  SELECT SELECT_list
FROM table_list
WHERE search_conditions;
```

Example:

```
SELECT * FROM customer
WHERE f_name = ‘Carl’;
```

- **Possible Search Conditions**
  - ✓ Comparison operators (=,<,>,!=.<>,<=,>=)

```
SELECT * FROM loan
WHERE principal > 100000000;
```

  - ✓ Ranges (between and not between; inclusive)

```
SELECT * FROM loan
WHERE rate BETWEEN 7.5 AND 8.5;
```

OR

```
SELECT * FROM loan
WHERE rate NOT BETWEEN 7.5 AND 8.5;
```

  - ✓ Lists (in and not in)

```
SELECT *
from Customer
where city IN ('Ahmedabad', 'Baroda', 'Delhi','Mumbai','Chennai');
```

OR

```
SELECT *
from Customer
where city NOT IN ('Ahmedabad', 'Baroda', 'Delhi','Mumbai','Chennai');
```

  - ✓ Null values

```
SELECT *
from Customer
where city is Null;
```

OR

```
SELECT *
from Customer
where city is Not Null;
```

  - ✓ Character matches (like and not like)

```
SELECT f_name, l_name
FROM customer
WHERE l_name LIKE 'Fos%';

SELECT f_name, l_name
FROM customer
WHERE l_name LIKE '_oster';
```

Note: "%" (matches any string of zero or more characters) and "_" (matches any one character). In addition to those, brackets can be used to include either ranges or sets of characters.
Combinations of previous options using logical operators and, or, and not etc.:

```
SELECT f_name, l_name
FROM customer
WHERE l_name LIKE 'San%' AND City NOT
IN ('Baroda','Delhi')
```

- **Some more examples:**
  - ✓ 'Am%' matches any string starting with Am.
  - ✓ '%Singh%' matches any string containing 'Singh'
  - ✓ '%a' matches any string ending with 'a'
  - ✓ '_ _ _' matches any string that is exactly 3 characters long.
  - ✓ '_ _ %' matches any string that has at least 2 characters long.
  - ✓ '_ _ _g' matches any string that is 4 characters along with 3 characters in the beginning but 'g' as the 4th character.
  - ✓

- **Viewing a tables structures**

  Describe/ Desc statement is used to see the structure of a table:

  ```
  Desc <tablename> ;
  Describe <tablename>;
  ```

- **Sorting records**

  The output of a SELECT query can be sorted in ascending or descending order on one or more columns, the default is ascending. This is important to note that the data in table is not sorted, only the results that appear on the screen are sorted.

  Syntax:

  ```
  SELECT <column name> [,<column name>, ….]
   FROM <table name>
   [WHERE <condition>]
   [ORDER BY <column name> [, <column name>…]];
  ```

  Example: (Sorting on single column)

  ```
  SELECT * FROM EMPL
  ORDER BY ENAME;
  ```

  Example : (Sorting on Multiple columns)

  ```
  SELECT * FROM EMPL
  ORDER BY ENAME, JOB;
  ```

- **Adding a column:**

  The ALTER TABLE command is used to change definitions of existing tables . It can add columns, delete columns or change their size.

  Syntax:

  ```
  ALTER TABLE <table name>
  ADD (<column name> <data type with size> <constraints>);
  ```

  Example:

  ```
  ALTER TABLE Students
  ADD ( age NUMBER (2) CHECK (age > 5));
  ```

- **Modify a column :**

Syntax :

```
ALTER TABLE <table name>
MODIFY ( column name newdatatype (newsize));
```

Example:

```
ALTER TABLE Students
MODIFY ( age NUMBER (1));
```

- **Changing a column name:**

```
ALTER TABLE <table name>
CHANGE <old_column_name> <new_column_name> <column definition> ;
```

Example:

```
ALTER TABLE Students
CHANGE age s_age NUMBER (2)
```

- **Removing table components**

  - ✓ To remove primary key constraints

```
ALTER TABLE Students
DROP primary key;
```

  - ✓ To remove column from the table

```
ALTER TABLE Students
DROP COLUMN age ;
```

- **Drop a table from database:**

```
DROP TABLE <table name> ;
```

Example:

```
DROP TABLE Students;
```

Operator Precedence :

All the operators have precedence. Precedence is the order in which different operators are evaluated. Various operators in descending order of precedence (top to bottom) are listed below:

1	!
2	(unary minus)
3	^
4	*,/,DIV,%, MOD
5	-,+
6	=, <=, >, >=, =, !=, IS, LIKE, IN
7	BETWEEN
8	NOT
9	&&, AND
10	||, OR

MySQL Functions

Functions

A function is a predefined command set that performs some operation and returns the single value.

Numeric Functions

- POWER() : Returns the argument raised to the specified power. POW () works the same way.

  Example:(i)POW(2,4):Result:16 (ii)POW(2,-2):Result:0.25 (iii)POW(-2,3):Result: -8

- ROUND() : ROUND(X) Rounds the argument to the zero decimal place, Where as ROUND(X,d) rounds the argument to d decimal places.

  Example : (i) ROUND(-1.23); Result: -1 (ii) ROUND(-1.58); Result: -2
  (iii) ROUND(1.58); Result: 2 (iv) ROUND(3.798, 1); Result: 3.8
  (v) ROUND(1.298, 0); Result: 1 (vi) ROUND(23.298, -1); Result: 20
  (vii) ROUND(25.298,-1); result: 30

- TRUNCATE() : Truncates the argument to specified number of decimal places.

  Example: (i) TRUNCATE (7.29,1) Result: 7.2 (ii) TRUNCATE(27.29,-1) Result: 20

- SIGN() : Returns sign of a given number.

  Example : (i) SIGN (15) Result : 1 : (ii) SIGN (-15) Result : -1 : (iii) SIGN (0) Result : 0.

- SQRT : Returns the square root of given number.

  Example : (i) SQRT (25) Result : 5

Character/String Functions

- LENGTH() : Returns the length of a string in bytes/no.of characters in string.

  Example: LENGTH('INFORMATICS'); Result:11

- CHAR() : Returns the corresponding ASCII character for each integer passed.

  Example : CHAR(65) ; Result : A

- CONCAT(): Returns concatenated string i.e. it adds strings.

  Example : CONCAT('Informatics',' ','Practices'); Result : Informatics Practices'

- INSTR(): Returns the index of the first occurrence of substring.

  Example : INSTR('Informatics',' mat'); Result : 6(since 'm' of 'mat' is at 6th place)

- LOWER()/ LCASE(): Returns the argument after converting it in lowercase.

  Example: LOWER('INFORMATICS'); Result : informatics

- UPPER()/ UCASE(): Returns the argument after converting it in uppercase.

  Example: UCASE('informatics'); Result : INFORMATICS

- LEFT() : Returns the given number of characters by extracting them from the left side of the given string

  Example : LEFT('INFORMATICS PRACTICES', 3); Result : INF

- RIGHT(): Returns the given number of characters by extracting them from the right side of the given string

Example : RIGHT('INFORMATICS PRACTICES',3); Result : CES

- MID(): Returns a substring starting from the specified position in a given string.

  Example: MID('INFORMATICS PRACTICES',3,4); Result : FORM

- SUBSTR() : Returns a substring from a given string.

  Example: SUBSTR('INFORMATICS' , 3 , 4) ; Result : FORM

- LTRIM() : Removes leading spaces.

  Example : LTRIM(' INFORMATICS'); Result: 'INFORMATICS'

- RTRIM(): Removes trailing spaces.

  Example : RTRIM('INFORMATICS '); Result: 'INFORMATICS'

- TRIM() : Removes leading and trailing spaces.

  Example: TRIM(' INFORMATICS '); Result: 'INFORMATICS'

Date/Time Functions

- CURDATE() : Returns the current date

  Example: CURDATE(); Result: '2012-09-18'

- NOW() : Returns the current date and time

  Example: NOW(); Result : '2010-07-21 13:58:11'

- SYSDATE() : Return the time at which the function executes

  Example: SYSDATE(); Result: '2010-07-21 13:59:23'

- DATE() : Extracts the date part of a date or datetime expression

  Example: DATE('2003-12-31 01:02:03'); Result:: '2003-12-31'

- MONTH() Returns the month from the date passed

  Example: MONTH('2010-07-21'); Result : 7

- YEAR() : Returns the year.

  Example: YEAR('2010-07-21'); Result : 2010

- DAYNAME() : Returns the name of the weekday

  Example: DAYNAME('2010-07-21'); Result : WEDNESDAY

- DAYOFMONTH() : Returns the day of the month (0-31)

  Example: DAYOFMONTH('2010-07-21'); Result: 21

- DAYOFWEEK() : Returns the weekday index of the argument

  Example: DAYOFWEEK('2010-07-21'); Result: 4 (Sunday is counted as 1)

- DAYOFYEAR() : Return the day of the year(1-366)

  Example: DAYOFYEAR('2010-07-21'); Result: 202