# LAB MANUAL

## *DATABASE SYSTEMS*


**DEPARTMENT OF SOFTWARE ENGINEERING**
**FACULTY OF ENGINEERING & COMPUTING**
**NATIONAL UNIVERSITY OF MODERN LANGUAGES**
**ISLAMABAD**

## Preface

This lab is about database management system. In this lab students will learn about Database from basic and they will also learn about the concept and usage of queries to manage and maintain the database using the SQL statements. In the lab SQL statements will be used to manage and maintain the database. Students will get the knowledge regarding creating table and performing different operations on the tables. The Theoretical Description is to familiarize students with SQL and to equip students with hands-on experience of implementing SQL for managing the database management system. This lab provides hands-on experience of implementing DBMS concepts using Structured Query Language, which ultimately helps students to understand the basic concept and usage of DBMS and provides them skills to manage and manipulate the database.

## Tools/ Technologies

- Mysql OR SQL management studio

# TABLE OF CONTENTS

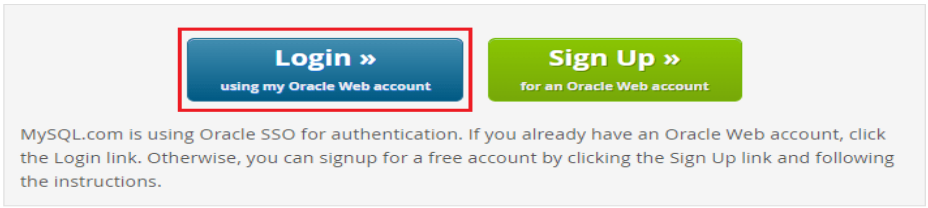
# LAB 1: Installation of Relational Database Management Systems

**Objectives**

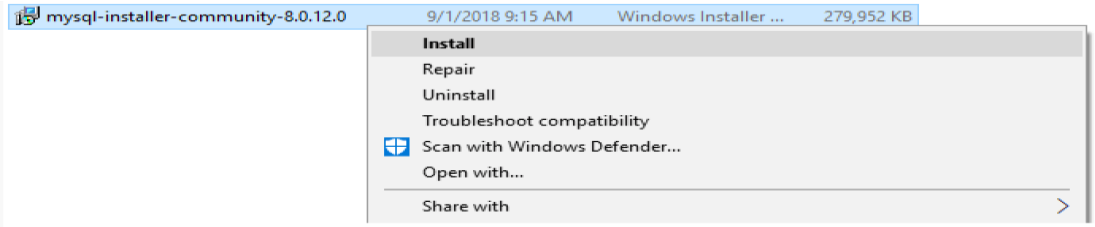
- To discuss the role of relational database management studio
- To install MySQL or Oracle or Sql server

**Theoretical description**

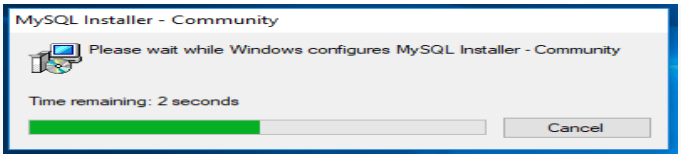
1. **Step 1:** Go to mysql.com
2. **Step 2:**Go to downloads and select mysql community gpl downloads and select mysqlinstaller for windows
3. **Step 3:**It will show you **Generally Available (GA) Releases**. Where we can see two different installers, one is a web community installer which comes as a little file and another one is MySQL installer community. Click the Download button on the second one (mysql-installer-community)
4. **Step 4** It will ask your MySQL credentials to download the .msi file. If you have your credentials, you can log in or else if you wish to sign up now you can click on thegreen coloured signup button.If you are not interested in login or sign up for now, you can directly go and click on **No thanks, just start my download** option. It will download selected MySQL for you on your local machine



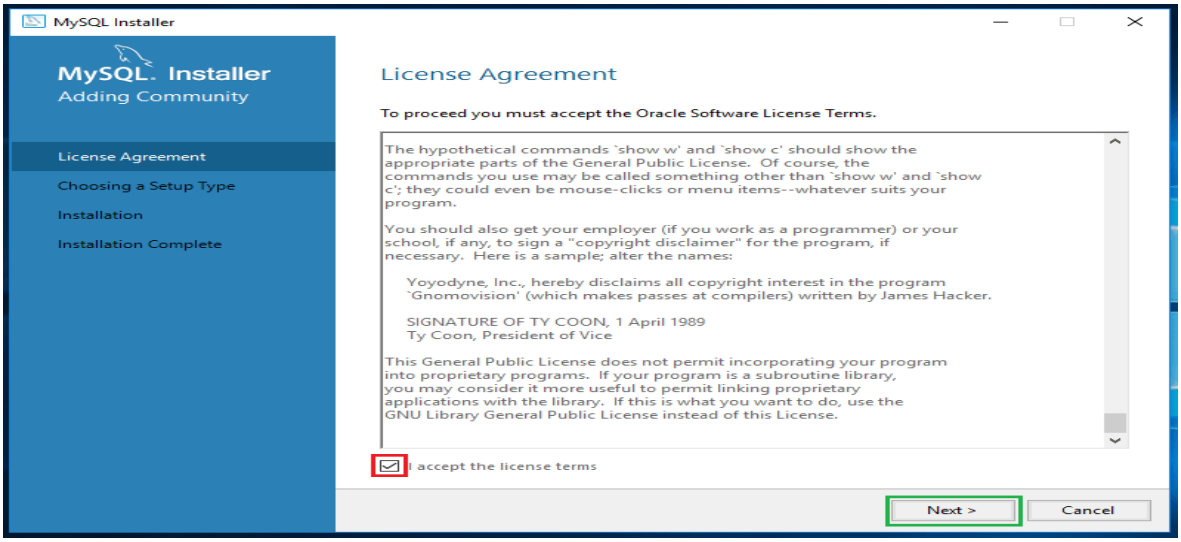

5. **Step 5:**Go to downloads folder where you can see the **mysql-installer- community** file, right click on that file and click **Install** option.



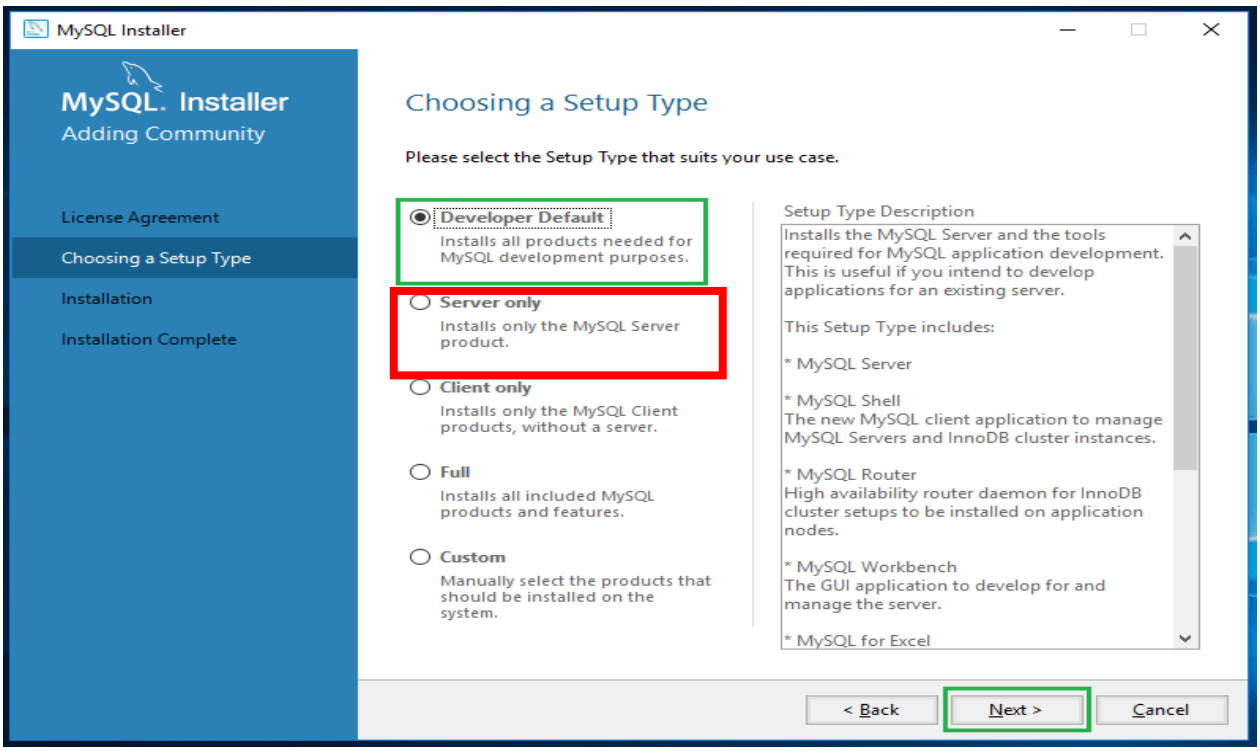

This window configures the installer, in the middle, it may ask you for permissions to change your computer settings or firewall confirmation, you can accept and then it will take a few seconds to configure the installer.



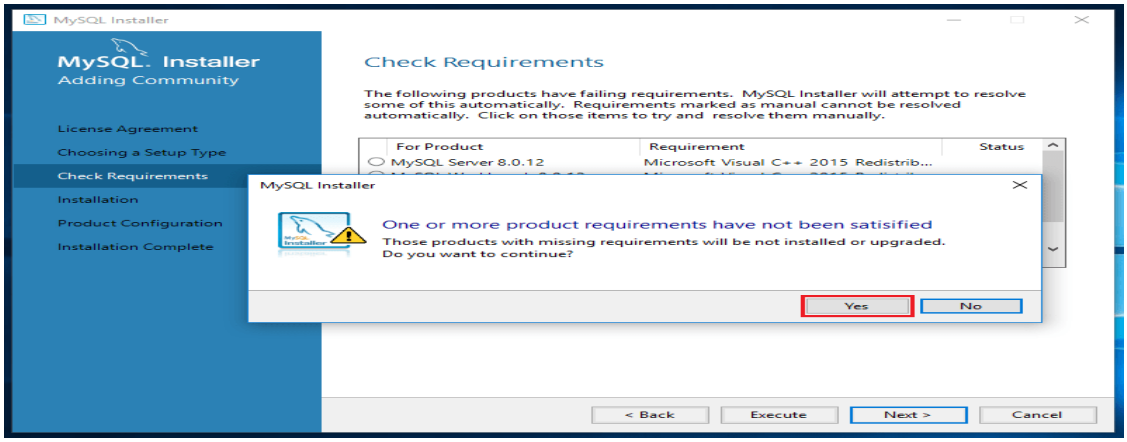
6. **Step 6:** Read the license agreement and accept the license terms.

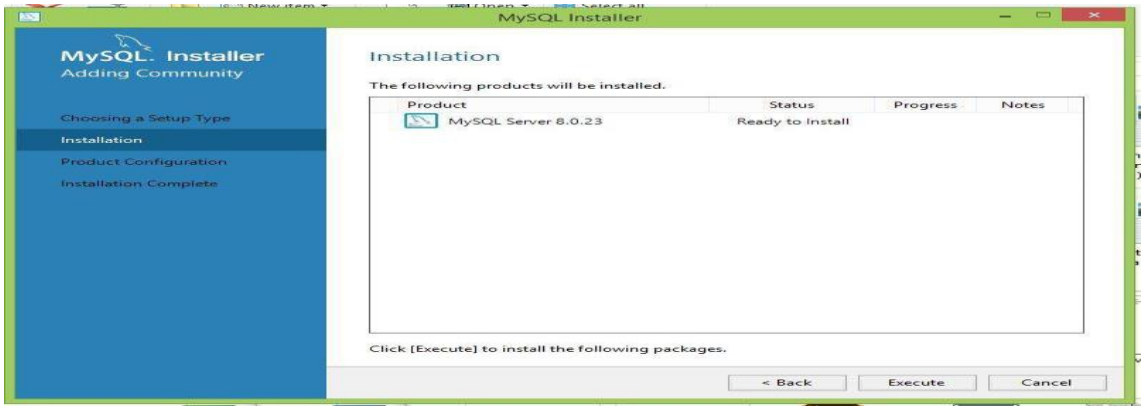



7. **Step 7:**This window provides you to set up different types of MySQL installations. You can set up Mysql in 5 different types as provided below. Now I am selecting the server only not I am a developer so that I need all the products which help my development Theoretical Descriptions. Click on Next.



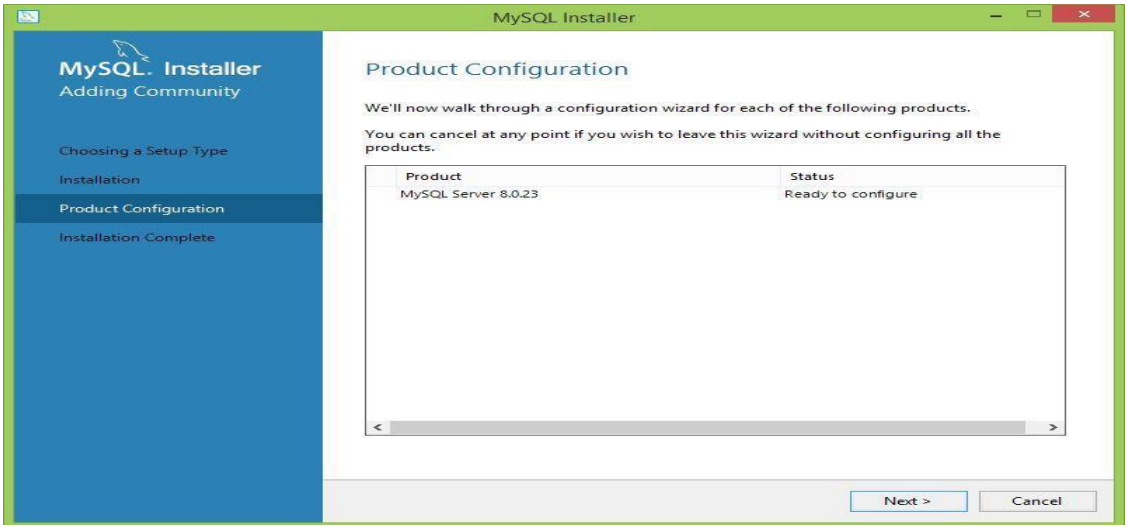
8. **Step 8:**Based on your Windows configuration, it may prompt you like “One or moreproduct requirements have not been satisfied”. You can just click on YES.

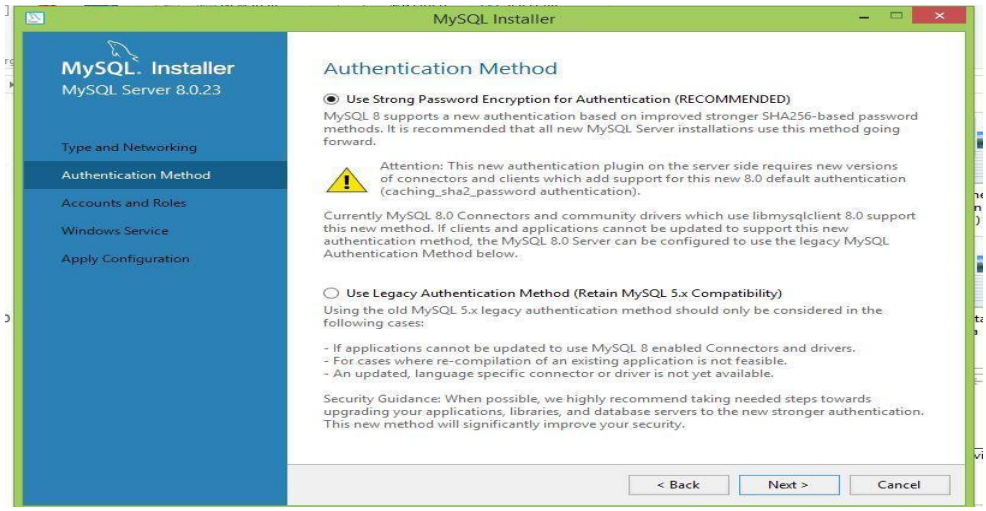



9. **Step 9: Click execute**

10. **Step 10:**Upon successful execution of all required products, now the MySQL allows us to configure the server settings. Click on Next to configure the server

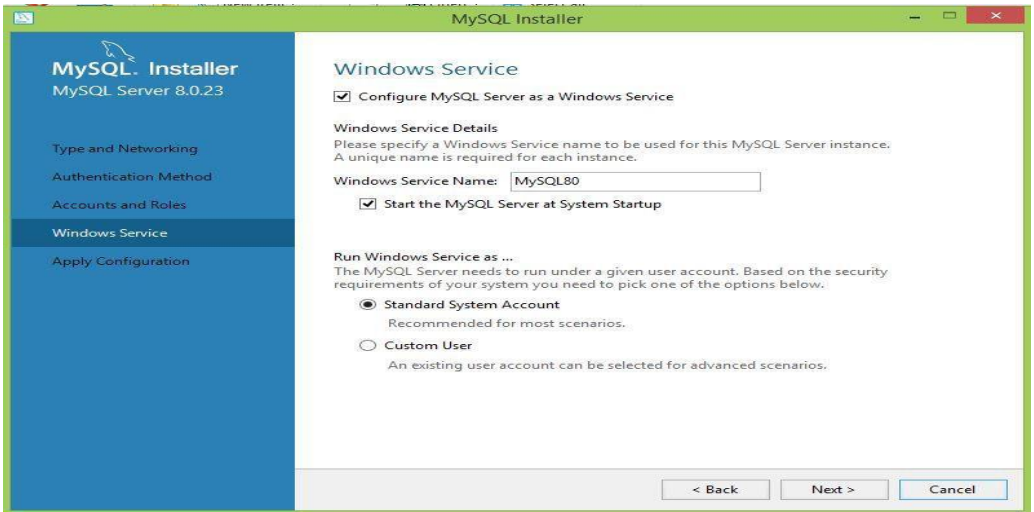




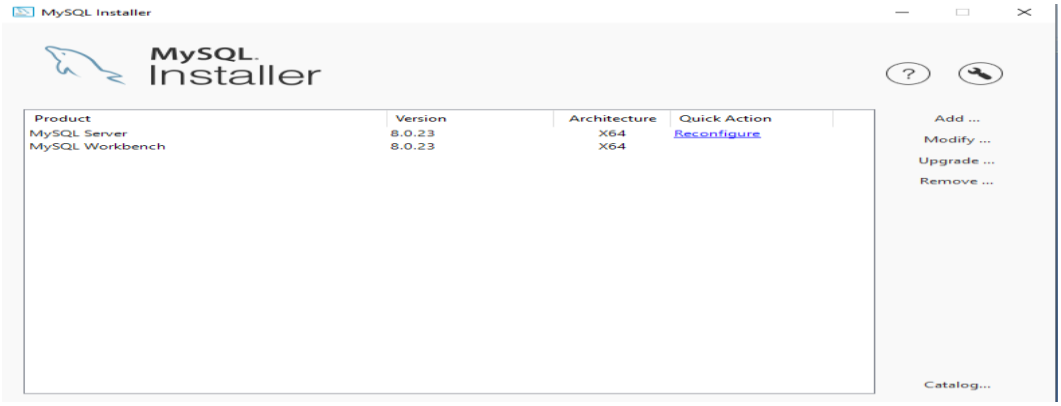

**11. Step 11**:Select default setting


**12. Step 12:Add password**

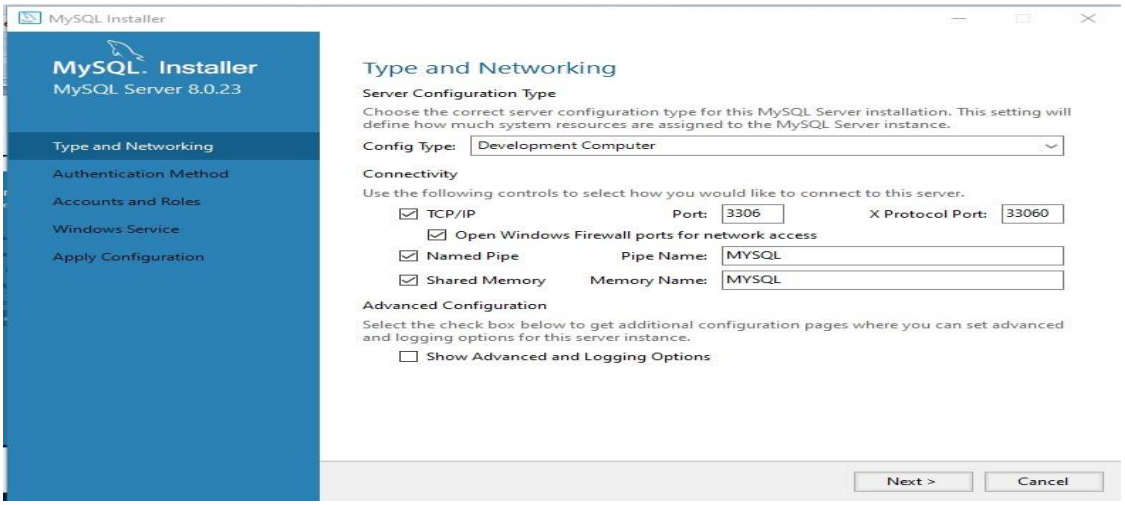
**13. Step 13:Select default setting**



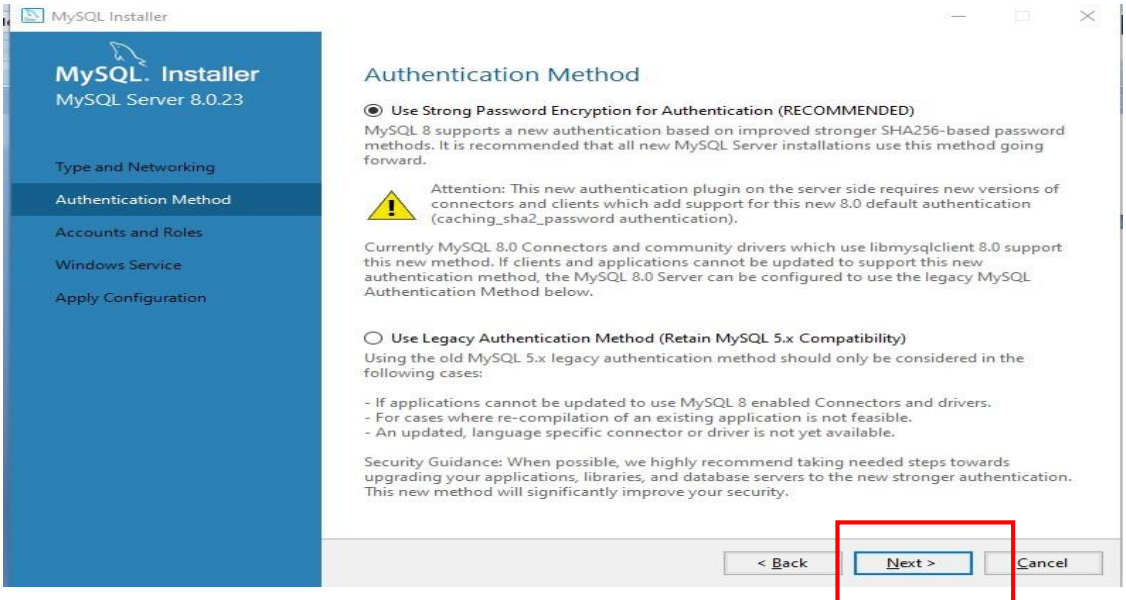
**14. Step 14:**Click on **reconfigure**



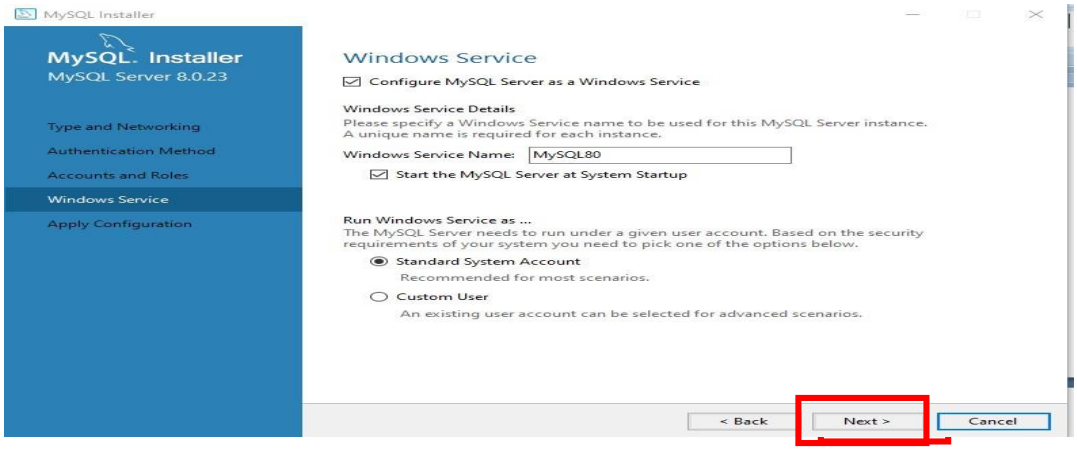

15. **Step 15**:Check named pipe and shared memory



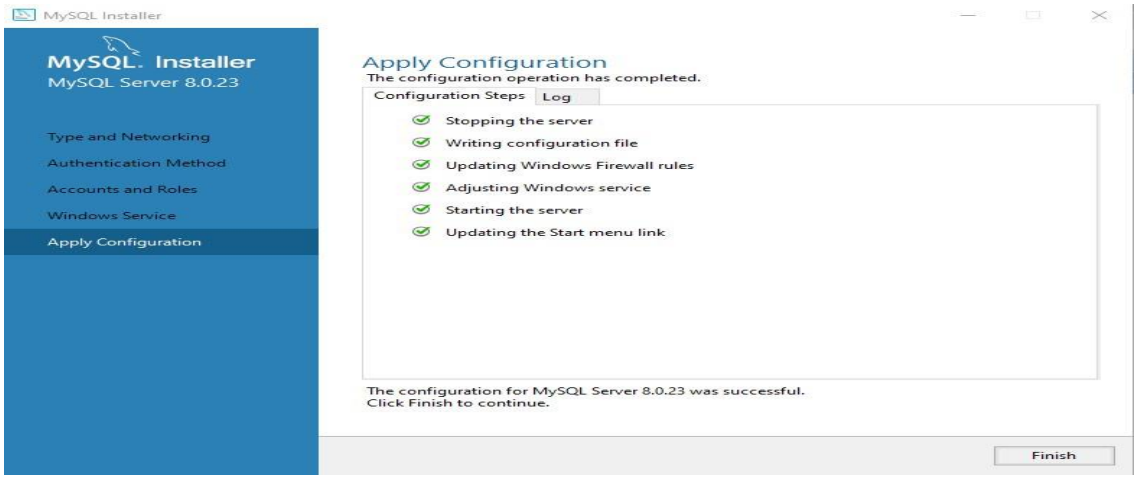

16. Step 16



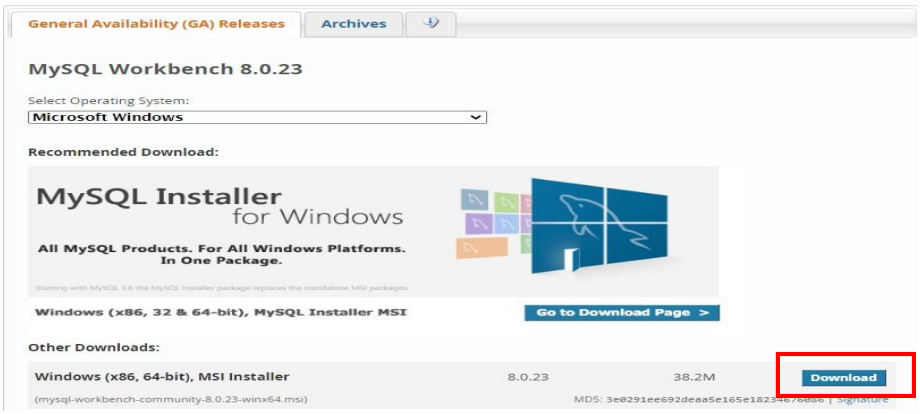
17. **Step 17:**Enter root password i.e password in my case and click check it which shows green symbol andclick next



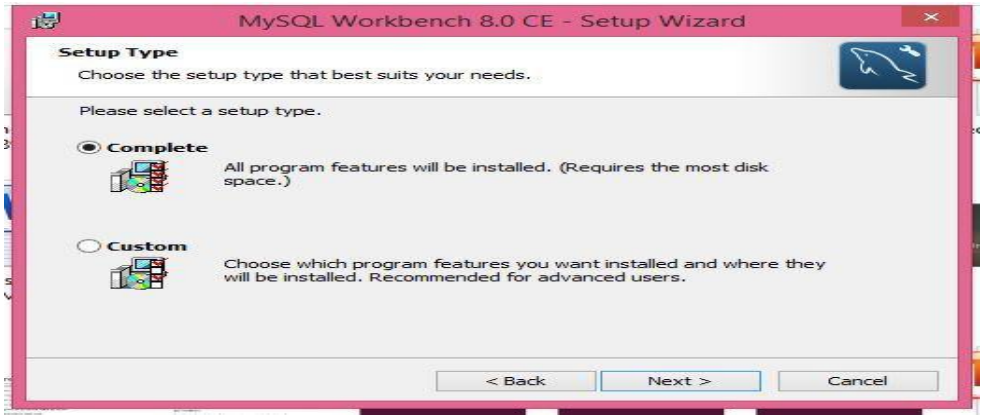

18. **Step 18**:Again select next
19. **Step 19:**Click on execute and than finish



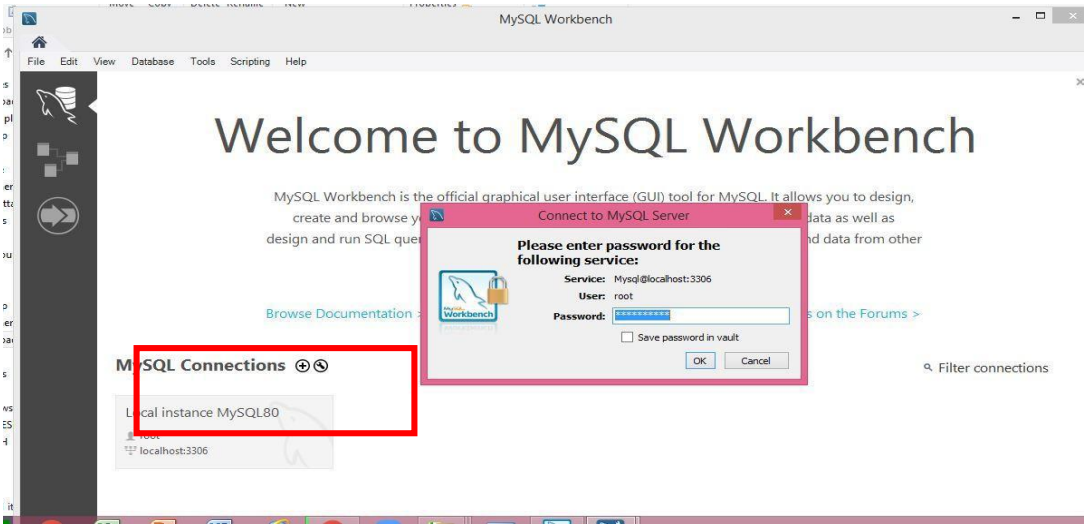

20. **Step 20:**Now go back to mysql.com and select downloads and select mysql community gpl downloadsand select mysql work bench and click on download button in below screen



**21. Step 21: select complete**




**22. Step 22:**Work bench is installed and screen shown is click on local instance and enter




your password

**Lab Task**

Q1.Observe the tool and explain all its findings in your own words with screebshots

Q2.Import any Mysql database and perform following tasks.Show output as well as give reasoning

1. Expand the database from left side of workbench window and list down the catagories of tables in the specific database
2. Click on any table of your choice to inspect table and show the schema attribute names and other metadata
3. Choose the above mentioned table and navigate through the contents of database tables
4. Observe multiple options in data view such as sorting
5. Observe errors warnings and messages within log pane

# LAB 2: Retrieving Data Using the SQL SELECT Statement

**Objectives**

- To list the capabilities of SQL SELECT statements
- To execute a basic SELECT statement

**Theoretical description**

**SQL KEYWORDS**

Basic SELECT Statement

```
SELECT *|{[DISTINCT] column|expression [alias],...}
FROM    table;
```

- SELECT identifies the columns to be displayed.
- FROM identifies the table containing those columns.

**Selecting All Columns**

```
SELECT *
FROM   departments;
```

| | DEPARTMENT_ID | DEPARTMENT_NAME | MANAGER_ID | LOCATION_ID |
|---|---|---|---|---|
| 1 | 10 | Administration | 200 | 1700 |
| 2 | 20 | Marketing | 201 | 1800 |
| 3 | 50 | Shipping | 124 | 1500 |
| 4 | 60 | IT | 103 | 1400 |
| 5 | 80 | Sales | 149 | 2500 |
| 6 | 90 | Executive | 100 | 1700 |
| 7 | 110 | Accounting | 205 | 1700 |
| 8 | 190 | Contracting | (null) | 1700 |

**Selecting Specific Columns**

```
SELECT department_id, location_id
FROM   departments;
```

| | DEPARTMENT_ID | LOCATION_ID |
|---|---|---|
| 1 | 10 | 1700 |
| 2 | 20 | 1800 |
| 3 | 50 | 1500 |
| 4 | 60 | 1400 |
| 5 | 80 | 2500 |
| 6 | 90 | 1700 |
| 7 | 110 | 1700 |
| 8 | 190 | 1700 |

**Writing SQL Statements**

- SQL statements are not case-sensitive.

For Example:

```
SELECT *
FROM DEPARTMENTS;
```

OR

```
select *
from departments;
```

SQL statements can be entered on one or more lines.

For Example:
```
SELECT *
FROM DEPARTMENTS;
```
OR
```
select * from departments;
```

- Keywords cannot be abbreviated or split across lines

For Example: SEL * FRO DEPARTMENTS; OR
```
SEL
ECT * FROM DEPARTMENTS;
```

Clauses are usually placed on separate lines

For Example:
```
SELECT *
FROM DEPARTMENTS;
```

- Indents are used to enhance readability.

For Example:
```
SELECT department_id, department_name
FROM   departments;
```

- In SQL*Plus, you are required to end each SQL statement with a semicolon (;).

**Arithmetic Expressions**

Create expressions with number and date data by using arithmetic operators.

| Operator | Description |
|---|---|
| + | Add |
| - | Subtract |
| * | Multiply |
| / | Divide |

**Using Arithmetic Operators**

```
SELECT 1+2+3+6;
```

| | 1+2+3+6 |
|---|---|
| ▶ | 12 |

```
SELECT last_name, salary, salary + 300
FROM   employees;
```

| | LAST_NAME | SALARY | SALARY+300 |
|---|---|---|---|
| 1 | King | 24000 | 24300 |
| 2 | Kochhar | 17000 | 17300 |
| 3 | De Haan | 17000 | 17300 |
| 4 | Hunold | 9000 | 9300 |
| 5 | Ernst | 6000 | 6300 |
| 6 | Lorentz | 4200 | 4500 |
| 7 | Mourgos | 5800 | 6100 |
| 8 | Rajs | 3500 | 3800 |
| 9 | Davies | 3100 | 3400 |
| 10 | Matos | 2600 | 2900 |

...

```
SELECT first_name, last_name, salary, salary – 300 FROM employees;
```

```
SELECT first_name, last_name, salary, salary * 2 FROM employees;
```

```
SELECT first_name, last_name, salary, salary / 1000 FROM employees;
```

**Operator Precedence**

- Multiplication and division take priority over addition and subtraction.
- Operators of the same priority are evaluated from left to right.
- Parentheses are used to force prioritized evaluation and to clarify statements

```
SELECT last_name, salary, 12*salary+100
FROM   employees;
```

(1)

| | LAST_NAME | SALARY | 12*SALARY+100 |
|---|---|---|---|
| 1 | King | 24000 | 288100 |
| 2 | Kochhar | 17000 | 204100 |
| 3 | De Haan | 17000 | 204100 |

...

```
SELECT last_name, salary, 12*(salary+100)
FROM   employees;
```

(2)

| | LAST_NAME | SALARY | 12*(SALARY+100) |
|---|---|---|---|
| 1 | King | 24000 | 289200 |
| 2 | Kochhar | 17000 | 205200 |
| 3 | De Haan | 17000 | 205200 |

...

**Defining a Column Alias**

A column alias:

- Renames a column heading
- Is useful with calculations
- Immediately follows the column name (There can also be the optional AS keyword between the column name and alias.)

- Requires double quotation marks if it contains spaces or special characters, or if it is case-sensitive

**Using Column Aliases**

```
SELECT last_name AS name, commission_pct comm
FROM   employees;
```

| | NAME | COMM |
|---|---|---|
| 1 | King | (null) |
| 2 | Kochhar | (null) |
| 3 | De Haan | (null) |

...

```
SELECT last_name "Name" , salary*12 "Annual Salary"
FROM   employees;
```

| | Name | Annual Salary |
|---|---|---|
| 1 | King | 288000 |
| 2 | Kochhar | 204000 |
| 3 | De Haan | 204000 |

...

**Concatenation Operator**

A concatenation operator:

- Links columns or character strings to other columns
- Is represented by command concat
- Creates a resultant column that is a character expression

```
select concat (customerName,  customerNumber) As Customerid from customers;
```

| Customerid |
|---|
| Atelier graphique103 |
| Signal Gift Stores112 |
| Australian Collectors, Co.114 |
| La Rochelle Gifts119 |
| Baane Mini Imports121 |
| Mini Gifts Distributors Ltd.124 |
| Havel & Zbyszek Co125 |

**Duplicate Rows**

The default display of queries is all rows, including duplicate rows.

```
SELECT department_id
FROM   employees;
```

| DEPARTMENT_ID |
|---|
| 90 |
| 90 |
| 90 |
| 60 |
| 60 |
| 60 |
| 50 |
| 50 |
| 50 |

**Eliminate duplicate rows by using the `DISTINCT` keyword in the `SELECT` clause.**

```
SELECT DISTINCT department_id
FROM   employees;
```

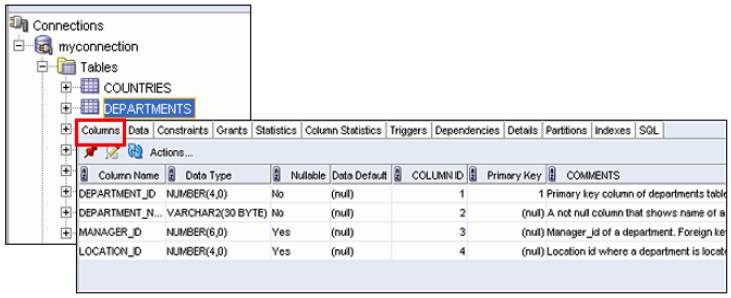
| DEPARTMENT_ID |
|---|
| 10 |
| 20 |
| 50 |
| 60 |
| 80 |
| 90 |
| 110 |
| |

8 rows selected.

**Displaying the Table Structure**

- Use the DESCRIBE command to display the structure of a table.
- Or, select the table in the Connections tree and use the Columns tab to view the table structure.

```
DESC[RIBE] tablename
```



**Using the DESCRIBE Command**

```
DESCRIBE employees
```

```
DESCRIBE employees
Name                           Null     Type
------------------------------ -------- ----------------------
EMPLOYEE_ID                    NOT NULL NUMBER(6)
FIRST_NAME                              VARCHAR2(20)
LAST_NAME                      NOT NULL VARCHAR2(25)
EMAIL                          NOT NULL VARCHAR2(25)
PHONE_NUMBER                            VARCHAR2(20)
HIRE_DATE                      NOT NULL DATE
JOB_ID                         NOT NULL VARCHAR2(10)
SALARY                                  NUMBER(8,2)
COMMISSION_PCT                          NUMBER(2,2)
MANAGER_ID                              NUMBER(6)
DEPARTMENT_ID                           NUMBER(4)

11 rows selected
```

**Lab Task**

**Consider the database given to you and perform following tasks on it**

Q1. Write an SQL query to display all records from city table

Q2. Write an SQL Query to display the following records. Give reasoning also that which table would be used get the following records

- Name
- local name
- government form
- Head of state
- Capital

Q3. Describe the structure of country language table and city table and also explain in your words about the output result

Q4. Consider country table and use arithmetic operators within SQL query to calculate 10 % on population column

Q5 Consider any one table of database. Take at least any one appropriate scenario and apply operator precedence. Also give reasoning

Q6. Consider country language table. Write SQL query to concatenate columns i.e. Country code and language .Also use column alias to rename the concatenated column as country details column

# LAB 3: Restricting using where clause and Sorting Data using order by clause

## Objectives

- To limit the rows that are retrieved by a query
- To sort the rows that are retrieved by a query
- To use ampersand substitution to restrict and sort output at run time

## Theoretical Description

### LIMITING ROWS USING SELECTION

EMPLOYEES

| | EMPLOYEE_ID | LAST_NAME | JOB_ID | DEPARTMENT_ID |
|---|---|---|---|---|
| 1 | 100 | King | AD_PRES | 90 |
| 2 | 101 | Kochhar | AD_VP | 90 |
| 3 | 102 | De Haan | AD_VP | 90 |
| 4 | 103 | Hunold | IT_PROG | 60 |
| 5 | 104 | Ernst | IT_PROG | 60 |
| 6 | 107 | Lorentz | IT_PROG | 60 |

...

"retrieve all employees in department 90"

| | EMPLOYEE_ID | LAST_NAME | JOB_ID | DEPARTMENT_ID |
|---|---|---|---|---|
| 1 | 100 | King | AD_PRES | 90 |
| 2 | 101 | Kochhar | AD_VP | 90 |
| 3 | 102 | De Haan | AD_VP | 90 |

- Restrict the rows that are returned by using the `WHERE` clause:

```
SELECT *|{[DISTINCT] column|expression [alias],...}
FROM   table
[WHERE condition(s)];
```

- The `WHERE` clause follows the `FROM` clause.

### Using the WHERE Clause

```
SELECT employee_id, last_name, job_id, department_id
FROM   employees
WHERE  department_id = 90 ;
```

| | EMPLOYEE_ID | LAST_NAME | JOB_ID | DEPARTMENT_ID |
|---|---|---|---|---|
| 1 | 100 | King | AD_PRES | 90 |
| 2 | 101 | Kochhar | AD_VP | 90 |
| 3 | 102 | De Haan | AD_VP | 90 |

## CHARACTER STRINGS AND DATES

- Character strings and date values are enclosed with single quotation marks.
- Character values are case-sensitive and date values are format-sensitive.

```
SELECT   ename, job, deptno
FROM     emp
WHERE    ename = 'JAMES';
```

### Comparison Operators

| Operator | Meaning |
|---|---|
| = | Equal to |
| > | Greater than |
| >= | Greater than or equal to |
| < | Less than |
| <= | Less than or equal to |
| <> | Not equal to |
| BETWEEN ...AND... | Between two values (inclusive) |
| IN(set) | Match any of a list of values |
| LIKE | Match a character pattern |
| IS NULL | Is a null value |

### OTHER COMPARISON OPERATORS

| Operator | Meaning |
|---|---|
| BETWEEN AND | Between two values inclusive |
| IN list | Match any of a list of values |
| LIKE | Match a character pattern |
| IS NULL | Is a null value |

## Range Conditions Using the BETWEEN Operator

Use the BETWEEN operator to display rows based on a range of values:

```
SELECT last_name, salary
FROM   employees
WHERE  salary BETWEEN 2500 AND 3500 ;
```

Lower limit  Upper limit

| | LAST_NAME | SALARY |
|---|---|---|
| 1 | Rajs | 3500 |
| 2 | Davies | 3100 |
| 3 | Matos | 2600 |
| 4 | Vargas | 2500 |

Use the IN operator to test for values in a list:

```
SELECT employee_id, last_name, salary, manager_id
FROM   employees
WHERE  manager_id IN (100, 101, 201);
```

| | EMPLOYEE_ID | LAST_NAME | SALARY | MANAGER_ID |
|---|---|---|---|---|
| 1 | 101 | Kochhar | 17000 | 100 |
| 2 | 102 | De Haan | 17000 | 100 |
| 3 | 124 | Mourgos | 5800 | 100 |
| 4 | 149 | Zlotkey | 10500 | 100 |
| 5 | 201 | Hartstein | 13000 | 100 |
| 6 | 200 | Whalen | 4400 | 101 |
| 7 | 205 | Higgins | 12000 | 101 |
| 8 | 202 | Fay | 6000 | 201 |

- Use the LIKE operator to perform wildcard searches of valid search string values.
- Search conditions can contain either literal characters or numbers:
  - % denotes zero or many characters.
  - _ denotes one character.

```
SELECT   first_name
FROM     employees
WHERE    first_name LIKE 'S%' ;
```

- You can combine the two wildcard characters (%, _) with literal characters for pattern matching:

```
SELECT last_name
FROM   employees
WHERE  last_name LIKE '_o%' ;
```

| | LAST_NAME |
|---|---|
| 1 | Kochhar |
| 2 | Lorentz |
| 3 | Mourgos |

## Using the AND Operator

AND requires both the component conditions to be true:

**Defining Conditions Using the Logical Operators**

| Operator | Meaning |
|---|---|
| AND | Returns TRUE if *both* component conditions are true |
| OR | Returns TRUE if *either* component condition is true |
| NOT | Returns TRUE if the condition is false |

```
> SELECT empno, ename, job, sal
 FROM    emp
 WHERE   sal> 1100
 AND     job='CLERK';
```

```
     EMPNO ENAME      JOB              SAL
---------- ---------- --------- ----------
      7876 ADAMS      CLERK            1100
      7934 MILLER     CLERK            1300
```

### Using the OR Operator

OR requires either component condition to be true:

```
SQL> SELECT empno, ename, job, sal
  2   FROM    emp
  3   WHERE   sal> 1100
  4   OR      job= 'CLERK';
```

| EMPNO | ENAME | JOB | SAL |
|---|---|---|---|
| 7839 | KING | PRESIDENT | 5000 |
| 7698 | BLAKE | MANAGER | 2850 |
| 7782 | CLARK | MANAGER | 2450 |
| 7566 | JONES | MANAGER | 2975 |
| 7654 | MARTIN | SALESMAN | 1250 |
| 7900 | JAMES | CLERK | 950 |

**USING NOT OPERATOR**

```
SELECT last_name, job_id
FROM   employees
WHERE  job_id
       NOT IN ('IT_PROG', 'ST_CLERK', 'SA_REP');
```

| | LAST_NAME | JOB_ID |
|---|---|---|
| 1 | De Haan | AD_VP |
| 2 | Fay | MK_REP |
| 3 | Gietz | AC_ACCOUNT |
| 4 | Hartstein | MK_MAN |
| 5 | Higgins | AC_MGR |
| 6 | King | AD_PRES |
| 7 | Kochhar | AD_VP |
| 8 | Mourgos | ST_MAN |
| 9 | Whalen | AD_ASST |
| 10 | Zlotkey | SA_MAN |

**RULES OF PRECENDENCE**

| Order Evaluated | Operator |
|---|---|
| 1 | All comparison operators |
| 2 | NOT |
| 3 | AND |
| 4 | OR |

```
SELECT last_name, job_id, salary
FROM   employees
WHERE  job_id = 'SA_REP'
OR     job_id = 'AD_PRES'
AND    salary > 15000;
```

1

| | LAST_NAME | JOB_ID | SALARY |
|---|---|---|---|
| 1 | King | AD_PRES | 24000 |
| 2 | Abel | SA_REP | 11000 |
| 3 | Taylor | SA_REP | 8600 |
| 4 | Grant | SA_REP | 7000 |

```
SELECT last_name, job_id, salary
FROM   employees
WHERE  (job_id = 'SA_REP'
OR     job_id = 'AD_PRES')
AND    salary > 15000;
```

2

| | LAST_NAME | JOB_ID | SALARY |
|---|---|---|---|
| 1 | King | AD_PRES | 24000 |

## Using the ORDER BY Clause

- Sort retrieved rows with the ORDER BY clause:
  - ASC: Ascending order, default
  - DESC: Descending order
- The ORDER BY clause comes last in the SELECT statement:

```
SELECT    last_name, job_id, department_id, hire_date
FROM      employees
ORDER BY hire_date ;
```

| | LAST_NAME | JOB_ID | DEPARTMENT_ID | HIRE_DATE |
|---|---|---|---|---|
| 1 | King | AD_PRES | 90 | 17-JUN-87 |
| 2 | Whalen | AD_ASST | 10 | 17-SEP-87 |
| 3 | Kochhar | AD_VP | 90 | 21-SEP-89 |
| 4 | Hunold | IT_PROG | 60 | 03-JAN-90 |
| 5 | Ernst | IT_PROG | 60 | 21-MAY-91 |
| 6 | De Haan | AD_VP | 90 | 13-JAN-93 |

...

**SORTING IN DESCENDING ORDER**

```
SQL> SELECT    ename, job, deptno, hiredate
  2  FROM      emp
  3  ORDER BY hiredate DESC;
```

**SORTING BY COLUMN ALIAS**

```
 SELECT    empno, ename, sal*12 annsal
FROM      emp
ORDER BY annsal;
```

**SORTING MULTIPLE COLUMNS**

```
 SELECT    ename, deptno, sal
FROM      emp
ORDER BY  deptno, sal DESC;
```

**Lab Task**

**Consider the database given and perform following tasks**

Q1. Write a Single Query to display local name and Gross National Product for Countries who specifically are in range of 372.00,4834,11705,60,650) Or those Countries who's Local Name starts off with B

Q2. Write an SQL Query to display Population, Expected Population after twenty years and Red Zone Areas with Population Exceeding 250,000.Also display information in descending order

Q3 Select Name, Capital, Region and Independence Year from Country Table Where Year Must be Equal to ‘1991’ and The Name Must be Start with ‘A’ and Capital Must be Less Than 500.

Q4 Select Surface Area and Population from Country Table Where Population Must be Equal to 500000 and Surface Area should Between 1000(lower) And 5000(Higher)

Q5.Write a single query which uses majority of concepts taught in lab 2 and lab 03

# LAB 4 : Using DDL Statements to Create and Manage Tables

**Objectives**

- Categorize the main database objects
- Review the table structure
- List the data types that are available for columns
- Create a simple table
- Explain how constraints are created at the time of table creation
- Describe how schema objects work

**Theoretical description**

## Database Objects

| Object | Description |
|---|---|
| Table | Basic unit of storage; composed of rows |
| View | Logically represents subsets of data from one or more tables |
| Sequence | Generates numeric values |
| Index | Improves the performance of some queries |
| Synonym | Gives alternative name to an object |

### Creation of Database:

To create a database, following MySQL query is used. In this query, a database is created named library.

**Query:**

mysql> create database hospital;

```
mysql> create database hospital;
Query OK, 1 row affected (0.13 sec)
```

## Naming Rules

Table names and column names:

- Must begin with a letter
- Must be 1–30 characters long
- Must contain only A–Z, a–z, 0–9, _, $, and #
- Must not duplicate the name of another object owned by the same user

## CREATE TABLE Statement

- You must have:
  - CREATE TABLE privilege
  - A storage area

```
CREATE TABLE [schema.]table
        (column datatype [DEFAULT expr][, ...]);
```

- You specify:
  - Table name
  - Column name, column data type, and column size

## Creating Tables

- Create the table:

```
CREATE TABLE dept
        (deptno      NUMBER(2),
         dname       VARCHAR2(14),
         loc         VARCHAR2(13),
         create_date DATE DEFAULT SYSDATE);
```

```
CREATE TABLE succeeded.
```

- Confirm table creation:

```
DESCRIBE dept
```

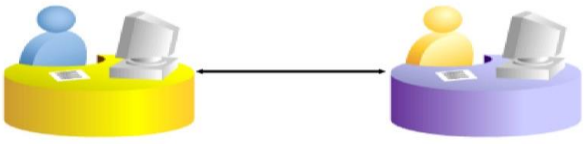
```
DESCRIBE dept
Name                          Null     Type
----------------------------- -------- -------------
DEPTNO                                 NUMBER(2)
DNAME                                  VARCHAR2(14)
LOC                                    VARCHAR2(13)
CREATE_DATE                            DATE
```

## Data Types

| Data Type | Description |
|---|---|
| VARCHAR2(*size*) | Variable-length character data |
| CHAR(*size*) | Fixed-length character data |
| NUMBER(*p*,*s*) | Variable-length numeric data |
| DATE | Date and time values |

## Referencing Another User's Tables

- Tables belonging to other users are not in the user's schema.
- You should use the owner's name as a prefix to those tables.




```
SELECT *
FROM userB.employees;
```

```
SELECT *
FROM userA.employees;
```

**Insert Command:**

The INSERT INTO statement of SQL is used to insert a new row in a table. To insert records into a table there are two ways.

- Enter the key words insert into followed by the table name, followed by an open parenthesis, followed by a list of column names separated by commas, followed by a closing parenthesis, followed by the keyword values, followed by the list of values enclosed in parenthesis. Data of only mentioned columns has to be entered and in order that is mentioned.
  - **Insert into tablename (column1, column2) values ('value1' ,'value2');**
- Enter the key words insert into followed by the table name, followed by the keyword values, followed by the list of values enclosed in parenthesis. Data for all the columns of the table.
  - **Insert into tablename values ('value1','value2');**

### Insert Data Only in Specified Columns

```
INSERT INTO Customers (CustomerName, City, Country)
VALUES ('Cardinal', 'Stavanger', 'Norway');
```

| CustomerID | CustomerName | ContactName | Address | City | PostalCode | Country |
|---|---|---|---|---|---|---|
| 89 | White Clover Markets | Karl Jablonski | 305 - 14th Ave. S. Suite 3B | Seattle | 98128 | USA |
| 90 | Wilman Kala | Matti Karttunen | Keskuskatu 45 | Helsinki | 21240 | Finland |
| 91 | Wolski | Zbyszek | ul. Filtrowa 68 | Walla | 01-012 | Poland |

**INSERT INTO Persons VALUES (4,'Nilsen', 'Johan', 'Bakken 2', 'Stavanger')**

| P_Id | LastName | FirstName | Address | City |
|---|---|---|---|---|
| 1 | Hansen | Ola | Timoteivn 10 | Sandnes |
| 2 | Svendson | Tove | Borgvn 23 | Sandnes |
| 3 | Pettersen | Kari | Storgt 20 | Stavanger |

**Lab Task**

Q1 Create tables of your project

Q2. Insert data that is 25 to 30 records in all tables in your project

# LAB 5 : Creating other schema objects like table level and column level constraints

## Objectives

- Understand table level and column level constraints and their working

## Theoretical Description

### SQL Constraints

- Constraints are used to limit the type of data that can go into a table.
- MySQL constraints are statements that can be applied at the column level or table level to specify rules for the data that can be entered into a column or data table

| SQL Constraint | Function |
|---|---|
| NOT NULL | It ensures that a column does not accept NULL values. |
| CHECK | It ensures that a column accepts values within the specified range of values. |
| UNIQUE | It ensures that a column does not accept duplicate values. |
| PRIMARY KEY | It uniquely identifies a row in the table. It is a combination of NOT NULL and UNIQUE constraints. |
| FOREIGN KEY | It is like a primary key constraint only. But it uniquely identifies a row in another table. |

**Defining Constraints**

- Example of a column-level constraint:

```
CREATE TABLE employees(
  employee_id   NUMBER(6)
    CONSTRAINT emp_emp_id_pk PRIMARY KEY,
  first_name    VARCHAR2(20),
  ...);
```

(1)

- Example of a table-level constraint:

```
CREATE TABLE employees(
  employee_id   NUMBER(6),
  first_name    VARCHAR2(20),
  ...
  job_id        VARCHAR2(10) NOT NULL,
  CONSTRAINT emp_emp_id_pk
    PRIMARY KEY (EMPLOYEE_ID));
```

(2)

### NOT NULL CONSTRAINT

- ❑ NOT NULL constraint is applied to a column, it ensures that the column will not accept NULL values.
- ❑ A null value means that a particular field has been left empty, and values such as zero or blank space do not come under Null values
- ❑ primary key column implicitly includes a not null constraint

| EMPLOYEE_ID | FIRST_NAME | LAST_NAME | EMAIL | HIRE_DATE | JOB_ID | COMMISSION_PCT |
|---|---|---|---|---|---|---|
| 100 | Steven | King | SKING | 17-JUN-87 | AD_PRES | (null) |
| 101 | Neena | Kochhar | NKOCHHAR | 21-SEP-89 | AD_VP | (null) |
| 102 | Lex | De Haan | LDEHAAN | 13-JAN-93 | AD_VP | (null) |
| 103 | Alexander | Hunold | AHUNOLD | 03-JAN-90 | IT_PROG | (null) |
| 104 | Bruce | Ernst | BERNST | 21-MAY-91 | IT_PROG | (null) |
| 107 | Diana | Lorentz | DLORENTZ | 07-FEB-99 | IT_PROG | (null) |
| 124 | Kevin | Mourgos | KMOURGOS | 16-NOV-99 | ST_MAN | (null) |
| 141 | Trenna | Rajs | TRAJS | 17-OCT-95 | ST_CLERK | (null) |
| 142 | Curtis | Davies | CDAVIES | 29-JAN-97 | ST_CLERK | (null) |
| 143 | Randall | Matos | RMATOS | 15-MAR-98 | ST_CLERK | (null) |
| 144 | Peter | Vargas | PVARGAS | 09-JUL-98 | ST_CLERK | (null) |
| 149 | Eleni | Zlotkey | EZLOTKEY | 29-JAN-00 | SA_MAN | 0.2 |
| 174 | Ellen | Abel | EABEL | 11-MAY-96 | SA_REP | 0.3 |

...

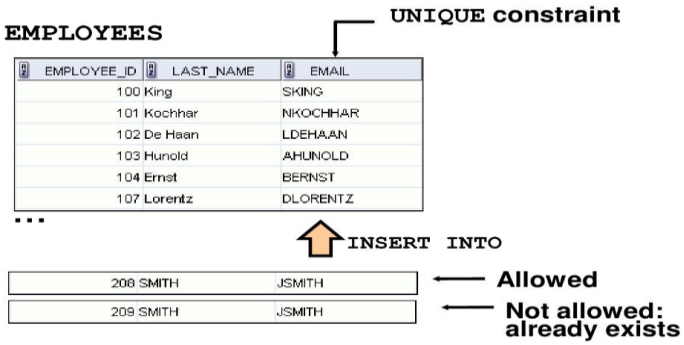
NOT NULL constraint (Primary Key enforces NOT NULL constraint.)

NOT NULL constraint

Absence of NOT NULL constraint (Any row can contain a null value for this column.)

## UNIQUE KEY CONSTRAINT

- A unique key is a constraint in SQL which helps in uniquely identifying a record in the data table.
- It is similar to the Primary key as both of them guarantees the uniqueness of a record.
- Unlike primary key, a unique key can accept NULL values, a unique Key can include a NULL value but only one NULL value per column is allowed.

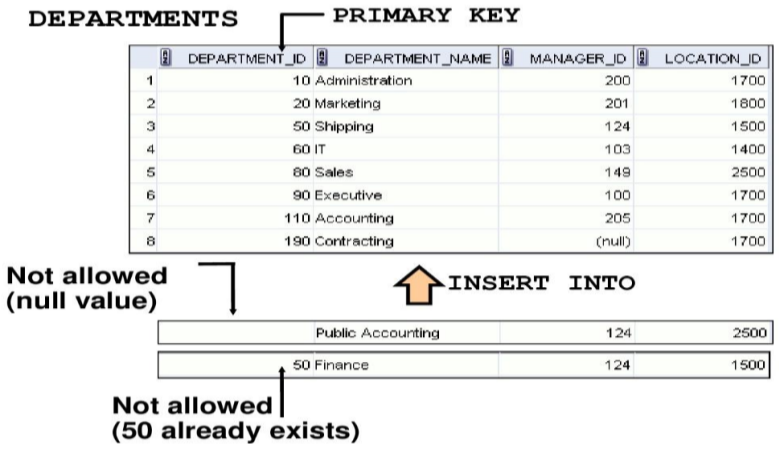



## EXAMPLE OF UNIQUE CONSTRAINT TABLE LEVEL AND COLUMN LEVEL

```
CREATE TABLE suppliers (
    supplier_id INT AUTO_INCREMENT,
    name VARCHAR(255) NOT NULL,
    phone VARCHAR(15) NOT NULL UNIQUE,
    address VARCHAR(255) NOT NULL,
    PRIMARY KEY (supplier_id),
    CONSTRAINT uc_name_address UNIQUE (name , address)
);
```

## PRIMARY KEY CONSTRAINT

- Each table must normally contain a column or set of columns that uniquely identifies rows of data that are stored in the table. This column or set of columns is referred to as the primary key.




```sql
-- Column level Primary key
CREATE TABLE Employee
 (
   ID INT (PRIMARY KEY

   NAME VARCHAR (50),

   EMAIL VARCHAR(60)

 )
```

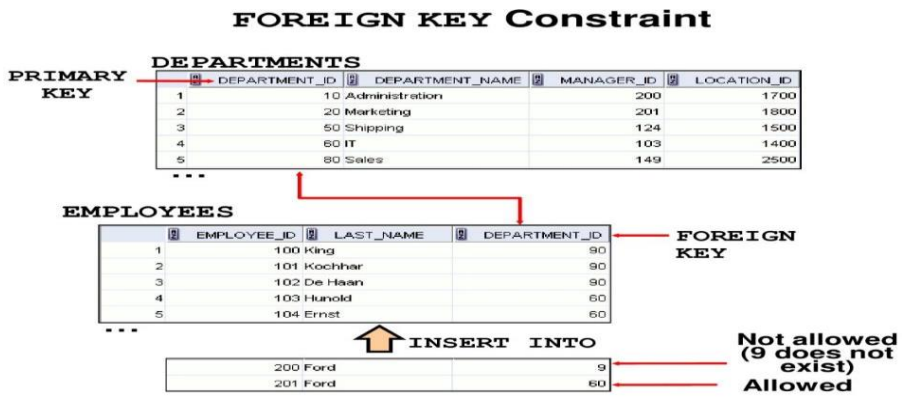
```sql
-- Table level Primary key
 CREATE TABLE Employee
  (
   ID INT NOT NULL,

   NAME VARCHAR (50),

   EMAIL VARCHAR(60)

   CONSTRAINT PK_ID PRIMARY KEY(ID)

  )
```




```sql
CREATE TABLE categories(
    categoryId INT AUTO_INCREMENT PRIMARY KEY,
    categoryName VARCHAR(100) NOT NULL
) ENGINE=INNODB;

CREATE TABLE products(
    productId INT AUTO_INCREMENT PRIMARY KEY,
    productName varchar(100) not null,
    categoryId INT,
    CONSTRAINT fk_category
    FOREIGN KEY (categoryId)
        REFERENCES categories(categoryId)
) ENGINE=INNODB;
```

## CHECK CONSTRAINT COLUMN LEVEL AND TABLE LEVEL

```
CREATE TABLE parts (
    part_no VARCHAR(18) PRIMARY KEY,
    description VARCHAR(40),
    cost DECIMAL(10,2 ) NOT NULL CHECK (cost >= 0),
    price DECIMAL(10,2) NOT NULL CHECK (price >= 0),
    CONSTRAINT parts_chk_price_gt_cost
        CHECK(price >= cost)
);
```

**Lab Task**

**Create a database named as Data definition Lab and create following table**

| Table name | Attributes | Constraints |
|---|---|---|
| Customer | Customer_ID<br>Customer_Name<br>Customer_Address<br>Customer_State | Customer_ID is a primary key<br>Customer_Name cannot be null and its size is 35 chracters<br>The size of customer_address is variable and is of 35 characters as well as it can be null<br>Customer_dtate is fixed two characters and cannot be Null |
| Sales_Order | Order_Id<br>Order_Date<br>Customer_ID | Order_Id is a primary key<br>For order_Date use date datatype<br>Customer Id is a foreign key |
| Products | Product_ID<br>Product_Name<br>Product_Line_ID | Product_ID is a primary key<br>Product_name is unique and contains 40 chracters<br>Product_line_id is always greater than or equal to 10 and less than or equal to 100 so use constraint to check this condition |
| Order_line | Order_ID<br>Product_ID | (order_id and product_id ) are primary keys<br>Order_id is a foreign key<br>Product_id is a foreign key and cannot be null |

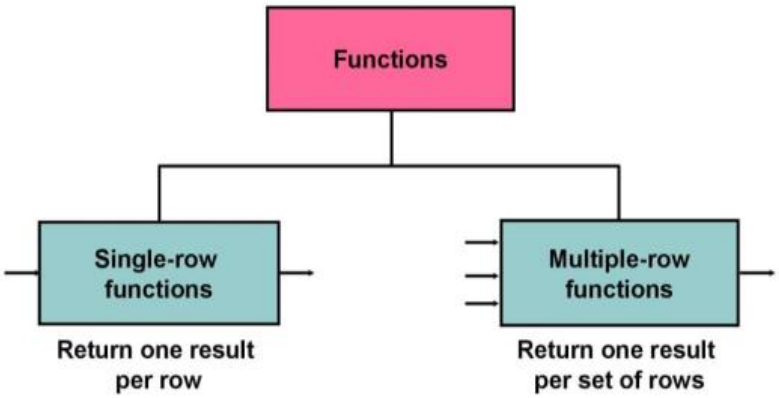
**Q2.** Insert 15 to 20 records in the above mentioned tables

# LAB 6: Using Single-Row Character Functions to Customize Output

**Objectives**

To describe various types of functions available in SQL
To use character, number, and date functions in SELECT statements

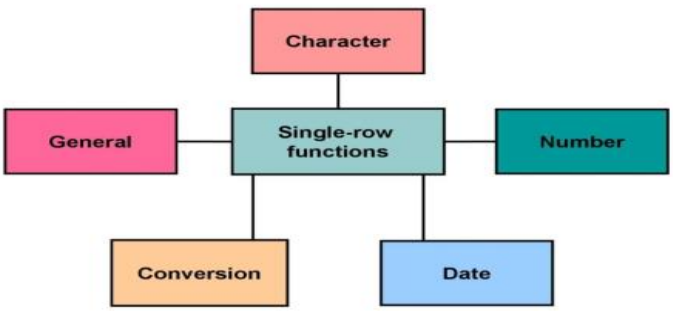
Two Types of SQL Functions



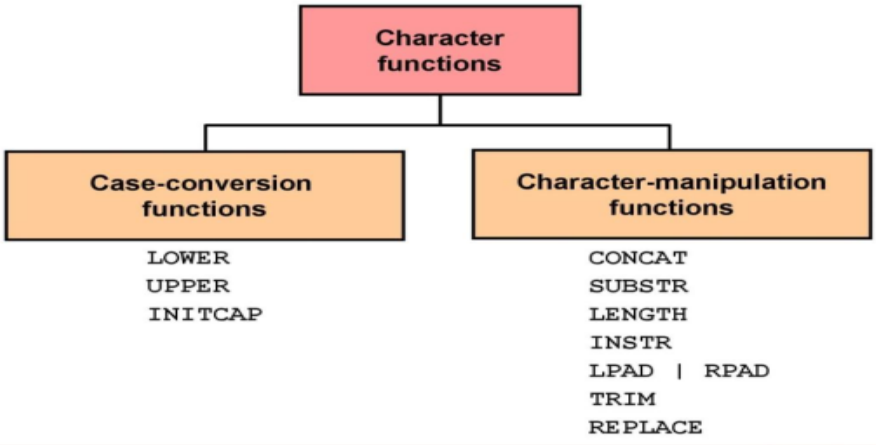

**Theoretical description**

Single row functions are the one who work on single row and return one output per row. For example, length and case conversion functions are single row functions. Single row functions can be character functions, numeric functions, date functions, and conversion functions. These functions require one or more input arguments and operate on each row, thereby returning one output value for each row.Single row functions can be used in SELECT statement, WHERE and ORDER BY clause.

Single-Row Functions



## Character Functions




## Case-Conversion Functions

These functions convert the case for character strings:

| Function | Result |
|---|---|
| LOWER('SQL Course') | sql course |
| UPPER('SQL Course') | SQL COURSE |

**USING CASE CONVERSION FUNCTIONS**

```
select upper(continent),lower(Name) from country;
```

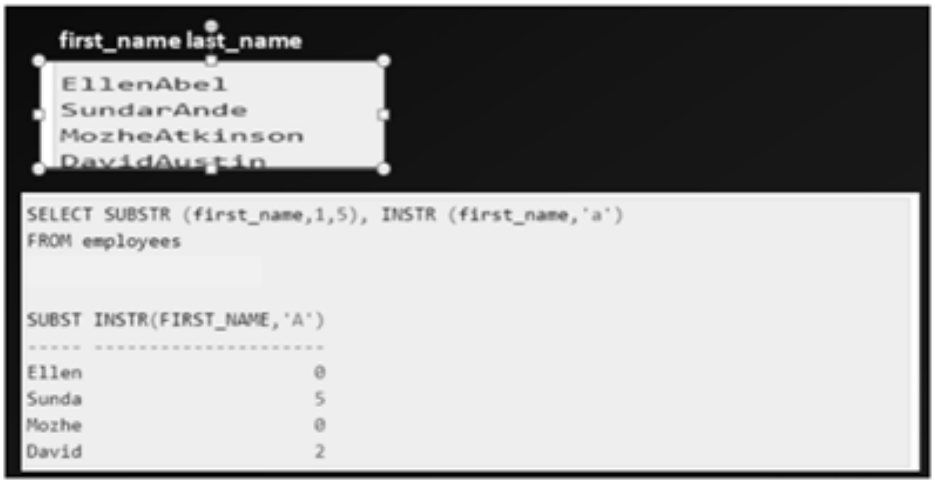
| upper(continent) | lower(Name) |
|---|---|
| NORTH AMERICA | aruba |
| ASIA | afghanistan |
| AFRICA | angola |

**CHRACTER MANUPULATION FUNCTIONS**

| Function | description | Result |
|---|---|---|
| **CONCAT** | Joins values together | CONCAT('Hello,World')<br>**Hello world** |
| **SUBSTR** | Extracts a string of determined length | SUBSTR('HelloWorld',1,5)<br>**Hello** |
| **LENGTH** | Shows length of string as numeric value | LENGTH('HelloWorld')<br>**10** |
| **INSTR** | Finds numeric position of named character | INSTR('HelloWorld','W')<br>**6** |
| **LPAD** | Pads the character value left justified | LPAD(salary,10,'*')<br>*****24000** |
| **RPAD** | Pads the character value right justified | RPAD(salary,10,'*')<br>**24000***** |
| **TRIM** | Trims heading or trailing characters or both from a character string | Select Trim(TRAILING 'trailing'FROM 'text trailing');<br>**text**<br>Select Trim(BOTH 'Leadtrail'FROM 'leadtrailtextlead trail'); |
| **REPLACE** | Replace the characters in given string | Select REPLACE('w3resource','ur','r');<br>**w3resorce** |

**Example of character manipulation functions**

1. **Substring function and instring function**



## 2. Replace function

Sample table: publisher

| pub_id | pub_name | pub_city | country | country_office | no_of_branch | estd |
|---|---|---|---|---|---|---|
| P001 | Jex Max Publication | New York | USA | New York | 15 | 1969-12-25 |
| P002 | BPP Publication | Mumbai | India | New Delhi | 10 | 1985-10-01 |
| P003 | New Harrold Publication | Adelaide | Australia | Sydney | 6 | 1975-09-05 |
| P004 | Ultra Press Inc. | London | UK | London | 8 | 1948-07-10 |
| P005 | Mountain Publication | Houstan | USA | Sun Diego | 25 | 1975-01-01 |
| P006 | Summer Night Publication | New York | USA | Atlanta | 10 | 1990-12-10 |
| P007 | Pieterson Grp. of Publishers | Cambridge | UK | London | 6 | 1950-07-15 |
| P008 | Novel Publisher Ltd. | New Delhi | India | Bangalore | 10 | 2000-01-01 |

Sample Output:

```
mysql> SELECT pub_city,country,
    -> REPLACE(country,'K','SA')
    -> FROM publisher
    -> WHERE country='UK';
+-----------+---------+--------------------------+
| pub_city  | country | REPLACE(country,'K','SA') |
+-----------+---------+--------------------------+
| London    | UK      | USA                      |
| Cambridge | UK      | USA                      |
+-----------+---------+--------------------------+
2 rows in set (0.05 sec)
```

## 3. RPAD

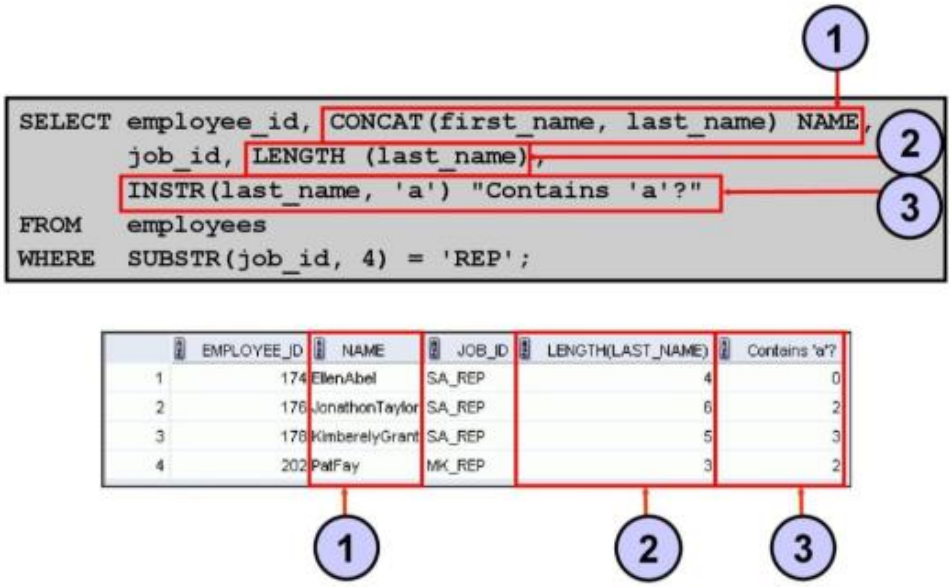
Sample table: author

| aut_id | aut_name | country | home_city |
|---|---|---|---|
| AUT001 | William Norton | UK | Cambridge |
| AUT002 | William Maugham | Canada | Toronto |
| AUT003 | William Anthony | UK | Leeds |
| AUT004 | S.B.Swaminathan | India | Bangalore |
| AUT005 | Thomas Morgan | Germany | Arnsberg |
| AUT006 | Thomas Merton | USA | New York |
| AUT007 | Piers Gibson | UK | London |
| AUT008 | Nikolai Dewey | USA | Atlanta |

Sample Output:

```
mysql> SELECT aut_name,RPAD(aut_name,25,'*')
    -> FROM author
    -> WHERE country='UK';
+-----------------+---------------------------+
| aut_name        | RPAD(aut_name,25,'*')     |
+-----------------+---------------------------+
| William Norton  | William Norton********** |
| William Anthony | William Anthony********** |
| Piers Gibson    | Piers Gibson************ |
| C. J. Wilde     | C. J. Wilde************** |
+-----------------+---------------------------+
```

## Using the Character-Manipulation Functions



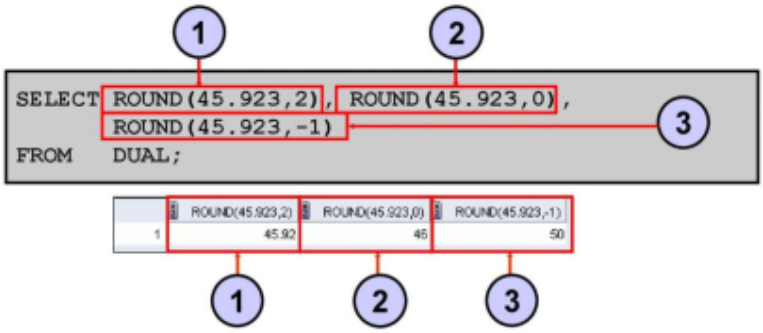

| | EMPLOYEE_ID | NAME | JOB_ID | LENGTH(LAST_NAME) | Contains 'a'? |
|---|---|---|---|---|---|
| 1 | 174 | EllenAbel | SA_REP | 4 | 0 |
| 2 | 176 | JonathonTaylor | SA_REP | 6 | 2 |
| 3 | 178 | KimberelyGrant | SA_REP | 5 | 3 |
| 4 | 202 | PatFay | MK_REP | 3 | 2 |

## Number Functions

- ROUND: Rounds value to a specified decimal
- TRUNC: Truncates value to a specified decimal
- MOD: Returns remainder of division

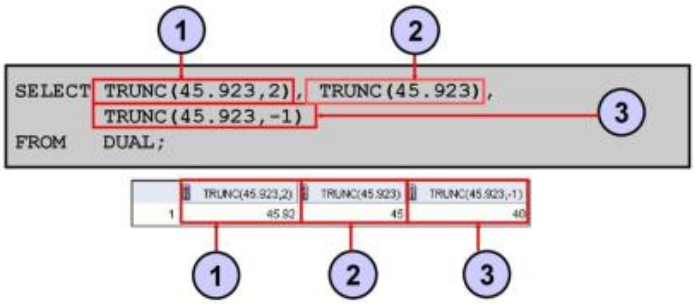
| Function | Result |
|---|---|
| ROUND(45.926, 2) | 45.93 |
| TRUNC(45.926, 2) | 45.92 |
| MOD(1600, 300) | 100 |

## Using the ROUND Function




DUAL is a dummy table that you can use to view results from functions and calculations.

## Using the TRUNC Function



## Using the MOD Function

For all employees with the job title of Sales Representative, calculate the remainder of the salary after it is divided by 5,000.

```
SELECT last_name, salary, MOD(salary, 5000)
FROM   employees
WHERE  job_id = 'SA_REP';
```

| | LAST_NAME | SALARY | MOD(SALARY,5000) |
|---|---|---|---|
| 1 | Abel | 11000 | 1000 |
| 2 | Taylor | 8600 | 3600 |
| 3 | Grant | 7000 | 2000 |

**Lab Task**

Consider database given to you and answer following questions

Q1.Write a single sql query to display employee first name, last name and job joined together, similarly display length of employee email address, and the numeric position of letter A in the employee last name, for all employees whose employee number is 1002 and job is president

Q2. Write a query to display the last names of all customers who have the letter p in their first name and the letter A at the 5th position in their last name.

Q3. Write a single query which covers all character manipulation functions such as substring ,instring, concat, upper case, trim,replace and lpad or rpad and should use where clause as well as comparison operators

Q4.Format the credit limit column to be 15 character long ,left padded with Rs of only those customers whose last name is king and their credit limit is greater than 10000 .Also label column as credit balance

Q5. Consider order table and round off the price of only those orders whose order number is 10100

# LAB 7: Using Single-Row Date Functions to Customize Output & Type conversion

**Objectives**

- To describe various types of functions available in SQL
- To use character, number, and date functions in SELECT statements

**Theoretical description**

**Date format functions**

This function allows the developer to format the date anyway that they wish by specifying a sequence of format strings. A string is composed of the percentage symbol '%' followed by a letter that signifies how we wish to display the date. These are some of the more common strings to use:

***Syntax:***

**DATE_FORMAT(date, sequence)**

**Date format functions**

| String | Displays | Example |
|---|---|---|
| %d | The numeric day of the month | 01....10....17....24 etc |
| %D | The day of the month with a suffix | 1st, 2nd, 3rd.... etc |
| %m | The numeric month | 01....04....08....11 etc |
| %M | The Month name | January....April....August etc |
| %b | The Abbreviated Month Name | Jan....Apr....Aug....Nov etc |
| %y | Two digit year | 98, 99, 00, 01, 02, 03 etc |
| %Y | Four digit year | 1998, 2000, 2002, 2003 etc |
| %W | Weekday name | Monday.... Wednesday....Friday etc |
| %a | Abbreviated Weekday name | Mon....Wed....Fri etc |
| %H | Hour (24 hour clock) | 07....11....16....23 etc |
| %h | Hour (12 hour clock) | 07....11....04....11 etc |
| %p | AM or PM | AM....PM |
| %i | Minutes | 01....16....36....49 etc |
| %s | Seconds | 01....16....36....49 etc |
| % c | Month | Numeric (0…12) |
| % e | Day of month | Numeric (0….31) |
| %f | Microseconds | |
| %r | Time 12 hour | hh:mm:ss followed by AM or PM |

**EXAMPLE**

```
+------------+------------+----------------+------------+------------+---------+---------------------------------+
| invoice_no | invoice_dt | ord_no         | ord_date   | receive_dt | book_id | book_name                       |
+------------+------------+----------------+------------+------------+---------+---------------------------------+
| INV0001    | 2008-07-15 | ORD/08-09/0001 | 2008-07-06 | 2008-07-19 | BK001   | Introduction to Electrodynamics |
| INV0002    | 2008-08-25 | ORD/08-09/0002 | 2008-08-09 | 2008-08-28 | BK004   | Transfer  of Heat and Mass      |
| INV0003    | 2008-09-20 | ORD/08-09/0003 | 2008-09-15 | 2008-09-23 | BK005   | Conceptual Physics              |
| INV0004    | 2007-08-30 | ORD/07-08/0005 | 2007-08-22 | 2007-08-30 | BK004   | Transfer  of Heat and Mass      |
| INV0005    | 2007-07-28 | ORD/07-08/0004 | 2007-06-25 | 2007-07-30 | BK001   | Introduction to Electrodynamics |
| INV0006    | 2007-09-24 | ORD/07-08/0007 | 2007-09-20 | 2007-09-30 | BK003   | Guide to Networking             |
```

Code:

```
1  SELECT invoice_no,ord_date, DATE_FORMAT(ord_date,'%W %D %M %Y')
2  FROM purchase;
```

Sample Output:

```
mysql> SELECT invoice_no,ord_date,DATE_FORMAT(ord_date,'%W %D %M %Y')
    ->       FROM purchase;
+------------+------------+-------------------------------------+
| invoice_no | ord_date   | DATE_FORMAT(ord_date,'%W %D %M %Y') |
+------------+------------+-------------------------------------+
| INV0001    | 2008-07-06 | Sunday 6th July 2008                |
| INV0002    | 2008-08-09 | Saturday 9th August 2008            |
| INV0003    | 2008-09-15 | Monday 15th September 2008          |
| INV0004    | 2007-08-22 | Wednesday 22nd August 2007          |
| INV0005    | 2007-06-25 | Monday 25th June 2007               |
| INV0006    | 2007-09-20 | Thursday 20th September 2007        |
+------------+------------+-------------------------------------+
6 rows in set (0.00 sec)
```

**DATE EXTRACTION FUNCTIONS**

As well as using DATE_FORMAT() there are other functions that allow we to extract specific information about a date (year, month, day etc). These include:

| Function | Displays | Example |
|---|---|---|
| DAYOFMONTH(date) | The numeric day of the month | 01....10....17....24 etc |
| DAYNAME(date) | The Name of the day | Monday.... Wednesday.... |
| MONTH(date) | The numeric month | 01....04....08....11 etc |
| MONTHNAME(date) | The Month name | January....April....August etc |
| YEAR(date) | Four digit year | 1998, 2000, 2002, 2003 etc |
| HOUR(time) | Hour (24 hour clock) | 07....11....16....23 etc |
| MINUTE(time) | Minutes | 01....16....36....49 etc |
| SECOND(time) | Seconds | 01....16....36....49 etc |
| DAYOFYEAR(date) | Numeric day of the year | 1.....366 |

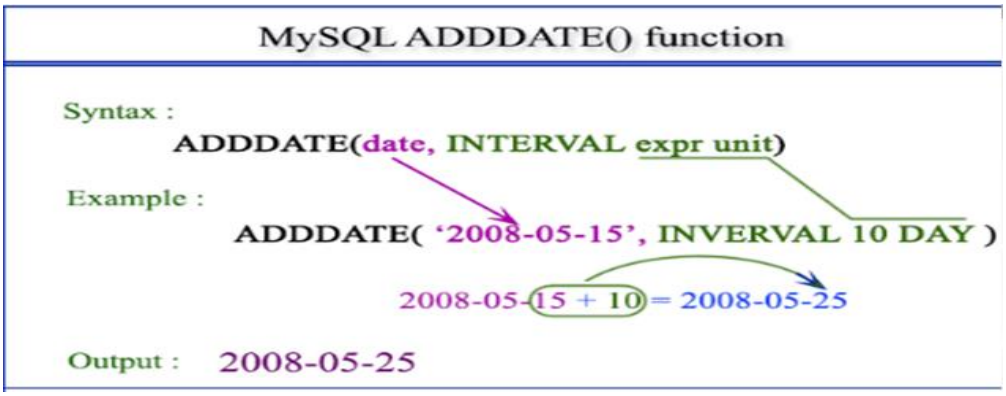
**EXAMPLE 1 DAY NAME**

```
mysql> SELECT DAYNAME('2008-05-15');
+-----------------------+
| DAYNAME('2008-05-15') |
+-----------------------+
| Thursday              |
+-----------------------+
```

```
select requiredDate, dayname(requiredDate) from orders;
```

| requiredDate | dayname(requiredDate) |
|---|---|
| 2003-01-13 | Monday |
| 2003-01-18 | Saturday |
| 2003-01-18 | Saturday |
| 2003-02-07 | Friday |
| 2003-02-09 | Sunday |
| 2003-02-21 | Friday |
| 2003-02-24 | Monday |
| 2003-03-03 | Monday |
| 2003-03-12 | Wednesday |
| 2003-03-19 | Wednesday |
| 2003-03-24 | Monday |

**EXAMPLE 2**
**CHANGING DATE VALUES**




```
select requiredDate, date_add(requiredDate,interval 1 month) from
orders;
```

**Lab Task**

Q1.Consider order table and round off the price of only those orders whose order number is 10100

Q2.Consider three major scenarios on classic model database in which major data format and time functions are covered?

Q3. Explain following functions and also explain how sql queries are written for them

- Time difference()
- Time stamp()
- Sysdate()

# LAB 8: Constructing ERD using VISIO or draw.io

**Objectives**

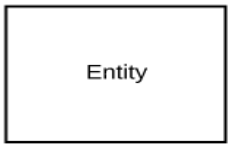
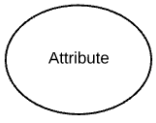
- To learn about Entity Relationship Diagram (ERD).
- To learn about the notations used in tools.
- We will make an ER Diagram using draw.io

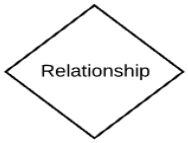
**Theoretical description**

**Entity Relationship Diagram:**

An entity–relationship model describes interrelated things of interest in a specific domain of knowledge. A basic ER model is composed of entity types and specifies relationships that can exist between entities. ER Diagrams are most often used to design or debug relational databases in the fields of software engineering, business information systems, education and research.

**Notations Used in ERD**:

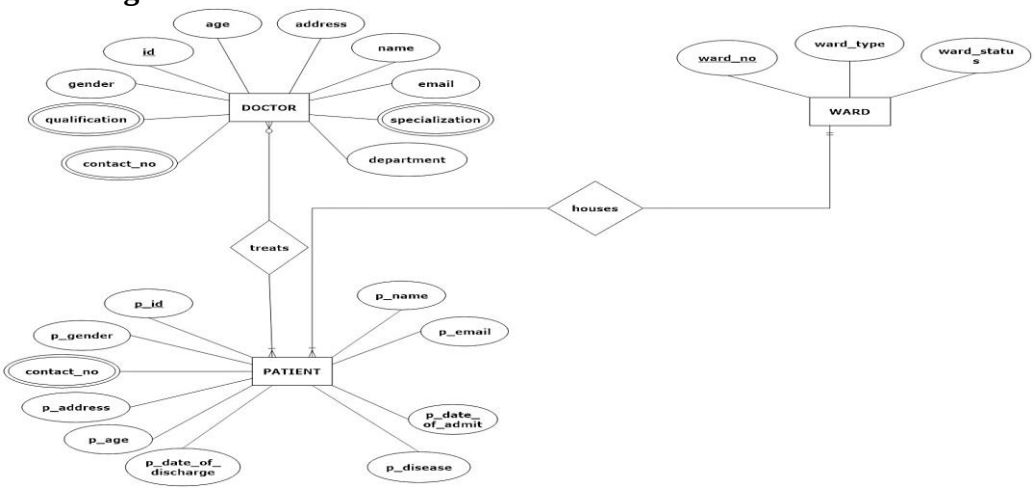
| | |
|---|---|
| 1.   Entity(strong) | These shapes are independent from other entities, and are often called parent entities, since they will often have weak entities that depend on them. They will also have a primary key, distinguishing each occurrence of the entity. |
| 2.   Entity(Weak) | Weak entities depend on some other entity type. They don't have primary keys, and have no meaning in the diagram without their parent entity. |
| 3.   Associative entity | Associative entities relate the instances of several entity types. They also contain attributes specific to the relationship between those entity instances. |
| 4.   Attributes | Attributes are characteristics of an entity, a many-to-many relationship, or a one-to-one relationship. |

| | |
|---|---|
| 5.   Relationship | Relationships are associations between or among entities. |

**SmartDraw:**

SmartDraw is a diagram tool used to make flowcharts, organization charts, mind maps, project charts, and other business visuals. SmartDraw allows you to draw and print architectural and engineering diagrams to scale. SmartDraw even provides an AutoCAD-like annotation layer that automatically resizes to match a diagram.

**ERD Diagram**:




**Lab Task**

Q1 Create enhanced er diagram using crow feet notation chenn notation for following

An MDXCS computer supply is a wholesale company which buys computers from suppliers and sells them to customers. MDXCS has a number of suppliers and each product is supplied by on supplier only. However, suppliers may supply more than one product. MDXCS also keeps record of suppliers who are not currently supplying any products. The company has divided their operation into different geographical areas. Each storehouse is responsible for each geographical area to deliver products to customers. Customers may be located in more than one place and every storehouse has at least one customer. There are a number of storehouses which hold stocks of products and these storehouses are assigned to deliver orders to customers. Suppliers supply product to a particular storehouse and then MDXCS stock in other depots. When a customer places an order, the company gives them a unique order reference number. One order can be for different products. Customer can pay by card or cash and orders are marked as delivered by entering a delivery date. Note: Specify all the entities, relationships and their cardinalities

## LAB 9: Constructing enhanced ERD using visio or draw.io

**Objectives**

- Identify enhanced erd notations
- Identify how to construct enhanced ERD

**Lab Task**

**Q1. Create an enhanced entity relationship diagram for following .**

Ali as a customer wants to purchase plot/plots in a housing scheme for investment. He can buy plots of different sizes and categories like commercial, residential and farm house along Marla details. On commercial plots tax payer ID details must be mentioned. System should also keep the record of Down Payment and quarterly installments. Regional office information with respect to cities should be kept. If regional office is removed then agent information should be removed. Notifications regarding new plots/schemes should also be shared with the registered and unregistered users.

**Q2. Consider Learning management system and list down business rules and create an enhanced entity relationship diagram for it**

Note: Apply all Enhanced ERD concepts

# LAB 10: Aggregating data using SQL Aggregate functions and group functions

## Objectives

- Identify aggregate functions
- Identify the available group functions
- Describe the use of group functions
- Group data by using the GROUP BY clause
- Include or exclude grouped rows by using the HAVING clause

**Theoretical description**

**Group functions**

- They give the user the ability to answer business questions such as:
  - What is the average salary of an employee in the company?
  - What were the total salaries for a particular year?
  - What are the maximum and minimum salaries in the Computer Department?
- They perform a variety of actions such as counting all the rows in a table, averaging a column's data, and summing numeric data.
- They can also search a table to find the highest "MAX" or lowest "MIN" values in a column.

**Example**

**Types of Group Functions**
**Using the AVG and SUM Functions**

You can use AVG and SUM for numeric data.

```
SELECT  AVG(salary), MAX(salary),
        MIN(salary), SUM(salary)
FROM    employees
WHERE   job_id LIKE '%REP%';
```

| | AVG(SALARY) | MAX(SALARY) | MIN(SALARY) | SUM(SALARY) |
|---|---|---|---|---|
| 1 | 8150 | 11000 | 6000 | 32600 |

- List of aggregate functions including their syntax and use.

| Function Syntax | Function Use |
|---|---|
| SUM( [ALL \| DISTINCT] expression ) | The total of the (distinct) values in a numeric column/expression. |
| AVG( [ALL \| DISTINCT] expression ) | The average of the (distinct) values in a numeric column/expression. |
| COUNT( [ALL \| DISTINCT] expression ) | The number of (distinct) non-NULL values in a column/expression. |
| COUNT(*) | The number of selected rows. |
| MAX(expression) | The highest value in a column/expression. |
| MIN(expression) | The lowest value in a column/expression. |

**U**
**sing**
**Mini**

**mum and Max Functions**

```
SELECT MIN(emp_last_name), MAX(emp_last_name)
FROM employee;

------------------------ ------------------------
Amin                     Zhu
```

You can use `MIN` and `MAX` for numeric, character, and date data types.

```
SELECT MIN(hire_date), MAX(hire_date)
FROM   employees;
```

| | MIN(HIRE_DATE) | MAX(HIRE_DATE) |
|---|---|---|
| 1 | 17-JUN-87 | 29-JAN-00 |

**Using count function**

```
SELECT COUNT(emp_superssn) "Number of Supervised
Employees"
FROM employee;

Number of Supervised Employees
------------------------------
7
```

In contrast the count(*) will count each row regardless of NULL values.

```
SELECT COUNT(*) "Number of Employees"
FROM employee;
Number of Employees
-------------------
8
```

## Using the DISTINCT Keyword

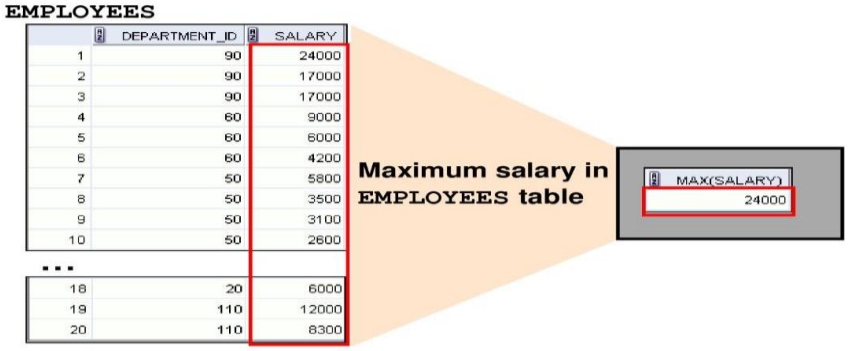
- `COUNT(DISTINCT expr)` returns the number of distinct non-null values of *expr*.
- To display the number of distinct department values in the `EMPLOYEES` table:

```
SELECT COUNT(DISTINCT department_id)
FROM   employees;
```

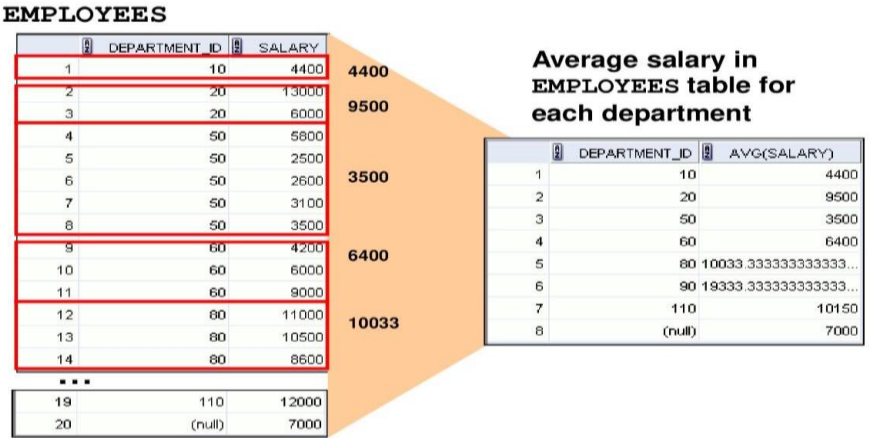
| | COUNT(DISTINCTDEPARTMENT_ID) |
|---|---|
| 1 | 7 |

## What Are Group Functions?

Group functions operate on sets of rows to give one result per group.




## Creating Groups of Data




## Creating Groups of Data: GROUP BY Clause Syntax

```
SELECT     column, group_function(column)
FROM       table
[WHERE     condition]
[GROUP BY  group_by_expression]
[ORDER BY  column];
```

You can divide rows in a table into smaller groups by using the GROUP BY clause.

**Example of group by clause**

```
SELECT    department_id, AVG(salary)
FROM      employees
GROUP BY department_id ;
```

| DEPARTMENT_ID | AVG(SALARY) |
|---|---|
| 10 | 4400 |
| 20 | 9500 |
| 50 | 3500 |
| 60 | 6400 |
| 80 | 10033.3333 |
| 90 | 19333.3333 |
| 110 | 10150 |
| | 7000 |

**Illegal queries using group by functions**

```
SELECT    department_id, AVG(salary)
FROM      employees
WHERE     AVG(salary) > 8000
GROUP BY department_id;
```

```
WHERE   AVG(salary) > 8000
        *
ERROR at line 3:
ORA-00934: group function is not allowed here
```

Cannot use the WHERE clause to restrict groups

## Grouping by More Than One Column

| DEPARTMENT_ID | JOB_ID | SALARY |
|---|---|---|
| 90 | AD_PRES | 24000 |
| 90 | AD_VP | 17000 |
| 90 | AD_VP | 17000 |
| 60 | IT_PROG | 9000 |
| 60 | IT_PROG | 6000 |
| 60 | IT_PROG | 4200 |
| 50 | ST_MAN | 5800 |
| 50 | ST_CLERK | 3500 |
| 50 | ST_CLERK | 3100 |
| 50 | ST_CLERK | 2600 |
| 50 | ST_CLERK | 2500 |
| 80 | SA_MAN | 10500 |
| 80 | SA_REP | 11000 |
| 80 | SA_REP | 8600 |
| 20 | MK_REP | 6000 |
| 110 | AC_MGR | 12000 |
| 110 | AC_ACCOUNT | 8300 |

20 rows selected.

"Add up the salaries in the EMPLOYEES table for each job, grouped by department.

| DEPARTMENT_ID | JOB_ID | SUM(SALARY) |
|---|---|---|
| 10 | AD_ASST | 4400 |
| 20 | MK_MAN | 13000 |
| 20 | MK_REP | 6000 |
| 50 | ST_CLERK | 11700 |
| 50 | ST_MAN | 5800 |
| 60 | IT_PROG | 19200 |
| 80 | SA_MAN | 10500 |
| 80 | SA_REP | 19600 |
| 90 | AD_PRES | 24000 |
| 90 | AD_VP | 34000 |
| 110 | AC_ACCOUNT | 8300 |
| 110 | AC_MGR | 12000 |
| | SA_REP | 7000 |

13 rows selected.

## Using the GROUP BY Clause on Multiple Columns

```
SELECT    department_id dept_id, job_id, SUM(salary)
FROM      employees
GROUP BY department_id, job_id
ORDER BY department_id;
```

| | DEPARTMENT_ID | JOB_ID | SUM(SALARY) |
|---|---|---|---|
| 1 | 10 | AD_ASST | 4400 |
| 2 | 20 | MK_MAN | 13000 |
| 3 | 20 | MK_REP | 6000 |
| 4 | 50 | ST_CLERK | 11700 |
| 5 | 50 | ST_MAN | 5800 |
| 6 | 60 | IT_PROG | 19200 |
| 7 | 80 | SA_MAN | 10500 |
| 8 | 80 | SA_REP | 19600 |
| 9 | 90 | AD_PRES | 24000 |
| 10 | 90 | AD_VP | 34000 |
| 11 | 110 | AC_ACCOUNT | 8300 |
| 12 | 110 | AC_MGR | 12000 |
| 13 | (null) | SA_REP | 7000 |

## Restricting Group Results

EMPLOYEES

| | DEPARTMENT_ID | SALARY |
|---|---|---|
| 1 | 10 | 4400 |
| 2 | 20 | 13000 |
| 3 | 20 | 6000 |
| 4 | 50 | 5800 |
| 5 | 50 | 2500 |
| 6 | 50 | 2600 |
| 7 | 50 | 3100 |
| 8 | 50 | 3500 |
| 9 | 60 | 4200 |
| 10 | 60 | 6000 |
| 11 | 60 | 9000 |
| 12 | 80 | 11000 |
| 13 | 80 | 10500 |
| 14 | 80 | 8600 |
| ... | | |
| 18 | 110 | 8300 |
| 19 | 110 | 12000 |
| 20 | (null) | 7000 |

**The maximum salary per department when it is greater than $10,000**

| | DEPARTMENT_ID | MAX(SALARY) |
|---|---|---|
| 1 | 20 | 13000 |
| 2 | 80 | 11000 |
| 3 | 90 | 24000 |
| 4 | 110 | 12000 |

## Restricting Group Results with the HAVING Clause

When you use the HAVING clause, the Oracle server restricts groups as follows:

1. Rows are grouped.
2. The group function is applied.
3. Groups matching the HAVING clause are displayed.

```
SELECT     column, group_function
FROM       table
[WHERE     condition]
[GROUP BY  group_by_expression]
[HAVING    group_condition]
[ORDER BY  column];
```

## Using the HAVING Clause

```
SELECT    department_id, MAX(salary)
FROM      employees
GROUP BY  department_id
HAVING    MAX(salary)>10000 ;
```

| | DEPARTMENT_ID | MAX(SALARY) |
|---|---|---|
| 1 | 90 | 24000 |
| 2 | 20 | 13000 |
| 3 | 110 | 12000 |
| 4 | 80 | 11000 |

## Using the HAVING Clause

```
SELECT    job_id, SUM(salary) PAYROLL
FROM      employees
WHERE     job_id NOT LIKE '%REP%'
GROUP BY job_id
HAVING    SUM(salary) > 13000
ORDER BY SUM(salary);
```

| | JOB_ID | PAYROLL |
|---|---|---|
| 1 | IT_PROG | 19200 |
| 2 | AD_PRES | 24000 |
| 3 | AD_VP | 34000 |

## Nesting Group Functions

Display the maximum average salary:

```
SELECT   MAX(AVG(salary))
FROM     employees
GROUP BY department_id;
```

| | MAX(AVG(SALARY)) |
|---|---|
| 1 | 19333.3333333333333333333333333333333333 |

**Lab Task**

**Q1 .Create following table with all these records and apply all types of aggregate functions you have learned so far on the appropriate data from all the columns in below mentioned table and show the SQL query and the resultant output also**

| *ID* | *name* | *salary* | *dept_name* | *building* | *budget* |
|---|---|---|---|---|---|
| 22222 | Einstein | 95000 | Physics | Watson | 70000 |
| 12121 | Wu | 90000 | Finance | Painter | 120000 |
| 32343 | El Said | 60000 | History | Painter | 50000 |
| 45565 | Katz | 75000 | Comp. Sci. | Taylor | 100000 |
| 98345 | Kim | 80000 | Elec. Eng. | Taylor | 85000 |
| 76766 | Crick | 72000 | Biology | Watson | 90000 |
| 10101 | Srinivasan | 65000 | Comp. Sci. | Taylor | 100000 |
| 58583 | Califieri | 62000 | History | Painter | 50000 |

**Q2. Create following table**

| eid | Name | Salary | Department | Date_Joined | Office_Code |
|---|---|---|---|---|---|
| 1 | Raj Patel | 60000 | Executive | 2020-11-01 | PUN |
| 2 | Radhika Sharma | 80000 | Operations | 2019-02-12 | MUM |
| 3 | Shreya Kulkarni | 55000 | Human Resources | 2018-05-30 | PUN |
| 4 | Vinay Jagdale | 75000 | Operations | 2020-01-03 | PUN |
| 5 | Jay Maini | 75000 | Marketing | 2020-08-17 | MUM |
| 6 | Vinisha Subramaniam | 90000 | Executive | 2018-04-05 | CHE |
| 7 | Ritwik Pawar | 90000 | Executive | 2018-04-05 | CHE |
| 8 | Arun Banerjee | 85000 | Marketing | 2018-04-05 | KOL |
| 10 | Devesh Panchal | 70000 | Operations | 2018-04-05 | PUN |

- **Consider an appropriate scenario and write an SQL query to apply count and sum function on only executive department**
- **Consider salary column and apply minimum and maximum function on it**

# LAB 11: Use of joins for displaying data from multiple tables

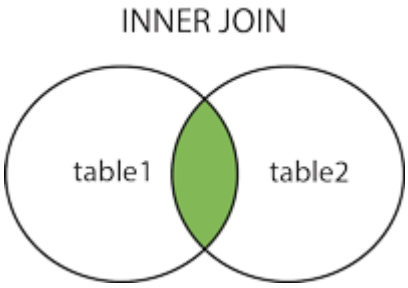
**Objectives**

- Retrieve data from different tables

**Theoretical Description**

Create different tables applying referential integrity constraints. Use foreign keys for integrity constraints and then apply different joins to overserve the retrieved data.

A **JOIN** clause is used to combine rows from two or more tables, based on a related column between them.




**Syntax:**

- SELECT column_name(s) FROM table1 **JOIN** table2 ON table1.column_name = table2.column_name;

JOIN and INNER JOIN are same in MySQL:

- SELECT column_name(s) FROM table1 **INNER JOIN** table2 ON table1.column_name = table2.column_name;

**Tables:**

```
mysql> select * from STUDENT;
+-----+-------+--------+
| sid | sname | CITY   |
+-----+-------+--------+
|   1 | ali   | lahore |
|   2 | asma  | psh    |
|   3 | ahmad | rwp    |
|   4 | sania | lhr    |
|   5 | shah  | psh    |
+-----+-------+--------+
5 rows in set (0.00 sec)
```

```
mysql> select * from TEACHER;
+-----+--------+------------+
| tid | name   | department |
+-----+--------+------------+
|  10 | akmal  | se         |
|  11 | ajmal  | se         |
|  12 | sara   | cs         |
|  13 | junaid | cs         |
|  14 | fawad  | mgm        |
|  15 | aqsa   | se         |
+-----+--------+------------+
6 rows in set (0.00 sec)
```

```
mysql> select * from COURSE;
+-----+------------+------+------+
| cid | coursename | sid  | tid  |
+-----+------------+------+------+
|   4 | bio        |    1 |   10 |
|   5 | phy        |    1 |   10 |
|   6 | chem       |    3 |   11 |
|   7 | eng        |    3 |   12 |
|   8 | math       |    4 |   13 |
|   9 | dbms       | NULL |   14 |
+-----+------------+------+------+
6 rows in set (0.02 sec)
```

**Example:**

1. Now, we create the following SQL statement (that contains an INNER JOIN), that selects records that have matching values in both tables:

**mysql> select STUDENT.sname,COURSE.coursename from STUDENT JOIN COURSE ON STUDENT.sid=COURSE.sid;**

```
mysql> select STUDENT.sname,COURSE.coursename from STUDENT JOIN COURSE ON STUDENT.sid=COURSE.sid;
+-------+------------+
| sname | coursename |
+-------+------------+
| ali   | bio        |
| ali   | phy        |
| ahmad | chem       |
| ahmad | eng        |
| sania | math       |
+-------+------------+
5 rows in set (0.00 sec)
```

The INNER JOIN keyword selects all rows from both tables as long as there is a match between the columns. If there are records in the "STUDENT" table that do not have matches in "COURSE", these will not be shown.

2. Now, we create the following SQL statement (that contains an INNER JOIN), that selects all records that have matching values in both tables:

**mysql> select * from STUDENT JOIN COURSE ON STUDENT.sid=COURSE.sid;**

```
mysql> select * from STUDENT JOIN COURSE ON STUDENT.sid=COURSE.sid;
+-----+-------+--------+-----+------------+------+------+
| sid | sname | CITY   | cid | coursename | sid  | tid  |
+-----+-------+--------+-----+------------+------+------+
|   1 | ali   | lahore |   4 | bio        |    1 |   10 |
|   1 | ali   | lahore |   5 | phy        |    1 |   10 |
|   3 | ahmad | rwp    |   6 | chem       |    3 |   11 |
|   3 | ahmad | rwp    |   7 | eng        |    3 |   12 |
|   4 | sania | lhr    |   8 | math       |    4 |   13 |
+-----+-------+--------+-----+------------+------+------+
5 rows in set (0.09 sec)
```

3. JOIN and INNER JOIN are same in MySQL:

**mysql> select * from STUDENT INNER JOIN COURSE ON STUDENT.sid=COURSE.sid;**

```
mysql> select * from STUDENT INNER JOIN COURSE ON STUDENT.sid=COURSE.sid;
+-----+-------+--------+-----+------------+------+------+
| sid | sname | CITY   | cid | coursename | sid  | tid  |
+-----+-------+--------+-----+------------+------+------+
|   1 | ali   | lahore |   4 | bio        |    1 |   10 |
|   1 | ali   | lahore |   5 | phy        |    1 |   10 |
|   3 | ahmad | rwp    |   6 | chem       |    3 |   11 |
|   3 | ahmad | rwp    |   7 | eng        |    3 |   12 |
|   4 | sania | lhr    |   8 | math       |    4 |   13 |
+-----+-------+--------+-----+------------+------+------+
5 rows in set (0.00 sec)
```

**JOIN Three Tables:**
**mysql> select * from STUDENT join COURSE ON STUDENT.sid=COURSE.sid join teacher on COURSE.tid=TEACHER.tid;**

```
mysql> select * from STUDENT join COURSE ON STUDENT.sid=COURSE.sid join teacher on COURSE.tid=TEACHER.tid;
+-----+-------+--------+-----+------------+------+------+-----+--------+------------+
| sid | sname | CITY   | cid | coursename | sid  | tid  | tid | name   | department |
+-----+-------+--------+-----+------------+------+------+-----+--------+------------+
|   1 | ali   | lahore |   4 | bio        |    1 |   10 |  10 | akmal  | se         |
|   1 | ali   | lahore |   5 | phy        |    1 |   10 |  10 | akmal  | se         |
|   3 | ahmad | rwp    |   6 | chem       |    3 |   11 |  11 | ajmal  | se         |
|   3 | ahmad | rwp    |   7 | eng        |    3 |   12 |  12 | sara   | cs         |
|   4 | sania | lhr    |   8 | math       |    4 |   13 |  13 | junaid | cs         |
+-----+-------+--------+-----+------------+------+------+-----+--------+------------+
5 rows in set (0.00 sec)
```

**mysql> select STUDENT.sname,COURSE.coursename,TEACHER.name from STUDENT join COURSE ON STUDENT.sid=COURSE.sid join teacher on COURSE.tid=TEACHER.tid;**

```
mysql> select STUDENT.sname,COURSE.coursename,TEACHER.name from STUDENT join COURSE ON STUDENT.sid=COURSE.sid join teacher on COURSE.tid=TEACHER.tid;
+-------+------------+--------+
| sname | coursename | name   |
+-------+------------+--------+
| ali   | bio        | akmal  |
| ali   | phy        | akmal  |
| ahmad | chem       | ajmal  |
| ahmad | eng        | sara   |
| sania | math       | junaid |
+-------+------------+--------+
5 rows in set (0.00 sec)
```

**4.** INNER JOIN with where clause:

**mysql> select STUDENT.sname,COURSE.coursename from STUDENT inner join COURSE ON STUDENT.sid=COURSE.sid where student.sid>2;**

```
mysql> select STUDENT.sname,COURSE.coursename from STUDENT join COURSE ON STUDENT.sid=COURSE.sid where student.sid>2;
+-------+------------+
| sname | coursename |
+-------+------------+
| ahmad | chem       |
| ahmad | eng        |
| sania | math       |
+-------+------------+
3 rows in set (0.02 sec)
```

**Lab Task**

Q1. Consider your project and apply joins on two tables as well as multiple tables

# LAB 12: Sub Queries and set operators

**Objectives**

- Learn different ways to extract data using sub queries
- Learn Nesting of subqueries
- Learn SET Operators (UNION, UNION ALL, INTERSECT, & EXCEPT) in MySQL.
- Implement SET operations examples.

**Theoretical Description**

**Subquery Syntax :**

```
Select      select_list
From        table
Where       expr operator
                        ( Select    select_list
                          From      table );
```

- The subquery (inner query) executes once before the main query (outer query) executes.
- The main query (outer query) use the subquery result.

**MySQL Subquery Example:**

Using a subquery, list the name of the employees, paid more than 'Alexander' from emp_details .

```
Select      first_name, last_name, salary
From        employees
Where       salary >                    9000
                    ( Select    salary
                      From      employees
                      Where     first_name='Alexander' );
```

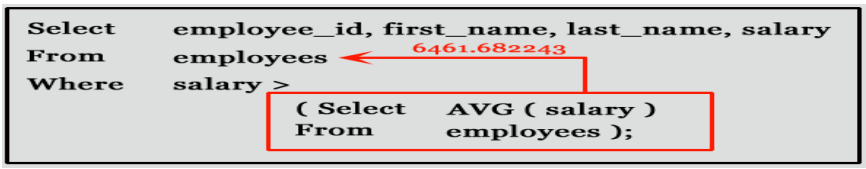
There are some guidelines to consider when using subqueries :
- A subquery must be enclosed in parentheses.
- Use single-row operators with single-row subqueries, and use multiple-row operators with multiple-row subqueries.
- If a subquery (inner query) returns a null value to the outer query, the outer query will not return any rows when using certain comparison operators in a WHERE clause.

**MySQL Subqueries: Using Comparisons**

A subquery can be used before or after any of the comparison operators. The subquery can return at most one value. The value can be the result of an arithmetic expression or a column function. SQL then compares the value that results from the subquery with the value on the other side of the comparison operator. You can use the following comparison operators:

| Operator | Description |
|---|---|
| = | Equal to |
| > | Greater than |
| >= | Greater than or equal to |
| < | Less than |
| <= | Less than or equal to |
| != | Not equal to |
| <> | Not equal to |
| <=> | NULL-safe equal to operator |

For example, suppose you want to find the employee id, first_name, last_name, and salaries for employees whose average salary is higher than the average salary throughout the company.




**MySQL Subqueries with ALL, ANY, IN, or SOME**

**Example: MySQL Subquery, ALL operator**

The following query selects the department with the highest average salary. The subquery finds the average salary for each department, and then the main query selects the department with the highest average salary.

```
mysql> SELECT department_id, AVG(SALARY)
FROM EMPLOYEES GROUP BY department_id
HAVING AVG(SALARY)>=ALL
(SELECT AVG(SALARY) FROM EMPLOYEES GROUP BY department_id);
+---------------+--------------+
| department_id | AVG(SALARY)  |
+---------------+--------------+
|            90 | 19333.333333 |
+---------------+--------------+
1 row in set (0.00 sec)
```

Note: Here we have used ALL keyword for this subquery as the department selected by the query must have an average salary greater than or equal to all the average salaries of the other departments.

The following query selects any employee who works in the location 1800. The subquery finds the department id in the 1800 location, and then the main query selects the employees who work in any of these departments.

```
mysql> SELECT first_name, last_name,department_id
FROM employees WHERE department_id= ANY
(SELECT DEPARTMENT_ID FROM departments WHERE location_id=1800);
+------------+-----------+---------------+
| first_name | last_name | department_id |
+------------+-----------+---------------+
| Michael    | Hartstein |            20 |
| Pat        | Fay       |            20 |
+------------+-----------+---------------+
2 rows in set (0.00 sec)
```

**MySQL Row Subqueries**

A row subquery is a subquery that returns a single row and more than one column value. You can use = , >, <, >=, <=, <>, !=, <=> comparison operators. See the following examples:

**Example: MySQL Row Subqueries**

In the following examples, queries shows differentr result according to above conditions :

```
mysql> SELECT first_name
FROM employees
WHERE ROW(department_id, manager_id) = (SELECT department_id, manager_id FROM
departments WHERE location_id = 1800);
+------------+
| first_name |
+------------+
| Pat        |
+------------+
1 row in set (0.00 sec)
```

The SET Operators in MySQL are basically used to combine the result of more than 1 select statement and return the output as a single result set. In SQL, 4 types of set operators are. They are as follows:

1. **UNION**: It is used to combine two or more result sets into a single set, without duplicates.
2. **UNION ALL**: It is used to combine two or more result sets into a single set, including duplicates.
3. **INTERSECT**: It is used to combine two result sets and returns the data which are common in both the result set.
4. **EXCEPT**: It is used to combine two result sets and returns the data from the first result set which is not present in the second result set.

We are going to use the following EmployeeUK and EmployeeUSA tables to understand the SET Operators in MySQL.

**EmployeeUK**

| EmployeeId | FirstName | LastName | Gender | Department |
|---|---|---|---|---|
| 1 | Pranaya | Rout | Male | IT |
| 2 | Priyanka | Dewangan | Female | IT |
| 3 | Preety | Tiwary | Female | HR |
| 4 | Subrat | Sahoo | Male | HR |
| 5 | Anurag | Mohanty | Male | IT |
| 6 | Rajesh | Pradhan | Male | HR |
| 7 | Hina | Sharma | Female | IT |

**EmployeeUSA**

| EmployeeId | FirstName | LastName | Gender | Department |
|---|---|---|---|---|
| 1 | James | Pattrick | Male | IT |
| 2 | Priyanka | Dewangan | Female | IT |
| 3 | Sara | Taylor | Female | HR |
| 4 | Subrat | Sahoo | Male | HR |
| 5 | Sushanta | Jena | Male | HR |
| 6 | Mahesh | Sindhey | Female | HR |
| 7 | Hina | Sharma | Female | IT |

**UNION Operator in MySQL**

The UNION operator is used to combine the result set of two or more SELECT statements into a single result set by removing the duplicate records. That means the UNION Operator selects only the distinct values. Following is the Syntax to use UNION Operator in MySQL.

```
SELECT column_list FROM table1
UNION
SELECT column_list FROM table2;
```

**MySQL UNION Operator Example:**

The following query combines two select statements using the UNION operator. In our example, both the EmployeeUK and EmployeeUSA tables having seven records.

```
SELECT FirstName, LastName, Gender, Department FROM EmployeeUK
UNION
SELECT FirstName, LastName, Gender, Department FROM EmployeeUSA;
```

Once you execute the above query, you will get the following result set. Please observe, here we don't have any duplicate data. Here, in the result set, we got a total of 11 rows out of 14 rows. This is because 3 rows are present in both the result set.

| FirstName | LastName | Gender | Department |
|---|---|---|---|
| Pranaya | Rout | Male | IT |
| Priyanka | Dewangan | Female | IT |
| Preety | Tiwary | Female | HR |
| Subrat | Sahoo | Male | HR |
| Anurag | Mohanty | Male | IT |
| Rajesh | Pradhan | Male | HR |
| Hina | Sharma | Female | IT |
| James | Pattrick | Male | IT |
| Sara | Taylor | Female | HR |
| Sushanta | Jena | Male | HR |
| Mahesh | Sindhey | Female | HR |

**UNION ALL Operator in MySQL**

The UNION ALL operator is used to combine the result set of two or more SELECT statements into a single result including the duplicate values. Following is the Syntax to use UNION ALL Operator in MySQL.

```
SELECT Column_list FROM table1
UNION ALL
SELECT Column_list FROM table2;
```

**MySQL UNION ALL Operator Example:**

The following query combines two select statements using the UNION ALL operator

```
SELECT FirstName, LastName, Gender, Department FROM EmployeeUK
UNION ALL
SELECT FirstName, LastName, Gender, Department FROM EmployeeUSA;
```

Once you execute the above query, you will get the following result set. Please observe, here we got all the 14 rows in the result set.

| FirstName | LastName | Gender | Department |
|---|---|---|---|
| Pranaya | Rout | Male | IT |
| Priyanka | Dewangan | Female | IT |
| Preety | Tiwary | Female | HR |
| Subrat | Sahoo | Male | HR |
| Anurag | Mohanty | Male | IT |
| Rajesh | Pradhan | Male | HR |
| Hina | Sharma | Female | IT |
| James | Pattrick | Male | IT |
| Priyanka | Dewangan | Female | IT |
| Sara | Taylor | Female | HR |
| Subrat | Sahoo | Male | HR |
| Sushanta | Jena | Male | HR |
| Mahesh | Sindhey | Female | HR |
| Hina | Sharma | Female | IT |

**Differences between UNION and UNION ALL Operator in MySQL**

From the output, it is very clear to us that UNION Operator removes duplicate rows whereas UNION ALL operator does not remove the duplicate rows. When we use a UNION operator, in order to remove the duplicate rows from the result set, it has to do a distinct operation which is time-consuming. For this reason, UNION ALL is much faster than UNION Operator in MySQL.

**UNION/UNION ALL with ORDER BY Clause in MySQL**

The UNION/UNION ALL Operator can be used with the ORDER BY clause to sort the result returned from the query. Suppose we want to sort the employees by First Name column values. ORDER BY clause should be part of the last select statement. The SQL statement will be:

```
SELECT FirstName, LastName, Gender, Department FROM EmployeeUK
UNION
SELECT FirstName, LastName, Gender, Department FROM EmployeeUSA
ORDER BY FirstName;
```

Now execute the query and you should get the following output.

| FirstName | LastName | Gender | Department |
|---|---|---|---|
| Anurag | Mohanty | Male | IT |
| Hina | Sharma | Female | IT |
| James | Pattrick | Male | IT |
| Mahesh | Sindhey | Female | HR |
| Pranaya | Rout | Male | IT |
| Preety | Tiwary | Female | HR |
| Priyanka | Dewangan | Female | IT |
| Rajesh | Pradhan | Male | HR |
| Sara | Taylor | Female | HR |
| Subrat | Sahoo | Male | HR |
| Sushanta | Jena | Male | HR |

Here you can see the employees are sorted according to their FirstName column values.

**MySQL EXCEPT Operator:**

The EXCEPT operator is used to combine two tables or two result sets and will return rows from the first select statement that are not present in the second select statement. Following is the syntax of EXCEPT Operator.

```
SELECT column1 [, column2 ] FROM table1 [, table2 ]
[WHERE condition]

EXCEPT

SELECT column1 [, column2 ] FROM table1 [, table2 ]
[WHERE condition]
```

But, the EXCEPT Operator is not supported by MYSQL. We can achieve the EXCEPT Operator functionality in MySQL using the following ways.

Using NOT IN Operator to achieve EXCEPT functionality:

Here, we are checking the FirstName column value only. Following is the SQL Query using the NOT IN Operator which returns the employees from the first EmployeeUK table that are not present in the EmployeeUSA table.

```
SELECT * FROM EmployeeUK
WHERE FirstName NOT IN (SELECT FirstName FROM EmployeeUSA);
```

Once you execute the above query, you will get the following result set.

| EmployeeId | FirstName | LastName | Gender | Department |
|---|---|---|---|---|
| 1 | Pranaya | Rout | Male | IT |
| 3 | Preety | Tiwary | Female | HR |
| 5 | Anurag | Mohanty | Male | IT |
| 6 | Rajesh | Pradhan | Male | HR |

**Using Join to achieve EXCEPT functionality in MySQL:**

We can use LEFT JOIN to achieve the functionality of EXCEPT Operator. Here, the join clause needs to contain all 4 columns FirstName, LastName, Gender, and Department. The where clause picks null values in EmployeeId in EmployeeUSA, which limits to rows that exist in EmployeeUK only.

```
SELECT t1.* FROM EmployeeUK AS t1
LEFT JOIN EmployeeUSA AS t2 ON
t1.FirstName=t2.FirstName
AND t1.LastName=t2.LastName
AND t1.Gender=t2.Gender
AND t1.Department=t2.Department
WHERE t2.EmployeeId IS NULL;
```

Once you execute the above join query, you will get the following result set.

| EmployeeId | FirstName | LastName | Gender | Department |
|---|---|---|---|---|
| 1 | Pranaya | Rout | Male | IT |
| 3 | Preety | Tiwary | Female | HR |
| 5 | Anurag | Mohanty | Male | IT |
| 6 | Rajesh | Pradhan | Male | HR |

**INTERSECT Operator in MySQL**

The INTERSECT operator is used to combine two result sets and returns the data which are common in both the result set. Following is the syntax of INTERSECT operator.

```
SELECT column_lists FROM table_name WHERE condition
INTERSECT
SELECT column_lists FROM table_name WHERE condition;
```

But the INTERSECT Operator is not supported by MYSQL. We can achieve the INTERSECT Operator functionality in MySQL using the following ways.

Using IN Operator to achieve INTERSECT functionality:

Here, we are checking the FirstName column value only. Following is the SQL Query using the IN Operator which returns the common employees i.e. the employees which are present in both t EmployeeUK and EmployeeUSA tables. Here, we are checking common based on the First Name column value.

```
SELECT * FROM EmployeeUK
WHERE FirstName IN (SELECT FirstName FROM EmployeeUSA);
```

Once you execute the above query, you will get the following result set.

| EmployeeId | FirstName | LastName | Gender | Department |
|---|---|---|---|---|
| 2 | Priyanka | Dewangan | Female | IT |
| 4 | Subrat | Sahoo | Male | HR |
| 7 | Hina | Sharma | Female | IT |

Using Join to achieve INTERSECT functionality in MySQL:

We can use INNER JOIN to achieve the functionality of INTERSECT Operator. Here, the join clause needs to contain all 4 columns FirstName, LastName, Gender, and Department.

```
SELECT t1.* FROM EmployeeUK AS t1
INNER JOIN EmployeeUSA AS t2 ON
t1.FirstName=t2.FirstName
AND t1.LastName=t2.LastName
AND t1.Gender=t2.Gender
AND t1.Department=t2.Department;
```

Once you execute the above join query, you will get the following result set.

| EmployeeId | FirstName | LastName | Gender | Department |
|---|---|---|---|---|
| 2 | Priyanka | Dewangan | Female | IT |
| 4 | Subrat | Sahoo | Male | HR |
| 7 | Hina | Sharma | Female | IT |

**Lab Task**

Q1 Practice union, union all, intersect and  except on your project

Q2. Consider your project and  write multiple scenarios to apply subqueries

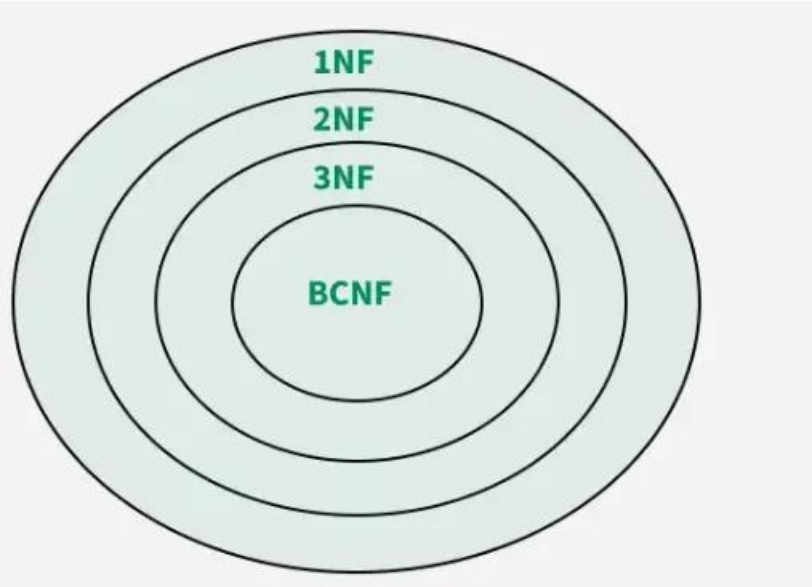
# LAB 13: Normalization

**Objectives**

- The students would apply 1NF 2NF 3NF normalization and denormalization on the projects assigned to them

**Theoretical description**

Normalization is a systematic approach to organize data in a database to eliminate redundancy, avoid anomalies and ensure data consistency. The process involves breaking down large tables into smaller, well-structured ones and defining relationships between them. This not only reduces the chances of storing duplicate data but also improves the overall efficiency of the database. Normalization consists of following forms




- A relation in BCNF is also in 3NF , a relation in 3NF is also in 2NF and a relation in 2NF is also in 1NF.
- A relation in BCNF is considered fully normalized.

**Lab Task**

Students will apply normalization on their projects

# Open Ended Lab (OEL)

**Course:** Database Systems **Total Marks: 15**

## OEL Title: Building a Library Management System

## OEL Level: Level 1

## Objectives:

To guide students through creating a library database system using predefined tables and SQL queries while encouraging them to generate their own outputs and analyze results.

- Hands-on experience in implementing a database structure.
- Writing SQL queries to fulfill functional requirements.
- Understanding how to analyze and present results from database queries.

## Background Information:

Libraries require efficient database systems to store and retrieve information about books, users, and borrowing activities. This lab focuses on implementing a library management system based on predefined requirements, with the freedom to interpret and present results independently.

## Problem:

Design a library management system that meets the following predefined specifications:

1. **Books**: Stores book details (e.g., BookID, Title, Author, PublicationYear, and Genre).
2. **Users**: Stores user details (e.g., UserID, Name, MembershipDate, and ContactInfo).
3. **Transactions**: Tracks borrowing activities (TransactionID, UserID, BookID, BorrowDate, ReturnDate, and FineAmount).

## Requirements:

1. Students must create three predefined tables
2. Students must populate tables with at least **10 entries** per table. Sample data can be based on the structure provided below but should be generated by the students:
    - Books: Details like titles, authors, and genres.
    - Users: Fictional user details.

- Transactions: Borrowing and returning records.

3. Students must write and execute SQL queries for the following tasks:

    - List all available books in the library.
    - Identify overdue books (assume the return due date is 7 days after borrowing).
    - Retrieve the borrowing history of a specific user (student's choice).
    - Calculate the total fines generated in the library.