**Database Management System Laboratory**

**GROUP A  
Experiment No: 3**

Name: Onasvee Banarse

TE COMPUTER SHIFT 1

Roll no: 09

**Aim**: Design at least 10 SQL queries for suitable database application using SQL DML and DDL statements: Insert, Select, Update, Delete, Index, View, Table, Sequence , Synonym .

**Objective:**

1. To learn and understand DML statements in MySQL
2. To learn and understand DDL statements in MySQL/SQL

**Requirement:** Any OS, MySQL, Workbench, Oracle Workbench (Online).

**Theory:**

Structured Query Language (SQL) as we all know is the database language by the use of which we can perform certain operations on the existing database and also we can use this language to create a database. SQL uses certain commands like Create, Drop, Insert, etc. to carry out the required tasks.

These SQL commands are mainly categorized into four categories as:

1. DDL – Data Definition Language
2. DQl – Data Query Language
3. DML – Data Manipulation Language
4. DCL – Data Control Language

**DDL (Data Definition Language):**

DDL or Data Definition Language consists of the SQL commands that can be used to define the database schema. It simply deals with descriptions of the database schema and is used to create and modify the structure of database objects in the database. DDL is a set of SQL commands used to create, modify, and delete database structures but not data. These commands are normally not used by a general user, who should be accessing the database via an application.

**List of DDL commands:**

1. **CREATE:** This command is used to create the database or its objects (like table, index, function, views, store procedure, and triggers).
2. **DROP:** This command is used to delete objects from the database.
3. **ALTER:** This is used to alter the structure of the database.
4. **TRUNCATE:** This is used to remove all records from a table, including all spaces allocated for the records are removed.
5. **COMMENT:** This is used to add comments to the data dictionary.
6. **RENAME:** This is used to rename an object existing in the database.

**DML(Data Manipulation Language):**

The SQL commands that deal with the manipulation of data present in the database belong to DML or Data Manipulation Language and this includes most of the SQL statements. It is the component of the SQL statement that controls access to data and to the database. Basically, DCL statements are grouped with DML statements.

**List of DML commands:**

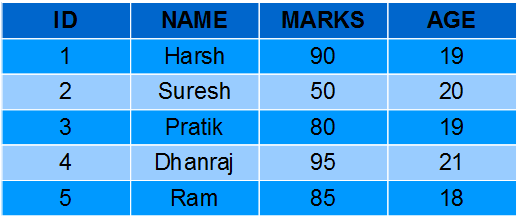
1. **INSERT** : It is used to insert data into a table.
2. **UPDATE**: It is used to update existing data within a table.
3. **DELETE** : It is used to delete records from a database table.
4. **LOCK**: Table control concurrency.
5. **CALL**: Call a PL/SQL or JAVA subprogram.
6. **EXPLAIN PLAN**: It describes the access path to data.

**SQL Functions (Aggregate and Scalar Functions)**

For doing operations on data sql has many built-in functions, they are categorised in two categories and further sub-categorised in different seven functions under each category. The categories are:

1. **Aggregate functions:**  
   These functions are used to do operations from the values of the column and a single value is returned.
   1. AVG()
   2. COUNT()
   3. FIRST()
   4. LAST()
   5. MAX()
   6. MIN()
   7. SUM()
2. **Scalar functions:**  
   These functions are based on user input; these too returns single value.
   1. UCASE()
   2. LCASE()
   3. MID()
   4. LEN()
   5. ROUND()
   6. NOW()
   7. FORMAT()

Students-Table



**Aggregate Functions**

* **AVG()**: It returns average value after calculating from values in a numeric column.  
  Syntax:

SELECT AVG(column\_name) FROM table\_name;

**Queries:**

* 1. Computing average marks of students.

SELECT AVG(MARKS) AS AvgMarks FROM Students;

Output:

|  |
| --- |
| **AvgMarks** |
| 80 |

* 1. Computing average age of students.

SELECT AVG(AGE) AS AvgAge FROM Students;

Output:

|  |
| --- |
| **AvgAge** |
| 19.4 |

* **COUNT():**It is used to count the number of rows returned in a SELECT statement. It can’t be used in MS ACCESS.  
  Syntax:

SELECT COUNT(column\_name) FROM table\_name;

**Queries**:

* 1. Computing total number of students.

SELECT COUNT(\*) AS NumStudents FROM Stuents;

Output:

|  |
| --- |
| **NumStudents** |
| 5 |

* 1. Computing number of students with unique/distinct age.

SELECT COUNT(DISTINCT AGE) AS NumStudents FROM Students;

Output:

|  |
| --- |
| **NumStudents** |
| 4 |

* **FIRST():**The FIRST() function returns the first value of the selected column.  
  Syntax:

SELECT FIRST(column\_name) FROM table\_name;

**Queries**:

* 1. Fetching marks of first student from the Students table.

SELECT FIRST(MARKS) AS MarksFirst FROM Students;

Output:

|  |
| --- |
| **MarksFirst** |
| 90 |

* 1. Fetching age of first student from the Students table.

SELECT FIRST(AGE) AS AgeFirst FROM Students;

Output:

|  |
| --- |
| **AgeFirst** |
| 19 |

* **LAST():**The LAST() function returns the last value of the selected column. It can be used only in MS ACCESS.  
  Syntax:

SELECT LAST(column\_name) FROM table\_name;

**Queries:**

* 1. Fetching marks of last student from the Students table.

SELECT LAST(MARKS) AS MarksLast FROM Students;

Output:

|  |
| --- |
| **MarksLast** |
| 82 |

* 1. Fetching age of last student from the Students table.

SELECT LAST(AGE) AS AgeLast FROM Students;

Output:

|  |
| --- |
| **AgeLast** |
| 18 |

* **MAX():**The MAX() function returns the maximum value of the selected column.  
  Syntax:

SELECT MAX(column\_name) FROM table\_name;

**Queries**:

* 1. Fetching maximum marks among students from the Students table.

SELECT MAX(MARKS) AS MaxMarks FROM Students;

Output:

|  |
| --- |
| **MaxMarks** |
| 95 |

* 1. Fetching max age among students from the Students table.

SELECT MAX(AGE) AS MaxAge FROM Students;

Output:

|  |
| --- |
| **MaxAge** |
| 21 |

* **MIN():**The MIN() function returns the minimum value of the selected column.  
  Syntax:

SELECT MIN(column\_name) FROM table\_name;

**Queries:**

1. Fetching minimum marks among students from the Students table.

SELECT MIN(MARKS) AS MinMarks FROM Students;

Output:

|  |
| --- |
| **MinMarks** |
| 50 |

1. Fetching minimum age among students from the Students table.

SELECT MIN(AGE) AS MinAge FROM Students;

Output:

|  |
| --- |
| **MinAge** |
| 18 |

* **SUM():**The SUM() function returns the sum of all the values of the selected column.  
  Syntax:

SELECT SUM(column\_name) FROM table\_name;

**Queries:**

1. Fetching summation of total marks among students from the Students table.

SELECT SUM(MARKS) AS TotalMarks FROM Students;

Output:

|  |
| --- |
| **TotalMarks** |
| 400 |

1. Fetching summation of total age among students from the Students table.

SELECT SUM(AGE) AS TotalAge FROM Students;

Output:

|  |
| --- |
| **TotalAge** |
| 9 |

**Scalar Functions**

* **UCASE()**: It converts the value of a field to uppercase.  
  Syntax:

SELECT UCASE(column\_name) FROM table\_name;

**Queries:**

* 1. Converting names of students from the table Students to uppercase.

SELECT UCASE(NAME) FROM Students;

Output:

|  |
| --- |
| **NAME** |
| HARSH |
| SURESH |
| PRATIK |
| DHANRAJ |
| RAM |

* **LCASE()**: It converts the value of a field to lowercase.  
  Syntax:

SELECT LCASE(column\_name) FROM table\_name;

**Queries:**

* 1. Converting names of students from the table Students to lowercase.

SELECT LCASE(NAME) FROM Students;

Output:

|  |
| --- |
| **NAME** |
| harsh |
| suresh |
| pratik |
| dhanraj |
| ram |

* **MID():**The MID() function extracts texts from the text field.  
  Syntax:

SELECT MID(column\_name,start,length) AS some\_name FROM table\_name;

specifying length is optional here, and start signifies start position ( starting from 1 )

**Queries:**

* 1. Fetching first four characters of names of students from the Students table.

SELECT MID(NAME,1,4) FROM Students;

Output:

|  |
| --- |
| **NAME** |
| HARS |
| SURE |
| PRAT |
| DHAN |
| RAM |

* **LEN():**The LEN() function returns the length of the value in a text field.  
  Syntax:

SELECT LENGTH(column\_name) FROM table\_name;

**Queries:**

* 1. Fetching length of names of students from Students table.

SELECT LENGTH(NAME) FROM Students;

Output:

|  |
| --- |
| **NAME** |
| 5 |
| 6 |
| 6 |
| 7 |
| 3 |

* **ROUND():**The ROUND() function is used to round a numeric field to the number of decimals specified.NOTE: Many database systems have adopted the IEEE 754 standard for arithmetic operations, which says that when any numeric .5 is rounded it results to the nearest even integer i.e, 5.5 and 6.5 both gets rounded off to 6.

Syntax:

SELECT ROUND(column\_name,decimals) FROM table\_name;

decimals- number of decimals to be fetched.

**Queries:**

* 1. Fetching maximum marks among students from the Students table.

SELECT ROUND(MARKS,0) FROM table\_name;

Output:

|  |
| --- |
| **MARKS** |
| 90 |
| 50 |
| 80 |
| 95 |
| 85 |

* **NOW():**The NOW() function returns the current system date and time.  
  Syntax:

SELECT NOW() FROM table\_name;

**Queries:**

* 1. Fetching current system time.

SELECT NAME, NOW() AS DateTime FROM Students;

Output:

|  |  |
| --- | --- |
| **NAME** | **DateTime** |
| HARSH | 1/13/2017 1:30:11 PM |
| SURESH | 1/13/2017 1:30:11 PM |
| PRATIK | 1/13/2017 1:30:11 PM |
| DHANRAJ | 1/13/2017 1:30:11 PM |
| RAM | 1/13/2017 1:30:11 PM |

* **FORMAT():**The FORMAT() function is used to format how a field is to be displayed.  
  Syntax:

SELECT FORMAT(column\_name,format) FROM table\_name;

**Queries:**

* 1. Formatting current date as ‘YYYY-MM-DD’.

SELECT NAME, FORMAT(Now(),'YYYY-MM-DD') AS Date FROM Students;

Output:

|  |  |
| --- | --- |
| **NAME** | **Date** |
| HARSH | 2017-01-13 |
| SURESH | 2017-01-13 |
| PRATIK | 2017-01-13 |
| DHANRAJ | 2017-01-13 |
| RAM | 2017-01-13 |

**Set Operators**

Set operators are used to join the results of two (or more) SELECT statements. The SET operators available in Oracle 11g are UNION,UNION ALL, INTERSECT and MINUS.

**UNION**

When multiple SELECT queries are joined using UNION operator, Oracle displays the combined result from all the compounded SELECT queries,after removing all duplicates and in sorted order (ascending by default), without ignoring the NULL values.

Consider the below five queries joined using UNION operator.The final combined result set contains value from all the SQLs. Note the duplication removal and sorting of data.

SELECT 1 NUM FROM DUAL

UNION

SELECT 5 FROM DUAL

UNION

SELECT 3 FROM DUAL

UNION

SELECT 6 FROM DUAL

UNION

SELECT 3 FROM DUAL;

NUM

-------

1

3

5

6

To be noted, the columns selected in the SELECT queries must be of compatible data type. Oracle throws an error message when the rule is violated.

SELECT TO\_DATE('12-OCT-03') FROM DUAL

UNION

SELECT '13-OCT-03' FROM DUAL;

SELECT TO\_DATE('12-OCT-03') FROM DUAL

\*

ERROR at line 1:

ORA-01790: expression must have same datatype as corresponding expression

**UNION ALL**

UNION and UNION ALL are similar in their functioning with a slight difference. But UNION ALL gives the result set without removing duplication and sorting the data. For example, in above query UNION is replaced by UNION ALL to see the effect.

Consider the query demonstrated in UNION section. Note the difference in the output which is generated without sorting and deduplication.

SELECT 1 NUM FROM DUAL

UNION ALL

SELECT 5 FROM DUAL

UNION ALL

SELECT 3 FROM DUAL

UNION ALL

SELECT 6 FROM DUAL

UNION ALL

SELECT 3 FROM DUAL;

NUM

-------

1

5

3

6

3

**INTERSECT**

Using INTERSECT operator, Oracle displays the common rows from both the SELECT statements, with no duplicates and data arranged in sorted order (ascending by default).

For example, the below SELECT query retrieves the salary which are common in department 10 and 20.As per ISO SQL Standards, INTERSECT is above others in precedence of evaluation of set operators but this is not still incorporated by Oracle.

SELECT SALARY

FROM employees

WHERE DEPARTMENT\_ID = 10

INTRESECT

SELECT SALARY

FROM employees

WHERE DEPARTMENT\_ID = 20

SALARY

---------

1500

1200

2000

**MINUS**

Minus operator displays the rows which are present in the first query but absent in the second query, with no duplicates and data arranged in ascending order by default.

SELECT JOB\_ID

FROM employees

WHERE DEPARTMENT\_ID = 10

MINUS

SELECT JOB\_ID

FROM employees

WHERE DEPARTMENT\_ID = 20;

JOB\_ID

-------------

HR

FIN

ADMIN

**Matching the SELECT statement**

There may be the scenarios where the compound SELECT statements may have different count and data type of selected columns. Therefore, to match the column list explicitly, NULL columns are inserted at the missing positions so as match the count and data type of selected columns in each SELECT statement. For number columns, zero can also be substituted to match the type of the columns selected in the query.

In the below query, the data type of employee name (varchar2) and location id (number) do not match. Therefore, execution of the below query would raise error due to compatibility issue.

SELECT DEPARTMENT\_ID "Dept", first\_name "Employee"

FROM employees

UNION

SELECT DEPARTMENT\_ID, LOCATION\_ID

FROM departments;

ERROR at line 1:

ORA-01790: expression must have same datatype as corresponding expression

Explicitly, columns can be matched by substituting NULL for location id and Employee name.

SELECT DEPARTMENT\_ID "Dept", first\_name "Employee", NULL "Location"

FROM employees

UNION

SELECT DEPARTMENT\_ID, NULL "Employee", LOCATION\_ID

FROM departments;

**Using ORDER BY clause in SET operations**

The ORDER BY clause can appear only once at the end of the query containing compound SELECT statements. It implies that individual SELECT statements cannot have ORDER BY clause. Additionally, the sorting can be based on the columns which appear in the first SELECT query only. For this reason, it is recommended to sort the compound query using column positions.

The compound query below unifies the results from two departments and sorts by the SALARY column.

SELECT employee\_id, first\_name, salary

FROM employees

WHERE department\_id=10

UNION

SELECT employee\_id, first\_name, salary

FROM employees

WHERE department\_id=20

ORDER BY 3;

**SQL Queries:**

**Conclusion:**