Structured Query Language (SQL)

What is SQL?

SQL is a database language designed for the retrieval and management of data in a relational database.

SQL is the standard language for database management. All the RDBMS systems like MySQL, MS Access, Oracle, Sybase, Postgres, and SQL Server use SQL as their standard database language.

SQL programming language uses various commands for different operations. We will learn about the like DCL, TCL, DQL, DDL and DML commands in SQL with examples.

Why Use SQL?

Here, are important reasons for using SQL

- It helps users to access data in the RDBMS system.
- It helps you to describe the data.
- It allows you to define the data in a database and manipulate that specific data.
- With the help of SQL commands in DBMS, you can create and drop databases and tables.
- SQL offers you to use the function in a database, create a view, and stored procedure.
- You can set permissions on tables, procedures, and views.

Brief History of SQL

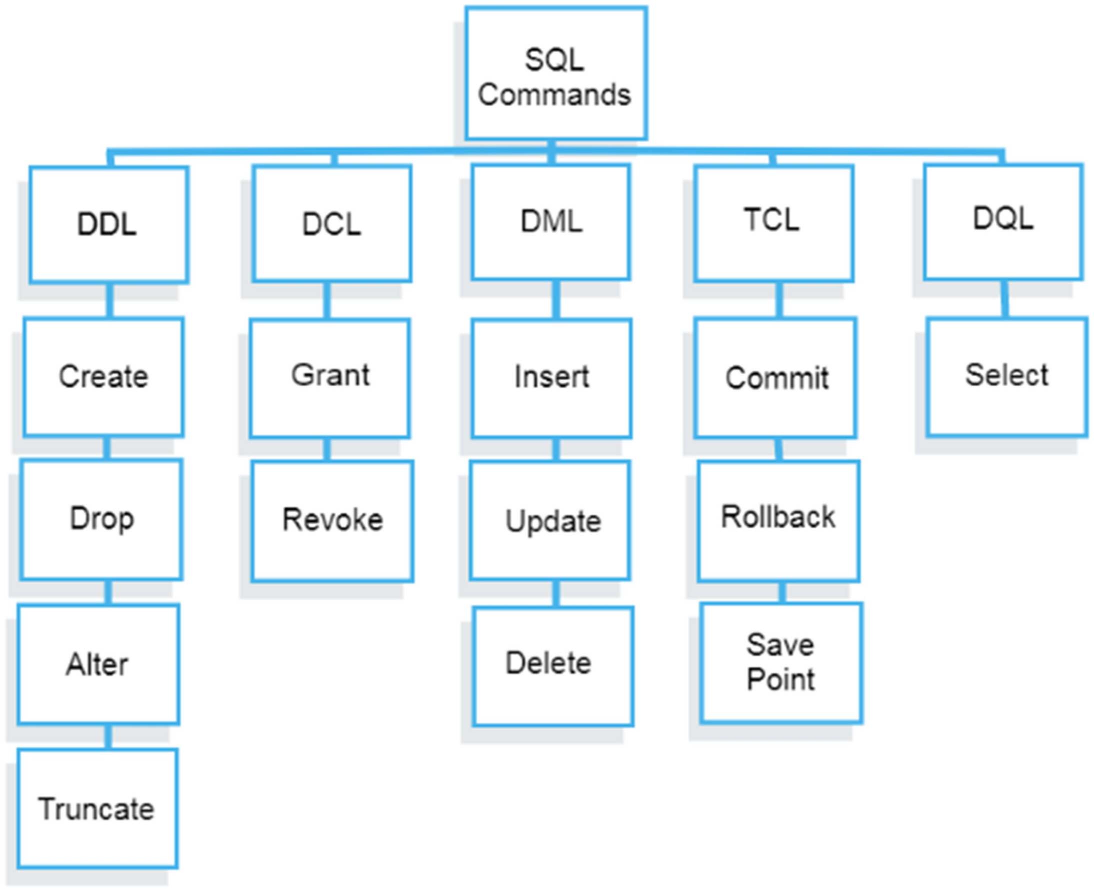
Here, are important landmarks from the history of SQL:

- 1970 – Dr. Edgar F. "Ted" Codd described a relational model for databases.
- 1974 – Structured Query Language appeared.
- 1978 – IBM released a product called System/R.
- 1986 – IBM developed the prototype of a relational database, which is standardized by ANSI.
- 1989- First ever version launched of SQL
- 1999 – SQL 3 launched with features like triggers, object-orientation, etc.
- SQL2003- window functions, XML-related features, etc.
- SQL2006- Support for XML Query Language
- SQL2011-improved support for temporal databases

Types of SQL

Here are five types of widely used SQL queries.

- Data Definition Language (DDL)
- Data Manipulation Language (DML)
- Data Control Language(DCL)
- Transaction Control Language(TCL)
- Data Query Language (DQL)



Structured Query Language (SQL)

DATA DEFINITION LANGUAGE (DDL):

The Data Definition Language (DDL) is used to create and destroy databases and database objects. These commands will primarily be used by database administrators during the setup and removal phases of a database project.

Let's take a look at the structure and usage of four basic DDL commands: CREATE, ALTER, DROP and RENAME

Practice No. 1

1. **Create Table**

Command	Syntax	Description
CREATE TABLE	CREATE TABLE table_name (column_1 datatype CONSTRAINT constraint_name constraint_desc, column_2 datatype, column_3 datatype);	It is used to create a new table in a database and specify the name of the table and columns inside it

DATA TYPES:

1. **CHAR (Size**): This data type is used to store character strings values of fixed length. The size in brackets determines the number of characters the cell can hold. The maximum number of character is 255 characters.
2. **VARCHAR (Size) / VARCHAR2 (Size):** This data type is used to store variable length alphanumeric data. The maximum character can hold is 2000 character.
3. **NUMBER (P, S):** The NUMBER data type is used to store number (fixed or floating point). Number of virtually any magnitude may be stored up to 38 digits of precision. Number as large as 9.99 * 10 124. The precision (p) determines the number of places to the right of the decimal. If scale is omitted then the default is zero. If precision is omitted, values are stored with their original precision up to the maximum of 38 digits.
4. **DATE**: This data type is used to represent date and time. The standard format is DDMM-YY as in 17-SEP-2009. To enter dates other than the standard format, use the appropriate functions. Date time stores date in the 24-Hours format. By default the time 9 in a date field

is 12:00:00 am, if no time portion is specified. The default date for a date field is the first day the current month.

5. **LONG**: This data type is used to store variable length character strings containing up to 2GB. Long data can be used to store arrays of binary data in ASCII format. LONG values cannot be indexed, and the normal character functions such as SUBSTR cannot be applied.
6. **RAW**: The RAW data type is used to store binary data, such as digitized picture or image. Data loaded into columns of these data types are stored without any further conversion. RAW data type can have a maximum length of 255 bytes. LONG RAW data type can contain up to 2GB.

CONSTRAINTS:

Constraints are used to specify rules for the data in a table. If there is any violation between the constraint and the data action, the action is aborted by the constraint. It can be specified when the table is created (using CREATE TABLE statement) or after the table is created (using ALTER TABLE statement).

1. **NOT NULL**: When a column is defined as NOTNULL, then that column becomes a mandatory column. It implies that a value must be entered into the column if the record is to be accepted for storage in the table.

Command	Syntax	Description
NOT NULL	CREATE TABLE table_name (column_1 datatype CONSTRAINT constraint_name constraint_desc, column_2 datatype, column_3 datatype);	It is used to create a new table in a database and specify the name of the table and columns inside it

Syntax:

CREATE TABLE Table_Name (column_name data_type (size) NOT NULL,);

Example:

CREATE TABLE student (sno NUMBER(3)NOT NULL, name CHAR(10));

2. **UNIQUE**: The purpose of a unique key is to ensure that information in the column(s) is unique i.e. a value entered in column(s) defined in the unique constraint must not be repeated across the column(s). A table may have many unique keys.

Command	Syntax	Description

UNIQUE	CREATE TABLE Table_Name(column_name data_type(size) UNIQUE, ….);	

3. **CHECK**: Specifies a condition that each row in the table must satisfy. To satisfy the constraint, each row in the table must make the condition either TRUE or unknown (due to a null).

Command	Syntax	Description
CHECK	CREATE TABLE Table_Name(column_name data_type(size) CHECK(logical expression), ….);	

4. **PRIMARY KEY**: A field which is used to identify a record uniquely. A column or combination of columns can be created as primary key, which can be used as a reference from other tables. A table contains primary key is known as Master Table.

- It must uniquely identify each record in a table.
- It must contain unique values.
- It cannot be a null field.
- It cannot be multi-port field.
- It should contain a minimum no. of fields necessary to be called unique.

Command	Syntax	Description
PRIMARY KEY	CREATE TABLE Table_Name(column_name data_type(size) PRIMARY KEY, ….);	

5. **FOREIGN KEY**: It is a table level constraint. We cannot add this at column level. To reference any primary key column from other table this constraint can be used. The table in which the foreign key is defined is called a detail table. The table that defines the primary key and is referenced by the foreign key is called the master table.

Command	Syntax	Description
PRIMARY KEY	CREATE TABLE Table_Name(column_name data_type(size) PRIMARY KEY, ….);	
	ALTER TABLE Table_Name ADD PRIMARY KEY (column_name)	To assign an attribute as a primary key after a table is created without a primary key
	ALTER TABLE table_name ADD CONSTRAINT constraint_name PRIMARY KEY(colname)	

6. **DEFAULT:** The DEFAULT constraint is used to insert a default value into a column. The default value will be added to all new records, if no other value is specified.

Command	Syntax	Description
PRIMARY KEY	CREATE TABLE Table_Name(col_name1,col_name2,col_name3 DEFAULT '<value>');	

CREATE TABLE DEPARTMENT

(

Dname varchar(50) CONSTRAINT c1 UNIQUE ,

Dnumber INT CONSTRAINT c2 NOT NULL PRIMARY KEY,

Mgr_Id INT CONSTRAINT c3 NULL,

Mgr_start_date date

);

CREATE TABLE EMPLOYEE

(

Fname varchar(50),

Minit varchar(4),

Lname varchar(50),

Id INT CONSTRAINT c4 NOT NULL PRIMARY KEY,

Bdate date,

Address varchar(50),

Sex varchar(10)CONSTRAINT c5 CHECK(Sex in('male','female')),

Salary float,

Super_Id int,

Dno INT,

CONSTRAINT c7 FOREIGN KEY(Dno) REFERENCES DEPARTMENT(Dnumber)

);

CREATE TABLE DEPT_LOCATIONS

(

Dnumber INT CONSTRAINT c8 UNIQUE ,

Dlocation INT CONSTRAINT c9 NOT NULL PRIMARY KEY,

CONSTRAINT c10 FOREIGN KEY(Dnumber) REFERENCES DEPARTMENT(Dnumber)

);

CREATE TABLE PROJECT

(

Pname varchar(50),

Pnumber INT CONSTRAINT c13 NOT NULL PRIMARY KEY,

Plocation varchar(50),

Dnumber INT,

CONSTRAINT c14 FOREIGN KEY(Dnumber) REFERENCES DEPARTMENT(Dnumber)

);

CREATE TABLE WORKS_ON

```
(
EId INT,
Pno INT,
Hours INT,
CONSTRAINT  c15 FOREIGN KEY(Pno) REFERENCES PROJECT(Pnumber),
CONSTRAINT  c16 FOREIGN KEY(EId) REFERENCES EMPLOYEE(Id)
);
```

CREATE TABLE DEPENDENT

```
(
EId INT,
Dependent_name varchar(50) CONSTRAINT  c17 NOT NULL CONSTRAINT c18 PRIMARY KEY ,
Sex varchar(10),
Bdate Date,
Relationship varchar(20)  DEFAULT 'father',
CONSTRAINT c19 FOREIGN KEY(EId) REFERENCES EMPLOYEE(Id)
);
```

Description of the above tables

```
DESC EMPLOYEE;
DESC DEPARTMENT;
DESC PROJECT;
DESC DEPENDENT;
DESC WORKS_ON;
```

DESC DEPT_LOCATIONS;

2. Alter table command

ALTER TABLE

Command	Syntax	Description
ALTER table	ALTER TABLE table_name ADD column_name datatype;	It is used to add columns to a existing table in a database

2.1. Adding column to project table

Example: ALTER TABLE PROJECT ADD P_duration varchar(10);

2.2. Adding multiple columns

Example: ALTER TABLE PROJECT ADD (P_duration varchar(10), P_desc varchar(100));

2.3. Changing column width with modify clause

Command	Syntax	Description
ALTER table	ALTER TABLE table_name MODIFY (column_name datatype, column_name datatype..) ;	This is used to change the width as well as data type of fields of existing relations.

Example: ALTER TABLE PROJECT MODIFY P_duration char(200);

2.4. Removing a Column

Command	Syntax	Description
ALTER table	ALTER TABLE table_name ADD column_name datatype;	It is used to add columns to a existing table in a database

Example: ALTER TABLE project DROP COLUMN p_duration;

2.5. Renaming a Table

Command	Syntax	Description
ALTER table	ALTER TABLE table_name RENAME TO new_relation/table _name;	This is used to change the name of existing relations.
RENAME	RENAME TABLE old_relation_name TO new_relation_name;	: It is used to modify the name of the existing database object.

Example: ALTER TABLE employee RENAME TO emp;

RENAME TABLE emp TO employee;

2.6. Renaming a Column

Command	Syntax	Description
ALTER table	ALTER TABLE relation_name RENAME COLUMN (OLD column_name) to (NEW column_name);	This is used to change the name fields/columns of existing relations.

Example: ALTER TABLE employee RENAME COLUMN fname TO first_name;

2.7. Changing a Column's Datatype

Command	Syntax	Description
ALTER table	ALTER TABLE relation_name ALTER COLUMN colunm_name TYPE varchar(25);	This is used to change the name data types of existing relations.

Example: ALTER TABLE employee ALTER COLUMN Fname TYPE varchar(25);

2.8. Adding a CONSTRAINT

Command	Syntax	Description
ALTER table	ALTER TABLE table_name ADD CONSTRAINT constraint_name constraint_clause;	This is used to add constraint to existing relations.

Example: ALTER TABLE employee ALTER COLUMN fname SET NOT NULL;

2.9. Adding primary key CONSTRAINT

Command	Syntax	Description
ALTER table	ALTER TABLE relation_nameADD CONSTRAINT constraint_name PRIMARY KEY(column_name);	This is used to add primary key CONSTRAINT to existing relations.

Example: ALTER TABLE employee ADD CONSTRAINT c11 PRIMARY KEY (ID);

2.10. Removing primary key CONSTRAINT

Command	Syntax	Description
ALTER table	ALTER TABLE relation_name DROP CONSTRAINT constraint_name ;	This is used to remove primary key CONSTRAINT from existing relations.

Example: ALTER TABLE employee DROP CONSTRAINT c11;

2.11. Adding foreign key reference

Command	Syntax	Description
ALTER table	ALTER TABLE relation_nameADD CONSTRAINT constraint_name FOREIGN KEY(column_name) REFERENCING relation_name(column_name) ;	This is used to add foreign key CONSTRAINT to existing relations.

Example: ALTER TABLE employee ADD CONSTRAINT c11

FOREIGNKEY (Super_Id) REFEREEING employee (Id);

2.12. Removing a CONSTRAINT

Command	Syntax	Description
ALTER table	ALTER TABLE relation_nameDROP CONSTRAINT constraint_name;	This is used to remove constraint from relation.

Example: ALTER TABLE employee DROP CONSTRAINT c5;

3. Dropping Tables

Command	Syntax	Description
DROP table	DROP TABLE relation_name;	This is used to delete the structure of a relation. It permanently deletes the records in the table.

Example: DROP TABLE employee;

3.1. Dropping a primary key that has dependent table

Command	Syntax	Description
ALTER TABLE	ALTER TABLE relation_name DROP PRIMARY KEY CASCADE	This is used to drop a relation that has dependent table.

Example: ALTER TABLE employee DROP PRIMARY KEY CASCADE;

4. Create VIEW

In SQL, a view is a virtual table based on the result-set of an SQL statement. A view contains rows and columns, just like a real table. The fields in a view are fields from one or more real tables in the database. You can add SQL functions, WHERE, and JOIN statements to a view and present the data as if the data were coming from one single table. A view is a virtual table, which consists of a set of columns from one or more tables. It is similar to a table but it does not store in the database. View is a query stored as an object.

Command	Syntax	Description
CREATE VIEW	CREATE VIEW <view_name> AS SELECT <set of fields> FROM relation_name WHERE (Condition)	Used to create view from a relation.

Exercise

1. Write SQL query to create view which will display name and address of employees?

3. DROPPING A VIEW:

A view can deleted with the DROP VIEW command. Syntax: DROP VIEW ;

Command	Syntax	Description
DROP VIEW	DROP VIEW view_name	Used to drop a view.

LAB PRACTICE 1:

1. Create a university database
2. Inside the university database create a student relation with following schema:
   Student (<u>SID</u>, S_name, S_address, S_ph_no, Dept_no)
   Department (<u>Dept_code</u>, Dept_name, Dept_location)
   Course (<u>C_dode</u>,C_name, C_hr, C_desc)

Student_Course(C_code,S_ID)

3. Add a new column; E_email to the existing relation.
4. Change the datatype of S_address from char to varchar2.
5. Change the name of column/field SID to S_no.
6. Modify the column width of the S_address field of student table

Practice No. 2

Title: Implementation of DML commands of SQL with suitable examples

- Insert table
- Update table
- Delete Table

Objective:

- To understand the different issues involved in the design and implementation of a database system
- To understand and use data manipulation language to query, update, and manage a database

Theory : DATA MANIPULATION LANGUAGE (DML): DML is used to retrieve, insert and modify database information. These commands will be used by all database users during the routine operation of the database. Let's take a brief look at the basic DML commands: INSERT, UPDATE and DELETE.

1. **INSERT INTO**: This is used to add records into a relation.

Command	Syntax	Description
INSERT INTO	INSERT INTO < relation/table name> (field_1,field_2……field_n)VALUES (data_1,data_2,........data_n);	This is used to add records into a relation
	INSERT INTO < relation/table name> VALUES (data_1,data_2,........data_n);	
	INSERT INTO relation_name_1 SELECT Field_1,field_2,field_n FROM relation_name_2 WHERE field_x=data;	Inserting all records from another relation
	INSERT INTO relation_name field_1,field_2,.....field_n) VALUES (&data_1,&data_2,........&data_n);	Inserting multiple records from users

1.1. Data insertion example to DEPARTMENT TABLE

```
INSERT INTO DEPARTMENT(Dname,Dnumber,Mgr_Id,Mgr_start_date)
VALUES('Finince',1111,9090,'5/8/7');

INSERT INTO DEPARTMENT(Dname,Dnumber,Mgr_Id,Mgr_start_date) VALUES('Human
Resources',2222,8080,'5/6/9');

INSERT INTO DEPARTMENT(Dname,Dnumber,Mgr_Id,Mgr_start_date)
VALUES('Design',3333,7070,'8/4/7');

INSERT INTO DEPARTMENT(Dname,Dnumber,Mgr_Id,Mgr_start_date)
VALUES('Programming',4444,6060,'5/9/9');
```

1.2. Data insertion example to EMPLOYEE TABLE

```
INSERT   INTO   EMPLOYEE(Fname,Minit,Lname,id,Bdate,Address,Sex,Salary,Super_id,Dno)
VALUES('HENOK','AG','GETACHEW',9090,'5/9/1','YEKA','Male',125478,NULL,2222);

INSERT   INTO   EMPLOYEE(Fname,Minit,Lname,Id,Bdate,Address,Sex,Salary,Super_Id,Dno)
VALUES('Jemal','AB','Sultan',7070,'5/9/12','ARADA','Female',124576,NULL,4444);

INSERT INTO EMPLOYEE(Fname,Minit,Lname,Id,Bdate,Address,Sex,Salary,Super_Id,Dno)
VALUES('Yonas','MO','Teshome',9785,'9/9/8','BOLE','Male',NULL,,3333);

INSERT INTO
EMPLOYEE(Fname,Minit,Lname,Id,Bdate,Address,Sex,Salary,Super_Id,Dno)VALUES('Toma
s','TB','Belete’, 6060,'8/9/7','ARADA','Male',147891,7070,4444);

INSERT INTO EMPLOYEE(Fname,Minit,Lname,Id,Bdate,Address,Sex,Salary,Super_Id,Dno)
VALUES('Saron','SA','Ayalew',9069,'5/9/9','GULELE','Female',NULL,98754,1111);

INSERT INTO EMPLOYEE(Fname,Minit,Lname,Id,Bdate,Address,Sex,Salary,Super_Id,Dno)
VALUES('Robel','RG','Goitom',8080,'5/9/1','YEKA','Male',9090,98745,2222);

INSERT INTO EMPLOYEE(Fname,Minit,Lname,Id,Bdate,Address,Sex,Salary,Super_Id,Dno)
VALUES('yohannis','YS','Solomon',3270,'5/9/8','N.L','Male',NULL,45756,3333);

INSERT INTO EMPLOYEE(Fname,Minit,Lname,Id,Bdate,Address,Sex,Salary,Super_Id,Dno)
VALUES('Selam','SS','Surafel',6098,'8/9/6','ARADA','Male',7070,65987,4444);
```

1.3. Insert statement into DEPT_LOCATIONS table

```
INSERT INTO DEPT_LOCATIONS(Dnumber,Dlocation) VALUES(1111,65878);

INSERT INTO DEPT_LOCATIONS(Dnumber,Dlocation) VALUES(2222,78564);

INSERT INTO DEPT_LOCATIONS(Dnumber,Dlocation) VALUES(3333,365478);
```

INSERT INTO DEPT_LOCATIONS(Dnumber,Dlocation) VALUES(4444,89874);

1.4. Insert statement into PROJECT table

INSERT INTO PROJECT(Pname,Pnumber,Plocation,Dnumber) VALUES('Game Development',9999,'North Wing',1111);

INSERT INTO PROJECT(Pname,Pnumber,Plocation,Dnumber) VALUES('System Development',8888,'South Wing',2222);

INSERT INTO PROJECT(Pname,Pnumber,Plocation,Dnumber) VALUES('Unit Testing',7777,'West Wing',3333);

INSERT INTO PROJECT(Pname,Pnumber,Plocation,Dnumber) VALUES('Code Writing',6666,'North-East Wing',4444);

1.5. Insert statement into WORKS_ON table

INSERT INTO WORKS_ON(Essn,Pno,Hours) VALUES(9090,9999,35);

INSERT INTO WORKS_ON(Essn,Pno,Hours) VALUES(8080,8888,30);

INSERT INTO WORKS_ON(Essn,Pno,Hours) VALUES(7070,7777,35);

INSERT INTO WORKS_ON(Essn,Pno,Hours) VALUES(6060,6666,25);

1.6. Insert statement into DEPENDENT table

INSERT INTO DEPENDENT(Essn,Dependent_name,Sex,Bdate,Relationship) VALUES(9090,'Jenny Peru','Female','5/9/8','Sister');

INSERT INTO DEPENDENT(Essn,Dependent_name,Sex,Bdate,Relationship) VALUES(8080,'James Brown','Male','5/9/7','Brother');

INSERT INTO DEPENDENT(Essn,Dependent_name,Sex,Bdate,Relationship) VALUES(7070,'Peter Miller','Female','5/9/8','Daughter');

INSERT INTO DEPENDENT(Essn,Dependent_name,Sex,Bdate,Relationship) VALUES(6060,'Billy Rend','Male','5/9/8','Son');

2. UPDATE-SET-WHERE

Command	Syntax	Description
UPDATE-SET	UPDATE relation name SET Field_name1=data,field_name2=data;	This is used to update the **all** the content of a specific

		field record in a relation.
UPDATE-SET-WHERE	UPDATE relation name SET Field_name1=data,field_name2=data, WHERE field_name=data;	This is used to update the content of a record in a relation.

Example:

UPDATE employee SET Fname = 'Seble'

UPDATE employee SET Fname = 'Seble' WHERE ID=6098

3. **DELETE-FROM:** This is used to delete all the records of a relation but it will retain the structure of that relation.

Command	Syntax	Description
DELETE-FROM	DELETE FROM relation_name;	This is used to delete all the records of a relation but it will retain the structure of that relation
DELETE -FROM-WHERE	DELETE FROM relation_name WHERE condition;	This is used to delete a selected record from a relation.

Example

DELETE FROM employee;

DELETE FROM employee WHERE ID =6098;

TRUNCATE: This command will remove the data permanently. But structure will not be removed.

Difference between Truncate & Delete:-

- Using truncate command data will be removed permanently & will not get back where as by using delete command data will be removed temporally & get back by using roll back command.
- Using delete command data will be removed based on the condition whereas by using truncate command there is no condition.
- Truncate is a DDL command & delete is a DML command.

Command	Syntax	Description

TRUNCATE	TRUNCATE TABLE <Table name>	This is used remove the data permanently but it will retain the structure of that relation

Example: TRUNCATE TABLE employee;

LAB EXERCISE 2:

1. Insert data in to the tables that you have created for lab exercise 1?
2. Delete some records from student table?
3. Delete some records from course table based on a condition?
4. Change the name of DEPARTMENT relation to DEPT and back to DEPARTMENT.
5. Rename the SID attribute of the student relation to SNO ?
6. Delete all records of student?

Practice No. 3

SELECT FROM command

Command	Syntax	Description

Structured Query Language (SQL)

SELECT FROM	SELECT * FROM relation_name;	This is used to display all fields for all records.
	SELECT a set of fields (Field_name1, Field_name2, Field_name3) FROM relation_name;	To display a set of fields for all records of relation.
SELECT - FROM - WHERE	SELECT a set of fields FROM relation_name WHERE condition;	This query is used to display a selected set of fields for a selected set of records of a relation.

Example

1. **Write SQL query to retrieve department information?**

SQL	Retional Algebra statement
SELECT * FROM Department ;	$\Pi_{Pname,Pnumber,Plocation,Dnumber}(Department)$

2. **Write SQL query to Retrieve department name**

SQL	Retional Algebra statement
SELECT Dname FROM Project;	$\Pi_{Dname}(Department)$

3. **Retrieve firstname, lastname and ssn of employee?**

SQL	Retional Algebra statement
SELECT Fname, Lname, SSN FROM employee;	$\Pi_{Fname,\ Lname,\ SSN}$ (Employee)

4. **Write SQL statement to retrieve all employees Ssn?**

SQL	Retional Algebra statement
SELECT SSN FROM employee;	Π_{SSN} (Employee)

5. **Write SQL statement to retrieve address of an employee?**

SQL	Retional Algebra statement
SELECT Address FROM Employee;	$\Pi_{Address}$ (Employee)

6. **Retrieve distinct values for address of employee?**

SQL	**Retional Algebra statement**
SELECT **DISTINCT** Address FROM Employee;	Π_{Address} (Employee)

Practice No: 4

Title: Implementation of different types of operators in SQL.

- Arithmetic Operator
- Logical Operator
- Comparison Operator
- Special Operator
- Set Operator

Objective: To learn different types of operator.

Theory:

ARIHMETIC OPERATORS:

(+): Addition - Adds values on either side of the operator.

(-): Subtraction - Subtracts right hand operand from left hand operand.

(*): Multiplication - Multiplies values on either side of the operator.

(/): Division - Divides left hand operand by right hand operand.

(^): Power- raise to power of.

(**%**): Modulus - Divides left hand operand by right hand operand and returns remainder.

LOGICAL OPERATORS:

AND: The AND operator allows the existence of multiple conditions in an SQL statement's WHERE clause.

OR: The OR operator is used to combine multiple conditions in an SQL statement's WHERE clause.

NOT: The NOT operator reverses the meaning of the logical operator with which it is used. Eg: NOT EXISTS, NOT BETWEEN, NOT IN, etc. This is a negate operator.

COMPARISION OPERATORS:

(=): Checks if the values of two operands are equal or not, if yes then condition becomes true.

(**!** =): Checks if the values of two operands are equal or not, if values are not equal then condition becomes true.

(< >): Checks if the values of two operands are equal or not, if values are not equal then condition becomes true.

(>): Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true

(<): Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true.

(<=): Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true.

SPECIAL OPERATOR:

BETWEEN: The BETWEEN operator is used to search for values that are within a set of values, given the minimum value and the maximum value.

IS NULL: The NULL operator is used to compare a value with a NULL attribute value.

ALL: The ALL operator is used to compare a value to all values in another value set

ANY: The ANY operator is used to compare a value to any applicable value in the list according to the condition.

LIKE: The LIKE operator is used to compare a value to similar values using wildcard operators. It allows to use percent sign (%) and underscore (_) to match a given string pattern.

Command	Syntax	Description

LIKE	SELECT column_name(s) FROM table_name WHERE column_name LIKE pattern;	It is an special operator used with the WHERE clause to search for a specific pattern in a column

Exercise

1. Retrieve all employees whose address is begins with A?
2. Find all employees who were born during the 1950s ?
3. Retrieve all employees whose name is begins with S?

IN: The IN operator is used to compare a value to a list of literal values that have been specified.

Command	Syntax	Description
LIKE	SELECT column_name(s) FROM table_name WHERE column_name LIKE pattern;	It is an special operator used with the WHERE clause to search for a specific pattern in a column

EXIST: The EXISTS operator is used to search for the presence of a row in a specified table that meets certain criteria.

1. **Write SQL query to retrieve department number for human resource department?**

SQL	Retional Algebra statement

SELECT Dnumber
FROM Department
Where Dname='Human Resource';
	$\Pi_{\text{Dnumber}}(\delta_{\text{Dname='Human Resource'}}(\text{Department}))$

2. **Retrieve list of employee whose salary is more than 1000000 ?**

SQL	Retional Algebra statement

SELECT *
FROM
Where Salary >1000000;
	$\Pi_{\text{Fname,Minit,Lname,Id,Bdate,Address,Sex,Salary,Super\_Id,Dno}}(\delta_{\text{Salary> 1000000}}(\text{Employee}))$

3. **Retrieve list of employee whose salary is between 10000and 400000?**

SQL	Retional Algebra statement
SELECT *	$\Pi_{\text{Fname,Minit,Lname,Id,Bdate,Address,Sex,Salary,Super\_Id,Dno}}$

FROM
Where Salary >10000
AND 40000;
	($\delta_{\text{Salary> 10000 and Salary<400000}}$ (Employee))

SELECT *
FROM EMPLOYEE
WHERE (Salary
BETWEEN 10000 AND
40000)	

7. **Write SQL query to retrieve first name, last name and ID of supervisee of Henok?**

SQL	Retional Algebra statement

SELECT Fname,Lname,ID
FROM Employee
Where SuperID=(SELECT ID
FROM employee WHERE
Fname='Henok');
	R $\leftarrow \Pi_{\text{ID}}(\delta_{\text{Fname='Henok'}}$ (Employee))
	$\Pi_{\text{Fname,Lname,ID}}$ (R)

8. **Write SQL statement to retrieve the birth date and address of the employee(s) whose name is 'John ws. Smith'?**

SQL	Retional Algebra statement

SELECT Bdate, Address
FROM EMPLOYEE
WHERE Fname = 'Shrafel'
AND Minit = 'SH' AND
Lname = 'Habib';
	R $\leftarrow \delta_{\text{Fname = 'Shrafel' AND Minit = 'SH' AND Lname = 'Habib'}}$ (Employee)
	Π Bdate, Address (R)

9. **Retrieve all employees who work in department 5?**

SQL	Rational Algebra statement

SELECT *
FROM Employee
WHERE Dno = 5;
	R $\leftarrow \delta_{\text{Dno=5}}$ (Employee)
	Π Fname,Minit,Lname,Id,Bdate,Address,Sex,Salary,Super_Id,Dno (R)

10. **Retrieve the name and address of all employees who work for the human resource department.**

SQL	Rational Algebra statement

SELECT
Fname,Lname,Minit,
Address
FROM Employee	

Where DNO=(SELECT Dno FROM Department WHERE Dname='human resource');	

Practice No: 5

Title: Implementation of different types of Joins

- Inner Join
- Outer Join
- Natural Join..etc

Objective: To implement different types of joins

Theory: The SQL Joins clause is used to combine records from two or more tables in a database. A JOIN is a means for combining fields from two tables by using values common to each.The join is actually performed by the 'where' clause which combines specified rows of tables. Types of Joins: Simple Join, Self Join and Outer Join

Command	Syntax	Description
Join	SELECT column 1, column 2, column 3... FROM table_name1, table_name2 WHERE table_name1.column name = table_name2.columnname;	used to combine records from two or more tables in a database

Simple Join: It is the most common type of join. It retrieves the rows from 2 tables having a common column and is further classified into
Equi-join: A join, which is based on equalities, is called equi-join.
Example: Select * from Department, Employee where Department.Dnumber=Employee.Dno; In the above statement, Department.Dnumber=Employee.Dno performs the join statement. It retrieves rows from both the tables provided they both have the same id as specified by the where clause. Since the where clause uses the comparison operator (=) to perform a join, it is said to be equijoin. It combines the matched rows of tables. It can be used as follows:

- To insert records in the target table.
- To create tables and insert records in this table.
- To update records in the target table.
- To create views.

Non Equi-join: It specifies the relationship between columns belonging to different tables by making use of relational operators other than'='.

Table Aliases Table aliases are used to make multiple table queries shorted and more readable. We give an alias name to the table in the 'from' clause and use it instead of the name throughout the query.

Self-join: Joining of a table to itself is known as self-join. It joins one row in a table to another. It can compare each row of the table to itself and also with other rows of the same table. Example: select * from employee x , employee y where x.salary >= (select avg(salary) from x where x. dno =y.dnumber);

Outer Join: It extends the result of a simple join. An outer join returns all the rows returned by simple join as well as those rows from one table that do not match any row from the table. The symbols (+) represents outer join.

- Left outer join

- Right outer join
- Full outer join

1. Write SQL statement to retrieve employees who works on Code Writing project?

SQL	Rational Algebra statement
SELECT Fname,Minit,Lname,Id,Bdate,Address,Sex,Salary,Super_Id,Dno Address FROM Employee E, WORKS_ON W, Project p Where E.ID=W.EID and W.Pid=W.Pnumber and P.Pname='Code Writing';	$\rho_{E(Employee)}$ $\rho_{W(WORKS\_ON)}$ $\rho_{P(Project)}$ R← EXWXP S←$\delta_{E.ID=W.EID\ AND\ W.Pid=W.Pnumber\ AND\ P.Pname='Code\ Writing'}$ (R) Π Fname,Minit,Lname,Id,Bdate,Address,Sex,Salary,Super_Id,Dno(S)

2. Retrieve the name and address of all employees who work for the human resource department.

SQL	Rational Algebra statement
SELECT Fname,Lname,Minit, Address FROM Employee E, Department DWHERE E.DNO=D.Dnumber AND Dname='human resource';	$\rho_{E(Employee)}$ $\rho_{D(Department)}$ R← EXD S←δ E.DNO=D.Dnumber AND Dname='human resource' (R) Π Fname,Lname,Minit, Address(R)

Practice Exercise

1. Write SQL statement to retrieve dependents of peter?
2. Write SQL statement to retrieve departments located in the same location?
3. Write SQL statements to retrieve managers of programming department?

4. For every project located in ‘west wing’, write SQL statement to list the project number, the controlling department number, and the department manager’s last name, address, and birth date.
5. For each employee, retrieve the employee’s first and last name and the first and last name of his or her immediate supervisor.
6. Make a list of all project numbers for projects that involve an employee whose last name is ‘habib’, either as a worker or as manager of the department that controls the project.

Practice No: 5

Aggregative operators: In addition to simply retrieving data, we often want to perform some computation or summarization. SQL allows the use of arithmetic expressions. We now consider a powerful class of constructs for computing aggregate values such as MIN and SUM.

Structured Query Language (SQL)

1. **Count:** COUNT following by a column name returns the count of tuple in that column. If DISTINCT keyword is used then it will return only the count of unique tuple in the column. Otherwise, it will return count of all the tuples (including duplicates)

Command	Syntax	Description
COUNT	SELECT COUNT(column_name), FROM table_name, WHERE condition	the count of tuple in that column

Example:

1.1. Write SQL query to retrieve number of employees

SQL	Rational Algebra statement
SELECT COUNT (*) FROM Employee;	$ɕ_{\text{COUNT (*)}}$(Employee)

1.2. Write SQL query to retrieve unique address of employee

SQL	Rational Algebra statement
SELECT DISTINCT COUNT (Address) FROM Employee;	

2. **SUM:** SUM followed by a column name returns the sum of all the values in that column.

Command	Syntax	Description
SUM	SELECT SUM(column_name), FROM table_name, WHERE condition	sum of all the values in that column

Example

2.1. Write SQL query to retrieve total salary of employees

SQL	Rational Algebra statement
SELECT COUNT (salary) FROM Employee;	$ɕ_{\text{sum(Salary)}}$(Employee)

3. **AVG**: AVG followed by a column name returns the average value of that column values.

Command	Syntax	Description
SUM	SELECT AVG(column_name) FROM	returns the average value of

	table_name, WHERE condition	that column values

Example

3.1. Write SQL query to retrieve average salary of employees ?

SQL	Rational Algebra statement
SELECT AVG (salary) FROM Employee;	$\varsigma_{AVG(Salary)}$(Employee)

4. **MAX**: MAX followed by a column name returns the maximum value of that column.

Command	Syntax	Description
MAX	SELECT MAX(column_name) FROM table_name, WHERE condition	returns the maximum value of that column values

Example

4.1. Write SQL query to retrieve average salary of employees ?

SQL	Rational Algebra statement
SELECT MAX (salary) FROM Employee;	$\varsigma_{MAX(Salary)}$(Employee)

5. **MIN**: MIN followed by a column name returns the maximum value of that column.

Command	Syntax	Description
MIN	SELECT MIN(column_name) FROM table_name, WHERE condition	returns the minimum value of that column values

Example

1.1. Write SQL query to retrieve minimum salary of employees ?

SQL	Rational Algebra statement
SELECT MIN (salary) FROM Employee;	$\varsigma_{MIN(Salary)}$(Employee)

Exercise

1. Write SQL query to retrieve an employee who earns the largest salary?
2. Write SQL query to retrieve an employee who earns the largest salary?

Practice No: 6

Title: Study Implementation of Group by & Having Clause and Order by Clause.

Objective: To learn the concept of group functions Theory:

1. **GROUP BY**: This query is used to group to all the records in a relation together for each and every value of a specific key(s) and then display them for a selected set of fields the relation.

Command	Syntax	Description
GROUP BY	SELECT column_name, COUNT(*) FROM	It is an clause in SQL used for

	table_name GROUP BY column_name;	aggregate functions in collaboration with the SELECT statement

Example

1. Write SQL query to retrieve total number of employees in each department?

SQL	Rational Algebra statement
SELECT COUNT (*) , Dno FROM Employee GROUP BY Dno;	

Exercise

1. Write SQL query to retrieve maximum, minimum, and average salary of employees in each department?
2. Write SQL query to retrieve maximum, minimum, and average salary of employee's human resource in department?

2. **HAVING**: The HAVING clause was added to SQL because the WHERE keyword could not be used with aggregate functions. The HAVING clause must follow the GROUP BY clause in a query and must also precedes the ORDER BY clause if used.

Command	Syntax	Description
HAVING	SELECT column_name, aggregate_function (column_name) FROM table_name WHERE column_name GROUP BY column_name HAVING aggregate_function(column_name) operator value;	It is an clause in SQL used for aggregate functions in collaboration with the SELECT statement

Exercise

1. Write SQL query to retrieve employees who has more than two dependents?
2. Write SQL query to retrieve employees who works on more than one project ?

3. **ORDER BY:** This query is used to display a selected set of fields from a relation in an ordered manner base on some field.

Command	Syntax	Description
ORDER BY	SELECT column_name FROM table_name ORDER BY column_name ASC | DESC;	It is a clause used to sort the result set by a particular column either numerically or

		alphabetically based on some field

Exercise

1. Write SQL query to retrieve employees name in ascending and descending order?
2. Write SQL query to retrieve employees salary ascending and descending order?

Practice No: 7

Title : Study & Implementation of

- Sub queries

Objective: – To perform nested Queries and joining Queries using DML command – To understand the implementation of views.

Theory: SUBQUERIES: The query within another is known as a sub query. A statement containing sub query is called parent statement. The rows returned by sub query are used by the parent statement or in other words A subquery is a SELECT statement that is embedded in a clause of another SELECT statement You can place the subquery in a number of SQL clauses: WHERE clause, HAVING clause, FROM clause, OPERATORS(IN.ANY,ALL,,>=,<= etc..)

Types

1. **Sub queries that return several values**

Sub queries can also return more than one value. Such results should be made use along with the operators in and any.

2. **Multiple queries**

Here more than one sub query is used. These multiple sub queries are combined by means of ‘and’ & ‘or’ keywords.

3. **Correlated sub query**

A sub query is evaluated once for the entire parent statement whereas a correlated Sub query is evaluated once per row processed by the parent statement.

Practice NO.8

SET OPERATORS

The Set operator combines the result of 2 queries into a single result. The following are the operators:

- Union
- Union all
- Intersect
- Minus

Union: Returns all distinct rows selected by both the queries Union all: Returns all rows selected by either query including the duplicates.

Intersect: Returns rows selected that are common to both queries.

Minus: Returns all distinct rows selected by the first query and are not by the second

Example:

1. Make a list of all project numbers for projects that involve an employee whose last name is 'Habib', either as a worker or as manager of the department that controls the project.

```
(SELECT DISTINCT Pnumber FROM PROJECT, DEPARTMENT, EMPLOYEE WHERE
Dnum = Dnumber AND Mgr_ssn = Ssn AND Lname = 'Habib' )

UNION

( SELECT DISTINCT Pnumber

FROM PROJECT, WORKS_ON, EMPLOYEE

WHERE Pnumber = Pno AND Essn = Ssn AND Lname = 'Habib' );
```

Exercise

1. Write SQL query to retrieve employees who participate in game development project only?
2. Write SQL query to retrieve employees who participate in game development and unit testing project only?