**EXNO :1**

**DATA DEFINITION LANGUAGE COMMANDS**

**AIM:**

To write a program for SQL Data Definition Language Commands on sample exercise

**PROCEDURE:**

**Data Definition Commands:**

DDL or Data Definition Language actually 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 database.

**Examples of DDL commands:**

* [**CREATE**](https://www.geeksforgeeks.org/sql-create/) – is used to create the database or its objects (like table, index, function, views, store procedure and triggers).
* [**DROP**](https://www.geeksforgeeks.org/sql-drop-truncate/) – is used to delete objects from the database.
* [**ALTER**](https://www.geeksforgeeks.org/sql-alter-add-drop-modify/)-is used to alter the structure of the database.
* [**TRUNCATE**](https://www.geeksforgeeks.org/sql-drop-truncate/)–is used to remove all records from a table, including all spaces allocated for the records are removed.
* [**COMMENT**](https://www.geeksforgeeks.org/sql-comments/) –is used to add comments to the data dictionary.
* [**RENAME**](https://www.geeksforgeeks.org/sql-alter-rename/) –is used to rename an object existing in the database.

**1 - i) CREATE TABLE**

It is used to create a table.

**Rules:**

1. Oracle reserved words cannot be used.
2. Underscore, numerals, letters are allowed but not blank space.
3. Maximum length for the table name is 30 characters.
4. Different tables should not have same name.
5. We should specify a unique column name.
6. We should specify proper data type along with width.

**Syntax:**

**SQL>**Create table tablename (column\_name1 data\_ type constraints, column\_name2 data\_ type constraints …);

**ii) DESC**

This is used to view the structure of the table.

**Syntax:**

**SQL>**desc tablename;

**iii) CREATING NEW TABLE FROM EXISTING TABLE:**

**Syntax:**

CREATE TABLE new\_table\_name AS SELECT column1, column2,... FROM existing\_table\_name WHERE ....;

**2- DROP TABLE**

It will delete the table .

**Syntax:**

**SQL>**DROP TABLE <TABLENAME>;

**3- ALTER COMMAND**

Alter command is used to:

1. Add a new column.

2. Modify the existing column definition.

3. To include or drop integrity constraint.

**i) - ADD COMMAND**

Add the new column to the existing table.

**Syntax:**

**SQL>**alter table tablename add/modify (attribute datatype(size));

**ii) - MODIFY COMMAND**

Modify the existing column definition.

**Syntax :**

**SQL>**alter table <tablename> modify(columnname constraint);

**SQL>**alter table <tablename>modify(columnnamedatatype);

**4- TRUNCATE TABLE**

If there is no further use of records stored in a table and the structure has to be retained then the records alone can be deleted.

**Syntax:**

**SQL>**TRUNCATE TABLE <TABLE NAME>;

**5-** [**COMMENT**](https://www.geeksforgeeks.org/sql-comments/)

Comments can be written in the following three formats:

1. Single line comments.
2. Multi line comments
3. In line comments

* **Single line comments:** Comments starting and ending in a single line are considered as single line comments.  
  Line starting with ‘–‘ is a comment and will not be executed.

**Syntax:**

--single line comment

--another comment

* **Multi line comments:**Comments starting in one line and ending in different line are considered as multi line comments. Line starting with ‘/\*’ is considered as starting point of comment and are terminated when ‘\*/’ is encountered.

**Syntax:**

/\* multi line comment

another comment \*/

* **In line comments:**In line comments are an extension of multi line comments, comments can be stated in between the statements and are enclosed in between ‘/\*’ and ‘\*/’.

**Syntax:**

**SQL>**SELECT \* FROM /\* table name; \*/

**6- RENAME**

If you want to change the name of the table in the SQL database because they want to give a more relevant name to the table. Any database user can easily change the name by using the RENAME TABLE and ALTER TABLE statement in Structured Query Language.

**Syntax:**

**SQL>**RENAME old\_table \_name To new\_table\_name ;

**Program:**

SQL> connect

Enter user-name: system

Enter password: admin

Connected.

SQL> create table emp(id number(10),name varchar(10));

Table created.

SQL> desc emp;

Name Null? Type

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

ID NUMBER(10)

NAME VARCHAR2(10)

SQL> alter table emp add(dept varchar(10));

Table altered.

SQL> desc emp;

Name Null? Type

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

ID NUMBER(10)

NAME VARCHAR2(10)

DEPT VARCHAR2(10)

SQL> alter table emp modify dept varchar(20);

Table altered.

SQL> desc emp;

Name Null? Type

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

ID NUMBER(10)

NAME VARCHAR2(10)

DEPT VARCHAR2(20)

SQL> alter table emp drop column dept;

Table altered.

SQL> desc emp;

Name Null? Type

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

ID NUMBER(10)

NAME VARCHAR2(10)

SQL> alter table emp rename to emp1;

Table altered.

SQL> desc emp1;

Name Null? Type

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

ID NUMBER(10)

NAME VARCHAR2(10)

SQL> desc emp2;

Name Null? Type

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

ID NUMBER(10)

NAME VARCHAR2(10)

DEPT VARCHAR2(10)

SQL> drop table emp2;

Table dropped.

SQL> select \* from emp2;

select \* from emp2

\*

ERROR at line 1:

ORA-00942: table or view does not exist

SQL> select \* from emp1;

ID NAME DEPT

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

1 aaa cse

2 aaa cse

3 aaa ece

4 aaa cse

5 aaa cse

SQL> truncate table emp1;

Table truncated.

SQL> select \* from emp1;

no rows selected

SQL> desc emp1;

Name Null? Type

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

ID NUMBER(10)

NAME VARCHAR2(10)

DEPT VARCHAR2(10)

SQL> drop table emp1;

Table dropped.

SQL> select \* from emp1;

select \* from emp1

\*

ERROR at line 1:

ORA-00942: table or view does not exist

SQL> desc emp1;

ERROR:

ORA-04043: object emp1 does not exist

**Result:**

Hence, the above query has been implemented successfully.

**EX.NO:2**

**DATE:**

**DATA MANIPULATION COMMANDS FOR INSERTING, DELETING, UPDATING AND RETRIEVING TABLES**

**AIM:**

To create a database using Data Manipulation Commands for inserting, deleting, updating and retrieving tables and Transaction Control statements.

**DESCRIPTION:**

**Data Manipulation Language:**

Data manipulation language (DML) statements access and manipulate data in existing tables. DML commands are the most frequently used SQL commands and is used to query and manipulate the existing database objects. Some of the commands are Insert, Select, Update, Delete.

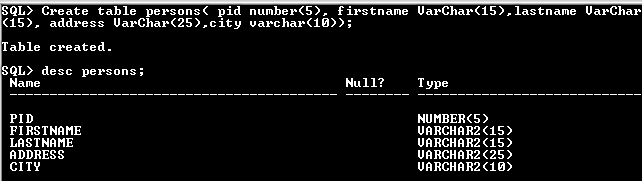
**Examples of DML:**

1. **Insert Command:** This is used to add one or more rows to a table. The values are separated by commas and the data types char and date are enclosed in apostrophes. The values must be entered in the same order as they are defined.
2. **Select Commands:** It is used to retrieve information from the table. It is generally referred to as querying the table. We can either display all columns in a table or only specify column from the table.
3. **Update Command:** It is used to alter the column values in a table. A single column may be updated or more than one column could be updated.
4. **Delete command:** After inserting row in a table we can also delete them if required. The delete command consists of a from clause followed by an optional where clause.

**INSERTING VALUES INTO TABLE**

**Create table:**

SQL>Create table persons( pid number(5), firstnameVarChar(15),lastnameVarChar(15), address VarChar(25),city varchar(10));



**INSERT COMMAND**

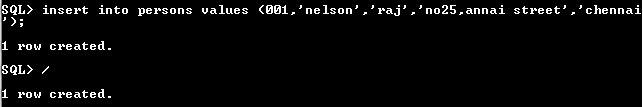
Insert command is used to insert values into table.

**Insert a single record into table.**

**Syntax:**SQL>insert into <table name> values (value list)

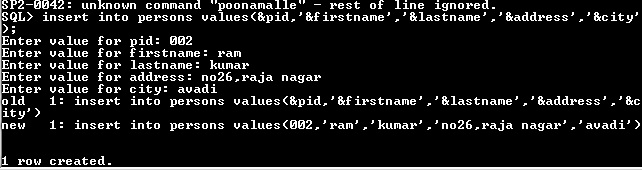
SQL>insert into persons values (001,'nelson','raj','no25,annai street','chennai');

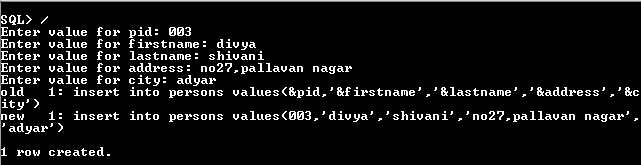
1 row created.

****

**Insert more than a record into persons table using a single insert command.**

SQL> insert into persons values(&pid,'&firstname','&lastname','&address','&city');





**Skipping the fields while inserting:**

SQL> insert into persons(pid,firstname) values(500,'prabhu');



**SELECT COMMAND**

It is used to retrieve information from the table. It is generally referred to as querying the

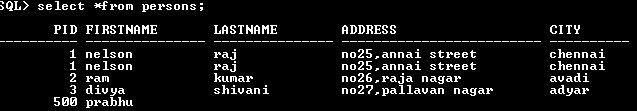
table. We can either display all columns in a table or only specify column from the table.

**Syntax:**

SQL> Select \* from tablename; // This query selects all rows from the table.

**Example:**

SQL>Select \* from persons;



**THE RETRIEVAL OF SPECIFIC COLUMNS FROM A TABLE:**

It retrieves the specified columns from the table

**Syntax:** SQL>Select column\_name1, …..,column\_name N from table name;

**Example:** SQL>Select pid, firstname from persons;



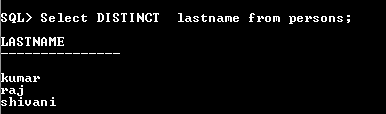
**Elimination of duplicates from the select clause:**

It prevents retrieving the duplicated values .Distinct keyword is to be used.

**Syntax:** SQL>Select DISTINCT col1, col2 from table name;

**Example:**

SQL>Select DISTINCT lastname from persons;



**SELECT COMMAND WITH WHERE CLAUSE:**

To select specific rows from a table we include ‘where’ clause in the select command. It

can appear only after the ‘from’ clause.

**Syntax:** SQL>Select column\_name1, …..,column\_name N from table name where condition;

**Example:** SQL>Select firstname, lastname from persons where pid>2;



**Select command with order by clause:**

**Syntax:** SQL>Select column\_name1, …..,column\_namen from table name where condition orderbycolmnname;

**Example:**

SQL>Select firstname, lastname from persons order by pid;



**Select command to create a table:**

**Syntax:**

SQL>create table tablename as select \* from existing\_tablename;

**Example:**

SQL>create table persons1 as select \* from persons;

Table created

**SELECT COMMAND TO INSERT RECORDS:**

**Syntax:**SQL>insert into tablename( select columns from existing\_tablename);

**Example:**SQL>insert into persons1( select \* from persons);

PID FIRSTNAME LASTNAMEADDRESS CITY PHONENO

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

001 nelson raj no25,annai street Chennai

100 niranjan kumar 10/25 krishna street Mumbai 999999999

102 arjun kumar 30 sundaram street coimbatore

300 gugan chand 5/10 mettu street Coimbatore

500 prabhu

**SELECT COMMAND USING IN KEYWORD:**

**Syntax:**SQL>Select column\_name1, …..,column\_namen from table name where colmnname IN (value1,value2);

**Example:** SQL>Select \* from persons where pid in (100,500);

(OR)

SQL>Select \* from persons where (pid=100 OR pid=500);

PID FIRSTNAME LASTNAMEADDRESS CITY PHONENO

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

100 niranjan kumar 10/25 krishna street Mumbai 999999999

500 prabhu

**SELECT COMMAND USING BETWEEN KEYWORD:**

**Syntax:**SQL>Select column\_name1, …..,column\_namen from table name where colmnname BETWEEN value1 AND value2;

**Example:** SQL>Select \* from persons where pid between 100 and 500;

PID FIRSTNAME LASTNAMEADDRESS CITY PHONENO

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

100 niranjan kumar 10/25 krishna street Mumbai 999999999

500 prabhu

**SELECT COMMAND USING PATTERN:**

**Syntax:**SQL>Select column\_name1, …..,column\_namen from table name where colmnname LIKE ‘% or \_‘;

**Example:** SQL>Select \* from persons where firstname like ‘nir\_n%’;

PID FIRSTNAME LASTNAMEADDRESS CITY PHONENO

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

100 niranjan kumar 10/25 krishna street Mumbai 999999999

**RENAMING THE FIELDNAME AT THE TIME OF DISPLAY USING SELECT STATEMENT:**

**Syntax:**SQL>Select old\_column\_namenew\_column\_namefrom table name where condition;

**Example:** SQL>Select pidpersonid from persons;

**SELECT COMMAND TO RETRIEVE NULL VALUES:**

**Syntax:**SQL>Select column\_name from table name where column\_name is NULL ;

**Example:** SQL>Select \* from persons where lastname is null;

PID FIRSTNAME LASTNAMEADDRESS CITY PHONENO

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

500 prabhu

**UPDATE COMMAND:**

**Syntax:**

UPDATE table\_nameSET column\_name = value [, column\_name = value]...

[ WHERE condition ];

**Example:**SQL>update persons set pid= 5 where firstname=’prabhu’;

Table updated.

**DELETE COMMAND**

**Syntax:** SQL>Delete from table where conditions;

**Example:**SQL>delete from persons where pid=500;

**1 row deleted.**

**RESULT:**

Thus the database has been created and the data has been inserted, deleted, modified, altered, updated and records are viewed based on conditions.

**EX.NO:3**

**DATE:**

**DATA CONTROL LANGUAGE COMMANDS AND TRANSACTION CONTROL COMMANDS**

**AIM:**

To create a database using Data Control Commands and Transaction Control Commands tomanage transactions in the database.

**DESCRIPTION:**

**Transaction Control statements**

**Transaction Control Language** (TCL) commands are used to managetransactions in the database. These are used to manage the changes made by DML-statements. It also allows statements to be grouped together into logical transactions.

**Examples of TCL:**

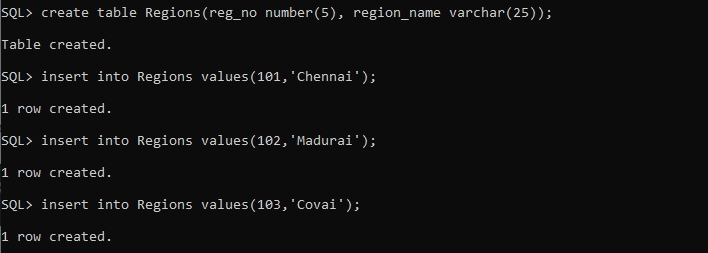
(i) Commit

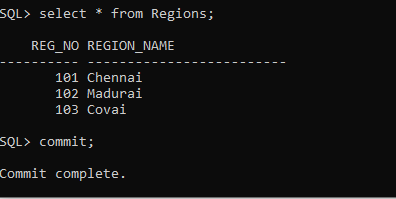
(ii) Rollback

(iii) Savepoint

**(i)Commit:** Commit command saves all the work done.

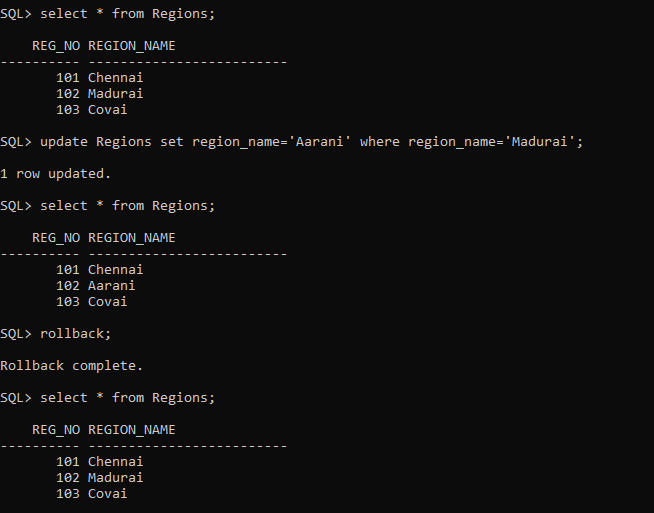
**Syntax:** commit;





**(ii)Rollback**: Rollback Command restores database to original since the last Commit.

**Syntax:** ROLLBACK TO SAVEPOINT <savepoint\_name>;

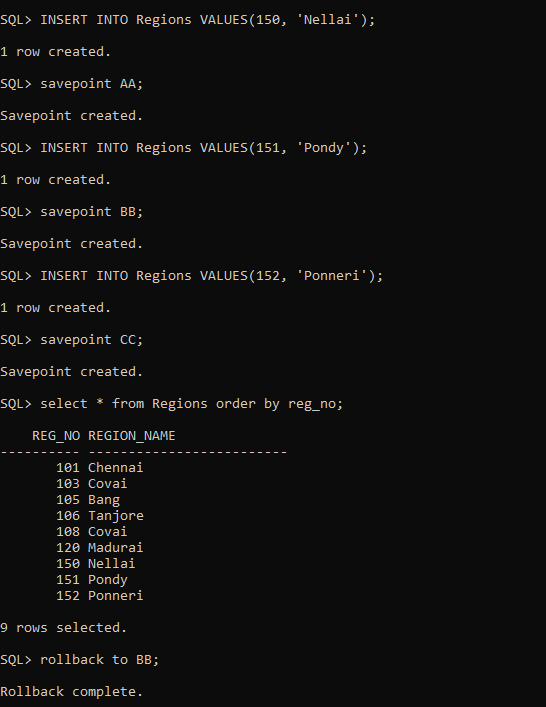


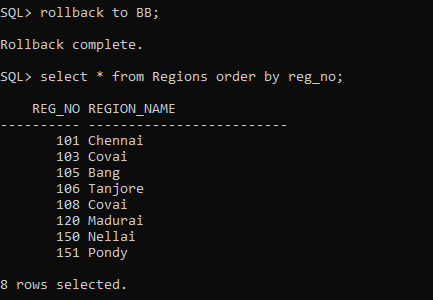
**(iii) Savepoint:**

SAVEPOINT command is used to temporarily save a transaction so that you can rollback to that point whenever required.

In short, using this command we can name the different states of our data in any table and then rollback to that state using the ROLLBACK command whenever required.

**Syntax:**SAVEPOINT <savepoint\_name>;

****



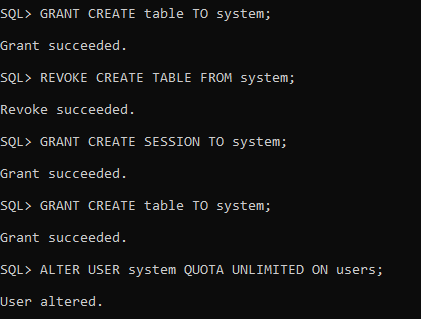
**Data Control Language**

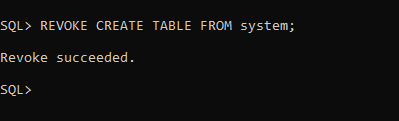
Data Control Language (DCL) is used to control privileges in Database. To perform any operation in the database, such as for creating tables, sequences or views, a user needs privileges. Privileges are of two types,

* **System:** This includes permissions for creating session, table, etc and all types of other system privileges.
* **Object:** This includes permissions for any command or query to perform any operation on the database tables.

In DCL we have two commands,

* **GRANT:** Used to provide any user access privileges or other privileges for the database.
* **REVOKE:** Used to take back permissions from any user.





**RESULT:**

Thus the Data Control Language commands and Transaction Control Language commands were executed successfully.

**EX.NO:4 SQL FUNCTIONS**

**AIM**

To study the various SQL Functions operations on the database.

**DESCRIPTION:**

**What are functions?**

Functions are methods used to perform [data operations](https://www.edureka.co/blog/sql-operators/). SQL has many in-built functions used to perform string concatenations, mathematical calculations etc.

SQL functions are categorized into the following two categories:

1. Aggregate Functions
2. Scalar Functions

Let us look into each one of them, one by one.

**AGGREGATE SQL FUNCTIONS**

The Aggregate Functions in SQL perform calculations on a group of values and then return a single value. Following are a few of the most commonly used Aggregate Functions:

|  |  |
| --- | --- |
| **Function** | **Description** |
| SUM() | Used to return the sum of a group of values. |
| COUNT() | Returns the number of rows either based on a condition, or without a condition. |
| AVG() | Used to calculate the average value of a numeric column. |
| MIN() | This function returns the minimum value of a column. |
| MAX() | Returns a maximum value of a column. |
| FIRST() | Used to return the first value of the column. |
| LAST() | This function returns the last value of the column. |

**SCALAR SQL FUNCTIONS**

The Scalar Functions in SQL are used to return a single value from the given input value.  Following are a few of the most commonly used Aggregate Functions:

|  |  |
| --- | --- |
| **Function** | **Description** |
| LCASE() | Used to convert string column values to lowercase |
| UCASE() | This function is used to convert a string column values to Uppercase. |
| LEN() | Returns the length of the text values in the column. |
| MID() | Extracts substrings in SQL from column values having String data type. |
| ROUND() | Rounds off a numeric value to the nearest integer. |
| NOW() | This function is used to return the current system date and time. |
| FORMAT() | Used to format how a field must be displayed. |

**EXAMPLE:**

**CHARACTER/STRING FUNCTION:**

SQL> select upper('welcome') from dual;

SQL> select upper('hai') from dual;

SQL> select lower('HAI') from dual;

SQL> select initcap(‘hello world') from dual;

SQL> select ltrim('hello world',’hell’) from dual;

SQL> select rtrim(''hello world',’ld’')from dual;

SQL> select concat('SRM',' University')from dual;

SQL> select length('Welcome’) from dual;

SQL> select replace('SRM University', 'University','IST')from dual;

SQL> select lpad('SRM University',20,'\*')from dual;

SQL> select rpad('SRM University',15,'$')from dual;

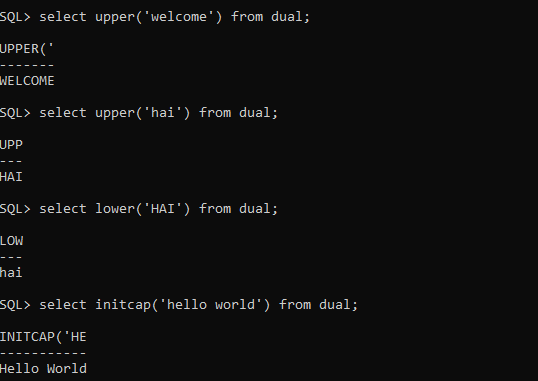
SQL> select substr('Welcome to SRM University', 4,7)from dual;

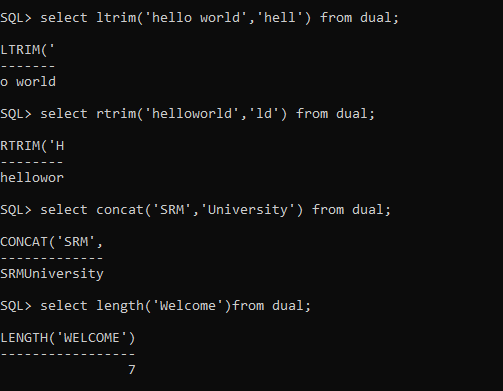
SQL> select replace('COMPUTER','O','AB')from dual;

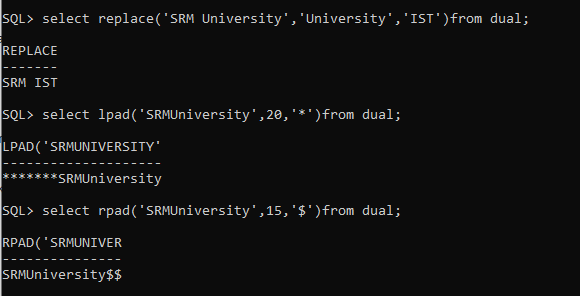
SQL> select replace('University','city’,'Inter')from dual;

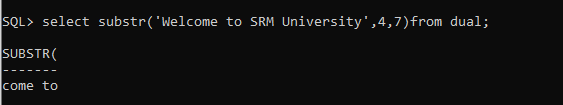
SQL> select translate('cold','ld','ol')from dual;

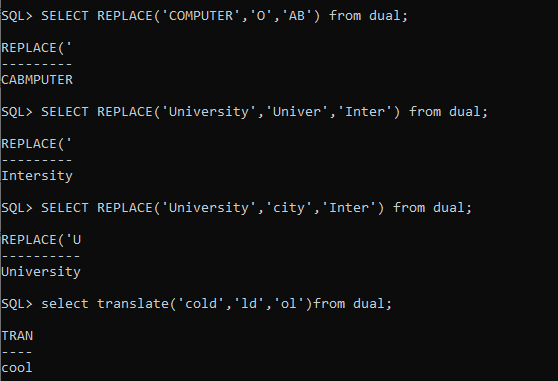
**OUTPUT:**

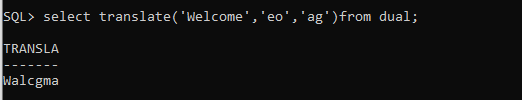












**DATE & TIME FUNCTION**

SQL> select sysdate from dual;

SQL> select round(sysdate)from dual;

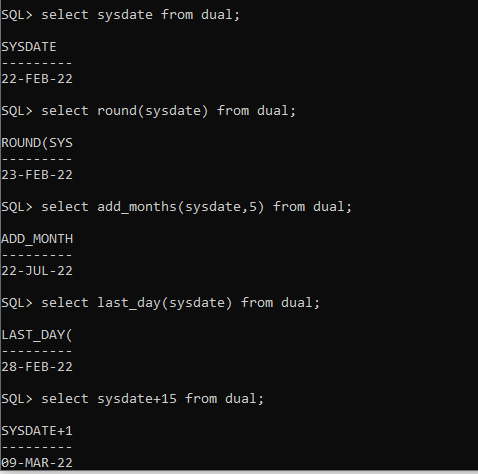
SQL> select add\_months(sysdate,3)from dual;

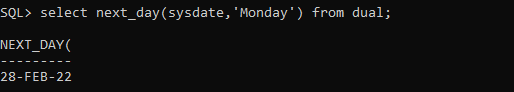
SQL> select last\_day(sysdate)from dual;

SQL> select sysdate+20 from dual;

SQL> select next\_day(sysdate,'tuesday')from dual;

**OUTPUT:**





**NUMERIC FUNCTION**

SQL> select round(15.6789)from dual;

SQL> select ceil(23.20)from dual;

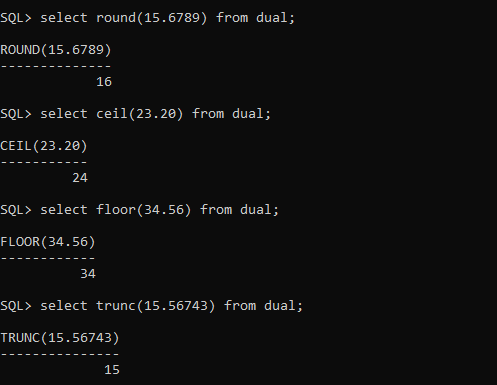
SQL> select floor(34.56)from dual;

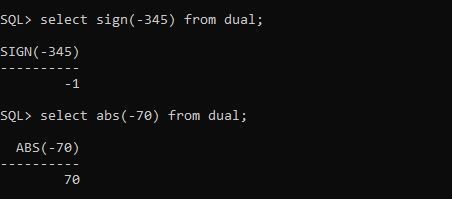
SQL> select trunc(15.56743)from dual;

SQL> select sign(-345)from dual;

SQL> select abs(-70)from dual;

**OUTPUT:**





**MATH FUNCTION:**

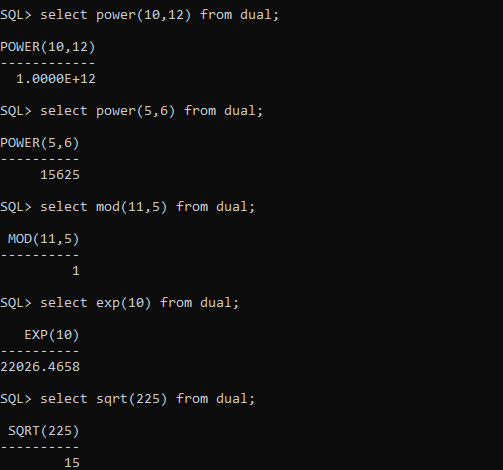
SQL> select power(10,12) from dual;

SQL> select power(5,6) from dual;

SQL> select mod(11,5) from dual;

SQL> select exp(10) from dual;

SQL> select sqrt(225) from dual;



**RESULT:**

Thus the SQL Functions have been executed successfully.

**Ex.No:5**

**CONSTRUCT A ER MODEL FOR THE APPLICATION TO BE CONSTRUCTED TO A DATABASE**

**AIM:**

To construct a ER model for the application to be constructed to a database.

**HOW TO DRAW AN ENTITY RELATIONSHIP DIAGRAM?:**

1. Determine the Entities in Your ERD.
2. Add Attributes to Each Entity.
3. Define the Relationships Between Entities.
4. Add Cardinality to Every Relationship in your ER Diagram.
5. Finish and Save Your ERD.

**Few Online Entity Relationship Diagram Tools**

1. Vertabelo
2. Creatly
3. ERDPlus
4. Lucidchart
5. Visual Paradigm Online
6. Draw.io
7. Microsoft Visio
8. Gliffy
9. SqlDBM ER Diagram Online Tool
10. ER Draw Max

|  |  |
| --- | --- |
| **S.No.** | **Problem Statement** |
| 1 | 1. Construct an E-R diagram for a hospital with a set of patients and a set of medical doctors. Associate with each patient a log of the various tests and examinations conducted.   **ER Diagram:**     1. Construct appropriate tables for the above ER Diagram :   **Hospital tables:**  patients (patient-id, name, insurance, date-admitted, date-checked-out)  doctors (doctor-id, name, specialization)  test (testid, testname, date, time, result)  doctor-patient (patient-id, doctor-id)  test-log (testid, patient-id)  performed-by (testid, doctor-id) |
| 2 | Design an E-R diagram for keeping track of the exploits of your favourite sports team. You should store the matches played, the scores in each match, the players in each match and individual player statistics for each match. Summary statistics should be modeled as derived attributes  **ER Diagram:** |
| 3 | Design an E-R diagram for bus reservation system.  Bus ER Diagram | EdrawMax Templates |
| 4 | 1. Construct an E-R diagram for a car-insurance company whose customers own one or more cars each. Each car has associated with it zero to any number of recorded accidents.      1. Construct appropriate tables for the above ER Diagram?   **Car insurance tables**:  person (driver-id, name, address)  car (license, year, model)  accident (report-number, date, location)  participated(driver-id, license, report-number, damage-amount) |
| 5 | Design an E-R diagram for University Database  ER Diagram Example: University Database |
| 6 | Consider a database used to record the marks that students get in different exams of different course offerings.   1. Construct an E-R diagram that models exams as entities, and uses a ternary relationship, for the above database.      1. Construct an alternative E-R diagram that uses only a binary relationship between students and course-offerings. Make sure that only one relationship exists between a particular student and course-offering pair, yet you can represent the marks that a student gets in different exams of a course offering.   **ER Diagram:** |
| 7 | Draw the E-R diagram which models an online bookstore. |
| 8 | Consider a university database for the scheduling of classrooms for -final exams. This database could be modeled as the single entity set exam, with attributes course-name, sectionnumber, room-number, and time. Alternatively, one or more additional entity sets could be defined, along with relationship sets to replace some of the attributes of the exam entity set, as   * course with attributes name, department, and c-number * section with attributes s-number and enrollment, and dependent as a weak entity set on course * room with attributes r-number, capacity, and building   Show an E-R diagram illustrating the use of all three additional entity sets listed. |
| 9 | Construct an ER Diagram for Company having following details :   * Company organized into DEPARTMENT. Each department has unique name and a particular employee who manages the department. Start date for the manager is recorded. Department may have several locations. * A department controls a number of PROJECT. Projects have a unique name, number and a single location. * Company’s EMPLOYEE name, ssno, address, salary, sex and birth date are recorded. An employee is assigned to one department, but may work for several projects (not necessarily controlled by her dept). Number of hours/week an employee works on each project is recorded; The immediate supervisor for the employee. * Employee’s DEPENDENT are tracked for health insurance purposes (dependent name, birthdate, relationship to employee). |
| 10 | Design an E-R diagram for Customer Account. |

**EX.NO:6 NESTED QUERIES**

**AIM:**

To study the various SQL nested queries operations on the database.

**DESCRIPTION:**

Nested query is one of the most useful functionalities of SQL. Nested queries are useful when we want to write complex queries where one query uses the result from another query. Nested queries will have multiple SELECT statements nested together. A SELECT statement nested within another SELECT statement is called a subquery.

**What is a Nested Query in SQL?**

A nested query in SQL contains a query inside another query. The result of the inner query will be used by the outer query. For instance, a nested query can have two **SELECT** statements, one on the inner query and the other on the outer query.

**What are the Types of Nested Queries in SQL?**

Nested queries in SQL can be classified into two different types:

1. Independent Nested Queries
2. Co-related Nested Queries
3. **Independent Nested Queries**

In independent nested queries, the execution order is from the innermost query to the outer query. An outer query won't be executed until its inner query completes its execution. The result of the inner query is used by the outer query. Operators such as **IN**, **NOT IN**, **ALL**, and **ANY** are used to write independent nested queries.

The **IN** operator checks if a column value in the outer query's result is **present** in the inner query's result. The final result will have rows that satisfy the **IN** condition.

The **NOT IN** operator checks if a column value in the outer query's result is **not present** in the inner query's result. The final result will have rows that satisfy the **NOT IN** condition.

The **ALL** operator compares a value of the outer query's result with **all the values** of the inner query's result and returns the row if it matches all the values.

The **ANY** operator compares a value of the outer query's result with all the inner query's result values and returns the row if there is a match with **any value**.

1. **Co-related Nested Queries**

In co-related nested queries, the inner query uses the values from the outer query so that the inner query is executed for every row processed by the outer query. The co-related nested queries run slowly because the inner query is executed for every row of the outer query's result.

**How to Write Nested Query in SQL?**

We can write a nested query in SQL by nesting a **SELECT** statement within another **SELECT** statement. The outer **SELECT** statement uses the result of the inner **SELECT** statement for processing.

**The general syntax of nested queries will be:**

*SELECT column\_name [, column\_name ]*

*FROM table1 [, table2 ]*

*WHERE column\_name OPERATOR*

*( SELECT column\_name [, column\_name ]*

*FROM table1 [, table2 ]*

*[WHERE]*

*)*

The **SELECT** query inside the brackets (**()**) is the inner query, and the **SELECT** query outside the brackets is the outer query. The result of the inner query is used by the outer query.

**EXAMPLE:**

**TABLE #1 - employeedata**

*SQL> CREATE TABLE employeedata(id NUMBER PRIMARY KEY, name VARCHAR2(25) NOT NULL, salary NUMBER NOT NULL, role VARCHAR2(15) NOT NULL);*

Table created.

*SQL> INSERT INTO employeedata VALUES (1, 'Augustine Hammond', 10000, 'Developer');*

1 row created.

SQL> INSERT INTO employeedata VALUES (2, 'Perice John', 10000, 'Manager');

1 row created.

*SQL> INSERT INTO employeedata VALUES (3, 'Ragu Delafoy', 30000, 'Developer');*

1 row created.

*SQL> INSERT INTO employeedata VALUES (4, 'Teakwood Saffen', 40000, 'Manager');*

1 row created.

*SQL> INSERT INTO employeedata VALUES (5, 'Freddy Malcom', 50000, 'Developer');*

1 row created.

*SQL> select \* from employeedata;*

**OUTPUT:**

ID NAME SALARY ROLE

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

1 Augustine Hammond 10000 Developer

2 Perice John 10000 Manager

3 Ragu Delafoy 30000 Developer

4 Teakwood Saffen 40000 Manager

5 Freddy Malcom 50000 Developer

**TABLE #2 - awards**

*SQL>CREATE TABLE awards( id NUMBER PRIMARY KEY, employee\_id NUMBER NOT NULL, award\_date DATE NOT NULL );*

Table created.

*SQL> INSERT INTO awards VALUES(1, 1, TO\_DATE('2022-04-01', 'YYYY-MM-DD'));*

1 row created.

*SQL> INSERT INTO awards VALUES(2, 3, TO\_DATE('2022-05-01', 'YYYY-MM-DD'));*

1 row created.

*SQL> select \* from awards;*

**OUTPUT:**

ID EMPLOYEE\_ID AWARD\_DAT

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

1 1 01-APR-22

2 3 01-MAY-22

**Independent Nested Queries**

**Example 1: IN**

* Select all employees who won an award.

*SQL> SELECT id, name FROM employeedata WHERE id IN (SELECT employee\_id FROM awards);*

**OUTPUT:**

ID NAME

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

1 Augustine Hammond

3 Ragu Delafoy

**Example 2: NOT IN**

* Select all employees who never won an award.

*SQL> SELECT id, name FROM employeedata WHERE id NOT IN (SELECT employee\_id FROM awards);*

**OUTPUT:**

ID NAME

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

2 Perice John

4 Teakwood Saffen

5 Freddy Malcom

**Example 3: ALL**

* Select all Developers who earn more than all the Managers

*SQL> SELECT \* FROM employeedata WHERE role = 'Developer' AND salary > ALL (SELECT salary FROM employeedata WHERE role = 'Manager');*

**OUTPUT:**

ID NAME SALARY ROLE

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

5 Freddy Malcom 50000 Developer

**Example 4: ANY**

* Select all Developers who earn more than any Manager

*SQL> SELECT \* FROM employeedata WHERE role = 'Developer' AND salary > ANY (SELECT salary FROM employeedata WHERE role = 'Manager');*

**OUTPUT:**

ID NAME SALARY ROLE

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

3 Ragu Delafoy 30000 Developer

5 Freddy Malcom 50000 Developer

**Co-related Nested Queries**

* Select all employees whose salary is above the average salary of employees in their role.

**Example:**

*SQL> SELECT \* FROM employeedata emp1 WHERE salary > (SELECT AVG(salary) FROM employeedata emp2 WHERE emp1.role = emp2.role);*

**OUTPUT:**

ID NAME SALARY ROLE

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

4 Teakwood Saffen 40000 Manager

5 Freddy Malcom 50000 Developer

**Explanation**

*The manager with id 4 earns more than the average salary of all managers (25000), and the developer with id 5 earns more than the average salary of all developers (30000). The inner query is executed for all rows fetched by the outer query. The role value (emp1.role) of every outer query's row is used by the inner query (emp1.role = emp2.role).*

* We can find the average salary of managers and developers using the below query:

*SQL> SELECT role, AVG(salary) FROM employeedata GROUP BY role;*

**OUTPUT:**

ROLE AVG(SALARY)

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

Developer 30000

Manager 25000

**RESULT:**

Thus the study the various SQL nested queries operations on the database is executed successfully.

**EX.NO:7 JOIN QUERIES**

**AIM**

To study the various SQL Join queries operations on the database.

**PROCEDURE:**

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

**Different Types of SQL JOINs**

Here are the different types of the JOINs in SQL:

1. **(INNER) JOIN: Returns records that have matching values in both tables**

***Syntax:***

SELECT *column\_name(s)*  
FROM *table1*  
INNER JOIN *table2*ON *table1.column\_name*=*table2.column\_name*;

1. **LEFT JOIN: Returns all records from the left table, and the matched records from the right table**

***Syntax:***

SELECT *column\_name(s)*  
FROM *table1*  
LEFT JOIN *table2*ON *table1.column\_name*=*table2.column\_name*;

1. **RIGHT (OUTER) JOIN: Returns all records from the right table, and the matched records from the left table**

***Syntax:***

SELECT *column\_name(s)*  
FROM *table1*  
RIGHT JOIN *table2*ON *table1.column\_name*=*table2.column\_name*;

1. **FULL (OUTER) JOIN: Returns all records when there is a match in either left or right table**

***Syntax:***

SELECT *column\_name(s)*  
FROM *table1*  
FULL OUTER JOIN *table2*ON *table1.column\_name*=*table2.column\_name*WHERE *condition*;

**EXAMPLE:**

**CREATE A TABLE PRODUCTORDERS**

*SQL> CREATE table productorders(Order\_Id number(5),Orderno number(5),P\_Id number(3));*

Table created.

*SQL> desc productorders;*

Name Null? Type

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

ORDER\_ID NUMBER(5)

ORDERNO NUMBER(5)

P\_ID NUMBER(3)

**INSERTING VALUES INTO THE TABLE PRODUCTORDERS**

*SQL> INSERT into productorders values(&Order\_Id,&Orderno,&P\_Id);*

Enter value for order\_id: 1

Enter value for orderno: 77895

Enter value for p\_id: 3

old 1: INSERT into productorders values(&Order\_Id,&Orderno,&P\_Id)

new 1: INSERT into productorders values(1,77895,3)

*1 row created.*

*SQL> INSERT into productorders values(&Order\_Id,&Orderno,&P\_Id);*

Enter value for order\_id: 2

Enter value for orderno: 44678

Enter value for p\_id: 3

old 1: INSERT into productorders values(&Order\_Id,&Orderno,&P\_Id)

new 1: INSERT into productorders values(2,44678,3)

1 row created.

*SQL> INSERT into productorders values(&Order\_Id,&Orderno,&P\_Id);*

Enter value for order\_id: 3

Enter value for orderno: 22456

Enter value for p\_id: 1

old 1: INSERT into productorders values(&Order\_Id,&Orderno,&P\_Id)

new 1: INSERT into productorders values(3,22456,1)

*1 row created.*

*SQL> INSERT into productorders values(&Order\_Id,&Orderno,&P\_Id);*

Enter value for order\_id: 4

Enter value for orderno: 24562

Enter value for p\_id: 1

old 1: INSERT into productorders values(&Order\_Id,&Orderno,&P\_Id)

new 1: INSERT into productorders values(4,24562,1)

*1 row created.*

*SQL> INSERT into productorders values(&Order\_Id,&Orderno,&P\_Id);*

Enter value for order\_id: 5

Enter value for orderno: 34764

Enter value for p\_id: 15

old 1: INSERT into productorders values(&Order\_Id,&Orderno,&P\_Id)

new 1: INSERT into productorders values(5,34764,15)

*1 row created.*

**DISPLAYING DATA FROM TABLE PRODUCTORDERS**

SQL> select \* from productorders;

ORDER\_ID ORDERNO P\_ID

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

1 77895 3

2 44678 3

3 22456 1

4 24562 1

5 34764 15

**CREATE A SECOND TABLE PERSON**

*SQL> CREATE table person(p\_Id number(5),LASTNAME varchar2(10),Firstname varchar2(15), Address varchar2(20),city varchar2(10));*

Table created.

**INSERTING VALUES INTO THE TABLE PERSON**

*SQL> INSERT into person values(&p\_Id,'&Lastname','&firstname','&Address','&city');*

Enter value for p\_id: 1

Enter value for lastname: Smith

Enter value for firstname: Jadon

Enter value for address: Ramapuram

Enter value for city: Chennai

old 1: INSERT into person values(&p\_Id,'&Lastname','&firstname','&Address','&city')

new 1: INSERT into person values(1,'Smith','Jadon','Ramapuram','Chennai')

*1 row created.*

*SQL> INSERT into person values(&p\_Id,'&Lastname','&firstname','&Address','&city');*

Enter value for p\_id: 2

Enter value for lastname: Hemal

Enter value for firstname: Elango

Enter value for address: Anna Nagar

Enter value for city: Tamilnadu

old 1: INSERT into person values(&p\_Id,'&Lastname','&firstname','&Address','&city')

new 1: INSERT into person values(2,'Hemal','Elango','Anna Nagar','Tamilnadu')

*1 row created.*

*SQL> INSERT into person values(&p\_Id,'&Lastname','&firstname','&Address','&city');*

Enter value for p\_id: 3

Enter value for lastname: Lanser

Enter value for firstname: Kim

Enter value for address: Hyderabad

Enter value for city: AP

old 1: INSERT into person values(&p\_Id,'&Lastname','&firstname','&Address','&city')

new 1: INSERT into person values(3,'Lanser','Kim','Hyderabad','AP')

*1 row created.*

**DISPLAYING DATA FROM TABLE PERSON**

*SQL> select \* from person;*

P\_ID LASTNAME FIRSTNAME ADDRESS CITY

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

1 Smith Jadon Ramapuram Chennai

2 Hemal Elango Anna Nagar Tamilnadu

3 Lanser Kim Hyderabad AP

**#1 INNER JOIN**

**OUTPUT**

*SQL> SELECT person.firstname,person.city FROM person INNER JOIN productorders ON person.p\_Id = productorders.p\_Id;*

FIRSTNAME CITY

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

Kim AP

Kim AP

Jadon Chennai

Jadon Chennai

**#2 LEFT JOIN**

**OUTPUT**

*SQL> SELECT person.lastname,person.firstname,productorders.orderno FROM person LEFT JOIN productorders ON person.p\_Id = productorders.p\_Id ORDER BY person.lastname;*

LASTNAME FIRSTNAME ORDERNO

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

Hemal Elango

Lanser Kim 77895

Lanser Kim 44678

Smith Jadon 24562

Smith Jadon 22456

**#3 RIGHT (OUTER) JOIN**

**OUTPUT**

*SQL> SELECT person.lastname,person.firstname,productorders.orderno FROM person RIGHT OUTER JOIN productorders ON person.p\_Id = productorders.p\_Id ORDER BY person.firstname;*

LASTNAME FIRSTNAME ORDERNO

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

Smith Jadon 22456

Smith Jadon 24562

Lanser Kim 77895

Lanser Kim 44678

34764

**#4 FULL OUTER JOIN**

**OUTPUT**

*SQL> SELECT person.lastname,person.address FROM person FULL OUTER JOIN productorders ON person.p\_Id = productorders.p\_Id ORDER BY person.firstname;*

LASTNAME ADDRESS

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

Hemal Anna Nagar

Smith Ramapuram

Smith Ramapuram

Lanser Hyderabad

Lanser Hyderabad

*6 rows selected.*

**RESULT:**

Thus the various SQL Join queries operations on the database have been executed successfully.

**Ex.No:8**

**Date:**

**SET OPERATORS AND VIEWS**

**AIM:**

To write a SQL queries to implement the set operators and views.

**PROCEDURE:**

**SQL SET OPERATION:**

The SQL Set operation is used to combine the two or more SQL SELECT statements.

**Types of Set Operation**

1. Union
2. UnionAll
3. Intersect
4. Minus

**1. Union**

* The SQL Union operation is used to combine the result of two or more SQL SELECT queries.
* In the union operation, all the number of datatype and columns must be same in both the tables on which UNION operation is being applied.
* The union operation eliminates the duplicate rows from its resultset.

**Syntax:**

*SELECT column\_name FROM table1   UNION   SELECT column\_name FROM table2;*

**2. Union All**

* Union All operation is equal to the Union operation. It returns the set without removing duplication and sorting the data.

**Syntax:**

*SELECT column\_name FROM table1  UNION ALL  SELECT column\_name FROM table2;*

**3. Intersect**

* It is used to combine two SELECT statements. The Intersect operation returns the common rows from both the SELECT statements.
* In the Intersect operation, the number of datatype and columns must be the same.
* It has no duplicates and it arranges the data in ascending order by default.

**Syntax:**

*SELECT column\_name FROM table1  INTERSECT  SELECT column\_name FROM table2;*

**4. Minus**

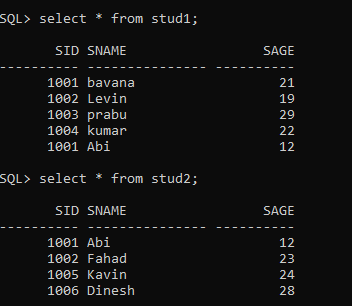
* It combines the result of two SELECT statements. Minus operator is used to display the rows which are present in the first query but absent in the second query.
* It has no duplicates and data arranged in ascending order by default.

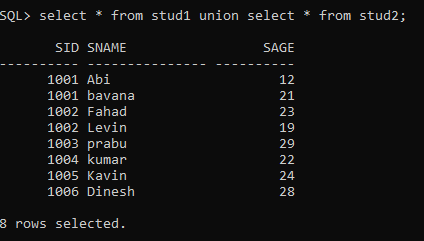
**Syntax:**

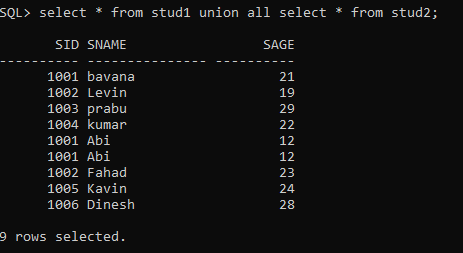
*SELECT column\_name FROM table1  MINUS  SELECT column\_name FROM table2;*

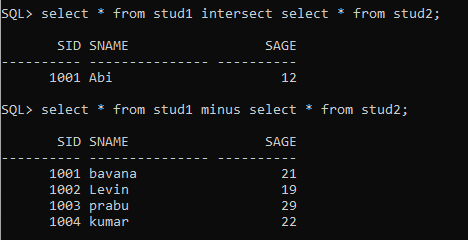
**Output:**

Creating two tables namely stud1 and stud2.









**SQL VIEW**

SQL provides the concept of VIEW, which hides the complexity of the data and restricts unnecessary access to the database.

It permits the users to access only a particular column rather than the whole data of the table.

The **View** in the Structured Query Language is considered as the virtual table, which depends on the result-set of the predefined SQL statement.

**Create a SQL View**

To create a View in Structured Query Language by using the CREATE VIEW statement. Creation of View from a *single table or multiple tables.*

**Syntax to Create View from Single Table**

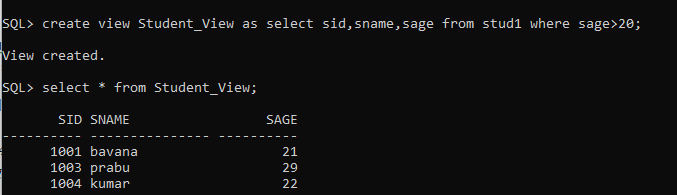
***CREATE******VIEW****View\_Name****AS***

***SELECT****Column\_Name1, Column\_Name2, ....., Column\_NameN*

***FROM****Table\_Name*

***WHERE****condition;*

**Example to Create a View from Single table**



**Syntax to Create View from Multiple Tables**

**To create a View from multiple tables by including the tables in the SELECT statement.**

*CREATE****VIEW****View\_Name****AS***

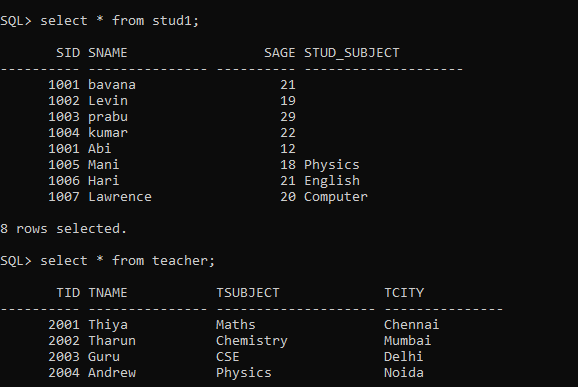
***SELECT****Table\_Name1.Column\_Name1, Table\_Name1.Column\_Name2,*

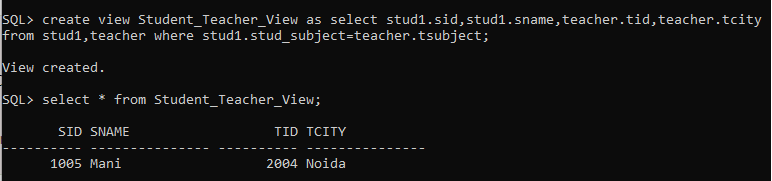
*Table\_Name2.Column\_Name2,  ....., Table\_NameN.Column\_NameN*

***FROM****Table\_Name1, Table\_Name2, ....., Table\_NameN*

***WHERE****condition;*

**Example to Create a View from Multiple tables**





**Update an SQL View**

*A view in SQL can only be modified if the view follows the following conditions:*

1. You can update that view which depends on only one table. SQL will not allow updating the view which is created more than one table.
2. The fields of view should not contain NULL values.
3. The view does not contain any subquery and DISTINCT keyword in its definition.
4. The views cannot be updatable if the SELECT statement used to create a View contains JOIN or HAVING or GROUP BY clause.
5. If any field of view contains any SQL aggregate function, you cannot modify the view.

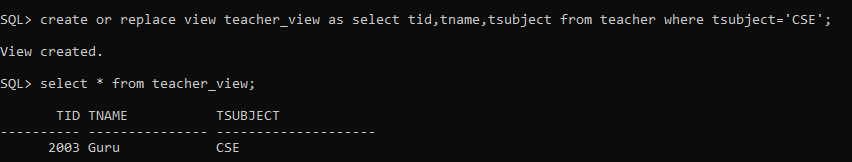
**Syntax to Update a View**

***CREATE****OR REPLACE****VIEW****View\_Name****AS***

***SELECT****Column\_Name1, Column\_Name2, ....., Column\_NameN*

***FROM****Table\_Name*

***WHERE****condition;*

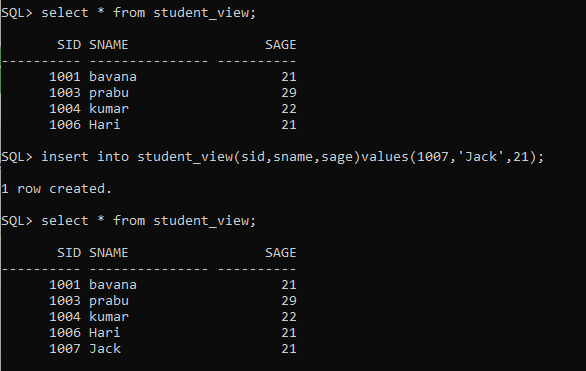


**Insert the new row into the existing view**

Just like the insertion process of database tables, we can also insert the record in the views. The following SQL INSERT statement is used to insert the new row or record in the view:

**Syntax:**

***INSERT******INTO****View\_Name(Column\_Name1, Column\_Name2 , Column\_Name3, ....., Column\_NameN)****VALUES****(value1, value2, value3, ...., valueN);*

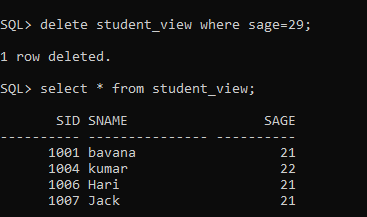


**Delete the existing row from the view**

Just like the deletion process of database tables, we can also delete the record from the views. The following SQL DELETE statement is used to delete the existing row or record from the view:

**Syntax**

***DELETE******FROM****View\_Name****WHERE****Condition;*

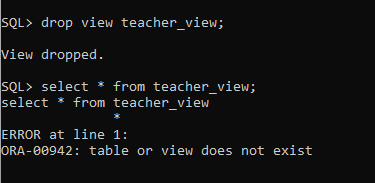


**Drop a View**

To delete the existing view from the database if it is no longer needed the following SQL DROP statement is used to delete the view:

**Syntax:**

***DROP******VIEW****View\_Name;*



**RESULT:**

Thus the SQL queries for implementing the set operators and views are executed successfully.

**EX.NO: 9**

**PL/SQL CONDITIONAL AND ITERATIVE STATEMENTS**

**AIM**

To study the various basic PL/SQL **Conditional and Iterative Statements** on the database.

**DESCRIPTION:**

The iterative statements are used to repeat the execution of certain statements multiple times. This is achieved with the help of loops. Loops in PL/SQL provide a mechanism to perform specific tasks multiple times without having to write them multiple times.

This article will discuss three main types of loops:

* ***Basic loop***
* ***WHILE loop***
* ***FOR loop***

**Basic loop**

The basic loop will execute the statement provided a certain number of times until the exit condition is met. It is necessary to have an EXIT statement so that the loop does not run indefinitely. There is also an increment statement that can be used to increase/decrease the changing variable in the loop.

**Syntax:**

***LOOP***

***Statements;***

***[increment\_statement]***

***EXIT condition;***

***END LOOP;***

**WHILE loop**

The WHILE loop in PL/SQL  is used to check the entry condition, and if the entry condition is true, only then is the loop executed. The basic loop executes at least once, whereas the WHILE loop will first check the condition provided in the boolean expression. If the condition is false, the control does not enter the loop.

**Syntax:**

***WHILE (boolean\_expression) LOOP***

***statements ;***

***[increment\_statement]***

***END LOOP;***

**FOR loop**

The FOR loop in PL/SQL provides implicit variable declaration, implicit incrementation of the variable by one, and implicit exit also. In the FOR loop, we do not have to declare the variable as we did in the previous two types of loop. While writing the loop statement, the variable is declared implicitly. The range consists of the starting value, from where the value of the iterating variable begins, and the end value, which determines the last value which the variable can have. In each loop, the variable is incremented by one.

**Syntax:**

***FOR variable IN range LOOP***

***Statements;***

***END LOOP;***

**EXAMPLES:**

**1. PL/SQL CODING FOR ADDITION OF TWO NUMBERS**

SQL> declare

a number;

b number;

c number;

begin

a:=&a;

b:=&b;

c:=a+b;

dbms\_output.put\_line('sum of'||a||'and'||b||'is'||c);

end;

/

**INPUT:**

Enter value for a: 23

old 6: a:=&a;

new 6: a:=23;

Enter value for b: 12

old 7: b:=&b;

new 7: b:=12;

**OUTPUT:**

sum of23and12is35

PL/SQL procedure successfully completed.

**2. PL/ SQL GENERAL SYNTAX FOR IF CONDITION:**

SQL> DECLARE

<VARIABLE DECLARATION>;

BEGIN

IF(CONDITION)THEN

<EXECUTABLE STATEMENT >;

END;

Coding for If Statement:

DECLARE

b number;

c number;

BEGIN

B:=10;

C:=20; if(C>B)

THEN

dbms\_output.put\_line('C is maximum');

end if;

end;

/

**OUTPUT:**

C is maximum

PL/SQL procedure successfully completed.

**3. PL/ SQL GENERAL SYNTAX FOR IF AND ELSECONDITION:**

SQL> DECLARE

<VARIABLE DECLARATION>;

BEGIN

IF (TEST CONDITION) THEN

<STATEMENTS>;

ELSE

<STATEMENTS>;

ENDIF;

END;

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Less then or Greater Using IF ELSE \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

SQL> declare

n number;

begin

dbms\_output. put\_line('enter a number');

n:=&number;

if n<5 then

dbms\_output.put\_line('entered number is less than 5');

else

dbms\_output.put\_line('entered number is greater than 5');

end if;

end;

/

**INPUT**

Enter value for number: 2

old 5: n:=&number;

new 5: n:=2;

**OUTPUT:**

entered number is less than 5

PL/SQL procedure successfully completed.

**4.PL/ SQL GENERAL SYNTAX FOR NESTED IF:**

SQL> DECLARE

<VARIABLE DECLARATION>;

BEGIN

IF (TEST CONDITION) THEN

<STATEMENTS>;

ELSEIF (TEST CONDITION) THEN

<STATEMENTS>;

ELSE

<STATEMENTS>;

ENDIF;

END;

\*\*\*\*\*\*\*\*\*\* GREATEST OF THREE NUMBERS USING IF ELSEIF\*\*\*\*\*\*\*\*\*\*\*\*

SQL> declare

a number;

b number;

c number;

d number;

begin

a:=&a;

b:=&b;

c:=&b;

if(a>b)and(a>c) then

dbms\_output.put\_line('A is maximum');

elsif(b>a)and(b>c)then

dbms\_output.put\_line('B is maximum');

else

dbms\_output.put\_line('C is maximum');

end if;

end;

/

**INPUT:**

Enter value for a: 21

old 7: a:=&a;

new 7: a:=21;

Enter value for b: 12

old 8: b:=&b;

new 8: b:=12;

Enter value for b: 45

old 9: c:=&b;

new 9: c:=45;

**OUTPUT:**

C is maximum

PL/SQL procedure successfully completed.

**5.PL/ SQL GENERAL SYNTAX FOR LOOPING STATEMENT:**

SQL> DECLARE

<VARIABLE DECLARATION>;

BEGIN

LOOP

<STATEMENT>;

END LOOP;

<EXECUTAVLE STATEMENT>;

END;

\*\*\*\*\*\*\*\*\*\*\*SUMMATION OF ODD NUMBERS USING FOR LOOP\*\*\*\*\*\*\*\*\*\*\*

SQL> declare

n number;

sum1 number default 0;

endvalue number;

begin

endvalue:=&endvalue;

n:=1;

for n in 1..endvalue

loop

if mod(n,2)=1

then

sum1:=sum1+n;

end if;

end loop;

dbms\_output.put\_line('sum ='||sum1);

end;

/

**INPUT:**

Enter value for endvalue: 4

old 6: endvalue:=&endvalue;

new 6: endvalue:=4;

**OUTPUT:**

sum =4

PL/SQL procedure successfully completed.

**6.PL/ SQL GENERAL SYNTAX FOR LOOPING STATEMENT:**

SQL> DECLARE

<VARIABLE DECLARATION>;

BEGIN

WHILE <condition>

LOOP

<STATEMENT>;

END LOOP;

<EXECUTAVLE STATEMENT>;

END;

\*\*\*\*\*\*\*\*\*SUMMATION OF ODD NUMBERS USING WHILE LOOP\*\*\*\*\*\*\*\*\*\*

SQL> declare

n number;

sum1 number default 0;

endvalue number;

begin

endvalue:=&endvalue;

n:=1;

while(n<endvalue)

loop

sum1:=sum1+n;

n:=n+2;

end loop;

dbms\_output.put\_line('sum of odd no. bt 1 and' ||endvalue||'is'||sum1);

end;

/

**INPUT:**

Enter value for endvalue: 4

old 6: endvalue:=&endvalue;

new 6: endvalue:=4;

**OUTPUT:**

sum of odd no. bt 1 and4is4

PL/SQL procedure successfully completed.

**RESULT:**

Thus the study the various basic PL/SQL Conditional and Iterative Statements (***Basic loop, WHILE loop, FOR loop) on*** the database is executed successfully.