18CSC303J - DATABASE MANAGEMENT SYSTEMS

#### SEMESTER – VI

**2021 – 2022 (EVEN)**

**Name :**

**Register No. :**

**Branch : CSE-CN**

**Section : P1**



**DEPARTMENT OF NETWORKING AND COMMUNICATIONS**

**SCHOOL OF COMPUTING**

**SRM INSTITUTE OF SCIENCE AND TECHNOLOGY**

**(Under Section 3 of UGC Act, 1956)**

#### S.R.M. NAGAR, KATTANKULATHUR – 603 203 CHENGALPATTU DISTRICT

**SRM INSTITUTE OF SCIENCE AND TECHNOLOGY**



###### (Under Section 3 of UGC Act, 1956)

**S.R.M. NAGAR, KATTANKULATHUR**

BONAFIDE CERTIFICATE

Register No. :

Certified to be the bonafide record of work done by **VARUN J** of **CSE-CN, B.Tech.** Degree course in the Practical of **18CSC303J - DATABASE MANAGEMENT SYSTEMS** in **SRM IST**, Kattankulathur during the academic year **2021 - 2022**.

**Staff In-Charge Head of the Department**

**Date :**

Submitted for University Examination held on ……….. **date**……………. at **SRMIST**, Kattankulathur.

**Internal Examiner I Internal Examiner II**

**TABLE OF CONTENTS**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Ex. No.** | **Date** | **Title** | **Page No.** | **Marks** |
| 1 | 24-01-22 | DDL commands in SQL |  |  |
| 2 | 29-01-22 | DML Commands in SQL |  |  |
| 3 | 04-02-22 | DCL and TCL commands in SQL |  |  |
| 4 | 08-02-22 | Built-in functions in SQL |  |  |
| 5 | 22-02-22 | Construction of an ER diagram |  |  |
| 6 | 23-02-22 | JOIN queries in SQL |  |  |
| 7 | 08-03-22 | SUB queries in SQL |  |  |
| 8 | 09-03-22 | SET operators and VIEWS in SQL |  |  |
| 9 | 18-03-22 | Simple PL/SQL |  |  |
| 10 | 22-03-22 | PROCEDURES in PL/SQL |  |  |
| 11 | 28-03-22 | FUNCTIONS in PL/SQL |  |  |
| 12 | 01-04-22 | CURSORS in PL/SQL |  |  |
| 13 | 03-04-22 | TRIGGERS in PL/SQL |  |  |
| 14 | 04-04-22 | EXCEPTIONAL HANDLING in  PL/SQL |  |  |

**EXERCISE – 1**

**Aim: Data Definition Language using SQL COMMANDS**

**Data Definition Language** (DDL) statements are used to define the database structure or schema. Some examples:

* CREATE - to create objects in the database
* ALTER - alters the structure of the database
* DROP - delete objects from the database
* TRUNCATE - remove all records from a table, including all spaces allocated for the records are removed
* COMMENT - add comments to the data dictionary
* RENAME - rename an object

###### The Create Table Command

The create table command defines each column of the table uniquely. Each column has minimum of three attributes.

* Name
* Data type
* Size (column width).

Each table column definition is a single clause in the create table syntax. Each table column definition is separated from the other by a comma. Finally, the SQL statement is terminated with a semicolon.

###### The Structure of Create Table Command Table name is Student

|  |  |  |
| --- | --- | --- |
| **Column name** | **Data type** | **Size** |
| Reg\_no | varchar2 | 10 |
| Name | char | 30 |
| DOB | date |  |
| Address | varchar2 | 50 |

**The DROP Command Syntax:**

###### The TRUNCATE Command Syntax:

**The RENAME Command Syntax:**

###### The ALTER Table Command

By The use of ALTER TABLE Command we can **modify** our exiting table.

###### Adding New Columns Syntax:

**Dropping a Column from the Table Syntax:**

###### Modifying Existing Table Syntax:

**ALTER TABLE <table\_name> MODIFY (<column\_name> <NewDataType>(<NewSize>))**

###### Restriction on the ALTER TABLE

Using the ALTER TABLE clause the following tasks cannot be performed.

* Change the name of the table
* Change the name of the column
* Decrease the size of a column if table data exists

###### Lab Experiment:

SQL> create table emp

2 (

3 empno int,

4 ename varchar(20) not null,

5 job varchar(10) not null,

6 deptno varchar(3),

7 sal int

8 );

Table created.

SQL> describe emp;

Name Null? Type

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

EMPNO NUMBER(38)

ENAME NOT NULL VARCHAR2(20)

JOB NOT NULL VARCHAR2(10)

DEPTNO VARCHAR2(3)

SAL NUMBER(38)

SQL> alter table emp

2 add (experience int);

Table altered.

SQL> describe emp;

Name Null? Type

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

EMPNO NUMBER(38)

ENAME NOT NULL VARCHAR2(20)

JOB NOT NULL VARCHAR2(10)

DEPTNO VARCHAR2(3)

SAL NUMBER(38)

EXPERIENCE NUMBER(38)

SQL> alter table emp

2 modify (job varchar(20));

Table altered.

SQL> describe emp;

Name Null? Type

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

EMPNO NUMBER(38)

ENAME NOT NULL VARCHAR2(20)

JOB NOT NULL VARCHAR2(20)

DEPTNO VARCHAR2(3)

SAL NUMBER(38)

EXPERIENCE NUMBER(38)

SQL> create table dept

2 (

3 deptno int,

4 dname varchar(10),

5 loc varchar(10)

6 );

Table created.

SQL> describe dept;

Name Null? Type

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

DEPTNO NUMBER(38)

DNAME VARCHAR2(10)

LOC VARCHAR2(10)

SQL> alter table dept

2 modify (deptno int primary key);

Table altered.

SQL> describe dept;

Name Null? Type

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

DEPTNO NOT NULL NUMBER(38)

DNAME VARCHAR2(10)

LOC VARCHAR2(10)

SQL> create table emp1 as

2 select ename, empno

3 from emp

4 ;

Table created.

SQL> drop table emp1;

Table dropped

SQL> values (1, 'Aditya', 'ML', 101, 100000, 5);

SP2-0734: unknown command beginning "values (1,..." - rest of line ignored.

SQL> insert into emp

2 values (1, 'Aditya', 'ML', 101, 100000, 5);

1 row created.

SQL> insert into emp

2 values (10, 'Rahul', 'AI', 111, 10000, 3);

1 row created.

SQL> insert into emp

2 values (200, 'Aman', 'AI', 111, 50000, 4);

1 row created.

SQL> insert into emp

2 values (250, 'Lenar', 'Mechanic', 001, 20000, 6);

1 row created.

SQL> describe emp;

Name Null? Type

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

EMPNO NUMBER(38)

ENAME NOT NULL VARCHAR2(20)

JOB NOT NULL VARCHAR2(20)

DEPTNO VARCHAR2(3)

SAL NUMBER(38)

EXPERIENCE NUMBER(38)

SQL> select \* from emp;

EMPNO ENAME JOB DEP SAL EXPERIENCE

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

1 Aditya ML 101 100000 5

10 Rahul AI 111 10000 3

200 Aman AI 111 50000 4

250 Lenar Mechanic 1 20000 6

SQL> create table emp1 as

2 select ename, empno

3 from emp

4 where empno > 100;

Table created.

SQL> describe emp1;

Name Null? Type

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

ENAME NOT NULL VARCHAR2(20)

EMPNO NUMBER(38)

SQL> select \* from emp1;

ENAME EMPNO

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

Aman 200

Lenar 250

SQL> alter table emp

2 drop column experience;

Table altered.

SQL> describe emp;

Name Null? Type

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

EMPNO NUMBER(38)

ENAME NOT NULL VARCHAR2(20)

JOB NOT NULL VARCHAR2(20)

DEPTNO VARCHAR2(3)

SAL NUMBER(38)

SQL> truncate table emp;

Table truncated.

SQL> describe emp;

Name Null? Type

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

EMPNO NUMBER(38)

ENAME NOT NULL VARCHAR2(20)

JOB NOT NULL VARCHAR2(20)

DEPTNO VARCHAR2(3)

SAL NUMBER(38)

SQL> select \* from emp;

no rows selected

SQL> drop table dept;

Table dropped.

SQL> describe dept;

ERROR:

ORA-04043: object dept does not exist

SQL> spool off;

**Result:** Data Definition Language using SQL COMMANDS has been studied and implemented

.

**EXERCISE – 2**

**Aim: To study DML (Data Manipulation Language) using SQL COMMANDS**

DML statements affect records in a table. These are basic operations we perform on data such as selecting a few records from a table, inserting new records, deleting unnecessary records, and updating/modifying existing records.

DML statements include the following:

**SELECT** – select records from a table

**INSERT** – insert new records

**UPDATE** – update/Modify existing records

**DELETE** – delete existing records

###### DML command

Data Manipulation Language (DML) statements are used for managing data in database. DML commands are not auto-committed. It means changes made by DML command are not permanent to database, it can be rolled back.

###### INSERT COMMAND

Insert command is used to insert data into a table. Following is its general syntax,

**INSERT** into *table-name* values(data1,data2,..)

###### UPDATE COMMAND

Update command is used to update a row of a table. Following is its general syntax,

**UPDATE** *table-name* set column-name = value *where* **condition**;

###### DELETE COMMAND

Delete command is used to delete data from a table. Delete command can also be used with conditions to delete a particular row. Following is its general syntax,

**DELETE** from *table-name*;

###### WHERE clause

***Where*** clause is used to specify condition while retrieving data from table. ***Where*** clause is used mostly with *Select*, *Update* and *Delete* query. If condition specified by ***where*** clause is true then only the result from table is returned.

|  |
| --- |
| ***Syntax for WHERE clause*** |
| *SELECT* column-name1, |
| column-name2, |
| column-name3, |
| column-nameN |
| from table-name **WHERE [condition]**; |

###### SELECT COMMAND

**SELECT Query**

Select query is used to retrieve data from a tables. It is the most used SQL query. We can retrieve complete tables, or partial by mentioning conditions using WHERE clause.

|  |
| --- |
| ***Syntax of SELECT Query*** |
| **SELECT** column-name1, column-name2, column-name3, column-nameN from *table-name*; |

###### Like Clause

**Like** clause is used as condition in SQL query. **Like** clause compares data with an expression using wildcard operators. It is used to find similar data from the table.

* **Percent sign %** : represents zero, one or more than one character.
* **Underscore sign \_** : represents only one character.

###### Order By Clause

Order by clause is used with the **Select** statement for arranging retrieved data in sorted order. The **Order by clause** by default sort data in ascending order. To sort data in descending order **DESC** keyword is used with **Order by** clause.

|  |
| --- |
| ***Syntax of Order By*** |
| *SELECT* column-list|\* from table-name **order by** *asc*|*desc*; |

###### Group By Clause

Group by clause is used to group the results of a SELECT query based on one or more columns. It is also used with SQL functions to group the result from one or more tables.

Syntax for using Group by in a statement. SELECT column\_name, function(column\_name) **HAVING Clause**

Having clause is used with SQL Queries to give more precise conditions for a statement. It is used to mention conditions in Group based SQL functions, just like WHERE clauses.

Syntax for having will be,

###### Distinct clause

The **distinct** keyword is used with **Select** statement to retrieve unique values from the table. **Distinct** Removes all the duplicate records while retrieving from database.

***Syntax for DISTINCT Keyword* AND** and **OR** operators are used with **Where** clause to make more precise conditions for fetching data from database by combining more than one condition together.

OR operator is also used to combine multiple conditions with the Where clause. The only difference between AND and OR is their behavior. When we use AND to combine two or more than two conditions, records satisfying all the conditions will be in the result. But in the case of OR, at least one condition from the conditions specified must be satisfied by any record to be in the result.

###### Lab Experiment:

SQL> create table student

2 (

3 RegNo int not null,

4 Name varchar(20) not null,

5 Gender varchar(1),

6 DOB date,

7 mobileno int constraint ten check (mobileno between 1000000000 and 9999999999),

8 City varchar(20),

9 primary key (RegNo)

10 );

Table created.

SQL> insert into student

2 values

3 (312, 'Randheer', 'M', to\_date('2000-12-20','yyyy-mm-dd'), 8096735597, 'Rajahmundry');

1 row created.

SQL> insert into student

2 values

3 (9531, 'Sarika', 'F', to\_date('2015-09-15','yyyy-mm-dd'), 9848035597, 'Rajahmundry');

1 row created.

SQL> insert into student

2 values

3 (8088, 'Satya Vani', 'F', to\_date('1986-12-31','yyyy-mm-dd'), 9705710159, 'Rajahmundry');

1 row created.

SQL> insert into student

2 values

3 (2609, 'Durga Rao', 'M', to\_date('1973-09-26','yyyy-mm-dd'), 9949028509, 'Rajahmundry');

1 row created.

SQL> insert into student

2 values

3 (3001, 'Sanju', 'M', to\_date('2002-01-30','yyyy-mm-dd'), 9884792252, 'Rajahmundry');

1 row created.

SQL> insert into student

2 values

3 (601, 'Varija Sri', 'F', to\_date('1999-01-06','yyyy-mm-dd'), 7674978787, 'Rajahmundry');

1 row created.

SQL> describe student

Name Null? Type

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

REGNO NOT NULL NUMBER(38)

NAME NOT NULL VARCHAR2(20)

GENDER VARCHAR2(1)

DOB DATE

MOBILENO NUMBER(38)

CITY VARCHAR2(20)

SQL> select \* from student;

REGNO NAME G DOB MOBILENO CITY

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

312 Randheer M 20-DEC-00 8096735597 Rajahmundry

9531 Sarika F 15-SEP-15 9848035597 Rajahmundry

8088 Satya Vani F 31-DEC-86 9705710159 Rajahmundry

2609 Durga Rao M 26-SEP-73 9949028509 Rajahmundry

3001 Sanju M 30-JAN-02 9884792252 Rajahmundry

601 Varija Sri F 06-JAN-99 7674978787 Rajahmundry

6 rows selected.

SQL> update student set Name = 'Ranveer' where RegNo = 312;

1 row updated.

SQL> select \* from student;

REGNO NAME G DOB MOBILENO CITY

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

312 Ranveer M 20-DEC-00 8096735597 Rajahmundry

9531 Sarika F 15-SEP-15 9848035597 Rajahmundry

8088 Satya Vani F 31-DEC-86 9705710159 Rajahmundry

2609 Durga Rao M 26-SEP-73 9949028509 Rajahmundry

3001 Sanju M 30-JAN-02 9884792252 Rajahmundry

601 Varija Sri F 06-JAN-99 7674978787 Rajahmundry

6 rows selected.

SQL> update student set dob = to\_date('1983-05-01','yyyy-mm-dd') where Name = 'Ram';

0 rows updated.

SQL> update student set Name = 'Ram' where RegNo = 312;

1 row updated.

SQL> select \* from student;

REGNO NAME G DOB MOBILENO CITY

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

312 Ram M 20-DEC-00 8096735597 Rajahmundry

9531 Sarika F 15-SEP-15 9848035597 Rajahmundry

8088 Satya Vani F 31-DEC-86 9705710159 Rajahmundry

2609 Durga Rao M 26-SEP-73 9949028509 Rajahmundry

3001 Sanju M 30-JAN-02 9884792252 Rajahmundry

601 Varija Sri F 06-JAN-99 7674978787 Rajahmundry

6 rows selected.

SQL> update student set dob = to\_date('1983-05-01','yyyy-mm-dd') where Name = 'Ram';

1 row updated.

SQL> select \* from student;

REGNO NAME G DOB MOBILENO CITY

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

312 Ram M 01-MAY-83 8096735597 Rajahmundry

9531 Sarika F 15-SEP-15 9848035597 Rajahmundry

8088 Satya Vani F 31-DEC-86 9705710159 Rajahmundry

2609 Durga Rao M 26-SEP-73 9949028509 Rajahmundry

3001 Sanju M 30-JAN-02 9884792252 Rajahmundry

601 Varija Sri F 06-JAN-99 7674978787 Rajahmundry

6 rows selected.

SQL> describe emp;

Name Null? Type

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

EMPNO NUMBER(38)

ENAME NOT NULL VARCHAR2(20)

JOB NOT NULL VARCHAR2(20)

DEPTNO VARCHAR2(3)

SAL NUMBER(38)

SQL> select \* from emp;

no rows selected

SQL> insert into emp

2 values (1, 'Aditya', 'ML', 101, 100000);

1 row created.

SQL> insert into emp

2 values (10, 'Rahul', 'AI', 111, 10000);

1 row created.

SQL> insert into emp

2 values (200, 'Aman', 'AI', 111, 50000);

1 row created.

SQL> insert into emp

2 values (200, 'Aman', 'AI', 111, 50000);

1 row created.

SQL> insert into emp

2 values (250, 'Lenar', 'Mechanic', 001, 20000);

1 row created.

SQL> select \* from emp;

EMPNO ENAME JOB DEP SAL

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

1 Aditya ML 101 100000

10 Rahul AI 111 10000

200 Aman AI 111 50000

200 Aman AI 111 50000

250 Lenar Mechanic 1 20000

SQL> insert into emp

2 values (50, 'Kil', 'Ass. Prof', 201, 2000);

1 row created.

SQL> select \* from emp;

EMPNO ENAME JOB DEP SAL

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

1 Aditya ML 101 100000

10 Rahul AI 111 10000

200 Aman AI 111 50000

200 Aman AI 111 50000

250 Lenar Mechanic 1 20000

50 Kil Ass. Prof 201 2000

6 rows selected.

SQL> update emp set sal = 15000 where job = 'Ass. Prof';

1 row updated.

SQL> select \* from emp;

EMPNO ENAME JOB DEP SAL

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

1 Aditya ML 101 100000

10 Rahul AI 111 10000

200 Aman AI 111 50000

200 Aman AI 111 50000

250 Lenar Mechanic 1 20000

50 Kil Ass. Prof 201 15000

6 rows selected.

SQL> create table employee as (select \* from emp where sal > 12000);

Table created.

SQL> describe employee;

Name Null? Type

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

EMPNO NUMBER(38)

ENAME NOT NULL VARCHAR2(20)

JOB NOT NULL VARCHAR2(20)

DEPTNO VARCHAR2(3)

SAL NUMBER(38)

SQL> select \* from employee;

EMPNO ENAME JOB DEP SAL

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

1 Aditya ML 101 100000

200 Aman AI 111 50000

200 Aman AI 111 50000

250 Lenar Mechanic 1 20000

50 Kil Ass. Prof 201 15000

SQL> select ename, job from emp;

ENAME JOB

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

Aditya ML

Rahul AI

Aman AI

Aman AI

Lenar Mechanic

Kil Ass. Prof

6 rows selected.

SQL> spool off;

###### Result:

DML (Data Manipulation Language) using SQL COMMANDS has been studied and implemented.

**EXERCISE-3**

**AIM:** To write SQL queries to execute different DCL and TCL commands.

**Explanation:** Database created for this exercise is:

**Data Control Language (DCL) Commands:**

DCL includes commands such as GRANT and REVOKE which mainly deal with the rights, permissions, and other controls of the database system.

List of DCL commands:

* [**GRANT:**](https://www.geeksforgeeks.org/mysql-grant-revoke-privileges/) This command gives users access privileges to the database.
* [**REVOKE:**](https://www.geeksforgeeks.org/difference-between-grant-and-revoke/)This command withdraws the user’s access privileges given by using the GRANT command.

**Transaction Control Language (TCL) Commands:**

* [**COMMIT**](https://www.geeksforgeeks.org/sql-transactions/)**:** Commits a Transaction.
* [**ROLLBACK**](https://www.geeksforgeeks.org/sql-transactions/)**:** Rollbacks a transaction in case of any error occurs.
* [**SAVEPOINT**](https://www.geeksforgeeks.org/sql-transactions/)**:**Sets a savepoint within a transaction.

**Lab Experiment:**

SQL> create table customers

2 (customer\_id int,

3 sale\_date date,

4 sale\_amount int,

5 salesperson varchar(25),

6 store\_state varchar(5),

7 order\_id int

8 );

Table created.

SQL> insert into customers values (1001, to\_date('2020-05-23', 'yyyy-mm-dd'), 1200, 'Raj K', 'KA', 1001);

1 row created.

SQL> insert into customers values (1001, to\_date('2020-05-22', 'yyyy-mm-dd'), 1200, 'M K', NULL, 1002);

1 row created.

SQL> insert into customers values (1002, to\_date('2020-05-23', 'yyyy-mm-dd'), 1200, 'Malika Rakesh', 'MH', 1003);

1 row created.

SQL> insert into customers values (1003, to\_date('2020-05-22', 'yyyy-mm-dd'), 1500, 'Malika Rakesh', 'MH', 1004);

1 row created.

SQL> insert into customers values (1004, to\_date('2020-05-22', 'yyyy-mm-dd'), 1210, 'M K', NULL, 1003);

1 row created.

SQL> insert into customers values (1005, to\_date('2019-12-12', 'yyyy-mm-dd'), 4200, 'RK Rakesh', 'MH', 1007);

1 row created.

SQL> insert into customers values (1002, to\_date('2020-05-21', 'yyyy-mm-dd'), 1200, 'Molly Samberg', 'DL', 1001);

1 row created.

SQL> describe customers;

Name Null? Type

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

CUSTOMER\_ID NUMBER(38)

SALE\_DATE DATE

SALE\_AMOUNT NUMBER(38)

SALESPERSON VARCHAR2(25)

STORE\_STATE VARCHAR2(5)

ORDER\_ID NUMBER(38)

SQL> select \* from customers;

CUSTOMER\_ID SALE\_DATE SALE\_AMOUNT SALESPERSON STORE ORDER\_ID

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

1001 23-MAY-20 1200 Raj K KA 1001

1001 22-MAY-20 1200 M K 1002

1002 23-MAY-20 1200 Malika Rakesh MH 1003

1003 22-MAY-20 1500 Malika Rakesh MH 1004

1004 22-MAY-20 1210 M K 1003

1005 12-DEC-19 4200 RK Rakesh MH 1007

1002 21-MAY-20 1200 Molly Samberg DL 1001

7 rows selected.

SQL> insert into customers values (1002, to\_date('2020-05-20', 'yyyy-mm-dd'), 1200, 'Molly Samberg', 'DL', 1005);

1 row created.

SQL> commit;

Commit complete.

SQL> select \* from customers;

CUSTOMER\_ID SALE\_DATE SALE\_AMOUNT SALESPERSON STORE ORDER\_ID

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

1001 23-MAY-20 1200 Raj K KA 1001

1001 22-MAY-20 1200 M K 1002

1002 23-MAY-20 1200 Malika Rakesh MH 1003

1003 22-MAY-20 1500 Malika Rakesh MH 1004

1004 22-MAY-20 1210 M K 1003

1005 12-DEC-19 4200 RK Rakesh MH 1007

1002 21-MAY-20 1200 Molly Samberg DL 1001

1002 20-MAY-20 1200 Molly Samberg DL 1005

8 rows selected.

SQL> rollback;

Rollback complete.

SQL> select \* from customers;

CUSTOMER\_ID SALE\_DATE SALE\_AMOUNT SALESPERSON STORE ORDER\_ID

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

1001 23-MAY-20 1200 Raj K KA 1001

1001 22-MAY-20 1200 M K 1002

1002 23-MAY-20 1200 Malika Rakesh MH 1003

1003 22-MAY-20 1500 Malika Rakesh MH 1004

1004 22-MAY-20 1210 M K 1003

1005 12-DEC-19 4200 RK Rakesh MH 1007

1002 21-MAY-20 1200 Molly Samberg DL 1001

1002 20-MAY-20 1200 Molly Samberg DL 1005

8 rows selected.

SQL> delete from customers where store\_state = 'MH' and customer\_id = 1002;

1 row deleted.

SQL> select \* from customers;

CUSTOMER\_ID SALE\_DATE SALE\_AMOUNT SALESPERSON STORE ORDER\_ID

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

1001 23-MAY-20 1200 Raj K KA 1001

1001 22-MAY-20 1200 M K 1002

1003 22-MAY-20 1500 Malika Rakesh MH 1004

1004 22-MAY-20 1210 M K 1003

1005 12-DEC-19 4200 RK Rakesh MH 1007

1002 21-MAY-20 1200 Molly Samberg DL 1001

1002 20-MAY-20 1200 Molly Samberg DL 1005

7 rows selected.

SQL> rollback;

Rollback complete.

SQL> select \* from customers;

CUSTOMER\_ID SALE\_DATE SALE\_AMOUNT SALESPERSON STORE ORDER\_ID

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

1001 23-MAY-20 1200 Raj K KA 1001

1001 22-MAY-20 1200 M K 1002

1002 23-MAY-20 1200 Malika Rakesh MH 1003

1003 22-MAY-20 1500 Malika Rakesh MH 1004

1004 22-MAY-20 1210 M K 1003

1005 12-DEC-19 4200 RK Rakesh MH 1007

1002 21-MAY-20 1200 Molly Samberg DL 1001

1002 20-MAY-20 1200 Molly Samberg DL 1005

8 rows selected.

SQL> savepoint sp1;

Savepoint created.

SQL> delete from customers where store\_state = 'MH' and customer\_id = 1002;

1 row deleted.

SQL> savepoint sp2;

Savepoint created.

SQL> rollback to sp1;

Rollback complete.

SQL> select \* from customers;

CUSTOMER\_ID SALE\_DATE SALE\_AMOUNT SALESPERSON STORE ORDER\_ID

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

1001 23-MAY-20 1200 Raj K KA 1001

1001 22-MAY-20 1200 M K 1002

1002 23-MAY-20 1200 Malika Rakesh MH 1003

1003 22-MAY-20 1500 Malika Rakesh MH 1004

1004 22-MAY-20 1210 M K 1003

1005 12-DEC-19 4200 RK Rakesh MH 1007

1002 21-MAY-20 1200 Molly Samberg DL 1001

1002 20-MAY-20 1200 Molly Samberg DL 1005

8 rows selected.

SQL> grant select on customers to RA1911026010101;

Grant succeeded.

SQL> grant select on customers to RA1911026010100;

Grant succeeded.

SQL> revoke select on customers from RA1911026010100;

Revoke succeeded.

SQL> spool off;

**Result:** Thus the DCL and TCL commands are used to modify or manipulate data records present in the customer database tables.

**EXERCISE – 4**

**Aim: Built-In functions in SQL**

###### Functions

Functions accept zero or more arguments and both return one or more results. Both are used to manipulate individual data items.Operators differ from functional in that they follow the format of function\_name(arg..). Functions can be classified into **single row functions and group functions**.

###### Single Row functions

The single row function can be broadly classified as, o Date Function o Numeric Function o Character Function o Conversion Function o Miscellaneous Function

The example that follows mostly uses the symbol table “**dual**”. It is a table, which is automatically created by oracle along with the data dictionary

###### Date Function

* **Add\_month**

This function returns a date after adding a specified date with a specified number of months.

**Syntax:** Add\_months(d,n); where d-date n-number of months

**Example:** *Select add\_months(sysdate,2) from dual;*

###### last\_day

It displays the last date of that month.

**Syntax:** last\_day (d); where d-date

**Example:** *Select last\_day (‘1-jun-2009’) from dual;*

###### Months\_between

It gives the difference in the number of months between d1 & d2.

**Syntax:** month\_between (d1,d2); where d1 & d2 –dates

**Example:** *Select month\_between (‘1-jun-2009’,’1-aug-2009’) from dual;*

###### next\_day

It returns a day followed the specified date.

**Syntax**: next\_day (d,day);

**Example:** *Select next\_day (sysdate,’wednesday’) from dual*

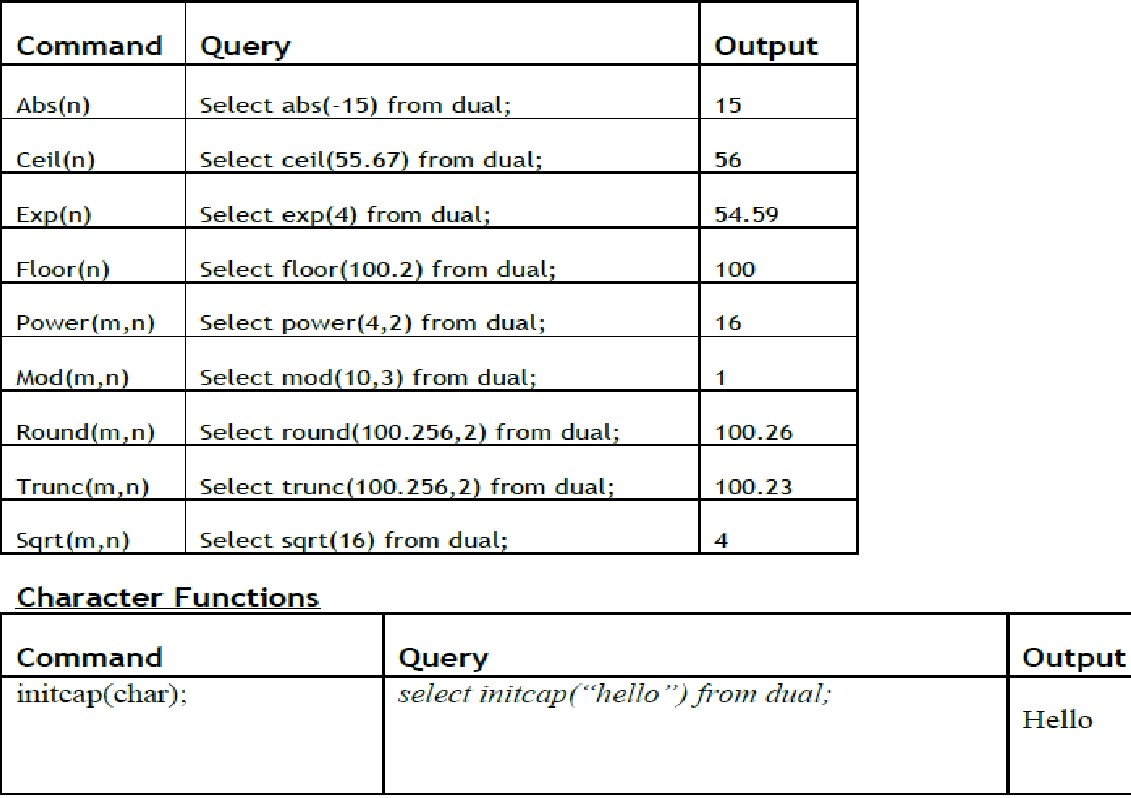
###### round

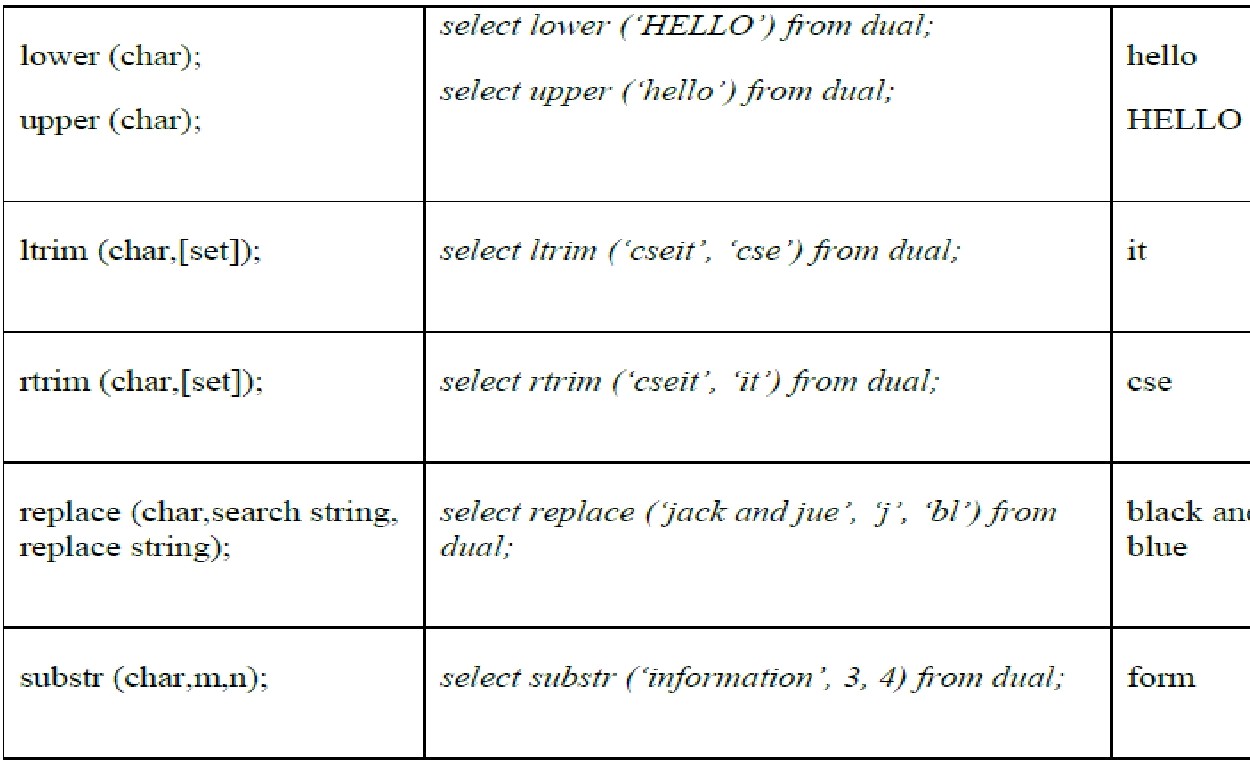
This function returns the date, which is rounded to the unit specified by the format model.

**Syntax :** round (d,[fmt]);

where d- date, [fmt] – optional. By default date will be rounded to the nearest day **Example:** *Select round (to\_date(‘1-jun-2009’,’dd-mm-yy’),’year’) from dual; Select round (‘1-jun-2009’,’year’) from dual;*

###### Numerical Functions





**Conversion Function**

###### to\_char()

**Syntax**: to\_char(d,[format]);

This function converts date to a value of varchar type in a form specified by date format. If format is negelected then it converts date to varchar2 in the default date format.

**Example**: *select to\_char (sysdate, ’dd-mm-yy’) from dual;*

###### to\_date()

**Syntax:** to\_date(d,[format]);

This function converts character to date data format specified in the form character.

**Example:** *select to\_date(‘aug 15 2009’,’mm-dd-yy’) from dual;*

###### Miscellaneous Functions

* **uid** – This function returns the integer value (id) corresponding to the user currently logged in.

**Example:** *select uid from dual;*

* **user** – This function returns the logins user name. **Example:** *select user from dual;*
* **nvl** – The null value function is mainly used in the case where we want to consider null values as zero.

**Syntax;** nvl(exp1, exp2)

If exp1 is null, return exp2. If exp1 is not null, return exp1.

**Example:** *select custid, shipdate, nvl(total,0) from order;*

* **vsize:** It returns the number of bytes in expression.

**Example:** *select vsize(‘tech’) from dual;*

###### Group Functions

A group function returns a result based on group of rows.

###### avg

**Example:** *select avg (total) from student;*

###### max

**Example**: *select max (percentagel) from student;*

###### min

**Example:** *select min (marksl) from student;*

###### sum

**Example:** *select sum(price) from product;*

###### Count Function

In order to count the number of rows, count function is used.

* **count(\*)** – It counts all, inclusive of duplicates and nulls.

**Example:** *select count(\*) from student;*

* **count(col\_name)–** It avoids null value. **Example**: *select count(total) from order;*
* **count(distinct col\_name)** – It avoids the repeated and null values. **Example:** *select count(distinct ordid) from order;*

###### Group by clause

This allows us to use simultaneous column name and group functions.

**Example:** *Select max(percentage), deptname from student group by deptname;*

###### Having clause

This is used to specify conditions on rows retrieved by using group by clause.

**Example:** *Select max(percentage), deptname from student group by deptname having count(\*)>=50;*

###### Special Operators:

**In / not in** – used to select a equi from a specific set of values

**Any -** used to compare with a specific set of values **Between / not between** – used to find between the ranges **Like / not like –** used to do the pattern matching

###### Lab Experiment:

SQL> select add\_months(sysdate, 2) from dual;

ADD\_MONTH

---------

08-APR-22

SQL> select last\_day ('1-jun-2009') from dual;

LAST\_DAY(

---------

30-JUN-09

SQL> select months\_between ('1-jun-2009', '1-aug-2009') from dual;

MONTHS\_BETWEEN('1-JUN-2009','1-AUG-2009')

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

-2

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

NEXT\_DAY(

---------

09-FEB-22

SQL> select round (to\_date('1-jun-2009', 'dd-mm-yyyy'), 'year') from dual;

ROUND(TO\_

---------

01-JAN-09

SQL> select to\_char (sysdate, 'dd-mm-yy') from dual;

TO\_CHAR(

--------

08-02-22

SQL> select uid from dual;

UID

----------

111

SQL> select user from dual;

USER

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

RA1911026010101

SQL> select \* from student;

REGNO NAME G DOB MOBILENO CITY

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

312 Ram M 01-MAY-83 8096735597 Rajahmundry

9531 Sarika F 15-SEP-15 9848035597 Rajahmundry

8088 Satya Vani F 31-DEC-86 9705710159 Rajahmundry

2609 Durga Rao M 26-SEP-73 9949028509 Rajahmundry

3001 Sanju M 30-JAN-02 9884792252 Rajahmundry

601 Varija Sri F 06-JAN-99 7674978787 Rajahmundry

6 rows selected.

SQL> select \* from employee;

EMPNO ENAME JOB DEP SAL

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

1 Aditya ML 101 100000

200 Aman AI 111 50000

200 Aman AI 111 50000

250 Lenar Mechanic 1 20000

50 Kil Ass. Prof 201 15000

SQL> select avg(sal) from employee;

AVG(SAL)

----------

47000

SQL> select max(sal) from employee;

MAX(SAL)

----------

100000

SQL> select min(sal) from employee;

MIN(SAL)

----------

15000

SQL> select sum(sal) from employee;

SUM(SAL)

----------

235000

SQL> select count(\*) from employee;

COUNT(\*)

----------

5

SQL> select count(\*) from student;

COUNT(\*)

----------

6

SQL> select count(regno) from student;

COUNT(REGNO)

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

6

SQL> select count(empno) from employee;

COUNT(EMPNO)

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

5

SQL> select count(distinct regno) from student;

COUNT(DISTINCTREGNO)

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

6

SQL> select count(distinct empno) from employee;

COUNT(DISTINCTEMPNO)

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

4

SQL> spool off;

SQL> select \* from employee;

EMPNO ENAME JOB DEP SAL

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

1 Aditya ML 101 100000

200 Aman AI 111 50000

200 Aman AI 111 50000

250 Lenar Mechanic 1 20000

50 Kil Ass. Prof 201 15000

SQL> select \* from student;

REGNO NAME G DOB MOBILENO CITY

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

312 Ram M 01-MAY-83 8096735597 Rajahmundry

9531 Sarika F 15-SEP-15 9848035597 Rajahmundry

8088 Satya Vani F 31-DEC-86 9705710159 Rajahmundry

2609 Durga Rao M 26-SEP-73 9949028509 Rajahmundry

3001 Sanju M 30-JAN-02 9884792252 Rajahmundry

601 Varija Sri F 06-JAN-99 7674978787 Rajahmundry

6 rows selected.

SQL> select \* from student where name like 'S%';

REGNO NAME G DOB MOBILENO CITY

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

9531 Sarika F 15-SEP-15 9848035597 Rajahmundry

8088 Satya Vani F 31-DEC-86 9705710159 Rajahmundry

3001 Sanju M 30-JAN-02 9884792252 Rajahmundry

SQL> select \* from student where name not like 'S%';

REGNO NAME G DOB MOBILENO CITY

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

312 Ram M 01-MAY-83 8096735597 Rajahmundry

2609 Durga Rao M 26-SEP-73 9949028509 Rajahmundry

601 Varija Sri F 06-JAN-99 7674978787 Rajahmundry

SQL> select \* from employee where empno between 150 and 250;

EMPNO ENAME JOB DEP SAL

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

200 Aman AI 111 50000

200 Aman AI 111 50000

250 Lenar Mechanic 1 20000

SQL> select sqrt(sal) from employee;

SQRT(SAL)

----------

316.227766

223.606798

223.606798

141.421356

122.474487

SQL> select count(\*) from employee;

COUNT(\*)

----------

5

SQL> select sum(sal), avg(sal) from employee;

SUM(SAL) AVG(SAL)

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

235000 47000

SQL> select max(sal) as max\_salary, min(sal) as min\_salary from employee;

MAX\_SALARY MIN\_SALARY

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

100000 15000

SQL> select sum(sal) from employee;

SUM(SAL)

----------

235000

SQL> select add\_months (sysdate, 2) from dual;

ADD\_MONTH

---------

08-APR-22

SQL> select add\_months (sysdate, -2) from dual;

ADD\_MONTH

---------

08-DEC-21

SQL> select \* from customers;

CUSTOMER\_ID SALE\_DATE SALE\_AMOUNT SALESPERSON STORE ORDER\_ID

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

1001 23-MAY-20 1200 Raj K KA 1001

1001 22-MAY-20 1200 M K 1002

1002 23-MAY-20 1200 Malika Rakesh MH 1003

1003 22-MAY-20 1500 Malika Rakesh MH 1004

1004 22-MAY-20 1210 M K 1003

1005 12-DEC-19 4200 RK Rakesh MH 1007

1002 21-MAY-20 1200 Molly Samberg DL 1001

1002 20-MAY-20 1200 Molly Samberg DL 1005

8 rows selected.

SQL> select min(sale\_amount) from customers group by salesperson;

MIN(SALE\_AMOUNT)

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

1200

1200

4200

1200

1200

SQL> select min(sale\_amount), salesperson from customers group by salesperson;

MIN(SALE\_AMOUNT) SALESPERSON

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

1200 Molly Samberg

1200 M K

4200 RK Rakesh

1200 Raj K

1200 Malika Rakesh

SQL> spool off;

**Result:**

The Built-in Functions in SQL have been implemented.

###### EXERCISE - 5

**Aim**: To draw an er-diagram for a Hotel Management System.

**Members**: Aditya Manoj Bhaskaran (RA1911026010101)

Abhisek Biswal (RA1911026010109)

**PROCEDURE:**

Entities:  Teacher, Supervisor (Weak Entity), Evaluator (Weak Entity), Room, Student, Paper

Relationships:

* The teacher is a generaliazation of the supervisor and the evaluator entities and has an ‘IS A’ relationship with them. The evaluator and supervisor entities are a specialization of the teacher entity.
* The supervisor has the relationship of ‘supervises’ with the room entity. It has a one-to-many relationship. It is a weak relationship.
* The evaluator has the relationship of ‘corrects’ with the paper entity. It has a many-to-many relationship. It is a weak relationship.
* The instances of the student entity sit in the room entity. It has a many-to-one relationship.
* The student entity has a relationship of ‘gives’ with the paper entity. It has a one-to-many relationship.

Attributes:

Teacher:

* Name
* Gender
* Teacher\_ID (Primary Key)

Supervisor:

* Room\_No (Foreign Key)

Evaluator:

* Course\_ID (Foreign Key)

Room:

* Room\_No (Primary Key)
* RegNo (Foreign Key)
* Teacher\_ID (Foreign Key)

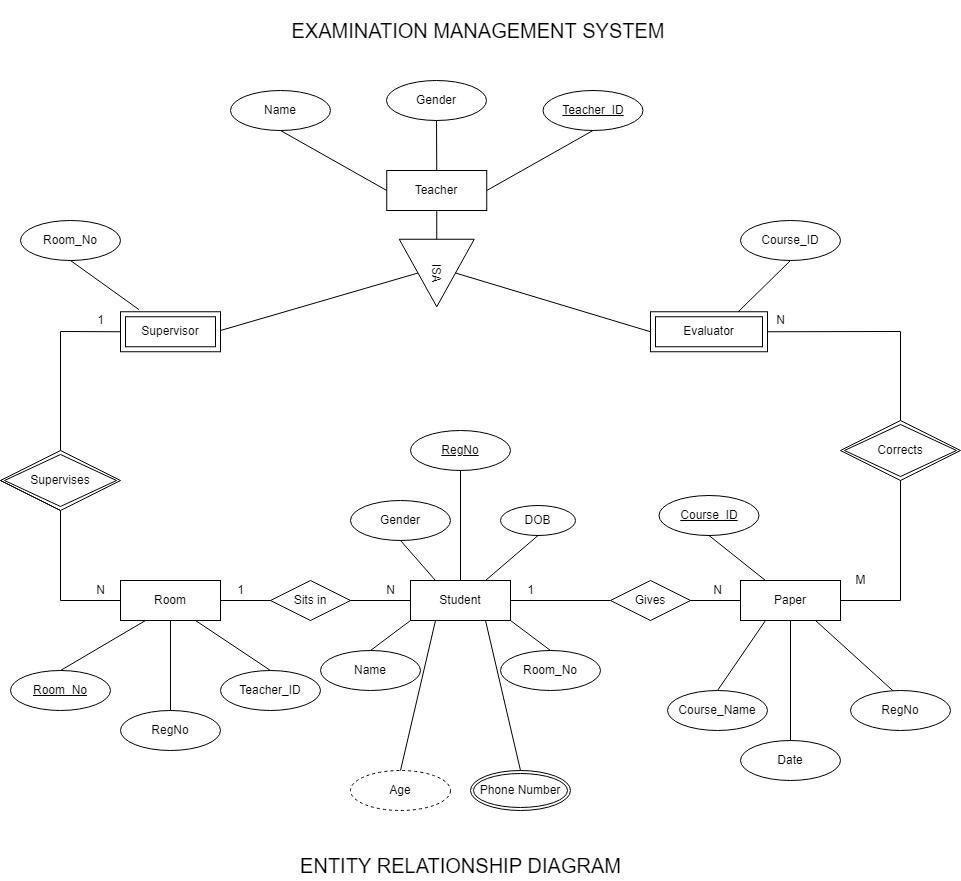
Paper:

* Course\_ID (Primary Key)
* Course\_Name
* Date
* RegNo (Foreign Key)

Student:

* RegNo (Primary Key)
* Gender
* DOB
* Name
* Room\_No (Foreign Key)
* Age (Derived Attribute)
* Phone Number (Multi-valued Attribute)

EER Diagram:



**RESULT** – The Extended ER Diagram for Bookstore Management System has been studied.

**EXERCISE – 6**

**AIM -** To study JOIN QUERIES in SQL.

**EXPLANATION -** SQL Join is used to fetch data from two or more tables, which is joined to appear as single set of data. SQL Join is used for combining column from two or more tables by using values common to both tables. **Join** Keyword is used in SQL queries for joining two or more tables. Minimum required condition for joining table, is **(n-1)** where **n**, is number of tables. A table can also join to itself known as, **Self Join**.

**Types of Join**

The following are the types of JOIN that we can use in SQL.

* Inner
* Outer
* Left
* Right

**Cross JOIN or Cartesian Product**

This type of JOIN returns the cartesian product of rows from the tables in Join. It will return a table which consists of records which combines each row from the first table with each row of the second table.

Cross JOIN Syntax is,

SELECT column-name-list from table-name1

**INNER Join or EQUI Join**

This is a simple JOIN in which the result is based on matched data as per the equality condition specified in the query.

Inner Join Syntax is,

SELECT column-name-list from table-name1

**Natural JOIN**

Natural Join is a type of Inner join which is based on column having same name and same datatype present in both the tables to be joined.

Natural Join Syntax is, SELECT \* from table-name1

**NATURAL JOIN**

**Natural join query will be,**

SELECT \* from class NATURAL JOIN class\_info;

**Outer JOIN**

Outer Join is based on both matched and unmatched data. Outer Joins subdivide further into,

* Left Outer Join
* Right Outer Join
* Full Outer Join

**Left Outer Join**

The left outer join returns a result table with the **matched data** of two tables then remaining rows of the **left** table and null for the **right** table's column.

Left Outer Join syntax is,

SELECT column-name-list from table-name1

**LEFT OUTER JOIN**

Left outer Join Syntax for **Oracle** is,

select column-name-list from table-name1, table-name2 on table-name1.column-name = table- name2.column-name(**+**);

**Left Outer Join** query will be,

SELECT \* FROM class LEFT OUTER JOIN class\_info ON ([class.id](http://class.id/)=class\_info.id);

**Right Outer Join**

The right outer join returns a result table with the **matched data** of two tables then remaining rows of the **right table** and null for the **left** table's columns.

Right Outer Join Syntax is, select column-name-list from table-name1

**RIGHT OUTER JOIN**

**Right Outer Join** query will be,

SELECT \* FROM class RIGHT OUTER JOIN class\_info on ([class.id](http://class.id/)=class\_info.id); The result table will look like,

**Full Outer Join**

The full outer join returns a result table with the **matched data** of two table then remaining rows of both **left** table and then the **right** table. Full Outer Join Syntax is, select column-name- list from table-name1

**FULL OUTER JOIN**

**Full Outer Join** query will be like,

SELECT \* FROM class FULL OUTER JOIN class\_info on ([class.id](http://class.id/)=class\_info.id);

###### Lab Experiment:

SQL> create table orders

2 (

3 order\_id int primary key,

4 order\_number int,

5 p\_id varchar(20) not null

6 );

Table created.

SQL> create table person

2 (

3 p\_id varchar(20) primary key,

4 firstname varchar(15),

5 lastname varchar(15),

6 city varchar(10)

7 );

Table created.

SQL> desc orders;

Name Null? Type

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

ORDER\_ID NOT NULL NUMBER(38)

ORDER\_NUMBER NUMBER(38)

P\_ID NOT NULL VARCHAR2(20)

SQL> desc person;

Name Null? Type

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

P\_ID NOT NULL VARCHAR2(20)

FIRSTNAME VARCHAR2(15)

LASTNAME VARCHAR2(15)

CITY VARCHAR2(10)

SQL> insert into orders values(101, 1, 'A1');

1 row created.

SQL> insert into orders values(102, 2, 'A2');

1 row created.

SQL> insert into orders values(105, 3, 'B2');

1 row created.

SQL> insert into orders values(115, 5, 'C2');

1 row created.

SQL> insert into orders values(120, 6, 'D1');

1 row created.

SQL> select \* from orders;

ORDER\_ID ORDER\_NUMBER P\_ID

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

101 1 A1

102 2 A2

105 3 B2

115 5 C2

120 6 D1

SQL> insert into person values('A1', 'Aditya', 'Bhaskaran', 'Kol');

1 row created.

SQL> insert into person values('A2', 'Alex', 'Kohr', 'Del');

1 row created.

SQL> insert into person values('B1', 'Chandan', 'Kamal', 'Jai');

1 row created.

SQL> insert into person values('C1', 'Venkata', 'Raman', 'Chen');

1 row created.

SQL> insert into person values('D1', 'John', 'Kramer', 'NY');

1 row created.

SQL> insert into person values('D2', 'Okabe', 'Rintarou', 'Shib');

1 row created.

SQL> select \* from person;

P\_ID FIRSTNAME LASTNAME CITY

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

A1 Aditya Bhaskaran Kol

A2 Alex Kohr Del

B1 Chandan Kamal Jai

C1 Venkata Raman Chen

D1 John Kramer NY

D2 Okabe Rintarou Shib

6 rows selected.

SQL> select \* from orders cross join person;

ORDER\_ID ORDER\_NUMBER P\_ID P\_ID

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

FIRSTNAME LASTNAME CITY

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

101 1 A1 A1

Aditya Bhaskaran Kol

101 1 A1 A2

Alex Kohr Del

101 1 A1 B1

Chandan Kamal Jai

ORDER\_ID ORDER\_NUMBER P\_ID P\_ID

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

FIRSTNAME LASTNAME CITY

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

101 1 A1 C1

Venkata Raman Chen

101 1 A1 D1

John Kramer NY

101 1 A1 D2

Okabe Rintarou Shib

ORDER\_ID ORDER\_NUMBER P\_ID P\_ID

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

FIRSTNAME LASTNAME CITY

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

102 2 A2 A1

Aditya Bhaskaran Kol

102 2 A2 A2

Alex Kohr Del

102 2 A2 B1

Chandan Kamal Jai

ORDER\_ID ORDER\_NUMBER P\_ID P\_ID

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

FIRSTNAME LASTNAME CITY

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

102 2 A2 C1

Venkata Raman Chen

102 2 A2 D1

John Kramer NY

102 2 A2 D2

Okabe Rintarou Shib

ORDER\_ID ORDER\_NUMBER P\_ID P\_ID

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

FIRSTNAME LASTNAME CITY

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

105 3 B2 A1

Aditya Bhaskaran Kol

105 3 B2 A2

Alex Kohr Del

105 3 B2 B1

Chandan Kamal Jai

ORDER\_ID ORDER\_NUMBER P\_ID P\_ID

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

FIRSTNAME LASTNAME CITY

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

105 3 B2 C1

Venkata Raman Chen

105 3 B2 D1

John Kramer NY

105 3 B2 D2

Okabe Rintarou Shib

ORDER\_ID ORDER\_NUMBER P\_ID P\_ID

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

FIRSTNAME LASTNAME CITY

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

115 5 C2 A1

Aditya Bhaskaran Kol

115 5 C2 A2

Alex Kohr Del

115 5 C2 B1

Chandan Kamal Jai

ORDER\_ID ORDER\_NUMBER P\_ID P\_ID

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

FIRSTNAME LASTNAME CITY

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

115 5 C2 C1

Venkata Raman Chen

115 5 C2 D1

John Kramer NY

115 5 C2 D2

Okabe Rintarou Shib

ORDER\_ID ORDER\_NUMBER P\_ID P\_ID

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

FIRSTNAME LASTNAME CITY

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

120 6 D1 A1

Aditya Bhaskaran Kol

120 6 D1 A2

Alex Kohr Del

120 6 D1 B1

Chandan Kamal Jai

ORDER\_ID ORDER\_NUMBER P\_ID P\_ID

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

FIRSTNAME LASTNAME CITY

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

120 6 D1 C1

Venkata Raman Chen

120 6 D1 D1

John Kramer NY

120 6 D1 D2

Okabe Rintarou Shib

30 rows selected.

SQL> select \* from orders inner join person using(p\_id);

P\_ID ORDER\_ID ORDER\_NUMBER FIRSTNAME LASTNAME

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

CITY

----------

A1 101 1 Aditya Bhaskaran

Kol

A2 102 2 Alex Kohr

Del

D1 120 6 John Kramer

NY

SQL> select \* from orders inner join person using(p\_id);

P\_ID ORDER\_ID ORDER\_NUMBER FIRSTNAME LASTNAME

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

CITY

----------

A1 101 1 Aditya Bhaskaran

Kol

A2 102 2 Alex Kohr

Del

D1 120 6 John Kramer

NY

SQL> select \* from orders natural join person;

P\_ID ORDER\_ID ORDER\_NUMBER FIRSTNAME LASTNAME

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

CITY

----------

A1 101 1 Aditya Bhaskaran

Kol

A2 102 2 Alex Kohr

Del

D1 120 6 John Kramer

NY

SQL> select \* from orders left outer join person on (orders.p\_id = person.p\_id);

ORDER\_ID ORDER\_NUMBER P\_ID P\_ID

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

FIRSTNAME LASTNAME CITY

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

101 1 A1 A1

Aditya Bhaskaran Kol

102 2 A2 A2

Alex Kohr Del

120 6 D1 D1

John Kramer NY

ORDER\_ID ORDER\_NUMBER P\_ID P\_ID

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

FIRSTNAME LASTNAME CITY

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

115 5 C2

105 3 B2

SQL> select \* from orders right outer join person on (orders.p\_id = person.p\_id);

ORDER\_ID ORDER\_NUMBER P\_ID P\_ID

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

FIRSTNAME LASTNAME CITY

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

101 1 A1 A1

Aditya Bhaskaran Kol

102 2 A2 A2

Alex Kohr Del

120 6 D1 D1

John Kramer NY

ORDER\_ID ORDER\_NUMBER P\_ID P\_ID

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

FIRSTNAME LASTNAME CITY

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

B1

Chandan Kamal Jai

C1

Venkata Raman Chen

D2

Okabe Rintarou Shib

6 rows selected.

SQL> select \* from orders full outer join person on (orders.p\_id = person.p\_id);

ORDER\_ID ORDER\_NUMBER P\_ID P\_ID

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

FIRSTNAME LASTNAME CITY

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

101 1 A1 A1

Aditya Bhaskaran Kol

102 2 A2 A2

Alex Kohr Del

B1

Chandan Kamal Jai

ORDER\_ID ORDER\_NUMBER P\_ID P\_ID

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

FIRSTNAME LASTNAME CITY

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

C1

Venkata Raman Chen

120 6 D1 D1

John Kramer NY

D2

Okabe Rintarou Shib

ORDER\_ID ORDER\_NUMBER P\_ID P\_ID

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

FIRSTNAME LASTNAME CITY

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

115 5 C2

105 3 B2

8 rows selected.

SQL> spool off;

**Result:** JOIN QUERIES in SQL have been successfully implemented.

**EXERCISE- 7**

**Aim: To study SQL SUBQUERIES**

**Subquery** or **Inner query** or **Nested query** is a query in a query. SQL subquery is usually added in the [WHERE](http://beginner-sql-tutorial.com/sql-where-clause.htm) Clause of the SQL statement. Most of the time, a subquery is used when you know how to search for a value using a SELECT statement, but do not know the exact value in the database.

**Subqueries** are an alternate way of returning data from multiple tables.

Subqueries can be used with the following SQL statements along with the comparision operators like =, <, >, >=, <= etc.

* [SELECT](http://beginner-sql-tutorial.com/sql-select-statement.htm)
* [INSERT](http://beginner-sql-tutorial.com/sql-insert-statement.htm)
* [UPDATE](http://beginner-sql-tutorial.com/sql-update-statement.htm)
* [DELETE](http://beginner-sql-tutorial.com/sql-delete-statement.htm)

2) Let's consider the student\_details table which we have used earlier. If you know the name of the students who are studying science subject, you can get their id's by using this query below,

SELECT id, first\_name FROM student\_details

WHERE first\_name IN ('Rahul', 'Stephen');

but, if you do not know their names, then to get their id's you need to write the query in this manner,

SELECT id, first\_name FROM student\_details

WHERE first\_name IN (SELECT first\_name FROM student\_details

WHERE subject= 'Science');

###### Subquery Output:

|  |  |
| --- | --- |
| **id** | **first\_name** |
|  |  |
| 100 | Rahul |
| 102 | Stephen |

In the above sql statement, first the inner query is processed first and then the outer query is processed.

SQL Subquery; INSERT Statement

* + Subquery can be used with INSERT statement to add rows of data from one or more tables to another table. Lets try to group all the students who study Maths in a table 'maths\_group'.

INSERT INTO maths\_group(id, name) SELECT id, first\_name || ' ' || last\_name

FROM student\_details WHERE subject= 'Maths'

SQL Subquery; SELECT Statement

* + A subquery can be used in the SELECT statement as follows. Lets use the product and order\_items table defined in the sql\_joins section.

|  |  |  |
| --- | --- | --- |
| select p.product\_name, p.supplier\_name, (select order\_id from order\_items where | | |
| product\_id = 101) as order\_id from product p where p.product\_id = 101 | | |
| **product\_name** | **supplier\_name** | **order\_id** |
|  |  |  |
| Television | Onida | 5103 |

###### Correlated Subquery

A query is called correlated subquery when both the inner query and the outer query are interdependent. For every row processed by the inner query, the outer query is processed as well. The inner query depends on the outer query before it can be processed.

SELECT p.product\_name FROM product p

WHERE p.product\_id = (SELECT o.product\_id FROM order\_items o WHERE o.product\_id = p.product\_id);

###### Subquery Notes Nested Subquery

* + You can nest as many queries you want but it is recommended not to nest more than 16 subqueries in oracle

###### Non-Correlated Subquery

* + If a subquery is not dependent on the outer query it is called a non-correlated subquery**Subquery Errors**
  + Minimize subquery errors: Use drag and drop, copy and paste to avoid running subqueries with spelling and database typos. Watch your multiple field SELECT comma use, extra or to few getting SQL error message "Incorrect syntax".

###### SQL Subquery Comments

Adding SQL Subquery comments are good habit (/\* your command comment \*/) which can save you time, clarify your previous work .. results in less SQL headaches

**Nested Queries and Performance Issues in SQL**

**Nested Queries** are queries that contain another complete SELECT statement nested within it, that is, in the WHERE clause. The nested SELECT statement is called an “inner query” or an “inner SELECT.” The main query is called “outer SELECT” or “outer query.” Many nested queries are equivalent to a simple query using JOIN operation. The use of nested query in this case is to avoid explicit coding of JOIN which is a very expensive database operation and to improve query performance. However, in many cases, the use of nested queries is necessary and cannot be replaced by a JOIN operation.

###### Nested queries that can be expressed using JOIN operations:

Example 1: (Library DB Query A) How many copies of the book titled the lost tribe are owned by the library branch whose name is “Sharptown”?

**Single Block Query Using Join:**

SELECT No\_Of\_Copies

FROM BOOK\_COPIES, BOOK, LIBRARY\_BRANCH

WHERE BOOK\_COPIES.BranchId = LIBRARY\_BRANCH.BranchId **AND**

BOOK\_COPIES.BookId = BOOK.BookId **AND**

BOOK.Title = "The Lost Tribe" **AND**

LIBRARY\_BRANCH.BranchName = "Sharpstown";

**Using Nested Queries:**

SELECT No\_Of\_Copies FROM BOOK\_COPIES

WHERE BranchID IN

(SELECT BranchID from LIBRARY\_BRANCH WHERE

LIBRARY\_BRANCH.BranchName = "Sharpstown")

AND BookID IN

(SELECT BookID from BOOK WHERE

BOOK.Title = "The Lost Tribe" ) ;

**Performance considerations:** The nested queries in this example involves simpler and faster operations. Each subquery will be executed once and then a simple select operation will be performed. On the other hands, the operations using join require Cartesian products of three tables and have to evaluate 2 join conditions and 2 selection conditions. Nested queries in this example also save internal temporary memory space for holding Cartesian join results.

###### ====================================================================

**=**

Rule of thumb:

* ***Correlated queries*** *where the inner query references some attribute of a relation declared in the outer query and use the” =“ or IN operators.*
* *Conversely, if the attributes in the projection operation of a single block query that joins several tables are from only one table, this query can always be translated into a nested query.*

====================================================================

=

Example 2: see Query 12 and Query 12A

Retrieve the name of each employee who has a dependent with the same first name and same sex as the employee.

**Single Block query using JOIN operation**

select A.fname, A.lname from employee A, dependent B where A.ssn = B.essn and

A.sex = B.sex and A.fname = B.dependent\_name;

**Correlated Query:** select A.fname, A.lname from employee A where A.ssn **IN** (SELECT essn

FROM dependent

WHERE **essn = A.ssn** and dependent\_name = A.fname and sex = A.sex); Computer Procedures:

Conceptually, think of this query as stepping through the EMPLOYEE table one row at a time, and then executing the inner query each time. The first row has A.fname = “John” and A.sex = “M” so that the inner query becomes **SELECT Essn FROM dependent where essn = 12345678, dependent\_name = “John” and sex = “M”**; The first run of the subquery returns nothing so it continues to proceed to the next tuple and executes the inner query again with the values of A.SSN, A.fname and A.sex for the second row, and so on for all rows of EMPLOYEE.

The term *correlated subquery* is used because its value depends on a variable (or variables) that receives its value from an outer query (e.g., A.SSN, A.fname, A.sex in this example; they are called **correlation variables**.). A correlated subquery thus cannot be evaluated once and for all. It must be evaluated repeatedly -- once for each value of the variable received from the outer query. This is different from non-correlated subqueries explained below.

**Non-correlated Subquery:**

A non-correlated subquery needs to be evaluated only once. For example: Query EMP-NQ2: find an employee that has the highest salary of the company.

SELECT fname, lname, bdate FROM EMPLOYEE

WHERE salary = **(SELECT max (salary) FROM Employee)**;

Here the inner query returns a value: 55000. The inner query will be executed first and only *once*

and then the entire query becomes SELECT fname, lname, bdate

FROM EMPLOYEE WHERE **salary = 55000**;

###### Nested Queries that cannot be directly translated into Join Operations

Rule of thumb:

* Unknown selection criteria: WHERE clause examines unknown value.

For example shown above (Query EMP-NQ2): find everybody in a department which has an employee that has the highest salary of the company.

Another example in section 7.2.5. finds employees who has salary higher than the highest salary in Department 5.

SELECT ssn, salary, dno from Employee where salary > (SELECT max (salary) from employee where dno = 5);

* Relational **set** operations such as Division or other comparison that involves EXISTS, NOT EXISTS, > , etc. (This may involve using paradox SET operation operators, such as NO, ONLY, EXACTLY and EVERY.)
* Outer Join that involves Null value operations. This is the equivalent of using NOT EXISTS. (See *SQL solution for queries on Library DB*: query C and C’).

###### General Discussion on SQL query formulation:

There are many ways to specify the same query in SQL. This flexibility in specifying queries has advantage and disadvantages.

* Advantage: You can choose a way to express the query that you prefer. **It is general preferable to write a query with as little nesting and implied ordering as possible.**
* Disadvantages:
  + the user may be confused
  + users may have the burden to figure out which way is more efficient due to different DBMS query optimization strategies. (Performance issues.)

###### Sample Correlated and Non-correlated Subqueries

Write SQL statements for the following queries on the Company Database and determine whether it’s a correlated or non-correlated query. (Please translate your SQL single-block join, if applicable, to subqueries.)

Tip: the term *correlated subquery* is used because its value depends on a variable (or variables) that receives its value from an outer query (e.g., A.SSN, A.fname, A.sex in the example shown in the previous handout; they are called **correlation variables**.). A correlated subquery thus cannot be evaluated once and for all. It must be evaluated repeatedly -- once for each value of the variable received from the outer query. A non-correlated subquery needs to be evaluated only once.

###### Lab Experiment

SQL> select \* from employee;

EMPNO ENAME JOB DEP SAL

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

1 Aditya ML 101 100000

200 Aman AI 111 50000

200 Aman AI 111 50000

250 Lenar Mechanic 1 20000

50 Kil Ass. Prof 201 15000

SQL> insert into employee values(30, 'Broth', 'Teacher', 80, 65000);

1 row created.

SQL> insert into employee values(430, 'Kristy', 'Teacher', 80, 45000);

1 row created.

SQL> insert into employee values(540, 'Brenda', 'Teacher', 80, 40000);

1 row created.

SQL> select \* from employee;

EMPNO ENAME JOB DEP SAL

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

1 Aditya ML 101 100000

200 Aman AI 111 50000

200 Aman AI 111 50000

250 Lenar Mechanic 1 20000

50 Kil Ass. Prof 201 15000

30 Broth Teacher 80 65000

430 Kristy Teacher 80 45000

540 Brenda Teacher 80 40000

8 rows selected.

SQL> select sal from employee;

SAL

----------

100000

50000

50000

20000

15000

65000

45000

40000

8 rows selected.

SQL> desc employee;

Name Null? Type

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

EMPNO NUMBER(38)

ENAME NOT NULL VARCHAR2(20)

JOB NOT NULL VARCHAR2(20)

DEPTNO VARCHAR2(3)

SAL NUMBER(38)

SQL> select sal from employee where deptno = 80;

SAL

----------

65000

45000

40000

SQL> select avg(sal) from employee where deptno = 80;

AVG(SAL)

----------

50000

SQL> select \* from employee where sal > (select avg(sal) from employee where deptno = 60);

no rows selected

SQL> select \* from employee where sal > 50000;

EMPNO ENAME JOB DEP SAL

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

1 Aditya ML 101 100000

30 Broth Teacher 80 65000

SQL> select \* from employee where sal > (select avg(sal) from employee where deptno = 60);

no rows selected

SQL> select \* from employee where sal > (select avg(sal) from employee where deptno = 80);

EMPNO ENAME JOB DEP SAL

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

1 Aditya ML 101 100000

30 Broth Teacher 80 65000

SQL> insert into emp values(214, 'Ghu', 'Teacher', 80, 46000);

1 row created.

SQL> insert into emp values(345, 'Bu', 'ML', 101, 94000);

1 row created.

SQL> select empno, ename from employee where deptno in (select deptno from employee where employee.ename like '%u');

no rows selected

SQL> select empno, ename from employee where deptno in (select deptno from employee where ename like '%u');

no rows selected

SQL> select deptno from employee where ename like '%u';

no rows selected

SQL> select \* from employee;

EMPNO ENAME JOB DEP SAL

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

1 Aditya ML 101 100000

200 Aman AI 111 50000

200 Aman AI 111 50000

250 Lenar Mechanic 1 20000

50 Kil Ass. Prof 201 15000

30 Broth Teacher 80 65000

430 Kristy Teacher 80 45000

540 Brenda Teacher 80 40000

8 rows selected.

SQL> insert into employee values(214, 'Ghu', 'Teacher', 80, 46000);

1 row created.

SQL> insert into employee values(345, 'Bu', 'ML', 101, 94000);

1 row created.

SQL> select empno, ename from employee where deptno in (select deptno from employee where ename like '%u');

EMPNO ENAME

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

30 Broth

430 Kristy

540 Brenda

214 Ghu

1 Aditya

345 Bu

6 rows selected.

SQL> insert into employee values(248, 'Ham', 'Transport', 170, 67000);

1 row created.

SQL> select ename, deptno, empno from employee where deptno = 170;

ENAME DEP EMPNO

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

Ham 170 248

SQL> create table product

2 ( catno int,

3 name varchar(10),

4 cost int,

5 id int

6 );

Table created.

SQL> insert into product values(100, 'Brush', 100, 64);

1 row created.

SQL> insert into product values(100, 'Paint', 300, 103);

1 row created.

SQL> insert into product values(100, 'Bucket', 500, 123);

1 row created.

SQL> insert into product values(60, 'Mouse', 900, 143);

1 row created.

SQL> insert into product values(60, 'File', 50, 243);

1 row created.

SQL> insert into product values(60, 'Folder', 100, 203);

1 row created.

SQL> insert into product values(80, 'Orange', 50, 23);

1 row created.

SQL> insert into product values(80, 'Apple', 100, 33);

1 row created.

SQL> select \* from product;

CATNO NAME COST ID

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

100 Brush 100 64

100 Paint 300 103

100 Bucket 500 123

60 Mouse 900 143

60 File 50 243

60 Folder 100 203

80 Orange 50 23

80 Apple 100 33

8 rows selected.

SQL> select \* from product where catno in (select catno from product where id = 64);

CATNO NAME COST ID

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

100 Brush 100 64

100 Paint 300 103

100 Bucket 500 123

SQL> select \* from product where cost > (select avg(cost) from product where catno = 60);

CATNO NAME COST ID

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

100 Bucket 500 123

60 Mouse 900 143

SQL> select \* from product where cost in (select cost from product where catno = 80);

CATNO NAME COST ID

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

60 File 50 243

80 Orange 50 23

100 Brush 100 64

60 Folder 100 203

80 Apple 100 33

SQL> select \* from product where cost > (select max(cost) from product where catno = 80);

CATNO NAME COST ID

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

100 Paint 300 103

100 Bucket 500 123

60 Mouse 900 143

SQL> spool off;

**Result -** SQL Subqueries have been studied and implemented.

**EXERCISE – 8**

**Aim: To study SET OPERATIONS and VIEWS in SQL**

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

· **Union**

###### Union all

* **Intersect**

###### Minus

***Rule:***

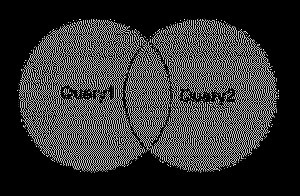
***The queries which are related by the set operators should have a same number of column and column definition.***

**Union:**

Returns all distinct rows selected by both the queries

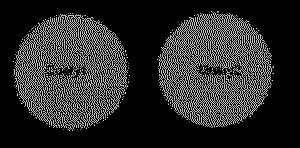
###### Syntax:

Query1 Union Query2;



**Exp:** SELECT \* FROM table1 UNION SELECT \* FROM table2;

**Union all:**



Returns all rows selected by either query including the duplicates.

###### Syntax:

Query1 Union all Query2;

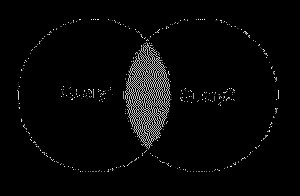
**Exp:** SELECT \* FROM table1 UNION ALL SELECT \* FROM table2;

**Intersect**

Returns rows selected that are common to both queries.

###### Syntax:

Query1 Intersect Query2;



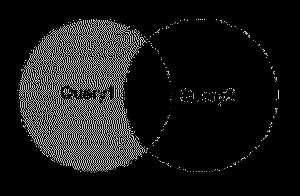
**Exp:** SELECT \* FROM table1 INTERSECT SELECT \* FROM table2;

**Minus**

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

###### Syntax:

Query1 minus Query2;



Exp: SELECT \* FROM table1 MINUS SELECT \* FROM table2;

**VIEWS**

A view is the tailored presentation of data contained in one or more table and can also be said as restricted view to the data’s in the tables. A view is a “virtual table” or a “stored query” which takes the output of a query and treats it as a table. The table upon which a view is created is called as base table.

###### Advantages of a view:

* Additional level of table security.
* Hides data complexity.
* Simplifies the usage by combining multiple tables into a single table.
* Provides data‟s in different perspective.

###### Types of view:

Horizontal -> enforced by where cause

Vertical -> enforced by selecting the required columns

###### SQL Commands for Creating and dropping view:

**Syntax:**

Create [or replace] view <view name> [column alias names] as <query> [with <options> conditions];

Drop view <view name>;

###### Lab Experiment:

SQL> desc employee;

Name Null? Type

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

EMPID NOT NULL NUMBER(38)

EMPNAME VARCHAR2(20)

LOCATION VARCHAR2(10)

DEPT VARCHAR2(10)

SALARY NUMBER(38)

SQL> insert into employee values(101, 'Aditya', 'Kol', 'AI', 1000000);

1 row created.

SQL> insert into employee values(102, 'Adi', 'Kol', 'AI', 500000);

1 row created.

SQL> insert into employee values(103, 'Kalim', 'Guj', 'Mech', 40000);

1 row created.

SQL> insert into employee values(104, 'Burj', 'Bihar', 'Civil', 1000);

1 row created.

SQL> insert into employee values(105, 'Lin', 'Kerela', 'Business', 100000);

1 row created.

SQL> insert into employee values(1000, 'Ayanokouji', 'Japan', 'HR', 10000000);

1 row created.

SQL> select \* from employee;

EMPID EMPNAME LOCATION DEPT SALARY

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

101 Aditya Kol AI 1000000

102 Adi Kol AI 500000

103 Kalim Guj Mech 40000

104 Burj Bihar Civil 1000

105 Lin Kerela Business 100000

1000 Ayanokouji Japan HR 10000000

6 rows selected.

SQL> select empid, empname from employee where salary > 300000 union select empid, empname from employee where salary < 100000;

EMPID EMPNAME

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

101 Aditya

102 Adi

103 Kalim

104 Burj

1000 Ayanokouji

SQL> select empid, empname from employee where dept = 'AI' union select empid, empname from employee where salary > 100000;

EMPID EMPNAME

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

101 Aditya

102 Adi

1000 Ayanokouji

SQL> select empid, empname from employee where dept = 'AI' union all select empid, empname from employee where salary > 100000;

EMPID EMPNAME

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

101 Aditya

102 Adi

101 Aditya

102 Adi

1000 Ayanokouji

SQL> select empid, empname from employee where dept = 'AI' union all select empid, empname from employee where salary > 100000 order by empname;

EMPID EMPNAME

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

102 Adi

102 Adi

101 Aditya

101 Aditya

1000 Ayanokouji

SQL> select empid, empname from employee where dept = 'AI' intersect select empid, empname from employee where salary > 100000 order by empname;

EMPID EMPNAME

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

102 Adi

101 Aditya

SQL> spool off;

SQL> create table student(

2 sid int primary key,

3 fname varchar(20),

4 lname varchar(20),

5 address varchar(20),

6 zip int,

7 phone int

8 );

Table created.

SQL> create table dept(

2 deptid int primary key,

3 deptname varchar(20)

4 );

Table created.

SQL> insert into student values(2, 'Ghaha', 'Bhhahah', 'Somalia', 23020, 1010101034);

1 row created.

SQL> insert into student values(3, 'Oveveveve', 'Ofuefuefuefue', 'Afrika', 23420, 2010101034);

1 row created.

SQL> insert into dept values(100, 'AI');

1 row created.

SQL> insert into dept values(101, 'ML');

1 row created.

SQL> insert into dept values(102, 'Mech');

1 row created.

SQL> insert into dept values(103, 'Anime');

1 row created.

SQL> select \* from student;

SID FNAME LNAME ADDRESS

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

ZIP PHONE

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

1 A Bh Ghana

10000 1010101010

2 Ghaha Bhhahah Somalia

23020 1010101034

3 Oveveveve Ofuefuefuefue Afrika

23420 2010101034

SQL> select \* from dept;

DEPTID DEPTNAME

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

100 AI

101 ML

102 Mech

103 Anime

SQL> create view stud as select student.sid, student.fname, student.lname, dept.deptid from student, dept;

View created.

SQL> select \* from stud;

SID FNAME LNAME DEPTID

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

1 A Bh 100

1 A Bh 101

1 A Bh 102

1 A Bh 103

2 Ghaha Bhhahah 100

2 Ghaha Bhhahah 101

2 Ghaha Bhhahah 102

2 Ghaha Bhhahah 103

3 Oveveveve Ofuefuefuefue 100

3 Oveveveve Ofuefuefuefue 101

3 Oveveveve Ofuefuefuefue 102

SID FNAME LNAME DEPTID

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

3 Oveveveve Ofuefuefuefue 103

12 rows selected.

**Result -** SET and View Operations have been studied and successfully implemented.

**Exercise 9**

### Aim : To perform PL/SQL Programs

In addition to SQL commands, PL/SQL can also process data using flow of statements. The flow of control statements are classified into the following categories.

* Conditional control -Branching
* Iterative control – looping
* Sequential control

BRANCHING in PL/SQL:

Sequences of statements can be executed on satisfying certain condition . If statements are being used and different forms of if are:

* Simple IF
* If-Else
* Nested IF

SIMPLE IF:

Syntax

IF condition THEN statement1; statement2; END IF;

##### IF-THEN-ELSE STATEMENT:

Syntax:

IF condition THEN statement1;

ELSE statement2;

END IF;

##### ELSIF STATEMENTS:

Syntax:

IF condition1 THEN statement1; ELSIF condition2 THEN statement2; ELSIF condition3 THEN statement3; ELSE statement;

END IF;

##### NESTED IF :

Syntax:

IF condition THEN statement1; ELSE

IF condition THEN statement2; ELSE statement3;

END IF; END IF;

ELSE statement3; END IF;

SELECTION IN PL/SQL(Sequential Controls) SIMPLE CASE

Syntax:

CASE SELECTOR

WHEN Expr1 THEN statement1; WHEN Expr2 THEN statement2;

:

ELSE Statement n;

END CASE; SEARCHED CASE:

CASE

WHEN searchcondition1 THEN statement1; WHEN

searchcondition2 THEN statement2;

:

ELSE statement n; END CASE;

##### ITERATIONS IN PL/SQL

Sequence of statements can be executed any number of times using loop construct. It is broadly classified into:

* Simple Loop
* For Loop
* While Loop SIMPLE LOOP

Syntax:

LOOP statement1; EXIT [ WHEN

Condition]; END LOOP;

WHILE LOOP

Syntax

WHILE condition LOOP statement1; statement2; END LOOP;

FOR LOOP

Syntax:

FOR counter IN [REVERSE] LowerBound..UpperBound LOOP

statement1; statement2; END LOOP;

SQL> declare

2 rev number := &rev;

3 comm number;

4 begin

5 if rev > 200000

6 then

7 comm := 0.1;

8 else

9 comm := 0.05;

10 end if;

11 dbms\_output.put\_line('Value of commission: ' || comm);

12 end;

13 /

Enter value for rev: 250000

old 2: rev number := &rev;

new 2: rev number := 250000;

Value of commission: .1

PL/SQL procedure successfully completed.

SQL> declare

2 monthly\_value number := &monthly\_value;

3 income\_level varchar(20);

4 begin

5 if monthly\_value <= 4000

6 then

7 income\_level := 'Low Income';

8 elsif monthly\_value <= 5000

9 then

10 income\_level := 'Avg Income';

11 else

12 income\_level := 'High Income';

13 end if;

14 dbms\_output.put\_line('Income level: ' || income\_level);

15 end;

16 /

Enter value for monthly\_value: 5600

old 2: monthly\_value number := &monthly\_value;

new 2: monthly\_value number := 5600;

Income level: High Income

PL/SQL procedure successfully completed.

SQL> select table\_name from user\_tables;

TABLE\_NAME

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

EMPLOYEE

STUDENT

DEPT

SQL> create table customer

2 (

3 customer\_id int primary key,

4 name varchar(20)

5 );

Table created.

SQL> insert into customer values (1, 'Aditya');

1 row created.

SQL> insert into customer values (2, 'Aman');

1 row created.

SQL> insert into customer values (3, 'Linu');

1 row created.

SQL> insert into customer values (4, 'Rahul');

1 row created.

SQL> select \* from customer;

CUSTOMER\_ID NAME

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

1 Aditya

2 Aman

3 Linu

4 Rahul

SQL> declare

2 names customer.name % type;

3 id customer.customer\_id % type;

4 begin

5 id := &id;

6 select name into names from customer where customer\_id = id;

7 dbms\_output.put\_line('Name of ' || id || ' is = ' || names);

8 end;

9 /

Enter value for id: 1

old 5: id := &id;

new 5: id := 1;

Name of 1 is = Aditya

PL/SQL procedure successfully completed.

SQL> spool off;

## Result :

The Basic of PL/SQL programs has been successfully implemented

# **Exercise-10**

Aim : To perform Procedures program in PL/SQL

Subprogram is a program unit that performs a particular task. These subprograms are combined to form larger programs. This is basically called the 'Modular design'. A subprogram can be invoked by another subprogram or program which is called a calling program.

A subprogram can be created −

* + - At the schema level
    - Inside a package
    - Inside a PL/SQL block

At the schema level, subprogram is a standalone subprogram. It is created with the CREATE PROCEDURE or the CREATE FUNCTION statement. It is stored in the database and can be deleted with the DROP PROCEDURE or DROP FUNCTION statement.

A subprogram created inside a package is a packaged subprogram. It is stored in the database and can be deleted only when the package is deleted with the DROP PACKAGE statement. We will discuss packages in the chapter 'PL/SQL - Packages'.

PL/SQL subprograms are named PL/SQL blocks that can be invoked with a set of parameters. PL/SQL provides two kinds of subprograms –

* Functions − These subprograms return a single value; mainly used to compute and return a value.
* Procedures − These subprograms do not return a value directly; mainly used to perform an action.

Parts of a PL/SQL Subprogram

Each PL/SQL subprogram has a name, and may also have a parameter list. Like anonymous PL/SQL blocks, the named blocks will also have the following three parts –



Creating a Procedure

A procedure is created with the CREATE OR REPLACE PROCEDURE statement. The simplified syntax for the CREATE OR REPLACE PROCEDURE statement is as follows –

CREATE [OR REPLACE] PROCEDURE procedure\_name [(parameter\_name [IN | OUT | IN OUT]

type [, ...])] {IS | AS}

BEGIN

< procedure\_body > END procedure\_name; Where,

* + procedure-name specifies the name of the procedure.
  + [OR REPLACE] option allows the modification of an existing procedure. • The optional parameter list contains name, mode and types of the parameters. IN represents the value that will be passed from outside and OUT represents the parameter that will be used to return a value outside of the procedure.
  + procedure-body contains the executable part.
  + The AS keyword is used instead of the IS keyword for creating a standalone procedure.

Deleting a Standalone Procedure

A standalone procedure is deleted with the DROP PROCEDURE statement. Syntax for deleting a procedure is −

DROP PROCEDURE procedure-name;

You can drop the greetings procedure by using the following statement

− DROP PROCEDURE greetings;

Parameter Modes in PL/SQL Subprograms

The following table lists out the parameter modes in PL/SQL subprograms − S.No Parameter Mode & Description

|  |  |
| --- | --- |
| 1 | IN  An IN parameter lets you pass a value to the subprogram. It is a read-only parameter. Inside the subprogram, an IN parameter acts like a constant. It cannot be assigned a value. You can pass a constant, literal, initialized variable, or expression as an IN parameter. You can also initialize it to a default value;  however, in that case, it is omitted from the subprogram call. It is the default mode of parameter passing. Parameters are passed by reference. |
| 2 | OUT |
|  |  |
|  | An OUT parameter returns a value to the calling program. Inside the subprogram, an OUT parameter acts like a variable. You can change its value and reference the value after assigning it. The actual parameter must be variable and it is passed by  value. |
| 3 | IN OUT  An IN OUT parameter passes an initial value to a subprogram and returns a updated value to the caller. It can be assigned a value and the value can be read.  The actual parameter corresponding to an IN OUT formal parameter must be a variable, not a constant or an expression. Formal parameter must be assigned avalue. Actual parameter is passed by value. |

Methods for Passing Parameters

Actual parameters can be passed in three ways −

* + Positional notation
  + Named notation
  + Mixed notation Positional Notation

In positional notation, you can call the procedure as −

findMin(a, b, c, d);

In positional notation, the first actual parameter is substituted for the first formal parameter; the second actual parameter is substituted for the second formal parameter, and so on. So,

a is substituted for x, b is substituted for y, c is substituted for z and d is substituted for m. Named Notation

In named notation, the actual parameter is associated with the formal parameter using the arrow symbol ( => ). The procedure call will be like the following −

findMin(x => a, y => b, z => c, m => d);

Mixed Notation

In mixed notation, you can mix both notations in procedure call; however, the positional notation should precede the named notation. The following call is legal − findMin(a, b, c, m => d);

However, this is not legal: findMin(x

=> a, b, c, d);

###### Write a PL/SQL block to get the salary of the employee who has empno=7369 and update his salary as specified below

* + - **if his/her salary < 2500, then increase salary by 25% • otherwise if salary lies between 2500 and 5000, then increase salary by 20% • otherwise increase salary by adding commission amount to the salary.**

SQL > Create table emp (Depid

number(3),

empno number(4), salary number(5), depname varchar(6));

SQL > insert into emp values(71,7369,2600,'Civil'); 1 Row(s) inserted

SQL > insert into emp values(73,7379,2800,'HRD'); 1 Row(s) inserted

SQL > insert into emp values(71,7379,2200,'HRD'); 1 Row(s) inserted

SQL > select \* from emp;

DEPID EMPNO SALARY DEPNAME 71 7369

2600 Civil

73 7379

2800 HRD 71

7379 2200 3

HRD rows selected.

SQL > create or replace procedure updatesalary (salary in number, Depid in number) as

begin

if salary < 2500 then

* + update emp set salary = salary+ 0.25\*salary where Depid = 7369;
  + elsif salary >= 2500 and salary < 5000 then
  + update emp set salary = salary+ 0.2\*salary where Depid = 7369;
  + else8 update emp set salary = salary + 0.1\*salary where Depid = 7369;
  + end if;
  + end;
  + /

Procedure created.

SQL > execute updatesalary(2600, 7369)

* 1. Write a PL/SQL Block to modify the department name of the department 71 if it is not ‘HRD’.

SQL > create or replace procedure updatedepartment (depname in varchar, Depid in number) as

begin

update emp set depname = 'HRD' where Depid = 71 and depname not in ('HRD');

end;

/

Procedure Created

SQL > execute updatesalary('Civil', 71)

## Result :

Programs related to Procedure in PL/SQL has been successfully implemented.

# **Exercise-11**

Aim: To perform functions in PL/SQL.

A function is the same as a procedure except that it returns a value. Therefore, all the discussions of the previous chapter are true for functions too.

Creating a Function

A standalone function is created using the CREATE FUNCTION statement. The simplified syntax for the CREATE OR REPLACE PROCEDURE statement is as follows −

CREATE [OR REPLACE] FUNCTION function\_name [(parameter\_name [IN | OUT | IN OUT] type [, ...])] RETURN return\_datatype

{IS | AS} BEGIN

< function\_body > END [function\_name];

Where,

* + - function-name specifies the name of the function.
    - [OR REPLACE] option allows the modification of an existing function. • The optional parameter list contains name, mode and types of the parameters. IN represents the value that will be passed from outside and OUT represents the parameter that will be used to return a value outside of the procedure.
    - The function must contain a return statement.
    - The RETURN clause specifies the data type you are going to return from the function. • function-body contains the executable part.
    - The AS keyword is used instead of the IS keyword for creating a standalone function. Calling a Function

While creating a function, you give a definition of what the function has to do. To use a function, you will have to call that function to perform the defined task. When a program calls a function, the program control is transferred to the called function.

SQL> set serveroutput on

Q.1 Write a PL/SQL Function to find factorial of a given number.

SQL> CREATE OR REPLACE FUNCTION FACT(N NUMBER) 2 RETURN NUMBER IS

* + I NUMBER(10);
  + F NUMBER:=1;
  + BEGIN
  + FOR I IN 1..N LOOP
  + F:=F\*I;
  + END LOOP;
  + RETURN F;
  + END;
  + /

Function created.

SQL> SELECT FAC(5) FROM DUAL; FACT(5)

120

Q.2 Write a PL/SQL Function that computes and returns the maximum of two values. SQL> DECLARE

* + a number;
  + b number;
  + c number;
  + FUNCTION findMax(x IN number, y IN number)
  + RETURN number
  + IS
  + z number;
  + BEGIN
  + IF x > y THEN
  + z:= x;
  + ELSE
  + Z:= y;
  + END IF;
  + RETURN z;
  + END;
  + BEGIN
  + a:= 23;
  + b:= 45;
  + c := findMax(a, b);
  + dbms\_output.put\_line(' Maximum of (23,45): ' || c);
  + END;

/

Maximum of (23,45): 45

PL/SQL procedure successfully completed. SQL> SPOOL OFF

Result:

Programs related to Function in PL/SQL have been successfully implemented.

## **Exercise-12**

### AIM:

To implement the concept of CURSORS in PL/SQL

A cursor is a pointer to this context area. PL/SQL controls the context area through a cursor. A cursor holds the rows (one or more) returned by a SQL statement. The set of rows the cursor holds is referred to as the active set.

You can name a cursor so that it could be referred to in a program to fetch and process the rows returned by the SQL statement, one at a time. There are two types of cursors −

###### Implicit cursors Explicit cursors

**Implicit Cursors:**

Implicit cursors are automatically created by Oracle whenever an SQL statement is executed, when there is no explicit cursor for the statement. Programmers cannot control the implicit cursors and the information in it.

Whenever a DML statement (INSERT, UPDATE and DELETE) is issued, an implicit cursor is associated with this statement. For INSERT operations, the cursor holds the data that needs to be inserted. For UPDATE and DELETE operations, the cursor identifies the rows that would be affected.

In PL/SQL, you can refer to the most recent implicit cursor as the SQL cursor, which always has attributes such as %FOUND, %ISOPEN, %NOTFOUND, and

%ROWCOUNT. The SQL cursor has additional attributes,

%BULK\_ROWCOUNT and %BULK\_EXCEPTIONS, designed for use with the FORALL statement. The following

table provides the description of the most used attributes −

S.No Attribute & Description

|  |  |
| --- | --- |
| 1 | %FOUND  Returns TRUE if an INSERT, UPDATE, or DELETE statement affected one or more rows or a SELECT INTO statement returned one or more rows. Otherwise,  it returns FALSE. |
| 2 | %NOTFOUND  The logical opposite of %FOUND. It returns TRUE if an INSERT, UPDATE, or DELETE statement affected no rows, or a SELECT INTO statement returned no |
|  | rows. Otherwise, it returns FALSE. |
| 3 | %ISOPEN  Always returns FALSE for implicit cursors, because Oracle closes the SQL cursor automatically after executing its associated SQL statement. |
| 4 | %ROWCOUNT  Returns the number of rows affected by an INSERT, UPDATE, or DELETE statement, or returned by a SELECT INTO statement. |

**Explicit Cursors:**

Explicit cursors are programmer-defined cursors for gaining more control over the context area. An explicit cursor should be defined in the declaration section of the PL/SQL Block. It is created on a SELECT Statement which returns more than one row.

The syntax for creating an explicit cursor is −

CURSOR cursor\_name IS select\_statement;

Working with an explicit cursor includes the following steps −

* Declaring the cursor for initializing the memory
* Opening the cursor for allocating the memory
* Fetching the cursor for retrieving the data
* Closing the cursor to release the allocated memory Declaring the Cursor

Declaring the cursor defines the cursor with a name and the associated SELECT

statement. For example −

CURSOR c\_customers IS

SELECT id, name, address FROM customers;

Opening the Cursor

Opening the cursor allocates the memory for the cursor and makes it ready for fetching the rows returned by the SQL statement into it. For example, we will open the above defined cursor as follows −

OPEN c\_customers; Fetching the Cursor

Fetching the cursor involves accessing one row at a time. For example, we will fetch rows from the above-opened cursor as follows − FETCH c\_customers INTO c\_id, c\_name, c\_addr;

Closing the Cursor

Closing the cursor means releasing the allocated memory. For example, we will close the above opened cursor as follows −

CLOSE c\_customers;

SQL> set serveroutput on

* 1. **Write a PL/SQL Block, to update salaries of all the employees who work in deptno 20 by 15%. If none of the employee’s salary are updated display a message 'None of the salaries were updated'. Otherwise display the total number of employee who got salary updated.**

##### SQL> create table emp(

* + dept\_id number(10),

##### salary number(10)); Table Created

SQL> desc emp

##### Column Null? Type

DEPT\_ID - NUMBER(10,0) SALARY

##### - NUMBER(10,0)

SQL > Insert into emp values(10,1000); 1 row(s) inserted.

##### SQL >Insert into emp values(20,1000); 1 row(s) inserted.

SQL >Insert into emp values(20,2000); 1 row(s) inserted.

##### SQL >Insert into emp values(30,1500); 1 row(s) inserted.

SQL > select \* from emp;

##### DEPT\_ID SALARY 1 0 1000

20 1000

##### 20 2000

30 1500

##### 4 rows selected.

Declare num number(5); Begin update emp set salary = salary +

##### salary\*0.15 where dept\_id=20; if SQL%NOTFOUND then dbms\_output.put\_line('none of the salaries were updated'); elsif SQL%FOUND then

num := SQL%ROWCOUNT;

##### dbms\_output.put\_line('salaries for '|| num || ' employees are updated'); end if;

End;

##### /

PL/SQL procedure successfully completed. salaries for 2 employees are updated

SQL > select \* from emp ;

DEPT\_ID SALARY 1 0 1000

20 1150

20 2300

30 1500

4 rows selected.

### Result:

Programs related to concept of cursors in PL/SQL has been successfully implemented.

## **Exercise-13**

Aim:

To perform the implementation of the concept TRIGGERS IN PL/SQL

Triggers are stored programs, which are automatically executed or fired when some events occur. Triggers are, in fact, written to be executed in response to any of the following events –

* A database manipulation (DML) statement (DELETE, INSERT, or UPDATE) A database definition (DDL) statement (CREATE, ALTER, or DROP).
* A database operation (SERVERERROR, LOGON, LOGOFF, STARTUP, or SHUTDOWN).

Triggers can be defined on the table, view, schema, or database with which the event is associated. Benefits of Triggers

Triggers can be written for the following purposes − • Generating some derived column values automatically

* Enforcing referential integrity
* Event logging and storing information on table access
* Auditing
* Synchronous replication of tables
* Imposing security authorizations
* Preventing invalid transactions

Creating Triggers

The syntax for creating a trigger is −

CREATE [OR REPLACE ] TRIGGER trigger\_name

{BEFORE | AFTER | INSTEAD OF }

{INSERT [OR] | UPDATE [OR] | DELETE}

[OF col\_name]

ON table\_name

[REFERENCING OLD AS o NEW AS n] [FOR EACH ROW]

WHEN (condition) DECLARE

Declaration-statements BEGIN

Executable-statements

|  |  |
| --- | --- |
| EXCEPTION |  |
| Exception-handling-statements END; | |

Where,

* CREATE [OR REPLACE] TRIGGER trigger\_name − Creates or replaces an existing trigger with the trigger\_name.

{BEFORE | AFTER | INSTEAD OF} − This specifies when the trigger will be executed. The INSTEAD OF clause is used for creating trigger on a DML operation. • [OF col\_name] − This specifies the column name that will be updated.

* [ON table\_name] − This specifies the name of the table associated with the trigger.
* [REFERENCING OLD AS o NEW AS n] − This allows you to refer new and old values for various DML statements, such as INSERT, UPDATE, and DELETE.
* [FOR EACH ROW] − This specifies a row-level trigger, i.e., the trigger will be executed for each row being affected. Otherwise the trigger will execute just once when the SQL statement is executed, which is called a table level trigger.
* WHEN (condition) − This provides a condition for rows for which the trigger would fire. This clause is valid only for row-level triggers.

The following points need to be considered here −

* OLD and NEW references are not available for table-level triggers, rather you can use them for record-level triggers.
* If you want to query the table in the same trigger, then you should use the AFTER keyword, because triggers can query the table or change it again only after the initial changes are applied and the table is back in a consistent state.
* The above trigger has been written in such a way that it will fire before any DELETE or INSERT or UPDATE operation on the table, but you can write your trigger on a single or multiple operations, for example BEFORE DELETE, which will fire whenever a record will be deleted using the DELETE operation on the table.
* view. • {INSERT [OR] | UPDATE [OR] | DELETE} − This specifies the

Triggering a Trigger

Let us perform some DML operations on the CUSTOMERS table. Here is one INSERT statement, which will create a new record in the table −

|  |  |  |  |
| --- | --- | --- | --- |
|  | | | |
|  | INSERT INTO CUSTOMERS (ID,NAME,AGE,ADDRESS,SALARY) | |  |
| VALUES (7, 'Kriti', 22, 'HP', 7500.00 ); |  |

When a record is created in the will display the following result

− CUSTOMERS table, the above trigger, create display\_salary\_changes will be fired.

Because this is a new record, old salary is not available and the above result comes as null. Let us now perform one more DML operation on the CUSTOMERS table. The

UPDATE statement will update an existing record in the table −

|  |  |
| --- | --- |
| UPDATE customers  SET salary = salary + 500 WHERE id = 2; |  |
| When a record is updated in the CUSTOMERS  table, the above trigger, display\_salary\_changes will be fired and it | will display the following result − create |
| Old salary: 1500 New  salary: 2000 |  |

1. **Create a Trigger to check the entered age is valid or not.**

create or replace trigger DateofBirth

after insert on Account

begin

if(NEW.age<18)

then dbms\_output.put\_line(‘Age should be more than 18’);

END;

/

1. **Create a row-level trigger for the customers table that would fire for INSERT or UPDATE or DELETE operations performed on that table.**

CREATE OR REPLACE TRIGGER display\_salary\_changes

BEFORE DELETE OR INSERT OR UPDATE ON customers

FOR EACH ROW

WHEN (NEW.ID > 0)

DECLARE

sal\_diff number;

BEGIN

sal\_diff := :NEW.salary - :OLD.salary;

dbms\_output.put\_line('Old salary: ' || :OLD.salary);

dbms\_output.put\_line('New salary: ' || :NEW.salary);

dbms\_output.put\_line('Salary difference: ' || sal\_diff);

END;

/

Result:

Programs related to Trigger in PL/SQL have been successfully implemented.

# **Exercise-14**

### Aim:

To implement the concept of exception handling in PL/SQL.

In PL/SQL a warning or error condition is called an exception. Exceptions can be internally defined (by the runtime system) or user-defined. Examples of internally defined exceptions include division by zero and out of memory.

Predefined Exceptions

CURSOR\_ALREADY\_OPEN is raised if you try to OPEN an already open cursor. DUP\_VAL\_ON\_INDEX is raised if you try to store duplicate values in a database column that is constrained by a unique index.

INVALID\_CURSOR is raised if you try an illegal cursor operation. For example, if you try to CLOSE an unopened cursor.

INVALID\_NUMBER is raised in a SQL statement if the conversion of a character string to a number fails.

LOGIN\_DENIED is raised if you try logging on to ORACLE with an invalid username/password.

NO\_DATA\_FOUND is raised if a SELECT INTO statement returns no rows or if you reference an uninitialized row in a PL/SQL table.

NOT\_LOGGED\_ON is raised if your PL/SQL program issues a database call without being logged on to ORACLE. PROGRAM\_ERROR is raised if PL/SQL has an internal problem.

STORAGE\_ERROR is raised if PL/SQL runs out of memory or if memory is corrupted. TIMEOUT\_ON\_RESOURCE is raised if a timeout occurs while ORACLE is waiting for a resource.

TOO\_MANY\_ROWS is raised if a SELECT INTO statement returns more than one row. VALUE\_ERROR is raised if an arithmetic, conversion, truncation, or constraint error occurs.

ZERO\_DIVIDE is raised if you try to divide a number by zero. Handling Raised Exception

Syntax : EXCEPTION

WHEN ... THEN

* + handle the error differently WHEN ... OR ... THEN
  + handle the error differently WHEN OTHERS THEN
  + handle the error differently END;

###### User Defined Exception:

Unlike predefined exceptions, user-defined exceptions must be declared and must be raised explicitly by RAISE statements. Exceptions can be declared only in the declarative part of a PL/SQL block, subprogram, or package. You declare an exception by introducing its name, followed by the keyword EXCEPTION.

Exception Declaration Ex. DECLARE

past\_due EXCEPTION; acct\_num NUMBER(5);

BEGIN

Exceptions and variable declarations are similar. But remember, an exception is an error condition, not an object. Unlike variables, exceptions cannot appear in assignment statements or SQL statements.

Syntax.

Exception-name Exception;

Using Raise statement

User-defined exceptions must be raised explicitly by RAISE statements. Syntax

RAISE exception-name;

Raise\_Application\_Error

This is a procedure to issue user-defined error messages from a stored

subprogram or database trigger.

Syntax : raise\_application\_error(error\_number, error\_message); where error\_number is a negative integer in the range -20000..- 20999 and error\_message is a character string up to 512 bytes in length.

Ex. IF salary is NULL THEN

raise\_application\_error(-20101, 'Salary is missing');

SQL> SET SERVEROUTPUT ON

**Q.1 Write a PL/SQL program that accepts a customer id as an input and returns the customer name using exception handling.**

##### SQL> create table customer(c\_id number(10), c\_name varchar2(10)); Table created.

SQL> insert into customer values(1, ‘Aditya’); 1 row(s) inserted.

##### SQL> insert into customer values(2, ‘Bhim’); 1 row(s) inserted. SQL> insert into customer values(3, ‘Varun’); 1 row(s) inserted.

SQL> insert into customer values(4, ‘Alan’); 1 row(s) inserted.

##### SQL> DECLARE

* + l\_name customer.c\_name%TYPE;

##### l\_customer\_id customer.c\_id%TYPE := 4;

* + BEGIN

##### SELECT c\_name INTO l\_name

* + FROM customer

##### WHERE c\_id = l\_customer\_id;

* + dbms\_output.put\_line('Customer name is ' || l\_name);

##### EXCEPTION

* + WHEN NO\_DATA\_FOUND THEN

##### dbms\_output.put\_line('Customer ' || l\_customer\_id || ' doesnot exist');

##### 12 END;

13 /

##### Customer name is Alan

##### SQL> DECLARE

* + l\_name customer.c\_name%TYPE;

##### l\_customer\_id customer.c\_id%TYPE := 10;

* + BEGIN

##### SELECT c\_name INTO l\_name

* + FROM customer
  + WHERE c\_id = l\_customer\_id;8 dbms\_output.put\_line('Customer name is ' || l\_name); 9 EXCEPTION
  + WHEN NO\_DATA\_FOUND THEN
  + dbms\_output.put\_line('Customer ' ||l\_customer\_id || ' does not exist'); 12 END;

13 /

Customer 10 does not exist SQL>SPOOL OFF

### Result:

The implementation of the concept exception handling in PL/SQL has been successfully implemented.