VIGNAN'S

INSTITUTE OF INFORMATION TECHNOLOGY (Autonomous)

VISAKHAPATNAM

DATABASE MANAGEMENT SYSTEMS LAB RECORD

Department of Computer Science and Engineering

II B. Tech II Sem VR-20

NAME:

REG.NO:

YEAR:_____ SEM:

VIGNAN'S

INSTITUTE OF INFORMATION TECHNOLOGY (Autonomous)

VISAKHAPATNAM

CERTIFICATE

This is to certify that this is the bonafied record of the work done in ……..

Laboratory by Mr/Ms. ……..

Bearing Reg no/Roll no. ……of……...course during……

Total number of Experiments held…….

Total number of Experiments held……..

LAB-IN-CHARGE

HEAD OF THE DEPARTMENT

INDEX

S.No	Date	Name Of Experiment/Program	Page No.	Remarks
1				
2				
3				
4				
5				
6				
7				
8				
9				
10				
11				
12				
13				
14				

EXPERIMENT: 1

Aim: Queries for Creating, Dropping, and Altering Tables and insert row into a table (use constraints while creating tables) examples using Select Command.

Procedure:

1. Creation of emp_ & dept table in Sql:

```
SQL>create table dept(
        deptno number(2,0) primary key,
        dname varchar2(14) NOT NULL,
        loc varchar2(13) NOT NULL,
);
```

Table created.

```
SQL>create table emp(
        empno    number(4,0),
        ename    varchar2(10) NOT NULL,
        job      varchar2(9) NOT NULL,
        mgr      number(4,0),
        hiredate date,
        sal      number(7,2) NOT NULL,
        comm     number(7,2),
        eptno  number(2,0),
        constraint pk_emp primary key (empno),
        constraint fk_deptno foreign key (deptno) references dept (deptno)
);
```

Table created.

2. View Structure/schema_of_emp_& dept table in sql:

SQL> select *from emp;

no rows selected

SQL> select *from dept;

no rows selected

SQL> desc emp;

Name	Null?	Type
EMPNO	NOT NULL	NUMBER(4)
ENAME		VARCHAR2(10)
JOB		VARCHAR2(9)
MGR		NUMBER(4)
HIREDATE		DATE
SAL		NUMBER(7,2)
COMM		NUMBER(7,2)
DEPTNO		NUMBER(2)

SQL> desc dept;

Name	Null?	Type
DEPTNO	NOT NULL	NUMBER(2)
DNAME		VARCHAR2(14)
LOC		VARCHAR2(13)

2. Insert the values in emp & dept table in sql:

There are several ways to insert the values in the existing table

Query to insert single record in the existing table:

SQL> insert into dept(deptno,dname,loc) values(20,'admin','hyd');

1 row created.

Query to insert multiple records in the existing table:

SQL>insert into dept values(&deptno,'&dname','&loc');

Enter value for deptno: 10

Enter value for dname: sales

Enter value for loc: vijayawada

old 1: insert into dept values(&deptno,'&dname','&loc')

new 1: insert into dept values(10,'sales','vijayawada')

1 row created.

SQL>/

Enter value for deptno: 20

Enter value for dname: admin

Enter value for loc: hyd

old 1: insert into dept values(&deptno,'&dname','&loc')

1 row created.

SQL> /

Enter value for deptno: 30

Enter value for dname: marketing

Enter value for loc: vzg

old 1: insert into dept values(&deptno,'&dname','&loc')

new 1: insert into dept values(30,'marketing','vzg')

1 row created.

4. Select Command: this command is used to print the record from the existing table.

View all records in dept table:

SQL> select *from dept;

```
  DEPTNO    DNAME            LOC
---------- -------------- -------------
        10 sales          vijayawada
        20 admin          hyd
        30 marketing      vzg
```

View records basing on given criteria on specific column.

1. View single column from existing table.

SQL>select dname from dept;

DNAME

```
----------
```

Sales

Admin

Marketing

2. View specific record(s) from existing table based on given condition.

SQL> select *from dept where dname='sales';

```
 DEPTNO    DNAME            LOC
---------- -------------- -------------
        10 sales          vijayawada
```

Types of SQL Commands:

DDL: DDL Commands (Data Definition Language)

1. CREATE 2. DESC 3. ALTER 4. DROP 5.TRUNCATE 6. RENAME

DML Commands (Data Manipulation Language)

1. SELECT 2. INSERT 3. UPDATE 4. DELETE

TCL(Transaction Control Language)

1. COMMIT 2. ROLLBACK 3. SAVEPOINT

DCL Commands (Data Control Language)

1. GRANT 2. REVOKE

1. CREATE:

CREATE TABLE: This is used to create a new relation and the corresponding

Syntax: CREATE TABLE relation_name (field_1 data_type(Size),field_2 data_type(Size), .. .);

Example:

SQL>CREATE TABLE Student (id number, name varchar2(10));

RESULT: Table created.

2. DESC: It is used to describe a schema as well as to retrieve rows from table in descending order.

SYNTAX: DESC

EX: SQL> DESC EMP1;

NAME	NULL?	TYPE
EMPNO	NOT NULL	NUMBER(10)
ENAME		VARCHAR2(15)
JOB		CHAR(10)
DEPTNAME		VARCHAR2(10)
DEPTNO		NUMBER(9)
HIREDATE		DATE
SALARY		NUMBER(8)
EXP		NUMBER(5**)**

3. ALTER: This is used for add, remove or modify the structure of the existing table

(a) ALTER TABLE ...ADD...: This is used to add some extra fields into existing relation.

Syntax: ALTER TABLE relation_name ADD(new field_1 data_type(size), new field_2data_type(size),..);

Example : SQL>ALTER TABLE emp1 ADD(Address CHAR(10));

TABLE ALTERED.

(b) ALTER TABLE...MODIFY...: This is used to change the width as well as data type of fields ofexisting relations.

Syntax: ALTER TABLE relation_name MODIFY (field_1 newdata_type(Size), field_2newdata_type(Size),. , field_newdata_type(Size));

Example:

SQL>ALTER TABLE emp1 MODIFY(ename VARCHAR2(20), salary NUMBER(5));

TABLE ALTERED.

SQL> DESC EMP1;

NAME	NULL?	TYPE
EMPNO	NOT NULL	NUMBER(10)
ENAME		**VARCHAR2(20)**
JOB		CHAR(10)
DEPTNAME		VARCHAR2(10)
DEPTNO		NUMBER(9)
HIREDATE		DATE
SALARY		**NUMBER(5)**
EXP		NUMBER(5)
ADDRESS		CHAR(10)

4. DROP TABLE: This is used to delete the structure of a relation. It permanently deletes the table.

Syntax: DROP TABLE tablename;

Example:

SQL>DROP TABLE EMP1;

Table dropped;

DROP: this command is used to remove the date from the existing tableDROP COLUMN IN TABLE

Syntax:

To DROP A COLUMN in an existing table, the Oracle ALTER TABLE **syntax is**:ALTER TABLE table_name DROP COLUMN column_name;

Example customers DROP COLUMN customer_name;

SQL> ALTER TABLE customers DROP COLUMN customer_name;

5. RENAME: It is used to modify the name of the existing database object.

Syntax: RENAME old_table_name TO new_table_name;

Example:

SQL>RENAME EMP1 TO EMP2;

Table renamed.

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

Syntax: TRUNCATE TABLE <Table name>

Example :

TRUNCATE TABLE EMP1;

EXPERIMENT: 2

QUERIES (ALONG WITH SUB QUERIES) USING ANY, ALL, IN, EXISTS, NOTEXISTS, UNION, INTERSECT

SOLUTION:

To Create employee table:

```
Sql> create table employee(
        Fname varchar2(20),
        Lname varchar2(20),
        Ssn number(4) primary key,
        B_date date,
        Address varchar2(30),
        Gender char(1),
        Salary number(7,2),
        Super_ssn references employee(ssn),
        Dno number(4)
         );
Table created.

SQL> INSERT INTO EMPLOYEE
       VALUES('SMITH',NULL,1111,'03-NOV-2016','BJD','M',2000,NULL,10)
1 row created.
SQL> INSERT INTO EMPLOYEE
      VALUES('ALLEN',NULL,2222,'03-NOV-2016','SBC','M',3000,1111,20)
1 row created.
SQL> INSERT INTO EMPLOYEE
      VALUES('MARTIN',NULL,3333,'03-NOV-2016','HYD','M',4000,1111,30)
1 row created.
```

Like this we can insert the values into the table. To view data in the table following query is used.

SQL> SELECT *FROM EMPLOYEE;

FNAME	LNAME	SSN	BDATE	ADDRESS	G	SALARY	SUPER_SSN	DNO
SMITH		1111	01-JAN-06	BZA	M	2000		10
ALLEN		2222	12-DEC-04	SBC	M	3000	1111	20
MARTIN		3333	15-DEC-07	HYD	M	4000	1111	20
JONES		4444	28-SEP-05	TNU	M	1500	2222	10
BLAKE		5555	04-SEP-04	VZA	M	2500	2222	10
TURNER		6666	21-OCT-99	GNT	M	6000	3333	20

6 rows selected.

Inserting values in the dependent table as follows

SQL> INSERT INTO DEPENDENT VALUES (1111,'SMITH',’G’)

1 row is created.

SQL> INSERT INTO DEPENDENT VALUES (2222,'POOJA',’F’)

1 row is created.

SQL> INSERT INTO DEPENDENT VALUES (3333,'MARTIN',’M’)

1 row is created.

SQL> INSERT INTO DEPENDENT VALUES (3333,'RAJA',’M’)

1 row is created.

To Create dependent table:

```
SQL> CREATE TABLE DEPENDENT (
        ESSN NUMBER (4) REFERENCES EMPLOYEE (SSN),
        DEPENDENT_NAME VARCHAR2 (20),
        GENDER CHAR (1),
        B_DATE DATE,
        RELATIONSHIP VARCHAR2 (20),
        PRIMARY KEY (ESSN, DEPENDENT_NAME)
        );
```

Table created.

To view data in the dependent table as follows.

SQL>SELECT * FROM DEPENDENT;

ESSN	DEPENDENT_NAME	G	B_DATE	RELATIONSHIP
1111	SMITH	M		
2222	POOJA	F		
3333	MARTIN	M		
3333	RAJA	M		

1. ALL:

Retrieve the names of employees whose salary is greater than the salary of all the employees in department 10

SQL> SELECT FNAME, LNAME FROM EMPLOYEE WHERE SALARY> ALL (SELECT SALARY FROM EMPLOYEE WHERE DNO=10);

FNAME	LNAME
ALLEN	
MARTIN	
TURNER	

2. ANY

Retrieve the names of employees whose salary is greater than the salary of any one of the employees in department 10

SQL> SELECT FNAME, LNAME FROM EMPLOYEE
WHERE SALARY> ANY(SELECT SALARY FROM EMPLOYEE WHERE DNO=10);

FNAME	LNAME
TURNER	
MARTIN	
ALLEN	
BLAKE	
SMITH	

3. IN

Retrieve the name of each employee who has a dependent with the firstname and same gender as the employee

SQL> **SELECT e.FNAME, e.LNAME FROM EMPLOYEE e WHERE e.SSN IN (SELECT ESSN FROM DEPENDENT WHERE e.GENDER=GENDER AND e.FNAME = DEPENDENT_NAME);**

FNAME LNAME

SMITH

MARTIN

4. EXISTS

Retrieve the name of each employee who has a dependent with the firstname and same gender as the employee

SQL> **SELECT e.FNAME, e.LNAME FROM EMPLOYEE e WHERE EXISTS (SELECT *FROM DEPENDENT WHERE e.SSN=ESSN AND e.GENDER=GENDER AND e.FNAME = DEPENDENT_NAME);**

FNAME LNAME

SMITH

MARTIN

5.NOT EXISTS

Retrieve the names of employees who have no dependents

SQL> **SELECT FNAME, LNAME FROM EMPLOYEE WHERE NOT EXISTS (SELECT * FROM DEPENDENT WHERE SSN=ESSN);**

FNAME LNAME

ALLEN

SQL Constraints

SQL constraints are used to specify rules for the data in a table.

Constraints are used to limit the type of data that can be insert into a table. This ensures the accuracy and reliability of the data in the table. If there is any violation between the constraint and the data action, the action is aborted.

Constraints can be column level or table level. Column level constraints apply to a column, and table level constraints apply to the whole table.

The following constraints are commonly used in SQL:

- **NOT NULL** - Ensures that a column cannot have a NULL value

  Example:

  SQL> create table person1 (id int, name varchar2 (10) not null, age int);

  Table created.

- **UNIQUE** - Ensures that all values in a column are different

  Example:

  SQL> **create table person(id int unique, name varchar2(10),age int);**

  Table created.

- **PRIMARY KEY** - A combination of a NOT NULL and UNIQUE. Uniquely identifies each row in a table

  Example:

  SQL> create table emp1(id number(10) primary key, name varchar2(10),sal int);

  Table created.

- **FOREIGN KEY** - Uniquely identifies a row/record in another table
  - A FOREIGN KEY is a key used to link two tables together.
  - A FOREIGN KEY is a field (or collection of fields) in one table that refers to the PRIMARY KEY in another table.
  - The table containing the foreign key is called the child table, and the table containing the candidate key is called the referenced or parent table.

  Example:

  SQL> create table emp2 (eid int, city varchar2(10),foreign key(eid) references emp1(id));

  Table created.

- **CHECK** - Ensures that all values in a column satisfies a specific condition

  Example:

  SQL> **CREATE TABLE person1(ID int ,Age int, City varchar(10), CONSTRAINT chk CHECK(Age>=18 AND City='vja');**

  Table created.

- **DEFAULT** - Sets a default value for a column when no value is specified.
  - The DEFAULT constraint is used to provide a default value for a column.
  - The default value will be added to all new records IF no other value is specified.

SQL DEFAULT on CREATE TABLE

SQL> create table emp(id number(10),name varchar2(10),city varchar2(10) default 'vja');

Table created.

EXPERIMENT: 3

QUERIES USING AGGREGATE FUNCTIONS (COUNT, SUM, AVG, MAX AND MIN) GROUP BY, HAVING and Creation and dropping of Views.

SOLUTION:

1. COUNT: Calculate the number of employees in dept 20.

```
SQL> SELECT COUNT (*) NO_EMP FROM EMP WHERE DEPTNO=20;

      NO_EMP
      -------
      5
```

2. SUM: Calculate the total salaries for each dept

```
SQL> SELECT DEPTNO, SUM (SAL) FROM EMP GROUP BY DEPTNO

 DEPTNO  SUM(SAL)
-------  -------
     30     9400
     20    10875
     10     8750
```

3. AVG: Calculate the average salaries for each dept

```
SQL> SELECT DEPT_NO, AVG (SAL) FROM EMP GROUP BY DEPT_NO;

  DEPT_NO AVG(SAL)
 ---------- ----------
    30        1566.66667
    20       2175
    10        2916.66667
```

4. MAX: Calculate the maximum salary for each dept

```
SQL> SELECT DEPTNO, MAX (SAL) FROM EMP GROUP BY DEPTNO;

  DEPTNO  MAX(SAL)
  -------  ----------
     30     2850
     20     3000
     10     5000
```

5. MIN

Calculate the minimum salary for each dept

SQL> SELECT DEPTNO, MIN(SAL) FROM EMP GROUP BY DEPTNO

DEPTNO	MIN(SAL)
30	950
20	800
10	1300

6. GROUP BY:

The GROUP BY clause is a SQL command that is used to **group rows that have the same values**.

The GROUP BY clause is used in the SELECT statement .Optionally it is used in conjunction with aggregate functions to produce summary reports from the database.

GROUP BY Syntax

SELECT statements… GROUPBY column_name1[column_name2,…];

Grouping using a Single Column:

Create a table called data with gender column and values as male and female.

SQL> select * from data;

GENDER
---
male
female
female
female
female
female
male
male
male
female
male
male
female
male
male
female

16 rows selected.

SQL> select gender from data GROUP BY gender;

```
GENDER
-------
male
female
```

SQL> select count (gender), gender from data GROUP BY gender;

```
COUNT (GENDER) GENDER
---------------  ----------
             8  male
             8  female
```

Grouping using Multiple Columns

Syntax

SELECT Column1, Column2, AGGREGATE_FUNCTION (Column3) FROM TABLE1 **GROUPBY** Column1, Column2

Examples :

SQL> select * from emp;

```
        ID NAME       DEPT       SAL
---------- ---------- ---------- ----------
         1          a        cse       1000
         2          b        ece       2000
         3          c        eee       3000
         4          d        cse       4000
         1          z        cse       5000
         5          a        ece       6000
         6          e        ece       7000
         2          b        eee       9000
8 rows selected.
```

SQL> select id, name from emp GROUP BY id, name;

```
        ID NAME
---------- ----------
         3          c
         4          d
         1          a
         2          b
         5          a
         1          z
         6          e
7 rows selected.
```

7. HAVING

- The HAVING clause was added to SQL because the WHERE keyword could not be used with aggregate functions.
- The WHERE clause places conditions on the selected columns, whereas the HAVING clause places conditions on groups created by the GROUP BY clause.
- The HAVING clause must follow the GROUP BY clause in a query and must also precede the ORDER BY clause if used

HAVING Syntax

SELECT *column_name(s)* FROM *table_name* WHERE *condition* GROUP BY *column_name(s)* HAVING *condition*

SQL> select * from emp;

ID	NAME	DEPT	SAL
1	a	cse	1000
2	b	ece	2000
3	c	eee	3000
4	d	cse	4000
5	e	ece	5000

SQL> select count (id), dept from emp GROUP BY dept having count (id)>1;

COUNT(ID)	DEPT
2	cse
2	ece

SQL> select * from emp;

ID	NAME	DEPT	SAL
1	a	cse	1000
2	b	ece	2000
3	c	eee	3000
4	d	cse	4000
5	e	ece	5000

SQL> select max (sal), dept from emp GROUP BY dept;

MAX(SAL)	DEPT
4000	cse
3000	eee
5000	ece

SQL> select max(sal),dept from emp GROUP BY dept having max(sal)>3000;

MAX(SAL)	DEPT
4000	cse
5000	ece

8. View :

- Views in SQL are considered as a virtual table. A view also contains rows and columns.
- To create the view, we can select the fields from one or more tables present in the database.
- A view can either have specific rows based on certain condition or all the rows of a table**.**

SQL> select *from emp;

ENO	ENAME	SALARY	LOC
101	ali	15000	vja
102	haji	20000	hyd
103	mohammad	42000	vja
104	ravi	23000	gnt
105	irfath	50000	hyd

SQL> create VIEW hyd as select *from emp where loc='hyd';

View created.

SQL> select *from hyd;

ENO	ENAME	SALARY	LOC
102	haji	20000	hyd
105	irfath	50000	hyd

SQL> drop VIEW hyd;

View dropped.

SQL> select *from hyd;

select *from hyd

*

ERROR at line 1:

ORA-00942: table or view does not exist

EXPERIMENT: 4

QUERIES USING CONVERSION FUNCTIONS (TO_CHAR, TO_NUMBER AND TO_DATE), STRING FUNCTIONS (CONCATENATION, LPAD, RPAD, LTRIM, RTRIM, LOWER, UPPER, INITCAP, LENGTH, SUBSTR AND INSTR), DATE FUNCTIONS (SYSDATE, NEXT_DAY, ADD_MONTHS, LAST_DAY, MONTHS_BETWEEN, LEAST, GREATEST, TRUNC, ROUND, TO_CHAR)

SQL> select *from emp;

ENO	ENAME	SALARY	LOC
101	ali	15000	vja
102	haji	20000	hyd
103	mohammad	42000	vja
104	ravi	23000	gnt
105	irfath	50000	hyd

a) Conversion Functions:

1. to_char: to_char is used to convert the attribute values to char.

SQL> select to_char(salary,'$99999.99') from emp;

TO_CHAR(SALARY)
$15000.00
$20000.00
$42000.00
$23000.00
$50000.00

SQL> SELECT TO_CHAR (123.4567, '99999.9') FROM DUAL;

TO_CHAR (
123.5

SQL> SELECT TO_CHAR(123.4567, '99999.99') FROM DUAL;

TO_CHAR(1
123.46

SQL> SELECT TO_CHAR(1234.56789,'9,999.00') FROM DUAL;

```
TO_CHAR(1
------
 1,234.57

SQL> SELECT TO_CHAR(SYSDATE, 'YYYY/MM/DD') FROM DUAL;

TO_CHAR(SY
-------
2021/07/09

SQL>  SELECT TO_CHAR (SYSDATE, 'DD/MM/YYYY') FROM DUAL;

TO_CHAR(SY
-------
09/07/2021

SQL> SELECT TO_CHAR (23, '000099') FROM DUAL;

TO_CHAR
-----
 000023

SQL> SELECT TO_CHAR (23, '0000999') FROM DUAL;

TO_CHAR(
------
 0000023

SQL>  SELECT TO_CHAR (23, '00009') FROM DUAL;

TO_CHA
----
 00023

SQL> SELECT TO_CHAR (23, '00000') FROM DUAL;

TO_CHA
----
 00023

SQL> SELECT TO_CHAR (234.5678, '00.00') FROM DUAL;

TO_CHA
----
######
```

```
SQL> SELECT TO_CHAR (234.5678, '000.000') FROM DUAL;

TO_CHAR(
-------
 234.568

SQL> SELECT TO_CHAR(2345.234566, '1,23.000') FROM DUAL;
SELECT TO_CHAR(2345.234566, '1,23.000') FROM DUAL
                            *
ERROR at line 1:
ORA-01481: invalid number format model

SQL> SELECT TO_CHAR (2345.2345, '9,000.00') FROM DUAL;

TO_CHAR(2
--------
 2,345.23

SQL> SELECT TO_CHAR (2345.2345, '$9,000.00') FROM DUAL;

TO_CHAR(23
---------
 $2,345.23
```

2. to_number: to_number is used to convert the attribute value to number.

```
SQL> SELECT  TO_NUMBER('1210.73', '9999.99') FROM DUAL;

        TO_NUMBER('1210.73','9999.99')
       ------------------------------
                        1210.73
```

3. to_date: to_date is used for convert and display the attribute values as date.

```
SQL> select to_date('01-01-2020', 'MM-DD-YYYY') from dual;

TO_DATE('
--------
01-JAN-20
```

b) String functions:

1. **Concatenation:** CONCAT is used to add two attribute values such as string.

SQL> select concat (eno, loc) from emp;

```
CONCAT(ENO,LOC)
--------------------------------
101vja
102hyd
103vja
104gnt
105hyd
```

2. **lpad:** LPAD() function is used to padding the left side of a string with a specific set of characters.

SQL> select lpad(ename,10,'*') from emp;

```
LPAD(ENAME,10,'*')
--------------------------
*******ali
******haji
**mohammad
******ravi
****irfath
```

3. **rpad**: RPAD() function is used to padding the right side of a string with a specific set of characters.

SQL> select rpad(ename,10,'*') from emp;

```
RPAD(ENAME,10,'*')
--------------------------
ali*******
haji******
mohammad**
ravi******
irfath****
```

4. **ltrim**: LTRIM() function is used to remove all specified characters from the left end side of a string

SQL> select ltrim('****hi********','*') from dual;**

```
LTRIM('***
-------
hi********
```

5. **rtrim:** RTRIM() function is used to remove all specified characters from the left end side of a string

   SQL> select rtrim('****hi********','*') from dual;**

```
RTRIM('*
------
******hi
```

6. **lower:** lower() function is used to convert the attribute value in to lower case.

   SQL> select lower(ename) from emp;

```
LOWER(ENAM
-------
ali
haji
mohammad
ravi
irfath
```

7. **upper**: upper() function is used to convert the attribute values in to upper case.

   SQL> select upper(ename) from emp;

```
UPPER(ENAM
-------
ALI
HAJI
MOHAMMAD
RAVI
IRFATH
```

8. **initcap**: initcap() is used to convert the attribute values first character in capital letter.

   SQL> select initcap (ename) from emp;

```
INITCAP(EN
-------
Ali
Haji
Mohammad
Ravi
Irfath
```

9. **length**: length() function is used to calculate the length of the given attribute.

   SQL> select ename,length(ename) from emp;

```
ENAME      LENGTH(ENAME)
---------- -------------
ali                    3
haji                   4
mohammad               8
ravi                   4
irfath                 6
```

10. **substr**:substr() function is used to find the substring of the given attribute value. It retuns size-1 of the given string/ attribute as a sub string.

    SQL> select ename, substr(ename,4) from emp;

```
ENAME      SUBSTR(ENAME,4)
---------- ---------------------------------
ali
haji                    i
mohammad   ammad
ravi                    i
irfath                  ath
```

11. **instr**: instr() function return the location of starting passion of the sub string in the existing value.

    SQL> select instr('welcome to CRRCOE','to') from dual;

INSTR('WELCOMETO CRRCOE','TO')

9

c) Date functions:

1. **Sysdate()**: sysdate() function returns the current system date.

   SQL> select sysdate from dual;

   SYSDATE

   28-APR-21

2. **next_day()**; it reurns the date of next coming day .

   SQL> select next_day(sysdate,'sunday') from dual;

   NEXT_DAY(

   02-MAY-21

3. **add_months()**: it returns the next date after adding number of months in the orguments.

   SQL> select add_months(sysdate,5) from dual;

   ADD_MONTH

   28-SEP-21

4. **last_day()**: The LAST_DAY() function takes a date value as argument and returns the last day of month in that date

   SQL> select last_day(sysdate) from dual;

   LAST_DAY(

   30-APR-21

   SQL> select last_day('02-FEB-2020') from dual;

   LAST_DAY(

29-FEB-20

5. **months_between()**: it returns the numbers of months between given two dates.

   SQL> select months_between('02-feb-2021','02-feb-2020') from dual;

   MONTHS_BETWEEN('02-FEB-2021','02-FEB-2020')

   12

   SQL> select months_between(sysdate,'02-feb-2020') from dual;

   MONTHS_BETWEEN(SYSDATE,'02-FEB-2020')

   14.8600769

6. **least()**: it retuns least value from the given argument or attributes.

   SQL> select least(300,450,100,440) from dual;

   LEAST(300,450,100,440)

   100

7. **greatest()**: it returns maximum values from the given arguments or attributes in the relation.

   SQL> select greatest(300,450,100,440) from dual;

   GREATEST(300,450,100,440)

   450

8. **trunc()**: The TRUNC() function returns a DATE value truncated to a specified unit.

   SQL> select trunc(sysdate,'mm') from dual;

   TRUNC(SYS

   01-APR-21

   SQL> select trunc(sysdate,'yyyy') from dual;

   TRUNC(SYS

01-JAN-21

9. **round()**: Round function round a number to a specified length or precision.

   SQL> select round(12.49,0) from dual;

   ROUND(12.49,0)

   12

   SQL> select round(12.51,0) from dual;

   ROUND(12.51,0)

   13

10. **to_char()**: it convert the given date type attribute values to text and return the date in the specific format.

    SQL> select to_char(sysdate,'yyyy-mm-dd') from dual;

    TO_CHAR(SY

    2021-04-28

EXPERIMENT: 5 AND EXPERIMENT:6

AIM:i. Create a simple PL/SQL program which includes declaration section, executable section and exception –Handling section (Ex. Student marks can be selected from the table and printed for those who secured first class and an exception can be raised if no records were found).

ii. Insert data into student table and use COMMIT, ROLLBACK and SAVEPOINT in PL/SQL block..

i). We have to create the student table and insert the records in to the table as follows:

SQL> create table student(sid number(10),sname varchar2(20),rank varchar(10));

Table created.

SQL> insert into student values(501,'Ravi','second');

1 row created.

SQL> insert into student values(502,'Raju','third');

1 row created.

SQL> insert into student values(503,'Ramu','');

1 row created.

SQL> select *from student;

SID	SNAME	RANK
501	Ravi	second
502	Raju	third
503	Ramu	

PL/SQL CODE:

SQL>ed 5a

Enter the following code into the text editor and save the file with .sql format

```
set serveroutput on;
declare
   temp1 number(10);
   temp2 varchar2(10);

begin
select sid,sname into temp1,temp2 from student where rank='first';
dbms_output.put_line('Student No:'|| temp1 ||'  Name:'||temp2||' got first
rank');
exception
when no_data_found then
dbms_output.put_line('*****************************************');
dbms_output.put_line('# Error: there is no student got first rank');
end;
/
```

SQL> @5a;
****
Error: there is no student got first rank

PL/SQL procedure successfully completed.

SQL> update student set rank='first' where sid=503;
1 row updated.

SQL> select *from student;

SID	SNAME	RANK
501	Ravi	second
502	Raju	third
503	Ramu	first

SQL> @5a
Student No:503 Name:Ramu got first rank

PL/SQL procedure successfully completed.

ii)

SQL> select *from student;

```
    SID SNAME          RANK
------- ---------- -------
      501 Ravi           second
      502 Raju           third
      503 Ramu           first
```

PL/SQL CODE:

SQL>ed 5b

Enter the following code into the text editor and save the file with .sql format

```
set serveroutput on;

DECLARE
        sno student.sid%type;
        name student.sname%type;
        srank student.rank%type;

BEGIN
        sno := &sno;
        name := '&name';
        srank := '&srank';
        INSERT into student values(sno,name,srank);
        dbms_output.put_line('One record inserted');
        COMMIT;
        -- adding savepoint
        SAVEPOINT s1;
        -- second time asking user for input
        sno := &sno;
        name := '&name';
        srank := '&srank';
        INSERT into student values(sno,name,srank);
        dbms_output.put_line('One record inserted');
        ROLLBACK TO SAVEPOINT s1;
END;
/
```

SQL> @5b;

```
SQL> @5b
Enter value for sno: 504 old
7:              sno := &sno;
new   7:        sno := 504;
Enter value for name: ali
old   8:        name := '&name';new
8:              name := 'ali'; Enter
value for srank: first old   9:  srank
:= '&srank';new   9:            srank
:= 'first';

Enter value for sno: 505 old
16:             sno := &sno;new
16:             sno := 505;
Enter value for name: haji
old  17:        name := '&name';new
17:             name := 'haji'; Enter
value for srank: third old  18: srank
:= '&srank';new  18:            srank
:= 'third'; One record inserted
One record inserted

PL/SQL procedure successfully completed.SQL>

select *from student;

      SID SNAME       RANK
--------- ---------- --------
      501 Ravi           second
      502 Raju           third
      503 Ramu           first
      504 ali            first
```

EXPERIMENT:7

AIM:Develop a program that includes the features NESTED IF, CASE and CASE expression. The program

can be extended using the NULLIF and COALESCE functions.

A. NESTED IF:

A nested if-then is an if statement that is the target of another if statement. Nested if-then statements mean an if statement inside another if statement

Syntax:-

```
if (condition1) then
  -- Executes when condition1 is true
  if (condition2) then
    -- Executes when condition2 is true
  end if;
end if;
```

PL/SQL CODE: PL/SQL Program to find biggest of three number using nested if.

SQL>ed 6a

Enter the following code into the text editor and save the file with .sql format

```
declare
        a number:=10;
        b number:=12;
        c number:=5;
begin
        dbms_output.put_line('a='||a||' b='||b||' c='||c);
        if a>b AND a>c then
                dbms_output.put_line('a is greatest');
        else
                if b>a AND b>c then
                        dbms_output.put_line('b is greatest');
                else
                        dbms_output.put_line('c is greatest');
                end if;
        end if;
end;
/
```

SQL> @6a

a=10 b=12 c=5

b is greatest

PL/SQL procedure successfully completed.

B. CASE and CASE Expression : CASE statement selects one sequence of statements to execute. However, to select the sequence, the **CASE** statement uses a selector rather than multiple Boolean expressions. A selector is an expression, the value of which is used to select one of several alternatives.

Syntax

```
CASE selector
  WHEN 'value1' THEN S1;
  WHEN 'value2' THEN S2;
  WHEN 'value3' THEN S3;
  ...
  ELSE Sn;  -- default case
END CASE;
```

SQL> create table emp(eno number(5), ename varchar2(10), loc varchar(10), salary number(10,2));

Table created.

SQL> insert into emp values(101,'ali','vja',15000);

1 row created.

SQL> insert into emp values(102,'ravi','hyd',25000);

1 row created.

SQL> insert into emp values(103,'raju','gnt',35000););

1 row created.

SQL> insert into emp values(104,'rakesh','vja',45000);

1 row created.

SQL> select *from emp;

ENO	ENAME	LOC	SALARY
101	ali	vja	15000
102	ravi	hyd	25000
103	raju	gnt	35000
104	rakesh	vja	45000

Example of CASE Expression:

SQL> select loc, case(loc) when 'vja' then salary+2000 when 'hyd' then salary+1000 else salary end "rev_salary" from emp;

LOC	rev_salary
vja	17000
hyd	26000
gnt	35000
vja	47000

PL/SQL CODE: PL/SQL CODE to demonstrate CASE

SQL> ed 6b

```
set serveroutput on;
declare
grade char(1);
begin
grade:='&grade';
case
        when grade='a' then
                dbms_output.put_line('Excellent');
        when grade='b' then
                dbms_output.put_line('very good');
        when grade='c' then
                dbms_output.put_line('good');
        when grade='d' then
                dbms_output.put_line('fair');
        when grade='f' then
                dbms_output.put_line('poor');
        else
                dbms_output.put_line('No such grade');
end case;
end;
/
```

SQL> @6b
Enter value for grade: c
old 4: grade:='&grade';
new 4: grade:='c';
good

PL/SQL procedure successfully completed.
SQL> @6b
Enter value for grade: g
old 4: grade:='&grade';

new 4: grade:='g';
No such grade

PL/SQL procedure successfully completed.

C. NULLIF: Takes two arguments. If the two arguments are equal, then NULL is returned. otherwise the first argument is returned.

Syntax: **select column_name, NULLIF(argument1,arguement2) from table_name;**

Example:

SQL> select ename, nullif('ali','ali1') from emp;

```
ENAME      NUL
-------    ----
ali          ali
ravi         ali
raju         ali
rakesh       ali
```

SQL> select ename, nullif('ali','ali') from emp;

```
ENAME      NUL
-------    --
ali
ravi
raju
rakesh
```

D. COALESCE: COALESCE () function accepts a list of arguments and returns the first one that evaluates to a non-null value.

Syntax: **coalesce("expression1","expression2",...);**

Example:

SQL> select coalesce(NULL,'CRRCOE','IT') from dual;

```
COALE

CRRCOE
```

EXPERIMENT:8

AIM:Program development using WHILE LOOPS, numeric FOR LOOPS, nested loops using ERROR

Handling, BUILT –IN Exceptions, USE defined Exceptions, RAISEAPPLICATION ERROR.

A. WHILE LOOP: A **WHILE LOOP** statement in PL/SQL programming language repeatedly executes a target statement as long as a given condition is true.

Syntax:

WHILE condition LOOP

sequence_of_statements

END LOOP;

PL/SQL Code: A PL/SQL Program to find sum of ODD number upto given number using While loop

SQL> ed 7a

```
set serveroutput on;
declare
        inval number;
        endval number;
        s number default 0;
begin
        inval:=1;
      endval:=&endval;
        while inval<endval loop
                s:=s+inval;
                inval:=inval+2;
        end loop;
dbms_output.put_line('sum of odd numbers between 1 and '||endval||' is '|| s);
end;
/
```

SQL> @7a

Enter value for endval: 100

old 7: endval:=&endval;

new 7: endval:=100;

sum of odd numbers between 1 and 100 is 2500

PL/SQL procedure successfully completed.

B. FOR Loop: A **FOR LOOP** is a repetition control structure that allows us to efficiently write a loop

that needs to execute a specific number of times.

Syntax

FOR counter IN initial_value .. final_value LOOP

sequence_of_statements;

END LOOP;

PL/SQL CODE: A PL/SQL code to print multiplication table using for loop

SQL> ed 7b

```
set serveroutput on;
DECLARE
   VAR1 NUMBER;
   VAR2 NUMBER;
BEGIN
  dbms_output.put_line('Enter number to print multiplication table');
  VAR1:=&VAR1;
  FOR VAR2 IN 1..10 LOOP
       DBMS_OUTPUT.PUT_LINE(VAR1||'X'||VAR2||'='||VAR1*VAR2);
   END LOOP;
END;
/
```

SQL> @7b

Enter value for var1: 2
old 6: VAR1:=&VAR1;
new 6: VAR1:=2;
Enter numer to print multiplication table
2X1=2
2X2=4
2X3=6
2X4=8
2X5=10
2X6=12
2X7=14
2X8=16
2X9=18
2X10=20

PL/SQL procedure successfully completed.

C. NESTED LOOP: PL/SQL allows using one loop inside another loop. It may be either basic, while or for loop.

Syntax:

```
WHILE condition1 LOOP
  sequence_of_statements1
      WHILE condition2 LOOP
        sequence_of_statements2
      END LOOP;
END LOOP;
```

PL/SQL CODE: A PL/SQL program to print n prime number using nested loop.

SQL> ed 7c

```
DECLARE
  i number(3);
  j number(3);
BEGIN
  i := 2;
  LOOP
    j:= 2;
    LOOP
      exit WHEN ((mod(i, j) = 0) or (j = i));
      j := j +1;
    END LOOP;
  IF (j = i ) THEN
    dbms_output.put_line(i || ' is prime');
  END IF;
  i := i + 1;
  exit WHEN i = 50;
  END LOOP;
END;
/
```

SQL> @7c

2 is prime
3 is prime
5 is prime
7 is prime
11 is prime
13 is prime
17 is prime
19 is prime
23 is prime
29 is prime
31 is prime
37 is prime
41 is prime
43 is prime
47 is prime

PL/SQL procedure successfully completed.

EXPERIMENT:9

AIM:Programs development using creation of procedures, passing parameters IN and OUT of PROCEDURES.

SQL> create table enquiry (enqno1 number(3), fname varchar2(30));

Table created.

SQL> insert into enquiry values (111,'sai');

1 row created.

SQL> insert into enquiry values (112,'sindhu');

1 row created.

PL/SQL CODE to create procedure

SQL> ed findname

```
create procedure findname(enquiryno1 IN number,fname1 OUT varchar2) is
fname2 varchar2(30);
begin
select fname into fname2 from enquiry where enqno1=enquiryno1;
fname1:=fname2;
exception when no_data_found then
raise_application_error(-20100,'The given number is not present');
end;
/
```

SQL> @findname

Procedure created.

PL/SQL Code for calling procedure in program

```
SQL> ed pro8
   set serveroutput on;
   declare
   enqno2 number(5);
   fname2 varchar2(30);
   begin
   enqno2:=&enqno2;
   findname(enqno2,fname2);
   dbms_output.put_line('Person name of equiry id '||enqno2||' is '||fname2);
   end;
   /

SQL> @pro8
Enter value for enqno2: 114
old  5: enqno2:=&enqno2;
new  5: enqno2:=114;
declare
*
ERROR at line 1:
ORA-20100: The given number is not present
ORA-06512: at "SYSTEM.FINDNAME", line 7
ORA-06512: at line 6

SQL> @pro8
Enter value for enqno2: 112
old  5: enqno2:=&enqno2;
new  5: enqno2:=112;

Person name of equiry id 112 is sindhu

PL/SQL procedure successfully completed.
```

EXPERIMENT:10

AIM:Program development using creation of stored functions, invoke functions in SQL statements and write

complex functions.

Sol:

SQL> create table dept(deptno int,dname varchar(10));

Table created.

SQL> insert into dept values(1219,'sai');

1 row created.

PL/SQL CODE to create user define function

```
create or replace function getname(dno number)
return varchar2 as
fname1 varchar2(30);
begin
select dname into fname1 from dept where deptno=dno;
return(fname1);
exception
when no_data_found then
raise_application_error(-20100,'Your entered Department number is not exists');
end;
/
```

SQL> @getname

Function created.

PL/SQL Code for calling function in program

SQL> ed pro9

```
set serveroutput on;
declare
fname2 varchar2(30);
deptno2 number(5);
begin
deptno2:=&deptno;
fname2:=getname(deptno2);
dbms_output.put_line(fname2||' is in dept no '||deptno2);
end;
/
```

SQL> @pro9

Enter value for deptno: 1219

old 5: deptno2:=&deptno;

new 5: deptno2:=1219;

sai is in dept no 1219

PL/SQL procedure successfully completed.

SQL> @pro9

Enter value for deptno: 1001

old 5: deptno2:=&deptno;

new 5: deptno2:=1001;

declare

*

ERROR at line 1:

ORA-20100: Your entered Department number is not exists

ORA-06512: at "SYSTEM.GETNAME", line 9

ORA-06512: at line 6

EXPERIMENT-11

AIM:Program development using creation of package specification,package bodies,private objects,package variables and cursors and calling stored packages.

(l)create a table deptl
->create tabke deptl(dname varchar2(10),deptno number);
->insert into dept values('accounting',10);
->insert into dept values('hr',20);

(2)create a table dept
->create table dept(dno number,vt varchar2(10),dloc varcar2(20));

(3)creating package header create or
replace package test

procedure savedept
(dno in number,dloc in varchar); end ;

(4)creating package body create or replace
package body test

function getdno(dno in number) return varchar

dnum varchar(20); begin select dname into
dnum from dept where deptno=dno; return
dnum; end ; procedure savedept
(dno in number,dloc in varchar)

Vt varchar(20) begin vt: = getno(dno); insert into
dept values(dno,vt,dloc); exception when dup val
on_index then raise_application_error(-
2007,'duplicate'); end ; end ;

(5)Executing procedure exec
test.savedept(10,'vijayawada');

(6)Display the table
->select * from dept;

EXPERIMENT:12

AIM:Develop programs using features parameters in a CURSOR, FOR UPDATE CURSOR, WHERE CURRENT of clause and CURSOR variables.

Sol:

```
SQL> create table customers(id number(3), name varchar2(10), age number(3), address
     varchar2(10), salary number(10,2));
Table created.

SQL> insert into customers values(1,'ramesh',32,'ahmedabad',2000);
1 row created.
SQL> insert into customers values(2,'khilan',25,'Delhi',1500);
1 row created.
SQL> insert into customers values(3,'kaushik',23,'Kota',2000);
1 row created.
SQL> insert into customers values(4,'chitali',25,'Mumbai',6500);
1 row created.
SQL> select *from customers;
```

ID	NAME	AGE	ADDRESS	SALARY
1	ramesh	32	ahmedabad	2000
2	khilan	25	Delhi	1500
3	kaushik	23	Kota	2000
4	chitali	25	Mumbai	6500

```
4 rows selected.

SQL> ed pro10

DECLARE
  c_id customers.id%type;
  c_name
  customers.name%type; c_addr
  customers.address%type;
```

```
SQL> @pro10
1 ramesh ahmedabad
2 khilan Delhi
3 kaushik Kota
4 chitali Mumbai
PL/SQL procedure successfully completed.
```

EXPERIMENT: 13

AIM:Develop programs using before and after triggers, row and statement triggers and instead of triggers.

Sol:

SQL> create table customers(id number(3), name varchar2(10), age number(3), address

varchar2(10), salary number(10,2));

Table created.

SQL> insert into customers values(1,'ramesh',32,'ahmedabad',2000);

1 row created.

SQL> insert into customers values(2,'khilan',25,'Delhi',1500);

1 row created.

SQL> insert into customers values(3,'kaushik',23,'Kota',2000);

1 row created.

SQL> insert into customers values(4,'chitali',25,'Mumbai',6500);

1 row created.

SQL> select *from customers;

ID	NAME	AGE	ADDRESS	SALARY
1	ramesh	32	ahmedabad	2000
2	khilan	25	Delhi	1500
3	kaushik	23	Kota	2000
4	chitali	25	Mumbai	6500

4 rows selected.

PL/SQL Code for creation of trigger while insert / update records into a table.SQL>ed pro11

```
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;
  /
```

SQL> @pro11

Trigger created.

SQL> insert into customers values(5,'Hardik',27,'Mumbai',5500);

Old salary:

New salary: 5500

Salary difference:

1 row created.

SQL> update customers set salary=salary+500 where id=2;

Old salary: 1500

New salary: 2000

Salary difference: 500

1 row updated.

EXPERIMENT-14

AIM:FOR a given set of relation tables perform the following:

a.Creating Views
b.Dropping Views
c.Selecting from a view

a.creating a view

Creating Views

Database views are created using the **CREATE VIEW** statement. Views can be created from a single table, multiple tables or another view.

To create a view, a user must have the appropriate system privilege according to the specific implementation.

The basic **CREATE VIEW** syntax is as follows –

```
CREATE VIEW view_name AS
SELECT column1, column2.....
FROM table_name
WHERE [condition];
```

b.Dropping a view

```
DROP VIEW view_name;
```

Example:

```
DROP VIEW Brazil
```

c.selecting from view

```
SQL > SELECT * FROM Brazil Customers;
```