Introduction To SQL

Data:

Collection of Raw facts. For Example objects like chair, table, computer etc.

Information:

Acquiring Knowledge by observing a meaning full report on the screen

DataBase:

collection of logically related data

DBMS:

It is the general software which facilitates the process of defining, constructing and manuplating Various DataBase applications.

DBMS Softwares:

ACCESS
ORACLE
SYBASE
MySQL
SQL SERVER

ORACLE: Oak Ridge Analytical (or) Automatic Computer logical Engine.

It consists of two parts

1. SQL (or) Sequel
2. PL/SQL

SQL: Structured Query Language

SQL is non-procedural language

- It is used for creating tables, inserting records, ~~modyfin~~ modifying the Records and displaying the required information.

PL/SQL:

PL/SQL stands for procedural logic (or) programmable logic SQL.

- It is the extension of SQL for performing run time activities like stored procedures, cursors etc

SQL Commands:

1. DDL Commands:

   a) CREATE: Used to create tables, views, indexs, procedures etc.

   b) ALTER: It is used to change the structure of the table.

      Eg: Adding new columns to the existing tabl adding constraints, Removing constraints.

   c) DROP: used to Remove table from memory (bot data and table).

   d) RENAME: used to change the name of table.

   e) DESC: used to describe the structure of table

   f) TRUNCATE: used to remove all the records from the table

2) DML Commands:

a) INSERT: Used to insert records into the table

b) DELETE: used to delete all the records from the table (or) selected records from the table

c) UPDATE: Used to modify the records in a table

d) SELECT: Used to retreive the information from one (or) more tables

3) DCL Commands:

a) GRANT:
Used to allocate different permission to the database users.

b) REVOKE:
Used to remove the previllages from Database Users

4) TCL Commands:

a) COMMIT:
Used to save the changes permanently in the database

b) ROLLBACK:
It acts like undo action. It is used to cancel the last action performed

c) SAVEPOINT:
It is used to save the required transactions from point to point

Experiment - 2

working with DDL, DML, DCL and Key Constraints

creating, Altering and Dropping of Tables and Inserting Rows into a Table (use constraints while creating tables) Examples using select Command.

STUDENT(RNO:NUMBER, NAME:CHAR, AGE:NUMBER, BRANCH:CHAR, RANK:NUMBER)

Creation:

```
CREATE TABLE STUDENT
( RNO NUMBER(3) PRIMARY KEY,
  NAME CHAR(20) NOT NULL,
  AGE NUMBER(2) NOT NULL,
  BRANCH CHAR(10) NOT NULL,
  RANK NUMBER(2)
);
```

Alter Command:

Adding a new column phone number (PHNO) to student table

```
ALTER TABLE STUDENT
ADD PHNO NUMBER (10) NOT NULL;
```

INSERTION:

```
INSERT INTO STUDENT VALUES(
&RNO, '&NAME', &AGE, '&BRANCH', &RANK, &PHNO);
```

Queries:

Output:

1) SELECT * FROM STUDENT

RNO	NAME	AGE	BRANCH	RANK	PHNO
1.	KAVYA	~~CSE~~ 19	CSE	1	7702862157
2	BHAVYA	13	IT	2	9491995736
3	SAHITHYA	14	LLB	3	9849402998
4.	REKHA	20	ECE	4	1234567890
5	SURIBABU	30	MBA	5	9087654321

2) SELECT RNO, NAME, BRANCH, PHNO FROM STUDENT.

RNO	NAME	BRANCH	PHNO
1.	KAVYA	CSE	7702862157
2	BHAVYA	IT	9491995736
3	SAHITHYA	LLB	9849402998
4.	REKHA	~~20~~ ECE	1234567890
5	SURIBABU	~~3~~MBA	9087654321

QUERIES:

1) Display all the records in Student table.

SELECT * FROM STUDENT

2) Display RNO, NAME, BRANCH AND PHONE NUMBER IN STUDENT table

SELECT RNO,NAME,BRANCH,PHNO FROM STUDENT;

3) Display All the Recordes with Branch CSE

SELECT * FROM STUDENT WHERE BRANCH = 'CSE'.

4) Display student Names whose Rank is more than 2

SELECT NAME FROM STUDENT WHERE RANK > 2;

~~SELECT~~ * FROM STUDENT;

5) SELECT BRANCH, COUNT (*) FROM STUDENT GROUP BY BRANCH;

3)

R.NO	NAME	AGE	BRANCH	RANK	PHNO
1.	KAVYA	12	CSE	1	7702862157

4) Name

SAHITHYA

REKHA

SURIBABU

5)

Branch	count(*)
CSE	1
IT	1
ECE	21
MBA	1
LLB	1

3) Display All

SELECT * FRO

4) Display s
than 2

SELECT NA

SELECT *

5) SELECT B
BY BRA

6)
```
UPDATE STUDENT2 SET RANK=1 WHERE BRANCH='CSE';
```

update student table with rank=1 whose branch is CSE

```
SELECT * FROM STUDENT2;
```

7) DELETE the records whose branch is ~~CE~~ LLB

```
DELETE FROM STUDENT 2 WHERE BRANCH="LLB';
SELECT * FROM STUDENT2;
```

8) Revoke the table student 2

```
DROP TABLE STUDENT2;
```

output

6)

RNO	NAME	AGE	BRANCH	RANK	PHNO
1	KAVYA	12	CSE	1	7702862157
2	BHAVYA	13	IT	2	9491995736
3	SAHITHYA	14	LLB	3	9849402992
4	REKHA	20	ECE	4	1234567890
5	SURIBABU	30	MBA	5	9087654321
6	DURGA	13	IT	3	9867543210
7	VARSHINI	12	CSE	1	1234509876
8	VAGDEVI	13	CSE	1	7890654321
9	SUPRIYA	11	LLB	6	9078563412
10	SRILATHA	14	MBA	2	6543217890

7) Output:
2 rows deleted

RNO	NAME	AGE	BRANCH	RANK	PHNO
1	KAVYA	12	CSE	1	7702862157
2	BHAVYA	13	IT	2	9491995736
4	REKHA	20	ECE	4	1234567890
5	SURIBABU	30	MBA	5	9087654321
6	DURGA	13	IT	3	9867543210
7	VARSHINI	12	CSE	1	1234509876
8	VAGDEVI	13	CSE	1	7890654321
10	SRILATHA	14	MBA	2	6543217890

8 rows selected.

Experi

6) UPDATE STUDENTS

update student table

CSE

SELECT * FROM STUDE

7) DELETE the r

DELETE FROM

SELECT * FRO

8) Revoke the tabl

DROP TABLE

DCL - DATA CONTROL LANGUAGE:

1. GRANT:

It is used to grant the permissions (or) previleges to the database users

2) REVOKE:

It is used to cancel the permissions from the database users.

These two are performed by database administrators

3) COMMIT:

It is used to store the records permanently in the Database.

4) ROLL BACK:

It is used to cancel the last recently performed transaction.

Experiment - 3

Working with Queries and Nested Queries

Queries (along with sub Queries) using ANY, ALL, IN, EXISTS, NOT EXISTS, UNION, INTERSECT and constraints

1. ANY:

This will return TRUE if any of the sub query Value meet the Condition

2. ALL:

This will return TRUE if all of the sub query Values meet the condition

3. IN:

This will return TRUE if operand is equal to one of the list of values

4. EXISTS:

This will return TRUE if sub query returns one (or) more records

5. NOT EXISTS:

This will return TRUE if sub query not returns one (or) more records

6) SET OPERATORS:

1. UNION:

This will combine the result of two (or) more SELECT statements. It avoids duplication

Syntax:

```
SELECT statement 1
UNION
SELECT statement 2;
```

2. INTERSECT:

This will return only common records returned by two (or) more SELECT statement.

Syntax:

```
SELECT statement 1
INTERSET
SELECT statement 2;
```

3. MINUS:

This will return records from the set which does not exists in another set

Syntax:

```
SELECT STATEMENT 1
MINUS
SELECT STATEMENT 2;
```

4) UNION ALL:

This will combine the result of two (or) more SELECT statements including duplications

Syntax:

```
SELECT STATEMENT 1
UNION ALL
SELECT STATEMENT 2;
```

1- Write a query to get empno, ename, job and deptno of managers

```
SELECT EMPNO, ENAME, JOB, DEPTNO FROM EMP E
WHERE EXISTS (SELECT EMPNO FROM EMP WHERE
EMP.MGR = E.EMPNO);
```

2) Write a query to get ename, job who are not managers

```
SELECT ENAME, JOB FROM EMP E WHERE NOT
EXISTS (SELECT MGR FROM EMP WHERE
              MGR = E.EMPNO);
```

output:

1)
```
EMPNO   ENAME    JOB         DEPTNO
-------------------------------------
7566    jones    manager     20
7698    blake    manager     30
7782    clark    manager     10
7839    King     president   10
```
4 rows selected

output

2)
```
ENAME     JOB
------    ------
turner    salesman
ward      salesman
martin    salesman
allen     salesman
millere   clerk
smith     clerk
adems     clerk
james     clerk
```
8 rows selected.

1-Write a query to
of managers

SELECT EMPNO, ENA
WHERE EXISTS (SELE
EMP.MGR = E.EMP

2) Write a query t
managers

SELECT ENAME, JO
EXISTS (SELECT M

3) Write a query to get employee names who are getting any salary of employees working in the department 20.

```
SELECT ENAME FROM EMP WHERE SAL=ANY (SELECT
SAL FROM EMP WHERE DEPTNO=20);
```

4) write a query to get employee names who are getting less salary of any employees working in the department 20

```
SELECT ENAME FROM EMP WHERE SAL<ANY (SELECT
SAL FROM EMP WHERE DEPTNO=20);
```

3) Output:

```
ENAME----
jones
scott
adems
ford
4 rows selected
```

output:

4)
```
ENAME----
james
adems
ward
millere
allen
marKn
clark
blake
jones
scott

10 rows selected
```

3) write a query to get
are getting any salary
in the department 20.

```
SELECT ENAME FROM E
SAL FROM EMP WHERE
```

4) write a query to
getting less salary
in the department

```
SELECT ENAME F
SAL FROM E
```

5) Write a query to get employee names who are belonging to department 10,20,30

```
SELECT ENAME FROM EMP WHERE DEPTNO IN
(10,20,30);
```

6) SET operators :

1. Display different designations in the department 20 and 30

```
SELECT JOB FROM EMP WHERE DEPTNO = 20
UNION
SELECT JOB FROM EMP WHERE DEPTNO = 30 ;
```

5) Output:

```
ENAME
-------
smith
allen
ward
jones
martin
blake
clark
scott
king
turner.
adems

ENAME
-------
james
ford
millere

14 rows selected
```

6) <u>output</u>

```
1 JOB
------
analyst
clerk
manager
salesman
```

5) Write
who ar

SELECT
(10,20,

6) Se

1. Dis
20

SELE
UNI
SEL

2. Display the jobs common to departments 20 and 30

```
SELECT JOB FROM EMP WHERE DEPTNO = 20
INTERSECT
SELECT JOB FROM EMP WHERE DEPTNO = 30;
```

3) Display the jobs unique to the department 20

```
SELECT JOB FROM EMP WHERE DEPTNO = 20
MINUS
SELECT JOB FROM EMP WHERE DEPTNO = 30
MINUS
SELECT JOB FROM EMP WHERE DEPTNO = 10;
```

4) Display the designations in the departments 20 and 30 duplications

```
SELECT JOB FROM EMP WHERE DEPTNO = 20
UNION ALL
SELECT JOB FROM EMP WHERE DEPTNO = 30;
```

2) Output:

```
JOB
---
clerk
manager
```

3) output

```
JOB
--------
analyst
```

4) Output:

```
JOB
-------
clerk
manager
analyst
clerk
analyst
salesman
salesman
salesman
manager
salesman
clerk
11 rows selected
```

2. Displa
20 a
SELEC
INTE
SELE

3) Displ
20
SELECT
MIN
SELEC
MIN
SEL

4) Dis
20
SEL
UN
SE

Experiment - 1
Working with ER Diagram

Basic notations used in E-R model:

strong entity:

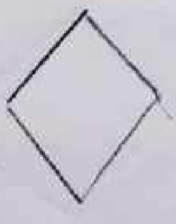
An entity that exists independently of the ~~order~~ other entity types is called strong entity

Weak entity:

An entity that is always dependent on the other entity is called weak entity.

Relationship:

A meaningful association between or among the entity types is called relationship



Experiment-5

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, trunk, round)

Numeric Functions:

a) ABS(n): It returns the absolute values of the x;

Syntax:

```
SELECT ABS(n) FROM DUAL;
```

Eg:

```
SELECT ABS(-2-6) FROM DUAL;
```

b) SQRT(n): It returns the square root value of n. if it is negative null value is returned.

Syntax:

```
SELECT SQRT(n) FROM DUAL;
```

Eg:

```
SELECT SQRT(4) FROM DUAL;
```

Output:

```
ABS(2-6)
--------
     2.6
```

output:

```
SQRT(4)
--------
       2
```

Queries

to-numbe

(concaten

initcap, le

(sysdate

months -

Numeric

a) ABS(n):

Syntax:

SELECT

Eg:

SELECT A

b) SQRT(n):

Syntax:

SELECT

Eg:

SELECT

c) CEIL(n): It returns the smallest integer greater than (or) equal to n

Syntax:

```
SELECT CEIL(n) FROM DUAL;
```

Eg:

```
           77.7
SELECT CEIL(~~[illegible]~~) FROM DUAL;
```

d) FLOOR(n): It returns the largest integer less than (or) equal to n

Syntax:

```
SELECT FLOOR(n) FROM DUAL;
```

Eg:

```
SELECT FLOOR (69.2) FROM DUAL;
```

e) POWER(m,n): It returns m raised to the power n.

Syntax:

```
SELECT POWER(m,n) FROM DUAL;
```

Eg:

```
SELECT POWER (7,2) FROM DUAL;
```

f) EXP(n): It returns exponential value of n

Syntax: SELECT EXP(n) from DUAL;

Eg:

```
SELECT EXP(4) from DUAL;
```

output:

```
CEIL(77.7)
------------
        78
```

output:

```
FLOOR(69.2)
------------
        69
```

output:

```
POWER(7,2)
-------------
        49
```

output:

```
EXP(4)
----------
 54.59815
```

c) CEI

Syntax

Eg:
SEL

d) FLO

Synta

Eg:
S

e) PO

Synt

Eg:

f) E

Synt

Eg:

g) MOD(m,n): It returns the remainder of m divided by n

Syntax:

```
SELECT MOD(m,n) from DUAL;
```

Eg:

```
SELECT MOD(79,10) from DUAL;
```

h) ROUND(m,n): It is used to round the n specify no.of digits after decimal

Syntax:

```
SELECT ROUND(M,N) FROM DUAL;
```

Eg:

```
SELECT ROUND(55.7438,1) FROM DUAL;
```

i) TRUNC(m,n): It is used to Truncates the n specify no.of digits after the decimal

Syntax:

```
SELECT TRUNC(m,n) FROM DUAL;
```

Eg:

```
SELECT TRUNC(79.128,2) FROM DUAL;
```

String functions:

a) length('string'): It is used to return the no.of characters in string.

Syntax:

```
SELECT LENGTH('STRING') FROM DUAL;
```

Eg:

```
SELECT LENGTH('second cse') FROM DUAL;
```

Output:

```
MOD(79,10)
-----------
        9
```

Output:

```
ROUND(55.438,1)
-----------
    55.4
```

output:

```
TRUNC(79.128,2)
-------------
     79.12
```

→ SELECT SIGN(-8), SIGN(9) FROM DUAL;

output:

```
SIGN(-8)        SIGN(9)
--------        -------
   -1              1
```

Output:

10

g) MO

Syntc

Eg:

h) RO

Synta

Eg:

S

i) TRU

n-

Syntax:

Eg:

SE

String f

a) length(

Syntax:

S

Eg:

SELEC

b) lower('string'): It is used to convert the string to lower case letters

Syntax:

```
SELECT LOWER('STRING') FROM DUAL;
```

Eg:

```
SELECT LOWER('CHEC') "result" FROM DUAL;
```

c) UPPER('string'): It converts a string to uppercase

Syntax:

```
SELECT UPPER('STRING') FROM DUAL;
```

Eg:

```
SELECT UPPER('chec') "output" from DUAL;
```

d) INITCAP('string'): It is used to convert the first letter into uppercase letter

Syntax:

```
SELECT INITCAP('STRING') FROM DUAL;
```

Eg:

```
SELECT INITCAR('chec') FROM DUAL;
```

e) REPLACE('string', 'source'string', 'Replace string'): It is used to replace the search string with Replace string

Syntax:

```
SELECT REPLACE('string', 'source string',
        'Replace string') from DUAL;
```

Eg:

```
SELECT REPLACE('midia and midia', 'mi', 'in')
  "replaced" FROM DUAL;
```

output:

```
result
-------
  chec
```

output:

```
output
-------
  CHEC
```

output:

```
output
-------
 Chec
```

output:

```
replaced
--------
india and india
```

b) lower('string'): It is
string to lower

Syntax:

```
SELECT LOWER (STRIN
```

Eg:

```
SELECT LOWER ('CHEC')
```

c) UPPER('string'): It conver

Syntax:

```
SELECT UPPER('STRIN
```

Eg:

```
SELECT UPPER('chec
```

d) INITCAP('string'): It is
first letter into uppe

Syntax:

```
SELECT INITCAP ('STR
```

Eg:

```
SELECT INITCAR(' chec'
```

e) REPLACE('string', 'source'

It is used to replace
with Replace string

Syntax:

```
SELECT REPLACE ('string',
      'Replace string')
```

Eg:

```
SELECT REPLACE ('midi
 "replaced" FROM DUA
```

f) SUBSTR('string', m,n):
It is used to display the searching string from m position to n position

Syntax:
```
SELECT SUBSTR('string',m,n) FROM DUAL;
```
~~g)~~ Eg:
```
SELECT SUBSTR('independence', 3,8) "substring"
from DUAL;
```
g) INSTR('string', 'char'): It returns the position of the first occurrence in the string.

Syntax:
```
SELECT INSTR('string','char') from DUAL;
```
Eg:
```
SELECT INSTR('aeroplane', 'p') "result" FROM DUAL;
```

h) LPAD('string', n, 'wildcard character'); It worked on leftside of the given string and fill that area with specified characters.

Syntax:
```
SELECT LPAD('string', 'n', 'wildcard character')
from DUAL;
```
Eg:
```
SELECT LPAP('cat', 5, '*')" padding output"
FROM DUAL;
```

output:

```
substring
---------
dependen
```

output:

```
result
-------
     5
```

Output:

```
Padding Output
--------------
**cat
```

f) SUBSTR('string', m,n)

It is used

string from m posit

Syntax:

SELECT SUBSTR('st

Eg:

g) SELECT SUBSTR('inde

from DUAL;

g) INSTR('string', 'char'):

of the first occurre

Syntax:

SELECT INSTR('string

Eg:

SELECT INSTR('aeroplan

h) LPAD('string', n, 'wildca

worked on leftside

and fill that area

characters

Syntax:

SELECT LPAD('string',

from DUA

Eg:

SELECT LPAD('cat', 5

FROM DUAL;

i, RPAD('string', n, 'wildcard character'):

It worked on right side of given string and fills that area with specified characters

Syntax:

```
SELECT RPAD('string', n, 'wildcard character')
            FROM DUAL;
```

Eg:

```
SELECT RPAD('doll', 9, '%') FROM DUAL;
```

j) LTRIM('string', 'char'): It is used to trim the character from the string

Syntax:

```
SELECT LTRIM('string', 'char') from DUAL;
```

Eg:

```
SELECT LTRIM('INTERNET', 'IN') "Result" FROM
DUAL;
```

k) RTRIM('string', 'char'): It is used to trim the character from the string

Syntax:

```
SELECT RTRIM('string', 'char') FROM DUAL;
```

Eg:

```
SELECT RTRIM ('internet', 't') "Result" from
dual;
```

DUAL;

<u>output:</u>

```
doll%%%%
```

<u>output:</u>

```
Result
--------
  TERNET
```

<u>output:</u>

```
Result
--------
internet
```

i, RPA

It

ar

ch

Sy

Eg:

j) LT

Sy

Eg:

S

k) RTI

th

Syn

S

Eg:

Miscellaneous Function:

a) GREATEST(list of values): It returns greatest value in given list of values

Syntax:

```
Select GREATEST(v1,v2,v3,...) FROM DUAL;
```

b) LEAST(list of values): It returns smallest value in the given list of values

Syntax:

```
SELECT LEAST(v1,v2,v3...) FROM DUAL;
```

Date functions:

a) SYSDATE: it returns the system date.

Syntax: `SELECT SYSDATE FROM DUAL;`

b) NEXT_DAY('date', 'day name'): it returns the date of next specified day of the week after the date.

Syntax: `SELECT NEXT_DAY('date', 'day name') FROM DUAL;`

c) ADD_MONTHS('date', n): it can add months to given 'date'.

Syntax: `SELECT ADD-MONTHS ('date', n) FROM DUAL;`

d) MONTHS-BETWEEN ('date1', 'date2'); it returns the no.of months between date1 and date2

syntax:

```
SELECT MONTHS-BETWEEN ('date1', 'date2') FROM
                                         DUAL;
```

Conversion Functions:

These functions are to convert the one data type to another data type.

a) TO-CHAR('date', format): it is used to convert the date into the specified character format.

Syntax:

```
SELECT TO-CHAR('Date specification', DDTH-MMTH-YYTH)
                          FROM DUAL;
SELECT TO-CHAR('date specifications', 'ddspth-
         mmspth-yyspth') FROM DUAL;
```

b) TO-DATE('char, format): it is used to convert the character into specified date format;

Syntax:

```
SELECT TO-DATE('date in character', 'date')
      from dual;
```

Queries

```
1) SELECT SYSDATE  FROM dUAL;
```

Output:

```
SYSDATE
---------
  31-MAY-22
```

Conversion Functions:
These functions ar
data type to anoth

a) TO-CHAR ('date',
convert the date in
format.

Syntax:

SELECT TO-CHAR ('Date

SELECT TO-CHAR ('d
mmspth - y

b) TO-DATE ('char, for
the charact
format;

Syntax:
SELECT TO-DATE (
from dua

Queries

1) SELECT SYSDATE

Output:

```
HIREDDATE   ADD-MONTHS(HIREDDA   ADD-MONTHS(HIRED
---------   ------------------   ----------------
09-JUN-81   09-OCT-81            09-FEB-81
17-NOV-81   17-MAR-82            17-JUL-81
23-JAN-82   23-MAY-82            23-SEP-81
```

output:

```
Nearest month
-------------
   01-Apr-71
```

Output:

```
Same months    Diff months
-----------    -----------
    0               2
```

output:

```
SYSDATE      LAST-DAY(SYSDATE)
-------      -----------------
13-AUG-15    31-AUG-15
```

Output:

```
SYSDATE      NEXT-DAY(SYSDATE
-------      ----------------
13-AUG-15    19-AUG-15
```

output:

```
SYSDATE      TO-CHAR(SYSDATE, 'DAY')
-------      -----------------------
13-AUG-15        THURSDAY
```

2) select hireddate, ADD-MONTHS (hireddate,4
ADD-MONTHS (hireddate, -4) from emp
WHERE DEPTNO = 10;

3) SELECT ROUND(TO-DATE ('12-apr-71'), 'MM')
"Nearest month" FROM DUAL;

4) SELECT MONTHS-BETWEEN ('05-jan-98', '05-jan-98') "Same Months",
MONTHS-BETWEEN ('05-mar-98', '05-jan-98')
"Diff months" FROM DUAL;

5) SELECT SYSDATE, LAST-DAY (SYSDATE) FROM DUAL;

6) SELECT SYSDATE, NEXT-DAY (SYSDATE, 'WEDNESDAY') FROM DUAL;

7) SELECT SYSDATE, TO-CHAR (SYSDATE, 'DAY') FROM DUAL;

Output:

```
GREATEST(10,'7',-1)
-------------------
    10
```

output:

```
least
-----
 ABCD
```

output:

```
LOWEST
-------
    -2
```

output:

```
ENAME       HIREDDATE
-----       ---------
smith       17/12/80
jones       02/04/81
scott       19/04/87
adams       23/05/87
ford        03/12/81
```

8) SELECT GREATEST (10, '7', -1) FROM DUAL;

9) SELECT LEAST ('abcd', 'ABCD', 'a', 'XYZ') "least"
FROM DUAL;

10) SELECT LEAST (9,3,56,89,23,1,0,-2,12,34,7,22)
as LOWEST FROM DUAL;

Example

Write a query to convert hireddate of employee as DD/mm/yy for department 20

SELECT ENAME, TO-CHAR (hireddate, 'DD/mm/yy')
AS hireddate FROM EMP WHERE dept no=20;

output:

```
ENO    ENAME    JOB         SALARY
----   ------   -----       -------
7369   smith    clerk       $800
7499   allen    salesman    $1600
7521   ward     salesman    $1250
7666   jones    manager     $12975
7654   martin   salesman    $1250
7698   blake    manager     $12850
7782   clark    manager     $2450
7788   scott    analyst     $3000
7839   king     president   $5000
7844   turner   salesman    $1500
7876   adams    clerk       $1100
7900   James    clerk       $950
7902   Ford     Analyst     $3000
7984   miller   clerk       $1300
```

Output:

```
YY        COUNT(*)
-----     --------
87        2
81        10
82        1
80        1
```

Write a query to display salary of employee with symbol '$'

```
SELECT ENO, ENAME, JOB, TO-CHAR (SAL, '$999')
AS SAlary FROM EMP;
```

Write a query to find the no-of employees who joined in the same year

```
select TO-CHAR (hireddate, 'YY') as YY,
COUNT(*) FROM emp GROUP BY TO-CHAR
(hired date, 'yy');
```

Experiment-6

Develop the programs using control structures.

PL/SQL has a number of control structures which includes:

- conditional controls
- Iterative (or) loop controls

Conditional controls

```
IF... THEN ... END IF;
IF... THEN... ELSE... END IF;
IF ... THEN ... ELSEIF ... THEN ... ELSE ... END IF;
```

Iterative (or) loop controls

PL/SQL Control structures

```
1) LOOP
        ... SQL Statements ...
        EXIT;
   END LOOP;

2) WHILE condition Loop
          ... SQL Statements ...
   END LOOP;

3) FOR <variable> <lowerbounds>.. <Upperbounds>
   loop
   ... ....
   END LOOP;
```

1) write a program to find the given number is even (or) odd number?

```
DECLARE
A NUMBER (5) := &A;
BEGIN
IF MOD(A,2) =0 THEN
DBMS-OUTPUT.PUT-LINE(A||' is a even number');
ELSE
DBMS-OUTPUT.PUT-LINE (A||' is a odd number');
END IF;
END;
```

2) write a program to find the largest/biggest among three numbers.

```
DECLARE
A NUMBER(10):=&A;
B NUMBER(10) :=&B;
C NUMBER (10) := &C;
BEGIN
IF(A>B) AND (A>C) THEN
DBMS-OUTPUT.PUT - LINE(A||' is the biggest number');
ELSE IF(B>C) THEN
DBMS-OUTPUT.PUT- LINE(B||' is the biggest number');
ELSE
DBMS-OUTPUT.PUT-LINE (C||' is the biggest number');
ENDIF;
ENDIF;
END;
```

Output:

```
Enter value for a : 10

old 2 :A NUMBER(5) : = &A;
new 2 :A NUMBER(5) : =10 ;

10 is a even number.
```

Output:

```
Enter value for a : 10
Enter value for b : 30
Enter value for c : 20

30 is the biggest number
```

3) write a program for printing 1 to 10 numbers by using WHILE?

```
DECLARE
A NUMBER(10):=1;
BEGIN
WHILE(A<=10)
LOOP
DBMS-OUTPUT.PUT_LINE(A);
~~A=A~~. A:=A+1;
END LOOP;
END;
```

4) program for printing first 10 natural numbers using FOOR LOOP;

```
DECLARE
A NUMBER(10);
BEGIN
 FOR A IN 1..10
 LOOP
DBMS_OUTPUT.PUT-LINE(A);
END LOOP;
END;
```

Output:

```
1
2
3
4
5
6
7
8
9
10
```

Output:

```
1
2
3
4
5
6
7
8
9
10
```

Experiment - 1

Working with Triggers using PL/SQL

Develop programs using BEFORE AND AFTER Triggers, Row and Statement Triggers and INSTEAD OF Triggers.

Definition

Triggers are stored procedures, which are automatically executed or fired when some eventsoccur. Triggers are, in fact, written to be executed in response to any of the following events -

→ A dabase manipulation (DML) Statement (DELETE, INSERT OR UPDATE)

→ A database definition (DDL) statement (CREATE, ALTER OR DROP)

Creating Triggers - syntax

```
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 view

{INSERT [OR] | UPDATE [OR] | DELETE} - This specifies the 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 trigger

{REFERENCING OLD AS o New AS n] - This allows you to refer new and old values for various DML statements.

[FOR EACH ROW] - This specifies a row-level trigger, i.e, the trigger will be executed for each row being affected

Example program

```
CREATE OR REPLACE TRIGGER update_sal
BEFORE, DELETE OR INSERT OR UPDATE ON EMP
For EACH ROW
WHEN(NEW.EMPNO >0)
DECLARE
   Sal-diff number;
BEGIN
   Sal-diff := New Sal - : Olb Sal;
  dbms_output.put_line('old Salary:'||:OLD.SAL);
 dbms-output.put_line('salary difference:'||
                                   Sal-diff);
  END;
 /
```

when the above record / code is executed at SQL prompt, it produces result as

Trigger Created

Executing a Trigger

```
INSERT INTO EMP(EMPNO,ENAME,SAL)
VALUES(8,'ABCD', 7500);
```

when a record is created in EMP table, the above create trigger, display-Salary-changes will be fired and it will display the following result

```
Old Salary:
    New Salary:7500
    Salary difference:
```

let us Perform one more DML operation on the EMP table. The UPDATE Statement will update an existing record.

```
UPDATE EMP SET SAL=SAL+500 WHERE EMPNO=8;
```

when a record is updated in EMP table, the above Creates trigger, display_Salary_changes will be fixed and it will display as

```
Old Salary: 1500
New Salary: 2000
Salary difference: 500
```

Experiment-8

working with PL/SQL Procedures

Develop Programs using Procedures, Passing Parameters IN and OUT of Procedures.

Definition

A Procedure is a subprogram unit that consists of a group of PL/SQL Statements. Each Procedure in oracle has its own unique name by which it can reffered The subprogram unit is stored as a object These subprograms donot return a value directly; mainly used to Perform an action.

Creating a Procedure

A Procedure is created with CREATE OR REPLACE PROCEDURE statement

Syntax

```
CREATE OR REPLACE PROCEDURE-name
[(Parameter-name [IN/OUT/IN OUT] type[,..])]
{IS/AS}
BEGIN
   <procedure-body>
END
   Procedure-name;
```

where,

→ Procedure - name Specifies the name of Procedure

→ [OR REPLACE] opinion allows the modification of an existing Procedure.

Program

```
DECLARE
 a number;
 b number;
 c number;
PROCEDURE findMIN(x IN number, y IN number,
IS                        z out number)
 BEGIN
  If x<y THEN
     z:=x;
  ELSE
     z:=y;
  END IF;
END;
BEGIN
 a:23;
 b:45;
 find MIn (a,b,c);
 dbms-output.put_line('Minimum of (23,45:'||c);
END;
/
```

output

```
Minimum of (23,45):23
PL/SQL Procedure successfully completed
```

→ Procedure - name specifie
Procedure

→ [OR REPLACE] opinion
of an existing Proced

Program

```
DECLARE
  a number;
  b number;
  c number;
PROCEDURE findMIN( x
                   z
IS
 BEGIN
  If x<y THEN
     z:=x;
   ELSE
     z:=y;
   END IF;
 END;
 BEGIN
   a:23;
   b:45;
   find MIN (a,b,c
   dbms-output.
   END;
  /
```

when the above code is executed at SQL Prompt, it Produces the result

Example Program-3 IN OUT Example.

```
DECLARE
   a number;
   PROCEDURE SquareNUm (x IN OUT number) IS
  BEGIN
   a:23;
   Square (Num (a));
   dbms-output.put-line('Square of (23:)'||a);
  END;
/
```

when the above code is executed at the SQL Prompt, it Produces the result

Program

```
CREATE OR REPLACE PROCEDURE GREETINGS
AS
BEGIN
DBMS-OUTPUT.PUT-LINE('Hello world');
END;
/
```

output

```
Square of (23): 529
PL/SQL Procedure Successfully Completed
```

output

```
Hello world
PL/SQL Procedure Successfully Completed
```

Experiment-9

Working with Functions using PL/SQL

Develop Programs using Stored Functions, invoke functions in SQL statements and Write Complex Functions.

Creating a function

A Standalone function is Created using the CREATE Function Statement

Syntax

```
CREATE [OR REPLACE] Function function-name
[(parameter-name[IN|OUT/INOUT]type[,...])]
Return return_datatype
{IS/AS}
BEGIN
      <function-body>
END;
```

where,

→function_name Specifies the name of the function

→The function must contain a return Statement

→The Return clause specifies the datatype you are going to return from the function

→function body contains the executable part

→ The AS keyword is used of IS keyword for Creating a Standalone function

Example Program-1

The following example illustrates how to create and call a standalone function

```
create or replace function total Employees
Return number IS
   total number(9):=0;
BEGIN
  Select COUNT(*) into total FROM EMP;
  Return total;
  END;
 /
```

when the above code is executed using SQL Prompt, it will Produce following result

Calling function

To call a function, we simplify needs to pass the required Parameters along with the function name and if the function on returns a value, then we can store the returned value

```
Declare
  c number(2);
Begin
```

<u>output</u>

Function Created

```
  C:=total Employees();
 dbms_output.put_line('Total no.of employees:'||c);
 END;
 /
```

Example Program-2 - - -> Recursive functions

```
Declare
  num mumber;
Function fact(x mumber)
  Return mumber
 IS
 K Number;
 Begin
  If k=0 then
     k:=1;
  ELSE
    K:=x*fact(x-1);
 END IF;
  Return k;
END;
BEGIN
  num:=6
 factorial:=fact(num);
 dbms_output.put.line('Factorial'||num 'is'||
                                  factorial);
 END;
 /
```

output

```
Total no.of employees: 14
PL/SQL Procedure successfully Completed
```

output

```
Factorial 6 is 720
PL/SQL Procedure successfully Completed
```

Experiment-10

working with cursors using PL/SQL

Develop Programs using cursors, Parameters in a cursor

what is cursor

A Cursor is a temporary work area created in the System memory when a SQL Statement is executed. A cursor contains information on a Select statement and the rows of data accessed by it. This temporary work area is used to Store the data received. The database and manipulate. This data A Cursor can hold more than one row, but can process only one row at a time.

There are two types of cursors in PL/SQL

A) Implicit cursors

These are created by default when DML statements like insert, update and delete Statements are executed

Implicit Cursor Attributes:

1) SQL % Found

The return value is True, if the DML statements like insert, delete and update affect atleast one raw and if Select_-info statement return atleast one row.

2. SQL%.Not found

The return value is False, if DML Statements like insert, delete and update atleast one row and it Select--info Statement return atleast one row

3 %. Row count

Return the number of rows affected by DML operations insert, delete, update, select

Program to demonstrate implicit cursors

```
Declare
    total_rows number(2);
Begin
    update emp set sal=sal+500;
if sql%.not found then
 dbms_output.put_line('no employees are selected
                      for updation');
ELSE if sql%.not found then
   total_rows:=sql%.rowcount;
dbms-output put_line('total rows||' employees
                      are updated);

END IF;

END;
/
```

output

```
14 employees are updated
```

B) Explicit cursors
They must be created when you are executing a select statement that returns more than one row. It is created on a select statement which returns more than one row.

Syntax

CURSOR cursor-name IS select-stmt;

Declaring the cursor
Declaring the cursor defines the cursor with a name and the associated SELECT statement.
For example - CURSOR C1 IS SELECT empno, ename, sal FROM emp;

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 C1;

Fetching the cursor
Fetching the cursor involves accessing one row at a time. For example, we will fetch rows from the above opened cusor as follows-

FETCH C1 INO C1_name, cl_sal;

closing the cursor

closing the cursor means releasing the allocated memory. For example, we will choose and close the above-opened cursor as follows.

Example Program to demonstrate Explicit cursors.

```
DECLARE
CURSOR RES IS SELECT * FROM EMP03;
BEGIN
R EMPO 3%. ROWTYPE;
OPEN RES;
loop
FETCH RES INTO R;
EXIT WHEN RES%. NOT FOUND;
DBMS-OUTPut.PUT-LINE(R.ENO||R.ENAME);
END LOOP;
CLOSE RES;
END;
/
```