

# DBMS LAB

## Lab 1 – Introduction to Oracle

### Topics

1. Oracle versions
2. Database Users (Scott and System), Login screen
3. Oracle follows client server architecture
4. SQL and PL/SQL
5. Introduction to DDL, DML, DCL and TCL
6. DQL select statement
7. Where clause (AND, OR and NOT), IN statement
8. Relational operators (=, <>, !=, ^=, <,>, <=,>=)
9. Removing duplicates (Unique and distinct)
10. Between and clause (Range Searching)
11. Like Clause (pattern matching)
12. Column Renaming
13. Null values
14. Dual table
15. Order by clause (ASC and DESC)

Q.1. Display the details of all employees.

SQL >

Select \* from EMP;

OUTPUT:								
	EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
-	7369	SMITH	CLERK	7902	17-DEC-80	800		
20.	7499	ALLEN	SALESMAN	7698	20-FEB-81	1600	300	
30	7521	WARD	SALESMAN	7698	22-FEB-81	1250	500	
30	7566	JONES	MANAGER	7839	02-APR-81	2975		
20	7654	MARTIN	SALESMAN	7698	28-SEP-81	1250	1400	
30	7698	BLAKE	MANAGER	7839	01-MAY-81	2850		
30	7782	CLARK	MANAGER	7839	09-JUN-81	2450		
10	7788	SCOTT	ANALYST	7566	09-DEC-82	3000		
20	7839	KING	PRESIDENT		17-NOV-81	5000		
10	7844	TURNER	SALESMAN	7698	08-SEP-81	1500	0	
30	7876	ADAMS	CLERK	7788	12-JAN-83	1100		
20	7900	JAMES	CLERK	7698	03-DEC-81	950		
30								

## DBMS LAB

20	7902 FORD	ANALYST	7566 03-DEC-81	3000
10	7934 MILLER	CLERK	7782 23-JAN-82	1300

14 rows selected.

Q.2 Display the details of all departments.

SQL >

Select \* from dept;

OUTPUT:

DEPTNO	DNAME	LOC
10	ACCOUNTING	NEW YORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO
40	OPERATIONS	BOSTON

Q.3 Display the name and job for all employees.

SQL >

Select ename, job from EMP;

OUTPUT:

ENAME	JOB
SMITH	CLERK
ALLEN	SALESMAN
WARD	SALESMAN
JONES	MANAGER
MARTIN	SALESMAN
BLAKE	MANAGER
CLARK	MANAGER
SCOTT	ANALYST
KING	PRESIDENT
TURNER	SALESMAN
ADAMS	CLERK
JAMES	CLERK
FORD	ANALYST
MILLER	CLERK

14 rows selected.

Q.4 Display the name of department which is located in 'CHICAGO'.

SQL >

Select dname from DEPT where loc = 'CHICAGO';

## DBMS LAB

OUTPUT:

DNAME

-----  
SALES

Q.5 Display the names of all employees who are working in department number 10.

SQL >

select ename from EMP where deptno = '10';

OUTPUT:

ENAME

-----  
CLARK

KING

MILLER

Q.6 Display the names of all employees working as clerks and drawing a salary greater than 1000.

SQL >

select ename from EMP where job = 'CLERK' and sal > 1000;

OUTPUT:

ENAME

-----  
ADAMS

MILLER

Q.7 Display employee number and names for employees who earn commission.

SQL >

select empno, ename from EMP where comm > 0;

OUTPUT:

EMPNO ENAME

-----  
7499 ALLEN

7521 WARD

7654 MARTIN

Q.8 Display names of employees who are not eligible for commission.

SQL >

select ename from EMP where comm is NULL;

OUTPUT:

ENAME

## DBMS LAB

```
SMITH  
JONES  
BLAKE  
CLARK  
SCOTT  
KING  
ADAMS  
JAMES  
FORD  
MILLER
```

10 rows selected.

Q.9 Display the names of employees who are working as clerk, salesman or analyst and drawing a salary more than 2500.

SQL>

select ename from EMP where job IN ('CLERK', 'ANALYST', 'SALESMAN') AND sal > 2500;

OUTPUT:

```
ENAME  
-----  
SCOTT  
FORD
```

Q.10 Display the names of employees who are working in department number 10 and are manager.

SQL>

select ename from EMP where deptno = '10' and job = 'MANAGER';

OUTPUT:

```
ENAME  
-----  
CLARK
```

Q.11 Display the list of employees who have joined the company before 30th june 90 or after 31st dec 90.

SQL>

select ename from EMP where hiredate BETWEEN '01-01-1980' and '31-12-1981';

OUTPUT:

```
ENAME  
-----  
SMITH  
ALLEN  
WARD  
JONES  
MARTIN  
BLAKE
```

## DBMS LAB

```
CLARK  
SCOTT  
KING  
TURNER  
ADAMS  
JAMES  
FORD  
MILLER
```

14 rows selected.

Q.12 Display the names of all employees who do not get commission.

SQL >

select ename from EMP where comm is NULL;

OUTPUT:

ENAME

```
-----  
SMITH  
JONES  
BLAKE  
CLARK  
SCOTT  
KING  
TURNER  
ADAMS  
JAMES  
FORD  
MILLER
```

11 rows selected.

Q.13 Display the names of employees working in department number 10 or 20 or --employees working as clerks, salesman or analyst.

SQL >

select ename from EMP where job IN ('CLERK', 'SALESMAN', 'ANALYST') or dept no IN (10, 20);

OUTPUT:

ENAME

```
-----  
SMITH  
ALLEN  
WARD  
JONES  
MARTIN  
CLARK  
SCOTT  
KING  
TURNER  
ADAMS  
JAMES
```

## DBMS LAB

FORD  
MILLER

13 rows selected.

Q.14 Display the names of employees whose name starts with alphabet S.

SQL >

Select ename from EMP where ename like 'S%';

OUTPUT:  
ENAME

-----  
SMITH  
SCOTT

Q.15 Display employee name from employees whose name ends with alphabet S.

SQL >

Select ename from EMP where ename like '%S';

OUTPUT:  
ENAME

-----  
JONES  
ADAMS  
JAMES

Q.16 Display the names of employees whose names have second alphabet A in their names.

SQL >

Select ename from EMP where ename like '\_A%';

OUTPUT:  
ENAME

-----  
WARD  
MARTIN  
JAMES

## DBMS LAB

Q.17 Display the names of employees whose name is exactly four characters in length.

SQL >

select ename from EMP where LENGTH(ename) = 4;

OUTPUT :

ENAME

-----

WARD

KING

FORD

Q.18 Display unique jobs available in company.

SQL >

select distinct job from EMP;

OUTPUT :

JOB

-----

CLERK

SALESMAN

PRESIDENT

MANAGER

ANALYST

Q.19 Display the names of employees and their salaries in descending order of salary.

SQL >

select ename, sal from EMP order by sal desc;

OUTPUT :

ENAME

SAL

-----

KING

5000

FORD

3000

SCOTT

3000

JONES

2975

BLAKE

2850

CLARK

2450

ALLEN

1600

TURNER

1500

MILLER

1300

WARD

1250

MARTIN

1250

ADAMS

1100

JAMES

950

SMITH

800

14 rows selected.

## DBMS LAB

Q.20 Display the names of employees in ascending order of their names.

SQL >

select ename from EMP order by ename;

OUTPUT:

ENAME

-----  
ADAMS  
ALLEN  
BLAKE  
CLARK  
FORD  
JAMES  
JONES  
KING  
MARTIN  
MILLER  
SCOTT  
SMITH  
TURNER  
WARD

14 rows selected.

Q.21 Display the names and salaries of employees in ascending order of salary and descending order of names.

SQL >

select ename, sal from EMP order by sal;

OUTPUT:

ENAME	SAL
SMITH	800
JAMES	950
ADAMS	1100
WARD	1250
MARTIN	1250
MILLER	1300
TURNER	1500
ALLEN	1600
CLARK	2450
BLAKE	2850
JONES	2975
SCOTT	3000
FORD	3000
KING	5000

14 rows selected.

## DBMS LAB

Q.22 Display the name of employees and job as "SCOTT - ANALYST".

SQL >

```
select concat(ename, ' - ', job) as "ename-job" from EMP;
```

OUTPUT:

ename-job

-----  
SMITH-CLERK  
ALLEN-SALESMAN  
WARD-SALESMAN  
JONES-MANAGER  
MARTIN-SALESMAN  
BLAKE-MANAGER  
CLARK-MANAGER  
SCOTT-ANALYST  
KING-PRESIDENT  
TURNER-SALESMAN  
ADAMS-CLERK  
JAMES-CLERK  
FORD-ANALYST  
MILLER-CLERK

14 rows selected.

Q.23 Display the name of the president.

SQL >

```
select ename from EMP where job = 'PRESIDENT';
```

OUTPUT:

ENAME

-----  
KING

Q.24 Display the names of employees who are not managers.

SQL >

```
select ename from EMP where job not IN ('MANAGER');
```

OUTPUT:

ENAME

-----  
SMITH  
ALLEN  
WARD  
MARTIN  
SCOTT  
KING

## DBMS LAB

```
TURNER  
ADAMS  
JAMES  
FORD  
MILLER
```

11 rows selected.

Q.25 Display the names, salary and department number of all employees and order them department wise using relative position of their select list.

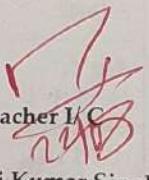
SQL >

~~select ename, sal, dept no from EMP order by deptno;~~

OUTPUT:

ENAME	SAL	DEPTNO
CLARK	2450	10
KING	5000	10
MILLER	1300	10
JONES	2975	20
FORD	3000	20
ADAMS	1100	20
SMITH	800	20
SCOTT	3000	20
WARD	1250	30
TURNER	1500	30
ALLEN	1600	30
JAMES	950	30
BLAKE	2850	30
MARTIN	1250	30

14 rows selected.

  
Teacher I/C  
Mr. Raj Kumar Singh

# DBMS LAB

## Lab 2 Oracle Functions

### 1. Arithmetic Functions

SN	Function	Description
1.	$\sin(x)$	Returns the sine of x where x is in radians.
2.	$\cos(x)$	Returns the cosine of x where x is in radians.
3.	$\tan(x)$	Returns the tangent of x where x is in radians.
4.	$\sinh(x)$	Returns the hyperbolic sine of x.
5.	$\cosh(x)$	Returns the hyperbolic cosine of x.
6.	$\tanh(x)$	Returns the hyperbolic tangent of x.
7.	$\text{asin}(x)$	Returns the arc sine of x.
8.	$\text{acos}(x)$	Returns the arc cosine of x.
9.	$\text{atan}(x)$	Returns the arc tangent of x.
10.	$\text{sign}(x)$	Returns the sign value of x.
11.	$\text{abs}(x)$	Returns the absolute value of x.
12.	$\text{mod}(m,n)$	Returns the remainder of m divided by n. $[\text{mod}(m, n) = m - n * \text{floor}(m/n)]$
13.	$\text{trunc}(m,n)$	Returns a number truncated to n number of decimal places.
14.	$\text{round}(m,n)$	Returns a number rounded to n number of decimal places.
15.	$\text{ceil}(x)$	Returns the smallest integer greater than or equal to the number x.
16.	$\text{floor}(x)$	Returns the greatest integer smaller than or equal to the number x.
17.	$\log(m,n)$	Returns the logarithm of n base m.
18.	$\ln(x)$	Returns the natural logarithm of x.
19.	$\sqrt{x}$	Returns the square root of x.
20.	$\exp(x)$	Returns e raised to the nth power, where e = 2.71828183.
21.	$\text{power}(m,n)$	Returns m raised to the nth power.
22.	$\text{greatest}(m,n,p,\dots)$	Returns the greatest value in a list of expressions.
23.	$\text{least}(m,n,p,\dots)$	Returns the least value in a list of expressions.

### 2. Aggregate Functions (Group Functions)

SN	Function	Description
1.	$\text{count}(\text{colname})$	Returns the number of rows in the column name specified.
2.	$\text{count}(\ast)$	Returns the number of rows in the table.
3.	$\text{min}(\text{colname})$	Returns the min value for an expression.
4.	$\text{max}(\text{colname})$	Returns the max value for an expression.
5.	$\text{avg}(\text{colname})$	Returns the avg value for an expression.
6.	$\text{sum}(\text{colname})$	Returns the sum value for an expression.
7.	$\text{stddev}(\text{colname})$	Returns the standard deviation for an expression.
8.	$\text{variance}(\text{colname})$	Returns the variance for an expression.

## DBMS LAB

Q.1. List name, salary and pf amount of all the employees. Pf is calculated as 10% of salary.

SQL >

Select ename, round (sal \* 1), round (sal \* .1) as "PF"  
from EMP;

OUTPUT:

ENAME	SAL	PF
SMITH	800	80
ALLEN	1600	160
WARD	1250	125
JONES	2975	297.5
MARTIN	1250	125
BLAKE	2850	285
CLARK	2450	245
SCOTT	3000	300
KING	5000	500
TURNER	1500	150
ADAMS	1100	110
JAMES	950	95
FORD	3000	300
MILLER	1300	130

14 rows selected.

Q.2 List the number of employees working in the company.

SQL >

Select count (ename) as "NOOFEML" from EMP;

OUTPUT:

NOOFEML

14

Q.3 List the number of jobs available in the company.

SQL >

Select count (distinct job) as "NOOFTJOBS" from EMP;

OUTPUT:

NOOFTJOBS

5

## DBMS LAB

Q.4 List the total salary payable to employees.

SQL >

select round (sum (sal)) as "TOTALSAL" from EMP;

OUTPUT:

TOTALSAL

-----  
29025

Q.5 List the minimum, maximum and average salary payable to employees.

SQL >

select round (min (sal)) as "MINSAL", round (max (sal)) as "MAXSAL", round (avg (sal), 2) as "AVGSAL" from EMP;

OUTPUT:

MINSAL

MAXSAL

AVGSAL

-----  
800 5000 2073.21429

Q.6 List the maximum salary and number of employees working as "SALESMAN".

SQL > select max(sal) as "MAXSAL", count(\*) as "NOOFEMP"  
from emp where job = 'SALESMAN';

OUTPUT:

MAXSAL

NOOFEMP

-----  
1600 4

Q.7 List the average salary and number of employees working on department number 20.

SQL > select round (avg (sal)) as "AVGSAL", count(\*) as "NOOFEMP"  
from emp where deptno = 20;

OUTPUT:

AVGSAL

NOOFEMP

-----  
2175 5

## DBMS LAB

Q.8 Display 10% increased salary of all the employees.

SQL >

select round((sal/10 + sal)) as "INCSAL" from emp;

OUTPUT:  
INCSAL

-----  
880  
1760  
1375  
3272.5  
1375  
3135  
2695  
3300  
5500  
1650  
1210  
1045  
3300  
1430

14 rows selected.

Q.9 Display the maximum salary paid to "CLERK".

SQL >

select max(sal) from emp where job = 'CLERK';

OUTPUT:  
MAXSAL

-----  
1300

Q.10 List the department numbers and number of employees in each department.

SQL > select deptno, count(\*) as "NOOFEML" from emp  
group by deptno Order by deptno desc;

OUTPUT:  
DEPTNO NOOFEML

-----  
30 6  
20 5  
10 3

## DBMS LAB

Q.11 List the department number and total salary payable to each department.

SQL> select deptno, sum(sal) as "TOTALSAL", avg(sal) as "AVGSAL", min(sal) from emp group by deptno order by deptno desc;

OUTPUT:

DEPTNO	TOTALSAL
30	9400
20	10875
10	8750

Q.12 List the jobs and the number of employees in each job. The result should be in the descending order of the number of jobs.

SQL> select job, count(\*) as "NUMOFEEMP" from emp group by job order by count(\*) desc;

OUTPUT:

JOB	NUMOFEEMP
CLERK	4
SALESMAN	4
MANAGER	3
ANALYST	2
PRESIDENT	1

Q.13 List the job wise total salary, average salary and minimum salary of employees.

SQL> select job, sum(sal) as "TOTAL SAL", avg(sal) as "AVGSAL", min(sal) as "MINSAL" from emp group by job;

OUTPUT:

JOB	TOTALSAL	AVGSAL	MINSAL
CLERK	4150	1037.5	800
SALESMAN	5600	1400	1250
PRESIDENT	5000	5000	5000
MANAGER	8275	2758.33333	2450
ANALYST	6000	3000	3000

## DBMS LAB

Q.14 List the total salary of employees, job wise for department 20 only.

SQL> *Select job, sum(sal) as "TOTALSAL" from emp where deptno = 20 group by job;*

OUTPUT:

JOB	TOTALSAL
CLERK	1900
MANAGER	2975
ANALYST	6000

Q.15 Find out maximum salaries department wise excluding those who are having salary less than 3000.

SQL> *select deptno, max(max(sal)) as "MAXSAL" from emp where sal < 3000 group by dept no;*

OUTPUT:

DEPTNO	MAXSAL
30	2850
20	2975
10	2450

Q.16 List the job wise total salary, average salary of employees of department number 20 and display only those rows having average salary greater than 1000.

SQL> *Select job, sum(sal) as "TOTALSAL", avg(sal) as "AVGSAL" from emp where deptno = 20 group by job having avg(sal) > 1000;*

OUTPUT:

JOB	TOTALSAL	AVGSAL
MANAGER	2975	2975
ANALYST	6000	3000

Q.17 Display the department numbers with more than three employees in each department.

SQL> *Select deptno, count(\*) as "NOOFEML" from emp group by deptno having count(\*) > 3;*

OUTPUT:

DEPTNO	NOOFEML
30	6
20	5

## DBMS LAB

Q.18 Display the various jobs along with total salary for each of the jobs where total salary is greater than 5000.

SQL> select job, round(sum(sal)) as "TOTALSAL" from emp  
group by job having sum(sal) > 5000;

OUTPUT:

JOB	TOTALSAL
SALESMAN	5600
MANAGER	8275
ANALYST	6000

Q.19 Display the various jobs along with total number of employees in each job. The output should contain only those jobs with more than three employees.

SQL> select job, count(\*) as "NOOFEMP" from emp group by job having count(\*) > 3;

OUTPUT:

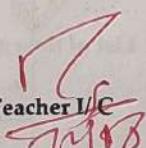
JOB	NOOFEMP
CLERK	4
SALESMAN	4

Q.20 Multiply all the department number from department table.

SQL> select round(exp(sum(ln(deptno)))) as "MULTIPLY" from dep2;

OUTPUT:

MULTIPLY  
-----  
240000

  
Teacher I/C  
Mr. Raj Kumar Singh

## DBMS LAB

### Lab 3 String and Date Functions

#### 3. String Functions

SN	Function	Description
1.	concat(s1, s2)	Concatenates string s1 with s2.
2.	length(s)	Finds the length of the string s.
3.	upper(s)	Converts into upper case.
4.	lower(s)	Converts into lower case.
5.	initcap(s)	Converts first letter into upper case and rest into lower case.
6.	replace(s, 'm', 'n')	Replaces every sequence of characters into another sequence.
7.	translate(s, 'm', 'n')	Replaces every sequence of characters into another sequence.
8.	ltrim(s, 'm')	Removes all specified characters from left hand side of string.
9.	rtrim(s, 'm')	Removes all specified characters from right hand side of string.
10.	trim(leading trailing both 'm' from s)	Removes all specified characters either from leading or trailing.
11.	lpad(s, len, 'm')	Pads the left side of the string a specified set of characters.
12.	rpad(s, len, 'm')	Pads the right side of the string a specified set of characters.
13.	soundex(s)	Returns a phonetic representation of a string.
14.	ascii(s)	Returns the ascii value of given character.
15.	chr(num)	Returns the character based on ascii value specified.
16.	instr(s, 'm', spos, occur)	Returns the location of substring m in string s.
17.	substr(s, spos, len)	Extracts a substring from a string.
18.	reverse(s)	Returns the reverse of the string.

#### 4. Date Functions

SN	Function	Description
1.	Sysdate	Returns the current date and time for the local database.
2.	current_date	Returns the current date in the time zone of the current SQL session.
3.	last_day(date)	Returns the last day of the months based on the date specified.
4.	next_day(date, weekday)	Returns the first weekday that is greater than a date.
5.	months_between(date1, date2)	Returns the number of months between date1 and date2.
6.	add_months(date, num)	Returns a date plus num months.
7.	to_date(date, 'fmt')	Converts a number or date to a string.
8.	to_char(date, 'fmt')	Converts a string to a date.
9.	new_time(date, 'tz1', 'tz2')	Returns a date in time zone tz1 to time zone tz2.

List of time zones for new\_time function

SN	Value	Description
1.	AST	Atlantic Standard Time
2.	ADT	Atlantic Daylight Time
3.	BST	Bering Standard Time
4.	BDT	Bering Daylight Time

## DBMS LAB

---

5.	CST	Central Standard Time
6.	CDT	Central Daylight Time
7.	EST	Eastern Standard Time
8.	EDT	Eastern Daylight Time
9.	GMT	Greenwich Mean Time
10.	HST	Alaska - Hawaii Standard Time
11.	HDT	Alaska - Hawaii Daylight Time
12.	MST	Mountain Standard Time
13.	MDT	Mountain Daylight Time
14.	NST	Newfoundland Standard Time
15.	PST	Pacific Standard Time
16.	PDT	Pacific Daylight Time
17.	YST	Yukon Standard Time
18.	YDT	Yukon Daylight Time

### Soundex algorithm

1. Capitalize all letters in the word and drop all punctuation marks. Pad the word with rightmost blanks as needed during each procedure step.
2. Retain the first letter of the word.
3. Change all occurrence of the following letters to '0' (zero):  
'A', 'E', 'I', 'O', 'U', 'H', 'W', 'Y'.
4. Change letters from the following sets into the digit given:  
1 = 'B', 'F', 'P', 'V'  
2 = 'C', 'G', 'J', 'K', 'Q', 'S', 'X', 'Z'  
3 = 'D', 'T'  
4 = 'L'  
5 = 'M', 'N'  
6 = 'R'
5. Remove all pairs of digits which occur beside each other from the string that resulted after step (4).
6. Remove all zeros from the string that results from step 5.0 (placed there in step 3)
7. Pad the string that resulted from step (6) with trailing zeros and return only the first four positions, which will be of the form <uppercase letter><digit><digit><digit>.

### List of valid parameters for to\_char function

Parameter	Explanation
YEAR	Year, spelled out.
YYYY	4-digit year.
YY	Last 3, 2, or 1 digit(s) of year.
Y	
IYY	Last 3, 2, or 1 digit(s) of ISO year.
IY	
I	
YYYY	4-digit year based on the ISO standard.
Q	Quarter of year (1, 2, 3, 4; JAN-MAR = 1).
MM	Month (01-12; JAN = 01).
MON	Abbreviated name of month.
MONTH	Name of month, padded with blanks to length of 9 characters.
RM	Roman numeral month (I-XII; JAN = I).

## DBMS LAB

WW	Week of year (1-53) where week 1 starts on the first day of the year and continues to the seventh day of the year.
W	Week of month (1-5) where week 1 starts on the first day of the month and ends on the seventh.
IW	Week of year (1-52 or 1-53) based on the ISO standard.
D	Day of week (1-7).
DAY	Name of day.
DD	Day of month (1-31).
DDD	Day of year (1-366).
DY	Abbreviated name of day.
J	Julian day; the number of days since January 1, 4712 BC.
HH	Hour of day (1-12).
HH12	Hour of day (1-12).
HH24	Hour of day (0-23).
MI	Minute (0-59).
SS	Second (0-59).
SSSS	Seconds past midnight (0-86399).
FF	Fractional seconds.

Q.1 Display the names of all employees in upper case, lower case and proper case.

SQL >

```
Select UPPER(ename) as "UP", LOWER(ename) as
"LOW", INITCAP(ename) as "PROPER" from emp;
```

OUTPUT:

	UP	LOW	PROPER
SMITH	smithSmith		
ALLEN	allen	Allen	
WARD	wardWard		
JONES	jonesJones		
MARTIN	martinMartin		
BLAKE	blake	Blake	
CLARK	clark	Clark	
SCOTT	scott	Scott	
KING	kingKing		
TURNER	turnerTurner		
ADAMS	adams	Adams	
JAMES	james	James	
FORD	fordFord		
MILLER	millerMiller		

14 rows selected.

Q.2 Display the names along with length of names of all employees.

SQL >

```
Select ename, LENGTH(ename) as "LENAME"
from emp;
```

## DBMS LAB

### OUTPUT:

ENAME	LENAME
SMITH	5
ALLEN	5
WARD	4
JONES	5
MARTIN	6
BLAKE	5
CLARK	5
SCOTT	5
KING	4
TURNER	6
ADAMS	5
JAMES	5
FORD	4
MILLER	6

14 rows selected.

Q.3 Display your name along with length of name.

```
SQL> select 'siddharth' as "SIDDHARTH", LENGTH('siddharth')
      as "LENGTH ('SIDDHARTH')";
```

### SAMPLE OUTPUT :

'RAJKUMAR LENGTH ('RAJKUMAR')
-----
rajkumar
8

Q.4 Display the name of employees concatenated with employee numbers as '7369-SMITH'.

```
SQL> select concat(empno, ' - ', ename) as "empno-ename"
      from emp;
```

### OUTPUT:

empno-ename
7369-SMITH
7499-ALLEN
7521-WARD
7566-JONES
7654-MARTIN
7698-BLAKE
7782-CLARK
7788-SCOTT
7839-KING
7844-TURNER
7876-ADAMS
7900-JAMES
7902-FORD
7934-MILLER

## DBMS LAB

14 rows selected.

Q.5 Use appropriate functions & extract 3 characters starting from 2nd character for the string 'ORACLE'.

SQL>

Select TRIM ('OLE') from 'ORACLE' as "Ext";

OUTPUT:

EXT  
---  
RAC

Q.6 Find the first occurrence of character 'a' from the string 'Computer Maintenance Corporation'.

SQL>

Select POSITION ('a' IN 'Computer Maintenance Corporation') as "FIRSTOCC";

OUTPUT:

FIRSTO

11

Q.7 Replace every occurrence of 'A' with 'B' in the string 'ALLENS'.

SQL>

Select REPLACE ('ALLENS', 'A', 'B') as "REPLAC";

OUTPUT:

REPLAC

BLENS

Q.8 Display the name of employees, job title and department numbers of all employees where job is 'MANAGER' it should be displayed as 'BOSS'.

SQL> Select ename as "REPLACEBOSS", REPLACE(job, 'MANAGER', 'BOSS') as "JOB", deptno as "DEPTNO" from emp;

OUTPUT:

REPLACEBOSS

SMITH  
ALLEN  
WARD  
JONES

JOB	DEPTNO
CLERK	20
SALESMAN	30
SALESMAN	30
BOSS20	

## DBMS LAB

MARTIN  
BLAKE  
CLARK  
SCOTT  
KING  
TURNER  
ADAMS  
JAMES  
FORD  
MILLER

SALESMAN 30  
BOSS30  
BOSS10  
ANALYST 20  
PRESIDENT 10  
SALESMAN 30  
CLERK 20  
CLERK 30  
ANALYST 20  
CLERK 10

14 rows selected.

Q.9 Display the first and second occurrence of 'C' in 'CHICAGO'.

SQL> Select POSITION ('C' IN 'CHICAGO') AS FIRSTOCC,  
POSITION ('C' IN 'SUBSTRING('CHICAGO', POSITION('C' IN 'CHICAGO')+1))  
+ POSITION ('C' IN 'KA(CAGO') AS SECONDOCC;

OUTPUT:

FIRSTOCC	SECONDOCC
1	4

Q.10 Display the substring from 5th character from string 'BHILAI INSTITUTE OF TECHNOLOGY'.

SQL> select TRIM ('BHIL' from 'BHILAI INSTITUTE OF  
TECHNOLOGY') AS "SUBS";

OUTPUT:

SUBS
AI INSTITUTE OF TECHNOLOGY

Q.11 Display the name of all employees right align 8 characters.

SQL> select LPAD (ename, 8, ' ') AS "RIGHTAL"  
from emp;

OUTPUT:  
RIGHTAL

SMITH
ALLEN
WARD
JONES
MARTIN
BLAKE
CLARK
SCOTT

## DBMS LAB

```
KING  
TURNER  
ADAMS  
JAMES  
FORD  
MILLER
```

14 rows selected.

Q.12 Display the names of employees who have joined the organization on Monday.

```
SQL> select ENAME from emp where TRIM(TO_CHAR(hiredate,  
'DAY')) = 'MONDAY'
```

OUTPUT:  
ENAME  
-----  
MARTIN

Q.13 Display your age in days.

```
SQL> select '2023-08-29' :: DATE - '2003-06-13' :: DATE  
as "AGE IN DAYS";
```

SAMPLE OUTPUT : (IF THE DOB IS 06-JUN-1983)  
AGEINDAYS  
-----  
13203

Q.14 Display your age in months.

```
SQL> select ('2023-08-29' :: DATE - '2003-06-13' :: DATE)/30  
as "AGE IN MONTHS";
```

SAMPLE OUTPUT : (IF THE DOB IS 06-JUN-1983)  
AGEINMON  
-----  
434

Q.15 Display the current day as '17th JULY TWO THOUSAND TWELVE'.

```
SQL> select TO_CHAR(CURRENT_DATE, 'DDTH MONTH YEAR');  
Output:  
NOTE : Not implemented in POSTGRESQL 12TH SEPTEMBER
```

## DBMS LAB

### OUTPUT:

```
TO_CHAR(SYSDATE, 'DDTHMONYEAR')
```

```
30th jul twenty nineteen
```

Q.16 Display the following output for each row from emp table.

'Scott has joined the company on Wednesday 9 Dec Nineteen Eighty Two'

```
SQL> select ename || ' Has Joined The Company On ' ||  
      TO_CHAR(HIREDATE, 'DD MON YYYY') as "OUTP" from  
      emp;
```

### OUTPUT:

```
OUTP
```

Smith Has Joined The Company On Wednesday	17 Dec Nineteen Eighty
Allen Has Joined The Company On Friday	20 Feb Nineteen Eighty-One
Ward Has Joined The Company On Sunday	22 Feb Nineteen Eighty-One
Jones Has Joined The Company On Thursday	02 Apr Nineteen Eighty-One
Martin Has Joined The Company On Monday	28 Sep Nineteen Eighty-One
Blake Has Joined The Company On Friday	01 May Nineteen Eighty-One
Clark Has Joined The Company On Tuesday	09 Jun Nineteen Eighty-One
Scott Has Joined The Company On Thursday	09 Dec Nineteen Eighty-Two
King Has Joined The Company On Tuesday	17 Nov Nineteen Eighty-One
Turner Has Joined The Company On Tuesday	08 Sep Nineteen Eighty-One
Adams Has Joined The Company On Wednesday	12 Jan Nineteen Eighty-Three
James Has Joined The Company On Thursday	03 Dec Nineteen Eighty-One
Ford Has Joined The Company On Thursday	03 Dec Nineteen Eighty-One
Miller Has Joined The Company On Saturday	23 Jan Nineteen Eighty-Two

14 rows selected.

Q.17 Find the date for nearest Saturday after current date.

```
SQL>
```

### SAMPLE OUTPUT:

```
CURRDATE  NEXTSAT
```

```
30-JUL-19 03-AUG-19
```

Q.18 Display current date and time.

```
SQL> select TO_CHAR(CURRENT_DATE, 'DD-MON-YYYY fmHH:MI:SS')  
      as "CURDT";
```

### SAMPLE OUTPUT:

```
CURDT
```

```
30-jul-2019 12:12:43
```

## DBMS LAB

Q.19 Display the date 3 months before the current date.

SQL >

```
Select CURRENT_DATE as "CURRDATE", CURRENT_DATE - 90  
as "BEF3MDN";
```

SAMPLE OUTPUT:

CURRDATE BEF3MDN.

-----  
30-JUL-19 30-APR-19

Q.20 Display the names of employees who are more than 38 years old in the organization.

SQL >

```
Select ename from emp where ((CURRENT_DATE - HIREDATE)  
/ (CURRENT_DATE - HIREDATE) / 365) > 40;
```

OUTPUT:

ENAME

-----  
SMITH  
ALLEN  
WARD  
JONES  
BLAKE  
CLARK

6 rows selected.

Q.21 Display the quarter of the year in which employees have joined.

SQL >

```
Select EXTRACT(quarter from HIREDATE) from emp;
```

OUTPUT:

Q  
-  
4  
1  
1  
2  
3  
2  
2  
4  
3  
1  
4  
4

14 rows selected.

## DBMS LAB

Q.22 Display the names of the employees who length of the name contains more than 4 characters.

SQL >

select ename from emp where length(ename) > 4;

OUTPUT :

ENAME

-----  
SMITH  
ALLEN  
JONES  
MARTIN  
BLAKE  
CLARK  
SCOTT  
TURNER  
ADAMS  
JAMES  
MILLER

11 rows selected.

Q.23 Display the name of those employees whose name contains 'A'. [Use instr function].

SQL >

select ename from emp where instr(ename, 'A') != 0;

OUTPUT :

ENAME

-----  
ALLEN  
WARD  
MARTIN  
BLAKE  
CLARK  
ADAMS  
JAMES

7 rows selected.

## DBMS LAB

Q.24 Display the half of the employee names in upper case and remaining in lower case.

SQL> *Select upper (substr(ename, 1, length(ename)/2)) || lower (substr(ename, length(ename)/2 )) as "EMPNAME" from emp*

OUTPUT:  
EMPNAME

SMith  
Allen  
WArd  
JOnes  
MARTin  
BLake  
CLark  
SCott  
KIng  
TURner  
ADams  
JAMES  
FOrd  
MILLer

14 rows selected.

Q.25 Display the name of those employees whose name starts with letter 'A'. [Use instr function].

SQL> *Select ename from emp where instr(ename, 'A')=1;*

OUTPUT:  
ENAME

-----  
ALLEN  
ADAMS

*RK*  
Teacher I/C

Mr. Raj Kumar Singh

# DBMS LAB

## Lab - 4 Multi Table Queries

### Topics

1. Set Operations (Union, intersection, set difference)
2. Types of Joins
  - (a) Cross Join (Cartesian Product)
  - (b) Inner Join (Equi and Non Equi)
  - (c) Natural Join
  - (d) Outer Join (Left, Right and Full)
  - (e) Self Join

Q.1. List employee number, name, his department and the department name.

SQL> select \* from emp  
natural join dept;

OUTPUT:

EMPNO	ENAME	DEPTNO	DNAME
7369	SMITH	20	RESEARCH
7499	ALLEN	30	SALES
7521	WARD	30	SALES
7566	JONES	20	RESEARCH
7654	MARTIN	30	SALES
7698	BLAKE	30	SALES
7782	CLARK	10	ACCOUNTING
7788	SCOTT	20	RESEARCH
7839	KING	10	ACCOUNTING
7844	TURNER	30	SALES
7876	ADAMS	20	RESEARCH
7900	JAMES	30	SALES
7902	FORD	20	RESEARCH
7934	MILLER	10	ACCOUNTING

14 rows selected.

Q.2 List employee name, his department name and the department location.

SQL> select ename, dname, loc from emp natural join dept;

OUTPUT:

ENAME	DNAME	LOC
SMITH	RESEARCH	DALLAS
ALLEN	SALES	CHICAGO
WARD	SALES	CHICAGO
JONES	RESEARCH	DALLAS
MARTIN	SALES	CHICAGO
BLAKE	SALES	CHICAGO
CLARK	ACCOUNTING	NEW YORK
SCOTT	RESEARCH	DALLAS

## DBMS LAB

KING	ACCOUNTING	NEW YORK
TURNER	SALES	CHICAGO
ADAMS	RESEARCH	DALLAS
JAMES	SALES	CHICAGO
FORD	RESEARCH	DALLAS
MILLER	ACCOUNTING	NEW YORK

14 rows selected.

Q.3 List employee name, department name for all the clerks in the company.

SQL>

Select ename, dname from emp NATURAL JOIN dept  
where job = 'CLERK';

OUTPUT:

ENAME	DNAME
MILLER	ACCOUNTING
ADAMS	RESEARCH
SMITH	RESEARCH
JAMES	SALES

Q.4 List employee number, name, job, his manager's name and manager's job.

SQL>

select e1.empno as "EMPLNO", e1.ename as 'EMPLNAME',  
e1.job as 'EMPJOB', e2.fname as 'MGRNAME', e2.job  
as 'MGRJOB' from emp e1, emp e2 where e1.mgr=e2.empno;

OUTPUT:

EMPLNO	EMPLNAME	EMPLJOB	MGRNAME	MGRJOB
7902	FORD	ANALYST	JONES	MANAGER
7788	SCOTT	ANALYST	JONES	MANAGER
7900	JAMES	CLERK	BLAKE	MANAGER
7844	TURNER	SALESMAN	BLAKE	MANAGER
7654	MARTIN	SALESMAN	BLAKE	MANAGER
7521	WARD	SALESMAN	BLAKE	MANAGER
7499	ALLEN	SALESMAN	BLAKE	MANAGER
7934	MILLER	CLERK	CLARK	MANAGER
7876	ADAMS	CLERK	SCOTT	ANALYST
7782	CLARK	MANAGER	KING	PRESIDENT
7698	BLAKE	MANAGER	KING	PRESIDENT
7566	JONES	MANAGER	KING	PRESIDENT
7369	SMITH	CLERK	FORD	ANALYST

13 rows selected.

Q.5 List the jobs common to department 20 and 30.

SQL> select distinct e1.job from emp e1, emp e2 where  
(e1.job = e2.job) and e1.deptno = 20 and e2.deptno = 30;

## DBMS LAB

OUTPUT:

JOB

CLERK

MANAGER

Q.6 List the jobs unique to department 20.

SQL> select job from emp EXCEPT (select job from emp where deptno != 20);

OUTPUT:

JOB

ANALYST

Q.7 List the employees belonging to the department of 'MILLER'.

SQL> select e1.ename from emp e1, emp e2 where  
e1.deptno = e2.deptno and e2.ename  
= 'MILLER';

OUTPUT:

EMPLNAME

CLARK

KING

MILLER

Q.8 List all the employees who have the same job as 'SCOTT'.

SQL> select e1.ename from emp e1, emp e2 where  
e1.job = e2.job and e2.ename = 'SCOTT';

OUTPUT:

EMPLNAME

SCOTT

FORD

Q.9 Display the names of the employees who are working in sales or research department.

SQL> select distinct e1.ename from emp e1 natural  
join dept d1, where d1.dname in ('SALES',  
'RESEARCH');

## DBMS LAB

**OUTPUT:**

ENAME

-----  
SMITH  
ALLEN  
WARD  
JONES  
MARTIN  
BLAKE  
SCOTT  
TURNER  
ADAMS  
JAMES  
FORD

11 rows selected.

Q.10 Display name and salary of the employees who is working in 'CHICAGO'.

SQL&gt;

select distinct e1.ename, e1.sal from emp e1 natural join dept d1 where d1.loc = 'CHICAGO';

**OUTPUT:**

ENAME	SAL
ALLEN	1600
WARD	1250
MARTIN	1250
BLAKE	2850
TURNER	1500
JAMES	950

6 rows selected.

Q.11 List the details of employees in department 10 who have the same job as in department 30.

SQL&gt;

select e1.empno, e1.ename, e1.job = e2.job  
and e2.deptno = 30 and e1.deptno = 10;

**OUTPUT:**

DEPTNO	EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM
10	7782	CLARK	MANAGER	7839	09-JUN-81	2450	
10	7934	MILLER	CLERK	7782	23-JAN-82	1300	

## DBMS LAB

Q.12 List the employee name, length of his name, his manager's name whose name length is greater than their managers name length.

SQL> *Select e1.ename as 'EMPLNAME', length (e1.ename) as 'LEMPL', e2.ename as 'MGRNAME' from emp e1, emp e2 where e1.mgr = e2.empno and length (e1.ename) > length (e2.ename);*

OUTPUT:

EMPLNAME	LEMPL	MGRNAME
TURNER	6	BLAKE
MARTIN	6	BLAKE
MILLER	6	CLARK
CLARK	5	KING
BLAKE	5	KING
JONES	5	KING
SMITH	5	FORD

7 rows selected.

Q.13 List employees and his manager's details, where that employee's salary is greater than his manager's salary.

SQL> *Select e1.empno, e1.ename, e2.ename, e2.job, e2.mgr, e2.hiredate, e2.sal, e2.comm, e2.deptno from emp e1, emp e2 where e1.mgr = e2.empno and e1.sal > e2.sal;*

OUTPUT:

EMPNAME	EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
FORD	7566	JONES	MANAGER		7839 02-APR-81	2975		20
SCOTT	7566	JONES	MANAGER		7839 02-APR-81	2975		20

Q.14 List those employee names whose manager name is 'JONES'.

SQL> *Select e1.ename from emp e1, emp e2 where e1.mgr = e2.empno and e2.ename = 'JONES';*

OUTPUT:

EMPNAME
SCOTT
FORD

Q.15 Display employee name, department name, salary and commission for those employees whose salary is between 2000 and 5000 while the department location is 'CHICAGO'.

SQL> *Select ename, dname, sal, comm from emp natural join dept where sal between 2000 and 5000 and loc='CHICAGO';*

## DBMS LAB

**OUTPUT:**

ENAME	DNAME	SAL	COMM
BLAKE	SALES	2850	

Q.16 Display those employees who are working in the same department where his manager is work.

SQL>

```
select e1.ename as 'EMPNAME' from emp e1, emp e2  
where e1.mgr = e2.empno and e1.deptno  
= e2.deptno;
```

**OUTPUT:**

EMPNAME

FORD  
SCOTT  
JAMES  
TURNER  
MARTIN  
WARD  
ALLEN  
MILLER  
ADAMS  
CLARK  
SMITH

11 rows selected.

Q.17 Display those employee names who joined the company before '31-Dec-82' while the department location is 'NEWYORK' or 'CHICAGO'.

SQL>

```
select ename from emp natural join dept  
where hiredate < '1982-12-31' and loc IN ('NEWYORK',  
CHICAGO);
```

**OUTPUT:**

ENAME

ALLEN  
WARD  
MARTIN  
BLAKE  
TURNER  
JAMES

6 rows selected.

Q.18 Display the employee name, job and his managers. Display also the employees who are without manager.

SQL> select e1.name, e1.job, e2.name, as mgr from emp e1, emp e2  
where e1.mgr = e2.empno union select e.ename, e.job, \*

## DBMS LAB

`' ' as mgr from emp e where e.mgr is null;`

OUTPUT:

EMPNAME	EMPJOB	MGRNAME
FORD	ANALYST	JONES
SCOTT	ANALYST	JONES
JAMES	CLERK	BLAKE
TURNER	SALESMAN	BLAKE
MARTIN	SALESMAN	BLAKE
WARD	SALESMAN	BLAKE
ALLEN	SALESMAN	BLAKE
MILLER	CLERK	CLARK
ADAMS	CLERK	SCOTT
CLARK	MANAGER	KING
BLAKE	MANAGER	KING
JONES	MANAGER	KING
SMITH	CLERK	FORD
KING	PRESIDENT	

14 rows selected.

Q.19 Display the name of the department where no employee is working.

SQL> `select dname from dept except (select distinct dname from emp natural join dept);`

OUTPUT:

DNAME

OPERATIONS

Q.20 Display the details of all the employees who are sub - ordinate to 'BLAKE'.

SQL> `select e1.empno, e1.ename, e1.job, e1.mgr, e1.hiredate, e1.sal, e1.comm, e1.deptno from emp e1, emp e2 where e1.mgr = e2.empno and e2.ename = 'BLAKE';`

OUTPUT:

DEPTNO	EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM
30	7499	ALLEN	SALESMAN	7698	20-FEB-81	1600	300
30	7521	WARD	SALESMAN	7698	22-FEB-81	1250	500
30	7654	MARTIN	SALESMAN	7698	28-SEP-81	1250	1400
30	7844	TURNER	SALESMAN	7698	08-SEP-81	1500	0
30	7900	JAMES	CLERK	7698	03-DEC-81	950	

## DBMS LAB

30

Q.21 Display the employee name and department name even if there are no employees working in a particular department.

SQL> select ename, dname from emp e right outer join dept d on  
e.deptno = d.deptno;  
[OR]  
(select '' as ename, dname from dept) union  
(select '' as ename, dname from dept except (select '' as ename,  
dname as d from emp natural join dept));

OUTPUT:

ENAME	DNAME
SMITH	RESEARCH
ALLEN	SALES
WARD	SALES
JONES	RESEARCH
MARTIN	SALES
BLAKE	SALES
CLARK	ACCOUNTING
SCOTT	RESEARCH
KING	ACCOUNTING
TURNER	SALES
ADAMS	RESEARCH
JAMES	SALES
FORD	RESEARCH
MILLER	ACCOUNTING
	OPERATIONS

15 rows selected.

Q.22 Display the department name and total number of employees in each department.

SQL> select dname, count(\*) as 'NOOFEPM' from emp  
natural join dept group by dname;

OUTPUT:

DNAME	NOOFEPM
ACCOUNTING	3
RESEARCH	5
SALES	6

Q.23 Display the department name along with total salary in each department.

SQL> select dname, sum(sal) from emp natural join  
dept group by dname;

OUTPUT:

## DBMS LAB

DNAME	TOTOLSAL
ACCOUNTING	8750
RESEARCH	10875
SALES	9400

Q.24 List the jobs common to department 'RESEARCH' and 'SALES'.

SQL > select job from emp natural join dept where dname = 'RESEARCH' intersect select job from emp natural join dept where dname = 'SALES';

OUTPUT:

JOB

CLERK

MANAGER

Q.25 List the jobs unique to department 'RESEARCH'.

SQL > select job from emp natural join dept where dname = 'RESEARCH' except select job from emp natural join dept where dname != 'RESEARCH';

OUTPUT:

JOB

ANALYST

Teacher I/C

Mr. Raj Kumar Singh  
10/10

## DBMS LAB

### LAB 5 Sub Query, Correlated Query, Top - N Analysis

#### Topics

1. What is Sub Query?
2. Types of Sub Query (Single Row Sub Query and Multi Row Sub Query)
3. What is Correlated Query?
4. Difference between Sub Query and Correlated Query.
5. Semi Join and Anti Join (Exists and Not Exists)
6. Top - N Analysis (Rowid and Rownum)

Q.1. Display the employee number and name of employee working as 'CLERK' and earning highest salary among clerks.

SQL > Select empno, ename from emp where sal = (select max(sal) from emp where job = 'CLERK') and job = 'CLERK';

OUTPUT:

EMPNO ENAME

-----  
7934 MILLER

Q.2 Display the names of 'SALESMAN' who earns a salary more than the highest salary of any 'CLERK'.

SQL > Select ename from emp where sal > (select max(sal) from emp where job = 'CLERK') and job = 'SALESMAN';

OUTPUT:

ENAME

-----  
ALLEN  
TURNER

Q.3 Display the names of clerks who earn a salary more than the lowest salary of any 'SALESMAN'.

SQL > Select ename from emp where sal > (select min(sal) from emp where job = 'SALESMAN') and job = 'CLERK';

OUTPUT:

ENAME

-----  
MILLER

## DBMS LAB

Q.4 Display the names of employees who earn a salary more than that of 'JONES' or that of salary greater than that of 'SCOTT'.

SQL > select ename from emp where sal > \* (select sal from emp where ename = 'JONES') and or  
sal > \* (select sal from emp where ename = 'SCOTT');

OUTPUT:

ENAME

-----  
SCOTT  
KING  
FORD

Q.5 Display the names of employees who earn highest salary in their respective departments.

SQL > select e.ename from emp e where e.sal  
IN (select max(sal) from emp group by  
deptno);

OUTPUT:

EMPNAME

-----  
BLAKE  
SCOTT  
KING  
FORD

Q.6 Display the names of the employees who earn highest salaries in their respective job groups.

SQL > select e.ename from emp e.sal IN (select  
max(sal) from emp group by job);

OUTPUT:

EMPNAME

-----  
ALLEN  
JONES  
SCOTT  
KING  
FORD  
MILLER

6 rows selected.

## DBMS LAB

Q.7 Display the employee names who are working in 'ACCOUNTING' department.

SQL >

```
Select ename from emp where deptno = (select deptno from dept where dname = 'ACCOUNTING')  
order by empno desc;
```

OUTPUT:

ENAME

MILLER  
KING  
CLARK

Q.8 Display the employee names who are working in 'CHICAGO'.

SQL >

```
Select ename from emp where deptno = (select deptno from dept where loc = 'CHICAGO')  
order by empno desc;
```

OUTPUT:

ENAME

JAMES  
TURNER  
BLAKE  
MARTIN  
WARD  
ALLEN

6 rows selected.

Q.9 Display the job groups having total salary greater than the maximum salary for managers.

SQL >

```
Select job from emp group by job having sum(sal) >  
(select max(sal) from emp where job = 'MANAGER');
```

OUTPUT:

JOB

CLERK  
SALESMAN  
PRESIDENT  
MANAGER  
ANALYST

## DBMS LAB

Q.10 Display the names of employees from department number 10 with salary greater than that of any employee working in other department.

SQL>

```
select ename from emp where deptno = 10  
and sal > any (select sal from emp where  
deptno != 10);
```

OUTPUT:

EMPNAME

-----

CLARK

KING

MILLER

Q.11 Display the names of the employees from department number 10 with salary greater than that of all employees working in other department.

SQL>

```
select ename from emp where deptno = 10 and  
sal > all (select sal from emp where deptno != 10);
```

OUTPUT:

EMPNAME

-----

KING

Q.12 Display the name of employee who is getting highest salary in the organization.

SQL>

```
select ename from emp where sal IN (select  
max (sal) from emp);
```

OUTPUT:

ENAME

-----

KING

Q.13 Display the name of employee who is getting second highest salary in the organization.

SQL>

```
select ename from emp where sal = (select  
max (sal) from emp where sal < (select max(sal)  
from emp));
```

OUTPUT:

ENAME

-----

SCOTT

FORD

## DBMS LAB

Q.14 Display the name of employee who is getting fourth highest salary in the organization.

SQL > *Select ename as 'empname' from emp e where  
4 = (select count (distinct sal) from emp where  
sal > e.sal);*

OUTPUT:  
EMPNAME

BLAKE

Q.15 Display first 5 rows from emp table.

SQL > *Select \* from emp where limit 5;*

OUTPUT:

	EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM
	DEPTNO						
20	7369	SMITH	CLERK	7902	17-DEC-80	800	
30	7499	ALLEN	SALESMAN	7698	20-FEB-81	1600	300
30	7521	WARD	SALESMAN	7698	22-FEB-81	1250	500
20	7566	JONES	MANAGER	7839	02-APR-81	2975	
30	7654	MARTIN	SALESMAN	7698	28-SEP-81	1250	1400

Q.16 Display last 5 rows from emp table.

SQL > *Select \* from emp offset 9 limit 5;*

OUTPUT:

	EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM
	DEPTNO						
30	7844	TURNER	SALESMAN	7698	08-SEP-81	1500	0
20	7876	ADAMS	CLERK	7788	12-JAN-83	1100	
30	7900	JAMES	CLERK	7698	03-DEC-81	950	
20	7902	FORD	ANALYST	7566	03-DEC-81	3000	

## DBMS LAB

10	7934 MILLER	CLERK	7782 23-JAN-82	1300
----	-------------	-------	----------------	------

Q.17 Display 3rd to 7th rows from emp table.

SQL >

select \* from emp offset 2 limit (7-2);

OUTPUT:

	EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM
	DEPTNO						
30.	7521	WARD	SALESMAN	7698	22-FEB-81	1250	500
20	7566	JONES	MANAGER	7839	02-APR-81	2975	
30	7654	MARTIN	SALESMAN	7698	28-SEP-81	1250	1400
30	7698	BLAKE	MANAGER	7839	01-MAY-81	2850	
10	7782	CLARK	MANAGER	7839	09-JUN-81	2450	

Q.18 Display even rows from emp table.

SQL >

select \* from emp where (rowid, 0) in  
 (select rowid, mod (rownum, 2) from emp);

OUTPUT:

	EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM
	DEPTNO						
30.	7499	ALLEN	SALESMAN	7698	20-FEB-81	1600	300
20	7566	JONES	MANAGER	7839	02-APR-81	2975	
30	7698	BLAKE	MANAGER	7839	01-MAY-81	2850	
20	7788	SCOTT	ANALYST	7566	09-DEC-82	3000	
30	7844	TURNER	SALESMAN	7698	08-SEP-81	1500	0
30	7900	JAMES	CLERK	7698	03-DEC-81	950	
10	7934	MILLER	CLERK	7782	23-JAN-82	1300	

7 rows selected.

## DBMS LAB

Q.19 Display odd rows from emp table.

SQL >

```
select * from emp where (rowid, 1) in
(select rowid, mod (rownum, 2) from emp)
```

OUTPUT :

	EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM
	DEPTNO						
20	7369	SMITH	CLERK	7902	17-DEC-80	800	
30	7521	WARD	SALESMAN	7698	22-FEB-81	1250	500
30	7654	MARTIN	SALESMAN	7698	28-SEP-81	1250	1400
10	7782	CLARK	MANAGER	7839	09-JUN-81	2450	
10	7839	KING	PRESIDENT		17-NOV-81	5000	
20	7876	ADAMS	CLERK	7788	12-JAN-83	1100	
20	7902	FORD	ANALYST	7566	03-DEC-81	3000	

7 rows selected.

Q.20 Display every 3rd row from emp table.

SQL >

```
select * from emp where (rowid, 0) in
(select rowid, mod (rownum, 3) from emp)
```

OUTPUT :

	EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM
	DEPTNO						
30	7521	WARD	SALESMAN	7698	22-FEB-81	1250	500
30	7698	BLAKE	MANAGER	7839	01-MAY-81	2850	
10	7839	KING	PRESIDENT		17-NOV-81	5000	
30	7900	JAMES	CLERK	7698	03-DEC-81	950	

Q.21 Display 3rd max salary from all the employees.

SQL > select sal from emp e where 3 = (select count(distinct sal) from emp where sal >= e.sal))

## DBMS LAB

OUTPUT:  
THIRDMAX

2975

Q.22 Display 3rd min salary from all the employees.

SQL> ~~select sal from emp e where 3 = (select count (distinct sal) from emp where sal <= e.sal);~~

OUTPUT:  
THIRDMIN

1100

Q.23 Who was the last employee hired in each department.

SQL> select ename from emp where (deptno , hiredate)  
in (select deptno, max(hiredate) from emp group by  
deptno);

OUTPUT:  
EMPNAME

ADAMS  
JAMES  
MILLER

Q.24 Display all the employees who have the same job as 'SCOTT'.

SQL> ~~select ename as empname from emp e where  
sal > (select avg(sal) from emp where  
deptno = e.deptno group by deptno);~~

OUTPUT:  
EMPNAME

FORD  
SCOTT

Q.25 List the employees who earn more than the average salary in their own department.

SQL> ~~select ename as empname from emp e where  
sal > (select avg(sal) from emp where deptno  
= e.deptno group by deptno);~~

## DBMS LAB

OUTPUT:  
EMPNAME

ALLEN  
JONES  
BLAKE  
SCOTT  
KING  
FORD

6 rows selected.

DEPARTMENT OF IT BIT DURG

Teacher I/C

Mr. Raj Kumar Singh

## DBMS LAB

---

### Lab 6 DDL, DML, DCL, Constraints

Table S

SN	NAME	ST	CITY
S1	Smith	20	London
S2	Jones	10	Paris
S3	Blake	30	Paris
S4	Clark	20	London
S5	Adams	30	Athens

Table J

JN	NAME	CITY
J1	Sorter	Paris
J2	Punch	Rome
J3	Reader	Athens
J4	Console	Athens
J5	Collator	London
J6	Terminal	Oslo
J7	Tape	London

Table P

PN	NAME	COLOR	WEIGHT	CITY
P1	Nut	Red	12	London
P2	Bolt	Green	17	Paris
P3	Screw	Blue	17	Rome
P4	Screw	Red	14	London
P5	Cam	Blue	12	Paris
P6	Cog	Red	19	London

Table SPJ

SN	PN	JN	QTY
S1	P1	J1	200
S1	P1	J4	700
S2	P3	J1	400
S2	P3	J2	200
S2	P3	J3	200
S2	P3	J4	500
S2	P3	J5	600
S2	P3	J6	400
S2	P3	J7	800
S2	P5	J2	100
S3	P3	J1	200
S3	P4	J2	500
S4	P6	J3	300
S4	P6	J7	300
S5	P2	J2	200
S5	P2	J4	100
S5	P5	J5	500
S5	P5	J7	100
S5	P6	J2	200
S5	P1	J4	100

## DBMS LAB

S5	P3	J4	200
S5	P4	J4	800
S5	P5	J4	400
S5	P6	J4	500

Q.1. Get all the data of all jobs.

SQL >

Select \* from j;

OUTPUT:

JNO	JNAME	CITY
J1	Sorter	Paris
J2	Punch	Rome
J3	Reader	Athens
J4	Console	Athens
J5	Collator	London
J6	Terminal	Oslo
J7	Tape	London

7 rows selected.

Q.2. Get all the data on all jobs in London.

SQL >

Select \* from j where city = 'LONDON';

OUTPUT:

JNO	JNAME	CITY
J5	Collator	London
J7	Tape	London

Q.3. Get supplier numbers for suppliers (S table) who supply (SPJ table) job J1, in supplier number order.

SQL >

Select sno from s natural join SPJ where  
jno = 'J1';

OUTPUT:

SNO
S1
S2
S3

## DBMS LAB

Q.4. Get all shipments (SPJ table) where the quantity is in the range 300 to 750 inclusive.

SQL >

select \* from SPJ where QTY between  
300 and 750;

OUTPUT :

SNO	PNO	JNO	QTY
S1	P1	J4	700
S2	P3	J1	400
S2	P3	J4	500
S2	P3	J5	600
S2	P3	J6	400
S3	P4	J2	500
S4	P6	J3	300
S4	P6	J7	300
S5	P5	J5	500
S5	P5	J4	400
S5	P6	J4	500

11 rows selected.

Q.5. For each supplier, tell the total number of parts supplied to some project.

SQL >

select sno, sum(QTY) from SPJ group by  
order by sno;

OUTPUT :

SNO	SUM(QTY)
S1	900
S2	3200
S3	700
S4	600
S5	3100

Q.6. Get a list of all part-color, part-city combinations, with duplicate (color, city) eliminated.

SQL >

select distinct color, city from p;

OUTPUT :

COLOR	CITY
Red	London
Blue	Paris
Green	Paris

## DBMS LAB

Blue Rome

Q.7. Get the total number of jobs supplied by supplier S1.

SQL >

Select count(distinct jno) from SPJ where  
Sno = 'S1';

OUTPUT:

COUNT(DISTINCTJNO)

2

Q.8. Get the total quantity of part P1 supplied by supplier S1.

SQL >

Select sum(QTY) from SPJ where sno = 'S1'  
and pno = 'P1';

OUTPUT:

SUM(QTY)

900

Q.9. For each part being supplied to a job, get the part number, the job number and the corresponding total quantity.

SQL >

Select jno, pno, sum(QTY) from SPJ group  
by jno, pno;

OUTPUT:

JNO	PNO	SUM(QTY)
J4	P1	800
J6	P3	400
J2	P6	200
J4	P5	400
J1	P1	200
J4	P6	500
J1	P3	600
J4	P3	700
J4	P2	100
J5	P3	600
J2	P5	100
J3	P6	300
J4	P4	800
J7	P6	300
J3	P3	200

## DBMS LAB

J7	P3	800
J2	P4	500
J2	P2	200
J2	P3	200
J5	P5	500
J7	P5	100

21 rows selected.

Q.10. Get part numbers for parts supplied to some job in the average quantity of more than 320.

SQL > select pno from spj group by pno showing avg (qty) > 320;

OUTPUT:

PNO
-----
P4
P1
P3
P6
P5

Q.11. Get all shipments where the quantity is non-null.

SQL >

select \* from spj where qty is not null;

OUTPUT:

SNO	PNO	JNO	QTY
S1	P1	J1	200
S1	P1	J4	700
S2	P3	J1	400
S2	P3	J2	200
S2	P3	J3	200
S2	P3	J4	500
S2	P3	J5	600
S2	P3	J6	400
S2	P3	J7	800
S2	P5	J2	100
S2	P5	J2	200
S3	P3	J1	200
S3	P4	J2	500
S3	P4	J2	300
S4	P6	J3	300
S4	P6	J7	300
S5	P2	J2	200
S5	P2	J4	100
S5	P5	J5	500

## DBMS LAB

```
S5    P5    J7      100  
S5    P6    J2      200  
S5    P1    J4      100  
S5    P3    J4      200  
S5    P4    J4      800  
S5    P5    J4      400  
S5    P6    J4      500
```

24 rows selected.

Q.12. Get project numbers and cities where the city has an "o" as the second letter of its name.

SQL >

select jno, city from j where city like '\_ - o %';

OUTPUT:

JNO	CITY
-----	------

```
-----  
J2    Rome  
J5    London  
J7    London
```

Q.13. Get supplier names which start with the letters 'Sm'.

SQL >

select sname from s where sname like 'sm %';

OUTPUT:

SNAME
-------

```
-----  
Smith
```

Q.14. Get supplier names that have a letter 'e' somewhere in their name.

SQL >

select sname from s where name like '% e %';

OUTPUT:

SNAME
-------

```
-----  
Jones  
Blake
```

## DBMS LAB

Q.15. Get the part number and total shipment quantity for each part.

SQL >

Select pno, sum(QTY) from SPJ group by pno;

OUTPUT:

PNO	SUM(QTY)
P4	1300
P1	1000
P2	300
P3	3500
P6	1300
P5	1100

6 rows selected.

Q.16. Get the last five shipments.

SQL >

Select \* from SPJ minus  
Select \* from SPJ where rownum <= Count(\*) - 5  
from SPJ;

OUTPUT:

SNO	PNO	JNO	QTY
S5	P6	J4	500
S5	P5	J4	400
S5	P4	J4	800
S5	P3	J4	200
S5	P1	J4	100

Q.17. Get the first five shipments.

SQL >

Select \* from SPJ where rownum <= 5;

OUTPUT:

SNO	PNO	JNO	QTY
S1	P1	J1	200
S1	P1	J4	700
S2	P3	J1	400
S2	P3	J2	200
S2	P3	J3	200

## DBMS LAB

Q.18. Get part numbers for all parts supplied by more than two supplier

SQL >

Select pno from SPJ group by pno having count (sno) > 2;

OUTPUT :

PNO  
-----  
P1  
P3  
P6  
P5

Q.19. Get part numbers for parts supplied to some project in Paris.

SQL >

Select distinct pno from SPJ natural join S where city = 'PARIS';

OUTPUT :

PNO  
-----  
P1  
P3

Q.20. Get part numbers for parts that are not supplied to any project in Paris.

SQL >

Select pno from P minus Select distinct pno from SPJ natural join S where city = 'PARIS';

OUTPUT :

PNO  
-----  
P2  
P4  
P5  
P6

Q.21. Get all supplier-number, part-number, job-number triples such that the indicated supplier, part, job are in the same city.

SQL >

Select sno, pno, jno from SPJ natural join S natural join P natural join J;

OUTPUT :

SNO	PNO	JNO
S1	P1	J7

## DBMS LAB

```
S1    P4    J7  
S1    P6    J7  
S1    P1    J5  
S1    P4    J5  
S1    P6    J5  
S2    P2    J1  
S2    P5    J1  
S3    P2    J1  
S3    P5    J1  
S4    P1    J7  
S4    P4    J7  
S4    P6    J7  
S4    P1    J5  
S4    P4    J5  
S4    P6    J5
```

16 rows selected.

Q.22. Get part number for parts supplied (SPJ table) by a supplier in London.

SQL > select distinct pno from SPJ natural join S where city = 'LONDON';

OUTPUT :

```
PNO  
---  
P1  
P6
```

Q.23. Get part numbers for parts supplied by a supplier in London to a job in Paris.

SQL > select pno from SPJ natural join S where city = 'LONDON' intersect select pno from SPJ natural join T where city = 'PARIS';

OUTPUT :

```
PNO  
---  
P1
```

Q.24. Get all pairs of city names such that a supplier in the first city supplies a job in the second city.

SQL > select distinct s.city as FCITY, t.city from S cross join T where (s.sno, t.jno) IN (select sno, jno from SPJ);

OUTPUT :

```
FCITY      SCITY
```

## DBMS LAB

```
Paris      Oslo
Athens    London
Paris      Rome
Paris     Athens
Paris      London
London    London
Athens    Rome
Paris      Paris
Athens    Athens
London    Paris
London    Athens
```

11 rows selected.

Q.25. Get part numbers for parts supplied to any job by a supplier in the same city as the job.

```
SQL> select pno from ppj where (rno, jno) in
      (select r.rno, j.jno from s cross join
       j where s.city = j.city);
```

OUTPUT:

```
PNO
-----
P3
P3
P6
P1
P2
P3
P4
P5
P6
```

9 rows selected.

Q.26. Get job numbers for jobs supplied by at least one supplier not in the same city.

```
SQL> select distinct jno from s cross join j
      where (r.rno, j.jno) in (select rno, jno
      from ppj) and j.city <> s.city;
```

OUTPUT:

```
JNO
-----
J2
J7
J3
```

## DBMS LAB

```
J6  
JS  
J1  
J4
```

7 rows selected.

Q.27. Get all pairs of part numbers such that the same supplier supplies both the indicated parts.

```
SQL> select distinct s1.pno, s2.pno from SPJ s1,  
      SPJ s2 where s1.pno = s2.pno and s1.pno  
      < s2.pno;
```

OUTPUT:

PNO	PNO
P3	P5
P2	P5
P3	P6
P2	P4
P4	P5
P1	P3
P1	P4
P1	P6
P2	P3
P2	P6
P4	P6
P3	P4
P1	P2
P5	P6
P1	P5

15 rows selected.

Q.28. Get job names for jobs supplied by supplier S1.

```
SQL> select jname from j natural join SPJ where  
      pno = 'S1';
```

OUTPUT:

JNAME
Sorter
Console

## DBMS LAB

Q.29. Get colors for parts supplied by supplier S1.

SQL >

Select distinct color from p natural join SPJ where sno = 'S1';

OUTPUT :

COLOR

-----  
Red

Q.30. Get part numbers for parts supplied to any job in London.

SQL >

Select distinct pno from SPJ natural join j where city = 'LONDON';

OUTPUT :

PNO

-----  
P3  
P6  
P5

Q.31. Get job numbers for jobs using at least one part available from supplier S1.

SQL >

Select jno from SPJ where sno = 'S1'  
group by jno having count(\*) >= 1;

OUTPUT :

JNO  
-----  
J1  
J4

Q.32. Get supplier numbers for suppliers supplying at least one part supplied by at least one supplier who supplies at least one red part.

SQL >

Select distinct pno from SPJ where pno  
IN (Select SPJ.pno from SPJ cross join  
p where p.color = 'RED');

OUTPUT :

## DBMS LAB

SNO

-----  
S3  
S4  
S5  
S2  
S1

Q.33. Get supplier numbers for suppliers with status lower than that of supplier S1.

SQL >

select sno from s where st < (select  
st from s where sno = 'S1');

OUTPUT:

SNO

-----  
S2

Q.34. Get job numbers for jobs whose city is first in the alphabetic list of such cities.

SQL >

select jno from J where city = (select  
city from J order by city where  
rownum = 1);

OUTPUT:

JNO

-----  
J3  
J4

Q.35. Get job numbers for jobs supplied with part P1 in an average quantity greater than the greatest quantity in which any part is supplied to project J3.

SQL >

select jno from SPJ where pno = 'P1'  
group by jno having avg(QTY) > all (select  
max(QTY) from SPJ where jno = 'J3');

OUTPUT:

JNO

-----  
J4

## DBMS LAB

Q.36. Get supplier numbers for suppliers supplying some job with part P1 in a quantity greater than the average shipment quantity of part P1 for that project.

SQL >

Select sno from SPJ s1 where pno = 'P1'  
and qty > (select avg(qty) from SPJ s2  
where pno = 'P1' and s2.jno = s1.jno);

OUTPUT :

SNO

S1

Q.37. Get part numbers from parts supplied to any job in London.

SQL >

Select pno from SPJ natural join J where  
city = 'LONDON';

OUTPUT :

PNO

P3

P6

P5

Q.38. Get job numbers for jobs using at least one part available from suppliers S1 i.e, we know that S1 shipped that part.

SQL >

Select distinct jno from SPJ where pno in  
(Select pno from SPJ where pno = 'S1');

OUTPUT :

JNO

J1

J4

Q.39. Get job numbers for jobs not supplied with any red part by any London supplier.

SQL >

Select jno from SPJ minus select jno from  
SPJ where (jno, pno) in (Select r.jno, p.pno  
from R cross join P where p.color = 'RED'  
and r.city = 'LONDON');

OUTPUT :

JNO

J2

## DBMS LAB

J5  
J6

Q.40. Get job numbers for jobs supplied entirely by supplier S2.

SQL > Select jno from SPJ where sno = 'S2'  
minus select jno from SPJ where sno != 'S2';

OUTPUT :

JNO  
-----  
J6

Q.41. Get part numbers for parts supplied to all jobs in London.

SQL > Select distinct pno from SPJ natural join where city = 'LONDON' minus select distinct pno from SPJ

OUTPUT : natural join p where city = 'LONDON';  
PNO  
-----  
P3  
P5

Q.42. Construct a list of all cities in which at least one supplier, part, or job is located.

SQL > select city from s union select city from j  
union select city from p;

OUTPUT :  
CITY  
-----  
Athens  
London  
Oslo  
Paris  
Rome

Q.43. Get the colors of parts either whose city is london, or paris or both.

SQL > Select distinct color from p where city = 'LONDON' or city = 'PARIS';

OUTPUT :  
COLOR

## DBMS LAB

Blue  
Green  
Red

Q.44. How many suppliers in Athens that supply red parts?

SQL >

select count(sno) from s natural join SPJ where city = 'ATHENS' and pno in (select pno from p where color = 'Red');

OUTPUT:

COUNT (SPJ.SNO)

4

Q.45. Find sname of suppliers who do not supply any part heavier than 18 (to any project).

SQL >

select sname from s minus select sname from s natural join SPJ where pno in (select pno from p where color = 'RED');

OUTPUT:

SNAME

Blake  
Jones  
Smith

Q.46. Find number and name of suppliers that supplies a 'Nut'.

SQL >

select distinct sname from s natural join SPJ where pno in (select pno from p where pname = 'NUT');

OUTPUT:

SNAME

Adams  
Smith

Q.47. Find number of projects that do not use any locally made parts (i.e., if the project takes place in city x, then it does not use any part made in city x).

SQL >

select count(jno) from j minus select count(jno) from j natural join SPJ where pno in (select pno from p where p.city = m.city);

OUTPUT:

PNO

## DBMS LAB

---

P1  
P2  
P4  
P5

Q.48. Get suppliers for whom the total shipment quantity, taken over all shipments for the supplier is less than 1000.

SQL > *Select \* from s where sno in (select sno from SPJ group by sno having sum(qty) < 1000);*

OUTPUT:

SNO	SNAME	STATUS	CITY
S1	Smith	20	London
S4	Clark	20	London
S3	Blake	30	Paris

Q.49. Get suppliers for whom the minimum shipment quantity is less than half the maximum shipment quantity (taken over all shipments for the supplier in both cases).

SQL > *Select \* from s where sno in (select sno from SPJ group by e.sno having min(qty) < all (select max(qty) from SPJ m where e.sno = m.sno);*

OUTPUT:

SNO	SNAME	STATUS	CITY
S1	Smith	20	London
S2	Jones	10	Paris
S3	Blake	30	Paris
S5	Adams	30	Athens

Q.50. For each supplier, find supplier details and total, maximum, and minimum shipment quantity, taken over all shipments for the supplier.

SQL > *Select \* from s natural join (select sno, sum(qty) as TOTQ, max(qty) as maxQ, min(qty) as minQ from SPJ group by sno);*

OUTPUT:

SNO	SNAME	STATUS	CITY	TOTQ	MAXQ	MINQ
S2	Jones	10	Paris	3200	800	100
S1	Smith	20	London	900	700	200
S4	Clark	20	London	600	300	300
S5	Adams	30	Athens	3100	800	100
S3	Blake	30	Paris	700	500	200

Teacher I/C

Mr. Raj Kumar Singh

## DBMS LAB

### Lab 7 Introduction to view, synonym, sequence, index

One of the important features of RDBMS is the Data Abstraction. The Data Abstraction gives different view of data to different users. All the information in a database need not be accessible to all the users. Sometimes, in an application, a different view of the data or the information is described and the relevant data changes from user to user. This is handled in ORACLE using the VIEWS.

A SYNONYM is a simple alias for a table, view, sequence, or other database objects. Synonyms can be used for giving meaningful alternative names for database objects.

A SEQUENCE is a database object used to generate the series of unique integers. Sequences are typically used with primary key or unique columns.

#### Views

##### What is a View?

The table of a database defines the structure and the organization of its data. Once they are defined they always present the data in a particular way. Sometimes, in an application, a different view of the data or the information in a different format is described. This is handled in ORACLE using the VIEWS.

A VIEW is a virtual table in the database. The contents of a view are defined by a query.

A view can represent a subset of the data in a table. This could be a horizontal subset consisting of some of the rows from the base table or vertical subset consisting of some of the columns from the base table. The data from multiple tables can be also combined together using a view.

#### Characteristics of a View

Views do not exist physically. Views are virtual tables that exist only as definitions in the system catalogue. Views are stored in the data dictionary in the table called USER\_VIEWS.

#### Advantages of Views

Views provide several advantages and can be useful in various ways. In small (desktop) applications, views can be used to simplify the data requests. In large database applications like production data views can be used to restrict access to the data and enforcing security.

The major advantages of views are

#### Security

Each user can be given permission to access the database only through a small set of views that contains the specific data the user is authorized to see, rather than the entire table, thus restricting the user's access to stored data.

## **DBMS LAB**

---

### **Query Simplicity**

A view can draw data from several different tables and present it to the user as a single table, turning what would have been multi-table queries into single table queries against the view.

### **Structural Simplicity**

Views can give a user a "personalized" view of the database structure, presenting the database as a set of virtual tables that make sense for the user.

### **Insulation from Change**

A view can present a consistent, unchanged image of the structure of the database, even if the underlying source tables are split, restructured or renamed.

### **Data Integrity**

If data is accessed and entered through a view, the DBMS can automatically check the data to ensure that it meets specified integrity constraints.

### **Disadvantages of Views**

While views provide substantial advantages as discussed above, there are also two major disadvantages of using a view instead of a real table.

### **Performance**

Views give the appearance of a table, but the DBMS must still translate the queries against the view into queries against the underlying source tables. If the view is defined by a complex, multi-table query, then even a simple query against the view becomes a complicated join, and it may take a long time to execute.

### **Update Restrictions**

When a user tries to update rows of a view, the DBMS must translate the request into an update on the rows of the underlying source tables. This is possible for simple views, but more complex views cannot be updated; they are "read-only".

The above disadvantages mean that we cannot indiscriminately define views and use them instead of the source tables. Instead, we must in each case weigh the pros and cons of creating a view for a given situation.

### **Creating a View**

A view can be created by using a CREATE VIEW command.

The general syntax of the command is

```
CREATE [OR REPLACE] [FORCE/ NOFORCE]
[(column_list)]
VIEW view_name
```

## DBMS LAB

### AS Query

[WITH CHECK OPTION]

[WITH READ ONLY];

Where

- **CREATE** – creates the view with the name specified.
- **OR REPLACE** – replaces the view if it already exists. This option is used to change the definition of an existing view.
- **FORCE** – creates the view (with compilation errors) regardless of whether the view's base table exists or the owner has the privilege on them. But to view the VIEW the owner must have the privileges and the base table must exist.
- **NOFORCE** – creates the view only if the base table exists and the owner has the privileges on them. (This is the default).
- **COLUMN\_LIST** – specifies different column names than their original names. These new column names can be only used while referring to views.

Example on creating a View:

Create a view on emp table which gives access to the employee number, employee name and designation information of employees working in the department 30 only. This can be done using following query.

```
CREATE OR REPLACE VIEW empview30 AS  
SELECT empno, ename, job  
FROM emp  
WHERE deptno = 30;
```

### Points to Remember

- The view's default column names are same as the table's column names.
- New column names if specified in the CREATE VIEW clause have one-to-one relationship with the column names in the SELECT clause of the query.
- The GROUP BY clause can be used in the definition of a view.
- Views may be joined or nested with other views or tables.
- Views may be used in the SELECT statement while defining other views.

### Querying a View

A view can be queried and used just like database tables. For example, if we want to see the structure of view (in the SQL\*PLUS environment) we use the DESCRIBE command. This is the same command we used earlier for tables.

Syntax:

```
Describ[ue] view_name
```

Example:

Display the structure of the view empview30.

Query

```
SQL>Desc empview30
```

We can query a view just like a database table. This can be seen in the following examples.

## DBMS LAB

### Example1

List the details of employees working as clerk from department 30.

Query

```
SQL> SELECT *
      FROM empview30
     WHERE job = 'CLERK';
```

### Example2

List the details of employees working as salesman from department 30 in the ascending order of their name.

Query

```
SQL> SELECT *
      FROM empview30
     WHERE job = 'SALESMAN'
   ORDER BY ename;
```

### Types of View

The definition of a view decides what operations can be performed on a view. On this basis views are categorized as

- Simple or Updateable Views.
- Complex or Non-updateable Views.

### Complex or Non-updateable Views

The Complex or Non-updateable views are used only to retrieve the corresponding data from the table. We cannot use the DML statements like INSERT, UPDATE, or DELETE with these views.

### Simple or Updateable View

When we refer to the term updating the views what actually implies is the updating of the underlying source table using the view. Views can be updated much the same way we update the tables i.e. by using the DML commands.

For an updateable view there are certain restrictions imposed by the ANSI/ ISO SQL standard.

These restrictions are as follows,

- The FROM clause must specify only one updateable tables.
- DISTINCT must not be specified i.e. duplicate rows must not be excluded from the query result.
- Each SELECT item must be a simple column reference; the SELECT list cannot contain expressions, calculated columns, or column function.
- The WHERE clause must not include a sub-query. Simple row-by-row search conditions may be used.
- The query must not include a GROUP BY or HAVING clause.

### Manipulating Base Tables Using Views

We can manipulate the base table using views. The DML commands Insert, Update, or Delete can be used with views.

The following restrictions apply, while manipulating base tables through views.

## DBMS LAB

- The view must be based on a single table.
- It must not have columns that are aggregate functions.
- It must not have expression in its definition.
- It must not use distinct in its definition.
- It must not use group by or having clause in its definition.
- It must not use sub queries.
- We cannot insert if the underlying table has any NOT NULL columns that don't appear in the view.

Some of the view definitions and remarks about the way they can be updated are as follows

Example 1:

```
SQL> CREATE OR REPLACE VIEW empview AS  
      SELECT *  
        FROM emp  
       WHERE deptno = 30;
```

Remark

The view empview is updateable because it does not violate any of the restrictions mentioned above.

Example 2:

```
SQL> CREATE OR REPLACE VIEW empsalview (empno, ename, sal, Totalsal)  
      AS  
      SELECT empno, ename, sal, sal + NVL(comm,0)  
        FROM emp  
       WHERE deptno = 30;
```

Remark

The view empsalview is not updateable because it contains an expression.

Specific restrictions on a view for DML operations are discussed in the following sections.

### Delete Restrictions

We cannot delete the rows when the row contains

- Group Function
- DISTINCT Clause
- GROUP BY Clause
- Join Condition

Example: The following view has a delete restriction because it is based on two tables.

```
SQL> CREATE OR REPLACE VIEW empview AS  
      SELECT emp.* , dept.dname  
        FROM emp, dept;
```

### Update Restrictions

We cannot update a view when view contains an expression such as SAL+COMM.

Other restrictions are same as stated for delete.

## DBMS LAB

For example, the following view has an update restriction because it contains an expression.

```
SQL> CREATE OR REPLACE VIEW empsalview (empno, ename, sal, Totalsal)
      AS SELECT empno, ename, sal, sal + NVL(comm,0) FROM emp;
```

### Insert Restrictions

We cannot insert a row using view when a view does not contain all the NOT NULL columns of the base table.

Other restrictions are same as stated for delete and update.

For example, the following view has an insert restriction. This is because it does not include empno column.

```
SQL> CREATE OR REPLACE VIEW empsalview (ename, sal, comm) AS
      SELECT ename, sal, comm.
            FROM emp;
```

### WITH CHECK OPTION

Sometimes, INSERT or UPDATE operations on a view can result in data that the view can't retrieve. In such case you might want to restrict the view so that the view does not accept the data that it can't display.

Example:

```
SQL> CREATE OR REPLACE VIEW empview30 AS
      SELECT empno, ename, deptno
            FROM emp
           WHERE deptno = 30;
```

The above view can display records for department number 30 only. Being Simple or updateable view you can execute following DML on it.

```
SQL> INSERT INTO empview30
```

However, when you query on empview30, it cannot display above inserted information. The WITH CHECK OPTION can be used with CREATE VIEW statement for preventing user against entering data through view which can't be displayed by view.

```
SQL> CREATE OR REPLACE VIEW empview30 AS
      SELECT empno, ename, deptno
            FROM emp
           WHERE deptno = 30
        WITH CHECK OPTION;
```

### WITH READ ONLY Option

Sometimes you may want to prevent users from making changes to the base table through view. In such case use WITH READ ONLY option with CREATE VIEW command.

## DBMS LAB

Example

```
SQL> CREATE OR REPLACE VIEW empview AS  
      SELECT empno, ename, job, deptno  
      FROM emp  
      WITH READ ONLY;
```

### Dropping a View

We can drop a view using following command.

Syntax:

```
DROP VIEW view_name;
```

### Synonyms

A SYNONYM is a simple alias for a database object. Synonyms can be used for giving meaningful alternative names for database objects. In this section we describe the concept and use of synonyms.

#### What is a Synonym?

A synonym is a simple alias for a table, view, sequence, or other database objects. Because a synonym is just an alternative name for an object it requires no storage space. ORACLE stores only definition of a synonym in the data dictionary. ORACLE allows us to create both, public or private synonyms. A public synonym is a synonym that is available to every user in a database. A private synonym is a synonym within the schema of a specific user.

#### Creating a Synonym

As we know, a synonym is an alternative name for a table, view, sequence, procedure, stored function, package, snapshot or another synonym.

To create a synonym CREATE SYNONYM command is used.

Syntax

```
CREATE [PUBLIC] SYNONYM [schema.]synonym_name  
FOR [schema.]object;
```

Where

- **PUBLIC** – Creates a public synonym that is accessible to all users in a database. If we omit this option, the synonym is private and is accessible only within our schema.  
For creating PUBLIC synonym you must have DBA privileges.
- **Schema** – This is the schema to contain the synonym. If we omit schema, ORACLE creates the synonym in our own schema.
- **Synonym\_name** – Name of the synonym to be created.
- **Object** – Identifies the object for which the synonym is created. This object can be of following types:
  1. Table
  2. View
  3. Sequence and
  4. Subprograms like a stored procedure, a function or a package.

## DBMS LAB

Example:

Create a synonym employee for emp table. This can be done as follows.

Query

```
SQL> CREATE SYNONYM employee  
      FOR emp;
```

Now, we can use the synonym employee for emp table and write queries. For example,

```
SQL> SELECT *  
      FROM employee;
```

### Deleting a Synonym

To delete the synonym from the database use the DROP command.

Syntax

```
DROP [PUBLIC] SYNONYM [schema.]synonym_name
```

Where

- **PUBLIC** – To drop the public synonym, we must specify the PUBLIC keyword.
- **Schema** – This is the schema name containing the synonym. ORACLE assumes the synonym is in our own schema. If we omit this option.
- **Synonym\_name** – Name of the synonym to be deleted from database.

### Sequences

A sequence is a database object used to generate the series of unique integers for use as primary keys. When an application inserts a new row into a table, the application simply requests a database sequence to provide the next available value in the sequence for the new row. Multiple users can use same sequence.

### Creating a Sequence

To create a sequence, use the CREATE SEQUENCE command.

Syntax

```
CREATE SEQUENCE sequence_name  
[INCREMENT BY n]  
[START WITH n];
```

Where

- **Sequence\_name** – The name of the sequence object to be created.
- **Incremented by** – Specifies the incremented value between sequence numbers. This value can be positive or negative, but it cannot be 0. If this value is negative, then the sequence is created in descending order. If the increment is positive, then the sequence is created in ascending order. If we omit this clause, the interval defaults to 1.

Creating a sequence is shown in the following example.

## DBMS LAB

Example:

Create a sequence for generating employee numbers starting with 7700.

Query

```
SQL> CREATE SEQUENCE empseq  
      INCREMENT BY 1  
      START WITH 7700;
```

### Using a Sequence

A sequence can be used with database operations. It is used with INSERT and UPDATE DML commands. Sequences provide two attributes for using the value generated using the sequence. These attributes are CURRVAL and NEXTVAL. Their use is as follows.

- **Sequence\_name.curval** – Returns the current value in the sequence.
- **Sequence\_name.nextval** – Increments are current value in the sequence and returns it.

The use of a sequence can be seen in the following example.

Example

Insert a new employee 'BILL' with a designation 'CLERK' and salary of 1000, He is to be posted to the department 20.

This can be done using the following query.

Query

```
SQL> INSERT INTO emp (empno, ename, job, sal, dept)  
      VALUES (empseq.nextval, 'BILL', 'CLERK', 1000, 20);
```

### Deleting a Sequence

To delete a sequence from the database use DROP SEQUENCE command.

Syntax

```
DROP SEQUENCE [schema.]sequence_name
```

Where

- **Schema** – It is the schema containing the sequence. The default assumes the sequence is in our own schema.
- **Sequence\_name** – The name of the sequence to be dropped.

For example, to drop the sequence empseq use the following command.

```
SQL> DROP SEQUENCE empseq;
```

Q.1. Create a view emp30 which display the name and job titles of employee working in department 30.

## DBMS LAB

SQL> create view emp30 as select ename, job  
from emp where deptno = 30;

OUTPUT:

SQL> select \* from emp30;

ENAME	JOB
ALLEN	SALESMAN
WARD	SALESMAN
MARTIN	SALESMAN
BLAKE	MANAGER
TURNER	SALESMAN
JAMES	CLERK

6 rows selected.

Q.2. Create a view empcount which counts the number of employees working in each department.

SQL> create view empcount as select deptno,  
count(ename) from emp group by deptno  
order by deptno desc;

OUTPUT:

SQL> select \* from empcount;

DEPTNO	NOOFEMP
30	6
20	5
10	3

Q.3. Create a view empsales which display the name and job titles of employee working in department 'SALES'.

SQL> create view empsales as select ename, job  
from emp natural join dept where dname  
= 'SALES';

OUTPUT:

SQL> select \* from empsales;

ENAME	JOB
-------	-----

## DBMS LAB

ALLEN	SALESMAN
WARD	SALESMAN
MARTIN	SALESMAN
BLAKE	MANAGER
TURNER	SALESMAN
JAMES	CLERK

6 rows selected.

Q.4. Create a view emploc which display the name and job of employees working in location 'CHICAGO'.

SQL>

create view emploc as select ename, job  
from emp natural join dept where loc

OUTPUT:

SQL> select \* from emploc;

ENAME	JOB
ALLEN	SALESMAN
WARD	SALESMAN
MARTIN	SALESMAN
BLAKE	MANAGER
TURNER	SALESMAN
JAMES	CLERK

6 rows selected.

= 'CHICAGO';

Q.5. Create a view emptotalsal which find the name and total salary employees department wise and display only those departments which have at least 5 employees.

SQL>

create view emptotal as select deptno, sum(sal)  
as TotalSal from emp group by deptno having

count(ename)

= 5;

OUTPUT:

SQL> select \* from emptotalsal;

DEPTNO	TOTALSAL
30	9400
20	10875

Teacher I/C

Mr. Raj Kumar Singh

## DBMS LAB

### Lab 8 Introduction to PL/SQL

Q.1. Write a PL/SQL program to input a name and display it.

PL/SQL Program:

```
declare
    name varchar2(20) = 'name';
begin
    dbms_output.put_line ("My name
is " || name);
end;
```

**OUTPUT:**

```
Enter value for name: rajkumar
old  4:   name := '&name';
new  4:   name := 'rajkumar';
Your name is rajkumar
```

PL/SQL procedure successfully completed.

## DBMS LAB

- Q.2. Write a PL/SQL program to input 2 numbers and display addition, subtraction, multiplication, division and modulus of these 2 numbers.

PL/SQL Program :

```
declare
    n1 number = &n1;
    n2 number = &n2;
begin
    dbms_output.put_line ("Addition = " || (n1+n2));
    dbms_output.put_line ("Subtraction = " || (n1-n2));
    dbms_output.put_line ("Multiplication = " || (n1*n2));
    dbms_output.put_line ("Division = " || (n1/n2));
    dbms_output.put_line ("Modulus = " || mod(n1, n2));
end;
/
```

### OUTPUT:

```
Enter value for a: 5
old 2: a number := &a;
new 2: a number := 5;
Enter value for b: 2
old 3: b number := &b;
new 3: b number := 2;
Addition = 7
Subtraction = 3
Multiplication = 10
Division = 2.5
Modulus = 1
```

PL/SQL procedure successfully completed.

## DBMS LAB

Q.3. Write a PL/SQL program to input a 5 digit number and display the reverse of that number.

PL/SQL Program :

```
declare
    num    number := &n;
    rev    number := 1;
begin
    while (num != 0)
loop
    rev := rev * 10 + mod (num, 10);
    num := floor (num / 10);
end loop;
dbms_output.put_line ('Reverse = ' || rev);
end;
```

**OUTPUT:**

```
Enter value for n: 12345
old 2: n number := &n;
new 2: n number := 12345;
Reverse = 54321
```

PL/SQL procedure successfully completed.

## DBMS LAB

- Q.4. Write a PL/SQL program to read the radius from the keyboard and insert it into table CIRCLE the radius along with area, the CIRCLE table has two columns defined as radius and area.

PL/ SQL Program :

```
declare
    r Number := &n;
    area Number;
begin
    area := 3.14 * r * r;
    insert into circle values(r, area);
    dbms_output.put_line ("Area of
    circle" || area);
    commit;
end;
```

**OUTPUT:**

```
Enter value for r: 5
old 2:   r number := &r;
new 2:   r number := 5;
Radius and area inserted in circle table
```

```
PL/SQL procedure successfully completed.
```

```
SQL> select * from circle;
```

RADIUS	AREA
5	78.5

## DBMS LAB

- Q.5. Write a PL/SQL program to input the employee number and display its total salary (i.e. sal + comm, and if comm is null assume it to be 0).

PL/SQL Program :

```
declare
    en Number := &en;
    Salary Number;
    Comm Number;
    sal Number;
begin
    select comm, sal into comm, salary from emp
    where empno = en;
    if comm is null then
        comm := 0;
    end if;
    it :.sal = salary + comm;
    dbms_output.put_line ('Total salary $' ||
                           '= ' || sal);
end;
```

### OUTPUT:

```
Enter value for eno: 7369
old 2:    enoemp.empno%type := &eno;
new 2:    enoemp.empno%type := 7369;
Totalsal = 800
```

PL/SQL procedure successfully completed.

## DBMS LAB

- Q.6. Write a PL/SQL program to input a number and find whether it is even or odd, if it is even insert it into the table EVEN or insert it into table ODD, both tables have only one column i.e. NUM.

PL/SQL Program :

```
declare
    n Number := &n;
begin
    if mod (n, 2) = 0 then
        dbms-output.put_line ('Number is even');
        insert into even values (n);
    else
        dbms-output.put_line ('Number is odd');
        insert into odd values (n);
    end if;
    commit;
end;
```

### OUTPUT:

```
Enter value for n: 5
old  2:  n number := &n;
new  2:  n number := 5;
```

```
PL/SQL procedure successfully completed.
```

```
SQL> select * from even;
```

```
no rows selected
```

```
SQL> select * from odd;
```

NUM
-----
5

## DBMS LAB

- Q.7. Write a PL/SQL program to input a number and find the factorial of that number using  
(i) For loop      (ii) Simple loop      (iii) While loop

PL/SQL Program using for loop :

```
declare
    n Number := sn;
    f Number := 1;
begin
    for i in 1 ... n
loop
    f = f * i;
end loop;
dbms_output.put_line('Factorial=' || f);
end;
/
```

PL/SQL Program using simple loop :

```
declare
    n Number := sn;
    f Number := 1;
    i Number;
begin
    i := n;
loop
    f = f * i;
    exit when (i <= 1);
    i := i - 1;
end loop;
dbms_output.put_line('Factorial=' || f);
end;
/
```

## DBMS LAB

PL/SQL Program using while loop :

```
declare
    n Number := 5;
    f Number := 1;
    i Number;
begin
    i := n;
    while (i > 0)
loop
    f = f * i;
    i = i - 1;
end loop;
dbms_output.put_line ("Factorial = " || f);
end;
```

### OUTPUT:

```
Enter value for n: 5
old 2: n number := &n;
new 2: n number := 5;
```

Factorial of 5 is = 120

PL/SQL procedure successfully completed.

## DBMS LAB

Q.8. Write a PL/SQL program to display the following patterns.

PL/SQL Program :

```
declare
    n Number := 8;
    c Number := 1;
begin
    for i in 1...n
    loop
        for j in 1...i
        loop
            dbms_output.put_line(c || ' ');
            c := c + 1;
        end loop;
        dbms_output.new_line();
    end loop;
end;
```

OUTPUT:

```
1
2   3
4   5   6
```

PL/SQL Program :

```
declare
    n Number := 8;
    c Number := 1;
    x Number;
begin
    for i in 1...n
    loop
        x := n - i;
        for j in 1...x
        loop
```

```
loop
    dbms_output.putline(' ');
end loop;
for k in 1...i
loop
    dbms_output.put(c);
    c := c + 1;
end loop;
dbms_output.new_line();
end loop;
end;
```

OUTPUT:

```
      1
     2   3
    4   5   6
```

## DBMS LAB

PL/SQL Program :

```
declare
    n Number := 5;
    x Number;
begin
    for i in 1..n
    loop
        n := n - i;
        for j in 1..n
        loop
            dbms_output.put(' ');
            end loop;
        for k in 1..i
        loop
            dbms_output.put(k || ' ');
            end loop;
        for m in reverse 2..i
        loop
            dbms_output.put(m - 1 || ' ');
        end loop;
        dbms_output.new_line();
    end loop;
end;
```

OUTPUT:

		1
1	2	1
2	3	2

## DBMS LAB

PL/SQL Program :

```
declare
    n Number := 5;
    x Number;
begin
    for i in 1 .... n
loop
    x := 65 + i - 1;
    for j in 65 .... x
loop
    dbms_output.put(chr(j) || ' ');
end loop;
dbms_output.new_line();
end loop;
end;
```

OUTPUT :

A		
A	B	
A	B	C

Teacher I/C  
Mr. Raj Kumar Singh

# DBMS LAB

## Lab 9

### Cursor

#### Topics

- (i) What is Cursor?
- (ii) Types of Cursors
- (iii) Cursor attributes
- (iv) Syntax of Cursor for loop

#### Cursors :

Oracle uses work area to execute SQL statements and to store processing information. Every time user executes SQL statements of any sort, there will be an activity on database that involves **cursor**.

Cursor is a memory (work) area that oracle engine uses for its internal processing for executing and storing the results of SQL statements and this work area is reserved for SQL's operations also called Oracle's private area or cursor.

The set of rows returned by a SQL query is called the result set. This result set is called **Active Data Set**, because data in cursor is ready to undergo any kind of processing. The size of cursor is the same as the size required by the number of rows in Active Data Set.

E.g., When a user fires a select statement as

```
Select empno, ename, job, sal from emp where deptno = 10;
```

All the rows returned by the query are stored in the cursor at the server and will be displayed at the client end.

#### **Types of Cursors**

Cursors may be categorized on the situations under which they are opened. Basically oracle has highlighted its two types :

1. Implicit Cursor
  2. Explicit Cursor
- 
1. **Implicit Cursor** : Implicit cursors are declared by PL/SQL implicitly for all SQL statements. They are opened and managed by oracle engine internally. So there is no need to open and manage by the users, these operations are performed automatically.
  2. **Explicit Cursor** : Explicit Cursors are user defined cursors for processing of multiple records returned by a query. Explicit cursors are declared explicitly, along with other identifiers to be used in a block, and manipulated through specific statements within the block's executable actions. These are user defined cursors defined in the declare section of the PL/SQL block. The user defined cursors needs to be opened, before the reading of the rows can be done, after which the cursor is closed. Cursor marks the current position in an active set.

#### **General Cursor Attributes :**

Whenever any cursor is opened and used, the oracle engine creates a set of four system variables, which keeps track of the current status of a cursor. These cursor variables can be accessed and used in a PL/SQL block. Both implicit and explicit cursor has four attributes. They described as

Attributes	Description
%isopen	It returns true if cursor is open, false otherwise.
%found	It returns true if record was fetched successfully from the opened cursor and false otherwise.
%notfound	It returns true if record was not fetched successfully and false otherwise.
%rowcount	It returns number of records processed from cursor.

## DBMS LAB

### Steps for explicit cursor

1. Declare a cursor mapped to a SQL select statement that receive data for processing.
2. Open the cursor.
3. Fetch data from the cursor one row at a time into memory variables.
4. Process the data held in the memory variables as required using a loop.
5. Exit from the loop after processing is complete.
6. Closes the cursor.

### PL/SQl Programs

Q.1. Write a PL/SQL program to display a message to check whether the record is deleted or not. [Using %found]

PL/SQL Program :

```
declare
    eno Number = &eno;
begin
    delete from emp where empno = eno;
    if SQL%found then
        dbms_output.put_line ('Record
            successfully deleted');
    else
        dbms_output.put_line ('Record not
            deleted');
    end if;
end;
/
```

### OUTPUT:

```
Enter value for eno: 1000
old  2: delete from emp where empno = &eno;
new  2: delete from emp where empno = 1000;
Record not deleted
```

PL/SQL procedure successfully completed.

## DBMS LAB

Q.2. Write a PL/SQL program to display a message to check whether the record is deleted or not.[Using %notfound]

PL/SQL Program :

```
declare
    eno Number := &eno;
begin
    delete from emp where empno = eno;
    if SQL%notfound then
        dbms_output.put_line('Record not deleted');
    else
        dbms_output.put_line('Record successfully deleted');
    end if;
end;
```

**OUTPUT:**

```
Enter value for eno: 1000
old  2:  delete from emp where empno = &eno;
new  2:  delete from emp where empno = 1000;
Record not deleted
```

PL/SQL procedure successfully completed.

## DBMS LAB

Q.3. Write a PL/SQL program to display a message to give the number of records deleted by the delete statement issued in a PL/SQL block.

PL/SQL Program

```
declare
    eno Number := &eno;
    N Number;
begin
    delete from emp where empno = eno;
    if SQL%Rowcount = 1 then
        DBMS_OUTPUT.PUT_LINE ('No of record deleted' || N);
    else
        DBMS_OUTPUT.PUT_LINE ('Not deleted');
    end if;
end;
```

**OUTPUT:**  
Enter value for eno: 7369  
old 4: delete from emp where empno = &eno;  
new 4: delete from emp where empno = 7369;  
Total number of records deleted 1

PL/SQL procedure successfully completed.

## DBMS LAB

- Q.4. Write a PL/SQL program to display the empno, ename, job of employees of department number 10.  
[Using variables]

PL/SQL Program :

```
declare
    eno    Emp.empno % type;
    emp    title emp.empname % type;
    jobtitle emp.job % type;
    dno    emp.deptno % type := 10;
    cursor pra is select empno, ename, job
                 from prat
                where deptno = dno;
begin
    open pra;
    loop
        fetch pra into eno; job title;
        Exit when pra%notfound;
        dbms_output.put_line ('empno :=' || eno);
        dbms_output.put_line ('jobtitle :=' || jobtitle);
    end loop;
    close pra;
end;
```

**OUTPUT:**

empno : 7782  
ename : CLARK  
job : MANAGER  
empno : 7839  
ename : KING  
job : PRESIDENT  
empno : 7934  
ename : MILLER  
job : CLERK

PL/SQL procedure successfully completed.

## DBMS LAB

- Q.5. Write a PL/SQL program to display the empno, ename, job of employees of department number 10.  
[Using records]

PL/ SQL Program :

```
declare
    rec prat%rowtype;
    dno prat.deptno%type := 10;
    cursor pra is select * from prat
    where deptno = dno;
begin
    open pra;
    loop
        fetch pra into rec;
        exit when pra%not found;
        dbms_output.put_line('empno:= ' || rec.empno);
        dbms_output.put_line('ename:= ' || rec.ename);
        dbms_output.put_line('Job title:= ' || rec.job);
    end loop;
    close pra;
end;
```

### OUTPUT:

empno : 7782  
ename :CLARK  
job :MANAGER  
empno : 7839  
ename :KING  
job :PRESIDENT  
empno : 7934  
ename :MILLER  
job :CLERK

PL/SQL procedure successfully completed.

## DBMS LAB

Q.6. Write a PL/SQL program to display the empno, ename, job of employees of department number 10.  
[Using cursor for loop]

PL/SQL Program :

```
declare
    dno emp.deptno % type := 10;
    cursor pra in select * from emp
    where deptno = dno;
begin
    for rec in pra
    loop
        dbms_output.put_line('empno := ' || rec.empno);
        dbms_output.put_line('ename := ' || rec.ename);
        dbms_output.put_line('Job title := ' || rec.job);
    end loop;
end;
```

OUTPUT:

empno : 7782  
ename : CLARK  
job : MANAGER  
empno : 7839  
ename : KING  
job : PRESIDENT  
empno : 7934  
ename : MILLER  
job : CLERK

PL/SQL procedure successfully completed.

## DBMS LAB

### Lab 9 Procedures and Functions

#### Stored Procedures

A **stored procedure** or in simple a **proc** is a named PL/SQL block which performs one or more specific task. This is similar to a procedure in other programming languages. A procedure has a header and a body. The header consists of the name of the procedure and the parameters or variables passed to the procedure. The body consists of declaration section, execution section and exception section similar to a general PL/SQL Block. A procedure is similar to an anonymous PL/SQL Block but it is named for repeated usage.

We can pass parameters to procedures in three ways.

- 1) IN-parameters
- 2) OUT-parameters
- 3) IN OUT-parameters

A procedure may or may not return any value.

General Syntax to create a procedure is:

```
CREATE [OR REPLACE] PROCEDURE proc_name [list of parameters]
```

```
IS
```

```
    Declaration section
```

```
BEGIN
```

```
    Execution section
```

```
EXCEPTION
```

```
    Exception section
```

```
END;
```

**IS** - marks the beginning of the body of the procedure and is similar to **DECLARE** in anonymous PL/SQL Blocks. The code between IS and BEGIN forms the Declaration section.

The syntax within the brackets [ ] indicate they are optional. By using **CREATE OR REPLACE** together the procedure is created if no other procedure with the same name exists or the existing procedure is replaced with the current code.

The below example creates a procedure 'employer\_details' which gives the details of the employee.

```
1> CREATE OR REPLACE PROCEDURE employer_details
2> IS
3> CURSOR emp_cur IS
4> SELECT first_name, last_name, salary FROM emp_tbl;
5> emp_rec emp%rowtype;
6> BEGIN
7> FOR emp_rec in sales_cur
8> LOOP
9> dbms_output.put_line(emp_rec.first_name || ' ' || emp_rec.last_name
10> || ' ' || emp_rec.salary);
11> END LOOP;
12>END;
13> /
```

#### How to execute a Stored Procedure?

There are two ways to execute a procedure.

- 1) From the SQL prompt.

```
EXECUTE [or EXEC] procedure_name;
```

- 2) Within another procedure - simply use the procedure name.

## DBMS LAB

**procedure\_name;**

**NOTE:** In the examples given above, we are using backward slash '/' at the end of the program. This indicates the oracle engine that the PL/SQL program has ended and it can begin processing the statements.

### PL/SQL Functions

A function is a named PL/SQL Block which is similar to a procedure. The major difference between a procedure and a function is, a function must always return a value, but a procedure may or may not return a value.

The General Syntax to create a function is:

**CREATE [OR REPLACE] FUNCTION function\_name [parameters]**

**RETURN return\_datatype;**

**IS**

**Declaration\_section**

**BEGIN**

**Execution\_section**

**Return return\_variable;**

**EXCEPTION**

**exception section**

**Return return\_variable;**

**END;**

- 1) **Return Type:** The header section defines the return type of the function. The return datatype can be any of the oracle datatype like varchar, number etc.
- 2) The execution and exception section both should return a value which is of the datatype defined in the header section.

For example, let's create a function called "employer\_details\_func" similar to the one created in stored proc

```
1> CREATE OR REPLACE FUNCTION employer_details_func  
2> RETURN VARCHAR(20);  
3> IS  
4> emp_nameVARCHAR(20);  
5> BEGIN  
6> SELECT first_name INTO emp_name  
7> FROM emp_tbl WHERE empID = '100';  
8> RETURN emp_name;  
9> END;  
10>/
```

In the example we are retrieving the 'first\_name' of employee with empID 100 to variable 'emp\_name'.

The return type of the function is VARCHAR which is declared in line no 2.

The function returns the 'emp\_name' which is of type VARCHAR as the return value in line no 9.

### How to execute a PL/SQL Function?

A function can be executed in the following ways.

- 1) Since a function returns a value we can assign it to a variable.

**employee\_name := employer\_details\_func;**

If 'employee\_name' is of datatype varchar we can store the name of the employee by assigning the return type of the function to it.

- 2) As a part of a SELECT statement

## DBMS LAB

```
SELECT employer_details_func FROM dual;  
3) In a PL/SQL Statements like,  
    dbms_output.put_line(employer_details_func);  
    This line displays the value returned by the function.
```

### Parameters in Procedure and Functions

In PL/SQL, we can pass parameters to procedures and functions in three ways.

- 1) **IN type parameter:** These types of parameters are used to send values to stored procedures.
- 2) **OUT type parameter:** These types of parameters are used to get values from stored procedures. This is similar to a return type in functions.
- 3) **IN OUT parameter:** These types of parameters are used to send values and get values from stored procedures.

**NOTE:** If a parameter is not explicitly defined a parameter type, then by default it is an IN type parameter.

#### 1) IN parameter:

This is similar to passing parameters in programming languages. We can pass values to the stored procedure through these parameters or variables. This type of parameter is a read only parameter. We can assign the value of IN type parameter to a variable or use it in a query, but we cannot change its value inside the procedure.

The General syntax to pass a IN parameter is

```
CREATE [OR REPLACE] PROCEDURE procedure_name (  
param_name1 IN datatype, param_name2 IN datatype ... )
```

Where

param\_name1, param\_name2... are unique parameter names.

datatype - defines the datatype of the variable.

IN - is optional, by default it is a IN type parameter.

#### 2) OUT Parameter:

The OUT parameters are used to send the OUTPUT from a procedure or a function. This is a write-only parameter i.e., we cannot pass values to OUT parameters while executing the stored procedure, but we can assign values to OUT parameter inside the stored procedure and the calling program can receive this output value.

The General syntax to create an OUT parameter is

```
CREATE [OR REPLACE] PROCEDURE proc2 (param_name OUT datatype)
```

The parameter should be explicitly declared as OUT parameter.

#### 3) IN OUT Parameter:

The IN OUT parameter allows us to pass values into a procedure and get output values from the procedure. This parameter is used if the value of the IN parameter can be changed in the calling program.

By using IN OUT parameter we can pass values into a parameter and return a value to the calling program using the same parameter. But this is possible only if the value passed to the procedure and output value have a same datatype. This parameter is used if the value of the parameter will be changed in the procedure.

The General syntax to create an IN OUT parameter is

```
CREATE [OR REPLACE] PROCEDURE proc3 (param_name IN OUT datatype)
```

The below examples show how to create stored procedures using the above three types of parameters.

Example1:

#### Using IN and OUT parameter:

Let's create a procedure which gets the name of the employee when the employee id is passed.

```
1> CREATE OR REPLACE PROCEDURE emp_name (id IN NUMBER, emp_name OUT NUMBER)  
2> IS  
3> BEGIN  
4> SELECT first_name INTO emp_name
```

## DBMS LAB

```
5> FROM emp_tbl WHERE empID = id;  
6> END;  
7> /
```

We can call the procedure 'emp\_name' in this way from a PL/SQL Block.

```
1> DECLARE  
2> empName varchar(20);  
3> CURSOR id_cur SELECT id FROM emp_ids;  
4> BEGIN  
5> FOR emp_rec in id_cur  
6> LOOP  
7> emp_name(emp_rec.id, empName);  
8> dbms_output.putline('The employee ' || empName || ' has id ' || emp_rec.id);  
9> END LOOP;  
10> END;  
11> /
```

In the above PL/SQL Block

In line no 3; we are creating a cursor 'id\_cur' which contains the employee id.

In line no 7; we are calling the procedure 'emp\_name', we are passing the 'id' as IN parameter and 'empName' as OUT parameter.

In line no 8; we are displaying the id and the employee name which we got from the procedure 'emp\_name'.

Example 2:

Using IN OUT parameter in procedures:

```
1> CREATE OR REPLACE PROCEDURE emp_salary_increase  
2> (emp_id IN emptbl.empID%type, salary_inc IN OUT emptbl.salary%type)  
3> IS  
4> tmp_sal number;  
5> BEGIN  
6> SELECT salary  
7> INTO tmp_sal  
8> FROM emp_tbl  
9> WHERE empID = emp_id;  
10> IF tmp_sal between 10000 and 20000 THEN  
11> salary_inout := tmp_sal * 1.2;  
12> ELSIF tmp_sal between 20000 and 30000 THEN  
13> salary_inout := tmp_sal * 1.3;  
14> ELSIF tmp_sal > 30000 THEN  
15> salary_inout := tmp_sal * 1.4;  
16> END IF;  
17> END;  
18> /
```

The below PL/SQL block shows how to execute the above 'emp\_salary\_increase' procedure.

```
1> DECLARE  
2> CURSOR updated_sal is  
3> SELECT empID,salary  
4> FROM emp_tbl;  
5> pre_sal number;  
6> BEGIN
```

## DBMS LAB

```
7> FOR emp_rec IN updated_sal LOOP  
8> pre_sal := emp_rec.salary;  
9> emp_salary_increase(emp_rec.empID, emp_rec.salary);  
10> dbms_output.put_line('The salary of ' || emp_rec.empID ||  
11> ' increased from ' || pre_sal || ' to ' || emp_rec.salary);  
12> END LOOP;  
13> END;  
14>/
```

Q.1 Write a PL/SQL program to create a procedure which input a number and find the factorial of a number.

PL/SQL Program :

```
declare  
    dno emp.deptno % type := cd_no;  
    cursor pk is select * from emp where deptno=dno;  
begin  
    for rec in pk  
    loop  
        dbms_output.put_line ('empno:' || rec.empno);  
        dbms_output.put_line ('empname : ' || rec.ename);  
        dbms_output.put_line ('job : ' || rec.job);  
    end loop;  
end;
```

## DBMS LAB

**OUTPUT:**

```
SQL> variable n number  
SQL> execute proc_fact(5,:n);  
PL/SQL procedure successfully completed.
```

```
SQL> print n
```

```
-----  
N
```

```
120
```

- Q.2 Write a PL/SQL procedure called MULTI\_TABLE that takes two numbers as parameter and displays the multiplication table of the first parameter till the second parameter.

**PL/SQL Program :**

```
create or replace procedure multitable (a in number,  
b in number) is  
f number;  
begin  
for i in 1...n;  
loop  
a = a * i;  
dbms_output.put_line('0/1'  
* '1/j 11'  
= '1/F);  
end loop;  
end;
```

## DBMS LAB

### OUTPUT:

```
SQL> execute multi_table(2,5)
2 * 1 = 2
2 * 2 = 4
2 * 3 = 6
2 * 4 = 8
2 * 5 = 10
```

PL/SQL procedure successfully completed.

- Q.3 Write a PL/SQL function called POW that takes two numbers as argument and return the value of the first number raised to the power of the second.

### PL/SQL Program :

```
p Number := 1;
begin
  for i in 1...n
  loop
    p := p * n;
  end loop;
  return f;
end;
```

## DBMS LAB

**OUTPUT:**

```
SQL> select pow(2,5) from dual;  
POW(2,5)  
-----  
32
```

Q.4 Write a PL/SQL program to create a function which input a number and find the factorial of a number.

**PL/SQL Program :**

```
f Number := 1;  
begin  
    for i in 1...n  
    loop  
        f := f * i;  
    end loop;  
    return p;  
end;  
/
```

**OUTPUT:**

```
SQL> select factorial(5) from dual;  
FACTORIAL(5)  
-----  
120
```

## DBMS LAB

Q.5 Write a PL SQL program to create a function that accepts radius of a circle and returns the area of that circle.

PL/SQL Program :

```
area number;  
begin  
    area := 3.14 * m * m;  
    return area;  
end;  
/
```

OUTPUT:  
SQL> select printarea(4) from dual;  
PRINTAREA(4)  
-----  
50.24

Teacher I/C  
Mr. Raj Kumar Singh

## DBMS LAB

### Lab 10

#### Trigger

##### Topics

1. Explain Trigger.
2. Syntax for creating trigger in Oracle.
3. Use of Trigger.
4. Types of Triggers
  - (a) Row Level Trigger with applications
  - (b) Statement Level Trigger with applications.

##### Trigger

A database trigger is a stored procedure that is fired when an INSERT, UPDATE or DELETE statements is issued against the associate table. The name trigger is appropriate, as these are triggered (fired) whenever the above mentioned commands are executed. A trigger defines an action the database should take when some database related event occurs. A trigger can include SQL and PL/SQL statements to execute as a unit and can invoke other stored procedures. Triggers may use to provide referential integrity, to enforce complex business rules, or to audit changes to data. The code within a trigger, called the trigger body is made up of PL/SQL blocks.

A trigger is automatically executed without any action required by the user. A stored procedure on the other hand needs to be explicitly invoked. This is the main difference between a trigger and a stored procedure.

##### Uses of Database Trigger

Database triggers can be used for the following purposes.

1. To derive column values automatically.
2. To enforce complex integrity constraints.
3. To enforce complex business rules.
4. To customize complex security authorizations.
5. To maintain replicate tables.
6. To audit data modifications.

##### Parts of a trigger

A database trigger has 3 parts

1. Triggering event or statement.
2. Triggering constraint (optional)
3. Trigger action

##### 1. Triggering event or statement

A triggering event or statement is the SQL statement that causes a trigger to be fired. A triggering event can be an INSERT, DELETE or UPDATE statement for a specific table.

##### 2. Trigger Constraint or Restriction

A trigger restriction specifies a Boolean (logical) expression that must be TRUE for the trigger to fire. The trigger action is not executed if the trigger restriction evaluates to false. A trigger restriction is an option available for triggers that are fired for each row. Its function is to conditionally control the execution of a trigger. A trigger restriction is specified using a WHEN clause. It is an optional part of trigger.

##### 3. Trigger Action

## DBMS LAB

A trigger action is the procedure (PL/SQL block) that contains the SQL statements and PL/SQL code to be executed when a triggering statement is issued and the trigger restriction is issued and the trigger restriction evaluates to TRUE.

### Type of Triggers

A trigger's type is defined by the type of triggering transactions and by the level at which the trigger is executed. Oracle has the following types of triggers depending on the different applications.

1. Row Level Triggers.
2. Statement Level Triggers.
3. Before Triggers.
4. After Triggers.

#### 1. Row Level Triggers

Row Level Triggers execute once for each row in a transaction, the commands that enables the trigger. For e.g., if an update statement updates multiple rows of a table, a row trigger is fired once for each row affected by the update statement. If the triggering statement affects no rows, the trigger is not executed at all. Row Level Triggers are created using the for each row clause in the create trigger command.

### Applications

Consider a case when our requirement is to prevent updation of 100 records of emp then whenever update statement update records there must be PL/SQL block that will be fired automatically by update statement to check that it must not be 100, so we have to use row level triggers for that type of applications.

#### 2. Statement Level Triggers

Statement Level Triggers are triggered only once for each transaction, for e.g., when a update command update 15 rows, the commands contains in the trigger are executed only once, and not with every processed row. Statement level triggers are the default types of triggers created via the create trigger commands.

### Applications

Consider a case where our requirement is to prevent the delete operation during Sunday. For this whenever delete statement deletes records, there must be PL/SQL block that will be fired only once by delete statement to check that day must not be Sunday by referencing system date, so we have to use statement level trigger for which fires only once for above application.

### Before and After Triggers

Since triggers are executed by event, they may be set to occur immediately before or after those events. When a trigger is defined, you can specify whether the trigger must occur before or after the triggering event, i.e., INSERT, UPDATE or DELETE commands.

Before trigger execute the trigger action before the triggering statement. These types of triggers are commonly used in the following situation.

Before triggers are used when the trigger action should determine whether or not the triggering statement should be allowed to complete. By using a before trigger, you can eliminate unnecessary processing of the triggering statement. For e.g., to prevent deletion on Sunday for this we have to use statement level before trigger on delete statement.

Before triggers are used to derive specific column values before completing a triggering INSERT or UPDATE statement.

After trigger executes the trigger action after the triggering statement is executed. After triggers are used when you want the triggering statements to complete before executing the trigger action for e.g., to perform cascade delete

## DBMS LAB

operation, it means that user delete the record for one table, but the corresponding records in other tables are deleted automatically by a trigger which fired after the execution of delete statement issued by the user.

When combining the different types of triggering actions, there are mainly 12 possible valid trigger types available to use. The possible configurations are :-

1. Before Insert Row
2. Before Insert Statement
3. After Insert Row
4. After Insert Statement
5. Before Update Row
6. Before Update Statement
7. After Update Row
8. After Update Statement
9. Before Delete Row
10. Before Delete Statement
11. After Delete Row
12. After Delete Statement

### Syntax for creating a Trigger

```
CREATE [OR REPLACE] TRIGGER TRIGGER_NAME  
[BEFORE/ AFTER]  
[DELETE [OR] INSERT OR UPDATE of column_name,...]  
On table_name  
[referencing [OLD as old, NEW as new]]  
[FOR EACH ROW [WHEN CONDITION]]  
DECLARE  
    VAR DECLARATION  
    CONSTANT DECLARATION  
BEGIN  
    PL/SQL SUBPROGRAM BODY  
[EXCEPTION]  
    EXCEPTION HANDLING CODE  
END;  
/
```

## DBMS LAB

1. Write a PL/SQL program to create a trigger for the student table which makes the entry in student name column in upper case.

PL/SQL Program :

```
create trigger name - upper
before insert on student
for each row
begin
    new.sname := upper (:new.sname);
end;
```

OUTPUT :  
SQL > insert into student values (1, 'PRIYA', 5, 'IT', 90, 121);

1 row inserted

SQL > select \* from student;

ROLLNO SNAME

8 PRIYA

SEM BRANCH

5 IT

MARKS

90

PNO

121

## DBMS LAB

2. Write a PL/SQL program to create a trigger on the student table which shows the old values and new values of student name after updation on student name of student table.

PL/SQL Program :

```
create trigger info-update
  after update of sname on student
  for each row
begin
  dbms_output.put_line('Old student name
    is = ' || :old.sname);
  dbms_output.put_line('New student name
    is = ' || :new.sname);
end;
```

OUTPUT :

SQL > update student set sname = 'RAJKUMAR' where sname = 'PRIYA';

Old student name is = PRIYA

New student name is = RAJKUMAR

## DBMS LAB

3. Write a PL/SQL program to create a trigger on student table which copies the row which is deleted from student table into student1 table.

PL/SQL Program :

```
create trigger backup
after delete on student
for each row
begin
insert into student1 (:old.rollno :old.
:old.sname, :old.branch, :old.marks,
:old.pno);
dbms_output.putline (' old data stored
in student1 table');
end;
```

OUTPUT:  
SQL > delete from student where rollno = 1;

Old data stored in backup table

SQL > select \* from student1;

ROLLNO SNAME

1 RAM

SEM BRANCH

3 IT

MARKS

40

PNO

121

## DBMS LAB

4. Write a PL/SQL program to create a trigger that displays a message prior to an insert operation on the student table.

PL/SQL Program :

```
create table insert info
before insert on student
for each row
begin
dbms_output.put_line ('New
student are about to be
added');
end;
/
```

### OUTPUT :

SQL > insert into student values (7, 'PRIYA', 5, 'IT', 90, 121);

New students are about to be added

1 row created

## DBMS LAB

5. Write a PL/SQL program to create a trigger for every insert, update and delete operation on the student table, a row is added to the log table recording the date, user and action.

PL/SQL Program :

```
create or replace trigger trigger
before insert or delete or update on
student
declare
    val declare (20);
begin
    if inserting then
        val := 'INSERT';
    end if;
    if updation then
        val := 'UPDATE';
    end if;
    if deleting then
        val := 'DELETE';
    end if;
    insert into logtable value (sysdate,
                                user,
                                val);
end;
```

OUTPUT:  
SQL > insert into student values (8, 'AWANEESH', 5, 'IT', 48, 122);  
SQL> select \* from logtable;  
DT USR ACTION  
-----  
30-JUL-19 RAJKUMAR INSERT

## DBMS LAB

6. Write a PL/SQL program to create a trigger so that no operation can be performed on student table on Sunday.

PL/SQL Program :

```
create trigger SUNDAY
before insert or delete or update on
student
for each row
begin
if to_char (sysdate, 'd') = '1' then
RAISE_APPLICATION_ERROR (-20022,
'No operation
on Sunday');
end if;
end;
```

### OUTPUT:

```
SQL> delete from student;
delete from student
*
ERROR at line 1:
ORA-20022: No Operation on Sunday
ORA-06512: at "STUDENT_SUNDAY", line 3
ORA-04088: error during execution of trigger 'STUDENT_SUNDAY'
```

## DBMS LAB

7. Write a PL/SQL program to create a trigger which will verify that no record has the marks value greater than 90 in the student table.

PL/SQL Program :

```
create or replace trigger rec-check  
before update of marks on student  
for each row  
begin  
if > new. mark > 10: then  
RAISE_APPLICATION_ERROR (-20017,  
'Updated marks in less  
than 10');  
end if;  
end;
```

**OUTPUT:**

```
SQL> update student set marks = 95 where branch = 'IT';  
update student set marks = 95 where branch = 'IT'  
*
```

ERROR at line 1:

ORA-20000: Record is illegal

ORA-06512: at "REC\_CHECK", line 3

ORA-04088: error during execution of trigger 'REC CHECK'

## DBMS LAB

8. Write a PL/SQL program which verifies that the updated marks of student is greater than his/ her previous marks.

PL/SQL Program :

```
create or replace trigger
before update of mark on student
for each row
begin
```

```
if :new.marks < :old.marks then
    RAISE_APPLICATION_ERROR (-20017,
                                'Updated marks is less');
end if;
```

```
end;
```

### OUTPUT:

```
SQL> update student set marks = 30 where rollno = 1;
update student set marks = 30 where rollno = 1
```

```
*
```

```
ERROR at line 1:
```

```
ORA-20107: Updated marks is less
```

```
ORA-06512: at "TRIGGER7", line 3
```

```
ORA-04088: error during execution of trigger 'TRIGGER7'
```

Teacher NC

Mr. Raj Kumar Singh