SRM INSTITUTE OF SCIENCE AND TECHNOLOGY
COLLEGE OF ENGINEERING AND TECHNOLOGY
SCHOOL OF COMPUTING

DEPARTMENT OF NETWORKING AND COMMUNICATIONS

18CSC209J_Data Base Management System and Cloud Integration Services

Ex.No. 1

SQL Data Definition Language Commands

Aim:

To perform creation of table, alter, modify and drop column using Data Definition Language (DDL) Commands

FOR CREATING TABLE
CREATE TABLE *tablename* (column_name data_ type constraints, …)

Used to create a table by defining its structure, the data type and name of the various columns, the relationships with columns of other tables etc.

Q1) Create the tables DEPT and EMP as described below

DEPT

Column name	*Data type*	*Description*
DEPTNO	Number	Department number
DNAME	Varchar	Department name
LOC	Varchar	Department Location

EMP

Column name	*Data type*	*Description*
EMPNO	Number	Employee number
ENAME	Varchar	Employee name
JOB	Char	Designation
MGR	Number	Manager's EMP.No.
HIREDATE	Date	Date of joining
SAL	Number	Basic Salary
COMM	Number	Commission
DEPTNO	Number	Department Number

Q2) Confirm table creation

Q3) List name of the tables created by the user

Q4) Describe tables owned by the user

Q5) View distinct object types owned by the user

Q6) View tables, views, synonyms, and sequences owned by the user

ALTER TABLE
ß *Add Column*

ß *Drop Column*

ß *Modify Column*

Q7) Add new columns COMNT and MISCEL in DEPT table of character type.

Q8) Modify the size of column LOC by 15 in the DEPT table

Q9) Set MISCEL column in the DEPT table as unused

Q10) Drop the column COMNT from the table DEPT

Q11) Drop unused columns in DEPT table

Q12) Rename the table DEPT to DEPT12

Q13) Remove all the rows in the table DEPT12 (Presently no records in DEPT12)

ADDING COMMENTS TO THE TABLES
ß You can add comments to a table or column by using the COMMENT statement.
ß Comments can be viewed through the data dictionary views.
• ALL_COL_COMMENTS
• USER_COL_COMMENTS
• ALL_TAB_COMMENTS
• USER_TAB_COMMENTS
Syntax :

COMMENT ON TABLE table **IS** 'xxxxxxxxxx';

Q14) Add some comment to the table DEPT12 and also confirm the inclusion of comment

Q15) Delete the table DEPT12 from the database.

Q16) Confirm the removal of table DEPT12 from the database.

Ex.No. 2

SQL Data Manipulation Language Commands

Aim:

To perform manipulate records of table using Data Manipulation Language (DML) Commands

DML

A DML statement is executed when you:

- Add new rows to a table
- Modify existing rows in a table
- Remove existing rows from a table

A transaction consists of a collection of DML statements that form a logical unit of work.

Data for EMP table

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	19-APR-87	3000		20
7839	KING	PRESIDENT		17-NOV-81	5000		10
7844	TURNER	SALESMAN	7698	08-SEP-81	1500	0	30
7876	ADAMS	CLERK	7788	23-MAY-87	1100		20
7900	JAMES	CLERK	7698	03-DEC-81	950		30
7902	FORD	ANALYST	7566	03-DEC-81	3000		20
7934	MILLER	CLERK	7782	23-JAN-82	1300		10

Data for DEPT table

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

INSERT STATEMENT

Add new rows to a table by using the INSERT statement.

(i) INSERT INTO *table* VALUES*(value1, value2,..);*

- Only one row is inserted at a time with this syntax.
- List values in the default order of the columns in the table
- Enclose character and date values within single quotation marks.
- Insert a new row containing values for each column

(ii) INSERT INTO *table(column1, column2,..)*
VALUES*(value1, value2,..);*

- Rows can be inserted with NULL values either
  - by omitting column from the column list or
  - by specifying NULL in the value field.

(iii) INSERT INTO *table(column1, column2,...)*
VALUES*(&value1,& value2,..);*

Substitution variable(&) helps us to write an interactive script for inserting rows

Q1) Insert the rows of DEPT table using syntax (i)

Q2) Insert first & second rows of EMP table using syntax (ii)

Q3) Insert the remaining rows of EMP table using syntax (iii)

Q4) Create a table MANAGER with the columns *mgr-id, name, salary* and *hiredate*

Q5)
Insert values into the table MANAGER by copying the values from EMP table where the designation of the employee is 'MANAGER'

UPDATE STATEMENT

- Modify existing rows with the UPDATE statement.
- Update more than one row at a time, if required.
- All rows in the table are modified if you omit the WHERE clause

UPDATE *table* SET *column = value*, *column = value, ….* WHERE *condition*;

Q6)
Change the LOC of all rows of DEPT table by 'NEW YORK'

Q7)
Change the LOC='DALLAS' for deptno=20 in DEPT table.

DELETE STATEMENT

ß You can remove existing rows from a table by using the DELETE statement.
ß All rows in the table are deleted if you omit the WHERE clause.

DELETE FROM *table* WHERE *condition*;

Q8) Delete the rows from EMP table whose employee name = 'PAUL'

SELECT STATEMENT

To perform a query we use select statement

SELECT
[DISTINCT] {*, *column* [*alias*],...}
FROM
table;

o *Select* Clause determines what columns
o *From* Clause determines which table.
o *Where* Clause specifies the conditions

Q9) List all the columns and rows of the table DEPT

Q10) List the name of the employee and salary of EMP table

Q11) Without duplication, list all names of the department of DEPT table.

Q12) Find out the name of an employee whose EMPNO is 7788

Q13) As a copy of DEPT table, create DEPT1 table using select command.

Q14) List ename and sal of EMP table with the column headings NAME and SALARY

Ex.No. 3

SQL DCL and TCL commands

Aim:

To perform manipulate records of table using Select command, Data Control Language (DCL) and Transaction Control Language (TCL) Commands

3 A BASIC SELECT STATEMENTS

Arithmetic Operators

+ Addition
- Subtraction
*Multiplication
/ Division

Comparison Operators

= Equal to
<> Not Equal to
< Less than
> Greater than
<= Less than or equal to
>= Greater than or equal to
IN (List) Match any of list of values
LIKE Match a character pattern (% ‡ any no. of characters, - ‡ One Character)
IS NULL Is a null value
BETWEEN…AND… Between two values

Logical Operators

AND Returns TRUE if *both* component conditions are TRUE
OR Returns TRUE if *either* component condition is TRUE
NOT Returns TRUE if the following condition is FALSE

Concatenation Operator (||)

ß Concatenates the Columns of any data type.
ß A Resultant column will be a Single column.

Operator Precedence

Order Evaluated	*Operators*
1	Parenthesis
2	All Arithmetic Operators (Multiplication and Division followed by Addition and subtraction)
3	All Comparison Operators
4	NOT
5	AND
6	OR

Where Clause

- Specify the Selection of rows retrieved by the WHERE Clause

  SELECT ***column1, column2, …***
  FROM ***table***
  WHERE ***condition*;**

- The WHERE clause follows the FROM clause

Order by Clause

- Sort rows specified by the order ASC / DESC

  SELECT ***column1, column2, … …***
  FROM ***table***
  ORDER BY ***sort-column*** **DESC;**

- Sorts *table* by *sort-column* in descending order
- Omitting the keyword DESC will sort the table in ascending order

Note :

- AS Keyword between the column name and the actual alias name
- Date and character literal values must be enclosed within single quotation marks
- Default date format is 'DD-MON-YY'
- Eliminate duplicate rows by using the DISTINCT keyword

Q1) Update all the records of *manager* table by increasing 10% of their salary as bonus.
SQL>

Q2) Delete the records from *manager* table where the salary less than 2750.
SQL>

Q3) Display each name of the employee as “Name” and annual salary as “Annual Salary” (Note: Salary in *emp* table is the monthly salary)
SQL>

Q4) List concatenated value of name and designation of each employee.
SQL>

Q5) List the names of Clerks from *emp* table.
SQL>

Q6) List the Details of Employees who have joined before 30 Sept 81.
SQL>

Q7) List names of employees who's employee numbers are 7369,7839,7934,7788.
SQL>

Q8) List the names of employee who are not Managers.
SQL>

Q9) List the names of employees not belonging to dept no 30,40 & 10
SQL>

Q10) List names of those employees joined between 30 June 81 and 31 Dec 81.
SQL>

Q11) List different designations in the company.
SQL>

Q12) List the names of employees not eligible for commission.
SQL>

Q13) List names and designations of employee who does not report to anybody
SQL>

Q14) List all employees not assigned to any department.
SQL>

Q15) List names of employee who are eligible for commission.
SQL>

Q16) List employees whose name either start or end with 's'.
SQL>

Q17) List names of employees whose names have 'i' as the second character.
SQL>

Q18) Sort *emp* table in ascending order by *hire-date* and list *ename, job, deptno* and *hire-date*.
SQL>

Q19) Sort *emp* table in descending order by annual salary and list *empno, ename, jo* and *annual-salary*. (Note : Salary in *emp* table is the monthly salary)
SQL>

Q20) List *ename, deptno* and *sal* after sorting *emp* table in ascending order by *deptn* and then descending order by *sal*. (Note : Sorting by multiple coluns)
SQL>

3 B DCL and TCL Commands

***DCL** is Data Control Language statements. Some examples:*
GRANT - gives user's access privileges to database
REVOKE - withdraw access privileges given with the GRANT command

DATABASE TRANSACTIONS

- Begin when the first executable SQL statement is executed
- End with one of the following events:
  - COMMIT or ROLLBACK
  - DDL or DCL statement executes (automatic commit)
  - User exits
  - System crashes

TCL (Transaction Control Language)

- TCL Statements are COMMIT, ROLLBACK & SAVE POINT.
- State of Data Before COMMIT or ROLLBACK
  - The previous state of the data can be recovered.
  - The current user can review the results of the DML operations by using the SELECT statement.
  - Other users *cannot* view the results of the DML statements by the current user.
  - The affected rows are *locked*; other users cannot change the data within the affected rows.
- State of Data After COMMIT
  - Data changes are made permanent in the database.
  - The previous state of the data is permanently lost.
  - All users can view the results.
  - Locks on the affected rows are released; those rows are available for other users to manipulate.
  - All savepoints are erased.

Q1) Change LOC='CHICAGO' for deptno=30 in DEPT table and COMMIT the transaction

State of Data After ROLLBACK
o Discard all pending changes by using the ROLLBACK statement.
o Data changes are undone.
o Previous state of the data is restored.
o Locks on the affected rows are released.
Q2) Delete all the rows from EMP table and ROLLBACK the transaction.

Rolling Back to a Marker
o Create a marker within a current transaction by using <SAVEPOINT savepoint-name>
o Roll back to that marker by using <ROLLBACK TO savepoint-name>
Q3) Do the following operations one after another
a) Change LOC='BOSTON' for deptno=40 in DEPT table

b) Create SAVEPOINT in the name 'update_over'
SQL>

c) Insert another row in DEPT table with your won values
SQL>
d) Rollback the transaction upto the point 'update_over'
SQL>

Ex.No. 3

Inbuilt functions in SQL

Aim:

To perform manipulate records of table using inbuilt functions in SQL.

SQL functions are of two types

(i) Single row functions or scalar functions

- Returns only one value for every row queried in the table
- Can be used in Select clause and where clause
- It can be broadly classified into 5 categories
  - Date Functions
  - Character Functions
  - Conversion functions
  - Numeric functions
  - Miscellaneous functions

(ii) Group functions or multiple-row functions

Discussed in the next exercise (ie.; Ex. No.6)

Note : The exercises that follow mostly uses system table 'dual'. It is a table which is automatically created by Oracle along with the data dictionary. Dual table has one column defined to be of varchar2 type and contains only one row with value 'x'.

SCALAR FUNCTIONS

Q1) List the hiredate of employees who work in deptno 20 in a format like 'WEDNESDAY JANUARY 12, 1983'

(Hint: DAY : Day of the week, MONTH : Name of the month, DD: Day of the month, and YYYY : Year)

SQL>

Q2) Display the hiredate with time of employess who work in deptno 20.

SQL>

Q3) Each employee receives a salary review after every 150 days of service. Now list employee name, hiredate and first salary review date of each employee who work in dept no 20.

SQL>

Q4. Date Functions

Functions	Value Returned	Input	Output
add_months(d,n)	'n' months added to date 'd'.	Select add_months(sysdate,2) from dual;	
last_day(d)	Date corresponding to the last day of the month	Select last_day(sysdate) from dual;	
to_date(str,'format')	Converts the string ina given format into Oracle date.	Select to_date('10-02-09','dd-mm-yy') from dual;	
to_char(date,'format')	Reformats date according to format	Select to_char(sysdate,'dy dd mon yyyy') from dual;	
months_between(d1,d2)	No. of months between two dates	Select months_between(sysdate, to_date('10-10-07','dd-mm-yy')) from dual;	
next_day(d,day)	Date of the 'day' that immediately follows the date 'd'	Select next_day(sysdate,'wednesday') from dual;	
round(d,'format')	Date will be rounded to the nearest day.	Select round(sysdate,'year') from dual;	
		Select round(sysdate,'month') from dual;	
		Select round(sysdate,'day') from dual;	
		Select round(sysdate) from dual;	
trunc(d,'format');	Date will be truncated to the nearest day.	Select trunc(sysdate,'year') from dual;	
		Select trunc(sysdate,'month') from dual;	
		Select trunc(sysdate,'day') from dual;	
		Select trunc(sysdate) from dual;	
greatest(d1,d2,...)	Picks latest of list of dates	Select greatest(sysdate, to_date('02-10-06','dd-mm-yy'), to-date('12-07-12','dd-mm-yy')) from dual;	
Date Arithmetic	Add /Subtract no. of days to a date	Select sysdate+25 from dual;	
		Select sysdate-25 from dual;	
	Subtract one date from another, producing a no. of days	Select sysdate - to_date('02-10-06','dd-mm-yy') from dual;	

Q5. Character Functions

Functions	Value Returned	Input	Output
initcap(char)	First letter of each word capitalized	Select initcap('jesus christ') from dual;	
lower(char)	Lower case	Select lower('DIED') from dual;	
upper(char)	Upper case	Select upper('for Us') from dual;	
ltrim(char, set)	Initial characters removed up to the character not in set.	Select ltrim('lordourgod','lord') from dual;	
rtrim(char, set)	Final characters removed after the last character not in set.	Select rtrim('godlovesyou','you') from dual;	
translate(char, from, to)	Translate 'from' by 'to' in char.	Select translate('jack','j','b') from dual;	
replace(char, search, repl)	Replace 'search' string by 'repl' string in 'char'.	Select replace('jack and jue','j','bl') from dual;	
substr(char, m, n)	Substring of 'char' at 'm' of size 'n' char long.	Select substr('wages of sin is death',10,3) from dual;	

Q6. Conversion Functions

Functions	Value Returned	Input	Output
to_date(str,'format')	Converts the string ina given format into Oracle date.	Select to_date('10-02-09','dd-mm-yy') from dual;	
to_char(date,'format')	Reformats date according to format	Select to_char(sysdate,'dy dd mon yyyy) from dual;	
to_char(number,'format')	Display number value as a char.	Select to_char(12345.5,'L099,999.99') from dual;	
to_number(char)	Char string to number form	Select to_number('123') from dual;	

Q7. Numeric Functions

Functions	Value Returned	Input	Output
Abs(n)	Absolute value of n	Select abs(-15) from dual;	
Ceil(n)	Smallest int >= n	Select ceil(33.645) from dual;	
Cos(n)	Cosine of n	Select cos(180) from dual;	
Cosh(n)	Hyperbolic cosine of n	Select cosh(0) from dual;	
Exp(n)	e^n	Select exp(2) from dual;	
Floor(n)	Largest int <= n	Select floor(100.2) from dual;	
Ln(n)	Natural log of n (base e)	Select ln(5) from dual;	
Log(b,n)	Log n base b	Select log(2,64) from dual;	
Mod(m,n)	Remainder of m divided by n	Select mod(17,3) from dual;	
Power(m,n)	m power n	Select power(5,3) from dual;	
Round(m,n)	m rounded to n decimal places	Select round(125.67854,2) from dual;	
Sign(n)	If n<0, -1 if n=0, 0 otherwise 1.	Select sin(-19) from dual;	
Sin(n)	Sin of n	Select sin(90) from dual;	
Sinh(n)	Hyperbolic sin of n	Select sinh(45) from dual;	
Sqrt(n)	Square root of n	Select sqrt(7) from dual;	
Tan(n)	Tangent of n	Select tan(45) from dual;	
Tanh(n)	Hyperbolic tangent of n	Select tanh(60) from dual;	
Trunc(m,n)	m truncated to n decimal places	Select trunc(125.5764,2) from dual;	

Q8. Miscellaneous Functions

Functions	Value Returned	Input	Output
Uid	User id	Select uid from dual;	
User	User name	Select user from dual;	
Vsize(n)	Storage size of v	Select vsize('hello') from dual;	
NVL(exp1,exp2)	Returns exp1 if not null, otherwise returns exp2.	Select nvl(comm,50) from emp where empno=7369;	

GROUP FUNCTIONS

Common Group Functions

- AVG : Average value of a set
- COUNT : Numbers of non null values
- MAX : Maximum of a set
- MIN : Minimum of a set
- STDDEV : Standard Deviation of a set
- SUM : Sum of a set
- VARIANCE : Variance of a set

Syntax :

SELECT *column, group_function(column)*
FROM *table*
[WHERE *condition*]
[GROUP BY *group_column_or_expression*]
[HAVING *group_condition*]
[ORDER BY *column*];

- Group functions ignore null values
- *Group by* Clause is used to modularize rows in a table into smaller groups
- Columns that are not a part of the Group Functions should be included in the Group by clause
- Any column or expression in the SELECT list that is not an aggregate function must be in the GROUP BY clause
- Group Functions cannot be placed in the where clause
- HAVING clause is to restrict groups Groups satisfying the HAVING condition are displayed

- Order of evaluation of the clauses :
  - WHERE clause
  - GROUP BY clause
  - HAVING clause

Q9) Find number of rows in the table EMP

SQL >

Q10) Find number of designations available in EMP table.
SQL>

Q11) Find number of employees who earn commission in EMP table.
SQL>

Q12) What is the difference between the following queries
SQL > select *count(comm)* from *emp*;
SQL > select *count(nvl(comm,0))* from *emp*;

Q13) Find the total salary paid to the employees.
SQL>

Q14) Find maximum, minimum and average salary in EMP table.
SQL>

Q15) Find number of employees who work in department number 30
SQL> select *count(*)* from *emp*
where deptno=30;

Q16) Find the maximum salary paid to a 'CLERK'
SQL>

Q17) List the department numbers and number of employees in each department
SQL>

Q18) List the jobs and number of employees in each job. The result should be in the descending order of the number of employees.
SQL> select *job, count(*)* from *emp*
group by *job*
order by *count(*)* desc;

Q19) List the total salary, maximum and minimum salary and average salary of the employees jobwise.

SQL>

Q20) List the total salary, maximum and minimum salary and average salary of the employees jobwise, for department 20 and display only those rows having an average salary > 1000.

```
SQL> select job,sum(sal), max(sal), min(sal), avg(sal)
     from emp
     group by job, deptno
     having deptno=20 and avg(sal) > 1000;
```

Q21) List the job and total salary of employees jobwise, for jobs other than 'PRESIDENT' and display only those rows having total salary > 5000.

SQL>

Q22) List the job, number of employees and average salary of employees jobwise. Display only the rows where the number of employees in each job is more than two.

SQL>

Ex.No. 5

ER Model

Aim:

To construct an Entity Relationship diagram for the "_____".

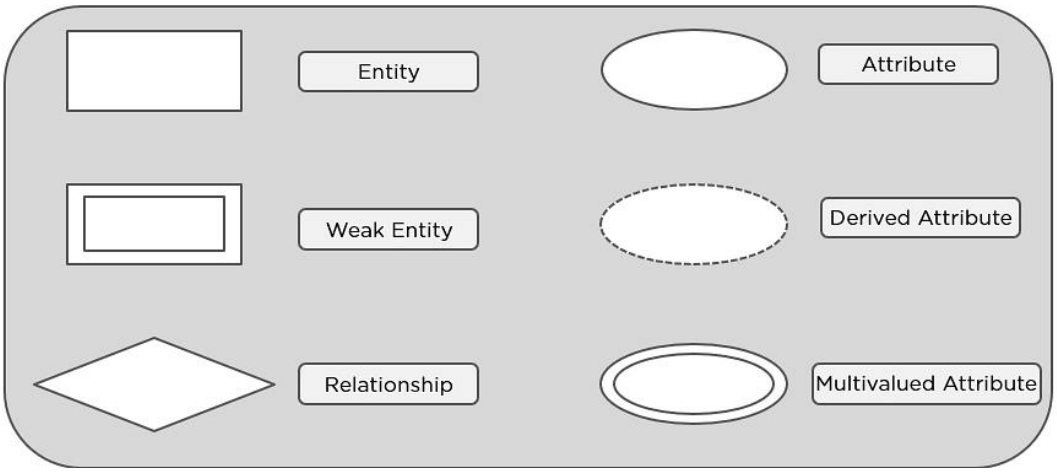
Entity Relationship model

ER Model stands for Entity Relationship Model is a high-level conceptual data model diagram. ER model helps to systematically analyze data requirements to produce a well-designed database. The ER Model represents real-world entities and the relationships between them.

ER Modeling helps to analyze data requirements systematically to produce a well-designed database. So, it is considered a best practice to complete ER modeling before implementing our database.

Symbols Used in ER Diagrams

- Rectangles: This Entity Relationship Diagram symbol represents entity types
- Ellipses: This symbol represents attributes
- Diamonds: This symbol represents relationship types
- Lines: It links attributes to entity types and entity types with other relationship types
- Primary key: Here, it underlines the attributes
- Double Ellipses: Represents multi-valued attributes



Entities

An entity can be either a living or non-living component. It showcases an entity as a rectangle in an ER diagram.

Weak Entity

An entity that makes reliance over another entity is called a weak entity

Attribute

An attribute exhibits the properties of an entity. You can illustrate an attribute with an oval shape in an ER diagram.

Key Attribute

Key attribute uniquely identifies an entity from an entity set. It underlines the text of a key attribute.

Composite Attribute

An attribute that is composed of several other attributes is known as a composite attribute. An oval showcases the composite attribute, and the composite attribute oval is further connected with other ovals.

Multivalued Attribute

Some attributes can possess over one value, those attributes are called multivalued attributes. The double oval shape is used to represent a multivalued attribute.

Derived Attribute

An attribute that can be derived from other attributes of the entity is known as a derived attribute. In the ER diagram, the dashed oval represents the derived attribute.

Relationship

The diamond shape showcases a relationship in the ER diagram. It depicts the relationship between two entities.

One-to-One Relationship

When a single element of an entity is associated with a single element of another entity, it is called a one-to-one relationship.

One-to-Many Relationship

When a single element of an entity is associated with more than one element of another entity, it is called a one-to-many relationship

Many-to-One Relationship

When more than one element of an entity is related to a single element of another entity, then it is called a many-to-one relationship.

Many-to-Many Relationship

When more than one element of an entity is associated with more than one element of another entity, this is called a many-to-many relationship.

Steps to draw ER diagram:

- Identify all the Entities. Embed all the entities in a rectangle and label them properly.
- Identify relationships between entities and connect them using a diamond in the middle, illustrating the relationship. Do not connect relationships with each other.
- Connect attributes for entities and label them properly.
- Eradicate any redundant entities or relationships.
- Make sure your ER Diagram supports all the data provided to design the database.

Ex.No. 6

Nested Queries

Aim:

To perform manipulate records of table using nested queries in SQL.

- Nesting of queries, one within the other is termed as sub query.

Syntax

```
SELECT    select_list
FROM      table
WHERE     expr operator  ( SELECT    select_list
                           FROM      table);
```

- The subquery (inner query) executes once before the main query.
- The result of the subquery is used by the main query (outer query).

Guidelines for Subqueries

- Enclose subqueries in parentheses.
- Place subqueries on the right side of the comparison operator.
- Do not add an ORDER BY clause to a subquery.
- Use single-row operators with single-row subqueries.
- Use multiple-row operators with multiple-row subqueries.

Single-Row Subqueries

- Return only one row
- Use single-row comparison operators (ie; relational operators)

Multiple-Row Subqueries

- Return more than one row
- Use multiple-row comparison operators

Operator	***Meaning***
IN	Equal to **any** member in the list
ANY	Compare value to **each value** returned by the subquery
ALL	Compare value to **every value** returned by the subquery

Note:

'**=any**' is equivalent to '**in**'
'**!=all**' is equivalent to '**not in**'

Q1) List the name of the employees whose salary is greater than that of employee with empno 7566.

```
SQL> select ename from employee
        where  sal > (select sal from employee
                        where  empno=7566);
```

Q2) List the name of the employees whose job is equal to the job of employee with empno 7369 and salary is greater than that of employee with empno 7876.

```
SQL>
```

Q3) List the *ename,job,sal* of the employee who get minimum salary in the company.

```
SQL> select ename, job, sal from employee
        where  sal = (select    min(sal) from employee);
```

Q4) List *deptno & min(salary)* departmentwise, only if *min(sal)* is greater than the *min(sal)* of *deptno* 20.

```
SQL>  select  deptno, min(sal) from employee
        group by deptno
        having min(sal) > (select min(sal) from employee
                             where deptno = 20);
```

Q5) List *empno, ename, job* of the employees whose *job* is not a 'CLERK' and whose *salary* is less than at least one of the salaries of the employees whose *job* is 'CLERK'.

```
SQL> select  empno, ename, job  from employee
        where sal < any (select sa  from employee
                    where job = 'CLERK')
                and job <> ' CLERK ';
```

Q6) List *empno, ename, job* of the employees whose salary is greater than the average salary of each department.

SQL>

Q7) Display the *name, dept. no, salary,* and *commission* of any employee whose *salary* and *commission* matches both the *commission* and *salary* of any employee in department 30.

```
SQL> select   ename, deptno, sal, comm
     from     employee
     where    (sal, nvl(comm,-1)) in  ( select sal, nvl(comm,-1)
                                        from   employee
                                        where  deptno = 30);
```

Q8) List *ename sal, deptno, average salary* of the dept where he/she works, if salary of the employee is greater than his/her department average salary.

```
SQL> select  a.ename, a.sal, a.deptno, b.salavg
     from    employee a,    ( select   deptno, avg(sal) salavg
                              from     employee
                              group by deptno) b
     where   a.deptno = b.deptno
     and     a.sal > b.salavg;
```

Q9) Execute and Write the output of the following query in words.

```
SQL> with summary as
     (select  dname,sum(sal) as dept_total   from employee a , department b
      where   a.deptno = b.deptno
      group by dname);
      select dname,dept_total  from summary
      where dept_total > (select sum(dept_total)*1/3  from summary)
      order by dept_total desc;
```

Q6) List *empno, ename, job* of the employees whose salary is greater than the average salary of each department.

SQL>

Q7) Display the *name, dept. no, salary,* and *commission* of any employee whose *salary* and *commission* matches both the *commission* and *salary* of any employee in department 30.

```
SQL> select   ename, deptno, sal, comm
     from     employee
     where    (sal, nvl(comm,-1)) in  ( select sal, nvl(comm,-1)
                                        from   employee
                                        where  deptno = 30);
```

Q8) List *ename sal, deptno, average salary* of the dept where he/she works, if salary of the employee is greater than his/her department average salary.

```
SQL> select  a.ename, a.sal, a.deptno, b.salavg
     from    employee a,    ( select   deptno, avg(sal) salavg
                              from     employee
                              group by deptno) b
     where   a.deptno = b.deptno
     and     a.sal > b.salavg;
```

Q9) Execute and Write the output of the following query in words.

```
SQL> with summary as
     (select  dname,sum(sal) as dept_total   from employee a , department b
      where   a.deptno = b.deptno
      group by dname);
      select dname,dept_total  from summary
      where dept_total > (select sum(dept_total)*1/3  from summary)
      order by dept_total desc;
```

Q10) List *ename, job, sal* of the employees whose salary is equal to any one of the salary of the employee 'SCOTT' and 'WARD'.

SQL>

Q11) List *ename, job, sal* of the employees whose salary and job is equal to the employee 'FORD'.

SQL>

Q12) List *ename, job, deptno, sal* of the employees whose job is same as 'JONES' and *salary* is greater than the employee 'FORD'.

SQL>

Q13) List *ename, job* of the employees who work in *deptno* 10 and his/her *job* is any one of the job in the department 'SALES'.

SQL>

Q14) Execute the following query and write the result in word

```
SQL> select job,ename,empno,deptno from emp s
     where exists (select * from emp
                   where s.empno=mgr)
     order by empno;
```

Ex.No. 7

Join Queries

Aim:

To perform manipulate records of table using nested queries in SQL.

Joins

- Used to combine the data spread across tables

Syntax

```
SELECT    table1.column, table2.column
FROM      table1, table2
WHERE     table1.column1 = table2.column2;
```

- A JOIN Basically involves more than one Table to interact with.
- Where clause specifies the JOIN Condition.
- Ambiguous Column names are identified by the Table name.
- If join condition is omitted, then a **Cartesian product** is formed. That is all rows in the first table are joined to all rows in the second table

Types of Joins

- Inner Join (Simple Join) : It retrieves rows from 2 tables having a common column.
  - Equi Join : A join condition with relationship = .
  - Non Equi Join : A join condition with relationship other than = .
- Self Join : Joining of a table to itself
- Outer Join : Returns all the rows returned by simple join as well as those rows from one table that do not match any row from the other table. The symbol (+) represents outer joins.

Create a table Student and Course and apply all types of join and wrte the output

Student

ROLL_NO	NAME	ADDRESS	Age
1	Ram	Delhi	18
2	RAMESH	GURGAON	18
3	SUJIT	ROHTAK	20
4	SURESH	Delhi	18

COURSE_ID	ROLL_NO
1	1
2	2
2	3
3	4

Q1) List *empno, ename, deptno* from *emp* and *dept* tables.

SQL>

Q2) Create a table *Salgrade* with the following data .

Grade	Losal	Hisal
1	700	1400
2	1401	2000
3	2001	5000
4	5001	9999

Now, list *ename, sal* and *salgrade* of all employees.

SQL>

Q3) List *ename, deptno* and *deptname* from *emp* and *dept* tables, including the rows of *emp* table that does not match with any of the rows in *dept* table.

SQL>

Q4) List *ename, deptno* and *deptname* from *emp* and *dept* tables, including the rows of *dept* table that does not match with any of the rows in *emp* table.

SQL>

Q5) List the names of the employee with name of his/her manager from *emp* table.

SQL>

Ex.No.8

Set Operators and Views

Aim:

To perform manipulate records of table using Set operators and Views in SQL.

Set operators

- Set operators combine the results of two queries into a single.

Operator	Function
Union	Returns all distinct rows selected by either query
Union all	Returns all rows selected by either query including duplicates
Intersect	Returns only rows that are common to both the queries
Minus	Returns all distinct rows selected only by the first query and not by the second.

Q1) Create the following tables :

depositor(cus_name,acno) & borrower(cus_name,loanno)

SQL>

SQL>

Q2) List the names of distinct customers who have either loan or account

SQL>

Q3) List the names of customers (with duplicates) who have either loan or account

SQL> (select *cus_name* from *borrower*)
union all (select *cus_name* from *depositor*)

Q4) List the names of customers who have both loan and account

SQL>

Q5) List the names of customers who have loan but not account

SQL>

VIEWS

- An Imaginary table contains no data and the tables upon which a view is based are called base tables.
- Logically represents subsets of data from one or more tables

Advantages of view

- To restrict database access
- To make complex queries easy
- To allow data independence
- To present different views of the same data

Syntax

CREATE [OR REPLACE] VIEW *view* [*col1 alias*, *col2 alias*,...]
AS *subquery*
[WITH CHECK OPTION [CONSTRAINT *constraint*]]
[WITH READ ONLY]

- You embed a subquery within the CREATE VIEW statement.
- The subquery can contain complex SELECT syntax.
- The subquery cannot contain an ORDER BY clause.

Q1) Create a view *empv10* that contains *empno, ename, job* of the employees who work in *dept* 10. Also describe the structure of the view.

SQL> create view *empv10* as
select *empno, ename, job* from *emp*
where *deptno=10;*

SQL> desc *empv10;*

Q2) Create a view with column aliases *empv30* that contains *empno, ename, sal* of the employees who work in *dept* 30. Also display the contents of the view.

SQL >

SQL> select * from *empv30;*

Rules for Performing DML Operations on a View

- You can perform DML operations on simple views.
- You **cannot remove** a row/ **modify** data/ **add** data if the view contains the following
  - Group functions
  - A GROUP BY clause
  - The DISTINCT keyword
- You cannot modify data in a view if it contains columns defined by expressions or it contains ROWNUM pseudo column
- You cannot add data if any NOT NULL columns in the base tables that are not selected by the view

Q3) Update the view *empv10* by increasing 10% salary of the employees who work as 'CLERK'. Also confirm the modifications in *emp table.*

SQL > update *empv10* set *sal = sal+0.10*sal*
where *job='CLERK'*;

SQL > select *empno,ename,job,sal* from emp;

Q4) Modify the view *empv10* which contains the data *empno, ename, job, sal.* Add an alias for each column name.

SQL > create or replace view *empv10*
***(employee_no, employee_name, job, salary)* as**
select *empno, ename, job,sal* from *emp* where *deptno=10;*

Q5) Using *emp* table, **c**reate a view *pay* which contains *ename, monthly_sal, annual_sal, deptno.*

SQL>

Q6) Create a view *dept_stat* which contains *department no., department name, minimum salary, maximum salary, total salary.*

SQL>

With check option

- You can ensure that DML on the view stays within the domain of the view by using the WITH CHECK OPTION.

Q7) Execute the following query and then try to delete the row with dept no 20. Now write in words that you understand

SQL> create or replace view *empv20*
as select * from *emp* where *deptno = 20*
with check option constraint *empv20_ck;*

Denying DML Operations

- You can ensure that no DML operations occur by adding the WITH READ ONLY option to your view definition.

Q8) Create a view *empv10* with all the details of employees who work in dept no. 10. Also ensure that no DML operations can be done with the view.

SQL> create or replace view *empv10*
as select * from *emp* where *deptno = 20*
with read only;

Deleting Views

Syntax

DROP VIEW view_*name*;

Q9) Delete the view *empv20*.

SQL>

Ex.No.9

PL/SQL Conditional and Iterative Statements

Aim:

To manipulate Conditional and Iterative statements in PL/SQL.

PL/SQL

PL/SQL is a combination of SQL along with the procedural features of programming languages.

Block Structure

PL/SQL code is grouped into structures called blocks.

The PL/SQL block divided into three section: declaration section, the executable section and the exception section

The structure of a typical PL/SQL block is shown in the listing:

```
declare
        < declaration section >
begin
        < executable  commands>
exception
        <exception handling>
end;
```

Declaration Section :

Defines and initializes the variables and cursor used in the block

Executable commands :

Uses flow-control commands (such as IF command and loops) to execute the commands and assign values to the declared variables

Exception handling :

Provides handling of error conditions

Creating and Executing PL/SQL Programs

Edit your PL/SQL program in a text editor as text file, and save with '.sql' extension.

Execute the following command once for a session to get displayed the output.

SQL> *set serveroutput on;*

Now execute the program using the following command.

SQL> *start* filename;

(or)

SQL> @filename;

Note : Give absolute path of the filename if you saved the file in some directory.

Ex.

SQL> *start* z:\plsql\ex11; (or) SQL> @ z:\plsql\ex11;

Control Structures

(i) IF Statements

There are three forms of IF statements: IF-THEN, IF-THEN-ELSE, and IF THEN-ELSIF. The third form of IF statement uses the keyword ELSIF (NOT ELSEIF) to introduce additional conditions, as follows:

```
IF condition1 THEN
sequence_of_statements1;
ELSIF condition2 THEN
sequence_of_statements2;
```

```
ELSE
sequence_of_statements3;
END IF;
```

(ii) LOOP and EXIT Statements

There are three forms of LOOP statements. They are LOOP, WHILE-LOOP, and FOR-LOOP.

LOOP

The simplest form of LOOP statement is the basic (or infinite) loop, which encloses a sequence of statements between the keywords LOOP and END LOOP, as follows:

```
LOOP
sequence_of_statements3;
...
END LOOP;
```

With each iteration of the loop, the sequence of statements is executed, then control resumes at the top of the loop. If further processing is undesirable or impossible, you can use the EXIT statement to complete the loop. You can place one or more EXIT statements anywhere inside a loop, but nowhere outside a loop. There are two forms of EXIT statements: EXIT and EXIT-WHEN.

(iii) WHILE-LOOP

The WHILE-LOOP statement associates a condition with a sequence of statements enclosed by the keywords LOOP and END LOOP, as follows:

```
WHILE condition LOOP
sequence_of_statements;
...
END LOOP;
```

(iv) FOR-LOOP

FOR loops iterate over a specified range of integers. The range is part of an iteration scheme, which is enclosed by the keywords FOR and LOOP.

```
FOR counter IN [REVERSE] lower_bound..upper_bound LOOP
sequence_of_statements;
...
END LOOP;
```

The lower bound need not be 1. However, the loop counter increment (or decrement) must be 1. PL/SQL lets you determine the loop range dynamically at run time, as the following example shows:

```
SELECT COUNT(empno) INTO emp_count FROM emp;
FOR i IN 1..emp_count LOOP
...
END LOOP;
```

The loop counter is defined only within the loop

(v) GOTO and NULL statements

The NULL statement can make the meaning and action of conditional statements clear and so improve readability.

```
BEGIN
...
```

```
GOTO insert_row;
...
<<insert_row>>
INSERT INTO emp VALUES ...
END;
```

A GOTO statement cannot branch into an IF statement, LOOP statement, or subblock. A GOTO statement cannot branch from one IF statement clause to another. A GOTO statement cannot branch out of a subprogram. Finally, a GOTO statement cannot branch from an exception handler into the current block.

Q1)Write a PL/SQL program to find the largest of three numbers.

```
Declare
      a number;
      b number;
      c number;
Begin
      dbms_output.put_line('Enter a:');
      a:=&a;
      dbms_output.put_line('Enter b:');
      b:=&b;
      dbms_output.put_line('Enter c:');
      c:=&c;
      dbms_output.put_line('NUMBERS');
      IF a>b AND a>c THEN
            dbms_output.put_line('A is Maximum');
      ELSIF (b>a) AND (b>c) then
            dbms_output.put_line('B is Maximum');
      ELSE
            dbms_output.put_line('C is Maximum');
      END IF;
End;
/
```

Q2) Write a PL/SQL program to swap two numbers
Q3) Write a PL/SQL program to find the factorial of a given number.
Q4) Write a PL / SQL program to check whether the given number is prime or not.
Q5) Write a PL/SQL Block to modify the department name of the department 71 if it is not ‘HRD’.

```
Declare
deptname dept.dname%type;
Begin

-- complete the block
End;
/
```

Ex.No. 10

PL/SQL Procedures

Aim:

To manipulate Stored procedures in PL/SQL.

PL/SQL Procedure

A PL/SQL procedure is a reusable unit that encapsulates specific business logic of the application. Technically speaking, a PL/SQL procedure is a named block stored as a schema object in the Oracle Database.

The following illustrates the basic syntax of creating a procedure in PL/SQL:

```
CREATE OR REPLACE PROCEDURE <procedure_name>
(<variable_name>IN/OUT/IN OUT <datatype>, <variable_name>IN/OUT/IN OUT
<datatype>,...)  IS/AS
variable/constant declaration;
BEGIN
-- PL/SQL subprogram body;
EXCEPTION
-- Exception Handling block ;
END <procedure_name>;
```

PL/SQL procedure header

A procedure begins with a header that specifies its name and an optional parameter list. Each parameter can be in either IN, OUT, or INOUT mode. The parameter mode specifies whether a parameter can be read from or write to.

- **IN** - An IN parameter is read-only. One can reference an IN parameter inside a procedure, but cannot change its value. Oracle uses IN as the default mode. It means that if you don't specify the mode for a parameter explicitly, Oracle will use the IN mode.
- **OUT** - An OUT parameter is writable. Typically, one set a returned value for the OUT parameter and return it to the calling program. Note that a procedure ignores the value that supply for an OUT parameter.
- **INOUT -** An INOUT parameter is both readable and writable. The procedure can read and modify it.

The **OR REPLACE** option allows you to overwrite the current procedure with the new code.

PL/SQL procedure body

Similar to an anonymous block, the procedure body has three parts. The executable part is mandatory whereas the declarative and exception-handling parts are optional. The executable part must contain at least one executable statement.

1) Declarative part

In this part, one can declare variables, constants, cursors, etc. Unlike an anonymous block, a declaration part of a procedure does not start with the DECLARE keyword.

2) **Executable part**

This part contains one or more statements that implement specific business logic. It might contain only a NULL statement.

3) Exception-handling part
This part contains the code that handles exceptions.

EXAMPLE:

```
CREATE OR REPLACE PROCEDURE greetings
AS BEGIN
   dbms_output.put_line('Hello World!'); END; /
```

To Execute Query from sqlplus terminal

```
EXECUTE greetings;

Hello World

PL/SQL procedure successfully completed.
```

Q1) Procedure to add Two numbers

```
CREATE OR REPLACE PROCEDURE sumTwoNum(num1 IN number, num2 IN
number) IS
DECLARE
 tot number;

BEGIN
   tot := num1 + num2;
END;
/
```

PL/SQL Block that invokes the Procedure `sumTwoNum`

```
set serveroutput on;
DECLARE
x number;
y number;
BEGIN
x := &x;
y := &y;
Sum(x,y);
END;
/
```

Q2) Write a PL/SQL Procedure to update the salary of employee whose job is Clerk.

```
create or replace procedure emp_sal_update
 IS
 BEGIN
 update emp
 set sal=sal+sal*0.10
 where job = 'Clerk';
 END;
```

Q3) Write a PL/SQL Procedure to find the number of managers in the employee table

Q4) Write a PL/SQL Procedure to display the details of employees from the emp table whose name are starting with ‘A’ and ‘M’.

Ex.No. 10

PL/SQL Functions

Aim:

To manipulate Functions in PL/SQL.

PL/SQL Function

A PL/SQL **function** is a reusable program unit stored as a schema object in the Oracle Database. A function is **same as a procedure except that it returns a value.** We can call the function anywhere within its scope inside the program, such as in Boolean expressions, assignment statements to assign the return value of the function to some variable or even inside the SQL statement, and queries such as inside where clause, etc.

The following illustrates the basic syntax of creating a function in PL/SQL:

```
CREATE [OR REPLACE] FUNCTION <function_name>
  ( (<variable name> IN <datatype>,
<variable name> IN <datatype>,…)

  RETURN <datatype>

IS | AS

variable/constant declaration;
BEGIN
-- PL/SQL subprogram body;
EXCEPTION
-- Exception Handling block ;
END <procedure_name>;
```

where

- OR REPLACE specifies the function that is to replace an existing function if present.
- type specifies the PL/SQL type of the parameter.
- The body of a function must return a value of the PL/SQL type specified in the RETURN clause.

The function header has the function name and a RETURN clause that specifies the datatype of the returned value.

The function body is the same as the procedure body which has three sections: declarative section, executable section, and exception-handling section.

- The declarative section is between the IS and BEGIN keywords. It is where you declare variables, constants, cursors, and user-defined types.

- The executable section is between the BEGIN and END keywords. It is where you place the executable statements. Unlike a procedure, you must have at least one RETURN statement in the executable statement.
- The exception-handling section is where you put the exception handler code.

In these three sections, only the executable section is required, the others are optional.

Q1) Function to add Two numbers

```
create or replace function adder(n1 in number, n2 in number)
return number
is
n3 number(8);
begin
n3 :=n1+n2;
return n3;
end;
/
```

PL/SQL Block that invokes the function `adder`

```
DECLARE
  n3 number(2);
BEGIN
  n3 := adder(11,22);
  dbms_output.put_line('Addition is: ' || n3);
END;
/
```

Output:

Addition is: 33

Statement processed.

0.05 seconds

Q2) Write a PL/SQL function to find factorial of a number.

```
SQL> create or replace function fact(n number)
return number is
i number(10);
f number:=1;
begin
for i in 1..N loop
f:=f*i;
end loop;
```

```
return f;
end;
 /
```

Q3) Write a PL/SQL function to check whether given number is prime or not. Return boolean value.

Q4) Write a PL/SQL function to display the salary of the employee from emp table by getting empid as input.

Ex.No. 12

PL/SQL Cursors

Aim:

To manipulate data using PL/SQL cursors.

Cursor

A cursor is a temporary work area created in the system memory when a SQL statement is executed. A cursor contains information on a select statement and the rows of data accessed by it. This temporary work area is used to store the data retrieved from the database, and manipulate this data. A cursor can hold more than one row, but can process only one row at a time.

There are two types of cursors in PL/SQL. They are Implicit cursors and Explicit cursors.

1. Implicit cursors

These are created by default when DML statements like, INSERT, UPDATE, and DELETE statements are executed. Oracle provides few attributes called as implicit cursor attributes to check the status of DML operations. The cursor attributes available are %FOUND, %NOTFOUND, %ROWCOUNT, and %ISOPEN.

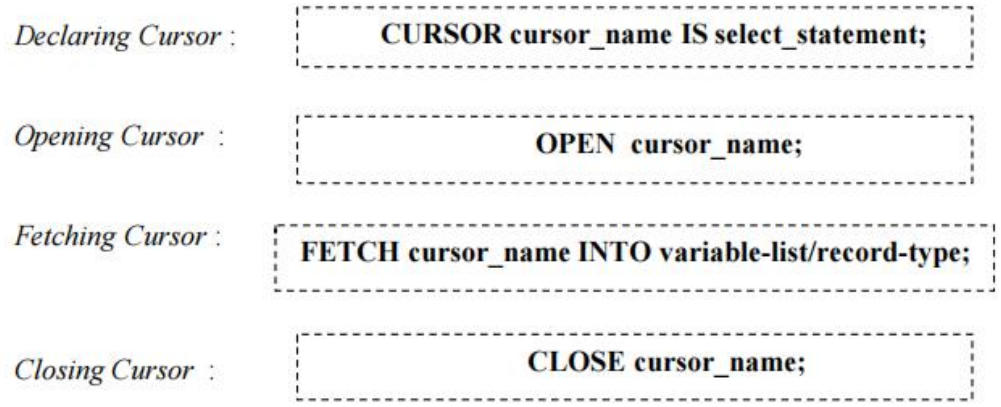
Implicit Cursor Attributes

- **%FOUND** - The return value is TRUE, if the DML statements like INSERT, DELETE and UPDATE affect at least one row or if SELECT ….INTO statement return at least one row.
- **%NOTFOUND -** The return value is FALSE, if DML statements affect at least one row or if SELECT. …INTO statement return at least one row.
- **%ROWCOUNT -** Return the number of rows affected by the DML operations

2. Explicit cursors

They must be created when you are executing a SELECT statement that returns more than one row. Even though the cursor stores multiple records, only one record can be processed at a time, which is called as current row. When you fetch a row the current row position moves to next row.

There are four steps in using an Explicit Cursor.



Explicit Cursor Attributes

- **%FOUND -** TRUE, if fetch statement returns at least one row.
- **%NOTFOUND** - TRUE, , if fetch statement doesn't return a row.
- **%ROWCOUNT -** The number of rows fetched by the fetch statement.
- **%ISOPEN** - TRUE, if the cursor is already open in the program.

Q1. Write a PL/SQL Block, to update salaries of all the employees who work in deptno 20 by 15%. If none of the employee's salary are updated display a message *'None of the salaries were updated'*. Otherwise display the total number of employee who got salary updated.

```
Declare
        num number(5);
Begin
    update emp set sal = sal + sal*0.15 where deptno=20;
    if SQL%NOTFOUND then
      dbms_output.put_line('none of the salaries were updated');
    elsif SQL%FOUND then
        num := SQL%ROWCOUNT;
      dbms_output.put_line('salaries for ' || num || 'employees are
updated');
    end if;
End;
```

Q2. Using cursors, write a PL/SQL block to display Employee Number, Name and Department Number from the Emp database.

```
DECLARE
CURSOR C1 IS SELECT EMPNO,ENAME ,DEPTNO FROM EMP;
EMPNUM EMP.EMPNO%TYPE;
EMPNAME EMP.ENAME%TYPE;
DEPTNUM EMP.DEPTNO%TYPE;
BEGIN OPEN C1;
LOOP FETCH C1 INTO EMPNUM,EMPNAME,DEPTNUM;
if c1%notfound then
        exit;
else
        dbms_output.put_line(EMPNUM||'                '||EMPNAME||'
'||DEPTNUM);
end if;
END LOOP;
end;
```

Q3. Using cursors, write a PL/SQL block to find the name and salary of first five highly paid employees.