Task 5: Perform queries using joins, sub query, set operations, group by and having.

Scenario : Many times you will find data are interdependent on each other. Thus Mohan realized to see its dependency by applying different queries & sub-queries resulting into output that expected.

Solve the following queries using sub queries:-

1) Find the product_no and description of non moving products i.e. product not being sold.
2) Find the customer name, address1, address2, city and pincode for the client who has placed order no 'O1901'.
3) Find out if the product 'Mouse' has been ordered by any client and print the client_no, name to whom it was sold.

Queries using Having and Group By Clause:

1) Print the description and total qty sold for each product.
2) Find the value of each product sold.
3) Find out the sum total of all the billed orders for the month of January

Queries on Joins and Correlation:

1) Find out the products, which have been sold to 'Ivan'.
2) Find the product_no and description of constantly sold i.e. rapidly moving products.
3) Find the names of clients who have purchased 'Cd Drive'.
4) List the product number and order number from customers who have ordered less than 4 units of 'Mouse'.

Queries on Constructing Sentences with data:

1) Print information from product-master, sales_order_detail tables in the following format for all records:-

   {description} Worth Rs. {total sales for the product} was sold.

NOTES

SUB QUERIES

A subquery is a form of an SQL statement that appears inside another SQL statement. It is also termed as nested query. The statement containing a subquery is called a parent statement. The parent statement uses the rows returned by the subquery. It can be used for the following:

- To insert records in target table
- To create tables and insert records in the table created.
- To update records in a target table.
- To create views
- To provide values for conditions in WHERE, HAVING, IN and so on used with SELECT, UPDATE, and DELETE statements.

It plays an important role in SQL mainly due to following reasons.

- SQL sub query is often the most natural way to express a query, because it most closely parallels the English-language description of the query.
- Sub queries lets you break a query down into pieces and then put the pieces together, hence very simple to implement.
- Some queries just can not be written without sub queries.
- Subquery must be enclosed in the parenthesis.

Subqueries can be divided into two broad groups-

- Single row suquery: This is subquery which returns only one value to the outer query.
- Multi row suquery: This is subquery which returns multiple values to the outer query.

IN or NOT IN

List the name, job and salary of people in department 2o who have the same job as people in department 30.

Sql>**SELECT** enmae, job, sal **FROM** EMP **WHERE** deptno=20
and Job **IN (SELECT** job **FROM** EMP **where** deptno=30**);**

ANY or ALL

If a suquery returns more than one row and the comparison other than equality is required (e.g. <,>, <=…etc) then the ANY, ALL operators are used.
If the relation required for the list of values returned by the inner query, then the ANY operator is used.

Example:
Find out the name, job and salary of people in department 20 who earn more than any one in department 30.

Sql>SELECT ename, job, sal FROM emp WHERE deptno=20
AND sal >ANY (SELECT sal FROM emp WHERE deptno=30);

If the relation required for the outer column in the WHERE clause is such that it should be TRUE for all the values in the list of values returned by the inner query; then the ALL operator is used.

Example:
Find out the name, job, salary of people who earn salary greater than all the people in department 30.

Sql>**SELECT** ename, job, sal **FROM** emp **WHERE** deptno=20
AND sal **>ALL** (**SELECT** sal **FROM** emp **WHERE** deptno=30);

Correlated Subqueries

A sub-query becomes correlated when the subquery references a column from a table in the parent query. A correlated subquery is evaluated once for each row processed by the parent statement, which can be any of SELECT, DELETE or UPDATE.

A correlated subquey is one way of reading every row in a table and comparing values in each row against related data. It is used whenever a subquery must return a different result for each candidate row considered by the parent query.

Consider the query to find out the employees who earn more than the average salary in their own department.

Sql>**SELECT** deptno, ename, sal **FROM** emp **WHERE**
Sal> **(SELECT** avg (sal) **FROM** emp **WHERE** deptno=deptno);

- A correlated suquery refers to a column selected by the main query.
- If a query performs a select from the same table as the main query the main query must define an alias for the table name and the subquery must use the alias to refer to the table name and the suquery must use the alias to refer the column value in the main query's candidate rows.
- HAVING clause can also be used in correlated suquery.

EXITS:-

The **EXITS** operator is usually used with correlated suqueries. This operator enables to test whether a value retrieved by the outer query exists in the results set of the values retrieved by the inner query. If the suqquery returns at least one row, the operator returns **TRUE.** If the value does not exit, it returns **FALSE.**

The EXISTS operator ensures that the search in inner query terminates when at least one match is found. Similarly, the NOT EXISTS operator enables to test whether a value retrieved by the outer

query is not a part of the result set of the values retrieved by the inner query.

Example:
Find all the departments that have employees yet.

Sql>**SELECT** deptno, dname, loc **FROM** dept **WHERE**
EXITS (SELECT deptno **FROM** emp **WHERE** deptno= emp.deptno**);**

Sub queries returning multiple columns:-

Inner queries can return more than one columns. Only one care should be taken to write the query in such a way that the order and the type of the column selected by the inner query.

Example:
Find out who are the highest paid employees in each department.

Sql>**SELECT** deptno, ename, sal **from** emp **WHERE** (deptno, sal)
IN (**SELECT** deptno, max (sal) **FROM** emp **GROUP BY** deptno);

JOINING TABLES

Sometimes it is necessary to work with multiple tables as though they were a single entity. Then a single SQL sentence can manipulate data from all the tables. Tables are joined on columns that have the same data type and data width in the tables.

JOINS

- The process of forming rows from two or more tables by comparing the contents of related column is called joining tables.
- The resulting table (containing data from both the original table) is called a join between the tables.
- Joins are foundation of multi-table query processing in SQL.

- Since SQL handles multi-table queries by matching columns, it uses the SELECT command for a search condition that specifies a column match.

Sql>SELECT columns FROM table1, table2,…
WHERE logical express

- The where clause specifies how to join or merge the tables together as well as the search criteria if any.
- Columns from tables may be named in SELECT clause.
- Columns which have the same name in multiple tale named in the from clause must be uniquely identified by specifying tablename.column name.
- Tablename.* is short cut to specify all the columns in the table. More than one pair of columns may be used to specify the join condition between two tables.

Using aliases

When using a join, it is good idea to identify all columns by using their table names. However, typing long table names can be difficult sometimes. So, to overcome this problem one can define short name aliases for tables.

Sql> SELECT t1.col1, t1.col2, t2.col3 FROM table t1, table t2
WHERE t1.col1= t2.col 1;

Here t1 and t2 are used as aliases for table 1 and table 2 respectively.

Types of JOIN

1. INNER JOIN OR EQUI JOIN

- A join that is formed as a result of an exact match between two columns is called an equi-join or a inner join. Statement generally compares two columns from two tables with the equivalences operator =.
- Consider query in which we want to known the name of the employee, his department and its location.

Sql > SELECT ename, deptno, locaion FROM Emp, Dept
WHERE Emp.deptno = Dept.deptno; (Theta Style)

Sql> SELECT ename, deptno, locaion FROM Emp E inner join Dept D

on E.deptno = D.deptno;

2. OUTER JOIN

Outer joins give more flexibility when selecting data from related tables. This type of join can be used in situations where it is desired, to select all rows from the table on the left(or right, or both) regardless of whether the other table has values in common and (usually) enter NULL where data is missing.

A) Left Outer Join

It returns all the rows from the first table, even if there are no matches in the second table. For example if there are no employees in Emp_m table that do not have any contacts in Cntc_d tale, those rows will also be listed.

Sql> SELECT E.name, E.dept, C.eno, C.cdata **FROM** Emp_m E LEFT JOIN Cntc_d C on E.eno = C.eno;

Sql> SELECT E.name, E.dept, C.eno, C.cdata **FROM** Emp_m E , Cntc_d C WHERE E.eno = C.eno(+);

B) Right Outer Join

It returns all the rows from the second table even if there are no matches in the first table.

Sql> SELECT E.name, E.dept, C.eno, C.cdata **FROM** Emp_m E RIGHT JOIN Cntc_d C on E.eno = C.eno;

Sql> SELECT E.name, E.dept, C.eno, C.cdata **FROM** Emp_m E , Cntc_d C WHERE E.eno(+) = C.eno;

C) Full Outer Join

It returns all the rows from the both tables even if there are no matches.

Sql> SELECT E.name, E.dept, C.eno, C.cdata **FROM** Emp_m E FULL JOIN Cntc_d C on E.eno = C.eno;

3.CROSS JOIN OR CARTESIAN JOIN:

This join combines every row from the left table with every row in the right table. This types of join can be used in situations where it is desired, to select all possible cominations of rows and columns from both tables. This kind of join is usually not preferred as it may run for a very long time and produce a huge result set that may not be useful.

Sql>SELECT ename, loc

FROM EMP Cross Join DEPT

WHERE job='CLERK'**;**

4. NON EQUI JOIN:-

- The join which uses comparison operation other than '=' while defining their joining criteria are called non-equi joins.
- Since this join can result in large number of rows, it is advisable to make a non-equi join in combination with a selection criteria to reduce the rows to a manageable range.

5. SELF-JOIN:-

- A self join is used to match and retrieve rows that have matching value in different columns of the same table.
- To join a table to itself, two copies of the very same tale have to opened in memory. Hence in the from clause, the table name needs to be mentioned twice. Since the table names are the same, each table is opened using an alias.
- These aliases table will cause two identical tables to be opened indifferent memory locations. These will result two identical tables to be physically present in the memory.

.

Example:- Consider relation Emp_m (E_no, Fname, Lname, m_no)

Retrieve the names of the employees and the names of their respective managers from the table

Sql > **SELECT** Emp.Fname, Mngr.Fname

FROM Emp_m Emp, Emp_m Mngr

WHERE Emp.m_no=Mngr.E_no.

- A table can be joined to itself as if it were two tables.
- As with any other join, the join is on columns that contain the same type of information.

- The table must be given s alias to synchronize which columns are coming from which tables.

GROUP BY Clause

This clause tells Oracle to group rows based on distinct values that exist for specified columns. i.e. It groups the selected rows based on the value of expression for each row and returns a single row of summary information for each group.

sql >SELECT grouped -by-columns functions **FROM** table name

GROUP BY column, column.,... ;

Example:

Retrieve the product numbers and the total quantity ordered for each product from the s_order_details.

sql >SELECT p_no, sum (qty_ordered) " total qty Ordered"

FROM s_order_details

GROUP BY p_no;

GROUPING FUNCTION

These functions act on a group or set of rows and return one Row of summary information per set.

SUM	Computes the total value of the group
AVG	Computes the average value of the group
MIN	Computes the minimum value of the group
MAX	Computes the maximum value of the group
STDDEV	Computes the standard deviation of the group
VARIANCE	Computes the variance of the group
COUNT	Counts the number of non-NULL values for the Specified group.
COUNT (*)	Counts the number of rows including those having NULL values for the given condition. .

HAVING CLAUSE

The HAVING clause can be used in conjunction with the GROUP BY clause. HAVING imposes a condition on the group by clause, which further filters the groups created by the group by clause.

HAVING and WHERE clauses work in a similar manner. The difference is that WHERE works

on rows, while HAVING works on groups. Expression in HAVING clause must be single value per group.

USING THE UNION, INTERSECT AND MINUS CLAUSE

Union Clause:

Multiple queries can be put together and their output combined using the union clause. The union clause merges the output of two or more queries into a single set of rows and columns.

Example:
Retrive the names of all the clients and salesman in the city of 'mumbai ' from the tables client_master and salesman_master.

```
SELECT salesman_no, name
        FROM salesman_master
        WHERE city='mumbai'
UNION
SELECT client_no, name
        FROM client_master
        WHERE city='mumbai';
```

Intersect Clause:

Multiple queries can be put together and their output combined using the intersect clause. The intersect clause outputs only rows produced by both the queries intersected i.e. the output in an intersect clause will include only those rows that are retrieved by both the queries.

Example:
Retrieves the salesman name in 'mumbai' whose efforts have resulted into at least one sales transaction.

```
SELECT salesman_no, name FROM salesman_master
                WHERE city='mumbai';
INTERSECT
SELECT salesman_master.salesman_no, name
    FROM salesman_master, sales_order
     WHERE salesman_master.salesman_no = sales_order.salesman_no;
```

The intersect clause picks up records that are common in both queries.

Minus Clause:

Multiple queries can be put together and their output combined using the minus clause. The minus clause outputs the rows produced by the first query, after filtering the rows retrieved by the second query.

Example:
Retrives all the product numbers of non-moving items from the product_master table.

Table name: sales_order_deatils

Order No	**Product No**
O111	P112
O112	P111
O113	P113
O114	P114

Table name: product_master

Product No	**Description**
P112	Monitors
P113	CD Drives
P116	Keyboards
P117	HDD

SELECT product no **FROM** product_master
MINUS
SELECT product no **FROM** sales_order_details;

The minus clause picks up records in the first query after filtering the records retrived by the second query. Thus, the output after applying the MINUS clause will be:

Output:

Product No

P113
P114

CONSTRUCTING AN ENGLISH SENTENCE WITH DATA FROM TABLE COLUMNS

Example:
Create an English sentence, by joining predetermined string values with column data retrieved from the sales_order table.

The string literals are : Order No.
Was placed by Client No. On

The columns are: Order_no
Client_no
Order_date

Table Name: Sales_order

Order_no	Client_no	Order_date
O1001	C001	04-aug-06
O1002	C002	05-aug-06
O1003	C003	06-aug-06

SELECT 'order no.'|| order_no || 'was placed by Client No'|| Client_no|| 'on' || order_date
FROM sales_order;

Output:

'orderno.'||Order_no||'wasplacedbyclientno.'||Client_no||'on'||Order_date

---

Order no. O1001 was placed by Client no. C001 on 04-aug-06 Order no. O1002 was placed by Client no. C002 on 05-aug-06 Order no. O1003 was placed by Client no. C003 on 06-aug-06

To avoid a data header that appears meaningless, use an alias as shown below:

SELECT 'order no.'|| order_no || 'was placed by Client No'|| Client_no|| 'on' ||
order_date "Orders placed"
FROM sales_order;

Output:

Orders placed

Order no. O1001 was placed by Client no. C001 on 04-aug-06