EXPERIMENT - VII
JOIN STATEMENTS, SET OPERATIONS, NESTED QUERIES AND GROUPING

August 16, 2019

ADITHYA D RAJAGOPAL
ROLL NO : 9
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
COLLEGE OF ENGINEERING TRIVANDRUM

AIM

To get introduced to

1. JOIN
2. UNION
3. INTERSECTION
4. MINUS
5. NESTED QUERIES
6. GROUP BY & HAVING

JOINS

A join is a query that combines rows from two or more tables, views, or materialized views ("snapshots"). Oracle performs a join whenever multiple tables appear in the query's FROM clause. The query's select list can select any columns from any of these tables. If any two of these tables have a column name in common, you must qualify all references to these columns throughout the query with table names to avoid ambiguity.

Join Conditions

Most join queries contain WHERE clause conditions that compare two columns, each from a different table. Such a condition is called a join condition. To execute a join, Oracle combines pairs of rows, each containing one row from each table, for which the join condition evaluates to TRUE. The columns in the join conditions need not also appear in the select list.
To execute a join of three or more tables, Oracle first joins two of the tables based on the join conditions comparing their columns and then joins the result to another table based on join conditions containing columns of the joined tables and the new table. Oracle continues this process until all tables are joined into the result. The optimizer determines the order in which Oracle joins tables based on the join conditions, indexes on the tables, and, in the case of the cost-based optimization approach, statistics for the tables.
In addition to join conditions, the WHERE clause of a join query can also contain other conditions that refer to columns of only one table. These conditions can further restrict the rows returned by the join query.

Equijoins

An equijoin is a join with a join condition containing an equality operator. An equijoin combines rows that have equivalent values for the specified columns. Depending on the internal algorithm the optimizer chooses to execute the join, the total size of the columns in the equijoin condition in a single table may be limited to the size of a data block minus some overhead. The size of a data block is specified by the initialization parameter DB_BLOCK_SIZE.

Self Joins

A self join is a join of a table to itself. This table appears twice in the FROM clause and is followed by table aliases that qualify column names in the join condition. To perform a self join, Oracle combines and returns rows of the table that satisfy the join condition.

Cartesian Products

If two tables in a join query have no join condition, Oracle returns their Cartesian product. Oracle combines each row of one table with each row of the other. A Cartesian product always generates many rows and is rarely useful. For example, the Cartesian product of two tables, each with 100 rows, has 10,000 rows. Always include a join condition unless you specifically need a Cartesian product. If a query joins three or more tables and you do not specify a join condition for a specific pair, the optimizer may choose a join order that avoids producing an intermediate Cartesian product.

Outer Joins

An outer join extends the result of a simple join. An outer join returns all rows that satisfy the join condition and those rows from one table for which no rows from the other satisfy the join condition. Such rows are not returned by a simple join. To write a query that performs an outer join of tables A and B and returns all rows from A, apply the outer join operator (+) to all columns of B in the join condition. For all rows in A that have no matching rows in B, Oracle returns NULL for any select list expressions containing columns of B.
Outer join queries are subject to the following rules and restrictions:

1. The (+) operator can appear only in the WHERE clause or, in the context of left-correlation (that is, when specifying the TABLE clause) in the FROM clause, and can be applied only to a column of a table or view.

2. If A and B are joined by multiple join conditions, you must use the (+) operator in all of these conditions. If you do not, Oracle will return only the rows resulting from a simple join, but without a warning or error to advise you that you do not have the results of an outer join.

3. The (+) operator can be applied only to a column, not to an arbitrary expression. However, an arbitrary expression can contain a column marked with the (+) operator.

4. A condition containing the (+) operator cannot be combined with another condition using the OR logical operator.

5. A condition cannot use the IN comparison operator to compare a column marked with the (+) operator with an expression.

6. A condition cannot compare any column marked with the (+) operator with a subquery.

If the WHERE clause contains a condition that compares a column from table B with a constant, the (+) operator must be applied to the column so that Oracle returns the rows from table A for which it has generated NULLs for this column. Otherwise Oracle will return only the results of a simple join.
In a query that performs outer joins of more than two pairs of tables, a single table can be the NULL-generated table for only one other table. For this reason, you cannot apply the (+) operator to columns of B in the join condition for A and B and the join condition for B and C.

SET OPERATORS

Set operators combine the results of two component queries into a single result. Queries containing set operators are called compound queries. Table lists SQL set operators.

Operator	Returns
UNION	all rows selected by either query.
UNION ALL	All rows selected by either query, including all duplicates.
INTERSECT	All distinct rows selected by both queries.
MINUS	All distinct rows selected by the first query but not the second

All set operators have equal precedence. If a SQL statement contains multiple set operators, Oracle evaluates them from the left to right if no parentheses explicitly specify another order.
The corresponding expressions in the select lists of the component queries of a compound query must match in number and datatype. If component queries select character data, the datatype of the return values are determined as follows:

1. If both queries select values of datatype CHAR, the returned values have datatype CHAR.

2. If either or both of the queries select values of datatype VARCHAR2, the returned values have datatype VARCHAR2.

NESTED QUERIES

Subquery

If a sql statement contains another sql statement then the sql statement which is inside another sql statement is called Subquery. It is also known as nested query. The Sql Statement which contains the other sql statement is called Parent Statement.

Nested Subquery

If a Subquery contains another subquery, then the subquery inside another subquery is called nested subquery.

Correlated Subquery

If the outcome of a subquery is depends on the value of a column of its parent query table then the Subquery is called Correlated Subquery.

GROUP BY & HAVING

The GROUP BY Statement in SQL is used to arrange identical data into groups with the help of some functions. i.e if a particular column has same values in different rows then it will arrange these rows in a group.
We can use HAVING clause to place conditions to decide which group will be the part of final result-set. Also we can not use the aggregate functions like SUM(), COUNT() etc. with WHERE clause. So we have to use HAVING clause if we want to use any of these functions in the conditions.

QUESTIONS

Amazon is one of the largest online stores operating in the United States of America. They are maintaining four tables in their database. The Items table, Customers table, Orders table and Delivery table. Each of these tables contains the following attributes:

1. Items
    (a) Itemid (primary key)
    (b) Itemname(type =varchar(50))
    (c) Category
    (d) Price
    (e) Instock (type=int, greater than or equal to zero)
2. Customers
    (a) Custid (primary key)
    (b) Custname
    (c) Address
    (d) State
3. Orders
    (a) Orderid (primary key)
    (b) Itemid (refers to Itemid of Items table)
    (c) Quantity (type=int)
    (d) Orderdate (type=date)
4. Delivery
    (a) Delivery id (primary key)
    (b) Custid (refers to Custid in customers table)
    (c) Orderid (refers to Orderid in orders table)

Create the above tables and populate them with appropriate data.

```
postgres=# SELECT * FROM ITEMS;
 itemid |      itemname      |  category   | price | instock
--------+--------------------+-------------+-------+---------
      3 | Samsung Galaxy S4  | Electronics |  2000 |       2
      5 | Sony z5 Premium    | Electronics |  5005 |       1
      2 | Redmi Note 4       | Electronics |  1000 |       4
      1 | A Study in Scarlet | Books       |   500 |       6
      4 | The Sign of Four   | Books       |  1500 |       2
(5 rows)

postgres=#
```

```
postgres=# SELECT * FROM CUSTOMERS;
 custid | custname |       address        |  state
--------+----------+----------------------+---------
    111 | Elvin    | 202 Jai Street       | Delhi
    112 | Patrick  | Street 1 Harinagar   | Chennai
    113 | Soman    | Puthumana p.o        | Kerala
    114 | Jaise    | Kottarakara          | Kerala
    115 | Mickey   | Juhu                 | Mumbai
(5 rows)

postgres=#
```

```
postgres=# SELECT * FROM ORDERS;
 orderid | itemid | quantity | orderdate
---------+--------+----------+------------
     201 |      4 |        3 | 2014-12-22
     202 |      1 |        2 | 2014-10-11
     203 |      3 |        1 | 2012-01-29
     204 |      5 |        1 | 2013-05-16
(4 rows)

postgres=#
```

```
postgres=# SELECT * FROM DELIVERY;
 deliveryid | custid | orderid |    status
------------+--------+---------+---------------
        301 |    111 |     202 | Delivered
        302 |    113 |     203 | Delivered
        303 |    115 |     204 | Delivered
        304 |    114 |     201 | Not Delivered
(4 rows)

postgres=#
```

1. List the details of all customers who have placed an order

```
postgres=# SELECT * FROM CUSTOMERS
postgres-# WHERE CUSTID IN(
postgres(# SELECT CUSTID FROM DELIVERY);
 custid | custname |    address      | state
--------+----------+-----------------+--------
    111 | Elvin    | 202 Jai Street | Delhi
    113 | Soman    | Puthumana p.o  | Kerala
    114 | Jaise    | Kottarakara    | Kerala
    115 | Mickey   | Juhu           | Mumbai
(4 rows)

postgres=#
```

2. List the details of all customers whose orders have been delivered

```
postgres=# SELECT * FROM CUSTOMERS
postgres-# WHERE CUSTID IN(
postgres(# SELECT CUSTID FROM DELIVERY
postgres(# WHERE STATUS='Delivered');
 custid | custname |    address      | state
--------+----------+-----------------+--------
    111 | Elvin    | 202 Jai Street | Delhi
    113 | Soman    | Puthumana p.o  | Kerala
    115 | Mickey   | Juhu           | Mumbai
(3 rows)

postgres=#
```

3. Find the orderdate for all customers whose name starts in the letter 'J'

```
postgres=# SELECT ORDERDATE FROM ORDERS
WHERE ORDERID=(SELECT ORDERID
FROM DELIVERY
WHERE CUSTID IN(SELECT CUSTID
FROM CUSTOMERS
WHERE CUSTNAME LIKE 'J%'));
 orderdate
------------
 2014-12-22
(1 row)

postgres=#
```

4. Display the name and price of all items bought by the customer 'Mickey'

```
postgres=# SELECT ITEMNAME,PRICE
postgres-# FROM ITEMS
postgres-# WHERE ITEMID IN(
postgres(# SELECT ITEMID FROM ORDERS
postgres(# WHERE ORDERID IN(
postgres(# SELECT ORDERID FROM DELIVERY
postgres(# WHERE CUSTID=(
postgres(# SELECT CUSTID FROM CUSTOMERS
postgres(# WHERE CUSTNAME='Mickey')));
    itemname     | price
-----------------+-------
 Sony z5 Premium |  5005
(1 row)

postgres=#
```

5. List the details of all customers who have placed an order after January 2013 and not received delivery of items.

```
postgres=# SELECT * FROM CUSTOMERS
postgres-# WHERE CUSTID IN(
postgres(# SELECT CUSTID FROM DELIVERY
postgres(# WHERE ORDERID IN(
postgres(# SELECT ORDERID FROM ORDERS
postgres(# WHERE ORDERDATE>='2013-02-01')
postgres(# AND STATUS='Not Delivered');
 custid | custname |   address   | state
--------+----------+-------------+--------
    114 | Jaise    | Kottarakara | Kerala
(1 row)

postgres=#
```

6. Find the itemid of items which has either been ordered or not delivered.

```
postgres=# (SELECT ITEMID FROM ORDERS
WHERE ITEMID IN(
SELECT ITEMID FROM ITEMS))
UNION (SELECT ITEMID
FROM ORDERS
WHERE ORDERID IN(
SELECT ORDERID FROM DELIVERY
WHERE STATUS='Not Delivered'));
 itemid
--------
      5
      4
      3
      1
(4 rows)

postgres=#
```

7. Find the name of all customers who have placed an order and have their orders delivered.

```
postgres=# (SELECT * FROM CUSTOMERS
postgres(# WHERE CUSTID IN(
postgres(# SELECT CUSTID FROM DELIVERY))
postgres-# INTERSECT (SELECT * FROM CUSTOMERS
postgres(# WHERE CUSTID IN(
postgres(# SELECT CUSTID FROM DELIVERY
postgres(# WHERE STATUS='Delivered'));
 custid | custname |    address      | state
--------+----------+-----------------+--------
    111 | Elvin    | 202 Jai Street | Delhi
    115 | Mickey   | Juhu            | Mumbai
    113 | Soman    | Puthumana p.o   | Kerala
(3 rows)

postgres=#
```

8. Find the custname of all customers who have placed an order but not having their orders delivered.

```
postgres=# (SELECT CUSTNAME FROM CUSTOMERS
WHERE CUSTID IN(
SELECT CUSTID FROM DELIVERY))
EXCEPT (SELECT CUSTNAME FROM CUSTOMERS
WHERE CUSTID IN(
SELECT CUSTID FROM DELIVERY
WHERE STATUS='Delivered'));
 custname
----------
 Jaise
(1 row)

postgres=#
```

9. Find the name of the customer who has placed the most number of orders.

```
postgres=# SELECT * FROM CUSTOMERS
postgres-# WHERE CUSTID IN(
postgres(# SELECT CUSTID FROM DELIVERY
postgres(# WHERE ORDERID IN(
postgres(# SELECT ORDERID FROM ORDERS
postgres(# WHERE QUANTITY>=ALL(
postgres(# SELECT QUANTITY FROM ORDERS)));
 custid | custname |   address   | state
--------+----------+-------------+--------
    114 | Jaise    | Kottarakara | Kerala
(1 row)

postgres=#
```

10. Find the details of all customers who have purchased items exceeding a price of 5000 $.

```
postgres=# SELECT * FROM CUSTOMERS
postgres-# WHERE CUSTID IN(
postgres(# SELECT CUSTID FROM DELIVERY
postgres(# WHERE ORDERid in(
postgres(# SELECT ORDERID FROM ORDERS O
postgres(# WHERE ITEMID IN(
postgres(# SELECT ITEMID FROM ITEMS
postgres(# WHERE O.QUANTITY*PRICE>5000)));
 custid | custname | address | state
--------+----------+---------+--------
    115 | Mickey   | Juhu    | Mumbai
(1 row)
```

11. Find the name and address of customers who has not ordered a 'Samsung Galaxy S4'

```
postgres=# SELECT CUSTNAME,ADDRESS FROM CUSTOMERS
WHERE CUSTID NOT IN(
SELECT CUSTID FROM DELIVERY
WHERE ORDERID IN(
SELECT ORDERID FROM ORDERS
WHERE ITEMID IN(
SELECT ITEMID FROM ITEMS
WHERE ITEMNAME='Samsung Galaxy S4')));
 custname |      address
----------+--------------------
 Elvin    | 202 Jai Street
 Patrick  | Street 1 Harinagar
 Jaise    | Kottarakara
 Mickey   | Juhu
(4 rows)

postgres=#
```

12. Perform Left Outer Join and Right Outer Join on Customers Orders Table.

```
postgres=# SELECT * FROM CUSTOMERS C
LEFT OUTER JOIN
(SELECT CUSTID,ORDERID FROM DELIVERY) D
ON (C.CUSTID=D.CUSTID)
LEFT OUTER JOIN ORDERS O
ON (D.ORDERID=O.ORDERID);
 custid | custname |     address      |  state  | custid | orderid | orderid | itemid | quantity | orderdate
--------+----------+------------------+---------+--------+---------+---------+--------+----------+------------
    111 | Elvin    | 202 Jai Street   | Delhi   |    111 |     202 |     202 |      1 |        2 | 2014-10-11
    113 | Soman    | Puthumana p.o    | Kerala  |    113 |     203 |     203 |      3 |        1 | 2012-01-29
    115 | Mickey   | Juhu             | Mumbai  |    115 |     204 |     204 |      5 |        1 | 2013-05-16
    114 | Jaise    | Kottarakara      | Kerala  |    114 |     201 |     201 |      4 |        3 | 2014-12-22
    112 | Patrick  | Street 1 Harinagar | Chennai |      |         |         |        |          |
(5 rows)

postgres=#
```

13. Find the details of all customers grouped by state

```
postgres=# SELECT * FROM CUSTOMERS
postgres-# WHERE STATE IN(
postgres(# SELECT STATE FROM CUSTOMERS
postgres(# GROUP BY STATE);
 custid | custname |      address        |  state
--------+----------+---------------------+---------
    111 | Elvin    | 202 Jai Street      | Delhi
    112 | Patrick  | Street 1 Harinagar  | Chennai
    113 | Soman    | Puthumana p.o       | Kerala
    114 | Jaise    | Kottarakara         | Kerala
    115 | Mickey   | Juhu                | Mumbai
(5 rows)

postgres=#
```

14. Display the details of all items grouped by category and having a price greater than the average price of all items.

```
postgres=# SELECT * FROM ITEMS
postgres-# WHERE (CATEGORY,PRICE)>=ALL(
postgres(# SELECT CATEGORY,AVG(PRICE)
postgres(# FROM ITEMS
postgres(# GROUP BY CATEGORY
postgres(# HAVING MAX(PRICE)>AVG(PRICE));
 itemid |    itemname     |  category   | price | instock
--------+-----------------+-------------+-------+---------
      5 | Sony z5 Premium | Electronics |  5005 |       1
(1 row)
```

RESULT

The query was executed and output was successfully obtained.