# EXP 4

Construct SQL queries using Group By, Order By, and Having Clauses.

# GROUP BY

- **Group By** clause is used for creating groups of records based on one or more columns.
- **GROUP BY** Clause is used to group the rows, which have the same values.
- Using these groups of records we can perform calculations with aggregate functions COUNT, MAX, MIN, SUM, AVG.

# GROUP BY

- **Syntax:**

  SELECT *column_name(s)*
  FROM *table_name*
  GROUP BY *column_name(s)*

# EMP table

| EID | ENAME | SALARY | DEPTNO |
|---|---|---|---|
| 1 | A | 1000 | 10 |
| 2 | B | 2000 | 20 |
| 3 | C | 1000 | 10 |
| 4 | D | 3000 | 20 |
| 5 | E | NULL | 20 |

# Sql>

- create table emp(eid number,ename varchar(16),salary number,deptno number);
- insert all
- into emp values(1,'A',1000,10)
- into emp values(2,'B',2000,20)
- into emp values(3,'C',1000,10)
- into emp values(4,'D',3000,20)
- into emp values(5,'E',NULL,20)
- select * from dual;
- select * from emp;

# GROUP BY

**Examples:**

- Find total salary of employees dept wise.

  **SELECT deptno, SUM(salary) AS totalsal**

  **FROM EMP**

  **GROUP BY deptno;**

| DEPTNO | TOTAL SAL |
|---|---|
| 10 | 2000 |
| 20 | 5000 |

# GROUP BY

- Find highest salary of employees dept wise.

  **SELECT deptno, MAX(salary) AS Highestsal**

  **FROM EMP**

  **GROUP BY deptno;**

| DEPTNO | HIGHESTSAL |
|---|---|
| 10 | 1000 |
| 20 | 3000 |

# GROUP BY

- Find total salary of employees dept wise considering employees with salary>1500.

```
SELECT  deptno, SUM(salary) AS totalsal
FROM  EMP
WHERE SALARY>1500
GROUP BY deptno;
```

| DEPTNO | TOTALSAL |
|---|---|
| 20 | 5000 |

# Having clause

- **HAVING** clause is used with **GROUP BY** clause to filter groups of rows based on certain conditions.
- **Having** clause is implemented after the **'GROUP BY'** clause in the **'SELECT'** statement.
- HAVING clause is used to filter grouped rows instead of single rows in the table.
- HAVING clause must always be placed after the ‘GROUP BY’ clause in the query.

# Having clause

- **HAVING** clause can use aggregate functions such as COUNT(), SUM(), AVG(), etc., whereas the **WHERE** clause cannot be used with them**.**
- **Having** clause eliminates non- matching groups of records.

# Having clause

**Syntax:**

```
SELECT   col1,col2,  aggregate_function(column)
FROM  table_name
GROUP BY  col1, col2
HAVING  condition;
```

# EMP table

| EID | ENAME | SALARY | DEPTNO |
|---|---|---|---|
| 1 | A | 1000 | 10 |
| 2 | B | 2000 | 20 |
| 3 | C | 1000 | 10 |
| 4 | D | 3000 | 20 |
| 5 | E | NULL | 20 |

# Having clause

- Find dept-wise sum of salaries of employees which are greater than 2000.

```
SELECT  deptno,  SUM(salary) AS totalsal
FROM EMP
GROUP BY deptno
HAVING  SUM(salary)>2000;
```

| DEPTNO | TOTALSAL |
|---|---|
| 20 | 5000 |

# ORDER BY

- **ORDER BY** clause is used to sort the records of the table based on one or more columns.
- Using the ORDER BY clause, we can sort the records in ascending or descending order.
- **ASC** keyword is used with “ORDER BY” clause to sort records in ascending order.
- **DESC** keyword will sort the records in descending order.
- If no keyword is specified ,the sorting will be done by default in the ascending order.

# ORDER BY

**Syntax**

**SELECT** *column1, column2, ...*
**FROM** *table_name*
**ORDER BY** *column1, column2, ...* [ASC|DESC];

# EMP table

| EID | ENAME | SALARY | DEPTNO |
|---|---|---|---|
| 1 | A | 1000 | 10 |
| 2 | B | 2000 | 20 |
| 3 | C | 1000 | 10 |
| 4 | D | 3000 | 20 |
| 5 | E | NULL | 20 |

# ORDER BY

- Display emp in ascending order of salary.

  **SELECT * FROM emp ORDER BY salary;**

| EID | ENAME | SALARY | DEPTNO |
|---|---|---|---|
| 1 | A | 1000 | 10 |
| 3 | C | 1000 | 10 |
| 2 | B | 2000 | 20 |
| 4 | D | 3000 | 20 |
| 5 | E | NULL | 20 |

# ORDER BY

- Display emp in descending order of salary.

  **SELECT * FROM emp ORDER BY salary DESC;**

| EID | ENAME | SALARY | DEPTNO |
|---|---|---|---|
| 5 | E | NULL | 20 |
| 4 | D | 3000 | 20 |
| 2 | B | 2000 | 20 |
| 1 | A | 1000 | 10 |
| 3 | C | 1000 | 10 |

# Exp 8

Construct SQL Queries on joins and Correlated subqueries

# JOINS

- JOIN is used to retrieve data from multiple tables.
- Join query is used to combine rows from two or more tables and creates a new table.
- There must be one join condition for joining two tables, result set conatins rows for which join condition is true.

# JOINS

There are  different types of  joins:

- Inner join
- Left outer join
- Right outer join
- Full outer join
- Cross join
- Self join

# Cross join

- CROSS JOIN of two tables makes a Cartesian product of the tables.
- CROSS JOIN combines all rows from first table with all of the rows of second table.
- If there are "x" rows in table1 and "y" rows in table2 then the CROSS JOIN result set have x*y rows.

# Cross join

**Syntax:**

```
SELECT  <column_list>
FROM
table1  CROSS JOIN  table2;
```

| ENO | ENAME | SAL | DNO |
|---|---|---|---|
| 1 | A | 1000 | 10 |
| 2 | B | 2000 | 20 |
| 3 | C | 1000 | 10 |
| 4 | D | 3000 | 20 |

| DNO | DNAME | LOCATION |
|---|---|---|
| 10 | PROD | HYD |
| 20 | FINAN | DEL |

# Sql>

- create table emp(eid number,ename varchar(16),sal number,dno number);
- insert all
- into emp values(1,'A',1000,10)
- into emp values(2,'B',2000,20)
- into emp values(3,'C',1000,10)
- into emp values(4,'D',3000,20)
- select * from dual;
- select * from emp;

- create table dept(dno number,dname varchar(16),location varchar(20));
- insert all
- into dept values(10,'prod','hyd')
- into dept values(20,'finan','del')
- select * from dual;
- select * from dept;

# CROSS JOIN

**Example:**

- SELECT * FROM EMP **CROSS JOIN** DEPT;

| ENO | ENAME | SAL | DNO | DNO | DNAME | LOCATION |
|---|---|---|---|---|---|---|
| 1 | A | 1000 | 10 | 10 | PROD | HYD |
| 2 | B | 2000 | 20 | 10 | PROD | HYD |
| 3 | C | 1000 | 10 | 10 | PROD | HYD |
| 4 | D | 3000 | 20 | 10 | PROD | HYD |
| 1 | A | 1000 | 10 | 20 | FINAN | DEL |
| 2 | B | 2000 | 20 | 20 | FINAN | DEL |
| 3 | C | 1000 | 10 | 20 | FINAN | DEL |
| 4 | D | 3000 | 20 | 20 | FINAN | DEL |

# JOINS

- JOIN is used to retrieve data from multiple tables.
- Join query is used to combine rows from two or more tables and creates a new table.
- There must be one join condition for joining two tables. It compares two columns from the different tables and combines rows, for which join condition is true to form the result set.

# Inner join

- Inner Join is the simplest and most common type of join.
- It returns all rows from multiple tables where the join condition is met.
- INNER JOIN is used to retrieve matching rows from tables.

# Inner join

**Syntax:**

```
SELECT <columns>
FROM
table1  INNER JOIN table2
ON  <join condition> ;
```

| ENO | ENAME | SAL | DNO |
|---|---|---|---|
| 1 | A | 1000 | 10 |
| 2 | B | 2000 | 20 |
| 3 | C | 1000 | 10 |
| 4 | D | 3000 | 20 |
| 5 | E | 2000 | - |

| DNO | DNAME | LOCATION |
|---|---|---|
| 10 | PROD | HYD |
| 20 | FINAN | DEL |
| 40 | SALES | MUM |

# Sql>

- create table emp(eid number,ename varchar(16),sal number,dno number);
- insert all
- into emp values(1,'A',1000,10)
- into emp values(2,'B',2000,20)
- into emp values(3,'C',1000,10)
- into emp values(4,'D',3000,20)
- into emp values(5,'E',2000,NULL)
- select * from dual;
- select * from emp;

- create table dept(dno number,dname varchar(16),location varchar(20));
- insert all
- into dept values(10,'prod','hyd')
- into dept values(20,'finan','del')
- into dept values(40,'sales','mum')
- select * from dual;
- select * from dept;

# Inner join

SELECT * FROM emp **INNER JOIN** dept

**ON** emp.dno = dept.dno;

| ENO | ENAME | SAL | DNO | DNO | DNAME | LOCATION |
|---|---|---|---|---|---|---|
| 1 | A | 1000 | 10 | 10 | PROD | HYD |
| 2 | B | 2000 | 20 | 20 | FINAN | DEL |
| 3 | C | 1000 | 10 | 10 | PROD | HYD |
| 4 | D | 3000 | 20 | 20 | FINAN | DEL |

# Outer join

- An outer join is similar to innerjoin but it also gets the non-matched rows from the table.

1. Left Outer Join
2. Right Outer Join
3. Full Outer Join

# Left Outer Join

- **Left Outer Join** returns all rows from the left side table specified in the **ON** condition and only those rows from the right table where the join condition is met.
- It is used to retrieve all the matching records from both the tables as well as non-matching records from the left side table only.

# Left Outer Join

- If there is no matching row found from the right table, the left join will have null values for the columns of the right table.

# Left Outer Join

**Syntax:**

SELECT <columns>

FROM table1 **LEFT [OUTER] JOIN** table2

ON table1.column = table2.column;

| ENO | ENAME | SAL | DNO |
|---|---|---|---|
| 1 | A | 1000 | 10 |
| 2 | B | 2000 | 20 |
| 3 | C | 1000 | 10 |
| 4 | D | 3000 | 20 |
| 5 | E | 2000 | - |

| DNO | DNAME | LOCATION |
|---|---|---|
| 10 | PROD | HYD |
| 20 | FINAN | DEL |
| 40 | SALES | MUM |

# Left Outer Join

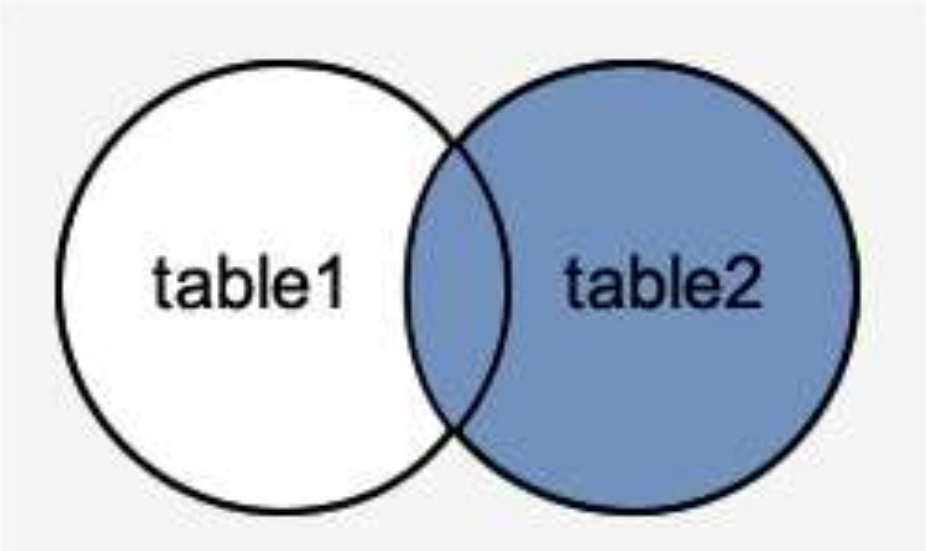
- SELECT * FROM emp LEFT OUTER JOIN dept ON emp.dno=dept.dno;

| ENO | ENAME | SAL | DNO | DNO | DNAME | LOCATION |
|---|---|---|---|---|---|---|
| 3 | C | 1000 | 10 | 10 | PROD | HYD |
| 1 | A | 1000 | 10 | 10 | PROD | HYD |
| 4 | D | 3000 | 20 | 20 | FINAN | DEL |
| 2 | B | 2000 | 20 | 20 | FINAN | DEL |
| 5 | E | 2000 | - | - | - | - |

# Right Outer Join

- Right Outer Join returns all rows from the right-hand table specified in the ON condition and only those rows from the left table where the join condition is met.
- It is used to retrieve all the matching records from both the tables as well as non-matching records from the right side table only.

# Right Outer Join



# Right Outer Join

**Syntax:**

SELECT <columns>

FROM table1 **RIGHT [OUTER] JOIN** table2

ON table1.column = table2.column;

| ENO | ENAME | SAL | DNO |
|---|---|---|---|
| 1 | A | 1000 | 10 |
| 2 | B | 2000 | 20 |
| 3 | C | 1000 | 10 |
| 4 | D | 3000 | 20 |
| 5 | E | 2000 | - |

| DNO | DNAME | LOCATION |
|---|---|---|
| 10 | PROD | HYD |
| 20 | FINAN | DEL |
| 40 | SALES | MUM |

# Right Outer Join

**Example:**

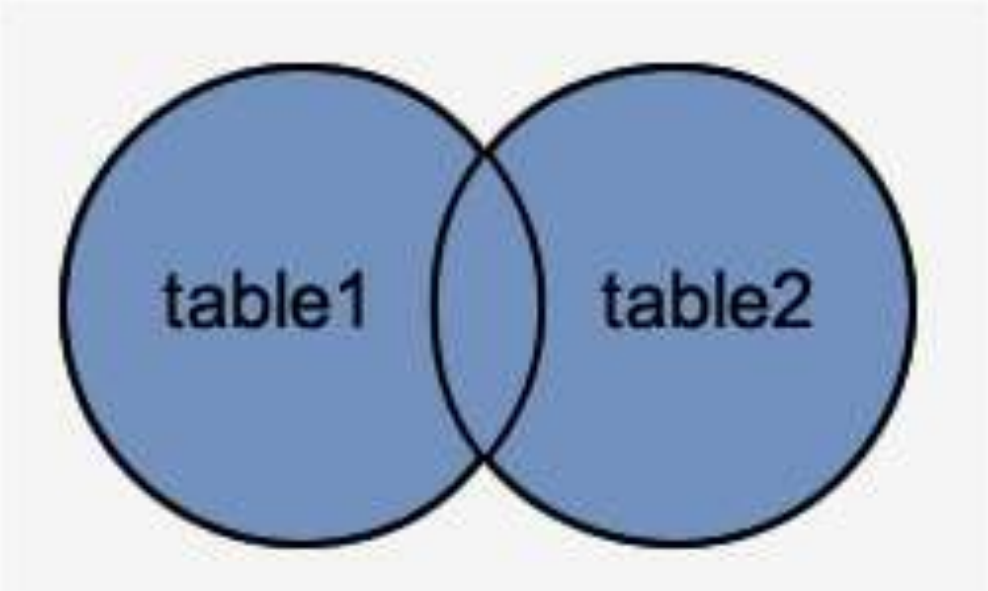
- SELECT * FROM emp RIGHT OUTER JOIN dept ON emp.dno=dept.dno;

| ENO | ENAME | SAL | DNO | DNO | DNAME | LOCATION |
|---|---|---|---|---|---|---|
| 1 | A | 1000 | 10 | 10 | PROD | HYD |
| 2 | B | 2000 | 20 | 20 | FINAN | DEL |
| 3 | C | 1000 | 10 | 10 | PROD | HYD |
| 4 | D | 3000 | 20 | 20 | FINAN | DEL |
| | | | | 40 | SALES | MUM |

# Full Outer Join

- **Full Outer Join** returns all rows from the left hand table and right hand table with the matching rows AND non matching rows from **both sides**.
- It places **NULL** where the join condition is not met.

# Full Outer Join



# Full Outer Join

**Syntax:**

SELECT <columns>

FROM table1 **FULL [OUTER] JOIN** table2

**ON** table1.column = table2.column;

| ENO | ENAME | SAL | DNO |
|---|---|---|---|
| 1 | A | 1000 | 10 |
| 2 | B | 2000 | 20 |
| 3 | C | 1000 | 10 |
| 4 | D | 3000 | 20 |
| 5 | E | 2000 | - |

| DNO | DNAME | LOCATION |
|---|---|---|
| 10 | PROD | HYD |
| 20 | FINAN | DEL |
| 40 | SALES | MUM |

# Full Outer Join

- **SELECT * FROM emp FULL OUTER JOIN dept ON emp.dno=dept.dno;**

| ENO | ENAME | SAL | DNO | DNO | DNAME | LOCATION |
|---|---|---|---|---|---|---|
| 3 | C | 1000 | 10 | 10 | PROD | HYD |
| 1 | A | 1000 | 10 | 10 | PROD | HYD |
| 4 | D | 3000 | 20 | 20 | FINAN | DEL |
| 2 | B | 2000 | 20 | 20 | FINAN | DEL |
| 5 | E | 2000 | - | - | - | - |
| -- | - | - | - | 40 | SALES | MUM |

# EQUIJOIN

- **EQUIJOIN** contains only equality operator in join condition.
- Equi join returns the matching rows based on join condition(=) of the associated tables.
- When we use EQUI join between two or more tables, there should be a common column.
- Common column names need not be the same name but datatype must be matched.

# EQUIJOIN

**Syntax:**

**SELECT** <column_list> **FROM** table1, table2
**WHERE** table1.column =table2.column;

**(or)**

**SELECT** <column_list> **FROM** table1 **JOIN** table2
**ON** table1.column =table2.column;

# EQUIJOIN

**SELECT** * FROM emp **JOIN** dept

**ON** emp.dno **=** dept.dno;

| ENO | ENAME | SAL | DNO | DNO | DNAME | LOCATION |
|---|---|---|---|---|---|---|
| 1 | A | 1000 | 10 | 10 | PROD | HYD |
| 2 | B | 2000 | 20 | 20 | FINAN | DEL |
| 3 | C | 1000 | 10 | 10 | PROD | HYD |
| 4 | D | 3000 | 20 | 20 | FINAN | DEL |

# EQUIJOIN

**SELECT** * FROM emp, dept

WHERE emp.dno **=** dept.dno;

| ENO | ENAME | SAL | DNO | DNO | DNAME | LOCATION |
|---|---|---|---|---|---|---|
| 1 | A | 1000 | 10 | 10 | PROD | HYD |
| 2 | B | 2000 | 20 | 20 | FINAN | DEL |
| 3 | C | 1000 | 10 | 10 | PROD | HYD |
| 4 | D | 3000 | 20 | 20 | FINAN | DEL |

# Self Join

- In Self Join, a table is joined with itself.
- A self join specifies that each rows of a table is combined with itself and every other row of the table.
- To perform self join, use table alias with different names for table in the query.

# Self Join

**Syntax:**

**SELECT** A1.column_name, A2.column_name...
**FROM** table1 **A1**, table1 **A2**
**WHERE** A1.common_filed = A2.common_field;

# Self Join

| ENO | ENAME | SAL | MGR_NO |
|---|---|---|---|
| 1 | A | 1000 | - |
| 2 | B | 2000 | 1 |
| 3 | C | 1000 | 1 |
| 4 | D | 3000 | 5 |
| 5 | E | 2000 | - |

# Sql>

- create table emp(eid number,ename varchar(16),sal number,mgr_no number);
- insert all
- into emp values(1,'A',1000,NULL)
- into emp values(2,'B',2000,1)
- into emp values(3,'C',1000,1)
- into emp values(4,'D',3000,5)
- into emp values(5,'E',2000,NULL)
- select * from dual;
- select * from emp;

# Self Join

- **Find the names of employees and their manager names.**

```
SELECT E.ENAME  AS EMPNAME ,
            M.ENAME AS MANAGER
            FROM EMP E, EMP M
WHERE E.MGR_NO = M.ENO;
```

| EMPNAME | MANAGER |
|---|---|
| C | A |
| B | A |
| D | E |

# Self Join

- **Find the employees whose salary is > their manager salary.**

```
SELECT  E.ENAME   AS  EMPNAME ,
                M.ENAME  AS  MANAGER
                FROM EMP E, EMP M
 WHERE E.MGR_NO = M.ENO
                AND E.SAL>M.SAL;
```

# NESTED QUERIES

- A **nested query** is a query that has another query inside it.
- The Query that lies inside or embedded is called **subquery**.
- subquery usually appears in WHERE clause of another query.
- Nested queries are used for writing complex queries ,Where we divide the complex query into isolated parts.

# NESTED QUERIES

- **Subquery** is also known as **Inner query** and the query containing it is the **outer** query.
- The result of inner query is used in execution of outer query.

**Outer Query**

- **Syntax:** SELECT <column list> FROM <tables>
  WHERE col_name OPERATOR (**SELECT Query**);

**Inner Query**

# Types of Nested Queries

**Independent Nested Queries**

- Execution of inner query is independent of outer query and the result of inner query is used in execution of outer query.

**Correlated Nested Queries**

- In correlated nested queries, the output of inner query **depends** on outer query.

  i.e. the row which is currently executed in outer query.

# Independent Nested Queries

- **Single row subquery** (single value)

- **multiple-row subquery (**multiple values)

# Independent Nested Queries

**1. Single row subquery** (single value)

- When a Subquery returns a single value is called a Single Row Subquery.
- For Single Row Subquery , we can use operators such as = , < , > , <= , >=, !=
- A single-row subquery can return only one row of results consisting of only one column to the outer query.

# Independent Nested Query

1. **Single row subquery** (single value)

Example:

**Find employees whose salary is greater than salary of john.**

```
SELECT * FROM EMP WHERE sal >
(SELECT sal FROM EMP WHERE ENAME='john');
```

# Independent Nested Query

**Single row subquery**

Example:

- **Find employees whose sal is minimum salary of all employees.**

  SELECT * FROM EMP

  WHERE  sal = (SELECT MIN(sal ) FROM EMP);

# Independent Nested Query

**Multiple-row subquery:**

- Subquery that return more than one value (multiple rows of only one column) to the outer query are called multiple-row subquery.
- For Multi Row Subquery , we use IN, ANY, or ALL operators.

# Independent Nested Query

**Multiple-row subquery:**

- Example
- Find all employees whose salary is more than all employees working in deptno 10;

```
SELECT  * FROM EMP WHERE sal >ALL
(SELECT sal FROM EMP WHERE deptno=10);
```

# Independent Nested Query

**Multiple-row subquery:**

- Example
- Find all employees whose salary is equal to employees working in deptno 10;

  SELECT * FROM EMP WHERE sal **IN**

  (SELECT sal FROM EMP WHERE deptno=10);

# Correlated Nested Query

- Correlated subquery uses the values from the outer query.
- A correlated subquery gets executed repeatedly once for each row of main query.
- With a normal nested subquery, the inner **SELECT** query runs first and executes only once, returning values to the main query.

# Correlated Nested Query

- A correlated subquery, however executes once for each candidate row of the outer query. In other words, the inner query is driven by the outer query.

# Correlated Nested Query

- **find all employees whose salary is higher than the average salary of the employees in their department:**

  SELECT Eid, Ename, salary, deptno

  FROM employees E

  WHERE sal > (SELECT AVG(sal) FROM employees  WHERE deptno = E.deptno)

# Execution strategy for Correlated Nested Query

1. First, the outer query selects a row.
2. Inner query executes once for row selected by outer query. It uses the value of the selected row and returns a result set.
3. Outer query uses the result set returned by the inner query. It determines whether the selected row should be included in the final output.
4. Steps 2 and 3 are repeated for each row in the outer query's result set.

# Correlated Nested Query

**Find the names of sailors who have reserved boat number 103.**

```
SELECT S.sname FROM Sailors S
WHERE EXISTS
( SELECT  * FROM Reserves R
   WHERE R.bid = 103 AND R.sid = S.sid )
```