#### **EXPERIMENT No: 5**

Title: Study & Implementation of

- Group by & Having Clause
- Order by Clause
- Indexing 🛭
- Sub quereis
- Views

# Theory.

#### **GROUP BY:**

The GROUP BY clause groups rows that have the same values into summary rows.

*Syntax:* SELECT column1, aggregate\_function(column2)

FROM table\_name GROUP BY column1;

**Example 1: Simple Grouping** 

- Count employees in each department

SELECT department\_id, COUNT(\*) as employee\_count

FROM employees

GROUP BY department\_id;

Example 2:

-- Calculate the sum of salaries of all employees

SELECT EMPNO, SUM (SALARY) as Total\_Salary

FROM EMPLOYEES

**GROUP BY EMPNO;** 

**Example 3: Grouping by Multiple Columns** 

- Count employees by department and job title

SELECT department\_id, iob\_id, COUNT(\*) as employee\_count

FROM employees

GROUP BY department\_id, job\_id;

#### **HAVING:**

The HAVING clause was added to SQL because the WHERE keyword could not be used with aggregate functions. The HAVING clause must follow the GROUP BY clause in a query and must also precede the ORDER BY clause if used.

**Syntax:** SELECT column1, aggregate\_function(column2)

FROM table\_name GROUP BY column1 HAVING condition;

**Example 1: Filtering Groups** 

- Find departments with more than 5 employees

SELECT department\_id, COUNT(\*) as employee\_count

FROM employees

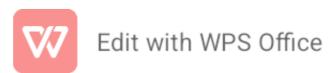
GROUP BY department\_id

HAVING COUNT(\*) > 5;

**Example 2: HAVING with Multiple Conditions** 

- Find departments with avg salary > 5000 and at least 3 employees

SELECT department\_id, AVG(salary) as avg\_salary, COUNT(\*) as emp\_count FROM employees



GROUP BY department\_id

HAVING AVG(salary) > 5000 AND COUNT(\*) >= 3;

#### **ORDER BY:**

The ORDER BY clause sorts the result set in ascending or descending order. **Syntax**:

SELECT column1, column2

FROM table\_name

ORDER BY column1 [ASC|DESC], column2 [ASC|DESC];

**Example 1: Simple Sorting** 

# - Sort employees by last name

SELECT employee\_id, last\_name, first\_name

FROM employees

ORDER BY last\_name;

#### **Example 2: Sorting by Multiple Columns**

# Sort by department then by salary (highest first)

SELECT employee\_id, last\_name, department\_id, salary

FROM employees

ORDER BY department\_id, salary DESC;

## Combining GROUP BY, HAVING, and ORDER BY

# Find departments with total salary > 20000, order by total salary descending

SELECT department\_id, SUM(salary) as total\_salary, COUNT(\*) as employee\_count

FROM employees

GROUP BY department\_id

HAVING SUM(salary) > 20000

ORDER BY total\_salary DESC;

# Find job titles with average salary between 5000 and 10000, ordered by avg salary

SELECT job\_id, AVG(salary) as avg\_salary, COUNT(\*) as employee\_count

FROM employees

GROUP BY job\_id

HAVING AVG(salary) BETWEEN 5000 AND 10000

ORDER BY avg\_salary DESC;

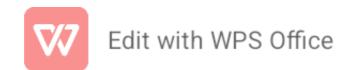
#### **INDEXING**:

Indexes in Oracle are database objects that improve data retrieval performance. They work like a book's index, allowing the database to find data without scanning the entire table.

#### Types of Indexes in Oracle

- 1) B-Tree Index(Balanced Tree Index): The most common type of index, suitable for high-cardinality columns (columns with many distinct values). Example:
- Create a basic B-Tree index

CREATE INDEX emp\_last\_name\_idx ON employees(last\_name);



# Composite index on multiple columns

CREATE INDEX emp\_dept\_job\_idx ON employees(department\_id, job\_id);

- 2) **Bitmap Index**: Best for low-cardinality columns (columns with few distinct values like gender or status flags).
- Create a bitmap index on gender column

CREATE BITMAP INDEX emp\_gender\_idx ON employees(gender);

Bitmap index on department\_id (if few departments)

CREATE BITMAP INDEX emp\_dept\_bitmap\_idx ON employees(department\_id);

3) Function-Based Index: Indexes the result of a function or expression.

CREATE INDEX idx\_employee\_upper\_name

ON employees (UPPER(last\_name));

4) Unique Index: Ensures that all values in the indexed column are unique.

CREATE UNIQUE INDEX idx\_employee\_email

ON employees (email);

5) Composite Index: Index on multiple columns.

CREATE INDEX idx\_employee\_name\_dept

ON employees (last\_name, department\_id);

6) **Reverse Key Index**: Reverses the bytes of the indexed column, useful for balancing I/O in certain scenarios.

CREATE INDEX idx\_employee\_id\_reverse

ON employees (employee\_id) REVERSE;

# **Index Management Examples**

## View existing indexes:

SELECT index\_name, index\_type, table\_name

FROM user\_indexes

WHERE table\_name = 'EMPLOYEES';

#### **Dropping an Index:**

To remove an index, use the DROP INDEX statement:

DROP INDEX index\_name:

DROP INDEX emp\_last\_name\_idx;

# Rebuild an index (for maintenance):

ALTER INDEX emp\_last\_name\_idx REBUILD;

Gather statistics for an index:

ANALYZE INDEX emp\_last\_name\_idx COMPUTE STATISTICS;

#### Rename an Index

ALTER INDEX idx\_employee\_name RENAME TO idx\_emp\_name;

#### When to Use Indexes

Columns frequently used in WHERE clauses.

Columns used in JOIN conditions.

Columns with high selectivity (many distinct values).

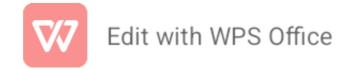
#### When Not to Use Indexes

On small tables.

On columns with low selectivity (few distinct values).

On columns frequently updated (indexes slow down INSERT, UPDATE, and DELETE operations).

# **Performance Considerations**



# Example: Force index usage (hint):

SELECT /\*+ INDEX(employees emp\_last\_name\_idx) \*/ \*

FROM employees

WHERE last\_name = 'Smith';

# Example: Check if index is used (explain plan):

**EXPLAIN PLAN FOR** 

SELECT \* FROM employees WHERE last\_name = 'Smith';

SELECT \* FROM TABLE(DBMS\_XPLAN.DISPLAY);

# Partial/Filtered index (Oracle 12c+):

CREATE INDEX emp\_active\_idx ON employees(employee\_id) WHERE status = 'ACTIVE':

## **SUBQUERIES:**

Subqueries (or nested queries) are queries embedded within other SQL statements. They are powerful tools for complex data retrieval in Oracle.

# **Types of Subqueries**

#### 1. Single Row Subquery

Returns a single row and is used with single-row comparison operators like =, >, <, etc.

# Find employees who earn more than the average salary.

SELECT employee\_id, first\_name, last\_name, salary

FROM employees

WHERE salary > (SELECT AVG(salary) FROM employees);

#### 2. Multi-Row Subqueries

Returns multiple rows

# -Find employees who work in the same department as employees with the last name 'Smith'.

SELECT employee\_id, first\_name, last\_name, department\_id

FROM employees

WHERE department\_id IN (SELECT department\_id FROM employees WHERE last\_name = 'Smith');

#### - Find employees in departments located in the US

SELECT employee\_id, last\_name, department\_id

FROM employees

WHERE department\_id IN (

SELECT department\_id FROM departments

WHERE location\_id IN (

SELECT location\_id FROM locations

WHERE country\_id = 'US' ));

#### 3 .Multi-column Subqueries

Return multiple columns.

# - Find employees with same job and department as employee 103

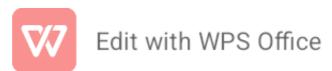
SELECT employee\_id, last\_name, job\_id, department\_id

FROM employees

WHERE (job\_id, department\_id) = (

SELECT job\_id, department\_id

FROM employees



WHERE employee\_id = 103

);

# 4. Correlated subquery

A subquery that references a column from the outer query.

Find employees who earn more than the average salary in their department.

SELECT employee\_id, first\_name, last\_name, salary, department\_id FROM employees e1

WHERE salary > (SELECT AVG(salary) FROM employees e2 WHERE e2.department\_id = e1.department\_id);

**5. Scalar Subquery:** Returns a single value and can be used in the SELECT clause or as part of an expression.

Retrieve the employee's name and the average salary of their department.

SELECT first\_name, last\_name, salary,

(SELECT AVG(salary) FROM employees e2 WHERE e2.department\_id = e1.department\_id) AS avg\_dept\_salary FROM employees e1;

#### VIEW:

Views are virtual tables that represent the result of a stored query. They don't store data physically but provide a way to simplify complex queries, enhance security, and present data differently to different users.

A view is a virtual table, which consists of a set of columns from one or more tables. It is similar to a table but it does not store in the database. View is a query stored as an object.

**Syntax:** CREATE VIEW <view\_name> AS SELECT <set of fields> FROM relation\_name WHERE (Condition)

#### Example:

SQL> CREATE VIEW employee AS SELECT empno,ename,job FROM EMP WHERE job = 'clerk'; SQL> View created.

#### Example:

CREATE VIEW [Current Product List] AS SELECT ProductID, ProductName FROM Products WHERE Discontinued=No;

**UPDATING A VIEW**: A view can updated by using the following syntax:

Syntax: CREATE OR REPLACE VIEW view\_name AS

SELECT column\_name(s)

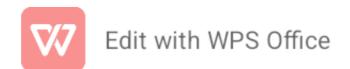
FROM table\_name WHERE condition

**DROPPING A VIEW:** A view can deleted with the DROP VIEW command.

Syntax: DROP VIEW <view\_name>;

#### LAB PRACTICE ASSIGNMENT:

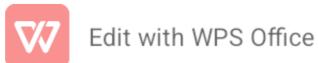
Consider the following schema:



```
Sailors (sid, sname, rating, age)
Boats (bid, bname, color)
Reserves (sid, bid, day(date))
1. Creating the Tables
-- Creating Sailors Table
CREATE TABLE Sailors
  sid NUMBER(5) PRIMARY KEY,
  sname VARCHAR2(50) NOT NULL,
  rating NUMBER(2) CHECK (rating >= 1 AND rating <= 10),
  age NUMBER(3,1) CHECK (age > 0)
);
-- Creating Boats Table
CREATE TABLE Boats
  bid NUMBER(5) PRIMARY KEY,
  bname VARCHAR2(50) NOT NULL,
  color VARCHAR2(20) NOT NULL
);
-- Creating Reserves Table
CREATE TABLE Reserves
  sid NUMBER(5),
  bid NUMBER(5),
  day DATE,
  PRIMARY KEY (sid, bid, day),
  FOREIGN KEY (sid) REFERENCES Sailors(sid) ON DELETE CASCADE,
  FOREIGN KEY (bid) REFERENCES Boats(bid) ON DELETE CASCADE
2. Inserting Sample Records
-- Inserting data into Sailors table
INSERT ALL
INTO Sailors VALUES (1, 'Alice', 7, 23.5)
INTO Sailors VALUES (2, 'Bob', 5, 29.0)
INTO Sailors VALUES (3, 'Charlie', 6, 35.2)
INTO Sailors VALUES (4, 'David', 8, 40.0)
INTO Sailors VALUES (5, 'Eve', 3, 21.7)
SELECT * FROM DUAL;
-- Inserting data into Boats table
INSERT ALL
 INTO Boats VALUES (101, 'Seafarer', 'red')
 INTO Boats VALUES (102, 'Ocean Wave', 'blue')
 INTO Boats VALUES (103, 'Sunshine', 'green')
INTO Boats VALUES (104, 'Speedster', 'red')
INTO Boats VALUES (105, 'Blue Pearl', 'blue')
SELECT * FROM DUAL;

    Inserting data into Reserves table

INSERT ALL
 INTO Reserves VALUES (1, 101, TO_DATE('2025-03-01', 'YYYY-MM-DD'))
 INTO Reserves VALUES (2, 103, TO_DATE('2025-03-02', 'YYYY-MM-DD'))
```



INTO Reserves VALUES (3, 104, TO\_DATE('2025-03-01', 'YYYY-MM-DD')) INTO Reserves VALUES (4, 102, TO\_DATE('2025-03-03', 'YYYY-MM-DD')) INTO Reserves VALUES (5, 101, TO\_DATE('2025-03-04', 'YYYY-MM-DD')) INTO Reserves VALUES (2, 105, TO\_DATE('2025-03-05', 'YYYY-MM-DD')) INTO Reserves VALUES (3, 101, TO\_DATE('2025-03-06', 'YYYY-MM-DD')) INTO Reserves VALUES (4, 103, TO\_DATE('2025-03-06', 'YYYY-MM-DD')) SELECT \* FROM DUAL:

# 1. Find all information of sailors who have reserved boat number 101.

SELECT S.\* FROM Sailors S
JOIN Reserves R ON S.sid = R.sid

WHERE R.bid = 101; sid sname rating age

sia	sname	rating	age
1	Alice	7	23.5
5	Eve	3	21.7
3	Charlie	6	35.2

#### 2. Find the name of boat reserved by Bob.

SELECT DISTINCT B.bname FROM Boats B JOIN Reserves R ON B.bid = R.bid JOIN Sailors S ON S.sid = R.sid WHERE S.sname = 'Bob';

bname
Sunshine
Blue Pearl

# 3. Find the names of sailors who have reserved a red boat, and list in the order of age.

SELECT S.sname,S.age FROM Sailors S JOIN Reserves R ON S.sid = R.sid JOIN Boats B ON R.bid = B.bid WHERE B.color = 'red' ORDER BY S.age;

sname Eve Alice Charlie

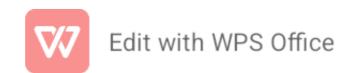
#### 4. Find the names of sailors who have reserved at least one boat.

SELECT DISTINCT S.sname FROM Sailors S JOIN Reserves R ON S.sid = R.sid;

sname
Alice
Bob
Charlie
David
Eve

#### 5. Find the ids of sailors who have reserved a red boat or a green boat.

SELECT DISTINCT R.sid FROM Reserves R JOIN Boats B ON R.bid = B.bid



WHERE B.color IN ('red', 'green');

sid		
1		
2		
3		
4		

6. Find the name and the age of the youngest sailor.

SELECT sname, age FROM Sailors
WHERE age = (SELECT MIN(age) FROM Sailors);
sname age

# **sname age** Eve 21.7

#### 7. Count the number of different sailor names

SELECT COUNT(DISTINCT sname) AS unique\_sailor\_names FROM Sailors;

unique_sailor_names	
5	

# 8. Find the average age of sailors for each rating level.

SELECT rating, AVG(age) AS avg\_age FROM Sailors GROUP BY rating;

rating	avg_age
3	21.7
5	29.0
6	35.2
7	23.5
8	40.0

# 9. Find the average age of sailors for each rating level that has at least two sailors.

SELECT rating, AVG(age) AS avg\_age FROM Sailors GROUP BY rating HAVING COUNT(sid) >= 2;

 rating
 avg\_age

 5
 28.5

 6
 34.7

