ORACLE SQL

Lesson 03: Group functions, Joins, Sub queries, set operators.

Lesson Objectives

To understand the following topics:

- Introduction to Functions
- Aggregate (Group) functions:
  - GROUP BY clause
  - HAVING clause
- Joins
  - Oracle Proprietary Joins
  - SQL: 1999 Compliant Joins
- Types of joins
- Sub-queries
- Use of Set Operations

3.1: Group Functions

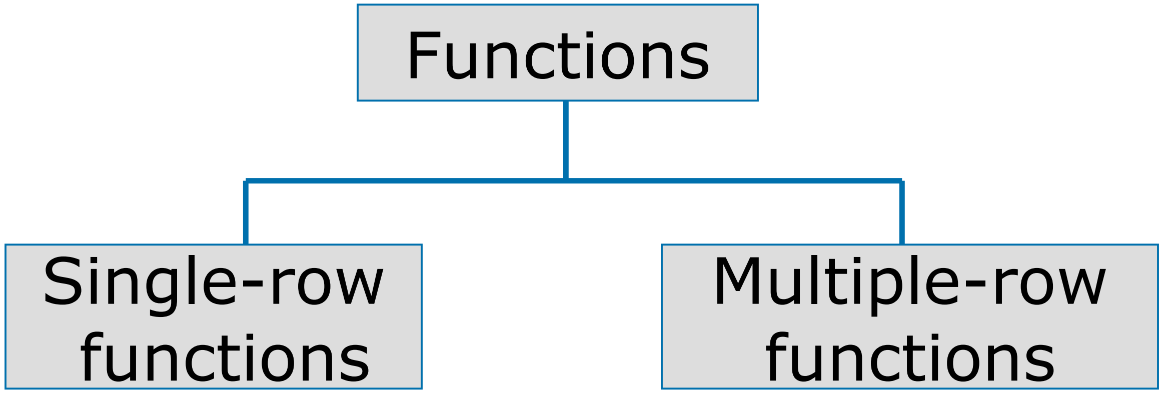
Types of SQL Functions

Single row functions :

- Operate on single rows only and return one result per row

Multiple row functions:

- Manipulates groups of rows to give one result per group of rows. Also called as group functions



3.1: Group Functions

The Group Functions

The Group functions are built-in SQL functions that operate on “groups of rows”, and return one value for the entire group.
The results are also based on groups of rows.
For Example, Group function called “SUM” will help you find the total marks, even if the database stores only individual subject marks.

Syntax : GROUP BY & HAVING clause

Syntax

```
SELECT          [column, ] aggregate function(column), ……
FROM       table
[WHERE          condition]
[GROUP BY       column]
[HAVING         condition]
[ORDER BY column] ;
```

Listing of Group Functions

- Given below is a list of Group functions supported by SQL:

Function	**Value returned**
SUM (expr)	Sum value of expr, ignoring NULL values.
AVG (expr)	Average value of expr, ignoring NULL values.
COUNT (expr)	Number of rows where expr evaluates to something other than NULL. COUNT(*) counts all selected rows, including duplicates and rows with NULLs.
MIN (expr)	Minimum value of expr.
MAX (expr)	Maximum value of expr.

3.1: Group Functions

Examples of using Group Functions

Example 1: Display the total number of records from student_marks.

Exampl

```
        SELECT COUNT( * )
                FROM Student_Marks;
```

```
SELECT  AVG(Student_sub1), AVG(Student_sub2), AVG(Student_sub3)
        FROM Student_Marks;
```

The GROUP BY clause

GROUP BY clause is used along with the Group functions to retrieve data that is grouped according to one or more columns.

- For example: Displays the average staff salary based on every department. The values are grouped based on dept_code

```
SELECT Dept_Code, AVG(Staff_sal)
        FROM Staff_Master
       GROUP BY  Dept_Code;
```

3.1: Group Functions

The HAVING clause

HAVING clause is used to filter data based on the Group functions.

- HAVING clause is similar to WHERE condition. However, it is used with Group functions.

Group functions cannot be used in WHERE clause. However, they can be used in HAVING clause.

Examples – GROUP BY and HAVING clause

For example: Display all department numbers having more than five employees.

```
SELECT Department_Code, Count(*)
        FROM Staff_Master
        GROUP BY Department_Code
        HAVING Count(*)> 5;
```

3.1: Group Functions
Quick Guidelines

All group functions except COUNT(*) ignores NULL values.
To substitute a value for NULL values use NVL functions.
DISTINCT clause makes the function consider only non duplicate values.
The AVG and SUM are used with numerica data.
The MIN and MAX functions used with any data type.

3.1: Group Functions
Quick Guidelines

All individual columns included in the SELECT clause other than group functions must be specified in the GROUP BY clause.
Any column other than selected column can also be placed in GROUP BY clause.
By default rows are sorted by ascending order of the column included in the GROUP BY list.
WHERE clause specifies the rows to be considered for grouping.

3.1: Group Functions
Quick Guidelines

Suppose your SELECT statement contains a HAVING clause. Then write your query such that the WHERE clause does most of the work (removing undesired rows) instead of the HAVING clause doing the work of removing undesired rows.

Use the GROUP BY clause only with an Aggregate function, and not otherwise.

- Since in other cases, you can accomplish the same end result by using the DISTINCT option instead, and it is faster.

3.2: Joins
What are Joins?

If we require data from more than one table in the database, then a join is used.

- Tables are joined on columns, which have the same “data type” and “data width” in the tables.
- The JOIN operator specifies how to relate tables in the query.
  - When you join two tables a Cartesian product is formed, by default.
- Oracle supports
  - Oracle Proprietary
  - SQL: 1999 Compliant Joins

Types of Joins

- Given below is a list of JOINs supported by Oracle:

Oracle Proprietary Joins	**SQL: 1999 Compliant Joins**
Cartesian Product	Cross Joins
Equijoin	Inner Joins (Natural Joins)
Outer-join	Left, Right, Full outer joins
Non-equijoin	Join on
Self-join	Join on

Cartesian Joins

A Cartesian product is a product of all the rows of all the tables in the query.
A Cartesian product is formed when the join condition is omitted or it is invalid
To avoid having Cartesian product always include a valid join condition
Example

```
SELECT Student_Name, Dept_Name
       FROM  Student_Master,
Department_Master;
```

3.2: Joins

Guidelines for Joining Tables

The JOIN condition is written in the WHERE clause

The column names which appear in more than one table should be prefixed with the table name

To improve performance of the query, table name prefix can be include for the other selected columns too

3.2: Joins

EquiJoin

In an Equijoin, the WHERE statement compares two columns from two tables with the equivalence operator "=".
This JOIN returns all rows from both tables, where there is a match.
Syntax :

```
SELECT <col1>, <col2>,...
    FROM <table1>,<table2>
    Where <table1>.<col1>=<table2>.<col2>
    [AND <condition>] [ORDER BY <col1>, <col2>,...]
```

3.2: Joins

EquiJoin - Example

Example 1: To display student code and name along with the department name to which they belong

Example 2: To display student and staff name along with the department name to which they belong

```
SELECT Student_Code,Student_name,Dept_name
  FROM Student_Master ,Department_Master
  WHERE Student_Master.Dept_code =Department_Master.Dept_code;
```

```
SELECT student_name,staff_name, dept_name
      FROM student_master, department_master,staff_master
      WHERE student_master.dept_code=department_master.dept_code
       and staff_master.dept_code=department_master.dept_code;
```

Non-EquiJoin

A non-equi join is based on condition other than an equality operator
Example: To display details of staff_members who receive salary
in the range defined as per grade

```
SELECT  s.staff_name,s.staff_sal,sl.grade
        FROM staff_master s,salgrade sl
        WHERE staff_sal BETWEEN sl.losal and sl.hisal
```

3.2: Joins
Outer Join

If a row does not satisfy a JOIN condition, then the row will not appear in the query result.
The missing row(s) can be returned by using OUTER JOIN operator in the JOIN condition.
The operator is PLUS sign enclosed in parentheses (+), and is placed on the side of the join(table), which is deficient in information.

3.2: Joins

Outer Join

Syntax

Table1.column = table2.column (+) means OUTER join is taken on table1.

The (+) sign must be kept on the side of the join that is deficient in information

Depending on the position of the outer join (+), it can be denoted as Left Outer or Right outer Join

```
WHERE table1 <OUTER JOIN INDICATOR> = table 2
```

Outer Join - Example

To display Department details which have staff members and also display department details who do not have any staff members

```
SELECT staff.staff_code,staff.Dept_Code,dept.Dept_name
 FROM Staff_master staff, Department_Master dept
 WHERE staff.Dept_Code(+) = dept.Dept_Code
```

3.2: Joins

Self Join

In Self Join, two rows from the "same table" combine to form a "resultant row".

- It is possible to join a table to itself, as if they were two separate tables, by using aliases for table names.
- This allows joining of rows in the same table.

Example: To display staff member information along with their manager information

```
SELECT staff.staff_code, staff.staff_name,
                mgr.staff_code, mgr.staff_name
FROM staff_master staff, staff_master mgr
WHERE staff.mgr_code  = mgr.staff_code;
```

SQL: 1999 Compliant Joins - Syntax

Syntax:

```
SELECT table1.column, table2.column
FROM table1
[CROSS JOIN table2] |
[NATURAL JOIN table2] |
[JOIN table2 USING (column_name)] |
[JOIN table2 ON (table1.column_name =
             table2.column_name)] |
[LEFT|RIGHT|FULL OUTER JOIN table2
ON (table1.column_name = table2.column_name)];
```

3.2: Joins
Cross Join

The Cross Join and Cartesian product are same which produces the cross-product of the tables
Example: Cross Join on Student_Master and Department_Master

```
SELECT student_name, dept_name
 FROM student_master
CROSS JOIN department_master;
```

Natural Join

The Natural Join is based on the all columns that have same name and datatype in the tables include in the query
All the rows that have equal values in the matched columns are fetched
Example: To display student details along with their department
details

```
SELECT  Student_Code,Student_name,Dept_Code, Dept_name
        FROM Student_Master
        NATURAL JOIN Department_Master
```

3.2: Joins

USING clause

The USING clause can be replace the NATURAL JOIN if the columns have same names but data types do not match.
The table name or aliases should not be used in the referenced columns
This clause should be used to match only one column when there are more than one column matches

USING clause - Example

Example 1: To display student details along with their department details. The department code does not match in datatype, hence the join is performed with the USING clause

```
SELECT student_code, student_name, dept_code, dept_name
        FROM student_master
        JOIN department_master
        USING (dept_code, dept_code);
```

ON clause

Explicit join condition can be specified by using ON clause
Other search conditions can be specified in addition to join condition
Example: To display student along with department details from Computer Science department

```
SELECT student.student_code, student.student_name,
student.dept_code, dept.dept_name
        FROM student_master student
         JOIN department_master dept
        ON (student.dept_Code = dept.dept_Code)
         AND dept.dept_Name =‘Computer Science’ ;
```

3.2: Joins

LEFT, RIGHT & FULL Outer Join

A join between two tables that return rows that match the join condition and also unmatched rows from left table is LEFT OUTER JOIN

A join between two tables that return rows that match the join condition and unmatched rows from the right table is RIGHT OUTER JOIN

A join between two tables that return rows that match the join condition and returns unmatched rows of both left and right table is a full outer join

3.2: Joins

LEFT, RIGHT & FULL Outer Join - Example

Example 1: Display student & department details and also those departments who do have students

Exan

```
SELECT s.student_code, s.dept_code, d.dept_name
        FROM student_master s
        RIGHT OUTER JOIN department_master d
        ON (s.dept_code = d.dept_code);
```

```
SELECT s.student_code, s.dept_code, d.dept_name
        FROM student_master s
        LEFT OUTER JOIN department_master d
        ON (s.dept_code = d.dept_code);
```

LEFT, RIGHT & FULL Outer Join - Example

Example 3: Display student & department details. Also those departments who do have students and students who are not assigned to any department

```
SELECT s.student_code,s.dept_code,d.dept_name
        FROM student_master s
        FULL OUTER JOIN department_master d
        ON (s.dept_code = d.dept_code );
```

3.3: Subqueries
What is a SubQuery?

A sub-query is a form of an SQL statement that appears inside another SQL statement.

- It is also called as a “nested query”.

The statement, which contains the sub-query, is called the “parent statement”.

The “parent statement” uses the rows returned by the sub-query.

Subquery - Examples

Example 1: To display name of students from "Mechanics" department.

- Method 1:

O/P : 40

```
SELECT Dept_Code
   FROM Department_Master
   WHERE Dept_name = 'Mechanics';
```

```
SELECT student_code,student_name
FROM student_master
WHERE dept_code=40;
```

Subquery - Examples

Example 1 (contd.):

- Method 2: Using sub-query

```
SELECT student_code, student_name
           FROM student_master
          WHERE dept_code = (SELECT dept_code
                          FROM department_master
                           WHERE dept_name = ’Mechanics’);
```

3.3: Subqueries

Where to use Subqueries?

Subqueries can be used for the following purpose :

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

Comparison Operators for Subqueries

- Types of SubQueries
  - Single Row Subquery
  - Multiple Row Subquery.
- Some comparison operators for subqueries:

Operator	**Description**
IN	Equals to any member of
NOT IN	Not equal to any member of
*ANY	compare value to every value returned by sub-query using operator *
*ALL	compare value to all values returned by sub-query using operator *

Using Comparison Operators - Examples

Example 1: To display all staff details of who earn salary least salary

Examp

```
    SELECT staff_name, staff_code, staff_sal
        FROM staff_master
          WHERE staff_sal  = (SELECT MIN(staff_sal)
                              FROM staff_master) ;
```

```
     SELECT staff_code,staff_sal
          FROM staff_master
          WHERE staff_sal > ANY (SELECT AVG(staff_sal)
                                FROM staff_master GROUP BY
dept_code);
```

What is a Co-related Subquery?

A sub-query becomes “co-related”, when the sub-query references a column from a table in the “parent query”.

- A co-related sub-query is evaluated once for each row processed by the “parent statement”, which can be either SELECT, UPDATE, or DELETE statement.
- A co-related sub-query is used whenever a sub-query must return a “different result” for each “candidate row” considered by the “parent query”.

Co-related Subquery -Examples

Example 2: To display staff details whose salary is greater than the average salary in their own department:

```
            SELECT staff_name, staff_sal , dept_code
              FROM staff_Master  s
              WHERE  staff_sal > (SELECT AVG(staff_sal)
                                  FROM staff_Master m
                                    WHERE  s.dept_code  = m.dept_code
      );
```

3.3: Subqueries

EXISTS/ NOT EXISTS Operator

The EXISTS / NOT EXISTS operator enables to test whether a value retrieved by the Outer query exists in the result-set of the values retrieved by the Inner query.

- The EXISTS / NOT EXISTS operator is usually used with a co-related sub-query.
  - If the query returns at least one row, the operator returns TRUE.
  - If the value does not exist, it returns FALSE.
- 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.

3.3: Subqueries

EXISTS/ NOT EXISTS Operator - Examples

Example 1: To display details of employees who have some other employees reporting to them.

E

```
SELECT staff_code, staff_name
      FROM staff_master staff
      WHERE  EXISTS (SELECT mgr_code FROM staff_master mgr
WHERE         mgr.mgr_code = staff.staff_code) ;
```

```
SELECT dept_code,dept_name
      FROM department_master
      WHERE  EXISTS ( SELECT dept_code FROM staff_master
                              WHERE staff_master.dept_code =
                              department_master.dept_code) ;
```

Tips and Tricks
Quick Guidelines

For Using Subqueries

- Should be enclosed in parenthesis
- They should be placed on the right side of the comparison condition
- Cannot use ORDER By clause in subquery unless performing top-n analysis
- Use operator carefully. Single Row operators for Single Row Subquery and Multiple Row operator for Multiple Row Subquery

Quick Guidelines

If Single row operators are used for a sub query that returns multiple rows, Oracle would throw an error
If one of the value returned by the sub query is NULL, then the whole sub query returns NULL.
Correlate sub queries hamper performance , use them only when absolutely required

Quick Guidelines

Restrict using the NOT IN clause, which offers poor performance because the optimizer has to use a nested table scan to perform this activity.

Instead try to use one of the following options, all of which offer better performance:

- Use EXISTS or NOT EXISTS
- Use IN
- Perform a LEFT OUTER JOIN and check for a NULL condition

Quick Guidelines

If you have a choice of using the IN or the EXISTS clauses in your SQL, use the EXISTS clause as it is usually more efficient and performs faster.

- Consider EXISTS in place of table joins.
- Consider NOT EXISTS in place of NOT IN.

3.4: Set Operation
SET Operators in Oracle

SQL supports the following four Set operations:

- UNION ALL
  - Combines the results of two SELECT statements into one result set.
- UNION
  - Same as UNION ALL. Eliminates duplicate rows from that result set.
- MINUS
  - Takes the result set of one SELECT statement, and removes those rows that are also returned by a second SELECT statement.
- INTERSECT
  - Returns only those rows that are returned by each of two SELECT statements.

3.4: Set Operation

SET Operators in Oracle

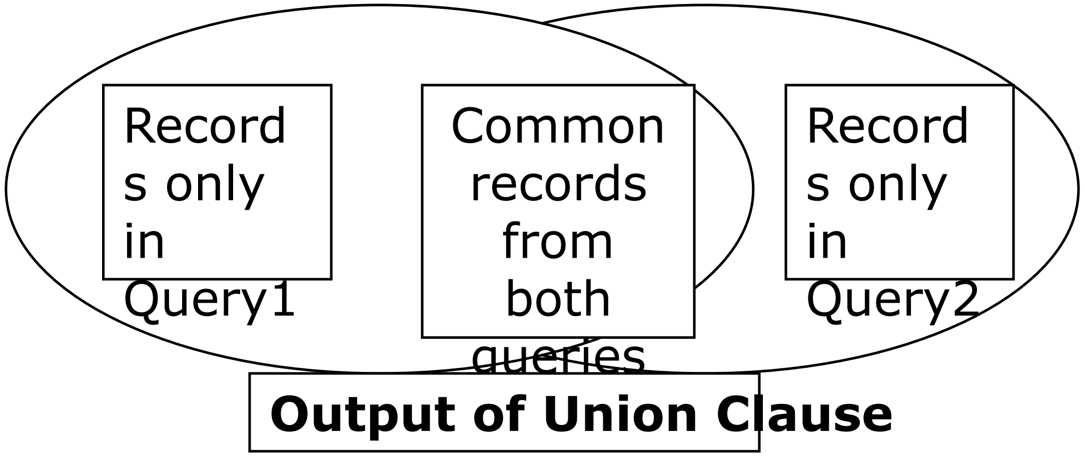
Each of these operations combines the results of two SELECT statements into a single result.

Note: While using SET operators, the column names from the first query appear in the result set.

3.4: Set Operation

UNION Operator

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



3.4: Set Operation

UNION Operator- Example

Example: To display all students who are listed for 2006, 2007 and both the years.

```
SELECT Student_Code
            FROM Student_Marks
                        WHERE Student_year=2006
             UNION
            SELECT Student_Code
            FROM  Student_Marks
                        WHERE Student_year=2007;
```

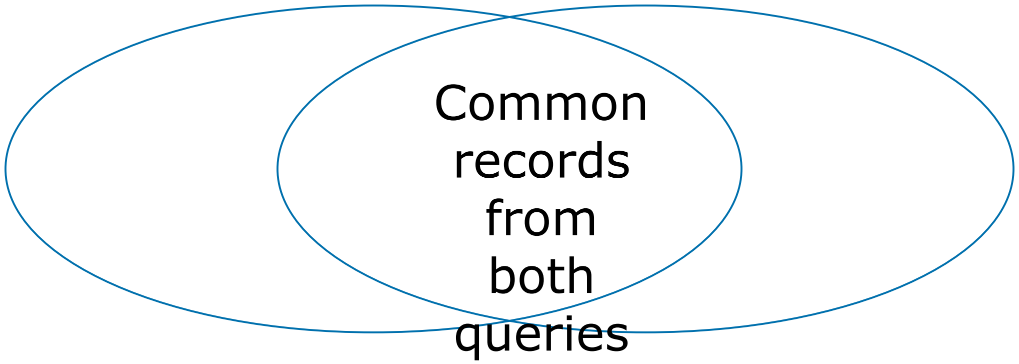
UNION Operator- Example

Some situations, if you need duplicate row as well use UNION ALL Operator

```
SELECT Student_Code
            FROM Student_Marks
            WHERE Student_year=2006
            UNION ALL
 SELECT Student_Code
            FROM  Student_Marks
            WHERE Student_year=2007;
```

INTERSECT Operator

The INTERSECT operator returns those rows, which are retrieved by both the queries.



Output of Intersect Clause

3.4: Set Operation

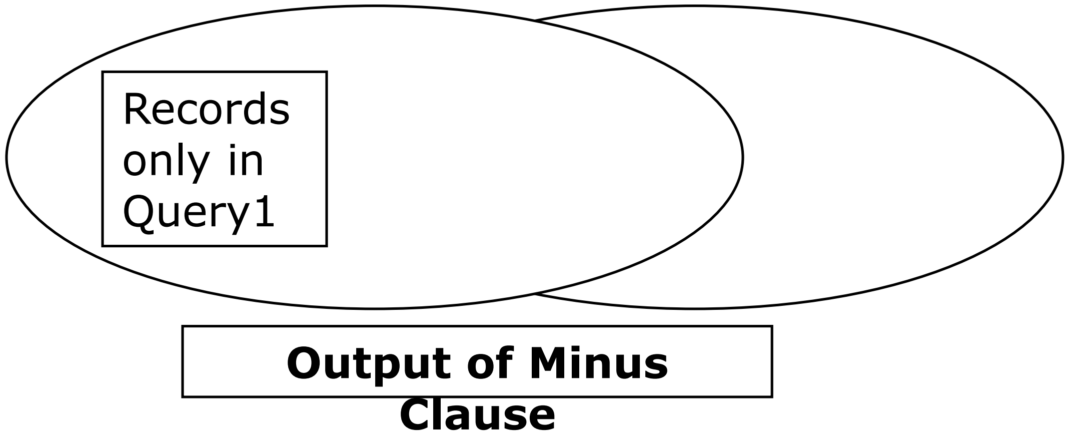
INTERSECT Operator – Example

Example : To display students who are listed for both the years

```
SELECT Student_Code
    FROM Student_Marks
   WHERE Student_year=2006
   INTERSECT
    SELECT Student_Code
    FROM Student_Marks
   WHERE Student_year=2007;
```

MINUS Operator

The MINUS operator returns all rows retrieved by the first query but not by the second query.



MINUS Operator - Example

Example: To display all students who are listed only for year 2006

```
SELECT Student_Code
   FROM Student_Marks
   WHERE Student_year=2006
   MINUS
    SELECT Student_Code
   FROM Student_Marks
   WHERE Student_year=2007;
```

Tips and Tricks
Quick Guidelines

Use UNION ALL in place of UNION.

- The UNION clause forces all rows returned by each portion of the union to be sorted, merged and filtered for duplicates before the first row is returned to the “calling module”.
- A UNION ALL simply returns all rows including duplicates. It does not perform SORT, MERGE and FILTER.

SUMMARY

- In this lesson you have learnt,
  - Aggregate (Group functions)
    - GROUP BY clause
    - HAVING clause
  - Joins
    - Oracle Proprietary Joins
    - SQL: 1999 Compliant Joins
  - Types of joins
  - Sub-queries
  - Use of Set Operations

Review – Questions

❖ Question 1: The AVG function ignores NULL values in the column.

- True / False

❖ Question 2: A sub-query can be used for creating and inserting records.

- True / False

❖ Question 3: The Set operation that will show all the rows from both the resultsets including duplicates is ____.

- Option 1: Union All
- Option 2: Union
- Option 3: Intersect
- Option 4: Minus

Review – Questions

- ❖ Question 4: The Intersect operator returns ____.
- ❖ Question 5: The output of set operators shows the columns names from ____.