Unit – II
SQL In built functions and Joins

2.1 Operators Arithmetic, Comparison, Logical SQL functions- Single row function

Operators are the foundation of any programming language. We can define operators as symbols that help us to perform specific mathematical and logical computations on operands. In other words, we can say that an operator operates the operands. SQL operators have three different categories.

- Arithmetic operator
- Comparison operator
- Logical operator

Arithmetic operators:

We can use various arithmetic operators on the data stored in the tables. Arithmetic Operators are:

Operator	Description
+	The addition is used to perform an addition operation on the data values.
–	This operator is used for the subtraction of the data values.
/	This operator works with the 'ALL' keyword and it calculates division operations.
*	This operator is used for multiply data values.
%	Modulus is used to get the remainder when data is divided by another.

Example:

SELECT employee_id, employee_name, salary, salary + 100 FROM employee;

Comparison operators:

❖ Another important operator in SQL is a comparison operator, which is used to compare one expression's value to other expressions.

❖ SQL supports different types of the comparison operator, which is described below:

Operator	Description
=	Equal to.
>	Greater than.
<	Less than.
>=	Greater than equal to.
<=	Less than equal to.
<>	Not equal to.

Example:

```
SELECT *  FROM Employee WHERE salary < 20000;
```

Logical operators:

The Logical operators are those that are true or false. They return true or false values to combine one or more true or false values.

Operator	Description
AND	Logical AND compares between two Booleans as expressions and returns true when both expressions are true.

OR	Logical OR compares between two Booleans as expressions and returns true when one of the expressions is true.
NOT	Not takes a single Boolean as an argument and changes its value from false to true or from true to false.

❖ AND Operator:

This operator displays only those records where both the conditions condition1 and condition2 evaluates to True.
Syntax:

SELECT * FROM table_name WHERE condition1 AND condition2 and ...conditionN;

❖ OR Operator:

This operator displays the records where either one of the conditions condition1 and condition2 evaluates to True. That is, either condition1 is True or condition2 is True.

Syntax:

SELECT * FROM table_name WHERE condition1 OR condition2 OR... conditionN;

❖ NOT Operator:

The **NOT operator** in SQL shows the record from the table if the condition evaluates to false. It is always used with the WHERE clause.

Syntax:

SELECT column1, column2 ...., columnN FROM table_Name WHERE NOT condition;

<u>SQL functions - Single row function.</u>

1. Date functions (add-months, months-between, round, truncate)

1) ADD_MONTHS

❖ The add_months function returns a new date after adding n months.

Syntax: ADD_MONTHS (Date1, n)

- Date1 is the starting date (before the n months have been added).
- n is the number of months to add to date1.

Example:

SELECT ADD_MONTHS ('01-Aug-23', 3) FROM dual;
Output: '01-Nov-23‘

SELECT ADD_MONTHS ('01-Aug-23', -3) FROM dual;
Output: '01-May-23'

SELECT ADD_MONTHS ('21-Aug-23', -3) FROM dual;
Output: '21-May-23'

SELECT ADD_MONTHS ('31-Jan-23', 1) FROM dual;
Output: '28-Feb-23'

2) MONTHS_BETWEEN

- Returns the months in between (separating) two given dates

Syntax: MONTHS_BETWEEN (Date1, Date2)

- Date1 and Date2 are the dates used to calculate the number of months.

Example:

SELECT MONTHS_BETWEEN (‘31-MAR-23‘, ‘31-DEC-22‘) FROM dual;
Output: 3

SELECT MONTHS_BETWEEN (‘2003/07/23‘, ‘2003/07/23‘) FROM dual;
Output: 0

SELECT MONTHS_BETWEEN (‘2003/08/23‘, ‘2003/06/23‘) FROM dual;
Output: 2

3) ROUND

- ROUND function returns a date rounded to a specific unit of measure.

Syntax: ROUND (date, [format]);

- Date is the date to round.
- Format is the unit of measure to apply for rounding.
- If the format parameter is omitted, the ROUND function will round to the nearest day.

Example:

ROUND (TO_DATE ('22-AUG-23'),'YEAR') would return '01-JAN-24'
ROUND (TO_DATE ('22-AUG-23'),'Q') would return '01-OCT-23'
ROUND (TO_DATE ('22-AUG-23'),'MONTH') would return '01-SEP-23'
ROUND (TO_DATE ('22-AUG-23'),'DDD') would return '22-AUG-23'
ROUND (TO_DATE ('22-AUG-23'),'DAY') would return '24-AUG-23'

4) TRUNCATE

- The Oracle/PLSQL TRUNC function returns a date truncated to a specific unit of measure.

Syntax: TRUNC (Date, [Format])

- Date is the date to truncate.
- Format is the unit of measure to apply for truncating.
- If the format parameter is omitted, the TRUNC function will truncate the date to the day value, so that any hours, minutes, or seconds will be truncated off.

Example:

TRUNC (TO_DATE ('22-AUG-23'), 'YEAR') would return '01-JAN-23'
TRUNC (TO_DATE ('22-AUG-23') 'MONTH') would return '01-AUG-23'

2. Numeric Functions (abs, power, mod, round, trunc, sqrt)

- Numeric functions are used to perform operations on numbers. They accept numeric values as input and return numeric values as output. Few of the Numeric functions are:

Function Name	**Description**	**Example**
ABS (x)	Absolute value of the number '*x*'	Select ABS(-15) from dual;
SQRT(x)	Return square root of ‘n’.	Select SQRT(64) from dual;
POWER(m,n)	Return m raised to nth Power.	Select POWER(2,3) from dual;
MOD(m,n)	Return remainder of m divided by n operations.	Select MOD(10,2) from dual;
CEIL (x)	Integer value that is Greater than or equal to the number '*x*'	Select CEIL(25.57) from dual;

FLOOR (x)	Integer value that is Less than or equal to the number '*x*'	Select FLOOR(25.57) from dual;
TRUNC (x, y)	Truncates value of number '*x*' up to '*y*' decimal places	Select TRUNC(25.57,2) from dual;
ROUND (x, y)	Rounded off value of the number '*x*' up to the number '*y*' decimal places	Select ROUND(25.57,2) from dual;

3. Character Fucntions (initcap, lower, upper, ltrim, rtrim, replace, substring, instr)

Function Name	**Examples**	**Return Value**
LOWER(string_value)	LOWER('Good Morning')	good morning
UPPER(string_value)	UPPER('Good Morning')	GOOD MORNING
INITCAP(string_value)	INITCAP('GOOD MORNING')	Good Morning
LTRIM(string_value, trim_text)	LTRIM ('Good Morning', 'Good)	Morning
RTRIM (string_value, trim_text)	RTRIM ('Good Morning', ' Morning')	Good
TRIM (trim_text FROM string_value)	TRIM ('o' FROM 'Good Morning')	Gd Mrning
SUBSTR (string_value, m, n)	SUBSTR ('Good Morning', 6, 7)	Morning
LENGTH (string_value)	LENGTH ('Good Morning')	12
LPAD (string_value, n, pad_value)	LPAD ('Good', 6, '*')	**Good
RPAD (string_value, n, pad_value)	RPAD ('Good', 6, '*')	Good**

4. Conversion Functions (to-char, to-date, to-number)

Function Name	**Examples**	**Return Value**
TO_CHAR ()	TO_CHAR (3000, '$9999') TO_CHAR (SYSDATE, 'Day, Month YYYY')	$3000 Monday, June 2023
TO_DATE ()	TO_DATE ('01-Jun-23')	01-Jun-23
TO_NUMBER()	TO_NUMBER('1234.56')	1234.56
NVL ()	NVL (null, 1)	1

2.2 Groupby, Having and Order by clause

1) SQL GROUP BY Clause

- ❖ The SQL GROUP BY Clause is used along with the group functions to retrieve data grouped according to one or more columns.
- ❖ **Syntax:**
  Select column1, column2……. columnN, Aggregate Function (argument) From TableName Group By column1, column2……. columnN;
- ❖ **Example:**
  If you want to know the total amount of salary spent on each department, the query would be:

Id	**Name**	**Dept**	**Age**	**Salary**	**Location**
100	Ramesh	Electrical	24	25000	Bangalore
101	Hrithik	Electronics	28	35000	Bangalore
102	Harsha	Aeronautics	28	35000	Mysore
103	Soumya	Electronics	22	20000	Bangalore
104	Priya	InfoTech	25	30000	Mangalore

Example:

SELECT dept, SUM (salary) FROM employee GROUP BY dept;

The output would be like:

Dept	**Salary**
Electrical	25000
Electronics	55000
Aeronautics	35000
InfoTech	30000

2) SQL ORDER BY

❖ The ORDER BY clause is used in a SELECT statement to sort results either in ascending or descending order. Oracle sorts query results in ascending order by default.

Syntax:

Select column1, column2……. columnN From TableName ORDER BY column1, column2……. columnN;

Id	**Name**	**Dept**	**Age**	**Salary**	**Location**
100	Ramesh	Electrical	24	25000	Bangalore
101	Hrithik	Electronics	28	35000	Bangalore
103	Soumya	Electronics	22	20000	Bangalore
104	Priya	InfoTech	25	30000	Mangalore

Example:

SELECT name, salary FROM employee ORDER BY salary;

❖ The output would be like

Name	**Salary**
Soumya	20000
Ramesh	25000
Priya	30000
Hrithik	35000

3) SQL HAVING Clause

❖ Having clause is used to filter data based on the group functions. This is similar to WHERE condition but is used with group functions.

❖ Group functions cannot be used in WHERE Clause but can be used in HAVING clause.

- ❖ The HAVING clause must follow the GROUP BY clause in a query and must also precedes the ORDER BY clause if used.
- ❖ The following is the syntax of the SELECT statement, including the HAVING clause:

Syntax:
SELECT column1, column2 FROM TableName WHERE [conditions] GROUP BY column1, column2 HAVING [conditions];

Example:
SELECT dept, SUM (salary) FROM employee GROUP BY dept HAVING SUM (salary) > 25000;

2.3 Joins: Simple, Equi-join, Non-equi, Self-Joins, Outer-joins.

- ❖ SQL Joins are used to relate information in different tables.
- ❖ A Join condition is a part of the sql query that retrieves rows from two or more tables.
- ❖ A SQL Join condition is used in the SQL WHERE Clause of select, update, delete statements.

Syntax:
SELECT col1, col2, col3...FROM table_name1, table_name2 WHERE table_name1.col2 = table_name2.col1;

- ❖ If a sql join condition is omitted or if it is invalid the joint operation will result in a Cartesian product.
- ❖ The Cartesian product returns a number of rows equal to the product of all rows in all the tables being joined.

Example, if the first table has 20 rows and the second table has 10 rows, the result will be 20 * 10 or 200 rows. This query takes a long time to execute.

- ❖ Let‘s use the below two tables to explain the sql join conditions.

Product_Id	**Product_Name**	**Supplier_Name**	**Price**
100	Camera	Nikon	300

101	Television	Onida	100
102	Refrigerator	Videocon	150
103	Ipod	Apple	75
104	Mobile	Nokia	50

Table Name: Product

Order_id	**Product_Id**	**Total_Unit**	**Custmer**
5100	104	30	Infosys
5101	103	5	Satyam
5102	102	25	Wipro
5103	101	10	TCS

Table Name: Order_Item

❖ SQL Joins can be classified into Equi join and Non Equi joins.

1) SQL Equi joins

❖ It is a simple sql join condition which uses the equal sign as the comparison operator. Two types of equi joins are SQL Outer join and SQL Inner join.

❖ **Example:** You can get the information about a customer who purchased a product and the quantity of product.

❖ **An equi-join is further classified into two categories: 1)** SQL Inner Join 2) SQL Outer Join

A) SQL Inner Join:

❖ All the rows returned by the sql query satisfy the sql join condition specified.
For example:

❖ If you want to display the product information for each order the query will be as given below. Since you are retrieving the data from two tables, you need to identify the common column between these two tables, which is the product_id.

❖ The query for this type of sql joins would be like,

SELECT order_id, product_name, unit_price, supplier_name, total_units FROM product, Oreder_Item WHERE order_items.product_id = product.product_id;

- The columns must be referenced by the table name in the join condition, because product_id is a column in both the tables and needs a way to be identified. This avoids ambiguity in using the columns in the SQL SELECT statement.
- The number of join conditions is (n-1), if there are more than two tables joined in a query where 'n' is the number of tables involved. The rule must be true to avoid Cartesian product.
- We can also use aliases to reference the column name, then the above query would be like,
  SELECT o.order_id, p.product_name, p.unit_price, p.supplier_name, o.total_units FROM product p, Oreder_Item oWHERE o.product_id = p.product_id;

B) SQL Outer Join:

- This sql join condition returns all rows from both tables which satisfy the join condition along with rows which do not satisfy the join condition from one of the tables.
- The sql outer join operator in Oracle is (+) and is used on one side of the join condition only.
- The syntax differs for different RDBMS implementation. Few of them represent the join conditions as "sql left outer join", "sql right outer join".
- If you want to display all the product data along with order items data, with null values displayed for order items if a product has no order item, the sql query for

Outer join would be as shown below:

SELECT p.product_id, p.product_name, o.order_id, o.total_units FROM Oreder_Item o, product p WHERE o.product_id (+) = p.product_id;

The output would be like,

product_id	product_name	order_id	total_units
100	Camera	5104	15
101	Television	5103	10
102	Refrigerator	5101	5
103	IPod	5102	25
104	Mobile	5100	30

NOTE:

If the (+) operator is used in the left side of the join condition it is equivalent to left outer join. If used on the right side of the join condition it is equivalent to right outer join.

C) SQL Self Join:

- ❖ A Self Join is a type of sql join which is used to join a table to it, particularly when the table has a FOREIGN KEY that references its own PRIMARY KEY.
- ❖ It is necessary to ensure that the join statement defines an alias for both copies of the table to avoid column ambiguity.

The below query is an example of a self join,

```
SELECT a.sales_person_id, a.name, a.manager_id, b.sales_person_id, b.name FROM
sales_person a, sales_person b WHERE a.manager_id = b.sales_person_id;
```

2) SQL Non equi joins

- ❖ A Non Equi Join is a SQL Join whose condition is established using all comparison operators except the equal (=) operator. Like >=, <=, <, >

For example:

```
SELECT first_name, last_name, subject FROM student_details WHERE subject! =
'computer'
```

- ❖ The output would be something like,

First_name	Last_name	Subject
Anajali	Bhagwat	Maths
Shekar	Gowda	Maths
Rahul	Sharma	Science
Stephen	Fleming	Science

2.4 Subqueries - Multiple, Correlated

- ❖ A Subquery or Inner query or Nested query is a query within another SQL query and embedded within the WHERE clause.
- ❖ A Subquery is used to return data that will be used in the main query as a condition to further restrict the data to be retrieved.
- ❖ Subquery can be used with the SELECT, INSERT, UPDATE, and DELETE statements along with the operators like =, <, >, >=, <=, IN, BETWEEN etc.

There are a few rules that Sub queries must follow:

- ❖ Subquery must be enclosed within parentheses.
- ❖ A Subquery can have only one column in the SELECT clause, unless multiple columns are in the main query for the Subquery to compare its selected columns.
- ❖ An ORDER BY cannot be used in a Subquery, although the main query can use an ORDER BY.
- ❖ The GROUP BY can be used to perform the same function as the ORDER BY in a Subquery.
- ❖ Subquery that return more than one row can only be used with multiple value operators, such as the IN operator.
- ❖ The SELECT list cannot include any references to values that evaluate to a BLOB, ARRAY, CLOB, or NCLOB.
- ❖ A Subquery cannot be immediately enclosed in a set function.
- ❖ The BETWEEN operator cannot be used with a Subquery; however, the BETWEEN operator can be used within the Subquery.

Syntax:

SELECT column_name [, column_name] FROM table1 [, table2] WHERE column_name OPERATOR (SELECT column_name [, column_name] FROM table1 [, table2] [WHERE])

Example: Consider the CUSTOMERS table having the following records:

Id	**Name**	**Dept**	**Age**	**Salary**	**Location**
100	Ramesh	Electrical	24	2500	Bangalore
101	Hrithik	Electronics	28	6500	Bangalore
102	Harsha	Aeronautics	28	5500	Mysore
103	Soumya	Electronics	22	2000	Bangalore
104	Priya	InfoTech	25	3000	Mangalore

- ✓ Now, let us check following Subquery with SELECT statement:

SELECT * FROM CUSTOMERS WHERE ID IN (SELECT ID FROM CUSTOMERS WHERE SALARY > 4500);

- ✓ This would produce the following result:

Id	**Name**	**Dept**	**Age**	**Salary**	**Location**
101	Hrithik	Electronics	28	6500	Bangalore
102	Harsha	Aeronautics	28	5500	Mysore

Correlated Subquery

- ❖ A query is called correlated Subquery when both the inner query and the outer query are interdependent.
- ❖ For every row processed by the inner query, the outer query is processed as well.
- ❖ The inner query depends on the outer query before it can be processed.

SELECT * FROM CUSTOMERS WHERE ID IN (SELECT ID, MAX (SALARY) FROM CUSTOMERS WHERE AGE > 25);

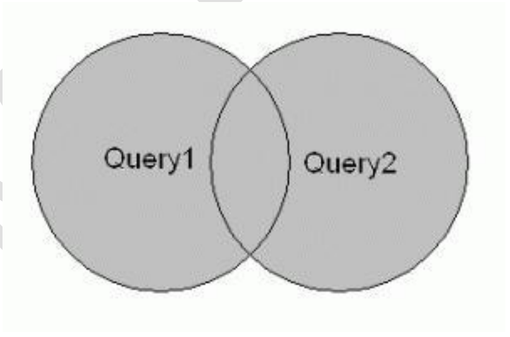
✓ This would produce the following result:

Id	Name	Dept	Age	Salary	Location
101	Hrithik	Electronics	28	6500	Bangalore

2.5 Implementation of Queries using SQL Set operators: Union, union all, Intersect, Minus

1) UNION Clause

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



- ❖ The output of both queries will be displayed as above.
  - o **Output :** Record only in query one + records only in query two + a single set of records which is common in both queries.

Syntax: SELECT * FROM table1 UNION SELECT * FROM table2;

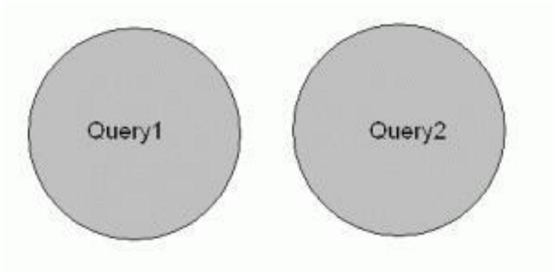
Example:

Select salesman_no 'ID', name from salesman_master Where city='bombay' **UNION**
Select client_no 'ID', name from client_master Where city='bombay';

Restriction on UNION clause:

- ✓ The number of column in all the queries should be same.
- ✓ The data type of columns in each query should be same
- ✓ Unions cannot be used in Sub queries
- ✓ Union cannot be used with aggregate functions

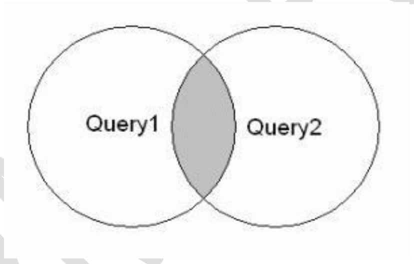
2) UNION ALL Clause



Syntax: SELECT * FROM table1 UNION ALL SELECT * FROM table2;

3) INTERSECT Clause

- ❖ The INTERSECT clause outputs only those rows produced by both queries intersected.
- ❖ The output of INTERSECT clause will include only those rows that are retrieved by both the queries



- ❖ The output of both queries will be displayed as above
- ❖ The final output of INTERSECT clause will be:
- ❖ A single set of Records which is common in both queries.

SELECT * FROM table1 INTERSECT SELECT * FROM table2;

Example:

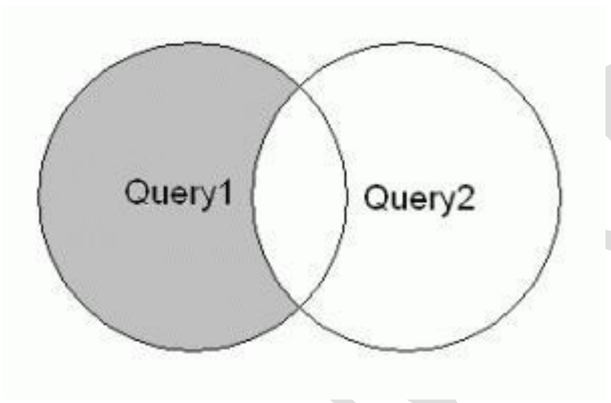
Select sman_no, name from salesman_master where city='bombay' **INTERSECT**
Select salesman_master. sman_no, name from salesman_master, sales_order where salesman_master. sman_no=sales_order. sman_no;

Output:

name	**Sman_no**
Kiran	S00001
Ravi	S00003

4) MINUS Clause

- ✓ The MINUS clause outputs the rows produced by the first query, after filtering the rows retrieved by the second query.



- ✓ The output of both queries will be displayed as above
- ✓ The final output of MINUS clause will be:
- ✓ Output: Records only in query one
- ✓ Example: Retrieve all the product number of non-moving items from the Product_Master table.

SELECT * FROM table1 MINUS SELECT * FROM table2;

Table Name: sales_order_details

Product_no	**Order_no**
P00001	O19001
P00002	O19002
P00003	O19003
P00004	O19004

Table Name: Product_Master

Description	**Product_no**
floppies	P00001
Monitors	P00002
Mouse	P00003
HDD	P00007
1.44 drive	P00008

Select product_no from Product_Master **MINUS**
Select product_no from sales_order_details;

Output:

Product_no
P00007
P00008