Oracle PL/SQL

Basics of PL/SQL

Agenda

Creating a Database

CREATE TABLE Statement

The CREATE TABLE statement allows you to create and define a table.

Syntax:

```
CREATE TABLE table_name
(
        column1  datatype [ NULL | NOT NULL ],
        column2  datatype [ NULL | NOT NULL ],
       ...
       column_n  datatype  [ NULL | NOT NULL ]
);
```

Altering a Database

ALTER TABLE Statement

The ALTER TABLE statement is used to add, modify, or drop/delete columns in a table. The Oracle

ALTER TABLE statement is also used to rename a table.

Add a Column in the Table

Syntax:

```
ALTER TABLE table_name
ADD column_name  column_definition;
```

Modify the Column in the Table

Syntax:

```
ALTER TABLE table_name
MODIFY  column_name  column_type;;
```

Add Multiple Columns in the Table

Syntax:

```
ALTER TABLE table_name
 ADD (column_1  column_definition,
          column_2  column_definition,
           ...
         column_n  column_definition);
```

Modify Multiple Columns in the Table

Syntax:

```
ALTER TABLE table_name
 MODIFY  (column_1  column_type,
           column_2  column_type,
           ...
           column_n  column_type);
```

Dropping a Column in the Table

The DROP COLUMN statement is used to drop a column in an existing table.

Drop a Column in the Table

Syntax:

```
ALTER TABLE table_name
 DROP COLUMN column_name;
```

Renaming a Column/Table

The RENAME COLUMN statement is used to rename a column in an existing table.

Rename a Column in the Table

Syntax:

```
ALTER TABLE table_name
RENAME COLUMN old_name TO new_name;
```

The RENAME TABLE statement is used to rename a table.

Rename a Table

Syntax:

```
ALTER TABLE table_name
RENAME TO new_table_name;
```

Querying a Database

Querying a Database: INSERT

The INSERT statement is used to insert a single record or multiple records into a table in Oracle.

Syntax:

```
INSERT INTO table
(column1, column2, ... column_n )
VALUES
(expression1, expression2, ... expression_n );
```

The syntax for the Oracle INSERT statement when inserting multiple records using a SELECT statement is:

Syntax:

```
INSERT INTO table
(column1, column2, ... column_n )
SELECT expression1, expression2, ... expression_n
FROM source_table
[WHERE conditions];
```

Querying a Database: SELECT

The SELECT statement is used to retrieve records from one or more tables in an Oracle Database.

Syntax:

```
SELECT expressions
FROM tables
[WHERE conditions];
```

- **expressions**

  Columns or calculations that you wish to retrieve. Use * if you wish to select all columns.
- **tables**

  Tables that you wish to retrieve the records from. There must be at least one table listed in the FROM clause.
- **WHERE conditions**

  Conditions that must be met for the records to be selected. If no conditions are provided, then all records will be selected (Optional).

Querying a Database: UPDATE

The UPDATE statement is used to update existing records in a table in an Oracle Database.

Syntax:

```
UPDATE table
SET column1  = expression1,
      column2  = expression2,
      ...
      column_n  = expression_n
[WHERE conditions];
```

Updating one table

Syntax:

```
UPDATE table1
SET column1  = (SELECT expression1
FROM table2
            WHERE conditions)
[WHERE conditions];
```

Updating one table with data from another table

GROUP BY Clause

It is used in a SELECT statement to collect data across multiple records and group the results by one or more columns.

Syntax:

```
SELECT expression1, expression2, ... expression_n,
  aggregate_function (aggregate_expression)
FROM tables
  [WHERE conditions]
GROUP BY expression1, expression2, ... expression_n;
```

ORDER BY Clause

It is used to sort records in your result set. The ORDER BY clause can only be used in SELECT statements.

Syntax:

```
SELECT expressions
  FROM tables
  [WHERE conditions]
ORDER BY expression [ ASC | DESC ];
```

HAVING Clause

It is used in combination with the GROUP BY clause to restrict the groups of returned rows to only those whose condition is TRUE.

Syntax:

```
SELECT expression1, expression2, ... expression_n,
 aggregate_function (aggregate_expression)
FROM tables
[WHERE conditions]
GROUP BY expression1, expression2, ... expression_n
HAVING having_condition;
```

BETWEEN Condition

It is used to retrieve values within a range in a SELECT, INSERT, UPDATE, or DELETE statement.

Syntax:

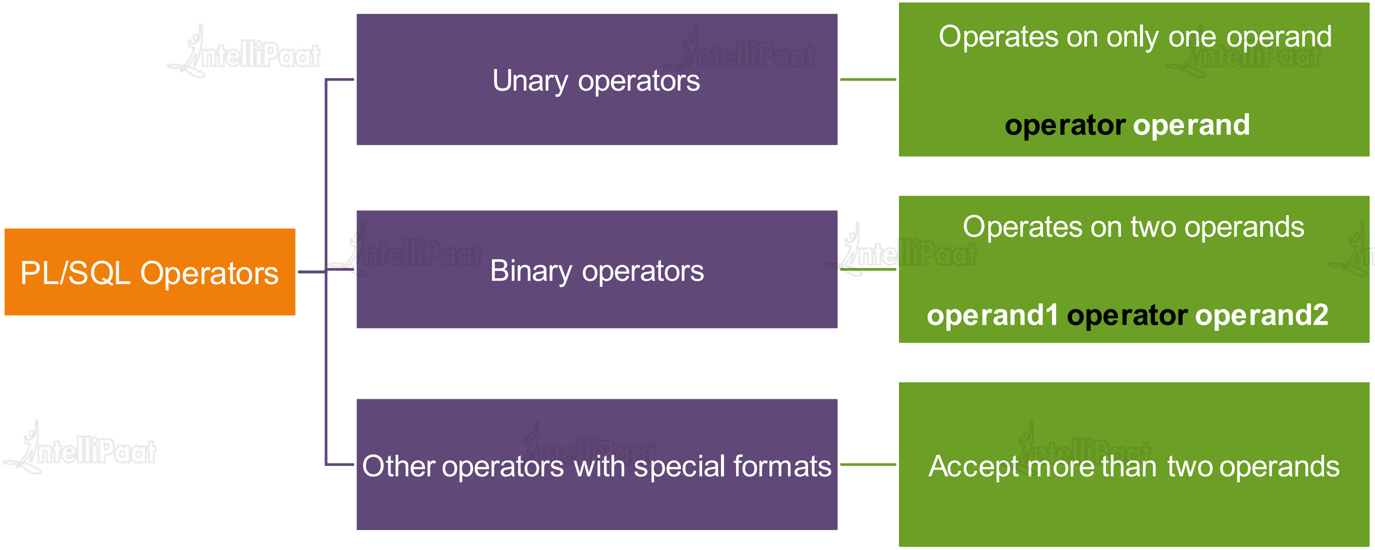
```
        expression BETWEEN value1  AND value2;
```

PL/SQL: Operators

PL/SQL: Operators

An operator is a symbol that manipulates individual data items called operands or arguments.

Operators are represented by special characters or by keywords.



PL/SQL Operators
Unary operators
Operates on only one operand
operator operand
Binary operators
Operates on two operands
operand1 operator operand2
Other operators with special formats
Accept more than two operands

Types of PL/SQL Operators

- **Arithmetic operators**
  - + , _ , * , / , MOD
- **Logical operators**
  - AND, OR, NOT
- **Relational operators**
  - >, <, >=, <=, !=,<>,^=, =
- **Concatenation operators**
  - | |
- **Assignment operators**
  - =
- **Miscellaneous operators**
  - Between, not between, like, not like, in not in, is null, not is null
- **Comparison operators**
  - Exists, not exists, any, all, some
- **Set operators**
  - Union, all Union, Intersect, minus

Arithmetic Operators

Operator	Purpose	Example
+ or -	It denotes a positive or negative expression. These are unary operators.	SELECT * FROM employee WHERE -salary < 0;
*** or /**	It multiplies and divides. These are binary operators.	UPDATE employee SET salary = salary * 1.1;
+ -	It adds and subtracts. These are binary operators.	SELECT salary + commission FROM employee WHERE SYSDATE – hiredate > 365;

Note: Do not use two consecutive minus signs with no separation (--) in arithmetic expressions to indicate double negation or the subtraction of a negative value

Operator Precedence

- Operations with higher precedence are applied first.
- Operators with the same precedence are applied in their text order:
    - You can change the execution order using parentheses.
    - If the expression includes parentheses, the execution starts with the innermost pair.

Operator	Description
**	Exponentiation
+, -	Identity and negation (unary operation)
*, /	Multiplication and division
+, -, ||	Addition, subtraction, and concatenation
=, <, >, < =, > =, <>, !=, ~= IS NULL, LIKE, BETWEEN, IN	Comparison
NOT	Logical negation
AND	Conjunction
OR	Inclusion

Logical Operators

A logical operator combines the results of two component conditions to produce a single result based on them or to invert the result of a single condition.

Operator	Function	Example

NOT	It returns TRUE if the following condition is FALSE, and vice versa. If it is UNKNOWN, it remains UNKNOWN.	SELECT * FROM emp WHERE NOT (job IS NULL);
		SELECT * FROM emp WHERE NOT (sal BETWEEN 1000 AND 2000);

AND	It returns TRUE if both component conditions are TRUE and returns FALSE if either is FALSE. Otherwise, it returns UNKNOWN.	SELECT * FROM emp WHERE job = 'CLERK' AND deptno = 10;
OR	It returns TRUE if either component condition is TRUE and returns FALSE if both are FALSE. Otherwise, it returns UNKNOWN.	SELECT * FROM emp WHERE job = 'CLERK' OR deptno = 10;

Comparison Operators with WHERE Clause

Operators that you can use in the WHERE clause to form more practical queries are:

- =, !=, or, <>, >, <= - Relational Operators

Operators that you can use in the WHERE clause to form more practical queries that are applied only on a single operand:

- BETWEEN, NOT BETWEEN
- IN, NOT IN
- Like, Not Like
- IS NULL, IS NOT NULL

Relational Operators

Operator	Description	Example
=	Checks if the values of two operands are equal or not. If yes, then the condition becomes true.	(A = B) is not true
!= <> ~=	Checks if the values of two operands are equal or not. If values are not equal, then the condition becomes true.	(A != B) is true
>	Checks if the value of the left operand is greater than the value of the right. If yes, then the condition becomes true.	(A > B) is not true
<	Checks if the value of the left operand is less than the value of the right. if yes, then the condition becomes true.	(A < B) is true
>=	Checks if the value of the left operand is greater than or equal to the value of the right. if yes, then the condition becomes true.	(A >= B) is not true
<=	Checks if the value of the left operand is less than or equal to the value of the right. If yes, then the condition becomes true.	(A <= B) is true

Defining a Null Value

- A Null is a value that is unavailable, unassigned, unknown, or inapplicable.
- A Null is not the same as zero or a blank space.

Find the name, job, salary, and commission of the employee

```
SELECT name, job, salary, comm From employees;
```

Null Values in Arithmetic Expressions

- Arithmetic expressions containing a null value evaluate to null

Find the total compensation given to each employee per annum

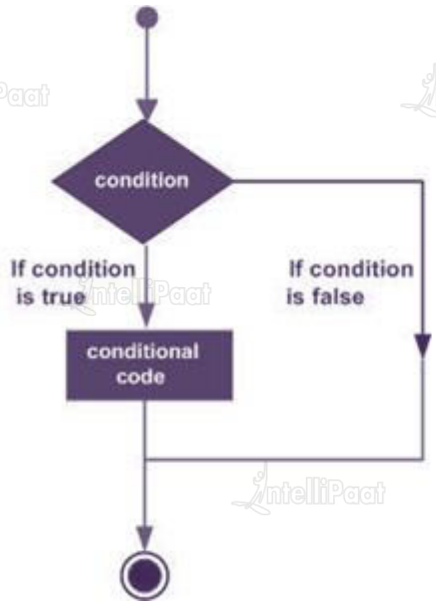
```
SELECT name, 12*(salary + commission) From employees;
```

PL/SQL: Conditions

Conditional Logic

- IF, THEN, ELSE, ELSIF, and END IF keywords are used in PL/SQL for performing conditional logic.

```
IF condition1 THEN
    statements1
ELSIF condition2 THEN
    statements2
ELSE
    statements3
END IF;
```



Conditional Logic: IF

Statement	Syntax	Description	Example

IF-THEN Statement	IF condition THEN
	Statement
	END IF;
		If the condition is true, then the statements get executed. And if the condition is false or NULL, then the IF statement does nothing.	IF (a <= 20) THEN
			c:= c+1;
			END IF;

IF-THEN-ELSE Statement	IF condition THEN
	Statement 1;
	ELSE
	Statement 2;
	END IF;
		If the condition is false or NULL, then only the alternative sequence of statements get executed. It ensures that either of the sequence of statements is executed.	IF color = red THEN
			dbms_output.put_line('You have chosen a red car')
			ELSE
			dbms_output.put_line('Please choose a color for your car');
			END IF;

Conditional Logic: IF

Statement	Syntax	Description	Example

IF-THEN-ELSIF Statement	
	```
	IF(boolean_expression
	1)THEN
	   Statement 1;
	ELSIF( boolean_expression  2)
	THEN
	   Statement 2;
	ELSIF( boolean_expression  3)
	THEN
	   Statement 3;
	ELSE
	   Statement 4;
	END IF;
	```
		It allows you to choose between several alternatives.	
			```
			DECLARE
			   a number(3)  := 100;
			BEGIN
			   IF ( a = 10 ) THEN
			      dbms_output.put_line('Value  of a is 10' );
			   ELSIF ( a = 20 ) THEN
			      dbms_output.put_line('Value  of a is 20' );
			   ELSIF ( a = 30 ) THEN
			      dbms_output.put_line('Value  of a is 30' );
			   ELSE
			       dbms_output.put_line('None  of the values  is
			matching');
			   END IF;
			   dbms_output.put_line('Exact  value  of a is: '|| a );
			END;
			```

Conditional Logic: IF

Statement	Syntax	Description	Example

Nested IF-THEN-ELSE Statement	IF(boolean_expression 1)THEN
	    IF(boolean_expression 2) THEN
	        sequence-of-statements;
	  END IF;
	ELSE
	    else-statements;
	END IF;
		You can use one IF-THEN or IF-THEN-ELSIF statement inside another IF-THEN or IF-THEN-ELSIF statement(s).	DECLARE
			  a number(3) := 100;
			  b number(3) := 200;
			BEGIN
			  -- check the boolean condition
			  IF(a = 100) THEN
			  -- if condition is true then check the following
			    IF(b = 200) THEN
			    -- if condition is true then print the following
			    dbms_output.put_line('Value of a is 100 and b is 200');
			    END IF;
			  END IF;
			  dbms_output.put_line('Exact value of a is : ' || a);
			  dbms_output.put_line('Exact value of b is : ' || b);
			END;

Conditional Logic: CASE

Statement	Syntax	Description	Example

CASE Statement	
	```
	CASE selector
	  WHEN 'value1'
	THEN  Statement 1;
	  WHEN 'value2'
	THEN  Statement2;
	  WHEN 'value3'
	THEN  Statement3;
	  ...
	  ELSE Statement n;
	-- default case
	END CASE;
	```
		Like the IF statement, the CASE statement selects one sequence of statements to execute.	
			```
			DECLARE
			  grade char(1)  := 'A';
			BEGIN
			  CASE grade
			    when 'A' then dbms_output.put_line('Excellent');
			    when 'B' then dbms_output.put_line('Very  good');
			    when 'C' then dbms_output.put_line('Well  done');
			    when 'D' then dbms_output.put_line('You  passed');
			    when 'F' then dbms_output.put_line('Better  try again');
			    else dbms_output.put_line('No  such grade');
			  END CASE;
			END;
			/
			```

Conditional Logic: CASE

Statement	Syntax	Description	Example

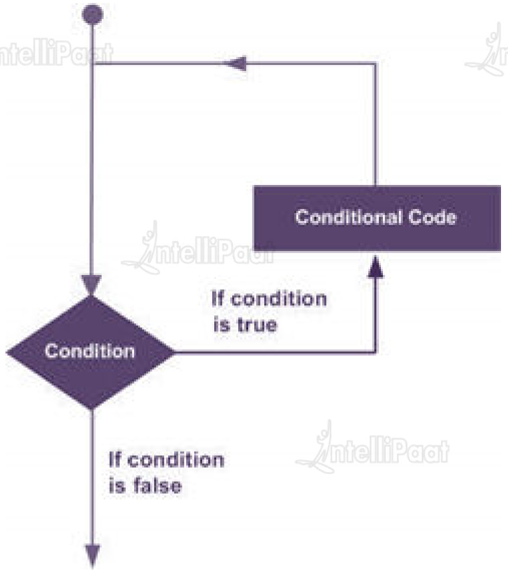
Searched CASE Statement	CASE
	  WHEN selector = 'value1' THEN S1;
	  WHEN selector = 'value2' THEN S2;
	  WHEN selector = 'value3' THEN S3;
	  ...
	  ELSE Sn; -- default case
	END CASE;
		The searched CASE statement has no selector, and its WHEN clauses contain search conditions that yield Boolean values.	DECLARE
			  grade char(1) := 'B';
			BEGIN
			  case
			    when grade = 'A' then dbms_output.put_line('Excellent');
			    when grade = 'B' then dbms_output.put_line('Very good');
			    when grade = 'C' then dbms_output.put_line('Well done');
			    when grade = 'D' then dbms_output.put_line('You passed');
			    when grade = 'F' then dbms_output.put_line('Better try again');
			    else dbms_output.put_line('No such grade');
			  end case;
			END;
			/

PL/SQL: Loops

LOOP Statement

- The LOOP statement is used when you are not sure of how many times you want the loop body to execute and whether you want the loop body to execute at least once.

```
LOOP
   {...statements...}
END LOOP;
```



LOOP Statement

1. Simple loops
   1. Run until you explicitly end the loop

1. WHILE loops
   1. Run until a specified condition occurs

1. FOR loops
   1. Run a predetermined number of times

1. Nested loops
   1. You can use one or more loops inside any other basic, while, or for loop.

Simple LOOP Statement

- With each iteration, the sequence of statements is executed. And then control resumes at the top of the loop

```
LOOP

  Sequence of statements;

END LOOP;
```

Syntax

```
DECLARE
  x number := 10;
BEGIN
  LOOP
    dbms_output.put_line(x);
    x := x + 10;
    IF x > 50 THEN
      exit;
    END IF;
  END LOOP;
  -- after exit, control resumes here
  dbms_output.put_line('After Exit x is: ' || x);
END;
/
```

Example

```
10
20
30
40
50
After Exit x is: 60
```

Output

WHILE LOOP Statement

- Repeatedly executes a target statement as long as a given condition is true

```
WHILE condition LOOP

  sequence_of_statements

END LOOP;
```

Syntax

```
DECLARE
  a number(2) := 10;
BEGIN
  WHILE a < 20 LOOP
    dbms_output.put_line('value of a: ' || a);
    a := a + 1;
  END LOOP;
END;
/
```

Example

```
value of a: 10
value of a: 11
value of a: 12
value of a: 13
value of a: 14
value of a: 15
value of a: 16
value of a: 17
value of a: 18
value of a: 19
```

Output

FOR LOOP Statement

- Allows you to efficiently write a loop that needs to execute a specific number of times

```
FOR counter IN initial_value .. final_value LOOP

  sequence_of_statements;

END LOOP;
```

Syntax

```
DECLARE
  a number(2);
BEGIN
  FOR a in 10 .. 20 LOOP
     dbms_output.put_line('value of a: ' || a);
 END LOOP;
END;
/
```

Example

```
value of a: 10
value of a: 11
value of a: 12
value of a: 13
value of a: 14
value of a: 15
value of a: 16
value of a: 17
value of a: 18
value of a: 19
value of a: 20
```

Output

Nested LOOP Statement

- Allows using one loop inside another loop

```
LOOP
  Sequence of statements1
  LOOP
    Sequence of statements2
  END LOOP;
END LOOP;
```

Nested Simple Loop

```
FOR counter1 IN initial_value1 .. final_value1 LOOP
  sequence_of_statements1
  FOR counter2 IN initial_value2 .. final_value2 LOOP
    sequence_of_statements2
  END LOOP;
END LOOP;
```

Nested FOR Loop

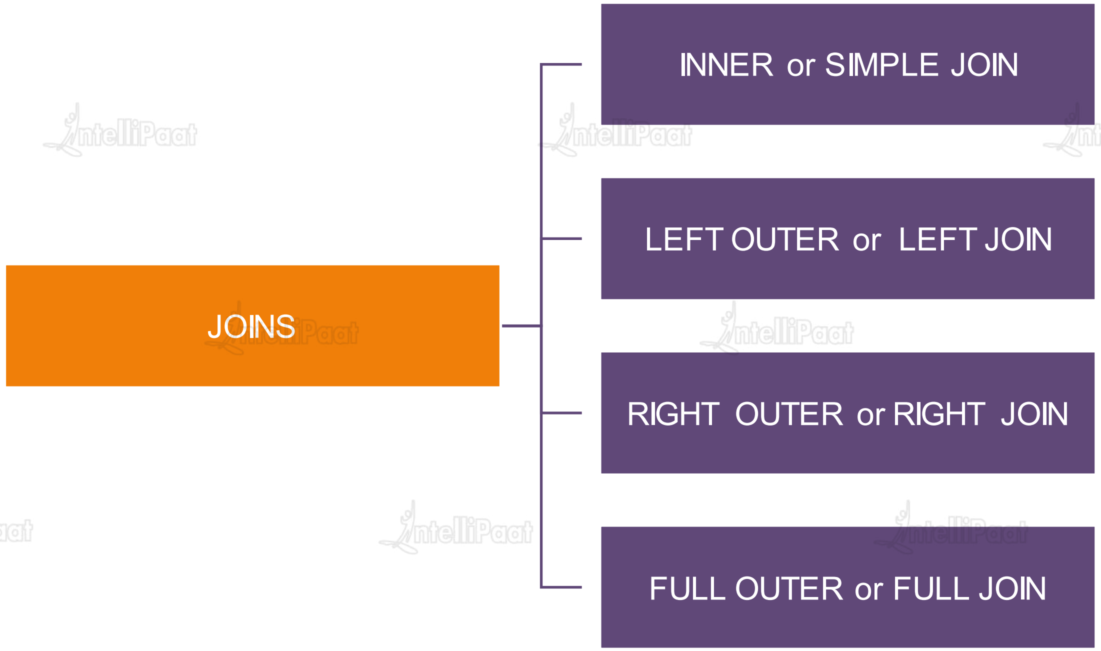
```
WHILE condition1 LOOP
  sequence_of_statements1
  WHILE condition2 LOOP
    sequence_of_statements2
  END LOOP;
END LOOP;
```

Nested WHILE Loop

PL/SQL: Joins

Joins

- Used to retrieve data from multiple tables

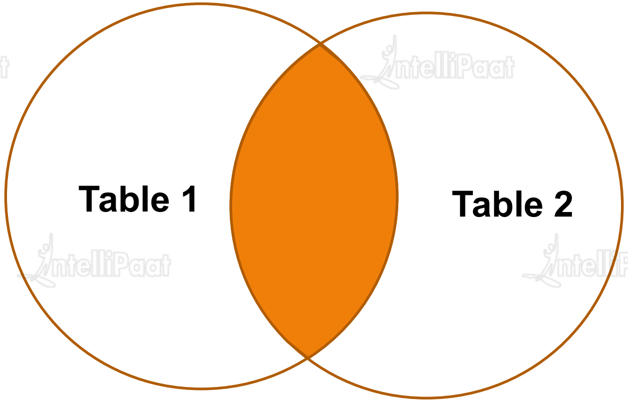


INNER JOIN (Simple Join)

- Returns all rows from multiple tables where the join condition is met

Syntax

```
SELECT columns
FROM table1
INNER JOIN table2
ON table1.column = table2.column;
```



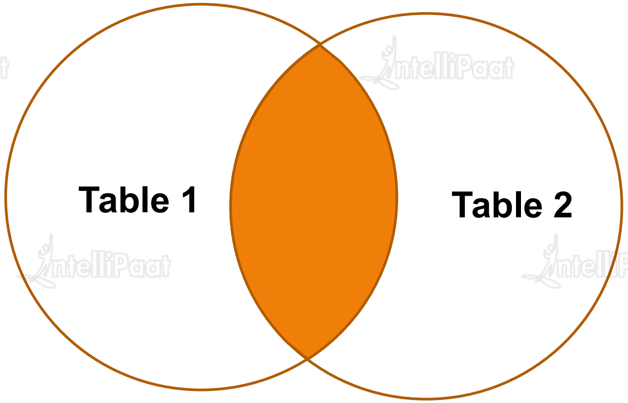
Different Ways of Joining Using Inner Join

Syntax

```
Select columns,..
from table1, table2, ...
where (Join Condition);
```

Example

```
SQL> select e.name, e.job, d.name
2 from employee e, dept d
3 where e.deptno = d.deptno;
```

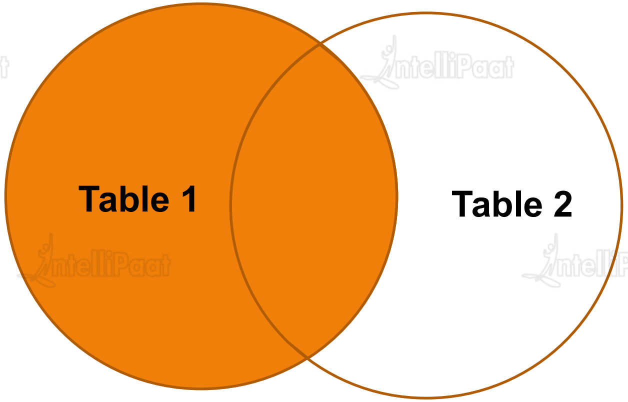


LEFT OUTER JOIN

- Returns all rows from the left-hand side table specified in the ON condition and only those rows from the other table where the joined fields are equal

Syntax

```
SELECT columns
FROM table1
LEFT [OUTER] JOIN table2
ON table1.column = table2.column;
```

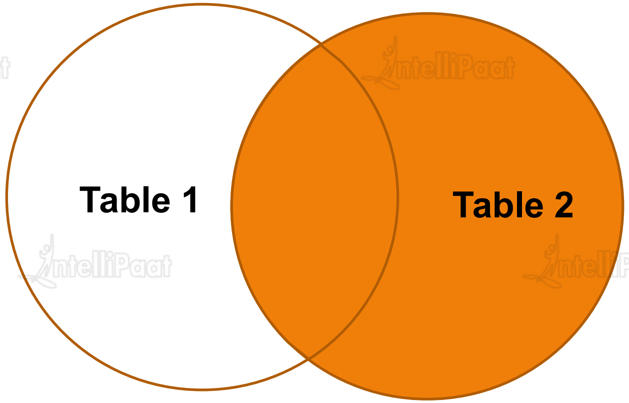


RIGHT OUTER JOIN

- Returns all rows from the right-hand side table specified in the ON condition and only those rows from the other table where the joined fields are equal

Syntax

```
SELECT columns
FROM table1
RIGHT [OUTER] JOIN table2
ON table1.column = table2.column;
```

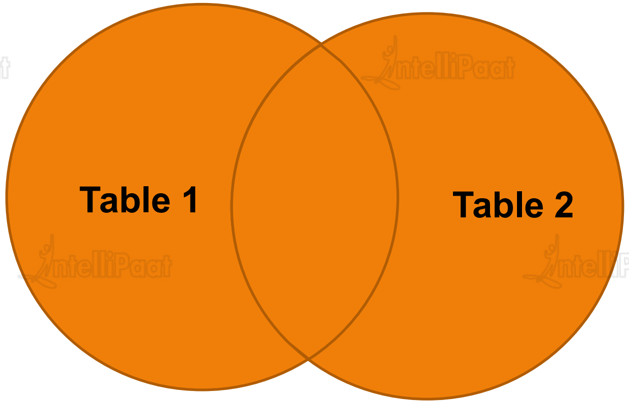


FULL OUTER JOIN

- Returns all rows from the left-hand table and the right-hand table with nulls in place where the join condition is not met.

Syntax

```
SELECT columns
FROM table1
FULL [OUTER] JOIN table2
ON table1.column = table2.column;
```



SELF JOIN

- Joining a table to itself is called a self join.
- It is useful when you want to join one row in a table to another in the same table.

List employee names along with their manager's name

```
SQL> select w.name as "Employee", m.name as "Manager“
 2 from employee w join employee m
3 on w.mgr = m.empno;
```

List employees whose salaries exceed their manager's salary

```
SQL> select w. name as "Employee , m. name as "Manager"
2 from employee w join employee m
3 on w.mgr = m.empno and w.salary > m.salary;
```

NON-EQUI JOIN

- In this join, we need to use only relational operators such as >,<,>=,<=,!= except 1=1 operator.
- The main advantage of non-equi join is that, even though there is no common column, we can perform the join operation.

```
SQL> Select e.empno, e.name e.salary s. grade from employee e, salarygrade s
2 where e.salary BETWEEN s.losai AND hisal;
```

Natural Join

- **This join was introduced from the Oracle 9i version onwards.**
- **Both equi join and natural join are exactly the same as far as the output is concerned.**
- **There are three differences between equi join and natural join.**

1)No need to check the WHERE condition

No need to mention the table name before the common column

Common column will be displayed in front of the output

```
Select * from dept natural join dept
```

Quiz

Quiz 1

The ORDER BY clause can only be used in ?

A SELECT queries

B INSERT queries

C GROUP BY queries

D HAVING queries

Quiz 1

The ORDER BY clause can only be used in ?

A	SELECT queries
B	INSERT queries
C	GROUP BY queries
D	HAVING queries

Quiz 2

Select the invalid PL/SQL looping construct ?

A	WHILE LOOP … END LOOP;
B	FOR rec IN some_cursor LOOP … END LOOP;
C	LOOP … UNTIL ; END LOOP;
D	LOOP … EXIT WHEN ; END LOOP;

Answer 2

Select the invalid PL/SQL looping construct ?

A	WHILE LOOP ... END LOOP;
B	FOR rec IN some_cursor LOOP ... END LOOP;
C	LOOP ... UNTIL ; END LOOP;
D	LOOP ... EXIT WHEN ; END LOOP;

Quiz 3

Which of the following way or ways before is/are correct to insert DATE in a table?

A	insert into Employee(Start_Date) values ('05-FEB-2020')
B	insert into Employee(Start_Date) values ('FEB-05-2020')
C	Both a and b
D	None of the above

Answer 3

Which of the following way or ways before is/are correct to insert DATE in a table?

[✓ **A** insert into Employee(Start_Date) values ('05-FEB-2020')]

B insert into Employee(Start_Date) values ('FEB-05-2020')

C Both a and b

D None of the above

Quiz 4

The following SQL is which type of join: SELECT CUSTOMER_T. CUSTOMER_ID, ORDER_T. CUSTOMER_ID, NAME, ORDER_ID FROM CUSTOMER_T,ORDER_T WHERE CUSTOMER_T. CUSTOMER_ID = ORDER_T. CUSTOMER_ID

A Equi-join

B Natural join

C Outer join

D Error in code

Answer 4

The following SQL is which type of join: SELECT CUSTOMER_T. CUSTOMER_ID, ORDER_T. CUSTOMER_ID, NAME, ORDER_ID FROM CUSTOMER_T,ORDER_T WHERE CUSTOMER_T. CUSTOMER_ID = ORDER_T. CUSTOMER_ID

[✓ **A** Equi-join]

B Natural join

C Outer join

D Error in code

Quiz 5

Which set operator can be used to join two sub-queries ?

A	Intersect
B	Union All
C	Union
D	Minus

Answer 5

Which set operator can be used to join two sub-queries ?

[✓ A Intersect]

B Union All

C Union

D Minus

India: +91-7847955955

US: 1-800-216-8930 (TOLL FREE)

sales@intellipaat.com

24/7 Chat with Our Course Advisor