

# Oracle SQL commands (Create , Alter, Drop , Temporary Tables, Views)

Thursday, March 17, 2022 8:08 PM

- Oracle database is a relational database management system (RDBMS) from Oracle Corporation.
- A database refers to the organized collection of structured data stored electronically in a device.
- Creating a Table

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

```
CREATE TABLE customers
( customer_id number(10) NOT NULL,
  customer_name varchar2(50) NOT NULL,
  city varchar2(50)
);
```

```
CREATE TABLE customers
( customer_id number(10) NOT NULL,
  customer_name varchar2(50) NOT NULL,
  city varchar2(50),
CONSTRAINT customers_pk PRIMARY KEY (customer_id
);
```

- Create table as is used to copying the existing table into a new table

```
CREATE TABLE newcustomers
AS (SELECT * FROM customers WHERE customer_id < 5000);
```

Table created.

## Create Table Example: copying selected columns from multiple tables

### Syntax:

```
CREATE TABLE new_table
AS (SELECT column_1, column2, ... column_n
FROM old_table_1, old_table_2, ... old_table_n);
```

```
CREATE TABLE newcustomers3
```

```
AS (SELECT regularcustomers.rcustomer_id, regularcustomers.rc_city, irregularcustomers.ircustomer_id  
FROM regularcustomers, irregularcustomers  
WHERE regularcustomers.rcustomer_id = irregularcustomers.ircustomer_id  
AND regularcustomers.rcustomer_id < 5000);
```

- Adding a new column into existing table

```
ALTER TABLE customers
```

```
ADD customer_age varchar2(50);
```

- We can add two or more columns at a time into a table

```
ALTER TABLE customers
```

```
ADD (customer_type varchar2(50),  
customer_address varchar2(50));
```

customer\_type and customer\_address will be added in the table customers.

- We can modify multiple columns of a table at a time.

```
ALTER TABLE table_name
```

```
MODIFY (column_1 column_type,  
column_2 column_type,  
...  
column_n column_type);
```

**Example:**

```
ALTER TABLE customers
```

```
MODIFY (customer_name varchar2(100) not null,  
city varchar2(100));
```

- We can drop the columns of the table.

```
ALTER TABLE table_name
```

```
DROP COLUMN column_name;
```

**Example:**

```
ALTER TABLE customers
```

```
DROP COLUMN customer_name;
```

- How to rename column name

```
ALTER TABLE customers
```

```
RENAME COLUMN customer_name to cname;
```

- How to rename the name of the table.

```
ALTER TABLE customers
```

```
RENAME TO retailers;
```

- If there are referential integrity constraints on table\_name and you do not specify the CASCADE CONSTRAINTS option, the DROP TABLE statement will return an error and Oracle will not drop the table

- Dropping the table.

## DROP TABLE Example

```
DROP TABLE customers;
```

This will drop the table named customers.

## DROP TABLE Example with PURGE parameter

```
DROP TABLE customers PURGE
```

This statement will drop the table called customers and issue a PURGE so that the space associated with the customers table is released and the customers table is not placed in recycle bin. So, it is not possible to recover that table if required.

## Oracle Global Temporary tables

Temporary tables generally contain all of the features that ordinary tables have like triggers, join cardinality, information about rows and block etc. the main difference is that the temporary tables can't have foreign keys related to other tables.

```
CREATE GLOBAL TEMPORARY TABLE students  
( student_id numeric(10) NOT NULL,  
  student_name varchar2(50) NOT NULL,  
  student_address varchar2(50)  
);
```

This will create a global temporary table called students

## Oracle Local Temporary tables

In Oracle, local temporary tables are distinct within modules. These tables are defined and scoped to the session in which you created it.

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

## Oracle View

In Oracle, view is a virtual table that does not physically exist. It is stored in Oracle data dictionary and do not store any data. It can be executed when called.

A view is created by a query joining one or more tables.

## What is the difference between view and table in SQL Server?



The main difference between them is that a table is an object that consists of rows and columns to store and retrieve data whenever the user needs it. In contrast, the view is a virtual table based on an SQL statement's result set and will disappear when the current session is closed.

### Create View Query:

```
CREATE VIEW sup_orders AS
SELECT suppliers.supplier_id, orders.quantity, orders.price
FROM suppliers
INNER JOIN orders
ON suppliers.supplier_id = supplier_id
WHERE suppliers.supplier_name = 'VOJO';
```

### Output:

```
View created.
0.21 seconds
```

You can now check the Oracle VIEW by this query:

```
SELECT * FROM sup_orders;
```

### Output:

```
SUPPLIER_ID    QUANTITY    PRICE
3              35          70
3              26          125
3              18          100
3 rows returned in 0.00 seconds
```

## Oracle Update VIEW

```
CREATE or REPLACE VIEW sup_orders AS
SELECT suppliers.supplier_id, orders.quantity, orders.price
FROM suppliers
INNER JOIN orders
ON suppliers.supplier_id = supplier_id
WHERE suppliers.supplier_name = 'HCL';
```

```
SELECT * FROM sup_orders;
```

Output:

SUPPLIER_ID	QUANTITY	PRICE
1	35	70
1	26	125
1	18	100

row(s) 1 - 3 of 3

## Oracle DROP VIEW

```
DROP VIEW sup_orders;
```

# Oracle Select Statement

Friday, March 18, 2022 2:02 PM

## Oracle SELECT Statement

### Select Example: select all fields

Let's take an example to select all fields from an already created table named customers

```
SELECT *  
FROM customers;
```

#### output

NAME	AGE	ADDRESS	SALARY
mohan	21	ghaziabad	20000
rohan	22	delhi	22000
sohan	25	noida	24000
alex	28	Paris	40000

4 rows returned in 0.02  
seconds

### Select Example: select specific fields

#### Example

```
SELECT age, address, salary  
FROM customers  
WHERE age < 25  
AND salary > '20000'  
ORDER BY age ASC, salary DESC;
```

AGE	ADDRESS	SALARY
22	delhi	22000

1 rows returned in 0.00  
seconds

## Select Example: select fields from multiple tables (JOIN)

```
SELECT customers.name, courses.trainer
FROM courses
INNER JOIN customers
ON courses.course_id = course_id
ORDER BY name;
```

### output

NAME	TRAINER
alex	sonoo jaiswal
alex	dd sharma
alex	swati uniyal
alex	rashmi desai
alex	mahesh sharma
mohan	mahesh sharma
mohan	rashmi desai
mohan	swati uniyal
mohan	sonoo jaiswal
mohan	dd sharma

More than 10 rows available. Increase rows selector to view more rows.

10 rows returned in 0.02  
seconds



# Oracle Insert/Insert All Statement

Friday, March 18, 2022 2:14 PM

**Syntax: (Inserting a single record using the Values keyword):**

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

**Syntax: (Inserting multiple records using a SELECT statement):**

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

## Oracle Insert Example: By VALUE keyword

----- Example -----

```
INSERT INTO suppliers  
(supplier_id, supplier_name)  
VALUES  
(50, 'Flipkart');
```

**Output:**

```
1 row(s) inserted.  
0.02 seconds
```

## Oracle Insert Example: By SELECT statement

**Execute this query:**

```
INSERT INTO suppliers  
(supplier_id, supplier_name)  
SELECT age, address  
FROM customers  
WHERE age > 20;
```

**Output:**

```
4 row(s) inserted.  
0.00 seconds
```



## Oracle INSERT ALL Example

This example specifies how to insert multiple records in one table. Here we insert three rows into the "suppliers" table.

```
INSERT ALL
  INTO suppliers (supplier_id, supplier_name) VALUES (20, 'Google')
  INTO suppliers (supplier_id, supplier_name) VALUES (21, 'Microsoft')
  INTO suppliers (supplier_id, supplier_name) VALUES (22, 'Apple')
SELECT * FROM dual;
```

### Output

```
3 row(s) inserted.
0.02 seconds
```

This is totally equivalent to the following three INSERT statements.

```
INSERT INTO suppliers (supplier_id, supplier_name) VALUES (1000, 'Google');
INSERT INTO suppliers (supplier_id, supplier_name) VALUES (2000, 'Microsoft');
INSERT INTO suppliers (supplier_id, supplier_name) VALUES (3000, 'Apple');
```

## Oracle INSERT ALL Example: (Insert into multiple tables)

The INSERT ALL statement can also be used to insert multiple rows into more than one table by one command only.

In the following example, we are going to insert records into the both "suppliers" and "customers" tables.

```
INSERT ALL
  INTO suppliers (supplier_id, supplier_name) VALUES (30, 'Google')
  INTO suppliers (supplier_id, supplier_name) VALUES (31, 'Microsoft')
  INTO customers (age, name, address) VALUES (29, 'Luca Warsi', 'New York')
SELECT * FROM dual;
```

### Output

```
3 row(s) inserted.
0.03 seconds
```

Here, total 3 rows are inserted, 2 rows are inserted into the suppliers table and one row into the customers table.

# Oracle Update Statement

Friday, March 18, 2022 2:22 PM

## Oracle Update Example: (Update single column)

```
UPDATE suppliers  
SET supplier_name = 'Kingfisher'  
WHERE supplier_id = 2;
```

This example will update the supplier\_name as "Kingfisher" where "supplier\_id" is 2.

## Oracle Update Example: (Update multiple columns)

The following example specifies how to update multiple columns in a table. In this example, two columns supplier\_name and supplier\_address is updated by a single statement.

```
UPDATE suppliers  
SET supplier_address = 'Agra',  
    supplier_name = 'Bata shoes'  
WHERE supplier_id = 1;
```

Output:

```
1 row(s) updated.  
0.06 seconds
```

## Oracle Update Example: (By selecting records from another table)

```
UPDATE customers  
SET name = (SELECT supplier_name  
            FROM suppliers  
            WHERE suppliers.supplier_name = customers.name)  
WHERE age < 25;
```

# Oracle Delete and Truncate Statement

Friday, March 18, 2022 2:25 PM

## Oracle Delete Example: On one condition

```
DELETE FROM customers  
WHERE name = 'Sohan';
```

This statement will delete all records from the customer table where name is "Sohan".

## Oracle Delete Example: On multiple conditions

```
DELETE FROM customers  
WHERE last_name = 'Maurya'  
AND customer_id > 2;
```

This statement will delete all records from the customers table where the last\_name is "Maurya" and the customer\_id is greater than 2.

## Oracle TRUNCATE TABLE

In Oracle, TRUNCATE TABLE statement is used to remove all records from a table. It works same as DELETE statement but without specifying a WHERE clause. It is generally used when you don't have to worry about rolling back

Once a table is truncated, it can't be rolled back. The TRUNCATE TABLE statement does not affect any of the table's indexes, triggers or dependencies.

```
TRUNCATE TABLE customers;
```

## TRUNCATE TABLE vs DELETE TABLE

Both the statements will remove the data from the "customers" table but the main difference is that you can roll back the DELETE statement whereas you can't roll back the TRUNCATE TABLE statement.

# Oracle Clauses

Friday, March 18, 2022 3:02 PM

## Oracle DISTINCT Clause

Oracle DISTINCT clause is used to remove the duplicate records from the result set. It is only used with SELECT statement.

### Oracle DISTINCT Example: (with single expression)

```
SELECT DISTINCT state
FROM customers
WHERE name = 'charu';
```

### Oracle DISTINCT Example: (with multiple expressions)

Execute this query:

```
SELECT DISTINCT name, age, salary
FROM customers
WHERE age >= '60';
```

## Oracle ORDER BY Clause

In Oracle, ORDER BY Clause is used to sort or re-arrange the records in the result set. The ORDER BY clause is only used with SELECT statement.

### Oracle ORDER BY Example: (sorting in descending order)

If you want to sort your result in descending order, you should use the DESC attribute in your ORDER BY clause:

Execute this Query:

```
SELECT *
FROM supplier
ORDER BY last_name DESC;
```

Output

SUPPLIER_ID	FIRST_NAME	LAST_NAME
2	kanchabhai	Motwani
3	lallu	lal
1	Dhirubhai	Ambani
row(s) 1 - 3 of 3		

### Oracle GROUP BY Example: (with SUM function)

Execute this query:

```
SELECT item, SUM(sale) AS "Total sales"
FROM salesdepartment
GROUP BY item;
```

Output

ITEM	Total Sales
Belts	105
Sari	5210
Shoes	1165
Medicines	250
Computer	210

The above example will show the total sales of every individual item.

```
SELECT state, COUNT(*) AS "Number of customers"
FROM customers
WHERE salary > 10000
GROUP BY state;
```

```
SELECT department,
MIN(salary) AS "Lowest salary"
FROM employees
GROUP BY department;
```

Output:

DEPARTMENT	Lowest Salary
Mechanical	12000
Software	10000
hardware	15000

```
SELECT department,
MAX(salary) AS "Highest salary"
FROM employees
GROUP BY department;
```

Output:

DEPARTMENT	Highest Salary
Mechanical	12000
Software	32000
hardware	15000

## Oracle HAVING Clause

In Oracle, HAVING Clause is used with GROUP BY Clause to restrict the groups of returned rows where 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;
```

**having\_conditions:** It specifies the conditions that are applied only to the aggregated results to restrict the groups of returned rows.

- We use having class to check the conditions after grouping has done on the table and if the having clause fails then the particular field will not be displayed.

```
SELECT item, SUM(sale) AS "Total sales"  
FROM salesdepartment  
GROUP BY item  
HAVING SUM(sale) < 1000;
```

```
SELECT state, COUNT(*) AS "Number of customers"  
FROM customers  
WHERE salary > 10000  
GROUP BY state  
HAVING COUNT(*) >= 2;
```

```
SELECT department,  
       MIN(salary) AS "Lowest salary"  
FROM employees  
GROUP BY department  
HAVING MIN(salary) < 15000;
```

```
SELECT department,  
       MAX(salary) AS "Highest salary"  
FROM employees  
GROUP BY department  
HAVING MAX(salary) > 30000;
```

# Oracle Operators

Friday, March 18, 2022 3:23 PM

## Oracle UNION Operator

In Oracle, UNION operator is used to combine the result sets of two or more Oracle SELECT statements. It combines the both SELECT statement and removes duplicate rows between them./p>

Each SELECT statement within the UNION operator must have the same number of fields in the result sets with similar data types.

### Oracle UNION Example: (Fetch single field)

```
SELECT supplier_id
FROM suppliers
UNION
SELECT supplier_id
FROM order_details
```

Note: If you don't want to remove duplicates, use Oracle UNION ALL operator.

### Oracle UNION Example: (Using ORDER BY)

The Oracle UNION operator can be used with ORDER BY clause to orders the results of the query.

```
SELECT supplier_id, supplier_name
FROM suppliers
WHERE supplier_id <= 20
UNION
SELECT s_id, s_name
FROM shopkeepers
WHERE s_name = 'dhirubhai'
ORDER BY 1;
```

#### Output

SUPPLIER_ID	SUPPLIER_NAME
1	Bata shoes
1	dhirubhai
2	Kingfisher
3	VOJO
20	Google

- Union all does not remove duplicate rows after union of the databases.



## Oracle UNION ALL Operator Example

```
SELECT supplier_id
FROM suppliers
UNION ALL
SELECT supplier_id
FROM order_details;
```

The above example will return the supplier\_id multiple times in the result set if the same value appeared in both the supplier\_id and order\_details table.

### Output

SUPPLIER_ID	
1	
2	
3	
50	
50	
21	
22	
25	
28	
20	
More than 10 rows available. Increase rows selector to view more rows.	
10 rows returned in 0.00 seconds	

## Oracle INTERSECT Operator






In Oracle, INTERSECT Operator is used to return the results of 2 or more SELECT statement. It picks the common or intersecting records from compound SELECT queries.

### Oracle INTERSECT Example: (with single expression)

#### Suppliers Table

Column Name	Data Type	Nullable	Default	Primary Key
SUPPLIER_ID	NUMBER	Yes	-	-
SUPPLIER_NAME	VARCHAR2(4000)	Yes	-	-
SUPPLIER_ADDRESS	VARCHAR2(4000)	Yes	-	-
				1 - 3




#### Suppliers Data

EDIT	SUPPLIER_ID	SUPPLIER_NAME	SUPPLIER_ADDRESS
	1	Bata shoes	Agra
	2	Kingfisher	Delhi
	3	VOJO	Lucknow
	4	Apple	-
	5	Flipkart	-

### Order\_details Table

Column Name	Data Type	Nullable	Default	Primary Key
SUPPLIER_ID	NUMBER	Yes	-	-
SUPPLY_DATE	DATE	Yes	-	-
SUPPLY_ADDRESS	VARCHAR2(4000)	Yes	-	-

### Order\_details Data

EDIT	SUPPLIER_ID	SUPPLY_DATE	SUPPLY_ADDRESS
	1	12-OCT-12	Ahmedabad
	2	23-OCT-12	Mumbai
	3	31-OCT-12	delhi

```
SELECT supplier_id
FROM suppliers
INTERSECT
SELECT supplier_id
FROM order_details;
```

In the above example, the supplier\_id appears in both the suppliers and order\_details table. Now the common entries will be returned in the result set.

### Output

SUPPLIER_ID
1
2
3

3 rows returned in 0.02 seconds

```
SELECT supplier_id, last_name, first_name
FROM supplier
WHERE first_name <> 'dhirubhai'
INTERSECT
SELECT customer_id, last_name, first_name
FROM customer
WHERE customer_id < 5;
```

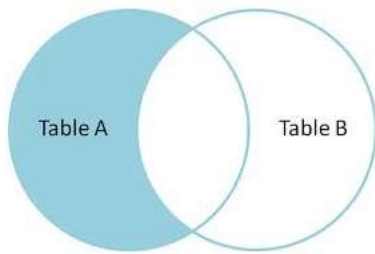
### Output

LAST_NAME	FIRST_NAME
maurya	ajeet

1 rows returned in 0.00 seconds

## Oracle MINUS operator

In Oracle, MINUS operator is used to return all rows in the first SELECT statement that are not returned by the second SELECT statement.



```
SELECT supplier_id
FROM suppliers
MINUS
SELECT supplier_id
FROM order_details;
```

#### Output

SUPPLIER_ID
20
21
22
25
28
30
31
50

8 rows returned in 0.11  
seconds

# Oracle Joins

Friday, March 18, 2022 6:04 PM

## Oracle INNER JOIN

Inner Join is the simplest and most common type of join. It is also known as simple join. It returns all rows from multiple tables where the join condition is met.

```
SELECT suppliers.supplier_id, suppliers.supplier_name, order1.order_number
FROM suppliers
INNER JOIN order1
ON suppliers.supplier_id = order1.supplier_id;
```

### Output

SUPPLIER_ID	SUPPLIER_NAME	ORDER_NUMBER
1	Bata shoes	101
2	Kingfisher	102

2 rows returned in 0.03 seconds

## Oracle OUTER JOIN

An outer join is similar to equijoin but it gets also the non-matched rows from the table. It is categorized in Left Outer Join, Right Outer Join and Full Outer Join by Oracle 9i ANSI/ISO 1999 standard.

- Left outer Join:
  - It returns all the matched and un matched rows of Table A and matched rows of table B

```
SELECT suppliers.supplier_id, suppliers.supplier_name, order1.order_number
FROM suppliers
LEFT OUTER JOIN order1
ON suppliers.supplier_id = order1.supplier_id;
```

- Right outer join
  - It returns all the matched rows of Table A and matched and un matched rows of table B.

```
SELECT order1.order_number, order1.city, suppliers.supplier_name
FROM suppliers
RIGHT OUTER JOIN order1
ON suppliers.supplier_id = order1.supplier_id;
```

## Full Outer Join

The Full Outer Join returns all rows from the left hand table and right hand table. It places NULL where the join condition is not met.

### Syntax

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

```

SELECT suppliers.supplier_id, suppliers.supplier_name, order1.order_number
FROM suppliers
FULL OUTER JOIN order1
ON suppliers.supplier_id = order1.supplier_id;

```

- EQUI JOIN

```

SELECT agents.agent_city, customer.last_name,
customer.first_name
FROM agents, customer
WHERE agents.agent_id = customer.customer_id;

```

## Oracle SELF JOIN

Self Join is a specific type of Join. In Self Join, a table is joined with itself (Unary relationship). A self join simply specifies that a row is combined with itself and every other row of the table.

```

SELECT a.name, b.age, a.SALARY
FROM CUSTOMERS a, CUSTOMERS b
WHERE a.SALARY < b.SALARY;

```

## Oracle Cross Join (Cartesian Products)

The CROSS JOIN specifies that all rows from first table join 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 will have x\*y rows. It normally happens when no matching join columns are specified.

In simple words you can say that if two tables in a join query have no join condition, then the Oracle returns their Cartesian product.

```

SELECT *
FROM table1
CROSS JOIN table2;

```

```

SELECT * FROM table1, table2

```

### Oracle Anti Join

- Anti-join between two tables returns rows from the first table where no matches are found in the second table. It is opposite of a semi-join. An anti-join returns one copy of each row in the first table for which no match is found.
- Anti-joins are written using the NOT EXISTS or NOT IN constructs.

#### Departments table

```

CREATE TABLE "DEPARTMENTS"
(
  "DEPARTMENT_ID" NUMBER(10,0) NOT NULL ENABLE,
  "DEPARTMENT_NAME" VARCHAR2(50) NOT NULL ENABLE,
  CONSTRAINT "DEPARTMENTS_PK" PRIMARY KEY ("DEPARTMENT_ID") ENABLE
)
/

```





```

EDIT DEPARTMENT_ID DEPARTMENT_NAME

```






## Departments table

```
CREATE TABLE "DEPARTMENTS"  
( "DEPARTMENT_ID" NUMBER(10,0) NOT NULL ENABLE,  
  "DEPARTMENT_NAME" VARCHAR2(50) NOT NULL ENABLE,  
  CONSTRAINT "DEPARTMENTS_PK" PRIMARY KEY ("DEPARTMENT_ID") ENABLE  
)  
/
```

EDIT	DEPARTMENT_ID	DEPARTMENT_NAME
	1	a
	2	b
	3	c
	4	d
row(s) 1 - 4 of 4		






## Customer table

```
CREATE TABLE "CUSTOMER"  
( "CUSTOMER_ID" NUMBER,  
  "FIRST_NAME" VARCHAR2(4000),  
  "LAST_NAME" VARCHAR2(4000),  
  "DEPARTMENT_ID" NUMBER  
)  
/
```

EDIT	CUSTOMER_ID	FIRST_NAME	LAST_NAME	DEPARTMENT_ID
	1	aryan	tomar	-
	2	ajeet	maurya	-
	3	richa	goyal	-
	4	dhirubhai	jotwani	-
	32	alex	pandian	1
row(s) 1 - 5 of 5				

## Customer table

```
CREATE TABLE "CUSTOMER"  
( "CUSTOMER_ID" NUMBER,  
  "FIRST_NAME" VARCHAR2(4000),  
  "LAST_NAME" VARCHAR2(4000),  
  "DEPARTMENT_ID" NUMBER  
)  
/
```

EDIT	CUSTOMER_ID	FIRST_NAME	LAST_NAME	DEPARTMENT_ID
	1	aryan	tomar	-
	2	ajeet	maurya	-
	3	richa	goyal	-
	4	dhirubhai	jotwani	-
	32	alex	pandian	1
row(s) 1 - 5 of 5				

```
SELECT departments.department_id, departments.department_name  
FROM departments  
WHERE NOT EXISTS  
(  
  SELECT 1  
  FROM customer  
  WHERE customer.department_id = departments.department_id  
)  
ORDER BY departments.department_id;
```

## Output

DEPARTMENT_ID	DEPARTMENT_NAME
2	b
3	c
4	d

3 rows returned in 0.02  
seconds

- Inner join can have equality (=) and other operators (like <,>,<>) in the join condition.
- Equi join only have equality (=) operator in the join condition.
- Equi join can be an Inner join, Left Outer join, Right Outer join

## Oracle Semi Join

Semi-join is introduced in Oracle 8.0. It provides an efficient method of performing a WHERE EXISTS sub-query.





A semi-join returns one copy of each row in first table for which at least one match is found.

Semi-joins are written using the EXISTS construct.








## Departments table

```
CREATE TABLE "DEPARTMENTS"  
( "DEPARTMENT_ID" NUMBER(10,0) NOT NULL ENABLE,  
  "DEPARTMENT_NAME" VARCHAR2(50) NOT NULL ENABLE  
  CONSTRAINT "DEPARTMENTS_PK" PRIMARY KEY ("DEPARTMENT_ID") ENABLE  
)  
/
```

EDIT	DEPARTMENT_ID	DEPARTMENT_NAME
	1	a
	2	b
	3	c
	4	d
		row(s) 1 - 4 of 4

## Customer table

```
CREATE TABLE "CUSTOMER"  
( "CUSTOMER_ID" NUMBER,  
  "FIRST_NAME" VARCHAR2(4000),  
  "LAST_NAME" VARCHAR2(4000),  
  "DEPARTMENT_ID" NUMBER  
)  
/
```

EDIT	CUSTOMER_ID	FIRST_NAME	LAST_NAME	DEPARTMENT_ID
	1	aryan	tomar	-
	2	ajeet	maurya	-
	3	richa	goyal	-
	4	dhirubhai	jotwani	-
	32	alex	pandian	1
		row(s) 1 - 5 of 5		

### Execute this query

```
SELECT departments.department_id, departments.department_name
FROM departments
WHERE EXISTS
(
    SELECT 1
    FROM customer
    WHERE customer.department_id = departments.department_id
)
ORDER BY departments.department_id;
```

### Output

DEPARTMENT_ID	DEPARTMENT_NAME
1	a

## Difference between anti-join and semi-join

While a semi-join returns one copy of each row in the first table for which at least one match is found, an anti-join returns one copy of each row in the first table for which no match is found.

# Oracle Procedures and Functions and where clause

Friday, March 18, 2022 7:48 PM

## Oracle Procedures

A procedure is a group of PL/SQL statements that can be called by name. The call specification (sometimes called call spec) specifies a java method or a third-generation language routine so that it can be called from SQL and PL/SQL.

### Create Procedure

#### Syntax

```
CREATE [OR REPLACE] PROCEDURE procedure_name  
  [ (parameter [,parameter]) ]  
IS  
  [declaration_section]  
BEGIN  
  executable_section  
[EXCEPTION  
  exception_section]  
END [procedure_name];
```

Following are the three types of procedures that must be defined to create a procedure.

- **IN:** It is a default parameter. It passes the value to the subprogram.
- **OUT:** It must be specified. It returns a value to the caller.
- **IN OUT:** It must be specified. It passes an initial value to the subprogram and returns an updated value to the caller.

#### Procedure Code:

```
create or replace procedure "INSERTUSER"  
(id IN NUMBER,  
 name IN VARCHAR2)  
is  
begin  
  insert into user values(id,name);  
end;  
/
```

## Oracle program to call procedure

Let's see the code to call above created procedure.

```
BEGIN  
  insertuser(101,'Rahul');  
  dbms_output.put_line('record inserted successfully');  
END;
```

## Oracle Drop Procedure

### Syntax

```
DROP PROCEDURE procedure_name;
```

## Oracle Function

A function is a subprogram that is used to return a single value. You must declare and define a function before invoking it. It can be declared and defined at a same time or can be declared first and defined later in the same block.

## CREATE function in Oracle

### Syntax

```
CREATE [OR REPLACE] FUNCTION function_name  
  [ (parameter [,parameter]) ]  
RETURN return_datatype  
IS | AS  
  [declaration_section]  
BEGIN  
  executable_section  
[EXCEPTION  
  exception_section]  
END [function_name];
```

You must have define some parametrs before creating a procedure or a function. These parameters are

- **IN:** It is a default parameter. It passes the value to the subprogram.
- **OUT:** It must be specified. It returns a value to the caller.
- **IN OUT:** It must be specified. It passes an initial value to the subprogram and returns an updated value to the caller.

```
create or replace function adder(n1 in number, n2 in number)  
return number  
is  
  n3 number(8);  
begin  
  n3 :=n1+n2;  
return n3;  
end;
```

```

DECLARE
  n3 number(2);
BEGIN
  n3 := adder(11,22);
  dbms_output.put_line('Addition is: ' || n3);
END;
/

```

Output:

```

Addition is: 33
Statement processed.
0.05 seconds

```

```

DECLARE
  a number;
  b number;
  c number;
FUNCTION findMax(x IN number, y IN number)
RETURN number
IS
  z number;
BEGIN
  IF x > y THEN
    z:= x;
  ELSE
    Z:= y;
  END IF;

  RETURN z;
END;
BEGIN
  a:= 23;
  b:= 45;

  c := findMax(a, b);
  dbms_output.put_line(' Maximum of (23,45): ' || c);
END;

```

Recursive function

**DECLARE**

```
num number;  
factorial number;
```

**FUNCTION** fact(x number)**RETURN** number**IS**

```
f number;
```

**BEGIN**

```
IF x=0 THEN
```

```
    f := 1;
```

**ELSE**

```
    f := x * fact(x-1);
```

```
END IF;
```

```
RETURN f;
```

```
END;
```

**BEGIN**

```
num:= 6;
```

```
factorial := fact(num);
```


```
dbms_output.put_line(' Factorial ' || num || ' is ' || factorial);
```

```
END;
```

Dropping a Function:

```
DROP FUNCTION function_name;
```

We can use the following commands in SQL to make queries efficient.

- ☞ ALIASES
- ☞ AND
- ☞ AND & OR
- ☞ **BETWEEN** 
- ☞ COMPARISON OPERATORS
- ☞ EXISTS
- ☞ IN
- ☞ INTERSECT
- ☞ IS NOT NULL
- ☞ IS NULL
- ☞ LIKE
- ☞ NOT
- ☞ OR
- ☞ SUBQUERY
- ☞ TRUNCATE
- ☞ WHERE

## Comparison Operators

Comparison operator	Description
=	Equal
< >	Not Equal
!=	Not equal
>	Greater than
>=	Greater than or equal
<	Less than
<=	Less than or equal

## ORACLE LIKE CONDITION

In oracle, like condition is used with select, insert, update, and delete in where clause using wildcard. It allows pattern matching.

### Syntax

```
expression LIKE pattern [ESCAPE 'escape_character' ]
```

### Parameters

**expression:** name of column.

**pattern:** patter to be matched in expression. Pattern can be in one of the following:-

Wildcard	Explanation
%	Used for matching string
-	Used for matching single character

## ORACLE NOT condition

In Oracle, NOT condition is used with SELECT, INSERT, UPDATE or DELETE statement. It is also called NOT operator. It is used to negate the given condition.

### Syntax

```
NOT condition
```



ID	NAME	AGE
1	shristee	23
2	vishal	20
3	shashank	26
4	dolly	18
5	charu	28
6	shweta	26
7	sid	22
8	avinash	27
9	heena	29

### Example 1

Query: select \* from table1 where name not like 26

ORACLE Database Express Edition

User: SYSTEM

Home > SQL > SQL Commands

☒ Autocommit Display: 10

select id from table1 where age not like 26

Results Explain Describe Saved SQL History

ID
1
2
4
5
7
8
9
10

8 rows returned in 0.00 seconds [CSV Export](#)

### Inserting multiple row values into a table at a time

```

INSERT ALL
  INTO student(id, name) VALUES (1, 'shristee')
  INTO student(id, name) VALUES (2, 'heena')
  INTO student(id, name) VALUES (3, 'mohit')
  INTO student(id, name) VALUES (4, 'shashank')
  INTO student(id, name) VALUES (5, 'avinash')
  INTO student(id, name) VALUES (6, 'shweta')
  INTO student(id, name) VALUES (7, 'suman')
  INTO student(id, name) VALUES (8, 'rohan')
  INTO student(id, name) VALUES (9, 'ali')
  INTO student(id, name) VALUES (10, 'dolly')
  INTO student(id, name) VALUES (11, 'mona')
  INTO student(id, name) VALUES (12, 'kiran')
SELECT * FROM dual;

```

# Oracle Primary Key constraint

Friday, March 18, 2022 8:22 PM

## Creating a table with primary key constraint

```
CREATE TABLE table_name  
(  
    column1 datatype null/not null,  
    column2 datatype null/not null,  
    ...  
    CONSTRAINT constraint_name PRIMARY KEY (column1, column2, ... column_n  
);
```

## Adding the primary key constraint later creating table

```
ALTER TABLE table_name  
ADD CONSTRAINT constraint_name PRIMARY KEY (column1, column2, ... column_n);
```

## Dropping the primary key constraint

```
ALTER TABLE table_name  
DROP CONSTRAINT constraint_name;
```

### Example

```
ALTER TABLE student DROP CONSTRAINT student_pk ;
```

## Disabling primary key constraint

```
ALTER TABLE table_name  
DISABLE CONSTRAINT constraint_name;
```

### Example

```
ALTER TABLE student DISABLE CONSTRAINT student_pk ;
```

## Enabling primary key constraint

```
ALTER TABLE table_name  
ENABLE CONSTRAINT constraint_name;
```

### Example

```
ALTER TABLE student ENABLE CONSTRAINT student_pk ;
```