What is Primary key

A primary key is a single field or combination of fields that contains a unique record. It must be filled. None of the field of primary key can contain a null value. A table can have only one primary key.

1. **CREATE** **TABLE** customers
2. ( customer\_id number(10) NOT NULL,
3. customer\_name varchar2(50) NOT NULL,
4. city varchar2(50),
5. **CONSTRAINT** customers\_pk **PRIMARY** **KEY** (customer\_id)
6. );

## SQL FOREIGN KEY Constraint

A FOREIGN KEY is a field (or collection of fields) in one table, that refers to the [PRIMARY KEY](https://www.w3schools.com/sql/sql_primarykey.asp) in another table.

The table with the foreign key is called the child table, and the table with the primary key is called the referenced or parent table.

Look at the following two tables:

### **Persons Table**

|  |  |  |  |
| --- | --- | --- | --- |
| **PersonID** | **LastName** | **FirstName** | **Age** |
| 1 | Hansen | Ola | 30 |
| 2 | Svendson | Tove | 23 |
| 3 | Pettersen | Kari | 20 |

### **Orders Table**

|  |  |  |
| --- | --- | --- |
| **OrderID** | **OrderNumber** | **PersonID** |
| 1 | 77895 | 3 |
| 2 | 44678 | 3 |
| 3 | 22456 | 2 |
| 4 | 24562 | 1 |

Notice that the "PersonID" column in the "Orders" table points to the "PersonID" column in the "Persons" table.

The "PersonID" column in the "Persons" table is the PRIMARY KEY in the "Persons" table.

The "PersonID" column in the "Orders" table is a FOREIGN KEY in the "Orders" table.

The FOREIGN KEY constraint prevents invalid data from being inserted into the foreign key column, because it has to be one of the values contained in the parent table.

## SQL FOREIGN KEY on CREATE TABLE

The following SQL creates a FOREIGN KEY on the "PersonID" column when the "Orders" table is created:

**SQL Server / Oracle / MS Access:**

CREATE TABLE Orders (  
    OrderID int NOT NULL PRIMARY KEY,  
    OrderNumber int NOT NULL,  
    PersonID int FOREIGN KEY REFERENCES Persons(PersonID)  
);

To allow naming of a FOREIGN KEY constraint, and for defining a FOREIGN KEY constraint on multiple columns, use the following SQL syntax:

**MySQL / SQL Server / Oracle / MS Access:**

CREATE TABLE Orders (  
    OrderID int NOT NULL,  
    OrderNumber int NOT NULL,  
    PersonID int,  
    PRIMARY KEY (OrderID),  
    CONSTRAINT FK\_PersonOrder FOREIGN KEY (PersonID)  
    REFERENCES Persons(PersonID)  
);

insert into customers values(2,'aa','thane');

# **CREATE TABLE AS Statement**

The CREATE TABLE AS statement is used to create a table from an existing table by copying the columns of existing table.

1. **CREATE** **TABLE** new\_table
2. **AS** (**SELECT** \* **FROM** old\_table);

Create Table Example: copying all columns of another table

In this example, we are creating a "newcustomers" table by copying all the columns from the already existing table "Customers".

1. **CREATE** **TABLE** newcustomers
2. **AS** (**SELECT** \*   **FROM** customers  **WHERE** customer\_id < 5000);

Table created.

This table is named as "newcustomers" and having the same columns of "customers" table.

# **Oracle ALTER TABLE Statement**

In Oracle, ALTER TABLE statement specifies how to add, modify, drop or delete columns in a table. It is also used to rename a table.

## How to add column in a table

**Syntax:**

1. **ALTER** **TABLE** table\_name
2. **ADD** column\_name **column**-definition;

**Example:**

Consider that already existing table customers. Now, add a new column customer\_age into the table customers.

1. **ALTER** **TABLE** customers
2. **ADD** customer\_age varchar2(50);

Now, a new column "customer\_age" will be added in customers table.

How to add multiple columns in the existing table

**Example**

1. **ALTER** **TABLE** customers
2. **ADD** (customer\_type varchar2(50),
3. customer\_address varchar2(50));

Now, two columns customer\_type and customer\_address will be added in the table customers.­

How to modify column of a table

**Example:**

1. **ALTER** **TABLE** customers
2. **MODIFY** customer\_name varchar2(100) not null;

Now the column column\_name in the customers table is modified

to varchar2 (100) and forced the column to not allow null values.

How to modify multiple columns of a table

1. **ALTER** **TABLE** customers
2. **MODIFY** (customer\_name varchar2(100) not null,
3. city varchar2(100));

This will modify both the customer\_name and city columns in the table.

How to drop column of a table

**Syntax:**

1. **ALTER** **TABLE** table\_name
2. **DROP** **COLUMN** column\_name;

**Example:**

1. **ALTER** **TABLE** customers
2. **DROP** **COLUMN** customer\_name;

This will drop the customer\_name column from the table.

How to rename column of a table

**Syntax:**

**Example:**

1. **ALTER** **TABLE** customers
2. RENAME **COLUMN** customer\_name **to** cname;

This will rename the column customer\_name into cname.

How to rename table

1. **ALTER** **TABLE** customers
2. RENAME **TO** retailers;

This will rename the customer table into "retailers" table

DROP TABLE Example

1. **DROP** **TABLE** customers;

This will drop the table named customers.

DROP TABLE Example with PURGE parameter

1. **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 SELECT Statement**

The Oracle SELECT statement is used to retrieve data from one or more than one tables, object tables, views, object views etc.

1. **SELECT** expressions
2. **FROM** tables
3. **WHERE** conditions;

Select Example: select all fields

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

1. **SELECT** \*
2. **FROM** customers;

Select Example: select specific fields

**Example**

1. **SELECT** age, address, salary
2. **FROM** customers
3. **WHERE**  age < 25
4. AND salary > '20000'
5. **ORDER** **BY** age **ASC**, salary **DESC**;

# **Oracle Insert Statement**

In Oracle, INSERT statement is used to add a single record or multiple records into the table.

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

1. **INSERT** **INTO** **table**
2. (column1, column2, ... column\_n )
3. **VALUES**
4. (expression1, expression2, ... expression\_n );
5. **INSERT** **INTO** suppliers
6. (supplier\_id, supplier\_name)
7. **VALUES**
8. (50, 'Flipkart');

we insert values to the "suppliers" table from "customers" table. Both tables are already created with their respective columns.

**Execute this query:**

1. **INSERT** **INTO** suppliers
2. (supplier\_id, supplier\_name)
3. **SELECT** age, address
4. **FROM** customers
5. **WHERE** age > 20;

You can even check the number of rows that you want to insert by following statement:

1. **SELECT** count(\*)
2. **FROM** customers
3. **WHERE** age > 20;

**Output:**

Count(\*)

4

# **Oracle INSERT ALL statement**

The Oracle INSERT ALL statement is used to insert multiple rows with a single INSERT statement. You can insert the rows into one table or multiple tables by using only one SQL command.

**Syntax**

1. **INSERT** ALL
2. **INTO** table\_name (column1, column2, column\_n) **VALUES** (expr1, expr2, expr\_n)
3. **INTO** table\_name(column1, column2, column\_n) **VALUES** (expr1, expr2, expr\_n)
4. **INTO** table\_name (column1, column2, column\_n) **VALUES** (expr1, expr2, expr\_n)
5. **SELECT** \* **FROM** dual;

Oracle INSERT ALL Example

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

1. **INSERT** ALL
2. **INTO** suppliers (supplier\_id, supplier\_name) **VALUES** (20, 'Google')
3. **INTO** suppliers (supplier\_id, supplier\_name) **VALUES** (21, 'Microsoft')
4. **INTO** suppliers (supplier\_id, supplier\_name) **VALUES** (22, 'Apple')
5. **SELECT** \* **FROM** dual;

## 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.

1. **INSERT** ALL
2. **INTO** suppliers (supplier\_id, supplier\_name) **VALUES** (30, 'Google')
3. **INTO** suppliers (supplier\_id, supplier\_name) **VALUES** (31, 'Microsoft')
4. **INTO** customers (age, **name**, address) **VALUES** (29, 'Luca Warsi', 'New York')
5. **SELECT** \* **FROM** dual;

# **Oracle UPDATE Statement**

In Oracle, UPDATE statement is used to update the existing records in a table. You can update a table in 2 ways.

## Traditional Update table method

**Syntax:**

1. **UPDATE** **table**
2. **SET** column1 = expression1,
3. column2 = expression2,
4. ...
5. column\_n = expression\_n
6. **WHERE** conditions;

## Update Table by selecting rocords from another table

**Syntax:**

1. **UPDATE** table1
2. **SET** column1 = (**SELECT** expression1
3. **FROM** table2
4. **WHERE** conditions)
5. **WHERE** conditions;

Oracle Update Example: (Update single column)

1. **UPDATE** suppliers
2. **SET** supplier\_name = 'Kingfisher'
3. **WHERE** supplier\_id = 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

1. **UPDATE** suppliers
2. **SET**  = 'Agra' supplier\_address ,
3. supplier\_name = 'Bata shoes'
4. **WHERE** supplier\_id = 1;

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

1. **UPDATE** customers
2. **SET** **name** = (**SELECT** supplier\_name
3. **FROM** suppliers
4. **WHERE** suppliers.supplier\_name = customers.**name**)
5. **WHERE** age < 25;

# **Oracle DELETE Statement**

In Oracle, DELETE statement is used to remove or delete a single record or multiple records from a table.

**yntax**

1. **DELETE** **FROM** table\_name
2. **WHERE** conditions

Oracle Delete Example: On one condition

1. **DELETE** **FROM** customers
2. **WHERE** **name** = 'Sohan';

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

Oracle Delete Example: On multiple conditions

1. **DELETE** **FROM** customers
2. **WHERE** last\_name = 'Maurya'
3. 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.

**Syntax**

1. **TRUNCATE** **TABLE** [schema\_name.]table\_name

Oracle TRUNCATE Table Example

Consider a table named "customers" and execute the following query to truncate this

1. **TRUNCATE** **TABLE** customers;

Oracle DELETE Table Example

1. **DELETE** **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 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.

1. **SELECT** expressions
2. **FROM** tables
3. **WHERE** conditions
4. **ORDER** **BY** expression [ **ASC** | **DESC** ];

Oracle ORDER BY Example: (without ASC/DESC attribute)

Let's take a table "supplier"

**Supplier table:**

1. **CREATE** **TABLE**  "SUPPLIER"
2. (    "SUPPLIER\_ID" NUMBER,
3. "FIRST\_NAME" VARCHAR2(4000),
4. "LAST\_NAME" VARCHAR2(4000)
5. )
6. **SELECT** \*
7. **FROM** supplier
8. **ORDER** **BY** last\_name;

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:**

1. **SELECT** \*
2. **FROM** supplier
3. **ORDER** **BY** last\_name **DESC**;

# **Oracle GROUP BY Clause**

In Oracle GROUP BY clause is used with SELECT statement to collect data from multiple records and group the results by one or more columns.

**Syntax:**

1. **SELECT** expression1, expression2, ... expression\_n,
2. aggregate\_function (aggregate\_expression)
3. **FROM** tables
4. **WHERE** conditions
5. **GROUP** **BY** expression1, expression2, ... expression\_n;

Oracle GROUP BY Example: (with SUM function)

Let's take a table "salesdepartment"

**Salesdepartment table:**

1. **CREATE** **TABLE**  "SALESDEPARTMENT"
2. (    "ITEM" VARCHAR2(4000),
3. "SALE" NUMBER,
4. "BILLING\_ADDRESS" VARCHAR2(4000)
5. )
6. **SELECT** item, SUM(sale) **AS** "Total sales"
7. **FROM** salesdepartment
8. **GROUP** **BY** item;

# **Oracle HAVING Clause**

In Oracle, HAVING Clause is used with GROUP BY Clause to restrict the groups of returned rows where condition is TRUE.

1. **CREATE** **TABLE**  "SALESDEPARTMENT"
2. (    "ITEM" VARCHAR2(4000),
3. "SALE" NUMBER,
4. "BILLING\_ADDRESS" VARCHAR2(4000)
5. )
6. /

**Execute this query:**

2. **SELECT** item, SUM(sale) **AS** "Total sales"
3. **FROM** salesdepartment
4. **GROUP** **BY** item
5. **HAVING** SUM(sale) < 1000;

SELECT COUNT(CustomerID), Country

FROM Customers

GROUP BY Country

HAVING COUNT(CustomerID) > 5;Number of Records: 5

|  |  |
| --- | --- |
| **COUNT(CustomerID)** | **Country** |
| 9 | Brazil |
| 11 | France |
| 11 | Germany |
| 7 | UK |
| 13 | USA |

# **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.

## Oracle CREATE VIEW

**Syntax:**

1. **CREATE** **VIEW** view\_name **AS**
2. **SELECT** columns
3. **FROM** tables
4. **WHERE** conditions;

**Example:**

Let's take an example to create view. In this example, we are creating two tables suppliers and orders first.

**Suppliers table:**

2. **CREATE** **TABLE**  "SUPPLIERS"
3. (    "SUPPLIER\_ID" NUMBER,
4. "SUPPLIER\_NAME" VARCHAR2(4000),
5. "SUPPLIER\_ADDRESS" VARCHAR2(4000)
6. )

**Orders table:**

1. **CREATE** **TABLE**  "ORDERS"
2. (    "ORDER\_NO." NUMBER,
3. "QUANTITY" NUMBER,
4. "PRICE" NUMBER
5. )

**Create View Query:**

1. **CREATE** **VIEW** sup\_orders **AS**
2. **SELECT** suppliers.supplier\_id, orders.quantity, orders.price
3. **FROM** suppliers
4. **INNER** JOIN orders
5. **ON** suppliers.supplier\_id = supplier\_id
6. **WHERE** suppliers.supplier\_name = 'VOJO';

**Output:**

View created.

You can now check the Oracle VIEW by this query:

1. **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

In Oracle, the CREATE OR REPLACE VIEW statement is used to modify the definition of an Oracle VIEW without dropping it.

**Syntax:**

1. **CREATE** OR REPLACE **VIEW** view\_name **AS**
2. **SELECT** columns
3. **FROM** **table**
4. **WHERE** conditions;

**Example:**

Execute the following query to update the definition of Oracle VIEW called sup\_orders without dropping it.

1. **CREATE** or REPLACE **VIEW** sup\_orders **AS**
2. **SELECT** suppliers.supplier\_id, orders.quantity, orders.price
3. **FROM** suppliers
4. **INNER** JOIN orders
5. **ON** suppliers.supplier\_id = supplier\_id
6. **WHERE** suppliers.supplier\_name = 'HCL';
7. **SELECT** \* **FROM** sup\_orders;

Output:

SUPPLIER\_ID QUANTITY PRICE

1 35 70

1 26 125

1 18 100

## Oracle DROP VIEW

The DROP VIEW statement is used to remove or delete the VIEW completely.

1. **DROP** **VIEW** view\_name;

**Example:**

1. **DROP** **VIEW** sup\_orders;

## SQL CREATE INDEX Statement

**The CREATE INDEX statement is used to create indexes in tables.**

**Indexes are used to retrieve data from the database more quickly than otherwise. The users cannot see the indexes, they are just used to speed up searches/queries.**

### **CREATE INDEX Syntax**

Creates an index on a table. Duplicate values are allowed:

CREATE INDEX index\_name  
ON table\_name (column1, column2, ...);

### **CREATE UNIQUE INDEX Syntax**

Creates a unique index on a table. Duplicate values are not allowed:

CREATE UNIQUE INDEX index\_name  
ON table\_name (column1, column2, ...);

**Note:** The syntax for creating indexes varies among different databases. Therefore: Check the syntax for creating indexes in your database.

## CREATE INDEX Example

The SQL statement below creates an index named "idx\_lastname" on the "LastName" column in the "Persons" table:

CREATE INDEX idx\_lastname  
ON Persons (LastName);

If you want to create an index on a combination of columns, you can list the column names within the parentheses, separated by commas:

CREATE INDEX idx\_pname  
ON Persons (LastName, FirstName);

## DROP INDEX Statement

The DROP INDEX statement is used to delete an index in a table.

**DB2/Oracle:**

DROP INDEX index\_name;

# **ORACLE ALIASES**

In Oracle, aliasing can also be done in column name as well as in table name. Aliasing is done to give a temporary to a column or table.

## Example 1

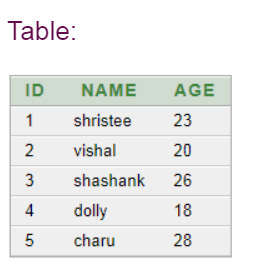
select id, name as Student\_name from table1

## Example 2

select s.id, s.name from table1s

# **ORACLE AND**

In Oracle, AND is used in select, insert, delete or update statement for checking two or more conditions.



select id, name from table1 where name='dolly' AND age=18

## Example 2

select id, name from table1 where name='shristee' AND id=1

# **ORACLE AND & OR**

In Oracle, AND & OR can be combined and used in select, insert, delete or update statement for checking two or more conditions.

## Example 1

select id, name, age from table1 where id>2 AND age<28 OR age>25

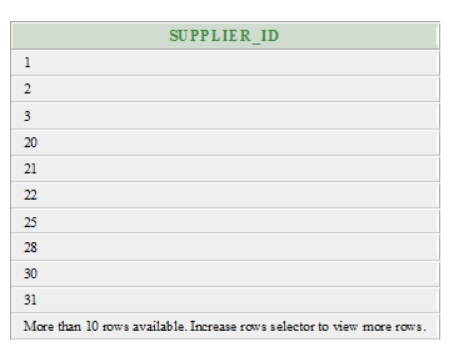
# **Oracle UNION Operator**

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)

1. **SELECT** supplier\_id
2. **FROM** suppliers
3. **UNION**
4. **SELECT** supplier\_id
5. **FROM** order\_details



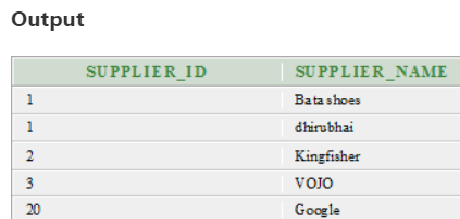
In this example, supplier\_id is defined in both of the table "suppliers" and "order\_details". After the UNION, it would appear once in the result set because Oracle UNION operator removes duplicate sets.

#### **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.

1. **SELECT** supplier\_id, supplier\_name
2. **FROM** suppliers
3. **WHERE** supplier\_id <= 20
4. **UNION**
5. **SELECT** s\_id, s\_name
6. **FROM** shopkeepers
7. **WHERE** s\_name = 'dhirubhai'
8. **ORDER** **BY** 1;



In the above example, result is sorted by supplier\_name/s\_name in ascending order, as denoted by ORDER BY 1.

# **Oracle UNION ALL Operator**

In Oracle, the UNION ALL operator is used to combine the result sets of 2 or more SELECT statements. It is different from UNION operator in a way that it does not remove duplicate rows between the various SELECT statements. It returns all of the rows.

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

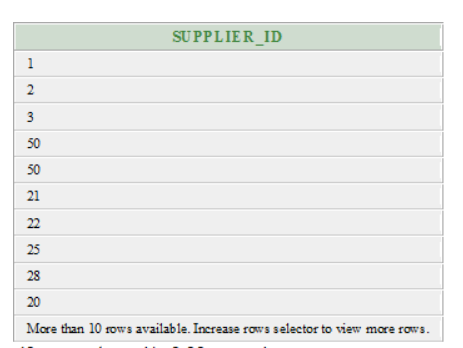
## Difference between UNION and UNION ALL operators

UNION operator removes duplicate rows while UNION ALL operator does not remove duplicate rows.

Oracle UNION ALL Operator Example

1. **SELECT** supplier\_id
2. **FROM** suppliers
3. **UNION** ALL
4. **SELECT** supplier\_id
5. **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.

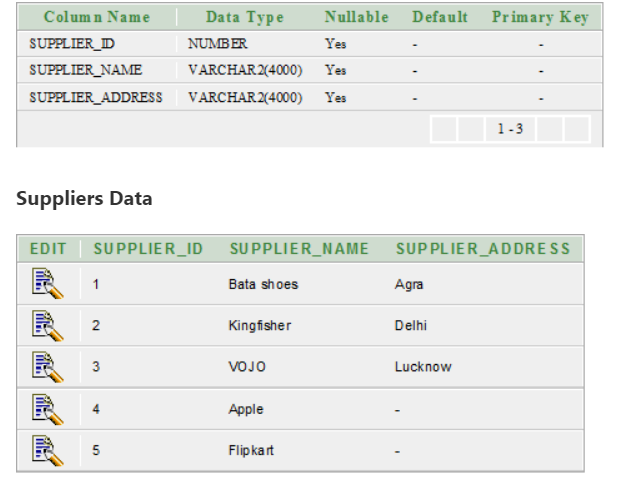


# **Oracle INTERSECT Operator**

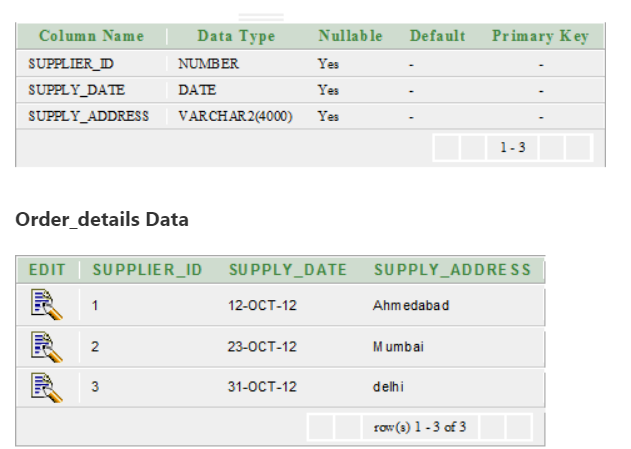
In Oracle, INTERSECT Operator is used to return the results of 2 or more SELECT statement. It picks the common or intersecting records from compound SELECT queries.

## Oracle INTERSECT Example: (with single expression)

**Suppliers Table**

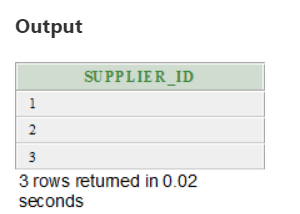


**Order\_details Table**



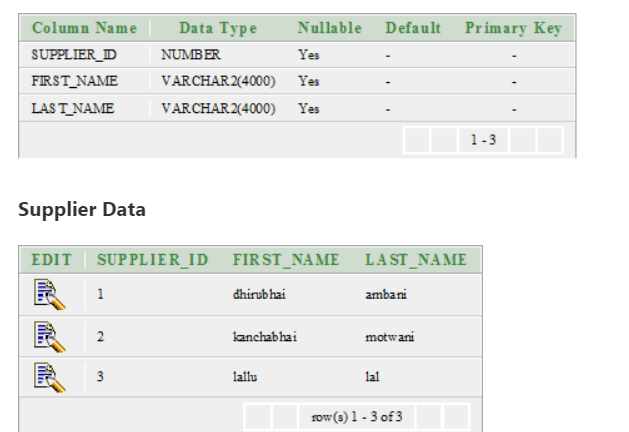
1. **SELECT** supplier\_id
2. **FROM** suppliers
3. **INTERSECT**
4. **SELECT** supplier\_id
5. **FROM** order\_details;

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.

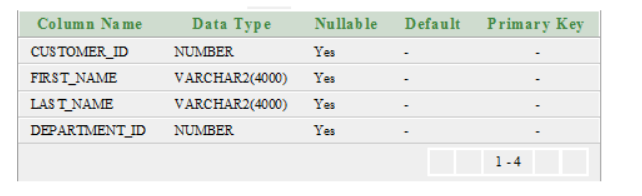


## Oracle INTERSECT Example: (with multiple expressions)

**Supplier Table**

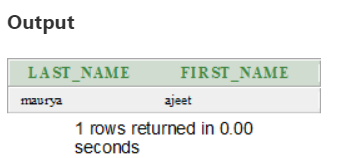


**Customer Table**





1. **SELECT** supplier\_id, last\_name, first\_name
2. **FROM** supplier
3. **WHERE** first\_name <> 'dhirubhai'
4. **INTERSECT**
5. **SELECT** customer\_id, last\_name, first\_name
6. **FROM** customer
7. **WHERE** customer\_id < 5;



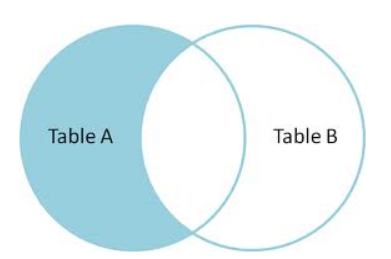
The above example returns the records from the supplier table where the supplier\_id, last\_name and first\_name values match the customer\_id, last\_name, and first\_name value of customer table.

# **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.

Each SELECT statement has a dataset and the MINUS operator returns all documents from the first dataset and then removes all documents from the second dataset.

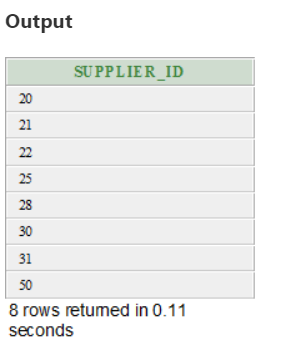
**For example**



Oracle MINUS Example

This example will return one field with the same datatype from two tables "suppliers" and "order\_details".

1. **SELECT** supplier\_id
2. **FROM** suppliers
3. MINUS
4. **SELECT** supplier\_id
5. **FROM** order\_details;



**AGGREGATE FUNCTIONS**