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.

### **3. Data Control Language**

DCL commands are used to grant and take back authority from any database user.

Here are some commands that come under DCL:

* Grant
* Revoke

**a. Grant:** It is used to give user access privileges to a database.

**Example**

1. GRANT SELECT, UPDATE ON MY\_TABLE TO SOME\_USER, ANOTHER\_USER;

**b. Revoke:** It is used to take back permissions from the user.

**Example**

1. REVOKE SELECT, UPDATE ON MY\_TABLE FROM USER1, USER2;

### **4. Transaction Control Language**

TCL commands can only use with DML commands like INSERT, DELETE and UPDATE only.

These operations are automatically committed in the database that's why they cannot be used while creating tables or dropping them.

Here are some commands that come under TCL:

* COMMIT
* ROLLBACK
* SAVEPOINT

**a. Commit:** Commit command is used to save all the transactions to the database.

**Syntax:**

1. COMMIT;

**Example:**

1. DELETE FROM CUSTOMERS
2. WHERE AGE = 25;
3. COMMIT;

**b. Rollback:** Rollback command is used to undo transactions that have not already been saved to the database.

**Syntax:**

1. ROLLBACK;

**Example:**

1. DELETE FROM CUSTOMERS
2. WHERE AGE = 25;
3. ROLLBACK;

**c. SAVEPOINT:** It is used to roll the transaction back to a certain point without rolling back the entire transaction.

**Syntax:**

1. SAVEPOINT SAVEPOINT\_NAME;

## DDL Commands

The Data Definition Language (DDL) commands are as follows −

### **GRANT Command**

It is employed to grant a privilege to a user. [**GRANT command**](https://www.tutorialspoint.com/mysql/mysql_grant_statement.htm) allows specified users to perform specified tasks

**Syntax**

GRANT privilege\_name on objectname to user;

Here,

* privilege names are SELECT,UPDATE,DELETE,INSERT,ALTER,ALL
* objectname is table name
* user is the name of the user to whom we grant privileges

### **REVOKE Command**

It is employed to remove a privilege from a user. [**REVOKE**](https://www.tutorialspoint.com/mysql/mysql_revoke_statement.htm) helps the owner to cancel previously granted permissions.

**Syntax**

 REVOKE privilege\_name on objectname from user;

Here,

* privilege names are SELECT,UPDATE,DELETE,INSERT,ALTER,ALL
* objectname is table name
* user is the name of the user whose privileges are removing

### **Example**

GRANT SELECT, UPDATE ON employees TO Bhanu

Explanation − Firstly, to give the permissions to user, we have to use **GRANT** command. The privileges are [**SELECT**](https://www.tutorialspoint.com/mysql/mysql-select-query.htm) because to view the records and [**UPDATE**](https://www.tutorialspoint.com/mysql/mysql-update-query.htm) to modify the records. The *objectname* is table name which is *Employee*. The user name is "bhanu".

REVOKE SELECT, UPDATE ON employees TO Bhanu

Explanation − Firstly, to revoke the permissions to user, we have to use ***REVOKE*** command. The privileges Need to revoke are ***SELECT*** because to view the records and ***UPDATE*** to modify the records. The *objectname* is table name which is *Employee*. The user name is "Bhanu".

**AGGREGATE FUNCTIONS**

## ****COUNT Function in Oracle with Examples****

In this article, I am going to discuss the **COUNT** **Function in Oracle** with Examples. Please read our previous article where we discussed [**Multiple Row** **Functions in Oracle**](https://dotnettutorials.net/lesson/multiple-row-functions-in-oracle/) with Examples.

##### **COUNT() Function in Oracle:**

The COUNT function in Oracle is used to count the data rows returned in the result set. The Oracle COUNT function counts distinct or all values in data rows returned in a result set. Following are the syntaxes to use the COUNT function in Oracle.

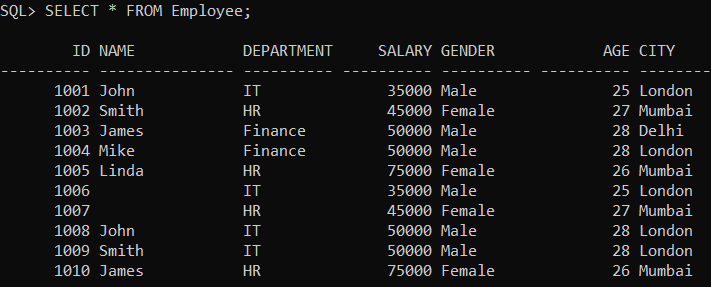
**SELECT COUNT(\*) FROM tablename;**

**SELECT COUNT(columnname) FROM tablename;**

**SELECT COUNT (DISTINCT columnname) FROM tablename;**

##### **Examples to Understand COUNT Function in Oracle:**

We are going to use the following Employee table to understand the need and use of the COUNT Function in Oracle with Examples.



Please execute the below SQL query to drop the existing Employee table and create a new Employee table with the required sample data.

**DROP** **Table** Employee;

**CREATE** **TABLE** Employee (

Id **INT** **PRIMARY KEY**,

Name **VARCHAR**(15),

Department **VARCHAR**(10),

Salary **NUMBER**(8, 2),

Gender **VARCHAR**(10),

Age **INT**,

City **VARCHAR**(10)

);

**INSERT** **INTO** Employee (Id, Name, Department, Salary, Gender, Age, City) **VALUES** (1001, 'John', 'IT', 35000, 'Male', 25, 'London');

**INSERT** **INTO** Employee (Id, Name, Department, Salary, Gender, Age, City) **VALUES** (1002, 'Smith', 'HR', 45000, 'Female', 27, 'Mumbai');

**INSERT** **INTO** Employee (Id, Name, Department, Salary, Gender, Age, City) **VALUES** (1003, 'James', 'Finance', 50000, 'Male', 28, 'Delhi');

**INSERT** **INTO** Employee (Id, Name, Department, Salary, Gender, Age, City) **VALUES** (1004, 'Mike', 'Finance', 50000, 'Male', 28, 'London');

**INSERT** **INTO** Employee (Id, Name, Department, Salary, Gender, Age, City) **VALUES** (1005, 'Linda', 'HR', 75000, 'Female', 26, 'Mumbai');

**INSERT** **INTO** Employee (Id, Name, Department, Salary, Gender, Age, City) **VALUES** (1006, **NULL**, 'IT', 35000, 'Male', 25, 'London');

**INSERT** **INTO** Employee (Id, Name, Department, Salary, Gender, Age, City) **VALUES** (1007, **NULL**, 'HR', 45000, 'Female', 27, 'Mumbai');

**INSERT** **INTO** Employee (Id, Name, Department, Salary, Gender, Age, City) **VALUES** (1008, 'John', 'IT', 50000, 'Male', 28, 'London');

**INSERT** **INTO** Employee (Id, Name, Department, Salary, Gender, Age, City) **VALUES** (1009, 'Smith', 'IT', 50000, 'Male', 28, 'London');

**INSERT** **INTO** Employee (Id, Name, Department, Salary, Gender, Age, City) **VALUES** (1010, 'James', 'HR', 75000, 'Female', 26, 'Mumbai');

##### **Example: Count the Total Number of employees in the Employee table.**

Now, let’s count the data rows in the Employee table using the COUNT function in Oracle by executing the below SQL Statement.

**SELECT COUNT(\*) FROM Employee;**

When you execute the above query, you will get the following output. As the Employee table contains 10 rows, so here you will get the output as 10.

##### **COUNT Function with Alias Name in Oracle**

We can also use Alias to give the name to the data column. By default, the column name would be **COUNT(\*)**that you can see in the previous example. Instead of **COUNT(\*)**, we want to provide the name as Total\_Employees. Then we can do the same by using the ALIAS name as shown in the below SQL Statement.

**SELECT COUNT(\*) AS “Total\_Employees” FROM Employee;**

Now once you execute the above SQL statement, you will get the following output. Notice, now the column name is Total\_Employees instead of **COUNT(\*).**

##### **Count Distinct Values in Oracle**

We already know we have 3 distinct values in the Department column of the Employee table. Let us get the number of employees in each department by using the COUNT function. Here, we also need to use the GROUP BY clause to group the employees by department and then apply the COUNT function to each department to get the desired result. Following is the SQL query.

**SELECT** Department, **COUNT**(\*) **AS** "Total\_Employees"

**FROM** Employee

**GROUP** **BY** Department;

Once you execute the above query, you will get the following output. As you can see, the above SQL statement will get only the distinct values from the column name department and its total number of appearances on the employee table.

**Note:** The COUNT(\*) will count all the rows (duplicates & nulls) in a table.

##### **COUNT(<COLUMN NAME>) in Oracle:**

This function is used for counting all values including duplicate values but not null values from a column. As you can see in the Employee table, two employees’ names are NULL and three employees’ names are duplicated. So, let us execute the below SQL statement and pass Name as a parameter to the COUNT function.

**SELECT COUNT(NAME) FROM Employee;**

Once you execute the above query, you will get the following output. As you can see, we are getting the output as 8. This is because the **COUNT(<COLUMN NAME>)**ignores the NULL values but includes the duplicate values.

##### **COUNT(DISTINCT <COLUMN NAME>) in Oracle:**

This function is used for counting the unique values from a column excluding the NULL values. Here “DISTINCT” keyword is eliminating duplicate values.

**SELECT COUNT(DISTINCT NAME) FROM Employee;**

Once you execute the above query, you will get the following output. If you look at the employee table we have 5 unique employee names excluding the NULL values and this is the reason why we get the output as 5.

## ****SUM Function in Oracle with Examples****

In this article, I am going to discuss the **SUM** **Function in Oracle** with Examples. Please read our previous article where we discussed [**COUNT** **Function in Oracle**](https://dotnettutorials.net/lesson/count-function-in-oracle/) with Examples.

##### **SUM Function in Oracle:**

The SUM Function in Oracle is used to return the total sum of a given numeric column. The SUM function will only work on numeric data types. For columns containing other than numeric values, it will give you an error i.e. Invalid Number. Following is the syntax to use the SUM Function in Oracle.

**SELECT SUM(column) FROM TableName;**

##### **Examples to Understand SUM Function in Oracle:**

We are going to use the following Employee table to understand the need and use of SUM Function in Oracle with Examples.

**SELECT SUM(Salary) As TotalSalary FROM Employee;**

Now when you execute the above query, you will get the following output.

##### **SUM Function with Group by Clause in Oracle**

Now we will calculate the Total Salary of each department from the Employee table. To do so, we need to pass the Salary column to the SUM function as well as we need to group the record by department using the GROUP BY clause as shown in the below SQL Query.

**SELECT** Department, **SUM**(Salary) **As** TotalSalary

**FROM** Employee

**GROUP** **BY** Department;

# **Subqueries in Oracle**

In Oracle, a subquery is a query within a query. We can create subqueries within our SQL statements. These subqueries can reside in the WHERE clause, in the FROM clause, or in the SELECT clause.

**CREATE** **TABLE** **EMPLOYEE**

(

**EMPNO** **NUMBER**(4) **NOT NULL**,

**ENAME** VARCHAR2(10),

**JOB** VARCHAR2(9),

**MGR** **NUMBER**(4),

**HIREDATE** **DATE**,

**SAL** **NUMBER**(7, 2),

**COMM** **NUMBER**(7, 2),

**DEPTNO** **NUMBER**(2)

);

**INSERT** **INTO** **EMPLOYEE** **VALUES** (7369, 'SMITH', 'CLERK', 7902, TO\_DATE('17-DEC-1980', 'DD-MON-YYYY'), 800, **NULL**, 20);

**INSERT** **INTO** **EMPLOYEE** **VALUES** (7499, 'ALLEN', 'SALESMAN', 7698, TO\_DATE('20-FEB-1981', 'DD-MON-YYYY'), 1600, 300, 30);

**INSERT** **INTO** **EMPLOYEE** **VALUES** (7521, 'WARD', 'SALESMAN', 7698, TO\_DATE('22-FEB-1981', 'DD-MON-YYYY'), 1250, 500, 30);

**INSERT** **INTO** **EMPLOYEE** **VALUES** (7566, 'JONES', 'MANAGER', 7839, TO\_DATE('2-APR-1981', 'DD-MON-YYYY'), 2975, **NULL**, 20);

**SELECT \* FROM EMPLOYEE WHERE SAL = (SELECT MAX( SAL ) FROM EMPLOYEE);**



the query that retrieves the max salary is called the subquery, and the query that selects the detailed employee data is called the outer query. We can say that the subquery or inner query is nested within the outer query. Note that a subquery must appear within parentheses (). Oracle evaluates the whole query in two steps:

1. First, execute the subquery.
2. Second, use the result of the subquery in the outer query.

**Note:** A subquery that is nested within the FROM clause of the SELECT statement is called an inline view. A subquery nested in the WHERE clause of the SELECT statement is called a nested subquery.

##### **Subqueries in Oracle**

A query inside another query is called a subquery or nested query in Oracle. The subquery is having two more queries. those are as follows,

1. **Inner / Child / Sub query**
2. **Outer / Parent / Main query**

**Syntax: SELECT \* FROM <TN> WHERE <CONDITION> (SELECT \* FROM ………………);**

As per the execution process of the subquery, it is again classified into two categorized.

1. **Non-Correlated Subqueries:** In non-correlated subqueries, first, the inner query will execute and return value, and later outer query will execute.
2. **Correlated Subqueries:** In Correlated subqueries first outer query will execute and return value and later inner query will execute.

The Non-Correlated Subqueries are again classified into 4 types are as follows.

1. [**Single Row Subquery**](https://dotnettutorials.net/lesson/single-row-subquery-in-oracle/)
2. [**Multiple Row Subquery**](https://dotnettutorials.net/lesson/multiple-row-subquery-in-oracle/)
3. [**Multiple Column Subquery**](https://dotnettutorials.net/lesson/multiple-column-subquery-in-oracle/)
4. [**Inline View Subquery**](https://dotnettutorials.net/lesson/inline-view-subquery-in-oracle/)

##### **Advantages of Oracle Subqueries**

The following are the main advantages of subqueries:

1. Provide an alternative way to query data that would require complex joins and unions.
2. Make the complex queries more readable.
3. Allow a complex query to be structured in a way that it is possible to isolate each part.

## subquery= outer + inner. The following is our subquery which includes the outer and inner query.

## ****Single Row Subquery in Oracle with Examples****

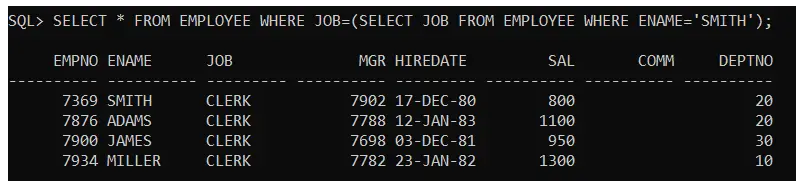
1. **SELECT \* FROM EMPLOYEE WHERE SAL = (SELECT MAX(SAL) FROM EMPLOYEE);**
2. When you execute the above subquery, you will get the following output which shows the first highest salary employee details from the Employee table.

###### **Example2:**

Whose employee job is the same as the job of ‘SMITH’?

**SELECT** \* **FROM** **EMPLOYEE** **WHERE** **JOB**=(**SELECT** **JOB** **FROM** **EMPLOYEE** **WHERE** **ENAME**='SMITH');

When you execute the above subquery, you will get the following output.

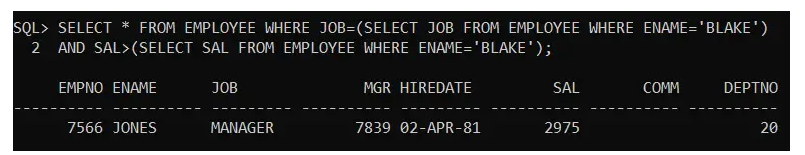


###### **Example4:**

Whose employee job is same as the job of “BLAKE” and who are earning Salary more than “BLAKE” salary?  
Solution:

**SELECT** \* **FROM** **EMPLOYEE** **WHERE** **JOB**=(**SELECT** **JOB** **FROM** **EMPLOYEE** **WHERE** **ENAME**='BLAKE') AND **SAL**>(**SELECT** **SAL** **FROM** **EMPLOYEE** **WHERE** **ENAME**='BLAKE');

When you execute the above subquery, you will get the following output.



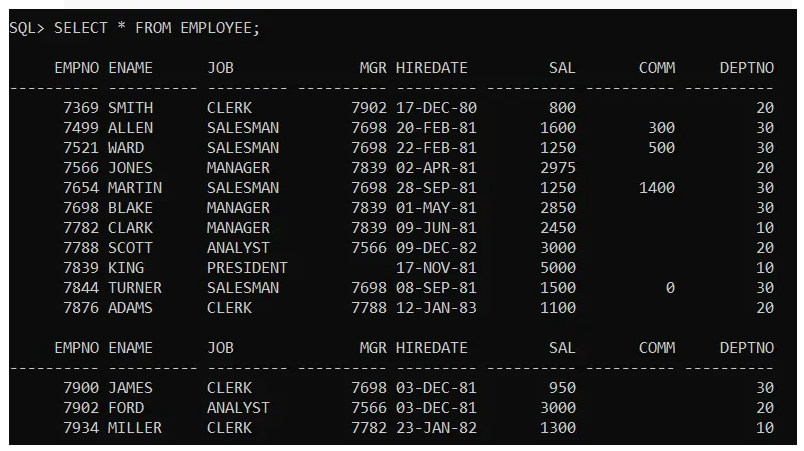
## ****Multiple Row Subquery in Oracle with Examples****

##### **What is Multiple Row Subquery in Oracle?**

When a Subquery returns more than one value is called a Multiple Row Subquery in Oracle. In this Multiple Row Subquery, we can use the operators such as IN, ANY, ALL.

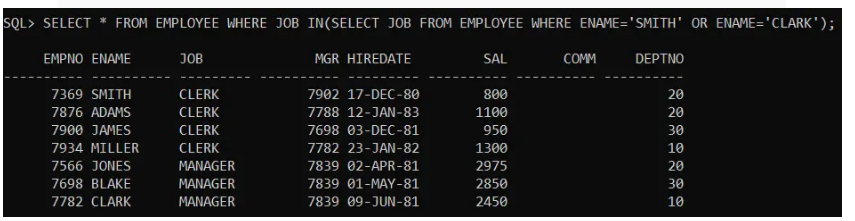
##### **Example to understand Multiple Row Subquery in Oracle**

We are going to use the following Employee table to understand Multiple Row Subquery with Examples.



###### **Example1:**

Employee details whose employee job is same as the job of the employee “SMITH”, and “CLARK”?  
Solution:  
**SELECT \* FROM EMPLOYEE WHERE JOB IN(SELECT JOB FROM EMPLOYEE WHERE ENAME=’SMITH’ OR ENAME=’CLARK’);**  
When you execute the above subquery, you will get the following output.



###### **Example2:**

Display employee details who are getting min, max salaries?  
Solution:

**SELECT** \* **FROM** **EMPLOYEE** **WHERE** **SAL** IN(

**SELECT** **MIN**(**SAL**) **FROM** **EMPLOYEE**

**UNION**

**SELECT** **MAX**(**SAL**) **FROM** **EMPLOYEE**);

When you execute the above query, you will get the following output.

