Different types of commands in SQL:

A). **DDL commands: -** To create a database objects
B). **DML commands: -** To manipulate data of a database objects
C). **DQL command: -** To retrieve the data from a database.
D). **DCL/DTL commands: -** To control the data of a database…

DDL commands:

1. The Create Table Command: - it defines each column of the table uniquely. Each column has minimum of three attributes, a name , data type and size.

Syntax:
Create table <table name> (<col1> <datatype>(<size>),<col2> <datatype><size>));

Ex:
```
   create table emp(empno number(4) primary key, ename char(10));
```

2. Modifying the structure of tables.
a)add new columns

Syntax:
Alter table <tablename> add(<new col><datatype(size),<new col>datatype(size));

Ex:
```
        alter table emp add(sal number(7,2));
```

3. Dropping a column from a table.

Syntax:
Alter table <tablename> drop column <col>;

Ex:
```
        alter table emp drop column sal;
```

4. Modifying existing columns.

Syntax:
Alter table <tablename> modify(<col><newdatatype>(<newsize>));

Ex:

```
alter table emp modify(ename varchar2(15));
```

5. Renaming the tables

Syntax:
Rename <oldtable> to <new table>;

Ex:

```
rename emp to emp1;
```

6. truncating the tables.

Syntax:
Truncate table <tablename>;

Ex:

```
trunc table emp1;
```

7. Destroying tables.

Syntax:
Drop table <tablename>;

Ex:

```
drop table emp;
```

<u>**DML commands:**</u>

8. Inserting Data into Tables: - once a table is created the most natural thing to do is load this table with data to be manipulated later.

Syntax:
insert into <tablename> (<col1>,<col2>) values(<exp>,<exp>);

9. Delete operations.

a) remove all rows

Syntax:

```
delete from <tablename>;
```

b) removal of a specified row/s

Syntax:

```
delete from <tablename> where <condition>;
```

10. Updating the contents of a table.

a) updating all rows

Syntax:

```
Update <tablename> set <col>=<exp>,<col>=<exp>;
```

b) updating seleted records.

Syntax:

```
Update <tablename> set <col>=<exp>,<col>=<exp>
    where <condition>;
```

<u>DQL Commands:</u>

12. Viewing data in the tables: - once data has been inserted into a table, the next most logical operation would be to view what has been inserted.

a) all rows and all columns

Syntax:

```
Select <col> to <col n> from tablename;

Select * from tablename;
```

13. Filtering table data: - while viewing data from a table, it is rare that all the data from table will be required each time. Hence, sql must give us a method of filtering out data that is not required data.

a) Selected columns and all rows:
Syntax:
select <col1>,<col2> from <tablename>;

b) selected rows and all columns:
Syntax:
select * from <tablename> where <condition>;

c) selected columns and selected rows
Syntax:
select <col1>,<col2> from <tablename> where<condition>;

14. Sorting data in a table.
Syntax:
Select * from <tablename> order by <col1>,<col2> <[sortorder]>;

The SQL ORDER BY

The ORDER BY keyword is used to sort the result-set in ascending or descending order.

Syntax

```
SELECT column1, column2, ...
FROM table_name
ORDER BY column1, column2, ... ASC|DESC;
```

Example

Sort the products by price:

```
SELECT * FROM Products
ORDER BY Price;
```

DESC

The ORDER BY keyword sorts the records in ascending order by default. To sort the records in descending order, use the DESC keyword.

Example

Sort the products from highest to lowest price:

```
SELECT * FROM Products
ORDER BY Price DESC;
```

Order Alphabetically

For string values the ORDER BY keyword will order alphabetically:

Example

Sort the products alphabetically by ProductName:

```
SELECT * FROM Products
ORDER BY ProductName;
```

Alphabetically DESC

To sort the table reverse alphabetically, use the DESC keyword:

Example

Sort the products by ProductName in reverse order:

```
SELECT * FROM Products
ORDER BY ProductName DESC;
```

ORDER BY Several Columns

The following SQL statement selects all customers from the "Customers" table, sorted by the "Country" and the "CustomerName" column. This means that it orders by Country, but if some rows have the same Country, it orders them by CustomerName:

Example

```
SELECT * FROM Customers
ORDER BY Country, CustomerName;
```

The SQL SELECT DISTINCT Statement

The SELECT DISTINCT statement is used to return only distinct (different) values.

Inside a table, a column often contains many duplicate values; and sometimes you only want to list the different (distinct) values.

Syntax

```
SELECT DISTINCT column1, column2, ...
FROM table_name;
```

The SQL SELECT DISTINCT Statement

The SELECT DISTINCT statement is used to return only distinct (different) values.

Example

Select all the different countries from the "Customers" table:

```
SELECT DISTINCT Country FROM Customers;
```

The SQL AND Operator

The WHERE clause can contain one or many AND operators.

The AND operator displays a record if *all* the conditions are TRUE.

Syntax

```
SELECT column1, column2, ...
FROM table_name
WHERE condition1 AND condition2 AND condition3 ...;
```

Example - With SELECT Statement

The first Oracle AND condition query involves a SELECT statement with 2 conditions.

For example:

```
SELECT *

FROM customers

WHERE state = 'Florida'

AND customer_id > 5000;
```

This Oracle AND example would return all customers that reside in the *state* of Florida and have a *customer_id* > 5000. Because the * is used in the SELECT statement, all fields from the *customers* table would appear in the result set.

he SQL OR Operator

The WHERE clause can contain one or more OR operators.

The OR operator is used to filter records based on more than one condition.

The OR operator displays a record if *any* of the conditions are TRUE.

Syntax

```
SELECT column1, column2, ...
FROM table_name
WHERE condition1 OR condition2 OR condition3 ...;
```

Example

Select all customers from Germany or Spain:

```
SELECT *
FROM Customers
WHERE Country = 'Germany' OR Country = 'Spain';
```

if you want to return all customers from Germany but also those from Spain:

The NOT Operator

The NOT operator is used in combination with other operators to give the opposite result, also called the negative result.

syntax

```
SELECT column1, column2, ...
FROM table_name
WHERE NOT condition;
```

In the select statement below we want to return all customers that are NOT from Spain:

Example

Select only the customers that are NOT from Spain:

```
SELECT * FROM Customers
WHERE NOT Country = 'Spain';
```

The SQL BETWEEN Operator

The `BETWEEN` operator selects values within a given range. The values can be numbers, text, or dates.

The `BETWEEN` operator is inclusive: begin and end values are included.

Syntax

```
SELECT column_name(s)
FROM table_name
WHERE column_name BETWEEN value1 AND value2;
```

Example

Selects all products with a price between 10 and 20:

```
SELECT * FROM Products
WHERE Price BETWEEN 10 AND 20;
```

SQL NOT BETWEEN Operator

The `NOT BETWEEN` operator is used to exclude the rows that match the values in the range. It returns all the rows except the excluded rows.
Let's look at an example.

```
-- exclude rows with amount between 300 and 500

SELECT item, amount
FROM Orders
WHERE amount NOT BETWEEN 300 AND 500;
```

SQL BETWEEN Dates

In SQL, we can also use BETWEEN to filter data between two dates.

Let's look at an example.

```
-- get the records of those teams
-- who registered between given dates

SELECT *
FROM Teams
WHERE registered BETWEEN '2021-01-01' AND '2022-11-01';
```

SQL IN Syntax

```
SELECT column1, column2, ...
FROM table
WHERE column IN (value1, value2, ...);
```

Here,

- `column1, column2, ...` are the table columns.
- `table` is the table name from where we select the data.
- `column` is where the values are compared against.
- `IN` operator specifies values that the `column` value should be compared against.
- `value1, value2, ...` are the values the `column` value is compared against.

Example: SQL IN

```
-- select rows if the country is either USA or UK

SELECT first_name, country
FROM Customers
WHERE country IN ('USA', 'UK');
```

Here, the SQL command selects rows if the `country` is either the **USA** or the **UK**.

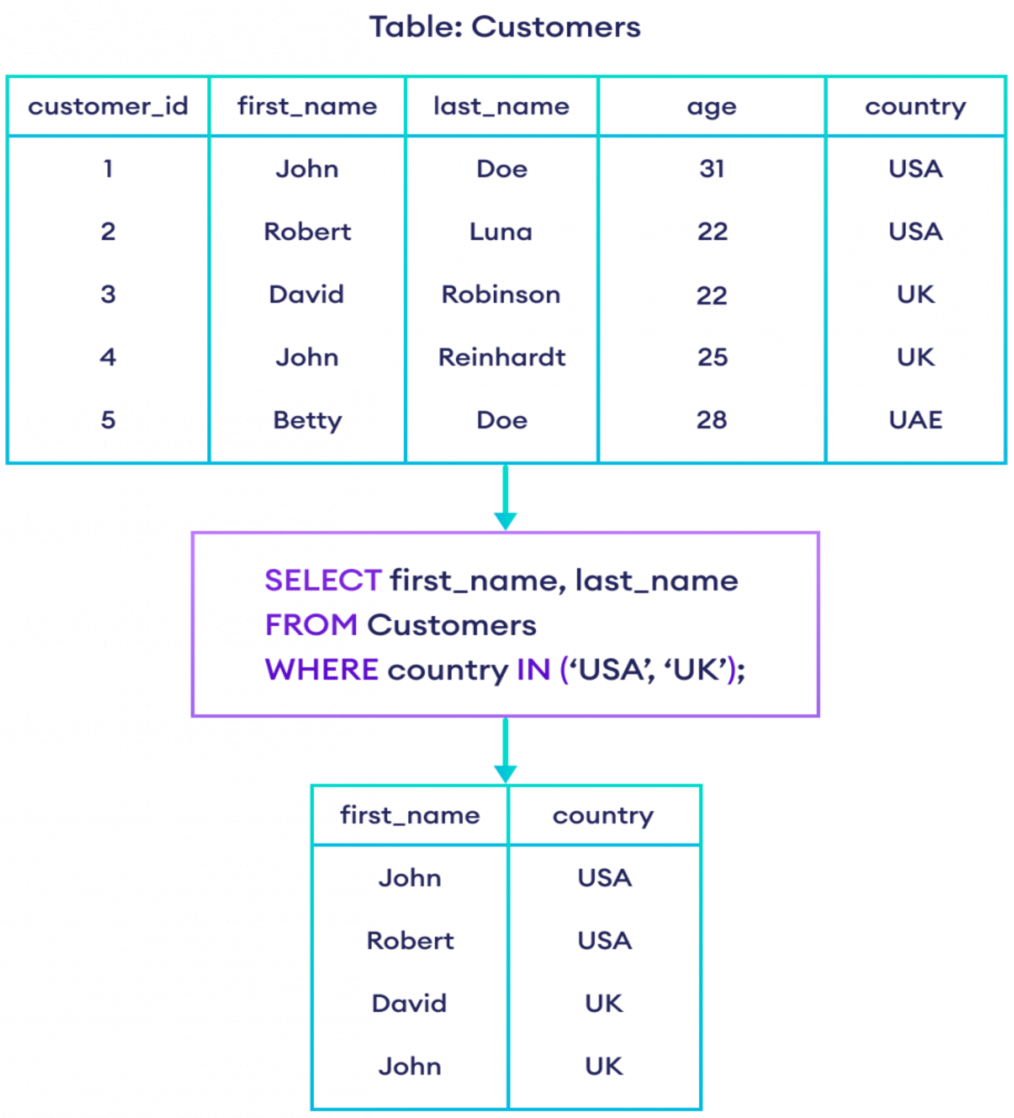
Table: Customers

customer_id	first_name	last_name	age	country
1	John	Doe	31	USA
2	Robert	Luna	22	USA
3	David	Robinson	22	UK
4	John	Reinhardt	25	UK
5	Betty	Doe	28	UAE

```
SELECT first_name, last_name
FROM Customers
WHERE country IN ('USA', 'UK');
```

first_name	country
John	USA
Robert	USA
David	UK
John	UK

Example: SQL IN Operator

SQL NOT IN Operator

The `NOT IN` operator excludes the rows that match values in the list. It returns all the rows except the excluded rows.

```
-- select rows where country is not in UK or UAE

SELECT first_name, country
FROM Customers
WHERE country NOT IN ('UK', 'UAE');
```

SQL IN and NOT IN Operators

We use the `IN` operator with the WHERE clause to match values in a list.

Example

```sql
-- select customers from the USA

SELECT first_name, country
FROM Customers
WHERE country IN ('USA');
```

Run Code

Here, the SQL command selects rows from the `Customers` table whose `country` value is `'USA'`.

SQL IN Syntax

```sql
SELECT column1, column2, ...
FROM table
WHERE column IN (value1, value2, ...);
```

Here,

- `column1, column2, ...` are the table columns.
- `table` is the table name from where we select the data.
- `column` is where the values are compared against.
- `IN` operator specifies values that the `column` value should be compared against.
- `value1, value2, ...` are the values the `column` value is compared against.

Example: SQL IN

```sql
-- select rows if the country is either USA or UK

SELECT first_name, country
FROM Customers
WHERE country IN ('USA', 'UK');
```

Run Code

Here, the SQL command selects rows if the `country` is either the **USA** or the **UK**.

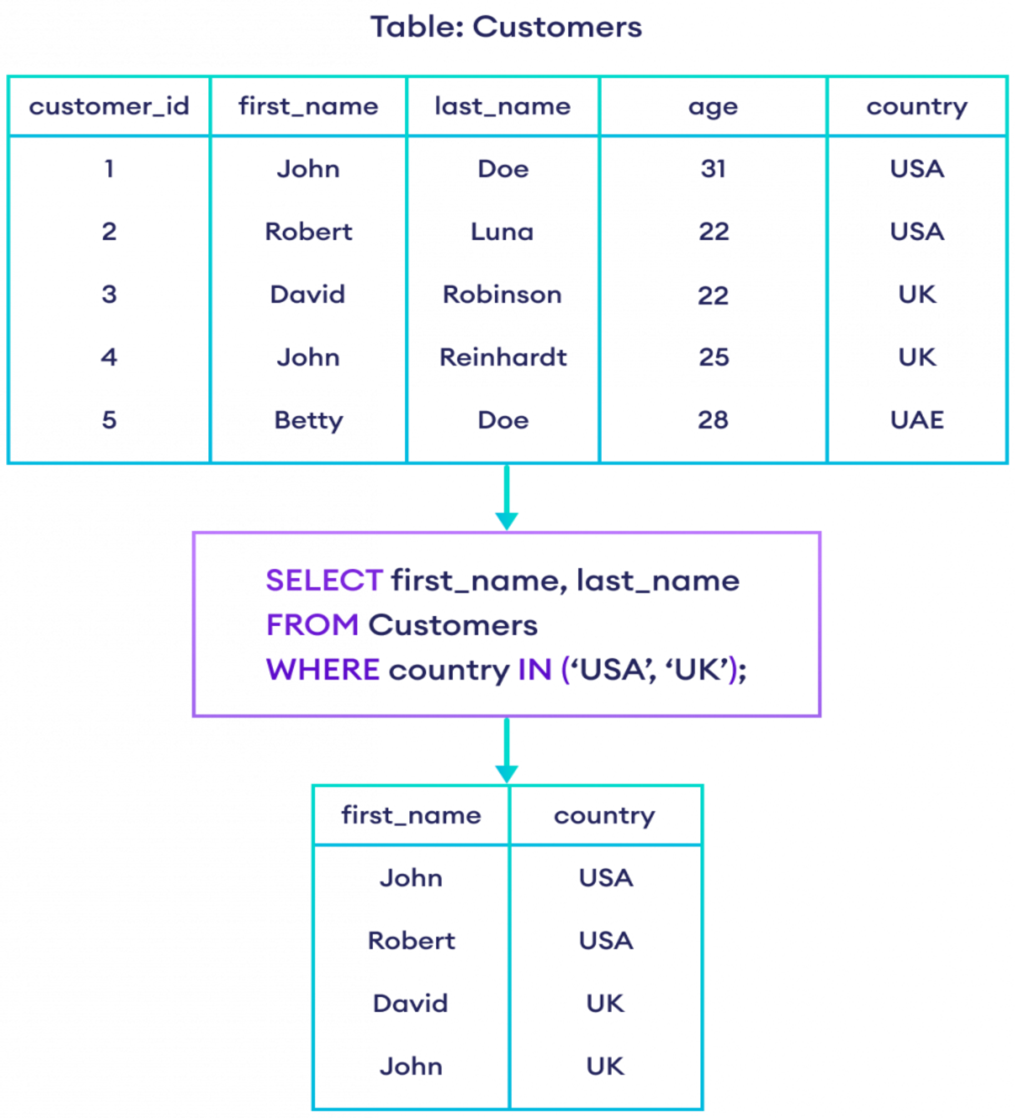
Table: Customers

customer_id	first_name	last_name	age	country
1	John	Doe	31	USA
2	Robert	Luna	22	USA
3	David	Robinson	22	UK
4	John	Reinhardt	25	UK
5	Betty	Doe	28	UAE

```
SELECT first_name, last_name
FROM Customers
WHERE country IN ('USA', 'UK');
```

first_name	country
John	USA
Robert	USA
David	UK
John	UK

Example: SQL IN Operator

Example: IN Operator to Select Rows Based on Country Value

The `IN` operator can be used to choose rows where a specific value is present in the specified field.

```
-- select rows with value 'USA' in the country column

SELECT first_name, country
FROM Customers
WHERE 'USA' IN (country);
```
Run Code

Here, the SQL command selects the rows if the `USA` value exists in the `country` field.

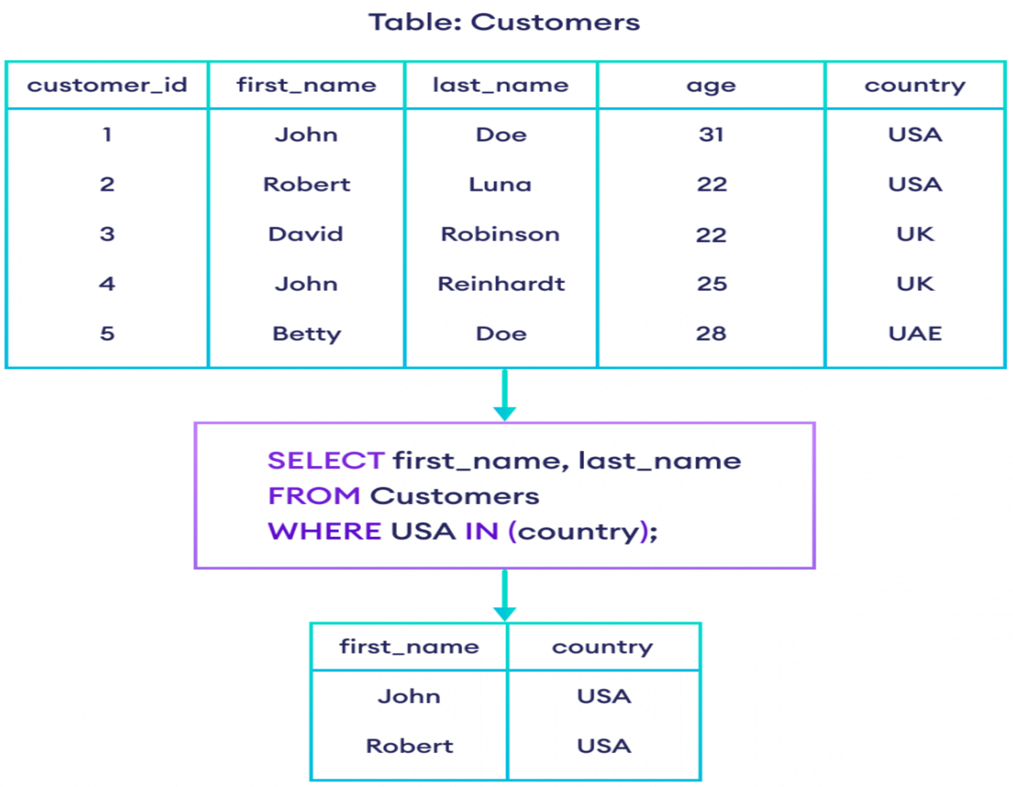

Table: Customers

customer_id	first_name	last_name	age	country
1	John	Doe	31	USA
2	Robert	Luna	22	USA
3	David	Robinson	22	UK
4	John	Reinhardt	25	UK
5	Betty	Doe	28	UAE

```
SELECT first_name, last_name
FROM Customers
WHERE USA IN (country);
```

first_name	country
John	USA
Robert	USA

Example: SQL IN Operator With Value

SQL NOT IN Operator

The `NOT IN` operator excludes the rows that match values in the list. It returns all the rows except the excluded rows.

```
-- select rows where country is not in UK or UAE

SELECT first_name, country
FROM Customers
WHERE country NOT IN ('UK', 'UAE');
```
Run Code

SQL LIKE and NOT LIKE Operators

We use the SQL `LIKE` operator with the WHERE clause to get a result set that matches the given string pattern.

Example

```
-- select customers who live in the UK

SELECT first_name
FROM Customers
WHERE country LIKE 'UK';
```

Here, the SQL command selects the first name of customers whose `country` is **UK**.

SQL LIKE Syntax

```
SELECT column1, column2, ...
FROM table
WHERE column LIKE value;
```

Here,

- `column1,column2, ...` are the columns to select the data from
- `table` is the name of the table
- `column` is the column we want to apply the filter to
- `LIKE` matches the `column` with `value`
- `value` is the pattern you want to match in the specified `column`

Example: SQL LIKE

```sql
-- select customers who live in the UK

SELECT *
FROM Customers
WHERE country LIKE 'UK';
```
Run Code

Here, the SQL command selects

custo

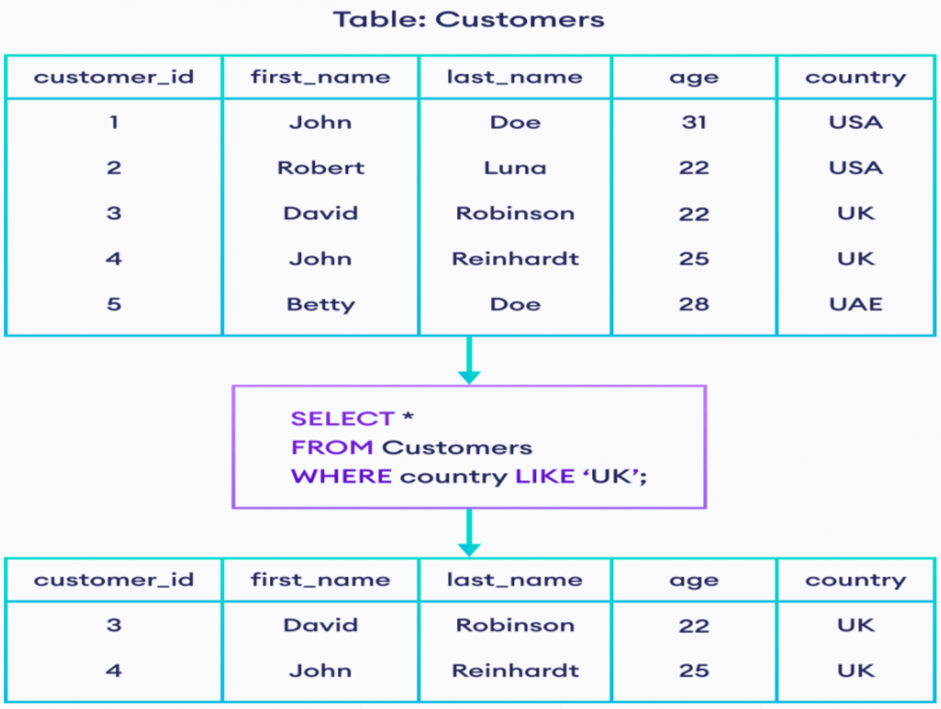
Table: Customers

customer_id	first_name	last_name	age	country
1	John	Doe	31	USA
2	Robert	Luna	22	USA
3	David	Robinson	22	UK
4	John	Reinhardt	25	UK
5	Betty	Doe	28	UAE

```sql
SELECT *
FROM Customers
WHERE country LIKE 'UK';
```

customer_id	first_name	last_name	age	country
3	David	Robinson	22	UK
4	John	Reinhardt	25	UK

Customers whose `country` is **UK**.
Example: SQL LIKE

Note: Although the `LIKE` operator behaves similarly to the `=` operator in this example, they are not the same. The `=` operator is used to check equality, whereas the `LIKE` operator is used to match string patterns only.

SQL LIKE With Wildcards

SQL LIKE With the % Wildcard

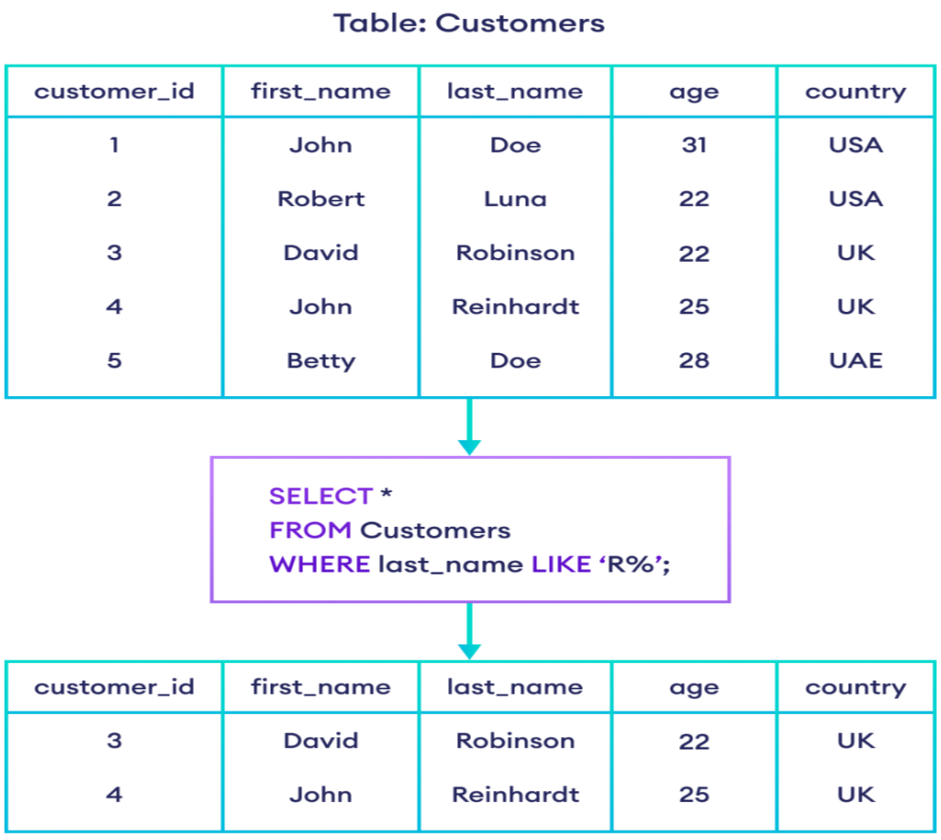
Table: Customers

customer_id	first_name	last_name	age	country
1	John	Doe	31	USA
2	Robert	Luna	22	USA
3	David	Robinson	22	UK
4	John	Reinhardt	25	UK
5	Betty	Doe	28	UAE

```
SELECT *
FROM Customers
WHERE last_name LIKE 'R%';
```

customer_id	first_name	last_name	age	country
3	David	Robinson	22	UK
4	John	Reinhardt	25	UK

SQL LIKE With the _ Wildcard

SQL NOT LIKE Operator

We can also invert the working of the `LIKE` operator by using the `NOT` operator with it. This returns a result set that doesn't match the given string pattern.

Syntax

```
SELECT column1, column2, ...
FROM table_name
WHERE column NOT LIKE value;
```

Here,

- `column1,column2, ...` are the columns to select the data from
- `table_name` is the name of the table
- `column` is the column we want to apply the filter to
- `NOT LIKE` ignores the match of the `column` with the `value`
- `value` is the pattern you don't want to match in the specified `column`

For example,

```
-- select customers who don't live in the USA

SELECT *
FROM Customers
WHERE country NOT LIKE 'USA';
```
Run Code

Here, the SQL command selects all customers except those whose `country` is **USA**.

SQL LIKE With Multiple Values

We can use the `LIKE` operator with multiple string patterns using the OR operator. For example,

```
-- select customers whose last_name starts with R and ends with t
-- or customers whose last_name ends with e

SELECT *
FROM Customers
WHERE last_name LIKE 'R%t' OR last_name LIKE '%e';
```
Run Code

Here, the SQL command selects customers whose `last_name` starts with **R** and ends with **t** or customers whose `last_name` ends with **e**.

The table given below has a few examples showing the WHERE clause having different LIKE operators with '%' and '_' −

S.No	Statement & Description
1	**WHERE SALARY LIKE '200%'** Finds any values that start with 200.
2	**WHERE SALARY LIKE '%200%'** Finds any values that have 200 in any position.
3	**WHERE SALARY LIKE '_00%'** Finds any values that have 00 in the second and third positions.
4	**WHERE SALARY LIKE '2_%_%'** Finds any values that start with 2 and are at least 3 characters in length.
5	**WHERE SALARY LIKE '%2'** Finds any values that end with 2.
6	**WHERE SALARY LIKE '_2%3'** Finds any values that have a 2 in the second position and end with a 3.
7	**WHERE SALARY LIKE '2___3'** Finds any values in a five-digit number that start with 2 and end with 3.

Learn **SQL** in-depth with real-world projects through our **SQL certification course.** Enroll and become a certified expert to boost your career.

The '%' Wildcard character

The % sign represents zero or multiple characters. The '%' wildcard matches any length of a string which even includes the zero length.

Example

To understand it better let us consider the CUSTOMERS table which contains the personal details of customers including their name, age, address and salary etc. as shown below −

Open Compiler

```
CREATE TABLE CUSTOMERS (
  ID INT NOT NULL,
  NAME VARCHAR (20) NOT NULL,
  AGE INT NOT NULL,
  ADDRESS CHAR (25),
  SALARY DECIMAL (18, 2),
  PRIMARY KEY (ID)
);
```

Now, insert values into this table using the INSERT statement as follows −

Open Compiler

```
INSERT INTO CUSTOMERS VALUES
(1, 'Ramesh', 32, 'Ahmedabad', 2000.00 ),
(2, 'Khilan', 25, 'Delhi', 1500.00 ),
(3, 'Kaushik', 23, 'Kota', 2000.00 ),
(4, 'Chaitali', 25, 'Mumbai', 6500.00 ),
(5, 'Hardik', 27, 'Bhopal', 8500.00 ),
(6, 'Komal', 22, 'Hyderabad', 4500.00 ),
(7, 'Muffy', 24, 'Indore', 10000.00 );
```

The table will be created as follows −

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000.00
2	Khilan	25	Delhi	1500.00
3	Kaushik	23	Kota	2000.00
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
6	Komal	22	Hyderabad	4500.00
7	Muffy	24	Indore	10000.00

Now, let us display all the records from the CUSTOMERS table, where the SALARY starts with 200 −

Open Compiler

```sql
SELECT * FROM CUSTOMERS WHERE SALARY LIKE '200%';
```

Output

This would produce the following result −

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000.00
3	Kaushik	23	Kota	2000.00

Example

Below is the query that displays all the records from the CUSTOMERS table previously created with the NAME that has 'al' in any position. Here, we are using multiple '%' wildcards in the LIKE condition −

Open Compiler

```sql
SELECT * FROM CUSTOMERS WHERE NAME LIKE '%al%';
```

Output

The following result is produced −

ID	NAME	AGE	ADDRESS	SALARY
4	Chaitali	25	Mumbai	6500.00
6	Komal	22	Hyderabad	4500.00

The '_' wildcard character

The **underscore** wild card represents a single number or character. A single '_' looks for exactly one character similar to the '%' wildcard.

Example

Following is the query which would display all the records from the CUSTOMERS table previously created, where the Name starts with **K** and is at least 4 characters in length −

Open Compiler

```sql
SELECT * FROM CUSTOMERS WHERE NAME LIKE 'K___%';
```

Output

The result obtained is given below −

ID	NAME	AGE	ADDRESS	SALARY
2	Khilan	25	Delhi	1500.00
3	Kaushik	23	Kota	2000.00
6	Komal	22	Hyderabad	4500.00

Example

Following is the query to display all the records from the CUSTOMERS table, where the NAME has 'm' in the third position −

Open Compiler

```sql
SELECT * FROM CUSTOMERS WHERE NAME LIKE '__m%';
```

Output

We get the following result on executing the above query −

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000.00
6	Komal	22	Hyderabad	4500.00

LIKE operator with OR

We can also use the LIKE operator with multiple string patterns for selecting rows by using the **AND** or **OR** operators.

Syntax

Following is the basic syntax of using LIKE operator with OR operator −

```sql
SELECT column1, column2, ...
FROM table_name
```

```
WHERE column1 LIKE pattern1 OR column2 LIKE pattern2 OR ...;
```

Example

Here, the SQL query retrieves the records of the customers whose name starts with **C** and ends with **i**, or customers whose name ends with **k** −

Open Compiler

```
SELECT * FROM CUSTOMERS WHERE NAME LIKE 'C%i' OR NAME LIKE '%k';
```

Output

This will produce the following result −

ID	NAME	AGE	ADDRESS	SALARY
3	Kaushik	23	Kota	2000.00
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00

NOT operator with the LIKE condition

We use the NOT operator with LIKE to extract the rows which does not contain a particular string provided in the search pattern.

Syntax

Following is the basic syntax of NOT LIKE operator in SQL −

```
SELECT column1, column2, ...
FROM table_name
WHERE column1 NOT LIKE pattern;
```

Example

In the query given below, we are fetching all the customers whose name does not start with **K** −

Open Compiler

```
SELECT * FROM CUSTOMERS WHERE NAME NOT LIKE 'K%';
```

Output

This will produce the following result −

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000.00
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
7	Muffy	24	Indore	10000.00

Escape characters with LIKE operator

The escape character in SQL is used to exclude certain wildcard characters from the expression of the **LIKE** operator. By doing so, we can use these characters in their general sense.

Using the escape character, we can also avoid using the characters that are reserved in SQL syntax to denote specific commands, such as the single quote ', % and _.

For example, if you need to search for % as a literal in the LIKE condition, then it is done using Escape character.

An escape character is only defined as a single character. It is suggested to choose the character which is not present in our data.

Syntax

The syntax for using the LIKE operator with escape characters is as follows −

```
SELECT column1, column2, ...
FROM table_name
WHERE column1 LIKE 'pattern ESCAPE escape_character';
```

Where,

- **pattern** is the pattern you want to match.
- **ESCAPE** is the keyword that indicates the escape character
- **escape_character** is the character that you want to use as the escape character.

Example

Let us create a new table EMPLOYEE using the query below −

Open Compiler

```
CREATE TABLE EMPLOYEE (
  SALARY DECIMAL (18,2) NOT NULL,
  BONUS_PERCENT VARCHAR (20)
);
```

Now, we can insert values into this empty tables using the INSERT statement as follows −

Open Compiler

```
INSERT INTO EMPLOYEE VALUES
(67000.00, '45.00'),
(54000.00, '20.34%'),
(75000.00, '51.00'),
(84000.00, '56.82%');
```

The **Employee** table consists of the salary of employees in an organization and the bonus percentage in their salary as shown below −

SALARY	BONUS_PERCENT
67000.00	45.00
54000.00	20.34%
75000.00	51.00
84000.00	56.82%

Now, we are displaying all the records from the EMPLOYEE table, where the BONUS_PERCENT contains the % literal −

Open Compiler

```
SELECT * FROM EMPLOYEE
WHERE BONUS_PERCENT LIKE'%!%%' ESCAPE '!';
```

Output

This will produce the following result −

SALARY	BONUS_PERCENT
54000.00	20.34%
84000.00	56.82%

Example

In here, we are retrieving the BONUS_PERCENT that starts with 2 and contains the % literal −

Open Compiler

```
SELECT * FROM EMPLOYEE
WHERE BONUS_PERCENT LIKE'2%!%%' ESCAPE '!';
```

Output

Following result is obtained −

SALARY	BONUS_PERCENT
54000.00	20.34%

Uses of LIKE Operator in SQL

The few uses of LIKE operators are given below −

- It helps us to extract data that matches with the required pattern.
- It helps us in performing complex regex-based queries on our data.
- It simplifies the complex queries.

Primary Keys

In Oracle, a **primary key** is a single field or combination of fields that uniquely defines a record. None of the fields that are part of the primary key can contain a null value. A table can have only one primary key.Note

- In Oracle, a primary key can not contain more than 32 columns.
- A primary key can be defined in either a CREATE TABLE statement or an ALTER TABLE statement.

Create Primary Key - Using CREATE TABLE statement

You can create a primary key in Oracle with the CREATE TABLE statement.

Syntax

The syntax to create a primary key using the CREATE TABLE statement in Oracle/PLSQL is:

```
CREATE TABLE table_name

(column1 datatype null/not null,

  column2 datatype null/not null,

  ...

CONSTRAINT constraint_name PRIMARY KEY (column1, column2, ... column_n)

);
```

Example

Let's look at an example of how to create a primary key using the CREATE TABLE statement in Oracle:

```
create table dept(

  deptno     number(2,0),
  dname      varchar2(14),
  loc        varchar2(13),
  constraint pk_dept primary key (deptno)
)
```

```
CREATE TABLE supplier
```

```
(
  supplier_id numeric(10) not null,
  supplier_name varchar2(50) not null,
  contact_name varchar2(50),
  CONSTRAINT supplier_pk PRIMARY KEY (supplier_id)
);
```

In this example, we've created a primary key on the supplier table called supplier_pk. It consists of only one field - the supplier_id field.We could also create a primary key with more than one field as in the example below:

```
CREATE TABLE supplier
( supplier_id numeric(10) not null,
  supplier_name varchar2(50) not null,
  contact_name varchar2(50),
  CONSTRAINT supplier_pk PRIMARY KEY (supplier_id, supplier_name)
);
```

Create Primary Key - Using ALTER TABLE statement

You can create a primary key in Oracle with the ALTER TABLE statement.

Syntax

The syntax to create a primary key using the ALTER TABLE statement in Oracle/PLSQL is:

```
ALTER TABLE table_name
```

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

Example

Let's look at an example of how to create a primary key using the ALTER TABLE statement in Oracle.

```
ALTER TABLE supplier

ADD CONSTRAINT supplier_pk PRIMARY KEY (supplier_id);
```

In this example, we've created a primary key on the existing supplier table called supplier_pk. It consists of the field called supplier_id.

We could also create a primary key with more than one field as in the example below:

```
ALTER TABLE supplier

ADD CONSTRAINT supplier_pk PRIMARY KEY (supplier_id, supplier_name);
```

Drop Primary Key

You can drop a primary key in Oracle using the ALTER TABLE statement.

Syntax

The syntax to drop a primary key using the ALTER TABLE statement in Oracle/PLSQL is:

```
ALTER TABLE table_name

DROP CONSTRAINT constraint_name;
```

Example

Let's look at an example of how to drop a primary key using the ALTER TABLE statement in Oracle.

```
ALTER TABLE supplier

DROP CONSTRAINT supplier_pk;
```

In this example, we're dropping a primary key on the supplier table called supplier_pk.

Disable Primary Key

You can disable a primary key in Oracle using the ALTER TABLE statement.

Syntax

The syntax to disable a primary key using the ALTER TABLE statement in Oracle/PLSQL is:

```
ALTER TABLE table_name

DISABLE CONSTRAINT constraint_name;
```

Example

Let's look at an example of how to disable a primary using the ALTER TABLE statement in Oracle.

```
ALTER TABLE supplier

DISABLE CONSTRAINT supplier_pk;
```

In this example, we're disabling a primary key on the supplier table called supplier_pk.

Enable Primary Key

You can enable a primary key in Oracle using the ALTER TABLE statement.

Syntax

The syntax to enable a primary key using the ALTER TABLE statement in Oracle/PLSQL is:

```
ALTER TABLE table_name

ENABLE CONSTRAINT constraint_name;
```

Example

Let's look at an example of how to enable a primary key using the ALTER TABLE statement in Oracle.

```
ALTER TABLE supplier

ENABLE CONSTRAINT supplier_pk;
```

In this example, we're enabling a primary key on the supplier table called supplier_pk.

What is a foreign key in Oracle?

A foreign key is a way to enforce referential integrity within your Oracle database. A foreign key means that values in one table must also appear in another table.

The referenced table is called the *parent table* while the table with the foreign key is called the *child table*. The foreign key in the child table will generally reference a primary key in the parent table.

A foreign key can be defined in either a CREATE TABLE statement or an ALTER TABLE statement.

Using a CREATE TABLE statement:Syntax

```
CREATE TABLE table_name
( column1 datatype null/not null,
  column2 datatype null/not null,
  CONSTRAINT fk_column
    FOREIGN KEY (column1, column2, ... column_n) REFERENCES parent_table
(column1, column2, ... column_n)
);
```

Example :

create table dept(
  deptno   number(2,0),
  dname   varchar2(14),
  loc   varchar2(13),
  constraint pk_dept **primary key**(deptno))

Example :

create table emp(
  empno   number(4,0),
  ename   varchar2(10),
  job   varchar2(9),
  mgr   number(4,0),
  hiredate date,
  sal   number(7,2),
  comm   number(7,2),
  deptno   number(2,0),
  constraint pk_emp primary key (empno),
constraint fk_deptno **foreign key** (deptno) references dept (deptno))

```
CREATE TABLE supplier
( supplier_id numeric(10) not null,
  supplier_name varchar2(50) not null,
  contact_name varchar2(50),
  CONSTRAINT supplier_pk PRIMARY KEY (supplier_id)
);



CREATE TABLE products
( product_id numeric(10) not null,
  supplier_id numeric(10) not null,
  CONSTRAINT fk_supplier
    FOREIGN KEY (supplier_id)
    REFERENCES supplier(supplier_id)
);
```

We could also create a foreign key with more than one field as in the example below:

```
CREATE TABLE supplier
( supplier_id numeric(10) not null,
  supplier_name varchar2(50) not null,
  contact_name varchar2(50),
  CONSTRAINT supplier_pk PRIMARY KEY (supplier_id, supplier_name)
);



CREATE TABLE products
```

```
( product_id numeric(10) not null,
  supplier_id numeric(10) not null,
  supplier_name varchar2(50) not null,
  CONSTRAINT fk_supplier_comp
    FOREIGN KEY (supplier_id, supplier_name)
    REFERENCES supplier(supplier_id, supplier_name)
);
```

Using an ALTER TABLE statement

Syntax

The syntax for creating a foreign key in an ALTER TABLE statement is:

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

Example

```
ALTER TABLE products
ADD CONSTRAINT fk_supplier
  FOREIGN KEY (supplier_id)
  REFERENCES supplier(supplier_id);
```

In this example, we've created a foreign key called *fk_supplier* that references the supplier table based on the supplier_id field.

We could also create a foreign key with more than one field as in the example below:

```
ALTER TABLE products
ADD CONSTRAINT fk_supplier
  FOREIGN KEY (supplier_id, supplier_name)
  REFERENCES supplier(supplier_id, supplier_name);
```

Oracle / PLSQL: Foreign Keys with Cascade Delete

This Oracle tutorial explains how to use **Foreign Keys with cascade delete** in Oracle with syntax and examples.

What is a foreign key with Cascade DELETE in Oracle?

A foreign key with cascade delete means that if a record in the parent table is deleted, then the corresponding records in the child table will automatically be deleted. This is called a cascade delete in Oracle.

A foreign key with a cascade delete can be defined in either a CREATE TABLE statement or an ALTER TABLE statement.

Syntax

The syntax for creating a foreign key with cascade delete using a CREATE TABLE statement in Oracle/PLSQL is:

```
CREATE TABLE table_name
( column1 datatype  column2 datatype ,
  ...CONSTRAINT fk_column FOREIGN KEY (column1, column2, ... column_n)
     REFERENCES parent_table (column1, column2, ... column_n)
     ON DELETE CASCADE
);
Example
```

Let's look at an example of how to create a foreign key with cascade delete using the CREATE TABLE statement in Oracle/PLSQL.

For example:

```
CREATE TABLE supplier
( supplier_id numeric(10) not null,
  supplier_name varchar2(50) not null,
  contact_name varchar2(50),
  CONSTRAINT supplier_pk PRIMARY KEY (supplier_id)
);
CREATE TABLE products
( product_id numeric(10) not null,
  supplier_id numeric(10) not null,
  CONSTRAINT fk_supplier
    FOREIGN KEY (supplier_id)
    REFERENCES supplier(supplier_id)
    ON DELETE CASCADE
);
```

In this example, we've created a primary key on the supplier table called *supplier_pk*. It consists of only one field - the supplier_id field. Then we've created a foreign key called *fk_supplier* on the products table that references the supplier table based on the supplier_id field.

Because of the cascade delete, when a record in the supplier table is deleted, all records in the products table will also be deleted that have the same supplier_id value.

We could also create a foreign key (with a cascade delete) with more than one field as in the example below:

```
CREATE TABLE supplier
( supplier_id numeric(10) not null,
  supplier_name varchar2(50) not null,
  contact_name varchar2(50),
  CONSTRAINT supplier_pk PRIMARY KEY (supplier_id, supplier_name)
);

CREATE TABLE products
( product_id numeric(10) not null,
  supplier_id numeric(10) not null,
  supplier_name varchar2(50) not null,
  CONSTRAINT fk_supplier_comp
    FOREIGN KEY (supplier_id, supplier_name)
    REFERENCES supplier(supplier_id, supplier_name)
    ON DELETE CASCADE
);
```

In this example, our foreign key called *fk_foreign_comp* references the supplier table based on two fields - the supplier_id and supplier_name fields.

The cascade delete on the foreign key called *fk_foreign_comp* causes all corresponding records in the products table to be cascade deleted when a record in the supplier table is deleted, based on supplier_id and supplier_name.

Using an ALTER TABLE statement

Syntax

The syntax for creating a foreign key with cascade delete in an ALTER TABLE statement in Oracle/PLSQL is:

```
ALTER TABLE table_name
```

```
ADD CONSTRAINT constraint_name
   FOREIGN KEY (column1, column2, ... column_n)
   REFERENCES parent_table (column1, column2, ... column_n)
   ON DELETE CASCADE;
```

Example

Let's look at an example of how to create a foreign key with cascade delete using the ALTER TABLE statement in Oracle/PLSQL.

For example:

```
ALTER TABLE products
ADD CONSTRAINT fk_supplier
  FOREIGN KEY (supplier_id)
  REFERENCES supplier(supplier_id)
  ON DELETE CASCADE;
```

In this example, we've created a foreign key (with a cascade delete) called *fk_supplier* that references the supplier table based on the supplier_id field.

We could also create a foreign key (with a cascade delete) with more than one field as in the example below:

```
ALTER TABLE products
ADD CONSTRAINT fk_supplier
  FOREIGN KEY (supplier_id, supplier_name)
  REFERENCES supplier(supplier_id, supplier_name)
  ON DELETE CASCADE;
```

What is a foreign key with "Set NULL on Delete" in Oracle?

A foreign key with "set null on delete" means that if a record in the parent table is deleted, then the corresponding records in the child table will have the foreign key fields set to null. The records in the child table will **not** be deleted.

A foreign key with a "set null on delete" can be defined in either a CREATE TABLE statement or an ALTER TABLE statement.

Using a CREATE TABLE statement

Syntax

The syntax for creating a foreign key using a CREATE TABLE statement is:

```
CREATE TABLE table_name
(column1 datatype null/not null,
  column2 datatype null/not null,
  ..  CONSTRAINT fk_column
     FOREIGN KEY (column1, column2, ... column_n)
     REFERENCES parent_table (column1, column2, ... column_n)
     ON DELETE SET NULL);
```

Example

```
CREATE TABLE supplier
( supplier_id numeric(10) not null,
  supplier_name varchar2(50) not null,
  contact_name varchar2(50),
  CONSTRAINT supplier_pk PRIMARY KEY (supplier_id)
);
```

```
CREATE TABLE products
( product_id numeric(10) not null,
  supplier_id numeric(10),
  CONSTRAINT fk_supplier
    FOREIGN KEY (supplier_id)
    REFERENCES supplier(supplier_id)
    ON DELETE SET NULL
);
```

In this example, we've created a primary key on the supplier table called *supplier_pk*. It consists of only one field - the supplier_id field. Then we've created a foreign key called *fk_supplier* on the products table that references the supplier table based on the supplier_id field.

Because of the set null on delete, when a record in the supplier table is deleted, all corresponding records in the products table will have the supplier_id values set to null.

We could also create a foreign key "set null on delete" with more than one field as in the example below:

```
CREATE TABLE supplier
( supplier_id numeric(10) not null,
  supplier_name varchar2(50) not null,
  contact_name varchar2(50),
  CONSTRAINT supplier_pk PRIMARY KEY (supplier_id, supplier_name)
);


CREATE TABLE products
( product_id numeric(10) not null,
  supplier_id numeric(10),
  supplier_name varchar2(50),
  CONSTRAINT fk_supplier_comp
    FOREIGN KEY (supplier_id, supplier_name)
    REFERENCES supplier(supplier_id, supplier_name)
    ON DELETE SET NULL
);
```

In this example, our foreign key called *fk_foreign_comp* references the supplier table based on two fields - the supplier_id and supplier_name fields.

The delete on the foreign key called *fk_foreign_comp* causes all corresponding records in the products table to have the supplier_id and supplier_name fields set to null when a record in the supplier table is deleted, based on supplier_id and supplier_name.

Using an ALTER TABLE statement

Syntax

The syntax for creating a foreign key in an ALTER TABLE statement is:

```
ALTER TABLE table_name
ADD CONSTRAINT constraint_name
   FOREIGN KEY (column1, column2, ... column_n)
   REFERENCES parent_table (column1, column2, ... column_n)
   ON DELETE SET NULL;
```

Example

```
ALTER TABLE products
ADD CONSTRAINT fk_supplier
```

```
  FOREIGN KEY (supplier_id)
  REFERENCES supplier(supplier_id)
  ON DELETE SET NULL;
```

In this example, we've created a foreign key "with a set null on delete" called *fk_supplier* that references the supplier table based on the supplier_id field.

We could also create a foreign key "with a set null on delete" with more than one field as in the example below:

```
ALTER TABLE products
ADD CONSTRAINT fk_supplier
  FOREIGN KEY (supplier_id, supplier_name)
  REFERENCES supplier(supplier_id, supplier_name)
  ON DELETE SET NULL;
```

Oracle / PLSQL: Drop a Foreign Key

This Oracle tutorial explains how to **drop a foreign key** in Oracle with syntax and examples.

Description

Once a foreign key has been created, you may find that you wish to drop the foreign key from the table. You can do this with the ALTER TABLE statement in Oracle.

Syntax

The syntax to drop a foreign key in Oracle/PLSQL is:

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

Example

If you had created a foreign key as follows:

```
CREATE TABLE supplier
( supplier_id numeric(10) not null,
  supplier_name varchar2(50) not null,
  contact_name varchar2(50),
  CONSTRAINT supplier_pk PRIMARY KEY (supplier_id)
);

CREATE TABLE products
( product_id numeric(10) not null,
  supplier_id numeric(10) not null,
  CONSTRAINT fk_supplier
    FOREIGN KEY (supplier_id)
    REFERENCES supplier(supplier_id)
);
```

In this example, we've created a primary key on the supplier table called *supplier_pk*. It consists of only one field - the supplier_id field. Then we've created a foreign key called *fk_supplier* on the products table that references the supplier table based on the supplier_id field.

If we then wanted to drop the foreign key called fk_supplier, we could execute the following command:

```
ALTER TABLE products

DROP CONSTRAINT fk_supplier;
```