SQL - Foreign Key

In SQL, a **Foreign Key** is a column (or combination of columns) in a table whose values match the values of a Primary Key column in another table. Using the Foreign key, we can link two tables together.

A **Foreign Key** is also known as a **Referencing key** of a table because it can reference any field defined as unique.

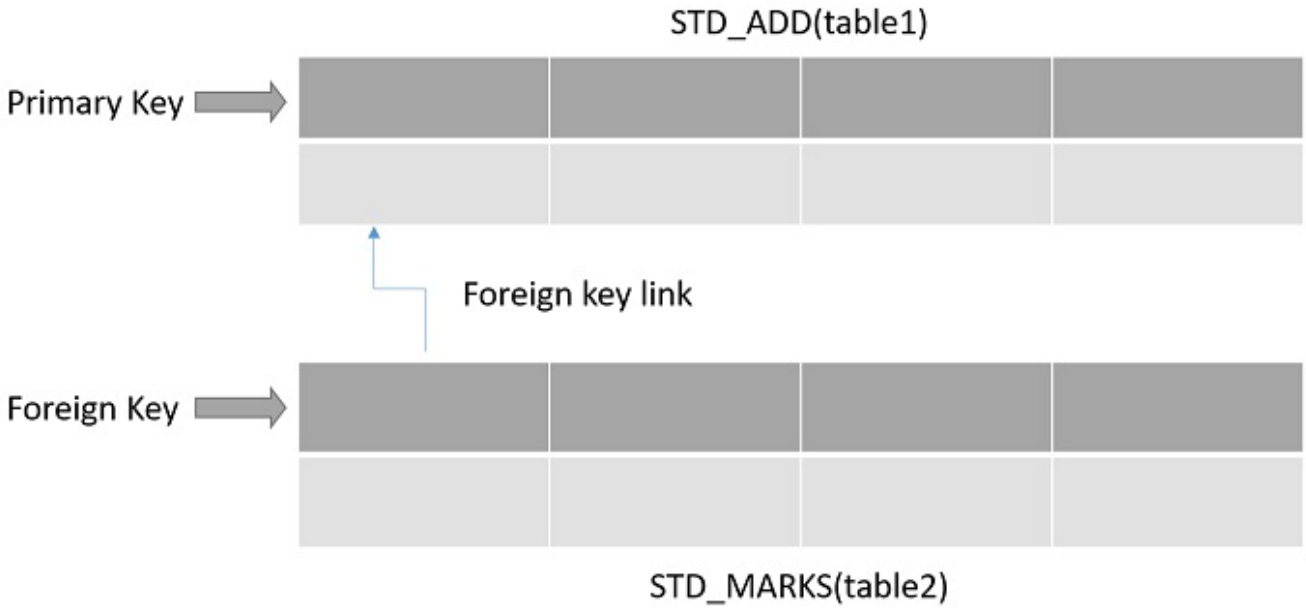
In addition to linking to tables, The **Foreign key** constraint ensures referential integrity by preventing changes to data in the primary key table from invalidating the link to data in the foreign key table. i.e, a Foreign key prevents operations, like **"dropping the table"**, that would eliminate the connection between two tables.

> A **KEY** is an attribute that allows us to uniquely identify a row in a table. In addition to avoiding redundant records, SQL keys are used to establish a relationship between multiple tables.

Let's consider an example scenario for a better understanding, suppose we have two tables namely, **STD_ADD & STD_MARKS** such that –

- **STU_ADD** contains the columns/attributes Roll_no, Name, Age, Address, and Pin.
- **STU_MARKS** contains the columns Roll_no, Subject, Marks and, Date.

Then, the column **Roll_no** of the **STU_ADD** table is the **Primary key**, whereas the column **Roll_no** of the **STD_MARKS** is a **Foreign key**. Following is the diagram to identify the Foreign key and Primary key in the above tables –



Here are some key points of the **Foreign Key** –

- A **Foreign Key** is used to **reduce the redundancy** (or duplicates) in the table.
- It helps to **normalize** (or organize the data in a database) the data in multiple tables.
- The table that has the primary key is known as the parent table and the key with the foreign key is known as the child table.

Primary key vs Foreign Key

Even though both the primary key and foreign key refer to the same column, there are many differences to be observed in the way they work. They are listed below.

Primary key	Foreign Key
The primary key is always unique.	The foreign key can be duplicated.
The primary key can not be NULL.	The Foreign can be NULL.
A table can contain only one Primary Key.	We can have more than one Foreign Key per table.

Syntax

Following is the syntax to add **Foreign key** constraints on a column of a table –

```
CREATE TABLE TABLE_2(COLUMN_NAME FOREIGN KEY REFERENCES TABLE_1(COLUMN_NAME));
```

Example

Let's create two tables with the names **CUSTOMERS** and **ORDERS**. The following query creates a table with the name **CUSTOMERS** –

```
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)
);
```

Output

Following is the output of the above SQL statement –

```
(0 rows affected)
```

Now, let us create the **ORDERS** table, While doing so, we add the foreign key constraint on column **CUSTOMER_ID** reference on column **ID** of the **CUSTOMERS** table as shown in the statement below –

```
CREATE TABLE ORDERS (
   ID INT NOT NULL,
   DATE DATETIME,
   CUSTOMER_ID INT FOREIGN KEY REFERENCES CUSTOMERS(ID),
   AMOUNT DECIMAL,
   PRIMARY KEY (ID)
);
```

Output

The above statement produces the following output –

```
(0 rows affected)
```

Verification

We have created a **Foreign Key** Constraint on a column named **CUSTOMER_ID** in the **ORDERS** table that references the column name **ID** of the **CUSTOMERS** table; so you can't drop table1 (CUSTOMERS) before dropping the table2 (ORDERS).

First of all, let's try to drop the **CUSTOMERS** table without dropping the **ORDERS** table by executing the following statement –

```
DROP TABLE CUSTOMERS;
```

If you verify the error message below, you will observe that it says that the table can not be dropped because it is referenced by a **FOREIGN KEY** constraint –

```
Could not drop object 'CUSTOMERS' because it is referenced by a FOREIGN KEY constr
```


Foreign key constraint on an existing column

We can also create a **Foreign key** constraint on a column of an existing table. This is useful when you forget to add a Foreign Key constraint on a column while creating a table, or when you want to add this constraint on another column even if one Foreign Key column exists in a table.

Syntax

Using the ALTER TABLE statement we can add a foreign key constraint on an existing column in a table as shown below –

```
ALTER TABLE TABLE1 ADD COLUMN_NAME INT FOREIGN KEY REFERENCES TABLE1(COLUMN_NAME)·
OR
```

```
ALTER TABLE TABLE2 ADD CONSTRAINT FK_ORDERS FOREIGN KEY(COLUMN_NAME) REFERENCES TA
```

Note – Here, **FK_ORDERS** is the name of the foreign key constraint. It is optional to specify the name of a constraint but it comes in handy while dropping the constraint.

Example

Assume the **CUSTOMERS** and **ORDERS** tables have already been created in the SQL database, then we will add a Foreign Key Constraint on any column in the **ORDERS** table.

Following is the SQL query to add the foreign key constraint on an existing column if the table has already been created –

```
ALTER TABLE ORDERS ADD CONSTRAINT FK_ORDERS FOREIGN KEY(ID) REFERENCES CUSTOMERS(ID
```

Output

Following is the output of the above program –

```
(0 rows affected)
```

Verification

We have created a **Foreign Key** Constraint on a column named **CUSTOMER_ID** in the **ORDERS** table that references the column name **ID** of the **CUSTOMERS** table; so you can't drop table1 (CUSTOMERS) before dropping the table2 (ORDERS).

First of all, let's try to drop the **CUSTOMERS** table without dropping the **ORDERS** table by executing the following statement –

```
DROP TABLE CUSTOMERS;
```

If you verify the error message below, you will observe that it says that the table can not be dropped because it is referenced by a **FOREIGN KEY** constraint –

```
Could not drop object 'CUSTOMERS' because it is referenced by a FOREIGN KEY constr
```

Dropping a FOREIGN KEY

Since we have created a foreign key on the column named **CUSTOMER_ID** in the **ORDERS** table, we can drop it as well whenever no longer needed it in the table.

Syntax

Using the ALTER TABLE statement, we can drop the FOREIGN key constraint from the column of the table –

```
ALTER TABLE TABLE_NAME DROP FK_NAME;
```

Where **FK_NAME** is the name of the foreign key constraint you need to drop.

Example

Following is the SQL query to drop the foreign key constraint from the column of a table –

```
ALTER TABLE ORDERS DROP FK_ORDERS;
```

Output

Following is the output of the above SQL query –

```
(0 rows affected)
```

Verification

Since, we have dropped the Foreign key constraint from the **ORDERS** table, now you can directly drop the **CUSTOMERS** table without dropping the ORDERS table.

Following is the SQL query to drop the **CUSTOMERS** table –

```
DROP TABLE CUSTOMERS;
```

If you verify the below status code thrown by the above SQL command, you observe that the CUSTOMERS table has dropped.

```
(0 rows affected)
```