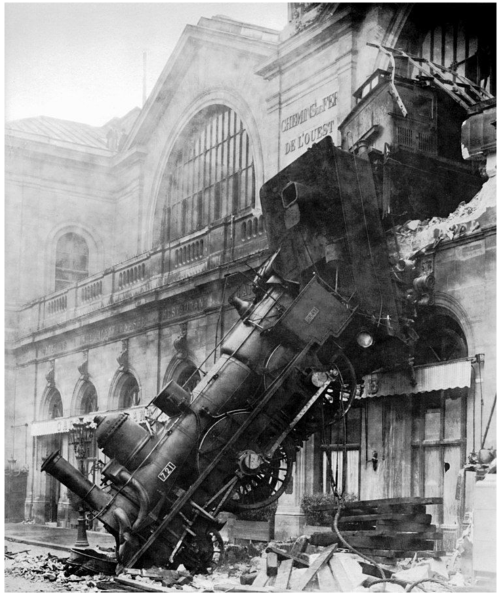
# A Crash Course in SQL

Part 2



# Data Types

NORMAL
FIRE
WATER
ELECTRIC
GRASS
ICE
FIGHTING
POISON
GROUND
FLYING
PSYCHIC
BUG
ROCK
GHOST
DRAGON
DARK
STEEL
FAIRY

## SQL Data Types

The data type of a column defines what value the column can hold: integer, character, money, date and time, binary, and so on.

Each **column** in a database table is required to have a **name** and a **data type**.

An SQL developer must decide what type of data that will be stored inside each column when creating a table. The data type is a guideline for SQL to understand what type of data is expected inside of each column, and it also identifies how SQL will interact with the stored data.

In MySQL there are three main data types:

- string,
- numeric
- date and time

## String Data Types

| | |
|---|---|
| **VARCHAR(size)** | A VARIABLE length string (can contain letters, numbers, and special characters). The *size* parameter specifies the maximum string length in characters - can be from 0 to 65535 |
| **TEXT(size)** | Holds a string with a maximum length of 65,535 bytes |
| **BLOB(size)** | For BLOBs (Binary Large Objects). Holds up to 65,535 bytes of data |

… more available at https://www.w3schools.com/sql/sql_datatypes.asp

## Numeric Data Types

| | |
|---|---|
| INT(*size*) | A medium integer. Signed range is from -2147483648 to 2147483647. Unsigned range is from 0 to 4294967295. The *size* parameter specifies the maximum display width (which is 255) |
| BOOL | Zero is considered as false, nonzero values are considered as true. |
| FLOAT(*size*, *d*) | A floating point number. The total number of digits is specified in *size*. The number of digits after the decimal point is specified in the *d* parameter. This syntax is deprecated in MySQL 8.0.17, and it will be removed in future MySQL versions |
| DOUBLE(*size*, *d*) | A normal-size floating point number. The total number of digits is specified in *size*. The number of digits after the decimal point is specified in the *d* parameter |

## Date and Time Data Types

| | |
|---|---|
| DATE | A date. Format: YYYY-MM-DD. The supported range is from '1000-01-01' to '9999-12-31' |
| DATETIME(*fsp*) | A date and time combination. Format: YYYY-MM-DD hh:mm:ss. The supported range is from '1000-01-01 00:00:00' to '9999-12-31 23:59:59'. Adding DEFAULT and ON UPDATE in the column definition to get automatic initialization and updating to the current date and time |
| TIMESTAMP(*fsp*) | A timestamp. TIMESTAMP values are stored as the number of seconds since the Unix epoch ('1970-01-01 00:00:00' UTC). Format: YYYY-MM-DD hh:mm:ss. The supported range is from '1970-01-01 00:00:01' UTC to '2038-01-09 03:14:07' UTC. Automatic initialization and updating to the current date and time can be specified using DEFAULT CURRENT_TIMESTAMP and ON UPDATE CURRENT_TIMESTAMP in the column definition |
| TIME(*fsp*) | A time. Format: hh:mm:ss. The supported range is from '-838:59:59' to '838:59:59' |
| YEAR | A year in four-digit format. Values allowed in four-digit format: 1901 to 2155, and 0000. MySQL 8.0 does not support year in two-digit format. |

## Create Table

## The SQL CREATE TABLE Statement

The `CREATE TABLE` statement is used to create a new table in a database.

```
CREATE TABLE table_name (

    column1 datatype,

    column2 datatype,

    column3 datatype,

   ....

);
```

The column parameters specify the names of the columns of the table.

The datatype parameter specifies the type of data the column can hold (e.g. varchar, integer, date, etc.).

## SQL CREATE TABLE Example

The following example creates a table called "Persons" that contains five columns: PersonID, LastName, FirstName, Address, and City:

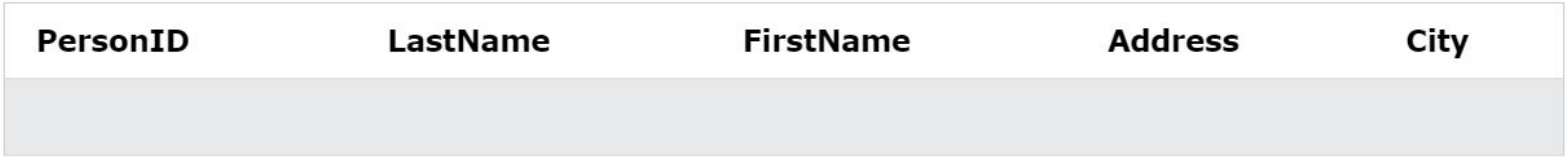
```
CREATE TABLE Persons (

     PersonID int,

     LastName varchar(255),

     FirstName varchar(255),

     Address varchar(255),

     City varchar(255)

);
```

The PersonID column is of type int and will hold an integer.

The LastName, FirstName, Address, and City columns are of type varchar and will hold characters, and the maximum length for these fields is 255 characters.

```
CREATE TABLE Persons (

     PersonID int,

     LastName varchar(255),

     FirstName varchar(255),

     Address varchar(255),

     City varchar(255)

);
```

Creates the empty table below, ready to receive some data:



| PersonID | LastName | FirstName | Address | City |
|---|---|---|---|---|
| | | | | |

## Drop Table

## The SQL DROP TABLE Statement

The `DROP TABLE` statement is used to delete an existing table in a database.

### Syntax

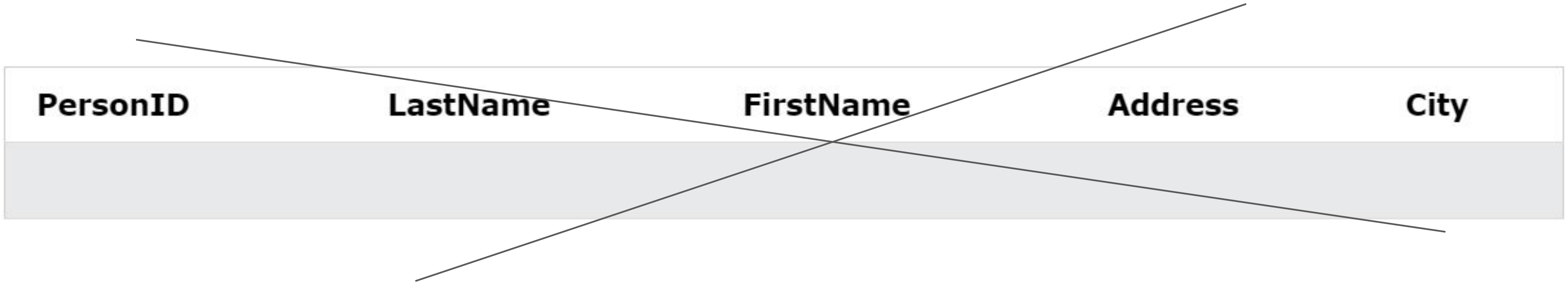
```
DROP TABLE table_name;
```

## SQL DROP TABLE Example

The following SQL statement drops the existing table "Shippers":

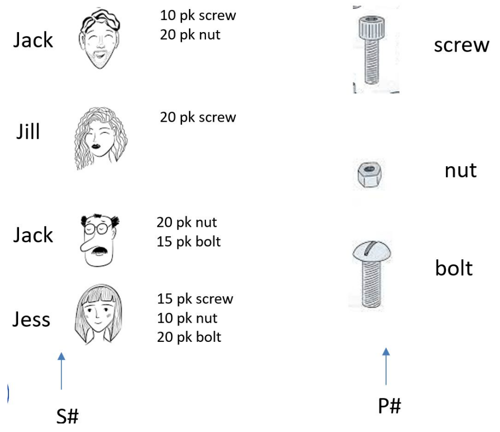
### Example

```sql
DROP TABLE Persons;
```



| PersonID | LastName | FirstName | Address | City |
|---|---|---|---|---|
| | | | | |

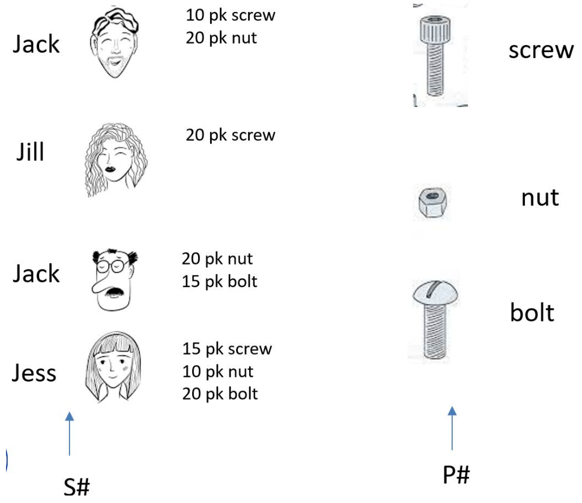
# SQL Constraints



# SQL Constraints

SQL constraints are used to specify rules for data in a table.

The constraints we apply when creating the tables, allow us to apply sensible restrictions to how data is entered into the tables.



## SQL Creation Constraints

Constraints can be specified when the table is created with the `CREATE TABLE` statement, or after the table is created with the `ALTER TABLE` statement.

### Syntax

```
CREATE TABLE table_name (

    column1 datatype constraint,

    column2 datatype constraint,

    column3 datatype constraint,

    ....

);
```

## SQL Constraints

SQL constraints are used to specify rules for the data in a table.

Constraints are used to limit the type of data that can go into a table. This ensures the accuracy and reliability of the data in the table. If there is any violation between the constraint and the data action, the action is aborted.

Constraints can be column level or table level. Column level constraints apply to a column, and table level constraints apply to the whole table.

The following constraints are commonly used in SQL:

- `NOT NULL` - Ensures that a column cannot have a NULL value
- `UNIQUE` - Ensures that all values in a column are different
- `PRIMARY KEY` - A combination of a `NOT NULL` and `UNIQUE`. Uniquely identifies each row in a table
- `FOREIGN KEY` - Prevents actions that would destroy links between tables
- `DEFAULT` - Sets a default value for a column if no value is specified

## SQL NOT NULL Constraint

By default, a column can hold NULL values.

The `NOT NULL` constraint enforces a column to NOT accept NULL values.

This enforces a field to always contain a value, which means that you cannot insert a new record, or update a record without adding a value to this field.

### Example

```
CREATE TABLE Persons (

     ID int NOT NULL,

    LastName varchar(255) NOT NULL,

     FirstName varchar(255) NOT NULL,

     Age int

);
```

## SQL UNIQUE Constraint

The `UNIQUE` constraint ensures that all values in a column are different.

Both the `UNIQUE` and `PRIMARY KEY` constraints provide a guarantee for uniqueness for a column or set of columns.

A `PRIMARY KEY` constraint automatically has a `UNIQUE` constraint.

However, you can have many `UNIQUE` constraints per table, but only one `PRIMARY KEY` constraint per table.

```
CREATE TABLE Persons (

     ID int NOT NULL,

     LastName varchar(255) NOT NULL,

     FirstName varchar(255),

     Age int,

     UNIQUE (ID, Lastname)

);
```

## SQL PRIMARY KEY Constraint

The `PRIMARY KEY` constraint uniquely identifies each record in a table.

Primary keys must contain UNIQUE values, and cannot contain NULL values.

A table can have only ONE primary key; and in the table, this primary key can consist of single or multiple columns (fields).

```
CREATE TABLE Persons (

    ID int NOT NULL,

    LastName varchar(255) NOT NULL,

    FirstName varchar(255),

    Age int,

   PRIMARY KEY (ID)

);
```

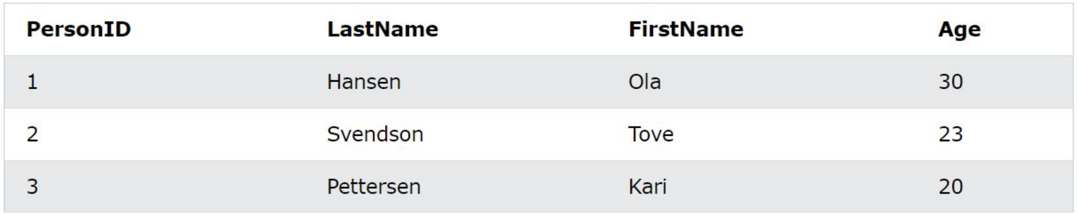
# SQL FOREIGN KEY Constraint

The `FOREIGN KEY` constraint is used to prevent actions that would destroy links between tables.

A `FOREIGN KEY` is a field (or collection of fields) in one table, that refers to the `PRIMARY KEY` in another table.

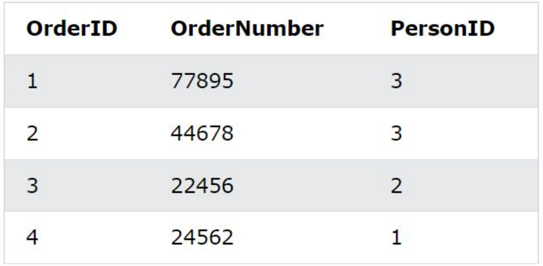
The table with the foreign key is called the child table, and the table with the primary key is called the referenced or parent table.

Persons Table



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

Orders Table



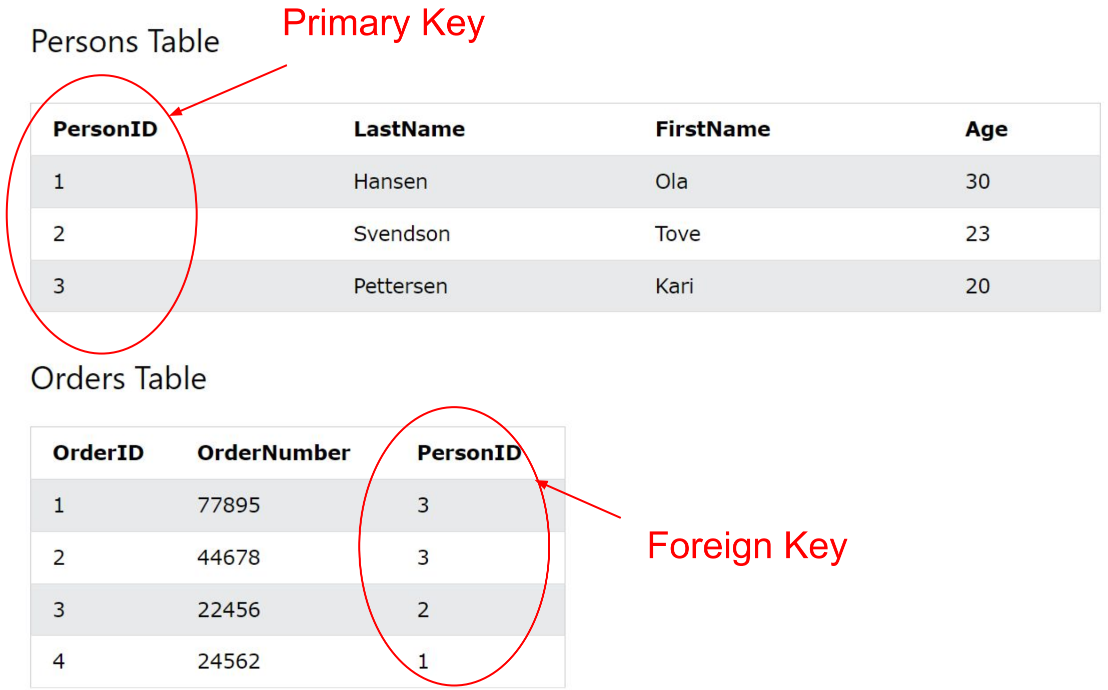
| OrderID | OrderNumber | PersonID |
|---|---|---|
| 1 | 77895 | 3 |
| 2 | 44678 | 3 |
| 3 | 22456 | 2 |
| 4 | 24562 | 1 |

# SQL FOREIGN KEY Constraint

The FOREIGN KEY constraint is used to prevent actions that would destroy links between tables.

A FOREIGN KEY is a field (or collection of fields) in one table, that refers to the PRIMARY KEY 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.


Primary Key
Foreign Key

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 |

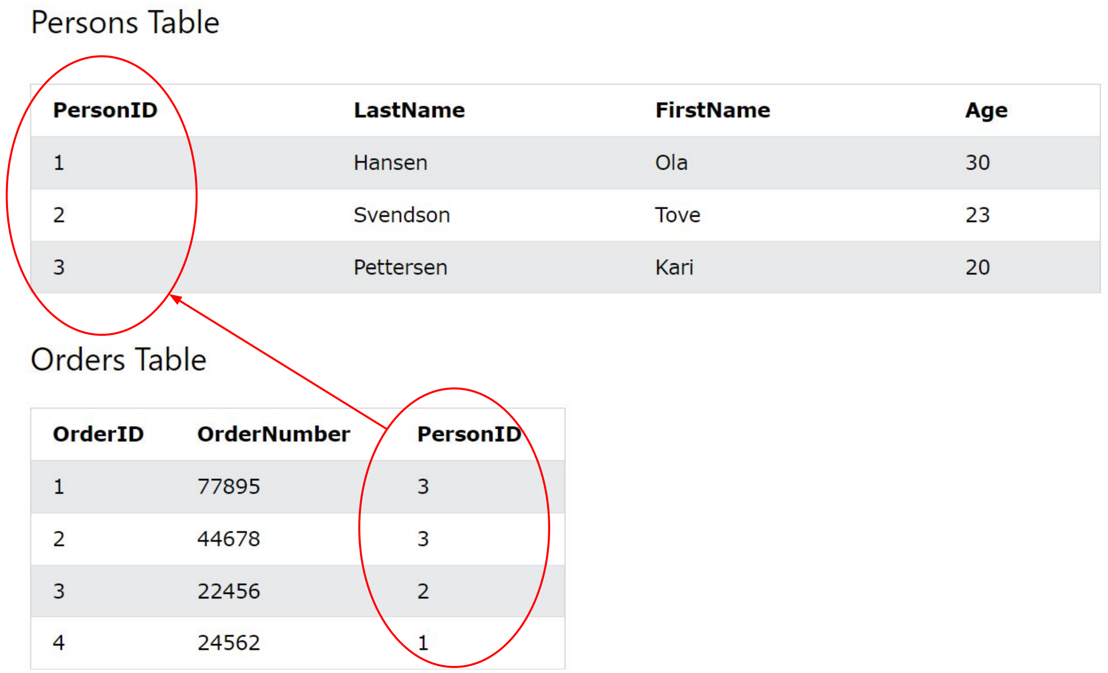
## SQL FOREIGN KEY Constraint

The `FOREIGN KEY`constraint is used to prevent actions that would destroy links between tables.

A `FOREIGN KEY`is a field (or collection of fields) in one table, that refers to the `PRIMARY KEY`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.

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.



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 |

## SQL FOREIGN KEY on CREATE TABLE

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

```
CREATE TABLE Orders (

     OrderID int NOT NULL,

     OrderNumber int NOT NULL,

     PersonID int,

    PRIMARY KEY (OrderID),

    FOREIGN KEY (PersonID) REFERENCES Persons(PersonID)

);
```

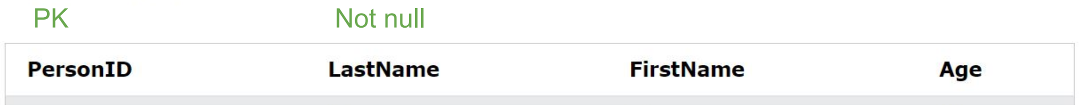
## Example Walkthrough of Creating Persons and Orders

```
CREATE TABLE Persons (

     ID int NOT NULL,

     LastName varchar(255) NOT NULL,

     FirstName varchar(255),

     Age int,

    PRIMARY KEY (ID)

);
```

## Example Walkthrough of Creating Persons and Orders

```
CREATE TABLE Persons (

     ID int NOT NULL,

     LastName varchar(255) NOT NULL

     FirstName varchar(255),

     Age int,

    PRIMARY KEY (ID)

);
```

Persons Table



| PK |  | Not null |  |  |
|---|---|---|---|---|
| PersonID | | LastName | FirstName | Age |

## Example Walkthrough of Creating Persons and Orders

```
CREATE TABLE Orders (

    OrderID int NOT NULL,

    OrderNumber int NOT NULL,

    PersonID int,

    PRIMARY KEY (OrderID),

    FOREIGN KEY (PersonID) REFERENCES Persons(PersonID)

);
```

Persons Table

PK Not null

| PersonID | LastName | FirstName | Age |
|---|---|---|---|

## Example Walkthrough of Creating Persons and Orders

```sql
CREATE TABLE Orders (

    OrderID int NOT NULL,

    OrderNumber int NOT NULL,

    PersonID int,

    PRIMARY KEY (OrderID),

    FOREIGN KEY (PersonID) REFERENCES Persons(PersonID)

);
```

Persons Table

| PK | Not null | | |
|---|---|---|---|
| **PersonID** | **LastName** | **FirstName** | **Age** |

Orders Table

| PK | Not null | FK → Persons (PersonID) |
|---|---|---|
| **OrderID** | **OrderNumber** | **PersonID** |

## Persons Table

| | | | |
|---|---|---|---|
| PK | Not null | | |
| **PersonID** | **LastName** | **FirstName** | **Age** |

## Orders Table

| | | |
|---|---|---|
| PK | Not null | FK → Persons (PersonID) |
| **OrderID** | **OrderNumber** | **PersonID** |

```sql
INSERT INTO Orders (OrderID, OrderNumber, PersonID) VALUES (1,77895,3);
```

Persons Table

| PersonID | LastName | FirstName | Age |
|---|---|---|---|
| PK | Not null | | |

Orders Table

| OrderID | OrderNumber | PersonID |
|---|---|---|
| PK | Not null | FK → Persons (PersonID) |

```
INSERT INTO Orders (OrderID, OrderNumber, PersonID) VALUES (1,77895,3);
```

Foreign Key constraint won't allow this – value 3 must exist in Persons table first!

Makes more sense to populate the Persons Table first :-)

## Example Walkthrough of Creating Persons and Orders

Start over inserting data!

Persons Table

PK Not null

| PersonID | LastName | FirstName | Age |
|---|---|---|---|

Orders Table

PK Not null FK → Persons (PersonID)

| OrderID | OrderNumber | PersonID |
|---|---|---|

```
INSERT INTO Persons (PersonID, LastName, FirstName, Age)VALUES (1,’Hansen’,’Ola’,30);
```

Persons Table

PK Not null

| PersonID | LastName | FirstName | Age |
|---|---|---|---|
| 1 | Hansen | Ola | 30 |

Orders Table

PK Not null FK → Persons (PersonID)

| OrderID | OrderNumber | PersonID |
|---|---|---|

```
INSERT INTO Persons (PersonID, LastName, FirstName, Age)VALUES (1,’Hansen’,’Ola’,30);
```

## Persons Table

PK Not null

| PersonID | LastName | FirstName | Age |
|---|---|---|---|
| 1 | Hansen | Ola | 30 |

## Orders Table

PK Not null FK → Persons (PersonID)

| OrderID | OrderNumber | PersonID |
|---|---|---|
| | | |

```
INSERT INTO Persons (PersonID, LastName, FirstName, Age)VALUES (2,'Svendson','Tove',23);
```

## Persons Table

| PK | Not null | | |
|---|---|---|---|
| **PersonID** | **LastName** | **FirstName** | **Age** |
| 1 | Hansen | Ola | 30 |
| 2 | Svendson | Tove | 23 |

## Orders Table

| PK | Not null | FK → Persons (PersonID) |
|---|---|---|
| **OrderID** | **OrderNumber** | **PersonID** |

```sql
INSERT INTO Persons (PersonID, LastName, FirstName, Age)VALUES (2,'Svendson','Tove',23);
```

Persons Table

PK Not null

| PersonID | LastName | FirstName | Age |
|---|---|---|---|
| 1 | Hansen | Ola | 30 |
| 2 | Svendson | Tove | 23 |

Orders Table

PK Not null FK → Persons (PersonID)

| OrderID | OrderNumber | PersonID |
|---|---|---|

```
INSERT INTO Persons (PersonID, LastName, FirstName, Age)VALUES (2,'Peterson','Cari',20);
```

Persons Table

PK Not null

| PersonID | LastName | FirstName | Age |
|---|---|---|---|
| 1 | Hansen | Ola | 30 |
| 2 | Svendson | Tove | 23 |

Orders Table

PK Not null FK → Persons (PersonID)

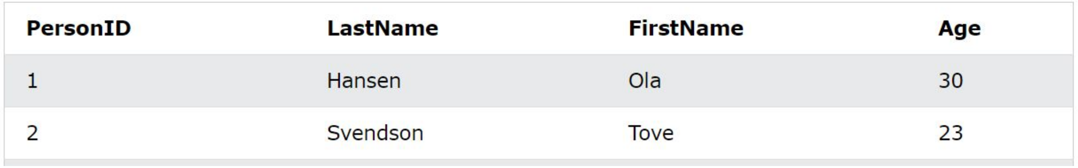
| OrderID | OrderNumber | PersonID |
|---|---|---|

```
INSERT INTO Persons (PersonID, LastName, FirstName, Age)VALUES (2,'Peterson','Cari',20);
```

Error! Constraint violation!

## Persons Table

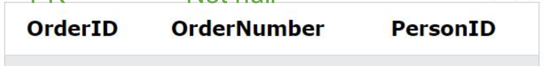
PK Not null



| PersonID | LastName | FirstName | Age |
|---|---|---|---|
| 1 | Hansen | Ola | 30 |
| 2 | Svendson | Tove | 23 |

## Orders Table

PK Not null FK → Persons (PersonID)



| OrderID | OrderNumber | PersonID |
|---|---|---|
| | | |

```
INSERT INTO Persons (PersonID, LastName, FirstName, Age)VALUES (3,’Petersen’,’Kari’,20);
```

## Persons Table

PK Not null

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

## Orders Table

PK Not null FK → Persons (PersonID)

| OrderID | OrderNumber | PersonID |
|---|---|---|
| | | |

```
INSERT INTO Persons (PersonID, LastName, FirstName, Age)VALUES (3,'Petersen','Kari',20);
```

## Persons Table

PK Not null

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

## Orders Table

PK Not null FK → Persons (PersonID)

| OrderID | OrderNumber | PersonID |
|---|---|---|

```
INSERT INTO Orders (OrderID, OrderNumber, PersonID) VALUES (1,77895,3);
```

Persons Table

PK Not null

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

Orders Table

PK Not null FK → Persons (PersonID)

| OrderID | OrderNumber | PersonID |
|---|---|---|
| 1 | 77895 | 3 |

```
INSERT INTO Orders (OrderID, OrderNumber, PersonID) VALUES (1,77895,3);
```

## Persons Table

PK Not null

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

## Orders Table

PK Not null FK → Persons (PersonID)

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

```
INSERT INTO Orders (OrderID, OrderNumber, PersonID) VALUES …;
```

## What we’ve covered today:

- Data Types
- Creating and Dropping Tables
- Table Constraints
    - UNIQUE
    - Primary Key
    - Foreign Key
    - NOT NULL