**ACTIVITY 4:** **SQL Constraints**

**Objectives:**

1. To know more about MySQL constraints and its underlying concepts
2. To create a database that enforce referential integrity between each table

**Materials:**

PC or Laptop

WAMP/XAMPP Installer

Web Browser or CLI

**Background**

**MySQL CONSTRAINT** is used to define rules to allow or restrict what values can be stored in columns. The purpose of inducing constraints is to enforce the integrity of a database. In addition, these are used to limit the type of data that can be inserted into a table. This can be classified into two types - ***column level and table level.***

The **column level** constraints can apply only to one column whereas **table level** constraints are applied to the entire table.

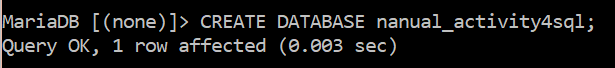
MySQL CONSTRAINT is declared at the time of creating a table. Below are as follows:

* [**PRIMARY KEY**](https://www.w3schools.com/sql/sql_primarykey.asp) - A combination of a NOT NULL and UNIQUE. Uniquely identifies each row in a table
* [**FOREIGN KEY**](https://www.w3schools.com/sql/sql_foreignkey.asp) - Uniquely identifies a row/record in another table
* [**NOT NULL**](https://www.w3schools.com/sql/sql_notnull.asp) - Ensures that a column cannot have a NULL value
* [**UNIQUE**](https://www.w3schools.com/sql/sql_unique.asp) - Ensures that all values in a column are different
* [**CHECK**](https://www.w3schools.com/sql/sql_check.asp) - Ensures that all values in a column satisfies a specific condition
* [**DEFAULT**](https://www.w3schools.com/sql/sql_default.asp) - Sets a default value for a column when no value is specified
* [**INDEX**](https://www.w3schools.com/sql/sql_create_index.asp) - Use to create and retrieve data from the database very quickly

## SQL PRIMARY KEY Constraint

The **PRIMARY KEY** constraint uniquely identifies each record in a database table. Primary keys must contain UNIQUE values and cannot contain NULL values. A table can have only one primary key, which may consist of single or multiple fields.

**My command prompt:**



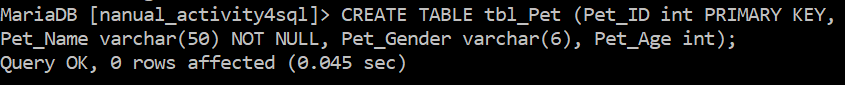
**NOTE:** I created a new database for this specific activity.

## SQL PRIMARY KEY on CREATE TABLE

The following SQL creates a PRIMARY KEY on the **"ID"** column when the **"tbl\_pet"** table is created:

CREATE TABLE tbl\_Pet (  
    pet\_id int PRIMARY KEY,  
    pet\_name varchar(50) NOT NULL,  
    pet\_gender varchar(6),  
    pet\_age int,  
);

**My command prompt:**



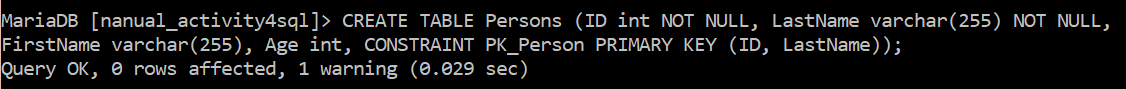
To allow naming of a PRIMARY KEY constraint, and for defining a PRIMARY KEY constraint on multiple columns, use the following SQL syntax:

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

CREATE TABLE Persons (  
    ID int NOT NULL,  
    LastName varchar(255) NOT NULL,  
    FirstName varchar(255),  
    Age int,  
    CONSTRAINT PK\_Person PRIMARY KEY (ID,LastName)  
);

**Note:** In the example above there is only ONE PRIMARY KEY (PK\_Person). However, the VALUE of the primary key is made up of TWO COLUMNS (ID + LastName).

**My command prompt:**



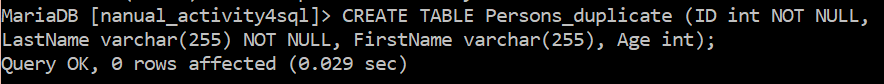
## SQL PRIMARY KEY on ALTER TABLE

To create a PRIMARY KEY constraint on the "ID" column when the table is already created, use the following SQL:

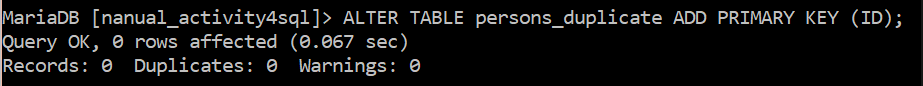
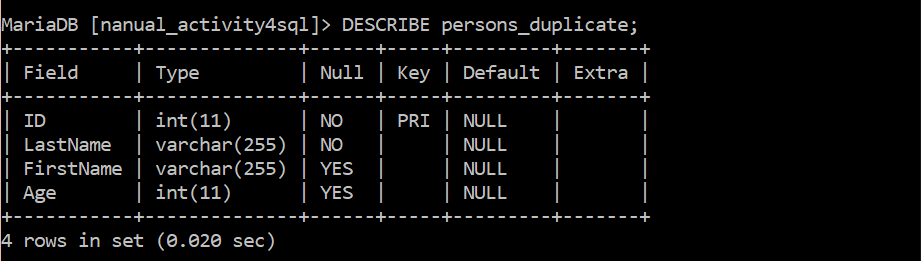
**MySQL / SQL Server / Oracle / MS Access:**

ALTER TABLE Persons  
ADD PRIMARY KEY (ID);

**My command prompt:**



**NOTE:** I created a duplicate table so I can apply the commands in this section.

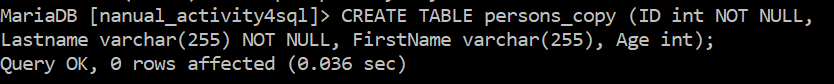
To allow naming of a PRIMARY KEY constraint, and for defining a PRIMARY KEY constraint on multiple columns, use the following SQL syntax:

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

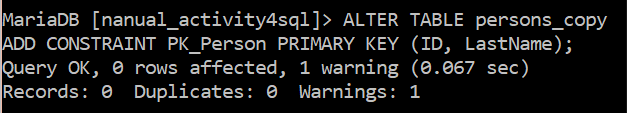
ALTER TABLE Persons  
ADD CONSTRAINT PK\_Person PRIMARY KEY (ID,LastName);

**Note:** If you use the ALTER TABLE statement to add a primary key, the primary key column(s) must already have been declared to not contain NULL values (when the table was first created).

**My command prompt:**



**NOTE:** I created another copy of the ‘Persons table’ so I can perform the command in this section.



## DROP a PRIMARY KEY Constraint

To drop a PRIMARY KEY constraint, use the following SQL:

**MySQL:**

ALTER TABLE Persons  
DROP PRIMARY KEY;

**My command prompt:**

## 

**SQL NOT NULL Constraint**

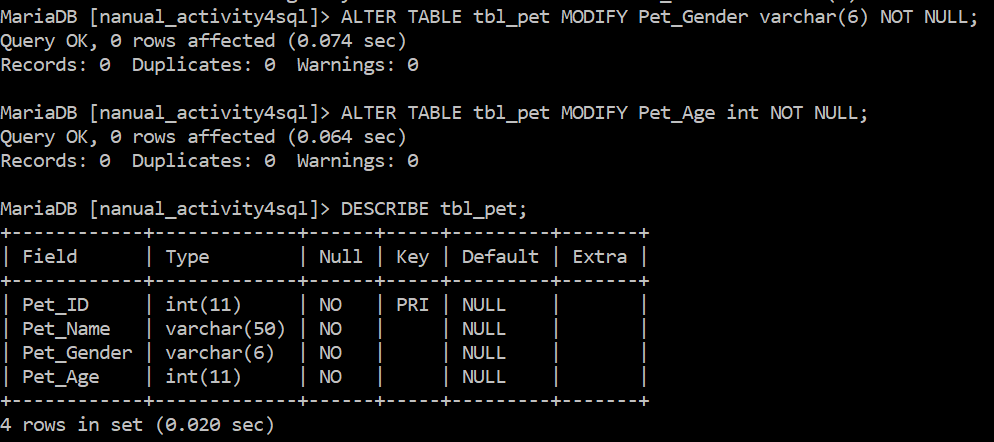
By default, a column can hold NULL values. The NOT NULL constraint enforces a column to NOT accept NULL values. This enforces a field to always contain a value, which means that you cannot insert a new record, or update a record without adding a value to this field.

The following SQL ensures that the ***"ID", "Name", and "Gender" columns*** will NOT accept NULL values:

CREATE TABLE tbl\_Pet (  
    ID int(8) NOT NULL,  
    Name varchar(255) NOT NULL,  
    Gender varchar(255) NOT NULL,  
    Age int  
);

**Tip:** If the table has already been created, you can add a NOT NULL constraint to a column with the [ALTER TABLE](https://www.w3schools.com/sql/sql_alter.asp) statement

**My command prompt:**



## 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.

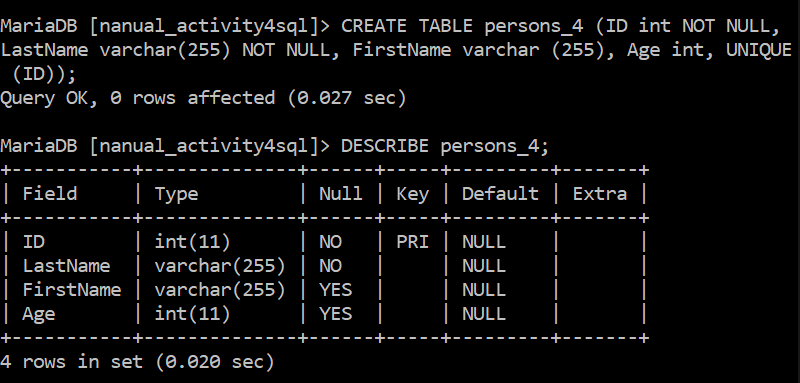
A PRIMARY KEY constraint automatically has a UNIQUE constraint. However, you can have many UNIQUE constraints per table, but only one PRIMARY KEY constraint per table.

## SQL UNIQUE Constraint on CREATE TABLE

The following SQL creates a UNIQUE constraint on the "ID" column when the "Persons" table is created:

CREATE TABLE Persons (  
    ID int NOT NULL,  
    LastName varchar(255) NOT NULL,  
    FirstName varchar(255),  
    Age int,  
    UNIQUE (ID)  
);

**My command prompt:**

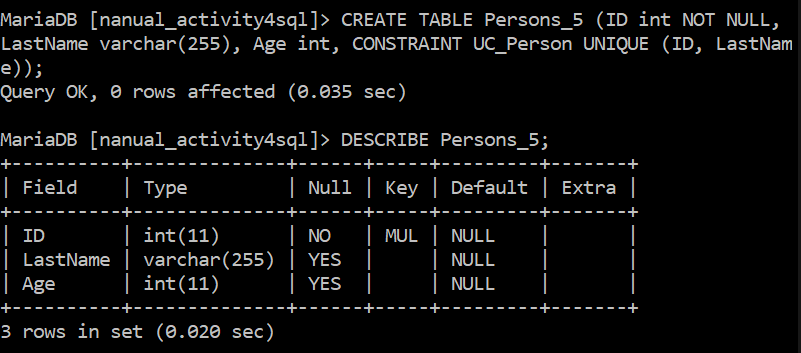


To name a UNIQUE constraint, and to define a UNIQUE constraint on multiple columns, use the following SQL syntax:

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

CREATE TABLE Persons (  
    ID int NOT NULL,  
    LastName varchar(255) NOT NULL,  
    FirstName varchar(255),  
    Age int,  
    CONSTRAINT UC\_Person UNIQUE (ID,LastName)  
);

**My command prompt:**



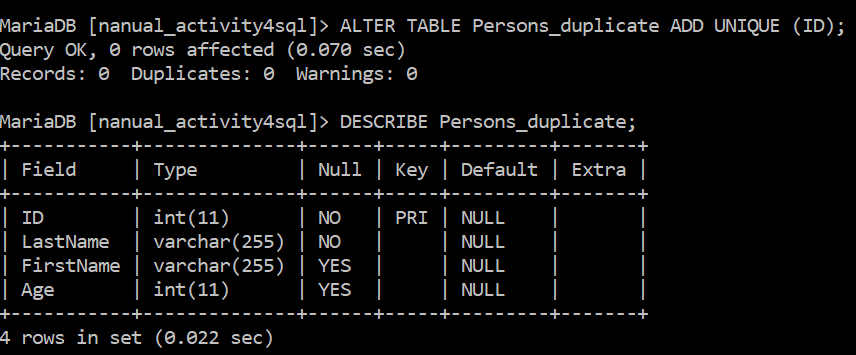
## SQL UNIQUE Constraint on ALTER TABLE

To create a UNIQUE constraint on the "ID" column when the table is already created, use the following SQL:

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

ALTER TABLE Persons  
ADD UNIQUE (ID);

**My command prompt:**

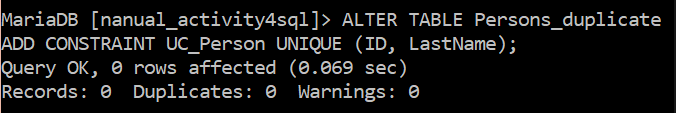


To name a UNIQUE constraint, and to define a UNIQUE constraint on multiple columns, use the following SQL syntax:

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

ALTER TABLE Persons  
ADD CONSTRAINT UC\_Person UNIQUE (ID,LastName);

**My command prompt:**



## DROP a UNIQUE Constraint

To drop a UNIQUE constraint, use the following SQL:

ALTER TABLE Persons  
DROP INDEX UC\_Person;

**My command prompt:**

## 

## SQL FOREIGN KEY Constraint

A FOREIGN KEY is a key used to link two tables together. A FOREIGN KEY in a table points to a PRIMARY KEY in another 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 is used to prevent actions that would destroy links between tables.

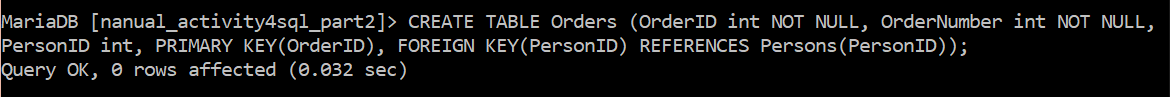
The FOREIGN KEY constraint also prevents invalid data from being inserted into the foreign key column, because it has to be one of the values contained in the table it points to.

## 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)  
);

**My command prompt:**

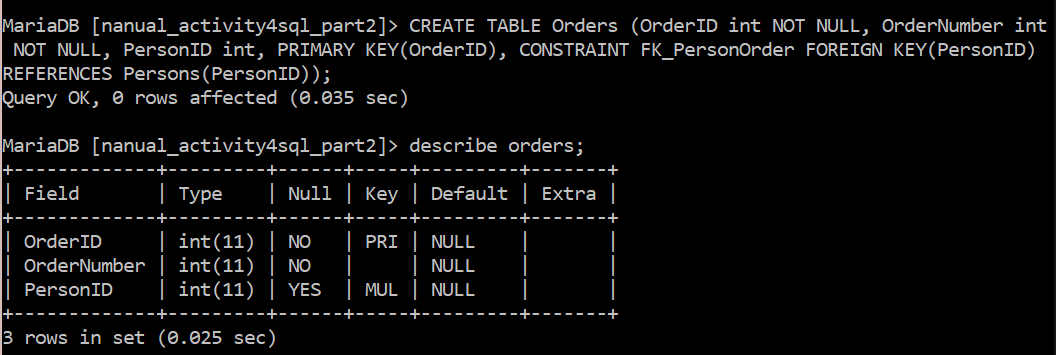


To allow naming of a FOREIGN KEY constraint, and for defining a FOREIGN KEY constraint on multiple columns, use the following SQL syntax:

**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)  
);

**My command prompt:**



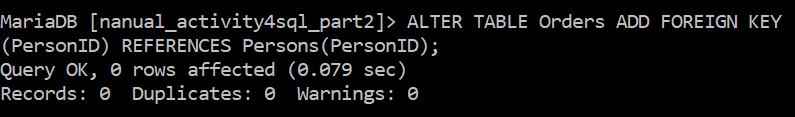
## SQL FOREIGN KEY on ALTER TABLE

To create a FOREIGN KEY constraint on the "PersonID" column when the "Orders" table is already created, use the following SQL:

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

ALTER TABLE Orders  
ADD FOREIGN KEY (PersonID) REFERENCES Persons(PersonID);

**My command prompt:**

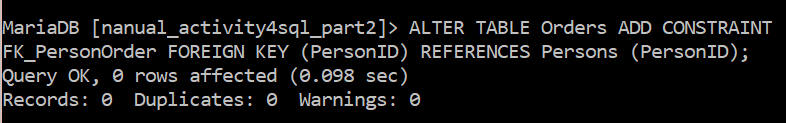


To allow naming of a FOREIGN KEY constraint, and for defining a FOREIGN KEY constraint on multiple columns, use the following SQL syntax:

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

ALTER TABLE Orders  
ADD CONSTRAINT FK\_PersonOrder  
FOREIGN KEY (PersonID) REFERENCES Persons(PersonID);

**My command prompt:**



## DROP a FOREIGN KEY Constraint

To drop a FOREIGN KEY constraint, use the following SQL:

ALTER TABLE Orders  
DROP FOREIGN KEY FK\_PersonOrder;

**My command prompt:**

## 

## SQL CHECK Constraint

The CHECK constraint is used to limit the value range that can be placed in a column.

If you define a CHECK constraint on a single column it allows only certain values for this column.

If you define a CHECK constraint on a table it can limit the values in certain columns based on values in other columns in the row.

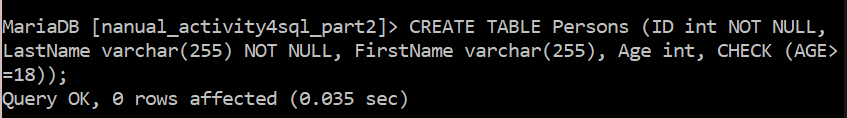
## SQL CHECK on CREATE TABLE

The following SQL creates a CHECK constraint on the "Age" column when the "Persons" table is created. The CHECK constraint ensures that you can not have any person below 18 years:

**MySQL:**

CREATE TABLE Persons (  
    ID int NOT NULL,  
    LastName varchar(255) NOT NULL,  
    FirstName varchar(255),  
    Age int,  
    CHECK (Age>=18)  
);

**My command prompt:**

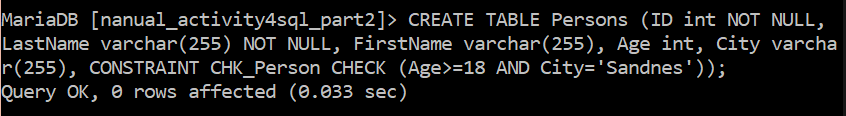


To allow naming of a CHECK constraint, and for defining a CHECK constraint on multiple columns, use the following SQL syntax:

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

CREATE TABLE Persons (  
    ID int NOT NULL,  
    LastName varchar(255) NOT NULL,  
    FirstName varchar(255),  
    Age int,  
    City varchar(255),  
    CONSTRAINT CHK\_Person CHECK (Age>=18 AND City='Sandnes')  
);

**My command prompt:**



## SQL CHECK on ALTER TABLE

To create a CHECK constraint on the "Age" column when the table is already created, use the following SQL:

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

ALTER TABLE Persons  
ADD CHECK (Age>=18);

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

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

ALTER TABLE Persons  
ADD CONSTRAINT CHK\_PersonAge CHECK (Age>=18 AND City='Sandnes');

## DROP a CHECK Constraint

To drop a CHECK constraint, use the following SQL:

**MySQL:**

ALTER TABLE Persons  
DROP CHECK CHK\_PersonAge;

## My command prompt:

## 

## SQL DEFAULT Constraint

The DEFAULT constraint is used to provide a default value for a column.

The default value will be added to all new records IF no other value is specified.

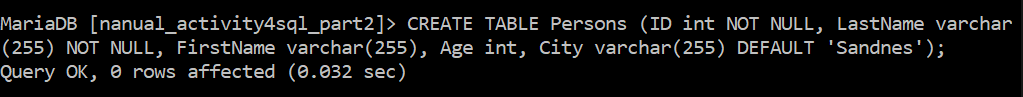
## SQL DEFAULT on CREATE TABLE

The following SQL sets a DEFAULT value for the "City" column when the "Persons" table is created:

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

CREATE TABLE Persons (  
    ID int NOT NULL,  
    LastName varchar(255) NOT NULL,  
    FirstName varchar(255),  
    Age int,  
    City varchar(255) DEFAULT 'Sandnes'  
);

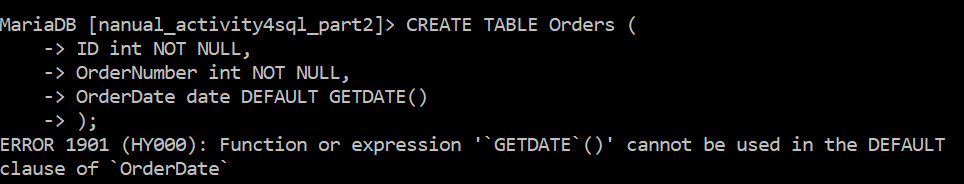
**My command prompt:**



The DEFAULT constraint can also be used to insert system values, by using functions like GETDATE():

CREATE TABLE Orders (  
    ID int NOT NULL,  
    OrderNumber int NOT NULL,  
    OrderDate date DEFAULT GETDATE()  
);

**My command prompt:**



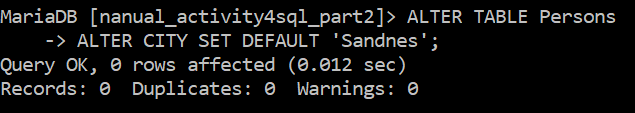
## SQL DEFAULT on ALTER TABLE

To create a DEFAULT constraint on the "City" column when the table is already created, use the following SQL:

**MySQL:**

ALTER TABLE Persons  
ALTER City SET DEFAULT 'Sandnes';

**My command prompt:**



## DROP a DEFAULT Constraint

To drop a DEFAULT constraint, use the following SQL:

**MySQL:**

ALTER TABLE Persons  
ALTER City DROP DEFAULT;

**My command prompt:**

## 

## SQL CREATE INDEX Statement

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

Indexes are used to retrieve data from the database very fast. The users cannot see the indexes, they are just used to speed up searches/queries.

**Note:** Updating a table with indexes takes more time than updating a table without (because the indexes also need an update). So, only create indexes on columns that will be frequently searched against.

### 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, ...);

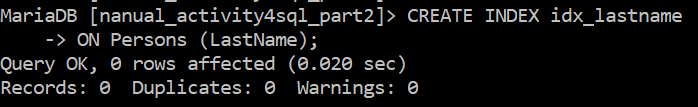
**Note:** The syntax for creating indexes varies among different databases. Therefore: Check the syntax for creating indexes in your database.

## 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);

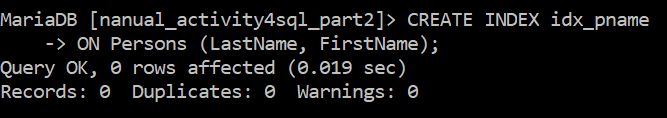
**My command prompt:**



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);

**My command prompt:**



## DROP INDEX Statement

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

**MySQL:**

ALTER TABLE table\_nameDROP INDEX index\_name;

## My command prompt:

## 

## AUTO INCREMENT Field

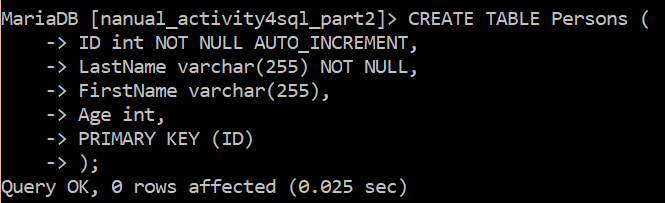
Auto-increment allows a unique number to be generated automatically when a new record is inserted into a table.

Often this is the primary key field that we would like to be created automatically every time a new record is inserted.

The following SQL statement defines the "ID" column to be an auto-increment primary key field in the "Persons" table:

CREATE TABLE Persons (  
    ID int NOT NULL AUTO\_INCREMENT,  
    LastName varchar(255) NOT NULL,  
    FirstName varchar(255),  
    Age int,  
    PRIMARY KEY (ID)  
);

**My command prompt:**



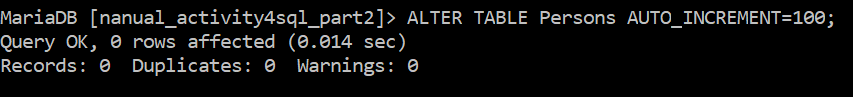
MySQL uses the AUTO\_INCREMENT keyword to perform an auto-increment feature.

By default, the starting value for AUTO\_INCREMENT is 1, and it will increment by 1 for each new record.

To let the AUTO\_INCREMENT sequence start with another value, use the following SQL statement:

ALTER TABLE Persons AUTO\_INCREMENT=100;

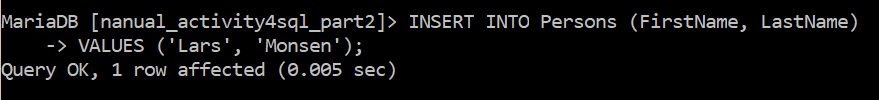
**My command prompt:**



To insert a new record into the "Persons" table, we will NOT have to specify a value for the "ID" column (a unique value will be added automatically):

INSERT INTO Persons (FirstName,LastName)  
VALUES ('Lars','Monsen');

**My command prompt:**



The SQL statement above would insert a new record into the "Persons" table. The "ID" column would be assigned a unique value. The "FirstName" column would be set to "Lars" and the "LastName" column would be set to "Monsen".

## SQL CREATE VIEW Statement

In SQL, a view is a virtual table based on the result-set of an SQL statement.

A view contains rows and columns, just like a real table. The fields in a view are fields from one or more real tables in the database.

You can add SQL functions, WHERE, and JOIN statements to a view and present the data as if the data were coming from one single table.

### CREATE VIEW Syntax

CREATE VIEW view\_name AS  
SELECT column1, column2, ...  
FROM table\_name  
WHERE condition;

**Note:** A view always shows up-to-date data! The database engine recreates the data, using the view's SQL statement, every time a user queries a view.

## SQL CREATE VIEW Examples

If you have the Northwind database you can see that it has several views installed by default.

The view "Current Product List" lists all active products (products that are not discontinued) from the "Products" table. The view is created with the following SQL:

**NOTE:** I can’t recreate the following commands since I don’t have access to the mentioned Northwind database.

CREATE VIEW [Current Product List] AS  
SELECT ProductID, ProductName  
FROM Products  
WHERE Discontinued = No;

Then, we can query the view as follows:

SELECT \* FROM [Current Product List];

Another view in the Northwind sample database selects every product in the "Products" table with a unit price higher than the average unit price:

CREATE VIEW [Products Above Average Price] AS  
SELECT ProductName, UnitPrice  
FROM Products  
WHERE UnitPrice > (SELECT AVG(UnitPrice) FROM Products);

We can query the view above as follows:

SELECT \* FROM [Products Above Average Price];

Another view in the Northwind database calculates the total sale for each category in 1997. Note that this view selects its data from another view called "Product Sales for 1997":

CREATE VIEW [Category Sales For 1997] AS  
SELECT DISTINCT CategoryName, Sum(ProductSales) AS CategorySales  
FROM [Product Sales for 1997]  
GROUP BY CategoryName;

We can query the view above as follows:

SELECT \* FROM [Category Sales For 1997];

We can also add a condition to the query. Let's see the total sale only for the category "Beverages":

SELECT \* FROM [Category Sales For 1997]  
WHERE CategoryName = 'Beverages';

## SQL Updating a View

You can update a view by using the following syntax:

### SQL CREATE OR REPLACE VIEW Syntax

CREATE OR REPLACE VIEW view\_name AS  
SELECT column1, column2, ...  
FROM table\_name  
WHERE condition;

Now we want to add the "Category" column to the "Current Product List" view. We will update the view with the following SQL:

CREATE OR REPLACE VIEW [Current Product List] AS  
SELECT ProductID, ProductName, Category  
FROM Products  
WHERE Discontinued = No;

## SQL Dropping a View

You can delete a view with the DROP VIEW command.

### SQL DROP VIEW Syntax

DROP VIEW view\_name;

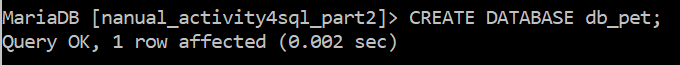
**Instructions:**

Based on the ERD given, convert it into Physical database by creating a table for each entity:

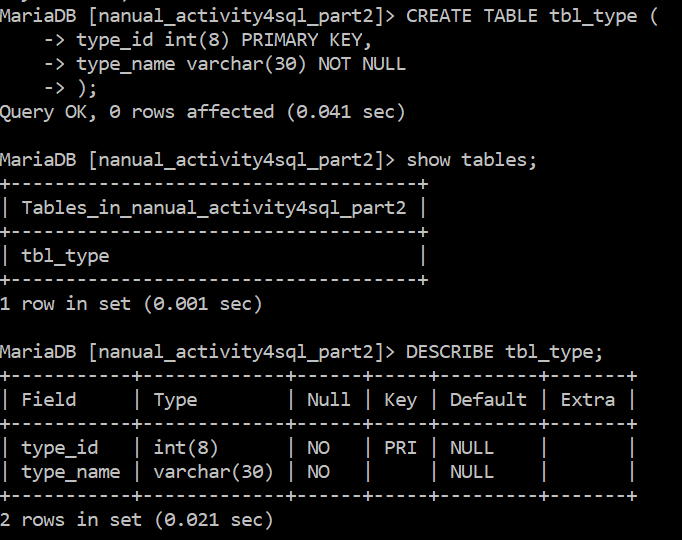
Graphical user interface, diagram

Description automatically generated

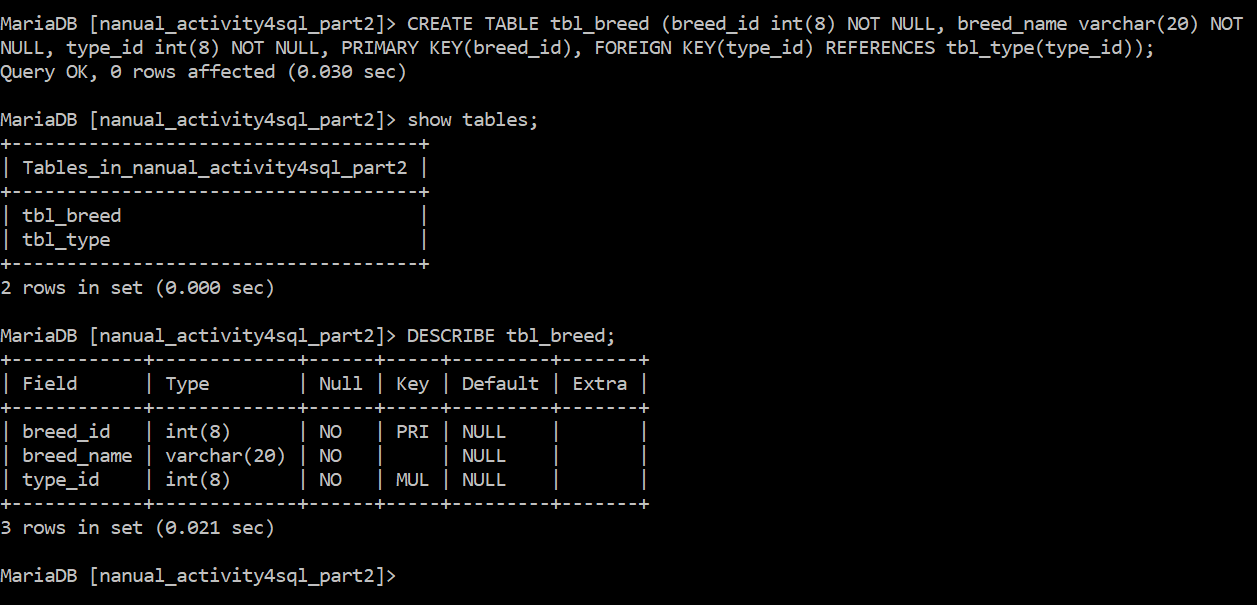
1. **Creating the database entitled ‘db\_pet’.**



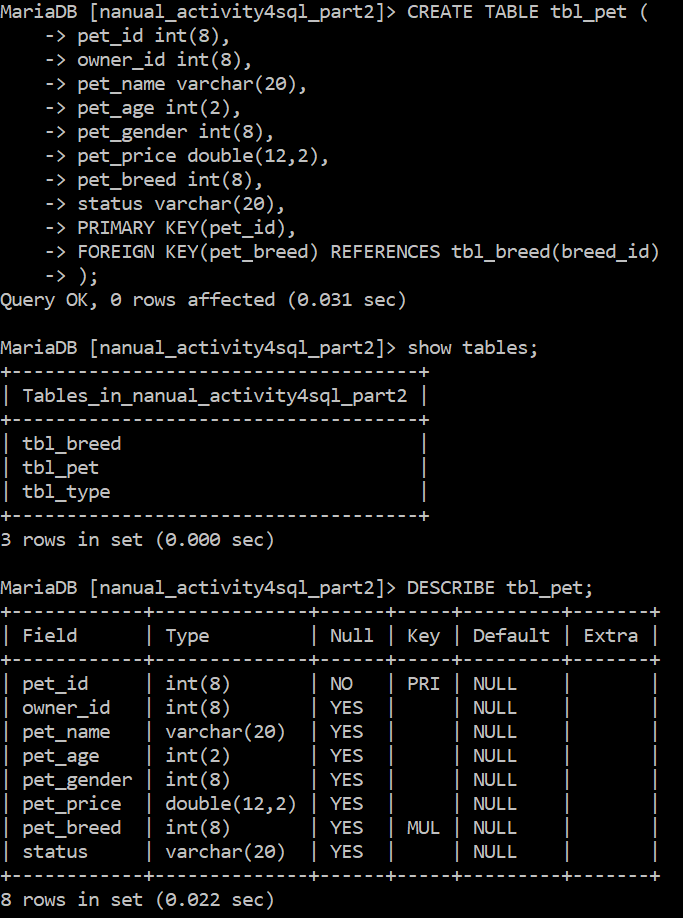
1. **Creating the ‘tbl\_type’ table and defining its ‘type\_id’ primary key.**



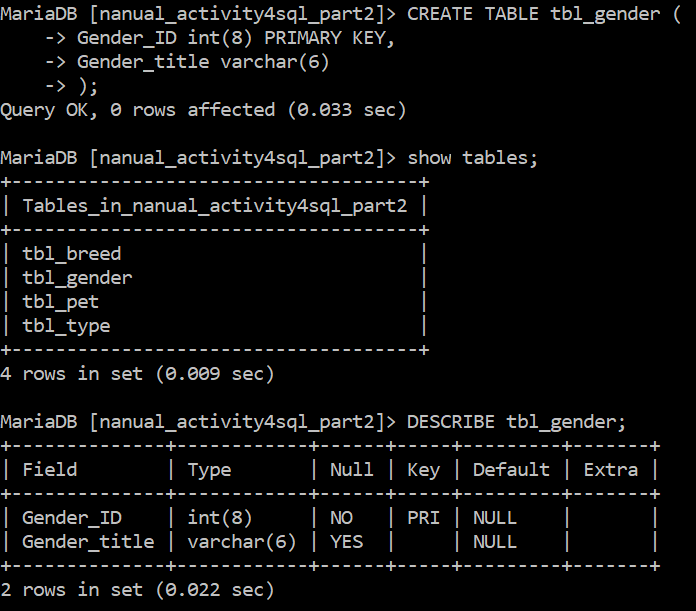
1. **Creating the ‘tbl\_breed’ table and defining its ‘breed\_id’ primary key and ‘type\_id’ foreign key (in reference to the ‘tbl\_type’ table).**



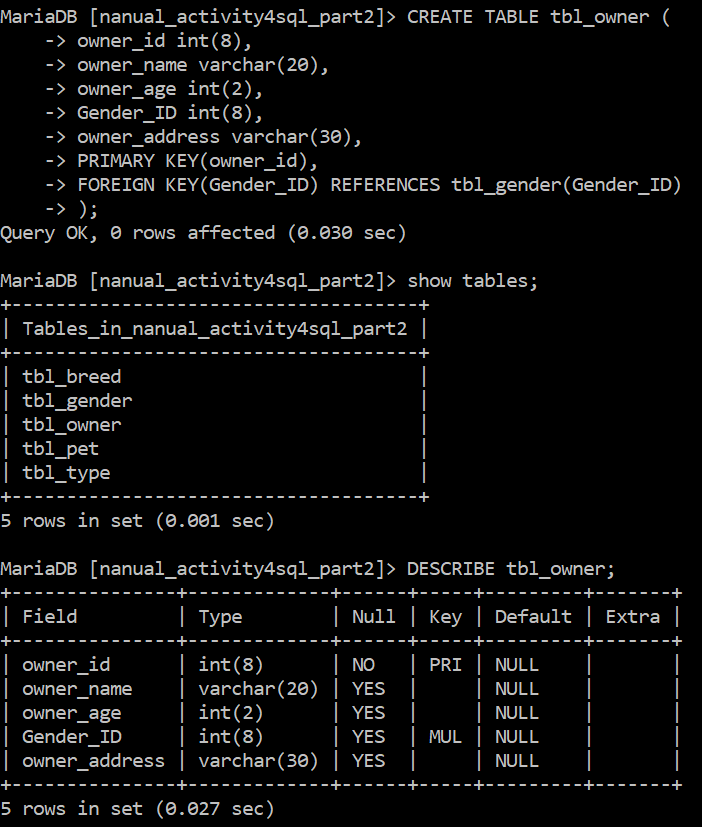
1. **Creating the ‘tbl\_pet’ table and defining its ‘pet\_id’ primary key and ‘pet\_breed’ foreign key (from ‘tbl\_breed’ table).**



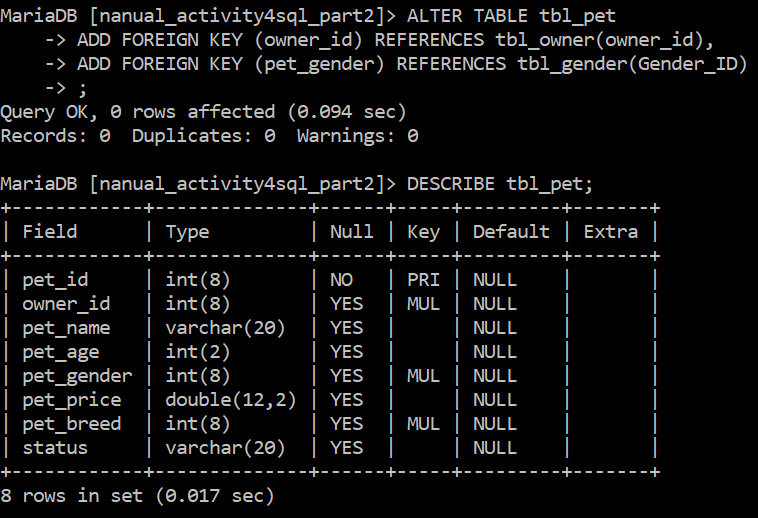
1. **Creating the ‘tbl\_gender’ table and defining its ‘Gender\_ID’ primary key.**



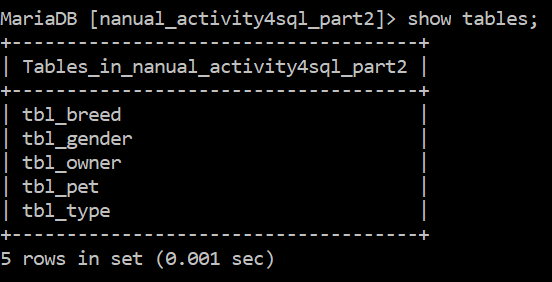
1. **Creating the ‘tbl\_owner’ table and defining its ‘owner\_id’ primary key and ‘Gender\_ID’ foreign key (from ‘tbl\_gender’ table).**



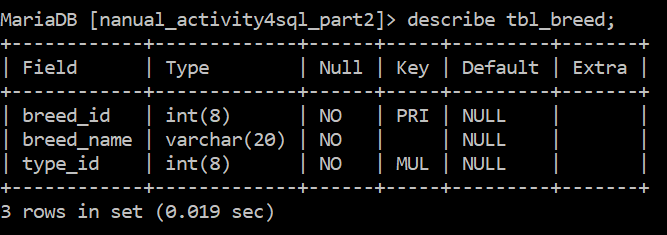
1. **Adding the ‘pet\_gender’ foreign key (from ‘tbl\_gender’ table) and ‘owner\_id’ foreign key (from ‘tbl\_owner’ table) to the ‘tbl\_pet’ table.**



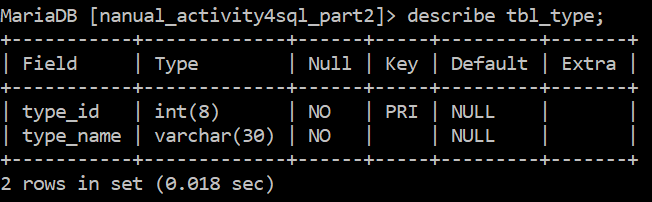
**[RECAP] Here are the tables in the database:**



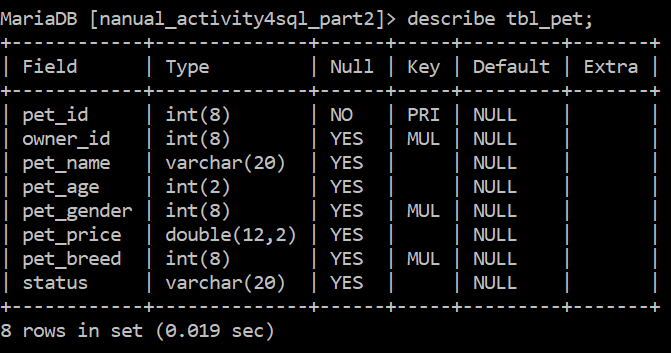
**tbl\_breed:**



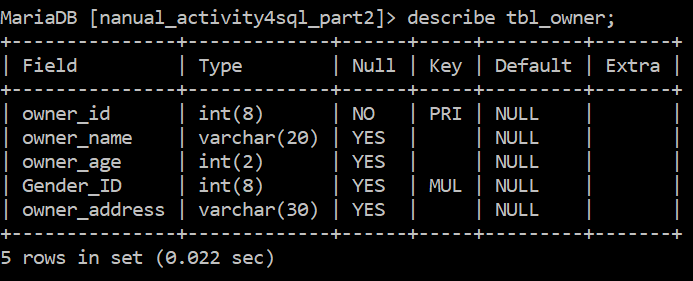
**tbl\_type:**



**tbl\_pet:**



**tbl\_owner:**



**tbl\_gender:**

