MYSQL CONSTRIANTS :

What Are Constraints in SQL?

Constraints in SQL are rules or conditions that are enforced on data in tables. They ensure the accuracy, validity, and consistency of the data. Constraints can be applied to one or more columns of a table during table creation or modification.

Types of Constraints in SQL

Here are the most common types of constraints, along with their explanations and examples:

1. PRIMARY KEY

- Ensures that each row in the table has a unique and non-null value for the specified column(s).
- A table can have only one primary key, and it uniquely identifies each record.

Example:

```
sql
Copy code
CREATE TABLE students (
    student_id INT PRIMARY KEY,
    name VARCHAR(50) NOT NULL
);
```

---

2. FOREIGN KEY

- Establishes a relationship between two tables. It links the column(s) in one table to the primary key column(s) in another table.
- Ensures referential integrity.

Example:

```
sql
Copy code
CREATE TABLE orders (
    order_id INT PRIMARY KEY,
    customer_id INT,
    FOREIGN KEY (customer_id) REFERENCES customers(customer_i
d)
);
```

---

3. UNIQUE

- Ensures that all values in a column are unique. Unlike the primary key, a table can have multiple unique constraints.

Example:

```
sql
Copy code
CREATE TABLE employees (
    employee_id INT PRIMARY KEY,
    email VARCHAR(100) UNIQUE
);
```

---

4. NOT NULL

- Ensures that a column cannot have null (empty) values.

Example:

```
sql
Copy code
CREATE TABLE products (
```

```
    product_id INT PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
    price DECIMAL(10, 2) NOT NULL
);
```

5. CHECK

- Ensures that all values in a column satisfy a specific condition.

Example:

```
sql
Copy code
CREATE TABLE accounts (
    account_id INT PRIMARY KEY,
    balance DECIMAL(10, 2),
    CHECK (balance >= 0)  -- Ensures the balance is not negat
ive
);
```

6. DEFAULT

- Provides a default value for a column if no value is specified during insertion.

Example:

```
sql
Copy code
CREATE TABLE inventory (
    item_id INT PRIMARY KEY,
    quantity INT DEFAULT 0
);
```

7. AUTO_INCREMENT

- Automatically generates a unique value for a column (commonly used for primary keys).

Example:

```
sql
Copy code
CREATE TABLE orders (
    order_id INT AUTO_INCREMENT PRIMARY KEY,
    order_date DATE NOT NULL
);
```

Combining Multiple Constraints

You can apply multiple constraints on a single column. For example:

```
sql
Copy code
CREATE TABLE users (
    user_id INT AUTO_INCREMENT PRIMARY KEY,
    username VARCHAR(50) UNIQUE NOT NULL,
    age INT CHECK (age >= 18)
);
```

Viewing Constraints

To view constraints applied to a table, use:

```
sql
Copy code
SHOW CREATE TABLE table_name;
```

When to Use Constraints

1. **PRIMARY KEY**: When you need a unique identifier for each record.
2. **FOREIGN KEY**: To maintain relationships between tables.
3. **NOT NULL**: For mandatory fields like names or email addresses.
4. **UNIQUE**: For fields that should not have duplicate values, like email or usernames.
5. **CHECK**: To enforce specific business rules, such as age restrictions.
6. **DEFAULT**: For columns that should have a preset value when no value is provided.

13. Primary key and Relationship schema:

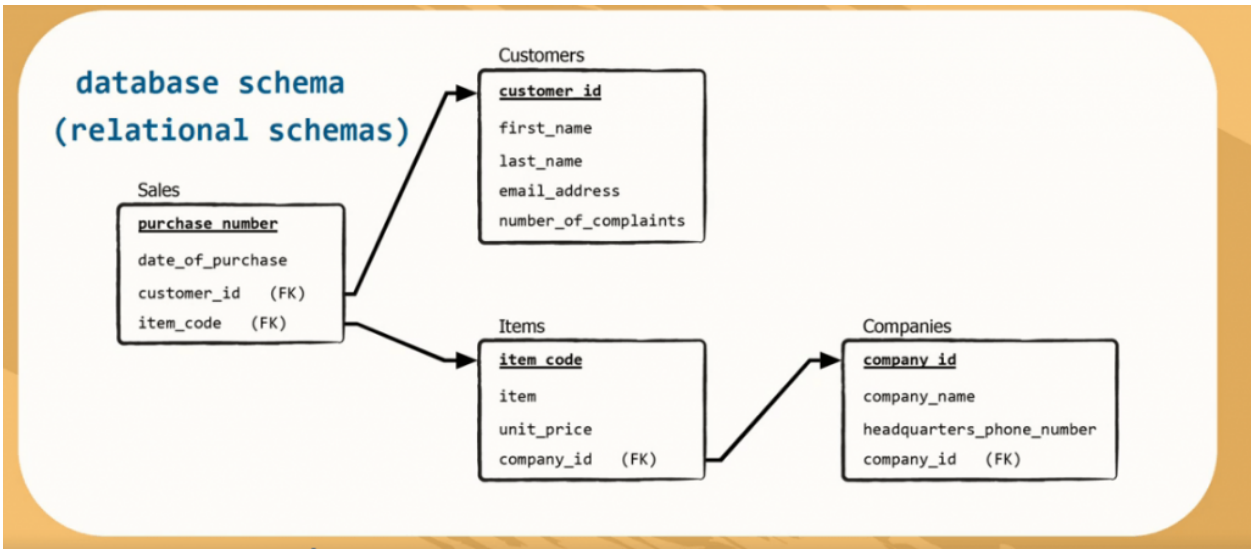

database schema
(relational schemas)
Customers
customer id
first_name
last_name
email_address
number_of_complaints
Sales
purchase number
date_of_purchase
customer_id (FK)
item_code (FK)
Items
item code
item
unit_price
company_id (FK)
Companies
company id
company_name
headquarters_phone_number
company_id (FK)

Table: orders

order_id	product	total	customer_id
1	Paper	500	5
2	Pen	10	2
3	Marker	120	3
4	Books	1000	1
5	Erasers	20	4

Table: customers

customer_id	first_name	last_name	phone	country
1	John	Doe	817-646-8833	USA
2	Robert	Luna	412-862-0502	USA
3	David	Robinson	208-340-7906	UK
4	John	Reinhardt	307-242-6285	UK
5	Betty	Doe	806-749-2958	UAE

What is a Primary Key in SQL?

In SQL, a **Primary Key** is a column (or a combination of columns) that uniquely identifies each row in a table. A table can have only one primary key, and it ensures the following properties:

1. **Uniqueness**: No two rows in a table can have the same value in the primary key column(s).
2. **Non-Null**: A primary key column cannot contain `NULL` values.

The primary key is essential for maintaining data integrity and ensuring each record can be uniquely identified. It also helps in establishing relationships between tables in a database.

Key Characteristics of a Primary Key

1. **Uniqueness**: Ensures data in the column(s) is unique for every record.
2. **Non-Null Constraint**: Primary key values cannot be `NULL`.
3. **Indexing**: A primary key automatically creates a unique clustered index on the column(s).
4. **One per Table**: A table can have only one primary key.

Important Rules and Features of Primary Keys

1. **Uniqueness**:
   - A primary key ensures that all values in the column(s) are unique.
   - Example: Two students cannot have the same `StudentID`.
2. **Non-NULL**:
   - A primary key column cannot contain `NULL` values.
   - Example: `StudentID` must have a value for every row.
3. **Single Table Rule**:
   - A table can have only one primary key.
   - Example: You cannot declare `PRIMARY KEY (StudentID)` and `PRIMARY KEY (FirstName)` in the same table.
4. **Composite Keys**:
   - A primary key can consist of one or more columns.

- Example: To uniquely identify enrollment records, you can combine `StudentID` and `CourseID`.

5. **Primary Key vs. Unique Key**:
   - A `PRIMARY KEY` is always unique and cannot be `NULL`.
   - A `UNIQUE KEY` can allow a single `NULL` value but must enforce uniqueness otherwise.
6. **Automatic Indexing**:
   - A primary key automatically creates a clustered index (in most SQL databases).
7. **Referencing Primary Keys**:
   - Primary keys are commonly used in **foreign key** relationships to establish connections between tables

Class coding Notes:

```
select database();

CREATE TABLE Students (
StudentID INT NOT NULL PRIMARY KEY,
FirstName VARCHAR(50),
LastName VARCHAR(50),
Age INT
);
DESCRIBE Students;

SHOW INDEX FROM Students;

CREATE TABLE Students1 (
StudentID INT NOT NULL,
FirstName VARCHAR(50),
LastName VARCHAR(50),
Age INT,
```

```
PRIMARY KEY (StudentID)
);

CREATE TABLE Enrollments (
StudentID INT NOT NULL,
CourseID INT NOT NULL,
PRIMARY KEY (StudentID, CourseID)
);

CREATE TABLE Students (
StudentID INT NOT NULL PRIMARY KEY,
FirstName VARCHAR(50),
LastName VARCHAR(50),
Age INT
);

ALTER TABLE students
Drop Primary Key;

ALTER TABLE Students
ADD PRIMARY KEY (StudentID);
```

Ways to Create a Primary Key

Here are the different ways to define a primary key in SQL:

---

1. Define the Primary Key Inline (Column Level)

You can define the primary key directly in the column definition.

```
CREATE TABLE Students (
```

```
    StudentID INT NOT NULL PRIMARY KEY,
    FirstName VARCHAR(50),
    LastName VARCHAR(50),
    Age INT
);
```

- **Inline Definition**: The `PRIMARY KEY` is declared alongside the column name (`StudentID`).
- **Benefit**: Simple for tables with a single primary key column.

---

2. Define the Primary Key as a Table Constraint

You can define the primary key separately at the table level after defining the columns.

```
CREATE TABLE Students (
    StudentID INT NOT NULL,
    FirstName VARCHAR(50),
    LastName VARCHAR(50),
    Age INT,
    PRIMARY KEY (StudentID)
);
```

- **Table-Level Definition**: The `PRIMARY KEY` constraint is applied to a column after all columns are defined.
- **Benefit**: Useful for single-column or multi-column primary keys.

---

3. Create a Composite Primary Key

You can define a primary key that spans multiple columns (composite key). This is done using a table-level constraint.

```
CREATE TABLE Enrollments (
    StudentID INT NOT NULL,
    CourseID INT NOT NULL,
    PRIMARY KEY (StudentID, CourseID)
);
```

- **Composite Primary Key**: Combines `StudentID` and `CourseID` to uniquely identify each row.

---

4. Add a Primary Key to an Existing Table

If a table is already created without a primary key, you can add it later.

```
sql
Copy code
ALTER TABLE Students
ADD PRIMARY KEY (StudentID);
```

---

5. Remove a Primary Key

You can drop the primary key if it is no longer required.

```
ALTER TABLE Students
DROP PRIMARY KEY;
```

Commands to Check and Describe the Primary Key

1. **Describe the Table**:

```
DESCRIBE Students;
```

2. **Show Table Schema**:

```
TABLE Students;
```

3. **Check Indexes (Including Primary Key)**:

```
SHOW INDEX FROM Students;
```

Foreign Key :

class code :

use sales;

- - CREATE TABLE Students (
  -- StudentID INT PRIMARY KEY,
  -- FirstName VARCHAR(50),
  -- LastName VARCHAR(50)
  --);
- - CREATE TABLE Enrollments (
  -- EnrollmentID INT PRIMARY KEY,

-- StudentID INT,
-- CourseName VARCHAR(50),
-- FOREIGN KEY (StudentID) REFERENCES Students(StudentID)
--);

- CREATE TABLE Enrollments4 (
-- EnrollmentID INT PRIMARY KEY,
-- StudentID INT,
-- CourseName VARCHAR(50)

- -);
- - ALTER TABLE Enrollments4
  -- ADD CONSTRAINT fk_student4
  -- FOREIGN KEY (StudentID)
  -- REFERENCES Students(StudentID)
  -- ON DELETE CASCADE
  -- ON UPDATE CASCADE
- - ;
- - ALTER TABLE Enrollments3
  -- DROP FOREIGN KEY fk_student2;

CREATE TABLE Orders (
OrderID INT,
CustomerID INT,
ProductID INT,
PRIMARY KEY (OrderID),
FOREIGN KEY (CustomerID, ProductID) REFERENCES Customers(CustomerID, ProductID)
);

What is a Foreign Key in SQL?

A **Foreign Key** is a column or a set of columns in a table that establishes a relationship between data in two tables. It refers to the **Primary Key** in another table, ensuring data integrity by enforcing a link between the two tables.

A foreign key:

- Creates a **parent-child relationship** between two tables.
- Ensures that the values in the foreign key column(s) match the primary key values in the referenced table.
- Prevents invalid data from being inserted into the foreign key column.

Key Properties of a Foreign Key

1. **Relationship**: A foreign key links two tables by referencing the primary key in the parent table.
2. **Data Integrity**: Ensures that only valid data (that exists in the parent table) can be inserted in the child table.
3. **Referential Integrity**: Prevents actions that would break the link between the tables, such as deleting a referenced row.
4. **Multiple Foreign Keys**: A table can have more than one foreign key.
5. **Cascade Actions**: Defines what happens when the parent table's data is updated or deleted (e.g., `ON DELETE CASCADE`).

Common Errors and Rules

1. **Mismatch in Data Types**:
   - Ensure that the foreign key column and the referenced primary key column have the same data type.
2. **Violation of Referential Integrity**:
   - You cannot delete or update a parent record if it would leave orphaned records in the child table (unless `CASCADE` is defined).
3. **Index Requirement**:

- The parent table must have a primary key or a unique key on the column being referenced.

4. **Nullability**:
   - Foreign key columns can allow `NULL` values, meaning the child row is not linked to a parent row.

Benefits of Using a Foreign Key

1. **Data Integrity**: Ensures data consistency between related tables.
2. **Referential Integrity**: Prevents orphaned rows in the child table.
3. **Logical Relationships**: Clearly defines relationships between tables.
4. **Simplifies Queries**: Helps in joining tables and retrieving related data.

Real-World Example

Tables: `Customers` and `Orders`

Customers Table

CustomerID	Name	Email
1	Alice	alice@email.com
2	Bob	bob@email.com

Orders Table

OrderID	CustomerID	Product	Quantity
101	1	Laptop	2
102	2	Headphones	1

- `CustomerID` in the `Orders` table is a foreign key referencing `CustomerID` in the `Customers` table.

Example of a Foreign Key

1. Defining a Foreign Key While Creating Tables

Let's create two related tables: **Students** and **Enrollments**.

Parent Table: `Students`

```
CREATE TABLE Students (
    StudentID INT PRIMARY KEY,
    FirstName VARCHAR(50),
    LastName VARCHAR(50)
);
```

Child Table: `Enrollments`

```
CREATE TABLE Enrollments (
    EnrollmentID INT PRIMARY KEY,
    StudentID INT,
    CourseName VARCHAR(50),
    FOREIGN KEY (StudentID) REFERENCES Students(StudentID)
);
```

- **Explanation**:
  - `Students.StudentID` is the **Primary Key** in the `Students` table.
  - `Enrollments.StudentID` is a **Foreign Key** that references `Students.StudentID`.
  - Each `StudentID` in `Enrollments` must exist in the `Students` table.

2. Add a Foreign Key to an Existing Table

If the table is already created, you can add the foreign key later using the `ALTER TABLE` statement.

```
sql
Copy code
ALTER TABLE Enrollments
ADD CONSTRAINT fk_student
FOREIGN KEY (StudentID)
REFERENCES Students(StudentID);
```

---

3. Composite Foreign Key

You can create a foreign key that consists of multiple columns.

```
sql
Copy code
CREATE TABLE Orders (
    OrderID INT,
    CustomerID INT,
    ProductID INT,
    PRIMARY KEY (OrderID),
    FOREIGN KEY (CustomerID, ProductID) REFERENCES Customers
(CustomerID, ProductID)
);
```

- **Composite Foreign Key**: `CustomerID` and `ProductID` together reference the `Customers` table.

Cascading Actions with Foreign Keys

Foreign keys can define actions when the parent table is updated or deleted:

1. **ON DELETE CASCADE**: Deletes the child rows if the referenced parent row is deleted.
2. **ON UPDATE CASCADE**: Updates the child rows when the referenced parent row is updated.
3. **ON DELETE SET NULL**: Sets the foreign key column to `NULL` if the referenced parent row is deleted.
4. **ON DELETE RESTRICT/NO ACTION**: Prevents deletion of the parent row if it is referenced by the child table.

Example with Cascading Actions

```
CREATE TABLE Students (
    StudentID INT PRIMARY KEY,
    FirstName VARCHAR(50),
    LastName VARCHAR(50)
);

CREATE TABLE Enrollments (
    EnrollmentID INT PRIMARY KEY,
    StudentID INT,
    CourseName VARCHAR(50),
    FOREIGN KEY (StudentID) REFERENCES Students(StudentID)
    ON DELETE CASCADE
    ON UPDATE CASCADE
);
```

- **ON DELETE CASCADE**: If a `StudentID` is deleted from the `Students` table, all related rows in the `Enrollments` table will also be deleted.
- **ON UPDATE CASCADE**: If the `StudentID` is updated in the `Students` table, it will automatically update the corresponding `StudentID` in the `Enrollments` table.
-

Dropping a Foreign Key

To drop a foreign key, use the `ALTER TABLE` statement. You need to know the name of the foreign key constraint.

1. **Find the Foreign Key Constraint Name**:

```
sql
Copy code
SHOW CREATE TABLE Enrollments;
```

2. **Drop the Foreign Key**:

```
sql
Copy code
ALTER TABLE Enrollments
DROP FOREIGN KEY fk_student;
```

Difference Between Primary Key and Foreign Key

Aspect	Primary Key	Foreign Key
Definition	A column or set of columns that uniquely identifies each row in a table.	A column or set of columns in one table that establishes a relationship with the primary key of another table.
Uniqueness	Ensures uniqueness; no two rows can have the same value in the primary key column(s).	Does not enforce uniqueness; multiple rows in the child table can have the same foreign key value.

Nullability	Cannot contain `NULL` values.	Can contain `NULL` values unless explicitly restricted.
Purpose	Used to uniquely identify each record in a table.	Used to enforce referential integrity and link two tables.
Table Scope	Defined within the table where it resides.	Refers to a primary key in another (or the same) table.
Indexing	Automatically creates a unique index for the primary key.	Does not create an index by default, but can be indexed manually for performance.
Relationship	Not dependent on any other table.	Always references the primary key of another table (parent-child relationship).
Modification Rules	Cannot be updated or deleted without cascading effects on dependent foreign keys.	Can be updated or deleted if `ON UPDATE` or `ON DELETE` actions are defined.

Key Differences in Usage

Primary Key:

- Ensures uniqueness within a single table.
- Every table must have one primary key for best practices.
- Critical for identifying records.

Foreign Key:

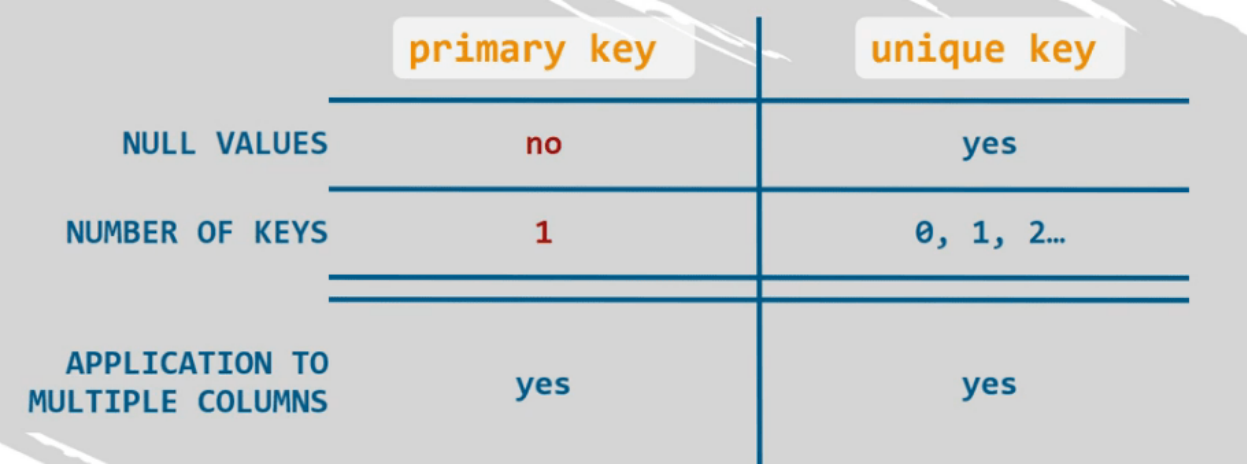
- Ensures relationships between tables.
- Improves data consistency across tables.
- Used for referential integrity.

Summary

- A **Foreign Key** links two tables and ensures data consistency and integrity.
- It references the **Primary Key** of another table.

- It supports cascading actions like `ON DELETE CASCADE` or `ON UPDATE SET NULL`.
- It's essential for modeling real-world relationships like `Customers` and `Orders` or `Students` and `Enrollments`

Primary Vs Unique Key :



	primary key	unique key
NULL VALUES	no	yes
NUMBER OF KEYS	1	0, 1, 2…
APPLICATION TO MULTIPLE COLUMNS	yes	yes

Unique Constraint in SQL

The **UNIQUE constraint** ensures that all values in a column or combination of columns are **unique** across all rows in a table. This is used to prevent duplicate values while still allowing `NULL` values (depending on database implementation).

Key Characteristics of UNIQUE Constraint

1. **Enforces Uniqueness**:
   - Ensures that no two rows have the same value in the specified column(s).
2. **Allows `NULL` Values**:
   - Unlike the primary key, UNIQUE constraints allow `NULL` values (though only one or a limited number of `NULL` values are allowed depending on the RDBMS).
3. **Multiple UNIQUE Constraints**:
   - A table can have multiple UNIQUE constraints applied to different columns.

Primary Key vs. Unique Constraint

Aspect	Primary Key	Unique Constraint
Uniqueness	Ensures all rows have unique values.	Ensures all rows have unique values but allows `NULL`.

Nullability	Cannot contain `NULL` values.	Allows `NULL` values.
Number of Keys	Only one primary key per table.	Multiple UNIQUE constraints can be added to a table.
Purpose	Uniquely identifies a record in the table.	Prevents duplicate values in a column or set of columns.
Automatic Index	Automatically creates a clustered index.	Automatically creates a non-clustered index.

Syntax for UNIQUE Constraint

1. Adding UNIQUE While Creating a Table

```
CREATE TABLE table_name (
    column1 data_type UNIQUE,
    column2 data_type,
    ...
);
```

Example:

```
CREATE TABLE Employees (
    EmployeeID INT PRIMARY KEY,
    Email VARCHAR(100) UNIQUE,
    PhoneNumber VARCHAR(15) UNIQUE
);
```

- Here, both `Email` and `PhoneNumber` must be unique across all rows.

2. Adding UNIQUE on Multiple Columns (Composite Unique Key)

```
s
CREATE TABLE table_name (
    column1 data_type,
    column2 data_type,
    ...
    CONSTRAINT constraint_name UNIQUE (column1, column2)
);
```

Example:

```
CREATE TABLE Orders (
    OrderID INT PRIMARY KEY,
    CustomerID INT,
    ProductID INT,
    CONSTRAINT unique_order UNIQUE (CustomerID, ProductID)
);
```

- Here, the combination of `CustomerID` and `ProductID` must be unique.

---

3. Adding UNIQUE to an Existing Table

You can modify an existing table to add a UNIQUE constraint:

```
ALTER TABLE table_name
ADD CONSTRAINT constraint_name UNIQUE (column_name);
```

Example:

```
ALTER TABLE Employees
```

```
ADD CONSTRAINT unique_email UNIQUE (Email);
```

Checking for UNIQUE Constraint

1. Using DESCRIBE Command

```
DESCRIBE table_name;
```

Example:

```
DESCRIBE Employees;
```

2. Using `SHOW CREATE TABLE`

```
SHOW CREATE TABLE table_name;
```

Example:

```
SHOW CREATE TABLE Employees;
```

This command will display the SQL statement used to create the table, including UNIQUE constraints.

Removing UNIQUE Constraint

To drop a UNIQUE constraint, use the `ALTER TABLE` command:

```
ALTER TABLE table_name
DROP INDEX constraint_name;
```

Example:

```
ALTER TABLE Employees
DROP INDEX unique_email;
```

class codig :

- - use sales ;
- - CREATE TABLE Employees (
  -- EmployeeID INT PRIMARY KEY,
  -- Email VARCHAR(100) UNIQUE,
  -- PhoneNumber VARCHAR(15) UNIQUE
  --);
- - DESCRIBE Employees;
- - CREATE TABLE Orders (
  -- OrderID INT PRIMARY KEY,
  -- CustomerID INT,
  -- ProductID INT,
  -- CONSTRAINT unique_order UNIQUE (CustomerID, ProductID)
  --);
- - DESCRIBE Orders;
- - CREATE TABLE Employees1 (
  -- EmployeeID INT PRIMARY KEY,
  -- Email VARCHAR(100) ,
  -- PhoneNumber VARCHAR(15)
  --);

- - DESCRIBE Employees1;
- - ALTER TABLE Employees1
  -- ADD CONSTRAINT unique_email UNIQUE (Email);
- - DESCRIBE Employees1
- - ALTER TABLE Employees1
  -- DROP INDEX unique_email;

DESCRIBE Employees1