LU06c - SQL-DDL: Constraint Management - Under Construction

Source:

- W3Schools | Constraints | Overview
- W3Schools | Constraints | PRIMARY KEY
- W3Schools | Constraints | FOREIGN KEY
- W3Schools | Constraints | NOT NULL
- W3Schools | Constraints | UNIQUE

Learning Objectives

1. Discuss what database Constraints are and for they are needed
2. Explain the four most important CONSTRAINTS in database systems
3. Apply constraints to entity and realation tables in databases

Overview

Sources:

- Youtube-DE | Übersicht Constraints
- Youtube-EN | CONSTRAINT Tutorial

MySQL constraints ensure data integrity, enforcing rules at the database level. Constraints restrict the type of data that can be inserted into tables, preventing invalid entries and ensuring relationships between tables remain accurate. The most common constraints in MySQL are

- Primary Key
- Foreign Key
- NOT NULL
- AUTO_INCREMENT
- UNIQUE

Let’s explore these constraints with their syntax and practical examples.

PRIMARY KEY

The Primary Key constraint uniquely identifies each record in a table. A primary key column (or a set of columns) must contain unique, non-null values. Each table can have only one primary key.

General Syntax

```
CREATE TABLE table_name (
  column_name1 datatype PRIMARY KEY,
```

```
  column_name2 datatype
);
```

Example

In the follwoing example, customer_id is the primary key, ensuring that every customer has a unique ID. It prevents any duplication of the customer_id.

```
CREATE TABLE customers (
    customer_id INT PRIMARY KEY,
    customer_name VARCHAR(50)
);
```

FOREIGN KEY

The Foreign Key constraint ensures referential integrity by linking a column in one table to the primary key of another. It establishes a relationship between two tables and enforces the rule that data in the foreign key column must match an existing primary key in the referenced table.

General Syntax

```
CREATE TABLE table_name (
  column_name1 datatype,
  column_name2 datatype,
  FOREIGN KEY (column_name1) REFERENCES another_table(primary_key_column)
);
```

Example

```
CREATE TABLE orders (
  order_id INT PRIMARY KEY,
  customer_id INT,
  FOREIGN KEY (customer_id) REFERENCES customers(customer_id)
);
```

In this example, customer_id in the orders table references the customer_id in the customers table. This relationship ensures that any customer_id in the orders table corresponds to an existing customer in the customers table.

NOT NULL

The NOT NULL constraint ensures that a column cannot contain a NULL value. It is used when the column must always have a value.

General Syntax

```
CREATE TABLE table_name (
  column_name datatype NOT NULL
```

```
);
```

Example

```
CREATE TABLE employees (
  employee_id INT PRIMARY KEY,
  employee_name VARCHAR(50) NOT NULL
);
```

In this example, the employee_name column is constrained to never contain a NULL value, ensuring every employee has a name.

AUTO INCREMENT

The AUTO_INCREMENT attribute automatically generates a unique number for new rows. It is typically used with the primary key to create unique identifiers without manual input.

General Syntax

```
CREATE TABLE table_name (
  column_name datatype AUTO_INCREMENT,
  PRIMARY KEY (column_name)
);
```

Example

```
CREATE TABLE products (
  product_id INT AUTO_INCREMENT PRIMARY KEY,
  product_name VARCHAR(50)
);
```

Here, product_id automatically increments each time a new row is inserted, ensuring a unique ID for every product.

UNIQUE

The UNIQUE constraint ensures that all values in a column (or a set of columns) are unique across the table. Unlike the primary key, a table can have multiple unique constraints.

General Syntax

```
CREATE TABLE table_name (
  column_name datatype UNIQUE
);
```

Example

```
CREATE TABLE users (
```

```
  user_id INT PRIMARY KEY,
  email VARCHAR(100) UNIQUE
);
```

In this case, the email column must contain unique values. No two users can have the same email address.

Practical Example with Multiple Constraints

Let’s create a students table to demonstrate multiple constraints in action.

```
CREATE TABLE students (
  student_id INT AUTO_INCREMENT PRIMARY KEY,
  student_name VARCHAR(50) NOT NULL,
  email VARCHAR(100) UNIQUE,
  class_id INT,
  FOREIGN KEY (class_id) REFERENCES classes(class_id)
);
```

Summary

- **AUTO_INCREMENT** is applied to student_id to automatically generate a unique ID for each student.
- **NOT NULL** is applied to student_name, requiring every student to have a name.
- **UNIQUE** ensures that no two students can register with the same email.
- **FOREIGN KEY** links class_id to another table classes, maintaining the relationship between students and their class.

Why Use Constraints?

- **Data Integrity:** Constraints ensure that the data stored in the database adheres to rules, such as ensuring unique IDs or valid foreign key relationships.
- **Preventing Errors:** Constraints like NOT NULL prevent the insertion of incomplete or invalid data.
- **Automation: AUTO_INCREMENT** simplifies the process of assigning unique identifiers without user input.

In summary, MySQL constraints like Primary Key, Foreign Key, NOT NULL, AUTO_INCREMENT, and UNIQUE are essential for maintaining data accuracy, integrity, and consistency across a database. They enforce rules at the database level, ensuring reliable data relationships and preventing errors.

Vocabulary

English	Deutsch
constraint	Bedingung, Beschränkung
to enforce	durchsetzen

English	Deutsch
to restrict	einschränken, beschränken
to prevent	verhindern
uniquely	eindeutig
