MySQL Constraints Overview

Constraints in MySQL are used to specify rules for the data in a table. These rules help ensure data integrity by enforcing certain conditions during data entry.

Let's explore each type of constraint, along with detailed explanations, syntax, and operations for deletion.

---

1. NOT NULL Constraint

- **Definition**: The `NOT NULL` constraint ensures that a column cannot have a NULL value.
- **Purpose**: Ensures mandatory fields are always filled.

Syntax:

```
CREATE TABLE emp (
    id INT,
    name VARCHAR(20) NOT NULL
);
```

Postmortem:

- **Why use it**: Ensures no record can be inserted without providing a value for the `name` field.
- **Behavior**: If you attempt to insert a record without a value for `name`, the operation will fail.

Deleting `NOT NULL` Constraint:

```
ALTER TABLE emp MODIFY name VARCHAR(20) NULL;
```

---

2. PRIMARY KEY Constraint

- **Definition**: The `PRIMARY KEY` uniquely identifies each record in a table. A table can have only one primary key, and it must be unique and not null.
- **Purpose**: Guarantees each row in the table is unique and easily identifiable.

Syntax:

```
CREATE TABLE emp (
    id INT,
    name VARCHAR(20),
    PRIMARY KEY (id)
);
```

Postmortem:

- **Why use it**: Ensures each employee has a unique identifier (`id`), preventing duplicate records.

- **Behavior**: The `PRIMARY KEY` column(s) automatically have both `NOT NULL` and `UNIQUE` constraints.

Deleting `PRIMARY KEY` Constraint:

```
ALTER TABLE emp DROP PRIMARY KEY;
```

---

3. **UNIQUE Constraint**

- **Definition**: The `UNIQUE` constraint ensures that all the values in a column are different.
- **Purpose**: Prevents duplicate values in a column but allows multiple NULL values.

Syntax:

```
CREATE TABLE emp (
    id INT,
    email VARCHAR(50) UNIQUE
);
```

Postmortem:

- **Why use it**: Ensures email addresses are unique, preventing duplicate records.
- **Behavior**: If you try to insert a duplicate email, the query will fail.

Deleting `UNIQUE` Constraint:

```
ALTER TABLE emp DROP INDEX email;
```

---

4. **CHECK Constraint**

- **Definition**: The `CHECK` constraint ensures that a value satisfies a specific condition.
- **Purpose**: Validates values before they are inserted into a column.

Syntax (MySQL 8.0+):

```
CREATE TABLE emp (
    id INT,
    age INT,
    CHECK (age >= 18)
);
```

Postmortem:

- **Why use it**: Ensures only employees who are 18 years or older are entered into the database.
- **Behavior**: If an employee younger than 18 is inserted, the operation will fail.

---

5. **DEFAULT Constraint**

- **Definition**: The `DEFAULT` constraint provides a default value for a column when no value is specified during insert.
- **Purpose**: Automatically fills in a value when a record is inserted without specifying that column.

Syntax:

```
CREATE TABLE emp (
    id INT,
    status VARCHAR(20) DEFAULT 'active'
);
```

Postmortem:

- **Why use it**: Ensures every employee gets a status of 'active' by default if no status is provided.
- **Behavior**: If no value is inserted for `status`, the value defaults to 'active'.

---

6. AUTO_INCREMENT Constraint

- **Definition**: The `AUTO_INCREMENT` constraint automatically generates a unique number when a new record is inserted.
- **Purpose**: Automatically assigns a unique ID to each new row, useful for primary keys.

Syntax:

```
CREATE TABLE emp (
    id INT AUTO_INCREMENT,
    name VARCHAR(20),
    PRIMARY KEY (id)
);
```

Postmortem:

- **Why use it**: Automatically generates a unique identifier without manual input, simplifying record insertion.
- **Behavior**: Each time a new employee is added, the `id` increments.

Setting `AUTO_INCREMENT` to Start from a Specific Number:

```
ALTER TABLE emp AUTO_INCREMENT = 100;
```

---

7. FOREIGN KEY Constraint

- **Definition**: The `FOREIGN KEY` constraint links two tables by referring to the `PRIMARY KEY` of another table.
- **Purpose**: Enforces referential integrity by ensuring that values in one table correspond to valid values in another.

Syntax:

```sql
CREATE TABLE emp (
    id INT PRIMARY KEY,
    dept_id INT,
    FOREIGN KEY (dept_id) REFERENCES department(dept_id)
);
```

Postmortem:

- **Why use it**: Ensures that `dept_id` in the `emp` table refers to an existing department, preventing invalid department assignments.
- **Behavior**: If you try to insert a `dept_id` that doesn't exist in the `department` table, the operation will fail.

Deleting `FOREIGN KEY` Constraint:

```sql
ALTER TABLE emp DROP FOREIGN KEY emp_dept_fk;
```

---

8. COMPOSITE KEY

- **Definition**: A `COMPOSITE KEY` is a combination of two or more columns used to create a unique identifier for a record.
- **Purpose**: Ensures uniqueness across multiple columns when a single column is insufficient.

Syntax:

```sql
CREATE TABLE emp (
    emp_id INT,
    dept_id INT,
    PRIMARY KEY (emp_id, dept_id)
);
```

Postmortem:

- **Why use it**: Used when multiple columns together form a unique identifier, like an employee assigned to a specific department.
- **Behavior**: Ensures that a specific employee cannot be assigned to the same department more than once.

Deleting `COMPOSITE KEY`:

```sql
ALTER TABLE emp DROP PRIMARY KEY;
```

Deleting Constraints in MySQL

1. **Delete `PRIMARY KEY`**:

   ```sql
   ALTER TABLE emp DROP PRIMARY KEY;
   ```

2. **Delete `UNIQUE` Key**:

   ```sql
   ALTER TABLE emp DROP INDEX column_name;
   ```

3. **Delete `NOT NULL` Constraint**:

```sql
ALTER TABLE emp MODIFY column_name data_type NULL;
```

4. **Delete `FOREIGN KEY`**:

```sql
ALTER TABLE emp DROP FOREIGN KEY fk_name;
```

Differences Between Primary Key and Unique Key

Feature	Primary Key	Unique Key
Uniqueness	Enforces uniqueness for each record	Ensures all values are unique
NULL Values	Cannot contain NULL values	Can contain multiple NULL values
Number Allowed	Only one primary key per table	Multiple unique keys per table
Auto-increment	Often used with `AUTO_INCREMENT`	Not commonly used with auto-increment
Example	`PRIMARY KEY (id)`	`UNIQUE (email)`

Difference Between `DELETE`, `TRUNCATE`, and `DROP`

Command	DELETE	TRUNCATE	DROP
Purpose	Removes specific rows based on a condition	Removes all rows from a table	Deletes the entire table (schema and data)
WHERE Clause	Can use `WHERE` to filter records	Cannot use `WHERE`; deletes all rows	Cannot use `WHERE`; deletes the entire table
Performance	Slower (logged operation)	Faster (non-logged operation)	Fast (completely removes table structure)
Rollback	Can be rolled back (if within a transaction)	Cannot be rolled back	Cannot be rolled back
Auto-increment Reset	Doesn't reset `AUTO_INCREMENT` values	Resets `AUTO_INCREMENT` values	Not applicable (table is dropped)

Example Usage

1. **DELETE Example**:

```sql
DELETE FROM emp WHERE id = 5;
```

2. **TRUNCATE Example**:

```sql
TRUNCATE TABLE emp;
```

3. **DROP Example**:

```sql
DROP TABLE emp;
```

Additional Concepts:

- **Indexes**: Used to retrieve data faster.

- **Views**: Virtual tables representing the result of a query.
- **Stored Procedures**: Reusable SQL code for common tasks.

This comprehensive guide provides a deep understanding of MySQL constraints and their practical usage.

In [ ]: