AROR UNIVERSITY
OF ART, ARCHITECTURE,
DESIGN & HERITAGE,
SUKKUR, SINDH

Department of Artificial Intelligence & Multimedia Gamming

CSC-207: Database Systems

Lab # 09: To Work with SQL Constraints

Objectives

1. To learn an application of Constraints on a Table & its columns.

Introduction of SQL Constraints

- SQL Constraints are rules used to limit the type of data that can go into a table, to maintain the accuracy and integrity of the data inside table. Constraints can be divided into the following two types,

1. **Column level constraints:** A rule defined directly within a column's specification in a table, restricting the type or values allowed in that single column.
2. **Table level constraints:** A rule defined separately from column specifications in a table, often involving multiple columns within the same table or establishing relationships with other tables.

```
CREATE TABLE Students (
   StudentID INT PRIMARY KEY AUTO_INCREMENT,    -- Column-level PRIMARY KEY
   FirstName VARCHAR(50) NOT NULL,              -- Column-level NOT NULL
   LastName VARCHAR(50) NOT NULL,               -- Column-level NOT NULL
   Email VARCHAR(100) UNIQUE,                   -- Column-level UNIQUE
   Major VARCHAR(50),
   EnrollmentDate DATE,
   CONSTRAINT UC_StudentMajor UNIQUE (StudentID, Major),          -- Table-level UNIQUE on
multiple columns
   CONSTRAINT FK_Major FOREIGN KEY (Major) REFERENCES Departments(DepartmentName) ON
DELETE SET NULL ON UPDATE CASCADE                                 -- Table-level FOREIGN KEY
);
```

Constraints are used to make sure that the integrity of data is maintained in the database. Following are the most used constraints that can be applied to a table.

NOT NULL: Ensures that a column cannot contain null values.

UNIQUE: Ensures that all values in a column are unique.

PRIMARY KEY: Defines a column (or combination of columns) that uniquely identifies each row in a table.

FOREIGN KEY: Establishes a link between data in two tables, enforcing referential integrity.

CHECK: Specifies a condition that must be met for the data to be valid.

AUTO_INCREMENT: Automatically generates a unique value for a column, typically used for generating primary key values.

DEFAULT: Assigns a default value to a column if no other value is specified.

NOT NULL Constraint

NOT NULL constraint restricts a column from having a NULL value. Once **NOT NULL** constraint is applied to a column, you cannot pass a null value to that column. It enforces a column to contain a proper value.

One important point to note about this constraint is that it cannot be defined at table level.

Example:

```
CREATE TABLE Employees (
    EmployeeID INT NOT NULL,
    FirstName VARCHAR(50) NOT NULL,
    LastName VARCHAR(50) NOT NULL
);
```

Example to check:

Insert into employees (EmployeeID,FirstName,LastName) values (2, 'jack', Null);

```
INSERT INTO Employees (EmployeeID, FirstName, LastName) VALUES (1, 'Jack', NULL);
 -- This should fail due to NOT NULL constraint on LastName
```

```
7  23:48:47  INSERT INTO Employees (EmployeeID, FirstName, LastName) VALUES (1, 'Jack', N...  Error Code: 1048. Column 'LastName' cannot be null
```

UNIQUE Constraint

UNIQUE constraint ensures that a field or column will only have unique values. A UNIQUE constraint field will not have duplicate data. This constraint can be applied at column level or table level.

Using UNIQUE constraint when creating a Table (Table Level)

Example:

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

The above query will declare that the StudentID field of Student table will only have unique values and wont take NULL value.

Example to check:
Insert into Students (StudentID,FirstName,LastName) values (3, 'jack','phill');
Insert into Students (StudentID,FirstName,LastName) values (3, 'jack','phill');

```
INSERT INTO Students (StudentID, FirstName, LastName) VALUES (1, 'Jack', 'Doe');
INSERT INTO Students (StudentID, FirstName, LastName) VALUES (1, 'Jane', 'Smith');
-- This should fail due to UNIQUE constraint on StudentID
```

```
11  23:56:24  INSERT INTO Students (StudentID, FirstName, LastName) VALUES (1, 'Jane', 'Smith')  Error Code: 1062. Duplicate entry '1' for key 'StudentID'
```

Using UNIQUE constraint after Table is created (Column Level)

ALTER TABLE Students Modify column LastName VARCHAR(50) unique;

ALTER TABLE Student ADD UNIQUE(StudentID);

The above query specifies that s_id field of Student table will only have unique value. You cannot add this constraint to column that already have duplicate values.

PRIMARY KEY Constraint

Primary key constraint uniquely identifies each record in a database. A Primary Key must contain unique value and it must not contain null value. Usually, Primary Key is used to index the data inside the table.

Using PRIMARY KEY constraint at Table Level

Example:

```
CREATE TABLE Orders (
    OrderID INT PRIMARY KEY,
    CustomerID INT,
    OrderDate DATE
);
```

Example to check:

-- This should fail due to duplicate PRIMARY KEY constraint on OrderID

```
INSERT INTO Orders (OrderID, CustomerID, OrderDate) VALUES (1, 1001, '2022-04-01');
INSERT INTO Orders (OrderID, CustomerID, OrderDate) VALUES (1, 1002, '2022-04-02');
-- This should fail due to duplicate PRIMARY KEY constraint on OrderID
```

Using PRIMARY KEY constraint at Column Level

ALTER table Student ADD PRIMARY KEY (StudentID);

The above command will creates a PRIMARY KEY on the StudentID.

For multiple columns:

```
PRIMARY KEY (StudentID, home_city)
CREATE TABLE Orders1 (
   OrderID INT, CustomerID INT,
   OrderDate DATE,
   PRIMARY KEY (OrderID,CustomerID )
);
```

FOREIGN KEY Constraint

FOREIGN KEY is used to relate two tables. FOREIGN KEY constraint is also used to restrict actions that would destroy links between tables. To understand FOREIGN KEY, let's see its use, with help of the below tables:

Customer_Detail Table

cid	CustomerName	address
101	Nida	Sukkur
102	Rameez	Karachi
103	Fabiha	Lahore

Order Table

Orderid	OrderName	cid
10	ProductID	10
11	Mobile	103
12	Order3	102

In **Customer_Detail** table, **cid** is the primary key which is set as foreign key in **Order** table. The value that is entered in **cid** which is set as foreign key in **Order** table must be present in **Customer_Detail** table where it is set as primary key. This prevents invalid data to be inserted into **cid** column of **Order** table.

If you try to insert any incorrect data, DBMS will return error and will not allow you to insert the data.

Example:

```
CREATE TABLE Customers (
    CustomerID INT PRIMARY KEY,
    FirstName VARCHAR(50),
    LastName VARCHAR(50)
);


CREATE TABLE Orderinfo (
    OrderID INT PRIMARY KEY,
    CustomerID INT,
    FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)
);
```

Example to check:

```
 -- Inserting a customer into the Customers table
INSERT INTO Customers (CustomerID, FirstName, LastName) VALUES (1001, 'ali', 'raza');

-- Attempting to insert an order with a non-existent CustomerID into the Orders table
INSERT INTO Orderinfo (OrderID, CustomerID) VALUES (1, 1001);  -- This will succeed

INSERT INTO Orderinfo (OrderID, CustomerID) VALUES (2, 1002);
 -- This will fail due to the FOREIGN KEY constraint on CustomerID
```

To show FK:

show create table *tablename*;

Behavior of Foreign Key Column on Delete/Update

- When two tables are connected with Foreign key, and certain data in the main table is deleted/updated, for which a record exists in the child table, then we must have some mechanism to save the integrity of data in the child table.
- MySQL allows creating a table with ***CASCADE, SET NULL and RESTRICT options.***

CASCADE: When a row in the parent table (containing the primary key) is deleted or updated, the corresponding rows in the child table (containing foreign keys referencing the primary key) are also automatically deleted or updated, respectively.

SET NULL: When a row in the parent table is deleted or updated, the foreign key columns in the child table are set to NULL. This is useful when you want to disassociate the child records from the parent records without deleting them.

RESTRICT: This option prevents the deletion or modification of rows in the parent table if there are matching rows in the child table. It acts as a constraint to enforce referential integrity and prevents operations that would result in orphaned records in the child table.

CASCADE option:

```
-- Create Teachers table
CREATE TABLE Teachers (
    TeacherID INT PRIMARY KEY,
    FirstName VARCHAR(50),
    LastName VARCHAR(50)
);
-- Create TeacherDetails table with CASCADE option
CREATE TABLE TeacherDetails (
    DetailID INT PRIMARY KEY,
    TeacherID INT,
```

```
    DetailInfo VARCHAR(255),
    FOREIGN KEY (TeacherID) REFERENCES Teachers(TeacherID) ON DELETE CASCADE ON
  UPDATE CASCADE
  );
  -- Insert data into Teachers table
  INSERT INTO Teachers (TeacherID, FirstName, LastName) VALUES (1, 'Ahmed', 'Khan');
  -- Insert related data into TeacherDetails table
  INSERT INTO TeacherDetails (DetailID, TeacherID, DetailInfo) VALUES (101, 1, 'Detail info
  for Ahmed Khan');
  -- Delete the record from Teachers table
  DELETE FROM Teachers WHERE TeacherID = 1;
  -- Check if related record is deleted from TeacherDetails table
  SELECT * FROM TeacherDetails; -- This should return an empty result set
-- Update record in Teachers table
UPDATE Teachers SET FirstName = 'Muhammad' WHERE TeacherID = 1;
-- Check if related record is updated in TeacherDetails table
SELECT * FROM TeacherDetails; -- This should show the updated record
```

RESTRICT option:

```
-- Create Lecturers table
CREATE TABLE Lecturers (
  LecturerID INT PRIMARY KEY,
  FirstName VARCHAR(50),
  LastName VARCHAR(50)
);
-- Create LecturerDetails table with RESTRICT option
CREATE TABLE LecturerDetails (
  DetailID INT PRIMARY KEY,
  LecturerID INT,
  DetailInfo VARCHAR(255),
  FOREIGN KEY (LecturerID) REFERENCES Lecturers(LecturerID) ON DELETE RESTRICT ON
UPDATE RESTRICT
```

```
);
-- Insert data into Lecturers table
INSERT INTO Lecturers (LecturerID, FirstName, LastName) VALUES (2, 'Fatima', 'Hussain');
-- Insert related data into LecturerDetails table
INSERT INTO LecturerDetails (DetailID, LecturerID, DetailInfo) VALUES (102, 2, 'Detail info for
Fatima Hussain');
-- Attempt to delete the record from Lecturers table
DELETE FROM Lecturers WHERE LecturerID = 2; -- This should fail due to RESTRICT option
-- Check if related record is still present in LecturerDetails table
SELECT * FROM LecturerDetails; -- This should still return the related record
-- Attempt to update record in Lecturers table
UPDATE Lecturers SET LastName = 'Hussain' WHERE LecturerID = 2; -- This should fail due to
RESTRICT option
-- Check if related record is still present in LecturerDetails table
SELECT * FROM LecturerDetails; -- This should still return the related record
```

CHECK Constraint

CHECK constraint is used to restrict the value of a column between a range. It performs check on the values, before storing them into the database. It's like condition checking before saving data into a column.

Using CHECK constraint at Table Level

```
-- Create a table with a CHECK constraint to ensure that age is greater than or equal to 18
CREATE TABLE Students (
    StudentID INT PRIMARY KEY,
    FirstName VARCHAR(50),
    LastName VARCHAR(50),
    Age INT,
    CHECK (Age >= 18)
);
-- Insert data into Students table
INSERT INTO Students (StudentID, FirstName, LastName, Age) VALUES (1, 'Ali', 'Khan', 20); --
Valid
```

```
INSERT INTO Students (StudentID, FirstName, LastName, Age) VALUES (2, 'Sana', 'Ahmed',
16); -- Invalid, will fail due to CHECK constraint
```

Example:

```
-- Create a table with a CHECK constraint to ensure that salary is greater than 0
CREATE TABLE Employees (
   EmployeeID INT PRIMARY KEY,
   FirstName VARCHAR(50),
   LastName VARCHAR(50),
   Salary DECIMAL(10, 2),
   CHECK (Salary > 0)
);
-- Insert data into Employees table
INSERT INTO Employees (EmployeeID, FirstName, LastName, Salary) VALUES (1, 'Ahmed', 'Ali',
50000.00); -- Valid
INSERT INTO Employees (EmployeeID, FirstName, LastName, Salary) VALUES (2, 'Fatima',
'Khan', -1000.00); -- Invalid, will fail due to CHECK constraint
```

Using CHECK constraint at Column Level

```
ALTER table Student ADD CHECK(s_id > 0);
```

CHECK CONSTRAINT using IN operator

MySQL CHECK CONSTRAINT can be applied to a column of a table, to set a limit for storing values within a range, along with IN operator.

```
CREATE TABLE patient(
        id INT(11) NOT NULL PRIMARY KEY,
        name VARCHAR(30),
        phone INT(15),
        country varchar(25) NOT NULL CHECK(country IN('UK', 'USA'))
);
```

You can use CHECK CONSTRAINT with LIKE, OR & AND operator and also with CASE statement.

AUTO_INCREMENT Constraint

MySQL allows you to set AUTO_INCREMENT to a column. Doing so will increase the value of that column by 1 automatically, each time a new record is added.

Example:

-- Create a new table with an AUTO_INCREMENT column

```
CREATE TABLE Products (
   ProductID INT AUTO_INCREMENT PRIMARY KEY,
   ProductName VARCHAR(255),
   Price DECIMAL(10, 2)
);
```

Example to check:

```
INSERT INTO Products (ProductName, Price) VALUES ('Product A', 10.99);

INSERT INTO Products (ProductName, Price) VALUES ('Product B', 19.99);

INSERT INTO Products (ProductName, Price) VALUES ('Product C', 15.49);
```

Example:

```
ALTER TABLE customer MODIFY id int AUTO_INCREMENT;
```

The above command will modify the id column of the customer table, to an auto increment.

DEFAULT Constraint

```
-- Create a table with a DEFAULT constraint

CREATE TABLE Books (
   BookID INT AUTO_INCREMENT PRIMARY KEY,
   Title VARCHAR(100),
   Author VARCHAR(100),
   Genre VARCHAR(50) DEFAULT 'Fiction'
);
-- Insert data into the table
INSERT INTO Books (Title, Author) VALUES ('To Kill a Mockingbird', 'Harper Lee'); -- Genre will default to 'Fiction'
```

INSERT INTO Books (Title, Author, Genre) VALUES ('1984', 'George Orwell', 'Dystopian'); -- Explicit genre value provided
INSERT INTO Books (Title, Author) VALUES ('Pride and Prejudice', 'Jane Austen'); -- Genre will default to 'Fiction'.

Exercises (Class)

1. Add here all the tasks performed in lab.

Exercises (Weekly)

1. Create a table DEPARTMENT with the following attributes:

DEPTNO number, DNAME varchar(10), LOC varchar(10). PRIMARY KEY constraint on DEPTNO.

2. Create a table EMPLOYEE with the following attributes:

EMPNO number, ENAME varchar(10),SAL number, DEPTNO number. Apply FOREIGN KEY constraint on DEPTNO referencing the DEPARTMENT table created in question 1 and PRIMARY KEY constraint on EMPNO and DEPTNO.

3. ALTER table EMPLOYEE created in question 2 and apply the constraint CHECK on

ENAME attribute such that ENAME should always be inserted in capital letters.

4. ALTER table DEPARTMENT created in question 1 and apply constraint on DNAME such that DNAME should not be entered empty.

5. ALTER table EMPLOYEE created in question 2 and apply the constraint on SAL attribute such that no two salaries of the employees should be similar.

6. ALTER table EMPLOYEE created in question 2 and apply the constraint on DEPTNO attribute such that on update, update a child value and on delete set null value to a child.