Constraints in SQL Server 🛡️

What are Constraints? 🤔

Constraints in SQL Server are predefined rules that enforce data integrity in the database tables. They ensure the accuracy and reliability of the data by imposing restrictions on the data that can be inserted, updated, or deleted in a table.

Need for Constraints in SQL Server:

The Data Types in SQL Server stops you from entering invalid data into a column. For example, you cannot insert string value into integer columns.

But that doesn't prevent users from entering invalid data. for example, the user may enter a negative number in the salary field. The same employee code may be given to two employees. Users may not provide a value to a required field like name, email , etc.

The constraint help us prevent those from happening.

Examples of Potential Issues without Constraints:

- Inputting a negative value in the **salary** field.
- Assigning the same **employee code** to two different employees.
- Leaving a required field like **name**, **email**, etc., blank.

Constraints come to the rescue here by preventing such inconsistencies and ensuring data integrity.

Types of Constraints:

1. Default constraint
2. Not Null constraint
3. Primary key constraints
4. Foreign Key constraints
5. Unique Key constraints
6. Check constraint

1-Default Constraint:

We use the Default Constraint to specify default values for a column. SQL Server uses the default value when the value for that column is absent in the insert query.

Example:

Without Default constraint

```
CREATE TABLE Employee (
  EmployeeID  [int]         ,
  FirstName    [varchar](50) ,
  LastName   [varchar](50) ,
  Department  [varchar](20) ,
)


insert into Employee (EmployeeID, FirstName, LastName) values (1,'Coder','Baba')


Select * from Employee


EmployeeID   FirstName   LastName   Department
----------   ---------   --------   ----------
1                  Coder       Baba       NULL
```

```
CREATE TABLE Employee (
  EmployeeID  [int]         ,
  Name       [varchar](50) ,
  Department  [varchar](20) Default 'Admin' ,
)
```

```
CREATE TABLE Employee (
  EmployeeID  [int]         ,
  Name       [varchar](50) ,
  Department  [varchar](20)  CONSTRAINT DF_Employee_Department  DEFAULT 'Admin'
)
```

2-Not Null Constraint:

Not null constraint specifies that the column cannot store a NULL value. All insert & updates to the column must specify a value. Attempting to insert or update NULL values will result in the error.

Example:

```
CREATE TABLE Employee (
   EmployeeID  [int]          ,
   Name       [varchar](50) ,
   Department  [varchar](20)  NOT NULL
)
```

3-Primary Key Constraint:

A primary key constraint ensures that each record in a table is unique and not null.

```
CREATE TABLE Employee (
   EmployeeID  [int] PRIMARY KEY  NOT NULL,
   Name       [varchar](50) NULL,
   Department  [varchar](20) NULL,
)
```

At table level

```
CREATE TABLE Employee (
   EmployeeID  [int] NOT NULL,
   Name       [varchar](50) NULL,
   Department  [varchar](20) NULL,
   PRIMARY KEY (EmployeeID )
)
```

Composite Primary Key

```
CREATE TABLE EmployeeLeave (
   EmployeeID  [int]  NOT NULL,
   LeaveID     [int]  NOT NULL,
   DateTaken     DateTime2] NOT NULL
   PRIMARY KEY ( EmployeeID, LeaveID )
)
```

Adding Primary key to an existing table.

```
Alter Table EmployeeLeave Add  PRIMARY KEY ( EmployeeID, LeaveID )
or

Alter Table EmployeeLeave
 Add CONSTRAINT PK_EmployeeLeave PRIMARY KEY ( EmployeeID, LeaveID )
```

4-Foreign Key Constraint: 🎯

A foreign key constraint establishes a relationship between two tables by referencing the primary key of another table

```
CREATE TABLE Customer (
   CustomerID  [int] NOT NULL ,
   Name varchar(10) NOT NULL,
   PRIMARY KEY (CustomerID)
)

CREATE TABLE Invoice (
   InvoiceID    [int]  NOT NULL,
   CustomerID  [int]  NOT NULL REFERENCES Customer(CustomerID),
   Amount     [decimal](18,2) NOT NULL,
   PRIMARY KEY (InvoiceID),
)
```

```
CREATE TABLE Invoice (
   InvoiceID    [int]  NOT NULL,
   CustomerID  [int]  NOT NULL ,
   Amount     [decimal](18,2) NOT NULL,
   PRIMARY KEY (InvoiceID),
   FOREIGN KEY (CustomerID) REFERENCES Customer(CustomerID)
)
```

Composite Foreign Key

```
CREATE TABLE TableA (
   TableID1  [int] NOT NULL ,
   TableID2  [int] NOT NULL ,
   PRIMARY KEY (TableID1,TableID2)
)

CREATE TABLE TableB (
   TableBID   [int]  NOT NULL,
   ID1  [int]  NOT NULL ,
   ID2  [int]  NOT NULL ,
   PRIMARY KEY (TableBID),
   CONSTRAINT FK_TableB_TableA_ID1_ID2 FOREIGN KEY (ID1,ID2) REFERENCES
TableA(TableID1,TableID2)
)
```

5-Unique Key Constraint:

Unique key constraint enforces the uniqueness of the column value.

i.e. no two rows of a table can have the same values

---Column Level

```
CREATE TABLE Employee (
   EmployeeID  int Primary Key      ,
   Name       varchar(50)  ,
  EmailID     varchar(50) UNIQUE

)
```

--- Table Level

```
CREATE TABLE Employee (
  EmployeeID  int        ,
  Name        varchar(50)  ,
  EmailID     varchar(50)  ,
  Primary Key (EmployeeID) ,
  UNIQUE (FirstName)
)
```

Composite Unique Key

```
CREATE TABLE Employee (
  EmployeeID  int NOT NULL ,
  FirstName    varchar(50)  ,
  LastName   varchar(50)  ,
  Primary Key (EmployeeID) ,
  UNIQUE (FirstName,LastName)
)
```

Naming the Unique Key

```
CREATE TABLE Employee (
  EmployeeID  int NOT NULL ,
  FirstName    varchar(50)  ,
  LastName   varchar(50)  ,
  Primary Key (EmployeeID) ,

  CONSTRAINT UK_FirstName_LastName UNIQUE(FirstName, LastName)
)
```

Adding Unique Key to Existing Table

```
Alter Table TableB

ADD CONSTRAINT UK_FirstName_LastName UNIQUE(FirstName,LastName)
```

6-Check Constraints:

The check constraints in SQL Server allows us to validate data values that are being inserted or updated in one or more columns. If the validation fails, then SQL Server will not insert or update the data.

“A check constraint ensures that the values in a column meet a specific condition.”

Example:

-- Check Constraint at column level

```
CREATE TABLE Employee (
   EmployeeID  int  primary key       ,
   Name       varchar(50)    CHECK (len(Name) > 15),
   Salary     Decimal(18,2) NOT NULL  CHECK (Salary > 0)
)
```

-- Check Constraint at Table level

```
CREATE TABLE Employee (
   EmployeeID  int         ,
   Name       varchar(50)  ,
   Salary     Decimal(18,2) NOT NULL    ,
   Primary Key (EmployeeID),
   CHECK (len(Name) > 15),
   CHECK (Salary > 0),
  )
```

Naming the Constraint

```
CREATE TABLE Employee (
   EmployeeID  int         ,
   Name       varchar(50)  ,
   Salary     Decimal(18,2) NOT NULL    ,
   Primary Key (EmployeeID),
   CONSTRAINT CK_Employee_Name    CHECK (len(Name) > 15),
   CONSTRAINT CK_Employee_Salary CHECK (Salary > 0),
)
```

Adding check constraint to an existing table

```
Alter Table Employee ADD CONSTRAINT CK_Employee Check (Salary > 0)
```

Drop a Constraints:

```
Alter table TableB drop CONSTRAINT UK_FirstName
```

Types of Constraints 📋

1. **Primary Key** 🔑
    - **Description**:
      A primary key constraint ensures that each record in a table is unique and not null.
    - **Example**:

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

Here, **StudentID** is a primary key, ensuring each student ID is unique.

2. **Foreign Key** 🎯
    - **Description**:
      A foreign key constraint establishes a relationship between two tables by referencing the primary key of another table.
    - **Example**:

```
CREATE TABLE Orders ( OrderID INT PRIMARY KEY, ProductID INT, FOREIGN KEY
(ProductID) REFERENCES Products(ProductID) );
```

Here, **ProductID** in the **Orders** table references the **ProductID** in the **Products** table.

3. **Unique** 🔄
    - **Description**:
      A unique constraint ensures that all values in a column are unique.
    - **Example**:

```
CREATE TABLE Employees ( EmployeeID INT UNIQUE, Name VARCHAR(50) );
```

Each **EmployeeID** must be unique in the **Employees** table.

4. **Check** ✅
    - **Description**:
      A check constraint ensures that the values in a column meet a specific condition.

- **Example**:

```
CREATE TABLE Products ( ProductID INT, Price DECIMAL(10, 2), CONSTRAINT CHK_Price
CHECK (Price > 0) );
```

The **CHK_Price** constraint ensures that the **Price** of a product is always greater than 0.

5. **Default** 🔄

   - **Description**:
     A default constraint provides a default value for a column when no value is specified.

   - **Example**:

```
CREATE TABLE Students ( StudentID INT PRIMARY KEY, Name VARCHAR(50), Status
VARCHAR(10) DEFAULT 'Active' );
```

If no **Status** is provided when inserting a student, it defaults to 'Active'.

Importance of Constraints 🌟

- **Data Integrity**: Constraints ensure data accuracy and consistency.
- **Relationship Establishment**: Foreign key constraints help in establishing relationships between tables.
- **Enhanced Performance**: Properly defined constraints can improve query performance.

Conclusion 📌

Understanding and implementing constraints is crucial for maintaining the quality and reliability of the data within a SQL Server database. Properly defined constraints not only ensure data integrity but also make the database more robust and efficient.

We are now on Telegram
Join us | Link in the bio

CB
CODER BABA

@coderbaba

Balram Yadav

/CODER BABA