DDL in MS-SQL Server

Soyeb Ghachi

Technical Trainer | Mentor |SME

Overview

DDL in MS-SQL Server

- -Create/Drop Database
- -Create/Drop Table
- -Alter Table

Constraints

- -Identity
- -Primary Key
- -Foreign Key
- -Unique Key
- -Not Null
- -Check

System Databases

🔧 System Databases in MS SQL Server:

System Database	Description
master	- Stores all system-level information such as login accounts, server configuration settings, information about all other databases, endpoints, and more. - If this database is corrupted or unavailable, SQL Server won't start.
model	- Serves as a **template** for all new databases. - Any objects or settings created in the model database will be copied to newly created databases.
msdb	- Used by **SQL Server Agent** for scheduling alerts and jobs. - Also stores backup and restore history, SQL Server Integration Services (SSIS) packages, and more.
tempdb	- A **temporary workspace** used for temporary tables, table variables, sorting operations, and other temporary needs. - It is recreated every time SQL Server is restarted.

DDL-Database

CREATE DATABASE

The CREATE DATABASE statement creates a new database. The following shows the minimal syntax of the CREATE DATABASE statement:

```
1 CREATE DATABASE database_name;
2
```

```
---TO SELECT ACTIVE DATABASE--
USE MASTER

--HOW TO CREATE A DATABASE
CREATE DATABASE MYAUTODB

--LIST DATABASES FROM CURRENT INSTANCE
SELECT * FROM SYS.databases WHERE NAME='MYAUTODB'
```

DROP DATABASE

The DROP DATABASE statement allows you to delete one or more databases with the following syntax:

```
DROP DATABASE  [ IF EXISTS ]
   database_name
   [,database_name2,...];
```

```
--DROP A DATABASE
USE MASTER
DROP DATABASE MYAUTODB
```

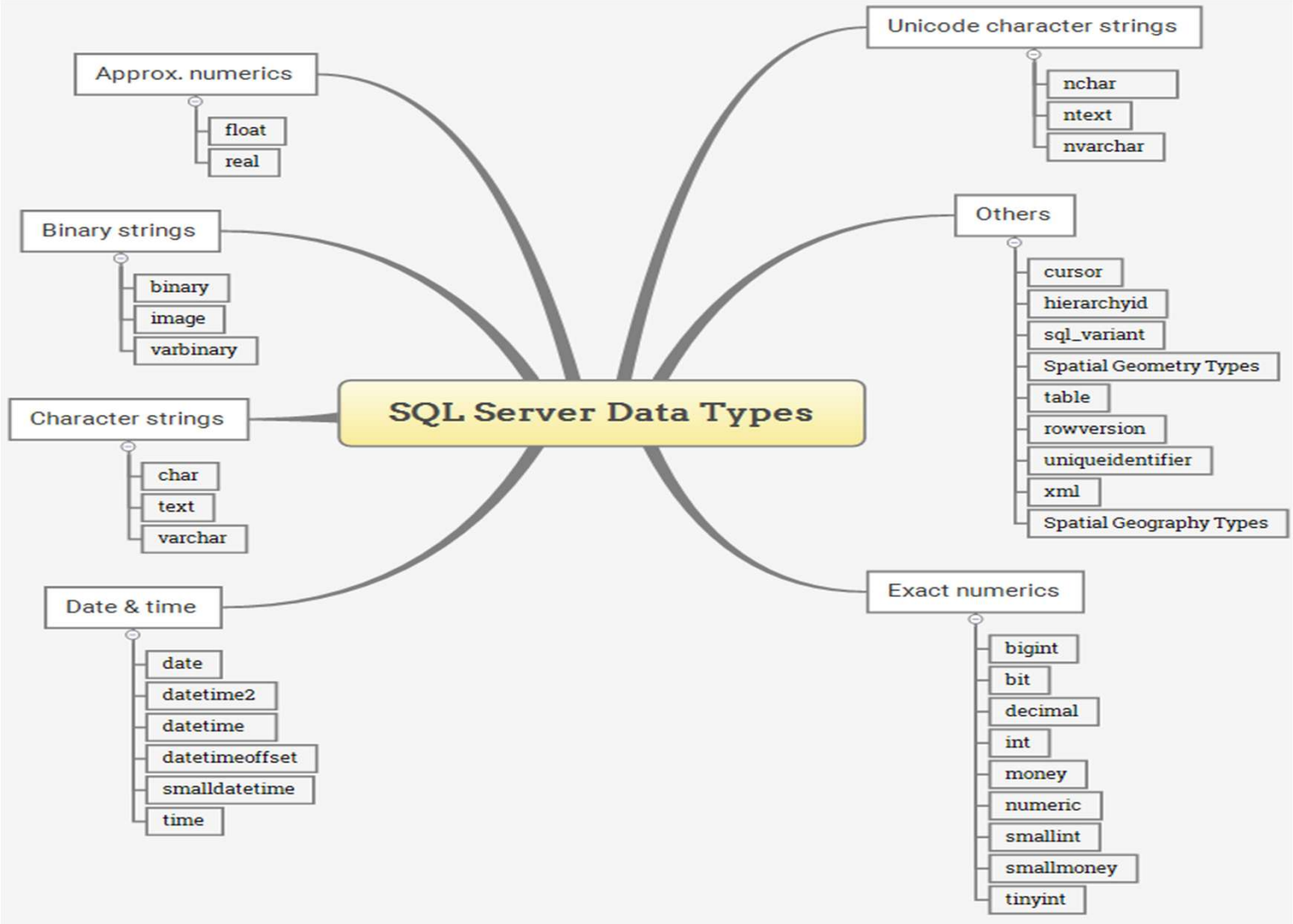
There are several scenarios in which **a database cannot be dropped** using the DROP DATABASE statement in MS SQL Server. Here are a few common scenarios:

- **Open connections:** If there are active connections to the database from user sessions or applications, you will not be able to drop the database. Make sure all connections to the database are closed before attempting to drop it.
- **System databases:** The system databases (such as master, model, msdb, and tempdb) cannot be dropped using the DROP DATABASE statement. These databases are essential for the functioning of the SQL Server instance.
- **Database in use:** If the database is currently in use by other processes or tasks, such as ongoing backups, restores, or replication, you will not be able to drop the database. Ensure that all operations involving the database are completed or stopped before attempting to drop it.

- **Database ownership:** If the login executing the DROP DATABASE statement is not the owner of the database or a member of the sysadmin fixed server role, the statement will be denied. You need to have the necessary permissions to drop the database.
- **Mirroring or Always On Availability Groups:** If the database is involved in database mirroring or is a part of an Always On Availability Group, you need to remove the mirroring or availability group configurations before dropping the database.
- **Encryption or Transparent Data Encryption (TDE):** If the database is encrypted or has Transparent Data Encryption (TDE) enabled, you cannot drop the database directly. You need to first disable the encryption or TDE before dropping the database.

Data Types

SqlServer Data Types



Create Table

TARGET DATABASE SCHEMA

Table Name: **Categories**

Field Name	Data Type	Constraints
Category_Id	int	PK
Category_Name	nvarchar(50)	NOT NULL

Table Name: **Brands**

Field Name	Data Type	Constraints
Brand_Id	int	PK
Brand_Name	nvarchar(50)	NOT NULL

Table Name: **Products**

Field Name	Data Type	Constraints
Product_Id	int	INDENTITY(100,1), PK
Product_Name	nvarchar(50)	NOT NULL
Category_Id	int	NOT NULL, FK
Brand_Id	int	NOT NULL, FK
List_Price	numeric(10,2)	NOT NULL

Table Name: **Customers**

Field Name	Data Type	Constraints
Customer_Id	int	INDENTITY(1,1), PK
First_Name	nvarchar(50)	NOT NULL
Last_Name	nvarchar(50)	NOT NULL
Name	AS(First_Name+ ` ’+ Last_Name)	
Email	nvarchar(100)	UNIQUE
Pin_Code	int	CHECK(Pin_Code>0), NOT NULL

Table Name: **Order_Details**

Field Name	Data Type	Constraints
Order_Id	int	PK
Order_Date	datetime	NOT NULL
Delivery_Date	Datetime	NOT NULL
Customer_Id	int	FK,NOT NULL

Table Name: **Order_Item_Info**

Field Name	Data Type	Constraints
Order_Id	int	FK , NOT NULL
Product_Id	int	FK, NOT NULL
Price	numeric(10,2)	NOT NULL
Qty	int	NOT NULL
Total	AS(Price*Qty)	

CREATE TABLE

In MS SQL Server, the CREATE TABLE statement is used to create a new table in a database. It defines the structure of the table, including the column names, data types, constraints, and any other properties associated with the table.

```
CREATE TABLE [database_name.][schema_name.]table_name (
   pk_column data_type PRIMARY KEY,
   column_1 data_type NOT NULL,
   column_2 data_type,
   ...,
   table_constraints
);
```

```
USE MYAUTODB
--HOW TO CREATE TABLES----
CREATE TABLE Categories(
Category_id          int             Primary Key,
Category_Name        nvarchar(50)    NOT NULL
)
```

```
CREATE TABLE Brands(
Brand_Id        int                 PRIMARY KEY,
Brand_Name      nvarchar(50)        NOT NULL
)
```

```
CREATE TABLE Products(
Product_Id          int                 IDENTITY(100,1) PRIMARY KEY,
Product_Name        nvarchar(50)        NOT NULL ,
Category_Id         int                 NOT NULL,
Brand_Id            int                 NOT NULL,
List_Price          numeric(10,2)       NOT NULL,
FOREIGN KEY(Category_Id) REFERENCES Categories(Category_id) ON DELETE CASCADE,
FOREIGN KEY(Brand_Id) REFERENCES Brands(Brand_id) ON DELETE CASCADE
)
```

```
CREATE TABLE Customers(
Customer_Id     int                                 IDENTITY(1,1)    PRIMARY KEY,
First_Name      nvarchar(50)                        NOT NULL,
Last_Name       nvarchar(50)                        NOT NULL,
Name            AS(first_name +' '+last_name),
Email           nvarchar(100)                       UNIQUE,
Pin_Code        int                                 CHECK(Pin_Code>0) NOT NULL
)
```

```
CREATE TABLE Order_Details(
Order_Id            int                             PRIMARY KEY,
Order_Date          datetime                        NOT NULL,
Delivery_date       datetime                        NOT NULL,
Customer_Id         int                             NOT NULL
FOREIGN KEY(Customer_Id) REFERENCES Customers(Customer_Id) ON DELETE CASCADE
)
```

```
CREATE TABLE Order_Item_Info(
Order_Id        int                                 NOT NULL,
Product_Id      int                                 NOT NULL,
Price           numeric(10,2)                       NOT NULL,
Qty             int                                 NOT NULL,
Total           AS(Price*Qty),
FOREIGN KEY(Order_Id) REFERENCES Order_Details(Order_Id) ON DELETE CASCADE,
FOREIGN KEY(Product_Id) REFERENCES Products(Product_Id) ON DELETE CASCADE
)
```

Constraints

- **Primary Key**

In MS SQL Server, a primary key constraint is used to uniquely identify each row in a table. It ensures that the values in the specified column or columns are unique and not null.

```
CREATE TABLE table_name (
    pk_column data_type PRIMARY KEY,
    ...
);
```

```
USE MYAUTODB
--HOW TO CREATE TABLES----
CREATE TABLE Categories(
Category_id          int             Primary Key,
Category_Name        nvarchar(50)    NOT NULL
)
```

Key Characteristics and benefits of using a primary key constraint:

- **Relationship:** A foreign key constraint defines a relationship between two tables, known as the parent table (referenced table) and the child table (referring table). The foreign key column(s) in the child table references the primary key column(s) in the parent table.

- **Referential integrity:** The foreign key constraint ensures referential integrity by preventing orphaned or inconsistent data. It guarantees that the values in the foreign key column(s) of the child table match valid values in the primary key column(s) of the parent table. If a foreign key value does not have a corresponding primary key value, the constraint will not allow the operation (such as an insert or update) to proceed.

Key Characteristics and benefits of using a primary key constraint:

- **Uniqueness:** A primary key constraint guarantees that each value in the specified column or columns is unique across all rows in the table. It ensures that no two rows can have the same value(s) for the primary key column(s).
- **Non-nullability:** A primary key constraint also enforces that the primary key column(s) cannot contain null values. This means that every row in the table must have a valid value for the primary key.
- **Indexing:** When a primary key constraint is defined on a column or set of columns, SQL Server automatically creates a clustered index on that column(s). This indexing helps improve the performance of queries that involve searching or sorting based on the primary key.

- **Cascading actions:** A foreign key constraint can also define cascading actions, such as ON DELETE CASCADE or ON UPDATE CASCADE. These actions specify what happens to the child table's records when the corresponding records in the parent table are deleted or updated. For example, if you delete a row from the parent table, the CASCADE action will automatically delete all related rows from the child table.

- **Foreign Key**

In MS SQL Server, a foreign key constraint is used to establish a relationship between two tables based on the values in a column or set of columns. It ensures referential integrity by enforcing that the values in the foreign key column(s) of one table correspond to the values in the primary key column(s) of another table.

To define a foreign key constraint in MS SQL Server, you can use the FOREIGN KEY keyword within the CREATE TABLE statement or modify an existing table using the ALTER TABLE statement.

```
CREATE TABLE child_table
--Column Schema
FOREIGN KEY (child_column(s))
REFERENCES parent_table (parent_column(s));
```

```
CREATE TABLE Products(
Product_Id          int                 IDENTITY(100,1) PRIMARY KEY,
Product_Name        nvarchar(50)        NOT NULL ,
Category_Id         int                 NOT NULL,
Brand_Id            int                 NOT NULL,
List_Price          numeric(10,2)       NOT NULL,
FOREIGN KEY(Category_Id) REFERENCES Categories(Category_id) ON DELETE CASCADE,
FOREIGN KEY(Brand_Id) REFERENCES Brands(Brand_id) ON DELETE CASCADE
)
```

- **Unique Constraint**

SQL Server UNIQUE constraints allow you to ensure that the data stored in a column, or a group of columns, is unique among the rows in a table.

```
ALTER TABLE table_name
ADD CONSTRAINT constraint_name
UNIQUE (column(s));
```

```
CREATE TABLE Customers(
Customer_Id     int                                  IDENTITY(1,1)    PRIMARY KEY,
First_Name      nvarchar(50)                         NOT NULL,
Last_Name       nvarchar(50)                         NOT NULL,
Name            AS(first_name +' '+last_name),
Email           nvarchar(100)                        UNIQUE,
Pin_Code        int                                  CHECK(Pin_Code>0) NOT NULL
)
```

UNIQUE constraint vs. PRIMARY KEY constraint

- Although both UNIQUE and PRIMARY KEY constraints enforce the uniqueness of data, you should use the UNIQUE constraint instead of PRIMARY KEY constraint when you want to enforce uniqueness of a column, or a group of columns, that are not the primary key columns.
- Different from PRIMARY KEY constraints, UNIQUE constraints allow NULL. Moreover, UNIQUE constraints treat the NULL as a regular value, therefore, it only allows one NULL per column.

- **NOT NULL Constraint**

Introduction to SQL Server NOT NULL constraint.

The SQL Server NOT NULL constraints simply specify that a column must not assume the NULL.

Alter Table

- **Add Column**

```
ALTER TABLE table_name
ADD column_name data_type column_constraint;
```

```
--ADD NEW COLUMN---
ALTER TABLE Products
ADD Pro_Description varchar(50) NULL
```

- **Modify Column**

  Modify column's data type

```
ALTER TABLE table_name
ALTER COLUMN column_name new_data_type(size);
```

```
--MODIFY A COLUMN--
ALTER TABLE Products
ALTER COLUMN Pro_Description varchar(30) NULL
```

- **Drop Column**

```
1 ALTER TABLE table_name
2 DROP COLUMN column_name;
```

```
---DROP A COLUMN--
ALTER TABLE Products
DROP COLUMN Pro_Description
```

- **Rename a Column**

```
--RENAME A COLUMN--

EXEC sp_rename 'Products.Product_Name','Pro_Name','COLUMN'
```

Drop Table

- **Drop Table**

To drop a table in MS SQL Server, you can use the DROP TABLE statement. This statement removes the entire table and all its data from the database.

Please exercise caution when using the DROP TABLE statement, as it permanently deletes the table and its data. Make sure you have a backup of the table or confirm that you intend to remove it before executing the statement.

```
DROP TABLE table_name;
```

```
DROP TABLE order_Items;
```

There are several scenarios in which a table cannot be dropped using the DROP TABLE statement in MS SQL Server. Here are a few common scenarios:

- **Dependencies:** If the table is referenced by other database objects such as views, stored procedures, functions, or foreign key constraints, you will not be able to drop the table until those dependencies are resolved or removed.
- **Permissions:** If you do not have the necessary permissions to drop the table, the DROP TABLE statement will be denied. Make sure you have the appropriate permissions, such as being a member of the sysadmin fixed server role or having the DROP permission on the table.
- **System tables:** Certain system tables, which are critical for the functioning of the database engine, cannot be dropped using the DROP TABLE statement. These system tables are protected and not meant to be modified or removed manually.

- **Replication:** If the table is involved in replication, you cannot directly drop the table using the DROP TABLE statement. You need to remove replication-related objects and configurations first before dropping the table.

- **In-use by active transactions:** If there are active transactions that have not been committed or rolled back and are accessing the table, the DROP TABLE statement will be blocked until those transactions are completed.

References

https://www.sqlshack.com/sql-ddl-getting-started-with-sql-ddl-commands-in-sql-server

https://www.sqlshack.com/sql-server-training/sql-server-basics/data-definition-language-ddl/

https://learn.microsoft.com/en-us/u-sql/data-definition-language-ddl-statements

https://www.geeksforgeeks.org/sql-ddl-dql-dml-dcl-tcl-commands/

https://learn.microsoft.com/en-us/training/modules/introduction-to-transact-sql/

https://www.techbrothersit.com/2016/01/how-to-generate-ddl-scripts-from-sql.html

https://www.javatpoint.com/ddl-commands-in-sql