MS SQL Server

Module 2

Data Definition Language Statements (DDL)

Module 2: Data Definition Language

- Lesson 1
  - Create, Drop and Select Database
  - Create and Drop Tables, Understanding Data Types
  - Inserting Values into the table
  - Modifying records in the table: Update Statement
- Lesson 2
  - Alter properties
  - Primary key and Foreign Key creation with Constraints
  - Truncate Table
- Lesson 3
  - Best Practices and key take away.

CREATE DATABASE

Syntax:

CREATE DATABASE *databasename*;

Example:

CREATE DATABASE dbCustomer;

SELECT a Database

In case multiple databases exists in **SQL** Schema, then before starting operation on DB, select a database where all the operations would be performed.

Syntax:

USE *databasename*;

Example:

USE dbCustomer;

DROP Database

Syntax:

DROP DATABASE *databasename*;

Example:

```
DROP DATABASE  dbCustomer;
```

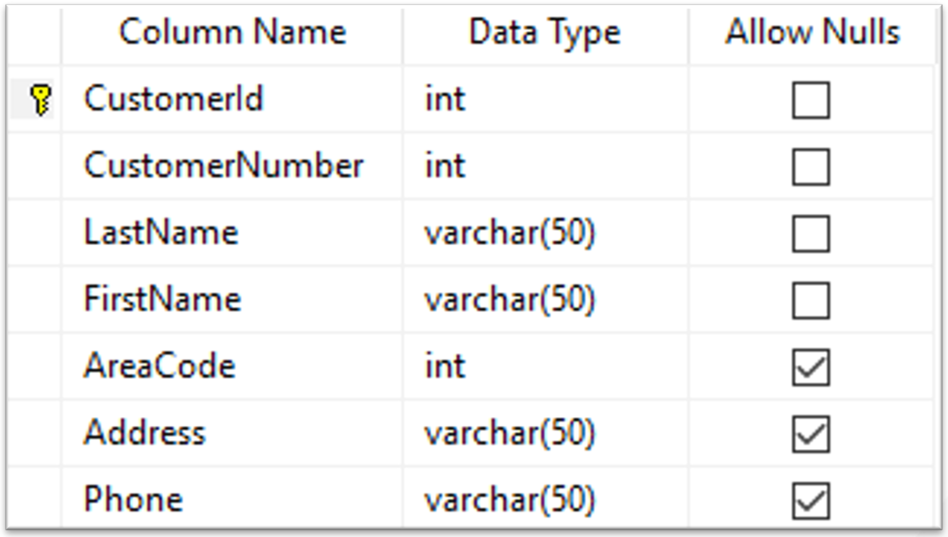
CREATE TABLE

Syntax:

```
CREATE TABLE tablename (
column1 datatype constraint,
column2 datatype constraint,
 .....
 columnN datatype constraint,
table constraint (one or more columns)
);
```

Example

Create a table “CUSTOMER” which has the following columns and data types.



	Column Name	Data Type	Allow Nulls
🔑	CustomerId	int	☐
	CustomerNumber	int	☐
	LastName	varchar(50)	☐
	FirstName	varchar(50)	☐
	AreaCode	int	☑
	Address	varchar(50)	☑
	Phone	varchar(50)	☑

```sql
CREATE TABLE CUSTOMER (
CustomerId int IDENTITY(1,1) PRIMARY KEY,
CustomerNumber int NOT NULL UNIQUE,
LastName varchar(50) NOT NULL,
FirstName varchar(50) NOT NULL,
AreaCode int NULL,
Address varchar(50) NULL,
Phone char(10) NULL DEFAULT '0000000000'
);
```

DROP TABLE

Syntax:

DROP TABLE *tablename;*

Example:

DROP TABLE *Sales;*

SQL Comments

Comments are used to explain sections of SQL statements, or to prevent execution of SQL statements.

Single Line Comments: Start with --

Any text between -- and the end of the line will be ignored (will not be executed).

Multi-line Comments: Start with /* and end with */

Any text between /* and */ will be ignored.

INSERT INTO

The INSERT INTO statement is used to insert a new row/s in a table

Syntax:

```
INSERT INTO table_name
        VALUES (value1, value2, value3,...)

INSERT INTO table_name (column1, column2, column3,...)
        VALUES (value1, value2, value3,...)
```

Example

```
INSERT INTO CUSTOMER VALUES ('100', 'Smith', 'John', 12, 'California', '11111111’);

INSERT INTO CUSTOMER
(CustomerNumber, LastName, FirstName, AreaCode, Address, Phone) VALUES
(‘101', 'Smith', 'John’, 14, 'California', '11111111’);

INSERT INTO CUSTOMER
(CustomerNumber, LastName, FirstName) VALUES
(‘102', 'Smith', 'John’);
```

*****You at least need to include all columns that cannot be NULL.***

Data types in MS SQL Server (not a comprehensive list)

Data type	Length	Description
bigint	8	Integer from -2^63 (-9 223 372 036 854 775 808) to 2^63-1 (9 223 372 036 854 775 807).
int	4	Integer from -2^31 (-2 147 483 648) to 2^31-1 (2 147 483 647).
smallint	2	Integer from -2^15 (-32 768) to 2^15-1 (32 767).
tinyint	1	Integer from 0 to 255.
bit	1 bit	Integer 0 or 1.
decimal(precision, scale)	5 - 17	Numeric data type with fixed precision and scale (accuracy 1-38, 18 by default and scale 0-p, 0 by default).
numeric	5 - 17	Same as data type 'decimal'.
money	8	Financial data type from -2^63 (-922 337 203 685 477.5808) to 2^63-1 (922 337 203 685 477.5807) with the precision of one ten-thousandth unit.
smallmoney	4	Financial data type from -2^31 (-214 748.3648) to 2^31-1 (214 748.3647) with the precision of one ten-thousandth unit.
datetime	8	Data type representing date and time from 1.1.1753 to 31.12.9999 with precision about 3ms. Values are rounded to .000, .003 and .007.
char	n	Char string of fixed length and max. length of 8000 chars.
varchar	n	Char string of variable length and max. length of 8000 chars.
text	n	Char string of variable length and max. length of 2^31-1 (2 147 483 647) chars.
nchar	2 * n	Unicode char string of fixed length and max. length of 4000 chars.
nvarchar	2 * n	Unicode char string of variable length and max. length of 4000 chars.

UPDATE

The UPDATE statement is used to update existing records in a table.

Syntax:

```
UPDATE table_name
SET column1 = value1, column2 = value2, ...
WHERE
some_column = some_value;
```

Notice the WHERE clause in the UPDATE syntax. The WHERE clause specifies which record or records should be updated.

If you omit the WHERE clause, all records will be updated!

Example

```
UPDATE CUSTOMER
SET
AreaCode = 46, LastName = ‘Fenn', FirstName = ‘John'
WHERE
CustomerId = 1;
```

Always include the WHERE clause when using the UPDATE command!

ALTER TABLE

The ALTER TABLE statement is used to add, delete, or modify columns in an existing table.

To add a column in a table, use the following syntax:

```
ALTER TABLE table_name ADD column_name datatype;
```

To delete a column in a table, use the following syntax:

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

To change the data type of a column in a table, use the following syntax:

```
ALTER TABLE table_name ALTER COLUMN column_name datatype;
```

SQL Constraints

Used to limit the type of data that can go into a table. This ensures the accuracy and reliability of the data in the table. Constraints can be column level or table level.

The following constraints are commonly used in SQL:

- NOT NULL - Ensures that a column cannot have a NULL value.
- UNIQUE - Ensures that all values in a column are different.
- PRIMARY KEY - A combination of a NOT NULL and UNIQUE. Uniquely identifies each row in a table.
- FOREIGN KEY - Uniquely identifies a row/record in another table.

Continued..

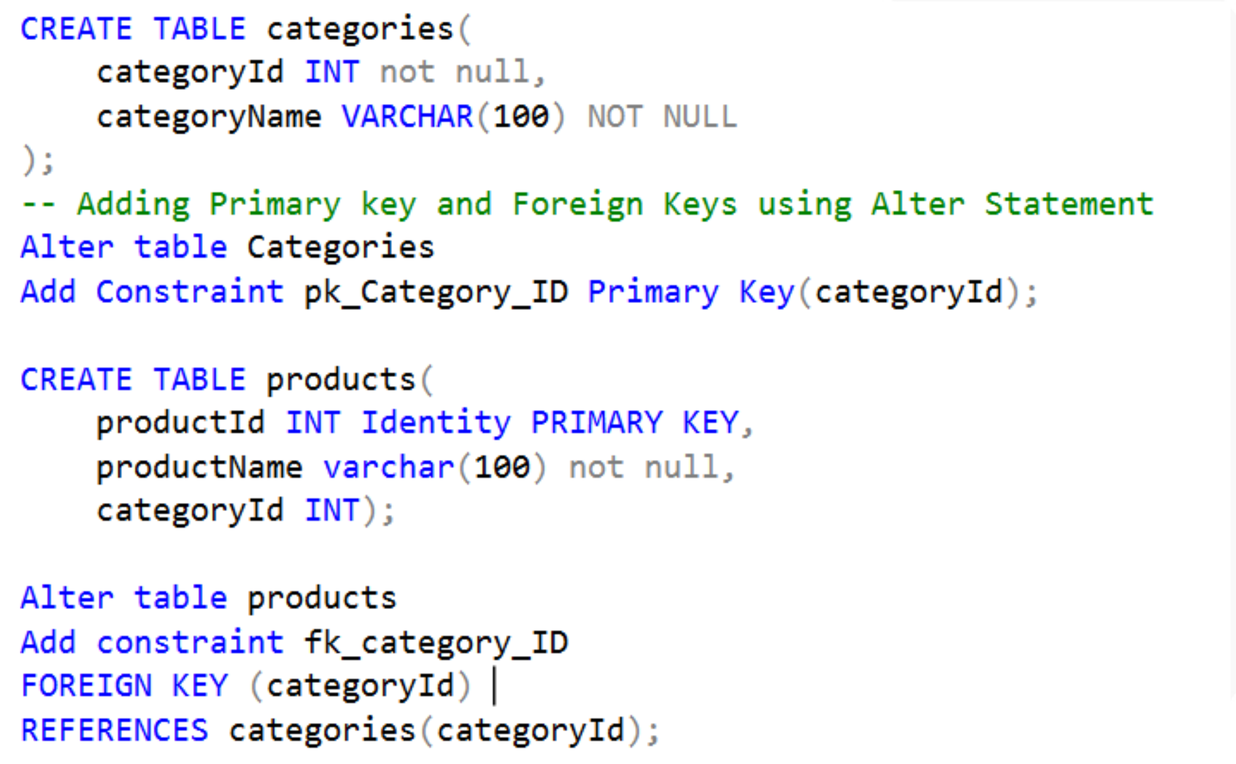
- CHECK - Ensures that all values in a column satisfies a specific condition.
- DEFAULT - Sets a default value for a column when no value is specified.
- IDENTITY - Value in the field would be auto generated by the system when we insert data in to the table.

Primary Key and Foreign key Creation

```
CREATE TABLE categories(
categoryId INT Identity PRIMARY KEY,
 categoryName VARCHAR(100) NOT NULL);

CREATE TABLE products(
productId INT Identity PRIMARY KEY,
productName varchar(100) not null, categoryId INT,
CONSTRAINT fk_category
FOREIGN KEY (categoryId)
REFERENCES categories(categoryId));
```

Adding Primary and Foreign Keys using Alter statements



```
CREATE TABLE categories(
    categoryId INT not null,
    categoryName VARCHAR(100) NOT NULL
);
-- Adding Primary key and Foreign Keys using Alter Statement
Alter table Categories
Add Constraint pk_Category_ID Primary Key(categoryId);

CREATE TABLE products(
    productId INT Identity PRIMARY KEY,
    productName varchar(100) not null,
    categoryId INT);

Alter table products
Add constraint fk_category_ID
FOREIGN KEY (categoryId)
REFERENCES categories(categoryId);
```

Truncate TABLE

Removes all the data from the table and retains the structure.

Syntax:

Truncate TABLE *tablename;*

Example:

Truncate Table *Categories*;

Note: A table cannot be truncated if it is being referenced by Foreign key constraints. To truncate such tables, you have to drop the constraint and then truncate the table

Best Practice

When creating tables you should consider following these guidelines:

- Use upper case and singular form in table names – not plural, e.g. CUSTOMER. Never use spaces inside the table names.
- Use Pascal notation for columns, e.g. “CustomerID”. Never use spaces inside the column names.
- Use Integer and Identity(1,1) columns for Primary Keys. Name primary key column the same as the table name + Id.
- Never Create Primary key when the table already has huge data, no matter how unique that column might be
- You cannot create primary key constraint on a nullable column.
- Avoid using alter column properties on tables already having data. It might result in the loss of Data.

Key Take away..

- Get familiar with creating tables
- Apply Alter properties on table
- Understand how to modify records in a table using Update statement
- Create Primary and Foreign key constraint
- Constraint enforcement
- Truncate table operation
- Best practices when using DDL statements