Intelligent Data Management with SQL Server

Session: 6

Creating Tables

Objectives

- Describe the procedure to create, modify, and drop tables in an SQL Server database
- Describe the procedure to add, modify, and drop columns in a table

Creating Tables

Syntax:

```
CREATE TABLE [database_name. [schema_name].| schema_name.]table_name
([<column_name>] [data_type] Null/Not Null,)
ON [filegroup | “default”]
```

where,

table_name: is the name of the new table, maximum of 128 characters.
column_name: is the name of a column in the table. up to 128 characters.
column_name are not specified for columns that are created with a timestamp data type. The default column name of a timestamp column is timestamp.

For example :

```
CREATE TABLE [dbo].[Customer_1](
[Customer_id number] [numeric](10, 0) NOT NULL,
[Customer_name] [varchar](50) NOT NULL)
ON [PRIMARY]
GO
```

Modifying Tables 1-2

- The ALTER TABLE statement is used to modify a table definition by altering, adding, or dropping columns and constraints, reassigning partitions, or disabling or enabling constraints and triggers.

Syntax:

```
ALTER TABLE [[database_name. [schema_name].| schema_name.]table_name
ALTER COLUMN ([<column_name>] [data_type] Null/Not Null,);
| DROP COLUMN ([<column_name>];
| ADD ([<column_name>] [data_type] Null/Not Null,);
```

where,

ALTER COLUMN: specifies that the particular column is to be changed or modified.

DROP COLUMN ([<column_name>]: specifies that column_name is to be removed from the table.

ADD: specifies that one or more column definitions are to be added.

Modifying Tables 2-2

- Following code snippet demonstrates altering the **Customer_id** column:

```
USE [CUST_DB]
ALTER TABLE [dbo].[Customer_1]
  ALTER Column [Customer_id number] [numeric](12, 0) NOT NULL;
```

- Following code snippet demonstrates adding the **Contact_number** column:

```
USE [CUST_DB]
ALTER TABLE [dbo].[Table_1]
  ADD [Contact_number] [numeric](12, 0) NOT NULL;
```

- Following code snippet demonstrates dropping the **Contact_number** column:

```
USE [CUST_DB]
ALTER TABLE [dbo].[Table_1]
  DROP COLUMN [Contact_name];
```

- Under certain conditions, columns cannot be dropped, such as, if they are used in a CHECK, FOREIGN KEY, UNIQUE, or PRIMARY KEY constraint, associated with a DEFAULT definition, and so forth.

Dropping Tables

- The DROP TABLE statement removes a table definition, its data, and all associated objects such as indexes, triggers, constraints, and permission specifications for that table.

Syntax:

```
DROP TABLE <Table_Name>
```

- For example:

```
USE [CUST_DB]
DROP TABLE [dbo].[Table_1]
```

Column Nullability

The nullability feature of a column determines whether rows in the table can contain a null value for that column.

Null value is not same as zero, blank, or a zero length character string (such as ' ').

Nullability can be defined either when creating a table or modifying a table.

When inserting a row, if no value is given for a nullable column, then, SQL Server automatically gives it a null value unless the column has been given a default value

- For example

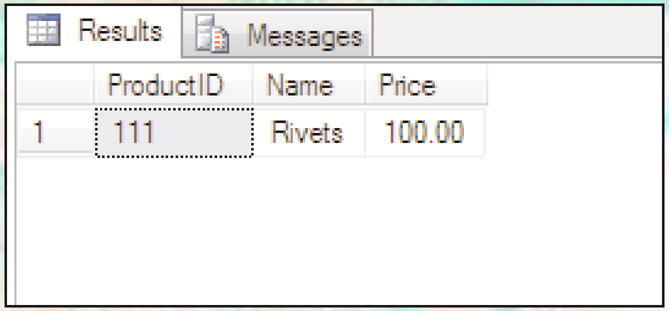
```
USE [CUST_DB]
CREATE TABLE StoreDetails ( StoreID int NOT NULL, Name varchar(40) NULL)
GO
```

DEFAULT Definition

assign a default value to the column if no value is given at the time of creation.

When a `DEFAULT` definition is added to an existing column, SQL Server applies the default values only to newly added rows of data.

- The following cannot be created on columns with DEFAULT definitions:
  - **A timestamp data type**
  - **An** IDENTITY **or** ROWGUIDCOL **property**
  - **An existing default definition or default object**



	ProductID	Name	Price
1	111	Rivets	100.00

- For example

```
USE [CUST_DB]
CREATE TABLE StoreProduct( ProductID int NOT NULL, Name varchar(40) NOT NULL, rice money NOT NULL DEFAULT (100))
GO
INSERT INTO dbo.StoreProduct (ProductID, Name) VALUES (111, 'Rivets')
GO
```

IDENTITY Property 1-3

- Is used to create columns that can contain auto-generated sequential values to uniquely identify each row within a table.
- Is often used for primary key values. The characteristics :

must be defined using one of the following data types: `decimal`**,** `int`**,** `numeric`**,** `smallint`**,** `bigint`**, or** `tinyint`**.**

need not have a seed and increment value specified. If they are not specified, a default value of 1 will be used for both.

A table cannot have more than one column with `IDENTITY` **property.**

must not allow null values and must not contain a `DEFAULT` **definition or object.**

cannot have their values updated.

The values can be explicitly inserted into the identity column only if the `IDENTITY_INSERT` **option is set** `ON`**.**

IDENTITY Property 2-3

- Once the IDENTITY property has been set, retrieving the values of the identifier column can be done by using the IDENTITYCOL keyword with the table name in a SELECT statement.
- To know if a table has an IDENTITY column, the OBJECTPROPERTY() function can be used.
- To retrieve the name of the IDENTITY column in a table, the COLUMNPROPERTY function is used.

Syntax:

```
CREATE TABLE <table_name> (
column_name data_type [ IDENTITY[(seed_value, increment_value)]] NOT NULL )
```

where,

seed_value: is the seed value from which to start generating identity values.

increment_value: is the increment value by which to increase each time.

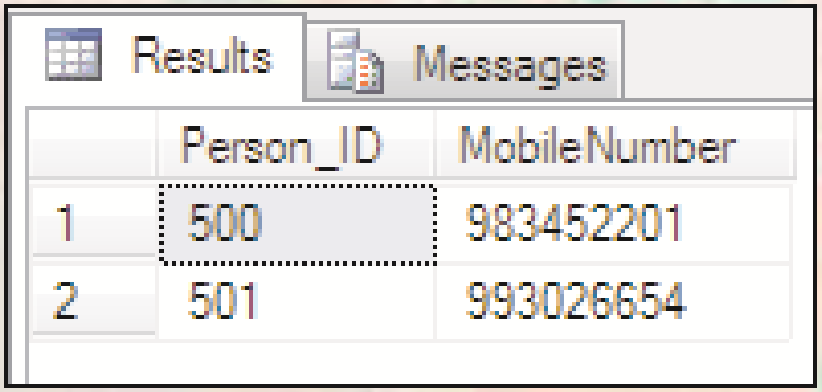
IDENTITY Property 3-3

- For example:

```
USE [CUST_DB]
GO
CREATE TABLE HRContactPhone ( Person_ID int IDENTITY(500,1) NOT NULL, MobileNumber
bigint NOT NULL )
GO

INSERT INTO HRContactPhone VALUES(983452201)
INSERT INTO HRContactPhone VALUES(993026654)
GO
```

- Following figure shows the output



	Person_ID	MobileNumber
1	500	983452201
2	501	993026654

Globally Unique Identifiers 1-3

Only one identifier column and one globally unique identifier column can be created for each table.

To create and work with globally unique identifiers, a combination of `ROWGUIDCOL`**,** `uniqueidentifier` **data type, and** `NEWID` **function are used.**

Values for a globally unique column are not automatically generated.

One has to create a `DEFAULT` **definition with a** `NEWID()` **function for a** `uniqueidentifier` **column to generate a globally unique value.**

Globally Unique Identifiers 2-3

The `NEWID()` function creates a unique identifier number which is a 16-byte binary string.

The column can be referenced in a `SELECT` list by using the `ROWGUIDCOL` keyword.

To know whether a table has a `ROWGUIDCOL` column, the `OBJECTPROPERTY` function is used.

The `COLUMNPROPERTY` function is used to retrieve the name of the `ROWGUIDCOL` column.

Globally Unique Identifiers 3-3

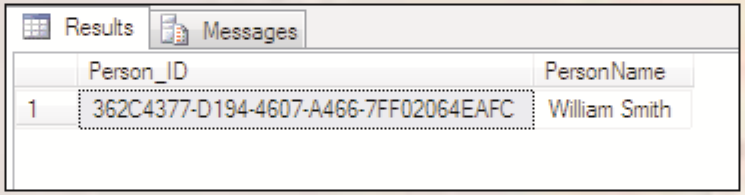
- For example

```
USE [CUST_DB]
CREATE TABLE EMP_CellularPhone( Person_ID uniqueidentifier DEFAULT NEWID() NOT NULL,
PersonName varchar(60) NOT NULL)
GO
```

- Following code snippet adds a value to **PersonName** column:

```
USE [CUST_DB]
INSERT INTO EMP_CellularPhone(PersonName) VALUES ('William Smith')
SELECT * FROM EMP_CellularPhone
GO
```

- Following figure shows the output where a unique identifier is displayed against a specific **PersonName**:



```
   Person_ID                              PersonName
1  362C4377-D194-4607-A466-7FF02064EAFC   William Smith
```

Constraints

- A constraint is a property assigned to a column or set of columns in a table to prevent certain types of inconsistent data values from being entered.

  - are used to apply business logic rules and enforce data integrity.
  - can be created when a table is created or added at a later stage
  - can be categorized as column constraints and table constraints.
  - A column constraint is specified as part of a column definition and applies only to that column.
  - A table constraint can apply to more than one column in a table and is declared independently from a column definition.
  - Table constraints must be used when more than one column is included in a constraint.

- SQL Server supports the following types of constraints:

  - PRIMARY KEY
  - UNIQUE
  - FOREIGN KEY
  - CHECK
  - NOT NULL

PRIMARY KEY

A table typically has a primary key comprising a single column or combination of columns to uniquely identify each row within the table.

The `PRIMARY KEY` constraint is used to create a primary key and enforce integrity of the entity of the table.

Only one primary key constraint can be created per table.

Column that is a primary key cannot have `NULL` values.

Syntax:

```
CREATE TABLE <table_name> ( Column_name datatype PRIMARY KEY [
column_list] )
```

```
CREATE TABLE <table_name> (<column_name> <datatype> [, column_list]
CONSTRAINT constraint_name PRIMARY KEY (column,..))
```

UNIQUE

- A UNIQUE constraint is used to ensure that only unique values are entered in a column or set of columns.
- UNIQUE constraints allow null values.
- A single table can have more than one UNIQUE constraint.

Syntax:

```
CREATE TABLE <table_name> ([column_list ] <column_name> <data_type> UNIQUE
[ column_list])
```

- For example:

```
USE [CUST_DB]

GO

CREATE TABLE EMP_ContactPhone(Person_ID int PRIMARY KEY, MobileNumber bigint
UNIQUE,ServiceProvider varchar(30),LandlineNumber bigint UNIQUE)
```

FOREIGN KEY

- **A foreign key in a table is a column that points to a primary key column in another table.**
- **Foreign key constraints are used to enforce referential integrity.**

Syntax:

```
CREATE TABLE <table_name1>([ column_list,] <column_name> <datatype>

FOREIGN KEY REFERENCES <table_name> (pk_column_name> [, column_list])
```

- For example

```
USE [CUST_DB]

GO

CREATE TABLE EMP_PhoneExpenses ( Expense_ID int PRIMARY KEY, MobileNumber
bigint FOREIGN KEY REFERENCES EMP_ContactPhone (MobileNumber), Amount
bigint)
```

CHECK

- A CHECK constraint limits the values that can be placed in a column.
- Check constraints enforce integrity of data.
- A CHECK constraint operates by specifying a search condition, which can evaluate to TRUE, FALSE, or unknown.
- Values that evaluate to FALSE are rejected.
- Multiple CHECK constraints can be specified for a single column.
- A single CHECK constraint can also be applied to multiple columns by creating it at the table level.
- Example:

```
USE [CUST_DB]
CREATE TABLE EMP_PhoneExpenses ( Expense_ID int PRIMARY KEY,
 MobileNumber bigint FOREIGN KEY REFERENCES EMP_ContactPhone
(MobileNumber), Amount bigint CHECK (Amount >10))
GO
```

NOT NULL

A `NOT NULL` constraint enforces that the column will not accept null values.

The `NOT NULL` constraints are used to enforce domain integrity, similar to `CHECK` constraints.

Data Modification Statements 1-3

- Used for modifying data, they are INSERT, UPDATE, and DELETE statements.

INSERT Statement

- adds a new row to a table.

Syntax:

```
INSERT [INTO] <Table_Name>
VALUES <values>
```

- For example

```
USE [CUST_DB]
INSERT INTO [dbo].[Table_2] VALUES (101, 'Richard Parker', 'Richy')
GO
```

Data Modification Statements 2-3

UPDATE Statement

- The UPDATE statement modifies the data in the table.

Syntax:

```
UPDATE <Table_Name>
SET <Column_Name = Value>
[WHERE <Search condition>]
```

where,

<Column_Name>: name of the column in which record is to be updated.
<Value>: specifies the new value for the modified column.
<Search condition>: the condition to be met for the rows to be updated.

- For examle:

```
USE [CUST_DB]
UPDATE [dbo].[Table_2] SET Contact_number = 5432679
 WHERE Contact_name LIKE 'Richy'
GO
```

Data Modification Statements 3-3

DELETE Statement

- The DELETE statement removes rows from a table.
- The syntax for DELETE statement is as follows:

Syntax:

```
DELETE FROM <Table_Name>
[WHERE <Search condition>]
```

Where,

The WHERE clause is used to specify the condition. If WHERE clause is not included in the DELETE statement, all the records in the table will be deleted.

- For example:

```
USE [CUST_DB]
DELETE FROM [dbo].[Customer_2] WHERE Contact_number = 5432679
GO
```

Summary

- Most tables have a primary key, made up of one or more columns of the table that identifies records uniquely.
- The nullability feature of a column determines whether rows in the table can contain a null value for that column.
- A DEFAULT definition for a column can be created at the time of table creation or added at a later stage to an existing table.
- The IDENTITY property of SQL Server is used to create identifier columns that can contain auto-generated sequential values to uniquely identify each row within a table.
- Constraints are used to apply business logic rules and enforce data integrity.
- A UNIQUE constraint is used to ensure that only unique values are entered in a column or set of columns.
- A foreign key in a table is a column that points to a primary key column in another table.
- A CHECK constraint limits the values that can be placed in a column.