Data Management Using Microsoft SQL Server

Session: 7

Creating Tables

Objectives

- List SQL Server 2012 data types
- 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

Introduction

- One of the most important types of database objects in SQL Server 2012 is a table.
- Tables in SQL Server 2012 contain data in the form of rows and columns.
- Each column may have data of a specific type and size.

Data Type

A data type is an attribute that specifies the type of data an object can hold, such as numeric data, character data, monetary data, and so on.

Once a column has been defined to store data belonging to a particular data type, data of another type cannot be stored in it.

Hence, if an attempt is made to enter character data into an integer column, it will not succeed.

Different Kinds of Data Types 1-6

- SQL Server 2012 supports three kinds of data types:

System data types

- These are provided by SQL Server 2012.

Alias data types

- These are based on the system-supplied data types.
- One of the typical uses of alias data types is when more than one table stores the same type of data in a column and has similar characteristics such as length, nullability, and type.
- In such cases, an alias data type can be created that can be used commonly by all these tables.

User-defined types

- These are created using programming languages supported by the .NET Framework, which is a software framework developed by Microsoft.

Different Kinds of Data Types 2-6

- Following table shows various data types in SQL Server 2012 along with their categories and description:

Category	Data Type	Description
Exact Numerics	int	Represents a column that occupies 4 bytes of memory space. Is typically used to hold integer values.
	smallint	Represents a column that occupies 2 bytes of memory space. Can hold integer data from –32,768 to 32,767.
	tinyint	Represents a column that occupies 1 byte of memory space. Can hold integer data from 0 to 255.

Different Kinds of Data Types 3-6

Category	Data Type	Description
Exact Numerics	bigint	Represents a column that occupies 8 bytes of memory space. Can hold data in the range -2^63 (-9,223,372,036,854,775,808) to 2^63-1 (9,223,372,036,854,775,807)
	numeric	Represents a column of this type that fixes precision and scale.
	money	Represents a column that occupies 8 bytes of memory space. Represents monetary data values ranging from (–2^63/10000) (–922,337,203,685,477.5808) through 2^63–1 (922,337,203,685,477.5807).

Different Kinds of Data Types 4-6

Category	Data Type	Description
Approximate Numerics	float	Represents a column that occupies 8 bytes of memory space. Represents floating point number ranging from –1.79E +308 through 1.79E+308.
	real	Represents a column that occupies 4 bytes of memory space. Represents floating precision number ranging from –3.40E+38 through 3.40E+38.
Date and Time	datetime	Represents date and time. Stored as two 4-byte integers.
	smalldatetime	Represents date and time.
Character String	char	Stores character data that is fixed-length and non-Unicode.
	varchar	Stores character data that is variable-length and non-Unicode.
	text	Stores character data that is variable-length and non-Unicode.
Unicode Types	nchar	Stores Unicode character data of fixed-length.
	nvarchar	Stores variable-length Unicode character data.

Different Kinds of Data Types 5-6

Category	Data Type	Description
Other Data Types	Timestamp	Represents a column that occupies 8 bytes of memory space. Can hold automatically generated, unique binary numbers that are generated for a database.
	binary(n)	Stores fixed-length binary data with a maximum length of 8000 bytes.
Other Data Types	varbinary(n)	Stores variable-length binary data with a maximum length of 8000 bytes.
	image	Stores variable-length binary data with a maximum length of 2^30–1 (1,073,741,823) bytes.
	uniqueidentifier	Represents a column that occupies 16 bytes of memory space. Also, stores a globally unique identifier (GUID).

Different Kinds of Data Types 6-6

- Alias data types can be created using the `CREATE TYPE` statement.
- The syntax for the `CREATE TYPE` statement is as follows:

Syntax:

```
CREATE TYPE[ schema_name.] type_name{FROM base_type[(
precision[,scale])][NULL|NOT NULL]}[;]
```

where,

`schema_name`: identifies the name of the schema in which the alias data type is being created.

`type_name`: identifies the name of the alias type being created.

`base_type`: identifies the name of the system-defined data type based on which the alias data type is being created.

`precision and scale`: specify the precision and scale for numeric data.

`NULL|NOT NULL`: specifies whether the data type can hold a null value or not.

- Following code snippet shows how to create an alias data type named **`usertype`** using the `CREATE TYPE` statement:

```
CREATE TYPE usertype FROM varchar(20) NOT NULL
```

Creating Tables 1-2

- The `CREATE TABLE` statement is used to create tables in SQL Server 2012.
- The syntax for `CREATE TABLE` statement is as follows:

Syntax:

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

where,

`database_name`: is the name of the database in which the table is created.

`table_name`: is the name of the new table. `table_name` can be a maximum of 128 characters.

`column_name`: is the name of a column in the table. `column_name` can be 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.

`data_type`: It specifies data type of the column.

Creating Tables 2-2

- Following code snippet demonstrates creation of a table named **dbo.Customer_1**:

```
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.
- The syntax for `ALTER TABLE` statement is as follows:

Syntax:

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

where,

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

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

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

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.
- The syntax for `DROP TABLE` statement is as follows:

Syntax:

```
DROP TABLE <Table_Name>
```

where,

`<Table_Name>`: is the name of the table to be dropped.

- Following code snippet demonstrates how to drop a table:

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

Data Modification Statements 1-4

- The statements used for modifying data are `INSERT`, `UPDATE`, and `DELETE` statements.
- These are explained as follows:

INSERT Statement

- The `INSERT` statement adds a new row to a table.
- The syntax for `INSERT` statement is as follows:

Syntax:

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

where,

`<Table_Name>`: is the name of the table in which row is to be inserted.

`[INTO]`: is an optional keyword used between `INSERT` and the target table.

`<Values>`: specifies the values for columns of the table.

Data Modification Statements 2-4

- Following code snippet demonstrates adding a new row to the **Table_2** table:

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

- The outcome of this will be that one row with the given data is inserted into the table.

UPDATE Statement

- The UPDATE statement modifies the data in the table.
- The syntax for UPDATE statement is as follows:

Syntax:

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

where,

<Table_Name>: is the name of the table where records are to be updated.

<Column_Name>: is the name of the column in the table in which record is to be updated.

Data Modification Statements 3-4

<Value>: specifies the new value for the modified column.
<Search condition>: specifies the condition to be met for the rows to be deleted.

- Following code snippet demonstrates the use of the UPDATE statement to modify the value in column **Contact_number**:

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

- Following figure shows the output of UPDATE statement:

Results | Messages

	Customer_id number	Customer_name	Contact_name	Contact_number
1	101	Richard Parker	Richy	5432679

Data Modification Statements 4-4

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,

<Table_Name>: is the name of the table from which the records are to be deleted.

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.

- Following code snippet demonstrates how to delete a row from the **Customer_2** table whose **Contact_number** value is **5432679**:

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

Column Nullability 1-2

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

Product table of the **AdventureWorks2012** database does not mean that the product has no color; it just means that the color for the product is unknown or has not been set.

The `NULL` keyword is used to indicate that null values are allowed in the column, and the `NOT NULL` keywords are used to indicate that null values are not allowed.

Column Nullability 2-2

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 definition.

Making a column non-nullable enforces data integrity by ensuring that the column contains data in every row.

- In the following code snippet, the `CREATE TABLE` statement uses the `NULL` and `NOT NULL` keywords with column definitions:

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

- The result of the code is that the **`StoreDetails`** table is created with **`StoreID`** and **`Name`** columns added to the table.

DEFAULT Definition 1-3

A DEFAULT definition can be given for the column to assign it as a default value if no value is given at the time of creation.

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.

DEFAULT Definition 2-3

- In the following code snippet, the `CREATE TABLE` statement uses the `DEFAULT` keyword to define the default value for `Price`:

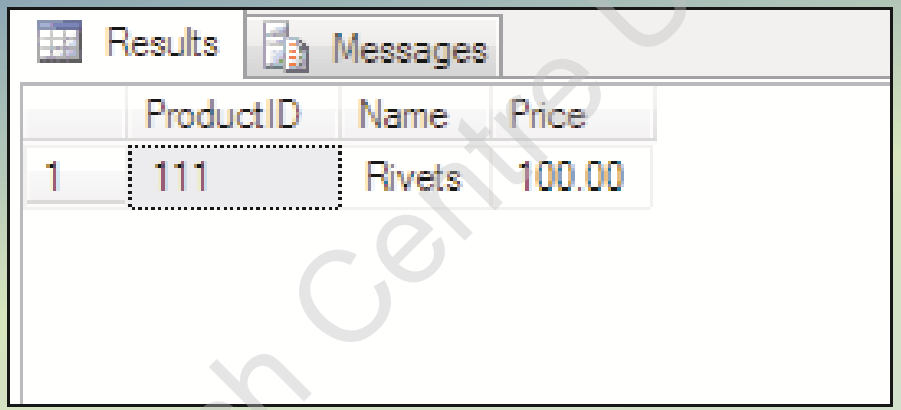
```
USE [CUST_DB]
CREATE TABLE StoreProduct( ProductID int NOT NULL, Name varchar(40) NOT NULL, Price money NOT NULL DEFAULT (100))
GO
```

- When a row is inserted using a statement as shown in the following code snippet, the value of **`Price`** will not be blank; it will have a value of **`100.00`** even though a user has not entered any value for that column.

```
USE [CUST_DB]
INSERT INTO dbo.StoreProduct (ProductID, Name) VALUES (111, 'Rivets')
GO
```

DEFAULT Definition 3-3

- Following figure shows the output, where though values are added only to the **ProductID** and **Name** columns, the **Price** column will still show a value of **100.00**.
- This is because of the DEFAULT definition.



- 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**

IDENTITY Property 1-4

- 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.
- An identity column is often used for primary key values. The characteristics of the IDENTITY property are as follows:

- A column having IDENTITY property must be defined using one of the following data types: decimal, int, numeric, smallint, bigint, or tinyint.
- A column having IDENTITY property 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.
- The identifier column in a table must not allow null values and must not contain a DEFAULT definition or object.
- Columns defined with IDENTITY property cannot have their values updated.
- The values can be explicitly inserted into the identity column of a table only if the IDENTITY_INSERT option is set ON.
- When IDENTITY_INSERT is ON, INSERT statements must supply a value.

IDENTITY Property 2-4

- Once the IDENTITY property has been set, retrieving the value 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.

- The syntax to add a IDENTITY property while creating a table is as follows:

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 3-4

- Following code snippet demonstrates the use of `IDENTITY` property:

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

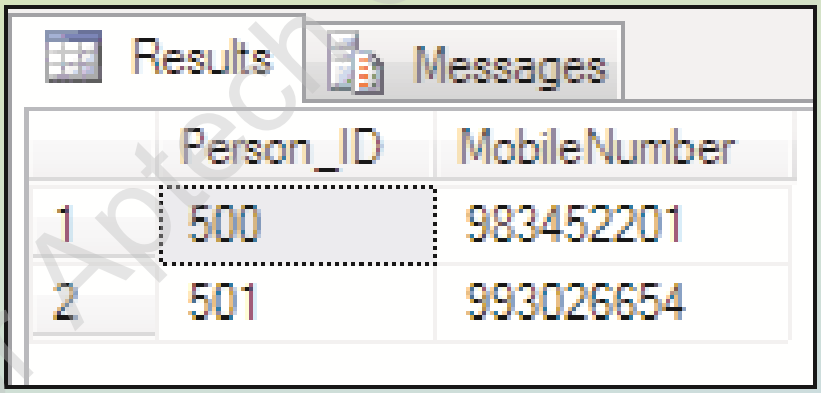
- **HRContactPhone** is created as a table with two columns in the schema **Person** that is available in the **CUST_DB** database.
- The **Person_ID** column is an identity column.
- The seed value is **500**, and the increment value is **1 .**
- While inserting rows into the table, if `IDENTITY_INSERT` is not turned on, then, explicit values for the `IDENTITY` column cannot be given.

IDENTITY Property 4-4

- Instead, statements similar to the following code snippet can be given:

```
USE [CUST_DB]
INSERT INTO HRContactPhone (MobileNumber) VALUES(983452201)
INSERT INTO HRContactPhone (MobileNumber) VALUES(993026654)
GO
```

- Following figure shows the output where `IDENTITY` property is incrementing **`Person_ID`** column values:



Globally Unique Identifiers 1-3

In addition to the `IDENTITY` property, SQL Server also supports globally unique identifiers.

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

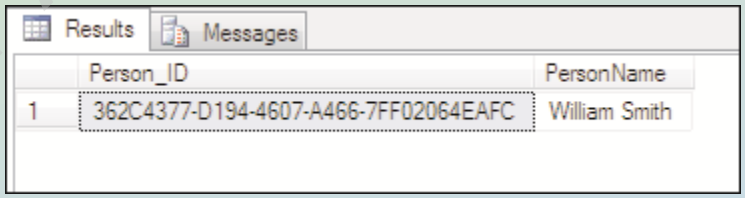
- Following code snippet demonstrates how to `CREATE TABLE` statement to create the **`EMPCellularPhone`** table.
- The **`Person_ID`** column automatically generates a `GUID` for each new row added to the table.

```
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`**:



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.

Constraints are used to apply business logic rules and enforce data integrity.

Constraints can be created when a table is created or added at a later stage using the `ALTER TABLE` statement.

Constraints 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 1-3

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

Only one primary key constraint can be created per table.

- The syntax to add a primary key while creating a table is as follows:

Syntax:

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

PRIMARY KEY 2-3

- Following code snippet demonstrates how to create a table **EMPContactPhone** to store the contact telephone details of a person.
- Since the column **EMP_ID** must be a primary key for identifying each row uniquely, it is created with the primary key constraint.

```
USE [CUST_DB]
CREATE TABLE EMPContactPhone ( EMP_ID int PRIMARY KEY, MobileNumber
bigint, ServiceProvider varchar(30), LandlineNumber bigint)
GO
```

- An alternative approach is to use the CONSTRAINT keyword. The syntax is as follows:

Syntax:

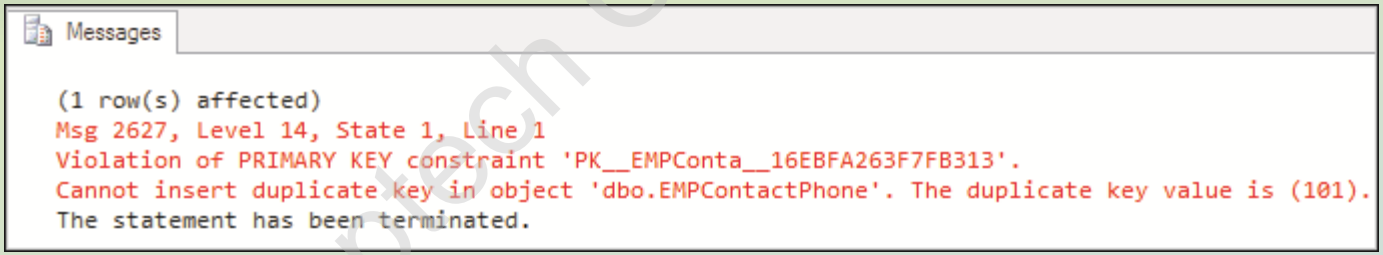
```
CREATE TABLE <table_name> (<column_name> <datatype> [, column_list]
CONSTRAINT constraint_name PRIMARY KEY)
```

PRIMARY KEY 3-3

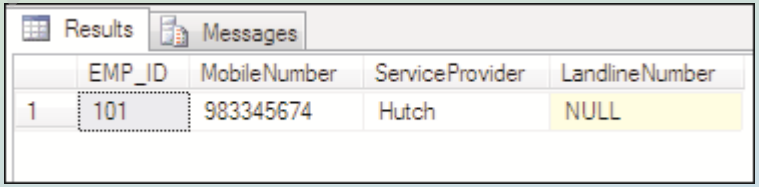
- Having created a primary key for EMP_ID, a query is written to insert rows into the table with the statements shown in the following code snippet:

```
USE [CUST_DB]
INSERT INTO dbo.EMPContactPhone values (101, 983345674,'Hutch', NULL)
INSERT INTO dbo.EMPContactPhone values (102, 989010002,'Airtel', NULL)
GO
```

- The first statement shown in the code snippet is executed successfully but the next INSERT statement will fail because the value for EMP_ID is duplicate as shown in the following figure:



- The output is shown in the following figure:



UNIQUE 1-2

- 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.
- The syntax to create UNIQUE constraint is as follows:

Syntax:

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

- Following code snippet demonstrates how to make the **MobileNumber** and **LandlineNumber** columns as unique:

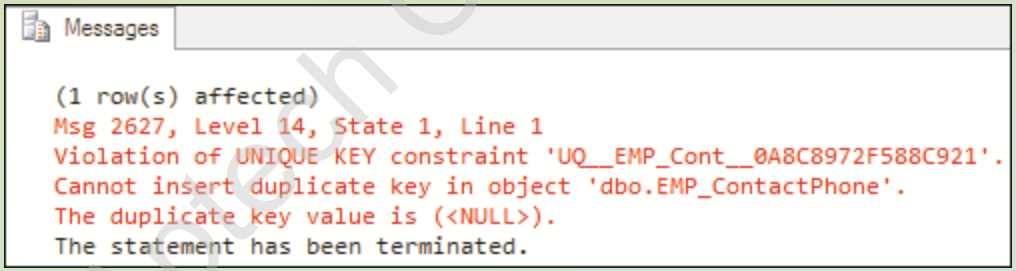
```
USE [CUST_DB]
GO
CREATE TABLE EMP_ContactPhone(Person_ID int PRIMARY KEY, MobileNumber
bigint UNIQUE,ServiceProvider varchar(30),LandlineNumber bigint UNIQUE)
```

UNIQUE 2-2

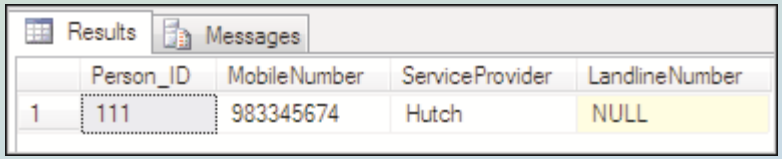
- Following code snippet demonstrates how to insert a row into the table:

```
USE [CUST_DB]
INSERT INTO EMP_ContactPhone values (111, 983345674, 'Hutch', NULL)
INSERT INTO EMP_ContactPhone values (112, 983345674, 'Airtel', NULL)
GO
```

- UNIQUE constraints check only for the uniqueness of values but do not prevent null entries.
- The second INSERT statement will fail because the value for **MobileNumber** is a duplicate as shown in the following figure:



- This is because the column **MobileNumber** is defined to be unique and disallows duplicate values. The output is shown in the following figure:



FOREIGN KEY 1-2

- 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.
- The syntax for foreign key is as follows:

Syntax:

```
CREATE TABLE <table_name1>([ column_list,] <column_name> <datatype>
FOREIGN KEY REFERENCES <table_name> (pk_column_name> [, column_list])
```

where,

`table_name`: is the name of the table from which to reference primary key.

`<pk_column_name>`: is the name of the primary key column.

- Following code snippet demonstrates how to create a foreign key constraint:

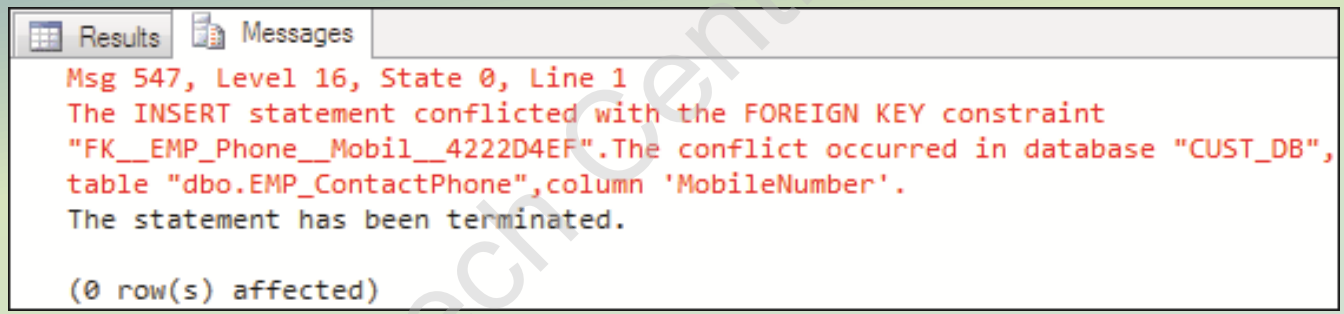
```
USE [CUST_DB]
GO
CREATE TABLE EMP_PhoneExpenses ( Expense_ID int PRIMARY KEY,
MobileNumber bigint FOREIGN KEY REFERENCES EMP_ContactPhone
(MobileNumber), Amount bigint)
```

FOREIGN KEY 2-2

- A row is inserted into the table such that the mobile number is the same as one of the mobile numbers in **EMP_ContactPhone**.
- The command that will be written is shown in the following code snippet:

```
INSERT INTO dbo.EMP_PhoneExpenses values(101, 993026654, 500)
SELECT * FROM dbo.EMP_PhoneExpenses
```

- The error message of the code snippet is shown in the following figure:



- If there is no key in the referenced table having a value that is being inserted into the foreign key, the insertion will fail as shown in the figure.

- It is, however, possible to add `NULL` value into a foreign key column.

CHECK 1-2

- 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.
- Following code snippet demonstrates creating a CHECK constraint to ensure that the **Amount** value will always be non-zero:

```
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
```

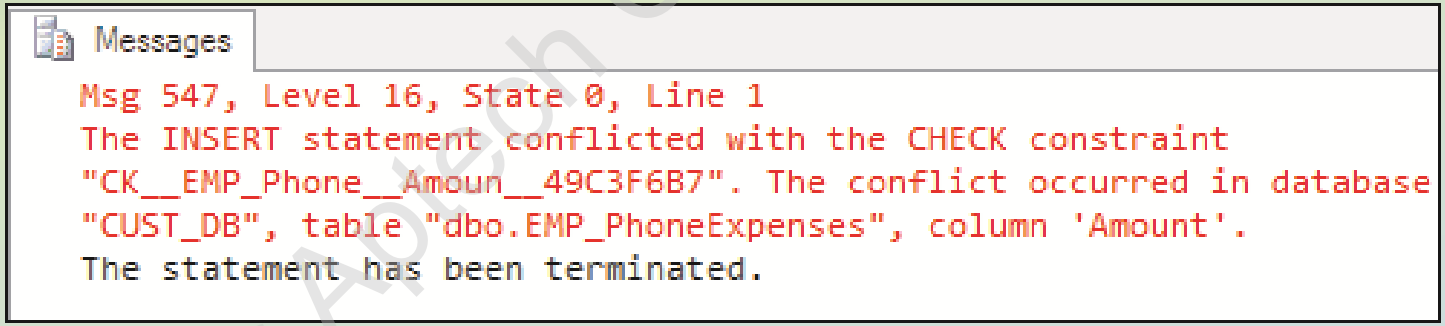
- A NULL value can, however, be added into **Amount** column if the value of **Amount** is not known.

CHECK 2-2

- Once a CHECK constraint has been defined, if an INSERT statement is written with data that violates the constraint, it will fail as shown in the following code snippet:

```
USE [CUST_DB]
INSERT INTO dbo.EMP_PhoneExpenses values (101, 983345674, 9)
GO
```

- The error message of the code snippet that appears when the **Amount** constraint is less than **10** is shown in the following figure:



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.

Summary 1-2

- A data type is an attribute that specifies the storage capacity of an object and the type of data it can hold, such as numeric data, character data, monetary data, and so on.

- SQL Server 2012 supports three kinds of data types:
  - System data types
  - Alias data types
  - User-defined types

- 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.

Summary 2-2

- 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.