SQL SERVER

Session Day-1

Creating Database and Tables

About the Author

Created By: Mohammad Salman

Experience: 17 Years +

Designation: Corporate Trainer .NET

Microsoft CERTIFIED Professional

Microsoft Specialist

Microsoft CERTIFIED Solutions Developer

Objectives

- Describe system and user-defined databases
- List SQL Server data types
- Describe the procedure to create, modify, and drop tables in an SQL Server database

Introduction 1-3

- A database is a collection of data stored in data files on a disk or some removable medium.
- A database consists of data files to hold actual data.
- An SQL Server database is made up of a collection of tables that stores sets of specific structured data.
- A table includes a set of rows (also called as records or tuples) and columns (also called as attributes).
- Each column in the table is intended to store a specific type of information, for example, dates, names, currency amounts, and numbers.
- A user can install multiple instances of SQL Server on a computer.
- Each instance of SQL Server can include multiple databases.
- Within a database, there are various object ownership groups called schemas.

Introduction 2-3

- Within each schema, there are database objects such as tables, views, and stored procedures.
- Some objects such as certificates and asymmetric keys are contained within the database, but are not contained within a schema.
- SQL Server databases are stored as files in the file system.
- These files are grouped into file groups.
- When people gain access to an instance of SQL Server, they are identified as a login.
- When people gain access to a database, they are identified as a database user.
- A user who has access to a database can be given permission to access the objects in the database.

Introduction 3-3

- Though permissions can be granted to individual users, it is recommended to create database roles, add the database users to the roles, and then, grant access permission to the roles.
- Granting permissions to roles instead of users makes it easier to keep permissions consistent and understandable as the number of users grow and continually change.
- SQL Server 2012 supports three kinds of databases, which are as follows:

System Databases

- SQL Server uses system databases to support different parts of the DBMS.
- Each database has a specific role and stores job information that requires to be carried out by SQL Server.
- The system databases store data in tables, which contain the views, stored procedures, and other database objects.
- They also have associated database files (for example, `.mdf` and `.ldf` files) that are physically located on the SQL Server machine.
- Following table shows the system databases that are supported by SQL Server 2012:

Database	Description
master	The database records all system-level information of an instance of SQL Server.
msdb	The database is used by SQL Server Agent for scheduling database alerts and various jobs.
model	The database is used as the template for all databases to be created on the particular instance of SQL Server 2012.
resource	The database is a read-only database. It contains system objects included with SQL Server 2012.
tempdb	The database holds temporary objects or intermediate result sets.

SQL Server 2016

Modifying System Data 1-3

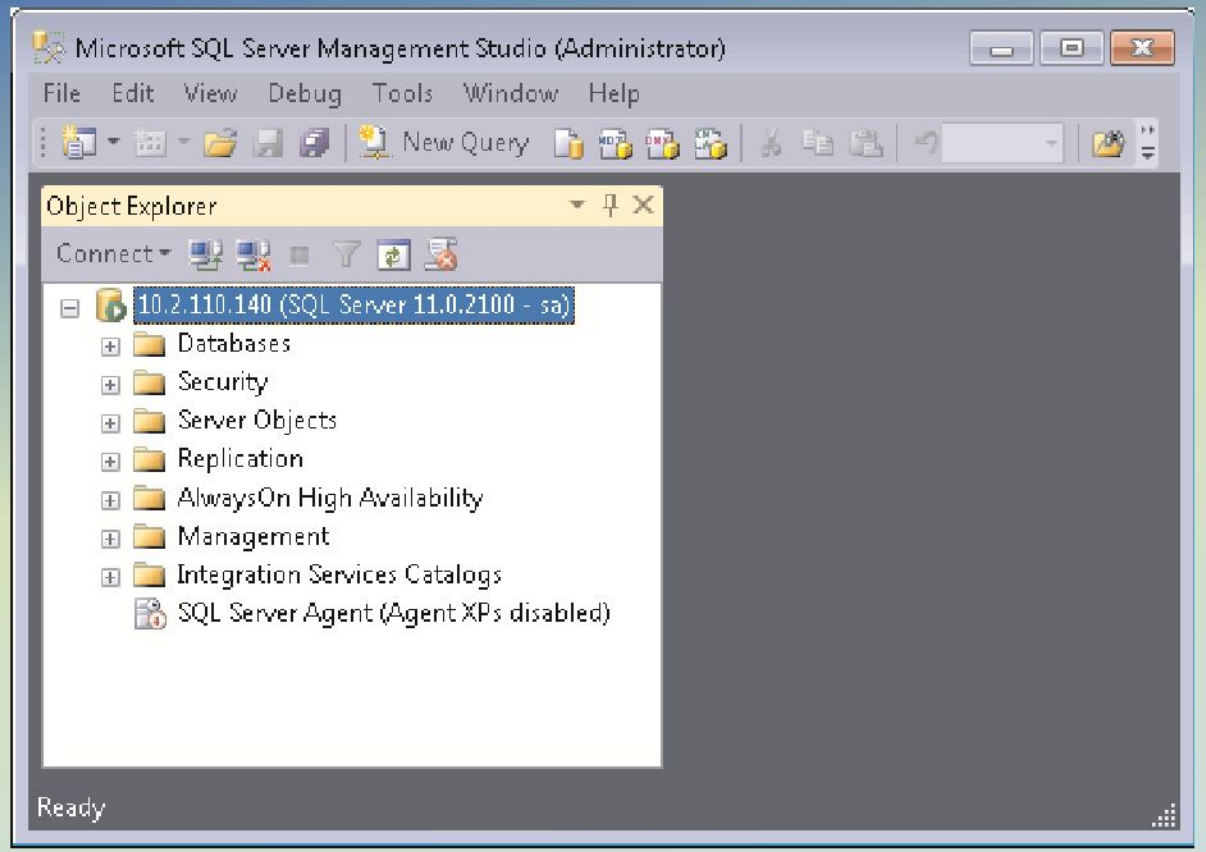
- Users are not allowed to directly update the information in system database objects, such as system tables, system stored procedures, and catalog views.
- However, users can avail a complete set of administrative tools allowing them to fully administer the system and manage all users and database objects.
- These are as follows:

Administration Utilities:

- From SQL Server 2005 onwards, several SQL Server administrative utilities are integrated into SSMS.
- It is the core administrative console for SQL Server installations.
- It enables to perform high-level administrative functions, schedule routine maintenance tasks, and so forth.

Modifying System Data 2-3

- Following figure shows the SQL Server 2012 Management Studio window:



SQL Server Management Objects (SQL-SMO) API:

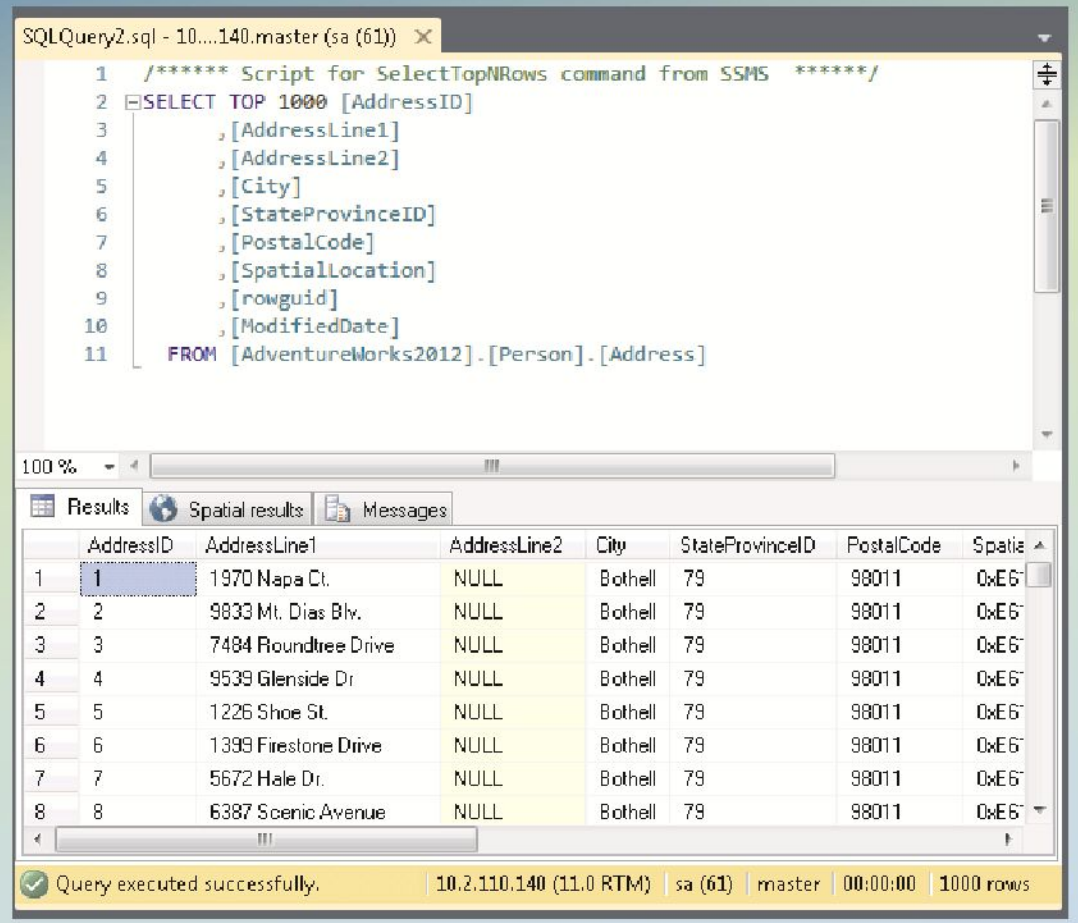
- Includes complete functionality for administering SQL Server in applications.

SQL Server 2016

Modifying System Data 3-3

Transact-SQL scripts and stored procedures:

- These use system stored procedures and Transact-SQL DDL statements. Following figure shows a Transact-SQL query window:



SQL Server 2016

Viewing System Database Data

- Database applications can determine catalog and system information by using any of these approaches:

System catalog views

- Views displaying metadata for describing database objects in an SQL Server instance.

SQL-SMO

- New managed code object model, providing a set of objects used for managing Microsoft SQL Server.

Catalog functions, methods, attributes, or properties of the data API

- Used in ActiveX Data Objects (ADO), OLE DB, or ODBC applications.

Stored Procedures and Functions

- Used in Transact-SQL as stored procedures and built-in functions.

SQL Server 2016

Creating Databases 1-4

- To create a user-defined database, the information required is as follows:

1. Name of the database
2. Owner or creator of the database
3. Size of the database
4. Files and filegroups used to store it

SQL Server 2016

Creating Databases 2-4

- The syntax to create a user-defined database is as follows:

Syntax:

```
CREATE DATABASE DATABASE_NAME
[ ON
[ PRIMARY ] [ <filespec> [ ,...n ]
[ , <filegroup> [ ,...n ] ]
[ LOG ON { <filespec> [ ,...n ] } ]
]
[ COLLATE collation_name ]
]
[;]
```

where,

`DATABASE_NAME`: is the name of the database to be created.

`ON`: indicates the disk files to be used to store the data sections of the database and data files.

`PRIMARY`: is the associated `<filespec>` list defining the primary file.

`<filespec>`: controls the file properties.

`<filegroup>`: controls filegroup properties.

Creating Databases 3-4

`LOG ON`: indicates disk files to be used for storing the database log and log files.
`COLLATE collation_name`: is the default collation for the database. A collation defines rules for comparing and sorting character data based on the standard of particular language and locale. Collation name can be either a Windows collation name or a SQL collation name.

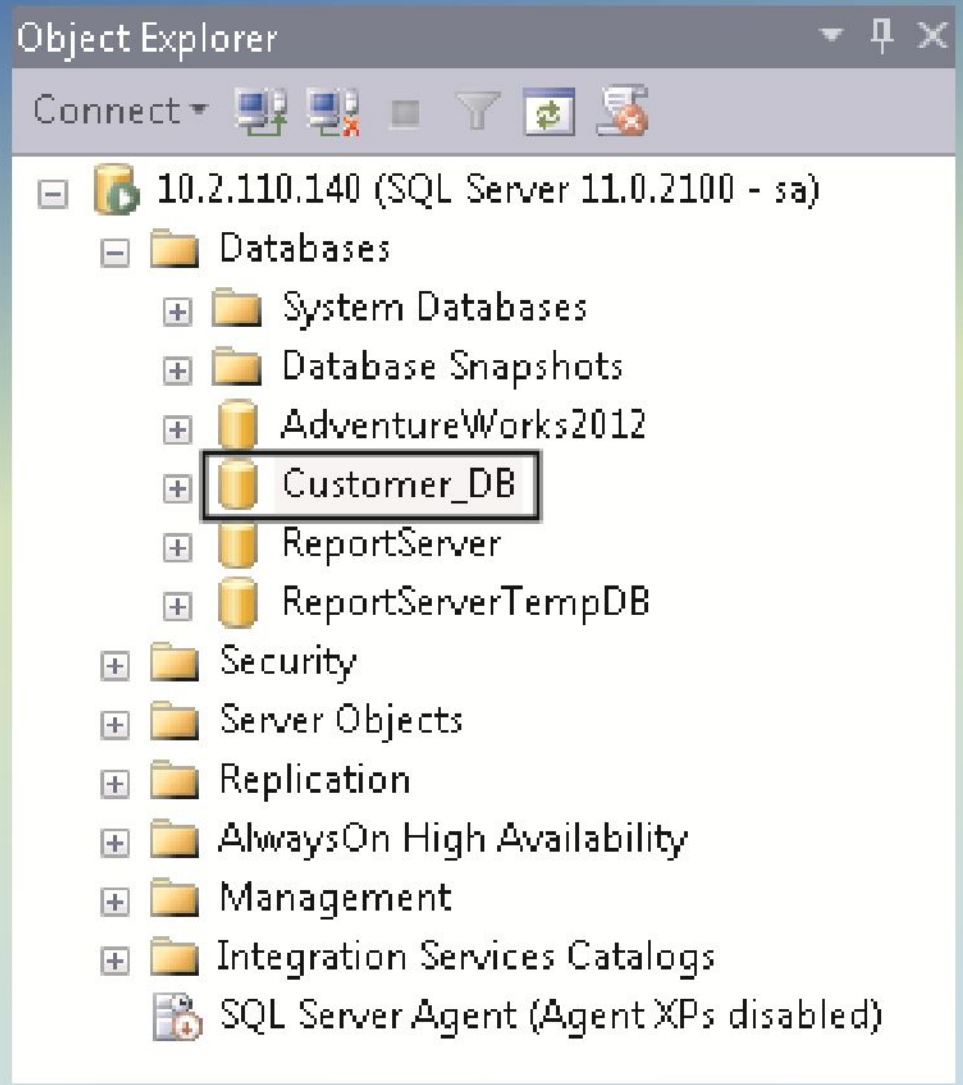
- Following code snippet shows how to create a database with database file and transaction log file with collation name:

```
CREATE DATABASE [Customer_DB] ON PRIMARY
( NAME = 'Customer_DB', FILENAME = 'C:\Program Files\Microsoft SQL Server\MSSQL11.MSSQLSERVER\MSSQL\DATA\Customer_DB.mdf')
LOG ON
( NAME = 'Customer_DB_log', FILENAME = 'C:\Program Files\Microsoft SQL Server\MSSQL11.MSSQLSERVER\MSSQL\DATA\Customer_DB_log.ldf')
COLLATE SQL_Latin1_General_CP1_CI_AS
```

- After executing the code, SQL Server 2012 displays the message 'Command(s) completed successfully'.

Creating Databases 4-4

- Following figure shows the database **`Customer_DB`** listed in the **Object Explorer**:



Modifying Databases 1-3

- As a user-defined database grows or diminishes, the database size will be expanded or be shrunk automatically or manually.
- The syntax to modify a database is as follows:

Syntax:

```
ALTER DATABASE database_name
{
<add_or_modify_files>
| <add_or_modify_filegroups>
| <set_database_options>
| MODIFY NAME = new_database_name
| COLLATE collation_name
}
[;]
```

where,

`database_name`: is the original name of the database.

`MODIFY NAME = new_database_name`: is the new name of the database to which it is to be renamed.

`COLLATE collation_name`: is the collation name of the database.

SQL Server 2016

Modifying Databases 2-3

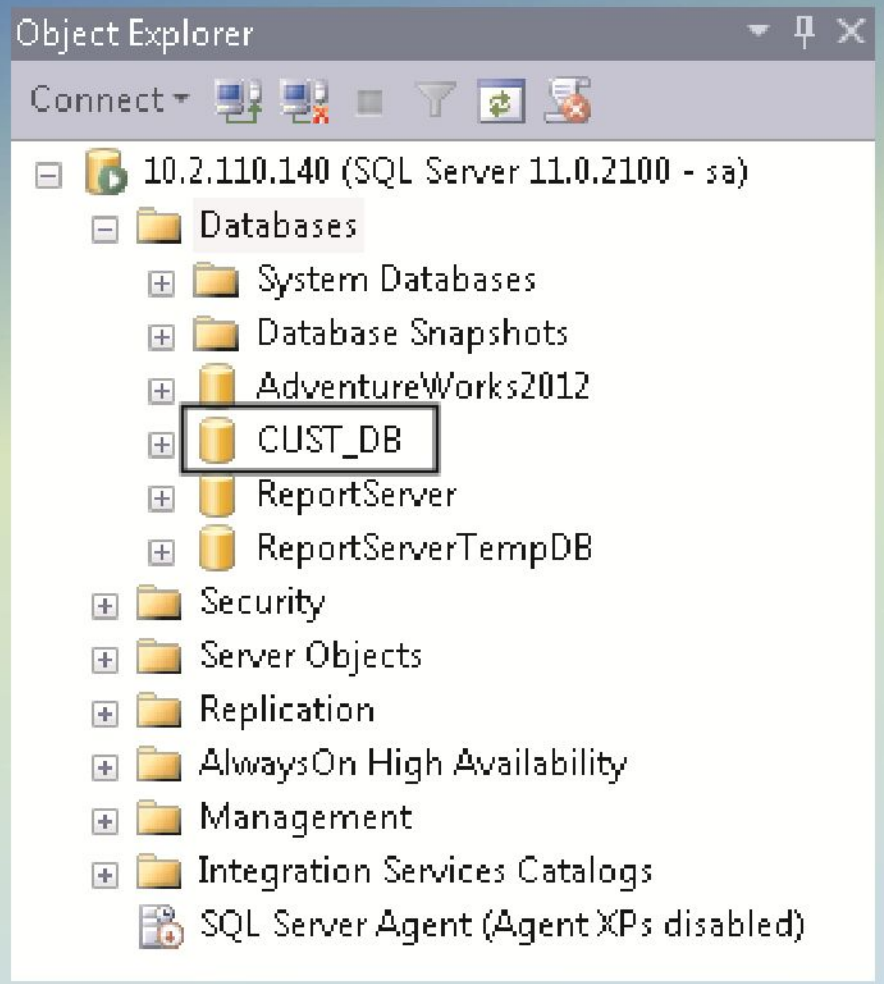
<add_or_modify_files>: is the file to be added, removed, or modified.
<add_or_modify_filegroups>: is the filegroup to be added, modified, or removed from the database.
<set_database_options>: is the database-level option influencing the characteristics of the database that can be set for each database. These options are unique to each database and do not affect other databases.

- Following code snippet shows how to rename a database **Customer_DB** with a new database name, **CUST_DB**:

```
ALTER DATABASE Customer_DB MODIFY NAME = CUST_DB
```

Modifying Databases 3-3

- Following figure shows database **Customer_DB** is renamed with a new database name, **CUST_DB**:



SQL Server 2016

Ownership of Databases 1-2

- In SQL Server 2012, the ownership of a user-defined database can be changed.
- Ownership of system databases cannot be changed.
- The system procedure `sp_changedbowner` is used to change the ownership of a database. The syntax is as follows:

Syntax:

```
sp_changedbowner [ @loginame = ] 'login'
```

where,

`login`: is an existing database username.

Ownership of Databases 2-2

- After `sp_changedbowner` is executed, the new owner is known as the `dbo` user inside the selected database.
- The `dbo` receives permissions to perform all activities in the database.
- The owner of the `master`, `model`, or `tempdb` system databases cannot be changed.
- Following code snippet, when executed, makes the login '`sa`' the owner of the current database and maps '`sa`' to existing aliases that are assigned to the old database owner, and will display 'Command(s) completed successfully':

```
USE CUST_DB
EXEC sp_changedbowner 'sa'
```

Setting Database Options 1-2

- Database-level options determine the characteristics of the database that can be set for each database.
- These options are unique to each database, so they do not affect other databases.
- These database options are set to default values when a database is first created, and can then, be changed by using the `SET` clause of the `ALTER DATABASE` statement.
- Following table shows the database options that are supported by SQL Server 2012:

Option Type	Description
Automatic options	Controls automatic behavior of database.
Cursor options	Controls cursor behavior.
Recovery options	Controls recovery models of database.
Miscellaneous options	Controls ANSI compliance.
State options	Controls state of database, such as online/offline and user connectivity.

Setting Database Options 2-2

- Following code snippet when executed sets AUTO_SHRINK option for the CUST_DB database to ON:

```
USE CUST_DB;
ALTER DATABASE CUST_DB
SET AUTO_SHRINK ON
```

- The AUTO_SHRINK options when set to ON, shrinks the database that have free space.

AdventureWorks2012 Database 1-2

- The `AdventureWorks2012` database consists of around 100 features.
- Some of the key features are as follows:

- Database Engine
- Analysis Services
- Integration Services
- Notification Services
- Reporting Services
- Replication Facilities
- A set of integrated samples for two multiple feature-based samples: `HRResume` and `Storefront`.

AdventureWorks2012 Database 2-2

- The sample database consists of these parts:

AdventureWorks2012: Sample OLTP database

AdventureWorks2012DW: Sample Data warehouse

AdventureWorks2012AS: Sample Analysis Services database

SQL Server 2016

Filegroups 1-2

In SQL Server, data files are used to store database files. The data files are further subdivided into filegroups for the sake of performance.

Each filegroup is used to group related files that together store a database object.

Every database has a primary filegroup by default. This filegroup contains the primary data file.

The primary file group and data files are created automatically with default property values at the time of creation of the database.

User-defined filegroups can then be created to group data files together for administrative, data allocation, and placement purposes.

SQL Server 2016

Filegroups 2-2

For example, three files named `Customer_Data1.ndf`, `Customer_Data2.ndf`, and `Customer_Data3.ndf` can be created on three disk drives respectively.

These can then be assigned to the filegroup `Customer_fgroup1`. A table can then be created specifically on the filegroup `Customer_fgroup1`.

Queries for data from the table will be spread across the three disk drives thereby, improving performance.

- Following table shows the filegroups that are supported by SQL Server 2012:

Filegroup	Description
Primary	The filegroup that consists of the primary file. All system tables are placed inside the primary filegroup.
User-defined	Any filegroup that is created by the user at the time of creating or modifying databases.

SQL Server 2016

Adding Filegroups to an Existing Database 1-5

- Filegroups can be created when the database is created for the first time or can be created later when more files are added to the database.
- However, files cannot be moved to a different filegroup after the files have been added to the database.
- A file cannot be a member of more than one filegroup at the same time.
- A maximum of 32,767 filegroups can be created for each database.
- Filegroups can contain only data files. Transaction log files cannot belong to a filegroup.

Adding Filegroups to an Existing Database 2-5

- The following is the syntax to add filegroups while creating a database:

Syntax:

```
CREATE DATABASE database_name
[ ON
[ PRIMARY ] [ <filespec> [ ,...n ]
[ , <filegroup> [ ,...n ] ]
[ LOG ON { <filespec> [ ,...n ] } ]
]
[ COLLATE collation_name ]
]
[;]
```

where,

`database_name`: is the name of the new database.

`ON`: indicates the disk files to store the data sections of the database, and data files.

`PRIMARY and associated <filespec> list`: define the primary file. The first file specified in the <filespec> entry in the primary filegroup becomes the primary file.

`LOG ON`: indicates the disk files used to store the database log files.

`COLLATE collation_name`: is the default collation for the database.

Adding Filegroups to an Existing Database 3-5

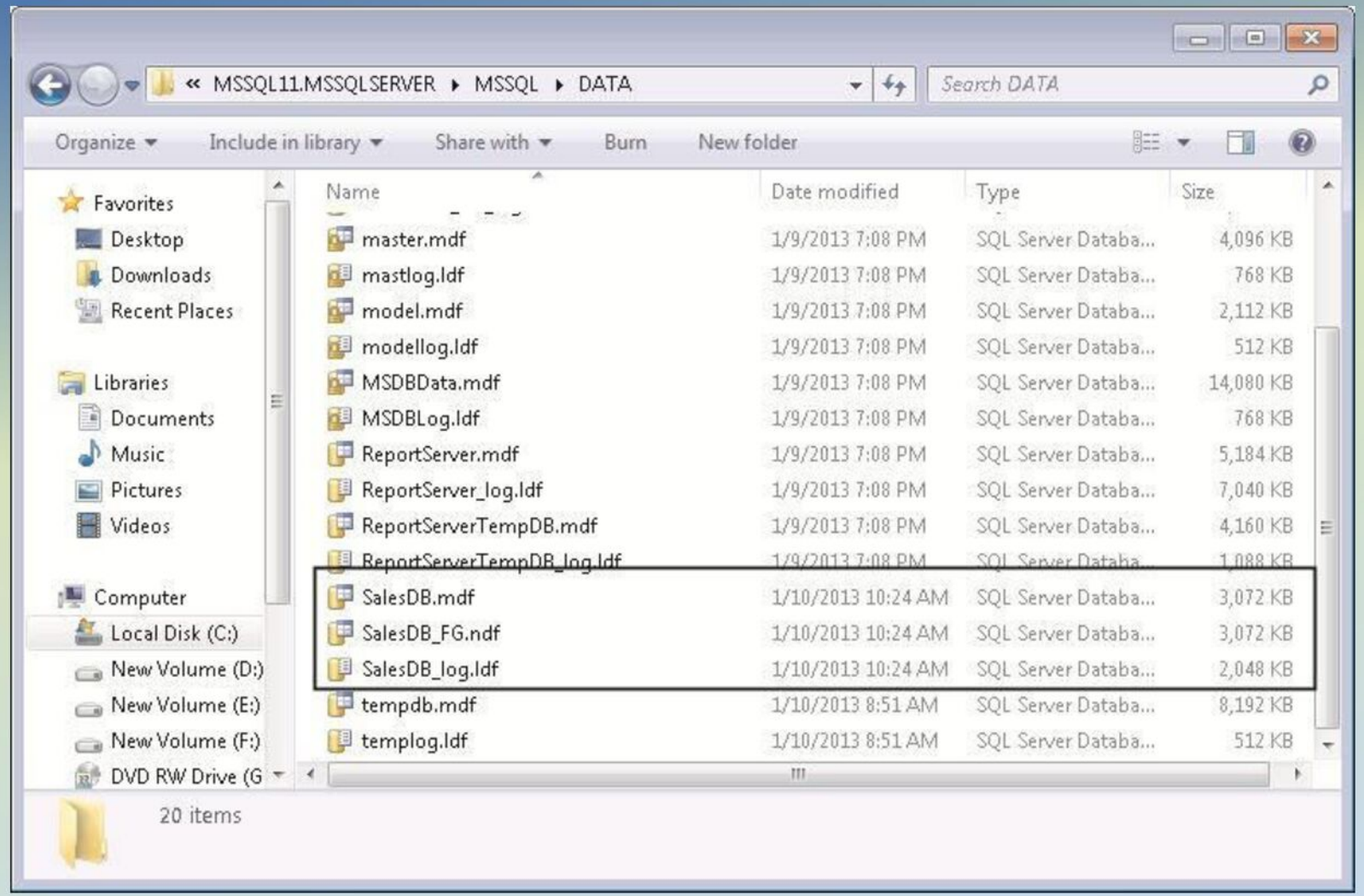
- Following code snippet shows how to add a filegroup (PRIMARY as default) while creating a database, called **SalesDB**:

```
CREATE DATABASE [SalesDB] ON PRIMARY
( NAME = 'SalesDB', FILENAME ='C:\Program Files\Microsoft SQL
Server\MSSQL11.MSSQLSERVER\MSSQL\DATA\SalesDB.mdf' , SIZE = 3072KB ,
MAXSIZE = UNLIMITED, FILEGROWTH = 1024KB ),
FILEGROUP [MyFileGroup]
( NAME = 'SalesDB_FG', FILENAME = 'C:\Program Files\Microsoft SQL
Server\MSSQL11.MSSQLSERVER\MSSQL\DATA\SalesDB_FG.ndf' , SIZE = 3072KB ,
MAXSIZE = UNLIMITED, FILEGROWTH = 1024KB )
LOG ON
( NAME = 'SalesDB_log', FILENAME = 'C:\Program Files\Microsoft SQL
Server\MSSQL11.MSSQLSERVER\MSSQL\DATA\SalesDB_log.ldf' , SIZE = 2048KB ,
MAXSIZE = 2048GB , FILEGROWTH = 10%)
COLLATE SQL_Latin1_General_CP1_CI_AS
```

SQL Server 2016

Adding Filegroups to an Existing Database 4-5

- Following figure shows the file groups when creating **`SalesDB`** database:



Adding Filegroups to an Existing Database 5-5

- The syntax to add a filegroup to an existing database is as follows:

```
ALTER DATABASE database_name
{ <add_or_modify_files>
| <add_or_modify_filegroups>
| <set_database_options>
| MODIFY NAME = new_database_name
| COLLATE collation_name
}
[;]
```

- Following code snippet shows how to add a filegroup to an existing database, called **CUST_DB**:

```
USE CUST_DB;
ALTER DATABASE CUST_DB
ADD FILEGROUP FG_ReadOnly0
```

- After executing the code, SQL Server 2012 displays the message 'Command(s) completed successfully' and the filegroup `FG_ReadOnly` is added to the existing database, **CUST_DB**.

SQL Server 2016

Default Filegroup 1-2

- Objects are assigned to the default filegroup when they are created in the database.
- The `PRIMARY` filegroup is the default filegroup. The default filegroup can be changed using the `ALTER DATABASE` statement.
- System objects and tables remain within the `PRIMARY` filegroup, but do not go into the new default filegroup.
- To make the `FG_ReadOnly` filegroup as default, it should contain at least one file inside it.
- Following code snippet shows how to create a new file, add it to the `FG_ReadOnly` filegroup, and make the `FG_ReadOnly` filegroup as the default filegroup:

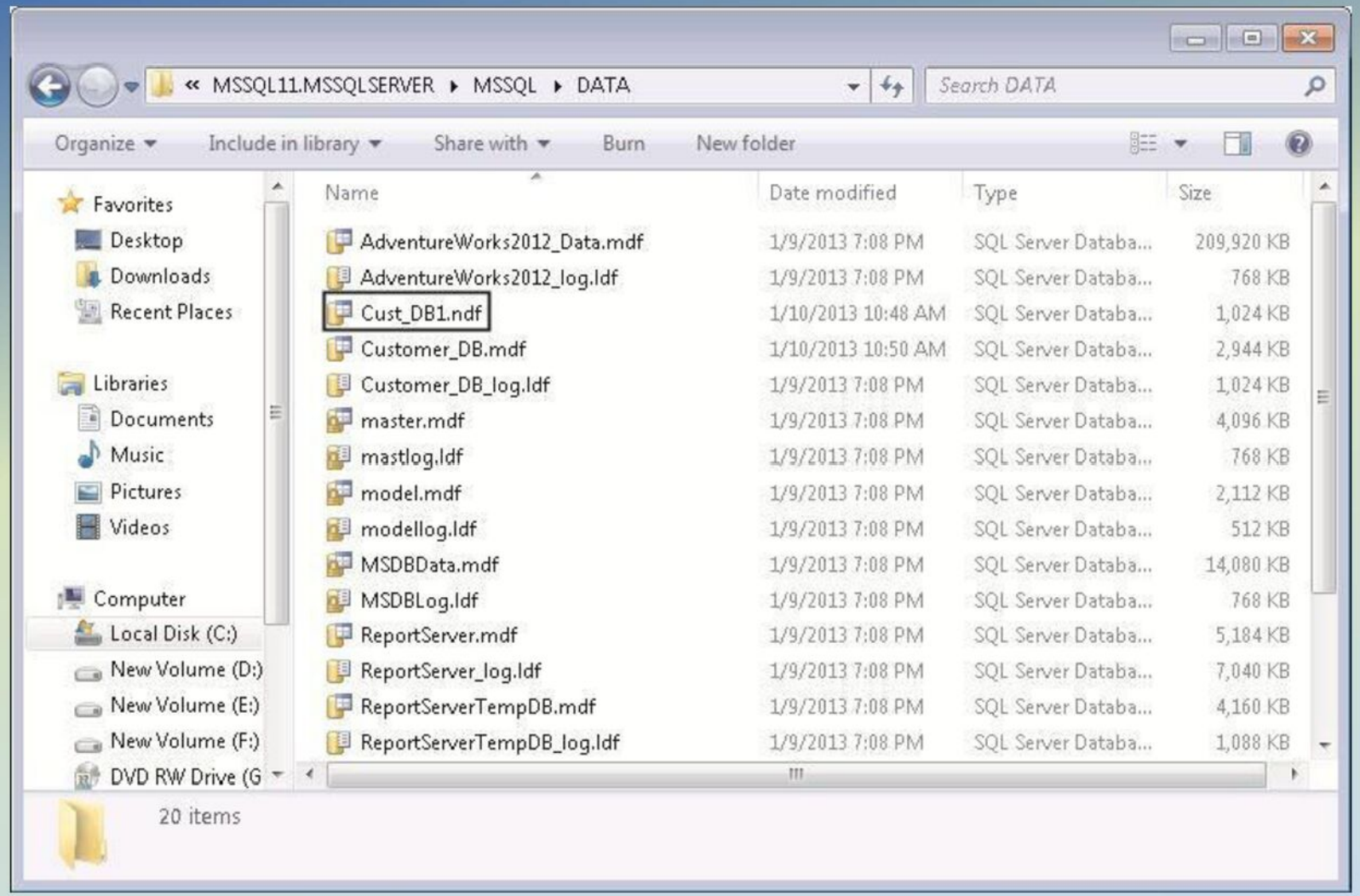
```
USE CUST_DB
ALTER DATABASE CUST_DB
ADD FILE (NAME = Cust_DB1, FILENAME = 'C:\Program Files\Microsoft SQL
Server\MSSQL11.MSSQLSERVER\MSSQL\DATA\Cust_DB1.ndf')
TO FILEGROUP FG_ReadOnly
ALTER DATABASE CUST_DB
MODIFY FILEGROUP FG_ReadOnly DEFAULT
```

- After executing the code, SQL Server 2012 displays the message saying the filegroup property 'DEFAULT' has been set.

SQL Server 2016

Default Filegroup 2-2

- Following figure shows a new file **`Cust_DB1`** created:



SQL Server 2016

Transaction Log 1-4

- A transaction log in SQL Server records all transactions and the database modifications made by each transaction.
- The transaction log is one of the critical components of the database.
- It can be the only source of recent data in case of system failure.
- The transaction logs support operations such as the following:

Recovery of individual transactions

- An incomplete transaction is rolled back in case of an application issuing a ROLLBACK statement or the Database Engine detecting an error.
- The log records are used to roll back the modifications.

Recovery of all incomplete transactions when SQL Server is started

- If a server that is running SQL Server fails, the databases may be left in an inconsistent state.
- When an instance of SQL Server is started, it runs a recovery of each database.

SQL Server 2016

Transaction Log 2-4

Rolling a restored database, file, filegroup, or page forward to the point of failure

- The database can be restored to the point of failure after a hardware loss or disk failure affecting the database files.

Supporting transactional replication

- The Log Reader Agent monitors the transaction log of each database configured for replications of transactions.

Supporting standby server solutions

- The standby-server solutions, database mirroring, and log shipping depend on the transaction log.

SQL Server 2016

Transaction Log 3-4

Working of Transaction Logs:

- A database in SQL Server 2012 has at least one data file and one transaction log file.
- Data and transaction log information are kept separated on the same file.
- Individual files are used by only one database.
- SQL Server uses the transaction log of each database to recover transactions.
- The transaction log is a serial record of all modifications that have occurred in the database as well as the transactions that performed the modifications.
- This log keeps enough information to undo the modifications made during each transaction.
- The transaction log records the allocation and deallocation of pages and the commit or rollback of each transaction.
- This feature enables SQL Server either to roll forward or to back out.

SQL Server 2016

Transaction Log 4-4

- The rollback of each transaction is executed using the following ways:

A transaction is rolled forward when a transaction log is applied.

A transaction is rolled back when an incomplete transaction is backed out.

Adding Log files to a database:

- The syntax to modify a database and add log files is as follows:

Syntax:

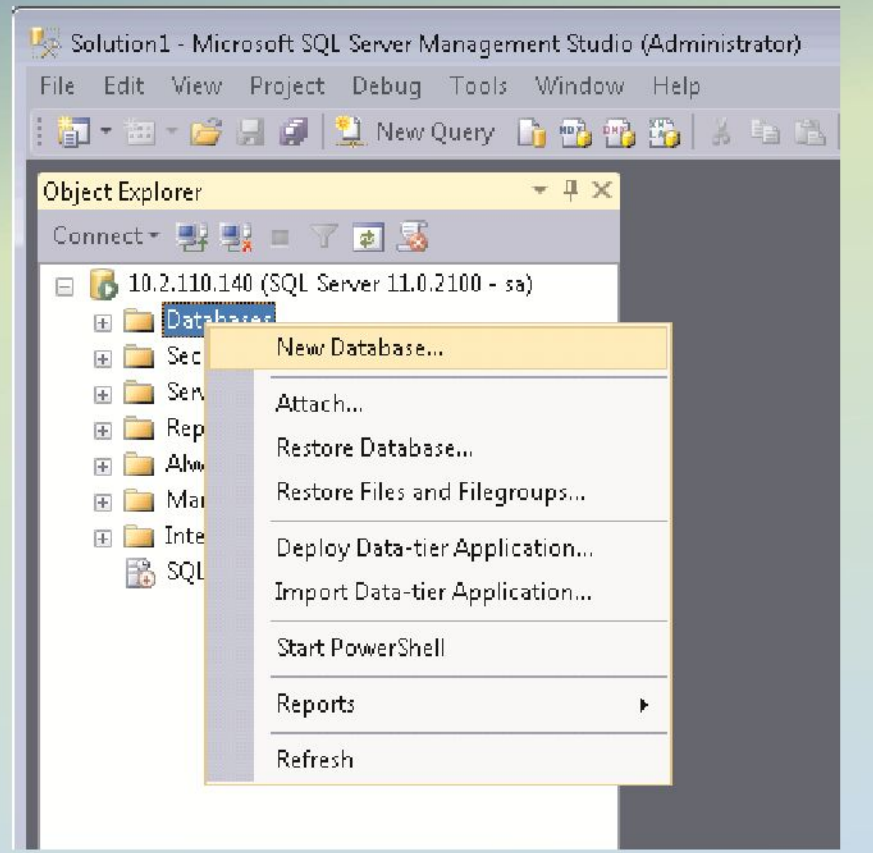
```
ALTER DATABASE database_name
{
...
}
[;]
```

```
<add_or_modify_files>::=
{
ADD FILE <filespec> [ ,...n ]
[ TO FILEGROUP { filegroup_name | DEFAULT } ]
| ADD LOG FILE <filespec> [ ,...n ]
| REMOVE FILE logical_file_name
| MODIFY FILE <filespec>
}
```

Create Database Procedure 1-7

- The steps to create a database using SSMS are as follows:

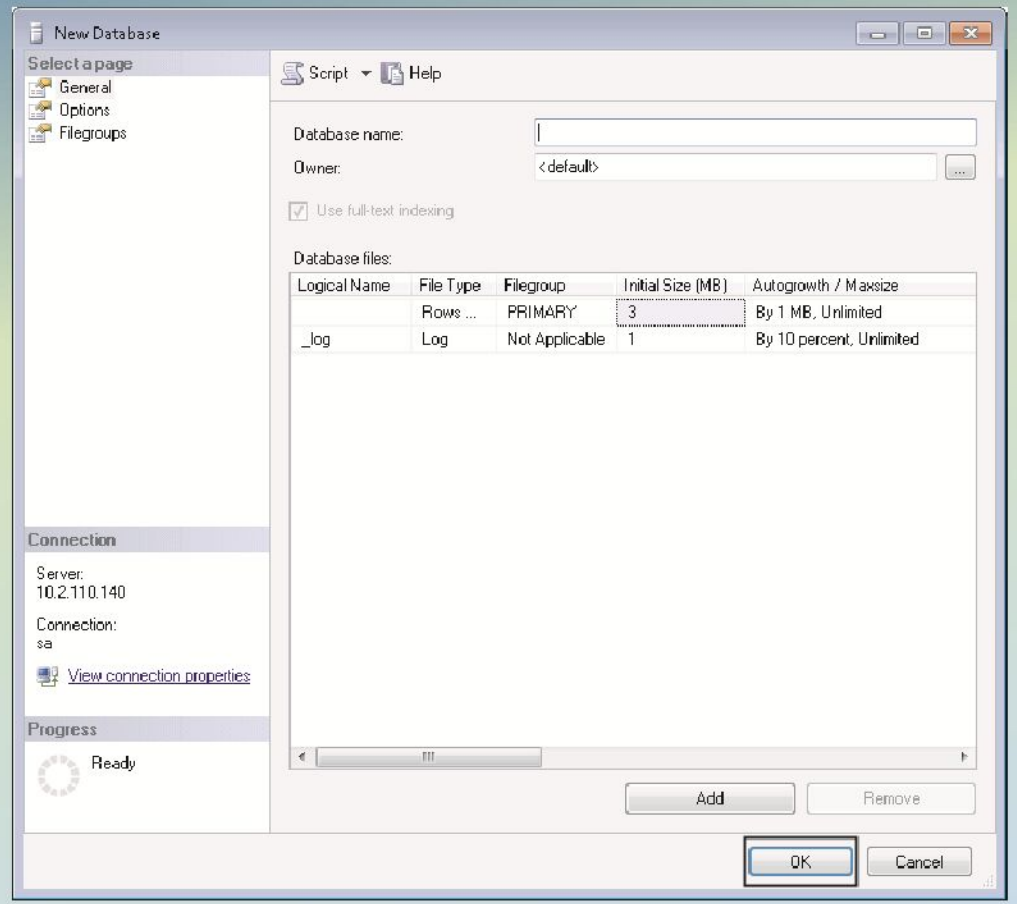
1. In **Object Explorer**, connect to an instance of the SQL Server Database Engine and then, expand that instance.
2. Right-click **Databases**, and then, click **New Database** as shown in the following figure:



Create Database Procedure 2-7

3. • In **New Database**, enter a database name.

4. • To create the database by accepting all default values, click **OK**, as shown in the following figure; otherwise, continue with the following optional steps:



SQL Server 2016

Create Database Procedure 3-7

5

- To change the owner name, click (…) to select another owner.

6

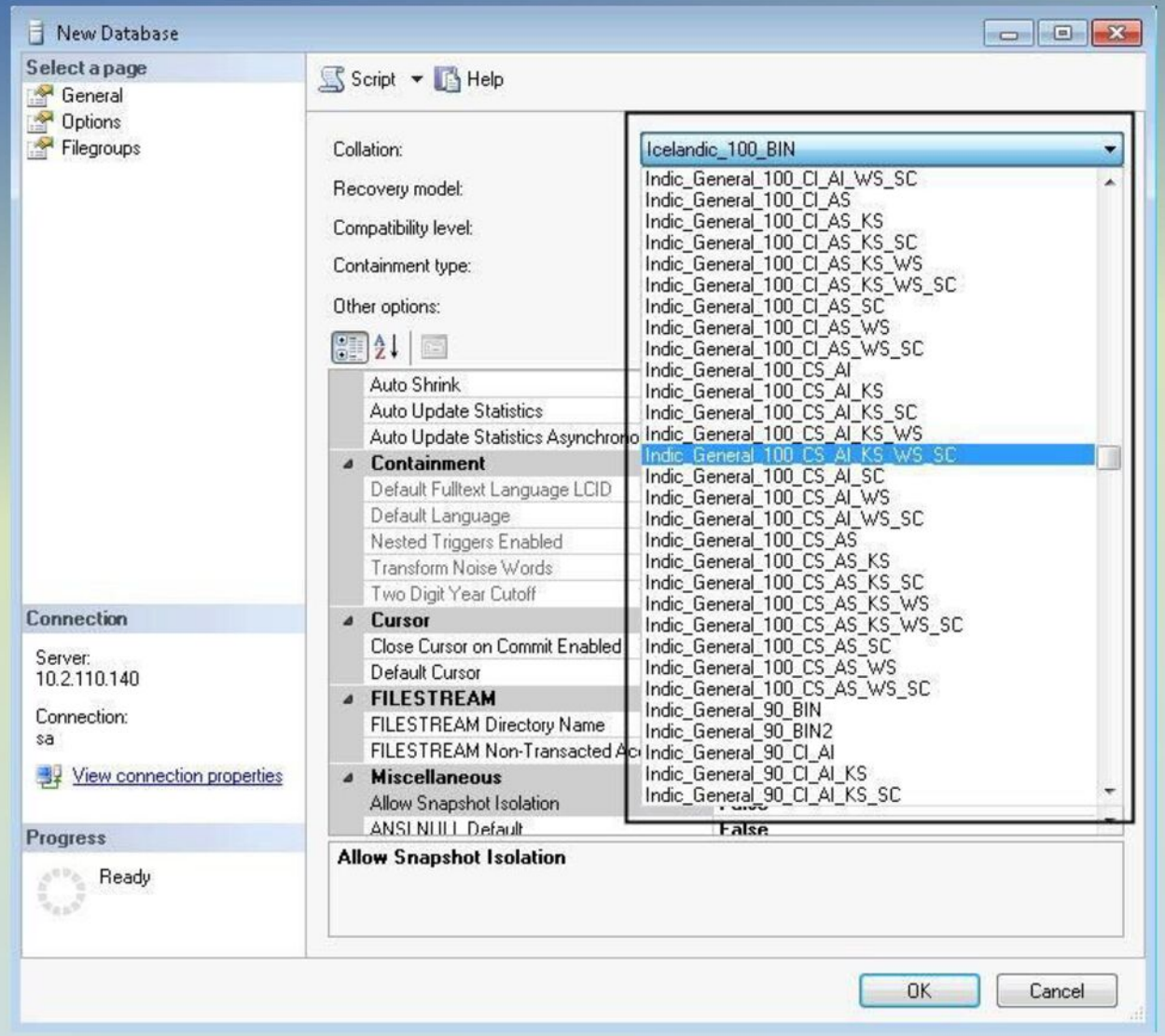
- To change the default values of the primary data and transaction log files, in the Database files grid, click the appropriate cell and enter the new value.

7

- To change the collation of the database, select the **Options** page, and then, select a collation from the list.

Create Database Procedure 4-7

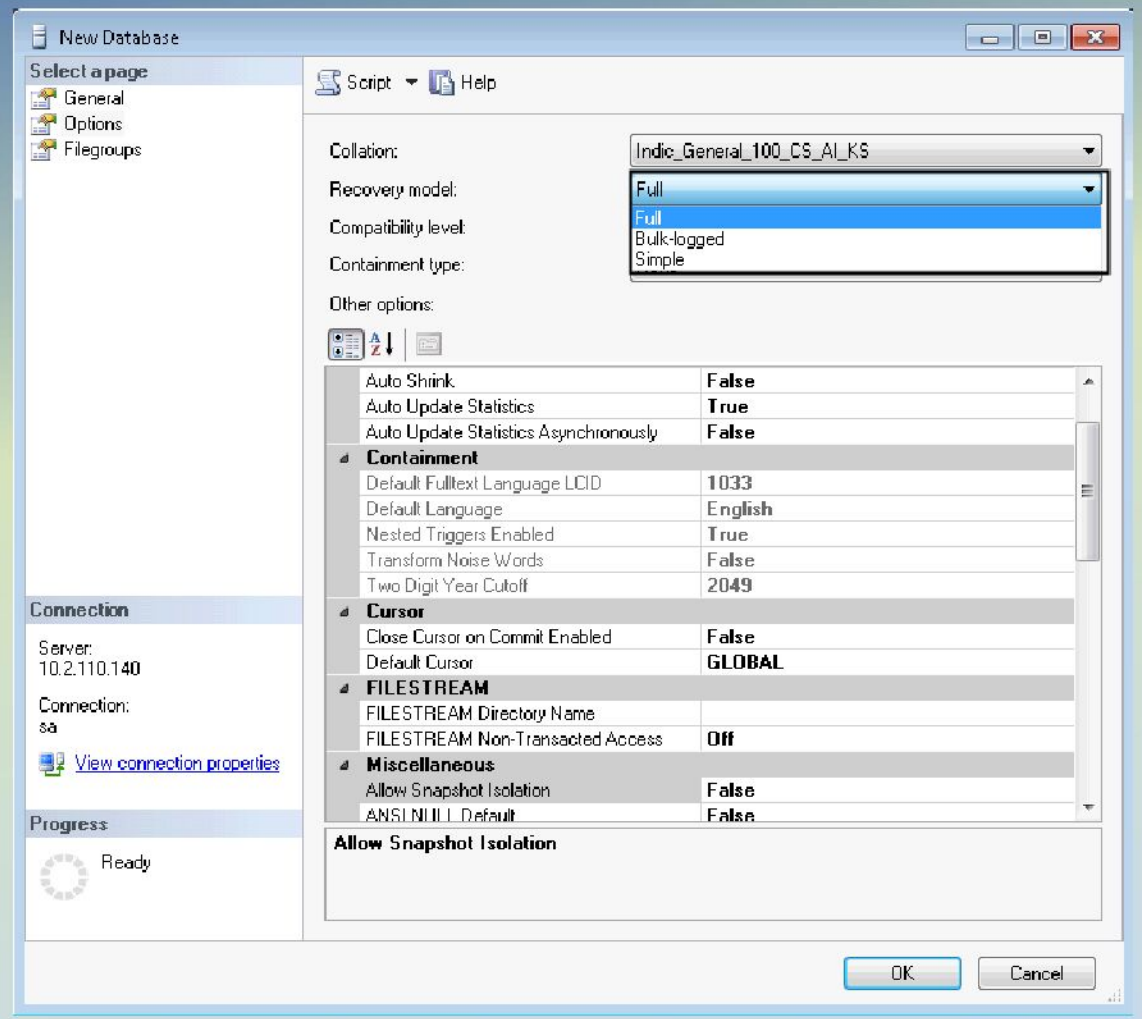
- This is shown in the following figure:



Create Database Procedure 5-7

8

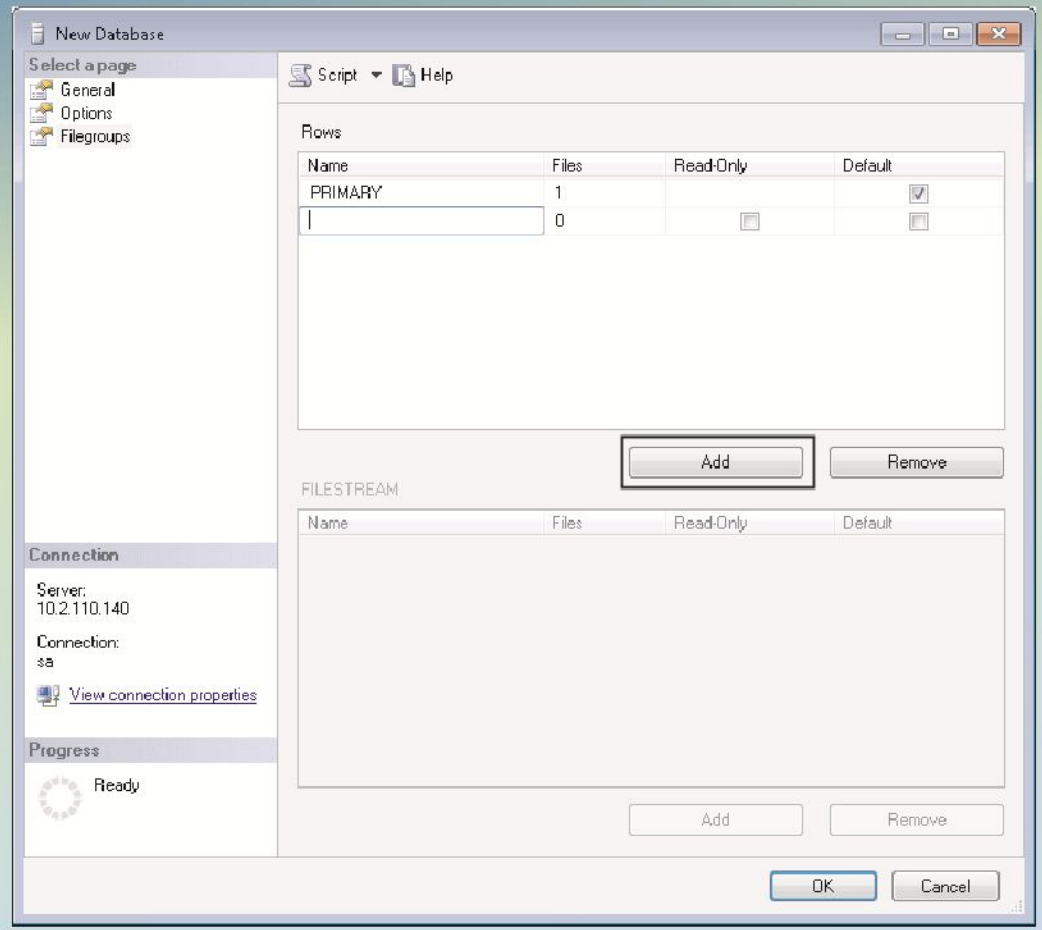
- To change the recovery model, select the **Options** page and then, select a recovery model from the list as shown in the following figure:



Create Database Procedure 6-7

9 • To change database options, select the **Options** page, and then, modify the database options.

10 • To add a new filegroup, click the **Filegroups** page. Click **Add** and then, enter the values for the filegroup as shown in the following figure:



SQL Server 2016

Create Database Procedure 7-7

11

- To add an extended property to the database, select the **Extended Properties** page.
- In the **Name** column, enter a name for the extended property.
- In the **Value** column, enter the extended property text. For example, enter one or more statements that describe the database.

12

- To create the database, click **OK**.

SQL Server 2016

Types of Database Modification Methods

- Following table lists and describes types of modifications of databases and modification methods:

Type of Modifications	Modification Methods
Increasing the size of a database	ALTER DATABASE statement or the database properties in SSMS
Changing the physical location of a database	ALTER DATABASE statement
Adding data or transaction log files	ALTER DATABASE statement or the database properties in SSMS
Shrinking a database	DBCC SHRINKDATABASE statement or the Shrink Database option in SSMS, accessed through the node for the specific database
Shrinking a database file	DBCC SHRINKFILE statement
Deleting data or log files	ALTER DATABASE statement or the database properties in SSMS
Adding a filegroup to a database	ALTER DATABASE statement or the database properties in SSMS
Changing the default filegroup	ALTER DATABASE statement
Changing database options	ALTER DATABASE statement or the database properties in SSMS
Changing the database owner	sp_changedbowner system stored procedure

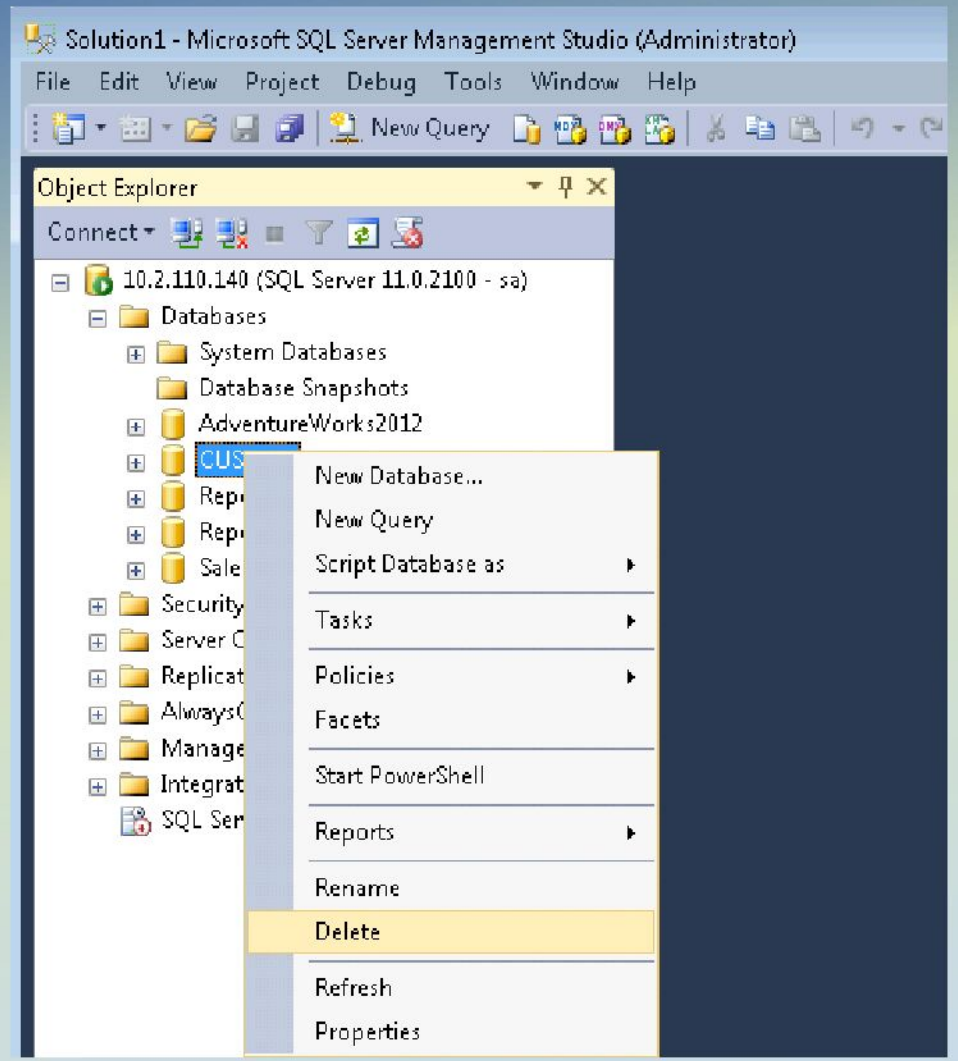
Drop Database Procedure 1-4

- A full backup of the database needs to be taken before dropping a database.
- A deleted database can be re-created only by restoring a backup.
- The steps to delete or drop a database using SSMS are as follows:

1. In **Object Explorer**, connect to an instance of the SQL Server Database Engine, and then, expand that instance.
2. Expand **Databases**, right-click the database to delete, and then, click **Delete**.

Drop Database Procedure 2-4

- This is shown in the following figure:



Drop Database Procedure 3-4

3

- Confirm that the correct database is selected, and then, click **OK**.

- The syntax to delete or drop a database using Transact-SQL is as follows:

```
CREATE DATABASE database_snapshot_name
ON
(
NAME = logical_file_name,
FILENAME = 'os_file_name'
) [ ,...n ]
AS SNAPSHOT OF source_database_name
[;]
```

where,

`database_snapshot_name`: is the name of the new database snapshot.

`ON ( NAME = logical_file_name, FILENAME = 'os_file_name' ) [ ,... n ]`: is the list of files in the source database. For the snapshot to work, all the data files must be specified individually.

Drop Database Procedure 4-4

`AS SNAPSHOT OF source_database_name`: is the database being created is a database snapshot of the source database specified by `source_database_name`.

- Following code snippet creates a database snapshot on the **`CUST_DB`** database:

```
CREATE DATABASE customer_snapshot01 ON
( NAME = Customer_DB, FILENAME = 'C:\Program Files\Microsoft SQL
 Server\MSSQL11.MSSQLSERVER\MSSQL\DATA\Customerdat_0100.ss')
AS SNAPSHOT OF CUST_DB;
```

Summary

- An SQL Server database is made up of a collection of tables that stores sets of specific structured data.
- SQL Server supports three kinds of databases:
  - System databases
  - User-defined databases
  - Sample databases
- SQL Server uses system databases to support different parts of the DBMS.
- A fictitious company, Adventure Works Cycles is created as a scenario and the AdventureWorks2012 database is designed for this company.
- The SQL Server data files are used to store database files, which are further subdivided into filegroups for the sake of performance.
- Objects are assigned to the default filegroup when they are created in the database. The PRIMARY filegroup is the default filegroup.
- A database snapshot is a read-only, static view of a SQL Server database

SQL SERVER

Session

Tables and Data Types

Objectives

Describe the procedure to

- add,
- Modify,
- & drop

Columns in tables.

Introduction

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

SQL Server 2016

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.

A data type also specifies the storage capacity of an object.

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

In this manner, data types enforce data integrity.

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

SQL Server 2016

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.

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.

SQL Server 2016

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.

SQL Server 2016

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).

SQL Server 2016

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.

SQL Server 2016

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).

SQL Server 2016

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
```

SQL Server 2016

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
```

SQL Server 2016

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.

SQL Server 2016

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]
```

SQL Server 2016

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.

SQL Server 2016

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.

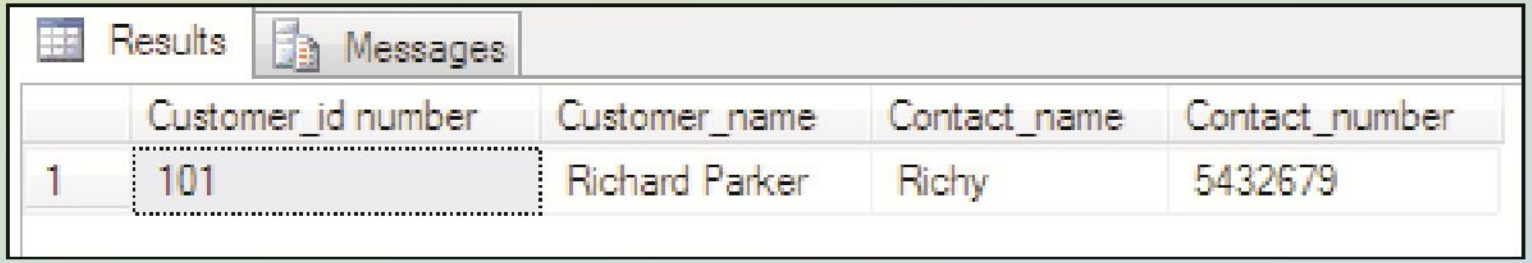
Data Modification Statements 3-4

<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:



	Customer_id number	Customer_name	Contact_name	Contact_number
1	101	Richard Parker	Richy	5432679

SQL Server 2016

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
```

SQL Server 2016

Column Nullability 1-2

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

In SQL Server, a null value is not same as zero, blank, or a zero length character string (such as ' '). For example, a null value in the **`Color`** column of the **`Production`**.

`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.

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

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.

SQL Server 2016

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.

It is also possible to explicitly enter a null value into a column regardless of what data type it is or whether it has a default associated with it.

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.

SQL Server 2016

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.

For example, it is common to specify zero as the default for numeric columns or 'N/A' or 'Unknown' as the default for string columns when no value is specified.

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.

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

SQL Server 2016

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
```

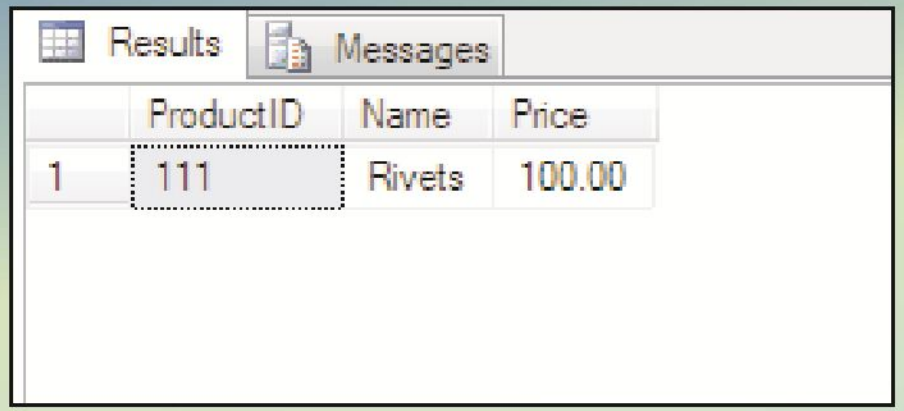
- When a row is inserted using a statement as shown in the following code snippet, the value of **`Price`** will not be blank; it will have a value of **`100.00`** even though a user has not entered any value for that column.

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

SQL Server 2016

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

SQL Server 2016

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.

SQL Server 2016

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.

SQL Server 2016

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.

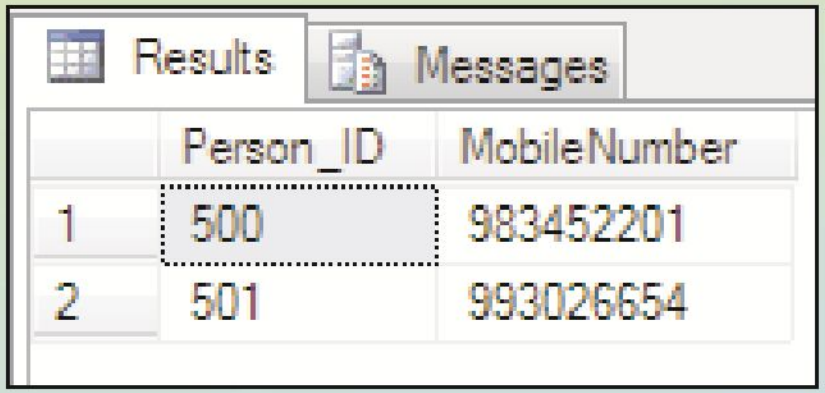
SQL Server 2016

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.

SQL Server 2016

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.

SQL Server 2016

Globally Unique Identifiers 3-3

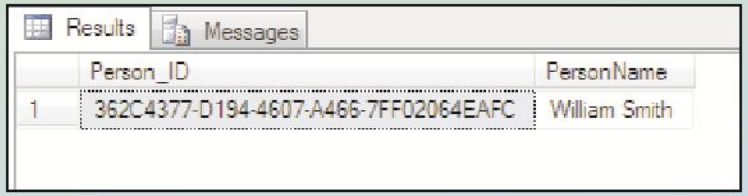
- Following code snippet demonstrates how to `CREATE TABLE` statement to create the **`EMPCellularPhone`** table.
- The **`Person_ID`** column automatically generates a `GUID` for each new row added to the table.

```
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`**:



SQL Server 2016

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`

SQL Server 2016

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.

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.

Two rows in a table cannot have the same primary key value and a column that is a primary key cannot have `NULL` values.

- 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] )
```

SQL Server 2016

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)
```

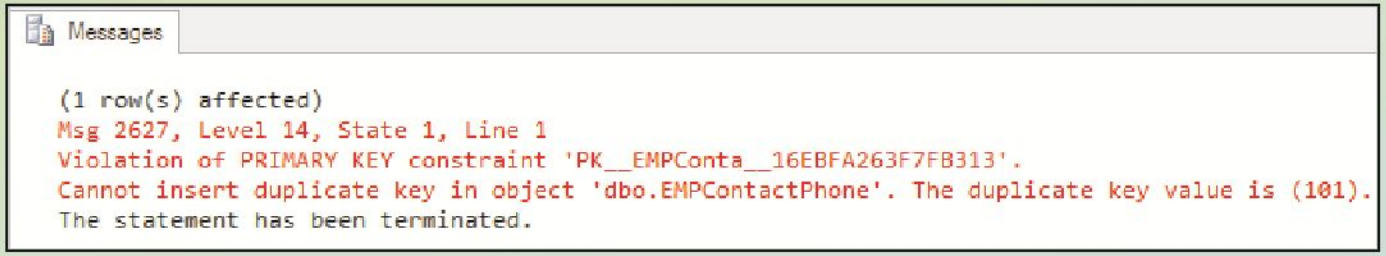
SQL Server 2016

PRIMARY KEY 3-3

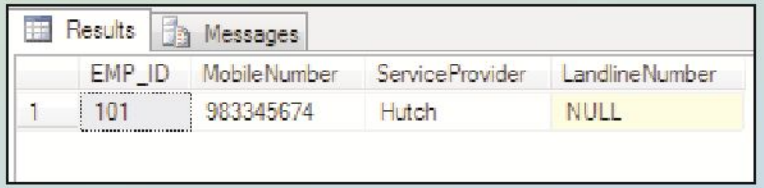
- Having created a primary key for `EMP_ID`, a query is written to insert rows into the table with the statements shown in the following code snippet:

```
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:



SQL Server 2016

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:

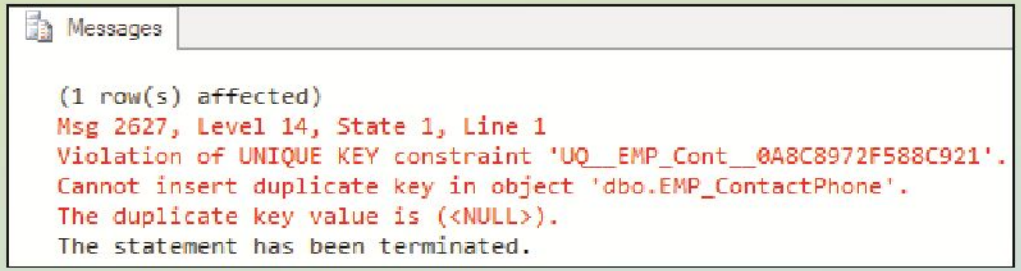
```
USE [CUST_DB]
GO

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

UNIQUE 2-2

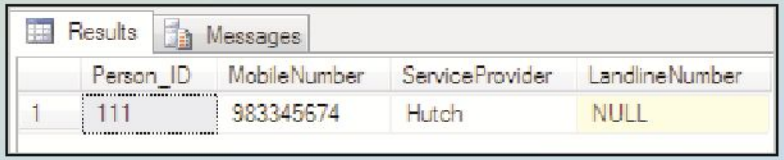
- Following code snippet demonstrates how to insert a row into the table:

```
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:



SQL Server 2016

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)
```

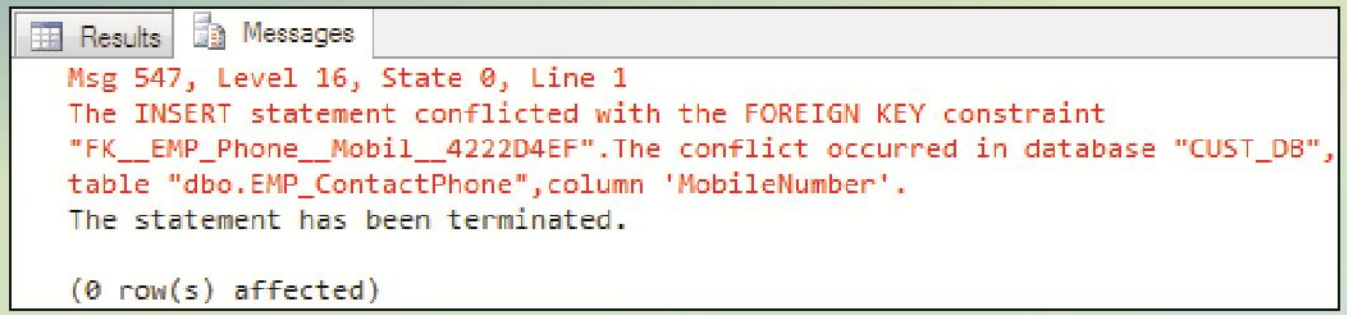
SQL Server 2016

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.

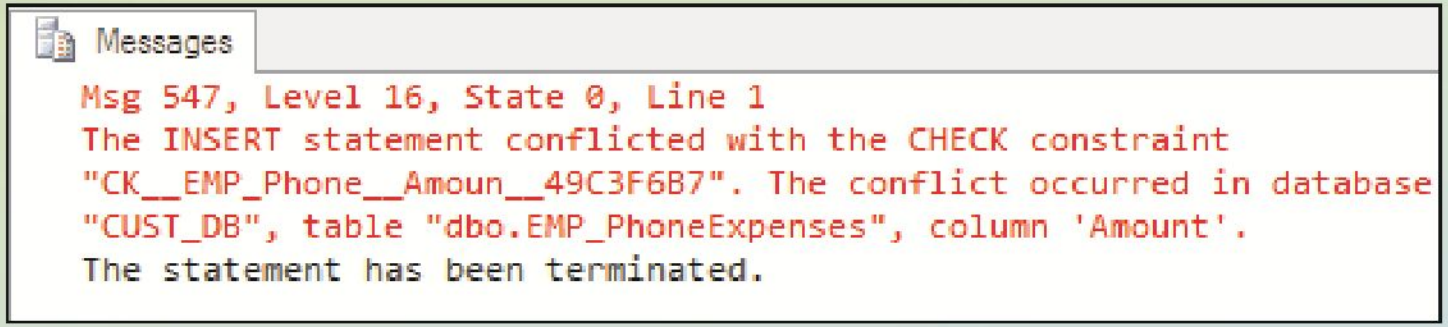
SQL Server 2016

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

- 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.

About the Author

Created By: Mohammad Salman

Experience: 17 Years +

Designation: Corporate Trainer .NET

Microsoft CERTIFIED Professional

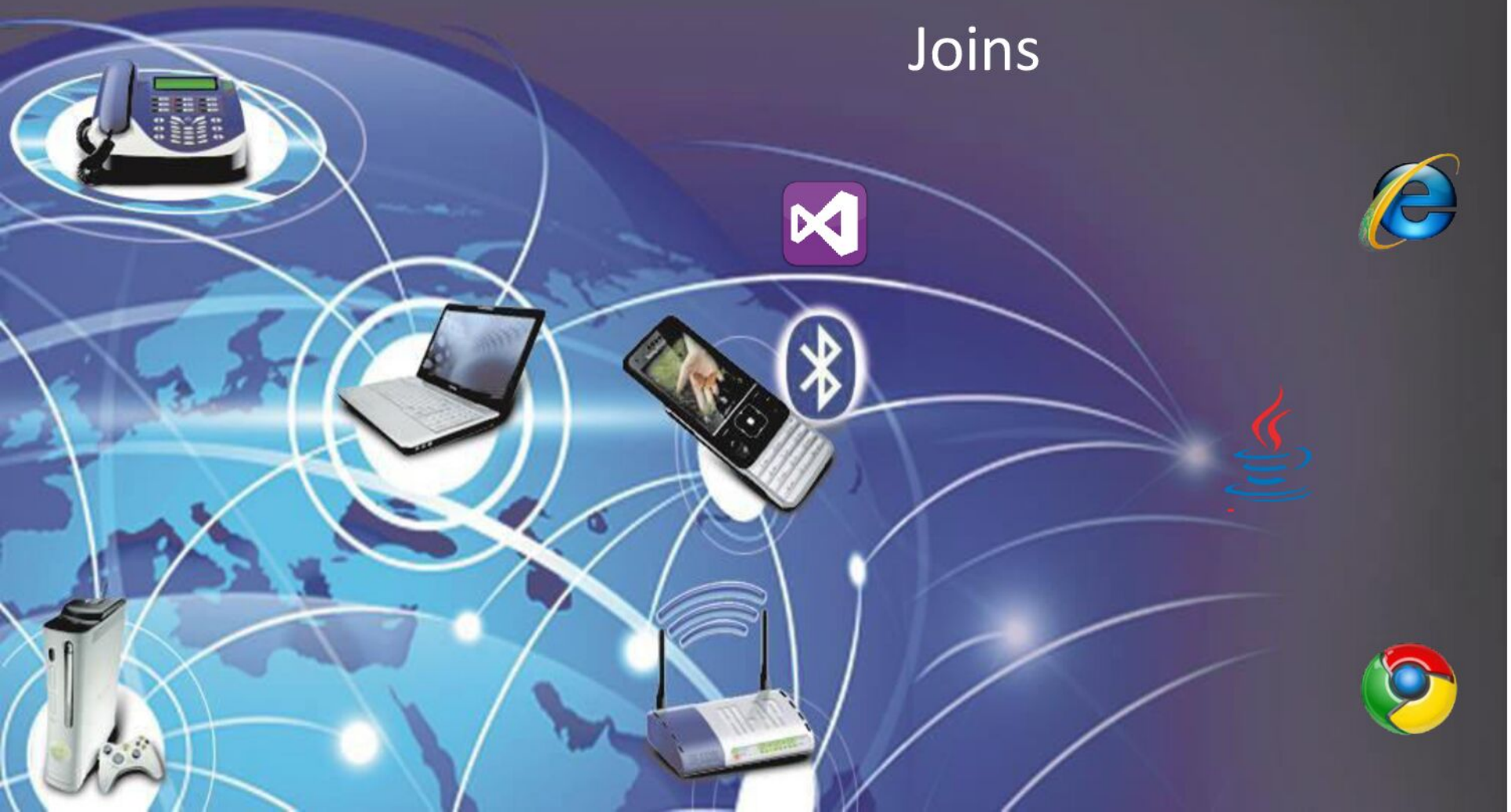
Microsoft Specialist

Microsoft CERTIFIED Solutions Developer

SQL SERVER

Session

Joins



Objectives

- Explain joins
- Describe various types of joins
- Explain the use of various set operators to combine data

SQL Server 2016

Joins 1-3

- Joins are used to retrieve data from two or more tables based on a logical relationship between tables.
- It defines the manner in which two tables are related in a query by:

Specifying the column from each table to be used for the join. A typical join specifies a foreign key from one table and its associated key in the other table.

Specifying a logical operator such as =, <> to be used in comparing values from the columns.

- Joins can be specified in either the `FROM` or `WHERE` clauses.
- The syntax of the `JOIN` statement is as follows:

Syntax:

```
SELECT <ColumnName1>, <ColumnName2>...<ColumnNameN>
FROM Table_A AS Table_Alias_A
JOIN
Table_B AS Table_Alias_B
ON
Table_Alias_A.<CommonColumn> = Table_Alias_B.<CommonColumn>
```

where,

`<ColumnName1>, <ColumnName2>`: Is a list of columns that need to be displayed.

SQL Server 2016

Joins 2-3

`Table_A`: Is the name of the table on the left of the JOIN keyword.
`Table_B`: Is the name of the table on the right of the JOIN keyword.
`AS Table_Alias`: Is a way of giving an alias name to the table. An alias defined for the table in a query can be used to denote a table so that the full name of the table need not be used.
`<CommonColumn>`: Is a column that is common to both the tables. In this case, the join succeeds only if the columns have matching values.

- Consider that you want to list employee first names, last names, and their job titles from the `HumanResources.Employee` and `Person.Person`.
- To extract this information from the two tables, you need to join them based on `BusinessEntityID` as shown in the following code snippet:

```
SELECT A.FirstName, A.LastName, B.JobTitle
FROM Person.Person A
JOIN
HumanResources.Employee B
ON
A.BusinessEntityID = B.BusinessEntityID;
```

SQL Server 2016

Joins 3-3

- Here, the tables `HumanResources.Employee` and `Person.Person` are given aliases `A` and `B`. They are joined together on the basis of their business entity ids.
- The `SELECT` statement then retrieves the desired columns through the aliases.
- Following figure shows the output:

	FirstName	LastName	JobTitle
1	Ken	Sánchez	Chief Executive Officer
2	Terri	Duffy	Vice President of Engineering
3	Roberto	Tamburello	Engineering Manager
4	Rob	Walters	Senior Tool Designer
5	Gail	Erickson	Design Engineer
6	Jossef	Goldberg	Design Engineer
7	Dylan	Miller	Research and Development Manager

SQL Server 2016

Inner Join

An inner join is formed when records from two tables are combined only if the rows from both the tables are matched based on a common column.

- The syntax of an inner join is as follows:

Syntax:

```
SELECT <ColumnName1>, <ColumnName2>...<ColumnNameN> FROM
Table_A AS Table_Alias_A
INNER JOIN
Table_B AS Table_Alias_B
ON
Table_Alias_A.<CommonColumn> = Table_Alias_B.<CommonColumn>
```

- Following code snippet demonstrates the use of inner join:

```
SELECT A.FirstName, A.LastName, B.JobTitle
FROM Person.Person A
INNER JOIN HumanResources.Employee B
ON
A.BusinessEntityID = B.BusinessEntityID;
```

- Here, an inner join is constructed between `Person.Person` and `HumanResources.Employee` based on common business entity ids.

SQL Server 2016

Outer Join

Outer joins are join statements that return all rows from at least one of the tables specified in the `FROM` clause, as long as those rows meet any `WHERE` or `HAVING` conditions of the `SELECT` statement.

- The two types of commonly used outer joins are as follows:

Left Outer Join

Right Outer Join

SQL Server 2016

Left Outer Join 1-3

Left outer join returns all the records from the left table and only matching records from the right table.

- The syntax of an outer join is as follows:

Syntax:

```
SELECT <ColumnList> FROM
Table_A AS Table_Alias_A
LEFT OUTER JOIN
Table_B AS Table_Alias_B
ON
Table_Alias_A.<CommonColumn> = Table_Alias_B.<CommonColumn>
```

- Consider that you want to retrieve all the customer ids from the `Sales.Customers` table and order information such as ship dates and due dates, even if the customers have not placed any orders.

- Since the record count would be very huge, it is to be restricted to only those orders that are placed before 2012.

SQL Server 2016

Left Outer Join 2-3

- The following code snippet achieves this by performing a left outer join:

```
SELECT A.CustomerID, B.DueDate, B.ShipDate
FROM Sales.Customer A LEFT OUTER JOIN
Sales.SalesOrderHeader B
ON
A.CustomerID = B.CustomerID AND YEAR(B.DueDate)<2012;
```

- In the query, the left outer join is constructed between the tables `Sales.Customer` and `Sales.SalesOrderHeader`.
- The tables are joined on the basis of customer ids.
- In this case, all records from the left table, `Sales.Customer` and only matching records from the right table, `Sales.SalesOrderHeader`, are returned.

SQL Server 2016

Left Outer Join 3-3

- Following figure shows the output:

	CustomerID	DueDate	ShipDate
3...	18178	2008-08-12 00:00:00.000	2008-08-07 00:00:00.000
3...	13671	2008-08-12 00:00:00.000	2008-08-07 00:00:00.000
3...	11981	2008-08-12 00:00:00.000	2008-08-07 00:00:00.000
3...	18749	2008-08-12 00:00:00.000	2008-08-07 00:00:00.000
3...	15251	2008-08-12 00:00:00.000	2008-08-07 00:00:00.000
3...	15868	2008-08-12 00:00:00.000	2008-08-07 00:00:00.000
3...	18759	2008-08-12 00:00:00.000	2008-08-07 00:00:00.000
3...	215	NULL	NULL
3...	46	NULL	NULL
3...	169	NULL	NULL
3...	507	NULL	NULL
3...	630	NULL	NULL

- As shown in the output, some records show the due dates and ship dates as `NULL`.
- This is because for some customers, no order is placed, hence, their records will show the dates as `NULL`.

SQL Server 2016

Right Outer Join 1-2

- The right outer join retrieves all the records from the second table in the join regardless of whether there is matching data in the first table or not.

- The syntax of a right outer join is as follows:

Syntax:

```
SELECT <ColumnList>
FROM Left_Table_Name
AS
Table_A AS Table_Alias_A
RIGHT OUTER JOIN
Table_B AS Table_Alias_B
ON
Table_Alias_A.<CommonColumn> = Table_Alias_B.<CommonColumn>
```

SQL Server 2016

Right Outer Join 2-2

- Consider that you want to retrieve all the product names from `Product` table and all the corresponding sales order ids from the `SalesOrderDetail` table even if there is no matching record for the products in the `SalesOrderDetail` table.

- The following code snippet achieve this by using a right outer join:

```
SELECT P.Name, S.SalesOrderID
FROM Sales.SalesOrderDetail S
RIGHT OUTER JOIN
Production.Product P
ON P.ProductID = S.ProductID;
```

- In the code, all the records from `Product` table are shown regardless of whether they have been sold or not.

SQL Server 2016

Self-Join 1-2

- A self-join is used to find records in a table that are related to other records in the same table. A table is joined to itself in a self-join.
- Consider that an **`Employee`** table in a database named Sterling has a column named **`mgr_id`** to denote information for managers whom employees are reporting to.
- Assume that the table has appropriate records inserted in it.
- A manager is also an employee. This means that the **`mgr_id`** in the table is the **`emp_id`** of an employee.
- For example, **Anabela** with **`emp_id`** as **ARD36773F** is an employee but **Anabela** is also a manager for Victoria, Palle, Karla, and other employees as shown in the following figure:

	emp_id	fname	minit	lname	job_id	job_lvl	pub_id	hire_date	mgr_id
1	PMA42628M	Paolo	M	Accorti	13	35	0877	1992-08-27 00:00:00.000	POK93028M
2	PSA89086M	Pedro	S	Afonso	14	89	1389	1990-12-24 00:00:00.000	POK93028M
3	VPA30890F	Victoria	P	Ashworth	6	140	0877	1990-09-13 00:00:00.000	ARD36773F
4	H-B39728F	Helen		Bennett	12	35	0877	1989-09-21 00:00:00.000	POK93028M
5	L-B31947F	Lesley		Brown	7	120	0877	1991-02-13 00:00:00.000	ARD36773F
6	F-C16315M	Francisco		Chang	4	227	9952	1990-11-03 00:00:00.000	MAS70474F
7	PTC11962M	Philip	T	Cramer	2	215	9952	1989-11-11 00:00:00.000	MAS70474F
8	A-C71970F	Aria		Cruz	10	87	1389	1991-10-26 00:00:00.000	POK93028M
9	AMD15433F	Ann	M	Devon	3	200	9952	1991-07-16 00:00:00.000	MAS70474F
10	ARD36773F	Anabela	R	Doming...	8	100	0877	1993-01-27 00:00:00.000	NULL
11	PHF38899M	Peter	H	Franken	10	75	0877	1992-05-17 00:00:00.000	POK93028M
12	PXH22250M	Paul	X	Henriot	5	159	0877	1993-08-19 00:00:00.000	MAS70474F

Self-Join 2-2

- To get a list of the manager names along with other details, you can use a self-join to join the employee table with itself and then, extract the desired records.
- Following code snippet demonstrates how to use a self-join:

```
SELECT TOP 7 A.fname + ' ' + A.lname AS 'Employee Name', B.fname + '
'+B.lname AS 'Manager'
FROM
Employee AS A
INNER JOIN
Employee AS B
ON A.mgr_id = B.emp_id
```

- In the code, the `Employee` table is joined to itself based on the `mgr_id` and `emp_id` columns.
- The following figure displays the output of the code:

	Employee Name	Manager
1	Paolo Accorti	Pirkko Koskitalo
2	Pedro Afonso	Pirkko Koskitalo
3	Victoria Ashworth	Anabela Domingues
4	Helen Bennett	Pirkko Koskitalo
5	Lesley Brown	Anabela Domingues
6	Francisco Chang	Margaret Smith
7	Philip Cramer	Margaret Smith

SQL Server 2016

UNION Operator 1-2

The results from two different query statements can be combined into a single resultset using the `UNION` operator.

The query statements must have compatible column types and equal number of columns.

The column names can be different in each statement but the data types must be compatible.

- The syntax of the `UNION` operator is as follows:

Syntax:

```
Query_Statement1
UNION [ALL]
Query_Statement2
```

where,

`Query_Statement1` and `Query_Statement2` are `SELECT` statements.

UNION Operator 2-2

- Following code snippet demonstrates the use of `UNION` operator:

```
SELECT Product.ProductId FROM Production.Product
UNION
SELECT ProductId FROM Sales.SalesOrderDetail
```

- This will list all the product ids of both tables that match with each other.
- If you include the `ALL` clause, all rows are included in the resultset including duplicate records.

```
SELECT Product.ProductId FROM Production.Product
UNION ALL
SELECT ProductId FROM Sales.SalesOrderDetail
```

- By default, the `UNION` operator removes duplicate records from the resultset.
- However, if you use the `ALL` clause with `UNION` operator, then all the rows are returned.
- Apart from `UNION`, the other operators that are used to combine data from multiple tables are `INTERSECT` and `EXCEPT`.

SQL Server 2016

INTERSECT Operator 1-2

The INTERSECT operator is used with two query statements to return a distinct set of rows that are common to both the query statements.

- The syntax of the INTERSECT operator is as follows:

Syntax:

```
Query_statement1
INTERSECT
Query_statement2
```

where,

Query_Statement1 and Query_Statement2 are SELECT statements.

- Following code snippet demonstrates the use of INTERSECT operator:

```
SELECT Product.ProductId FROM Production.Product
INTERSECT
SELECT ProductId FROM Sales.SalesOrderDetail
```

SQL Server 2016

INTERSECT Operator 2-2

- The basic rules for using `INTERSECT` are as follows:

  - The number of columns and the order in which they are given must be the same in both the queries.
  - The data types of the columns being used must be compatible.

- The result of the intersection of the `Production.Product` and `Sales.SalesOrderDetail` tables would be only those product ids that have matching records in `Production.Product` table.

SQL Server 2016

EXCEPT Operator 1-2

The EXCEPT operator returns all the distinct rows from the query given on the left of the EXCEPT operator and removes all the rows from the resultset that match the rows on the right of the EXCEPT operator.

- The syntax of the EXCEPT operator is as follows:

Syntax:

```
Query_statement1
EXCEPT
Query_statement2
```

where,

Query_Statement1 and Query_Statement2 are SELECT statements.

- The two rules that apply to INTERSECT operator are also applicable for EXCEPT operator.

EXCEPT Operator 2-2

- Following code snippet demonstrates the use of `EXCEPT`:

```
SELECT Product.ProductId FROM Production.Product
EXCEPT
SELECT ProductId FROM Sales.SalesOrderDetail
```

- If the order of the two tables in this example is interchanged, only those rows are returned from `Production`.
- `Product` table which do not match with the rows present in `Sales.SalesOrderDetail`.
- Thus, `EXCEPT` operator selects all the records from the first table except those which match with the second table.
- Hence, when you are using `EXCEPT` operator, the order of the two tables in the queries is important.
- Whereas, with the `INTERSECT` operator, it does not matter which table is specified first.

SQL Server 2016

GROUPING SETS 1-3

The `GROUPING SETS` operator allows you to group together multiple groupings of columns followed by an optional grand total row, denoted by parentheses, `()`.

It is more efficient to use `GROUPING SETS` operators instead of multiple `GROUP BY` with `UNION` clauses because the latter adds more processing overheads on the database server.

- The syntax of the `GROUPING SETS` operator is as follows:

Syntax:

```
GROUP BY
GROUPING SETS ( <grouping set list> )
```

where,

`grouping set list`: consists of one or more columns, separated by commas.

- A pair of parentheses, `()`, without any column name denotes grand total.

SQL Server 2016

GROUPING SETS 2-3

- Following code snippet demonstrates the `GROUPING SETS` operator.
- It assumes that a table **`Students`** is created with fields named **`Id`**, **`Name`**, and **`Marks`** respectively.

```
SELECT Id, Name, AVG(Marks) Marks
FROM Students
GROUP BY
GROUPING SETS
(
(Id, Name, Marks),
(Id),
()
)
```

SQL Server 2016

GROUPING SETS 3-3

- Following figure shows the output of the code:

	Id	Name	Marks
1	91	Sasha Goldsmith	78
2	91	NULL	78
3	92	Karen Hues	55
4	92	NULL	55
5	93	William Pinter	67
6	93	NULL	67
7	94	Yuri Gogol	89
8	94	NULL	89
9	NULL	NULL	72

- Here, the code uses `GROUPING SETS` to display average marks for every student.
- `NULL` values in **`Name`** indicate average marks for every student.
- `NULL` value in both **`Id`** and **`Name`** columns indicate grand total.

Summary

- The GROUP BY clause and aggregate functions enabled to group and/or aggregate data together in order to present summarized information.
- Spatial aggregate functions are newly introduced in SQL Server 2012.
- A subquery allows the result set of one SELECT statement to be used as criteria for another SELECT statement.
- Joins help you to combine column data from two or more tables based on a logical relationship between the tables.
- Set operators such as UNION and INTERSECT help you to combine row data from two or more tables.

About the Author

Created By: Mohammad Salman

Experience: 17 Years +

Designation: Corporate Trainer .NET

Microsoft CERTIFIED Professional

Microsoft Specialist

Microsoft CERTIFIED Solutions Developer

SQL SERVER

Session

Stored Procedures and Functions

Objectives

- Define stored procedures
- Explain the types of stored procedures
- Describe the procedure to create, alter, and execute stored procedures

Introduction

- An SQL Server database has two main categories of objects:
  - those that store data.
  - those that access, manipulate, or provide access to data.

- Views and stored procedures belong to this latter category.

SQL Server 2017

Stored Procedures

A stored procedure is a group of Transact-SQL statements that act as a single block of code that performs a specific task.

This block of code is identified by an assigned name and is stored in the database in a compiled form.

A stored procedure may also be a reference to a .NET Framework Common Language Runtime (CLR) method.

Stored procedures are useful when repetitive tasks have to be performed.

Stored procedures can accept values in the form of input parameters and return output values as defined by the output parameters.

- The advantages of using stored procedures are as follows:
  - Improved Security
  - Precompiled Execution
  - Reduced Client/Server Traffic
  - Reuse of Code

SQL Server 2017

Types of Stored Procedures 1-3

- SQL Server supports the following types of stored procedures:

User-Defined Stored Procedures

- User-defined stored procedures are also known as custom stored procedures.
- These procedures are used for reusing Transact-SQL statements for performing repetitive tasks.
- There are two types of user-defined stored procedures, the Transact-SQL stored procedures and the Common Language Runtime (CLR) stored procedures.
- Transact-SQL stored procedures consist of Transact-SQL statements whereas the CLR stored procedures are based on the .NET framework CLR methods.
- Both the stored procedures can take and return user-defined parameters.

SQL Server 2017

Types of Stored Procedures 2-3

Extended Stored Procedures

- Extended stored procedures help SQL Server in interacting with the operating system.
- Extended stored procedures are not resident objects of SQL Server.
- They are procedures that are implemented as dynamic-link libraries (DLL) executed outside the SQL Server environment.
- The application interacting with SQL Server calls the DLL at run-time. The DLL is dynamically loaded and run by SQL Server.
- SQL Server allots space to run the extended stored procedures.
- Extended stored procedures use the '`xp`' prefix.
- Tasks that are complicated or cannot be executed using Transact-SQL statements are performed using extended stored procedures.

Types of Stored Procedures 3-3

System Stored Procedures

System stored procedures are commonly used for interacting with system tables and performing administrative tasks such as updating system tables.

The system stored procedures are prefixed with '`sp_`'. These procedures are located in the Resource database.

These procedures can be seen in the sys schema of every system and user-defined database.

System stored procedures allow `GRANT`, `DENY`, and `REVOKE` permissions.

A system stored procedure is a set of pre-compiled Transact-SQL statements executed as a single unit.

System procedures are used in database administrative and informational activities.

When referencing a system stored procedure, the sys schema identifier is used.
System stored procedures are owned by the database administrator.

SQL Server 2017

Classification of System Stored Procedures 1-2

Catalog Stored Procedures

- All information about tables in the user database is stored in a set of tables called the system catalog.
- Information from the system catalog can be accessed using catalog procedures.
- For example, the `sp_tables` catalog stored procedure displays the list of all the tables in the current database.

Security Stored Procedures

- Security stored procedures are used to manage the security of the database.
- For example, the `sp_changedbowner` security stored procedure is used to change the owner of the current database.

Cursor Stored Procedures

- Cursor procedures are used to implement the functionality of a cursor.
- For example, the `sp_cursor_list` cursor stored procedure lists all the cursors opened by the connection and describes their attributes.

SQL Server 2017

Classification of System Stored Procedures 2-2

Distributed Query Stored Procedures

- Distributed stored procedures are used in the management of distributed queries.
- For example, the `sp_indexes` distributed query stored procedure returns index information for the specified remote table.

Database Mail and SQL Mail Stored Procedures

- Database Mail and SQL Mail stored procedures are used to perform e-mail operations from within the SQL Server.
- For example, the `sp_send_dbmail` database mail stored procedure sends e-mail messages to specified recipients.
- The message may include a query resultset or file attachments or both.

Temporary Stored Procedures

- Stored procedures created for temporary use within a session are called temporary stored procedures.
- These procedures are stored in the `tempdb` database.
- The `tempdb` system database is a global resource available to all users connected to an instance of SQL Server.
- SQL Server supports two types of temporary stored procedures namely, local and global.
- The table lists the differences between the two types of stored procedures.

Local Temporary Procedure	Global Temporary Procedure
Visible only to the user that created it	Visible to all
Dropped at the end of the current session	Dropped at the end of the last session
Local Temporary Procedure	Global Temporary Procedure
Can only be used by its owner	Can be used by any user
Uses the # prefix before the procedure name	Uses the ## prefix before the procedure name

Remote Stored Procedures

Stored procedures that run on remote SQL Servers are known as remote stored procedures.

Remote stored procedures can be used only when the remote server allows remote access.

When a remote stored procedure is executed from a local instance of SQL Server to a client computer, a statement abort error might be encountered.

When such an error occurs, the statement that caused the error is terminated but the remote procedure continues to be executed.

SQL Server 2017

Extended Stored Procedures

Extended stored procedures are used to perform tasks that are unable to be performed using standard Transact-SQL statements.

Extended stored procedures use the 'xp_' prefix. These stored procedures are contained in the dbo schema of the master database.

- The syntax used to execute an extended stored procedure is as follows:

Syntax:

```
EXECUTE <procedure_name>
```

- Following code snippet executes the extended stored procedure xp_fileexist to check whether the MyTest.txt file exists or not.

```
EXECUTE xp_fileexist 'c:\MyTest.txt'
```

SQL Server 2017

Custom or User-defined Stored Procedures 1-3

In SQL Server, users are allowed to create customized or user-defined stored procedures for performance of various tasks.

The `CREATE PROCEDURE` permission is required to create a procedure and the `ALTER` permission on the schema in which the procedure is being created.

- The syntax used to create a custom stored procedure is as follows:

Syntax:

```
CREATE { PROC | PROCEDURE } procedure_name
[ { @parameter data_type } ]
AS <sql_statement>
```

where,

`procedure_name`: specifies the name of the procedure.
`@parameter`: specifies the input/output parameters in the procedure.
`data_type`: specifies the data types of the parameters.
`sql_statement`: specifies one or more Transact-SQL statements to be included in the procedure.

Custom or User-defined Stored Procedures 2-3

- Following code snippet creates and then executes a custom stored procedure, **`uspGetCustTerritory`**, which will display the details of customers such as customer id, territory id, and territory name.

```
CREATE PROCEDURE uspGetCustTerritory
AS
SELECT TOP 10 CustomerID, Customer.TerritoryID,
Sales.SalesTerritory.Name
FROM Sales.Customer JOIN Sales.SalesTerritory ON
Sales.Customer.TerritoryID = Sales.SalesTerritory.TerritoryID
```

- The following code snippet executes the stored procedure using the `EXEC` command.

```
EXEC uspGetCustTerritory
```

Custom or User-defined Stored Procedures 3-3

- The output is shown in the following figure:

	CustomerID	TerritoryID	Name
1	15	9	Australia
2	33	9	Australia
3	51	9	Australia
4	69	9	Australia
5	87	9	Australia
6	105	9	Australia
7	123	9	Australia
8	141	9	Australia
9	159	9	Australia
10	177	9	Australia

SQL Server 2017

Using Parameters

The real advantage of a stored procedure comes into picture only when one or more parameters are used with it.

Data is passed between the stored procedure and the calling program when a call is made to a stored procedure.

- This data transfer is done using parameters. Parameters are of two types that are as follows:

Input Parameters	• Input parameters allow the calling program to pass values to a stored procedure. • These values are accepted into variables defined in the stored procedure.
Output Parameters	• Output parameters allow a stored procedure to pass values back to the calling program. • These values are accepted into variables by the calling program.

SQL Server 2017

Input Parameters 1-2

- Values are passed from the calling program to the stored procedure and these values are accepted into the input parameters of the stored procedure.
- The input parameters are defined at the time of creation of the stored procedure.
- The values passed to input parameters could be either constants or variables.
- These values are passed to the procedure at the time of calling the procedure.
- The stored procedure performs the specified tasks using these values.
- The syntax used to create a stored procedure is as follows:

Syntax:

```
CREATE PROCEDURE <procedure_name>
@parameter <data_type>
AS <sql_statement>
```

where,

`data_type`: specifies the system defined data type.

- The syntax used to execute a stored procedure and pass values as input parameters is as follows:

Syntax:

```
EXECUTE <procedure_name> <parameters>
```

SQL Server 2017

Input Parameters 2-2

- Following code snippet creates a stored procedure, **uspGetSales** with a parameter territory to accept the name of a territory and display the sales details and salesperson id for that territory.
- Then, the code executes the stored procedure with Northwest being passed as the input parameter.

```
CREATE PROCEDURE uspGetSales
@territory varchar(40)
AS
SELECT BusinessEntityID, B.SalesYTD, B.SalesLastYear
FROM Sales.SalesPerson A
JOIN Sales.SalesTerritory B
ON A.TerritoryID = B.TerritoryID
WHERE B.Name = @territory;
--Execute the stored procedure
EXEC uspGetSales 'Northwest'
```

- The output is shown in the following figure:

	BusinessEntityID	SalesYTD	SalesLastYear
1	280	7887186.7882	3298694.4938
2	283	7887186.7882	3298694.4938
3	284	7887186.7882	3298694.4938

SQL Server 2017

Output Parameters 1-3

Output parameters are defined at the time of creation of the procedure.

To specify an output parameter, the `OUTPUT` keyword is used while declaring the parameter.

Also, the calling statement has to have a variable specified with the `OUTPUT` keyword to accept the output from the called procedure.

- The following syntax is used to pass output parameters in a stored procedure and then, execute the stored procedure with the `OUTPUT` parameter.

Syntax:

```
EXECUTE <procedure_name> <parameters>
```

SQL Server 2017

Output Parameters 2-3

- Following code snippet creates a stored procedure, `uspGetTotalSales` with input parameter `@territory` to accept the name of a territory and output parameter `@sum` to display the sum of sales year to date in that territory.

```
CREATE PROCEDURE uspGetTotalSales
@territory varchar(40), @sum int OUTPUT
AS
SELECT @sum= SUM(B.SalesYTD)
FROM Sales.SalesPerson A
JOIN Sales.SalesTerritory B
ON A.TerritoryID = B.TerritoryID
WHERE B.Name = @territory
```

- Following code snippet declares a variable `sumsales` to accept the output of the procedure `uspGetTotalSales`.

```
DECLARE @sumsales money;
EXEC uspGetTotalSales 'Northwest', @sum = @sum OUTPUT;
PRINT 'The year-to-date sales figure for this territory is ' +
convert(varchar(100),@sumsales);
GO
```

SQL Server 2017

Output Parameters 3-3

- `OUTPUT` parameters have the following characteristics:
  - The parameter cannot be of text and image data type.
  - The calling statement must contain a variable to receive the return value.
  - The variable can be used in subsequent Transact-SQL statements in the batch or the calling procedure.
  - Output parameters can be cursor placeholders.
- The `OUTPUT` clause returns information from each row on which the `INSERT`, `UPDATE`, and `DELETE` statements have been executed.
- This clause is useful to retrieve the value of an identity or computed column after an `INSERT` or `UPDATE` operation.

SQL Server 2017

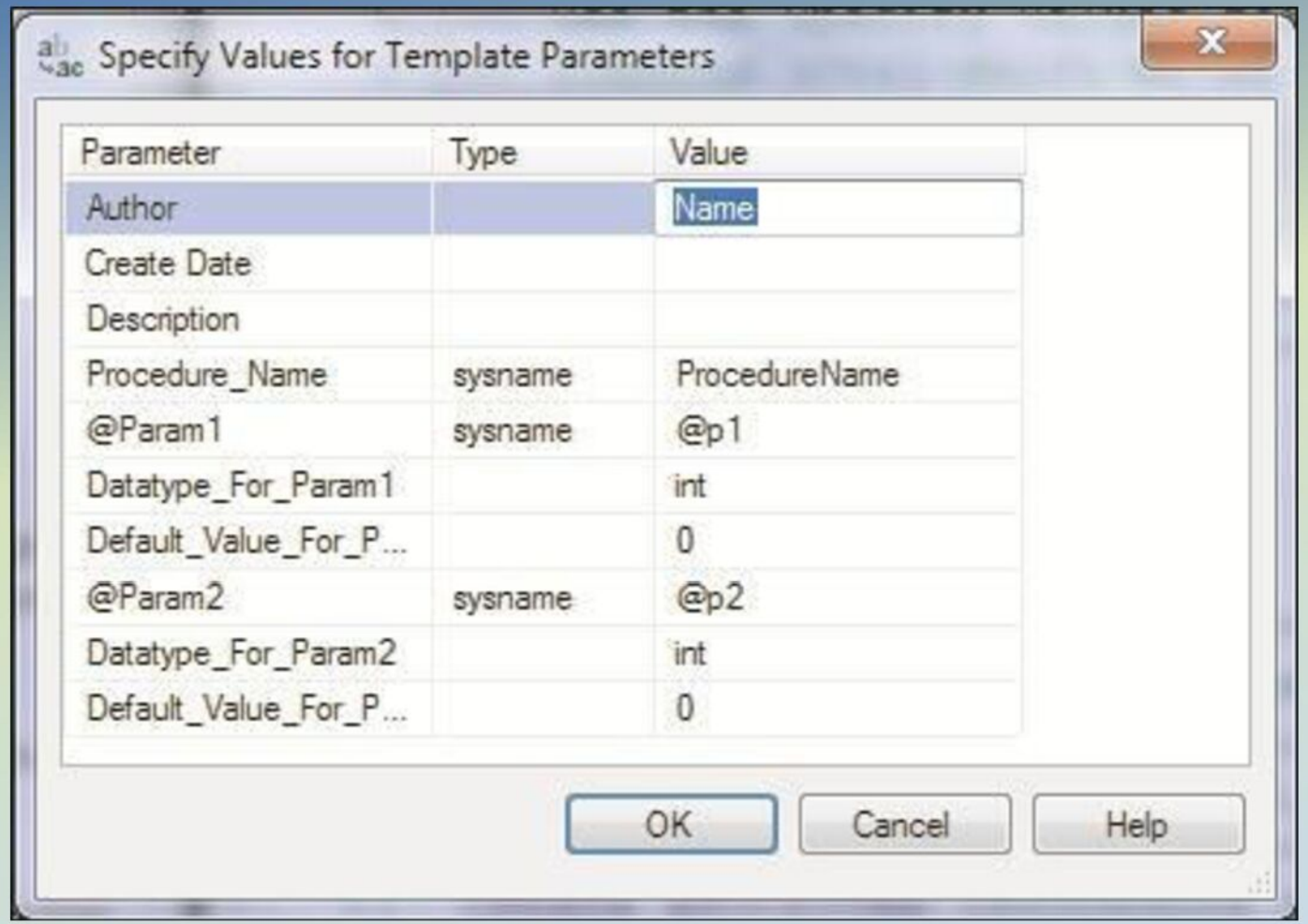
Using SSMS to Create Stored Procedures 1-5

- You can also create a user-defined stored procedure using SSMS by performing the following steps:

1. Launch **Object Explorer**.
2. In **Object Explorer**, connect to an instance of Database Engine.
3. After successfully connecting to the instance, expand that instance.
4. Expand **Databases** and then, expand the **AdventureWorks2012** database.
5. Expand **Programmability**, right-click **Stored Procedures**, and then, click **New Stored Procedure**.
6. On the **Query** menu, click **Specify Values for Template Parameters**. The **Specify Values for Template Parameters** dialog box is displayed.

Using SSMS to Create Stored Procedures 2-5

- This is shown in the following figure:



SQL Server 2017

Using SSMS to Create Stored Procedures 3-5

7

- In the **Specify Values for Template Parameters** dialog box, enter the values for the parameters as shown in the following table:

Parameter	Value
Author	Your name
Create Date	Today's date
Description	Returns year to sales data for a territory
Procedure_Name	uspGetTotals
@Param1	@territory
@Datatype_For_Param1	varchar(50)
Default_Value_For_Param1	NULL
@Param2	
@Datatype_For_Param2	
Default_Value_For_Param2	

8

- After entering these details, click **OK**.

SQL Server 2017

Using SSMS to Create Stored Procedures 4–5

9 • In the Query Editor, replace the `SELECT` statement with the following statement:

```
SELECT BusinessEntityID, B.SalesYTD, B.SalesLastYear
FROM Sales.SalesPerson A
JOIN Sales.SalesTerritory B
ON A.TerritoryID = B.TerritoryID
WHERE B.Name = @territory;
```

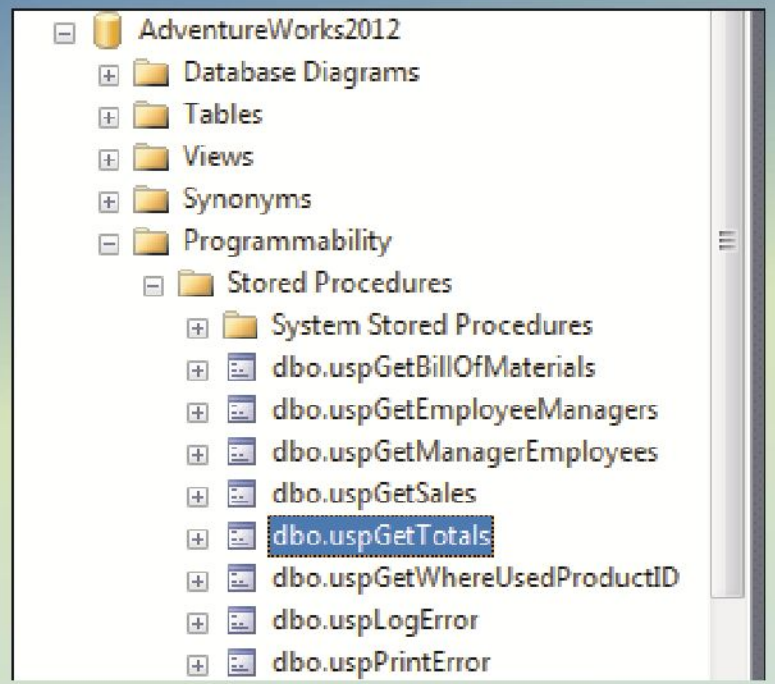
10 • To test the syntax, on the **Query** menu, click **Parse**. If an error message is returned, compare the statements with the information and correct as needed.

11 • To create the procedure, from the **Query** menu, click **Execute**. The procedure is created as an object in the database.

12 • To see the procedure listed in **Object Explorer**, right-click **Stored Procedures** and select **Refresh**.

Using SSMS to Create Stored Procedures 5-5

- The procedure name will be displayed in the **Object Explorer** tree as shown in the following figure:



10. To run the procedure, in **Object Explorer**, right-click the stored procedure name `uspGetTotals` and select **Execute Stored Procedure**.
11. In the **Execute Procedure** window, enter **Northwest** as the value for the parameter `@territory`.

SQL Server 2017

Viewing Stored Procedure Definitions

- The definition of a stored procedure can be viewed using the sp_helptext system stored procedure.
- To view the definition, you must specify the name of the stored procedure as the parameter when executing sp_helptext.
- This definition is in the form of Transact-SQL statements.
- The Transact-SQL statements of the procedure definition include the `CREATE PROCEDURE` statement as well as the SQL statements that define the body of the procedure.
- The syntax used to view the definition of a stored procedure is as follows:

Syntax:

```
sp_helptext '<procedure_name>'
```

- Following code snippet displays the definition of the stored procedure named **`uspGetTotals`**.

```
EXEC sp_helptext uspGetTotals
```

SQL Server 2017

Modifying and Dropping Stored Procedures 1-2

The permissions associated with the stored procedure are lost when a stored procedure is re-created.

When a stored procedure is altered, the permissions defined for the stored procedure remain the same even though the procedure definition is changed.

A procedure can be altered using the `ALTER PROCEDURE` statement.

- The syntax used to modify a stored procedure is as follows:

Syntax:

```
ALTER PROCEDURE <procedure_name>
@parameter <data_type> [ OUTPUT ]
[ WITH { ENCRYPTION | RECOMPILE } ]
AS <sql_statement>
```

where,

`ENCRYPTION`: encrypts the stored procedure definition.

`RECOMPILE`: indicates that the procedure is compiled at run-time.

`sql_statement`: specifies the Transact-SQL statements to be included in the body of the procedure.

Modifying and Dropping Stored Procedures 2-2

- Following code snippet modifies the definition of the stored procedure named **uspGetTotals** to add a new column **CostYTD** to be retrieved from **Sales.SalesTerritory**.

```
ALTER PROCEDURE [dbo].[uspGetTotals]
@territory varchar = 40
AS
SELECT BusinessEntityID, B.SalesYTD, B.CostYTD, B.SalesLastYear
FROM Sales.SalesPerson A
JOIN Sales.SalesTerritory B
ON A.TerritoryID = B.TerritoryID
WHERE B.Name = @territory;
GO
```

Guidelines for Using ALTER PROCEDURE Statement

When a stored procedure is created using options such as the `WITH ENCRYPTION` option, these options should also be included in the `ALTER PROCEDURE` statement.

The `ALTER PROCEDURE` statement alters a single procedure.
When a stored procedure calls other stored procedures, the nested stored procedures are not affected by altering the calling procedure.

The creators of the stored procedure, members of the `sysadmin` server role and members of the `db_owner` and `db_ddladmin` fixed database roles have the permission to execute the `ALTER PROCEDURE` statement.

It is recommended that you do not modify system stored procedures. If you need to change the functionality of a system stored procedure, then create a user-defined system stored procedure by copying the statements from an existing stored procedure and modify this user-defined procedure.