# Session 18

SQL Server 2016

# Enhancements in SQL Server 2016

# Objectives

- Describe the importance of configuring tempdb database
- files
- Describe how to configure tempdb during installation of an
- instance of SQL Server 2016
- Outline the allocation and autogrowth of tempdb files in
- SQL Server 2016
- List the differences between the two execution plans
- Explain how to enable system-versioning on tables

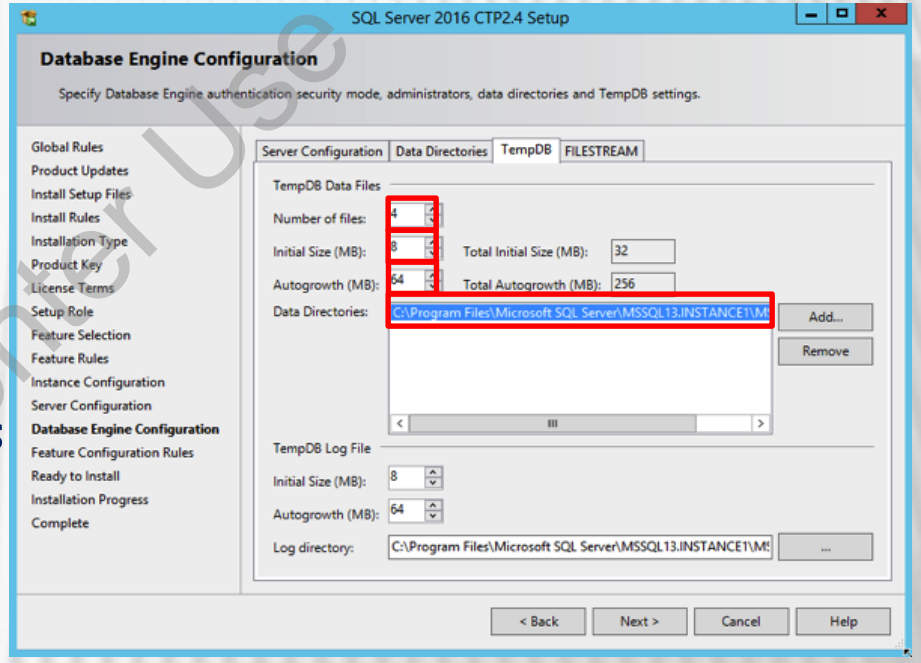
# Importance of tempdb Configuration


- The tempdb is a system database used to store local and global temporary tables, temporary procedures, table variables, and cursors.
- SQL Server 2016 allows you to configure the tempdb database during the installation of the SQL Server instance.
- Accurate configuration of tempdb is important to ensure that performance is not impacted.
- Multiple tempdb files should be configured with the same size to ensure that 'writes' are evenly distributed across a set of data files.

# Enhancements to tempdb Configuration 1-2

SQL Server 2016

- By default, the setup process adds as many tempdb files as the CPU Count, with a maximum of eight files.
- Following parameters can be specified during setup:
  - Number of tempdb database files
  - Initial size of the files, autogrowth, and directory placement
  - Initial size of the log files, autogrowth, and directory placement
- Multiple volumes or directories for the tempdb database files.



# Enhancements to tempdb Configuration 2-2


- Number of tempdb database files are based on number of available CPU cores.
- If the number of CPU cores is less than or equal to eight, then the number of tempdb files configured is equal to the number of CPU cores.
- If the number of CPU cores is greater than eight, then the number of tempdb files configured is only eight.

| Number of Available CPU Cores | Number of tempdb Data Files |
|---|---|
| 2 | 2 |
| 4 | 4 |
| 8 | 8 |
| 32 | 8 |

# Handling Allocations and Autogrowth

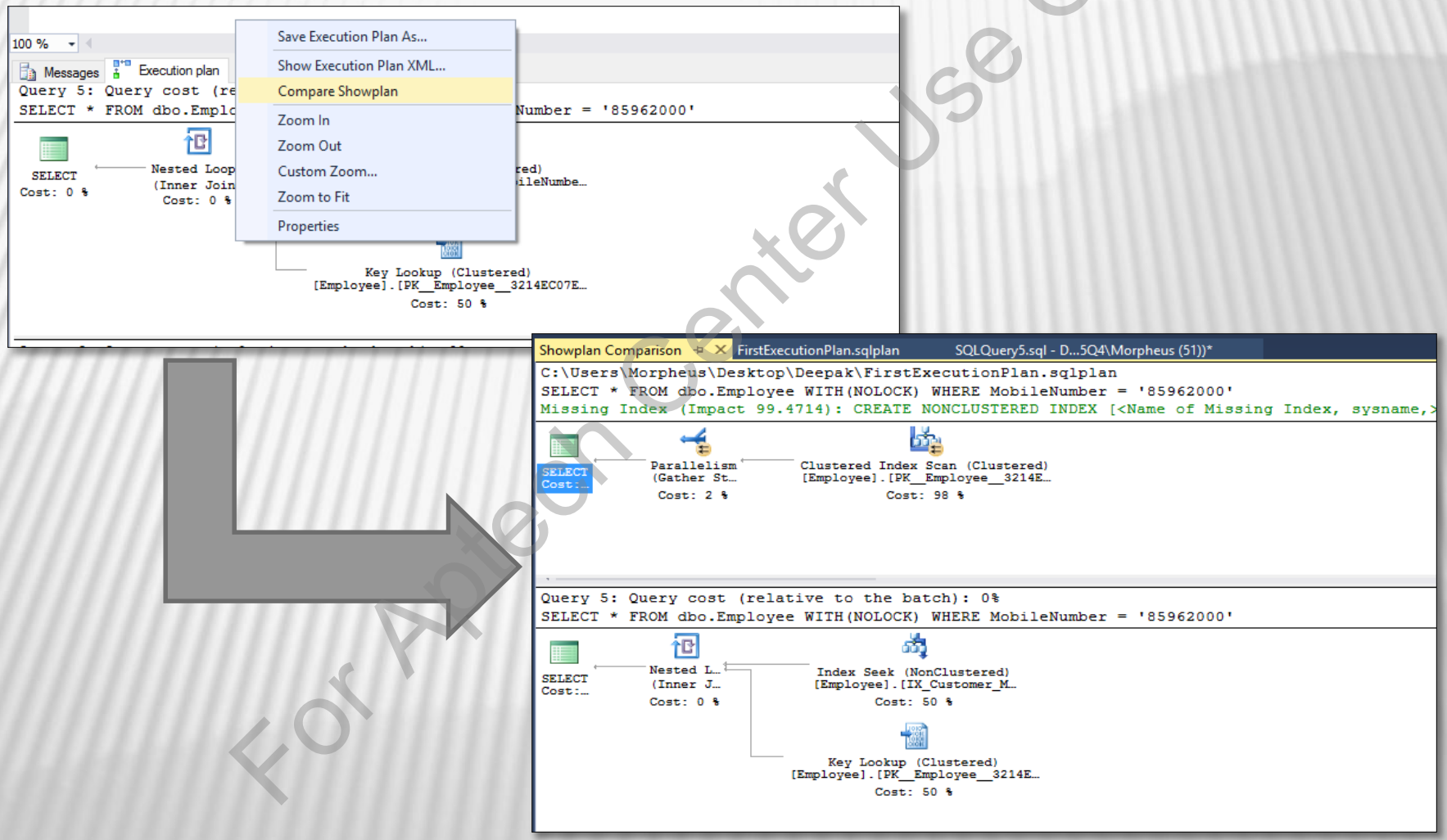

- Enhancements in SQL Server 2016 as compared to previous versions:

| Earlier SQL Server Versions | SQL Server 2016 |
| --- | --- |
| • Used trace 1117 and 1118 flags to define allocation of pages in databases and handling of autogrowth across multiple data files<br>• First eight pages for a database object allocated to Mixed extent (64KB)<br>• Ninth page onwards allocated to Uniform extent (64KB) | • No need to use trace flags<br>• Mixed extents are not used<br>• Temp table pages are always allocated directly to Uniform extents |

# Comparing Execution Plans 1-2


- Is useful to test differences in execution when changes are made.
- Helps to troubleshoot and debug issues in production environment by comparing with test environment.
- In SQL Server 2016, you can save two execution plans and compare them by using the Compare Showplan option.
- Properties of nodes can be compared.
- Colored highlights help to identify differences easily.

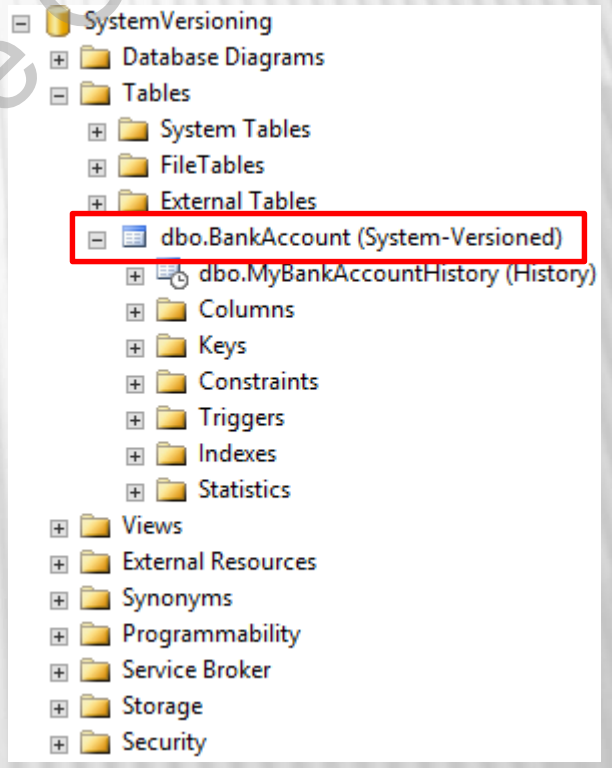
# Comparing Execution Plans 2-2



# Temporal Tables and System-Versioning

SQL Server 2016

- In SQL Server 2016, temporal tables that are system-versioned tables allowing storage of different versions of rows.
- A table can be enabled to be 'System-Versioned'.
- A history table is created and linked to the base table.
- When the base table is updated, the older version of data is moved to the history table.
- These different versions of the data or 'Time-Specific' data can be retrieved by specifying required time parameters.



# Creating System-Versioned Tables


| | |
|---|---|
| Create a table and add columns to store 'From-date' and 'To-date' of the record | • Add two **datetime2** columns<br>• Include keyword **PERIOD FOR SYSTEM_TIME** |
| Enable System Versioning on the table | • Use **ENABLE SYSTEM_VERSIONING = ON** |

# Example to Create System-Versioned Table

SQL Server 2016

```
ALTER TABLE dbo.ExampleTable
   ADD ValidFrom datetime2 GENERATED ALWAYS AS ROW START
HIDDEN NOT NULL,
   ValidTo datetime2 GENERATED ALWAYS AS ROW END HIDDEN
NOT NULL,
   PERIOD FOR SYSTEM_TIME (ValidFrom, ValidTo);
```

```
ALTER TABLE dbo.ExampleTable
        SET (SYSTEM_VERSIONING = ON (HISTORY_TABLE =
dbo.ExampleHistoryTable));
```

```
INSERT INTO dbo.ExampleTable
 (ExampleNumber, ExampleValue)
 VALUES
 (20500, 90100)
```

# Querying Time-Specific Data 1-2


- The '**FOR SYSTEM_TIME'** clause is used to retrieve time specific data as follows:

| To retrieve data at a specific time | **FOR SYSTEM_TIME AS OF** |
|---|---|

**Example: SELECT * FROM dbo.BankAccount FOR SYSTEM_TIME AS OF @DateTimeInHistory**

| To retrieve data in 'from' and 'to' times | **FOR SYSTEM_TIME FROM** |
|---|---|

```
SELECT PrimaryAccountNumber, CurrentAccountBalance, ValidFrom, ValidTo
FROM dbo.BankAccount
FOR  SYSTEM_TIME  FROM '2016-07-04 06:26:00' TO '2016-07-04 06:28:00'
```

# Querying Time-Specific Data 2-2

SQL Server 2016

- The '**FOR SYSTEM_TIME'** clause is used to retrieve time specific data as follows:

| To retrieve data contained in a time range | FOR SYSTEM_TIME CONTAINED IN |
|---|---|

```
Example: SELECT PrimaryAccountNumber, CurrentAccountBalance, ValidFrom, ValidTo
FROM dbo.BankAccount
FOR SYSTEM_TIME CONTAINED IN ('2016-07-04 06:26:00', '2016-07-04 06:28:00')
```

| To retrieve data between two times | FOR SYSTEM_TIME BETWEEN |
|---|---|

```
Example: SELECT PrimaryAccountNumber, CurrentAccountBalance, ValidFrom, ValidTo
FROM dbo.BankAccount
FOR SYSTEM_TIME BETWEEN '2016-07-04 06:26:00' AND '2016-07-04 06:28:00'
```

# Making Schema Changes to System-Versioned Tables 1-2


- To make changes to system-versioned tables :

1. Disable System-Versioning on the table.

```
ALTER TABLE dbo.ExampleTable SET (SYSTEM_VERSIONING = OFF)
GO
```

2. Make the same change to the history table.

```
ALTER TABLE dbo.ExampleHistoryTable
ADD NewColumn VARCHAR(10)
 GO
```

3. Enable System-Versioning on the table again.

```
ALTER TABLE dbo.ExampleTable
SET (SYSTEM_VERSIONING = ON (HISTORY_TABLE =
dbo.ExampleHistoryTable))
GO
```

# Making Schema Changes to System-Versioned Tables 2-2


- Disabling system-versioning for a short period means some historical data loss.
- Disabling the system-versioning would not result in removal of historical data.
- Temporal tables can also be indexed in the same way as normal tables.
- If data on temporal tables need to be removed at any time, it can only be done by executing a simple query that would go through the temporal tables and delete the records.

# Summary


- Enhancements in SQL Server 2016 include:
  - Configuring tempdb databases during setup: Accurate sizing of tempdb is important for optimal system performance. SQL Server 2016 simplifies this process with default allocation of the number of tempdb files as well as their initial sizes.
  - Comparing two execution plans: SQL Server 2016 provisions easy comparison of changes in execution plans and their impact with easy-to-analyze highlights.
  - Retrieving time-specific versions of data through temporal or history tables: SQL Server 2016 enables storing table data in different versions which helps in case of a need to roll-back to a previous version.
- SQL Server 2016 also eliminates the need to use Trace Flags 1117 and 1118.
- Comparing two execution plans proves useful to troubleshoot errors in production environment.
- Time-specific data retrieval that is made possible in SQL Server 2016 is also very useful to analyze data for business decisions.