Intelligent Data Management with SQL Server

Session: 10

Views, Stored Procedures, and Querying Metadata

Objectives

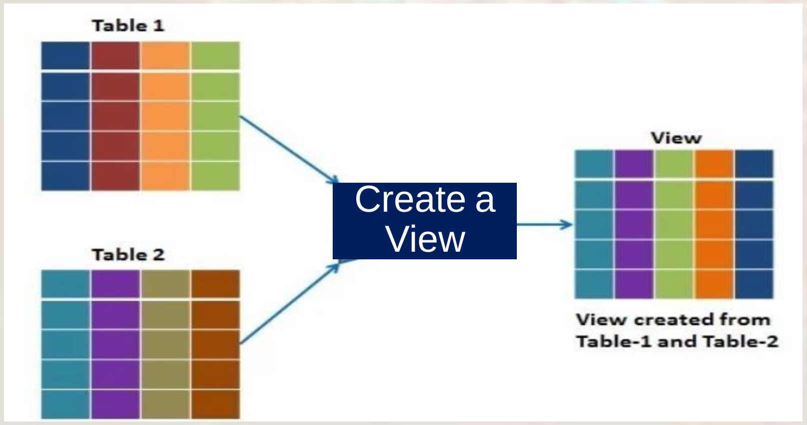
- Define views
- Describe the technique to create, alter, and drop views
- Define stored procedures and its types
- Describe the procedure to create, alter, and execute stored procedures
- Describe nested stored procedures
- Describe querying SQL Server metadata System Catalog views and functions

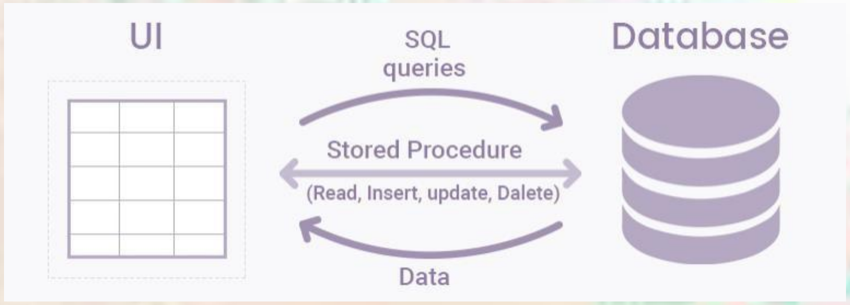
Introduction

- 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 the latter category.





Views

A view is a virtual table that is made up of selected columns from one or more tables.
.

The tables from which the view is created are referred to as base tables.

A view can also include columns from other views created in the same or a different database.

A view can have a maximum of 1,024 columns.

Creating Views

A user can create a view using columns from tables or other views only if the user has permission to access these tables and views.

A view is created using CREATE VIEW statement and it can be created only in the current database.

SQL Server verifies the existence of objects that are referenced in the view definition.

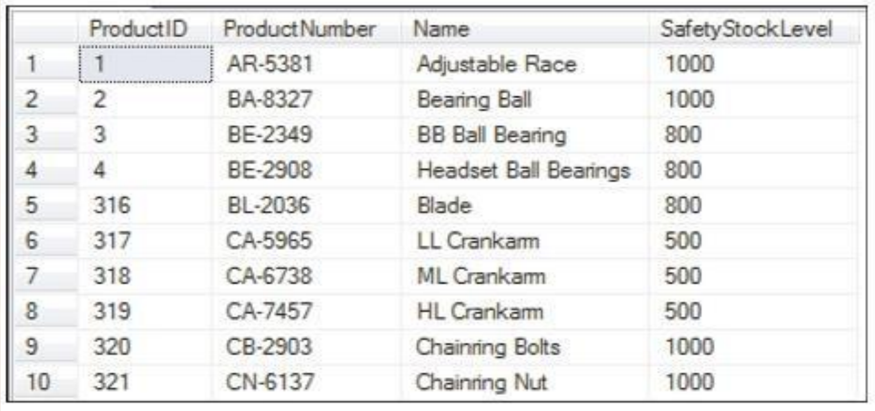
Syntax:

```
CREATE VIEW <view_name>
  AS <select_statement>
```

- For example

```
CREATE VIEW vwProductInfo AS
  SELECT ProductID,
  ProductName,
  SafetyStockLevel
  FROM Production.Product;
GO

SELECT * FROM vwProductInfo
GO
```



	ProductID	ProductNumber	Name	SafetyStockLevel
1	1	AR-5381	Adjustable Race	1000
2	2	BA-8327	Bearing Ball	1000
3	3	BE-2349	BB Ball Bearing	800
4	4	BE-2908	Headset Ball Bearings	800
5	316	BL-2036	Blade	800
6	317	CA-5965	LL Crankarm	500
7	318	CA-6738	ML Crankarm	500
8	319	CA-7457	HL Crankarm	500
9	320	CB-2903	Chainring Bolts	1000
10	321	CN-6137	Chainring Nut	1000

Records from a View

Creating Views Using JOIN Keyword 1-3

The JOIN keyword can also be used to create views.

The CREATE VIEW statement is used along with JOIN keyword to create a view using columns from multiple tables.

Syntax:

```
CREATE VIEW <view_name> AS
SELECT * FROM table_name1 JOIN
table_name2
ON table_name1.column_name = table_name2.column_name
```

where,

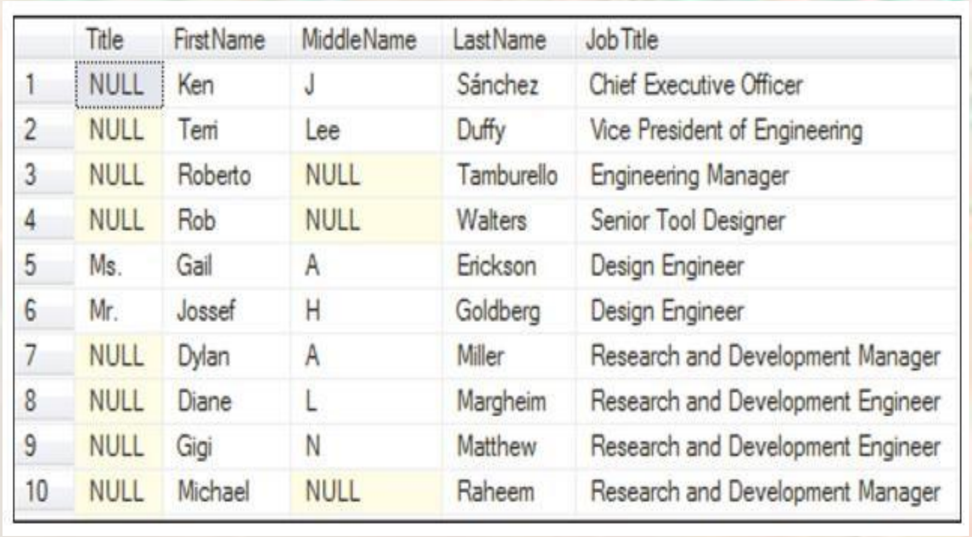
view_name: specifies the name of the view.
table_name1: specifies the name of first table.
JOIN: specifies that two tables are joined using JOIN keyword.
table_name2: specifies the name of the second table.

Creating Views Using JOIN Keyword 2-3

Code Snippet 3:

```
CREATE VIEW vwPersonDetails  AS
SELECT
   p.Title
  ,p.[FirstName]
  ,p.[MiddleName]
  ,p.[LastName]
  ,e.[JobTitle]
FROM [HumanResources].[Employee] e
     INNER JOIN [Person].[Person] p
     ON p.[BusinessEntityID] = e.[BusinessEntityID]
GO
```

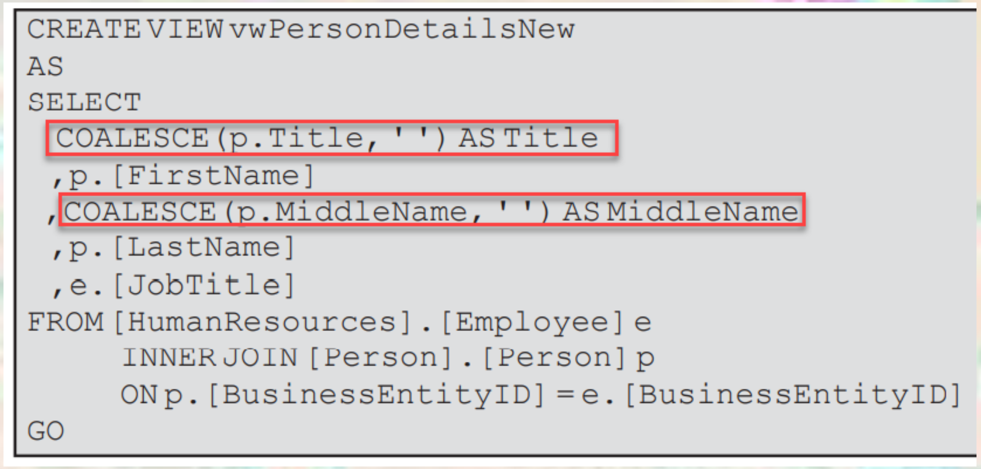
```
SELECT * FROM vwPersonDetails
```



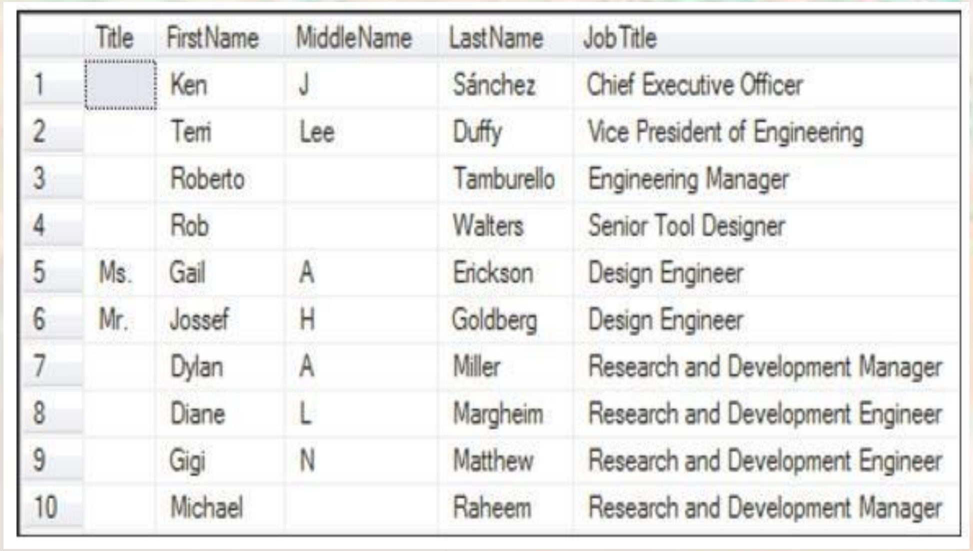
	Title	FirstName	MiddleName	LastName	JobTitle
1	NULL	Ken	J	Sánchez	Chief Executive Officer
2	NULL	Terri	Lee	Duffy	Vice President of Engineering
3	NULL	Roberto	NULL	Tamburello	Engineering Manager
4	NULL	Rob	NULL	Walters	Senior Tool Designer
5	Ms.	Gail	A	Erickson	Design Engineer
6	Mr.	Jossef	H	Goldberg	Design Engineer
7	NULL	Dylan	A	Miller	Research and Development Manager
8	NULL	Diane	L	Margheim	Research and Development Engineer
9	NULL	Gigi	N	Matthew	Research and Development Engineer
10	NULL	Michael	NULL	Raheem	Research and Development Manager

Creating Views Using JOIN Keyword 3-3

- To replace all the NULL valuesin the output with a nullstring, the COALESCE() function



```
CREATE VIEW vwPersonDetailsNew
AS
SELECT
  COALESCE(p.Title, '') AS Title
  ,p.[FirstName]
  ,COALESCE(p.MiddleName, '') AS MiddleName
  ,p.[LastName]
  ,e.[JobTitle]
FROM [HumanResources].[Employee] e
     INNER JOIN [Person].[Person] p
     ON p.[BusinessEntityID] = e.[BusinessEntityID]
GO
```



	Title	FirstName	MiddleName	LastName	JobTitle
1		Ken	J	Sánchez	Chief Executive Officer
2		Terri	Lee	Duffy	Vice President of Engineering
3		Roberto		Tamburello	Engineering Manager
4		Rob		Walters	Senior Tool Designer
5	Ms.	Gail	A	Erickson	Design Engineer
6	Mr.	Jossef	H	Goldberg	Design Engineer
7		Dylan	A	Miller	Research and Development Manager
8		Diane	L	Margheim	Research and Development Engineer
9		Gigi	N	Matthew	Research and Development Engineer
10		Michael		Raheem	Research and Development Manager

Using COALESCE Function in a View

Guidelines and Restrictions on Views

A view is created only in current database. The base tables and views from which the view is created can be from other databases or servers.

View names must be unique and cannot be same as table names in the schema.

A view cannot be created on temporary tables.

A view cannot have a full-text index.

A view cannot contain DEFAULT definition.

The CREATE VIEW statement can include ORDER BY clause only if TOP keyword is used.

Views cannot reference more than 1,024 columns.

The CREATE VIEW statement cannot include INTO keyword.

The CREATE VIEW statement cannot be combined with other Transact-SQL statements in a single batch.

Modifying Data through Views

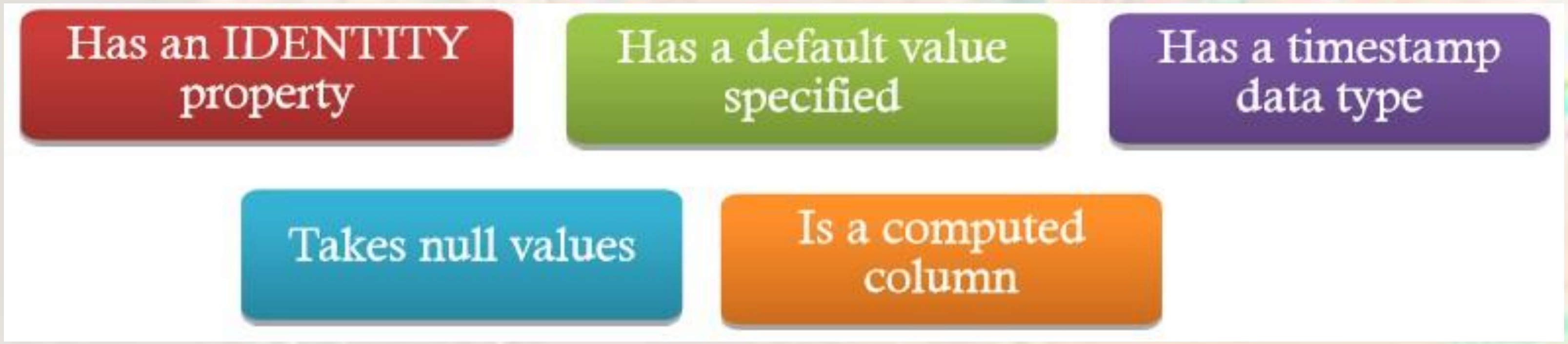
Views can be used to modify data in database tables.

Data can be inserted, modified, or deleted through a view using following statements:

- INSERT
- UPDATE
- DELETE

INSERT with Views

- The INSERT statement is used to add a new row to a table or a view.
- During the execution of the statement, if the value for a column is not provided, the SQL Server Database Engine must provide a value based on the definition of the column.
- The value for the column is provided automatically if the column:



INSERT with Views

- Example

```
USE StrongHold
CREATE TABLE Employee_Personal_Details (
EmpID int NOT NULL,
FirstName varchar(30)NOT NULL,
LastName varchar(30) NOT NULL,
Address varchar(30) NULL
)
CREATE TABLE Employee_Salary_Details (
EmpID int NOT NULL,
Designation varchar(30),
Salary int NOT NULL
)
CREATE VIEW vwEmployee_Personal_Details
AS
SELECT e1.EmpID,FirstName,LastName, Designation,Salary
FROM Employee_Personal_Details e1
JOIN Employee_Salary_Details e2
ON e1.EmpID = e2.EmpID
CREATE VIEW vwEmpDetails AS
SELECT *
FROM Employee_Personal_Details
INSERT INTO vwEmpDetails VALUES (1,'Jack','NYC','123 Florida')
```

UPDATE with Views

UPDATE statement can be used to change data in a view.

Updating a view also updates underlying table.

The value of a column with an IDENTITY property cannot be updated.

Records cannot be updated if the base table contains a TIMESTAMP column.

When there is a self-join with the same view or base table, the UPDATE statement does not work.

While updating a row, if a constraint or rule is violated, the statement is terminated, an error is returned, and no records are updated.

UPDATE with Views

- Example

```
USE StrongHold
CREATE TABLE Product_Details (
ProductID int,
ProductName varchar(30),
Rate money
)

INSERT INTO Product_Details
VALUES ('1','DVD Writer',2250),
       ('2','External Hard Drive',4250)

CREATE VIEW vwProduct_Details
AS
SELECT ProductName, Rate
FROM Product_Details

UPDATE vwProduct_Details
SET Rate=3000
WHERE ProductName='DVD Writer'
```

DELETE with Views

- SQL Server enables you to delete rows from a view. Rows can be deleted from the view using the DELETE statement.
- When rows are deleted from a view, corresponding rows are also deleted from the base table.

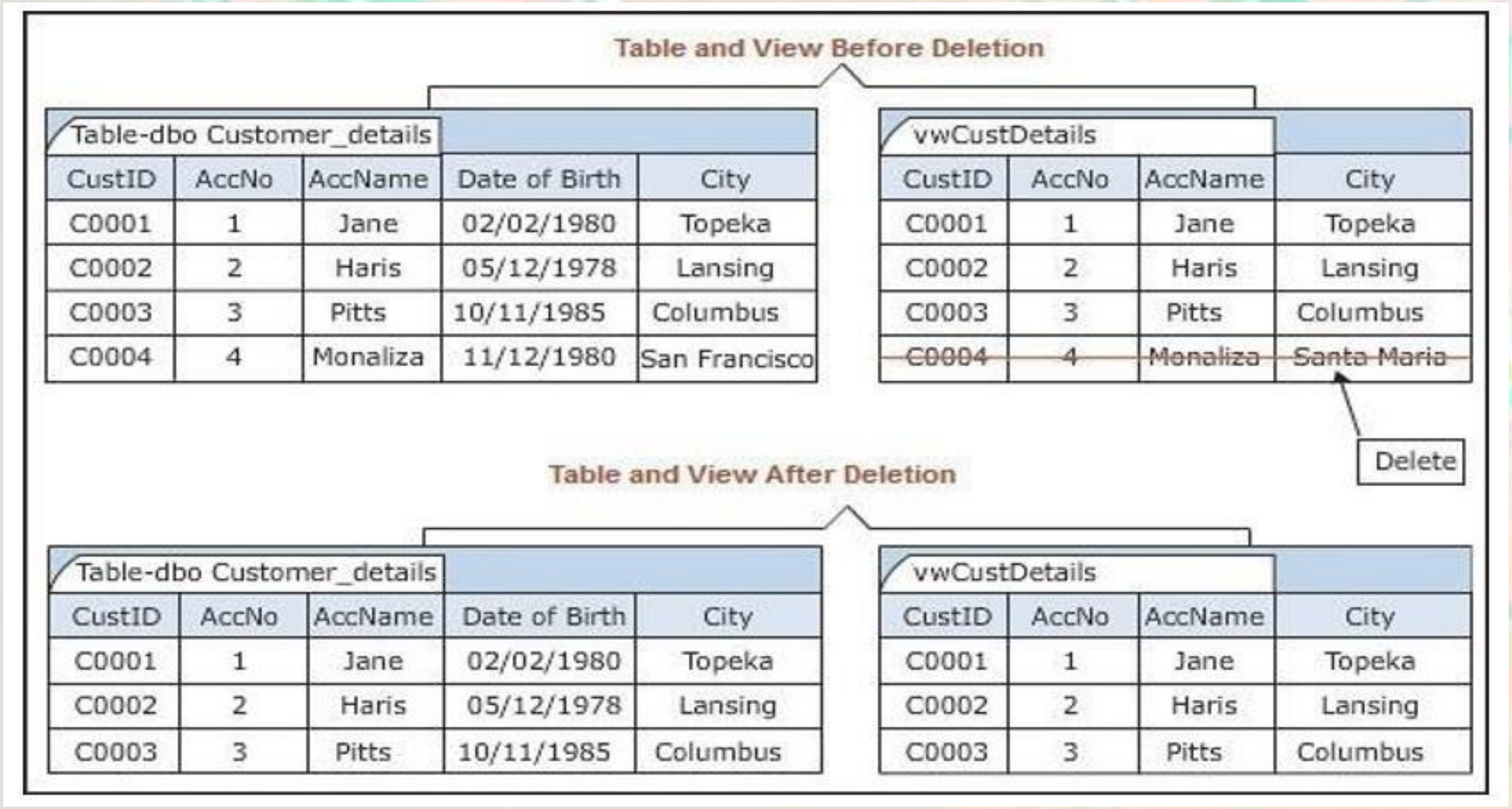


Table-dbo Customer_details (before deletion)

CustID	AccNo	AccName	Date of Birth	City
C0001	1	Jane	02/02/1980	Topeka
C0002	2	Haris	05/12/1978	Lansing
C0003	3	Pitts	10/11/1985	Columbus
C0004	4	Monaliza	11/12/1980	San Francisco

vwCustDetails (before deletion)

CustID	AccNo	AccName	City
C0001	1	Jane	Topeka
C0002	2	Haris	Lansing
C0003	3	Pitts	Columbus
~~C0004~~	~~4~~	~~Monaliza~~	~~Santa Maria~~

Table-dbo Customer_details (after deletion)

CustID	AccNo	AccName	Date of Birth	City
C0001	1	Jane	02/02/1980	Topeka
C0002	2	Haris	05/12/1978	Lansing
C0003	3	Pitts	10/11/1985	Columbus

vwCustDetails (after deletion)

CustID	AccNo	AccName	City
C0001	1	Jane	Topeka
C0002	2	Haris	Lansing
C0003	3	Pitts	Columbus

Deleting from Views

DELETE with Views

- Example

```
USE StrongHold
GO

CREATE VIEW vwProductDetails
AS
SELECT ProductID
FROM Product_Details
WHERE ProductID=1

GO
DELETE FROM vwProductDetails

GO
SELECT *
FROM Product_Details
```

Deleting from Views

Altering Views

- A view can be modified or altered by dropping and recreating it or executing the ALTER VIEW statement.

- ALTER VIEW can be applied to indexed views; however, it unconditionally drops all indexes on the view.

- Views are often altered when a user requests for additional information or makes changes in the underlying table definition.

Altering Views

- Syntax:

```
ALTER VIEW <view_name>
 AS <select_statement>
```

- Example

```
ALTER VIEW vwProduct_Details
AS
SELECT ProductID,ProductName, Rate
FROM Product_Details

GO

SELECT *
FROM vwProduct_Details
GO
```

Dropping Views

A view can be removed from the database if it is no longer required. This is done using the DROP VIEW statement.

The definition of the view and other information associated with the view is deleted from the system catalog.

- The syntax :

```
DROP VIEW <view_name>
```

- Example

```
DROP VIEW vwProduct_Details
```

Definition of a View

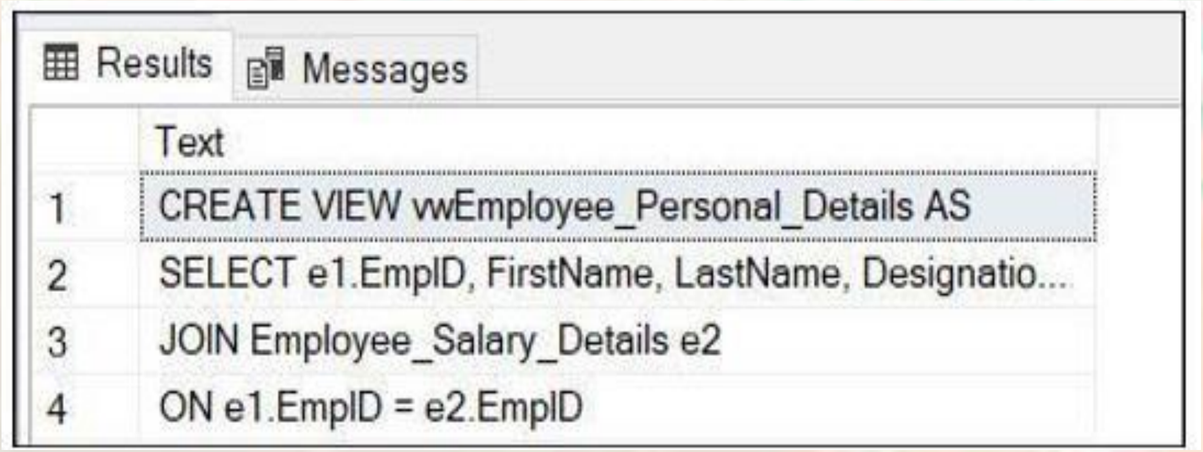
- The definition of a view helps to understand how its data is derived from the source tables.

Syntax:

```
sp_helptext <view_name>
```

Example:

```
EXEC sp_helptext vwProductPrice
```



```
Results   Messages
     Text
1    CREATE VIEW vwEmployee_Personal_Details AS
2    SELECT e1.EmpID, FirstName, LastName, Designatio...
3    JOIN Employee_Salary_Details e2
4    ON e1.EmpID = e2.EmpID
```

Using sp_helptext to display View Definitions

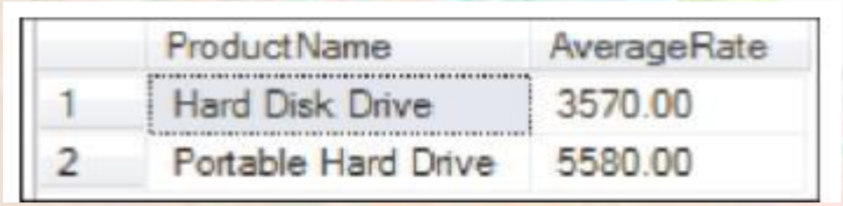
Creating a View Using Built-in Functions

Views can be created using built-in functions of SQL Server.

When functions are used, the derived column must include the column name in the CREATE VIEW statement.

```
CREATE VIEW vwProduct_Details AS
SELECT ProductName, AVG(Rate) AS
AverageRate
FROM Product_Details
GROUP BY ProductName
GO

SELECT *
FROM vwProduct_Details
GO
```



	ProductName	AverageRate
1	Hard Disk Drive	3570.00
2	Portable Hard Drive	5580.00

Using Built-in Functions with Views

CHECK OPTION

The CHECK OPTION is an option associated with the CREATE VIEW statement.

It is used to ensure that all the updates in the view satisfy the conditions mentioned in the view definition.

If the conditions are not satisfied, the database engine returns an error.

Syntax:

```
CREATE VIEW <view_name>
AS select_statement [ WITH CHECK OPTION ]
```

```
CREATE VIEW vwProductInfo AS
  SELECT * FROM Production.Product
  WHERE SafetyStockLevel <=1000
  WITH CHECK OPTION;
GO
```

```
UPDATE vwProductInfo SET SafetyStockLevel = 2500
  WHERE ProductID=321
```

ERROR !

CHECK OPTION

- Example

```
CREATE VIEW vwProductInfo AS
SELECT * FROM Product_Details
WHERE Rate <= 4000
WITH CHECK OPTION
GO

UPDATE vwProductInfo SET Rate =
10000
  WHERE ProductID=3
```

SCHEMABINDING Option

- A view can be bound to the schema of the base table using the SCHEMABINDING option.
- This option can be used with CREATE VIEW or ALTER VIEW statements.

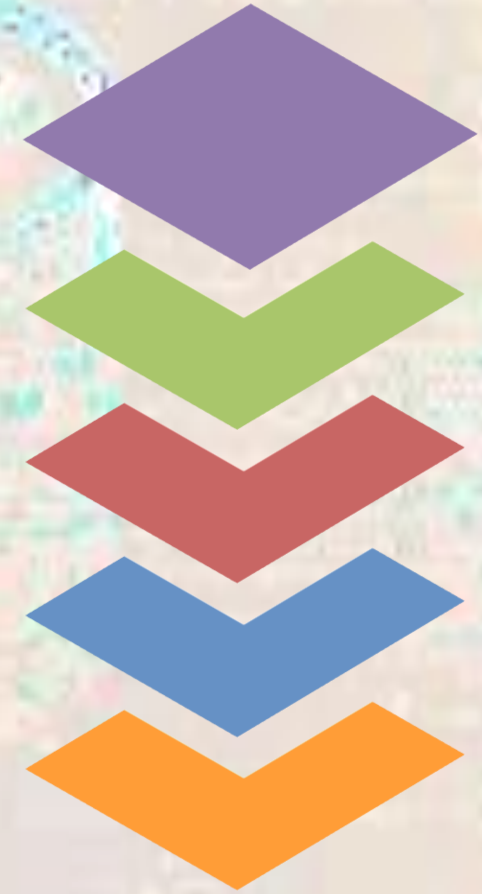
- When **SCHEMABINDING** option is specified, the base tables cannot be modified if that would affect the view definition.

- The view definition must be first modified or deleted to remove dependencies on the table that is to be modified.

```
CREATE VIEW vwNewProductInfo
WITH SCHEMABINDING AS
SELECT ProductID, ProductName,
Rate
FROM dbo.Product_Details
GO
```

Using sp_refreshview

During the creation of a view, the SCHEMABINDING option is used to bind the view to the schema of the tables that are included in the view.



If changes are made to the underlying objects (tables or views) on which the view depends, the sp_refreshview stored procedure should be executed.

The sp_refreshview stored procedure updates the metadata for the view.

Stored Procedures 1-2

- 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. This eliminates the need for repetitively typing out multiple Transact-SQL statements and then repetitively compiling them.

Stored Procedures 2-2

Improved Security

The database administrator can improve security by associating database privileges with stored procedures. Users can be given permission to execute a stored procedure even if user does not have permission to access tables or views.

Precompiled Execution

Stored procedures are compiled during the first execution. For every subsequent execution, SQL Server reuses this precompiled version. This reduces the time and resources required for compilation.

Reduced Client/Server Traffic

Stored procedures help in reducing network traffic. When Transact-SQL statements are executed individually, there is network usage separately for execution of each statement. When a stored procedure is executed, Transact-SQL statements are executed together as a single unit. This reduces network traffic.

Reuse of code

Stored procedures can be used multiple times. This eliminates need to repetitively type out hundreds of Transact-SQL statements every time a similar task is to be performed.

Types of Stored Procedures 1-3

- SQL Server supports various types of stored procedures such as:
  - **User-Defined Stored Procedures**
  - **Extended Stored Procedures**

User-Defined Stored Procedures

- Also known as custom stored procedures
- Used for reusing Transact-SQL statements for performing repetitive tasks

Two types of user-defined stored procedures

Transact-SQL stored procedures

Common Language Runtime (CLR) stored procedures.

Types of Stored Procedures 2-3

Extended Stored Procedures

- Helps SQL Server in interacting with the operating system
- Not resident objects of SQL Server
- They are procedures that are implemented as Dynamic-Link Libraries (DLL) executed outside the SQL Server environment
- Use the 'xp' prefix

Types of Stored Procedures 3-3

System Stored Procedures

- ➢ Used for interacting with system tables and performing administrative tasks such as updating system tables
- ➢ These procedures are located in the Resource 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.
- Is used in database administrative and informational activities.
- Provide easy access to the metadata information about database objects such as system tables, user-defined tables, views, and indexes.

Classification of System Stored Procedures

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.
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

Local Temporary Procedure	Global Temporary Procedure
Visible only to the user that created it	Visible to all users
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.

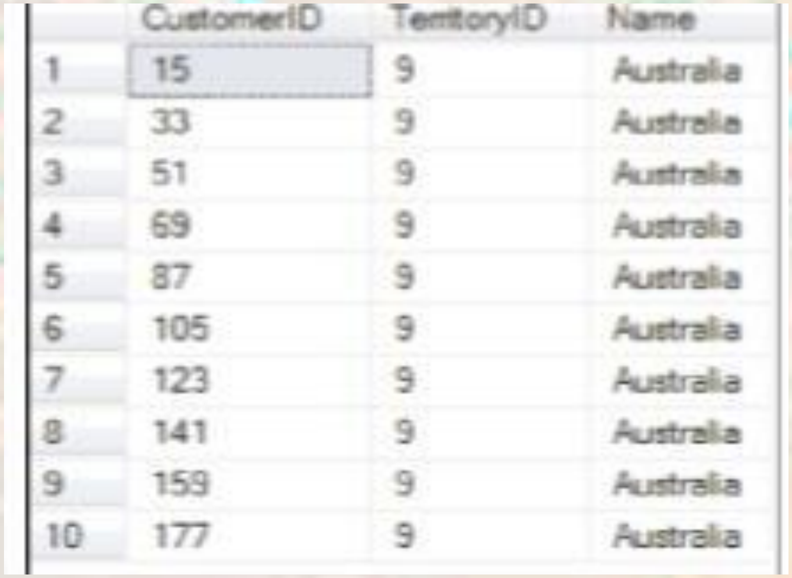
When such an error occurs, the statement that caused the error is terminated, but the remote procedure continues to be executed.

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.

Custom or User-defined Stored Procedures

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



	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

Output of a Simple Stored Procedure

Using Parameters

Parameters are of two types:

Input Parameters

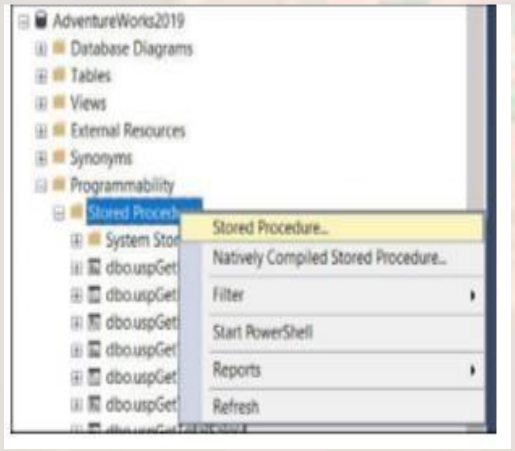
Allow the calling program to pass values to a stored procedure. These values are accepted into variables defined in the stored procedure.

Output Parameters

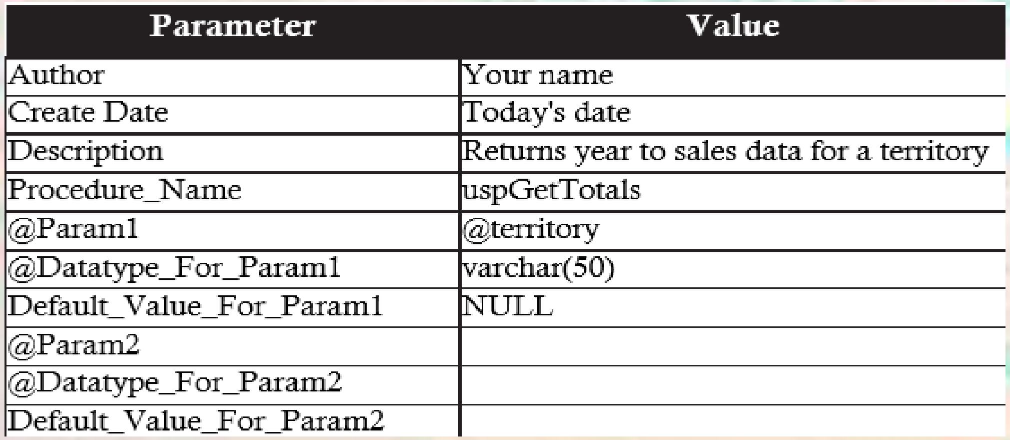
Allow a stored procedure to pass values back to the calling program. These values are accepted into variables by the calling program.

Using SSMS to Create Stored Procedures

- Steps to create a user-defined stored procedure using SSMS are:
  1. Launch Object Explorer.
  2. In Object Explorer, connect to an instance of Database Engine and after successfully connection, expand the instance.
  3. Expand the Databases node and then, expand the AdventureWorks2019 database.



Creating a Stored Procedure



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	

Parameter Values

Modifying and Dropping Stored Procedures

- Permissions associated with stored procedure are lost when a stored procedure is re-created. However, when a stored procedure is altered, permissions defined for the stored procedure remain same even though procedure definition is changed.

Dropping stored procedures

- Stored procedures can be dropped if they are no longer required. If another stored procedure calls a deleted procedure, an error message is displayed.

- If a new procedure is created using the same name as well as the same parameters as the dropped procedure, all calls to the dropped procedure will be executed successfully.

Nested Stored Procedures

- SQL Server 2019 enables stored procedures to be called inside other stored procedures.
- The called procedures can in turn call other procedures.
- This architecture of calling one procedure from another procedure is referred to as nested stored procedure architecture.
- When a stored procedure calls another stored procedure, the level of nesting is said to be increased by one.
- Similarly, when a called procedure completes its execution and passes control back to the calling procedure, the level of nesting is said to be decreased by one.
- The maximum level of nesting supported by SQL Server 2019 is 32.

Querying System MetaData 1-3

The properties of an object such as a table or a view are stored in special system tables.

These properties are referred to as metadata. All SQL objects produce metadata.

This metadata can be viewed using system views, which are predefined views of SQL Server.

System Catalog Views

- These contain information about the catalog in a SQL Server system.
- A catalog is similar to an inventory of objects.
- These views contain a wide range of metadata.

Querying System MetaData 2-3

Information Schema Views

- Users can query information schema views to return system metadata.
- These views are useful to third-party tools that may not be specific for SQL Server.
- Information schema views provide an internal, system table independent view of the SQL Server metadata.

Information Schema Views	SQL Server System Views
They are stored in their own schema, INFORMATION_SCHEMA.	They appear in the sys schema.
They use standard terminology instead of SQL Server terms. For example, they use catalog instead of database and domain instead of user-defined data type.	They adhere to SQL Server terminology.
They may not expose all the metadata available to SQL Server's own catalog views. For example, sys.columns includes attributes for identity property and computed column property, while INFORMATION_SCHEMA.columns does not.	They can expose all the metadata available to SQL Server's catalog views.

Information Schema Views and SQL Server System Views

Querying System MetaData 3-3

System Metadata Functions

- Built-in functions can return metadata to a query.
- These include scalar functions and table-valued functions.
- SQL Server metadata functions come in a variety of formats.

Function Name	Description	Example
OBJECT_ID(<object_name>)	Returns the object ID of a database object.	OBJECT_ID('Sales.Customer')
OBJECT_NAME(<object_id>)	Returns the name corresponding to an object ID.	OBJECT_ NAME(197575742)
@@ERROR	Returns 0 if the last statement succeeded; otherwise returns the error number.	@@ERROR
SERVERPROPERTY (<property >)	Returns the value of the specified server property.	SERVERPROPERTY('Collation')

Common System Metadata Functions

Querying Dynamic Management Objects

- Dynamic Management Views (DMVs) and Dynamic Management Functions (DMFs) are dynamic management objects that return server and database state information.

- DMVs and DMFs are collectively referred to as dynamic management objects.

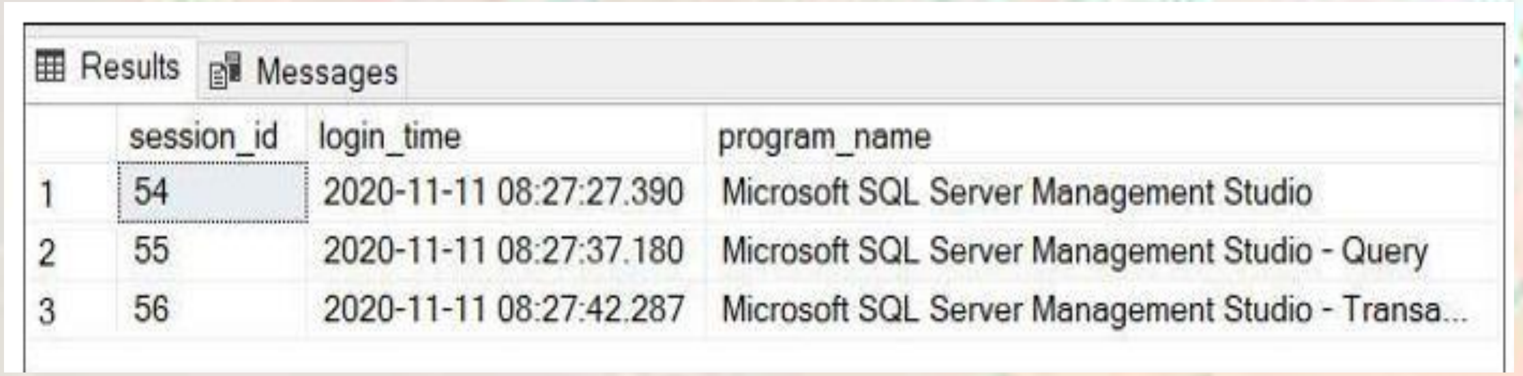
- Used for examining the state of SQL Server instance, troubleshooting, and performance tuning.

- In order to query DMVs, it is required to have VIEW SERVER STATE or VIEW DATABASE STATE permission, depending on the scope of the DMV.

Categorizing and Querying DMVs

Naming Pattern	Description
db	Database related
io	I/O statistics
Os	SQL Server Operating System Information
'tran'	Transaction-related
'exec'	Query execution-related metadata

Organizing DMVS by Function



	session_id	login_time	program_name
1	54	2020-11-11 08:27:27.390	Microsoft SQL Server Management Studio
2	55	2020-11-11 08:27:37.180	Microsoft SQL Server Management Studio - Query
3	56	2020-11-11 08:27:42.287	Microsoft SQL Server Management Studio - Transa...

Querying the sys.dm_exec_sessions DMV

Summary

- A view is a virtual table that is made up of selected columns from one or more tables and is created using the CREATE VIEW command in SQL Server.
- Users can manipulate the data in views, such as inserting into views, modifying the data in views, and deleting from views.
- A stored procedure is a group of Transact-SQL statements that act as a single block of code that performs a specific task.
- SQL Server supports various types of stored procedures, such as User-Defined Stored Procedures, Extended Stored Procedures, and System Stored Procedures.
- System stored procedures can be classified into different categories such as Catalog Stored Procedures, Security Stored Procedures, and Cursor Stored Procedures.
- Input and output parameters can be used with stored procedures to pass and receive data from stored procedures.
- The properties of an object such as a table or a view are stored in special system tables and are referred to as metadata.
- DMVs and DMFs are dynamic management objects that return server and database state information. DMVs and DMFs are collectively referred to as dynamic management objects.