



SQL Server Views & Stored Procedures



© Copyright Microsoft Corporation. All rights reserved.

Agenda

- SQL Server Views
- Stored Procedures

© Copyright Microsoft Corporation. All rights reserved.

What is a View?

Views are simply SQL queries, that are saved as any other database object.

You can then query the views just like a database table.

A view acts as a filter on the underlying tables referenced in the view:

- The query that defines the view can be from one or more tables or from other views in the current or other databases

To query a view and retrieve results from it, refer to it in the FROM clause of a SELECT statement, as you would refer to a table.

© Copyright Microsoft Corporation. All rights reserved.

```
CREATE VIEW Sales.CustOrders
AS
SELECT
    O.custid,
    DATEADD(month, DATEDIFF(month, 0, O.orderdate), 0)
AS ordermonth,
    SUM(OD.qty) AS qty
FROM Sales.Orders AS O
    JOIN Sales.OrderDetails AS OD
    ON OD.orderid = O.orderid
GROUP BY custid, DATEADD(month, DATEDIFF(month, 0,
O.orderdate), 0);
```

```
SELECT custid, ordermonth, qty
FROM Sales.CustOrders;
```

<https://learn.microsoft.com/en-us/training/modules/create-tables-views-temporary-objects/3-create-query-views>

Views (cont...)

- You can use views as a source for your queries in much the same way as tables.
- Views are parsed and validated when created but not executed until queried.
- **Benefits of Views**
 - **Simplicity:** Hide complex joins and calculations from end users.
 - **Reusability:** Use the same logic in multiple queries without rewriting.
 - **Security:** Restrict access to sensitive columns or rows.
 - Views can be used as security mechanisms by letting users access data through the view, without granting users permissions to directly access the underlying tables of the query
 - **Data abstraction:** Present only relevant data to users.
 - Views can be considered as an application programming interface (API) to a database for purposes of retrieving data.
 - **Consistency:** Centralize business logic and calculations.

© Copyright Microsoft Corporation. All rights reserved.

Updatable View

You can create a view that has SQL command to update the underlying data IF and ONLY IF:

- UPDATE, INSERT, and DELETE statements reference columns from only one base table.

The columns being modified in the view must directly reference the underlying data in the table columns. The columns cannot be derived in any other way, such as through the following:

- An aggregate function: AVG, COUNT, SUM, MIN, MAX, GROUPING, STDEV, STDEVP, VAR, and VARP.
- A computation. The column cannot be computed from an expression that uses other columns. Columns that are formed by using the set operators UNION, UNION ALL, CROSSJOIN, EXCEPT, and INTERSECT amount to a computation and are also not updatable.
- The columns being modified are not affected by **GROUP BY**, **HAVING**, or **DISTINCT** clauses.
- TOP is not used anywhere in the select statement of the view together with the WITH CHECK OPTION clause.

Updatable View

```
CREATE VIEW SimpleEmployeeView AS  
SELECT EmpID, EmpName, Dept  
FROM EmployeeProjects;
```

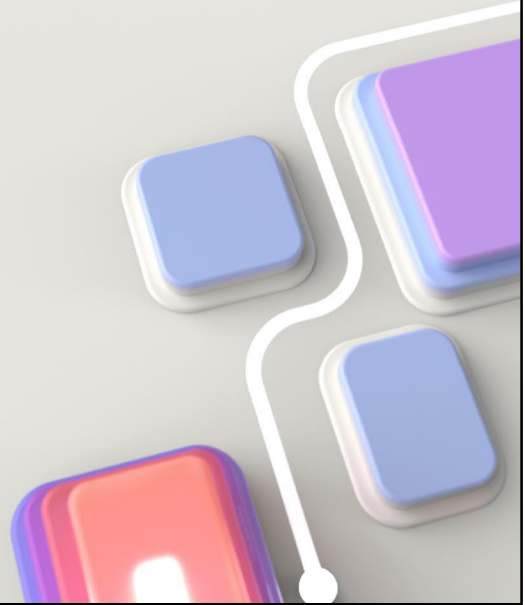
Same above view is being used, but with an UPDATE statement:

```
UPDATE SimpleEmployeeView  
SET Dept = 'Marketing'  
WHERE EmpID = 101;
```

© Copyright Microsoft Corporation. All rights reserved.

Stored Procedure

© Copyright Microsoft Corporation. All rights reserved.



Some Pre-requisites (as always)

- **@@ Functions**
 - In SQL Server, **@@ functions** are **global variables** that return system-level information.
 - They're built-in and start with two @ symbols (@@).
 - You don't need to declare them—they're always available.
- Usage & Benefits:
 - Track session-level or server-level values.
 - Monitor performance, errors, or settings.
 - Control flow logic in stored procedures or scripts.
- Example:
 - **@@IDENTITY**
 - Last identity value inserted
 - **SELECT @@IDENTITY**
 - **@@ROWCOUNT**
 - Number of rows affected by the last statement.
 - **IF @@ROWCOUNT > 0 PRINT 'Rows updated'**

© Copyright Microsoft Corporation. All rights reserved.

Common @@ Functions

Function	Description	Example
@@ROWCOUNT	Number of rows affected by the last statement	IF @@ROWCOUNT > 0 PRINT 'Rows updated'
@@ERROR	Error code from the last statement	IF @@ERROR <> 0 PRINT 'Error occurred'
@@IDENTITY	Last identity value inserted	SELECT @@IDENTITY
@@TRANCOUNT	Number of active transactions	IF @@TRANCOUNT > 0 COMMIT
@@VERSION	SQL Server version info	SELECT @@VERSION
@@SERVERNAME	Name of the SQL Server instance	SELECT @@SERVERNAME

© Copyright Microsoft Corporation. All rights reserved.

SET NOCOUNT ON

- When you run a SQL statement like INSERT, UPDATE, or DELETE, SQL Server by default returns a message like:
 - (3 rows affected)
- This message is called the **row count message**.
- SET NOCOUNT ON tells SQL Server **not to send these messages**.
- Benefits:
 - Performance Improvement
 - Cleaner Output
 - Avoids Interference in Applications
- Returns the number of rows affected by the last statement
- The @@ROWCOUNT function is updated even when SET NOCOUNT is ON.
- Setting SET NOCOUNT to ON can provide a significant performance boost, because network traffic is greatly reduced.

© Copyright Microsoft Corporation. All rights reserved.

```
-- Without NOCOUNT
UPDATE Employees SET Status = 'Active' WHERE Status =
'Pending';

-- Now add NOCOUNT
SET NOCOUNT ON;
UPDATE Employees SET Status = 'Active' WHERE Status =
'Pending';
SET NOCOUNT OFF;
```

Stored Procedure (aka stored-procs, s-procs)

A stored procedure in SQL Server is a group of:

- One or more Transact-SQL statements, or
- A reference to a Microsoft .NET Framework common runtime language (CLR) method.

Procedures resemble functions/methods in other programming languages because they can:

- Accept input parameters and return multiple values in the form of output parameters to the calling program.
- Contain programming statements that perform operations in the database. These include calling other procedures.
- Return a status value to a calling program to indicate success or failure (and the reason for failure).

© Copyright Microsoft Corporation. All rights reserved.

Benefits of S-Proc

- **Reduced server/client network traffic**
- **Stronger security**
 - Multiple users and client programs can perform operations on underlying database objects through a procedure, even if the users and programs don't have direct permissions on those underlying objects.
- **Reuse of code**
- **Easier maintenance**
- **Improved performance**
 - A procedure compiles the first time it's executed and creates an execution plan that is reused for subsequent executions.

© Copyright Microsoft Corporation. All rights reserved.

```
CREATE PROCEDURE SalesLT.uspGetCustomerCompany
    @LastName nvarchar(50),
    @FirstName nvarchar(50)
AS
    SET NOCOUNT ON;
    SELECT FirstName, LastName, CompanyName
    FROM SalesLT.Customer
    WHERE
        FirstName = @FirstName AND
        LastName = @LastName;
GO
```

Reduced server/client network traffic

The commands in a procedure are executed as a single batch of code. This can significantly reduce network traffic between the server and client because only the call to execute the procedure is sent across the network. Without the code encapsulation provided by a procedure, every individual line of code would have to cross the network.

Stronger security

Multiple users and client programs can perform operations on underlying database objects through a procedure, even if the users and programs don't have direct permissions on those underlying objects. The procedure controls what processes and activities are performed and protects the underlying database objects. This eliminates the requirement to grant permissions at the individual object level and simplifies the security layers.

Reuse of code

The code for any repetitious database operation is the perfect candidate for encapsulation in procedures. This eliminates needless rewrites of the same

code, decreases code inconsistency, and allows the access and execution of code by any user or application possessing the necessary permissions.

Easier maintenance

When client applications call procedures and keep database operations in the data tier, only the procedures must be updated for any changes in the underlying database. The application tier remains separate and doesn't have to know how about any changes to database layouts, relationships, or processes.

Improved performance

By default, a procedure compiles the first time it's executed, and creates an execution plan that is reused for subsequent executions. Since the query processor doesn't have to create a new plan, it typically takes less time to process the procedure.

If there are significant changes to the tables or data referenced by the procedure, the precompiled plan might actually cause the procedure to perform slower. In this case, recompiling the procedure and forcing a new execution plan can improve performance.

Stored Procedures w/Return Values

```
CREATE PROCEDURE CheckStock
    @ProductID INT
AS
BEGIN
    DECLARE @Stock INT
    SELECT @Stock = Quantity FROM Inventory WHERE ProductID = @ProductID
    IF @Stock > 0
        RETURN 1
    ELSE
        RETURN 0
END
```

```
DECLARE @Result INT
EXEC @Result = CheckStock @ProductID = 101
```

Stored Procedures w/Output Parameters

```
CREATE PROCEDURE GetCustomerSummary
    @CustomerID INT,
    @TotalOrders INT OUTPUT
AS
BEGIN
    SELECT @TotalOrders = COUNT(*) FROM Orders WHERE CustomerID = @CustomerID

    SELECT OrderID, OrderDate, OrderAmount
    FROM Orders
    WHERE CustomerID = @CustomerID

    RETURN @TotalOrders
END
```

```
DECLARE @Orders INT, @RetVal INT
EXEC @RetVal = GetCustomerSummary @CustomerID = 101, @TotalOrders = @Orders OUTPUT
PRINT 'Output: ' + CAST(@Orders AS VARCHAR)
PRINT 'Return: ' + CAST(@RetVal AS VARCHAR)
```

© Copyright Microsoft Corporation. All rights reserved.

Stored Procedures w/Exception Handling

```
CREATE PROCEDURE SafeDivideWithThrow
    @Numerator INT,
    @Denominator INT,
    @Result INT OUTPUT
AS
BEGIN
    BEGIN TRY
        IF @Denominator = 0
        BEGIN
            THROW 50001, 'Division by zero is not allowed.', 1;
        END

        SET @Result = @Numerator / @Denominator;
    END TRY
    BEGIN CATCH
        -- You can log the error or re-throw it
        THROW; -- Re-throws the original error
    END CATCH
END
```

© Copyright Microsoft Corporation. All rights reserved.

TRY...CATCH Block

TRY contains the code that might cause an error.

CATCH handles the error if one occurs.

THROW

error_number: Must be a number ≥ 50000 (user-defined error).

message: Custom error message.

state: A number from 0 to 255 used to identify the error location.

Why Use THROW Instead of RAISERROR?

THROW is newer (introduced in SQL Server 2012).

It automatically preserves the original error details when used without parameters.

It's cleaner and more consistent with modern error handling.

Stored Procedures w/Exception Handling

```
CREATE PROCEDURE SafeDivideWithThrow
    @Numerator INT,
    @Denominator INT,
    @Result INT OUTPUT
AS
BEGIN
    BEGIN TRY
        IF @Denominator = 0
        BEGIN
            THROW 50001, 'Division by zero is not allowed.', 1;
        END

        SET @Result = @Numerator / @Denominator;
    END TRY
    BEGIN CATCH
        -- You can log the error or re-throw it
        THROW; -- Re-throws the original error
    END CATCH
END
```

© Copyright Microsoft Corporation. All rights reserved.

TRY...CATCH Block

TRY contains the code that might cause an error.

CATCH handles the error if one occurs.

THROW

error_number: Must be a number ≥ 50000 (user-defined error).

message: Custom error message.

state: A number from 0 to 255 used to identify the error location.

Why Use THROW Instead of RAISERROR?

THROW is newer (introduced in SQL Server 2012).

It automatically preserves the original error details when used without parameters.

It's cleaner and more consistent with modern error handling.

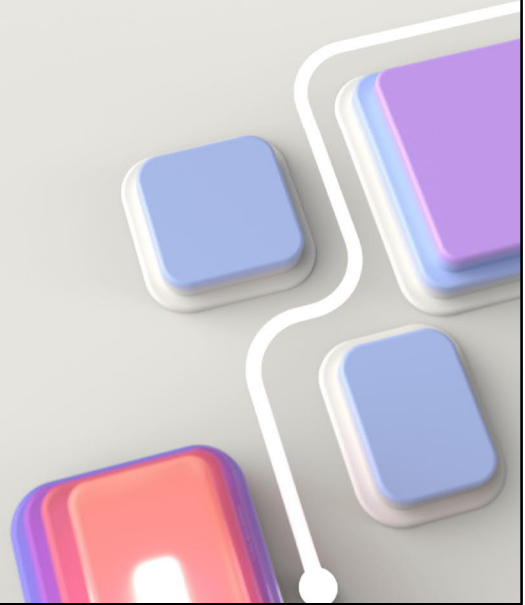
SQL Server Views vs. Stored Procedures

Feature	View	Stored Procedure
Purpose	Display data	Perform operations or logic
Can accept parameters	✗ No	✓ Yes
Can modify data	✗ No	✓ Yes (Insert, Update, Delete)
Can contain logic	✗ Limited (only SELECT)	✓ Full logic (IF, WHILE, etc.)
Returns	Table-like result	Can return data, output parameters, or nothing

© Copyright Microsoft Corporation. All rights reserved.

Triggers

© Copyright Microsoft Corporation. All rights reserved.



Triggers

- Special type of stored procedure that automatically runs when an event occurs in the database server
- DML triggers run when a user tries to modify data through a data manipulation language (DML) event
 - DML events are INSERT, UPDATE, or DELETE statements on a table or view.
- DDL triggers run in response to various data definition language (DDL) events.
 - Correspond to Transact SQL CREATE, ALTER, and DROP statements
- Logon triggers fire in response to the LOGON event that is raised when a user's session is being established.

© Copyright Microsoft Corporation. All rights reserved.

```
CREATE TRIGGER trg_AuditInsert
ON Employees
AFTER INSERT
AS
BEGIN
    INSERT INTO AuditLog (ActionType, ActionTime)
    VALUES ('INSERT', GETDATE())
END
```

AFTER Trigger

Executes after an INSERT, UPDATE, or DELETE.

INSTEAD OF Trigger

Replaces the action (used mostly with views).

Return results

The ability to return results from triggers will be removed in a future version of SQL Server. Triggers that return result sets might cause unexpected behavior in applications that aren't designed to work with them.

Multiple triggers

SQL Server lets you create multiple triggers for each DML, DDL, or LOGON event. For example, if CREATE TRIGGER FOR UPDATE is run for a table that already has an UPDATE trigger, an additional update trigger is created. In earlier versions of SQL Server, only one trigger for each INSERT, UPDATE, or DELETE data modification event is allowed for each table.

INSTEAD OF Trigger

An INSTEAD OF trigger replaces the default action of INSERT, UPDATE, or DELETE on a table or view. It lets you customize what happens when someone tries to modify data.

Usage & Benefits:

- Enable DML on Views
- Control Data Changes
- Prevent Certain Actions
- Implement Business Rules

```
CREATE TRIGGER trg_InsertEmployeeView
ON EmployeeView
INSTEAD OF INSERT
AS
BEGIN
    INSERT INTO Employees (EmpID, Name,
    Department)
    SELECT EmpID, Name, Department FROM INSERTED
END
```

© Copyright Microsoft Corporation. All rights reserved.

Enable DML on Views: Views don't support direct INSERT, UPDATE, or DELETE unless you use triggers.

Control Data Changes: You can validate or transform data before it's inserted or updated.

Prevent Certain Actions: You can block unwanted changes (e.g., deleting critical rows).

Implement Business Rules: Apply custom logic like logging, conditional updates, or cascading changes.