# Complete Guide to Views in SQL Server Management Studio (SSMS)

## Table of Contents

1. [What is a View?](#what-is-a-view)
2. [Types of Views](#types-of-views)
3. [Basic View Syntax](#basic-view-syntax)
4. [Creating Simple Views](#creating-simple-views)
5. [Views with Joins](#views-with-joins)
6. [Views with Aggregations](#views-with-aggregations)
7. [Views with Calculations](#views-with-calculations)
8. [Parameterized Views (Table-Valued Functions)](#parameterized-views)
9. [Indexed Views (Materialized Views)](#indexed-views)
10. [Updating Data Through Views](#updating-data-through-views)
11. [View Security and Permissions](#view-security-and-permissions)
12. [View Management in SSMS](#view-management-in-ssms)
13. [Best Practices](#best-practices)
14. [Views vs Alternatives](#views-vs-alternatives)

## What is a View?

A **View** in SQL Server is a virtual table that doesn’t store data physically. Instead, it stores a SQL query that generates data dynamically when the view is accessed. Views provide a way to present data from one or more tables in a specific format without duplicating the underlying data.

### Key Benefits:

* **Data Security**: Hide sensitive columns and restrict data access
* **Simplification**: Present complex joins in a simple format
* **Consistency**: Standardize data presentation across applications
* **Abstraction**: Hide database schema complexity from users
* **Reusability**: Reuse common query logic across multiple applications

## Types of Views

### 1. Simple Views

* Based on a single table
* Can perform DML operations (INSERT, UPDATE, DELETE)

### 2. Complex Views

* Based on multiple tables with joins
* May include GROUP BY, aggregate functions
* Limited DML operations

### 3. Indexed Views (Materialized Views)

* Physically store data for performance
* Automatically updated when base tables change

## Basic View Syntax

-- Create View  
CREATE VIEW view\_name AS  
SELECT column1, column2, ...  
FROM table\_name  
WHERE condition;  
  
-- Use View  
SELECT \* FROM view\_name;  
  
-- Modify View  
ALTER VIEW view\_name AS  
SELECT column1, column2, ...  
FROM table\_name  
WHERE new\_condition;  
  
-- Drop View  
DROP VIEW view\_name;

## Creating Simple Views

Let’s start with sample data:

-- Create sample tables  
CREATE TABLE employees (  
 employee\_id INT PRIMARY KEY,  
 first\_name VARCHAR(50),  
 last\_name VARCHAR(50),  
 email VARCHAR(100),  
 phone VARCHAR(20),  
 department VARCHAR(50),  
 position VARCHAR(50),  
 salary DECIMAL(10,2),  
 hire\_date DATE,  
 manager\_id INT,  
 is\_active BIT DEFAULT 1  
);  
  
CREATE TABLE departments (  
 dept\_id INT PRIMARY KEY,  
 dept\_name VARCHAR(50),  
 location VARCHAR(50),  
 budget DECIMAL(12,2)  
);  
  
-- Insert sample data  
INSERT INTO employees VALUES  
(1, 'John', 'Doe', 'john.doe@company.com', '555-1001', 'IT', 'Developer', 75000, '2020-01-15', NULL, 1),  
(2, 'Jane', 'Smith', 'jane.smith@company.com', '555-1002', 'IT', 'Senior Developer', 85000, '2019-03-10', 1, 1),  
(3, 'Bob', 'Johnson', 'bob.johnson@company.com', '555-1003', 'Sales', 'Sales Rep', 55000, '2021-06-20', NULL, 1),  
(4, 'Alice', 'Brown', 'alice.brown@company.com', '555-1004', 'HR', 'HR Manager', 70000, '2018-09-05', NULL, 1),  
(5, 'Charlie', 'Wilson', 'charlie.wilson@company.com', '555-1005', 'IT', 'Junior Developer', 60000, '2022-02-14', 2, 1),  
(6, 'Diana', 'Ross', 'diana.ross@company.com', '555-1006', 'Sales', 'Sales Manager', 80000, '2017-11-30', NULL, 1),  
(7, 'Elvis', 'King', 'elvis.king@company.com', '555-1007', 'Finance', 'Accountant', 65000, '2020-08-12', NULL, 0);  
  
INSERT INTO departments VALUES  
(1, 'Information Technology', 'New York', 500000),  
(2, 'Sales', 'Chicago', 300000),  
(3, 'Human Resources', 'Los Angeles', 200000),  
(4, 'Finance', 'Boston', 250000);

### Basic Employee View

-- Create a view showing active employees only  
CREATE VIEW vw\_active\_employees AS  
SELECT   
 employee\_id,  
 first\_name,  
 last\_name,  
 first\_name + ' ' + last\_name AS full\_name,  
 email,  
 department,  
 position,  
 salary,  
 hire\_date  
FROM employees  
WHERE is\_active = 1;  
  
-- Use the view  
SELECT \* FROM vw\_active\_employees;  
SELECT \* FROM vw\_active\_employees WHERE department = 'IT';

### Public Employee Directory View

-- Create a view hiding sensitive information  
CREATE VIEW vw\_employee\_directory AS  
SELECT   
 employee\_id,  
 first\_name + ' ' + last\_name AS full\_name,  
 department,  
 position,  
 email,  
 -- Hide salary and other sensitive data  
 CASE   
 WHEN phone IS NOT NULL THEN 'Available'  
 ELSE 'Not Available'  
 END AS contact\_available  
FROM employees  
WHERE is\_active = 1;  
  
-- Usage  
SELECT \* FROM vw\_employee\_directory ORDER BY full\_name;

## Views with Joins

### Employee Department View

-- Create view joining employees with departments  
CREATE VIEW vw\_employee\_department AS  
SELECT   
 e.employee\_id,  
 e.first\_name + ' ' + e.last\_name AS employee\_name,  
 e.email,  
 e.position,  
 e.salary,  
 e.hire\_date,  
 d.dept\_name AS department\_name,  
 d.location AS department\_location,  
 d.budget AS department\_budget  
FROM employees e  
LEFT JOIN departments d ON e.department = d.dept\_name  
WHERE e.is\_active = 1;  
  
-- Usage  
SELECT \* FROM vw\_employee\_department;  
SELECT \* FROM vw\_employee\_department WHERE department\_location = 'New York';

### Employee Hierarchy View

-- Create view showing manager-employee relationships  
CREATE VIEW vw\_employee\_hierarchy AS  
SELECT   
 e.employee\_id,  
 e.first\_name + ' ' + e.last\_name AS employee\_name,  
 e.department,  
 e.position,  
 e.salary,  
 m.first\_name + ' ' + m.last\_name AS manager\_name,  
 m.position AS manager\_position  
FROM employees e  
LEFT JOIN employees m ON e.manager\_id = m.employee\_id  
WHERE e.is\_active = 1;  
  
-- Usage  
SELECT \* FROM vw\_employee\_hierarchy ORDER BY manager\_name, employee\_name;

### Complex Multi-Table View

-- Create comprehensive employee view  
CREATE VIEW vw\_employee\_complete AS  
SELECT   
 e.employee\_id,  
 e.first\_name + ' ' + e.last\_name AS employee\_name,  
 e.email,  
 e.phone,  
 e.position,  
 e.salary,  
 e.hire\_date,  
 DATEDIFF(YEAR, e.hire\_date, GETDATE()) AS years\_of\_service,  
 d.dept\_name AS department\_name,  
 d.location AS office\_location,  
 d.budget AS department\_budget,  
 m.first\_name + ' ' + m.last\_name AS manager\_name,  
 CASE   
 WHEN e.salary >= 80000 THEN 'Senior'  
 WHEN e.salary >= 65000 THEN 'Mid-Level'  
 ELSE 'Junior'  
 END AS employee\_level  
FROM employees e  
LEFT JOIN departments d ON e.department = d.dept\_name  
LEFT JOIN employees m ON e.manager\_id = m.employee\_id  
WHERE e.is\_active = 1;  
  
-- Usage  
SELECT \* FROM vw\_employee\_complete WHERE employee\_level = 'Senior';

## Views with Aggregations

### Department Statistics View

CREATE VIEW vw\_department\_stats AS  
SELECT   
 department,  
 COUNT(\*) AS employee\_count,  
 AVG(salary) AS avg\_salary,  
 MIN(salary) AS min\_salary,  
 MAX(salary) AS max\_salary,  
 SUM(salary) AS total\_salary\_cost,  
 MAX(salary) - MIN(salary) AS salary\_range  
FROM employees  
WHERE is\_active = 1  
GROUP BY department;  
  
-- Usage  
SELECT \* FROM vw\_department\_stats ORDER BY avg\_salary DESC;

### Monthly Hiring Trends View

CREATE VIEW vw\_hiring\_trends AS  
SELECT   
 YEAR(hire\_date) AS hire\_year,  
 MONTH(hire\_date) AS hire\_month,  
 DATENAME(MONTH, hire\_date) AS month\_name,  
 COUNT(\*) AS employees\_hired,  
 AVG(salary) AS avg\_starting\_salary  
FROM employees  
GROUP BY YEAR(hire\_date), MONTH(hire\_date), DATENAME(MONTH, hire\_date);  
  
-- Usage  
SELECT \* FROM vw\_hiring\_trends ORDER BY hire\_year, hire\_month;

### Salary Analysis View

CREATE VIEW vw\_salary\_analysis AS  
SELECT   
 position,  
 department,  
 COUNT(\*) AS position\_count,  
 AVG(salary) AS avg\_salary,  
 MIN(salary) AS min\_salary,  
 MAX(salary) AS max\_salary,  
 STDEV(salary) AS salary\_std\_dev,  
 CASE   
 WHEN COUNT(\*) > 1 THEN (MAX(salary) - MIN(salary)) / NULLIF(AVG(salary), 0) \* 100  
 ELSE 0  
 END AS salary\_variation\_percent  
FROM employees  
WHERE is\_active = 1  
GROUP BY position, department;  
  
-- Usage  
SELECT \* FROM vw\_salary\_analysis   
WHERE position\_count > 1   
ORDER BY salary\_variation\_percent DESC;

## Views with Calculations

### Employee Performance Metrics View

CREATE VIEW vw\_employee\_metrics AS  
SELECT   
 employee\_id,  
 first\_name + ' ' + last\_name AS employee\_name,  
 department,  
 position,  
 salary,  
 hire\_date,  
 DATEDIFF(DAY, hire\_date, GETDATE()) AS days\_employed,  
 DATEDIFF(YEAR, hire\_date, GETDATE()) AS years\_of\_service,  
 salary / 12.0 AS monthly\_salary,  
 salary / 52.0 AS weekly\_salary,  
 salary / 2080.0 AS hourly\_rate, -- Assuming 40 hours/week \* 52 weeks  
 CASE   
 WHEN DATEDIFF(YEAR, hire\_date, GETDATE()) < 1 THEN 'New Hire'  
 WHEN DATEDIFF(YEAR, hire\_date, GETDATE()) < 3 THEN 'Junior'  
 WHEN DATEDIFF(YEAR, hire\_date, GETDATE()) < 8 THEN 'Experienced'  
 ELSE 'Veteran'  
 END AS experience\_level,  
 salary \* 1.15 AS total\_cost\_with\_benefits -- Assuming 15% benefits cost  
FROM employees  
WHERE is\_active = 1;  
  
-- Usage  
SELECT \* FROM vw\_employee\_metrics WHERE experience\_level = 'Veteran';

### Budget Analysis View

CREATE VIEW vw\_budget\_analysis AS  
SELECT   
 d.dept\_name AS department,  
 d.location,  
 d.budget AS allocated\_budget,  
 COUNT(e.employee\_id) AS employee\_count,  
 SUM(e.salary) AS total\_salaries,  
 SUM(e.salary \* 1.15) AS total\_cost\_with\_benefits,  
 d.budget - SUM(e.salary \* 1.15) AS remaining\_budget,  
 CASE   
 WHEN d.budget - SUM(e.salary \* 1.15) > 0 THEN 'Under Budget'  
 WHEN d.budget - SUM(e.salary \* 1.15) = 0 THEN 'On Budget'  
 ELSE 'Over Budget'  
 END AS budget\_status,  
 (SUM(e.salary \* 1.15) / NULLIF(d.budget, 0)) \* 100 AS budget\_utilization\_percent  
FROM departments d  
LEFT JOIN employees e ON d.dept\_name = e.department AND e.is\_active = 1  
GROUP BY d.dept\_name, d.location, d.budget;  
  
-- Usage  
SELECT \* FROM vw\_budget\_analysis ORDER BY budget\_utilization\_percent DESC;

## Parameterized Views (Table-Valued Functions)

Since views can’t accept parameters directly, we use Table-Valued Functions:

### Inline Table-Valued Function (Acts like a parameterized view)

-- Create a function that works like a parameterized view  
CREATE FUNCTION fn\_employees\_by\_department(@department VARCHAR(50))  
RETURNS TABLE  
AS  
RETURN  
(  
 SELECT   
 employee\_id,  
 first\_name + ' ' + last\_name AS employee\_name,  
 email,  
 position,  
 salary,  
 hire\_date  
 FROM employees  
 WHERE department = @department AND is\_active = 1  
);  
  
-- Usage  
SELECT \* FROM fn\_employees\_by\_department('IT');

### Multi-Statement Table-Valued Function

CREATE FUNCTION fn\_employee\_salary\_range(  
 @min\_salary DECIMAL(10,2),  
 @max\_salary DECIMAL(10,2)  
)  
RETURNS @result TABLE (  
 employee\_id INT,  
 employee\_name VARCHAR(100),  
 department VARCHAR(50),  
 position VARCHAR(50),  
 salary DECIMAL(10,2),  
 salary\_category VARCHAR(20)  
)  
AS  
BEGIN  
 INSERT INTO @result  
 SELECT   
 employee\_id,  
 first\_name + ' ' + last\_name,  
 department,  
 position,  
 salary,  
 CASE   
 WHEN salary >= (@min\_salary + @max\_salary) / 2 THEN 'High'  
 ELSE 'Low'  
 END  
 FROM employees  
 WHERE salary BETWEEN @min\_salary AND @max\_salary  
 AND is\_active = 1;  
   
 RETURN;  
END;  
  
-- Usage  
SELECT \* FROM fn\_employee\_salary\_range(60000, 80000);

## Indexed Views (Materialized Views)

Indexed views physically store data and can significantly improve performance for complex queries.

### Creating an Indexed View

-- First create the view with specific requirements  
CREATE VIEW vw\_department\_summary  
WITH SCHEMABINDING  
AS  
SELECT   
 department,  
 COUNT\_BIG(\*) AS employee\_count,  
 SUM(salary) AS total\_salary,  
 AVG(salary) AS avg\_salary  
FROM dbo.employees  
WHERE is\_active = 1  
GROUP BY department;  
  
-- Create unique clustered index on the view  
CREATE UNIQUE CLUSTERED INDEX IX\_vw\_department\_summary   
ON vw\_department\_summary (department);  
  
-- Optionally create additional indexes  
CREATE NONCLUSTERED INDEX IX\_vw\_department\_summary\_salary   
ON vw\_department\_summary (total\_salary);  
  
-- Usage (SQL Server may automatically use the indexed view)  
SELECT \* FROM vw\_department\_summary;

### Requirements for Indexed Views:

* Must use SCHEMABINDING
* Must reference tables with schema prefix (dbo.table\_name)
* Cannot use certain functions (e.g., GETDATE(), USER)
* First index must be unique and clustered
* Must use COUNT\_BIG instead of COUNT

## Updating Data Through Views

### Simple View Updates

-- Create updatable view  
CREATE VIEW vw\_employee\_basic AS  
SELECT   
 employee\_id,  
 first\_name,  
 last\_name,  
 email,  
 phone,  
 position,  
 salary  
FROM employees  
WHERE is\_active = 1;  
  
-- Update through view  
UPDATE vw\_employee\_basic   
SET salary = 77000   
WHERE employee\_id = 1;  
  
-- Insert through view  
INSERT INTO vw\_employee\_basic (first\_name, last\_name, email, position, salary)  
VALUES ('New', 'Employee', 'new.employee@company.com', 'Analyst', 50000);  
  
-- Delete through view  
DELETE FROM vw\_employee\_basic WHERE employee\_id = 7;

### View with CHECK OPTION

-- Create view with CHECK OPTION to maintain data integrity  
CREATE VIEW vw\_high\_salary\_employees AS  
SELECT   
 employee\_id,  
 first\_name,  
 last\_name,  
 email,  
 salary,  
 department  
FROM employees  
WHERE salary >= 70000  
WITH CHECK OPTION;  
  
-- This will succeed  
UPDATE vw\_high\_salary\_employees SET salary = 75000 WHERE employee\_id = 1;  
  
-- This will fail due to CHECK OPTION  
UPDATE vw\_high\_salary\_employees SET salary = 65000 WHERE employee\_id = 1;

### INSTEAD OF Triggers for Complex Views

-- Create view that can't be directly updated  
CREATE VIEW vw\_employee\_department\_info AS  
SELECT   
 e.employee\_id,  
 e.first\_name + ' ' + e.last\_name AS full\_name,  
 e.email,  
 e.salary,  
 d.dept\_name,  
 d.location  
FROM employees e  
JOIN departments d ON e.department = d.dept\_name;  
  
-- Create INSTEAD OF trigger to handle updates  
CREATE TRIGGER tr\_update\_employee\_department  
ON vw\_employee\_department\_info  
INSTEAD OF UPDATE  
AS  
BEGIN  
 UPDATE employees   
 SET   
 email = i.email,  
 salary = i.salary  
 FROM employees e  
 INNER JOIN inserted i ON e.employee\_id = i.employee\_id;  
END;  
  
-- Now we can update through the view  
UPDATE vw\_employee\_department\_info   
SET salary = 80000   
WHERE employee\_id = 1;

## View Security and Permissions

### Column-Level Security

-- Create view that hides sensitive columns  
CREATE VIEW vw\_employee\_public AS  
SELECT   
 employee\_id,  
 first\_name + ' ' + last\_name AS full\_name,  
 department,  
 position,  
 -- Hide exact salary, show only range  
 CASE   
 WHEN salary >= 80000 THEN '$80,000+'  
 WHEN salary >= 60000 THEN '$60,000-$79,999'  
 WHEN salary >= 40000 THEN '$40,000-$59,999'  
 ELSE 'Below $40,000'  
 END AS salary\_range,  
 hire\_date  
FROM employees  
WHERE is\_active = 1;

### Row-Level Security with Views

-- Create view for department managers (they can only see their department)  
CREATE VIEW vw\_manager\_employees AS  
SELECT   
 employee\_id,  
 first\_name,  
 last\_name,  
 email,  
 position,  
 salary,  
 hire\_date,  
 department  
FROM employees  
WHERE department = (  
 SELECT department   
 FROM employees   
 WHERE employee\_id = USER\_ID() -- Assuming USER\_ID() returns current user's employee\_id  
) AND is\_active = 1;

### Granting Permissions on Views

-- Grant permissions on view without giving access to underlying tables  
GRANT SELECT ON vw\_employee\_public TO [PublicUsers];  
GRANT SELECT, INSERT, UPDATE ON vw\_employee\_basic TO [HRManagers];  
GRANT SELECT ON vw\_department\_stats TO [Executives];  
  
-- Revoke permissions  
REVOKE SELECT ON vw\_employee\_public FROM [PublicUsers];

## View Management in SSMS

### Using SSMS Interface

1. **Creating Views through SSMS:**
   * Right-click on “Views” folder → “New View”
   * Use graphical query designer or write SQL directly
   * Save with appropriate naming convention
2. **Modifying Views:**
   * Right-click view → “Design” for graphical editor
   * Right-click view → “Script View as” → “ALTER To” for SQL script
3. **Viewing Dependencies:**
   * Right-click view → “View Dependencies”
   * Shows which objects depend on this view and what this view depends on
4. **Checking View Definition:**
   * Right-click view → “Script View as” → “CREATE To”
   * Use system views: SELECT \* FROM INFORMATION\_SCHEMA.VIEWS

### System Views for View Information

-- List all views in database  
SELECT   
 TABLE\_SCHEMA,  
 TABLE\_NAME,  
 VIEW\_DEFINITION  
FROM INFORMATION\_SCHEMA.VIEWS;  
  
-- Get view definition  
SELECT OBJECT\_DEFINITION(OBJECT\_ID('vw\_employee\_directory'));  
  
-- Check view dependencies  
SELECT   
 o.name AS dependent\_object,  
 o.type\_desc,  
 d.referenced\_entity\_name  
FROM sys.sql\_dependencies d  
JOIN sys.objects o ON d.object\_id = o.object\_id  
WHERE d.referenced\_entity\_name = 'employees';

## Advanced View Examples

### Pivot View for Salary Comparison

CREATE VIEW vw\_salary\_by\_department\_position AS  
SELECT   
 position,  
 ISNULL([IT], 0) AS IT\_salary,  
 ISNULL([Sales], 0) AS Sales\_salary,  
 ISNULL([HR], 0) AS HR\_salary,  
 ISNULL([Finance], 0) AS Finance\_salary  
FROM (  
 SELECT position, department, AVG(salary) AS avg\_salary  
 FROM employees  
 WHERE is\_active = 1  
 GROUP BY position, department  
) AS SourceTable  
PIVOT (  
 AVG(avg\_salary)  
 FOR department IN ([IT], [Sales], [HR], [Finance])  
) AS PivotTable;  
  
-- Usage  
SELECT \* FROM vw\_salary\_by\_department\_position;

### Ranking View

CREATE VIEW vw\_employee\_rankings AS  
SELECT   
 employee\_id,  
 first\_name + ' ' + last\_name AS employee\_name,  
 department,  
 salary,  
 ROW\_NUMBER() OVER(ORDER BY salary DESC) AS overall\_rank,  
 ROW\_NUMBER() OVER(PARTITION BY department ORDER BY salary DESC) AS dept\_rank,  
 DENSE\_RANK() OVER(ORDER BY salary DESC) AS salary\_dense\_rank,  
 PERCENT\_RANK() OVER(ORDER BY salary) AS salary\_percentile,  
 NTILE(4) OVER(ORDER BY salary) AS salary\_quartile  
FROM employees  
WHERE is\_active = 1;  
  
-- Usage  
SELECT \* FROM vw\_employee\_rankings WHERE dept\_rank <= 3;

### Time-Based Analysis View

CREATE VIEW vw\_employee\_tenure\_analysis AS  
SELECT   
 employee\_id,  
 first\_name + ' ' + last\_name AS employee\_name,  
 department,  
 salary,  
 hire\_date,  
 DATEDIFF(MONTH, hire\_date, GETDATE()) AS months\_employed,  
 CASE   
 WHEN DATEDIFF(MONTH, hire\_date, GETDATE()) <= 6 THEN 'New (0-6 months)'  
 WHEN DATEDIFF(MONTH, hire\_date, GETDATE()) <= 24 THEN 'Growing (6-24 months)'  
 WHEN DATEDIFF(MONTH, hire\_date, GETDATE()) <= 60 THEN 'Experienced (2-5 years)'  
 ELSE 'Veteran (5+ years)'  
 END AS tenure\_category,  
 salary / NULLIF(DATEDIFF(MONTH, hire\_date, GETDATE()), 0) AS salary\_per\_month\_tenure,  
 CASE   
 WHEN DATEPART(QUARTER, hire\_date) = 1 THEN 'Q1'  
 WHEN DATEPART(QUARTER, hire\_date) = 2 THEN 'Q2'  
 WHEN DATEPART(QUARTER, hire\_date) = 3 THEN 'Q3'  
 ELSE 'Q4'  
 END AS hire\_quarter  
FROM employees  
WHERE is\_active = 1;  
  
-- Usage  
SELECT tenure\_category, COUNT(\*) as employee\_count, AVG(salary) as avg\_salary  
FROM vw\_employee\_tenure\_analysis  
GROUP BY tenure\_category;

## Best Practices

### 1. Naming Conventions

-- Use consistent prefixes  
CREATE VIEW vw\_employee\_summary AS ... -- "vw\_" prefix  
CREATE VIEW v\_department\_stats AS ... -- "v\_" prefix (alternative)

### 2. Documentation and Comments

/\*  
View: vw\_employee\_complete  
Purpose: Comprehensive employee information for reporting  
Author: Database Team  
Created: 2023-01-01  
Last Modified: 2023-06-01  
Notes: Includes department and manager information  
\*/  
CREATE VIEW vw\_employee\_complete AS  
SELECT   
 -- Employee basic info  
 e.employee\_id,  
 e.first\_name + ' ' + e.last\_name AS employee\_name,  
 -- ... other columns  
FROM employees e  
-- Join with departments for location info  
LEFT JOIN departments d ON e.department = d.dept\_name  
-- Join with managers  
LEFT JOIN employees m ON e.manager\_id = m.employee\_id  
WHERE e.is\_active = 1;

### 3. Performance Considerations

-- Create indexes on base tables for columns used in views  
CREATE INDEX IX\_employees\_department ON employees(department) WHERE is\_active = 1;  
CREATE INDEX IX\_employees\_salary ON employees(salary) WHERE is\_active = 1;  
  
-- Use appropriate WHERE clauses in views  
CREATE VIEW vw\_active\_employees AS  
SELECT \* FROM employees   
WHERE is\_active = 1 -- Filter early for performance  
AND hire\_date IS NOT NULL;

### 4. Avoid SELECT \* in Views

-- Good: Specify columns explicitly  
CREATE VIEW vw\_employee\_basic AS  
SELECT   
 employee\_id,  
 first\_name,  
 last\_name,  
 email,  
 department  
FROM employees;  
  
-- Avoid: SELECT \* makes view fragile  
CREATE VIEW vw\_employee\_all AS  
SELECT \* FROM employees; -- Avoid this

## Views vs Alternatives

| Feature | Views | CTEs | Stored Procedures | Table Functions |
| --- | --- | --- | --- | --- |
| **Reusability** | High | Single Query | High | High |
| **Parameters** | No | No | Yes | Yes |
| **Performance** | Good | Good | Excellent | Good |
| **Security** | Excellent | N/A | Good | Good |
| **Data Persistence** | No | No | No | No |
| **Indexed** | Yes (with restrictions) | No | No | No |
| **DML Operations** | Limited | No | Yes | No |

### When to Use Views:

* **Data Security**: Hide sensitive columns or rows
* **Simplification**: Present complex joins simply
* **Standardization**: Consistent data presentation
* **Legacy System Integration**: Abstract schema changes
* **Reporting**: Pre-built queries for reports

### When Not to Use Views:

* **Parameter Requirements**: Use functions instead
* **Complex Logic**: Use stored procedures
* **Temporary Results**: Use CTEs
* **Performance Critical**: Consider indexed views or tables

## Troubleshooting Common Issues

### Issue 1: View Not Updatable

-- Problem: Complex view with joins can't be updated  
-- Solution: Create INSTEAD OF triggers or use stored procedures  
  
CREATE TRIGGER tr\_update\_complex\_view  
ON vw\_complex\_view  
INSTEAD OF UPDATE  
AS  
BEGIN  
 -- Custom update logic here  
END;

### Issue 2: View Performance Problems

-- Problem: View is slow  
-- Solutions:  
-- 1. Add indexes to base tables  
CREATE INDEX IX\_employees\_dept\_salary ON employees(department, salary);  
  
-- 2. Create indexed view (if possible)  
-- 3. Use NOEXPAND hint for indexed views  
SELECT \* FROM vw\_department\_summary WITH (NOEXPAND);

### Issue 3: Schema Binding Issues

-- Problem: Can't create indexed view  
-- Solution: Use SCHEMABINDING and proper syntax  
CREATE VIEW vw\_indexed\_view  
WITH SCHEMABINDING  
AS  
SELECT   
 department,  
 COUNT\_BIG(\*) AS employee\_count  
FROM dbo.employees -- Must use schema prefix  
GROUP BY department;

## Summary

Views in SQL Server are powerful tools for: - **Data Security**: Controlling access to sensitive information - **Code Reusability**: Creating reusable query logic - **Simplification**: Making complex queries accessible - **Performance**: Improving query performance with indexed views - **Maintenance**: Centralizing business logic in the database

Master views to create more secure, maintainable, and efficient database applications!