SQL Views Simplifying Complex Queries

AusMine Resources Mining Operations

Video| Duration: 15 minutes | Level: Intermediate

THE ANALYST NIGHTMARE

Monday 8:00 AM - New Analyst Sarah

"I need the monthly site performance report by 10 AM"

The Problem:

- ❌ 60+ lines of complex SQL to write
- ❌ 5 LEFT JOINs across multiple tables
- ❌ Complex date filters and calculations
- ❌ Junior analysts struggle for hours
- ❌ Copy-paste errors everywhere

Result:

- 2 hours to create ONE report
- Inconsistent calculations
- Frequent mistakes
- Reports often late

THE BUSINESS CHALLENGES

AusMine Resources Pain Points

1

Complexity Crisis

- 15 different tables to join
- Same query written 50 ways
- Junior analysts can't write reports

2

Security Nightmare

- Salary data exposed to everyone
- Equipment costs visible
- Compliance risks

3

Inconsistency Problem

- "Active employees" defined differently
- Cost per tonne calculated inconsistently
- Management gets conflicting numbers

4

Performance Issues

- Complex queries take 5-10 minutes
- Reports timeout frequently

WHAT ARE SQL VIEWS?

Virtual Tables for Data Access

Definition:

A view is a saved SELECT query that acts like a virtual table

Think of it as:

- Pre-built "window" into your data
- Saved query with a simple name
- Virtual table (no data stored)
- Reusable across applications

Creating a View:

```
CREATE VIEW vw_ActiveSites AS
SELECT
  site_id,
  site_name,
  state,
  resource_type
FROM sites
WHERE status = 'Active';
```

Using the View:

```
SELECT * FROM vw_ActiveSites;
```

That's it! Hide complexity behind simple names.

DEMO 1 - SIMPLE VIEW

Security View: Hiding Salary Data

The Problem:

Employees table contains sensitive data (salaries, superannuation)

Junior analysts need directory info but NOT salary data

The Solution: Create a Security View

```
CREATE VIEW vw_EmployeeDirectory AS
SELECT
  employee_code,
  first_name + ' ' + last_name AS full_name,
  email,
  job_title,
  department,
  site_name,
  hire_date
  -- NOTE: annual_salary EXCLUDED
FROM employees e
JOIN sites s ON e.site_id = s.site_id
WHERE employment_status = 'Active';
```

Now junior analysts query:

```
SELECT * FROM vw_EmployeeDirectory;
```

✓ No salary data visible

✓ No access to base table needed

✓ Centralized security control

DEMO 2 - COMPLEX DASHBOARD VIEW

Executive Dashboard Simplified

Before Views (60+ lines): Complex query with 5 JOINs, multiple aggregations, date filters

After Views (1 simple line):

```
CREATE VIEW vw_SitePerformanceDashboard AS
SELECT
  site_name,
  state,
  resource_type,
  COUNT(DISTINCT employees) AS total_employees,
  SUM(tonnes_produced) AS december_production,
  SUM(costs) AS december_costs,
  COUNT(incidents) AS safety_incidents,
  ROUND(costs / tonnes_produced, 2) AS cost_per_tonne
FROM [complex joins and filters]
GROUP BY site_name, state, resource_type;
```

Executives now query:

```
SELECT * FROM vw_SitePerformanceDashboard
ORDER BY december_production DESC;
```

Transformation:

BEFORE: 60 lines → **AFTER:** 1 line

Time:

30 minutes → 30 seconds

VIEW TYPES & USE CASES

Four Types of Views

1

Simple Views

- Single table filtering
- Example: vw_ActiveSites
- Use: Simplify common WHERE clauses

2

Security Views

- Hide sensitive columns
- Example: vw_EmployeeDirectory (no salaries)
- Use: Column-level security

3

Analytical Views

- Complex aggregations and JOINs
- Example: vw_SitePerformanceDashboard
- Use: Reports and dashboards

4

Indexed Views (Advanced)

- Physically stored results
- Example: vw_DailyProductionSummary
- Use: Performance optimization for large datasets

Best Practice: Choose the right view type for your scenario!

INDEXED VIEWS - PERFORMANCE BOOST

When Regular Views Aren't Fast Enough

Problem:

Large aggregation queries still slow even with views

Solution:

Indexed Views (Materialized Views)

```
CREATE VIEW vw_DailyProductionSummary
WITH SCHEMABINDING -- Required for indexed views
AS
SELECT
  site_id,
  production_date,
  resource_type,
  SUM(tonnes_produced) AS total_tonnes,
  SUM(operating_hours) AS total_hours
FROM dbo.production
GROUP BY site_id, production_date, resource_type;

-- Add unique clustered index
CREATE UNIQUE CLUSTERED INDEX IX_DailyProdSummary
ON vw_DailyProductionSummary(site_id, production_date);
```

✓ Results pre-calculated and stored physically

✓ Queries run 10-100x faster

✓ Automatically maintained by SQL Server

✓ Perfect for large dataset aggregations

VIEW BEST PRACTICES

Production-Ready Guidelines

✓ DO:

- Use clear naming: vw_PurposeDescription
- Add WITH SCHEMABINDING for indexed views
- Document complex logic with comments
- Create views for repeated queries
- Use views for security (hide sensitive columns)
- Grant permissions on views, not base tables

❌ DON'T:

- Put SELECT * in views (specify columns)
- Create views on views on views (3+ levels)
- Use ORDER BY in views (add when querying)
- Forget to consider performance impact
- Expose sensitive data through views

Managing Views:

- ALTER VIEW to modify existing views
- DROP VIEW to remove views
- sp_helptext 'vw_Name' to see definition
- sp_refreshview to update metadata

THE TRANSFORMATION - BEFORE vs AFTER

Business Impact Summary

BEFORE SQL VIEWS:

- ❌ 60+ line queries written repeatedly
- ❌ Junior analysts struggle (2+ hours per report)
- ❌ Inconsistent calculations across reports
- ❌ Security risks (sensitive data exposed)
- ❌ Poor performance on complex queries
- ❌ High maintenance (change logic in 50 places)

AFTER SQL VIEWS:

- ✓ Simple SELECT * FROM vw_SitePerformance
- ✓ Anyone can query complex data (5 minutes)
- ✓ Consistent logic across ALL reports
- ✓ Secure (sensitive columns hidden)
- ✓ Better performance (indexed views)
- ✓ Easy maintenance (change view once)

Business Metrics:

75%
Query Development Time
reduction

100%
Report Consistency
(same view for all)

3x
Junior Analyst Productivity
increase

90%
Security Incidents
reduction

80%
Maintenance Effort
reduction

Career Impact:

- ✓ Enterprise database design patterns mastered
- ✓ Data security implementation skills
- ✓ Self-service analytics enablement
- ✓ Production-ready SQL development