# SQL Server Hidden Dependencies Resolution Guide

This guide provides methods for identifying and resolving hidden dependencies on SQL Server columns, steps to troubleshoot, document dependencies, and ensure smooth schema modifications in complex databases.

## Table of Contents

1. Overview of Hidden Dependencies in SQL Server.......................................4

2. Use Case: Dropping a Column with Unidentified Dependencies.........................5

3. Step-by-Step Guide for Identifying Dependencies....................................6

3.1 Query for General Dependencies...............................................6

3.2 Query for Indexes............................................................7

3.3 Query for Constraints.........................................................8

3.4 Query for Triggers............................................................9

4. Example Scenario and Resolution Steps: Troubleshooting RoleLevel in core.Type.....10

5. Conclusion and Best Practices.....................................................12

## Overview of Hidden Dependencies in SQL Server

In SQL Server, hidden dependencies, such as indexes, constraints, and triggers, can cause issues when modifying or dropping columns. This guide provides a systematic approach for identifying these dependencies, allowing you to handle them effectively.

## Use Case: Dropping a Column with Unidentified Dependencies

Consider a scenario where attempts to drop the 'RoleLevel' column from a table ('core.Type') fail due to SQL Server reporting dependencies. Initial checks may reveal no dependencies in sys.sql\_expression\_dependencies, but hidden dependencies, such as indexes or constraints, may still exist.

## Step-by-Step Guide for Identifying Dependencies

Below are SQL queries for identifying different types of dependencies on a specific column.

### 1. Query for General Dependencies

Use the following query to locate common dependencies, like views or functions, referencing a column.

DECLARE @SchemaName NVARCHAR(128) = 'schema\_name';  
DECLARE @TableName NVARCHAR(128) = 'table\_name';  
DECLARE @ColumnName NVARCHAR(128) = 'column\_name';  
  
SELECT o.name AS referencing\_object\_name,  
 o.type\_desc AS referencing\_object\_type,  
 col.name AS referenced\_column,  
 dep.referenced\_entity\_name AS referenced\_table  
FROM sys.sql\_expression\_dependencies AS dep  
JOIN sys.objects AS o ON dep.referencing\_id = o.object\_id  
JOIN sys.columns AS col ON dep.referenced\_id = col.object\_id  
 AND dep.referenced\_minor\_id = col.column\_id  
WHERE SCHEMA\_NAME(o.schema\_id) = @SchemaName  
 AND dep.referenced\_entity\_name = @TableName  
 AND col.name = @ColumnName;

### 2. Query for Indexes

To find indexes referencing the column, use the query below. It searches sys.indexes, sys.index\_columns, and sys.columns to list all indexes that include the specified column.

DECLARE @SchemaName NVARCHAR(128) = 'schema\_name';  
DECLARE @TableName NVARCHAR(128) = 'table\_name';  
DECLARE @ColumnName NVARCHAR(128) = 'column\_name';  
  
SELECT i.name AS index\_name,  
 i.type\_desc AS index\_type,  
 c.name AS column\_name  
FROM sys.indexes AS i  
JOIN sys.index\_columns AS ic ON i.object\_id = ic.object\_id  
 AND i.index\_id = ic.index\_id  
JOIN sys.columns AS c ON ic.object\_id = c.object\_id  
 AND ic.column\_id = c.column\_id  
WHERE OBJECT\_SCHEMA\_NAME(i.object\_id) = @SchemaName  
 AND OBJECT\_NAME(i.object\_id) = @TableName  
 AND c.name = @ColumnName;

### 3. Query for Constraints

Constraints, such as primary keys, unique constraints, and foreign keys, may also reference columns.

DECLARE @SchemaName NVARCHAR(128) = 'schema\_name';  
DECLARE @TableName NVARCHAR(128) = 'table\_name';  
DECLARE @ColumnName NVARCHAR(128) = 'column\_name';  
  
SELECT con.name AS constraint\_name,  
 con.type\_desc AS constraint\_type,  
 col.name AS column\_name  
FROM sys.objects AS con  
JOIN sys.columns AS col ON con.parent\_object\_id = col.object\_id  
JOIN sys.sysconstraints AS syscon ON con.object\_id = syscon.constid  
WHERE OBJECT\_SCHEMA\_NAME(con.parent\_object\_id) = @SchemaName  
 AND OBJECT\_NAME(con.parent\_object\_id) = @TableName  
 AND col.name = @ColumnName;

### 4. Query for Triggers

If the table has triggers, they might include operations referencing the specified column. This query identifies any triggers on the table.

DECLARE @SchemaName NVARCHAR(128) = 'schema\_name';  
DECLARE @TableName NVARCHAR(128) = 'table\_name';  
  
SELECT trg.name AS trigger\_name,  
 trg.type\_desc AS trigger\_type  
FROM sys.triggers AS trg  
JOIN sys.objects AS tbl ON trg.parent\_id = tbl.object\_id  
WHERE OBJECT\_SCHEMA\_NAME(tbl.schema\_id) = @SchemaName  
 AND tbl.name = @TableName;

## Example Scenario and Resolution Steps: Troubleshooting RoleLevel in core.Type

In our use case, attempts to drop the 'RoleLevel' column in 'core.Type' failed due to hidden dependencies. To proceed with the column drop operation, follow these steps:

1. \*\*Drop the identified Index\*\*: Use the following command to drop the index:  
  
 DROP INDEX IX\_Type\_RoleLevel ON core.Type;  
2. \*\*Drop the Primary Key Constraint\*\*: If the column is part of the primary key, drop the key constraint:  
  
 ALTER TABLE core.Type DROP CONSTRAINT PK\_\_Type\_\_516F03B56BB44873;