Cross–Database Ownership Chaining (CDOC) is a security feature in **SQL Server** that allows stored procedures, views, or functions in one database to access objects in another database *without requiring explicit permissions* on the underlying objects **as long as the owners of all involved objects are the same**.

It can simplify application logic, but if misused, it can also unintentionally expose data across databases. Understanding it deeply is important for DBAs and developers.

✅ What Is Ownership Chaining?

An **ownership chain** is created when SQL Server checks permissions on a sequence of objects that call each other.

Example (inside a single DB):

EXEC ProcA → SELECT from ViewB → SELECT from TableC

If **TableC**, **ViewB**, and **ProcA** share the **same owner** (usually dbo), SQL Server skips permission checks on the underlying objects. It only checks permission on the *initial* object (ProcA).
This is called **ownership chaining**.

🔗 Cross–Database Ownership Chaining

When this chain crosses **database boundaries**, it becomes a **cross-database ownership chain**.
Example:

DatabaseA.dbo.Proc1 → DatabaseB.dbo.Table1

SQL Server will allow this access **only if both objects have the same owner** (usually dbo *AND* both db owners match).
Otherwise, the chain is broken.

✔ Conditions for CDOC to Work

Condition	**Required?**	**Notes**
Same user owns objects in both DBs	**Yes**	In practice, both objects owned by dbo and both DBs owned by the *same login*.
Cross-database ownership chaining enabled	**Maybe**	At instance or per-database level.
User has permission on the *first* object	**Yes**	SQL Server doesn't check deeper objects.

⚙ How to Enable or Disable Cross-DB Ownership Chaining

Server-wide (instance-level) setting

```
sp_configure 'cross db ownership chaining', 1;
RECONFIGURE;
```

Database-level

```
ALTER DATABASE DatabaseA SET DB_CHAINING ON;
ALTER DATABASE DatabaseB SET DB_CHAINING ON;
```

📘 Real-Time Example Scenario 1: Multi-Database Application (ERP/CRM)

🎯 Situation

A company has:

- **AppDB** → Contains stored procedures
- **AuditDB** → Stores transactional logs

Application users should **not have direct access** to AuditDB tables.

🟦 Without CDOC (Default Behavior)

User calls a stored procedure:

EXEC AppDB.dbo.InsertOrder
Inside InsertOrder:
INSERT INTO AuditDB.dbo.OrderAuditLog (...)
SELECT ...
FROM AppDB.dbo.Orders

❌ This fails **unless the user has INSERT permission on AuditDB.dbo.OrderAuditLog**, which is undesirable (security risk).

🟩 **With CDOC Enabled**

- Both DBs owned by dbo.
- DB_CHAINING = ON.
- User only needs EXECUTE on the procedure.

Now:
EXEC AppDB.dbo.InsertOrder

✓ Successfully writes into AuditDB.dbo.OrderAuditLog
✓ User has **no direct access** to AuditDB tables
✓ Permissions simplified

📘 **Real-Time Example Scenario 2: Microservices Database Pattern**

A company uses separate databases for each module:

- **BillingDB**
- **InventoryDB**
- **CustomerDB**

A stored procedure in BillingDB:
BillingDB.dbo.GetInvoiceForCustomer
…needs to read customer address from CustomerDB.
Without CDOC, you must grant Billing app users **SELECT on CustomerDB** tables → security issue.

With CDOC:

- Enable DB_CHAINING on both DBs
- Ensure same owner

Now:
EXEC BillingDB.dbo.GetInvoiceForCustomer 123
→ Procedure reads from CustomerDB.dbo.Customers
→ User has no direct access to CustomerDB

📘 **Real-Time Example 3: Shared Security Database**

Large enterprise uses:

- **SecurityDB** → Stores users, roles, audit logs
- **ApplicationDB** → Regular OLTP

Stored procedures in ApplicationDB need to log authentication attempts in SecurityDB.
CDOC allows:
ApplicationDB.dbo.LogAttempt
→ writes to SecurityDB.dbo.LoginAttempts
No need to grant the application user direct rights.

⚠ **Security Risks & Why Microsoft Recommends Caution**

🚨 **Risk 1: Implicit Access Your Admins Didn't Expect**

If someone creates new objects owned by dbo in both DBs:
DB1.dbo.NewProc → DB2.dbo.SensitiveTable

The chain works automatically, potentially exposing sensitive data.

🚨 **Risk 2: Database Owner = Login**

If databases have different owners, cross-chaining **won't work**, but if accidentally changed to same owner, access suddenly becomes possible.

🚨 **Risk 3: Privilege Escalation via "Trojan Horse" Stored Procedures**

Malicious developer inserts:

```
CREATE PROCEDURE dbo.GetAllPayrollData AS
SELECT * FROM PayrollDB.dbo.EmployeeSalary;
```

If both DBs are dbo-owned and chaining enabled → user may get sensitive data.

🎯 **Best Practices**

✔ Avoid enabling CDOC at server level

✔ Enable DB_CHAINING **only on specific trusted DB pairs**

✔ Ensure strict change management

✔ Audit cross-database access

✔ Prefer **signed stored procedures** or **module signing** (more secure)

📝 **Summary in Simple Words**

Cross-DB Ownership Chaining lets SQL Server skip permission checks across databases, **as long as the same owner exists**. It is useful for multi-database applications but risky if misunderstood.

Below is a **complete, hands-on, DBA-friendly guide** including:

1. **Step-by-step demo scripts** for Cross-Database Ownership Chaining
2. **Secure alternatives** (module signing with certificates)
3. **Troubleshooting broken ownership chains** with real causes + fixes

🔷 1. Step-by-Step Demo Scripts for Cross-Database Ownership Chaining

This demo uses two databases:

- **AppDB** — application code
- **AuditDB** — stores audit logs

Goal: A user executes a stored procedure in AppDB, which writes into AuditDB **without giving the user permissions on AuditDB**.

🧰 Step 1 — Create Database

```
CREATE DATABASE AppDB;
CREATE DATABASE AuditDB;
```

🧰 Step 2 — Enable DB Chaining on Both Databases

Required because we are crossing databases.

```
ALTER DATABASE AppDB SET DB_CHAINING ON;
ALTER DATABASE AuditDB SET DB_CHAINING ON;
```

🧰 Step 3 — Create Tables & Stored Procedures

In AppDB

```
USE AppDB;
GO
CREATE TABLE dbo.Orders
(
   OrderId INT IDENTITY PRIMARY KEY,
   CustomerName VARCHAR(50)
);
GO

CREATE PROCEDURE dbo.InsertOrder
(
   @CustomerName VARCHAR(50)
)
AS
BEGIN
   INSERT INTO dbo.Orders (CustomerName)
   VALUES (@CustomerName);

   -- Write audit log in other database
   INSERT INTO AuditDB.dbo.OrderAuditLog (OrderId, ActionTime)
   SELECT SCOPE_IDENTITY(), GETDATE();
END
GO
```

In AuditDB

```
USE AuditDB;
GO
```

```
CREATE TABLE dbo.OrderAuditLog
(
   AuditId INT IDENTITY PRIMARY KEY,
   OrderId INT,
   ActionTime DATETIME
);
GO
```

🧰 **Step 4 — Create a Test User**

```
USE master;
GO
CREATE LOGIN TestUser WITH PASSWORD = 'StrongPassword123!';
GO

USE AppDB;
CREATE USER TestUser FOR LOGIN TestUser;
GRANT EXECUTE ON dbo.InsertOrder TO TestUser;
GO
```

👉 **Important**:

TestUser is **NOT** given any access to AuditDB.

🧰 **Step 5 — Test Behavior**

As TestUser:

```
EXEC AppDB.dbo.InsertOrder 'John Doe';
```

✔ **This works successfully**

✔ User inserts into AuditDB **without permissions** on AuditDB

✔ This is cross-database ownership chaining working properly

🔷 **2. Security Alternatives (Better and Safer Than CDOC)**

CDOC is convenient but risky.

The **recommended secure alternative** is:

⭐ **Use Module Signing with Certificates**

This allows stored procedures to access other databases **without enabling DB_CHAINING** and without granting users extra privileges.

🧰 **Example: Replace CDOC with Module Signing**

Goal: AppDB.dbo.InsertOrder should insert into AuditDB.dbo.OrderAuditLog **without granting user any permissions** on AuditDB.

✔ **Step 1 — Create Certificate in AppDB**

```
USE AppDB;
GO

CREATE CERTIFICATE AppDBCert
   ENCRYPTION BY PASSWORD = 'StrongPassword123!'
   WITH SUBJECT = 'Cross DB Access Certificate';
GO
```

✔ **Step 2 — Create a Signing Login (Stored Procedure Identity)**

```
CREATE LOGIN AppDBCertUser FROM CERTIFICATE AppDBCert;
GO
```

✓ Step 3 — Give Certificate Login Required Permissions in AuditDB

```
USE AuditDB;
GO
CREATE USER AppDBCertUser FOR LOGIN AppDBCertUser;
GRANT INSERT ON dbo.OrderAuditLog TO AppDBCertUser;
```

✓ Step 4 — Sign the Procedure with the Certificate

```
USE AppDB;
GO

ADD SIGNATURE TO dbo.InsertOrder
   BY CERTIFICATE AppDBCert
   WITH PASSWORD = 'StrongPassword123!';
GO
```

✓ Step 5 — Disable DB_CHAINING if you want maximum security

```
ALTER DATABASE AppDB SET DB_CHAINING OFF;
ALTER DATABASE AuditDB SET DB_CHAINING OFF;
GO
```

🎉 **Result:**

- User only needs EXECUTE on the stored procedure
- SQL Server impersonates the certificate user internally
- No cross-db chaining required
- No excessive permissions granted
- Secure and audit-friendly

🔷 3. Troubleshooting Broken Ownership Chains

Below are the **4 most common issues** that break CDOC and how to fix them.

❌ Issue 1 — Databases Have Different Owners

Check owner:

```
SELECT name, SUSER_SNAME(owner_sid) FROM sys.databases;
```

Fix:

```
ALTER AUTHORIZATION ON DATABASE::AppDB TO sa;
ALTER AUTHORIZATION ON DATABASE::AuditDB TO sa;
```

❌ Issue 2 — Object Owners Are Not the Same (dbo vs custom schema)

If object is owned by different schemas:

AppDB.dbo.Proc1 → AuditDB.audit.OrderAuditLog

Ownership chain **breaks**.

Fix: Use dbo schema on both sides:

```
ALTER SCHEMA dbo TRANSFER AuditDB.audit.OrderAuditLog;
```

❌ Issue 3 — DB_CHAINING Is OFF

Check:

```
SELECT name, is_db_chaining_on
FROM sys.databases;
```

Fix:

```
ALTER DATABASE AppDB SET DB_CHAINING ON;
ALTER DATABASE AuditDB SET DB_CHAINING ON;
```

❌ Issue 4 — Dynamic SQL breaks ownership chaining

Example inside procedure:

```
EXEC('INSERT INTO AuditDB.dbo.OrderAuditLog VALUES (...)');
```

🚨 Dynamic SQL always breaks ownership chaining

SQL Server must check permissions again.

Fix options:

- Avoid dynamic SQL
- OR use module signing (recommended)
- OR explicitly grant permissions (less secure)

🎯 Final Summary

Method	**Security**	**Maintenance**	**Risk**	**Recommended?**
Cross-DB Ownership Chaining	Medium	Easy	High	❌ Only for trusted systems
Explicit Permissions	Low	Hard	Low	❌ Too much access given
Module Signing (Certificates)	**High**	Medium	Very Low	✅ **Best practice**
Application-level Security	High	App-dependent	Medium	Good alternative

Below is a **ready-to-run auditing toolkit** that helps you identify **ALL cross-database ownership chains**, dangerous access patterns, broken chains, database owners, and objects referencing other databases.
These are DBA-level scripts for SQL Server audits.

🛠 1. Check Database-Level Cross-DB Chaining Settings

```
SELECT
   name AS DatabaseName,
   CASE WHEN is_db_chaining_on = 1 THEN 'ON' ELSE 'OFF' END AS DBChaining,
   SUSER_SNAME(owner_sid) AS DatabaseOwner
FROM sys.databases
ORDER BY name;
```

✓ Shows which databases have **DB_CHAINING ON**

✓ Lists **database owners** (must match for chaining to work)

🛠 2. Find Dangerous Cross-Database References (Views, Procedures, Functions)

This identifies objects in one DB referencing objects in another DB.

```
SELECT
   DB_NAME() AS CurrentDB,
   o.type_desc AS ObjectType,
   o.name AS ObjectName,
   referenced_database_name AS ReferencedDB,
   referenced_schema_name AS ReferencedSchema,
   referenced_entity_name AS ReferencedObject
FROM sys.sql_expression_dependencies d
JOIN sys.objects o
     ON d.referencing_id = o.object_id
WHERE referenced_database_name IS NOT NULL
ORDER BY ReferencedDB, o.name;
```

✓ Shows **which objects call other databases**

✓ Helps detect **accidental or hidden cross-database calls**

🛠 3. Find Dynamic SQL That Breaks Ownership Chaining

Dynamic SQL breaks secure chaining and often introduces privilege escalation risks.

```
SELECT
   o.name AS ObjectName,
   o.type_desc AS ObjectType,
   m.definition AS Code
FROM sys.sql_modules m
JOIN sys.objects o ON m.object_id = o.object_id
WHERE m.definition LIKE '%EXEC(%'
  OR m.definition LIKE '%sp_executesql%'
  OR m.definition LIKE '%EXECUTE(%'
ORDER BY o.name;
```

✓ Flags risky code

✓ Useful for reviewing potential permission bypasses

🛠 4. Find Objects Owned by Non-dbo Owners

Mismatched owners break cross-database chains.

```
SELECT
   name AS ObjectName,
   type_desc AS ObjectType,
   USER_NAME(schema_id) AS SchemaName
FROM sys.objects
```

```
WHERE schema_id != SCHEMA_ID('dbo')
ORDER BY SchemaName, ObjectName;
```

✓ Identifies objects not owned by dbo

✓ Useful when chains mysteriously break

🛠 5. Check for Mixed Database Owners (Common Cause of Broken Chains)

Cross-db chaining requires databases to have **matching owners**.

```
SELECT
   name AS DatabaseName,
   SUSER_SNAME(owner_sid) AS Owner
FROM sys.databases
ORDER BY Owner, name;
```

✓ Helps you verify if databases involved in chaining have the same owner

🛠 6. Detect Dangerous Cross-Database Write Operations

Identifies procedures or views that **insert/update/delete** objects in other DBs.

```
SELECT
   o.name AS ObjectName,
   o.type_desc AS ObjectType,
   m.definition AS Definition
FROM sys.objects o
JOIN sys.sql_modules m ON o.object_id = m.object_id
WHERE m.definition LIKE '%insert into %.%'
  OR m.definition LIKE '%update %.%'
  OR m.definition LIKE '%delete from %.%'
ORDER BY o.name;
```

✓ Helps find hidden or unapproved write operations across DBs

🛠 7. Comprehensive Cross-Database Dependency Map (Best Audit Script)

This creates a full dependency list with DB-to-DB links.

```
SELECT
   DB_NAME() AS SourceDB,
   SCHEMA_NAME(o.schema_id) AS SourceSchema,
   o.name AS SourceObject,
   o.type_desc AS SourceObjectType,
   d.referenced_database_name AS TargetDB,
   d.referenced_schema_name AS TargetSchema,
   d.referenced_entity_name AS TargetObject,
   d.referenced_class_desc AS ReferenceType
FROM sys.sql_expression_dependencies d
JOIN sys.objects o
     ON d.referencing_id = o.object_id
WHERE d.referenced_database_name IS NOT NULL
ORDER BY TargetDB, SourceObject;
```

✓ Full map of cross-database relationships

✓ Ideal for security audits and application dependency analysis

🛠 8. Show Which Objects Would Break If Chaining Was Disabled

```
SELECT
   o.name AS ObjectName,
   o.type_desc AS ObjectType,
   d.referenced_database_name AS ExternalDatabase,
    d.referenced_entity_name AS ExternalObject
FROM sys.objects o
JOIN sys.sql_expression_dependencies d
    ON o.object_id = d.referencing_id
WHERE d.referenced_database_name IS NOT NULL
ORDER BY ExternalDatabase;
```

✓ Identifies objects that depend on CDOC

✓ Useful before disabling DB_CHAINING

🛠 9. Find External Access via 3-Part Names

```
SELECT
    o.name AS ObjectName,
    o.type_desc AS ObjectType,
    m.definition
FROM sys.objects o
JOIN sys.sql_modules m ON o.object_id = m.object_id
WHERE m.definition LIKE '%[%].[%].[%]%'
   OR m.definition LIKE '%[A-Za-z0-9_]%.dbo.%'
ORDER BY o.name;
```

✓ Finds code using explicit 3-part names

✓ Helps locate cross-db calls buried in stored procedure code

🛠 10. DBA-Level Summary Report (Everything in One)

Paste and run in **master**:

```
EXEC sp_MSforeachdb '
USE [?];
PRINT ''=== Database: [?] ===''; 

-- DB chaining
SELECT
    DB_NAME() AS DBName,
    is_db_chaining_on AS DBChaining,
    SUSER_SNAME(owner_sid) AS Owner
FROM sys.databases WHERE name = DB_NAME();

-- Cross-database references
SELECT
    o.name AS ObjectName,
    o.type_desc AS ObjectType,
    d.referenced_database_name AS RefDB,
    d.referenced_schema_name AS RefSchema,
    d.referenced_entity_name AS RefEntity
FROM sys.sql_expression_dependencies d
JOIN sys.objects o ON o.object_id = d.referencing_id
WHERE d.referenced_database_name IS NOT NULL;
';
```

✓ Scans *all databases*

✓ Produces a unified audit log

✓ Excellent for penetration testing and compliance