Creatio Academy

Instruments for working with database

Version 8.0

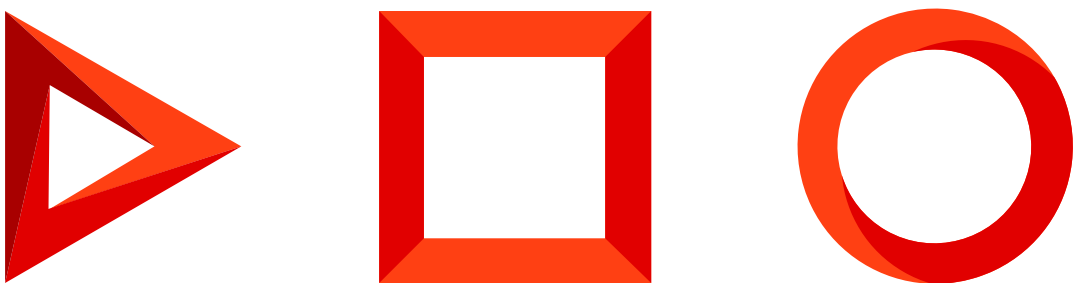


Table of Contents

Special features of working with PostgreSQL

Medium

General recommendations

- Create triggers, views, and functions using the `DROP … IF EXISTS` expression (if needed, you can use the `CASCADE` command), then the `CREATE OR REPLACE` expression. Do not use the `CREATE OR REPLACE` command alone.
- Use `"public"` instead of the `"dbo"` schema.
- Note that system names are case-sensitive. Use quotes (`""`) for names of tables, columns, and other entities.
- Use the `BOOL` type instead of the `BIT` type used in Microsoft SQL. You do not have to use the `WHERE "boolColumn" = true` expression to match the value of a `BOOL` type field. You can use the `WHERE "boolColumn"` or `WHERE NOT "boolColumn"` expression instead.
- You can use the short form of the `::TEXT` explicit conversion.
- String matching in PostgreSQL is case-sensitive. You can use the `iLIKE` keyword to execute a case-insensitive matching. Note that matching that uses this keyword is slower than matching that uses the `UPPER+LIKE` expression. The `UPPER+LIKE` expression has less strict index applicability rules than `iLIKE`.
- You can use the `CREATE CAST` command to cast types if source code has no implicit type casting. Learn more about the type casting in the official PostgreSQL documentation.
- Create a special procedure parameter to store the current recursion level because PostgreSQL recursive procedures do not have a built-in `NESTLEVEL` function.
- Use the `NAME` type in PostgreSQL instead of the `SYSNAME` type used in Microsoft SQL.
- Create rules instead of empty `INSTEAD` triggers. For example:

```
CREATE RULE RU_VwAdministrativeObjects AS
ON UPDATE TO "VwAdministrativeObjects"
DO INSTEAD NOTHING;
```

- Convert `INT` type to `BOOL` type explicitly. Implicit conversion of `INT` type to `BOOL` type does not work in PostgreSQL if the corresponding `CAST` statement is present and the `UPDATE` command is executed.
- Use allowed ways to format string literals. Learn more about string literals in the official PostgreSQL documentation (quote_ident, quote_literal, format).
- Use the following expression

```
DECLARE rowsCount BIGINT = 0;
GET DIAGNOSTICS rowsCount = row_count;
```

instead of `@@ROWCOUNT`.

- Use the following expression

```
EXISTS (
 SELECT 1
 FROM "SysSSPEntitySchemaAccessList"
 WHERE "EntitySchemaUId" = BaseSchema."UId"
) "IsInSSPEntitySchemaAccessList"
```

instead of the following Microsoft SQL expression

```
(CASE
    WHEN EXISTS
        (SELECT 1
            FROM [SysSSPEntitySchemaAccessList]
            WHERE [SysSSPEntitySchemaAccessList].[EntitySchemaUId] = [BaseSchemas].[UId] ) TH
    ELSE 0
END) AS [IsInSSPEntitySchemaAccessList]
```

The response field will have a `BOOL` type.

Data type matching

Creatio, Microsoft SQL, and PostgreSQL data type matching

Type value in the Creatio object Designer	Data type	
	Microsoft SQL	PostgreSQL
BLOB	`VARBINARY`	`BYTEA`
Boolean	`BIT`	`BOOLEAN`
Color	`NVARCHAR`	`CHARACTER VARYING`
CRC	`NVARCHAR`	`CHARACTER VARYING`
Currency	`DECIMAL`	`NUMERIC`
Date	`DATE`	`DATE`
Date/Time	`DATETIME2`	`TIMESTAMP WITHOUT TIME ZONE`
Decimal (0.00000001)	`DECIMAL`	`NUMERIC`

Decimal (0.0001)	DECIMAL	NUMERIC
Decimal (0.001)	DECIMAL	NUMERIC
Decimal (0.01)	DECIMAL	NUMERIC
Decimal (0.1)	DECIMAL	NUMERIC
Encrypted string	NVARCHAR	CHARACTER VARYING
File	VARBINARY	BYTEA
Image	VARBINARY	BYTEA
Image Link	UNIQUEIDENTIFIER	UUID
Integer	INTEGER	INTEGER
Lookup	UNIQUEIDENTIFIER	UUID
Text (250 characters)	NVARCHAR(250)	CHARACTER VARYING
Text (50 characters)	NVARCHAR(50)	CHARACTER VARYING
Text (500 characters)	NVARCHAR(500)	CHARACTER VARYING
Time	TIME	TIME WITHOUT TIME ZONE
Unique identifier	UNIQUEIDENTIFIER	UUID
Unlimited length text	NVARCHAR(MAX)	TEXT

MSSQL and PostgreSQL script examples

Medium

Example 1 (views)

Example. Example of an SQL script that creates a view and triggers to add, modify, and delete records from the target table.

MSSQL

```
-- View and triggers that let you modify the target table
-- MSSQL
IF EXISTS (SELECT * FROM sys.views WHERE object_id = OBJECT_ID(N'[dbo].[VwSysAdminUnit]'))
DROP VIEW [dbo].[VwSysAdminUnit]
GO
CREATE VIEW [dbo].[VwSysAdminUnit]
AS
SELECT [SysAdminUnit].[Id]
    ,[SysAdminUnit].[CreatedOn]
    ,[SysAdminUnit].[CreatedById]
    ,[SysAdminUnit].[ModifiedOn]
    ,[SysAdminUnit].[ModifiedById]
    ,[SysAdminUnit].[Name]
    ,[SysAdminUnit].[Description]
    ,[SysAdminUnit].[ParentRoleId]
    ,[SysAdminUnit].[ContactId]
    ,[SysAdminUnit].[IsDirectoryEntry]
    ,[TimeZone].[Id] AS [TimeZoneId]
    ,[SysAdminUnit].[UserPassword]
    ,[SysAdminUnitType].[Id] AS [SysAdminUnitTypeId]
    ,[SysAdminUnit].[AccountId]
    ,[SysAdminUnit].[Active]
    ,[SysAdminUnit].[LoggedIn]
    ,[SysAdminUnit].[SynchronizeWithLDAP]
    ,[SysAdminUnit].[LDAPEntry]
    ,[SysAdminUnit].[LDAPEntryId]
    ,[SysAdminUnit].[LDAPEntryDN]
    ,[SysAdminUnit].[SysCultureId]
    ,[SysAdminUnit].[ProcessListeners]
    ,[SysAdminUnit].[PasswordExpireDate]
    ,[SysAdminUnit].[HomePageId]
    ,[SysAdminUnit].[ConnectionType]
    ,[ConnectionType].[Id] AS [UserConnectionTypeId]
    ,[SysAdminUnit].[ForceChangePassword]
    ,[SysAdminUnit].[DateTimeFormatId]
    ,[SysAdminUnit].[Id] as [SysAdminUnitId]
    ,[SysAdminUnit].[SessionTimeout] as [SessionTimeout]
FROM [SysAdminUnit]
INNER JOIN [SysAdminUnitType] ON [SysAdminUnitType].[Value] = [SysAdminUnit].[SysAdminUnitTypeVa
LEFT JOIN [ConnectionType] AS [ConnectionType] ON [ConnectionType].[Value] = [SysAdminUnit].[Cor
LEFT JOIN [TimeZone] AS [TimeZone] ON [TimeZone].[Code] = [SysAdminUnit].[TimeZoneId]
GO
CREATE TRIGGER [dbo].[ITR_VwSysAdminUnit_I]
ON [dbo].[VwSysAdminUnit]
    INSTEAD OF INSERT
```

```
AS
BEGIN
SET NOCOUNT ON;
INSERT INTO [SysAdminUnit](
    [Id]
    ,[CreatedOn]
    ,[CreatedById]
    ,[ModifiedOn]
    ,[ModifiedById]
    ,[Name]
    ,[Description]
    ,[ParentRoleId]
    ,[ContactId]
    ,[IsDirectoryEntry]
    ,[TimeZoneId]
    ,[UserPassword]
    ,[SysAdminUnitTypeValue]
    ,[AccountId]
    ,[Active]
    ,[LoggedIn]
    ,[SynchronizeWithLDAP]
    ,[LDAPEntry]
    ,[LDAPEntryId]
    ,[LDAPEntryDN]
    ,[SysCultureId]
    ,[ProcessListeners]
    ,[PasswordExpireDate]
    ,[HomePageId]
    ,[ConnectionType]
    ,[ForceChangePassword]
    ,[DateTimeFormatId]
    ,[SessionTimeout])
SELECT [Id]
    ,[CreatedOn]
    ,[CreatedById]
    ,[ModifiedOn]
    ,[ModifiedById]
    ,[Name]
    ,[Description]
    ,[ParentRoleId]
    ,[ContactId]
    ,[IsDirectoryEntry]
    ,(SELECT COALESCE(
        (SELECT [TimeZone].[Code] FROM [TimeZone]
            WHERE [TimeZone].[Id] = [INSERTED].[TimeZoneId]), ''))
    ,[UserPassword]
    ,ISNULL((SELECT [SysAdminUnitType].[Value] FROM [SysAdminUnitType]
        WHERE [SysAdminUnitType].[Id] = [INSERTED].[SysAdminUnitTypeId]), 4)
    ,[AccountId]
```

```
    ,[Active]
    ,ISNULL([LoggedIn], 0)
    ,[SynchronizeWithLDAP]
    ,[LDAPEntry]
    ,[LDAPEntryId]
    ,[LDAPEntryDN]
    ,[SysCultureId]
    ,[ProcessListeners]
    ,[PasswordExpireDate]
    ,[HomePageId]
    ,COALESCE([INSERTED].[ConnectionType],
        (SELECT [ConnectionType].[Value] FROM [ConnectionType]
        WHERE [ConnectionType].[Id] = [INSERTED].[UserConnectionTypeId]), 0)
    ,ISNULL([ForceChangePassword], 0)
    ,[DateTimeFormatId]
    ,[SessionTimeout]
FROM [INSERTED]
END
GO
CREATE TRIGGER [dbo].[ITR_VwSysAdminUnit_U]
ON [dbo].[VwSysAdminUnit]
    INSTEAD OF UPDATE
AS
BEGIN
SET NOCOUNT ON;
UPDATE [SysAdminUnit]
SET [SysAdminUnit].[CreatedOn] = [INSERTED].[CreatedOn]
    ,[SysAdminUnit].[CreatedById] = [INSERTED].[CreatedById]
    ,[SysAdminUnit].[ModifiedOn] =[INSERTED].[ModifiedOn]
    ,[SysAdminUnit].[ModifiedById] = [INSERTED].[ModifiedById]
    ,[SysAdminUnit].[Name] = [INSERTED].[Name]
    ,[SysAdminUnit].[Description] = [INSERTED].[Description]
    ,[SysAdminUnit].[ParentRoleId] = [INSERTED].[ParentRoleId]
    ,[SysAdminUnit].[ContactId] = [INSERTED].[ContactId]
    ,[SysAdminUnit].[IsDirectoryEntry] = [INSERTED].[IsDirectoryEntry]
    ,[SysAdminUnit].[TimeZoneId] =
        (SELECT COALESCE(
            (SELECT [TimeZone].[Code] FROM [TimeZone]
                WHERE [TimeZone].[Id] = [INSERTED].[TimeZoneId]), ''))
    ,[SysAdminUnit].[UserPassword] = [INSERTED].[UserPassword]
    ,[SysAdminUnit].[SysAdminUnitTypeValue] =
        (SELECT [SysAdminUnitType].[Value] FROM [SysAdminUnitType]
            WHERE [SysAdminUnitType].[Id] = [INSERTED].[SysAdminUnitTypeId])
    ,[SysAdminUnit].[AccountId] = [INSERTED].[AccountId]
    ,[SysAdminUnit].[Active] = [INSERTED].[Active]
    ,[SysAdminUnit].[LoggedIn] = [INSERTED].[LoggedIn]
    ,[SysAdminUnit].[SynchronizeWithLDAP] = [INSERTED].[SynchronizeWithLDAP]
    ,[SysAdminUnit].[LDAPEntry] = [INSERTED].[LDAPEntry]
```

```
    ,[SysAdminUnit].[LDAPEntryId] = [INSERTED].[LDAPEntryId]
    ,[SysAdminUnit].[LDAPEntryDN] = [INSERTED].[LDAPEntryDN]
    ,[SysAdminUnit].[SysCultureId] = [INSERTED].[SysCultureId]
    ,[SysAdminUnit].[ProcessListeners] = [INSERTED].[ProcessListeners]
    ,[SysAdminUnit].[PasswordExpireDate] = [INSERTED].[PasswordExpireDate]
    ,[SysAdminUnit].[HomePageId] = [INSERTED].[HomePageId]
    ,[SysAdminUnit].[ConnectionType] = COALESCE([INSERTED].[ConnectionType],
        (SELECT [ConnectionType].[Value] FROM [ConnectionType]
        WHERE [ConnectionType].[Id] = [INSERTED].[UserConnectionTypeId]), 0)
    ,[SysAdminUnit].[ForceChangePassword] = [INSERTED].[ForceChangePassword]
    ,[SysAdminUnit].[DateTimeFormatId] = [INSERTED].[DateTimeFormatId]
    ,[SysAdminUnit].[SessionTimeout] = [INSERTED].[SessionTimeout]
FROM [SysAdminUnit]
INNER JOIN [INSERTED] ON [SysAdminUnit].[Id] = [INSERTED].[Id]
END
GO
CREATE TRIGGER [dbo].[ITR_VwSysAdminUnit_D]
ON [dbo].[VwSysAdminUnit]
    INSTEAD OF DELETE
AS
BEGIN
SET NOCOUNT ON;
DELETE FROM [SysAdminUnit]
WHERE EXISTS(SELECT * FROM [DELETED] WHERE [SysAdminUnit].[Id] = [DELETED].[Id])
END
GO
```

PostgreSQL

```
-- View and triggers that let you modify the target table
-- PostgreSQL
DROP FUNCTION IF EXISTS "public"."ITR_VwSysLookup_IUD_Func" CASCADE;
DROP VIEW IF EXISTS "public"."VwSysLookup";

CREATE VIEW "public"."VwSysLookup" AS
SELECT "SysLookup"."Id"
    ,"SysLookup"."CreatedOn"
    ,"SysLookup"."CreatedById"
    ,"SysLookup"."ModifiedOn"
    ,"SysLookup"."ModifiedById"
    ,"SysLookup"."Name"
    ,"SysLookup"."Description"
    ,"SysLookup"."SysFolderId"
    ,"SysLookup"."SysEntitySchemaUId"
    ,"SysLookup"."SysGridPageSchemaUId"
    ,"SysLookup"."SysEditPageSchemaUId"
    ,"VwSysSchemaInfo"."SysWorkspaceId"
```

```
    ,"SysLookup"."ProcessListeners"
    ,"SysLookup"."IsSystem"
    ,"SysLookup"."IsSimple"
FROM "public"."SysLookup"
INNER JOIN "public"."VwSysSchemaInfo" ON "SysLookup"."SysEntitySchemaUId" = "VwSysSchemaInfo"."U

CREATE FUNCTION "public"."ITR_VwSysLookup_IUD_Func"() RETURNS TRIGGER AS $$
    BEGIN
        IF TG_OP = 'INSERT' THEN
            INSERT INTO "public"."SysLookup"(
                "Id"
                ,"CreatedOn"
                ,"CreatedById"
                ,"ModifiedOn"
                ,"ModifiedById"
                ,"Name"
                ,"Description"
                ,"SysFolderId"
                ,"SysEntitySchemaUId"
                ,"SysGridPageSchemaUId"
                ,"SysEditPageSchemaUId"
                ,"ProcessListeners"
                ,"IsSystem"
                ,"IsSimple")
            SELECT NEW."Id"
                ,NEW."CreatedOn"
                ,NEW."CreatedById"
                ,NEW."ModifiedOn"
                ,NEW."ModifiedById"
                ,NEW."Name"
                ,NEW."Description"
                ,NEW."SysFolderId"
                ,NEW."SysEntitySchemaUId"
                ,NEW."SysGridPageSchemaUId"
                ,NEW."SysEditPageSchemaUId"
                ,NEW."ProcessListeners"
                ,NEW."IsSystem"
                ,NEW."IsSimple";
            RETURN NEW;
        ELSIF TG_OP = 'UPDATE' THEN
            UPDATE "public"."SysLookup"
            SET "CreatedOn" = NEW."CreatedOn"
                ,"CreatedById" = NEW."CreatedById"
                ,"ModifiedOn" = NEW."ModifiedOn"
                ,"ModifiedById" = NEW."ModifiedById"
                ,"Name" = NEW."Name"
                ,"Description" = NEW."Description"
                ,"SysFolderId" = NEW."SysFolderId"
```

```
                ,"SysEntitySchemaUId" = NEW."SysEntitySchemaUId"
                ,"SysGridPageSchemaUId" = NEW."SysGridPageSchemaUId"
                ,"SysEditPageSchemaUId" = NEW."SysEditPageSchemaUId"
                ,"ProcessListeners" = NEW."ProcessListeners"
                ,"IsSystem" = NEW."IsSystem"
                ,"IsSimple" = NEW."IsSimple"
            WHERE "SysLookup"."Id" = NEW."Id";
            RETURN NEW;
        ELSIF TG_OP = 'DELETE' THEN
            DELETE FROM "public"."SysLookup" WHERE OLD."Id" = "SysLookup"."Id";
            RETURN OLD;
        END IF;
        RETURN NEW;
    END;
$$ LANGUAGE plpgsql;
CREATE TRIGGER "ITR_VwSysLookup_IUD"
    INSTEAD OF INSERT OR UPDATE OR DELETE ON "public"."VwSysLookup"
    FOR EACH ROW EXECUTE PROCEDURE "public"."ITR_VwSysLookup_IUD_Func"();
```

Example 2 (views)

Example. Example of an SQL script that demonstrates how to use a RULE instead of the INSTEAD OF trigger in PostgreSQL.

MSSQL

```
-- Use the RULE instead of the INSTEAD OF trigger
-- MSSQL
IF EXISTS (SELECT * FROM sys.views WHERE object_id = OBJECT_ID(N'[dbo].[VwAdministrativeObjects]
DROP VIEW [dbo].[VwAdministrativeObjects]
GO
CREATE VIEW [dbo].[VwAdministrativeObjects]
AS
WITH
[SysSchemaAdministrationProperties] AS (
SELECT [AdministrationPropertiesAll].[Id] AS [SysSchemaId],
    max([AdministrationPropertiesAll].[AdministratedByOperations]) AS [AdministratedByOperations
    max([AdministrationPropertiesAll].[AdministratedByColumns]) AS [AdministratedByColumns],
    max([AdministrationPropertiesAll].[AdministratedByRecords]) AS [AdministratedByRecords],
    max([AdministrationPropertiesAll].[IsTrackChangesInDB]) AS [IsTrackChangesInDB]
FROM (
    SELECT [SysSchema].[Id],
            (CASE WHEN EXISTS (
```

```
                SELECT 1
                FROM [SysSchemaProperty]
                WHERE (([SysSchemaProperty].[SysSchemaId] = [SysSchema].[Id] AND [SysSchema].[Ex
                    OR [SysSchemaProperty].[SysSchemaId] = [DerivedSysSchema].[Id])
                    AND [SysSchemaProperty].[Name] = 'AdministratedByOperations'
                    AND [SysSchemaProperty].[Value] = 'True'
                    AND [SysSchemaProperty].[SysSchemaId] IS NOT NULL
                )
            THEN 1 ELSE 0 END) AS [AdministratedByOperations],
            (CASE WHEN EXISTS (
                SELECT 1
                FROM [SysSchemaProperty]
                WHERE (([SysSchemaProperty].[SysSchemaId] = [SysSchema].[Id] AND [SysSchema].[Ex
                    OR [SysSchemaProperty].[SysSchemaId] = [DerivedSysSchema].[Id])
                    AND [SysSchemaProperty].[Name] = 'AdministratedByColumns'
                    AND [SysSchemaProperty].[Value] = 'True'
                    AND [SysSchemaProperty].[SysSchemaId] IS NOT NULL
                )
            THEN 1 ELSE 0 END) AS [AdministratedByColumns],
            (CASE WHEN EXISTS (
                SELECT 1
                FROM [SysSchemaProperty]
                WHERE (([SysSchemaProperty].[SysSchemaId] = [SysSchema].[Id] AND [SysSchema].[Ex
                    OR [SysSchemaProperty].[SysSchemaId] = [DerivedSysSchema].[Id])
                    AND [SysSchemaProperty].[Name] = 'AdministratedByRecords'
                    AND [SysSchemaProperty].[Value] = 'True'
                    AND [SysSchemaProperty].[SysSchemaId] IS NOT NULL
                )
            THEN 1 ELSE 0 END) AS [AdministratedByRecords],
            (CASE WHEN EXISTS (
                SELECT 1
                FROM [SysSchemaProperty]
                WHERE (([SysSchemaProperty].[SysSchemaId] = [SysSchema].[Id] AND [SysSchema].[Ex
                    OR [SysSchemaProperty].[SysSchemaId] = [DerivedSysSchema].[Id])
                    AND [SysSchemaProperty].[Name] = 'IsTrackChangesInDB'
                    AND [SysSchemaProperty].[Value] = 'True'
                    AND [SysSchemaProperty].[SysSchemaId] IS NOT NULL
                )
            THEN 1 ELSE 0 END) AS [IsTrackChangesInDB]
        FROM [SysSchema]
        LEFT OUTER JOIN [SysSchema] AS [DerivedSysSchema] ON ([SysSchema].[Id] = [DerivedSysSchema].
        WHERE [SysSchema].[ManagerName] = 'EntitySchemaManager'
            AND [SysSchema].[ExtendParent] = 0
    ) AS [AdministrationPropertiesAll]
    GROUP BY [AdministrationPropertiesAll].[Id]
    )
    SELECT [BaseSchemas].[UId] AS [Id],
        [BaseSchemas].[UId],
        [BaseSchemas].[CreatedOn],
```

```
    [BaseSchemas].[CreatedById],
    [BaseSchemas].[ModifiedOn],
    [BaseSchemas].[ModifiedById],
    [BaseSchemas].[Name],
    [VwSysSchemaExtending].[TopExtendingCaption] as Caption,
    [BaseSchemas].[Description],
    (CASE WHEN EXISTS (
        SELECT 1
        FROM [SysLookup]
        WHERE [SysLookup].[SysEntitySchemaUId] = [BaseSchemas].[UId])
        THEN 1 ELSE 0 END) AS [IsLookup],
    (CASE WHEN EXISTS (
        SELECT 1 FROM [SysModule]
        INNER JOIN [SysModuleEntity] ON [SysModuleEntity].[Id] = [SysModule].[SysModuleEntityId]
        WHERE [BaseSchemas].[UId] = [SysModuleEntity].[SysEntitySchemaUId])
        THEN 1 ELSE 0 END) AS [IsModule],
    [SysSchemaAdministrationProperties].[AdministratedByOperations],
    [SysSchemaAdministrationProperties].[AdministratedByColumns],
    [SysSchemaAdministrationProperties].[AdministratedByRecords],
    [SysSchemaAdministrationProperties].[IsTrackChangesInDB],
    [SysWorkspaceId],
    [BaseSchemas].[ProcessListeners],
    (CASE WHEN EXISTS (
        SELECT 1
        FROM [SysSSPEntitySchemaAccessList]
        WHERE [SysSSPEntitySchemaAccessList].[EntitySchemaUId] = [BaseSchemas].[UId]
        )
        THEN 1 ELSE 0 END) AS [IsInSSPEntitySchemaAccessList]
FROM  [SysSchema] as [BaseSchemas]
INNER JOIN [VwSysSchemaExtending] ON BaseSchemas.[Id] = [VwSysSchemaExtending].[BaseSchemaId]
INNER JOIN [SysPackage] on [BaseSchemas].[SysPackageId] = [SysPackage].[Id]
INNER JOIN [SysSchemaAdministrationProperties] ON [BaseSchemas].[Id] = [SysSchemaAdministrationP
GO
CREATE TRIGGER [dbo].[TRVwAdministrativeObjects_IU]
ON [dbo].[VwAdministrativeObjects]
    INSTEAD OF UPDATE
AS
BEGIN
    SET NOCOUNT ON;
    RETURN
END
GO
```

PostgreSQL

```
-- Use the RULE instead of the INSTEAD OF trigger
```

```
-- PostgreSQL
DROP VIEW IF EXISTS public."VwAdministrativeObjects";
DROP RULE IF EXISTS RU_VwAdministrativeObjects ON "VwAdministrativeObjects";

CREATE VIEW public."VwAdministrativeObjects" AS
WITH SysSchemaAdministrationProperties AS (
    SELECT AdministrationPropertiesAll.Id "SysSchemaId",
        MAX(AdministrationPropertiesAll.AdministratedByOperations) "AdministratedByOperations",
        MAX(AdministrationPropertiesAll.AdministratedByColumns) "AdministratedByColumns",
        MAX(AdministrationPropertiesAll.AdministratedByRecords) "AdministratedByRecords",
        MAX(AdministrationPropertiesAll.IsTrackChangesInDB) "IsTrackChangesInDB"
    FROM (
        SELECT ss."Id" Id
            ,(CASE WHEN EXISTS (
                SELECT 1
                FROM "SysSchemaProperty" ssp
                WHERE ((ssp."SysSchemaId" = ss."Id" AND NOT ss."ExtendParent") OR ssp."SysSchema
                    AND ssp."Name" = 'AdministratedByOperations'
                    AND ssp."Value" = 'True'
                    AND ssp."SysSchemaId" IS NOT NULL
            ) THEN 1 ELSE 0 END) AdministratedByOperations
            ,(CASE WHEN EXISTS (
                SELECT 1
                FROM "SysSchemaProperty" ssp
                WHERE ((ssp."SysSchemaId" = ss."Id" AND NOT ss."ExtendParent") OR ssp."SysSchema
                    AND ssp."Name" = 'AdministratedByColumns'
                    AND ssp."Value" = 'True'
                    AND ssp."SysSchemaId" IS NOT NULL
            ) THEN 1 ELSE 0 END) AdministratedByColumns
            ,(CASE WHEN EXISTS (
                SELECT 1
                FROM "SysSchemaProperty" ssp
                WHERE ((ssp."SysSchemaId" = ss."Id" AND NOT ss."ExtendParent") OR ssp."SysSchema
                    AND ssp."Name" = 'AdministratedByRecords'
                    AND ssp."Value" = 'True'
                    AND ssp."SysSchemaId" IS NOT NULL
            ) THEN 1 ELSE 0 END) AdministratedByRecords
            ,(CASE WHEN EXISTS (
                SELECT 1
                FROM "SysSchemaProperty" ssp WHERE ((ssp."SysSchemaId" = ss."Id" AND NOT ss."Ext
                    OR ssp."SysSchemaId" = DerivedSysSchema."Id")
                    AND ssp."Name" = 'IsTrackChangesInDB'
                    AND ssp."Value" = 'True'
                    AND ssp."SysSchemaId" IS NOT NULL
            ) THEN 1 ELSE 0 END) IsTrackChangesInDB
        FROM "SysSchema" ss
        LEFT OUTER JOIN "SysSchema" DerivedSysSchema ON (ss."Id" = DerivedSysSchema."ParentId" A
        WHERE ss."ManagerName" = 'EntitySchemaManager' AND NOT ss."ExtendParent"
    ) AdministrationPropertiesAll
```

```
    GROUP BY AdministrationPropertiesAll.Id
)
SELECT BaseSchema."UId" "Id"
    ,BaseSchema."UId"
    ,BaseSchema."CreatedOn"
    ,BaseSchema."CreatedById"
    ,BaseSchema."ModifiedOn"
    ,BaseSchema."ModifiedById"
    ,BaseSchema."Name"
    ,public."VwSysSchemaExtending"."TopExtendingCaption" "Caption"
    ,BaseSchema."Description"
    ,EXISTS (
        SELECT 1
        FROM "SysLookup"
        WHERE "SysEntitySchemaUId" = BaseSchema."UId"
    ) "IsLookup"
    ,EXISTS (
        SELECT 1
        FROM "SysModule" sm
        INNER JOIN "SysModuleEntity" sme ON sme."Id" = sm."SysModuleEntityId"
        WHERE BaseSchema."UId" = sme."SysEntitySchemaUId"
    ) "IsModule"
    ,SysSchemaAdministrationProperties."AdministratedByOperations"::BOOLEAN
    ,SysSchemaAdministrationProperties."AdministratedByColumns"::BOOLEAN
    ,SysSchemaAdministrationProperties."AdministratedByRecords"::BOOLEAN
    ,SysSchemaAdministrationProperties."IsTrackChangesInDB"::BOOLEAN
    ,"SysWorkspaceId"
    ,BaseSchema."ProcessListeners"
    ,EXISTS (
        SELECT 1
        FROM "SysSSPEntitySchemaAccessList"
        WHERE "EntitySchemaUId" = BaseSchema."UId"
    ) "IsInSSPEntitySchemaAccessList"
FROM "SysSchema" BaseSchema
INNER JOIN "VwSysSchemaExtending" ON BaseSchema."Id" = "VwSysSchemaExtending"."BaseSchemaId"
INNER JOIN "SysPackage" on BaseSchema."SysPackageId" = "SysPackage"."Id"
INNER JOIN SysSchemaAdministrationProperties ON BaseSchema."Id" = SysSchemaAdministrationPropert

CREATE RULE RU_VwAdministrativeObjects AS
    ON UPDATE TO "VwAdministrativeObjects"
    DO INSTEAD NOTHING;
```

Example 3 (stored procedures)

Example. Example of an SQL script that creates a stored procedure. The stored procedure uses loops,

cursors, and temporary tables.

MSSQL

```
-- Stored procedure that uses loops, cursors, and temporary tables
-- MSSQL
IF NOT OBJECT_ID('[dbo].[tsp_ActualizeUserRoles]') IS NULL
BEGIN
    DROP PROCEDURE [dbo].[tsp_ActualizeUserRoles]
END
GO

CREATE PROCEDURE dbo.tsp_ActualizeUserRoles (@UserId uniqueidentifier)
AS
BEGIN
    SET NOCOUNT ON

    IF OBJECT_ID('tempdb..#AdminUnitListTemp') IS NOT NULL
    BEGIN
        DROP TABLE [#AdminUnitListTemp];
    END;
    CREATE TABLE [#AdminUnitListTemp] (
        [UserId] uniqueidentifier NOT NULL,
        [Id] uniqueidentifier NOT NULL,
        [Name] NVARCHAR(250) NOT NULL,
        [ParentRoleId] uniqueidentifier NULL,
        [Granted] BIT NULL
    );

    DECLARE @GetAdminUnitList TABLE (
        [Id] uniqueidentifier NOT NULL,
        [Name] nvarchar(260) NOT NULL,
        [ParentRoleId] uniqueidentifier NULL
    );
    DECLARE @NewRoles TABLE ([Id] uniqueidentifier NOT NULL);
    DECLARE @OldUserRoles TABLE ([Id] uniqueidentifier NOT NULL);

    DECLARE @getUserAdminUnits CURSOR;
    DECLARE @SysAdminUnitRoles TABLE (
        [Id] uniqueidentifier,
        [Name] nvarchar(260),
        [ParentRoleId] uniqueidentifier
    );
    DECLARE @ManagersBeforeActualization TABLE ([Id] uniqueidentifier NOT NULL);
    DECLARE @ManagersAfterActualization TABLE ([Id] uniqueidentifier NOT NULL);
    DECLARE @StillManagers TABLE ([Id] uniqueidentifier NOT NULL);
    DECLARE @NoLongerManagers TABLE ([Id] uniqueidentifier NOT NULL);
```

```
DECLARE @NewManagers TABLE ([Id] uniqueidentifier NOT NULL);
DECLARE @SysAdminUnitId uniqueidentifier;

-- Old user roles
INSERT INTO @OldUserRoles
    SELECT DISTINCT [SysAdminUnitInRole].[SysAdminUnitRoleId] [Id]
    FROM [SysAdminUnitInRole]
    WHERE [SysAdminUnitInRole].[SysAdminUnitId] = @UserId

-- Old user managers
INSERT INTO @ManagersBeforeActualization
    SELECT DISTINCT [SysUserInRole].[SysUserId] [Id]
    FROM [SysAdminUnitInRole]
    INNER JOIN [SysAdminUnit] [Roles]
        ON [SysAdminUnitInRole].[SysAdminUnitRoleId] = [Roles].[Id]
    INNER JOIN @OldUserRoles
        ON [Roles].[ParentRoleId] = [@OldUserRoles].[Id]
    INNER JOIN [SysUserInRole]
        ON [SysUserInRole].[SysRoleId] = [Roles].[Id]
    WHERE [Roles].[SysAdminUnitTypeValue] = 2

-- Get and insert new user roles
INSERT INTO @GetAdminUnitList EXEC [tsp_GetAdminUnitList] @UserId=@UserId;
INSERT INTO @NewRoles SELECT [Id] FROM @GetAdminUnitList;
DELETE FROM [SysAdminUnitInRole] WHERE [SysAdminUnitId] = @UserId;
INSERT INTO [SysAdminUnitInRole] ([SysAdminUnitId], [SysAdminUnitRoleId])
    SELECT DISTINCT @UserId, [Id] FROM @NewRoles;

-- User managers after actualization
INSERT INTO @ManagersAfterActualization
    SELECT DISTINCT
        [SysUserInRole].[SysUserId] [Id]
    FROM [SysAdminUnitInRole]
    INNER JOIN [SysAdminUnit] [Roles]
        ON [SysAdminUnitInRole].[SysAdminUnitRoleId] = [Roles].[Id]
    INNER JOIN @NewRoles NewRoles
        ON [Roles].[ParentRoleId] = NewRoles.[Id]
    INNER JOIN [SysUserInRole]
        ON [SysUserInRole].[SysRoleId] = [Roles].[Id]
    WHERE [Roles].[SysAdminUnitTypeValue] = 2;

-- New (who were not but become) user managers
INSERT INTO @NewManagers
    SELECT [Id] FROM @ManagersAfterActualization AS managersAfterActualization
        WHERE NOT EXISTS (
            SELECT NULL
            FROM @ManagersBeforeActualization AS managersBeforeActualization
            WHERE managersBeforeActualization.[Id] = managersAfterActualization.[Id]
        );
```

```
-- Add all user roles to new managers and their grantee-users, if they arent already have
SET @getUserAdminUnits = CURSOR FOR
    SELECT DISTINCT [Id] FROM (
        SELECT [Id] FROM @NewManagers
        UNION
        SELECT [GranteeSysAdminUnitId]
        FROM [SysAdminUnitGrantedRight]
        WHERE EXISTS (
            SELECT NULL FROM @NewManagers as newManagers
            WHERE [SysAdminUnitGrantedRight].[GrantorSysAdminUnitId] = newManagers.[Id]
        )
    ) Roles;

OPEN @getUserAdminUnits;
FETCH NEXT
FROM @getUserAdminUnits INTO @SysAdminUnitId;
WHILE @@FETCH_STATUS = 0
BEGIN
    INSERT INTO [SysAdminUnitInRole] ([SysAdminUnitId], [SysAdminUnitRoleId])
        SELECT DISTINCT @SysAdminUnitId, [Id]
        FROM @NewRoles AS newRoles
        WHERE NOT EXISTS (
            SELECT 1
            FROM [SysAdminUnitInRole]
            WHERE [SysAdminUnitInRole].[SysAdminUnitId] = @SysAdminUnitId
            AND [SysAdminUnitInRole].[SysAdminUnitRoleId] = newRoles.[Id]
        );
    FETCH NEXT FROM @getUserAdminUnits INTO @SysAdminUnitId;
END;
CLOSE @getUserAdminUnits;
DEALLOCATE @getUserAdminUnits;

DECLARE @isUserLostAtLeastOneRole INT = (
    SELECT COUNT(*)
    FROM @OldUserRoles AS oldUserRoles
    WHERE NOT EXISTS (
        SELECT 1
        FROM @NewRoles AS newUserRoles
        WHERE newUserRoles.[Id] = oldUserRoles.[Id]
    )
);

-- Still (who were and remained) user managers
INSERT INTO @StillManagers
    SELECT  DISTINCT managersAfterActualization.[Id] AS [Id]
    FROM @ManagersAfterActualization AS managersAfterActualization
        JOIN @ManagersBeforeActualization AS managersBeforeActualization
```

```
                ON managersAfterActualization.[Id] = managersBeforeActualization.[Id];

    -- If user lost at least one role, we need to actualize all his still-managers.
    -- If not (user only gained new roles) - we just add to still-managers and their grantee-use
    IF (@isUserLostAtLeastOneRole = 0)
    BEGIN
        -- Add all new user roles to his still-managers and to their grantee-users
        SET @getUserAdminUnits = CURSOR FOR
            SELECT DISTINCT [Id] FROM (
                SELECT stillManagers.[Id] AS [Id]
                FROM @StillManagers AS stillManagers
                UNION
                SELECT [GranteeSysAdminUnitId]
                FROM [SysAdminUnitGrantedRight]
                WHERE EXISTS (
                    SELECT NULL
                    FROM @StillManagers AS stillManagers
                    WHERE stillManagers.[Id] = [GrantorSysAdminUnitId]
                )
            ) Roles;

        OPEN @getUserAdminUnits;
        FETCH NEXT
        FROM @getUserAdminUnits INTO @SysAdminUnitId;
        WHILE @@FETCH_STATUS = 0
        BEGIN
            INSERT INTO [SysAdminUnitInRole] ([SysAdminUnitId], [SysAdminUnitRoleId])
                SELECT DISTINCT @SysAdminUnitId, [Id]
                FROM @NewRoles AS newRoles
                WHERE NOT EXISTS (
                    SELECT 1
                    FROM [SysAdminUnitInRole]
                    WHERE [SysAdminUnitInRole].[SysAdminUnitId] = @SysAdminUnitId
                        AND [SysAdminUnitInRole].[SysAdminUnitRoleId] = newRoles.[Id]
                );
            FETCH NEXT FROM @getUserAdminUnits INTO @SysAdminUnitId;
        END;
        CLOSE @getUserAdminUnits;
        DEALLOCATE @getUserAdminUnits;
    END ELSE
    BEGIN
        --Actualize all roles for still-managers
        SET @getUserAdminUnits = CURSOR FOR
            SELECT DISTINCT [Id]
            FROM @StillManagers
            UNION
                SELECT [GranteeSysAdminUnitId]
                FROM [SysAdminUnitGrantedRight]
                WHERE EXISTS (
```

```
                SELECT NULL
                FROM @StillManagers AS stillManagers
                WHERE stillManagers.[Id] = [GrantorSysAdminUnitId]
                );

        OPEN @getUserAdminUnits;
        FETCH NEXT
        FROM @getUserAdminUnits INTO @SysAdminUnitId;
        WHILE @@FETCH_STATUS = 0
        BEGIN
            DELETE FROM @SysAdminUnitRoles;
            INSERT INTO @SysAdminUnitRoles
                EXEC [tsp_GetAdminUnitList] @UserId=@SysAdminUnitId;
            BEGIN TRAN;
                DELETE FROM [dbo].[SysAdminUnitInRole] WHERE SysAdminUnitId = @SysAdminUnitId;
                INSERT INTO [dbo].[SysAdminUnitInRole] (SysAdminUnitId, SysAdminUnitRoleId)
                    SELECT @SysAdminUnitId, [Id] FROM @SysAdminUnitRoles;
            COMMIT;
            FETCH NEXT
                FROM @getUserAdminUnits INTO @SysAdminUnitId;
        END;
        CLOSE @getUserAdminUnits;
        DEALLOCATE @getUserAdminUnits;
    END;

    -- No longer (who were but not remained) user managers
    INSERT INTO @NoLongerManagers
        SELECT [Id] FROM @ManagersBeforeActualization as managersBeforeActualization
            WHERE NOT EXISTS (
                SELECT NULL
                FROM @ManagersAfterActualization AS managersAfterActualization
                WHERE managersAfterActualization.[Id] = managersBeforeActualization.[Id]
            );

    -- Actualize roles for all noLonger-managers, his grantee-users and all grantee-users of use
    SET @getUserAdminUnits = CURSOR FOR
        SELECT DISTINCT [Id] FROM (
            SELECT [Id] FROM @NoLongerManagers
            UNION
            SELECT [GranteeSysAdminUnitId]
            FROM [SysAdminUnitGrantedRight]
            WHERE EXISTS (
                SELECT NULL
                FROM @NoLongerManagers AS noLongerManagers
                WHERE noLongerManagers.[Id] = [GrantorSysAdminUnitId]
            )
            UNION ALL
            SELECT GranteeSysAdminUnitId
```

```
            FROM SysAdminUnitGrantedRight
            WHERE GrantorSysAdminUnitId = @UserId
        ) Roles;

    OPEN @getUserAdminUnits;
    FETCH NEXT
        FROM @getUserAdminUnits INTO @SysAdminUnitId;
    WHILE @@FETCH_STATUS = 0
    BEGIN
        DELETE FROM @SysAdminUnitRoles;
        INSERT INTO @SysAdminUnitRoles
            EXEC [tsp_GetAdminUnitList] @UserId=@SysAdminUnitId;
        BEGIN TRAN;
            DELETE FROM [dbo].[SysAdminUnitInRole] WHERE SysAdminUnitId = @SysAdminUnitId;
            INSERT INTO [dbo].[SysAdminUnitInRole] (SysAdminUnitId, SysAdminUnitRoleId)
                SELECT @SysAdminUnitId, [Id] FROM @SysAdminUnitRoles;
        COMMIT;
        FETCH NEXT
            FROM @getUserAdminUnits INTO @SysAdminUnitId;
    END;
    CLOSE @getUserAdminUnits;
    DEALLOCATE @getUserAdminUnits;

    IF OBJECT_ID('tempdb..#AdminUnitListTemp') IS NOT NULL
    BEGIN
        DROP TABLE [#AdminUnitListTemp];
    END;
END;
GO
```

PostgreSQL

```
-- Stored procedure that uses loops, cursors, and temporary tables
-- PostgreSQL
DROP FUNCTION IF EXISTS "tsp_ActualizeUserRoles";
CREATE FUNCTION "tsp_ActualizeUserRoles"(
    UserId UUID
)
RETURNS VOID
AS $$
DECLARE
    getUserNewManagers CURSOR FOR
        SELECT DISTINCT "Id" FROM (
            SELECT "Id" FROM "NewManagers"
            UNION
            SELECT "GranteeSysAdminUnitId"
            FROM "SysAdminUnitGrantedRight"
```

```
            WHERE EXISTS (
                SELECT NULL FROM "NewManagers" as "newManagers"
                WHERE "SysAdminUnitGrantedRight"."GrantorSysAdminUnitId" = "newManagers"."Id"
            )
        ) "Roles";
    lostUserRolesCount INT;
    getUserStillManagers CURSOR FOR
        SELECT DISTINCT "stillManagers"."Id" AS "Id"
        FROM "StillManagers" AS "stillManagers"
        UNION
        SELECT "GranteeSysAdminUnitId"
        FROM "SysAdminUnitGrantedRight"
        WHERE EXISTS (
            SELECT NULL
            FROM "StillManagers" AS "stillManagers"
            WHERE "stillManagers"."Id" = "GrantorSysAdminUnitId"
        );
    getUserNoLongerManagers CURSOR FOR
        SELECT DISTINCT "Id" FROM (
            SELECT "Id"
            FROM "NoLongerManagers"
            UNION
            SELECT "GranteeSysAdminUnitId"
            FROM "SysAdminUnitGrantedRight"
            WHERE EXISTS (
                SELECT NULL
                FROM "NoLongerManagers" AS "noLongerManagers"
                WHERE "noLongerManagers"."Id" = "GrantorSysAdminUnitId"
            )
            UNION ALL
            SELECT "GranteeSysAdminUnitId"
            FROM "SysAdminUnitGrantedRight"
            WHERE "GrantorSysAdminUnitId" = UserId
        ) "Roles";
BEGIN
    DROP TABLE IF EXISTS "GetAdminUnitListTmp";
    CREATE TEMP TABLE "GetAdminUnitListTmp" (
        "Id" UUID,
        "Name" VARCHAR(250),
        "ParentRoleId" UUID
    );

    DROP TABLE IF EXISTS "SysAdminUnitRoles";
    CREATE TEMP TABLE "SysAdminUnitRoles" (
        "Id" UUID,
        "Name" VARCHAR(250),
        "ParentRoleId" UUID
    );
```

```
-- Old user roles
DROP TABLE IF EXISTS "OldUserRoles";
CREATE TEMP TABLE "OldUserRoles" (
    "Id" UUID
);
INSERT INTO "OldUserRoles"
    SELECT DISTINCT "SysAdminUnitInRole"."SysAdminUnitRoleId" "Id"
    FROM "SysAdminUnitInRole"
    WHERE "SysAdminUnitInRole"."SysAdminUnitId" = UserId;

-- Old user managers
DROP TABLE IF EXISTS "ManagersBeforeActualization";
CREATE TEMP TABLE "ManagersBeforeActualization" (
    "Id" UUID
);
INSERT INTO "ManagersBeforeActualization"
    SELECT DISTINCT "SysUserInRole"."SysUserId" "Id"
    FROM "SysAdminUnitInRole"
    INNER JOIN "SysAdminUnit" "Roles"
        ON "SysAdminUnitInRole"."SysAdminUnitRoleId" = "Roles"."Id"
    INNER JOIN "OldUserRoles"
        ON "Roles"."ParentRoleId" = "OldUserRoles"."Id"
    INNER JOIN "SysUserInRole"
        ON "SysUserInRole"."SysRoleId" = "Roles"."Id"
    WHERE "Roles"."SysAdminUnitTypeValue" = 2;

-- Get and insert new user roles
DROP TABLE IF EXISTS "GetAdminUnitList";
CREATE TEMP TABLE "GetAdminUnitList" (
    "Id" UUID,
    "Name" VARCHAR(250),
    "ParentRoleId" UUID
);
DROP TABLE IF EXISTS "NewRoles";
CREATE TEMP TABLE "NewRoles" (
    "Id" UUID
);
INSERT INTO "GetAdminUnitList" SELECT * FROM "tsp_GetAdminUnitList"(UserId);
INSERT INTO "NewRoles" SELECT "Id" FROM "GetAdminUnitList";
DELETE FROM "SysAdminUnitInRole" WHERE "SysAdminUnitId" = UserId;
INSERT INTO "SysAdminUnitInRole" ("SysAdminUnitId", "SysAdminUnitRoleId")
    SELECT DISTINCT UserId, "Id" FROM "NewRoles";

-- User managers after actualization
DROP TABLE IF EXISTS "ManagersAfterActualization";
CREATE TEMP TABLE "ManagersAfterActualization" (
    "Id" UUID
);
```

```
INSERT INTO "ManagersAfterActualization"
    SELECT DISTINCT
        "SysUserInRole"."SysUserId" "Id"
    FROM "SysAdminUnitInRole"
    INNER JOIN "SysAdminUnit" "Roles"
        ON "SysAdminUnitInRole"."SysAdminUnitRoleId" = "Roles"."Id"
    INNER JOIN "NewRoles" "NewRoles"
        ON "Roles"."ParentRoleId" = "NewRoles"."Id"
    INNER JOIN "SysUserInRole"
        ON "SysUserInRole"."SysRoleId" = "Roles"."Id"
    WHERE "Roles"."SysAdminUnitTypeValue" = 2;

-- New (who were not but become) user managers
DROP TABLE IF EXISTS "NewManagers";
CREATE TEMP TABLE "NewManagers" (
    "Id" UUID
);
INSERT INTO "NewManagers"
    SELECT "Id" FROM "ManagersAfterActualization" AS "managersAfterActualization"
        WHERE NOT EXISTS (
            SELECT NULL
            FROM "ManagersBeforeActualization" AS "managersBeforeActualization"
            WHERE "managersBeforeActualization"."Id" = "managersAfterActualization"."Id"
        );

-- Add all user roles to new managers and their grantee-users, if they arent already have
FOR UserNewManager IN getUserNewManagers LOOP
    EXIT WHEN UserNewManager = NULL;
    INSERT INTO "SysAdminUnitInRole" ("SysAdminUnitId", "SysAdminUnitRoleId")
        SELECT DISTINCT UserNewManager."Id", "Id"
        FROM "NewRoles" AS "newRoles"
        WHERE NOT EXISTS (
            SELECT 1
            FROM "SysAdminUnitInRole"
            WHERE "SysAdminUnitInRole"."SysAdminUnitId" = UserNewManager."Id"
            AND "SysAdminUnitInRole"."SysAdminUnitRoleId" = "newRoles"."Id"
        );
END LOOP;

SELECT COUNT(*) INTO lostUserRolesCount
FROM "OldUserRoles" AS "oldUserRoles"
WHERE NOT EXISTS (
    SELECT 1
    FROM "NewRoles" AS "newUserRoles"
    WHERE "newUserRoles"."Id" = "oldUserRoles"."Id"
);

-- Still (who were and remained) user managers
```

```
DROP TABLE IF EXISTS "StillManagers";
CREATE TEMP TABLE "StillManagers" (
    "Id" UUID
);
INSERT INTO "StillManagers"
    SELECT DISTINCT "managersAfterActualization"."Id" AS "Id"
    FROM "ManagersAfterActualization" AS "managersAfterActualization"
        JOIN "ManagersBeforeActualization" AS "managersBeforeActualization"
            ON "managersAfterActualization"."Id" = "managersBeforeActualization"."Id";

-- If user lost at least one role, we need to actualize all his still-managers.
-- If not (user only gained new roles) - we just add to still-managers and their grantee-use
IF lostUserRolesCount = 0 THEN

    -- Add all new user roles to his still-managers and to their grantee-users
    FOR UserStillManager IN getUserStillManagers LOOP
        EXIT WHEN UserStillManager = NULL;
        INSERT INTO "SysAdminUnitInRole" ("SysAdminUnitId", "SysAdminUnitRoleId")
            SELECT DISTINCT UserStillManager."Id", "Id"
            FROM "NewRoles" AS "newRoles"
            WHERE NOT EXISTS (
                SELECT 1
                FROM "SysAdminUnitInRole"
                WHERE "SysAdminUnitInRole"."SysAdminUnitId" = UserStillManager."Id"
                    AND "SysAdminUnitInRole"."SysAdminUnitRoleId" = "newRoles"."Id"
            );
    END LOOP;
ELSE

    --Actualize all roles for still-managers
    FOR UserStillManager IN getUserStillManagers LOOP
        EXIT WHEN UserStillManager = NULL;
        DELETE FROM "SysAdminUnitRoles";
        INSERT INTO "SysAdminUnitRoles"
            SELECT * FROM "tsp_GetAdminUnitList"(UserStillManager."Id");
            DELETE FROM "SysAdminUnitInRole" WHERE "SysAdminUnitId" = UserStillManager."Id";
            INSERT INTO "SysAdminUnitInRole" ("SysAdminUnitId", "SysAdminUnitRoleId")
                SELECT UserStillManager."Id", "Id" FROM "SysAdminUnitRoles";
    END LOOP;
END IF;

-- No longer (who were but not remained) user managers
DROP TABLE IF EXISTS "NoLongerManagers";
CREATE TEMP TABLE "NoLongerManagers" (
    "Id" UUID
);
INSERT INTO "NoLongerManagers"
    SELECT "Id" FROM "ManagersBeforeActualization" AS "managersBeforeActualization"
        WHERE NOT EXISTS (
```

```
                SELECT NULL
                FROM "ManagersAfterActualization" AS "managersAfterActualization"
                WHERE "managersAfterActualization"."Id" = "managersBeforeActualization"."Id"
            );


    -- Actualize roles for all noLonger-managers, his grantee-users and all grantee-users of use
    FOR UserNoLongerManager IN getUserNoLongerManagers LOOP
        EXIT WHEN UserNoLongerManager = NULL;
        DELETE FROM "SysAdminUnitRoles";
        INSERT INTO "SysAdminUnitRoles"
            SELECT * FROM "tsp_GetAdminUnitList"(UserNoLongerManager."Id");
            DELETE FROM "SysAdminUnitInRole"
                WHERE "SysAdminUnitId" = UserNoLongerManager."Id";
            INSERT INTO "SysAdminUnitInRole" ("SysAdminUnitId", "SysAdminUnitRoleId")
                SELECT UserNoLongerManager."Id", "Id" FROM "SysAdminUnitRoles";
    END LOOP;

    DROP TABLE IF EXISTS "GetAdminUnitListTmp";
END;
$$ LANGUAGE plpgsql;
```

Example 4 (stored procedures)

Example. Example of a recursive stored procedure that returns a table and uses `PERFORM`.

MSSQL

```
-- Recursive stored procedure that returns a table and uses PERFORM:
-- MSSQL
IF NOT OBJECT_ID('[dbo].[tsp_GetAdminUnitList]') IS NULL
BEGIN
    DROP PROCEDURE [dbo].[tsp_GetAdminUnitList];
END;
GO

CREATE PROCEDURE dbo.tsp_GetAdminUnitList (
    @UserId uniqueidentifier, @Granted BIT = 0
)
AS
BEGIN
    SET NOCOUNT ON;
```

```
DECLARE @StartNestedLevel INT;

IF object_id('tempdb..#AdminUnitList') IS NULL
BEGIN
    CREATE TABLE [#AdminUnitList]
    (
        [Id] uniqueidentifier NOT NULL,
        [Name] NVARCHAR(250) NULL,
        [ParentRoleId] uniqueidentifier NULL,
        [Granted] BIT NULL,
        Level INT NOT NULL
    );
    SET @StartNestedLevel = @@NESTLEVEL;
END;

DECLARE @ConnectionType INT = (SELECT [ConnectionType] FROM SysAdminUnit WHERE [Id] = @UserI

-- #AdminUnitListTemp should be created in tsp_ActualizeUserRoles or in tsp_ActualizeAdminUr
DECLARE @IsAdminUnitListTempExists BIT = OBJECT_ID('tempdb..#AdminUnitListTemp');

IF (@IsAdminUnitListTempExists IS NULL)
BEGIN
    WITH
    [MainSelect] AS (
        SELECT
            [Id] [Id],
            [Name] [Name],
            [ParentRoleId] [ParentRoleId]
        FROM
            [dbo].[SysAdminUnit]
        WHERE
            ([SysAdminUnitTypeValue] <= 4 OR [SysAdminUnitTypeValue] = 6)
        AND [ConnectionType] = @ConnectionType
        UNION ALL
        SELECT
            [Id] [Id],
            [Name] [Name],
            [ParentRoleId] [ParentRoleId]
        FROM
            [dbo].[SysAdminUnit]
        WHERE
            [Id] = @UserId),
    [ChiefUnitsSelect] AS (
        (
            SELECT
                [Chief].[ParentRoleId] [Id]
            FROM
                [dbo].[SysUserInRole] userInRole
                INNER JOIN [dbo].[SysAdminUnit] sau ON (sau.[Id] = userInRole.[SysUserId])
```

```
                    INNER JOIN [dbo].[SysAdminUnit] [Chief] ON ([Chief].[Id] = userInRole.[SysRo
                WHERE
                    sau.[Id] = @UserId AND NOT (userInRole.[SysRoleId] IS NULL) AND [Chief].[Sys
                UNION ALL
                SELECT
                    [Chief].[ParentRoleId] [Id]
                FROM
                    [dbo].[SysAdminUnit] [Chief]
                WHERE
                    [Chief].[Id] = @UserId AND [Chief].[SysAdminUnitTypeValue] = 2
            )
            UNION ALL
            SELECT
                sau.[Id]
            FROM
                [ChiefUnitsSelect]
                INNER JOIN [dbo].[SysAdminUnit] sau ON (sau.[ParentRoleId] = [ChiefUnitsSelect].
            WHERE
                sau.[SysAdminUnitTypeValue] < 4
        ),
        [HierarchicalSelect] AS (
            SELECT
                [Id],
                [Name],
                [ParentRoleId],
                0 [Level]
            FROM
                [MainSelect] [SelectStartLevel]
            WHERE
                [Id] IN (
                    SELECT
                        userInRole.[SysRoleId]
                    FROM
                        [dbo].[SysUserInRole] userInRole
                        INNER JOIN [dbo].[SysAdminUnit] sau ON (sau.[Id] = userInRole.[SysUserId
                    WHERE
                        sau.[Id] = @UserId
                    UNION ALL
                    SELECT [Id] FROM [ChiefUnitsSelect]
                    UNION ALL
                    SELECT
                        [Id]
                    FROM
                        [dbo].[SysAdminUnit]
                    WHERE
                        ([ParentRoleId] IS NULL OR [Id] = @UserId)
                        AND [SysAdminUnitTypeValue] < 4
                    UNION ALL
```

```
                    SELECT
                        [FuncRoleId]
                    FROM
                        [dbo].[SysFuncRoleInOrgRole]
                    WHERE
                        [SysFuncRoleInOrgRole].[OrgRoleId] = @UserId
                    )
            UNION ALL
            SELECT
                [SelectPriorLevel].[Id],
                [SelectPriorLevel].[Name],
                [SelectPriorLevel].[ParentRoleId],
                [Level] + 1 level
            FROM
                [MainSelect] [SelectPriorLevel]
                INNER JOIN [HierarchicalSelect] hierSelect ON (hierSelect.[ParentRoleId] = [Sele
        ),
        [FuncRoleHierarchicalSelect] AS (
            SELECT
                [Id],
                [Name],
                [ParentRoleId],
                0 [Level]
            FROM
                [MainSelect] [StartLevel]
            WHERE EXISTS (
                SELECT NULL
                FROM [dbo].[SysFuncRoleInOrgRole] funcRoleInOrgRole
                    INNER JOIN [HierarchicalSelect] hierSelect ON funcRoleInOrgRole.[OrgRoleId]
                WHERE funcRoleInOrgRole.[FuncRoleId] = [StartLevel].[Id]
            )
            UNION ALL
            SELECT
                [PriorLevel].[Id],
                [PriorLevel].[Name],
                [PriorLevel].[ParentRoleId],
                [Level] + 1 level
            FROM
                [MainSelect] [PriorLevel]
                INNER JOIN [FuncRoleHierarchicalSelect] funcRoleHierSelect ON (funcRoleHierSelec
        ),
        [DependentUserSelect] AS (
            SELECT
                mainSelect.[Id] [Id],
                mainSelect.[Name] [Name],
                mainSelect.[ParentRoleId] [ParentRoleId],
                0 [Level]
            FROM
                [MainSelect] mainSelect
```

```
            INNER JOIN [SysUserInRole] userInRole
                ON mainSelect.[Id] = userInRole.[SysUserId]
            INNER JOIN [ChiefUnitsSelect] [AllUnits]
                ON [AllUnits].[Id] = userInRole.[SysRoleId]
            WHERE
                NOT EXISTS (
                        SELECT
                            [UserUnits].[Id]
                        FROM [ChiefUnitsSelect] [UserUnits]
                        INNER JOIN [SysUserInRole] [UserInRole]
                            ON [UserUnits].[Id] = [UserInRole].[SysRoleId]
                        INNER JOIN [SysAdminUnit] sau
                            ON sau.[Id] = [UserUnits].[Id]
                        WHERE sau.[SysAdminUnitTypeValue] = 2
                            AND [UserInRole].[SysUserId] = @UserId
                            AND [UserUnits].[Id] = [AllUnits].[Id])
        )
        INSERT INTO [#AdminUnitList] ([Id], [Name], [ParentRoleId], [Granted], [Level])
        SELECT DISTINCT
            [Id],
            [Name],
            [ParentRoleId],
            @Granted,
            @@NESTLEVEL
        FROM
            (
                SELECT
                    [Id],
                    [Name],
                    [ParentRoleId]
                FROM
                    [HierarchicalSelect]
                UNION ALL
                SELECT
                    [Id],
                    [Name],
                    [ParentRoleId]
                FROM
                    [dbo].[SysAdminUnit]
                WHERE
                    [Id] = @UserId
            UNION ALL
                SELECT
                    [Id],
                    [Name],
                    [ParentRoleId]
                FROM
                    [FuncRoleHierarchicalSelect]
```

```
            UNION ALL
                SELECT
                    [Id],
                    [Name],
                    [ParentRoleId]
                FROM
            [DependentUserSelect]
        ) [AdminUnitList];
    END ELSE
    BEGIN
        DECLARE @alreadyGotRolesForThisUser bit = 0;

        IF (@IsAdminUnitListTempExists = 1)
        BEGIN
            SET @alreadyGotRolesForThisUser = (SELECT CAST( CASE WHEN EXISTS(SELECT 1 FROM [#Adm
                    WHERE [UserId] = @UserId
                    )
                    THEN 1
                    ELSE 0
                END
            AS BIT));
        END;

        IF (@alreadyGotRolesForThisUser = 1)
        BEGIN
            INSERT INTO [#AdminUnitList] ([Id], [Name], [ParentRoleId], [Granted], [Level])
                SELECT DISTINCT
                    [Id],
                    [Name],
                    [ParentRoleId],
                    @Granted,
                    @@NESTLEVEL
                FROM [#AdminUnitListTemp] WHERE UserId = @UserId;
        END ELSE
        BEGIN
            WITH
            [MainSelect] AS (
                SELECT
                    [Id] [Id],
                    [Name] [Name],
                    [ParentRoleId] [ParentRoleId]
                FROM
                    [dbo].[SysAdminUnit]
                WHERE
                    ([SysAdminUnitTypeValue] <= 4 OR [SysAdminUnitTypeValue] = 6)
                AND [ConnectionType] = @ConnectionType
                UNION ALL
                SELECT
                    [Id] [Id],
```

```
                [Name] [Name],
                [ParentRoleId] [ParentRoleId]
            FROM
                [dbo].[SysAdminUnit]
            WHERE
                [Id] = @UserId),
        [ChiefUnitsSelect] AS (
            (
                SELECT
                    [Chief].[ParentRoleId] [Id]
                FROM
                    [dbo].[SysUserInRole] sysUserInRole
                    INNER JOIN [dbo].[SysAdminUnit] sau ON (sau.[Id] = sysUserInRole.[SysUse
                    INNER JOIN [dbo].[SysAdminUnit] [Chief] ON ([Chief].[Id] = sysUserInRole
                WHERE
                    sau.[Id] = @UserId AND NOT (sysUserInRole.[SysRoleId] IS NULL) AND [Chie
                UNION ALL
                SELECT
                    [Chief].[ParentRoleId] [Id]
                FROM
                    [dbo].[SysAdminUnit] [Chief]
                WHERE
                    [Chief].[Id] = @UserId AND [Chief].[SysAdminUnitTypeValue] = 2
            )
            UNION ALL
            SELECT
                sau.[Id]
            FROM
                [ChiefUnitsSelect] ChiefUnitsSelect
                INNER JOIN [dbo].[SysAdminUnit] sau ON (sau.[ParentRoleId] = [ChiefUnitsSele
            WHERE
                sau.[SysAdminUnitTypeValue] < 4
        ),
        [HierarchicalSelect] AS (
            SELECT
                [Id],
                [Name],
                [ParentRoleId],
                0 [Level]
            FROM
                [MainSelect] [SelectStartLevel]
            WHERE EXISTS (
                SELECT NULL
                FROM (
                    SELECT [SysUserInRole].[SysRoleId] AS RoleId
                    FROM [dbo].[SysUserInRole]
                        INNER JOIN [dbo].[SysAdminUnit] ON ([SysAdminUnit].[Id] = [SysUserIr
                    WHERE [SysAdminUnit].[Id] = @UserId
```

```
                UNION ALL

                SELECT [Id] AS RoleId
                FROM [ChiefUnitsSelect]

                UNION ALL

                SELECT [Id] AS RoleId
                FROM [dbo].[SysAdminUnit]
                WHERE ([ParentRoleId] IS NULL OR [Id] = @UserId)
                    AND [SysAdminUnitTypeValue] < 4

                UNION ALL

                SELECT [FuncRoleId] AS RoleId
                FROM [dbo].[SysFuncRoleInOrgRole]
                WHERE [SysFuncRoleInOrgRole].[OrgRoleId] = @UserId
            ) AS Roles
            WHERE Roles.RoleId = [SelectStartLevel].[Id]

        )
        UNION ALL
        SELECT
            [SelectPriorLevel].[Id],
            [SelectPriorLevel].[Name],
            [SelectPriorLevel].[ParentRoleId],
            [Level] + 1 level
        FROM
            [MainSelect] [SelectPriorLevel]
            INNER JOIN [HierarchicalSelect] hierSelect ON (hierSelect.[ParentRoleId] = [
    ),
    [FuncRoleHierarchicalSelect] AS (
        SELECT
            [Id],
            [Name],
            [ParentRoleId],
            0 [Level]
        FROM
            [MainSelect] [StartLevel]
        WHERE EXISTS (
            SELECT NULL
            FROM [dbo].[SysFuncRoleInOrgRole] funcRoleInOrgRole
                INNER JOIN [HierarchicalSelect] hierSelect ON funcRoleInOrgRole.[OrgRole
            WHERE funcRoleInOrgRole.[FuncRoleId] = [StartLevel].[Id]
        )
        UNION ALL
        SELECT
            [PriorLevel].[Id],
```

```
                [PriorLevel].[Name],
                [PriorLevel].[ParentRoleId],
                [Level] + 1
            FROM
                [MainSelect] [PriorLevel]
                INNER JOIN [FuncRoleHierarchicalSelect] funcRolesHierSelect ON (funcRolesHie
        ),
        [DependentUserSelect] AS (
            SELECT
                [MainSelect].[Id] [Id],
                [MainSelect].[Name] [Name],
                [MainSelect].[ParentRoleId] [ParentRoleId],
                0 [Level]
            FROM
                [MainSelect]
            INNER JOIN [SysUserInRole] sysUserInRole
                ON [MainSelect].[Id] = sysUserInRole.[SysUserId]
            INNER JOIN [ChiefUnitsSelect] [AllUnits]
                ON [AllUnits].[Id] = sysUserInRole.[SysRoleId]
            WHERE
                NOT EXISTS (
                        SELECT
                            [UserUnits].[Id]
                        FROM [ChiefUnitsSelect] [UserUnits]
                        INNER JOIN [SysUserInRole] [UserInRole]
                            ON [UserUnits].[Id] = [UserInRole].[SysRoleId]
                        INNER JOIN [SysAdminUnit] sau
                            ON sau.[Id] = [UserUnits].[Id]
                        WHERE sau.[SysAdminUnitTypeValue] = 2
                            AND [UserInRole].[SysUserId] = @UserId
                            AND [UserUnits].[Id] = [AllUnits].[Id])
        )
        INSERT INTO #AdminUnitListTemp ([UserId], [Id], [Name], [ParentRoleId], [Granted])
        SELECT DISTINCT
            @UserId,
            [Id],
            [Name],
            [ParentRoleId],
            @Granted
        FROM
            (
                SELECT
                    [Id],
                    [Name],
                    [ParentRoleId]
                FROM
                    [HierarchicalSelect]
                UNION ALL
```

```
                    SELECT
                        [Id],
                        [Name],
                        [ParentRoleId]
                    FROM
                        [dbo].[SysAdminUnit]
                    WHERE
                        [Id] = @UserId
                UNION ALL
                    SELECT
                        [Id],
                        [Name],
                        [ParentRoleId]
                    FROM
                        [FuncRoleHierarchicalSelect]
                UNION ALL
                    SELECT
                        [Id],
                        [Name],
                        [ParentRoleId]
                    FROM
                [DependentUserSelect]
                ) [AdminUnitList];

            INSERT INTO [#AdminUnitList] ([Id], [Name], [ParentRoleId], [Granted], [Level])
            SELECT DISTINCT
                [Id],
                [Name],
                [ParentRoleId],
                @Granted,
                @@NESTLEVEL
            FROM [#AdminUnitListTemp] WHERE UserId = @UserId;
        END;
    END;

    DECLARE @DependentUserId uniqueidentifier;
    DECLARE @DependentUsersList CURSOR;
    SET @DependentUsersList = CURSOR FOR
        SELECT
            [#AdminUnitList].[Id]
        FROM
            [#AdminUnitList]
        INNER JOIN [SysAdminUnit] ON [#AdminUnitList].[Id] = [SysAdminUnit].[Id]
        WHERE
            [SysAdminUnit].[SysAdminUnitTypeValue] = 4 AND [#AdminUnitList].[Id]  @UserId
            AND [#AdminUnitList].[Granted]  1 AND [#AdminUnitList].[Level] >= @@NESTLEVEL;
    OPEN @DependentUsersList;
    FETCH NEXT
    FROM @DependentUsersList INTO @DependentUserId;
```

```
    WHILE @@FETCH_STATUS = 0
    BEGIN
        EXEC [tsp_GetAdminUnitList] @UserId=@DependentUserId, @Granted=1;
    FETCH NEXT
    FROM @DependentUsersList INTO @DependentUserId;
    END;
    CLOSE @DependentUsersList;
    DEALLOCATE @DependentUsersList;

    DECLARE @GrantorSysAdminUnitId uniqueidentifier;
    DECLARE @getGrantorSysAdminUnitList CURSOR;
    SET @getGrantorSysAdminUnitList = CURSOR FOR
        SELECT
            [GrantorSysAdminUnitId]
        FROM
            [dbo].[SysAdminUnitGrantedRight]
        WHERE
            [GranteeSysAdminUnitId] = @UserId
            AND NOT EXISTS(SELECT * FROM [#AdminUnitList] WHERE [Id] = @UserId AND [Granted] = 1
    OPEN @getGrantorSysAdminUnitList;
    FETCH NEXT
    FROM @getGrantorSysAdminUnitList INTO @GrantorSysAdminUnitId;
    WHILE @@FETCH_STATUS = 0
    BEGIN
        EXEC [tsp_GetAdminUnitList] @UserId=@GrantorSysAdminUnitId, @Granted=1;
    FETCH NEXT
    FROM @getGrantorSysAdminUnitList INTO @GrantorSysAdminUnitId;
    END;
    CLOSE @getGrantorSysAdminUnitList;
    DEALLOCATE @getGrantorSysAdminUnitList;

    IF @@NESTLEVEL = @StartNestedLevel
    BEGIN
        WITH QQ ([Id], [Name], [ParentRoleId], SysAdminUnitTypeValue) as (
            SELECT DISTINCT adminUnitList.[Id],
                adminUnitList.[Name],
                adminUnitList.[ParentRoleId],
                sau.SysAdminUnitTypeValue
            FROM [#AdminUnitList] adminUnitList
            INNER JOIN SysAdminUnit sau on sau.Id = adminUnitList.[Id]
        )
        SELECT [Id], [Name], [ParentRoleId] FROM QQ
        ORDER BY SysAdminUnitTypeValue DESC;
    END;
END;
GO
```

PostgreSQL

```
-- Recursive stored procedure that returns a table and uses PERFORM:
-- PostgreSQL
DROP FUNCTION IF EXISTS "tsp_GetAdminUnitList";
CREATE FUNCTION "tsp_GetAdminUnitList"(
    UserId UUID,
    IsGranted BOOLEAN = FALSE,
    NestLevel INT = 0
)
RETURNS TABLE (
    "Id" UUID,
    "Name" VARCHAR(250),
    "ParentRoleId" UUID
)
AS $$
DECLARE
    ConnectionType INT;
    IsAdminUnitListTempExists BOOLEAN = FALSE;
    DependentUserId UUID;
    DependentUsersList CURSOR FOR
        SELECT
            "AdminUnitList"."Id"
        FROM
            "AdminUnitList"
        INNER JOIN "SysAdminUnit" ON "AdminUnitList"."Id" = "SysAdminUnit"."Id"
        WHERE
            "SysAdminUnit"."SysAdminUnitTypeValue" = 4
            AND "AdminUnitList"."Id"  UserId
            AND "AdminUnitList"."Granted" = FALSE
            AND "AdminUnitList"."Level" >= NestLevel;
    GrantorSysAdminUnitId UUID;
    GetGrantorSysAdminUnitList CURSOR FOR
        SELECT
            "GrantorSysAdminUnitId" AS "Id"
        FROM
            "SysAdminUnitGrantedRight"
        WHERE
            "GranteeSysAdminUnitId" = UserId
            AND NOT EXISTS (
                SELECT *
                FROM "AdminUnitList"
                WHERE "AdminUnitList"."Id" = UserId
                    AND "AdminUnitList"."Granted" = TRUE
                    AND "AdminUnitList"."Level" < NestLevel
            );
    ParentRoleId UUID = NULL;
BEGIN
```

```
IF NestLevel = 0 THEN
    CREATE TEMPORARY TABLE IF NOT EXISTS "AdminUnitList" (
        "Id" UUID,
        "Name" VARCHAR(250),
        "ParentRoleId" UUID,
        "Granted" BOOLEAN,
        "Level" INT
    );
    TRUNCATE TABLE "AdminUnitList";
END IF;

SELECT "ConnectionType" INTO ConnectionType FROM "SysAdminUnit" WHERE "SysAdminUnit"."Id" =

WITH RECURSIVE "MainSelect" AS (
    SELECT
        "SysAdminUnit"."Id" "Id",
        "SysAdminUnit"."Name" "Name",
        "SysAdminUnit"."ParentRoleId" "ParentRoleId"
    FROM
        "SysAdminUnit"
    WHERE
        ("SysAdminUnitTypeValue" <= 4 OR "SysAdminUnitTypeValue" = 6)
    AND "ConnectionType" = ConnectionType
    UNION ALL
    SELECT
        "SysAdminUnit"."Id" "Id",
        "SysAdminUnit"."Name" "Name",
        "SysAdminUnit"."ParentRoleId" "ParentRoleId"
    FROM
        "SysAdminUnit"
    WHERE
        "SysAdminUnit"."Id" = UserId),
    "ChiefUnitsSelect" AS (
        SELECT
            "chief"."ParentRoleId" "Id"
        FROM
            "SysUserInRole" AS "userInRole"
            INNER JOIN "SysAdminUnit" AS "sau" ON ("sau"."Id" = "userInRole"."SysUserId")
            INNER JOIN "SysAdminUnit" AS "chief" ON ("chief"."Id" = "userInRole"."SysRoleId"
        WHERE
            "sau"."Id" = UserId
            AND "userInRole"."SysRoleId" IS NOT NULL
            AND "chief"."SysAdminUnitTypeValue" = 2
        UNION ALL
        SELECT
            "chief"."ParentRoleId" "Id"
        FROM
            "SysAdminUnit" "chief"
```

```
            WHERE
                "chief"."Id" = UserId AND "chief"."SysAdminUnitTypeValue" = 2
            UNION ALL
            SELECT
                "sau"."Id"
            FROM
                "ChiefUnitsSelect"
                INNER JOIN "SysAdminUnit" "sau" ON ("sau"."ParentRoleId" = "ChiefUnitsSelect"."I
            WHERE
                "sau"."SysAdminUnitTypeValue" < 4
        ),
        "HierarchicalSelect" AS (
            SELECT
                "SelectStartLevel"."Id",
                "SelectStartLevel"."Name",
                "SelectStartLevel"."ParentRoleId",
                0 "Level"
            FROM
                "MainSelect" "SelectStartLevel"
            WHERE
                "SelectStartLevel"."Id" IN (
                    SELECT
                        "userInRole"."SysRoleId"
                    FROM
                        "SysUserInRole" AS "userInRole"
                        INNER JOIN "SysAdminUnit" AS "sau" ON ("sau"."Id" = "userInRole"."SysUse
                    WHERE
                        "sau"."Id" = UserId
                    UNION ALL
                    SELECT "ChiefUnitsSelect"."Id"
                    FROM "ChiefUnitsSelect"
                    UNION ALL
                    SELECT
                        "SysAdminUnit"."Id"
                    FROM
                        "SysAdminUnit"
                    WHERE
                        ("SysAdminUnit"."ParentRoleId" IS NULL OR "SysAdminUnit"."Id" = UserId)
                        AND "SysAdminUnitTypeValue" < 4
                    UNION ALL
                    SELECT
                        "FuncRoleId"
                    FROM
                        "SysFuncRoleInOrgRole"
                    WHERE
                        "SysFuncRoleInOrgRole"."OrgRoleId" = UserId
                    )
            UNION ALL
```

```
            SELECT
                "SelectPriorLevel"."Id",
                "SelectPriorLevel"."Name",
                "SelectPriorLevel"."ParentRoleId",
                "Level" + 1 "level"
            FROM
                "MainSelect" "SelectPriorLevel"
                INNER JOIN "HierarchicalSelect" AS "hierSelect" ON ("hierSelect"."ParentRoleId"
        ),
        "FuncRoleHierarchicalSelect" AS (
            SELECT
                "StartLevel"."Id",
                "StartLevel"."Name",
                "StartLevel"."ParentRoleId",
                0 "Level"
            FROM
                "MainSelect" "StartLevel"
            WHERE EXISTS (
                SELECT NULL
                FROM "SysFuncRoleInOrgRole" AS "funcRoleInOrgRole"
                    INNER JOIN "HierarchicalSelect" AS "hierSelect" ON "funcRoleInOrgRole"."OrgR
                WHERE "funcRoleInOrgRole"."FuncRoleId" = "StartLevel"."Id"
            )
            UNION ALL
            SELECT
                "PriorLevel"."Id",
                "PriorLevel"."Name",
                "PriorLevel"."ParentRoleId",
                "Level" + 1 "level"
            FROM
                "MainSelect" "PriorLevel"
                INNER JOIN "FuncRoleHierarchicalSelect" AS "funcRoleHierSelect" ON ("funcRoleHie
        ),
        "DependentUserSelect" AS (
            SELECT
                "mainSelect"."Id" "Id",
                "mainSelect"."Name" "Name",
                "mainSelect"."ParentRoleId" "ParentRoleId",
                0 "Level"
            FROM
                "MainSelect" AS "mainSelect"
            INNER JOIN "SysUserInRole" AS "userInRole"
                ON "mainSelect"."Id" = "userInRole"."SysUserId"
            INNER JOIN "ChiefUnitsSelect" AS "AllUnits"
                ON "AllUnits"."Id" = "userInRole"."SysRoleId"
            WHERE
                NOT EXISTS (
                        SELECT
                            "UserUnits"."Id"
```

```
                    FROM "ChiefUnitsSelect" AS "UserUnits"
                    INNER JOIN "SysUserInRole" AS "UserInRole"
                        ON "UserUnits"."Id" = "UserInRole"."SysRoleId"
                    INNER JOIN "SysAdminUnit" AS "sau"
                        ON "sau"."Id" = "UserUnits"."Id"
                    WHERE "sau"."SysAdminUnitTypeValue" = 2
                        AND "UserInRole"."SysUserId" = UserId
                        AND "UserUnits"."Id" = "AllUnits"."Id")
    )
    INSERT INTO "AdminUnitList" ("Id", "Name", "ParentRoleId", "Granted", "Level")
    SELECT DISTINCT
        "AdminUnitList"."Id",
        "AdminUnitList"."Name",
        "AdminUnitList"."ParentRoleId",
        IsGranted,
        NestLevel
    FROM (
        SELECT
            "HierarchicalSelect"."Id",
            "HierarchicalSelect"."Name",
            "HierarchicalSelect"."ParentRoleId"
        FROM "HierarchicalSelect"
        UNION ALL
        SELECT
            "SysAdminUnit"."Id",
            "SysAdminUnit"."Name",
            "SysAdminUnit"."ParentRoleId"
        FROM "SysAdminUnit"
        WHERE
            "SysAdminUnit"."Id" = UserId
        UNION ALL
        SELECT
            "FuncRoleHierarchicalSelect"."Id",
            "FuncRoleHierarchicalSelect"."Name",
            "FuncRoleHierarchicalSelect"."ParentRoleId"
        FROM "FuncRoleHierarchicalSelect"
        UNION ALL
        SELECT
            "DependentUserSelect"."Id",
            "DependentUserSelect"."Name",
            "DependentUserSelect"."ParentRoleId"
        FROM "DependentUserSelect"
    ) AS "AdminUnitList";

DependentUsersList := 'DependentUsersList' || NestLevel ;
FOR DependentUser IN DependentUsersList LOOP
    EXIT WHEN DependentUser = NULL;
    DependentUserId = DependentUser."Id";
```

```
        PERFORM "tsp_GetAdminUnitList"(DependentUserId, 1, NestLevel + 1);
    END LOOP;

    GetGrantorSysAdminUnitList := 'GetGrantorSysAdminUnitList' || NestLevel ;
    FOR GrantorSysAdminUnit IN GetGrantorSysAdminUnitList LOOP
        EXIT WHEN GrantorSysAdminUnit = NULL;
        GrantorSysAdminUnitId = GrantorSysAdminUnit."Id";
        PERFORM "tsp_GetAdminUnitList"(GrantorSysAdminUnitId, 1, NestLevel + 1);
    END LOOP;

    IF NestLevel = 0 THEN
        RETURN QUERY
        SELECT "QQ"."Id",
                "QQ"."Name",
                "QQ"."ParentRoleId"
        FROM (
            SELECT DISTINCT
                "AdminUnitList"."Id",
                "AdminUnitList"."Name",
                "AdminUnitList"."ParentRoleId",
                "sau"."SysAdminUnitTypeValue"
            FROM "AdminUnitList"
            INNER JOIN "SysAdminUnit" AS "sau" ON "sau"."Id" = "AdminUnitList"."Id") AS "QQ"
        ORDER BY "QQ"."SysAdminUnitTypeValue" DESC;
    END IF;

END;
$$ LANGUAGE plpgsql;
```

Example 5 (stored procedures)

Example. Example of a stored procedure that uses exception handling and executes a custom script.

MSSQL

```
-- Stored procedure that uses exception handling and executes a custom script
-- MSSQL
IF  EXISTS (SELECT * FROM sys.objects WHERE object_id = OBJECT_ID(N'[dbo].[tsp_CanConvertData]')
DROP PROCEDURE [dbo].[tsp_CanConvertData]
GO
CREATE PROCEDURE [dbo].[tsp_CanConvertData]
    @EntitySchemaName SYSNAME,
    @SourceColumnName SYSNAME,
```

```
    @NewColumnDataType SYSNAME,
    @Result BIT OUT
AS
BEGIN
    SET NOCOUNT ON

    SET @Result = 0

    DECLARE @sql NVARCHAR(MAX)
    DECLARE @unicodeCharLength INT = 2
    DECLARE @dataTypeName SYSNAME
    DECLARE @dataTypeSize INT
    DECLARE @dataTypePrecision INT

    SELECT
        @dataTypeName = UPPER(DATA_TYPE),
        @dataTypeSize =
        CASE
            WHEN CHARACTER_MAXIMUM_LENGTH IS NULL THEN NUMERIC_PRECISION
            ELSE CHARACTER_MAXIMUM_LENGTH
        END,
        @dataTypePrecision = ISNULL(NUMERIC_SCALE, 0)
    FROM INFORMATION_SCHEMA.COLUMNS
    WHERE TABLE_NAME = @EntitySchemaName
    AND COLUMN_NAME = @SourceColumnName

    IF (@dataTypeName IS NULL)
    BEGIN
        RETURN
    END

    DECLARE @newDataTypeName SYSNAME
    DECLARE @newDataTypeSize INT
    DECLARE @newDataTypePrecision INT
    DEClARE @i INT
    DECLARE @newDataTypeSizeDefinition NVARCHAR(MAX)

    SET @i = CHARINDEX('(', @NewColumnDataType)
    IF (@i = 0)
    BEGIN
        SET @newDataTypeName = @NewColumnDataType
        SET @newDataTypeSize = 0
        SET @newDataTypePrecision = 0
    END ELSE
    BEGIN
        SET @newDataTypeName = UPPER(LTRIM(RTRIM(SUBSTRING(@NewColumnDataType, 1, @i - 1))))
        SET @newDataTypeSizeDefinition = LTRIM(RTRIM(SUBSTRING(@NewColumnDataType, @i + 1,
            LEN(@NewColumnDataType))))
        SET @i = CHARINDEX(')', @newDataTypeSizeDefinition)
```

```
        IF (@i > 0)
        BEGIN
            SET @newDataTypeSizeDefinition = LTRIM(RTRIM(SUBSTRING(@newDataTypeSizeDefinition, 1
        END
        SET @i = CHARINDEX(',', @newDataTypeSizeDefinition)
        IF (@i > 0)
        BEGIN
            SET @newDataTypeSize = CAST(LTRIM(RTRIM(SUBSTRING(@newDataTypeSizeDefinition, 1, @i
            SET @newDataTypePrecision = CAST(LTRIM(RTRIM(SUBSTRING(@newDataTypeSizeDefinition, @
                LEN(@newDataTypeSizeDefinition)))) AS INT)
        END ELSE
        BEGIN
            SET @newDataTypePrecision = 0
            IF (UPPER(@newDataTypeSizeDefinition) = 'MAX')
            BEGIN
                SET @newDataTypeSize = -1
            END ELSE
            BEGIN
                SET @newDataTypeSize = CAST(@newDataTypeSizeDefinition AS INT)
            END
        END
    END

    DECLARE @ImplicitDataConvertTable TABLE (
        SourceDataType SYSNAME,
        DestinationDataType SYSNAME
    )
    INSERT INTO @ImplicitDataConvertTable
    SELECT 'INT', 'INT'
    UNION ALL
    SELECT 'INT', 'BIT'
    UNION ALL
    SELECT 'INT', 'DECIMAL'
    UNION ALL
    SELECT 'INT', 'VARCHAR'
    UNION ALL
    SELECT 'INT', 'NVARCHAR'
    UNION ALL
    SELECT 'INT', 'VARBINARY'
    UNION ALL
    SELECT 'BIT', 'BIT'
    UNION ALL
    SELECT 'BIT', 'INT'
    UNION ALL
    SELECT 'BIT', 'DECIMAL'
    UNION ALL
    SELECT 'BIT', 'VARCHAR'
    UNION ALL
```

```
	SELECT 'BIT', 'NVARCHAR'
	UNION ALL
	SELECT 'BIT', 'VARBINARY'
	UNION ALL
	SELECT 'DECIMAL', 'BIT'
	UNION ALL
	SELECT 'UNIQUEIDENTIFIER', 'UNIQUEIDENTIFIER'
	UNION ALL
	SELECT 'UNIQUEIDENTIFIER', 'VARBINARY'
	UNION ALL
	SELECT 'VARCHAR', 'INT'
	UNION ALL
	SELECT 'VARCHAR', 'BIT'
	UNION ALL
	SELECT 'VARCHAR', 'UNIQUEIDENTIFIER'
	UNION ALL
	SELECT 'DATETIME2', 'DATETIME2'
	UNION ALL
	SELECT 'DATETIME2', 'DATE'
	UNION ALL
	SELECT 'DATETIME2', 'TIME'
	UNION ALL
	SELECT 'DATETIME2', 'VARCHAR'
	UNION ALL
	SELECT 'DATE', 'DATE'
	UNION ALL
	SELECT 'DATE', 'DATETIME2'
	UNION ALL
	SELECT 'DATE', 'VARCHAR'
	UNION ALL
	SELECT 'DATE', 'NVARCHAR'
	UNION ALL
	SELECT 'TIME', 'TIME'
	UNION ALL
	SELECT 'TIME', 'DATETIME2'
	UNION ALL
	SELECT 'TIME', 'VARCHAR'
	UNION ALL
	SELECT 'TIME', 'NVARCHAR'
	UNION ALL
	SELECT 'VARBINARY', 'INT'
	UNION ALL
	SELECT 'VARBINARY', 'BIT'
	UNION ALL
	SELECT 'VARBINARY', 'UNIQUEIDENTIFIER'

	IF EXISTS(SELECT * FROM @ImplicitDataConvertTable
	    WHERE SourceDataType = @dataTypeName AND DestinationDataType = @newDataTypeName)
	BEGIN
```

```
        SET @Result = 1
        RETURN
    END

    DECLARE @ImplicitDataOverflowConvertTable TABLE (
        SourceDataType SYSNAME,
        DestinationDataType SYSNAME
    )
    INSERT INTO @ImplicitDataOverflowConvertTable
    SELECT 'DECIMAL', 'INT'
    UNION ALL
    SELECT 'DECIMAL', 'DECIMAL'
    UNION ALL
    SELECT 'DECIMAL', 'VARCHAR'
    UNION ALL
    SELECT 'DECIMAL', 'NVARCHAR'
    UNION ALL
    SELECT 'DECIMAL', 'VARBINARY'
    UNION ALL
    SELECT 'UNIQUEIDENTIFIER', 'VARCHAR'
    UNION ALL
    SELECT 'UNIQUEIDENTIFIER', 'NVARCHAR'
    UNION ALL
    SELECT 'VARCHAR', 'INT'
    UNION ALL
    SELECT 'VARCHAR', 'BIT'
    UNION ALL
    SELECT 'VARCHAR', 'DECIMAL'
    UNION ALL
    SELECT 'VARCHAR', 'VARCHAR'
    UNION ALL
    SELECT 'VARCHAR', 'NVARCHAR'
    UNION ALL
    SELECT 'NVARCHAR', 'INT'
    UNION ALL
    SELECT 'NVARCHAR', 'BIT'
    UNION ALL
    SELECT 'NVARCHAR', 'DECIMAL'
    UNION ALL
    SELECT 'NVARCHAR', 'VARCHAR'
    UNION ALL
    SELECT 'NVARCHAR', 'NVARCHAR'
    UNION ALL
    SELECT 'VARBINARY', 'VARCHAR'
    UNION ALL
    SELECT 'VARBINARY', 'NVARCHAR'
    UNION ALL
    SELECT 'VARBINARY', 'VARBINARY'
```

```
IF EXISTS(SELECT * FROM @ImplicitDataOverflowConvertTable
    WHERE SourceDataType = @dataTypeName AND DestinationDataType = @newDataTypeName)
BEGIN
    SET @sql = N'IF EXISTS(SELECT * FROM [' + @EntitySchemaName + ']) SET @Result = 0 ELSE S
    EXEC sp_executesql @sql, N'@Result BIT OUT', @Result = @Result OUT

    IF (@Result = 1)
    BEGIN
        RETURN
    END
    BEGIN TRY
        IF (@dataTypeName = 'DECIMAL' AND @newDataTypeName = 'INT') OR
            (@dataTypeName = 'DECIMAL' AND @newDataTypeName = 'VARCHAR') OR
            (@dataTypeName = 'DECIMAL' AND @newDataTypeName = 'NVARCHAR') OR
            (@dataTypeName = 'DECIMAL' AND @newDataTypeName = 'VARBINARY') OR
            (@dataTypeName = 'DECIMAL' AND @newDataTypeName = 'DECIMAL')
        BEGIN
            DECLARE @cnt INT
            DECLARE @ConvertDescription NVARCHAR(MAX)
            SET @ConvertDescription = 'CONVERT(' + @NewColumnDataType +', [' + @SourceColumn
            SET @sql = N'IF EXISTS(SELECT * FROM [' + @EntitySchemaName + '] WHERE ' +
            @ConvertDescription + ' = ' + @ConvertDescription + ') SET @cnt = 1 ELSE SET @cn
            EXEC sp_executesql @sql, N'@cnt INT OUT', @cnt = @cnt OUT
            SET @Result = 1
        END ELSE
        BEGIN
            DECLARE @dl INT
            SET @sql = N'SELECT @dl = MAX(DATALENGTH([' + @SourceColumnName + ''])) ' +
            'FROM [' + @EntitySchemaName + ']'
            EXEC sp_executesql @sql, N'@dl INT OUT', @dl = @dl OUT
            IF (@newDataTypeName IN ('VARCHAR', 'NVARCHAR', 'VARBINARY') AND @newDataTypeSiz
            BEGIN
                SET @Result = 1
            END ELSE
            IF (@dl <= @newDataTypeSize OR (
                @newDataTypeName IN ('NVARCHAR', 'NCHAR') AND (@dl / @unicodeCharLength) <=
            BEGIN
                SET @Result = 1
            END ELSE
            BEGIN
                SET @Result = 0
            END
        END
    END TRY
    BEGIN CATCH
        SET @Result = 0
    END CATCH
END ELSE
```

```
    BEGIN
        SET @Result = 0
    END
END
GO
```

PostgreSQL

```
-- Stored procedure that uses exception handling and executes a custom scrip
-- PostgreSQL
DROP FUNCTION IF EXISTS public."tsp_CanConvertData" CASCADE;
CREATE FUNCTION public."tsp_CanConvertData"(
    EntitySchemaName NAME,
    SourceColumnName NAME,
    NewColumnDataType NAME,
    CanConvert OUT BOOLEAN)
AS $BODY$
DECLARE
    dataTypeName NAME;
    newDataTypeName NAME;
    newDataTypeSize INTEGER;
    countRow INTEGER;
    dataLength INTEGER;
    convertDescription TEXT;
    unicodeCharLength INTEGER = 2;
    sqlQuery TEXT;
    castQuery TEXT;
BEGIN
    CanConvert = FALSE;
    dataTypeName = (
        SELECT UPPER(data_type) FROM information_schema.columns
        WHERE table_name = EntitySchemaName AND column_name = SourceColumnName);
    IF dataTypeName IS NULL THEN
        RETURN;
    END IF;

    SELECT "fn_ParseDataType".DataTypeName, "fn_ParseDataType".DataTypeSize
    FROM public."fn_ParseDataType"(NewColumnDataType)
    INTO newDataTypeName, newDataTypeSize;

    DROP TABLE IF EXISTS "NotConvertTable";
    CREATE TEMP TABLE "NotConvertTable" (
        SourceDataType NAME,
        DestinationDataType NAME
    );
    INSERT INTO "NotConvertTable" VALUES
```

```
        ('INTEGER', 'UUID'),
        ('INTEGER', 'TIMESTAMP WITHOUT TIME ZONE'),
        ('INTEGER', 'DATE'),
        ('INTEGER', 'TIME WITHOUT TIME ZONE'),
        ('NUMERIC', 'UUID'),
        ('NUMERIC', 'TIMESTAMP WITHOUT TIME ZONE'),
        ('NUMERIC', 'DATE'),
        ('NUMERIC', 'TIME WITHOUT TIME ZONE'),
        ('BOOLEAN', 'UUID'),
        ('BOOLEAN', 'TIMESTAMP WITHOUT TIME ZONE'),
        ('BOOLEAN', 'DATE'),
        ('BOOLEAN', 'TIME WITHOUT TIME ZONE'),
        ('UUID', 'INTEGER'),
        ('UUID', 'NUMERIC'),
        ('UUID', 'BOOLEAN'),
        ('UUID', 'TIMESTAMP WITHOUT TIME ZONE'),
        ('UUID', 'DATE'),
        ('UUID', 'TIME WITHOUT TIME ZONE'),
        ('TIMESTAMP WITHOUT TIME ZONE', 'INTEGER'),
        ('TIMESTAMP WITHOUT TIME ZONE', 'NUMERIC'),
        ('TIMESTAMP WITHOUT TIME ZONE', 'BOOLEAN'),
        ('TIMESTAMP WITHOUT TIME ZONE', 'UUID'),
        ('DATE', 'INTEGER'),
        ('DATE', 'NUMERIC'),
        ('DATE', 'BOOLEAN'),
        ('DATE', 'UUID'),
        ('DATE', 'TIME WITHOUT TIME ZONE'),
        ('TIME WITHOUT TIME ZONE', 'INTEGER'),
        ('TIME WITHOUT TIME ZONE', 'NUMERIC'),
        ('TIME WITHOUT TIME ZONE', 'BOOLEAN'),
        ('TIME WITHOUT TIME ZONE', 'UUID'),
        ('TIME WITHOUT TIME ZONE', 'DATE');
    IF EXISTS(SELECT SourceDataType, DestinationDataType FROM "NotConvertTable"
        WHERE SourceDataType = dataTypeName AND DestinationDataType = newDataTypeName) THEN
        RETURN;
    END IF;

    DROP TABLE IF EXISTS ImplicitDataConvertTable;
    CREATE TEMP TABLE ImplicitDataConvertTable (
        SourceDataType NAME,
        DestinationDataType NAME
    );
    INSERT INTO ImplicitDataConvertTable VALUES
        ('INTEGER', 'INTEGER'),
        ('INTEGER', 'NUMERIC'),
        ('INTEGER', 'BOOLEAN'),
        ('INTEGER', 'CHARACTER VARYING'),
        ('INTEGER', 'TEXT'),
        ('NUMERIC', 'CHARACTER VARYING'),
```

```
        ('NUMERIC', 'TEXT'),
        ('BOOLEAN', 'INTEGER'),
        ('BOOLEAN', 'BOOLEAN'),
        ('BOOLEAN', 'CHARACTER VARYING'),
        ('BOOLEAN', 'TEXT'),
        ('CHARACTER VARYING', 'TEXT'),
        ('CHARACTER VARYING', 'BYTEA'),
        ('TEXT', 'TEXT'),
        ('TEXT', 'BYTEA'),
        ('BYTEA', 'BYTEA'),
        ('UUID', 'CHARACTER VARYING'),
        ('UUID', 'TEXT'),
        ('UUID', 'UUID'),
        ('TIMESTAMP WITHOUT TIME ZONE', 'CHARACTER VARYING'),
        ('TIMESTAMP WITHOUT TIME ZONE', 'TEXT'),
        ('TIMESTAMP WITHOUT TIME ZONE', 'TIMESTAMP WITHOUT TIME ZONE'),
        ('DATE', 'CHARACTER VARYING'),
        ('DATE', 'TEXT'),
        ('DATE', 'TIMESTAMP WITHOUT TIME ZONE'),
        ('DATE', 'DATE'),
        ('TIME WITHOUT TIME ZONE', 'CHARACTER VARYING'),
        ('TIME WITHOUT TIME ZONE', 'TEXT'),
        ('TIME WITHOUT TIME ZONE', 'TIMESTAMP WITHOUT TIME ZONE'),
        ('TIME WITHOUT TIME ZONE', 'TIME WITHOUT TIME ZONE'),
        ('TIMESTAMP WITHOUT TIME ZONE', 'DATE'),
        ('TIMESTAMP WITHOUT TIME ZONE', 'TIME WITHOUT TIME ZONE'),
        ('INTEGER', 'BYTEA'),
        ('NUMERIC', 'BOOLEAN'),
        ('NUMERIC', 'BYTEA'),
        ('BOOLEAN', 'NUMERIC'),
        ('BOOLEAN', 'BYTEA'),
        ('UUID', 'BYTEA'),
        ('TIMESTAMP WITHOUT TIME ZONE', 'BYTEA'),
        ('DATE', 'BYTEA'),
        ('TIME WITHOUT TIME ZONE', 'BYTEA'),
        ('NUMERIC', 'INTEGER'),
        ('NUMERIC', 'NUMERIC');
    IF EXISTS(SELECT SourceDataType, DestinationDataType FROM ImplicitDataConvertTable
        WHERE SourceDataType = dataTypeName AND DestinationDataType = newDataTypeName) THEN
        CanConvert = TRUE;
        RETURN;
    END IF;

    EXECUTE FORMAT('SELECT count(*) FROM %1$I', EntitySchemaName) INTO countRow;
    CanConvert = (countRow = 0);
    IF CanConvert THEN
        RETURN;
    END IF;
```

```
DROP TABLE IF EXISTS "ExplicitDataConvertTable";
CREATE TEMP TABLE "ExplicitDataConvertTable" (
    SourceDataType NAME,
    DestinationDataType NAME
);
INSERT INTO "ExplicitDataConvertTable" VALUES
    ('CHARACTER VARYING', 'INTEGER'),
    ('CHARACTER VARYING', 'NUMERIC'),
    ('CHARACTER VARYING', 'BOOLEAN'),
    ('CHARACTER VARYING', 'UUID'),
    ('CHARACTER VARYING', 'TIMESTAMP WITHOUT TIME ZONE'),
    ('CHARACTER VARYING', 'DATE'),
    ('CHARACTER VARYING', 'TIME WITHOUT TIME ZONE'),
    ('TEXT', 'INTEGER'),
    ('TEXT', 'NUMERIC'),
    ('TEXT', 'BOOLEAN'),
    ('TEXT', 'UUID'),
    ('TEXT', 'TIMESTAMP WITHOUT TIME ZONE'),
    ('TEXT', 'DATE'),
    ('TEXT', 'TIME WITHOUT TIME ZONE'),
    ('BYTEA', 'INTEGER'),
    ('BYTEA', 'NUMERIC'),
    ('BYTEA', 'BOOLEAN'),
    ('BYTEA', 'UUID'),
    ('BYTEA', 'TIMESTAMP WITHOUT TIME ZONE'),
    ('BYTEA', 'DATE'),
    ('BYTEA', 'TEXT'),
    ('BYTEA', 'TIME WITHOUT TIME ZONE'),
    ('NUMERIC', 'BOOLEAN');
IF EXISTS(SELECT SourceDataType, DestinationDataType FROM "ExplicitDataConvertTable"
    WHERE SourceDataType = dataTypeName AND DestinationDataType = newDataTypeName) THEN
    castQuery = FORMAT('CAST(%1$I%3$s AS %2$s)', SourceColumnName, NewColumnDataType,
        CASE
            WHEN dataTypeName = 'BYTEA' THEN '::TEXT'
            WHEN dataTypeName = 'NUMERIC' THEN '::INTEGER'
            ELSE ''
        END);
    sqlQuery = FORMAT('SELECT COUNT(*) FROM %1$I WHERE %2$s = %2$s',
        EntitySchemaName, castQuery);
    BEGIN
        EXECUTE sqlQuery;
        CanConvert = TRUE;
    EXCEPTION WHEN OTHERS THEN
        CanConvert = FALSE;
    END;
    RETURN;
END IF;
```

```
    DROP TABLE IF EXISTS "ImplicitDataOverflowConvertTable";
    CREATE TEMP TABLE "ImplicitDataOverflowConvertTable" (
        SourceDataType NAME,
        DestinationDataType NAME
    );
    INSERT INTO "ImplicitDataOverflowConvertTable" VALUES
        ('CHARACTER VARYING', 'CHARACTER VARYING'),
        ('TEXT', 'CHARACTER VARYING'),
        ('BYTEA', 'CHARACTER VARYING');
    IF EXISTS(SELECT SourceDataType, DestinationDataType FROM "ImplicitDataOverflowConvertTable"
        WHERE SourceDataType = dataTypeName AND DestinationDataType = newDataTypeName) THEN
        EXECUTE FORMAT('SELECT count(*) FROM %1$I', EntitySchemaName) INTO countRow;
        CanConvert = (countRow = 0);
        IF CanConvert THEN
            RETURN;
        END IF;
        BEGIN
            EXECUTE FORMAT('SELECT MAX(PG_COLUMN_SIZE(%1$I)) FROM %2$I', SourceColumnName, Entit
            INTO dataLength;
            IF (dataLength <= newDataTypeSize) THEN
                CanConvert = TRUE;
            ELSE
                CanConvert = FALSE;
            END IF;
        EXCEPTION WHEN OTHERS THEN
            CanConvert = FALSE;
        END;
    END IF;
END;
$BODY$
LANGUAGE 'plpgsql';
```

Example 6 (functions)

Example. Example of a function.

MSSQL

```
-- Function
-- MSSQL
IF  EXISTS (SELECT * FROM sys.objects
        WHERE object_id = OBJECT_ID(N'[dbo].[fn_IsGuid]') AND type = N'FN')
DROP FUNCTION [dbo].[fn_IsGuid]
```

```
GO

CREATE FUNCTION [dbo].[fn_IsGuid] (
    @ValidateValue NVARCHAR(MAX))
RETURNS BIT
AS
BEGIN
    DECLARE @hasLeftBraces BIT
    IF @ValidateValue LIKE '{%'
    BEGIN
        SET @ValidateValue = SUBSTRING(@ValidateValue, 2, LEN(@ValidateValue) - 1)
        SET @hasLeftBraces = 1
    END ELSE
    BEGIN
        SET @hasLeftBraces = 0
    END
    DECLARE @hasRightBraces BIT
    IF @ValidateValue LIKE '%}'
    BEGIN
        SET @ValidateValue = SUBSTRING(@ValidateValue, 1, LEN(@ValidateValue) - 1)
        SET @hasRightBraces = 1
    END ELSE
    BEGIN
        SET @hasRightBraces = 0
    END
    DECLARE @Result BIT
    IF @ValidateValue LIKE '[0-9a-fA-F][0-9a-fA-F][0-9a-fA-F][0-9a-fA-F][0-9a-fA-F][0-9a-fA-F][0
    BEGIN
        SET @Result = 1
    END ELSE
    BEGIN
        SET @Result = 0
    END
    IF @hasLeftBraces = @hasRightBraces
    BEGIN
        RETURN @Result
    END ELSE
    BEGIN
        SET @Result = 0
    END
    RETURN @Result
END
GO
```

PostgreSQL

```
-- Function
```

```
-- PostgreSQL
DROP FUNCTION IF EXISTS "public"."fn_IsGuid";
CREATE OR REPLACE FUNCTION public."fn_IsGuid"(ValidateValue IN VARCHAR) RETURNS BOOLEAN AS $$
    BEGIN
        IF (regexp_matches(ValidateValue, '^\{?[0-9a-fA-F]{8}-[0-9a-fA-F]{4}-[0-9a-fA-F]{4}-[0-9
            RETURN TRUE;
        ELSE
            RETURN FALSE;
        END IF;
    END;
$$ LANGUAGE plpgsql;
```

Backward compatible SQL scripts

Medium

Backward compatible SQL scripts are available in Creatio version 8.0.3 and later.

If the package installation ends with an error, you cannot roll the configuration back completely since SQL scripts make irreversible changes to the database as part of the package installation. To prevent that, use backward compatible SQL scripts instead.

A **backward compatible SQL script** (safe SQL script) is an SQL script whose execution does not make irreversible changes to the database. Changes made by backward compatible SQL scripts as part of the package installation do not affect Creatio's operability. Use backward compatible SQL scripts to ensure you can roll back to the previous configuration state completely in case of an error. Learn more in a separate article: Delivery management process.

Note. Since you can roll back to the previous configuration state from the package backup after the package installation is complete, restoration is accomplished by re-executing the previous version of the SQL script rather than by canceling the transaction.

Creatio version 8.0.5 Atlas and later lets you customize delivery process to execute CRUD operations with data as part of package installation in a way that does not cause irreparable changes to the database. Learn more in a separate article: Customize delivery process.

To develop a backward compatible SQL script, follow the recommendations listed in a different section during the development: Development recommendations for backward compatible SQL scripts.

Usage recommendations for backward compatible SQL scripts

Creatio lets you roll back to the previous configuration state from the package backup if the package installation ends with an error or a successful installation results in issues with Creatio functionality. Learn more in a separate article: Delivery management. To successfully roll back to the previous configuration state from the package

backup, [*SQL script*] configuration items must meet the guidelines.

Recommendations to ensure successful rollback to the previous configuration state from the package backup:

- No new or changed package configuration elements of the [*SQL script*] type. I.e., the modification date of the configuration element of the [*SQL script*] type is the same as date of its modification on the production environment to install the package.
- New or changed package configuration elements of the [*SQL script*] type are backward compatible.

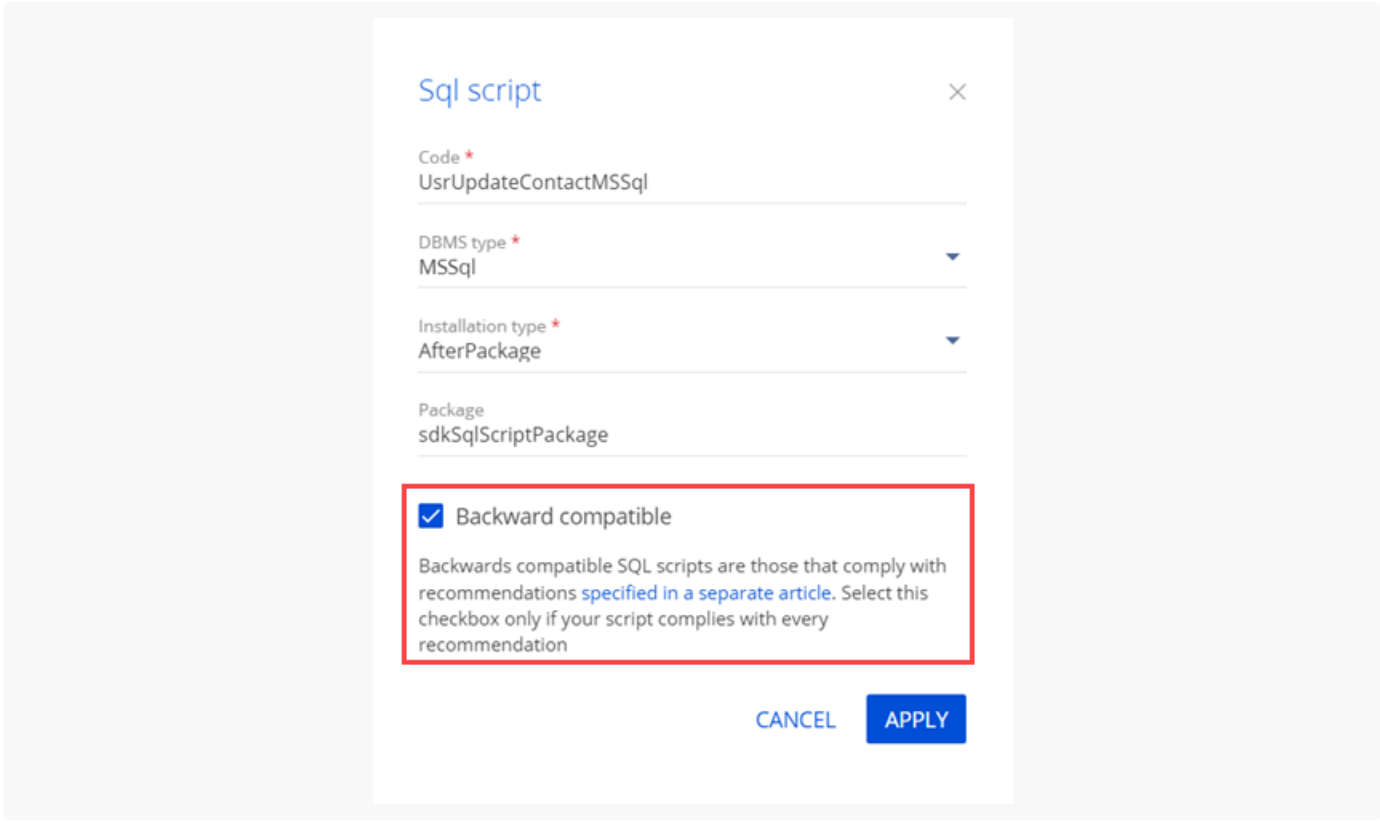
Compliance with one of the recommendations ensures successful rollback to the previous configuration state from the package backup.

Implement a backward compatible SQL script

1. Create a configuration element of the [*SQL script*] type. To do this, follow the instructions in a separate article: SQL script.
2. Implement a script that complies with the recommendations. Learn more about the recommendations in a different section: Development recommendations for backward compatible SQL scripts.

   Follow these **practices** when you develop a backward compatible SQL script:

   - Analyze the potential behavior of Creatio after it executes the SQL script.
   - Take into account the way the SQL script affects Creatio operability after the configuration rollback.
3. Click in the setup area of the Script Designer and select the [*Backward compatible*] checkbox. The checkbox flags the SQL scenario as backward compatible. The developer that implements the SQL scenario is in charge of selecting the checkbox and ensuring the SQL script complies with the recommendations.



Attention. If you install a package that contains at least one configuration element of the [*SQL script*] type without the [*Backward compatible*] checkbox selected, fails, rollback to the previous configuration state from the package backup via the WorkspaceConsole utility will be suspended. Learn more in another article: Delivery management.

Development recommendations for backward compatible SQL scripts

Follow the development **recommendations** to create backward compatible SQL scripts that do not cause issues with Creatio functionality after the configuration rollback. The recommendations are based on open sources as well as the official documentation of Microsoft SQL, PostgreSQL, and Oracle database management systems supported by Creatio.

Recommendation. Follow the ACID principle when you develop SQL scripts.

The **ACID principle** is a set of properties that ensures the reliability of database transactions. Learn more about the ACID principle in Wikipedia.

The ACID principle is based on the following **properties**:

- **A**tomicity
- **C**onsistency
- **I**solation
- **D**urability

A single SQL script must correspond to a single entity. To **execute several SQL scripts in a set order**, follow the instructions in a separate article: SQL script.

Recommendation. Do not use SQL scripts to change database entities.

A database entity is a view, function, procedure, or trigger. When you develop SQL scripts that change entities, keep in mind that Creatio might not recognize certain database tables or database table fields when you roll back to the previous configuration state from the package backup. Use SQL scripts to recreate entities instead of editing them.

To **recreate an entity using an SQL script**:

1. Make sure the needed entity exists in the database tables.
2. Delete the current entity version.
3. Create a new entity version. Use the `CREATE` operator instead of `ALTER` operator to do this.
4. Set the [*Installation type*] property of the configuration element of the [*SQL script*] type to “`AfterPackage`” or “`AfterSchemaData`.” Learn more about the available values of the [*Installation type*] property in a separate article: SQL script.

Recommendation. Do not use SQL scripts to create triggers.

For example, if you develop an SQL script that creates a field management trigger, Creatio behaves as follows. After Creatio executes an SQL script and you roll back to the previous configuration version from the package backup, issues with Creatio functionality can occur. Since the fields created by the trigger are new for the version to which you rolled back, you cannot delete the trigger. As such, this can lead to issues with Creatio functionality.

Recommendation. Do not use SQL scripts to delete database entities.

A database entity is a view, procedure, function, trigger, or database table.

We do not recommend deleting unused entities. Deleting unused entities violates dependencies in previous versions, thus causing issues with Creatio functionality. After Creatio executes the SQL script and you roll back to the previous configuration version from the package backup, deleted entities are not restored unless the SQL script recreates them in the previous package version. If you decide to delete a database entity regardless, make sure Creatio has not used it for a significant time.

Situations when you need to delete entities when deleting an app from Creatio, for example, when deleting Marketplace apps, are an exception.

To **delete an entity using an SQL script when deleting an app**, set the [*Installation type*] property of the configuration element of the [*SQL script*] type to "`UninstallApp`." Learn more about the available values of the [*Installation type*] property in a separate article: SQL script.

Recommendation. Do not use SQL scripts to create and edit database tables.

Use **objects** instead of SQL scripts to create and edit database tables. Learn more in a separate article: Object.

You can use SQL scripts to create and edit the following **database tables**:

- temporary tables
- service tables, i. e., tables that do not contain business data and are required to solve technical problems

Follow these **practices** when creating and editing such database tables using SQL scripts:

- Make sure changes described by the SQL script do not cause issues with Creatio functionality.
- Use table versioning that lets you switch logic between table versions.
- Make sure fields, tables, and other objects required for the SQL scenario exist.
- Keep in mind the use of such tables and their fields in indexes, primary keys, procedures, functions, triggers, and views.

Recommendation. Do not use SQL scripts to change the field type or requirement flag in database tables.

Create new fields using **objects** to change the field type or requirement flag in database tables instead of changing the existing fields using SQL scripts. Learn more in a separate article: Object.

For example, if you execute an SQL script that changes the field type, Creatio behaves as follows. After Creatio executes the SQL script and you roll back to the previous configuration version from the package backup, the mismatched data type error occurs.

For example, if you execute an SQL script that changes the field requirement flag, Creatio behaves as follows. After Creatio executes the SQL script and you roll back to the previous configuration version from the package backup, the field remains required and you still have to fill it out to save the record.

> **Recommendation.** Do not use SQL scripts to add required fields that have default values to database tables.

Use **objects** instead of SQL scripts to add required fields that have default values. Learn more in a separate article: Object.

> **Recommendation.** Do not use SQL scripts to add data to database tables.

Use **data bindings** or install scripts instead of SQL scripts to add data to database tables. Learn more in separate articles: Packages basics, Customize delivery process.

Make sure database tables do not have data that must be added using a complex filter or special procedure. This is relevant for key fields first and foremost. Creatio can execute an SQL script multiple times as part of debugging. Learn more in a separate guide: Debugging. Creatio does not delete data added by the SQL script when you roll back to the previous configuration version from the package backup. This can lead to issues with Creatio functionality.

> **Recommendation.** Do not use SQL scripts to change database table data.

Use **data bindings** or install scripts instead of SQL scripts to change database table data. Learn more in separate articles: Packages basics, Customize delivery process.

Some changes to database table data executed using an SQL script cannot be reversed using another SQL script or via a rollback to the previous configuration version from the package backup. For example, if Creatio executes an SQL script that changes the value of the `[IsActive]` column in the `[SysProcess]` database table from `true` to `false`, Creatio behaves as follows.

```
UPDATE SysProcess SET IsActive = true WHERE IsActive = false
```

After Creatio executes the SQL script, the `[IsActive]` column of the `[SysProcess]` database table has no `false` values. I. e., Creatio cannot determine which column values the SQL script changed. You can develop an SQL script that reverses the changes, but this requires significant effort on your end.

> **Recommendation.** Do not use SQL scripts to delete business data from database tables.

We do not recommend deleting business data. For example, if you execute an SQL script that deletes field data, Creatio behaves as follows. After Creatio executes the SQL script and you roll back to the previous configuration version from the package backup, the field data remains deleted. I. e., the script deletes data permanently. The fact that the SQL script of the package to the version of which you roll back added this data is irrelevant. Use **objects** or install scripts instead of SQL scripts to create and change database table data. Learn more in separate articles: Object, Customize delivery process.

Service data, for example, the cleanup of the scheduler queue, is an exception. However, keep in mind that the deletion is permanent.

Recommendation. Ensure that the changes made by backward compatible SQL scripts do not affect the data integrity.

Recommendation. Check whether database tables contain procedures and functions before you call them in SQL scripts.

In most cases, you do not need to call procedures and functions in SQL scripts during a package installation. If such a need arises, make sure that the database tables contain the required procedures and functions.