/\* ========= EDIT THESE TWO VALUES ========= \*/

DECLARE @SchemaName sysname = N'dbo'; -- e.g., 'dbo'

DECLARE @TableName sysname = N'OrgCatalog'; -- e.g., the table you found

/\* ========================================== \*/

/\* Optional: sample peek \*/

DECLARE @sql nvarchar(MAX) =

N'SELECT TOP (5) \* FROM ' + QUOTENAME(@SchemaName) + N'.' + QUOTENAME(@TableName) + N';';

EXEC sp\_executesql @sql;

/\* 1) Create the user→company mapping table (adjust names if you prefer) \*/

IF OBJECT\_ID(N'dbo.UserCompanyAccess', N'U') IS NULL

BEGIN

CREATE TABLE dbo.UserCompanyAccess

(

UserName nvarchar(256) NOT NULL, -- store lower-case UPN/login

Company int NOT NULL, -- must match your table''s Company column

AccessLevel nvarchar(50) NULL, -- optional (Reader/Manager/Admin)

CONSTRAINT PK\_UserCompanyAccess PRIMARY KEY (UserName, Company)

);

CREATE INDEX IX\_UserCompanyAccess\_User ON dbo.UserCompanyAccess(UserName, Company);

END

/\* Seed a few demo rows — change Company IDs to ones that exist in your table \*/

TRUNCATE TABLE dbo.UserCompanyAccess;

INSERT dbo.UserCompanyAccess(UserName, Company, AccessLevel)

VALUES

(N'alice@company.com', 4988, N'Reader'),

(N'bob@company.com', 5055, N'Reader'),

(N'manager@company.com', 4988, N'Manager'),

(N'manager@company.com', 5055, N'Manager');

/\* 2) Create (or replace) the predicate function \*/

IF OBJECT\_ID(N'dbo.fn\_rls\_company', N'IF') IS NOT NULL

DROP FUNCTION dbo.fn\_rls\_company;

GO

CREATE FUNCTION dbo.fn\_rls\_company (@Company int)

RETURNS TABLE

WITH SCHEMABINDING

AS

RETURN

SELECT 1 AS AllowRow

WHERE EXISTS

(

SELECT 1

FROM dbo.UserCompanyAccess AS a

WHERE a.Company = @Company

AND a.UserName = LOWER(COALESCE(SESSION\_CONTEXT(N'upn'), SUSER\_SNAME()))

);

GO

/\* 3) Create the security policy (drop if exists, then create fresh) \*/

IF OBJECT\_ID(N'dbo.rls\_company\_policy', N'SP') IS NOT NULL

DROP SECURITY POLICY dbo.rls\_company\_policy;

GO

DECLARE @ddl nvarchar(MAX) =

N'CREATE SECURITY POLICY dbo.rls\_company\_policy

ADD FILTER PREDICATE dbo.fn\_rls\_company(Company) ON ' + QUOTENAME(@SchemaName) + N'.' + QUOTENAME(@TableName) + N',

ADD BLOCK PREDICATE dbo.fn\_rls\_company(Company) ON ' + QUOTENAME(@SchemaName) + N'.' + QUOTENAME(@TableName) + N'

WITH (STATE = ON);';

EXEC sp\_executesql @ddl;

GO

/\* 4) Test using SESSION\_CONTEXT to simulate users (no user creation required) \*/

/\* Alice: should see only Company=4988 rows \*/

EXEC sys.sp\_set\_session\_context @key=N'upn', @value=N'alice@company.com';

DECLARE @sqlAlice nvarchar(MAX) =

N'SELECT DISTINCT Company, FName FROM ' + QUOTENAME(@SchemaName) + N'.' + QUOTENAME(@TableName) + N' ORDER BY Company;';

EXEC sp\_executesql @sqlAlice;

EXEC sys.sp\_set\_session\_context @key=N'upn', @value=NULL;

/\* Bob: should see only Company=5055 rows \*/

EXEC sys.sp\_set\_session\_context @key=N'upn', @value=N'bob@company.com';

EXEC sp\_executesql @sqlAlice;

EXEC sys.sp\_set\_session\_context @key=N'upn', @value=NULL;

/\* Manager: should see both 4988 and 5055 rows \*/

EXEC sys.sp\_set\_session\_context @key=N'upn', @value=N'manager@company.com';

EXEC sp\_executesql @sqlAlice;

EXEC sys.sp\_set\_session\_context @key=N'upn', @value=NULL;

Cleanup

ALTER SECURITY POLICY dbo.rls\_company\_policy WITH (STATE = OFF);

DROP SECURITY POLICY dbo.rls\_company\_policy;

DROP FUNCTION dbo.fn\_rls\_company;

DROP TABLE dbo.UserCompanyAccess;  
  
  
  
“We implemented a pilot RLS on the **OrgCatalog** table using **Company** as the access key. A small mapping table defines user→company access. A schema-bound predicate function + security policy enforces row filtering and blocks unauthorized writes. We validated with session context to simulate different users.”

**📝 Note to add *before starting (what you will do)***

**Task: Implement Row-Level Security (RLS) proof-of-concept**  
I will implement a test of RLS on the [Org table] using the **Company** column as the filter key.

* Create a **mapping table** (UserCompanyAccess) to store which test users can see which companies.
* Define a **predicate function** to check the current user’s access against the mapping table.
* Create a **security policy** that enforces filtering (only rows allowed for each user are visible).
* Simulate test users (Alice, Bob, Manager) using SESSION\_CONTEXT to validate the policy.
* Verify that each test user only sees the rows for their assigned company.

**✅ Note to add *after completion (results/outcome)***

**Result: RLS test completed**  
I successfully implemented Row-Level Security (RLS) on the [Org table].

* Built the **UserCompanyAccess** mapping table and seeded test users.
* Created a schema-bound **predicate function** and **security policy** tied to the Company column.
* Tested using session context:
  + *Alice* saw only Company 4988 rows.
  + *Bob* saw only Company 5055 rows.
  + *Manager* saw both 4988 and 5055.
* The security policy filtered data automatically without changing any queries.
* This validates that RLS can enforce **fine-grained access control** at the database layer.

# Other option. **Before / While Implementing**

You want to show **intent + scope**:

*“Implementing a proof-of-concept of Row-Level Security (RLS) on the OrgCatalog table.  
I will create a user-to-company access mapping table, a predicate function, and a security policy that restricts query results by Company.  
Since we don’t yet have actual employee data, I’ll populate the mapping with a few test users (Alice, Bob, Manager) to simulate different access patterns.”*

**📝 After Completing**

You want to show **results + validation**:

\*“RLS pilot completed on OrgCatalog.  
The security policy now enforces filtering by Company automatically:

* Alice only sees rows for Company 4988 (Walnut Creek Hospital).
* Bob only sees rows for Company 5055 (Lovelace Insurance Company).
* Manager sees both, as expected.  
  Validation was done using session context to simulate logins.  
  This demonstrates that we can restrict sensitive data access row-by-row. The next step is to decide how we’ll maintain real user mappings (e.g., HR source, AD groups, or domain mapping).”\*

CREATE OR ALTER FUNCTION dbo.fn\_rls\_company (@Company INT)

RETURNS TABLE

WITH SCHEMABINDING

AS

RETURN

SELECT 1 AS AllowRow

WHERE

-- normal per-company access

EXISTS (

SELECT 1

FROM dbo.UserCompanyAccess a

WHERE a.Company = @Company

AND a.UserName = LOWER(COALESCE(SESSION\_CONTEXT(N'upn'), SUSER\_SNAME()))

)

-- OR global admin access

OR EXISTS (

SELECT 1

FROM dbo.UserCompanyAccess a

WHERE a.AccessLevel = N'Admin'

AND a.UserName = LOWER(COALESCE(SESSION\_CONTEXT(N'upn'), SUSER\_SNAME()))

);

No change needed to the security policy.

Insert the admin user (use any non-real company ID, e.g., 0):

sql

Copy code

INSERT dbo.UserCompanyAccess(UserName, Company, AccessLevel)

VALUES (N'superuser@company.com', 0, N'Admin'); -- 0 must not be a real Company

Test

sql

Copy code

EXEC sys.sp\_set\_session\_context 'upn', 'superuser@company.com';

SELECT DISTINCT Company, FName FROM dbo.YourRealTable ORDER BY Company;

EXEC sys.sp\_set\_session\_context 'upn', NULL; -- should see all companies