Row-level Security (RLS)

Leonard Lobel
CTO, SLEEK TECHNOLOGIES

@lennilobel

Introducing Row-level Security (RLS)

Restrict access to individual rows in a table

- Create predicate functions
- Write custom logic to control user access to every row

Security policy

- Bind the functions to tables as a filter or block predicate
- SQL Server filters and blocks user access to individual rows
- Can enable/disable the policy as desired

Filter and Block Predicates

Filter predicate

- `SELECT`, `UPDATE`, `DELETE`
  - Can’t select, update, or delete rows that violate the predicate

Block predicate

- `AFTER INSERT`, `AFTER UPDATE`
  - Can’t insert or update rows to values that would violate the predicate
- `BEFORE UPDATE`, `BEFORE DELETE`
  - Can’t update or delete rows that violate the predicate
  - Implied when combined with filter predicate

Combining RLS Predicates

Predicate	SELECT/UPDATE/DELETE rows that violate the predicate	INSERT rows with violating values	UPDATE rows to violating values

Combining RLS Predicates

Predicate	SELECT/UPDATE/DELETE rows that violate the predicate	INSERT rows with violating values	UPDATE rows to violating values
Filter	No	Yes	Yes

Combining RLS Predicates

Predicate	SELECT/UPDATE/DELETE rows that violate the predicate	INSERT rows with violating values	UPDATE rows to violating values
Filter	No	Yes	Yes
AFTER INSERT block	Yes	No	Yes

Combining RLS Predicates

Predicate	SELECT/UPDATE/DELETE rows that violate the predicate	INSERT rows with violating values	UPDATE rows to violating values
Filter	No	Yes	Yes
AFTER INSERT block	Yes	No	Yes
AFTER UPDATE block	Yes	Yes	No

Combining RLS Predicates

Predicate	SELECT/UPDATE/DELETE rows that violate the predicate	INSERT rows with violating values	UPDATE rows to violating values
Filter	No	Yes	Yes
AFTER INSERT block	Yes	No	Yes
AFTER UPDATE block	Yes	Yes	No
BEFORE UPDATE block	No	N/A	N/A

Combining RLS Predicates

Predicate	SELECT/UPDATE/DELETE rows that violate the predicate	INSERT rows with violating values	UPDATE rows to violating values
Filter	No	Yes	Yes
AFTER INSERT block	Yes	No	Yes
AFTER UPDATE block	Yes	Yes	No
BEFORE UPDATE block	No	N/A	N/A
BEFORE DELETE block	No	N/A	N/A

Creating Security Predicate Functions

Write a security predicate function

- Ordinary inline table-valued function (TVF)
  - Must be schema-bound
- Accept any parameters of any type
  - Map these parameters to column values

Implement your own custom logic in T-SQL

- Examine the row via the columns passed in as parameters
  - Determine if access should be allowed or denied
- Return a scalar 1 (allow) or nothing at all (deny)
- Encapsulate logic inside WHERE clause of a single SELECT statement inside the TVF

Creating Security Predicate Functions

```
CREATE FUNCTION sec.fn_MySecurityPredicate(@Parm1 AS int, ...)
 RETURNS TABLE
 WITH SCHEMABINDING
AS
    -- SQL Server passes in column values of each row via parameters

    RETURN
      SELECT 1 AS Result
      WHERE ...
        -- Custom logic here examines the parameters (column values)
        -- passed in, and determines the row's accessibility
```

Creating Security Predicate Functions

```
CREATE FUNCTION sec.fn_MySecurityPredicate(@Parm1 AS int, ...)
 RETURNS TABLE
 WITH SCHEMABINDING
AS
    -- SQL Server passes in column values of each row via parameters

    RETURN
      SELECT 1 AS Result
      WHERE ...
        -- Custom logic here examines the parameters (column values)
        -- passed in, and determines the row's accessibility
```

Creating Security Policies

Create a security policy

- Add filter and block predicates to the policy

Bind each predicate function to a table

- Map table columns to the TVF parameters
  - SQL Server will call the TVF to determine the accessibility of each row

Security Policy Examples

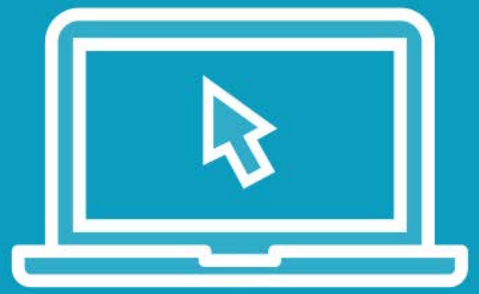
With filter predicate

```
CREATE SECURITY POLICY sec.MySecurityPolicy
 ADD FILTER PREDICATE sec.fn_MySecurityPredicate(Col1, ...)
  ON dbo.MyTable
 WITH (STATE = ON)
```

With AFTER INSERT and AFTER UPDATE block predicates

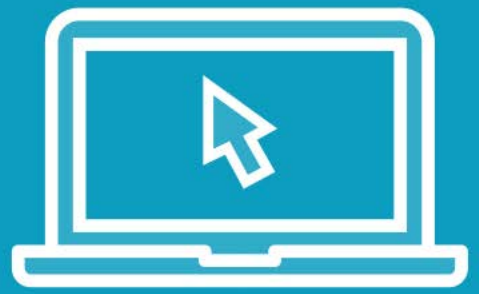
```
CREATE SECURITY POLICY sec.MySecurityPolicy
 ADD FILTER PREDICATE sec.fn_MySecurityPredicate(Col1, ...)
  ON dbo.MyTable,
 ADD BLOCK PREDICATE sec.fn_MySecurityPredicate(Col1, ...)
  ON dbo.MyTable AFTER INSERT,
 ADD BLOCK PREDICATE sec.fn_MySecurityPredicate(Col1, ...)
  ON dbo.MyTable AFTER UPDATE
 WITH (STATE = ON)
```

Demo



Creating and testing an RLS security predicate

Demo



Implementing an RLS security policy

Identifying Users for RLS

Credentials supplied for the database connection

- SQL Server login (username and password)
- Windows authentication
- Obtain the username from DATABASE_PRINCIPAL_ID

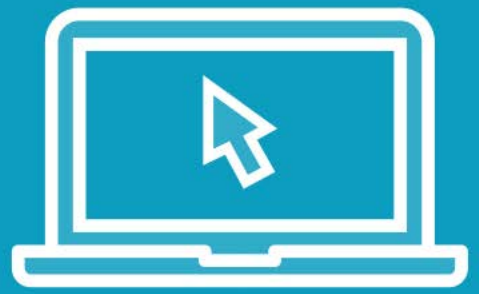
Different strategy required for n-tier applications

- Typically, all users connect to the database using the same service account from the application tier
- DATABASE_PRINCIPAL_ID is the same for every user

Solution: Use session context

- Store the application level user ID as a readonly value in session context

Demo



Using Row-level Security in n-tier applications

Summary

- Row-level security (RLS)
- Filter and block predicates
- Predicate functions
- Security policies