Database Security

CT069-3-3

ASIA PACIFIC UNIVERSITY
OF TECHNOLOGY & INNOVATION

User Administration

Topics

- Authentication vs Authorization
- SQL Login, User, Role
- Authorization Matrix
- Levels : Table, Column, Row
- Grant vs Deny vs Revoke

Security Requirements

Let's say, we have these security requirements

Create user accounts for Patients so that they can access the system and perform the below actions

- check and update their personal details
- check their own medical details such as diagnosis and medication
- must not have access to check other patients' details

Table below shows sample users. We need to create user accounts for them.

PatientNo	PatientName	Address	Telephone	DOB	Gender	RegistrationDate	MaritalStatus
Pat100	John	KL	0112345678	1995-05-11	Male	2022-06-18	Married
Pat200	Mary	Ipoh	0123456789	1998-02-04	Female	2021-05-14	Single

Authentication & Authorization

- Authentication: A mechanism that determines whether a user is who he or she claims to be.

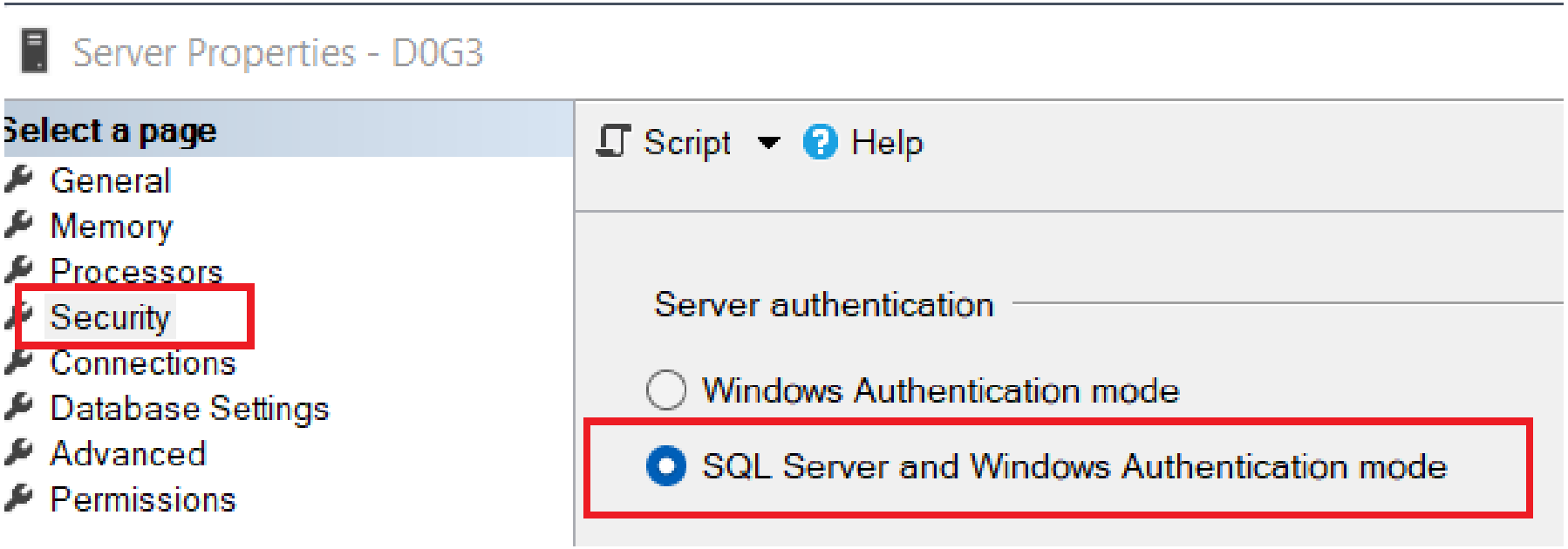
- Authorization: The granting of a right or privilege that enables a user to have legitimate access to a system or a system's object.

Authentication vs Authorization

Authentication	**Authorization**
Determines whether users are who they claim to be	Determines what users can and cannot access
Implemented using several types/techniques – single factor (password) , 2FA, SSO, MFA - Challenges the user to validate credentials (for example, through passwords, answers to security questions, or facial recognition)	Implemented through policies and rules - by configuring id, object that the id can access and actions that the id can perform on the action
Usually done before authorization	Usually done after successful authentication
Example: Employees in a company are required to authenticate through the network before accessing their company resources such as email, database, application etc	Example: After an employee successfully authenticates, the system determines what information the employees are allowed to access

Pre-requisite: Enable Mixed Model Authentication

When you install/setup SQL Server, you will be prompted to choose if you want to enable Windows Integrated login or Mixed Mode (Windows + SQL)



Create SQL Login for Authentication

PatientNo	PatientName	Address	Telephone	DOB	Gender	RegistrationDate	MaritalStatus
Pat100	John	KL	0112345678	1995-05-11	Male	2022-06-18	Married
Pat200	Mary	Ipoh	0123456789	1998-02-04	Female	2021-05-14	Single

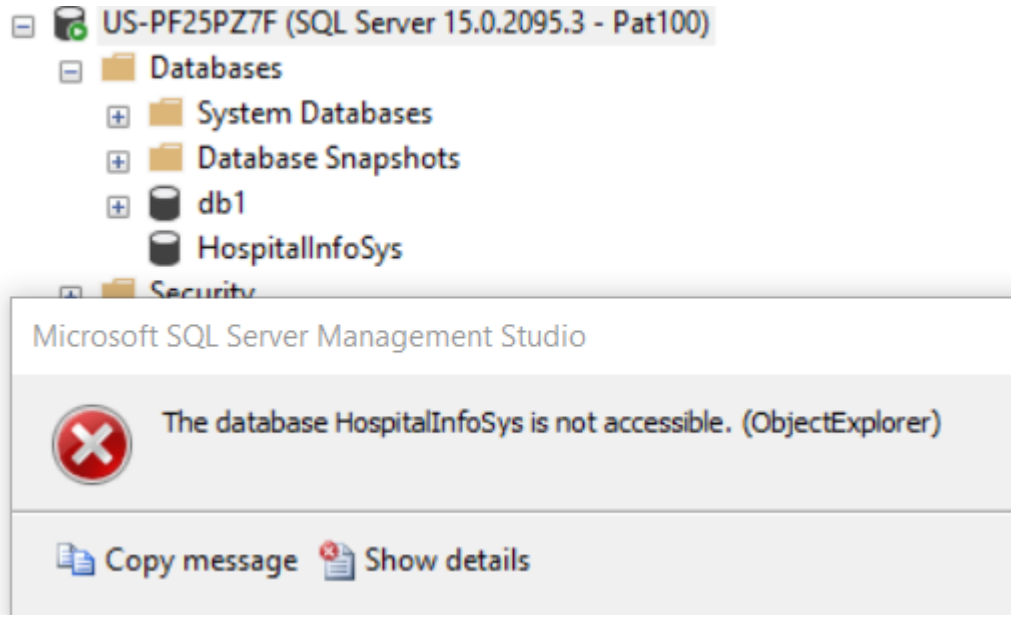
```
Create Login Pat100
With Password = 'QwErTy12345!@#$%'
```

```
Select [LoginName], * From SysLogins Where
[LoginName] = 'Pat100'
```

Newly Created Login (Pat200)

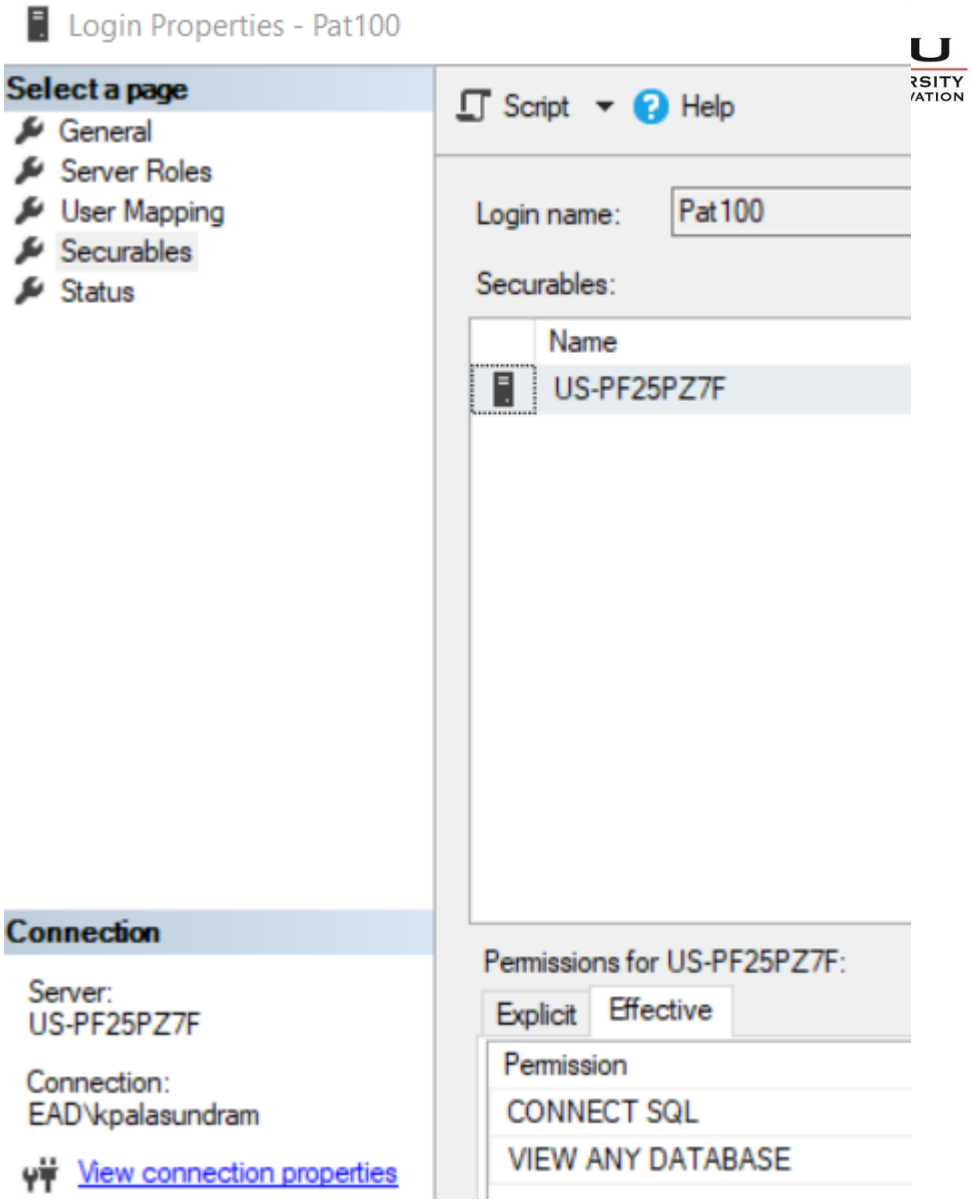
- Can login to server and view the databases

- but not able to access the database – Why ?



Authenticated but Not Authorized

Pat200 was given **authentication** permission to login to the server but not given the **authorization** to access the database



SQL Login vs SQL User

- SQL Login
  - Authentication
  - Created at server/instance level
- SQL User
  - Authorization
  - Created on each database

SQL User

- To enable a SQL Login to access a database (in other words to grant authorization to a SQL login), we need to
  - Create a SQL User in that database for the SQL Login
  - And perform some additional statements to grant access accordingly

Create SQL User for Authorization

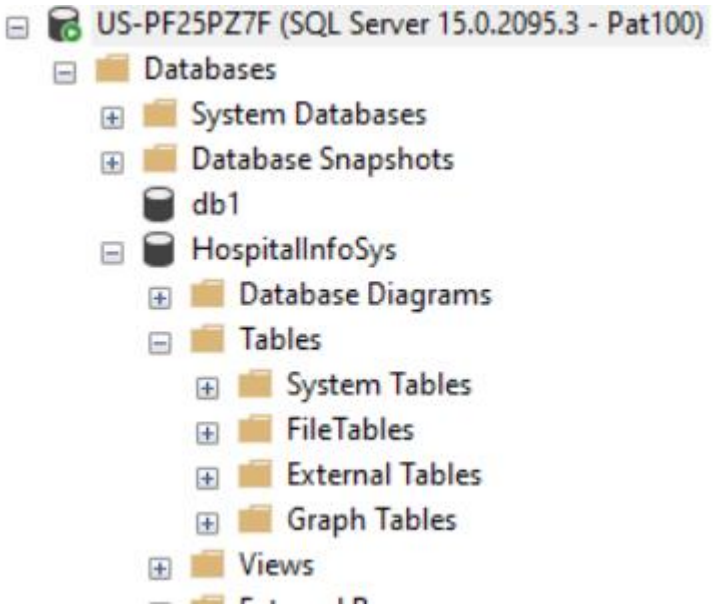
PatientNo	PatientName	Address	Telephone	DOB	Gender	RegistrationDate	MaritalStatus
Pat100	John	KL	0112345678	1995-05-11	Male	2022-06-18	Married
Pat200	Mary	Ipoh	0123456789	1998-02-04	Female	2021-05-14	Single

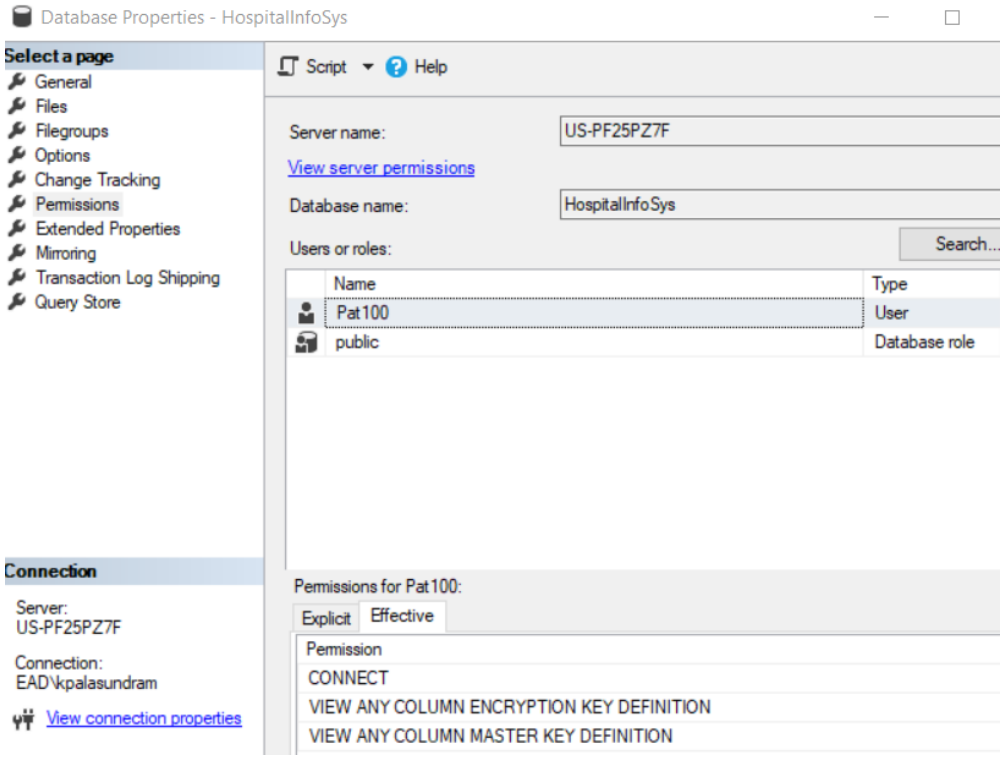
```
Use [HospitalInfoSys]
Create User [Pat100] For Login [Pat100]
Go

Select [name],* From sys.sysusers Where [Name] =
'Pat100'
```

Pat200 Authorization – Step 1

- Pat200 can now access the database but cannot view the tables yet





Create SQL User Without Login

Besides the common way of creating SQL Login and then SQL user, we can also cut short the steps by creating only the SQL user without creating the SQL Login. **This is usually done for testing purposes.**

<u>Without Login</u>

Create User [Pat200] Without Login

Pat200

- Even if Pat200 user knows the table name, he/she still cannot view the contents because no permission (authorization) was given

```
Execute as User = 'Pat100'
Select * From Patient
```

Messages

```
Msg 229, Level 14, State 5, Line 49
The SELECT permission was denied on the object 'Patient',
database 'HospitalInfoSys', schema 'dbo'.
```

Note: Each DB user must be **authorized** to view and/or manipulate (adding, removing / updating) the database contents such as tables, views and columns accordingly

Authorize Pat200 to view the contents of the Patient Table

```
GRANT SELECT ON Tutorial.dbo.Country to Pat200

GRANT SELECT ON Tutorial to Pat200
```

Grant – SQL key word which means give authorization

Select – refers to ability to read data

`Tutorial` or **`Tutorial.dbo.Country`** – refers to the object that is being granted upon

Pat200 – refers to subject (account that is being granted permission)

Table Properties - Patient

Select a page

- General
- Permissions
- Change Tracking
- Storage
- Security Predicates
- Extended Properties

Script Help

Schema: dbo

View schema permissions

Table name: Patient

Users or roles:

Name
Pat100

Connection

Server:
US-PF25PZ7F

Connection:
EAD\kpalasundram

View connection properties

Permissions for Pat100:

Explicit Effective

Permission	Grantor	Grant
References		☐
Select		☐
Select	dbo	☑

Authorization Table

- Permission requirements are captured in **Authorization Table**
- Authorization table specifies the users and the exact permission that is given to the users
- Sample:

Account	Type	Object(s)	Privilege(s)
Sam	SQL User	Table: Employee	Read, Add, Modify, Remove
Bill	SQL User	Table: Employee	Read
Bill	SQL User	Table: Orders	Read
Clerks	SQL Role	Columns: Employee(Col1, Col2)	Read
Administrators	SQL Role	Database: APU	Control, Grant

User and Role

- There are 2 types of SQL accounts
  - User – actual account that can login to a system
  - Role
    - just a grouping of users for easier and efficient permission management
    - *RBAC – role based access control*
    - **Note**: Using roles is the best practice for user management. We should NOT grant permission directly for a user.

Creating Role and Adding Users

- Similar to user accounts, roles must also be created and granted permission. However, you cannot login using a Role. It is a just a grouping name.

```
CREATE ROLE [Patients];
GO

GRANT SELECT ON HospitalInfoSys.dbo.Patient to Patients
Go

ALTER ROLE [Patients] ADD MEMBER Pat100
GO
```

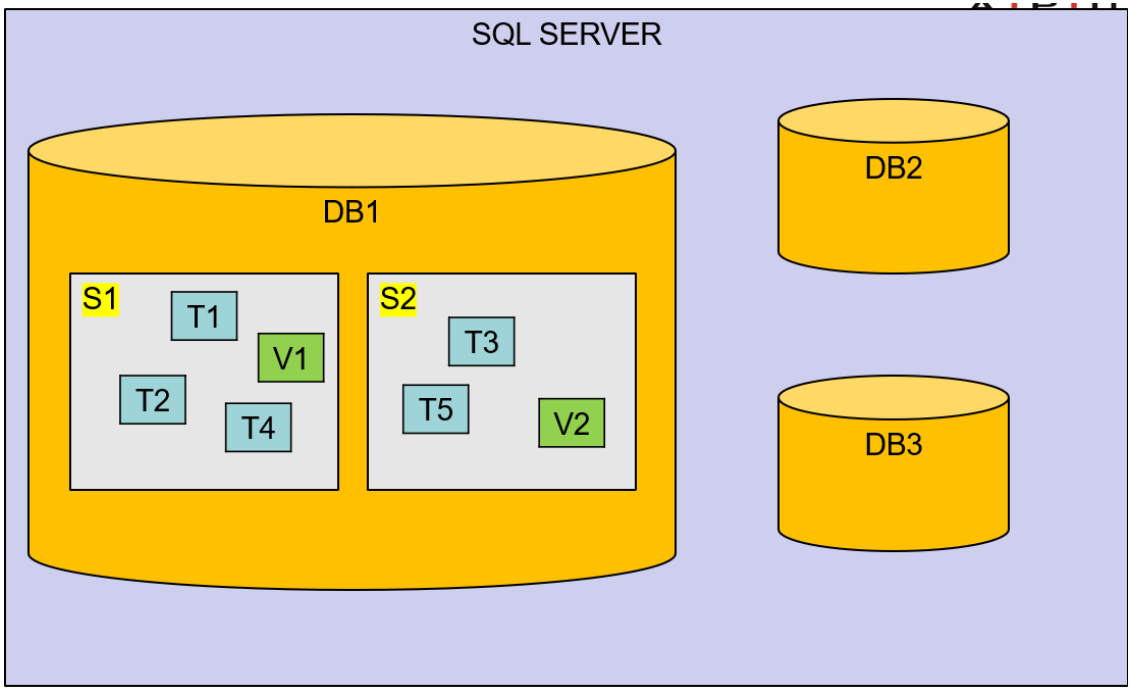
Privileges

Privileges refers to what the user can do in the data

Privileges	SQL keyword
Read	Select
Add data	Insert
Remove data	Delete
Modify data	Update
Allow user to share his/her permission with another user	Grant
Read, add, remove, modify, references	Control

Levels of Permission Settings

- Database
- Schema
- Table, View
- Column



Recap: Permission Management Using SQL Commands

- Create Login
- Create DB User
- Granting Permission

<u>RBAC – Role Based Access Control</u>

- Creating Custom Role
- Adding User to Role

Check Role Member

```
SELECT roles.[name] as role_name, members.[name] as user_name
FROM sys.database_role_members
INNER JOIN sys.database_principals roles
ON database_role_members.role_principal_id = roles.principal_id
INNER JOIN sys.database_principals members
ON database_role_members.member_principal_id = members.principal_id
WHERE roles.name = ‘Patients'
```

Authorization Rules

Commands to create and revoke rules

- GRANT and DENY create a permission rule
- REVOKE removes a permission rule

Note: DENY blocks access. DENY trumps all other access. If a user has both a GRANT and a DENY on a given object, by whatever means, the DENY will take effect.

Authorization rules take into accounts a THREE concepts:

1. **Users:** Individuals who perform some activity on the database.
   - permission settings - users or roles
2. **Objects:** Database units that require authorization in order to manipulate, eg. DB, Table, Columns, View
3. **Privileges:** Any action that might be performed on an object by a user. Eg. Read, Update, Insert, Delete.

Objects and Privileges

- Database: BACKUP DATABASE, CREATE TABLE, and CREATE VIEW

- Table/View: DELETE, INSERT, REFERENCES, SELECT, UPDATE, ALTER, CONTROL

Permission Action Types

- **Select:** Grants user the ability to perform Select operations on the table.
- **Insert:** Grants user the ability to perform the insert operations on the table.
- **Update**: Grants user the ability to perform the update operations on the table.
- **Delete:** Grants user the ability to perform the delete operations on the table.

Permission Action Types

- **Alter:** Grants user permission to alter the table definitions.
- **References**: References permission is needed to create a Foreign key constraint on a table.
- **Control**: Grants SELECT, INSERT, UPDATE, DELETE, and REFERENCES permission to the User on the table.

Some Granting Permissions Examples

- To User

GRANT SELECT ON DATABASE:: [APU Database] TO Test2

GRANT CONTROL ON SCHEMA:: [BookStore] TO Test3

GRANT INSERT, UPDATE, DELETE ON [APU Database].BookStore.Book TO Test4

GRANT SELECT ON [APU Database].BookStore.Book(Category) TO Test5

- To Role

GRANT SELECT ON DATABASE:: [APU Database] TO Testers

Revoking Permission

- REVOKE is used to revoke authorization from subjects.

  REVOKE action1, action2, ….
  ON object1, object2, ….
  FROM subject1, subject2, ….

- If John leaves the company, then we should revoke his permissions

  REVOKE SELECT, INSERT, UPDATE
  ON Employee FROM John;

Revoking Examples

```
REVOKE CONTROL ON SCHEMA:: [BookStore] FROM Test3
```

Denying Permissions Examples

```
GRANT SELECT ON DATABASE:: [APU Database] TO Test2
DENY SELECT ON BookStore.Member To Test2

execute as user = 'Test2'
Select * from BookStore.Book
Select * From BookStore.Member
Revert;
```

We can use DENY to manage permission is a more granular level such as selectively deny to certain tables

Row Level Security (RLS)

PatientNo	PatientName	Address	Telephone	DOB	Gender	RegistrationDate	MaritalStatus
Pat100	John	KL	0112345678	1995-05-11	Male	2022-06-18	Married
Pat200	Mary	Ipoh	0123456789	1998-02-04	Female	2021-05-14	Single

- We need to let each patient to see his/her own details only and not other patient's details.
- To achieve this, we need to create row level security (RLS) for this table.
- RLS enables implementation of more granular access control on table **data row level.**
- RLS is implemented in conjunction with
  - Security Policy
  - Execute As command

Implement RLS - Step 1

- Create a function that checks the current user and returns true (1) if the row value matches the current user

```
CREATE SCHEMA Security;
GO
CREATE FUNCTION Security.fn_securitypredicate
       (@UserName AS nvarchar(100))
RETURNS TABLE
WITH SCHEMABINDING
AS
   RETURN SELECT 1 AS fn_securitypredicate_result
   WHERE @UserName = USER_NAME()
       OR USER_NAME() = 'dbo';
GO
```

Implement RLS - Step 2

- Create a security policy for the table

```
CREATE SECURITY POLICY [DemoSecurityPolicy]
ADD FILTER PREDICATE
[Security].[fn_securitypredicate]([PatientName])
ON [dbo].[Patient]
```

Testing RLS

```
Execute as User = 'Pat100'
Select * From Patient
Revert
```

PatientNo	PatientName	Address	Telephone	DOB	Gender	RegistrationDate	MaritalStatus
Pat100	John	KL	0112345678	1995-05-11	Male	2022-06-18	Married

Permission on View

- Let’s say instead of granting permission directly to the table, we can also grant permission to a view
- The method to grant permission to a view is the same as to granting permission to a table

Checking Database Level Permissions

```
SELECT dp.NAME      AS principal_name,
       dp.TYPE_DESC AS principal_type_desc,
       o.NAME       AS object_name,
       o.type_descas object_type,
       p.PERMISSION_NAME,
       p.STATE_DESC AS permission_state_desc
FROM sys.database_permissions p
    LEFT OUTER JOIN sys.all_objects o
        ON p.MAJOR_ID = o.OBJECT_ID
    INNER JOIN sys.database_principals dp
        ON p.GRANTEE_PRINCIPAL_ID = dp.PRINCIPAL_ID
and dp.is_fixed_role=0
and dp.Name NOT in ('public','dbo')
```

Database Level Permissions

principal_name	principal_type_desc	object_name	object_type	PERMISSION_NAME	permission_state_desc
AppUser	SQL_USER	NULL	NULL	CONNECT	GRANT
dbuser_john	SQL_USER	NULL	NULL	CONNECT	GRANT
dbuser_sara	SQL_USER	NULL	NULL	CONNECT	GRANT
dbuser_raja	SQL_USER	NULL	NULL	CONNECT	GRANT
dbuser_chong	SQL_USER	NULL	NULL	CONNECT	GRANT
dbuser_john	SQL_USER	StudentDetails	USER_TABLE	SELECT	GRANT
dbuser_sara	SQL_USER	StudentDetails	USER_TABLE	SELECT	GRANT
dbuser_raja	SQL_USER	StudentDetails	USER_TABLE	SELECT	GRANT
crole1	DATABASE_ROLE	StudentDetails	USER_TABLE	SELECT	GRANT
dbuser_raja	SQL_USER	TEST1	VIEW	SELECT	GRANT
dbuser_raja	SQL_USER	RestrictedColumns	VIEW	SELECT	GRANT
crole1	DATABASE_ROLE	RestrictedColumns	VIEW	SELECT	GRANT

Checking Affective Permission For a User

```sql
GRANT CONTROL ON SCHEMA:: [BookStore] TO Test3
execute as user = 'Test3'
select * from sys.fn_my_permissions('BookStore','schema')
revert;
```

7 %

Results | Messages

	entity_name	subentity_name	permission_name
	BookStore		SELECT
2	BookStore		INSERT
3	BookStore		UPDATE
4	BookStore		DELETE
5	BookStore		REFERENCES
6	BookStore		EXECUTE
7	BookStore		CREATE SEQUENCE
8	BookStore		VIEW CHANGE TRACKING
9	BookStore		VIEW DEFINITION
10	BookStore		ALTER
11	BookStore		TAKE OWNERSHIP
12	BookStore		CONTROL

Checking Affective Permission For a User

```
Execute as User = 'Pat100'
Select * from sys.fn_my_permissions('Patient','object')
Revert
```

entity_name	subentity_name	permission_name
Patient		SELECT
Patient	PatientNo	SELECT
Patient	PatientName	SELECT
Patient	Address	SELECT
Patient	Telephone	SELECT
Patient	DOB	SELECT
Patient	Gender	SELECT
Patient	RegistrationDate	SELECT
Patient	MaritalStatus	SELECT

Disabling/Removing SQL Login or User

SQL Login

- Alter login [dblogin1] disable
- Drop login [dblogin1]

SQL User

- Deny connect to dbuser_sara
- Revoke connect to dbuser_sara
- Drop user dbuser_sara

Q & A