1. Database Security

1. 1. General Overview

Today's organizations rely on database systems as the key data management technology for a large variety of tasks ranging from regular business operations to critical decision making. The information in the databases is used, shared, and accessed by various users. It needs to be protected and managed because any changes to the database can affect it or other databases.

The main role of a security system is to preserve integrity of an operational system by enforcing a security policy that is defined by a security model. These security models are the basic theoretical tools to start with when developing a security system.

Database security models include the following elements:

- Subject: Individual who performs some activity on the database. It can be a user or process.
- Object: Database unit that requires authorization in order to manipulate. It is a resource or data in a system.
- Access mode/action: Any activity that might be performed on an object by a subject
- Authorization: Specification of access modes for each subject on each object
- Administrative rights: Who has rights in system administration and what responsibilities administrators have
- Policies: Enterprise-wide accepted security rules
- Constraint: A more specific rule regarding an aspect of an object and action

1. 2. Database Security Approaches

A typical DBMS supports basic approaches of data security—discretionary control, mandatory control, and role-based access control.

Discretionary Control:

Discretionary access control (DAC) is an identity-based access control model that provides users a certain amount of control over their data. Data owners (or any users authorized to control data) can define access permissions for specific users or groups of users.

Access permissions for each piece of data are stored in an access-control list (ACL). This list can be generated automatically when a user grants access to somebody or can be created by an administrator. An ACL includes users and groups that might access data and levels of access they might have. An ACL can also be enforced by a system administrator. In this case, the ACL acts as a security policy and regular users can't edit or overrule it.

A given user typically has different access rights, also known as privileges, for different objects. For discretionary access control, we need a language to support the definition of rights—for example, SQL.

Gaining access in the DAC model works like this:

- User 1 creates a file and becomes its owner or obtains access rights to an existing file.
- User 2 requests access to this file.
- User 1 grants access at its own discretion. However, user 1 can't grant access rights that exceed their own. For example, if user 1 can only read a document, they can't allow user 2 to edit it.

If there's no contradiction between the ACL created by an administrator and the decision made by user 1, access is granted.

Discretionary access control is quite a popular model because it allows a lot of freedom for users and doesn't cause administrative overhead.

Mandatory Control:

Each data object is labelled with a certain classification level, and a given object can be accessed only by a user with a sufficient clearance level. Mandatory access control is applicable to the databases in which data has a rather static or rigid classification structure.

Mandatory access control is a method of limiting access to resources based on the sensitivity of the information that the resource contains and the authorization of the user to access information with that level of sensitivity.

Mandatory Access Control uses a hierarchical approach: Each object in a file system is assigned a security level, based on the sensitivity of the data. When a user tries to access a resource, the system automatically checks whether or not they are allowed access.

In both discretionary and mandatory control cases, the unit of data and the data object to be protected can range from the entire database to a specific tuple.

Example of mandatory model can be when the operating system provides users with access based on data confidentiality and user clearance levels. Here, the administrator may assign each subject (user or resource that accesses data) and object (file, database, port, etc.) a set of attributes. In this case, data objects with top-secrete confidentiality level will be accessed by users with top-secrete clearance level.

Role-Based Access Control (RBAC):

Permissions are associated with roles, and users are made members of appropriate roles. However, a role brings together a set of users on one side and a set of permissions on the other, whereas user groups are typically defined as a set of users only.

Role-based security provides the flexibility to define permissions at a high level of granularity in Microsoft SQL, thus greatly reducing the attack surface area of the database system.

1. 3. SQL Server Security

SQL Server has many powerful features for security and protecting data, but planning and effort are required to properly implement them.

Security can be one of the most complex issues when managing a SQL Server instance, yet it's also one of the most important, especially when sensitive and personal data are on the line.

SQL Server includes a variety of tools for protecting data from theft, destruction, and other types of malicious behaviour.

1.3.1. SQL Server Authentication and Authorization

Protecting data starts with the ability to authenticate users and authorize their access to specific data. To this end, SQL Server includes an authentication mechanism for verifying the identities of users trying to connect to a SQL Server instance, as well as an authorization mechanism that determines which data resources that authorized users can access and what actions they can take.

SQL Server supports two authentication modes: Windows Authentication, sometimes referred to as integrated security, and SQL Server and Windows Authentication, sometimes referred to as mixed mode.

Windows authentication is integrated with Windows user and group accounts, making it possible to use a local or domain Windows account to log into SQL Server. When a Windows user connects to a SQL Server instance, the database engine validates the login credentials against the Windows principal token, eliminating the need for separate SQL Server credentials.

In some cases, however, you might require SQL Server Authentication. For example, users might connect from non-trusted domains, or the server on which SQL Server is hosted is not part of a domain, in which case, you can use the login mechanisms built into SQL Server, without linking to Windows accounts.

1.3.2. Principals, Securables, and Permissions

Authentication and authorization are achieved in SQL Server through a combination of security principals, securables, and permissions. They must be configured to enable users to access the data they need, while preventing unauthorized users from accessing data they shouldn't. These components are used for controlling which users can log onto SQL Server, what data they can access, and which operations they can carry out.

You can view and work with principals, securables, and permissions through SQL Server Management Studio (SSMS).

Principals:

Principals are individuals, groups, or processes that are granted access to the SQL Server instance, either at the server level or database level. Server-level principals include logins and server roles, which are listed in the Logins and Server Roles subfolders in the Security folder:

- A login is an individual user account for logging into the SQL Server instance. A login can be a local or domain Windows account or a SQL Server account. You can assign server-level permissions to a login, such as granting a user the ability to create databases or logins.
- A server role is a group of users that share a common set of server-level permissions. SQL Server supports fixed server roles and user-defined server roles. You can assign logins to a fixed server role, but you cannot change its permissions. You can do both with a user-defined server role.

Database-level principals include users and database roles, which are listed in the Users and Roles subfolders in the database's Security folder:

- A database user is an individual user account for logging into a specific database. The database user commonly maps to a corresponding server login in order to provide access to the SQL Server instance as well as the data itself. However, you can create database of any logins, which can be useful for developing and testing data-driven applications, as well as for implementing contained databases.
- A database role is a group of users that share a common set of database-level permissions. As with server roles, SQL Server supports both fixed and user-defined database roles.

Securables:

For each security principal, you can grant rights that allow that principal to access or modify a set of securables. Securables are the objects that make up the database and server environment. They can include anything from functions to database users to endpoints. SQL Server scopes the objects hierarchically at the server, database and schema levels:

- Server-level securables include databases as well as objects such as logins, server roles, and availability groups.
- Database-level securables include schemas as well as objects such as database users, database roles, and full-text catalogs.
- Schema-level securables include objects such as tables, views, functions, and stored procedures.

Permissions:

Permissions define the level of access permitted to principals on specific securables. You can grant or deny permissions to securables at the server, database, or schema level. The permissions you grant at a higher level of the hierarchy can also apply to the children objects, unless you specifically override the permissions at the lower level.

For example, if you grant the **`SELECT`** permission to **`user1`** principal on the **`Registrar`** database, the user will be able to query all table data in the database. However, if you then deny the **`SELECT`** permission on the **`Student`** table, the user will not be able to query that table, but will still be able to access the other tables in that database.

You can use SSMS or T-SQL to view the permissions that have been explicitly granted to a user on a securable.

In most of the cases, you can perform all of the following, or partly:

1. At the server level, create a login for each user that should be able to log into SQL Server. You can create Windows authentication logins that are associated with Windows user or group accounts, or you can create SQL Server authentication logins that are specific to that instance of SQL Server.
2. Create user-defined server roles if the fixed server roles do not meet your configuration requirements. (must be newer version of sql server)
3. Assign logins to the appropriate server roles (either fixed or user-defined).
4. For each applicable server-level securable, grant or deny permissions to the logins and server roles.
5. At the database level, create a database user for each login. A database user can be associated with only one server login. You can also create database users that are not associated with logins.
6. Create user-defined database roles if the fixed database roles do not meet your configuration requirements.
7. Assign users to the appropriate database roles (either fixed or user-defined).
8. For each applicable database-level or schema-level securable, grant or deny permissions to the database users and roles.

You do not need to follow these steps in the exact order. You might grant permissions to server logins or database users when you create them, or you might create server roles and database roles before creating the logins or users.

1. 4. Working with Principals

How to create Logins using T-SQL:

Syntax with Example:

```
CREATE LOGIN [mypc\xyz] FROM Windows
```

The above code creates a server login named 'mypc\xyz', where 'mypc' is the name of the computer and 'xyz' is an existing windows user account.

To define a SQL Server login (one that is not associated with a Windows account), in which case, do not include the FROM WINDOWS clause. However, you must include a WITH clause that specifies a password like in the following:

Syntax with Example:

```
CREATE LOGIN xxx WITH PASSWORD='any_password'
     DEFAULT_DATABASE = master, DEFAULT_LANGUAGE = us_english;
GO
```

1. 5. Creating Server Roles

You can create server role, grant permissions to the role, and then add logins—or you can add the logins first and then grant the permissions.

Syntax with Example:

```
CREATE SERVER ROLE manager
GRANT ALTER ANY DATABASE TO manager
ALTER SERVER ROLE manager ADD MEMBER [mypc\xyz];
WITH DEFAULT_DATABASE = master, DEFAULT_LANGUAGE = us_english;
```

1. 6. Creating Database User

You can create database users having the same name as their Logins or giving them a different name:

Syntax with Example:

```
USE your_database;
GO
CREATE USER [mypc\xyz]  --same name as the login [mypc\xyz]
                                OR
CREATE USER NEW_NAME FOR LOGIN [mypc\xyz] --- different name
```

To create database user without login:

Syntax with Example:

```
USE your_databse;
GO
CREATE USER abcd WITHOUT login
```

1. 7. Creating Database Role

A database role is a group of users that share a common set of database-level permissions.

SQL Server supports both fixed and user-defined database roles. To set up a user-defined database role, you must create the role, grant permissions to the role, and add members to the role (or add members and then grant per

Exa

Syntax with Example:

```
USE your_databse;

GO

CREATE ROLE abcd_role

GRANT SELECT ON DATABASE::your DB name TO abcd role
```