Permissions (GRANT , DENY , REVOKE)

Permissions

- SQL Server provides the ability to grant users the access they need at the level they need it. The GRANT, DENY, and REVOKE statements—along with the wide assortments of permissions (230 in SQL Server 2016 and 237 in SQL Server 2017)
- SQL Server actually provides three T-SQL statements for working with permissions:
  - Use a GRANT statement to enable principals to access specific securables.
  - Use a DENY statement to prevent principals from accessing specific securables. A DENY statement overrides any granted permissions.
  - Use a REVOKE statement to remove permissions that have been granted to principals on specific securables.
- Permissions are cumulative in that the user receives all permissions granted specifically to the database user as well as to its associated login.
- if the user has been assigned to a database role or if the login has been assigned to a server role, the user receives the role permissions as well.
- Permissions are also transitive, based on the hierarchical nature of the server, database, and schema securables.
- For example, if you grant the UPDATE permission to a user for a specific database, the user will also be granted the UPDATE permission on all schemas and schema objects such as tables and views.

GRANT Permissions

- Some permissions are covering, that is, they include multiple permissions under a single name.
- A good example of this is the CONTROL permission, which includes such permissions as INSERT, UPDATE, DELETE, EXECUTE, and several others.
- A good example of this is the CONTROL permission, which includes such permissions as INSERT, UPDATE, DELETE, EXECUTE, and several others.
- GRANT CONTROL ON SCHEMA::Sales TO sqluser01;
- After granting the CONTROL permission, you can again use the fn_my_permissions function to view the effective permissions for that user on the Sales schema:
- The SELECT permission was granted to the user at the database level through the dbdev role and at the schema level as part of the CONTROL permission, as are the rest of the permissions shown in the results.
- Covering permissions help simplify the process of granting access to the database objects.

DENY Permissions

- We can also deny permissions on securables.
- This can be useful when you want to grant permissions at a higher level in the object hierarchy but want to prevent those permissions from extending to a few of the child objects.
- For example, you can deny the CONTROL permission to sqluser01 on an individual table within the Sales schema, as shown in the following example:
  - DENY CONTROL ON OBJECT::Sales.BuyingGroups TO sqluser01;
- When you deny the CONTROL permission, you deny all permissions that are part of CONTROL, including the SELECT permission.
- The SELECT statement returns an empty result set, indicating that sqluser01 no longer has any type of permissions on the BuyingGroups table.
- The DENY permission takes precedence over all granted permissions, no matter where in the object hierarchy permissions are granted or denied.
- However, denying permissions on one object does not impact other objects unless they’re child objects.

DENY Permissions

- But, if you deny permissions on an object that contains child objects, the permissions are also denied on the child objects.
- The following DENY statement denies sqluser01 the ALTER permission on the Sales schema:
- DENY ALTER ON SCHEMA::Sales TO sqluser01;
- If you now run the following SELECT statement, you’ll find that the ALTER permission is no longer granted at the Sales schema:

REVOKE Permissions

- In some cases, you will need to roll back the permissions that have been granted on an executable, in which case, you can use the REVOKE statement.
- The following REVOKE statement removes the CONTROL permission from the Sales schema for sqluser01:
- REVOKE CONTROL ON SCHEMA::Sales TO sqluser01;
- After revoking the CONTROL permission, you can once again use the fn_my_permissions function to view the effective permissions for that user on the Sales schema:
- When working with permissions, be careful not to confuse the DENY statement with the REVOKE statement.
- You could end up unintended consequences when users receive permissions from multiple sources, as in the examples above.
- For example, if you had denied sqluser01 the CONTROL permission to the Sales schema, rather than revoke the permission, the user would no longer have SELECT permissions to the schema and its objects.