More Advanced Topics on SQL SERVER

Views in SQL server

- In SQL, a view is a virtual table or a stored query based on the result-set of an SQL statement.
- Are Ways to simplify tables appropriate for end users
- A view contains rows and columns, just like a real table.
- The fields in a view are fields from one or more real tables in the database.

Views in SQL server…

- The data accessible through a view is not stored in the database as a distinct object. What is stored in the database is a SELECT statement.
- The result set of the SELECT statement forms the virtual table returned by the view.
- A user can use this virtual table by referencing the view name in Transact-SQL statements the same way a table is referenced.

view

- A view is used to do any or all of these functions:
- Restrict a user to specific rows in a table.
- Allow an employee to see only the rows recording his or her work.
- Restrict a user to specific columns.
- For employees who do not work in payroll, do not allow them to see any columns with salary information or personal information.
- Join columns from multiple tables so that they look like a single table.
- Aggregate information instead of supplying details.

For example, present the sum of a column, or the maximum or minimum value from a column.

How to create views

First write your query such as:

- SELECT Sname, Sid, Test1+Test2+mid+Final as Total From Stud Course

Create your view:

- CREATE VIEW View Name AS your query

Now this is saved in the views with name VW Total_Result

- Create view VW Total_Result

  As

SELECT sfname, stud_id, c_name, c_code, Test1+Test2+mid+Final as Total

From course_taken, student s, course c

Where ct.stud_id=s.sidnoand c.c_code=ct.course_code

Adding and Dropping constraints

Data Integrity

- Each of these categories of the data integrity can be enforced by the appropriate constraints.

Microsoft SQL Server supports the following constraints:

- PRIMARY KEY
- UNIQUE
- FOREIGN KEY
- CHECK
- NOT NULL

You can create constraints at the time of Creating a TABLE statement or later on using ALTER TABLE.

Data Integrity….

A PRIMARY KEY constraint:

- is a unique identifier for a row within a database table.
- Every table should have a primary key constraint to uniquely identify each row and only one primary key constraint can be created for each table.
- The primary key constraints are used to enforce entity integrity.

A UNIQUE constraint:

- Enforces the uniqueness of the values in a set of columns, so no duplicate values are entered.
- The primary key and unique key together are called super keys. So super key constraint includes both.
- The super key is applied on a single relation

Data Integrity ….

A FOREIGN KEY constraint:

- prevents any actions that would destroy linkbetween tables with the corresponding data values.
- A foreign key in one table & a primary key in another table.
- The foreign key constraints are used to enforce referential integrity.
- This constraint involve two relations the referencing relation (fk) and the referenced relation (pk).

A CHECK constraint:

- Is used to limit the values that can be placed in a column. The check constraints are used to enforce domain integrity.
- It is also called semantic integrity constraint

Data Integrity

A NOT NULL constraint:

- Enforces that the column will not accept null values. The not null constraints are used to enforce domain integrity, as the check constraints.
- The following example creates a check sal CHECK constraint on teachers table:

Alter table teacher add tsalary money constraint check sal check (tsalary>1000)

- If the field already exists simply add constraint

Alter table teacher add constraint check sal check (tsalary>1000)

Note: To drop the column first drop the constraint

- alter table teacher drop constraint check sal
- alter table teacher drop column tsalary

Add primary key constraint

- You can add constraints to an existing table by using the ALTER TABLE statement.

The following example adds a pk_Teacher primary key constraint on an Teacher table if it was not created earlier:

```
ALTER TABLE Teacher ADD CONSTRAINT pk_Teacher PRIMARY KEY (tidno)
```

Note:

- You can add the primary or unique key constraint into an existing table only when there are no duplicate rows in the table.

Drop constraint

- You can drop constraints in an existing table by using the ALTER TABLE statement.

The following example drops the pk_employee constraint:

Drop constraint

- ALTER TABLE Table_name DROP CONSTRAINT Constraint_name
- ALTER TABLE Teacher DROP CONSTRAINT pk_Teacher24 T.G.

Disable constraint

Check and no check constraint

- Sometimes you need to perform some actions that require the FOREIGN KEY or CHECK constraints be disabled
- You need to disable the constraint by using the ALTER TABLE statement.
- You can re enable the constraints by using the ALTER TABLE statement again.

Disable constraint

- Disable Primary key constraint created earlier

```
ALTER TABLE Teacher NO CHECK CONSTRAINT Pk_TEacher
```

- The following example disables the check sal constraint in the Teacher table and enables this constraint later:

- Disable the check sal constraint in the Teacher table

```
ALTER TABLE Teacher NO CHECK CONSTRAINT check sal
```

- Enable the check sal constraint in the Teacher table

```
ALTER TABLE Teacher CHECK CONSTRAINT check sal
```

Referential constraint -SQL Server Constraints Enhancements

You can control it by using the new

- ON DELETE and
- ON UPDATE

The REFERENCES clause can be on the CREATE TABLE or ALTER TABLE statements.

- ALTER TABLE (this table_name)ADD CONSTRAINT (constraint name)FOREIGN KEY(this field name)REFERENCES primary_table_name (that primary_table_primary_index_field);

Cont…

Dropping foreign key constraint

- ALTER TABLE (this table_name) drop CONSTRAINT (constraint name)

Note do not forget

- primary_table_primary_index_field
- Should be a primary key in that table

Referential integrity

Delete Rules

- Restrict–don't allow delete of "parent" side if related rows exist in "dependent" side. This is the default for SQL server
- Cascade–automatically delete "dependent" side rows that correspond with the "parent" side row to be deleted
- Set-to-Null–set the foreign key in the dependent side to null if deleting from the parent side
- not allowed for weak entities.
- This works on oracle: Restrict, Cascade, Set-to-Null
- SQL-server : Cascade works and restrict is the default

Referential integrity…

- A referential constraint is the rule that the non null values of a foreign key are valid only if they also appear as values of a parent key.
- The table that contains the parent key is called the parent table of the referential constraint
- The table that contains the foreign key is a dependent of that table.

The following example is used to:

- Create the Books and the Authors tables, and
- Create a foreign key constraint which will perform the cascade delete action, therefore, when a row in the Authors table is deleted, the corresponding rows in the Books are also deleted:

Example

Creating relations without relationship

- One way to add referential integrity is at the time of creating tables;

delete the books table and create again

- CREATE TABLE Books (Book ID INT NOT NULL PRIMARY KEY, Author IDINT NOT NULL, Book Name VARCHAR(100) NOT NULL, Price MONEY NOT NULL, **FOREIGN KEY (Author ID) REFERENCES Authors (Author ID));**

Adding Referential Integrity as constraint

Adding Referential Integrity as constraint

```
CREATE TABLE Books
( BookID INT NOT NULL PRIMARY KEY,
  AuthorID INT NOT NULL,
  BookName VARCHAR(100) NOT NULL,
  Price MONEY  NOT NULL)
```

```
CREATE TABLE Authors
( AuthorID INT NOT NULL PRIMARY KEY,
  Name VARCHAR(100) NOT NULL)
```

```
ALTER TABLE Books
ADD CONSTRAINT fk_author
FOREIGN KEY (AuthorID)
REFERENCES Authors  (AuthorID)
```

Adding constraint -ON DELETE CASCADE

```
CREATE TABLE Books
( BookID INT NOT NULL PRIMARY KEY,
  AuthorID INT NOT NULL,
  BookName VARCHAR(100) NOT NULL,
  Price MONEY  NOT NULL)
```

```
CREATE TABLE Authors
( AuthorID INT NOT NULL PRIMARY KEY,
  Name VARCHAR(100) NOT NULL)
```

```
ALTER TABLE Books
ADD CONSTRAINT fk_author
FOREIGN KEY (AuthorID)
REFERENCES Authors  (AuthorID) ON DELETE CASCADE
```

```
ALTER TABLE books DROP
CONSTRAINT fk_author
```

Triggers

Triggers…

- A trigger is a form of a stored procedure that is executed when a specified (Data Manipulation Language) action is performed on a table.
- The trigger can be executed before or after an INSERT, DELETE, or UPDATE.
- Triggers can also be used to check data integrity before an INSERT, DELETE, or UPDATE.
- Triggers can roll back transactions, and they can modify data in one table and read from another table in another database.

Triggers…

- Triggers are Flashing message or data movements while you perform an action on your relation

Can be designed for:

  - Insert
  - Delete
  - Update

- Triggers, for the most part, are very good functions to use;
- However, cause more I/O overhead.

Syntax of triggers

- Create trigger trigger_name On table_name for{insert, update, delete}

As SQL_Statements

Basically, triggers are classified into two main types:

- After Triggers (For Triggers)
- Instead Of Triggers

After Trigger (using FOR/AFTER CLAUSE)

- This type of trigger fires after SQL Server finish the execution of the action successfully that fired it.
- These triggers run after an insert, update or delete on a table. They are not supported for views.

AFTER TRIGGERS can be classified further into three types as:

- AFTER INSERT Trigger.
- AFTER UPDATE Trigger.
- AFTER DELETE Trigger.

```
CREATE TRIGGER Date_insert ON department FOR INSERT
AS PRINT GETDATE()
go
```

After Trigger (using FOR/AFTER CLAUSE)

- **Simple popup trigger message**

```
create trigger check_insert on department  for insert, update
as
begin
print(' one record is inserted to your table')
end
go
```

After Trigger (using FOR/AFTER CLAUSE)

For insert Trigger -shows a record inserted

```
create trigger show_insert
on department
for insert
as
begin
select * from department
where d_number= (select d_number from inserted)
end
Go
```

```
insert into department
values('1017','drawing','8430')
```

Results | Messages

	d_head	d_name	d_number
1	1017	drawing	8430

After Trigger (using FOR/AFTER CLAUSE)

After insert Trigger

```
create trigger check_insert

on department

after insert

as

begin

print(' one record is inserted to your table')

end

go
```

After Trigger (using FOR/AFTER CLAUSE)

After update Trigger

```
CREATE TRIGGER up_record
ON department
after update
AS
begin
select * from department
where d_number=(select d_number from inserted)
end
go
```

After Trigger (using FOR/AFTER CLAUSE)

After Delete Trigger

```
create trigger after_delete

on department

for delete

as

PRINT 'AFTER DELETE TRIGGER fired.'

go
```

```
AFTER DELETE TRIGGER fired.
(1 row(s) affected)
```

- delete from department where d_number=1234

Instead of Trigger (using INSTEAD OF CLAUSE)

- This type of trigger fires before SQL Server starts the execution of the action that fired it.
- This is differ from the AFTER trigger, which fires after the action that caused it to fire.
- We can have an INSTEAD OF insert/update/delete trigger on a table that successfully executed but does not include the actual insert/update/delete to the table.
- These can be used as an interceptor for anything that anyone tried to do on our table or view.

Instead of Trigger (using INSTEAD OF CLAUSE)

- If you define an Instead Of trigger on a table for the Delete operation, they try to delete rows, and they will not actually get deleted (unless you issue another delete instruction from within the trigger)
- INSTEAD OF TRIGGERS can be classified further into three types as:
  - INSTEAD OF INSERT Trigger.
  - INSTEAD OF UPDATE Trigger.
  - INSTEAD OF DELETE Trigger.

Instead of Trigger (using INSTEAD OF CLAUSE)

- These triggers helps to check what will happen without inserting record to relations

```
create trigger show_before_insert
on department
instead of insert, update
as
begin
select * from inserted
end
go
```

```
insert into department
values
('1012','Abebe','222'),
('1013','Alemu','223'),
('1014','Fatuma','224')
```

Results | Messages

	d_head	d_name	d_number
1	1012	Abebe	222
2	1013	Alemu	223
3	1014	Fatuma	224

- Shows the records on a column instead of inserting them on the table

Instead of Trigger (using INSTEAD OF CLAUSE)

- Instead of Delete Trigger

```
CREATE TRIGGER Instead_of_Delete

ON department

INSTEAD OF DELETE

AS

select * from deleted

go
```

```
delete from department
where d_number like '1%'
```

Results | Messages

	d_head	d_name	d_number
1	1011	ICT	101
2	1012	Accounting	102
3	1014	Geography	103
4	1016	Biotechnology	104
5	1018	Mathematics	105
6	1017	Amharic	106

Triggers List

- Sometimes it is difficult to remember all your triggers.
- You can See the list of triggers:
- SELECT * FROM sys.triggers

Stored Procedures

Stored Procedures…

- A stored procedure is nothing more than prepared SQL code that you save so you can reuse the code over and over again.
- In addition to running the same SQL code over and over again you also have the ability to pass parameters to the stored procedure.

How to write Stored procedures

- Create procedure procedure_name as your query

```
CREATE PROCEDURE procedure1 AS
    BEGIN
        SELECT * from Course_Taken
    END
GO
```

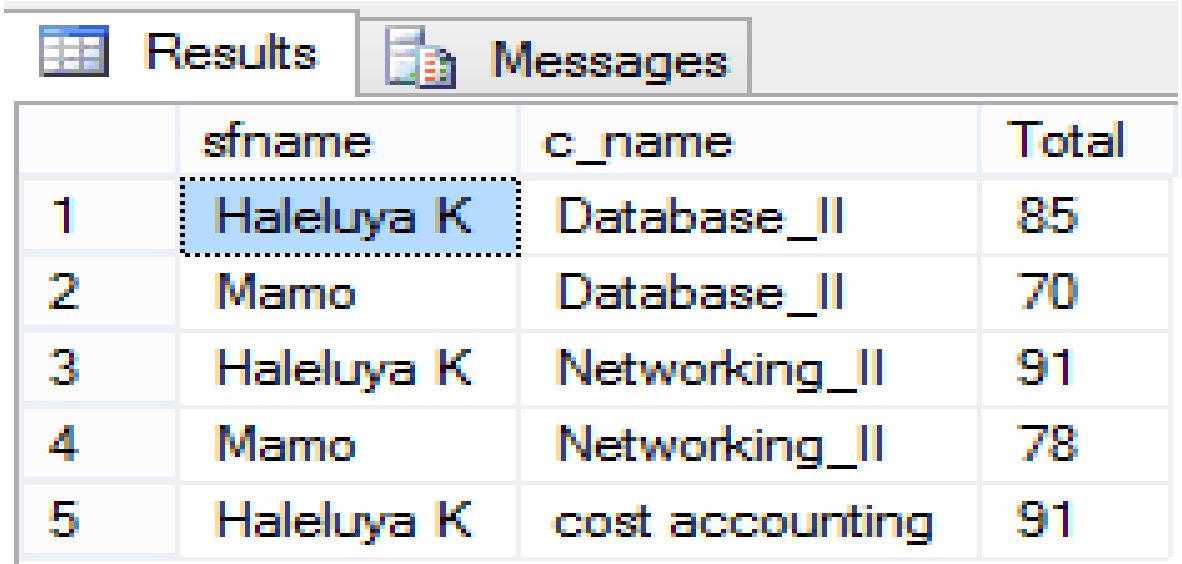
```
CREATE PROC procedure1 AS
   BEGIN
    SELECT * from Course_Taken
    END
GO
```

- To call this stored procedure we would execute it as follows:
  - Execute procedure1
  - Exec procedure1
- How to drop procedures use - Drop procedure procedure_name
  - Drop procedure procedure1

How to write Stored procedures

```
CREATE PROCEDURE top_five
AS
    SELECT TOP 5 sfname, c_name, Total
FROM total_result
```

exec top_five



Results | Messages

	sfname	c_name	Total
1	Haleluya K	Database_II	85
2	Mamo	Database_II	70
3	Haleluya K	Networking_II	91
4	Mamo	Networking_II	78
5	Haleluya K	cost accounting	91

Here is a good summary

A stored procedure:

- accepts parameters
- can NOT be used as building block in a larger query
- can contain several statements, loops, IF ELSE, etc.
- can perform modifications to one or several tables
- can NOT be used as the target of an INSERT, UPDATE or DELETE statement.

- A view:
  - does NOT accept parameters
  - can be used as building block in a larger query
  - can contain only one single SELECT query
  - can NOT perform modifications to any table
  - but can (sometimes) be used as the target of an INSERT, UPDATE or DELETE statement

Database Security

DML*

- SELECT
- DELETE
- UPDATE
- INSERT
- REFERENCES

*Data Manipulation Language

DDL*

- ALTER
- CONTROL
- VIEW DEFINITION
- CREATE
- DROP
- EXECUTE

*Data Definition Language

DCL*

Modify Security

- GRANT
- REVOKE
- DENY
- IMPERSONATE

* Data Controllin Language

Data Control Language (DCL) Statements

- Data Control Language Statements are used to grant privileges on tables, views, sequences, synonyms, procedures to other users or roles.

The DCL statements are:

- GRANT:-Use to grant privileges to other users or roles.
- REVOKE:- Use to take back privileges granted to other users and roles.

Types of Privileges

There are two types of privileges

- SYSTEMPRIVILEGES
- OBJECTPRIVILEGES

1. SYSTEMPRIVILEGES

- System Privileges are normally granted by a DBA to users.
- Examples of system privileges are CREATE SESSION, CREATE TABLE, CREATE USER etc.

Usually about DDL

- Are privileges that do not relate to a specific schema or object.

2. OBJECTPRIVILEGES

- Object privileges means privileges on objects such as tables, views, synonyms, procedure.
- These are granted by owner of the object.

Usually about DML.

- Owner already create an object, he can further decide who can manipulate it.

OBJECTP RIVILEGES…

- The set of actions that a user can carry out against a database object are called the privileges for the object.

The SQL1 standard specifies four basic privileges for tables and views:

- The SELECT privilege
- The INSERT privilege
- The DELETE privilege
- The UPDATE privilege

Creating database Users

To grant a user database privileges:

- First create the login to the user

```
create login login_name with password='password';
```

- Second create the user for the database

```
create user user_name for login login_name;
```

Create the login

- CREATE LOGIN <login name> WITH PASSWORD = ‘<password>’
- CREATE LOGIN Talegeta WITH PASSWORD=‘Geta'

CONT…

Try to create the following logins on SQL server 2012

- Create login user1 with password='12345'
- Create login user2 with password='12345'
- Create login user3 with password='12345'
- Create login user4 with password='12345‘

Cannot find the user ‘user1', because it does not exist or you do not have permission.

Therefore users should be created from logins

CONT…

Create the user for HU database

use HU

go

- create user Tale for login Geta
- create user user1 for login user1
- create user user2 for login user2
- create user user3 for login user3
- create user user4 for login user4

To easily remember you are recommended to make users with similar name for login names

EXAMPLE….

- create login Getaneh with password ='123';
- create user user1 for login Getaneh;

grant select, update,delete on student to user1;

- create login gech with password ='123'
- create user user2 for login gech;

grant select, delete on department to user1;

- create login selamawit with password ='123'
- create user user3 for login selamawit;

grant all on exam to user3;

EXAMPLE…

- create login tsion with password ='123';
- create user user4 for login tsion;

grant select, insert on examroom to user4;

- create login talegeta with password ='123';
- create user user5 for login talegeta;

grant select, update on question to user5;

- create login Zenebech with password ='123'
- create user user6 for login zenebech;

grant select on examin_student to user5,user4,user3,user2;

EXAMPLE…

- create login kassahun with password ='123'
- create user user7 for login kassahun;

grant update, alter, delete on teacher to user7,user5,user4;

- create login gg with password ='123';
- create user user9 for login gg;

grant select, update, delete, insert on teacher to user1,user2,user3,user8,user6;

- create login Haleluya with password ='123'
- create user user8 for login Haleluya;

grant select, update, delete, insert on teacher to user1,user2,user3,user8,user6;

Creating Roles

- Role is a random set of privileges that is granted to users.

There are three types of roles in SQL server:

- Fixed server roles
- Fixed database roles
- User defined database roles

- We cannot create or change server level roles, but it is possible database level role.

Creating Role cont'd

- After you create a database level role, configure the database-level permissions of the role by using GRANT, DENY, and REVOKE.
- Users can be added to a fixed server level role using sp_addsrvrolemember stored procedure.

  Syntax: sp_addsrvrolemember 'role, 'user_nane'

  Example: sp_addsrvrolemember 'dbcreater','u1'
- To add members to a database role, use the sp_addrolemember stored procedure.

Syntax: CREATE ROLE role_name [AUTHORIZATION owner_name]

- Role_name is the name of the role to be created.
- AUTHORIZATION owner_name is the database user or role that is to own the new role.

Example: CREATE ROLE student

Creating Role cont'd

- To add user u1 to be the member of student role,

EXECUTE sp_addrolemember 'student','u1'

- To add user u1 to be the member of fixed database role,

EXECUTE sp_addrolemember

Example: sp_addrolemember 'db_accessadmin','u1'

- We can drop roles using the Drop role role_name code
- We can remove membership from roles using sp_droprolemember stored procedure

Example; sp_droprolemember db_accessadmin, **'u1'**

Grant

SQL Grant command

- SQL Grant command is used to provide access or privileges on the database objects to the users.

The syntax for the GRANT command is:

- GRANT privilege_name ON object_name TO {user_name| PUBLIC | role_name} [with GRANT option];

Here,

- privilege_name: is the access right or privilege granted to the user.
- object_name: is the name of the database object like table, view etc.,.
- user_name: is the name of the user to whom an access right is being granted.

Grant Permission to users

- use HU
- Go

Grant read of student table for user1

- Grant select on student to user1,user2,user3,user4

Grant read of student table for many users

- Grant select on student to user1,user2,user3,user4

Grant read of department, student table for many users

- Grant select on student, department to user1,user2,user3,user4

Grant many privileges on student relation

- Grant update, insert, select on student to user1,user2,user3,user4

Note: if you revoke select and grant update this contradicts

Grant Users

- "Public is used to grant rights to all the users.
- With Grant option: allows users to grant access rights to other "users.
- You can also use the all keyword to indicate that you wish all permissions to be granted.
- You can grant all Permissions at once TO USER1

GRANT ALL (PRIVILEGES) ON courses TO user1;

- Give all users SELECT access to the Student table.

For example:

GRANT SELECT ON STUDENT TO PUBLIC;

Column Privileges

- Let user1 change department names.

```
GRANT UPDATE (d_name) ON department TO user1;
```

- Give Abebe read-only access to the sfname, smname, slname columns of the student table.

```
GRANT SELECT (sfname, smname, slname) ON student TO Abebe;
```

Column level privileges

- As the owner of a table, you can control at column level at which you specify which columns are manipulatable by other schema owners.
- Suppose you want to grant update and insert privilege on only certain columns not on all the columns then include the column names in grant statement.
- For example you want to grant update privilege on sfname column only and insert privilege on smname and slname columns only.

Then give the following statement:

Grant update (sfname),insert (smname, slname)on student to Abebe;

Passing Privileges (GRANT OPTION)

- To grant select statement on student table to Abebe and to make Abebe be able further pass on this privilege you have to give WITH GRANT OPTION clause in GRANT statement like this.

Grant select on student to Abebe with grant option;

- When you create a database object and become its owner, you are the only person who can grant privileges to use the object.
- you may want to allow other users to grant privileges on an object that you own GRANT statement:

GRANT SELECT ON STUDENT TO Abebe;

WITH GRANT OPTION…

- The WITH GRANT OPTION clause privilege the right to grant those privileges to other users.

GRANT SELECT ON STUDENT TO ABEBE WITH GRANT OPTION;

- Abebe can now issue this GRANT statement:

GRANT SELECT ON STUDENT TO USER1;

Revoke

Revoke cont'd

- The revoke command removes user access rights or privileges to the database objects.
- The syntax for the REVOKE command is:
- "REVOKE privilege_name ON object_name FROM {User_name| PUBLIC | Role_name}

For Example:

GRANT SELECT ON Student TO user1

- This command grants a SELECT permission on Student table to user1.

REVOKE SELECT ON Student FROM user1

- This command will revoke a SELECT privilege on Student table from user1.

Revoke cont’d

Revoke from users

- Revoke select on student from user1,user2,user3,user4
- Revoke update on student from user1,user2,user3,user4
- Revoke select, update on student from user4

Revoking Privileges (REVOKE)

In most SQL-based databases, the privileges that you have granted with the GRANT statement can be taken away with the REVOKE statement

- A REVOKE statement may take away all or some of the privileges that you previously granted to a user-id
- Grant and then revoke some STUDENT table privileges.

```
GRANT SELECT, INSERT, UPDATE ON STUDENT TO ABEBE, USER1;

REVOKE INSERT, UPDATE ON STUDENT FROM USER1;
```

- The INSERT and UPDATE privileges on the STUDENT table are first given to the two users and then revoked from one of them. However, the SELECT privilege remains for both user-ids.

Denying Permission

- For example, if you wanted to revoke delete privileges on a table called STUDENT from a user named USER1, you would execute the following statement:

```
Revoke delete on STUDENT from USER1;
```

- If you wanted to revoke all privileges on a table, you could use the all keyword. For example:

```
Revoke all on STUDENT from USER1;
```

- If you had granted privileges to public (all users) and you wanted to revoke these privileges, you could execute the following statement:

```
Revoke all on STUDENT from public;
```

Denying Permission

- Take away all privileges granted earlier on the OFFICES table.

```
REVOKE ALL PRIVILEGES ON OFFICES FROM ARUSER;
```

- Take away UPDATE and DELETE privileges for two user-ids.

```
REVOKE UPDATE, DELETE ON OFFICES FROM ARUSER, OPUSER;
```

- Take away all privileges on the OFFICES table that were formerly granted to all users.

```
REVOKE ALL PRIVILEGES ON OFFICES FROM PUBLIC;
```

- When you issue a REVOKE statement, you can take away only those privileges that you previously granted to another user

Denying Permission…

- DENY revokes a permission so that it cannot be inherited.
- DENY takes precedence over all permissions, except DENY does not apply to object owners or members of sys admin.
- If you DENY permissions on an object to the public role it is denied to all users and roles except for object owners and sys admin members.

How TWO GRANTS CONFLICT

- Suppose that Haleluya, the DBMS teacher, gives Jerry Fitch SELECT privileges for the STUDENT table and SELECT and UPDATE privileges for the COURSE_TAKEN table, using the following statements:

```
GRANT SELECT
ON STUDENT
TO JERRY;
```

```
GRANT SELECT, UPDATE
ON COURSE_TAKEN
TO JERRY;
```

- A few days later KASSAHUN, the OS Teacher, gives Jerry the following privilege:

```
GRANT SELECT, DELETE
ON COURSE_TAKEN
TO JERRY;
```

```
GRANT SELECT
ON STUDENT
TO JERRY;
```

- A few days later Kassahun revokes the privileges he previously granted, What Will Happen?

Do not hesitate to communicate me, if you face any problem on the above SQL server feature