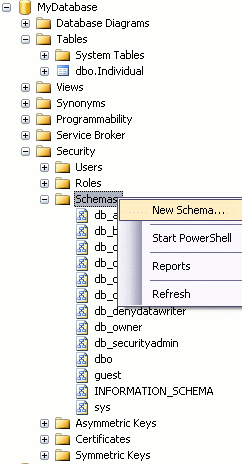
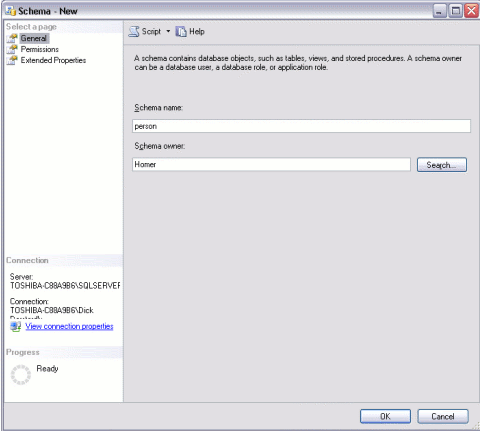
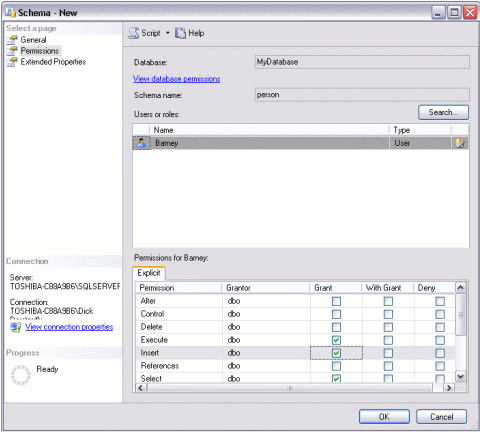
**What is a Database Schema?**

A database schema is a way to logically group objects such as tables, views, stored procedures etc. Think of a schema as a container of objects.

You can assign a user login permissions to a single schema so that the user can only access the objects they are authorized to access.

Schemas can be created and altered in a database, and users can be granted access to a schema. A schema can be owned by any user, and schema ownership is transferable.

Creating a Database Schema

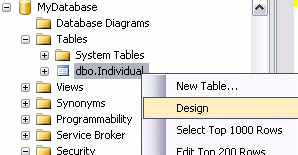
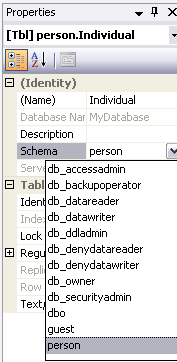
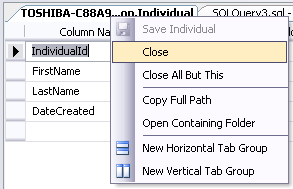
1. Navigate to *Security > Schemas*
2. Right click on *Schemas* and select *New Schema...*. Like this:  
   
3. Complete the details in the *General* tab for the new schema. In this example, the schema name is "person" and the schema owner is "Homer".  
   
4. Add users to the schema as required and set their permissions:  
   
5. Add any extended properties (via the *Extended Properties tab*)
6. Click *OK*.

## Add a Table to the New Schema

Now that we have a new schema, we can add objects such as tables, views, and stored procedures to it. For example, we could transfer the table that we created in the earlier lesson to the new schema.

When we created that table (called "Individual"), it was created in the default database schema ("dbo"). We know this because it appears in our object browser as "dbo.Individual".

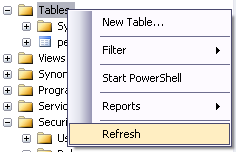
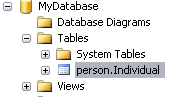
To transfer the "Individual" table to the "person" schema:

1. In Object Explorer, right click on the table name and select "Design":  
   
2. From Design view, press F4 to display the Properties window.
3. From the Properties window, change the schema to the desired schema:  
   
4. Close Design View by right clicking the tab and selecting "Close":  
   
5. Click "OK" when prompted to save

Your table has now been transferred to the "person" schema.

## Confirm your Change

To confirm the change:

1. Refresh the Object Browser view:  
   
2. You will now see that Object Browser displays the new schema for the table (person.Individual):  
   

## Using Transact-SQL

#### To create a schema

1. In **Object Explorer**, connect to an instance of Database Engine.
2. On the Standard bar, click **New Query**.
3. The following example creates a schema named Chains, and then creates a table named Sizes.

CREATE SCHEMA Chains;

GO

CREATE TABLE Chains.Sizes (ChainID int, width dec(10,2));

1. Additional options can be performed in a single statement. The following example creates the schema Sprockets owned by Annik that contains table NineProngs. The statement grants SELECT to Mandar and denies SELECT to Prasanna.

CREATE SCHEMA Sprockets AUTHORIZATION Annik

CREATE TABLE NineProngs (source int, cost int, partnumber int)

GRANT SELECT ON SCHEMA::Sprockets TO Mandar

DENY SELECT ON SCHEMA::Sprockets TO Prasanna;

GO

1. Execute the following statement, to view the schemas in this database:

SELECT \* FROM sys.schemas;

23

You can GRANT schema permissions that are effective for everything existing **and everything that will exist** in that schema.

Grant Schema Permissions

GRANT SELECT, INSERT, UPDATE, DELETE ON SCHEMA :: <schema> TO <user>;

Further to that, if you want to then deny permissions on a certain object within that schema, you can do.

Denying Object Permissions

DENY INSERT ON OBJECT::<schema>.<object> TO <user>;

##### Problem

I have a table where some of the columns should not be queryable by all users. How can I filter the data appropriately so that not everyone can select the data? In a previous tip, Filtering Columns in SQL Server Using Views we looked at using Views. In this tip we cover how this can be done with column-level permissions.

##### Solution

One of the easiest ways to do this is through the use of views. However, if that isn't possible, there is another way: column-level permissions. They are a little harder to see and require a bit more diligence to keep track of, but they work just fine. So what's the difference between column level permissions and, say, table level permissions?

First, let's set up a table to use:

CREATE TABLE dbo.Employee (

EmployeeID INT IDENTITY(1,1),

FirstName VARCHAR(20) NOT NULL,

MiddleName VARCHAR(20) NULL,

SurName VARCHAR(20) NOT NULL,

SSN CHAR(9) NOT NULL,

Salary INT NOT NULL,

CONSTRAINT PK\_Employee PRIMARY KEY (EmployeeID)

);

And we'll go ahead and load it up with a couple of entries for a proof of concept:

INSERT INTO dbo.Employee (FirstName, MiddleName, SurName, SSN, Salary)

VALUES ('John', 'Mark', 'Doe', '111223333', 50000);

INSERT INTO dbo.Employee (FirstName, MiddleName, SurName, SSN, Salary)

VALUES ('Jane', 'Eyre', 'Doe', '222334444', 65000);

Let's go ahead and set up two users and two roles for this demonstration:

CREATE ROLE HR\_Employee;

GO

CREATE ROLE HR\_Intern;

GO

CREATE USER SalaryPerson WITHOUT LOGIN;

GO

EXEC sp\_addrolemember @membername = 'SalaryPerson', @rolename = 'HR\_Employee';

GO

CREATE USER SummerIntern WITHOUT LOGIN;

GO

EXEC sp\_addrolemember @membername = 'SummerIntern', @rolename = 'HR\_Intern';

GO

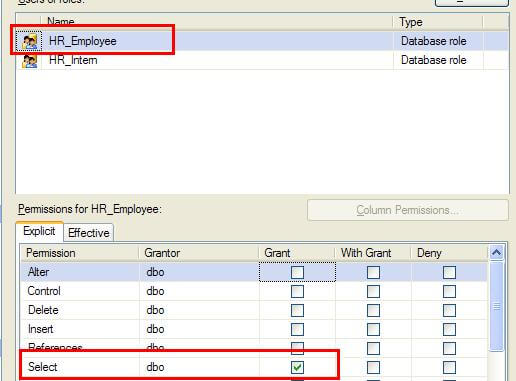
This sets up two levels of users: HR Employees (role HR\_Employee, of which SalaryPerson is one) and HR Interns (role HR\_Intern, played by SummerIntern). Now, when we normally grant permissions, we do so against the whole object or schema. For instance, this grants SELECT permission against the dbo.Employee table to HR\_Employee role members:

GRANT SELECT ON dbo.Employee TO HR\_Employee;

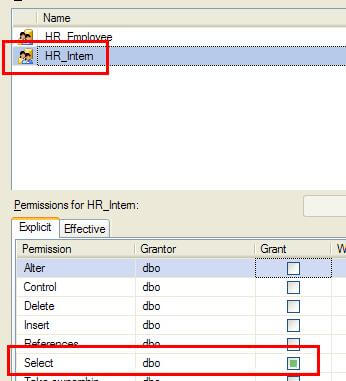
Now, we don't want interns to have this level of permissions. We only want them to have access to specific columns. There's a way to do this. Immediately after the table name, we can specify the columns we want to grant permission to (or DENY, if we needed to do that) within a set of parentheses, like so:

GRANT SELECT ON dbo.Employee (EmployeeID, FirstName, MiddleName, SurName) TO HR\_Intern;

Now, if you prefer the GUI, you can do and see the same thing in SQL Server Management Studio, it just takes a little closer eye. Note the difference between Figure 1 (a checkbox, signifying complete permissions against the table) and Figure 2 (a green square, indicating that there are some permissions, but we'll have to take a closer look).

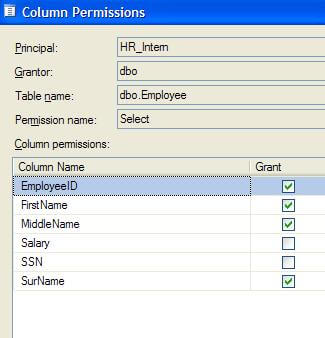


(Figure 1)



(Figure 2)

If we click on the Select row, the button for Column Permissions activates. Clicking on that shows that we do indeed have permissions at the column level. Note there is no checkbox beside SSN nor Salary (Figure 3).



(Figure 3)

Therefore, the HR\_Intern role cannot query these columns. They can find out that they are there, but they can't retrieve data. If you want to see these permissions in action, execute the following snippets. This should work just fine, because HR\_Employees can SELECT against the whole table:

EXECUTE AS USER = 'SalaryPerson';

GO

SELECT \* FROM dbo.Employee;

GO

REVERT;

GO

This will fail with a couple of access denied errors, listing the columns the user cannot access:

EXECUTE AS USER = 'SummerIntern';

GO

SELECT \* FROM dbo.Employee;

GO

REVERT;

GO

The errors you should see:

Msg 230, Level 14, State 1, Line 2

The SELECT permission was denied on the column 'SSN' of the

object 'Employee", database 'MSSQLTips', schema 'dbo'.

Msg 230, Level 14, State 1, Line 2

The SELECT permission was denied on the column 'Salary' of the

object 'Employee", database 'MSSQLTips', schema 'dbo'.

This will work, because the columns in the query are accessible to HR\_Intern:

EXECUTE AS USER = 'SummerIntern';

GO

SELECT EmployeeID, FirstName, SurName FROM dbo.Employee;

GO

REVERT;

GO

And that's how to restrict using column permissions. Incidentally, you can do the same for DENY. Therefore, if a group of users already have access to columns they shouldn't, and you can't rework security in this manner, you could use DENY if you had to, like so:

DENY SELECT ON dbo.Employee (SSN, Salary) TO HR\_Intern;

Since DENY trumps any other permissions, this will effectively block access to those columns. This should be used as a last resort, obviously, because the use of DENY is not intuitive. And DENY at the column level is another step removed from what we're used to when looking at permissions.