DB Security

YOU HAVE THE POWER!

DB Security

Security

- Discretionary Access Control
- Views & Stored Procedures

- Backup and Recovery
- Integrity
- Encryption
- Duplication/Raid

DB Security

DISCRETIONARY ACCESS CONTROL (DAC)

DB Security – Access Control

Access control defines what a user can and cannot do

- Discretionary Access Control (DAC)
  - Involves GRANT and REVOKE of privileges to specific database functions
  - Can be tailored to individual users
  - Can be changes by specific users with sufficient privileges

- Can be different for each attribute, table or even query, view or stored procedure in a database as well as connecting to a database

- System Privilege Keywords
  - GRANT – give access to a database object (table, attribute, view etc.…)
  - DENY – outright prevent access to a database object
  - REVOKE – remove access to a database object

Access Control

Access Control describes what a user can do via the DBMS

1. What resource(s) the user has access to (tables, views, stored procedures)
   - Make sure each user only sees the data they should have access to

2. What
   - Types of
   - Make su

3. If the

Users have specific privileges and can only operate on the data for which they are authorised to access

As the d

assign privileges to other users

- A predefined user represents the DBA and has complete access to all resources
  - "dbo" (Database Owner) for MS-SQL
  - "system" for Oracle

Access Control - Privileges

SQL offers several types of privilege

1. INSERT - `INSERT INTO R VALUES ('x', 'y', 'z')`
2. UPDATE - `UPDATE R SET attr1 = 'x' WHERE pk = 'x'`
3. DELETE- `DELETE FROM R WHERE attr1 = 'x'`
4. SELECT - `SELECT * FROM R WHERE attr1 = 'x'`
5. REFERENCES - to build a referential integrity constraint with the resource (foreign key!) as well as ALTER a table
6. EXECUTE - for stored procedures: `GRANT EXECUTE ON GetCharacters TO testUser`
7. CREATE - `GRANT CREATE TABLE TO testUser     // DROP and ALTER as well` ☺

- These are the basic privileges, others may differ between DBMS
  - Some DBMS even allow authorisation rules to be applied to each specific attribute

Access Control – Obtaining Privileges

The creator of a relation is the **d**ata**b**ase **o**wner

- The **dbo** is a user that has permissions to perform all activities in the database

- The **dbo** can GRANT privileges (permissions) to any other users

```
GRANT <privileges> ON Relations TO Users [WITH GRANT OPTION]
```

Privileges = INSERT, UPDATE, SELECT, DELETE etc..

Users = a list of user logins or roles (eg public) that contain users

Access Control – Obtaining Privileges

Two main methods

1. Granted to users specifically
   - Previous slide
   - Very specific to each user

2. Privileges granted to roles
   - A role is a named group of privileges
   - A user can then be placed into one or more different roles
   - Easier to maintain for large number of similar users

Access Control – GRANT Privileges

Privileges can be **GRANTED** on a per user basis

- To grant a privilege to a user
- GRANT <Privileges | ALL [privileges]> ON Resource to User1, …Usern [with grant option]

```
GRANT SELECT ON Actors TO testUser;

GRANT SELECT, INSERT, UPDATE, DELETE ON Actors TO testUser, mary;

GRANT ALL ON Actors TO testUser; -- Deprecated in SQL Server
```

- With grant option specifies whether the grantee can pass the privilege on to other users:

```
GRANT SELECT, INSERT ON Actors TO testUser WITH GRANT OPTION;
```

Access Control – GRANT Privileges

Privileges can be **GRANTED** on a per user basis

- These privileges can be very specific – they can apply to specific attributes

```
GRANT SELECT ON Actors(actorName) TO testUser;

GRANT UPDATE ON Actors(actorName) TO testUser, mary;
```

This is not so for DELETE/INSERT

```
GRANT DELETE ON Actors(actorName) TO testUser;

GRANT INSERT ON Actors(actorName) TO testUser;
```

Sub-entity lists (such as column or security expressions) cannot be specified for entity-level permissions.

Access Control – Grant Privileges

Privileges can be **GRANTED** to other users

- These privileges can be the same or less

```
GRANT SELECT ON Actors TO testUser WITH GRANT OPTION;

--Run following queries as testUser:
EXECUTE AS USER = 'testUser'

--testUser SQL Query:
GRANT SELECT ON Actors TO testUserNo2

--Or lesser priviledges:
GRANT SELECT ON Actors(actorName) TO testUserNo2
```

Access Control – Revoke Privileges

Privileges can also be **REVOKED** on a per user basis

- To revoke a privilege (permission) from a user
- REVOKE <Privileges | ALL [privileges]> ON Resource to User1, …Usern [CASCADE]

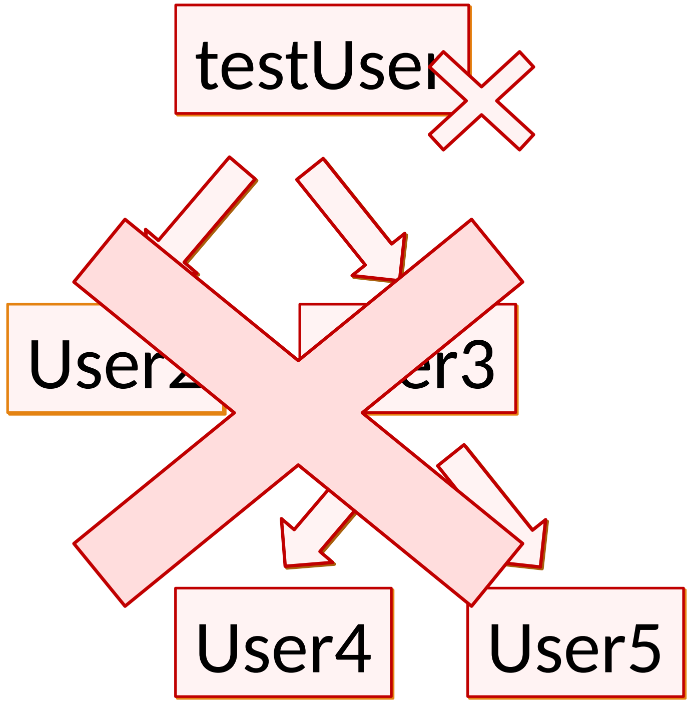
```
REVOKE SELECT ON Actors TO testUser;

REVOKE SELECT, INSERT, UPDATE, DELETE ON Actors TO testUser, mary;

REVOKE ALL ON Object::Actors TO testUser; -- Deprecated in SQL Server
```

- This is the default behaviour/state of new users
- A permission revoked from a user can still be inherited from other users (or roles) via the GRANT command

Access Control – Revoke Privileges

Where the grant option was specified, the removal of privileges must be cascaded to other affected users:

```
REVOKE SELECT, INSERT ON Actors TO testUser CASCADE;
```

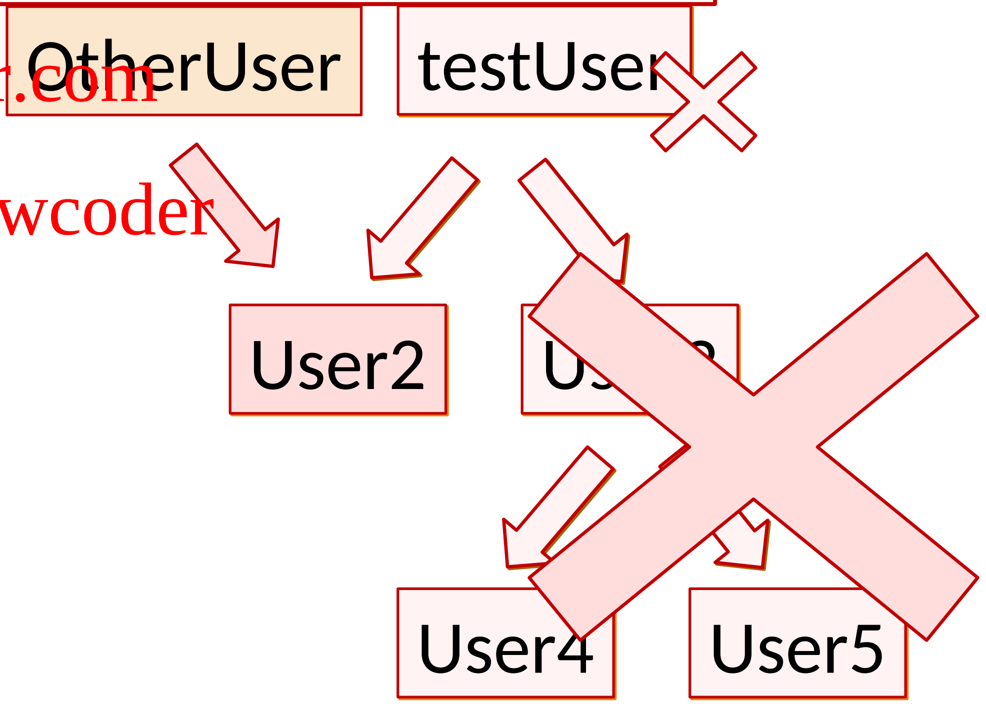


Access Control – Revoke Privileges

Where the grant option was specified, the removal of privileges must be cascaded to other affected users:

```
GRANT SELECT, INSERT ON Actors TO User2;
REVOKE SELECT, INSERT ON Actors TO testUser CASCADE;
```

- What if the same privilege was granted by another user to User2?
  - CASCADE will revoke privileges even if they are duplicated (granted) by other users



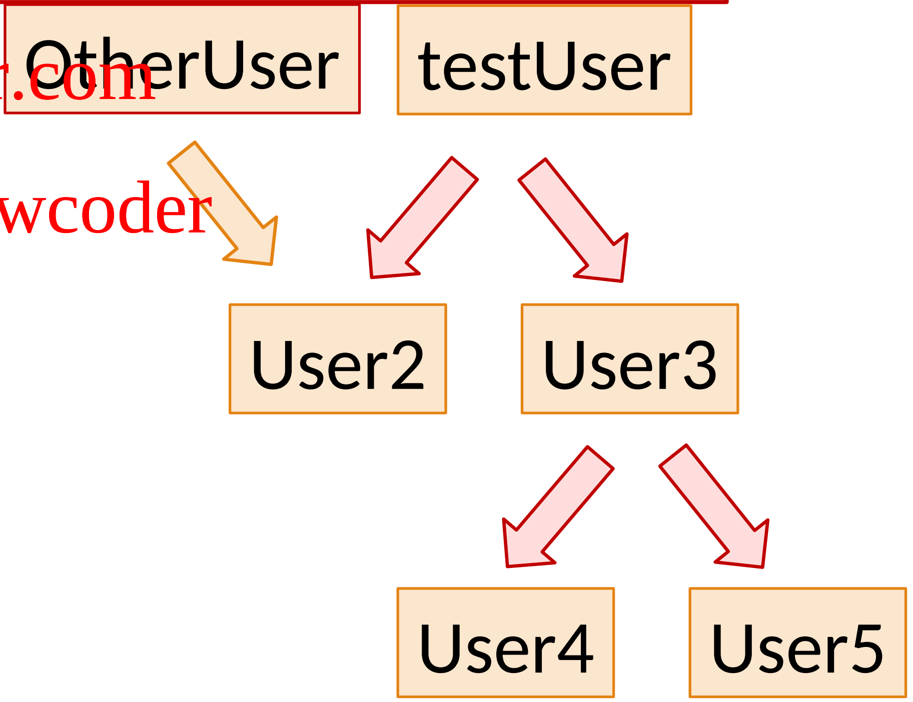
Access Control – Revoke Privileges

Where the grant option was specified, the removal of privileges must be cascaded to other affected users:

```
GRANT SELECT, INSERT ON Actors TO User2;
REVOKE SELECT, INSERT ON Actors TO testUser RESTRICT;
```

- What if the same privilege was granted by another user to User2?
  - RESTRICT **will throw an error** if the grantee has delegated the specified privileges to other users

DEFAULT BEHAVIOUR OF MS-SQL Server and does not need to be specified
> The keyword **RESTRICT** is not recognised



Access Control – Revoke Privileges

Privileges can be assigned to VIEWS and Stored Procedures

```
GRANT SELECT ON SimpsonsView TO testUser, mary; -- for a view
```

Privileges can also be assigned to Stored Procedures

```
ON GetSimpsons TO testUser; -- for a stored
```

Access Control – DENY Privileges

Access can be denied

```
DENY SELECT ON People(taxFileNo) TO userGroup1;

DENY DELETE ON Actors TO userGroup1;

DENY SELECT ON SimpsonsView TO userGroup1;

DENY EXECUTE ON GetSimpsons TO userGroup1;
```

- Unlike revoke, DENY revokes a permission so that it **cannot be inherited**
- **DENY takes precedence over all permissions**
  - Even if another user GRANTs the permission, it is still denied!

Access Control – DENY Privileges

Regaining a denied access

- REVOKE the original access so that it may later be GRANTED by others
- GRANT the access via dbo/owner or sa

```
REVERT
REVOKE SELECT ON Characters TO testUser2
OR
GRANT SELECT ON Characters TO testUser2
```

- DENY prevents others from using GRANT to give permission to a user…
  - Note that MS-SQL allows the grant but when the user goes to execute a denied action it fails

SECURITY

SETTING MINIMUM PERMISSIONS

Least-Privileged User Account

LUA – Least privileged User Account

- Important strategy to ensuring users only have access to the data and objects in the database that they should have access too

- New users have all privileges to all objects REVOKED until otherwise GRANTED

- An existing user (either one with GRANT OPTION) or the owner/sa must GRANT the necessary permissions to a new user

Example – Determine Min Privileges

Query to Execute

```
EXECUTE AS User='testUser';
SELECT CharacterName, DateAired, EpisodeName
FROM Characters AS C JOIN Episodes AS E
ON C.EpisodeID = E.EpisodeID
```

Min Permissions Required

Example – Determine Min Privileges

Query to Execute

```
EXECUTE AS User='testUser';
SELECT CharacterName, DateAired, EpisodeName
FROM Characters AS C JOIN Episodes AS E
ON C.EpisodeID = E.EpisodeID
```

Min Permissions Required

Characters: SELECT(CharacterName, EpisodeID)
Episodes: SELECT(EpisodeName, DateAired, EpisodeID)

Example – Determine Min Privileges

Query to Execute

```
INSERT INTO CharacterVoices (CharacterID, ActorID)
SELECT  C.CharacterID, A.ActorID FROM Characters C
JOIN CharactersImport CI ON
C.CharacterName = CI.Character
JOIN Actors a ON CI.ActorName = CI.[Voice actor(s)]
```

Min Permissions Required

Example – Determine Min Privileges

Query to Execute

```
INSERT INTO CharacterVoices (CharacterID, ActorID)
SELECT  C.CharacterID, A.ActorID FROM Characters C
JOIN CharactersImport CI ON
C.CharacterName = CI.Character
JOIN Actors a ON CI.ActorName = CI.[Voice actor(s)]
```

Min Permissions Required

CharacterVoices: INSERT // no attribute list
Characters: SELECT(CharacterID, CharacterName)
CharactersImport: SELECT(Character, [Voice actor(s)])
Actors: SELECT(ActorName, ActorID)

Example – Determine Min Privileges

Query to Execute

```
UPDATE Characters SET EpisodeID = E2.EpisodeID
FROM Characters C2 JOIN CharactersImport CI ON
C2.CharacterName = CI.Character
JOIN Episodes E2 ON E2.EpisodeName = CI.Episode
AND E2.DateAired = CI.OriginalAirDate;
```

Min Permissions Required

Example – Determine Min Privileges

Query to Execute

```
UPDATE Characters SET EpisodeID = E2.EpisodeID
FROM Characters C2 JOIN CharactersImport CI ON
C2.CharacterName = CI.Character
JOIN Episodes E2 ON E2.EpisodeName = CI.Episode
AND E2.DateAired = CI.originalAirDate;
```

Min Permissions Required

Characters: UPDATE(EpisodeID)
Characters: SELECT(CharacterName)
Episodes: SELECT(EpisodeName, EpisodeID)
CharactersImport: SELECT(originalAirDate, Character, Episode)

Example – Determine Min Privileges

Query to Execute

```
DELETE FROM Customers WHERE CustomerID > 149
```

Min Permissions Required

Example – Determine Min Privileges

Query to Execute

```
DELETE FROM Customers WHERE CustomerID > 149
```

Min Permissions Required

Customers: DELETE
Customers: SELECT (CustomerID)

Security

VIEWS

DB Security – VIEWS

Each user can be given the privilege to access the database only through a small set of views that contain the data appropriate for that user

- Views can restrict what a user knows to exist in a database
  - If it's not displayed then the user doesn't know the data exists

- Views can redefine the attributes a user sees
  - Every attribute can be renamed
  - The user does not know the actual attribute names

DB Security – VIEWS

```
CREATE VIEW SimpsonsCharacters (ID, Name, [Role], Episode, AiredDate)
AS

SELECT
CharacterID, CharacterName, CharacterRole,
EpisodeName, DateAired
FROM Characters AS C
JOIN Episodes AS E
ON C.EpisodeID = E.EpisodeID
```

What the user sees as the tables:

SimpsonsCharacters(<u>ID</u>, Name, Role, Episode, DateAired)
- Do not know that there are two tables (Characters and Episodes)
- Do not know any of the real attribute names

Security

STORED PROCEDURES

DB Security – Stored Procedures

Stored Procedures can be used to restrict the types of queries a user can execute

- Like views, they can be used to mask the names of underlying attributes
- The can also specify the exact queries a user can utilise

DB Security – Stored Procedures

```
CREATE Procedure GetSimpsonsCharacters
@name varchar(100)
AS

IF (@name IS NOT NULL)
BEGIN
SELECT TOP 5
CharacterID AS ID,
CharacterName AS Name,
CharacterRole AS [Role],
EpisodeName AS Episode,
DateAired AS DateAired
FROM Characters AS C
JOIN Episodes AS E
ON C.EpisodeID = E.EpisodeID
RETURN -- return the results
END
```

```
ELSE
BEGIN
RETURN-- give them nothing!!
END
```

Using a stored procedure you can specify:

- The user must have a search term
- How many records the user gets back
- The name of the attributes
- The exact search a user can perform

DB Security – Stored Procedures

```
CREATE Procedure InsertCharacter3
@name varchar(100),
@role varchar(100),
@episode varchar(100),
@airedDate date,
@ID int OUTPUT
AS

-- check if character does not already exist
SET @ID = (SELECT characterID
                              FROM Characters
                              WHERE
CharacterName = @name
                              )

-- check if epsiode does not already exist
DECLARE @episodeID int = (
         SELECT EpisodeID FROM Episodes
         WHERE EpisodeName = @episode
         )
```

```
IF(@episodeID IS NULL)
BEGIN
 INSERT INTO Episodes (EpisodeName, DateAired)
 VALUES (@episode, @airedDate)
END

IF (@ID IS  NULL)
     BEGIN
         INSERT INTO Characters (
                  CharacterName,
                  CharacterRole,
                  EpisodeID
                  ) VALUES (
                  @name,
                  @role,
                  @episodeID
         );
         SET @ID = SCOPE_IDENTITY();
     END
RETURN
```

DB Security – Stored Procedures

Users can be granted access to execute a Stored Procedure (SP) but nothing else

- Even if the SP contains INSERT, UPDATE, DELETE, SELECT queries on various tables the EXECUTE privilege can be granted to the SP only
  - This gives a user permission to use the SP regardless of the tables affected

- If the user tries to write a query (even if it exists in the SP) and execute it, the query will be denied
  - The privilege to execute an SP does not give a user the privileges to access to any base tables.