Security and Authorization

Introduction to DB Security

- Secrecy: Users shouldn’t be able to see things they are not supposed to.
  - E.g., A student can’t see other students’ grades.

- Integrity: Users shouldn’t be able to modify things they are not supposed to.
  - E.g., Only instructors can assign grades.

- Availability: Users should be able to see and modify things they are allowed to.

Access Controls

- Discretionary access control (DAC)
- Mandatory access control (MAC)

Discretionary Access Control

- Based on concept of privileges for objects (tables and views) and mechanisms for granting and revoking privileges

- Creator of an object automatically gets all privileges on it.

- DMBS keeps track of who subsequently gains and loses privileges.

GRANT Command

GRANT privileges ON object TO users [WITH GRANT OPTION]

- privileges can be:
  - **SELECT** Can read all columns

    including those added later via ALTER TABLE command
  - **INSERT(column-name)** Can insert tuples with **non-null** or **nondefault** values in this column.
  - **INSERT** means same right with respect to all columns.
  - **DELETE** Can delete tuples.
  - **REFERENCES (column-name)** Can define foreign keys (in other tables) that refer to this column.

- If you want the recipient(s) to be able to pass the privilege(s) to others add:

  WITH GRANT OPTION

Grant Examples I

- Suppose **Joe** has created the tables

  Sailors(<u>*sid*</u>*, sname, rating, age***)**

  Boats(<u>*bid*</u>*, bname, color***)**

  Reserves(<u>*sid, bid, day*</u>**)**

- **Joe** now executes the following:

```
GRANT INSERT, DELETE ON Reserves TO Yuppy WITH GRANT OPTION;
```

- Yuppy can now insert or delete **Reserves** rows and authorize someone else to do the same.

Grant Examples II

- **Joe** further executes:

```
GRANT SELECT ON Reserves TO Michael;
GRANT SELECT ON Sailors TO Michael WITH GRANT OPTION;
```

- **Michael** can create a view:

```
   CREATE VIEW ActiveSailors (name, age, day) AS
    SELECT S.sname, S.age, R.day
    FROM   Sailors S, Reserves R
    WHERE  S.sid = R.sid AND S.rating > 6;
```

- However, Michael cannot grant SELECT on **ActiveSailors** to others. Why?

Grant Examples III

- On the other hand, suppose that Michael executes:

```
CREATE VIEW YoungSailors (sid, age, rating) AS
  SELECT S.sid, S.age, S.rating
  FROM Sailors S
  WHERE S.age < 18;
```

- Michael can then say:

```
GRANT SELECT ON YoungSailors TO Eric, Guppy;
```

- Eric and Guppy can exec. SELECT queries on **YoungSailors**.
  - – however, not directly on the underlying **Sailor** table.

Grant Examples IV

- Suppose now **Joe** executes:

```
GRANT UPDATE (rating) ON Sailors TO Leah;
```

- Leah can update only the *rating* column of Sailors. E.g.

```
 UPDATE Sailors
 SET rating = 8;
```

- However, she cannot execute:

```
 UPDATE Sailors
 SET age = 25;
```

- She cannot execute either:

```
 UPDATE Sailors
 SET rating = rating-1;
```

- Why?

Grant Examples V

- Suppose now that **Joe** executes:

```
GRANT SELECT, REFERENCES(bid) ON Boats TO Bill;
```

- **Bill** can then refer to the *bid* column of Boats as a foreign key in another table. E.g.

```
CREATE TABLE BillTable (
     bid INTEGER,
     …
     FOREIGN KEY (bid) REFERENCES Boats
);
```

But, why the SQL standard chose to introduce the REFERENCES privilege rather than to simply allow the SELECT privilege to be used when creating a Foreign Key?

This is because now Joe is restricted in what he can do with the tuples in his own table. For example, Joe cannot now delete a tuple in Boats which is referenced by a tuple in BillTable.
In other words, Bill can now constrain Joe wrt to his table. This goes well beyond a simple SELECT privilege.

Role-Based Authorization

- Privileges can also be assigned to roles.
- Roles can then be granted to users and to other roles.
- Roles reflect how real organizations work.

Example.

```
CREATE ROLE some_role;

GRANT SELECT ON Reserves TO some_role;
GRANT INSERT ON Sailors TO some_role;
GRANT UPDATE ON Boats TO some_role;

GRANT some_role TO Michael;
GRANT some_role TO Bill;
```

Revoke Examples

REVOKE [GRANT OPTION FOR] privileges

ON object FROM users {RESTRICT | CASCADE}

Suppose **Joe** is the creator of Sailors.

Joe executes:

GRANT SELECT ON Sailors TO Art WITH GRANT OPTION

Art executes:

GRANT SELECT ON Sailors TO Bob WITH GRANT OPTION

Joe executes:

REVOKE SELECT ON Sailors FROM Art CASCADE

Art loses the SELECT privilege on Sailors.
Then Bob, who received this privilege from Art, and only Art, also loses this privilege.
Bob's privilege is said to be **abandoned**

If CASCADE is specified, all abandoned privileges are also revoked possibly causing privileges held by other users to become abandoned and thereby revoked recursively.
If RESTRICT is specified, the command is rejected if revoking privileges causes other privileges to become abandoned.

Revoke Examples

Joe executes:

```
GRANT SELECT ON Sailors TO Art WITH GRANT OPTION
```

Joe executes:

```
GRANT SELECT ON Sailors TO Bob WITH GRANT OPTION
```

Art executes:

```
GRANT SELECT ON Sailors TO Bob WITH GRANT OPTION
```

Joe executes:

```
REVOKE SELECT ON Sailors FROM Art CASCADE
```

- As before, Art loses the SELECT privilege on Sailors.
- But what about Bob?
- Bob received this privilege from Art, but he also received it independently from Joe. So, he doesn’t lose the privilege.

Revoke Examples

Joe executes:

```
GRANT SELECT ON Sailors TO Art WITH GRANT OPTION
GRANT SELECT ON Sailors TO Art WITH GRANT OPTION
REVOKE SELECT ON Sailors FROM Art CASCADE
```

- Since Joe granted the privilege to Art twice and only revoked it once, does Art get to keep the privilege?
- NO. It doesn't matter how many times we grant a privilege.

Privilege Descriptors

- When a GRANT is executed, a privilege descriptor is added to a descriptors table.
- Privilege descriptor specifies the:
  - *grantor,*
  - *grantee,*
  - *granted privilege*
  - *grant option*
- When a user creates a table or view he 'automatically' gets all privileges from system.

Authorization Graphs

Nodes: users. **Arcs:** privileges passed.

WGO abbr. for WITH GRANT OPTION

Joe:

GRANT SELECT ON Sailors TO Art WGO

Art:

GRANT SELECT ON Sailors TO Bob WGO

Bob:

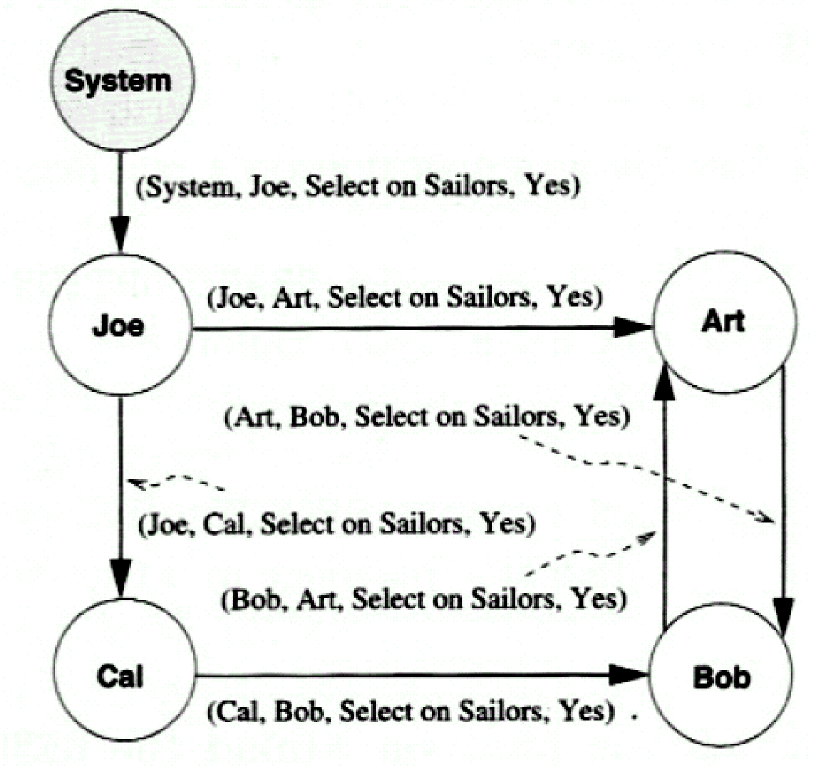
GRANT SELECT ON Sailors TO Art WGO

Joe:

GRANT SELECT ON Sailors TO Cal WGO

Cal:

GRANT SELECT ON Sailors TO Bob WGO


System
(System, Joe, Select on Sailors, Yes)
Joe
(Joe, Art, Select on Sailors, Yes)
Art
(Art, Bob, Select on Sailors, Yes)
(Joe, Cal, Select on Sailors, Yes)
(Bob, Art, Select on Sailors, Yes)
Cal
Bob
(Cal, Bob, Select on Sailors, Yes)

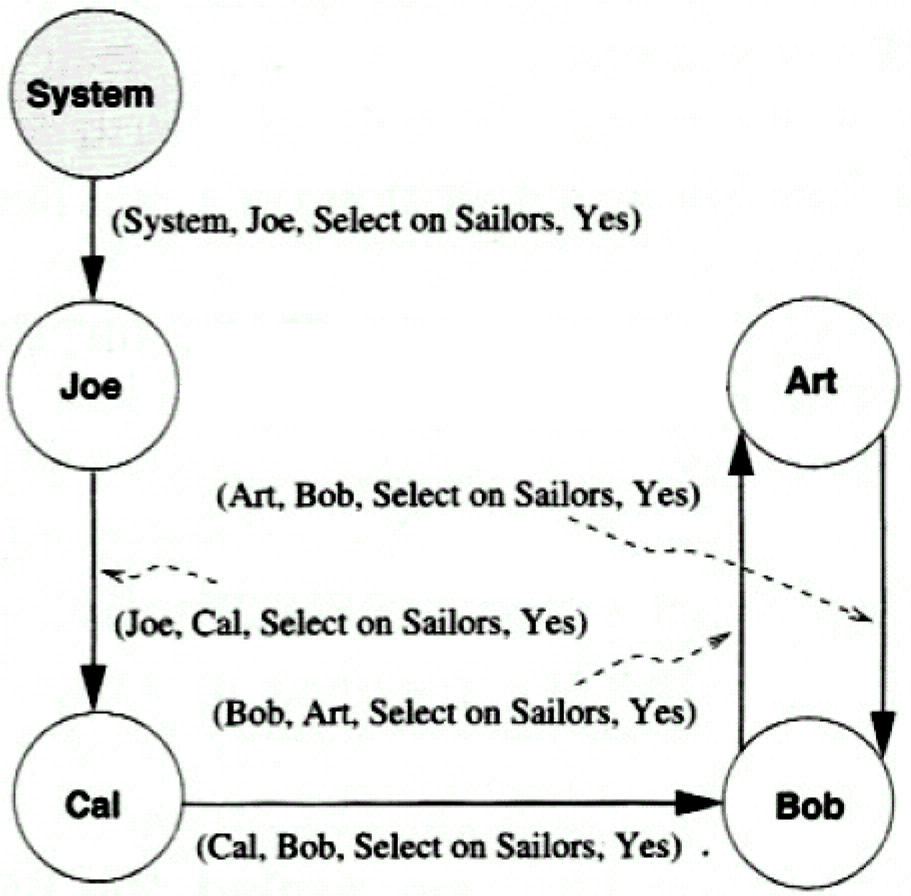
Effects of Revocations I

- Suppose Joe executes:

  REVOKE SELECT ON
  Sailors FROM Art
  CASCADE

- The arc from Joe to Art is removed.

- Art still has the privilege
  - Got it independently from Bob.


System
(System, Joe, Select on Sailors, Yes)
Joe
Art
(Art, Bob, Select on Sailors, Yes)
(Joe, Cal, Select on Sailors, Yes)
(Bob, Art, Select on Sailors, Yes)
Cal
Bob
(Cal, Bob, Select on Sailors, Yes)

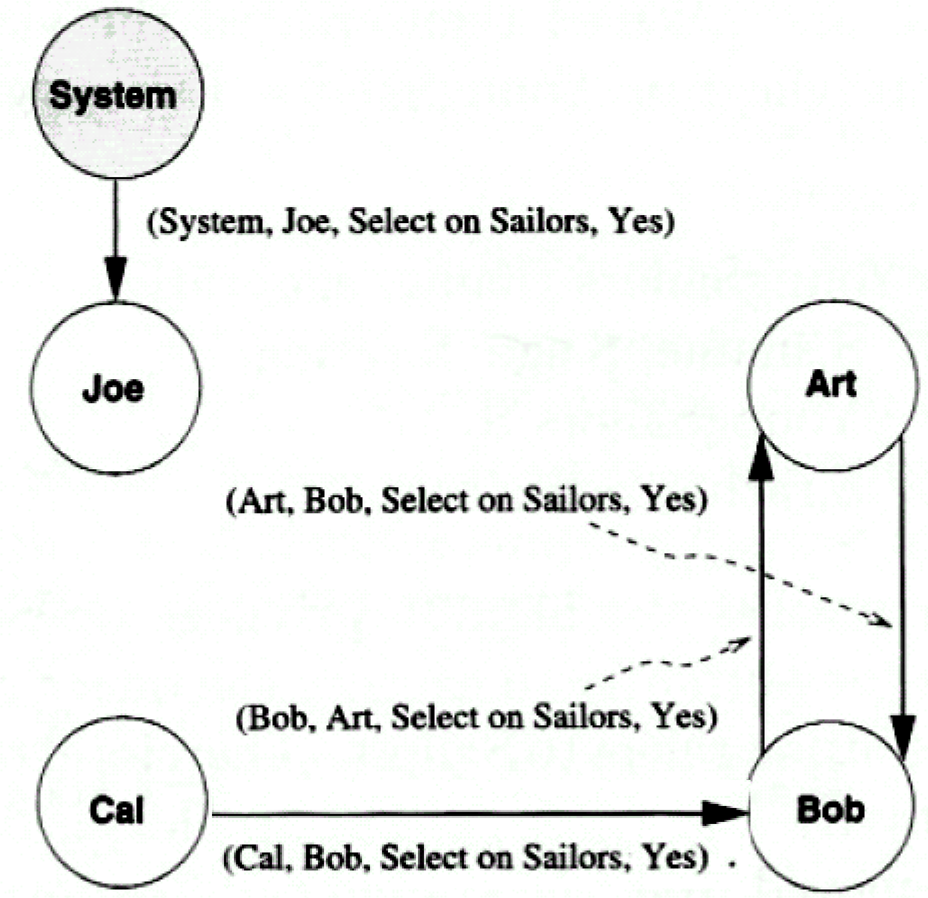
Effects of Revocations II

- Suppose Joe now executes:

  REVOKE SELECT ON
  Sailors FROM Cal
  CASCADE

- The arc from Joe to Cal is removed.

- Cal, Art and Bob lose privileges because **there isn't a path from System**.


System
(System, Joe, Select on Sailors, Yes)
Joe
Art
(Art, Bob, Select on Sailors, Yes)
(Bob, Art, Select on Sailors, Yes)
Cal
Bob
(Cal, Bob, Select on Sailors, Yes)

Grant and Revoke on Views

Suppose that

- Joe, the creator of table **Sailors,** gave Michael SELECT privilege on the table with grant option.
- Michael created view **YoungSailors** and gave Eric SELECT privilege on it.
- Eric created view **FineYoungSailors**:

```
CREATE VIEW FineYoungSailors (name, age, rating) AS
        SELECT S.sname, S.age, S.rating
        FROM YoungSailors S
        WHERE S.rating> 6;
```

- What happens if Joe revokes the SELECT privilege on **Sailors** from Michael?

- Michael no longer has the authority to execute the query used to define **YoungSailors** because the definition refers to **Sailors.**
  - Therefore, the view **YoungSailors** is dropped.
  - In turn, **FineYoungSailors** is dropped as well.

Revoking REFERENCES privilege

- Recall, Joe had executed:

```
GRANT REFERENCES(bid) ON Boats TO Bill;
```

- Bill referred to the *bid* column of Boats as a foreign key in BillTable table.

```
CREATE TABLE BillTable (
     bid INTEGER,
     …
     FOREIGN KEY (bid) REFERENCES Boats
);
```

- If now Joe revokes the REFERENCES privilege from Bill, then the Foreign Key constraint referencing the **Boat** table will be dropped from the Bill's **Reserves** table.

The problem with DAC

- Discretionary control has some flaws, e.g., the *Trojan horse* problem:
- Dick creates table **Horsie** and gives **INSERT** privilege to Justin (who doesn’t know about this).
- Dick modifies the code of an application program used by Justin to additionally write some secret data to table **Horsie**.
- Now, Dick can see the secret info.

The modification of the code is beyond the DBMS’s control, but DBMS can try and prevent the use of the database as a channel for secret information.

Mandatory Access Control

- Based on system-wide policies that cannot be changed by individual users.
- Each DB object is assigned a security class.
- Each subject (user or user program) is assigned a clearance for a security class.

Bell-LaPadula Model

- **Objects** (e.g., tables, views)
- **Subjects** (e.g., users, user programs)
- **Security classes**:
  - Top secret (TS), secret (S), confidential (C), unclassified (U):
  - TS > S> C > U
- Each object and subject is assigned a class.

- Simple Security Property: Subject *S* can read object *O* only if

  class(S) >=class(O)
- *-Property: Subject *S* can write object *O* only if

  class(S) <= class(O)

Idea is to ensure that information can never flow from a higher to a lower security level. E.g.,

If Dick has security class C, Justin has class S, and the secret table has class S:
Dick's table, Horsie, has Dick's clearance, C.
Justin's application has his clearance, S.
So, the program cannot write into table Horsie.

Statistical Databases

Statistical DB: Contains information about individuals, but allows only **aggregate** queries (e.g., average age, rather than Joe's age).

New problem: It may be possible to infer some secret information!

Example.

- Suppose Sneaky Pete wants to know the **rating** of Admiral Horntooter
  - It happens that Pete knows that Horntooter is the oldest sailor in the club.
  - Pete repeatedly asks "How many sailors are older than an age X" for various values of *X,* until the answer is 1.
  - Obviously, this sailor is Horntooter, the oldest sailor.
  - Hence the Horntooter's age is discovered.
- Each of these queries is a valid statistical query and is permitted. Let the value of *X* at this point be, say, 65.

- Pete now asks the query, "What is the average rating of all sailors whose age is greater or equal to 65?"

  …discovering so the Horntooter's rating.