Chapter 10 Advanced topics in relational databases

- Security and user authorization in SQL
- Recursion in SQL
- Object-relational model
  1. User-defined types in SQL
  2. Operations on object-relational data
- Online analytic processing & data cubes

Security and user authorization in SQL

Authorization

Aim:

- Make sure users only see the data they’re suppose to
- Guard the database against updates by malicious users

How SQL control it?

  - Authorization ID
  - Privileges

Authorization ID

A user is referred to by *authorization ID*, typically their name.

- An element of SQL environment
- A user or a group of users who may be granted some particular privileges on objects
  - User ID: personal security account on behalf of individuals, applications, system services
    - Not defined in SQL standard regarding its creation,
  - Role: a defined set of privileges granted to users or other roles
    - CREATE ROLE
- PUBLIC: a special built-in authorization ID
  - Granting a privilege to PUBLIC makes it available to any authorization ID.

Authorization in SQL

- File systems identify certain access privileges on files, e.g., read,write,execute.
- In partial analogy, SQL identifies nine types of privileges:

1. SELECT = the right to query the relation

Authorization in SQL (cont.)

2. INSERT = the right to insert tuples into the relation, may refer to one attribute, in which case the privilege is to specify only one column of the inserted tuple.
3. DELETE = the right to delete tuples from the relation.
4. UPDATE = the right to update tuples of the relation, may refer to one attribute.

Example: What privileges are needed for this statement?

```
INSERT INTO Beers(name)
 SELECT beer FROM Sells
 WHERE NOT EXISTS
     (SELECT * FROM Beers
      WHERE name = beer);
```

beers that do not appear in Beers. We add them to Beers with a NULL manufacturer.

- We require privileges SELECT on Sells and Beers, and INSERT on Beers or Beers.name.

Obtaining Privileges

- How to grant privilege?
- Owner vs. granted user
  - Owner has all privileges and may GRANT them to others

Privilege-Checking

- Each module, schema, and session has an associated authorization ID.
- Let agent A executes a module that operates on a DB element: A's privileges derive from the *current auth. ID* that is either
  - module auth. ID if there is one, or
  - session auth. ID if not.

We may execute the SQL operation only if the current auth. ID possesses all the privileges.

Principles for Privilege-Checking

- The current authorization ID is the owner of the data.
- The current authorization ID has been granted by the owner or been granted to user PUBLIC
- Executing a module owned by the owner or by someone who has been granted of needed privileges and EXECUTE privilege on the module.
- Session auth. ID with the needed privileges to publicly available module.

Granting Privileges

- You have all possible privileges to the relations you create. (owner)
- You may grant privileges to any user if you have those privileges” **with grant option**.” You have this option to your own relations.

Example

1) Sally can query Sells and can change prices, but cannot pass on this power:

```
GRANT SELECT ON Sells, UPDATE (price)
   ON Sells TO sally;
```

2) Sally can also pass these privileges to whom she chooses;

```
GRANT SELECT ON Sells, UPDATE (price)
   ON Sells TO sally WITH GRANT OPTION;
```

Grant diagrams

- An SQL system maintains a representation of this diagram to keep track of both privileges and their origins.
- The nodes of a grant diagram correspond to a user and a privilege.
- A privilege with and without the grant option must be represented by two different nodes.

Grant Diagrams

- Node: user/privilege
- Arc: grants
- * = WITH GRANT OPTION

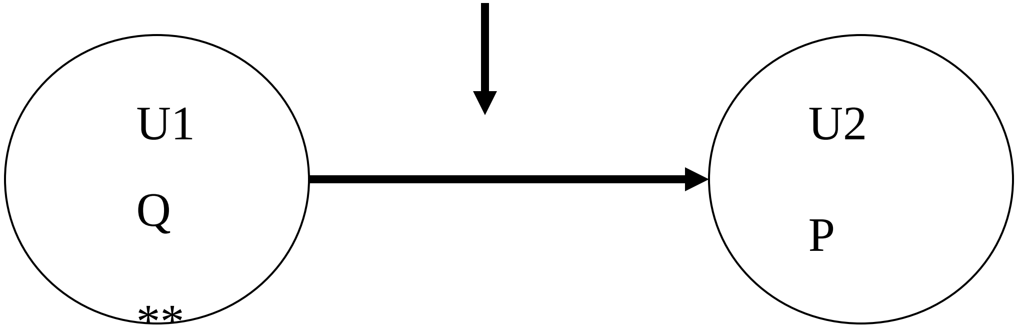
  ** = derived from ownership

For example:

Q: is UPDATE ON R

P: UPDATE(a) on R

User U1 grants privilege P to user U2



Q is more general than P

Revoking Privileges

- Syntax

  REVOKE *privileges* ON *relation* FROM *users*

  [CASCADE | RESTRICT]

  - CASCADE: transitively revoking.
  - RESTRICT: Revoke not allowed if it would cause any node unreachable from an owner.

Revoking Privileges (cont.)

a) If you have been given a privilege by several different people, then all of them have to revoke in order for you to lose the privilege.

b) Revocation is transitive （传递的）. If A granted P to B, who granted P to C, and then A revokes P from B, it is as if B also revoked P from C.

Revoking Privileges (cont.)

c) Revoke with RESTRICT： the revoke statement cannot be executed if the cascading rule would result in the revoking of any privileges due to the revoked privileges having been passed on to others.

Revoking GRANT OPTION

- Syntax

  REVOKE GRANT OPTION FOR *privilege*

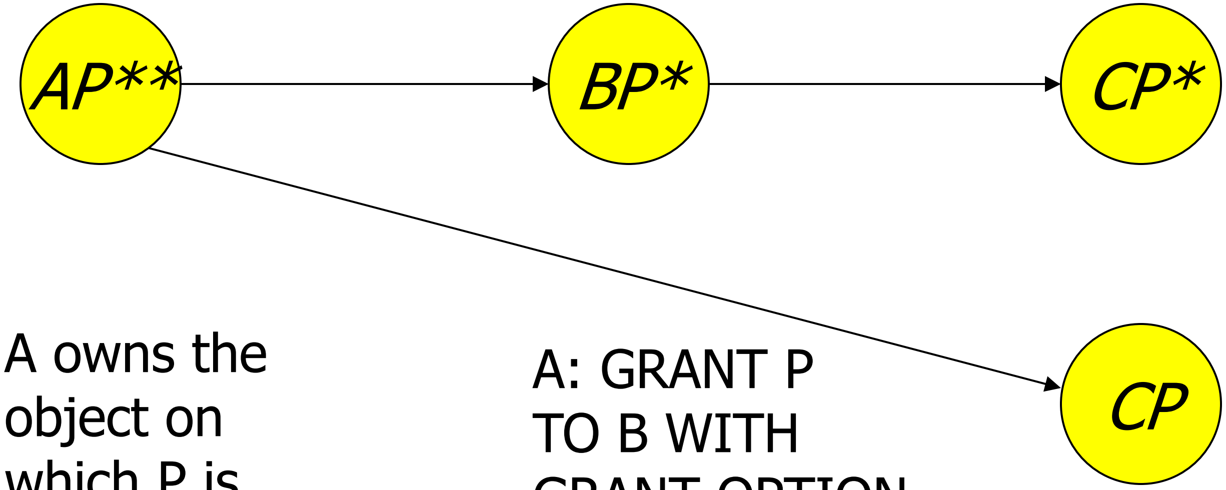
  ON *relation* FROM *users*

  [CASCADE | RESTRICT]

  - Only revoking the grant option, not the privilege itself.

Example: Grant Diagram

B: GRANT P
TO C WITH
GRANT OPTION



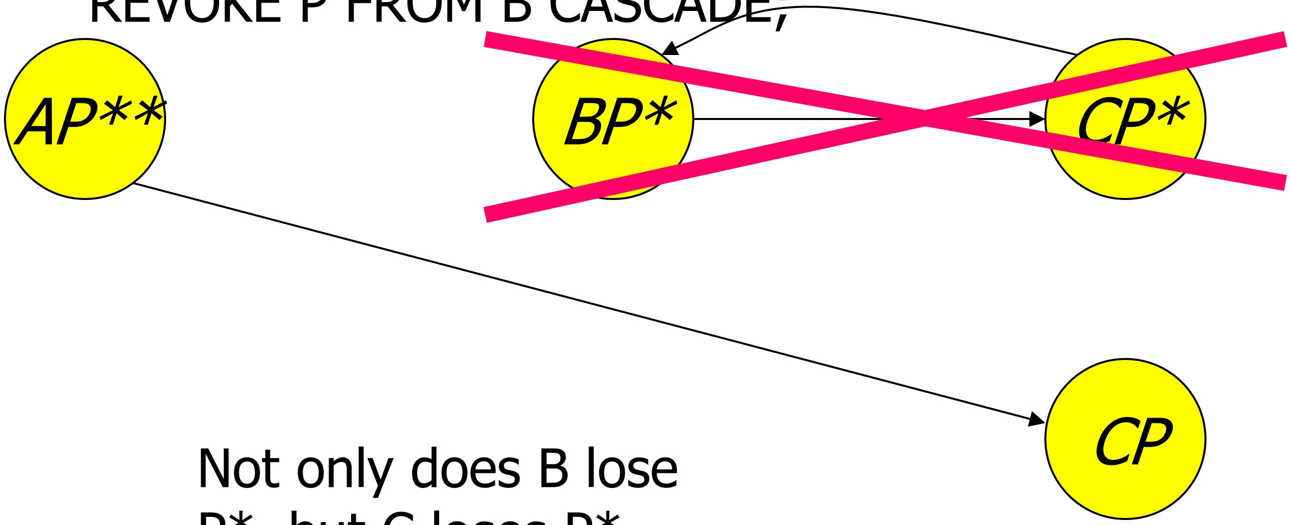
A owns the
object on
which P is
a privilege

A: GRANT P
TO B WITH
GRANT OPTION

A: GRANT P
TO C

Example: Grant Diagram

A executes
REVOKE P FROM B CASCADE;



Even had C passed P to B, both nodes are still cut off.

Not only does B lose P*, but C loses P*. Delete BP* and CP*.

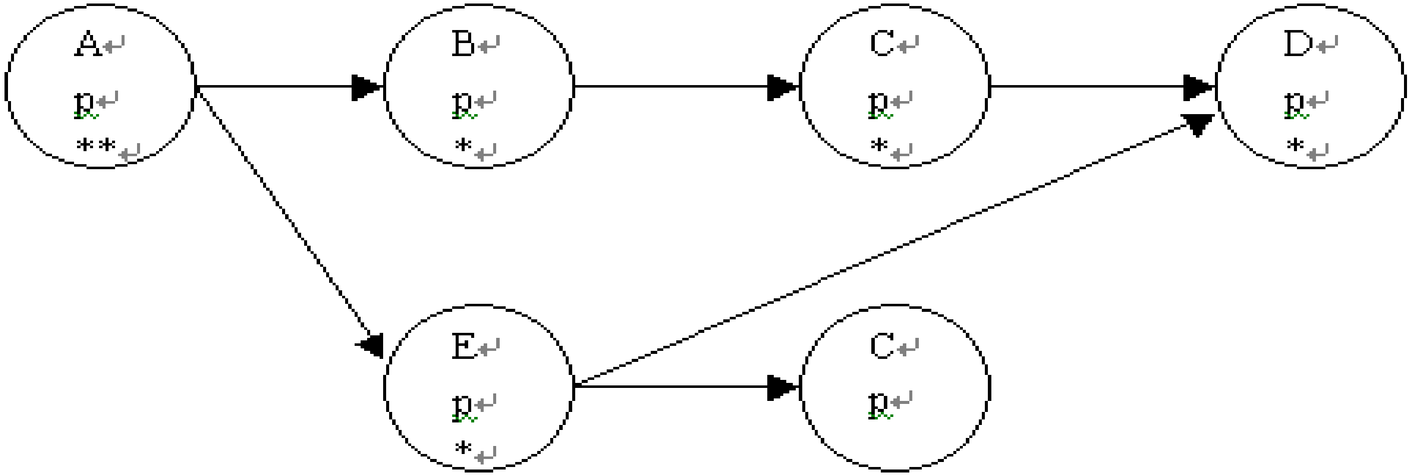
However, C still has P without grant option because of the direct grant.

If A executes

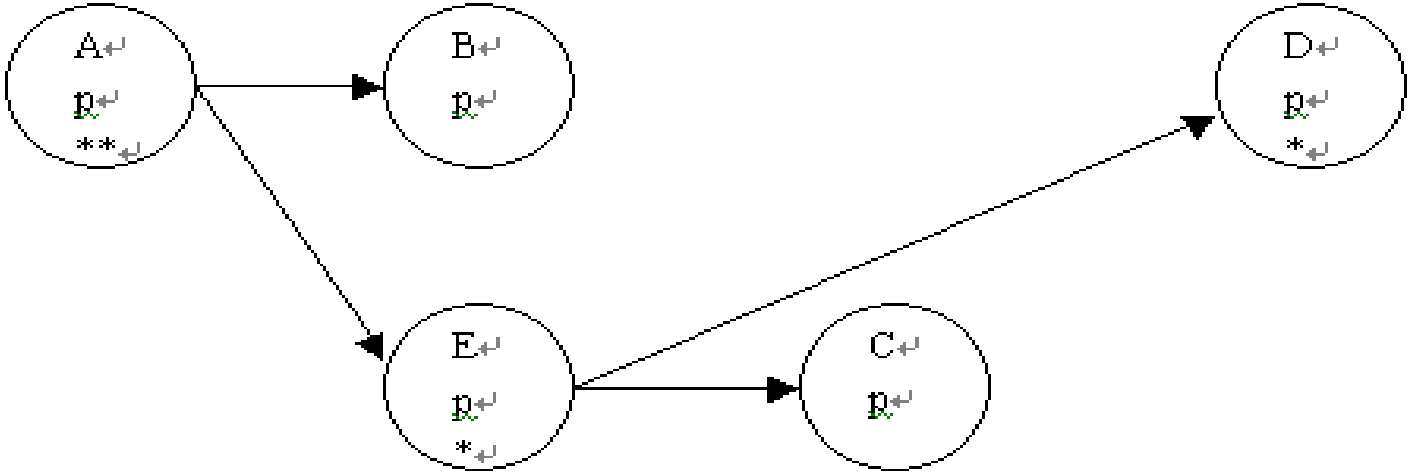
REVOKE P FROM B RESTRICT ??

Classroom Exercises (10.1.3)

- User A(owner): grant p to B, E with grant option
- User B: grant p to C with grant option
- User C: grant p to D with grant option
- User E: grant p to C
- User E: grant p to D with grant option
- User A: revoke grant option for p from B cascade



After step (6):


A
p
**
B
p
D
p
*
E
p
*
C
p