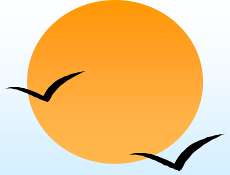
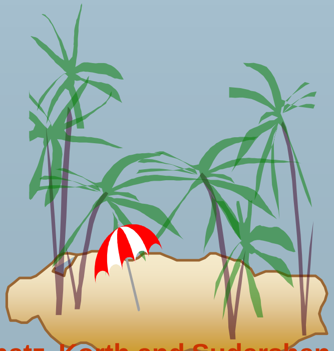


Security

- **Security** - protection from malicious attempts to steal or modify data.
  - Database system level
    - Authentication and authorization mechanisms to allow specific users access only to required data
    - We concentrate on authorization in the rest of this chapter
  - Operating system level
    - Operating system super-users can do anything they want to the database!   Good operating system level security is required.
  - Network level: must use encryption to prevent
    - Eavesdropping (unauthorized reading of messages)
    - Masquerading (pretending to be an authorized user or sending messages supposedly from authorized users)



Security (Cont.)

- Physical level
  - Physical access to computers allows destruction of data by intruders; traditional lock-and-key security is needed
  - Computers must also be protected from floods, fire, etc.
    - More in Chapter 17 (Recovery)
- Human level
  - Users must be screened to ensure that an authorized users do not give access to intruders
  - Users should be trained on password selection and secrecy

Authorization

Forms of authorization on parts of the database:

- **Read authorization** - allows reading, but not modification of data.
- **Insert authorization** - allows insertion of new data, but not modification of existing data.
- **Update authorization** - allows modification, but not deletion of data.
- **Delete authorization** - allows deletion of data

Authorization (Cont.)

Forms of authorization to modify the database schema:

- **Index authorization** - allows creation and deletion of indices.
- **Resources authorization** - allows creation of new relations.
- **Alteration authorization** - allows addition or deletion of attributes in a relation.
- **Drop authorization** - allows deletion of relations.

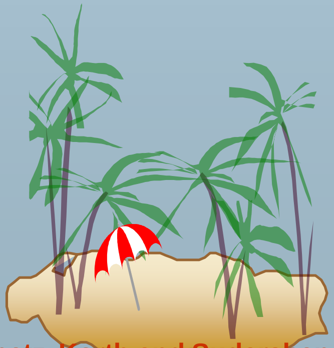
Authorization and Views

- Users can be given authorization on views, without being given any authorization on the relations used in the view definition
- Ability of views to hide data serves both to simplify usage of the system and to enhance security by allowing users access only to data they need for their job
- A combination or relational-level security and view-level security can be used to limit a user’s access to precisely the data that user needs.

View Example

- Suppose a bank clerk needs to know the names of the customers of each branch, but is not authorized to see specific loan information.
  - Approach: Deny direct access to the *loan* relation, but grant access to the view *cust-loan*, which consists only of the names of customers and the branches at which they have a loan.
  - The *cust-loan* view is defined in SQL as follows:

create view *cust-loan* **as**
  select *branchname*, *customer-name*
  from   *borrower, loan*
  where *borrower.loan-number* = *loan.loan-number*



View Example (Cont.)

- The clerk is authorized to see the result of the query:

  select *
  from *cust-loan*

- When the query processor translates the result into a query on the actual relations in the database, we obtain a query on *borrower* and *loan*.
- Authorization must be checked on the clerk's query before query processing begins.

Authorization on Views

- Creation of view does not require **resources** authorization since no real relation is being created
- The creator of a view gets only those privileges that provide no additional authorization beyond that he already had.
- E.g. if creator of view *cust-loan* had only **read** authorization on *borrower* and *loan*, he gets only **read** authorization on *cust-loan*

Granting of Privileges

- The passage of authorization from one user to another may be represented by an authorization graph.
- The nodes of this graph are the users.
- The root of the graph is the database administrator.
- Consider graph for update authorization on loan.
- An edge $U_i \rightarrow U_j$ indicates that user U_i has granted update authorization on loan to U_j.

$$\begin{tikzcd}
 & U_1 \arrow[r] \arrow[rd] & U_4 \\
DBA \arrow[ru] \arrow[r] \arrow[rd] & U_2 \arrow[r] & U_5 \\
 & U_3 &
\end{tikzcd}$$

Authorization Grant Graph

- *Requirement*: All edges in an authorization graph must be part of some path originating with the database administrator
- If DBA revokes grant from U_1:
  - Grant must be revoked from U_4 since U_1 no longer has authorization
  - Grant must not be revoked from U_5 since U_5 has another authorization path from DBA through U_2
- Must prevent cycles of grants with no path from the root:
  - DBA grants authorization to U_7
  - U7 grants authorization to U_8
  - U8 grants authorization to U_7
  - DBA revokes authorization from U_7
- Must revoke grant U_7 to U_8 and from U_8 to U_7 since there is no path from DBA to U_7 or to U_8 anymore.

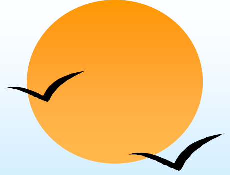
Security Specification in SQL

- The grant statement is used to confer authorization

  grant <privilege list>

  on <relation name or view name> to <user list>

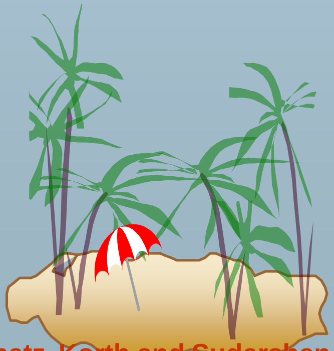
- <user list> is:
  - a user-id
  - *public*, which allows all valid users the privilege granted
  - A role (more on this later)
- Granting a privilege on a view does not imply granting any privileges on the underlying relations.
- The grantor of the privilege must already hold the privilege on the specified item (or be the database administrator).



Privileges in SQL

- **select:** allows read access to relation,or the ability to query using the view
  - Example: grant users U_1, U_2, and U_3 **select** authorization on the *branch* relation:

    grant select on *branch* **to** U_1, U_2, U_3
- **insert**: the ability to insert tuples
- **update**: the ability to update using the SQL update statement
- **delete**: the ability to delete tuples.
- **references**: ability to declare foreign keys when creating relations.
- **usage**: In SQL-92; authorizes a user to use a specified domain
- **all privileges**: used as a short form for all the allowable privileges

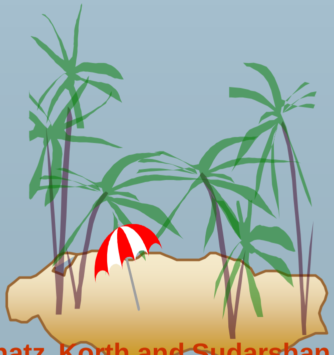


Privilege To Grant Privileges

- **with grant option**: allows a user who is granted a privilege to pass the privilege on to other users.
  - Example:

    grant select on *branch* **to** U_1 **with grant option**

    gives U_1 the **select** privileges on branch and allows U_1 to grant this privilege to others



Roles

- Roles permit common privileges for a class of users can be specified just once by creating a corresponding "role"
- Privileges can be granted to or revoked from roles, just like user
- Roles can be assigned to users, and even to other roles
- SQL:1999 supports roles

create role *teller*
create role *manager*

grant select on *branch* **to** *teller*
grant update (*balance***) on** *account* **to** *teller*
grant all privileges on *account* **to** *manager*

grant *teller* **to** *manager*

grant *teller* **to** *alice, bob*
grant *manager* **to** *avi*

Revoking Authorization in SQL

- The **revoke** statement is used to revoke authorization.

  revoke<privilege list>

  on <relation name or view name> **from** <user list> [**restrict**|**cascade**]

- Example:

  revoke select on *branch* **from** U_1, U_2, U_3 **cascade**

- Revocation of a privilege from a user may cause other users also to lose that privilege; referred to as cascading of the **revoke**.

- We can prevent cascading by specifying **restrict**:

  revoke select on *branch* **from** U_1, U_2, U_3 **restrict**

  With **restrict**, the **revoke** command fails if cascading revokes are required.

Revoking Authorization in SQL (Cont.)

- <privilege-list> may be **all to** revoke all privileges the revokee may hold.
- If <revokee-list> includes **public** all users lose the privilege except those granted it explicitly.
- If the same privilege was granted twice to the same user by different grantees, the user may retain the privilege after the revocation.
- All privileges that depend on the privilege being revoked are also revoked.

Limitations of SQL Authorization

- SQL does not support authorization at a tuple level
  - E.g. we cannot restrict students to see only (the tuples storing) their own grades
- All end-users of an application (such as a web application) may be mapped to a single database user
- The task of authorization in above cases falls on the application program, with no support from SQL
  - Authorization must be done in application code, and may be dispersed all over an application
  - Checking for absence of authorization loopholes becomes very difficult since it requires reading large amounts of application code

Encryption

- Data may be *encrypted* when database authorization provisions do not offer sufficient protection.
- Properties of good encryption technique:
  - Relatively simple for authorized users to encrypt and decrypt data.
  - Encryption scheme depends not on the secrecy of the algorithm but on the secrecy of a parameter of the algorithm called the encryption key.
  - Extremely difficult for an intruder to determine the encryption key.

Encryption (Cont.)

- *Data Encryption Standard* (DES) substitutes characters and rearranges their order on the basis of an encryption key which is provided to authorized users via a secure mechanism. Scheme is no more secure than the key transmission mechanism since the key has to be shared.
- Advanced Encryption Standard (AES) is a new standard replacing DES, and is based on the Rijndael algorithm, but is also dependent on shared secret keys
- *Public-key encryption* is based on each user having two keys:
  - *public key* – publicly published key used to encrypt data, but cannot be used to decrypt data
  - *private key* -- key known only to individual user, and used to decrypt data. Need not be transmitted to the site doing encryption.

  Encryption scheme is such that it is impossible or extremely hard to decrypt data given only the public key.
- The RSA public-key encryption scheme is based on the hardness of factoring a very large number (100's of digits) into its prime components.

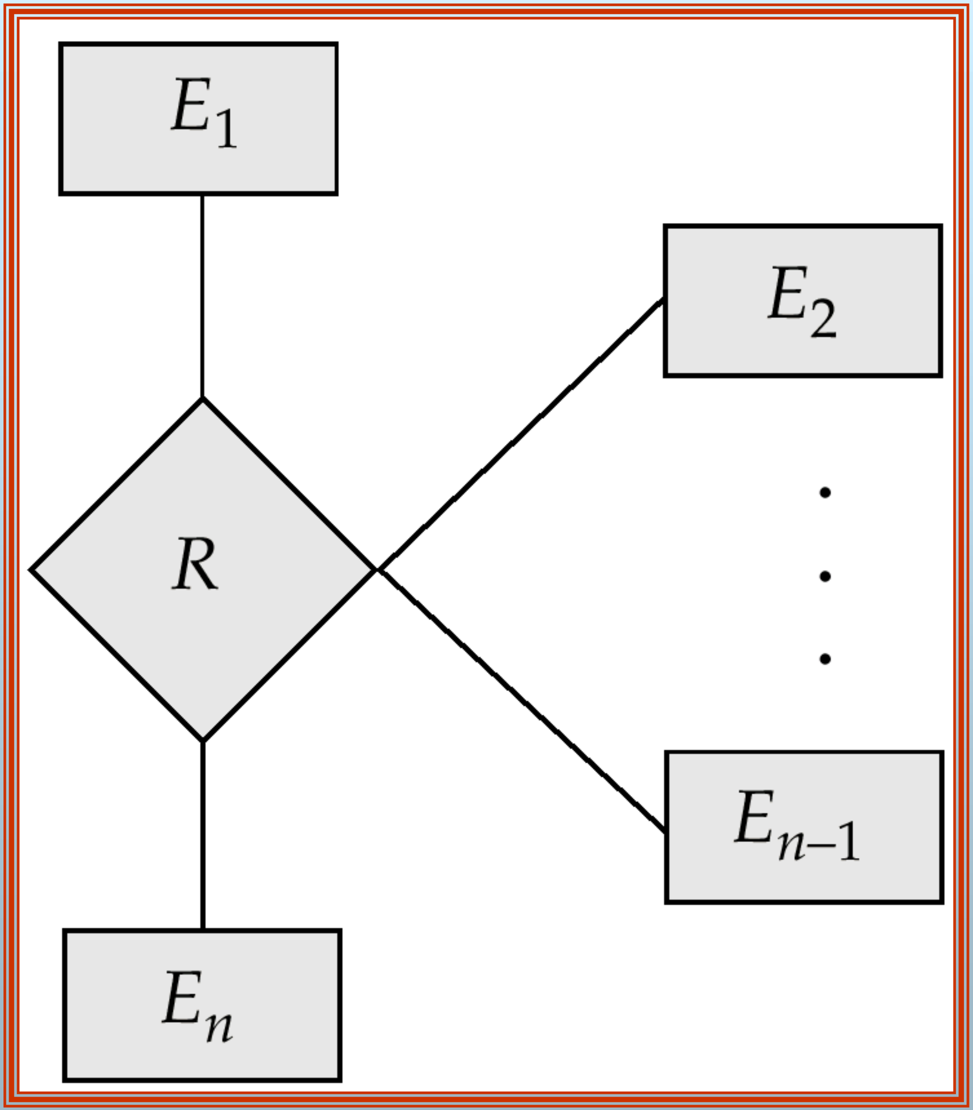
Authentication

- Password based authentication is widely used, but is susceptible to sniffing on a network
- **Challenge-response** systems avoid transmission of passwords
  - DB sends a (randomly generated) challenge string to user
  - User encrypts string and returns result.
  - DB verifies identity by decrypting result
  - Can use public-key encryption system by DB sending a message encrypted using user's public key, and user decrypting and sending the message back
- **Digital signatures** are used to verify authenticity of data
  - E.g. use private key (in reverse) to encrypt data, and anyone can verify authenticity by using public key (in reverse) to decrypt data. Only holder of private key could have created the encrypted data.
  - Digital signatures also help ensure **nonrepudiation:** sender cannot later claim to have not created the data

Statistical Databases

- Problem: how to ensure privacy of individuals while allowing use of data for statistical purposes (e.g., finding median income, average bank balance etc.)
- Solutions:
  - System rejects any query that involves fewer than some predetermined number of individuals.
    - Still possible to use results of multiple overlapping queries to deduce data about an individual
  - *Data pollution* -- random falsification of data provided in response to a query.
  - Random modification of the query itself.
- There is a tradeoff between accuracy and security.

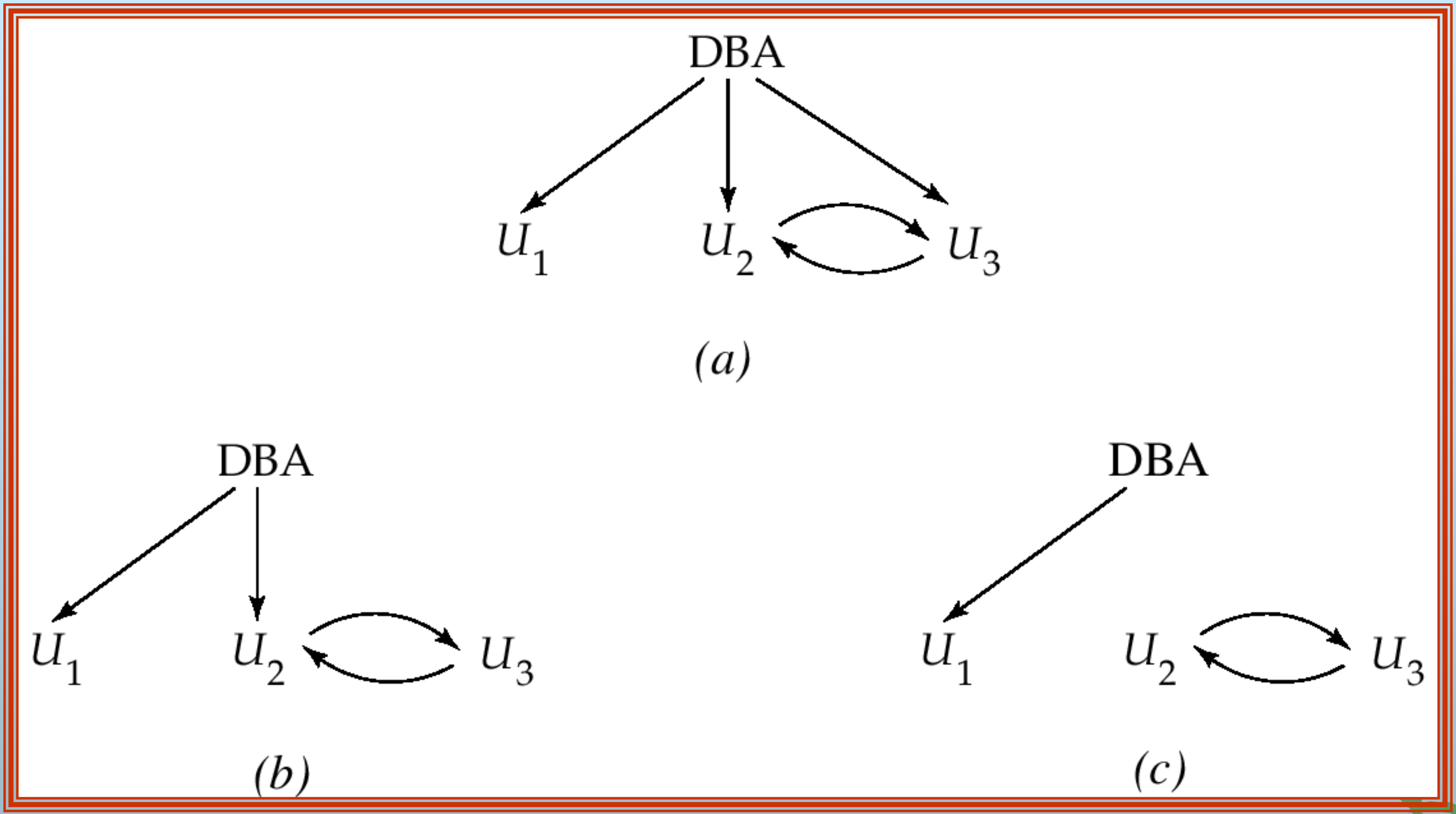
An *n*-ary Relationship Set



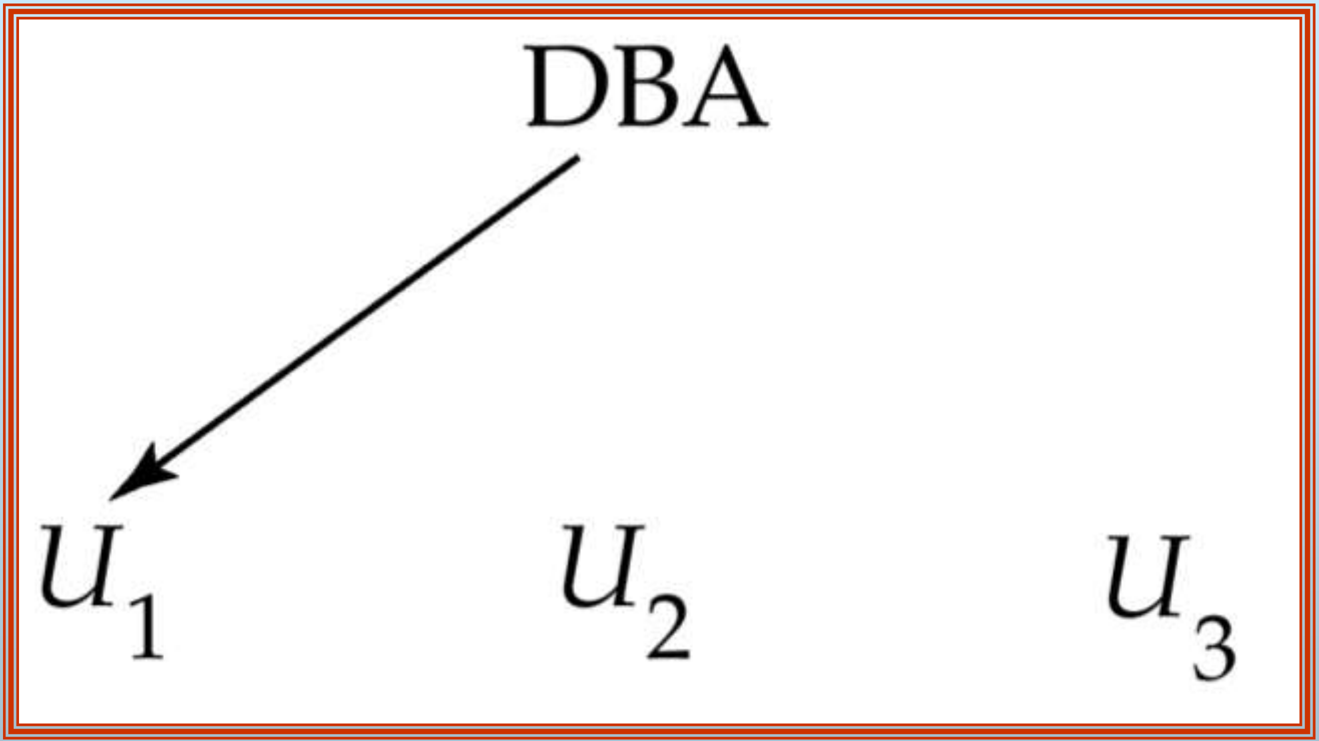
Authorization-Grant Graph

$$\begin{tikzcd}
 & U_1 \arrow[r] \arrow[rd] & U_4 \\
\text{DBA} \arrow[ru] \arrow[r] \arrow[rd] & U_2 \arrow[r] & U_5 \\
 & U_3 &
\end{tikzcd}$$

Attempt to Defeat Authorization Revocation



Authorization Graph

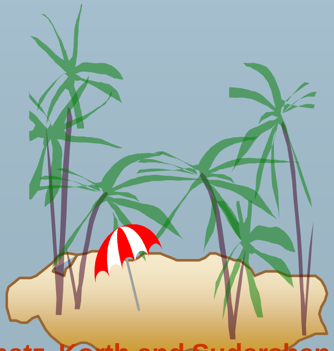

$$\begin{tikzcd}
 & DBA \arrow[dl] & \\
U_1 & U_2 & U_3
\end{tikzcd}$$

Physical Level Security

- Protection of equipment from floods, power failure, etc.
- Protection of disks from theft, erasure, physical damage, etc.
- Protection of network and terminal cables from wiretaps non-invasive electronic eavesdropping, physical damage, etc.

Solutions:

- Replicated hardware:
  - mirrored disks, dual busses, etc.
  - multiple access paths between every pair of devises
- Physical security: locks,police, etc.
- Software techniques to detect physical security breaches.

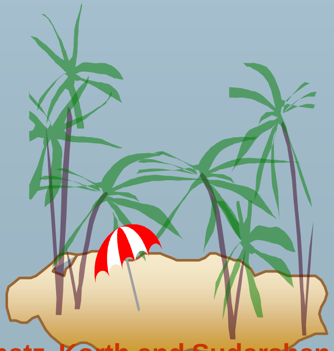


Human Level Security

- Protection from stolen passwords, sabotage, etc.
- Primarily a management problem:
  - Frequent change of passwords
  - Use of “non-guessable” passwords
  - Log all invalid access attempts
  - Data audits
  - Careful hiring practices

Operating System Level Security

- Protection from invalid logins
- File-level access protection (often not very helpful for database security)
- Protection from improper use of “superuser” authority.
- Protection from improper use of privileged machine intructions.



Network-Level Security

- Each site must ensure that it communicate with trusted sites (not intruders).
- Links must be protected from theft or modification of messages
- Mechanisms:
  - Identification protocol (password-based),
  - Cryptography.

Database-Level Security

- Assume security at network, operating system, human, and physical levels.
- Database specific issues:
  - each user may have authority to read only part of the data and to write only part of the data.
  - User authority may correspond to entire files or relations, but it may also correspond only to parts of files or relations.
- Local autonomy suggests site-level authorization control in a distributed database.
- Global control suggests centralized control.