Chapter 6: Integrity and Security

- Domain Constraints
- Referential Integrity
- Assertions
- Triggers
- Security
- Authorization
- Authorization in SQL

Domain Constraints

- Integrity constraints guard against accidental damage to the database, by ensuring that authorized changes to the database do not result in a loss of data consistency.
- Domain constraints are the most elementary form of integrity constraint.
- They test values inserted in the database, and test queries to ensure that the comparisons make sense.
- New domains can be created from existing data types
  - `create domain Dollars numeric(12, 2)`
  - `create domain Pounds numeric(12, 2)`
- We cannot assign or compare a value of type Dollars to a value of type Pounds.
  - However, we can convert type as below
    - `(cast r:A as Pounds)`
    - (Should also multiply by the dollar-to-pound conversion-rate)

Domain Constraints (Cont.)

- The `check` clause in SQL-92 permits domains to be restricted:
  - Use `check` clause to ensure that an hourly-wage domain allows only values greater than a specified value.
    - `create domain hourly-wage numeric(5, 2)`
    - `constraint value-test check (value > = 4.00)`
  - The domain has a constraint that ensures that the hourly-wage is greater than 4.00
  - The clause `constraint value-test` is optional; useful to indicate which constraint an update violated.
  - Can have complex conditions in domain check
    - `check (branch-name in (select branch-name from branch))`

Referential Integrity

- Ensures that a value that appears in one relation for a given set of attributes also appears for a certain set of attributes in another relation.
  - Example: If “Perryridge” is a branch name appearing in one of the tuples in the account relation, then there exists a tuple in the branch relation for branch “Perryridge”.
- Formal Definition
  - Let $r_1(R_1)$ and $r_2(R_2)$ be relations with primary keys $K_1$ and $K_2$ respectively.
  - The subset $\alpha$ of $R_2$ is a foreign key referencing $K_1$ in relation $r_1$, if for every $t_1 \in r_1$ there must be a tuple $t_2 \in r_2$ such that $t_1[K_1] = t_2[\alpha]$.
- Referential integrity constraint also called subset dependency since its can be written as $\Pi_{\alpha}(r_2) = \Pi_{K_1}(r_1)$

Referential Integrity in the E-R Model

- Consider relationship set $R$ between entity sets $E_1$ and $E_2$. The relational schema for $R$ includes the primary keys $K_1$ of $E_1$ and $K_2$ of $E_2$.
  - Then $K_1$ and $K_2$ form foreign keys on the relational schemas for $E_1$ and $E_2$ respectively.
  - Weak entity sets are also a source of referential integrity constraints.
    - For the relation schema for a weak entity set must include the primary key attributes of the entity set on which it depends.

Database Modification (Cont.)

- Update. There are two cases:
  - If a tuple $t_1$ is updated in relation $r_1$ and the update modifies values for foreign key $\alpha$, then a test similar to the insert case is made:
    - Let $t_1$ denote the new value of tuple $t_1$. The system must ensure that:
      - $t_1[\alpha] \in \Pi_{\alpha}(r_1)$
  - If a tuple $t_1$ is updated in $r_1$, and the update modifies values for the primary key $(K)$, then a test similar to the delete case is made:
    1. The system must compute $\sigma_{t_1[K]}(r_2)$
    2. If this set is not empty
      - the update may be rejected as an error, or
      - the update may be cascaded to the tuples in the set, or
      - the tuples in the set may be deleted.

Checking Referential Integrity on Database Modification

- The following tests must be made in order to preserve the following referential integrity constraint:
  - $\Pi_{\alpha}(r_2) = \Pi_{K_1}(r_1)$
- Insert. If a tuple $t_1$ is inserted into $r_1$, the system must ensure that there is a tuple $t_2$ in $r_2$ such that $t_1[K_1] = t_2[\alpha]$. That is $t_1[\alpha] \in \Pi_{\alpha}(r_2)$
- Delete. If a tuple $t_1$ is deleted from $r_1$, the system must compute the set of tuples in $r_2$ that reference $t_1$:
  - $\sigma_{t_1[K]}(r_2)$
    - If this set is not empty
      - either the delete command is rejected as an error, or
      - the tuples that reference $t_1$ must themselves be deleted

Database Modification (Cont.)

- Primary and candidate keys and foreign keys can be specified as part of the SQL `create table` statement:
  - The primary key clause lists attributes that comprise the primary key.
  - The unique key clause lists attributes that comprise a candidate key.
  - The foreign key clause lists the attributes that comprise the foreign key and the name of the relation referenced by the foreign key.
- By default, a foreign key references the primary key attributes of the referenced table
  - foreign key (account-number, references account)
- Short form for specifying a single column as foreign key
  - account-number char (10) references account
- Reference columns in the referenced table can be explicitly specified
  - must be declared as primary/candidate keys
  - foreign key (account-number, references account, account-number)
Referential Integrity in SQL – Example

create table customer
(customer-name char(20),
customer-street char(30),
customer-city char(30),
primary key (customer-name))
create table branch
(branch-name char(15),
branch-city char(30),
assets integer,
primary key (branch-name))

Cascading Actions in SQL

create table account
(account-number char(10),
branch-name char(15),
balance integer,
primary key (account-number),
foreign key (branch-name) references branch)
create table depositor
(customer-name char(20),
account-number char(10),
primary key (customer-name, account-number),
foreign key (account-number) references account,
foreign key (customer-name) references customer)
create table customer
(customer-name, account-number, asset balance)
create table branch
(branch-name, branch-city, assets)

Referential Integrity in SQL (Cont.)

- Alternative to cascading:
  - on delete set null
  - on delete set default
- Null values in foreign key attributes complicate SQL referential integrity semantics, and are best prevented using not null
- If any attribute of a foreign key is null, the tuple is defined to satisfy the foreign key constraint!

Cascading Actions in SQL (Cont.)

- If there is a chain of foreign-key dependencies across multiple relations, with on delete cascade specified for each dependency, a deletion or update at one end of the chain can propagate across the entire chain.
- If a cascading update to delete causes a constraint violation that cannot be handled by a further cascading operation, the system aborts the transaction.
  - As a result, all the changes caused by the transaction and its cascading actions are undone.
  - Referential integrity is only checked at the end of a transaction
  - Intermediate steps are allowed to violate referential integrity provided later steps remove the violation
  - Otherwise it would be impossible to create some database states, e.g.
    - Insert two tuples whose foreign keys point to each other
      - E.g. spouse attribute of relation marriedperson(name, address, spouse)

Cascading Annotations

- An assertion is a predicate expressing a condition that we wish the database always to satisfy.
- An assertion in SQL takes the form
  - create assertion <assertion-name> check <predicate>
- When an assertion is made, the system tests it for validity, and tests it again on every update that may violate the assertion.
  - This testing may introduce a significant amount of overhead; hence assertions should be used with great care.
- Asserting
  - for all X, P(X)
  - is achieved in a round-about fashion using not exists X such that not P(X)

Assertion Example

- The sum of all loan amounts for each branch must be less than the sum of all account balances at the branch.

create assertion sum-constraint check
(not exists (select * from branch
  where (select sum(amount) from loan
    where loan.branch-name = branch.branch-name) >= (select sum(amount) from account
    where loan.branch-name = branch.branch-name)))

Assertion Example

- Every loan has at least one borrower who maintains an account with a minimum balance or $1000.00

create assertion balance-constraint check
(not exists (select * from loan
  where not exists (select * from borrower, depositor, account
    where loan.loan-number = borrower.loan-number
    and borrower.customer-name = depositor.customer-name
    and depositor.account-number = account.account-number
    and account.balance >= 1000))
A trigger is a statement that is executed automatically by the system as a side effect of a modification to the database.

To design a trigger mechanism, we must:
- Specify the conditions under which the trigger is to be executed.
- Specify the actions to be taken when the trigger executes.

Triggers introduced to SQL standard in SQL:1999, but supported even earlier using non-standard syntax by most databases.

Trigger Example in SQL:1999

```sql
create trigger overdraft-trigger after update on account
referencing new row as nrow
for each row
when nrow.balance < 0
begin atomic
    insert into borrower
    (select customer-name, account-number
     from depositor
     where nrow.account-number = depositor.account-number);
    insert into loan values
    (n.row.account-number, nrow.branch-name, 
     n.row.phone-number = ' ');
    update account set balance = 0
    where account.account-number = nrow.account-number
end
```

Statement Level Triggers

- Instead of executing a separate action for each affected row, a single action can be executed for all rows affected by a transaction
  - Use for each statement instead of for each row
  - Use referencing old table or referencing new table to refer to temporary tables (called transition tables) containing the affected rows
  - Can be more efficient when dealing with SQL statements that update a large number of rows

External World Actions

- We sometimes require external world actions to be triggered on a database update
  - E.g. re-ordering an item whose quantity in a warehouse has become small, or turning on an alarm light.
- Triggers cannot be used to directly implement external-world actions, BUT
  - Triggers can be used to record actions-to-be-taken in a separate table
  - Have an external process that repeatedly scans the table, carries out external-world actions and deletes action from table

E.g. Suppose a warehouse has the following tables

- inventory(item, level): How much of each item is in the warehouse
- minlevel(item, level): What is the minimum desired level of each item
- reorder(item, amount): What quantity should we re-order at a time
- orders(item, amount): Orders to be placed (read by external process)

```sql
create trigger reorder-trigger after update of amount on inventory
referencing old row as orow, new row as nrow
for each row
when nrow.level <= (select level
from minlevel
where minlevel.item = orow.item)
and orow.level > (select level
from minlevel
where minlevel.item = orow.item)
begin
    insert into orders
    (select item, amount
     from reorder
     where reorder.item = orow.item)
end
```

Triggers in MS-SQLServer Syntax

```sql
create trigger overdraft-trigger on account
for update
as
if inserted.balance < 0
begin
    insert into borrower
    (select customer-name, account-number
     from depositor, inserted
     where inserted.account-number = depositor.account-number)
    insert into loan values
    (inserted.account-number, inserted.branch-name, 
     inserted.balance)
    update account set balance = 0
    from account, inserted
    where account.account-number = inserted.account-number
end
```
When Not To Use Triggers

- Triggers were used earlier for tasks such as:
  - maintaining summary data (e.g., total salary of each department)
  - replicating databases by recording changes to special relations (called change or delta relations) and having a separate process that applies the changes over to a replica
- There are better ways of doing these now:
  - Databases today provide built-in materialized view facilities to maintain summary data
  - Databases provide built-in support for replication
- Encapsulation facilities can be used instead of triggers in many cases:
  - Define methods to update fields
  - Carry out actions as part of the update methods instead of through a trigger

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 user does 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 of 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:
  ```sql
  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:
  ```sql
  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 replaces a view by the definition of the view.
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 what 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.

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 U1:
  - Grant must be revoked from U2 since U1 no longer has authorization.
  - Grant must not be revoked from U3 since U3 has another authorization path from DBA through U2.
- Must prevent cycles of grants with no path from the root:
  - DBA grants authorization to U1.
  - U7 grants authorization to U1.
  - U7 revokes authorization from U1.
- Must revoke grant U1 to U4 and from U2 to U5 since there is no path from DBA to U4 or to U5 anymore.

Privileges in SQL

- select: allows read access to relation or the ability to query using the view.
  - Example: grant users U1, U2, and U3 select authorization on the branch relation.
- 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.

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).

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 users.
- Roles can be assigned to users, and even to other roles.
- SQL:1999 supports roles:
  - create role:
    - create role role
  - grant select on branch to teller
  - grant update (balance) on account to teller
  - grant all privileges on account to manager
  - grant manager to teller
  - grant teller to alice, bob
  - grant manager to av

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 U1, U2, U3 cascade
  - Revocation of a privilege from a user may cause other users also to lose that privilege; referred to as cascading.
  - We can prevent cascading by specifying restrict.
    - revoke select on branch from U1, U2, U3 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.

Audit Trails

- An audit trail is a log of all changes (inserts/deletes/updates) to the database along with information such as which user performed the change, and when the change was performed.
- Used to track erroneous/fraudulent updates.
- Can be implemented using triggers, but many database systems provide direct support.

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.

Digital Certificates

- **Digital certificates** are used to verify authenticity of public keys.
- Problem: when you communicate with a web site, how do you know if you are talking with the genuine web site or an imposter?
  - Solution: use the public key of the web site
  - Problem: how to verify if the public key itself is genuine?
- Solution:
  - Every client (e.g. browser) has public keys of a few root-level certification authorities
  - A site can get its name/URL and public key signed by a certification authority: signed document is called a certificate
  - Client can use public key of certification authority to verify certificate
  - Multiple levels of certification authorities can exist. Each certification authority:
    - presents its own public-key certificate signed by a higher level authority, and
    - Uses its private key to sign the certificate of other web sites/authorities

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

End of Chapter
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

Authorization Graph

Physical Level Security

- Protection 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 devices.
  - 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 instructions.
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.