Chapter 19: Special Topics

- Security and Integrity
- Standardization
- Performance Benchmarks
- Performance Tuning
- Time In Databases
- User Interfaces
- Active Databases

Security and Integrity

- Integrity – protection from accidental loss of consistency.
  - Crashes during transaction processing
  - Concurrency control.
  - Anomalies caused by the distribution of data over several computers.
  - Logical error that violates assumption of database consistency.
- Security – protection from malicious attempts to steal or modify data.
  - Physical level
  - Human level
  - Operating system level
  - Network
  - Database system level

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 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 communicates 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.

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.
- **Resource 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

- Views – means of providing a user with a “personalized” model of the database.
- Ability of views to hide data serves both to simplify usage of the system and to enhance security.
- 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

- A view is defined in SQL using the **create view** command.

  create view *v* **as** <query expression>

- A bank clerk needs to know the names of the customers of each branch, but is not authorized to see specific loan information. 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** *branch-name, 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.

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.
  - An edge $U_i \rightarrow U_j$ indicates that user U_i has granted update authorization on *loan* to U_j.
  - The root of the graph is the database administrator.
- All edges in an authorization graph must be part of some path originating with the database administrator.

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
- Granting a privilege on a view does not require 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).

Privilege List

- **select**: allows read access to the 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**: restricts a user's ability to declare foreign keys when creating relations

Privilege List (Cont.)

- **all privileges**: used as a short form for all the allowable privileges
- **usage**: In SQL-92; authorizes a user to use a specified domain
- **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** privilege on *branch* and allows U_1 to grant this privilege to others

Revoking Authorization

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

  revoke <privilege list>
  on <relation name or view name> **from** <user list>

- 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**. Prevent cascading by specifying **restrict**:

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

Revoking Authorization (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.

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 a parameter of the algorithm called the *encryption key*.
  - Extremely difficult for an intruder to determine the encryption key.

Encryption (Cont.)

- *Data Encryption Standard* substitutes characters and rearranges their order on the basis of an encryption key which is provided to authorized users via a secure mechanism. Scheme only as secure as the mechanism.
- *Public-key encryption* based on each user having two keys:
  - *public key* – published key used to encrypt data
  - *private key* – key known only to individual user used to decrypt data.
- Must be a encryption scheme that can be made public without making it easy to figure out the decryption scheme.
  - Algorithm for testing whether or not a number is prime.
  - No efficient algorithm is known for finding the prime factors of a number.

Statistical Databases

- Problem: how to ensure privacy of individuals while allowing use of data for statistical purposes.
- Solutions:
  - System rejects any query that involves fewer than some predetermined number of individuals.
  - *Data pollution* – random falsification of data provided in response to a query.
  - Random modification of the query itself.
- Tradeoff between accuracy and security.

Standardization

- The complexity of contemporary database systems and the need for their interoperation require a variety of standards.
  - syntax and semantics of programming languages
  - functions in application program interfaces
  - data models (i.e., object oriented database standards)
- *Formal standards* are standards developed by a standards organization (ANSI, ISO), or by industry groups, through a public process.

  Formal standards exist for SQL.
- De facto standards are generally accepted as standards without any formal process of recognition; they have played an important role in the growth of client-server database systems.
- Industry groups are developing standards for O-O databases.

Performance Benchmarks

- Suites of tasks used to quantify the performance of software systems; important in comparing database systems, especially as systems become more standards compliant.
- Database-application classes:
  - Online transaction processing (OLTP) requires high concurrency and clever techniques to speed up commit processing to support a high rate of update transactions.
  - Decision support (also called online analytical processing, or (OLAP) applications require good query evaluation algorithms and query optimization.

Performance Benchmarks (Cont.)

- The TPC benchmark suites are widely used. Benchmarks such as average transactions-per-second and transactions-per-second-per-dollar are useful for comparing the performance of databases under different workloads.
- OODB transactions require a different set of benchmarks.

  The OO7 benchmark provides a set of numbers, containing a separate benchmark number for each kind of operation from a set of different kinds of operations.

Performance Tuning

- Adjusting various parameters and design choices to improve system performance for a specific application.
- Tuning is best done by identifying bottlenecks, and eliminating them.
- Bottlenecks in a database system typically show up as very high utilizations for a particular service.
- Can tune a database system at 3 levels:
  - Hardware – add disks, add memory, move to a faster processor.
  - Database system parameters – e.g., buffer size, checkpointing intervals; system may have automatic tuning.
  - Higher level database design, such as the schema, indices and transactions.

Performance Tuning (Cont.)

- Schema tuning
  - Vertically partition relations to isolate the data that is accessed most often – only fetch needed information.
  - Improve performance by storing a denormalized relation for frequently required joins.
  - Cluster records that would match in a frequently required join together on the same disk page; compute join very efficiently when required.

Performance Tuning (Cont.)

- Indices tuning
  - Create appropriate indices on bottleneck queries.
  - Speed updates by removing excess indices.
  - Choose type of index appropriate for most frequent types of queries.
- Transaction tuning
  - Combine frequent calls into a single set-oriented query.
  - Stored procedures.
  - Limit number of updates that a single transaction can carry out.
  - *Mini-batch* transactions.

Time In Databases

- While most databases tend to model reality at a point in time (at the "current" time), temporal databases model the states of the real world across time.
- Facts in temporal relations have associated times when they are *valid*, which can be represented as a union of intervals.
- The *transaction time* for a fact is the time interval during which the fact is current within the database system.
- In a *temporal relation*, each tuple has an associated time when it is true; the time may be either valid time or transaction time.
- Temporal query languages have been proposed to simplify modeling of time as well as time related queries.

Time Specification in SQL-92

- **date**: four digits for the year (1–9999), two digits for the month (1–12), and two digits for the date (1–31).
- **time**: two digits for the hour, two digits for the minute, and two digits for the second, plus optional fractional digits.
- **timestamp**: the fields of **date** and **time**, with six fractional digits for the seconds field.
- Times are specified in the *Universal Coordinated Time*, abbreviated UTC (from the French); supports **time with time zone**.
- **interval**: refers to a period of time (e.g., 2 days and 5 hours), without specifying a particular time when this period starts; could more accurately be termed a *span*.

Temporal Query Languages

- A temporal relation at time t consists of the tuples that are true at time t, with the time-interval attributes projected out.
- *Temporal selection*: involves time attributes.
- *Temporal projection*: the tuples in the projection inherit their times from the tuples in the original relation
- *Temporal join*: the time of a tuple in the result is the intersection of the times of the tuples from which it is derived.
- *Precedes*, *overlaps*, and *contains* can be applied on intervals.
- *Intersect* can be applied on two intervals, to give a single (possibly empty) interval; the union of two intervals may or may not be a single interval.
- TSQL2 extends SQL-92 to improve support of temporal data.

User Interfaces

- Categories of user interfaces to a database:
  - Line-oriented interfaces: most basic interface; good for ad hoc querying, but not for repetitive tasks.
  - Forms interfaces: used to construct forms in a simple declarative manner.
  - Report writers: used for generating reports based on data in the database.
  - Graphical user interfaces: point-and-click paradigm using features such as menus and icons; Web interfaces based on HTML.
- 4GLs: collections of application-development tools (e.g., forms packages, report writers)

Active Databases

- Support the specification and execution of rules in the database.
- *Event–condition–action* model:

  on *event*
  if *condition*
  then *action*

  - Rules are triggered by events
  - Database system checks the conditions of the triggered rules; if the conditions are satisfied, the database system executes the specified actions.
- Rules used for diverse purposes (e.g. alerting users to unusual activity, reordering stock, enforcing integrity constraints).

Active Databases (Cont.)

- We must ensure that the rule set will terminate, if the action of a rule can cause an event that triggers the same rule.
- Multiple rules are executed in the order of their priority value.
- Event-execution binding specifies when a rule gets executed (e.g., *immediate*, *deferred*, *decoupled*).
- Error handling with immediate or deferred event-execution binding is handled as part of transaction recovery.
- Because error recovery for decoupled executions is so difficult, the rule system should be designed such that run-time errors do not occur during the execution of decoupled rules.