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*.

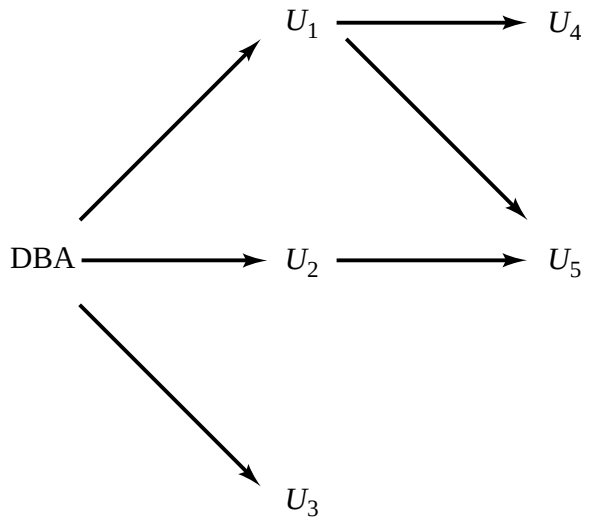
- Authorization must be checked on the clerk's query before query processing begins.

Authorization on Views

- Creation of view does not require **resource** authorization
- 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.



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
  - U_7 grants authorization to U_8
  - U_8 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
- 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 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**: ability to declare foreign keys when creating relations

Privileges in SQL (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 in SQL

- 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**

  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.

SQL-3 Extensions

- SQL-3 provides notion of **roles**
- Privileges can be granted to or revoked from roles, just like users
- Roles can be assigned to users, and to other roles
- E.g.

create role *teller*
create role *manager*
grant select on *branch* **to** *teller*
grant *teller* **to** *alice*
grant *teller* **to** *manager*

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* 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.
- *Public-key encryption* based on each user having two keys:
  - *public key* – 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.

  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.

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.

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.
- **De facto standards** are generally accepted as standards without any formal process of recognition
  - Standards defined by dominant vendors (IBM, Microsoft) often become de facto standards

Standardization (Cont.)

- **Anticipatory standards** lead the market place, defining features that vendors then implement
  - Ensure compatibility of future products
  - But at times become very large and unwieldy since standards bodies may not pay enough attention to ease of implementation (e.g., SQL-92 or the forthcoming SQL-3)
- **Reactionary standards** attempt to standardize features that vendors have already implemented, possibly in different ways.
  - Can be hard to convince vendors to change already implemented features
  - De facto standards often go through a formal process of recognition and become formal standards

SQL Standards History

- SQL developed by IBM in late 70s/early 80s
- SQL-86 first formal standard
- IBM SAA standard for SQL in 1987
- SQL-89 added features to SQL-86 that were already implemented in many systems (reactionary standard)
- SQL-92 added many new features to SQL-89 (anticipatory standard)
  - Defines levels of compliance (*entry, intermediate* and *full*)
  - Even as of 1997, few database vendors had full SQL-92 implementation
- SQL-3 standard currently under development
  - Adds variety of new features — extended data types, object orientation, procedures, triggers, multimedia, etc.

Other Standards

- Microsoft *Open DataBase Connectivity* (ODBC) standard for database interconnectivity
  - based on *Call Level Interface* (CLI) developed by X/Open consortium
  - defines application programming interface, and SQL features that must be supported at different levels of compliance
- *X/Open XA* standards define transaction management standards for supporting distributed 2-phase commit

Object Oriented Database Standards

- *Object Database Management Group* (ODMG) standard for object-oriented databases
  - version 1 in 1993 and version 2 in 1997
  - provides language independent *Object Definition Language* (ODL) as well as several language specific *bindings* (Chapter 8)
- *Object Management Group* (OMG) standard for distributed software based on objects
  - *Object Request Broker* (ORB) provides transparent message dispatch to distributed objects
  - *Interface Definition Language* (IDL) for defining language-independent data types
  - *Common Object Request Broker Architecture* (CORBA) defines specifications of ORB and IDL

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 applications (including online analytical processing, or (OLAP applications) require good query evaluation algorithms and query optimization.

Performance Benchmarks (Cont.)

- Commonly used performance measures:
  - **Throughput** (transactions per second, or tps)
  - **Response time** (delay from submission of transaction to return of result)
  - **Availability** or **mean time to failure**
- Suites of tasks used to characterize performance (single task not enough for complex systems)
- Must not compute average of throughput
  - E.g., suppose a system runs transaction type A at 99 tps and transaction type B at 1 tps. Given an equal mixture of types A and B, throughput is **not** (99 + 1)/2 = 50 tps.
  - Running one transaction of each type takes time $1 + \frac{1}{99}$ seconds, giving a throughput of 1.98 tps.
  - Must use harmonic mean: $\frac{n}{1/t_1 + 1/t_2 + \ldots + 1/t_n}$

Benchmark Suites

- The Transaction Processing Council (TPC) benchmark suites are widely used.
  - TPC-A: simple OLTP application modeling a bank teller application, with end-to-end measurements, including communication with terminals
  - TPC-B: same teller application, but without communication
  - TPC-C: more complex OLTP application modeling an inventory system
  - TPC-D: complex decision support application (involves plenty of aggregation)
- TPC performance measures
  - *transactions-per-second* with specified constraints on response time
  - *transactions-per-second-per-dollar* accounts for cost of owning system

Benchmark Suites (Cont.)

- TPC benchmark database sizes scale up with increasing transactions-per-second, to reflect real world applications.
- External audit of TPC performance numbers mandatory (TPC performance claims can be trusted)
- OODB transactions require a different set of benchmarks.
  - OO7 benchmark has several different operations, and provides a separate benchmark number for each kind of operation.

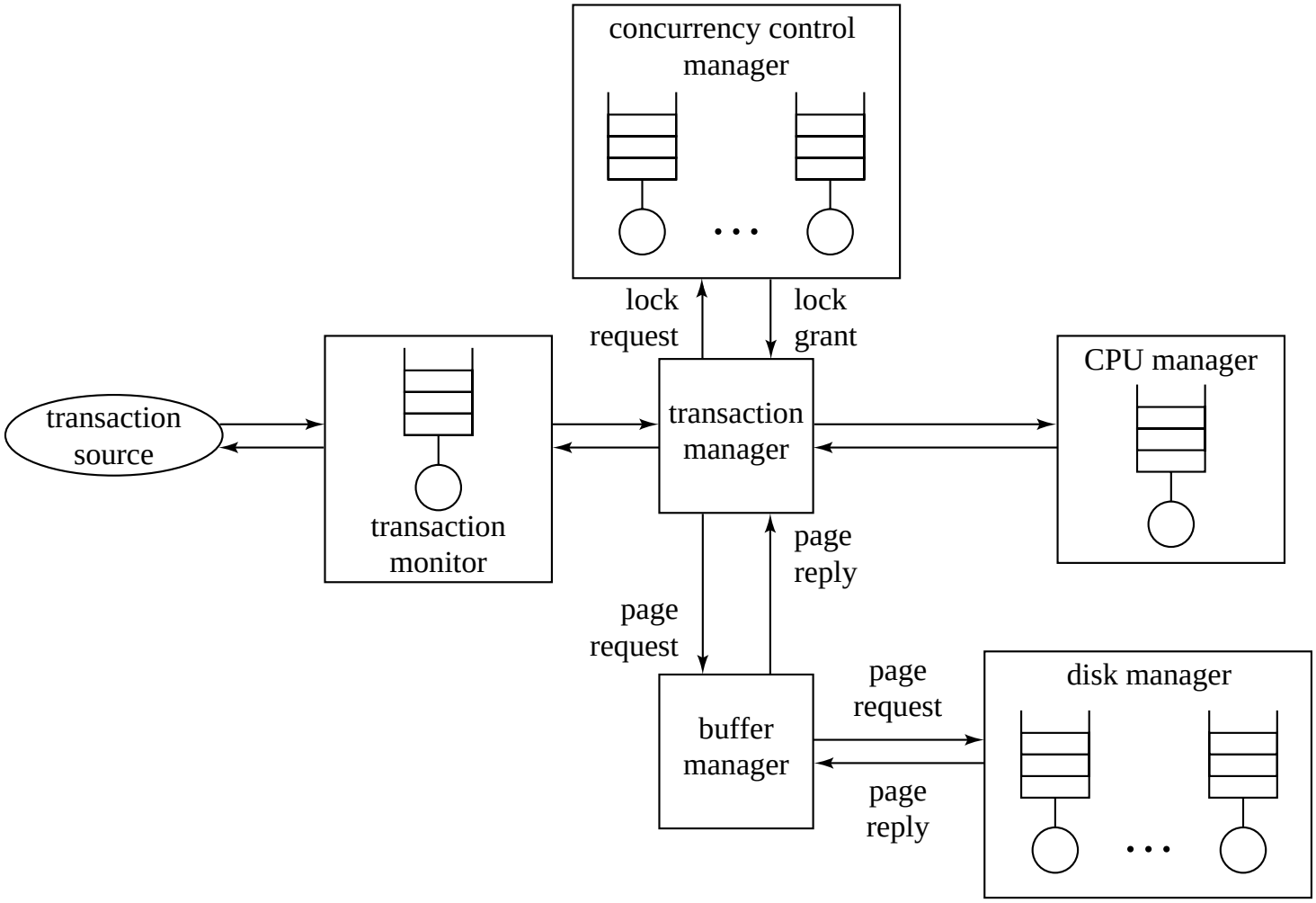
Performance Tuning

- Adjusting various parameters and design choices to improve system performance for a specific application.
- Tuning is best done by identifying bottlenecks, and eliminating them.
- Can tune a database system at 3 levels:
  - **Hardware** – e.g., add disks to speed up I/O, add memory to increase buffer hits, move to a faster processor.
  - **Database system parameters** – e.g., set buffer size to avoid paging of buffer, set checkpointing intervals to limit log size. System may have automatic tuning.
  - **Higher level database design**, such as the schema, indices and transactions (more later)

Identifying Bottlenecks

- Databases are complex systems
- Transactions request a sequence of services (e.g. CPU, Disk I/O, locks)
- With concurrent transactions, transactions may have to wait for a requested service while other transactions are being served
- Can model database as a *queueing system* with a queue for each service; transactions repeatedly do the following: request a service, wait in queue for the service, and get serviced
- Bottlenecks in a database system typically show up as very high utilizations (and correspondingly, very long queues) for a particular service.
- **Performance simulation** using queueing model useful to predict bottlenecks as well as the effects of tuning changes, even without access to real system

Queues in a Database System



Tuning the Database Design

- **Schema tuning**
  - Vertically partition relations to isolate the data that is accessed most often – only fetch needed information.
    - E.g., split *account* into two, one having (*account-number, branch-name*) pairs, and the other having (*account-number, balance*) pairs. Branch-name need not be fetched unless required
  - Improve performance by storing a denormalized relation
    - E.g., store join of *account* and *depositor*; branch-name and balance information is repeated for each holder of an account, but join need not be computed repeatedly.
  - Cluster records that would match in a frequently required join together on the same disk page; compute join very efficiently when required.

Tuning the Database Design (Cont.)

- **Index tuning**
  - Create appropriate indices to speed up slow queries
  - Speed up slow updates by removing excess indices (tradeoff between queries and updates)
  - Choose type of index (B-tree/hash) appropriate for most frequent types of queries.

Tuning the Database Design (Cont.)

- **Transaction tuning**
  - Combine frequent calls into a single set-oriented query: fewer calls to database
  - Use stored procedures: avoid need to re-parse and re-optimize query
  - Use *mini-batch* transactions to limit number of updates that a single transaction can carry out. E.g., if a single large transaction updates every record of a very large relation, log may grow too big.
    - Split large transaction into batch of "mini-transactions," each performing part of the updates
    - Hold locks across transactions in a mini-batch to ensure serializability
    - In case of failure during a mini-batch, must complete its remaining portion on recovery, to ensure atomicity.

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.
- A *bi-temporal relation* stores both valid and transaction time.

Time In Databases (Cont.)

- Example of a temporal relation:

branch-name	*account-number*	*balance*	*from*		*to*	
Downtown	A-101	500	94/1/1	9:00	94/1/24	11:30
Downtown	A-101	100	94/1/24	11:30	*	
Mianus	A-215	700	95/6/2	15:30	95/8/8	10:00
Mianus	A-215	900	95/8/8	10:00	95/9/5	8:00
Mianus	A-215	700	95/9/5	8:00	*	
Brighton	A-217	750	94/7/5	11:00	95/5/1	16:00

- 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

- Predicates *precedes*, *overlaps*, and *contains* can be applied on time 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.
- A **snapshot** of 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 time-intervals from the tuples in the original relation
- **Temporal join**: the time-interval of a tuple in the result is the intersection of the time-intervals of the tuples from which it is derived. If intersection is empty, tuple is discarded from join.

Temporal Query Languages (Cont.)

- Functional dependencies must be used with care: adding a time field may invalidate functional dependency
- A **temporal functional dependency** $X \xrightarrow{\tau} Y$ holds on a relation schema R if, for all legal instances r of R, all snapshots of r satisfy the functional dependency $X \rightarrow Y$.
- TSQL2 is a proposed extension to SQL-92 to improve support of temporal data.

User Interfaces

- Several categories of user interfaces to a database:
- **Line-oriented interfaces**: most basic interface; good for ad hoc querying using, e.g., SQL, but not good for repetitive tasks such as data entry.
- **Forms interfaces**
  - Widely used for data entry and other similar repetitive tasks.
  - Actions can be associated with user inputs. E.g., entering customer number may fetch and fill in customer name and address.
  - Simple error checks can be performed in the form interface
  - Form editor programs/packages allow forms to be created in a simple declarative manner, without programming.
- **Graphical user interfaces**: point-and-click paradigm using features such as menus and icons; Web interfaces based on HTML.

An Order Entry Form

Acme Supply Company Inc. **Component Order Form**

Customer Name: _____ Date: _____

Customer Number: _____ Order No.: _____

Address: _____

Phone Number: _____

Part Number	Part Description	Unit Price	Quantity	Subtotal
			Total	

User Interfaces (Cont.)

- *Report writers*: used for generating periodic reports based on data in the database.

Acme Supply Company Inc.
Quarterly Sales Report

Period: Jan. 1 to March 31, 1996

Region	Category	Sales	Subtotal
North	Computer Hardware	1,000,000	
	Computer Software	500,000	
	All categories		1,500,000
South	Computer Hardware	200,000	
	Computer Software	400,000	
	All categories		600,000

Total Sales 2,100,000

- *Fourth Generation Languages* (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.

Active Databases (Cont.)

- Rules used for diverse purposes (e.g. alerting users to unusual activity, reordering stock, enforcing integrity constraints).
- Example of a trigger to enforce the constraint
  "salary of employee $<$ salary of manager":

define trigger *employee-sal*
on insert *employee*
if *employee.salary* $>$
(**select** *E.salary* **from** *employee* **as** *E*
where *E.name = employee.manager*)
then *abort*

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:
  - *immediate*: as soon as event occurs
  - *deferred*: at end of transaction, before it commits
  - *decoupled*: some time after transaction has completed
- Error handling with immediate or deferred event-execution binding is handled as part of transaction recovery.
- Error recovery for decoupled executions is very difficult — the rule system should be designed such that run-time errors do not occur during the execution of decoupled rules.