# Practical Database Design Methodology and Use of UML Diagrams

## 406.426 Design & Analysis of Database Systems

Jonghun Park

jonghun@snu.ac.kr

Dept. of Industrial Engineering

Seoul National University

# chapter outline

- information system life cycle
- phases of database design
- UML diagrams
  - Rational Rose
  - other tools
- design tools

# IT as a key to successful business

- data is regarded as a **corporate resource**, and its management and control is considered central to the effective working of an organization
- more functions in organizations are computerized, increasing the need to keep **large volumes** of data available in an **up-to-the-minute** current state
- as the complexity of the data and applications grows, **complex relationships** among the data need to be modeled and maintained
- there's a tendency toward **consolidation** of information resources in many organizations
- many organizations are reducing their personnel costs by letting the **end-user perform business transactions**

# characteristics of database systems

- **data independence** from changes in the underlying logical organization and in the physical access paths and storage structures
- **external schemas** that allow the same data to be used for multiple applications
- **integration** of data across multiple applications into a single DB
- **simplicity** of developing new applications using high-level languages like SQL
- possibility of supporting **casual access** for browsing and querying by managers while supporting major production-level **TP**

# trends in DB systems

- personal DBs is gaining popularity
  - Excel, MySQL, Access, …
  - check-out and check-in
- advent of distributed & client-server DBMSs
  - for better local control and faster local processing
  - emergence of Web-based applications
- using data dictionary systems (or information repositories)
  - data about DB
  - DB structure, constraints, applications, authorizations, …
- **performance-critical TP** systems
  - around-the-clock nonstop operation
  - hundreds of transactions per min.

# information system life cycle

- feasibility analysis
  - cost-benefit studies, setting up priorities, scopes,…
- requirements collection and analysis
  - interacting with potential users
- design
  - design of DB system, design of application systems
- implementation
- validation and acceptance testing
  - against performance criteria and behavior specifications
- deployment, operation and maintenance
  - new requirements or applications crop up

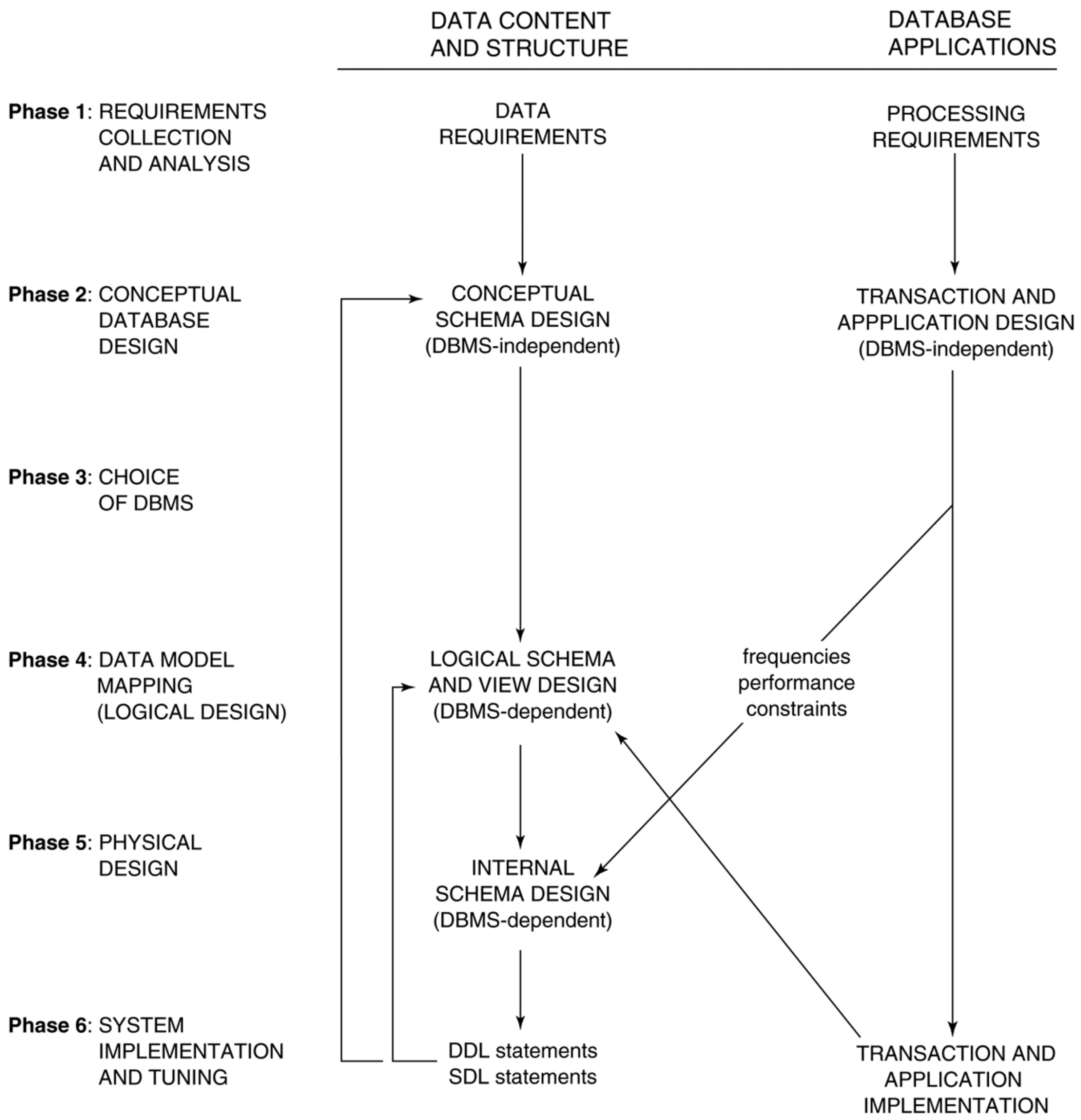
# DB application system life cycle

- system definition
- DB design
- DB implementation
- loading or data conversion
  - time consuming
- application conversion
  - time consuming
- testing and validation
- operation
  - usually the old and the new systems are operated in parallel for some time
- monitoring and maintenance

# database design

- problem definition: design the **logical** and **physical** structure of one or more databases to accommodate the **information needs** of the users in an organization for a defined set of applications
- goals
  - satisfy the information **content requirements**
  - provide a natural and easy-to-understand **structuring of information**
  - support **processing requirements** and any **performance objectives**
  - **tradeoff** between "understandability" and "performance"

# phases of DB design and implementation



# phase 1: requirements collection and analysis

- major activities
  - **application areas** and **user groups** are identified
  - existing **documentation** concerning the applications is analyzed
  - current **operating environment** and planned use of the information is studied
    - types of transactions and their frequencies, the flow of information, geographic characteristics, origin of transactions, destination of reports, input and output data for the transactions, …
  - written responses to sets of questions are sometimes collected from the potential DB users
- requirements are **subject to change**!
  - JAD (Joint Application Design)
  - contextual design

# phase 1: requirements collection and analysis

- requirement specification techniques
  - diagramming techniques
    - OOA
    - DFD
  - formal specification methods
    - e.g., Z
    - hardly used
- upper CASE tools
  - help check the consistency and completeness of specifications
- **correcting a requirement error is much more expensive than correcting an error made during implementation**

# phase 2: conceptual DB design

- involves two parallel activities: **conceptual schema design** and **transaction** and **application design**
- conceptual schema design is **DBMS-independent** because
  - complete understanding of the DB structure, semantics, interrelationships, and constraints can be best achieved **independently of a specific DBMS**
  - choice of DBMS and later design decision may **change**
  - high-level data model is **more expressive** and general than the data models of individual DBMS
  - diagrammatic description of the conceptual schema can serve as an excellent **vehicle of communication** among database users, designers, and analysts

# phase 2: conceptual DB design

- desired characteristics of a conceptual data model
  - expressiveness
  - simplicity and understandability
  - minimality
  - diagrammatic representation
  - formality
- the above characteristics usually result in conflicts
- output
  - entity types, relationship types, attributes
  - key attributes, cardinality and participation constraints on relationships, weak entity types, specialization/generalization hierarchies, …

# phase 2: conceptual DB design

- approaches to conceptual schema design
  - centralized (or one-shot) schema design approach
    - **requirements** of the different applications and user groups from Phase 1 are **merged** into a **single set of requirements** before schema design begins
    - **single schema** corresponding to the merged set of requirements is then designed
  - view integration approach
    - schema is designed for each user group or application based only on its own requirements
    - during a subsequent **view integration** phase, the **schemas are merged or integrated** into a global conceptual schema for the entire DB
    - more popular

# phase 2: conceptual DB design

- strategies for schema design
  - top-down strategy
    - start with a schema containing **high-level abstractions** and then apply successive top-down refinements
  - bottom-up strategy
    - start with a schema containing **basic abstractions** and then combine or add to these abstractions
  - inside-out strategy
    - special case of a bottom-up strategy, where attention is focused on a **central set of concepts** that are most evident
    - modeling then spreads outward by considering new concepts in the vicinity of existing ones
  - mixed strategy
    - requirements are partitioned according to a top-down strategy, and part of the schema is designed for each partition according to a bottom-up strategy

# example of top-down refinement

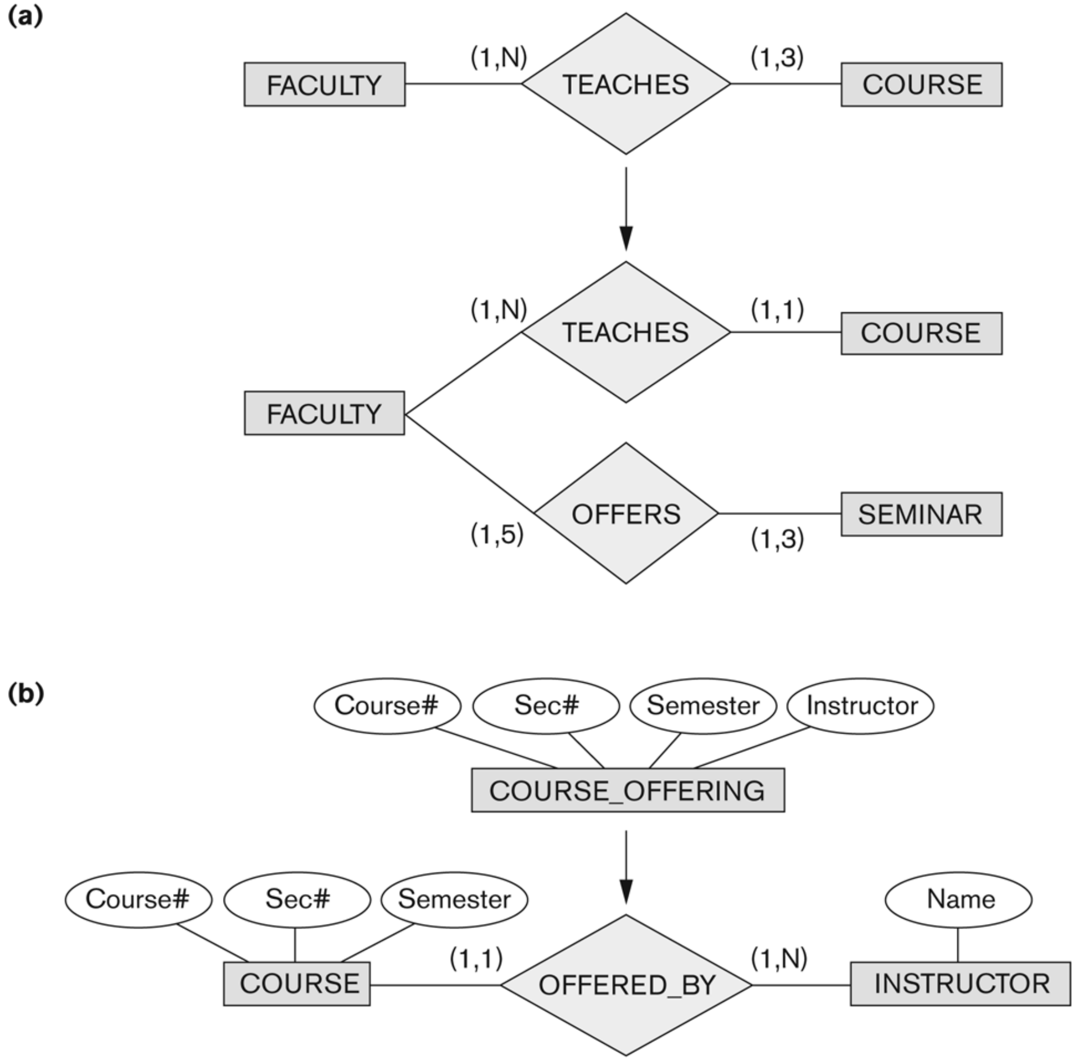



**Figure 12.2**
Examples of top-down refinement. (a) Generating a new entity type.
(b) Decomposing an entity type into two entity types and a relationship type.

# example of bottom-up refinement

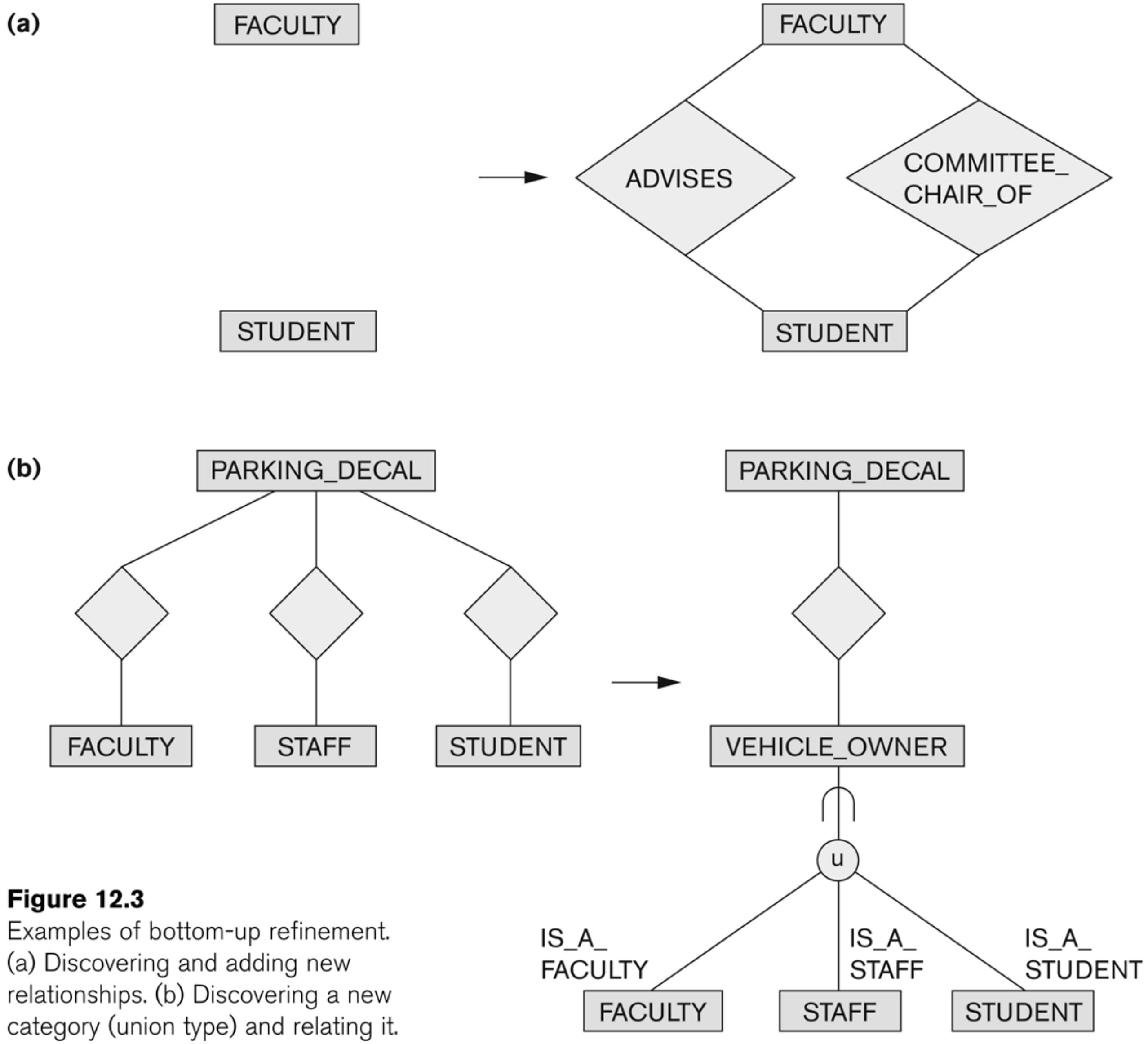
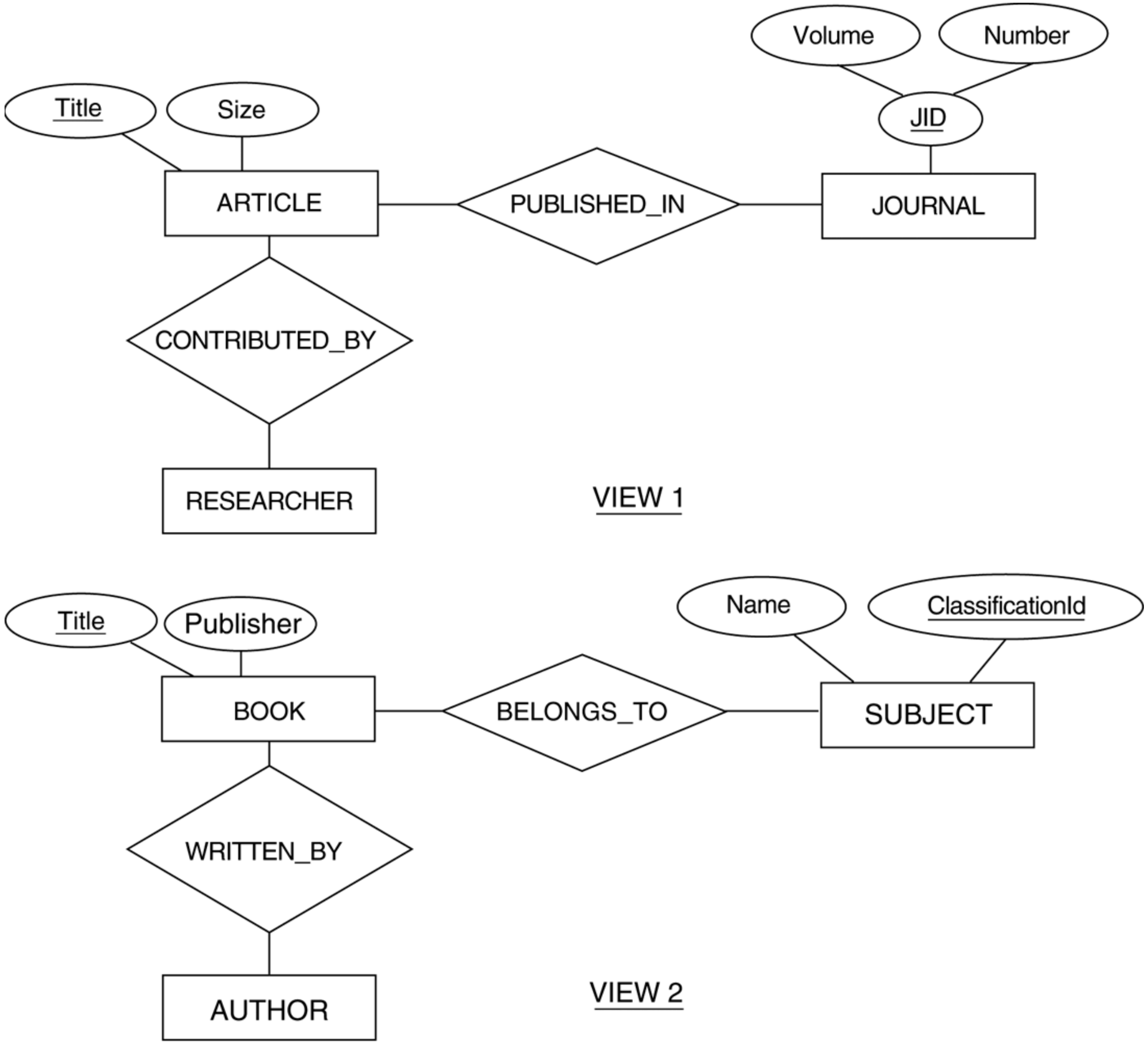



**Figure 12.3**
Examples of bottom-up refinement. (a) Discovering and adding new relationships. (b) Discovering a new category (union type) and relating it.

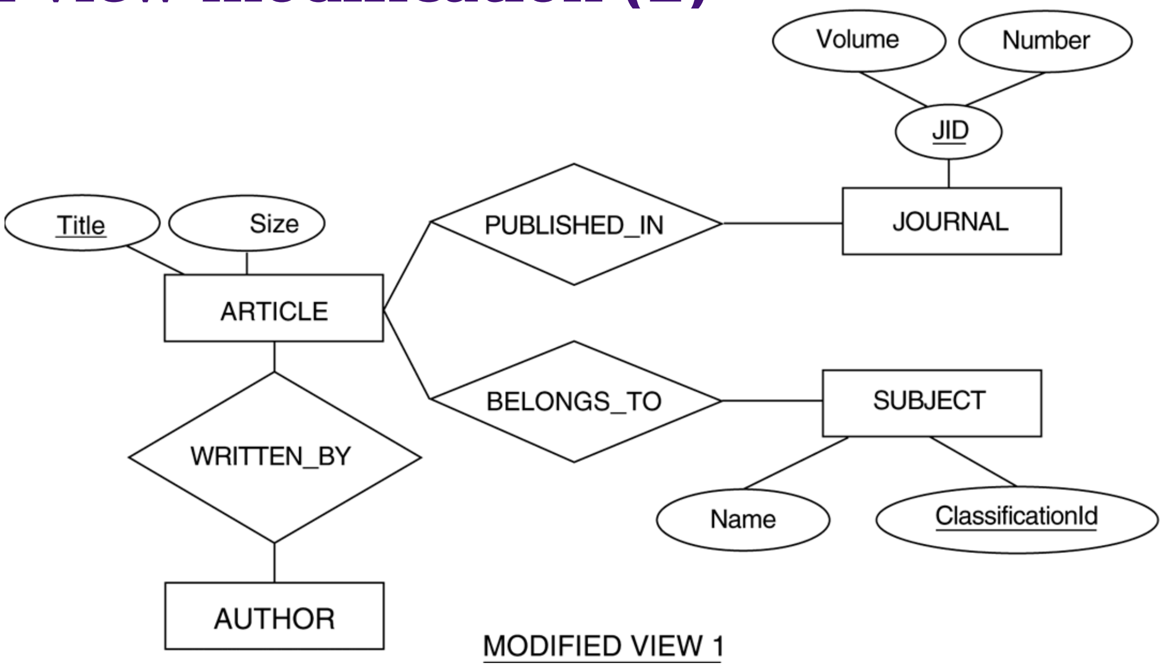
# phase 2: conceptual DB design

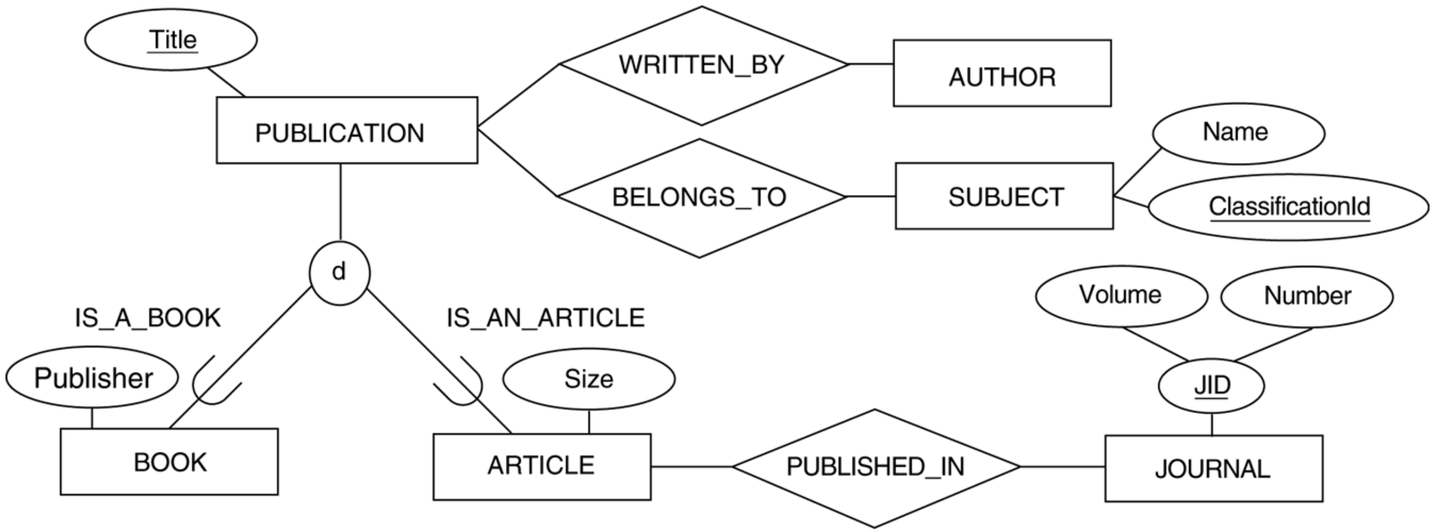
- schema integration
  - identifying **correspondences** and **conflicts** among the schemas
    - naming conflicts: synonyms, homonyms
    - type conflicts: e.g., entity vs. attribute
    - domain conflicts
    - conflicts among constraints
  - modifying views to conform to one another
  - merging of views
    - involves a considerable amount of human intervention and negotiation to **resolve conflicts**
  - restructuring
    - to remove any redundancies and unnecessary complexity

# example of view modification (1)



# example of view modification (2)

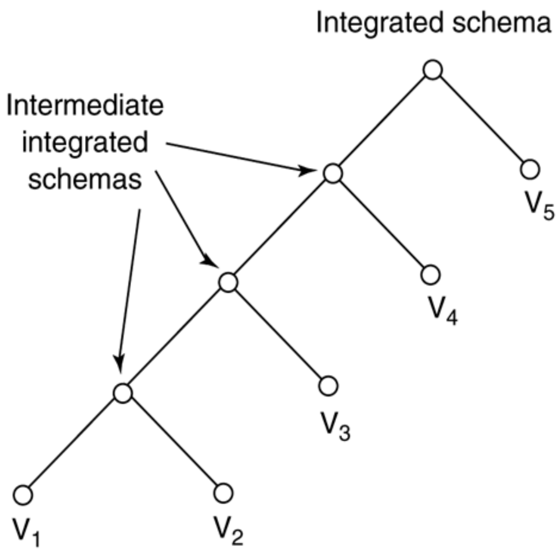


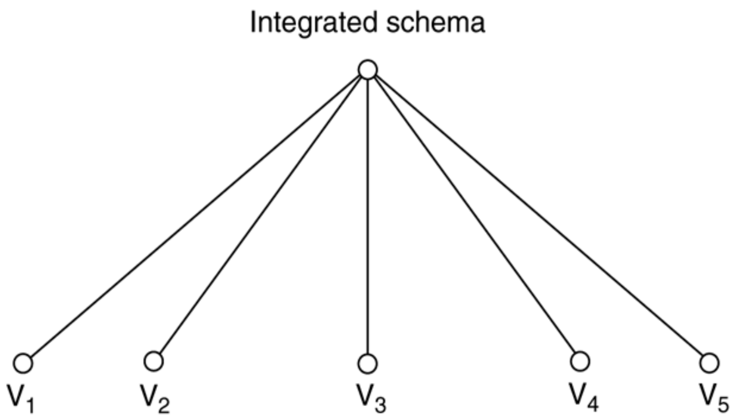
# phase 2: conceptual DB design

- strategies for the view integration process
  - binary ladder integration
    - 2 schemas that are quite **similar** are integrated first
  - N-ary integration
    - all the views are integrated in one procedure
  - binary balanced strategy
    - pairs of schemas are integrated first, then the resulting schemas are paired for further integration
  - mixed strategy
    - schemas are **partitioned** into groups based on their similarity, and each group is integrated separately

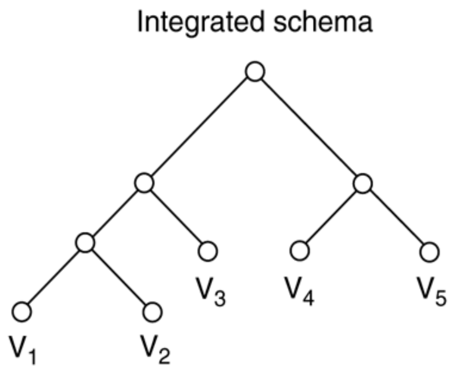
# different strategies for the view integration



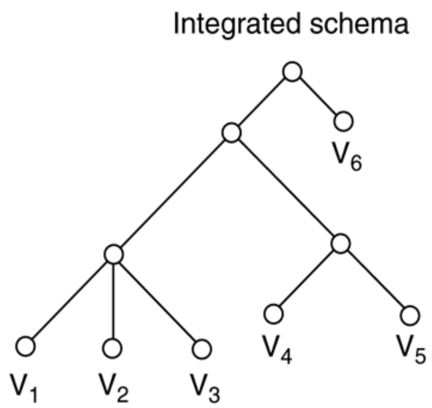

**Binary Ladder Integration**




**N-ary Integration**




**Binary Balanced Integration**




**Mixed Integration**

# phase 2: conceptual DB design

- phase 2b: transaction design
  - to design the **functional characteristics** of known DB transactions (applications) in a **DBMS-independent** way
  - **80-20 rule**: 80 % of the workload is represented by 20 % of the most frequently used transactions
  - identifying the transaction's I/O
    - retrieval, update, and mixed transactions
  - identifying the transaction's functional behavior
    - notation for specifying processes
    - activities, events, operations, sequencing, synchronizations, …
    - still remains an active area of research

# phase 3: choice of a DBMS

- technical considerations
  - type of DBMS, the storage structures and access paths, UI, APIs, the types of high-level languages, availability of development tools, ability to interface with other DBMSs, architectural options related to CS operation, DBMS portability
- nontechnical considerations
  - financial status and the support organization of the vendor, availability of vendor services, organization-wide adoption of a certain philosophy, familiarity of personnel with the system
- economic considerations
  - software acquisition cost, maintenance cost, hardware acquisition cost, DB creation and conversion cost, personnel cost, training cost, operating cost

# phase 3: choice of a DBMS

- drivers for DBMS
  - data complexity, data sharing among applications, dynamically evolving or growing data, frequency of ad hoc requests for data, data volume and need for control
- common built-in features of DBMSs
  - text editors and browsers
  - report generators and listing utilities
  - communication software
  - data entry and display features such as forms, screens, and menus with automatic editing features
  - inquiry and access tools that can be used on WWW
  - graphical DB design tools

# phase 4: data model mapping

- to create a **conceptual schema** and **external schemas** in the data model of the selected DBMS
- two stages
  - **system-independent mapping**: e.g., EER -> relational schemas
  - tailoring the schemas to a specific DBMS
- result: **DDL statements** in the language of the chosen DBMS that specify the conceptual and external level schemas of the DB system
- many automated CASE design tools can generate DDL from a conceptual schema design

# phase 5: physical database design

- process of choosing specific **storage structures** and **access paths** for the DB files to achieve good **performance** for the various DB applications
- usually include various types of indexing, clustering of related records on disk blocks, linking related records via pointers, and various types of hashing
- frequently used criteria
  - response time
    - elapsed time between submitting a DB transaction for execution and receiving a response
  - space utilization
    - amount of storage space used by the DB files and their access path structures on disk
  - transaction throughput
    - average # of transactions processed per min
- cf. benchmark test

# phase 6: DB system implementation and tuning

- typically the responsibility of the DBA and is carried out in conjunction with the DB designers
- language statements in DDL including SDL of the selected DBMS are compiled and used to create the DB schemas and DB files
- DB can then be loaded (populated) with the data
- conversion routines may be needed
- DB transactions must be implemented by the application programmers, and then writing and testing program code with **embedded DML commands**

# UML as a design specification standard

- even though its concepts are based on object-oriented techniques, the resulting models of structure and behavior can be used to design both relational, object-oriented, and object-relational DBs
- UML defines 9 types of diagrams
  - **structural** diagrams
    - describe the structural or static relationships among components
    - class diagram, object diagram, component diagram, and deployment diagram
  - **behavioral** diagrams
    - describe the behavioral or dynamic relationships among components
    - use case diagram, sequence diagram, collaboration diagram, statechart diagram, and activity diagram

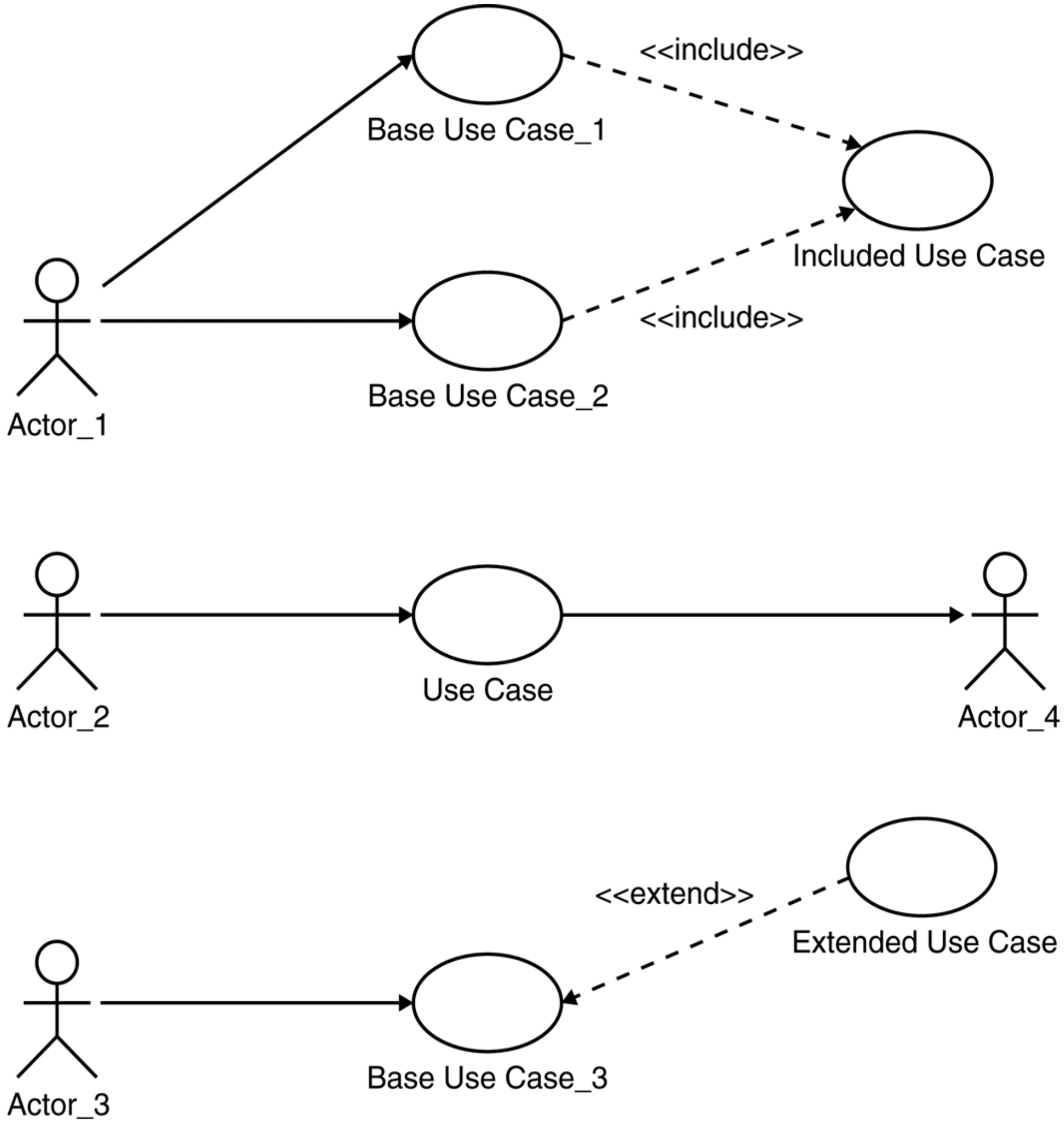
# UML diagrams

- class diagrams
  - capture the **static structure** of the system and act as foundation for other models
  - show classes, interfaces, collaborations, dependencies, generalizations, association and other relationships
- object diagrams
  - show a set of objects and their relationships
  - correspond to instance diagrams
- component diagrams
  - illustrate the organizations and dependencies among software components
  - consists of components, interfaces, and dependency relationships
- deployment diagrams
  - represent the distribution of components across the hardware topology

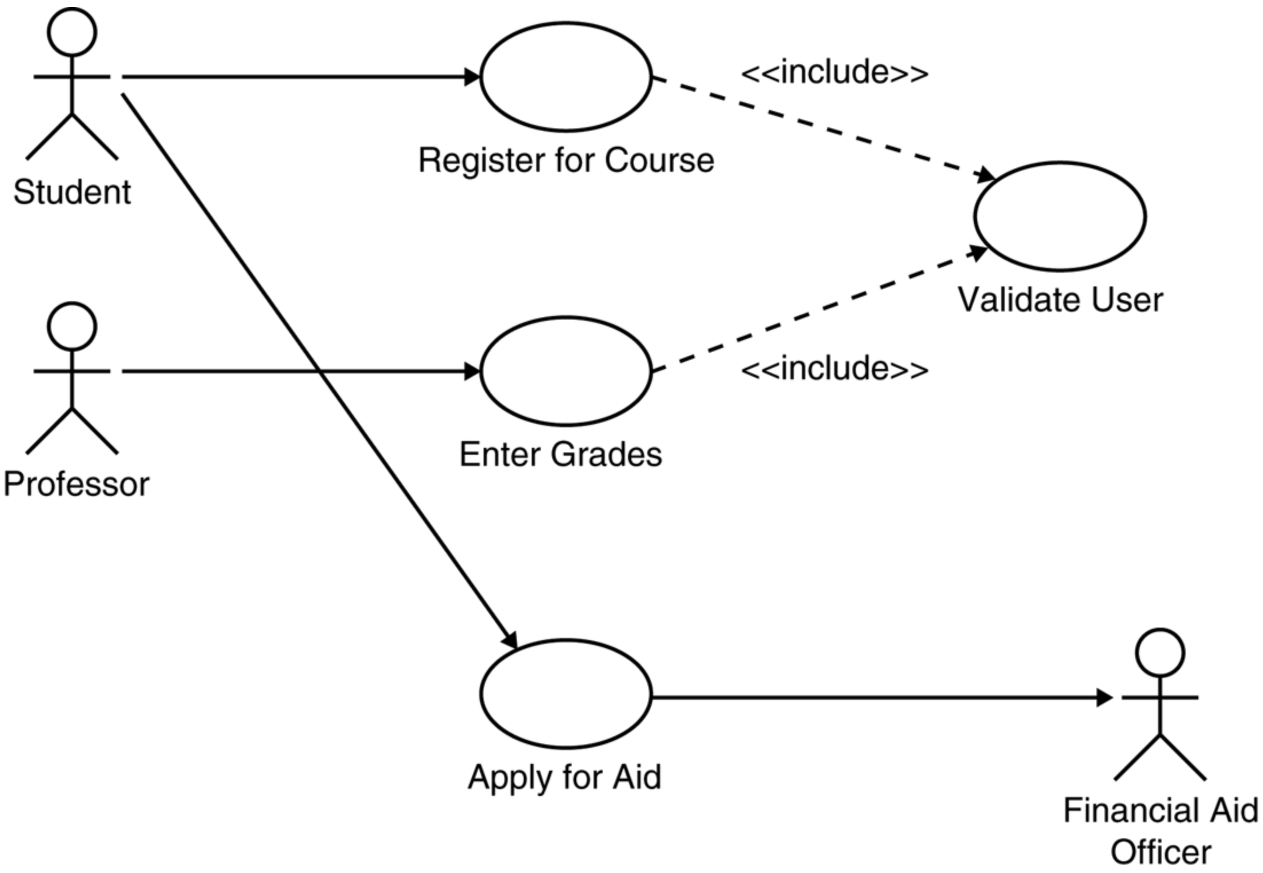
# UML diagrams

- use case diagrams
  - model the functional interactions between users and the system
  - use case is a set of scenarios that have a common goal
- sequence diagrams
  - describe the interactions between various objects over time
  - give a dynamic view of the system by showing the flow of messages between objects
- collaboration diagrams
  - represent interactions between objects as a series of sequenced messages
  - show objects as icons and number the messages
- statechart diagrams
  - describe how an object's state changes in response to external **events**
  - show all the possible states an object can get into in its lifetime
- activity diagrams
  - present a dynamic view of the system by modeling the flow of control from activity to activity
  - can be considered as flowcharts with states

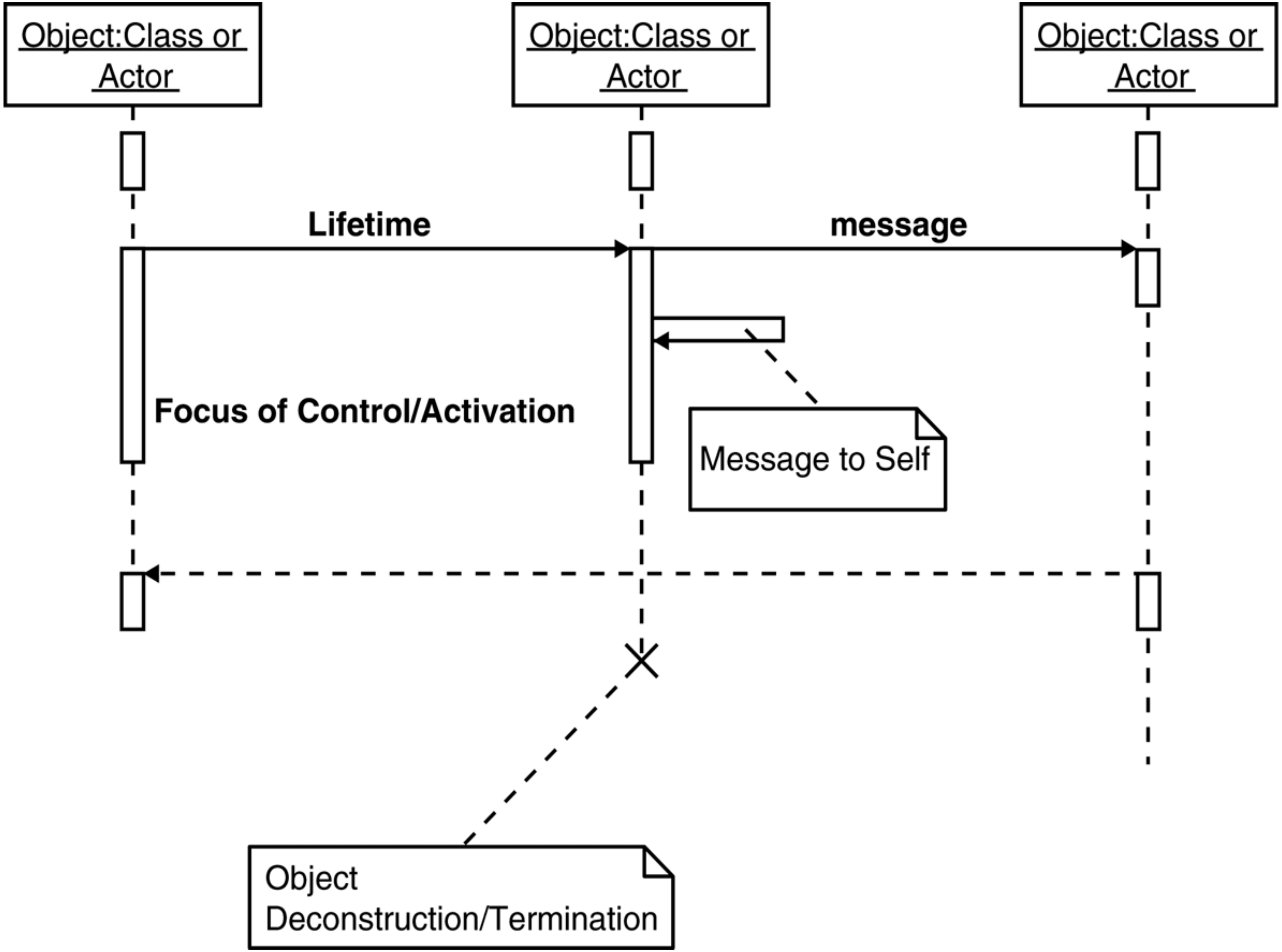
# use-case diagram notation



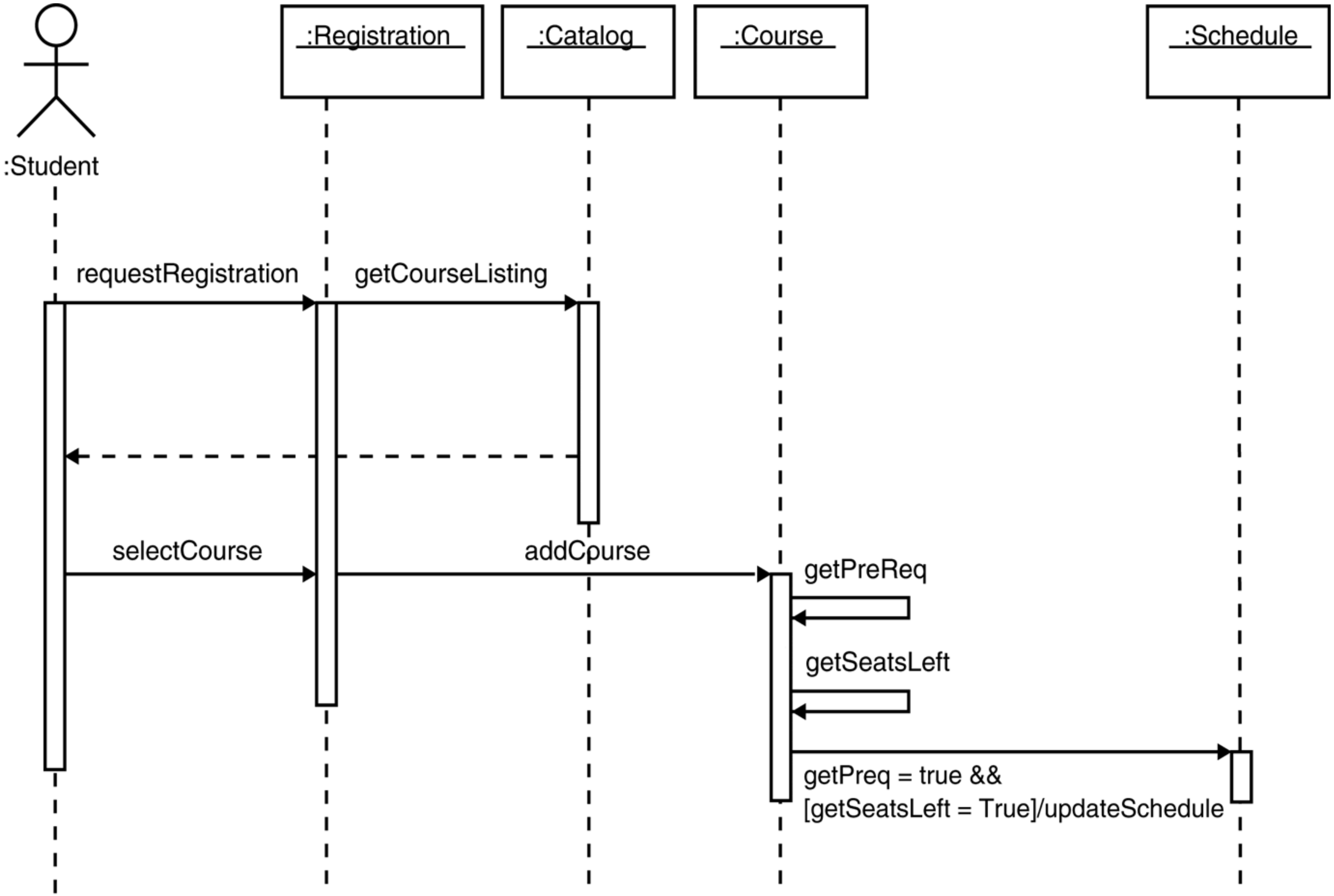
# example use case diagram



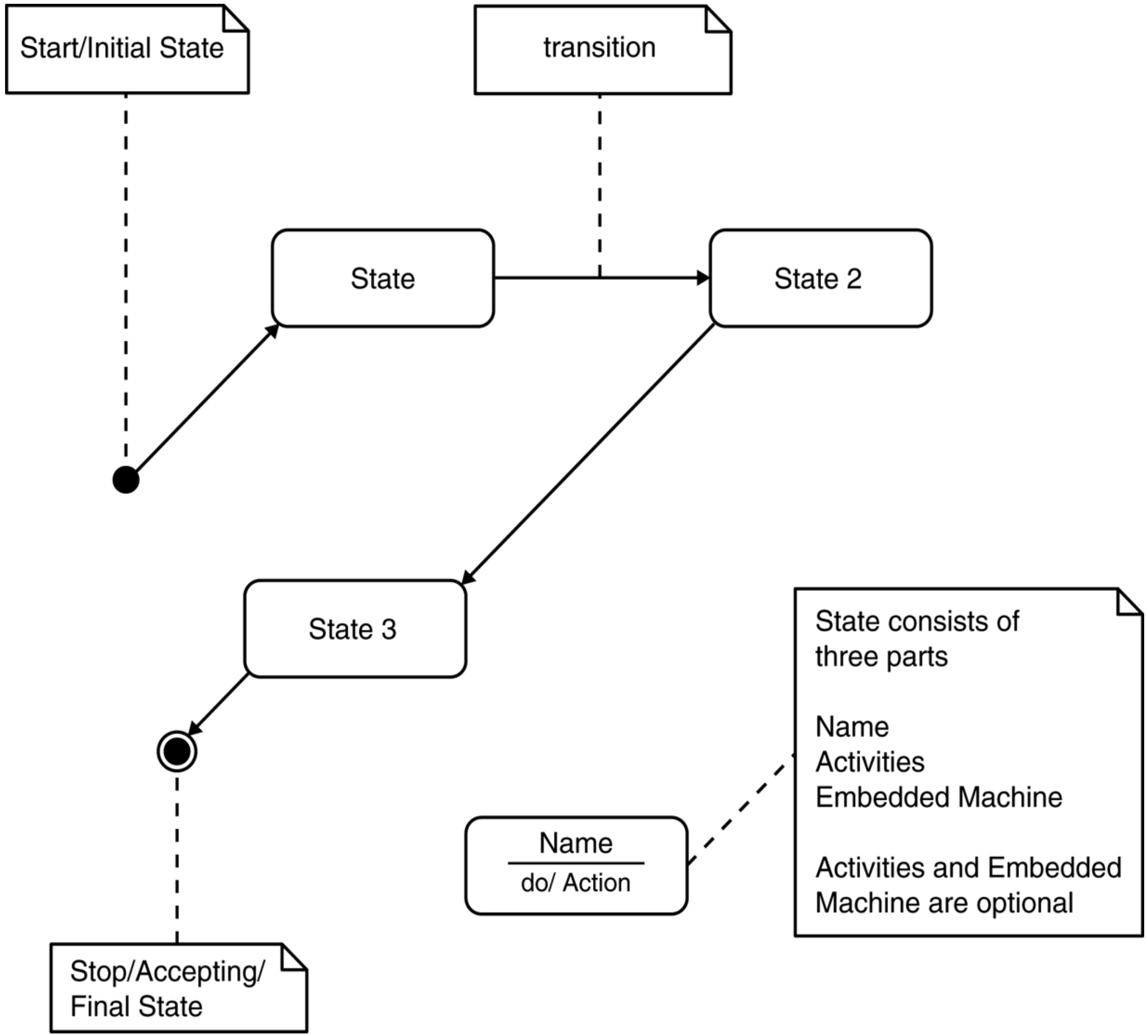
# sequence diagram notation



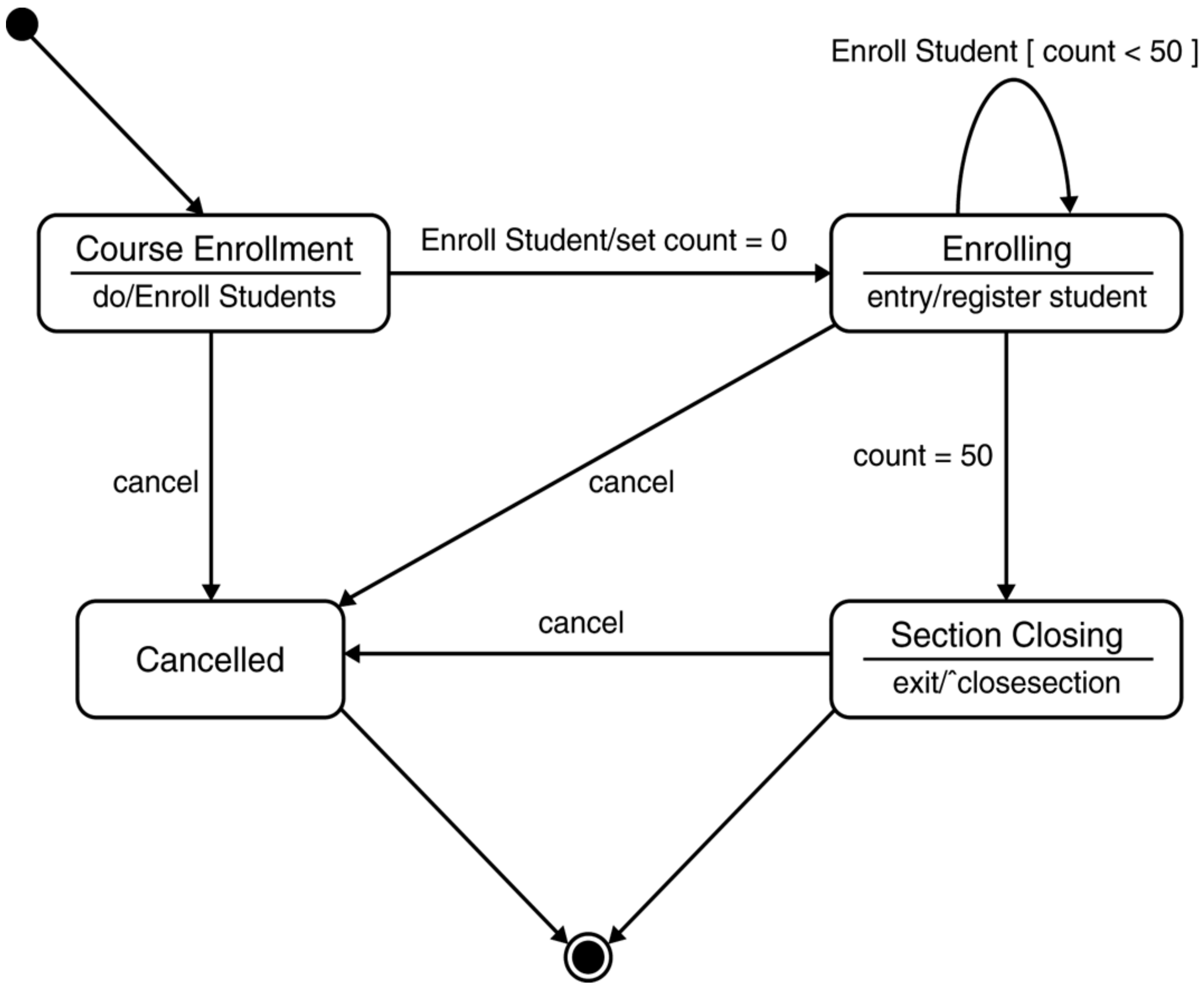
# example of a sequence diagram



# statechart diagram notation



# example of statechart diagram



# data modeling using Rational Rose

- reverse engineering
  - create a conceptual data model based on the DB structure
- forward engineering
  - generate the DDL in a specific DBMS from a data model
- conceptual design in UML notation
- supported DBs: IBM DB2, Oracle, SQL server, Sybase
- converting logical data model to object model and vice versa
- synchronization between the conceptual design and the actual DB
- extensive domain support
- easy communication among design teams

# graphical data model in Rational Rose

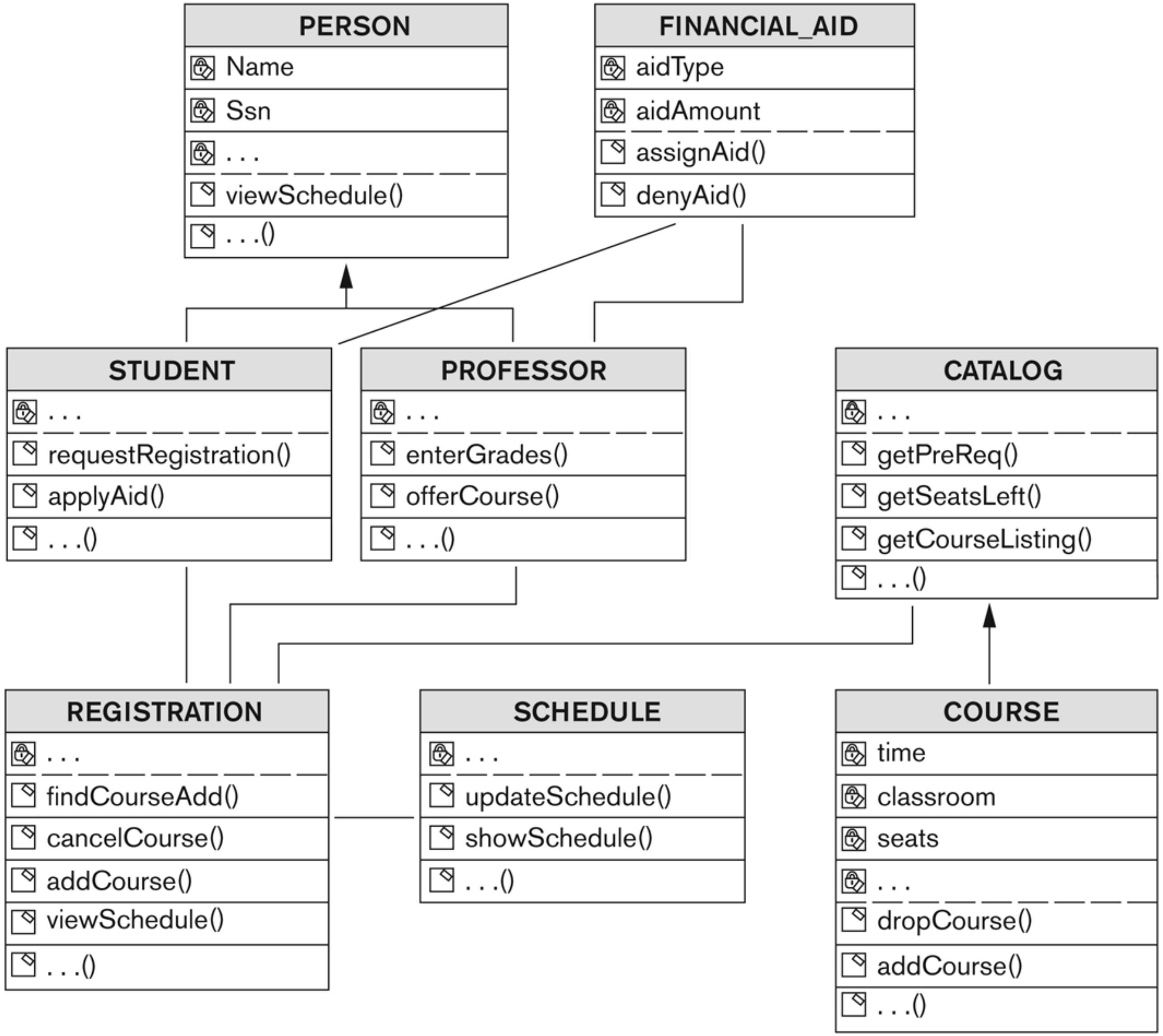



**Figure 12.13**
The design of the UNIVERSITY database as a class diagram.

# logical data model diagram in Rational Rose

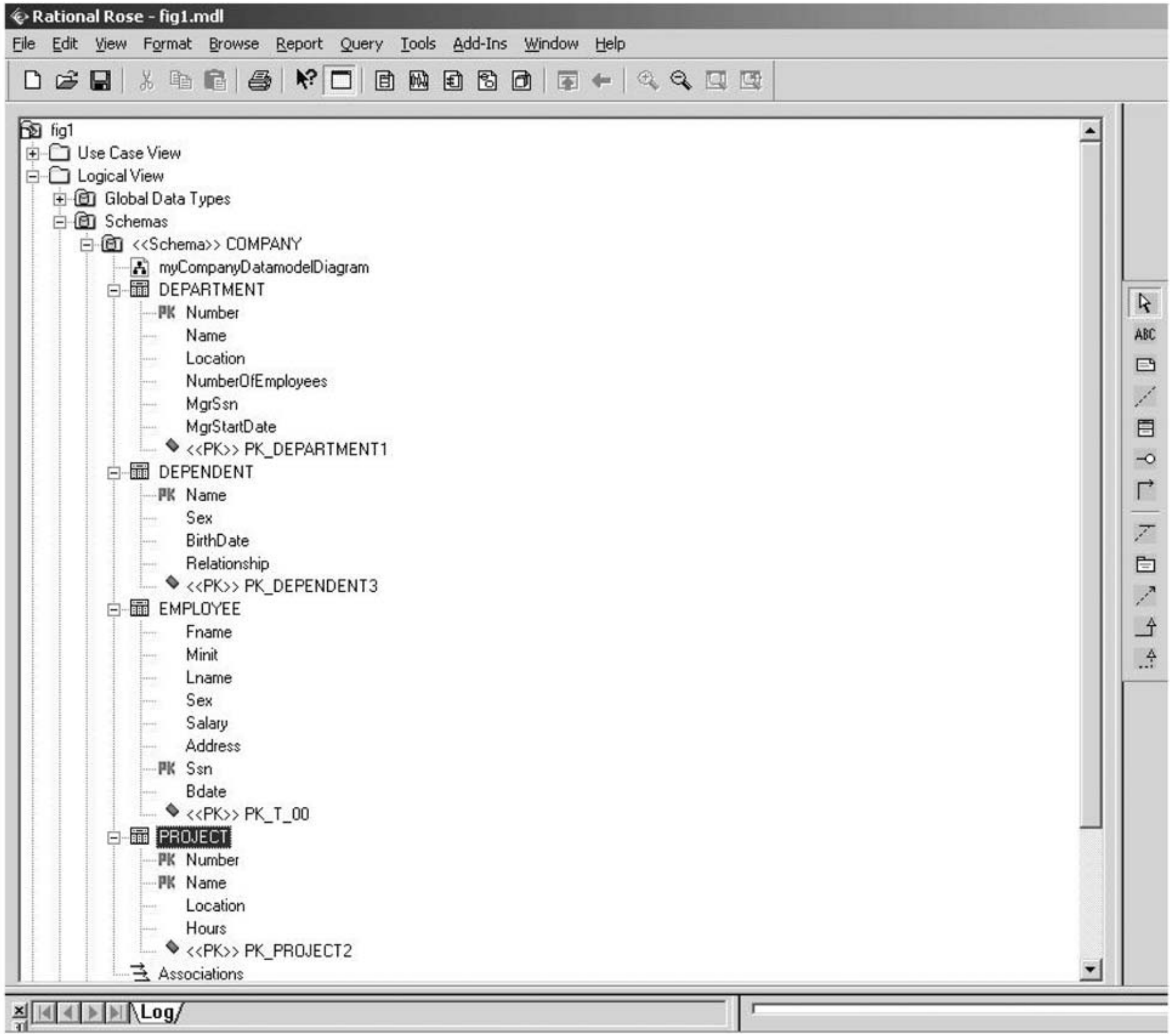



**Figure 12.14**
A logical data model diagram definition in Rational Rose.

# CASE tools

- provided facilities
  - diagramming
  - model mapping
  - design normalization
- desired characteristics
  - easy-to-use interface
  - analytical components
  - heuristic components
  - trade-off analysis
  - display of design results
  - design verification