Lecture 22: Database Design

Kenneth M. Anderson
Object-Oriented Analysis and Design
CSCI 6448 - Spring Semester, 2002

Credit where Credit is Due

- Some material presented in this lecture is taken from section 8 of Maciaszek's "Requirements Analysis and System Design". © Addison Wesley, 2000

Goals for this Lecture

- Cover the information presented in Section 8 of the textbook
  - Discuss
    - Persistent Database Layer
      - Data Models
    - Object database model
    - Object-relational database model
    - Relational database model

Last Lecture

- Covered
  - User Interface Design
    - UI Design Process
    - UI Guidelines
    - Use of OO Models in UI Design

Quiz

1. Draw a sequence diagram for the following scenario

   A registrar object creates a student object and adds the student to its school. The registrar object instructs the student to register for classes, so the student creates an empty schedule object and adds three courses to its schedule. The registrar object then sets the student's status to "registered"
2. Create the equivalent collaboration diagram

Databases and Information Systems

- Information systems have multiple users that require acess to information stored in a database
  - typically one database can support many applications (perhaps viewing different "slices" of the database's information)
- In OO Design, entity classes represent data that must be made persistent
  - these classes must be mapped to a database; the particular mapping used depends on the data model employed by the target database

Persistent Database Layer

- Maciaszek distinguishes between application design and database design
  - The class models in his BCED packages represent application classes, not database classes
    - as such, we still need to design what Maciaszek calls "the persistent database layer"
      - this layer can be provided by a relational, object-relational, or object database

Data Models

- Databases store data
  - and present data models for representing the data that they store
    - they also supply behavior models for things such as stored procedures and triggers
- Data models have three layers
  - External (conceptual) data model
    - typically, entity-relationship models
  - Logical data model
    - a model of the storage structures of a DBMS
  - Physical data model
    - specific to a database; it defines the physical storage of data on a disk

Data Models, continued

- Figure 8.1 on page 277 demonstrates the relationships between application packages and the data models of a database
  - Entity classes represent the external model
    - UML class diagrams can repace ER diagrams
  - Database package classes separate the entity classes from any particular database
    - it communicates to the logical model on the database side, which then makes use of the physical model to store information on disk
    - primary responsibility is to "map" objects in an OO system to data in the database

Object mapping

- Mapping objects to a database can be problematic
  - Firstly, the storage structures of a database may not be object-oriented
    - this means that entity classes must be mapped to non-object-oriented structures
      - even, sometimes, for object databases
  - Secondly, a database is almost never designed for a single application (in modern information system applications)
    - as such, database structures may be tuned to a high-priority information system and thus other information systems must make due with these structures and determine a way to "fit in"
    - in addition, database structures must be designed with future applications in mind, which leads to more generic structures, which are less likely to be optimal for any one information system

Object databases

- present the least troublesome mapping between objects in an application and objects in a database
  - indeed object databases try to make the mapping as transparent to a programmer as possible
  - in addition, a query language, Object SQL is provided when making queries on objects outside of a programming environment

ODB Model

- Basic Modeling Primitives of ODB
  - object (has OID)
    - defined by classes, which have properties and operations
    - atomic, structured, and collection objects
  - literal (has no OID)
    - atomic
      - numbers, strings, etc.
    - structured
      - date, time, timestamp, interval
    - collection
      - set, bag, list, array, dictonary
    - null

ODB Model, continued

- The ODB model builds on the previous modeling primitives
  - three relationship types
    - association, aggregation, generalization
      - see figure 8.3 on page 281
    - ODB defines two types of generalization
      - ISA (interface inheritance)
      - EXTENDS (implementation inheritance)
      - see figure 8.4 on page 282
  - built in operations
    - listed on pages 282-284

Mapping objects with ODB

- Mapping UML class models to ODB is relatively straightforward
  - See example 8.1 on page 285 for details in mapping entity classes
    - The example invents some questions to ask of the analysis class model of the Contact Management case study
      - e.g. "How do I manage an employee having more than one phone number?" or "Can I model an employee name as a single attribute with internal structure (first, middle, last) without creating a separate employee name class?
  - Example 8.2 on pg 286 demonstrates mapping assoications between objects to ODB

Mapping aggregations in ODB

- ODB does not directly support aggregations
  - aggregations are simply modeled as associations (e.g. by using ODB collection classes)
    - semantics must be enforced using stored procedures
  - compositions are modeled by creating a separate ODB class that represents the container
    - this container contains an attribute using an ODB collection class that stores values of the component class
  - Example 8.3 on pages 288 and 289 demonstrate mapping aggregation and composition relationships

Mapping generalizations in ODB

- Very straightforward
  - Inheriting from an ODB interface results in an ISA relationship
  - Inheriting from an ODB class results in an EXTENDS operation
- Example 8.4 on page 289 presents an example of this type of mapping

Object-Relational Data Model

- Object-Relational models combine the "new" object model with the "old" relational model
  - A single DBMS can process relational tables and object tables
- The ORDB model was standardized in 1999, also known as SQL'99
  - upward compatible with SQL'92
  - extends relational model to allow objects to be stored in SQL tables
  - also extends model to allow the specification of arbitrarily complex structured types (e.g. classes)

ORDB modeling primitives

- ORDB provides a structured type that corresponds to ODB's interface and UML's class
  - structured type is defined by specifying attributes and operations
    - subtype relationships can be specified
  - structured types are stored in tables
    - columns in tables can be structured types (object tables)
    - ODB structured and collection types are supported
    - row types can be used to nest structured types and provide navigation between structured types

Mapping to ORDB

- More difficult to do, because existing database systems do not completely support the SQL'99 standard
  - as such, a mapping must be done to a particular DBMS, not to the standard
  - I will, hence, skip this section of chapter 8

Relational Data Model

- The relational model has been dominant for over 20 years
  - as such, inertia will keep relational databases in business for years (decades) to come
- RDB was last standardized with SQL'92

Relational Modeling Primitives

- Tables consist of columns
- Columns can only contain values of atomic types
  - ODB's object types, structured types, collections, and references are not supported
  - references between tables are maintained by comparing values in columns
- Complete set of primitives shown in figure 8.16 on page 302

Mapping objects to RDB

- See figure 8.17 on page 304 for an example of modeling an entity class as a table
- See figure 8.18 on page 305 for an example of modeling a relationship between classes
  - A referential integrity link says that the values for a particular column in one table must come from the values of a column (often with the same name) in another table
    - constraints may be placed on this referential integrity link to determine what happens when a table is updated or deleted
- Many to Many relationships are modeled by introducing a new table (see figure 8.19 on page 306)
- Note, other than these basic examples, I do not require in-depth knowledge of this subject