Chapter 9:
Data Management Layer Design
Objectives

- Become familiar with several object-persistence formats.
- Be able to map problem domain objects to different object-persistence formats.
- Be able to apply the steps of normalization to a relational database.
- Be able to optimize a relational database for object storage and access.
- Become familiar with indexes for relational databases.
- Be able to estimate the size of a relational database.
- Understand the effect of nonfunctional requirements on the data management layer
- Be able to design the data access and manipulation classes.
Introduction

- Applications are of little use without data
  - Data must be stored and accessed efficiently
- The data management layer includes:
  - Data access and manipulation logic
  - Storage design
- Four-step design approach:
  - Selecting the format of the storage
  - Mapping problem-domain objects to object persistence format
  - Optimizing the object persistence format
  - Designing the data access & manipulation classes
Object Persistence Formats

- Files (sequential and random access)
- Object-oriented databases
- Object-relational databases
- Relational databases
- “NoSQL” data stores
Electronic Files

- Sequential access files
  - Operations (read, write and search) are conducted one record after another (in sequence)
  - Efficient for report writing
  - Inefficient for searching (an average of 50% of records have to be accessed for each search)
  - Unordered files add records to the end of the file
  - Ordered files are sorted, but additions & deletions require additional maintenance

- Random access files
  - Efficient for operations (read, write and search)
  - Inefficient for report writing
Application File Types

- Master Files
  - Store core information (e.g., order and customer data)
  - Usually held for long periods
  - Changes require new programs
- Look-up files (e.g., zip codes with city and state names)
- Transaction files
  - Information used to update a master file
  - Can be deleted once master file is updated
- Audit file—records data before & after changes
- History file—archives of past transactions
Relational Databases

- Most popular way to store data for applications
- Consists of a collection of tables
  - Primary key uniquely identifies each row
  - Foreign keys establish relationships between tables
    - Referential integrity ensures records in different tables are matched properly
    - Example: you cannot enter an order for a customer that does not exist
- Structured Query Language (SQL) is used to access the data
  - Operates on complete tables vs. individual records
  - Allows joining tables together to obtain matched data
Object-Relational Databases

- A standard relational database with ability to store objects added
- Must create a mapping from UML class diagrams to database schema is required.
- Accomplished using user-defined data types
  - SQL extended to handle complex data types
  - Support for inheritance varies
Object-Oriented Databases

Two approaches:
- Add persistence extensions to OO programming language
- Create a separate OO database

Utilize extents—a collection of instances of a class
- A table associated with a class.
- Each row an instance of the class, and having an Object ID
- Object IDs relate objects together
- Another primary key may still be desirable

Inheritance is supported but is language dependent

Represent a small market share due to its steep learning curve
NoSQL Data Stores

- Newest type; used primarily for complex data types
  - Does not support SQL
  - No standards exist
  - Support very fast queries
- Data may not be consistent since there are no locking mechanisms
- Types
  - Key-value data stores
  - Document data stores
  - Columnar data stores
- Immaturity of technology prevents traditional business application support
## Selecting Persistence Formats

<table>
<thead>
<tr>
<th></th>
<th>Sequential and Random Access Files</th>
<th>Relational DBMS</th>
<th>Object Relational DBMS</th>
<th>Object-Oriented DBMS</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Major Strengths</strong></td>
<td>Usually part of an object-oriented programming language</td>
<td>Leader in the database market</td>
<td>Based on established, proven technology, e.g., SQL</td>
<td>Able to handle complex data</td>
</tr>
<tr>
<td></td>
<td>Files can be designed for fast performance</td>
<td>Can handle diverse data needs</td>
<td>Able to handle complex data</td>
<td>Direct support for object orientation</td>
</tr>
<tr>
<td></td>
<td>Good for short-term data storage</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Major Weaknesses</strong></td>
<td>Redundant data</td>
<td>Cannot handle complex data</td>
<td>Limited support for object orientation</td>
<td>Technology is still maturing Skills are hard to find</td>
</tr>
<tr>
<td></td>
<td>Data must be updated using programs, i.e., no manipulation or query language</td>
<td>No support for object orientation</td>
<td>Impedance mismatch between tables and objects</td>
<td></td>
</tr>
<tr>
<td></td>
<td>No access control</td>
<td>Impedance mismatch between tables and objects</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Data Types Supported</strong></td>
<td>Simple and Complex</td>
<td>Simple</td>
<td>Simple and Complex</td>
<td>Simple and Complex</td>
</tr>
<tr>
<td><strong>Types of Application Systems Supported</strong></td>
<td>Transaction processing</td>
<td>Transaction processing and decision making</td>
<td>Transaction processing and decision making</td>
<td>Transaction processing and decision making</td>
</tr>
<tr>
<td><strong>Existing Storage Formats</strong></td>
<td>Organization dependent</td>
<td>Organization dependent</td>
<td>Organization dependent</td>
<td>Organization dependent</td>
</tr>
<tr>
<td><strong>Future Needs</strong></td>
<td>Poor future prospects</td>
<td>Good future prospects</td>
<td>Good future prospects</td>
<td>Good future prospects</td>
</tr>
</tbody>
</table>
Mapping Problem-Domain Objects to Object-Persistence Formats

- Map objects to an OODBMS format
  - Each concrete class has a corresponding object persistence class
  - Add a data access and manipulation class to control the interaction
- Map objects to an ORDBMS format
  - Procedure depends on the level of support for object orientation by the ORDBMS
- Map objects to an RDBMS format
Mapping to an OODBMS

Adapts between

PD Layer

Patient
Appointment

DM Layer

Patient-DAM
Appointment-DAM

Patient-OODBMS
Appointment-OODBMS

Adapts between
Mapping to an OODBMS

May need to factor out multiple inheritance if not supported by OODBMS

Techniques similar to removing it from the design to write code when required:

Either change extra base class objects to attributes, or absorb their attributes.
Rule 1: Map all concrete Problem Domain classes to the ORDBMS tables. Also, if an abstract problem domain class has multiple direct subclasses, map the abstract class to an ORDBMS table.

Rule 2: Map single-valued attributes to columns of the ORDBMS tables.

Rule 3: Map methods and derived attributes to stored procedures or to program modules.

Rule 4: Map single-valued aggregation and association relationships to a column that can store an Object ID. Do this for both sides of the relationship.

Rule 5: Map multivalued attributes to a column that can contain a set of values.

Rule 6: Map repeating groups of attributes to a new table and create a one-to-many association from the original table to the new one.

Rule 7: Map multivalued aggregation and association relationships to a column that can store a set of Object IDs. Do this for both sides of the relationship.

Rule 8: For aggregation and association relationships of mixed type (one-to-many or many-to-one), on the single-valued side (1..1 or 0..1) of the relationship, add a column that can store a set of Object IDs. The values contained in this new column will be the Object IDs from the instances of the class on the multivalued side. On the multivalued side (1..' or 0..*), add a column that can store a single Object ID that will contain the value of the instance of the class on the single-valued side.

For generalization/inheritance relationships:

Rule 9a: Add a column(s) to the table(s) that represents the subclass(es) that will contain an Object ID of the instance stored in the table that represents the superclass. This is similar in concept to a foreign key in an RDBMS. The multiplicity of this new association from the subclass to the “superclass” should be 1..1. Add a column(s) to the table(s) that represents the superclass(es) that will contain an Object ID of the instance stored in the table that represents the subclass(es). If the superclasses are concrete, that is, they can be instantiated themselves, then the multiplicity from the superclass to the subclass is 0..1, otherwise, it is 1..1. An exclusive-or (XOR) constraint must be added between the associations. Do this for each superclass.

or

Rule 9b: Flatten the inheritance hierarchy by copying the superclass attributes down to all of the subclasses and remove the superclass from the design.*

*It is also a good idea to document this modification in the design so that in the future, modifications to the design can be maintained easily.
Mapping to an ORDBMS

ORDBMS Tables

Participant Table
- lastname[1..1]
- firstname[1..1]
- address[1..1]
- phone[1..1]
- birthdate[1..1]
- SubClassObjects[1..1]

Patient Table
- amount[1..1]
- Participant[1..1]
- Appts[0..*]
- Symptoms[1..*]
- Insurance carrier[0..*]
- Primary Insurance Carrier[0..*]

Appointment Table
- Patient[1..1]
- time[1..1]
- date[1..1]
- reason[1..1]

Problem Domain Classes

Participant
- lastname
- firstname
- address
- phone
- birthdate
- age

Patient
- amount
- insurance carrier
- make appointment()
- calculate last visit()
- change status()
- provides medical history()

Appointment
- time
- date
- reason
- cancel without notice()

Symptom
- name

Symptom Table
- name[1..1]
- Patients[0..*]
Mapping to an RDBMS

**Rule 1:** Map all concrete-problem domain classes to the RDBMS tables. Also, if an abstract Problem Domain class has multiple direct subclasses, map the abstract class to a RDBMS table.

**Rule 2:** Map single-valued attributes to columns of the tables.

**Rule 3:** Map methods to stored procedures or to program modules.

**Rule 4:** Map single-valued aggregation and association relationships to a column that can store the key of the related table, i.e., add a foreign key to the table. Do this for both sides of the relationship.

**Rule 5:** Map multivalued attributes and repeating groups to new tables and create a one-to-many association from the original table to the new ones.

**Rule 6:** Map multivalued aggregation and association relationships to a new associative table that relates the two original tables together. Copy the primary key from both original tables to the new associative table, i.e., add foreign keys to the table.

**Rule 7:** For aggregation and association relationships of mixed type, copy the primary key from the single-valued side (1..1 or 0..1) of the relationship to a new column in the table on the multivalued side (1..* or 0..*) of the relationship that can store the key of the related table, i.e., add a foreign key to the table on the multivalued side of the relationship.

For generalization/inheritance relationships:

**Rule 8a:** Ensure that the primary key of the subclass instance is the same as the primary key of the superclass. The multiplicity of this new association from the subclass to the “superclass” should be 1..1. If the superclasses are concrete, that is, they can be instantiated themselves, then the multiplicity from the superclass to the subclass is 0..1, otherwise, it is 1..1. Furthermore, an exclusive-or (XOR) constraint must be added between the associations. Do this for each superclass.

**OR**

**Rule 8b:** Flatten the inheritance hierarchy by copying the superclass attributes down to all of the subclasses and remove the superclass from the design.

* It is also a good idea to document this modification in the design so that in the future, modifications to the design can be maintained easily.
Mapping to an RDBMS
Optimizing RDBMS-Based Object Storage

- Primary (often conflicting) dimensions:
  - Improve storage efficiency
    - Normalize the tables
    - Reduce redundant data and the occurrence of null values
  - Improve speed of access
    - De-normalize some tables to reduce processing time
    - Place similar records together (clustering)
    - Add indexes to quickly locate records
Normalization

- Store each data fact only once in the database
- Reduces data redundancies and chances of errors
- First four levels of normalization are
  - 0 Normal Form: normalization rules not applied
  - 1 Normal Form: no multi-valued attributes (each cell has only a single value)
    - Eliminate fields which are arrays or repeated.
    - They can be new tables.
Form 1 Violation

Sample Records:

<table>
<thead>
<tr>
<th>Order Number</th>
<th>Date</th>
<th>Cust ID</th>
<th>Last Name</th>
<th>First Name</th>
<th>State</th>
<th>Tax Rate</th>
<th>Prod. 1 Number</th>
<th>Prod. 1 Desc.</th>
<th>Prod. 1 Price</th>
<th>Prod. 1 Qty</th>
<th>Prod. 2 Number</th>
<th>Prod. 2 Desc.</th>
<th>Prod. 2 Price</th>
<th>Prod. 2 Qty</th>
<th>Prod. 3 Number</th>
<th>Prod. 3 Desc.</th>
<th>Prod. 3 Price</th>
<th>Prod. 3 Qty</th>
</tr>
</thead>
<tbody>
<tr>
<td>239</td>
<td>11/23/00</td>
<td>1035</td>
<td>Black</td>
<td>John</td>
<td>MD</td>
<td>0.05</td>
<td>555</td>
<td>Cheese Tray</td>
<td>$65.00</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>260</td>
<td>11/24/00</td>
<td>1035</td>
<td>Black</td>
<td>John</td>
<td>MD</td>
<td>0.05</td>
<td>444</td>
<td>Wine Gift Pack</td>
<td>$60.00</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>273</td>
<td>11/27/00</td>
<td>1035</td>
<td>Black</td>
<td>John</td>
<td>MD</td>
<td>0.05</td>
<td>222</td>
<td>Bottle Opener</td>
<td>$12.00</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>241</td>
<td>11/23/00</td>
<td>1123</td>
<td>Williams</td>
<td>Mary</td>
<td>CA</td>
<td>0.08</td>
<td>444</td>
<td>Wine Gift Pack</td>
<td>$60.00</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>262</td>
<td>11/24/00</td>
<td>1123</td>
<td>Williams</td>
<td>Mary</td>
<td>CA</td>
<td>0.08</td>
<td>222</td>
<td>Bottle Opener</td>
<td>$12.00</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>287</td>
<td>11/27/00</td>
<td>1123</td>
<td>Williams</td>
<td>Mary</td>
<td>CA</td>
<td>0.08</td>
<td>222</td>
<td>Bottle Opener</td>
<td>$12.00</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>290</td>
<td>11/30/00</td>
<td>1123</td>
<td>Williams</td>
<td>Mary</td>
<td>CA</td>
<td>0.08</td>
<td>555</td>
<td>Cheese Tray</td>
<td>$45.00</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>234</td>
<td>11/23/00</td>
<td>2242</td>
<td>DeBerry</td>
<td>Ann</td>
<td>DC</td>
<td>0.065</td>
<td>555</td>
<td>Cheese Tray</td>
<td>$45.00</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>237</td>
<td>11/23/00</td>
<td>2242</td>
<td>DeBerry</td>
<td>Ann</td>
<td>DC</td>
<td>0.065</td>
<td>111</td>
<td>Wine Guide</td>
<td>$15.00</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>238</td>
<td>11/23/00</td>
<td>2242</td>
<td>DeBerry</td>
<td>Ann</td>
<td>DC</td>
<td>0.065</td>
<td>444</td>
<td>Wine Gift Pack</td>
<td>$60.00</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>245</td>
<td>11/24/00</td>
<td>2242</td>
<td>DeBerry</td>
<td>Ann</td>
<td>DC</td>
<td>0.065</td>
<td>222</td>
<td>Bottle Opener</td>
<td>$12.00</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>250</td>
<td>11/24/00</td>
<td>2242</td>
<td>DeBerry</td>
<td>Ann</td>
<td>DC</td>
<td>0.065</td>
<td>222</td>
<td>Bottle Opener</td>
<td>$12.00</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>252</td>
<td>11/24/00</td>
<td>2242</td>
<td>DeBerry</td>
<td>Ann</td>
<td>DC</td>
<td>0.065</td>
<td>222</td>
<td>Bottle Opener</td>
<td>$12.00</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>253</td>
<td>11/24/00</td>
<td>2242</td>
<td>DeBerry</td>
<td>Ann</td>
<td>DC</td>
<td>0.065</td>
<td>222</td>
<td>Bottle Opener</td>
<td>$12.00</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>297</td>
<td>11/30/00</td>
<td>2242</td>
<td>DeBerry</td>
<td>Ann</td>
<td>DC</td>
<td>0.065</td>
<td>333</td>
<td>Jams &amp; Jellies</td>
<td>$20.00</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>243</td>
<td>11/24/00</td>
<td>4254</td>
<td>Bailey</td>
<td>Ryan</td>
<td>MD</td>
<td>0.05</td>
<td>555</td>
<td>Cheese Tray</td>
<td>$45.00</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>246</td>
<td>11/24/00</td>
<td>4254</td>
<td>Bailey</td>
<td>Ryan</td>
<td>MD</td>
<td>0.05</td>
<td>333</td>
<td>Jams &amp; Jellies</td>
<td>$20.00</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>248</td>
<td>11/24/00</td>
<td>4254</td>
<td>Bailey</td>
<td>Ryan</td>
<td>MD</td>
<td>0.05</td>
<td>222</td>
<td>Bottle Opener</td>
<td>$12.00</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>235</td>
<td>11/23/00</td>
<td>9500</td>
<td>Chin</td>
<td>April</td>
<td>KS</td>
<td>0.05</td>
<td>222</td>
<td>Bottle Opener</td>
<td>$12.00</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>242</td>
<td>11/23/00</td>
<td>9500</td>
<td>Chin</td>
<td>April</td>
<td>KS</td>
<td>0.05</td>
<td>333</td>
<td>Jams &amp; Jellies</td>
<td>$20.00</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>244</td>
<td>11/24/00</td>
<td>9500</td>
<td>Chin</td>
<td>April</td>
<td>KS</td>
<td>0.05</td>
<td>222</td>
<td>Bottle Opener</td>
<td>$12.00</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>251</td>
<td>11/24/00</td>
<td>9500</td>
<td>Chin</td>
<td>April</td>
<td>KS</td>
<td>0.05</td>
<td>111</td>
<td>Wine Guide</td>
<td>$15.00</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Redundant Data: Order Number: unsigned long
Date: Date
Cust ID: unsigned long
Last Name: String
First Name: String
State: String
Tax Rate: float
Product 1 Number: unsigned long
Product 1 Desc.: String
Product 1 Price: double
Product 1 Qty.: unsigned long
Product 2 Number: unsigned long
Product 2 Desc.: String
Product 2 Price: double
Product 2 Qty.: unsigned long
Product 3 Number: unsigned long
Product 3 Desc.: String
Product 3 Price: double
Product 3 Qty.: unsigned long

Null Cells:
Normalization

First four levels of normalization are

- 2 Fields depend on whole primary key
  - New order: primary key is order and product number
  - Description depends only on product
  - New table for product info

- 3 None of the fields depend on the primary key
  - No field depends on non-primary key.
  - Tax rate depends on state.
  - Add table of states and rates.
Form 2 Violation

Sample Records:

**Order Table**

<table>
<thead>
<tr>
<th>Order Number</th>
<th>Date</th>
<th>Cust ID</th>
<th>Last Name</th>
<th>First Name</th>
<th>State</th>
<th>Tax Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>239</td>
<td>11/23/00</td>
<td>1035</td>
<td>Black</td>
<td>John</td>
<td>MD</td>
<td>0.05</td>
</tr>
<tr>
<td>260</td>
<td>11/24/00</td>
<td>1035</td>
<td>Black</td>
<td>John</td>
<td>MD</td>
<td>0.05</td>
</tr>
<tr>
<td>273</td>
<td>11/27/00</td>
<td>1035</td>
<td>Black</td>
<td>John</td>
<td>MD</td>
<td>0.05</td>
</tr>
<tr>
<td>241</td>
<td>11/23/00</td>
<td>1123</td>
<td>Williams</td>
<td>Mary</td>
<td>CA</td>
<td>0.08</td>
</tr>
<tr>
<td>262</td>
<td>11/24/00</td>
<td>1123</td>
<td>Williams</td>
<td>Mary</td>
<td>CA</td>
<td>0.08</td>
</tr>
<tr>
<td>287</td>
<td>11/27/00</td>
<td>1123</td>
<td>Williams</td>
<td>Mary</td>
<td>CA</td>
<td>0.08</td>
</tr>
<tr>
<td>290</td>
<td>11/30/00</td>
<td>1123</td>
<td>Williams</td>
<td>Mary</td>
<td>CA</td>
<td>0.08</td>
</tr>
<tr>
<td>234</td>
<td>11/23/00</td>
<td>2242</td>
<td>DeBerry</td>
<td>Ann</td>
<td>DC</td>
<td>0.065</td>
</tr>
<tr>
<td>237</td>
<td>11/23/00</td>
<td>2242</td>
<td>DeBerry</td>
<td>Ann</td>
<td>DC</td>
<td>0.065</td>
</tr>
<tr>
<td>238</td>
<td>11/23/00</td>
<td>2242</td>
<td>DeBerry</td>
<td>Ann</td>
<td>DC</td>
<td>0.065</td>
</tr>
<tr>
<td>245</td>
<td>11/24/00</td>
<td>2242</td>
<td>DeBerry</td>
<td>Ann</td>
<td>DC</td>
<td>0.065</td>
</tr>
<tr>
<td>250</td>
<td>11/24/00</td>
<td>2242</td>
<td>DeBerry</td>
<td>Ann</td>
<td>DC</td>
<td>0.065</td>
</tr>
<tr>
<td>252</td>
<td>11/24/00</td>
<td>2242</td>
<td>DeBerry</td>
<td>Ann</td>
<td>DC</td>
<td>0.065</td>
</tr>
<tr>
<td>253</td>
<td>11/24/00</td>
<td>2242</td>
<td>DeBerry</td>
<td>Ann</td>
<td>DC</td>
<td>0.065</td>
</tr>
<tr>
<td>297</td>
<td>11/30/00</td>
<td>2242</td>
<td>DeBerry</td>
<td>Ann</td>
<td>DC</td>
<td>0.065</td>
</tr>
<tr>
<td>243</td>
<td>11/24/00</td>
<td>4254</td>
<td>Bailey</td>
<td>Ryan</td>
<td>MD</td>
<td>0.05</td>
</tr>
<tr>
<td>246</td>
<td>11/24/00</td>
<td>4254</td>
<td>Bailey</td>
<td>Ryan</td>
<td>MD</td>
<td>0.05</td>
</tr>
<tr>
<td>248</td>
<td>11/24/00</td>
<td>4254</td>
<td>Bailey</td>
<td>Ryan</td>
<td>MD</td>
<td>0.05</td>
</tr>
<tr>
<td>235</td>
<td>11/23/00</td>
<td>9500</td>
<td>Chin</td>
<td>April</td>
<td>KS</td>
<td>0.05</td>
</tr>
<tr>
<td>242</td>
<td>11/23/00</td>
<td>9500</td>
<td>Chin</td>
<td>April</td>
<td>KS</td>
<td>0.05</td>
</tr>
<tr>
<td>244</td>
<td>11/24/00</td>
<td>9500</td>
<td>Chin</td>
<td>April</td>
<td>KS</td>
<td>0.05</td>
</tr>
<tr>
<td>251</td>
<td>11/24/00</td>
<td>9500</td>
<td>Chin</td>
<td>April</td>
<td>KS</td>
<td>0.05</td>
</tr>
</tbody>
</table>

**Product Order Table**

<table>
<thead>
<tr>
<th>Order Number</th>
<th>Product Number</th>
<th>Product Desc</th>
<th>Product Price</th>
<th>Product Qty</th>
</tr>
</thead>
<tbody>
<tr>
<td>239</td>
<td>555</td>
<td>Cheese Tray</td>
<td>$45.00</td>
<td>2</td>
</tr>
<tr>
<td>260</td>
<td>444</td>
<td>Wine Gift Pack</td>
<td>$60.00</td>
<td>1</td>
</tr>
<tr>
<td>273</td>
<td>222</td>
<td>Bottle Opener</td>
<td>$12.00</td>
<td>1</td>
</tr>
<tr>
<td>241</td>
<td>444</td>
<td>Wine Gift Pack</td>
<td>$60.00</td>
<td>2</td>
</tr>
<tr>
<td>262</td>
<td>222</td>
<td>Bottle Opener</td>
<td>$12.00</td>
<td>2</td>
</tr>
<tr>
<td>287</td>
<td>222</td>
<td>Bottle Opener</td>
<td>$12.00</td>
<td>2</td>
</tr>
<tr>
<td>290</td>
<td>555</td>
<td>Cheese Tray</td>
<td>$45.00</td>
<td>3</td>
</tr>
<tr>
<td>234</td>
<td>555</td>
<td>Cheese Tray</td>
<td>$45.00</td>
<td>2</td>
</tr>
<tr>
<td>237</td>
<td>111</td>
<td>Wine Guide</td>
<td>$15.00</td>
<td>1</td>
</tr>
<tr>
<td>237</td>
<td>444</td>
<td>Wine Gift Pack</td>
<td>$60.00</td>
<td>1</td>
</tr>
<tr>
<td>238</td>
<td>444</td>
<td>Wine Gift Pack</td>
<td>$60.00</td>
<td>1</td>
</tr>
<tr>
<td>245</td>
<td>222</td>
<td>Bottle Opener</td>
<td>$12.00</td>
<td>1</td>
</tr>
<tr>
<td>250</td>
<td>222</td>
<td>Bottle Opener</td>
<td>$12.00</td>
<td>1</td>
</tr>
<tr>
<td>252</td>
<td>222</td>
<td>Bottle Opener</td>
<td>$12.00</td>
<td>1</td>
</tr>
<tr>
<td>253</td>
<td>444</td>
<td>Wine Gift Pack</td>
<td>$60.00</td>
<td>2</td>
</tr>
<tr>
<td>253</td>
<td>222</td>
<td>Bottle Opener</td>
<td>$12.00</td>
<td>1</td>
</tr>
<tr>
<td>253</td>
<td>444</td>
<td>Wine Gift Pack</td>
<td>$60.00</td>
<td>1</td>
</tr>
<tr>
<td>297</td>
<td>333</td>
<td>Jams &amp; Jellies</td>
<td>$20.00</td>
<td>2</td>
</tr>
<tr>
<td>243</td>
<td>555</td>
<td>Cheese Tray</td>
<td>$45.00</td>
<td>2</td>
</tr>
<tr>
<td>246</td>
<td>333</td>
<td>Jams &amp; Jellies</td>
<td>$20.00</td>
<td>3</td>
</tr>
<tr>
<td>248</td>
<td>333</td>
<td>Jams &amp; Jellies</td>
<td>$20.00</td>
<td>2</td>
</tr>
<tr>
<td>248</td>
<td>111</td>
<td>Wine Guide</td>
<td>$15.00</td>
<td>1</td>
</tr>
<tr>
<td>235</td>
<td>222</td>
<td>Bottle Opener</td>
<td>$12.00</td>
<td>1</td>
</tr>
<tr>
<td>242</td>
<td>333</td>
<td>Jams &amp; Jellies</td>
<td>$20.00</td>
<td>3</td>
</tr>
<tr>
<td>244</td>
<td>222</td>
<td>Bottle Opener</td>
<td>$12.00</td>
<td>2</td>
</tr>
<tr>
<td>251</td>
<td>111</td>
<td>Wine Guide</td>
<td>$15.00</td>
<td>2</td>
</tr>
</tbody>
</table>
Form 3 Violation

Sample Records:

Customer Table
- Cust ID: unsigned long
- Last Name: String
- First Name: String

Order Table
- Order Number: unsigned long
- Date: Date
- Cust ID: unsigned long
- State: String
- Tax Rate: Real

Product Order Table
- Order Number: unsigned long
- Product Number: unsigned long
- Qty: unsigned long

Product Table
- Product Number: unsigned long
- Product Desc: String
- Price: double

Note: Cust ID will serve as the primary key of Customer.

Note: Order Number will serve as the primary key of Order.
Note: Cust ID will serve as a foreign key in Order.

Note: Product Number will serve as part of the primary key of Product Order.
Note: Order Number will also serve as a foreign key in Product Order.
Note: Cust ID will serve as a foreign key in Product Order.
Note: Product Number will serve as part of the primary key of Product Order.
Note: Product Number also will serve as a foreign key in Product Order.

Last Name and First Name was moved to the Customer table to eliminate redundancy

Product Desc and Price was moved to the Product table to eliminate redundancy
Steps of Normalization

0 Normal Form

| Do any tables have repeating fields? Do some records have a different number of columns from other records? | Yes: Remove the repeating fields. Add a new table that contains the fields that repeat. |
| No: The data model is in 1NF |

First Normal Form

| Is the primary key made up of more than one field? If so, do any fields depend on only a part of the primary key? | Yes: Remove the partial dependency. Add a new table that contains the fields that are partially dependent. |
| No: The data model is in 2NF |

Second Normal Form

| Do any fields depend on another nonprimary key field? | Yes: Remove the transitive dependency. Add a new table that contains the fields that are transitively dependent. |
| No: The data model is in 3NF |

Third Normal Form
Optimizing Data Access Speed

- **De-normalization**
  - Table joins require processing
  - Add some data to a table to reduce the number of joins required (Increases data retrieval speed)
  - Creates redundancy and should be used sparingly

- **Clustering**
  - Place similar records close together on the disk
  - Reduces the time needed to access the disk
  - Ordering records in a table?
Denormalization

Last name is now in both classes
Optimizing Data Access Speed (cont.)

- **Indexing**
  - A small file with attribute values and a pointer to the record on the disk
  - Search the index file for an entry, then go to the disk to retrieve the record
  - Accessing a file in memory is much faster than searching a disk
  - Adds overhead to update; most efficient when lookup >> update.

Use indexes sparingly for transaction systems.
Use many indexes to increase response times in decision support systems.
For each table, create a unique index that is based on the primary key.
For each table, create an index that is based on the foreign key to improve the performance of joins.
Create an index for fields that are used frequently for grouping, sorting, or criteria.
Optimizing Data Access Storage

Estimating Data Storage Size

- Use volumetrics to estimate amount of raw data + overhead requirements
- This helps determine the necessary hardware capacity

<table>
<thead>
<tr>
<th>Field</th>
<th>Average Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Order Number</td>
<td>8</td>
</tr>
<tr>
<td>Date</td>
<td>7</td>
</tr>
<tr>
<td>Cust ID</td>
<td>4</td>
</tr>
<tr>
<td>Last Name</td>
<td>13</td>
</tr>
<tr>
<td>First Name</td>
<td>9</td>
</tr>
<tr>
<td>State</td>
<td>2</td>
</tr>
<tr>
<td>Amount</td>
<td>4</td>
</tr>
<tr>
<td>Tax Rate</td>
<td>2</td>
</tr>
<tr>
<td>Record Size</td>
<td>49</td>
</tr>
<tr>
<td>Overhead</td>
<td>30%</td>
</tr>
<tr>
<td>Total Record Size</td>
<td>63.7</td>
</tr>
<tr>
<td>Initial Table Size</td>
<td>50,000</td>
</tr>
<tr>
<td>Initial Table Volume</td>
<td>3,185,000</td>
</tr>
<tr>
<td>Growth Rate/Month</td>
<td>1,000</td>
</tr>
<tr>
<td>Table Volume @ 3 years</td>
<td>5,478,200</td>
</tr>
</tbody>
</table>
Designing Data Access & Manipulation Classes

- Classes that translate between the problem domain classes and object persistent classes
- ORDBMS: create one DAM for each concrete PD class
- RDBMS: may require more classes since data is spread over more tables
  - Class libraries (e.g., Hibernate) are available to help
ORDBMS DAM classes

- Participant Table
  - last_name
  - first_name
  - address
  - phone
  - birth_date
  - SubClassObjects

- Patient Table
  - amount
  - participant
  - appts
  - symptoms
  - insurance_carrier
  - primary_insurance_carrier

- Appointment Table
  - patient
  - time
  - date
  - reason

- Symptom Table
  - name
  - patients

- Appointment-DAM
  + ReadAppointmentTable()
  + WriteAppointmentTable()
  + ReadAppointment()
  + WriteAppointment()

- Patient-DAM
  + ReadPatientTable()
  + WritePatientTable()
  + ReadPatient()
  + WritePatient()

- Symptom-DAM
  + ReadSymptomTable()
  + WriteSymptomTable()
  + ReadSymptom()
  + WriteSymptom()

- Appointment-DAM
  + ReadApptTable()
  + WriteApptTable()
  + ReadAppt()
  + WriteAppt()

- ORDBMS Tables

- Data Access and Manipulation Classes

- Problem Domain Classes
Nonfunctional Requirements & Data Management Layer Design

- Operational requirements: affected by choice in hardware and operating system
- Performance requirements: speed & capacity issues
- Security requirements: access controls, encryption, and backup
- Cultural & political requirements: may affect the data storage (e.g., expected number of characters for data field, required format of a data field, local laws pertaining to data storage, etc….)
VERIFYING AND VALIDATING THE DATA MANAGEMENT LAYER

- Test the fidelity of the design before implementation
- Verifying and validating the design of the data management layer falls into three basic groups:
  1. Verifying and validating any changes made to the problem domain
  2. Dependency of the object persistence instances on the problem domain must be enforced
  3. The design of the data access and manipulation classes need to be tested
Summary

- Object Persistence Formats
- Mapping Problem-Domain Objects to Object-Persistence Formats
- Optimizing RDBMS-Based Object Storage
- Nonfunctional Requirements and Data Management Layer Design
- Designing Data Access and Manipulation Classes
- Nonfunctional Requirements & Data Management Layer Design
- Verifying and validating the data management layer