Data Management Design

Chapter 18

In this Lecture you will Learn:

- The different ways of storing persistent objects
- The differences between object and relational databases
- How to design data management objects
- How to extend sequence diagrams to include data management objects

Transient Object

- Some objects are Transient, exist in memory and are discarded when an application terminates. It called **Transient Object**.
  - Examples of transient objects include instances of boundary and control classes and the objects used in applications like on-screen calculators.

Persistence Object

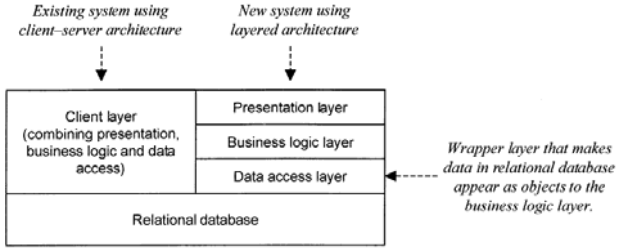
- Some objects must exist from one execution of an application to another or be shared among different instances of applications. Such objects are **Persistent Objects**.
  - Example of persistent objects include the books, borrowers and loans in a library system. A loan of a book to a borrower may be created on one occasion on one machine, and is accessed from another machine on another occasion, for example to send out overdue book reminders.

Persistence Mechanisms

- Files hold data, typically on magnetic media such as disks and tapes
- Database Management Systems (DBMS) hold tables of data (relational DBMS) or objects (object DBMS)
- DBMS use files to store data or objects, but they hide the physical processes for storing data beneath a layer of abstraction
- Objects can also be serialized directly to files

Persistence Architecture

- The choice of the architecture for persistence is a system design issue.
- The design of storage for specific classes and associations within the framework of that architecture is a class design issue.
- The overall design may be constrained by having to operate with existing systems or using existing DBMS



Persistence Design Questions

- For the architect designing the persistent storage of a system there are a number of questions to be answered.
  - Can files be used for some storage?
  - Is it truly an OO system or just an interface to a relational database using Java or C++?
  - Will the system use an existing DBMS?
  - Will it use a relational DBMS?
  - Will it use an object DBMS?
  - What is the logical layering of the system?
  - What is the physical layering of the system?
  - Is the system distributed? Does this include distributed data storage?
  - What protocols will be used to communicate within the system?

File Systems and Record Structure

- Record in files can take different form:
  - Fixed Length
  - Variable Length
  - Header and Detail
  - Tagged Data

File Systems and Record Structure

- Fixed Length (Padded)
  - Easy to process, but wastes storage

```
12345678901234567890123456789012345
Simon          Bennett        Leice
ster      GB 21322012002
```


File Systems and Record Structure

- Variable Length (Delimited)
  - More difficult to process (looking for delimiters), but saves storage space

```
”Simon”,”Bennett”,”Leicester”,”GB”,
213,”22-01-2002”
```


File Systems and Record Structure

- Header and Detail
  - Allows for structure in data, may need line type and line no.

```
1,”Simon”,”Bennett”
2,1,”0077098641”,2002
2,2,”0077096738”,2001
```


File Systems and Record Structure

- Tagged Data (XML)
  - Self-describing, but storage overhead for tags.

```
<Author>
  <Forename>Simon</Forename>
  <Surname>Bennett</Surname>
</Author>
```

File Organization in File Systems

- Serial
  - New records appended
- Sequential
  - Records ordered in file, usually according to a numeric key
- Random
  - Uses an algorithm to convert a key to an address in the file

File Access Methods in File Systems

- Serial
  - To read serial and sequential files
- Index-sequential
  - Using indexes to find records in a sequential file and improve access time
- Direct
  - Using relative or hashed addressing to move directly to the required record in the file

Example: Searching for "Hamer" using Index-sequential File Access

Master Index

Record key	Block index address
Feng	1
Patel	2
Zarzycki	3

Block Index (Index block 2)

Record key	Block address
Finlayson	55
Gomez	56
Hanson	57
Jacobson	58
....	....
Patel	84

Data Records in blocks (for simplicity only the keys are shown)

Block address	Records				
55	Fern	Finch	Finlayson		
56	Finn	Firmin	Ford	Gangar	Gomez
57	Gordon	Govan	Hamer	Hanson	
58	Ho	Ibrahim	Jacobson		
....	....				

Explanation

- In order to look for the record for "Hamer"
- Search the Master Index until you reach a key that is greater than Hamer (in string comparison terms).
- Go to the Block Index for that key (2).
- Search the Block Index until you reach a key that is greater than Hamer (in string comparison terms).
- Goto the Block for that key.
- Search through the block until you find the right record with the required key.
- (If you get to the end of the block without finding the record then it doesn't already exist.)

Direct Access

Records hashed on first three characters of key field

Khan → Kha
ASCII Values = 75, 104, 97
75 + 104 + 97 = 276
276 divided by 7 leaves a modulo of 3
So Khan will be added in Block 3.

Data records in blocks (for simplicity only the keys are shown)

Block no. in file	Records
0	Hao
1	Ford Farmer
2	Firmin Firth
3	Harris
4	Hastings Gomez Jacobson
5	Ibrahim Finch Fern Gangar
6	Hanson

Improving Access

- Creating a linked list in random files to make it possible to read records in sequential order
- Adding a secondary index keyed on a field that is not the key on which the main access method is based
- Indexes can be inverted files or use other techniques such as B-trees

File Types

- Master Files
  - Stores relatively permanent data about an entity
- Transaction Files
  - Stores records that contain day-to-day business and operational data
- Index Files
- Temporary File or Work Files
  - It is a temporary file created for a single task
- Backup Files
  - A compressed version of the original file and its locations created by Backup
- Parameter Files

File Using Example

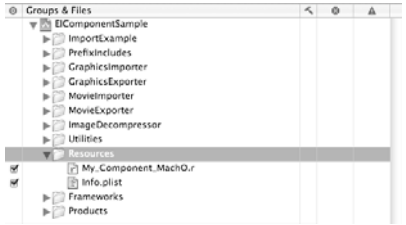
- Using files in Java to handle localization of prompts and messages
- Use the **java.util.Locale** class to hold information about the current locale
  - language_country_variant
  - fr_FR_EURO
  - fr_CA
  - en_UK
  - en_AU

File Using Example

- The Java class **java.util.ResourceBundle** uses the locale to load a file with locale-specific values.
  - E.g. UIResources_fr_FR_EURO
- Java code to use this:
  - resources = ResourceBundle.getBundle(”UIResources”, currentLocale);
  - Button cancelButton = new Button(resources.getString(”Cancel”);

Resource File

- File for French is UIResources_fr_FR_EURO
- It Contains
  - Cancel = Annuler
  - OK = OK
  - File = Fichier
  - …

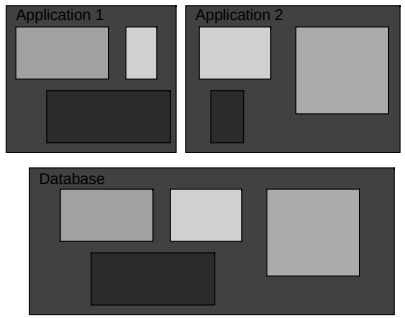


Problems with Files

- **Redundancy** – Number of files grows with applications, and data is duplicated
- **Inconsistence** – Data is updated in one application’s files, but not in another’s
- **Maintenance Problems** – Changes to data structures mean changes to many programs
- **Difficulty Combining Data** – Business needs may mean users want data from different applications

Database Management System (DBMS)

- Corporate database consolidates data for different applications
- Each application then has its own view of a subset of the data

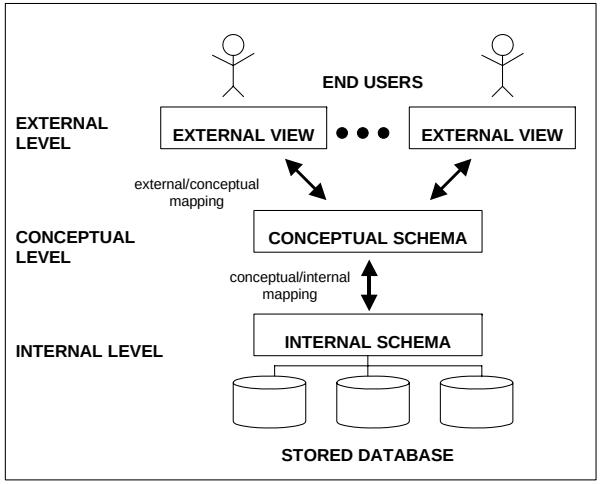


DBMS Schema

- Ultimately data in databases is stored in files, but their structure is hidden from developers

External Schema	The view on data used by application programs.
Conceptual Schema	The logical model of data that is separate from how it is used.
Internal Schema	The physical storage of data in files and indexes.

The Three Levels Architecture



DBMS Features

- Data Definition Language (DDL)
- Data Manipulation Language (DML)
- Integrity Constraints
- Transaction Management
- Concurrency
- Security
- Tuning of Storage

Advantages of DBMS

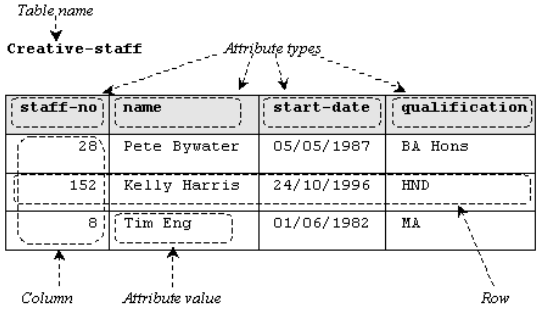
- Eliminate unnecessary duplication of data
- Enforce data integrity through constraints
- Changes to conceptual schema need not affect external schema
- Changes to internal schema need not affect the conceptual schema
- Many tools are available to manage the database

Disadvantages of DBMS

- Cost of investing in the DBMS
- Running cost, including staff (Database Administrators) to manage the DBMS
- Processing overhead in converting data to format required by programs

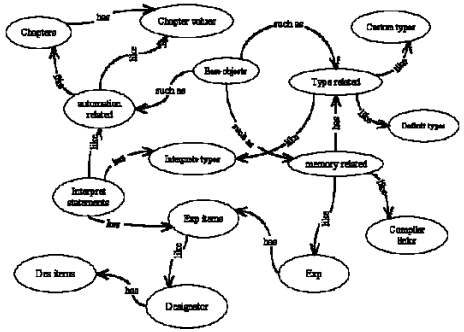
Types of DBMS: Relational

- Represent data in tables
- Tables consist of rows of data organized in columns



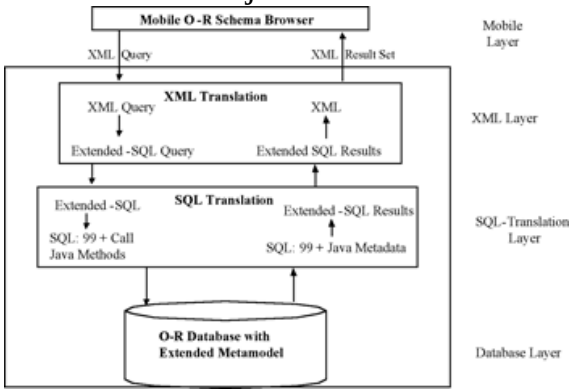
Types of DBMS: Object

- Store objects as objects
- Designed to handle complex nested objects for graphical and multimedia applications



Types of DBMS: Object-Relational

- Hybrid databases that can store data in tables but can also store objects in tables



Relational DBMS

- To store objects in a relational database, the objects have to be ‘flattened’ into tables
- Complex objects have to be taken apart and the parts stored in different tables
- When retrieved from the database, the object has to be reassembled from the parts in different tables

Normalization

- Data from complex structures is ‘flattened’ into tables
- Typically normalization is carried out as far as ‘Third Normal Form’
- In an object-oriented system, we may use normalization to convert classes to table schemas

Normalization Example

SMGL:InternationalCampaign

```
campaignCode = SMGL
campaignTitle = Soong Motors
                Granda Launch

locationsList =
[
 locationCode = HK
 locationName = Hong Kong
 locationMgr = Vincent Sieuw
 locationMgrTel = ext. 456

 locationCode = NY
 locationName = New York
 locationMgr = Martina Duarte
 locationMgrTel = ext. 312

 locationCode = TO
 locationName = Toronto
 locationMgr = Pierre Dubois
 locationMgrTel = ext. 37
]
```

YPSC:InternationalCampaign

```
campaignCode = YPSC
campaignTitle = Yellow Partridge
                Summer Collection

locationsList =
[
 locationCode = HK
 locationName = Hong Kong
 locationMgr = Jenny Lee
 locationMgrTel = ext. 413

 locationCode = NY
 locationName = New York
 locationMgr = Martina Duarte
 locationMgrTel = ext. 312
]
```


Normalization Example: Objects as a Table

- To get to First Normal Form, break out the repeating groups

InternationalCampaign

campaign Code	campaign Title	location Code	location Name	locationMgr	location MgrTel
SMGL	Soong Motors Granda Launch	HK NY TO	Hong Kong New York Toronto	Vincent Sieuw Martina Duarte Pierre Dubois	456 312 37
YPSC	Yellow Partridge Summer Collection	HK NY	Hong Kong New York	Jenny Lee Martina Duarte	413 312

Normalization Example: First Normal Form

InternationalCampaign-1

campaign Code	campaign Title	location Code	location Name	locationMgr	location MgrTel
SMGL	Soong Motors Granda Launch	HK	Hong Kong	Vincent Sieuw	456
SMGL	Soong Motors Granda Launch	NY	New York	Martina Duarte	312
SMGL	Soong Motors Granda Launch	TO	Toronto	Pierre Dubois	37
YPSC	Yellow Partridge Summer Collection	HK	Hong Kong	Jenny Lee	413
YPSC	Yellow Partridge Summer Collection	NY	New York	Martina Duarte	312

Normalization Example: Second Normal Form

InternationalCampaign-2

campaign Code	location Code	locationMgr	location MgrTel
SMGL	HK	Vincent Sieuw	456
SMGL	NY	Martina Duarte	312
SMGL	TO	Pierre Dubois	37
YPSC	HK	Jenny Lee	413
YPSC	NY	Martina Duarte	312

Campaign

campaign Code	campaignTitle
SMGL	Soong Motors Granda Launch
YPSC	Yellow Partridge Summer Collection

Location

location Code	location Name
HK	Hong Kong
NY	New York
TO	Toronto

Normalization Example: Third Normal Form

InternationalCampaign-3

campaign Code	location Code	locationMgr
SMGL	HK	Vincent Sieuw
SMGL	NY	Martina Duarte
SMGL	TO	Pierre Dubois
YPSC	HK	Jenny Lee
YPSC	NY	Martina Duarte

LocationManager

locationMgr	location MgrTel
Vincent Sieuw	456
Martina Duarte	312
Pierre Dubois	37
Jenny Lee	413

Campaign

campaign Code	campaignTitle
SMGL	Soong Motors Granda Launch
YPSC	Yellow Partridge Summer Collection

Location

location Code	location Name
HK	Hong Kong
NY	New York
TO	Toronto

Alternative Approach

- Classes with simple data structure become tables
- Object ID become primary keys
- Where classes contain another class as an attribute create a table for the embedded class
- For collections create two tables, one for the objects in the collection, the other to hold Object ID of the containing objects and the contained objects

Alternative Approach for Association

- **One-to-Many Associations** can be treated like collections
- **Many-to-Many Associations** become two separate tables for the objects and a table to hold pairs of Object ID
- **One-to-One Associations** are implemented as foreign-key attributes – each class gains an extra attribute for the Object ID of the other

Alternative Approach to Implement Inheritance

- Only implement the superclass as a table including all subclass attributes
- Only implement the subclasses as tables, duplicating superclass attributes in each
- Implement superclass and subclasses as tables with shared primary keys
- **Each approach has Disadvantages**

Object DBMS

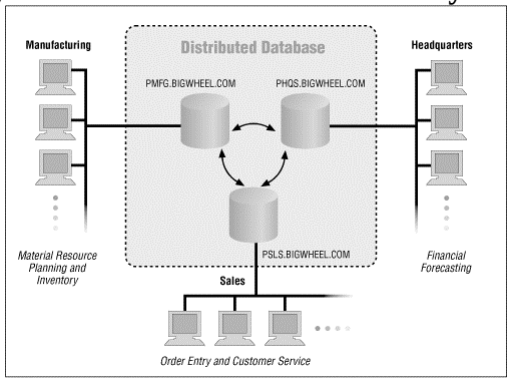
- ODBMS have the advantage that objects can be stored directly
- Based on the Object Data Management Group (ODMG) standard
- Not all object databases conform to the standard
- Object databases are closely linked to programming languages with ways of navigating through the database
- Some will transparently 'materialize' objects from the database when they are referred to
- Update transactions need to be bracketed with start and finish transaction methods
- Operations are still implemented in object-oriented languages

Sample C++ ObjectStore Operation

```
IntCampaign * CreativeStaff::findIntCampaign( string
   campaignCode )
{
   IntCampaign * intCampaignPointer;
   intCampaignPointer =
   staffIntCampaignList.getValue().query_pick(
        "IntCampaign*",
        "campaignCode == this->campaignCode",
        os_database::of(this));
   return intCampaignPointer;
}
```

Distributed Databases

- Organizations that need to decentralize their computer processing may also need to decentralized their database.
- When computing databases are scattered rather than centralized, they are called **Distributed Database Systems**.

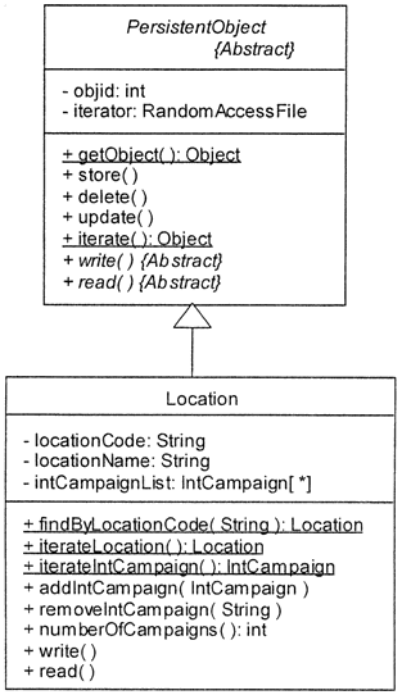


Designing Data Management Classes

- Alternatives:
  - Add save and retrieve operations to classes
  - Make save and retrieve class-scope methods
  - Allow all persistent objects to inherit from a PersistentObject superclass
  - Use collection classes to manage persistence
  - Use broker classes to manage persistence
  - Use a parameterized class to handle persistence for different classes

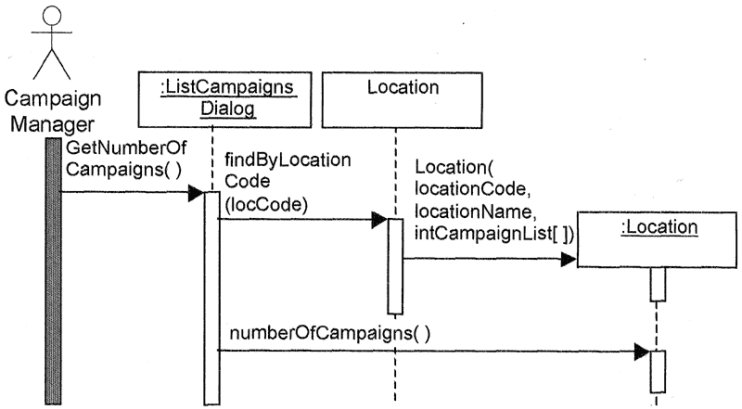
PersistentObject

- Create an abstract superclass and make all persistent classes inherit from it



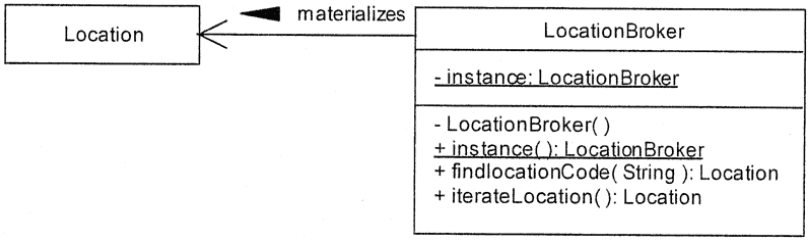
PersistentObject Materialization as Class Method

- Sequence diagram for Get number of campaigns for location showing Location retrieving (or materializing) an instance



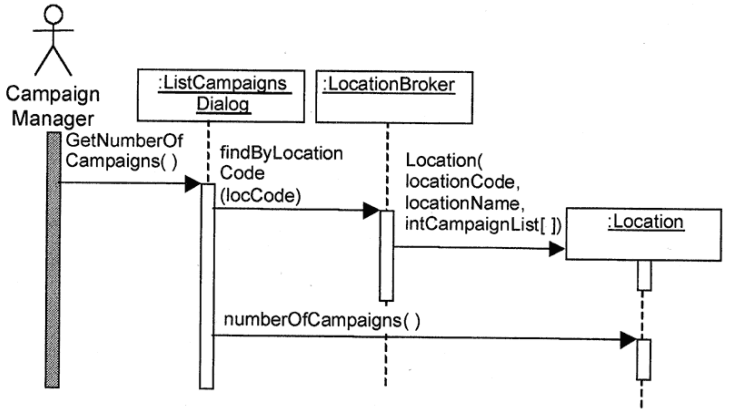
Database Broker

- Use a broker class responsible for materializing instances of each class from the database



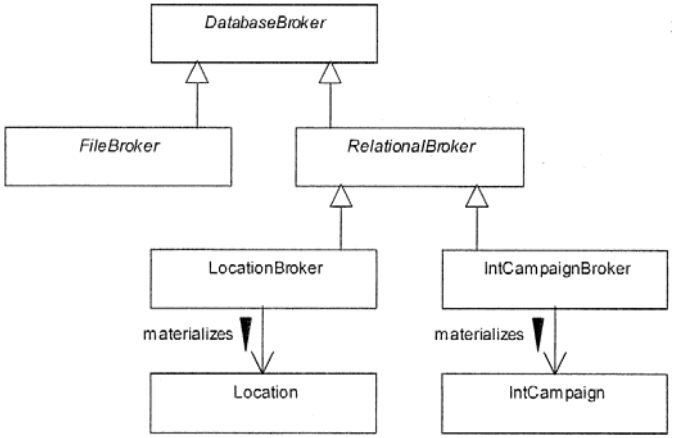
Database Broker Materializes Instances

- Sequence diagram for Get number of Campaigns for Location showing LocationBroker retrieving an instance of Location



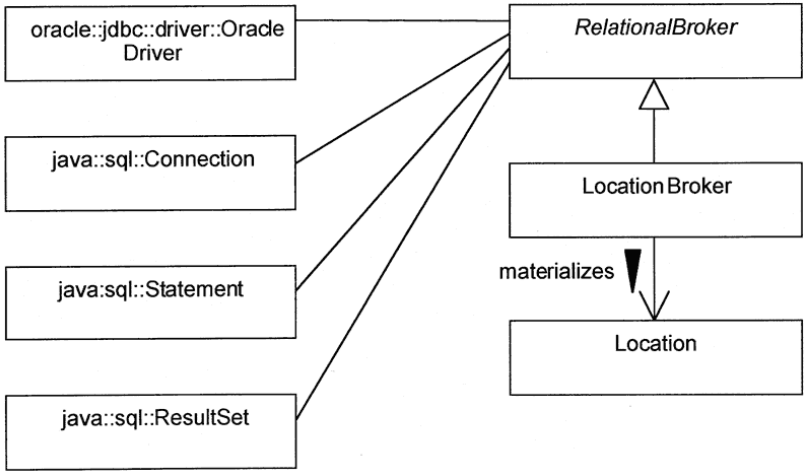
Inheritance Hierarchy of Database Brokers

- Simplified version of inheritance hierarchy for database brokers.



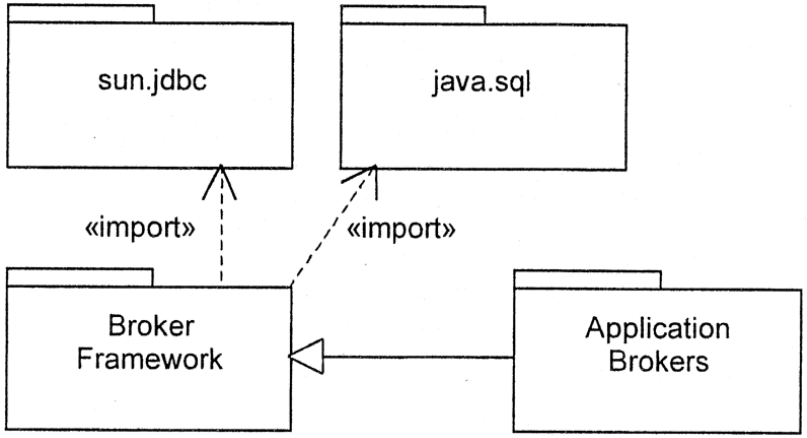
RelationalBroker and Other Classes

- RelationalBroker class and classes from other package



Package Diagram

- Class diagram showing packages for database brokers package.

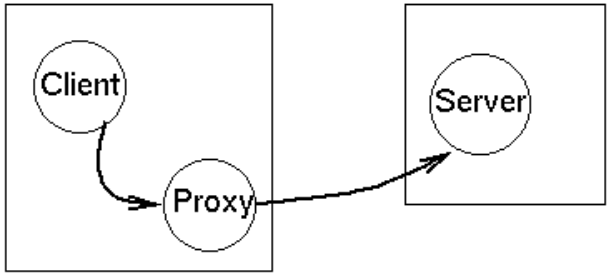


Proxy Pattern

- Proxy objects act as placeholders for the real objects, e.g. IntCampaigns in Locations
- The IntCampaignProxy has the same interface as IntCampaign, but no data
- When a Location requires data about one of its IntCampaigns, it sends a message to the Proxy
- The Proxy requests the Broker to materialize the IntCampaign and passes the message on

Proxy Pattern

- The Proxy can then replace the reference to itself in the Location with a reference to the real materialized object
- This approach can be combined with caching of objects
- The caches can be used by the Broker to check whether an object is already in memory and save materializing it from the database if it is

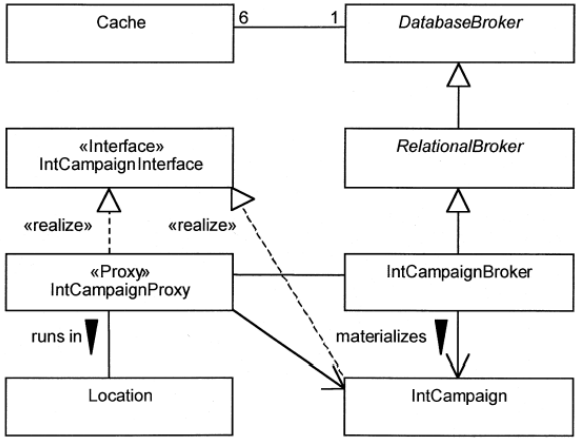


Transaction Commit Cache Actions

- Six caches
  - New clean cache
  - New dirty cache
  - New deleted cache
  - Old clean cache
  - Old dirty cache
  - Old deleted cache
- Cache actions
  - – write to database
  - – write to database
  - – delete from cache
  - – delete from cache
  - – write to database
  - – delete from database

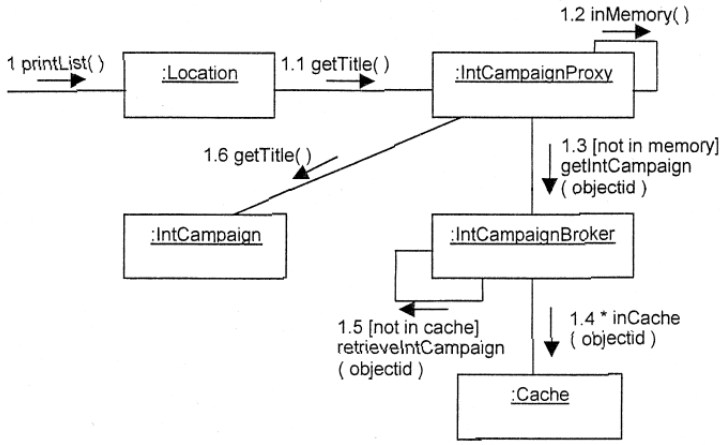
Class Diagram with Caches and Proxies

- Extension of the database broker framework to include caches and proxies.



Collaboration Diagram

- Collaboration diagram showing proxy, broker and cache objects collaborating to retrieve an object instance



Explanation

- The Location sends the message getTitle() to the IntCampaignProxy.
- The IntCampaignProxy checks whether the IntCampaign is in memory.
- It is not, so it then requests the object by object identifier from the broker, IntCampaignBroker.
- The IntCampaignBroker checks if the object is in a Cache.
- It is not, so the IntCampaignBroker retrieves the object from the database and returns it to the IntCampaignProxy.
- The IntCampaignProxy sends the getTitle() message to the IntCampaign object and returns the result to the Location.

Using a Framework

- Why develop a framework when you can use an existing one?
- Object-table mappings: Toplink & CocoBase
  - Products that will map attributes of classes to columns in a relational database table
- J2EE Application Servers
  - Enterprise JavaBeans (EJB) can be used – Entity Beans for business objects
  - Container-Managed Persistence (CMP) is a framework for J2EE containers to handle the persistence of instances of entity beans
  - Use J2EE Patterns