

CS 580 Client-Server Programming
Spring Semester, 2009
Doc 14 Databases and Architecture
19 March, 2009

Copyright ©, All rights reserved. 2009 SDSU & Roger Whitney, 5500 Campanile Drive, San Diego, CA 92182-7700 USA. OpenContent (<http://www.opencontent.org/opl.shtml>) license defines the copyright on this document.

References

Patterns of Enterprise Application Architecture, Martin Folwer, Addison-Wesley, 2003

CS 683 Lecture document 27 Hibernate Example, Fall 2005, <http://www.eli.sdsu.edu/courses/fall04/cs683/notes/hibernate/hibernate.html>

The Vietnam of Computer Science, Ted Neward, <http://blogs.tedneward.com/2006/06/26/The+Vietnam+Of+Computer+Science.aspx>

Databases & Architecture

How to keep SQL isolated?

How to isolate database connection details?

How to keep dealing with the database under control?

How to structure programs that use databases?

Topics

Organizing Domain Logic

Structuring code that accesses database

Organizing Domain Logic

How to organize an application that uses a database

Fowler provides the following methods

Transaction Script

Domain Model

Table Module

Service Layer

Transaction Script

Each request from GUI or client can be viewed as a separate transaction

Each request/transaction is handled by a separate method

Consequences

Very simple to implement

As application grows in complexity, becomes overly complex and hard to manage

Domain Model

Implement classes that incorporates both behavior & data

Classes represent objects in the domain

Program becomes collection of interacting objects

Objects map to tables

- A single object may span many tables

- A table row may contain multiple objects

Consequences

Overly complex for simple applications

Scales well to complex applications

Database organizes data differently

Table Module

For each table (or view) implement a class

Each class holds the business logic related to the data in the table

Consequences

Classes are organized around database structure rather than OO principles

Handles more complex situations than Transaction Script

Not as scalable as Domain Model

Structuring code that accesses database

Hiding database connection details

Organizing Access to Database

Issues about Database Connections

Database usernames and passwords should not be scattered in code

How much database connection detail should be scattered in the code

DatabaseConnector

Hide the details

```
public class DatabaseConnector {
    private String databaseUrl;
    private String user;
    private String password;
    private ArrayList connectionPool;

    private static DatabaseConnector instance =
        DatabaseConnector("filename");

    public static DatabaseConnector instance() {
        return instance;
    }

    private DatabaseConnector(String filename) {
        read file for database info
        set private fields
    }

    public ResultSet executeQuery( String sql ) {
        return getStatement().executeQuery( sql);
    }

    public Statement getStatement() {
        return getConnection().createStatement();
    }

    private Connection getConnection() { return a connection}

    etc
}
```

For Future Examples - Office Hours

Common Operations

- Find Office hours for instructor X
- Find office hours of any graduate advisor
- Find office hours of any undergraduate advisor
- Find office hours of any TA
- Who has office hours at time X
- What times are there no office hours
- Add office hours
- Modify office hours

Tables for Example

Faculty			
Id	Name	Office	Phone
1	Eckberg	GMCS-543	594-6834
2	Donald	GMCS-541	594-7248
3	Carroll	GMCS-537	594-7242

RoleTypes	
ID	Role
1	Undergraduate Advisor
2	Graduate Advisor
3	TA

OfficeHours				
Id	StartTime	EndTime	Day	FacultyId
1	10:00	11:00	Tuesday	1
2	10:00	11:00	Thursday	1

Roles	
FacultyId	TypeId
1	2
2	2
3	1

Organizing Access to Database

Table Data Gateway

Row Data Gateway

Active Record

Data Mapper

Table Data Gateway

One object handles all the rows in a table or view

Each table has one class that knows the table

One object represents the table – all the rows

Gateway hides all the Sql from the rest of the program

Works well with

- Table Module

- Transaction Script

OfficeHoursGateway

```
public class OfficeHoursGateway {

    private static String addOfficeHoursSql =
        "INSERT
        INTO officeHours ( startTime, endTime, day, facultyId )
        VALUES ( ? , ?, '?', ?)";

    private static String officeHoursSql =
        "SELECT startTime, endTime, day
        FROM officeHours
        WHERE facultyId = ?";

    public ResultSet officeHoursFor(int facultyId, ) {
        Statement hoursStatement = DatabaseConnector.instance().
            prepareStatement(officeHoursSql);
        hoursStatement.setObject( 1, facultyId);
        return hoursStatement.executeQuery();
    }

    public int setOfficeHoursFor(int facultyId, Time start, Time end, String day) {

        Statement addOfficeHours = DatabaseConnector.instance().
            prepareStatement(addOfficeHoursSql);

        addOfficeHours.setObject(1, start);
        addOfficeHours.setObject(2, end);
        addOfficeHours.setObject(3, day);
        addOfficeHours.setObject(4, facultyId);
        return addOfficeHours.executeQuery();
    }
}
```


Transaction Script + Table Gateway

```
public class OfficeHoursServer {
    private OfficeHoursGateway officeHours;
    private FacultyGateway faculty;
    etc.

    public Vector officeHoursFor(String facultyName) {

        int facultyId = faculty.idFor(facultyName,);

        ResultSet officeHoursRows = officeHours.officeHoursFor( facultyId);
        Vector officeHours = new Vector();
        while (officeHoursRows.next() ) {
            Dictionary officeHour = new Dictionary();
            officeHour.put( "start", officeHoursRows.getObject( "start"));
            officeHour.put( "end", officeHoursRows.getObject( "end"));
            officeHour.put( "day", officeHoursRows.getObject( "day"));
            officeHours.add( officeHour);
        }
        officeHoursRows.close();
        return officeHours;
    }
}
```

Row Data Gateway

One object handles or represents a single row in a table or view

Each table has one class that knows the table

Gateway hides all the Sql from the rest of the program

A class provides just accessor methods to data in a row

Works well with Transaction script

sdsu.sql.DatabaseTable

Utility for Row Access

Part of SDSU Java library

```
Connection db = DriverManager.getConnection( dbUrl, user, password);
```

```
DatabaseTable rows;
```

```
//Get rows from table Faculty with column Name = Donald
```

```
rows = DatabaseTable.getRow("Faculty", "Name", "Donald", db);  
rows.elementAt(rowIndex, "Office");
```

Active Record

Each domain object know how add/remove/find its state in the database

In simple cases

- Class for each table

- An object represents one row in the table

- Similar to Row Data Gateway with domain logic

Faculty Example

```
public class Facutly {
    String name;
    String phoneNumber;
    int id;
    etc.

    private final static String findByNameSql =
        "SELECT *
        FROM faculty
        WHERE name = '?'";

    public static Faculty findByName(String name ) {
        Statement find =
            dabaseConnector.prepareStatement(findByNameSql);
        find.setObject( 1, name);
        ResultSet facultyRow = find.executeQuery();
        return load(facultyRow);
    }

    private static Faculty load( ResultSet facultyRow) {
        create faculty object.
        get data out of Resultset.
        Put data into faculty object.
    }
}
```

Faculty Example

```
public boolean hasOfficeHoursAt(Time anHour) {  
    Iterator hours = officeHours().iterator();  
    while (hours.hasNext() ) {  
        OfficeHour officeHour = (OfficeHour) hours.next();  
        if (officeHour.contains( anHour ) ) return true;  
    }  
    return false;  
}
```

```
public ArrayList officeHours() {  
    if( officeHours = nil ) {  
        officeHours = OfficeHour.findFor( id );  
    }  
    return officeHours;  
}
```

etc.

Object-Relational Mapping Layers

Data Mapper

Implementing a good object-relational layer is a lot of work

Use existing tools to save a lot of time

Read/Write objects from tables without SQL

Some existing object-relational layers

- JDO – Java Data Object (Java framework)

- TopLink (Commercial - Java)

- Hibernate (Open source - Java)

- Cayenne (Open source - Java)

- GLORP (Open source - Smalltalk)

Hibernate Simple Example

Storing Person objects in table

Database Table

id	first_name	last_name

SQL Used to Create Table

```
CREATE TABLE PEOPLE  
  (FIRST_NAME varchar(50) NULL ,  
   LAST_NAME varchar(50) NULL ,  
   ID int NOT NULL ,  
   PRIMARY KEY (id));
```


Person Class

```
package sample;

public class Person {
    String firstName;
    String lastName;
    long id;

    public Person () {super(); }

    public Person(String first, String last) {
        firstName = first;
        lastName = last;
    }
    public String getLastName() { return lastName; }
    public String getFirstName() { return firstName; }
    public void setFirstName( String name) { firstName = name; }
    public void setLastName( String name) { lastName = name; }
    public long getId() { return id; }
    public void setId(long l) {id = l; }
    public String toString() {return firstName + " " + lastName + id;}
}
```

Mapping – Person.hbm.xml

Indicates how to map object fields to table columns

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC
    "-//Hibernate/Hibernate Mapping DTD//EN"
    "http://hibernate.sourceforge.net/hibernate-mapping-2.0.dtd" >

<hibernate-mapping package="sample">
    <class
        name="Person"
        table="people" >
        <id
            name="id"
            type="java.lang.Long"
            column="id" >
            <generator class="assigned"/>
        </id>
        <property
            name="lastName"
            column="last_name"
            type="string"
            not-null="false"
            length="50" />
        <property
            name="firstName"
            column="first_name"
            type="string"
            not-null="false"
            length="50" />
        </class>
    </hibernate-mapping>
```

```
import lots of stuff;
```

Sample Connection

```
public class Main {  
    public static void main(String[] args) throws Exception {  
        sampleRead();  
        sampleWrite();  
    }  
  
    static Session getHibernateSession() throws  
        MappingException, HibernateException, Exception {  
        some code to get HibernateSession  
    }  
  
    static void sampleWrite() throws  
        MappingException, HibernateException, Exception {  
        Session session = getHibernateSession();  
        Transaction save = session.beginTransaction();  
        Person newPerson = new Person("Jack", "Frost");  
        newPerson.setId(1);  
        session.save(newPerson);  
        newPerson = new Person("Jack", "Ripper");  
        newPerson.setId(2);  
        session.save(newPerson);  
        save.commit();  
        session.close();  
    }  
}
```

Sample Connection Continued

```
static void sampleRead() throws
    MappingException, HibernateException, Exception {
    Session session = getHibernateSession();
    Query getByLastName =
        session.createQuery(
            "from People p where p.lastName = :var");
    getByLastName.setString("var", "Frost");
    List result = getByLastName.list();
    System.out.println("Number of Objects: " + result.size());
    Person frost = (Person) result.get( 0);
    System.out.println(frost);
    session.close();
}
}
```

O-R Mapping - Vietnam of Computer Science

<http://blogs.tedneward.com/2006/06/26/The+Vietnam+Of+Computer+Science.aspx>

<http://www.codinghorror.com/blog/archives/000621.html>

Last mile problem & OR mapping Problem

Dual-Schema Problem

Entity Identity Issues

The Data Retrieval Mechanism

Query-By-Example (QBE)

Query-By-API (QBA)

Query-By-Language (QBL)

Some Solutions

Abandon relational databases - store objects

Abandon objects

Abandon OR-layers

Accept OR-Layer limitations

 Use SQL when easier

Add relational concepts to language

Add relational concepts to frameworks