References

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

How to keep Sql isolated?

How to isolate database connection details?
Example – 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

<table>
<thead>
<tr>
<th>Tables</th>
</tr>
</thead>
<tbody>
<tr>
<td>Faculty</td>
</tr>
<tr>
<td>Id</td>
</tr>
<tr>
<td>----</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>OfficeHours</th>
</tr>
</thead>
<tbody>
<tr>
<td>Id</td>
</tr>
<tr>
<td>----</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>RoleTypes</td>
</tr>
<tr>
<td>-----------</td>
</tr>
<tr>
<td><strong>ID</strong></td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Roles</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>FacultyId</strong></td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
</tbody>
</table>
DatabaseConnector

Hides username and password
Can we hide the connections completely?
Should we hide connections?

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
}
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
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

```java
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;
    }

    etc.
}
```
Active Record

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

In simple cases

• Class for each table
• An object represents one row in the table
public class Faculty {
    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 = 
            databaseConnector.prepareStatement(findByNameSql);
        find.setObject(1, name);
        ResultSet facultyRow = find.executeQuery();
        return load(facultyRow);
    }

    public static Faculty load(ResultSet facultyRow) {
        create faculty object.
        get data out of ResultSet.
        Put data into faculty object.
        Return faculty object.
    }
}
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;
}
Domain Model + Active Record

public class OfficeHoursServer {

    public Vector officeHoursFor(String facultyName) {

        Faculty X = Faculty.findByName (facultyName,);

        ArrayList officeHours = X.officeHours();

        Convert contents of officeHours to XML-RPC acceptable types
        return vector of valid XML-RPC types;
    }

    etc.
}

Object-Relational Mapping Layers

Tools to automate and perform the object-relational mapping

- JDO – Java Data Object
- Smalltalk GLORP – Generic Lightweight Object-Relational Persistence
Object Databases

Store and retrieve objects from the database

A partial list

- Gemstone (http://www.gemstone.com/)
- Objectivity (http://www.objectivity.com/)
- Matisse (http://www.matisse.com/)
- OmniBase (http://www.gorisek.com/homepage/index.html)
- Versant (http://www.versant.com/index)
- ObjectStore (http://www.objectstore.net/index.ssp)
- Zope Object Database (http://zope.org/Wikis/ZODB/FrontPage)

No need to convert between objects and sql
OmniBase Example

database := OmniBase createOn: 'examples'.

[OmniBase root
 at: 'restaurantTypes'
 put: Set newPersistent

OmniBase root is a Dictionary

Entry point to data

Server>>addType: aString

(aString isNil or: [aString isEmpty]) ifTrue:[^false].

[types := OmniBase root at: 'restaurantTypes'.
types add: aString.

^true

Server>>types

[^(OmniBase root at: 'restaurantTypes') asSortedCollection]
evaluateIn: database newTransaction.
Simplistic Restaurants

[OmniBase root
  at: 'restaurants'
  put: OrderedCollection newPersistent

Server>>restaurantById: anInteger
  | restaurant |
  |

[restaurants := OmniBase root at: 'restaurants'.
  restaurant := restaurants
    detect: [:each | each id = anInteger]
    ifNone: [Dictionary new].
  ^restaurant asDictionary