J2EE Access of Relational Data

- **Direct JDBC**
  - Direct SQL calls, uses rows and result sets directly

- **Object view**
  - Accessed as objects or components, transparent that the data is stored in relational database
  - Need persistence layer in middle tier to handle the object-relational mapping and conversion
Direct Access with JDBC

- Java standard for accessing databases
- JDBC is simply the database connection utilities Java developers need to build upon
Direct Access with JDBC

- Developer codes all interactions with database
  - Low level JDBC for query, insert, update, delete operations
- Low overhead from container
  - Developer is responsible for opening/closing resources
  - Transactions managed programmatically, no CMT
- Leads towards database model coded into application
  - Use of meta data is possible but requires implementation
Data Access Object

- Common design pattern for direct JDBC access
  - Encapsulates the JDBC calls to persist an object
  - Separates storage from business logic
  - DAO interface allows for different implementations
  - Callable from Web tier, Session and BMP beans
  - Easy to test

- Provides familiar CRUD methods
Data Access Object

- Design pattern for direct JDBC access

Class: TopicDAOImpl

```java
public class TopicDAOImpl implements TopicDAO {
    public int createTopic(…) {
        // direct JDBC code
    }
}
```

Interface: TopicDAO

- int createTopic(…)
- Topic queryById(…)
- int updateTopic (…)
- int deleteTopic(…)

implements
DEMONSTRATION

JDBC Example
Direct Access with JDBC

Issues:
- Concurrency considerations
- DBA Concerns: Review SQL
**Object View**

- Abstracts persistence details from the application layer

- Object-level querying and creation results are objects

- Results are returned as raw data

- Persistence Layer

- JDBC

- API uses SQL or database specific calls

- Object creation and updates through object-level API
Entity Beans

- Used to represent data from a data source
- Sharable across multiple clients
- Persistent:
  - Bean-Managed Persistence (BMP)
  - Container-Managed Persistence (CMP)
## Entity Bean Lifecycle and Database

<table>
<thead>
<tr>
<th>Method</th>
<th>Operation</th>
<th>DB Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>ejbCreate()</td>
<td>Create an entity bean instance</td>
<td><strong>INSERT</strong> to persist the bean instance</td>
</tr>
<tr>
<td>ejbPostCreate()</td>
<td>Set relationship</td>
<td><strong>UPDATE</strong> foreign key or <strong>INSERT</strong> if deferred write</td>
</tr>
<tr>
<td>ejbLoad()</td>
<td>Load a bean instance using the supplied primary key</td>
<td><strong>SELECT</strong> statement to retrieve a database row</td>
</tr>
<tr>
<td>ejbStore()</td>
<td>Update the corresponding bean in the database</td>
<td><strong>UPDATE</strong> to a database row</td>
</tr>
<tr>
<td>ejbRemove()</td>
<td>Remove a bean instance</td>
<td><strong>DELETE</strong> the record</td>
</tr>
<tr>
<td>findByXXX()</td>
<td>Find the beans based on a certain condition</td>
<td><strong>SELECT</strong> statement to fetch one or multiple rows</td>
</tr>
</tbody>
</table>
Bean Managed Persistence

• Using BMP
  – All database interactions are manually coded
  – Container calls your code when it performs an operation
  – Container still ultimately in control

• Provides some implementation flexibility
  – Map to 1 or more tables, different storage systems
  – Make use of existing stored procedures

• Performance perceptions and reality
  – Developer coded *may* be better than container generated
public Integer ejbCreate(Integer id, String name) throws CreateException {
    log(this, "ejbCreate pk=" + id);
    String SQL = "insert into " + m_table + "(id,name) values (" + id + ", " + name + ")";

    Connection con = null;
    Statement stmt = null;

    try {
        setId(id);
        setName(name);

        con = getConnection();
        stmt = con.createStatement();
        int rows = stmt.executeUpdate(SQL);
        return getId();
    }
}
BMP Implementation

EJB Bean Methods

- ejbCreate
- ejbLoad
- ejbStore
- ejbRemove
- ejbFindByPrimaryKey
- ejbFindByXYZ

Code

```java
public Integer ejbFindByPrimaryKey(Integer primaryKey) throws FinderException {
    Connection con = null;
    PreparedStatement stmt = null;
    ResultSet rset = null;
    try {
        con = getConnection();
        stmt = con.prepareStatement("select * from "+ m_table + " where id=?");
        stmt.setInt(1, primaryKey);
        rset = stmt.executeQuery(SQL);
        rset.next();
        return primaryKey;
    } catch (Exception ex) {
        ex.printStackTrace();
        throw new FinderException(ex.getMessage());
    }
}
```
BMP Reality Check

- Requires
  - Same level of coding as JDBC/DAO
  - Implementation of additional EJB interfaces and semantics
  - Construction of EJB XML deployment descriptor
  - Additional packaging and deployment
- Harder to test than simple JDBC/DAO
- Preferred use is historically motivated and based on outdated information
- How many ECPref submissions used BMP?

Don’t use BMP in modern EJB containers!
Container Managed Persistence

- Using CMP
  - Developer specifies persistent attributes of object
  - Developer specifies how entity object maps to database
  - All JDBC code generated by container
- Container is responsible for managing persistence
- Container maintains relationships between objects
- Requires less code than JDBC and BMP
- Choice for persistence in J2EE Applications
Container Managed Relationships

- Only between CMP EJBs
- Supports 1-1, 1-M, and M-M
- Relationships can be unidirectional or bi-directional
- Relationships are persistent just like EJBs
- Can have DELETE-CASCADE property

**Customer**
- id: int
- name: String
- creditRating: int

**Address**
- id: int
- city: String
- zip: String
Performance Concerns

- Container generates the SQL
  - SQL statement that can cause performance degradation by making full table scans or unnecessary joins
  - Unnecessary Extra SQL statements
- Developers specify finder methods that can make full table scans
- How to tune database for use with CMPs
Optimizing CMP Entity beans for Oracle database

- Reduce database operations
- Use right concurrency mode and locking strategy
- Tune your SQL/database
DEMONSTRATION

CMP Example
Next Generation of J2EE Tools

- Reach a Broader Audience
  - Bring the power of J2EE to corporate developers
- Increase Productivity and Ease of Use
  - Visual and declarative development
  - Less coding, more reuse
  - Bridge the usability gap with C/S tools
- Reduce Learning Curve / Staffing Issues
  - Provide abstraction on top of J2EE
Oracle ADF
Visual and Declarative Development

- End-to-end Application Development
  - J2EE & Services
- Visual
  - WYSIWYG editors
  - UML modelers
  - Structure pane
- Declarative
  - Structure pane
  - Property inspector
- Code View/Design View Synchronization
  - No separate generation step - always synchronized
  - Underlying code always accessible
ADF Architecture Overview

- Rich Clients
- Web and Wireless Clients
- Controller
- Model
- Business Services
Business Services

- Manages Persistence
  - O/R mapping
  - Queries/DML
- Performs Validation
  - Data validation
  - Business logic
- Choices of implementation
  - EJB, ADF Business Components, Toplink, Web Services, Java classes
ADF Business Components

- Complete business tier development framework
- Implements J2EE design patterns
- Best practice solutions for:
  - Business logic implementation
  - Scalability and performance
  - Optimized database interaction
- Standard deployment options:
  - EJB, web tier, web services
Enterprise JavaBeans (EJB)

- Integrated environment to model, code, test, and debug EJBs
- Productively develop EJBs
  - EJB Module Editor
  - Design patterns support
    - Data Transfer Object
    - Session Façade
  - EJB Verifier
TopLink

- Object-Relational Persistence Architecture
  - Java Classes – “POJO”
  - CMP Entity Beans
- TopLink Mapping Editor
- Mappings verifier
- Unit test mappings
Web Services

- **Enterprise-Class J2EE Web Services**
  - Asynchronous conversations
  - Interoperability with .NET Web Services
  - Productive development
    - UML Modeler
    - TCP packet monitor
    - PL/SQL Web Service wizard
    - WS-I compliance tester
    - ADF Data Control Binding
ADF Architecture Overview

Rich Clients

Web and Wireless Clients

Controller

Model

Business Services
ADF Model

- Data Controls
- Data Binding
- JSR-227: “A Standard Data Binding & Data Access Facility for J2EE”
Drag-and-Drop Binding

Data Control Palette
- Expose any business service
- Available to any client
DEMOnSTRATION

ADF Example
J2EE Applications Have Inherent Complexity

Client Side Presentation
- Browser
  - Pure HTML
  - Java Applet
- Desktop
  - Java Application
- Device
  - J2EE Client

Server-Side Presentation
- Web Server
  - JSP
  - Servlet
- EJB Container
  - EJB

Server-Side Business Logic
- EJB Container
  - EJB
  - J2EE Platform

Enterprise Information Systems
- Database
  - Database
  - Database
Tuning JDBC Performance: Start with the Obvious

- Use connection pooling
  - Connection objects are expensive
  - Tailor min and max connections to your application
- Avoid cycling physical database connections
  - Look for database connections timing out
- Tune statement caching
  - Cache distinct SQL statements
No Connection Pooling

J2EE Container

FAQEJB
- create
- findby
- setQuestion
- getQuestion
- setTopics
- getTopics

Make connection, do Query

Return result and disconnect (unless application itself does connection pooling)
With Connection Pooling

J2EE Container
FAQEJB
- create
- findBy
- setQuestion
- getQuestion
- setTopics
- getTopics

Application uses available connections from the container.
How Can You Tell?

• Use your database management console
• Use your application server JDBC debug options
• Use your application server monitoring console

OC4J Spy
Tune Your SQL!

- Easier said than done
  - What is the real SQL running in CMP EJB?
- Look at the SQL on the wire
  - Tools like P6Spy, Oracle Enterprise Manager
- Become good friends with your DBA
  - Tune using traditional techniques
    - Explain plan
    - Tools like SQLPlus and Oracle Jdeveloper 10g
DEMONSTRATION

OC4J Spy, P6Spy and Oracle JDeveloper
Tuning EJB Performance

- Configure data sources appropriately
- Use appropriate locking-mode and isolation (CMP Entity Beans)
- Understand transaction usage across EJB types
- Take advantage of container optimizations
Choose the Right CMP
Locking-Mode and Isolation

<table>
<thead>
<tr>
<th>locking-mode</th>
<th>isolation</th>
<th>When to use</th>
<th>Performance Impact</th>
</tr>
</thead>
<tbody>
<tr>
<td>pessimistic</td>
<td>committed</td>
<td>When data consistency must be guaranteed and frequent updates to the same rows are expected</td>
<td>**</td>
</tr>
<tr>
<td>pessimistic</td>
<td>serializable</td>
<td>Recommend that this combination not be used.</td>
<td>*</td>
</tr>
<tr>
<td>optimistic</td>
<td>committed</td>
<td>If concurrent reads and updates to the same rows with read-committed semantics is sufficient</td>
<td>***</td>
</tr>
<tr>
<td>optimistic</td>
<td>serializable</td>
<td>If data consistency must be guaranteed but infrequent concurrent updates to the same rows</td>
<td>**</td>
</tr>
<tr>
<td>read-only</td>
<td>committed</td>
<td>If repeatable read is not required</td>
<td>****</td>
</tr>
<tr>
<td>read-only</td>
<td>serializable</td>
<td>If repeatable read is required</td>
<td>****</td>
</tr>
</tbody>
</table>

Note: All settings dependent on application requirements
Legend: * Slowest  ** Slow  *** Faster  **** Fastest
Transactions and Performance

- Entity beans load/store data at transaction boundaries
  - Transactions settings affect how often database is accessed
  - Poor performance can be caused by transaction settings
- Rules of thumb
  - Always use REQUIRED for entity bean methods
  - Cover the unit of work in from the SessionFacade with a REQUIRED transaction
System.out.println("<Create Test>":UIButtonType
for(int i=0;i<3;i++)
{
    TopicLocal topic =
    topicHome.create(
       new Integer(i),("topic " + i));
    topic.setDesc("desc" + i);
}
System.out.println("</Create Test>"uxtaposed
Resulting Transactional Activity

<Tx create>
TopicBean: ejbCreate id = 0
TopicBean: ejbStore id = 0
TopicBean: ejbLoad id = 0
TopicBean: ejbStore id = 0
TopicBean: ejbCreate id = 1
TopicBean: ejbStore id = 1
TopicBean: ejbLoad id = 1
TopicBean: ejbStore id = 1
TopicBean: ejbCreate id = 2
TopicBean: ejbStore id = 2
TopicBean: ejbLoad id = 2
TopicBean: ejbStore id = 2
</Tx create>

Requires:
12 lifecycle calls
Session Bean - Tx:Required
Entity Bean - Tx:Required

TopicSessionFacade
- createTopicSet
- printTopicSet
- deleteTopicSet

Topic
- create
- findBy
- getTopicId
- getTopicDesc
- getTopicName
- setTopicId
- setTopicDesc
- setTopicName

```java
System.out.println("<Create Test>");
for(int i=0;i<3;i++)
{
    TopicLocal topic =
        topicHome.create(
            new Integer(i),("topic " + i));
    topic.setDesc("desc" + i);
}
System.out.println("</Create Test>");
```
Resulting Transactional Activity

Tx : createTopic

Same code: 6 lifecycle calls
Take Advantage of Your EJB Container Configuration

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Performance Characteristic Impacted</th>
</tr>
</thead>
<tbody>
<tr>
<td>call-timeout</td>
<td>Session &amp; Entity</td>
<td>Specifies the maximum time to wait for any resource that the EJB container needs before the container calls the EJB method (excluding DB).</td>
</tr>
<tr>
<td>max-tx-retries</td>
<td>Session &amp; Entity</td>
<td>Specifies the number of times to re-try a transaction that was rolled back due to system level failures.</td>
</tr>
<tr>
<td>do-select-beforeinsert</td>
<td>CMP</td>
<td>Recommend setting to false to avoid the extra select before insert which checks if the entity already exists before doing the insert. This will then detect a duplicate, if there is one, during the insert.</td>
</tr>
<tr>
<td>read-only</td>
<td>CMP</td>
<td>Multiple users can execute the entity bean in parallel. The container does not allow any updates to the bean’s state.</td>
</tr>
<tr>
<td>update-changed field-only</td>
<td>CMP</td>
<td>Specifies whether the container updates only modified fields or all fields to when ejbStore is invoked. Default true.</td>
</tr>
<tr>
<td>cache-timeout</td>
<td>Stateless Session</td>
<td>Specifies how long to keep stateless sessions cached in the pool.</td>
</tr>
</tbody>
</table>

**Recommended Configuration:**

- Set `call-timeout` to a value that ensures timely response without causing timeouts.
- Set `max-tx-retries` to a value that allows retries without exceeding system resources.
- Set `do-select-beforeinsert` to false to avoid unnecessary checks.
- Set `read-only` to false to allow updates to the bean’s state.
- Set `update-changed field-only` to true to optimize performance.
- Set `cache-timeout` to a value that ensures sessions are kept in memory for an appropriate duration.