Data Management

Dr. Alejandro Zunino
ISISTAN – UNICEN

Layered Application Architecture

- Presentation layer
  - Concerned with presenting the results of a computation to system users and with collecting user inputs
- Application processing layer
  - Concerned with providing application specific functionality, e.g., in a banking system, banking functions such as open account, close account, etc.
- Data management layer
  - Concerned with managing the system databases

Underestimation

- Managing persistence related issues is the most underestimated challenge in enterprise Java today - in terms of complexity, effort and maintenance

What do RDBs do well?

- Work with large amounts of data
  - Searching, sorting
- Work with sets of data
  - Joining, aggregating
- Sharing
  - Concurrency (Transactions)
  - Many applications
- Integrity
  - Constraints
  - Transaction isolation

What do RDBs do badly?

- Modeling
  - No polymorphism
  - Fine grained models are difficult
- Business logic
  - Stored procedures kinda suck
- Distribution
  - (arguable, I suppose)

Persistence Layer

- Accessed as objects or components
- Transparent that the data is stored in RDB
- Persistence layer in middle tier handles object-relational mapping and infrastructure
- Required if doing business logic in the middle tier!
### Persistence Layer

- Abstracts persistence from the application

---

### Impedance Mismatch

- The differences in relational and object technology is known as the “object-relational impedance mismatch”
- Challenging problem to address because it requires a combination of relational database and object expertise

---

### Impedance Mismatch

<table>
<thead>
<tr>
<th>Factor</th>
<th>J2EE</th>
<th>RDB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Logical View</td>
<td>Objects, methods, inheritance</td>
<td>Tables, SQL</td>
</tr>
<tr>
<td>Scale</td>
<td>Hundreds of MB</td>
<td>GB, TB</td>
</tr>
<tr>
<td>Relationships</td>
<td>Memory references</td>
<td>Foreign keys</td>
</tr>
<tr>
<td>Uniqueness</td>
<td>Internal OID</td>
<td>Primary keys</td>
</tr>
<tr>
<td>Key skills</td>
<td>Object modeling, OOP</td>
<td>SQL, relational modeling</td>
</tr>
</tbody>
</table>

---

### J2EE Developer Desires

- Data model should not constrain object model
- Don’t want database code in object/component code
- Accessing data should be fast
- Minimize calls to database
- Object-based queries, not SQL
- Isolate J2EE application from schema changes

---

### DBA Desires

- Adhere to rules of database – referential integrity, stored procedures, sequences, etc.
- Build the J2EE application but do NOT expect to change schema
- Build the J2EE application but the schema might change
- Let DBA influence/change database calls/SQL generated to optimize
- Be able to profile all SQL calls to database
- Leverage database features where appropriate (outer joins, sub queries, specialized database functions)

---

### Differences

- Desires are contradictory
  - “Insulate application from details of database but let me leverage the full power of it”
  - Different skill sets
  - Different methodologies
  - Different tools
- Technical differences must also be considered!
 Persistence Checklist

- Mappings
- Object traversal
- Queries
- Transactions
- Optimized database interaction
- Database Features
- Cascade Deletes and Integrity
- Caching
- Locking

Mapping

- Object model and Schema must be mapped
  - True for any persistence approach
- Most contentious issue facing designers
  - Which classes map to which table(s)?
  - How are relationships mapped?
  - What data transformations are required?

Good and Poor Mapping Support

- Good mapping support:
  - Domain classes don’t have to be “tables”
  - References should be to objects, not foreign keys
    - Database changes (schema and version) easily handled
- Poor mapping support:
  - Classes must exactly mirror tables
  - Middle tier needs to explicitly manage foreign keys
  - Classes are disjointed
  - Change in schema requires extensive application changes

An Example

1:1 Relationship

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

Address
- id: int
- city: String
- zip: String

Typical 1:1 Relationship Schema

<table>
<thead>
<tr>
<th>CUST</th>
<th>ADOR</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>ID</td>
</tr>
<tr>
<td>NAME</td>
<td>CITY</td>
</tr>
<tr>
<td>C_RATING</td>
<td>ZIP</td>
</tr>
</tbody>
</table>

Other possible Schemas...
**Mapping Summary**

- Just showed several valid ways a 1-1 relationship could be represented in a database
  - Most persistence layers and application servers will only support one
- Without good support, designs will be forced
- Imagine the flexibility needed for other mappings like 1-M and M-M

**Object Traversal – Lazy Reads**

- J2EE applications work on the scale of a few hundreds of megabytes
- Relational databases routinely manage gigabytes and terabytes of data
- Persistence layer must be able to transparently fetch data “just in time,” usually called “lazy reads” or “lazy instantiation.”

**Lazy Reads**

![Diagram: Customer accessing relationship for first time and getting related object based on PK]

**Relationships Traversal**

- Even with lazy reads, object traversal is not always ideal
  - To find a phone number for the manufacturer of a product that a particular customer bought, may do several queries:
    - Get customer in question
    - Get orders for customer
    - Get parts for order
    - Get manufacturer for part
    - Get address for manufacturer
  - Very natural object traversal results in 5 queries to get data that can be done in 1

**N+1 Reads Problem**

- Many persistence layers and application servers have an N+1 reads problem
- Causes N subsequent queries to fetch related data when a collection is queried for
- This is usually a side effect of the impedance mismatch and poor mapping and querying support in persistence layers

**N+1 Reads Problem**

- In order to read data from N entity beans
  - call a finder method (one database call).
  - the container will then execute ejbLoad() individually on each entity bean returned by the finder method
  - This means that ejbLoad() (which will execute a database call) will need to be called for each entity bean.
  - Thus, a simple database query operation requires N + 1 database calls when going through the entity bean layer!
N+1 Reads Problem

- Must have solution to minimize queries
- Need flexibility to reduce to 1 query, 1+1 query or N+1 query where appropriate
  - 1 Query when displaying list of customers and addresses – known as a "Join Read"
  - 1+1 Query when displaying list of customers and user may click button to see addresses – known as a "Batch Read"
  - N+1 Query when displaying list of customers but only want to see address for selected customer

Queries

- Java developers are not usually SQL experts
  - Maintenance and portability become a concern when schema details hard-coded in application
- Allow Java based queries that are translated to SQL and leverage database options
  - EJB QL,
  - object-based proprietary queries: HQL
  - query by example

Cascaded Deletes

- Cascaded deletes done in the database have a real effect on what happens at J2EE layer
- Middle tier app must:
  - Be aware a cascaded delete is occurring
  - Determine what the "root" object is
  - Configure persistence settings or application logic to avoid deleting related objects already covered by cascaded delete

Integrity

- Java developers manipulate object model in a manner logical to the business domain
- May result in ordering of INSERT, UPDATE and DELETE statements that violate database constraints
- Persistence layer should automatically manage this and allow options for Java developer to influence order of statements
### Transaction Management

- J2EE apps typically support many clients sharing small number of db connections
- Ideally would like to minimize length of transaction on database

### Locking

- J2EE developers want to think of locking at the object level
- Databases may need to manage locking across many applications
- Persistence layer or application server must be able to respect and participate in locks at database level

### Other Issues

- Use of special types
  - BLOB, Object Relational
- Open Cursors
- Batch Writing
- Sequence number allocations
- Connection pooling
- Audit logging

### Do you Still Want to Write Code for Managing Persistence...

### Technological Alternatives

- Direct JDBC:
  - by hand...
  - only for very small systems
- Object/Relational:
  - Heavyweight:
    - EJB BMP or EJB CMP (2.1)
  - Lightweight:
    - EJB 3 (still beta...)
    - Hibernate
    - JDO
    - OJB
    - Toplink