

CS520 Web Programming

Object-Relational Mapping with Hibernate

Chengyu Sun
California State University, Los Angeles

The Object-Oriented Paradigm

- ◆ The world consists of objects
- ◆ So we use object-oriented languages to write applications
- ◆ We want to store some of the application objects (a.k.a. persistent objects), e.g. *accounts*, *customers*, *employees*
- ◆ So we use a Object Database?

The Reality of DBMS

- ◆ Relational DBMS are still predominant
 - Best performance
 - Most reliable
 - Widest support
- ◆ Bridge between OO applications and relational databases
 - CLI and embedded SQL
 - Object-Relational Mapping (ORM) tools

Call-Level Interface (CLI)

- ◆ Application interacts with database through functions calls

```
String sql = "select name from items where id = 1";
```

```
Connection c = DriverManager.getConnection( url );  
Statement stmt = c.createStatement();  
ResultSet rs = stmt.executeQuery( sql );
```

```
if( rs.next() ) System.out.println( rs.getString("name") );
```

Embedded SQL

- ◆ SQL statements are embedded in host language

```
String name;  
#sql {select name into :name from items where id = 1};  
System.out.println( name );
```

Employee – Application Object

```
public class Employee {  
  
    Integer id;  
    String name;  
    Employee supervisor;  
  
}
```

Employee – Database Table

```
create table employees (  
    id            integer primary key,  
    name         varchar(255),  
    supervisor   integer references employees(id)  
);
```

From Database to Application

- ◆ So how do we construct an Employee object based on the data from the database?

```
public class Employee {  
    Integer    id;  
    String     name;  
    Employee  supervisor;  
  
    public Employee( Integer id )  
    {  
        // access database to get name and supervisor  
        ...  
    }  
}
```

Problems with CLI and Embedded SQL ...

- ◆ SQL statements are hard-coded in applications

```
public Employee( Integer id ) {  
    ...  
    PreparedStatement p;  
    p = connection.prepareStatement(  
        "select * from employees where id = ?"  
    );  
    ...  
}
```

... Problems with CLI and Embedded SQL ...

- ◆ Tedious translation between application objects and database tables

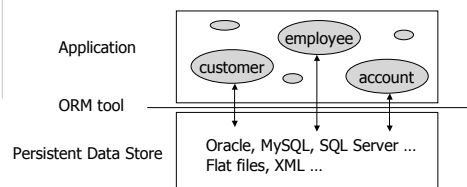
```
public Employee( Integer id ) {  
    ...  
    ResultSet rs = p.executeQuery();  
    if( rs.next() )  
    {  
        name = rs.getString("name");  
        ...  
    }  
}
```

... Problems with CLI and Embedded SQL

- ◆ Application design has to work around the limitations of relational DBMS

```
public Employee( Integer id ) {  
    ...  
    ResultSet rs = p.executeQuery();  
    if( rs.next() )  
    {  
        ...  
        supervisor = ??  
    }  
}
```

The ORM Approach



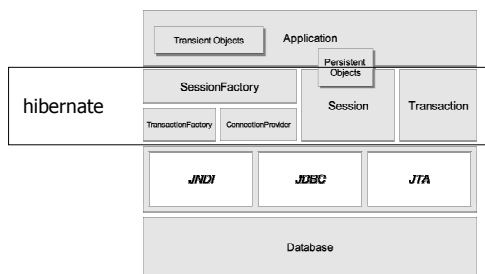
Advantages of ORM

- ◆ Make RDBMS look like ODBMS
- ◆ Data are accessed as objects, not rows and columns
- ◆ Simplify many common operations. E.g. `System.out.println(e.supervisor.name)`
- ◆ Improve portability
 - Use an object-oriented query language (OQL)
 - Separate DB specific SQL statements from application code
- ◆ Caching

Common ORM Tools

- ◆ Java Data Object (JDO)
 - One of the Java specifications
 - Flexible persistence options: RDBMS, OODBMS, files etc.
- ◆ Hibernate
 - Most popular Java ORM tool right now
 - Persistence by RDBMS only
- ◆ Others
 - http://en.wikipedia.org/wiki/Object-relational_mapping
 - http://www.theserverside.net/news/thread.tss?thread_id=29914

Hibernate Application Architecture



Setup Hibernate

- ◆ Download *hibernate-3.0.5.zip* from <http://www.hibernate.org/6.html>
- ◆ Add the following jar files to CLASSPATH
 - *hibernate-3.0\hibernate3.jar*
 - All the jar files under *hibernate-3.0\lib*
 - The JDBC driver of your DBMS

A Simple Hibernate Application

- ◆ Java classes
 - `Employee.java`
- ◆ O/R Mapping files
 - `Employee.hbm.xml`
- ◆ Hibernate configuration file
 - `hibernate.cfg.xml`
- ◆ (Optional) Logging configuration files
 - `log4j.properties`
- ◆ Code to access the persistent objects
 - `EmployeeTest1.java`

Java Classes

- ◆ Plain Java classes (POJOs); however, it is *recommended* that
 - Each persistent class have an identity field
 - Each persistent field have a pair of getter and setter, *which don't have to be public*
- ◆ The identity field is used to uniquely identify an object
- ◆ The persistent fields are accessed as bean *properties*

O/R Mapping Files

- ◆ Describe how class fields are mapped to table columns
- ◆ Three important types of elements in a mapping file
 - <id>
 - <property> - when the field is of simple type
 - Association – when the field is of a class type
 - <one-to-one>
 - <many-to-one>
 - <many-to-many>

Hibernate Configuration Files

- ◆ Tell hibernate about the DBMS and other configuration parameters
- ◆ Either hibernate.properties or hibernate.cfg.xml or both
 - Sample files under *hibernate-3.0/etc*

Logging

- ◆ Use print statements to assist debugging
 - Why do we want to do that when we have GUI debugger??

```
public void foo()
{
    System.out.println( "loop started" );
    // some code that might get into infinite loop
    ...
    System.out.println( "loop finished" );
}
```

Requirements of Good Logging Tools

- ◆ Minimize performance penalty
- ◆ Support different log output
 - Console, file, database, ...
- ◆ Support different message levels
 - Fatal, error, warn, info, debug, trace
- ◆ Easy configuration

Log4j and Commons-logging

- ◆ Log4j
 - A logging tool for Java
 - <http://logging.apache.org/log4j/docs/>
- ◆ Commons-logging
 - A wrapper around different logging implementations to provide a consistent API
 - <http://jakarta.apache.org/commons/logging/>

Log4j Configuration File

- ◆ log4j.properties specifies
 - Output type
 - Output format } Appender
 - Class
 - Message level } Logger

Logging Example

- ◆ hex.test.LogTest

Access Persistent Objects

- ◆ Session
- ◆ Query
- ◆ Transaction
 - A transaction is required for updates

Hibernate Query Language (HQL)

- ◆ A query language that looks like SQL, but for accessing *objects*
- ◆ Automatically translated to DB-specific SQL statements
- ◆

```
select e from Employee e where e.id = :id
```

 - From all the Employee objects, find the one whose id matches the given value

CRUD Example

- ◆ EmployeeTest2.java
 - "from Employee"
 - load() or get()?
 - How does hibernate tell whether an object is new??
 - Caching

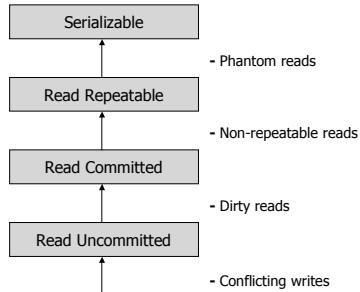
load() vs. get()

- ◆ load() raises an exception if an object cannot be found; get() would return null
- ◆ load() may return a proxy but get() never does

Caching

- ◆ Object cache and query cache
- ◆ Cache scopes
 - Session
 - Process
 - Cluster

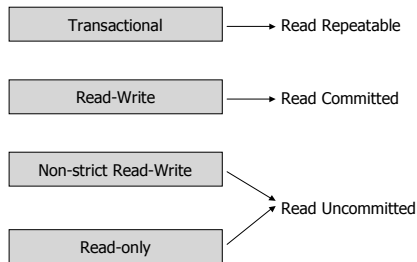
Transaction Isolation Levels



Caching in Hibernate

- ◆ First-level cache
 - Session scope
 - Always on (and cannot be turned off)
- ◆ Second-level cache
 - Pluggable *Cache Providers*
 - Process cache
 - EHCACHE and OSCache
 - Cluster cache
 - SwarmCache and JBossCache

Hibernate Cache Concurrency Policies



Currency Support of Hibernate Cache Providers

	Read-only	Non-strict Read-Write	Read-Write	Transactional
EHCACHE	X	X	X	
OSCache	X	X	X	
SwarmCache	X	X		
JBossCache	X			X

hbm2ddl

- ◆ Generate DDL statements from Java classes and mapping files
- ◆ db/
 - hex.ddl – generated automatically by hbm2ddl
 - hex.sql – based on hex.ddl but maintained manually

More About Mapping

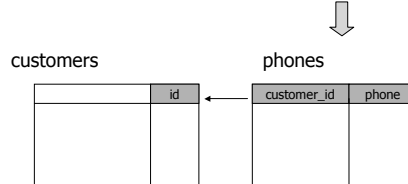
- ◆ Basic mapping
 - <id>
 - <property>
 - Association
 - many-to-one
 - one-to-many
 - one-to-one
 - many-to-many
- ◆ Collections
 - ◆ Subclasses
 - ◆ Components
 - ◆ Other
 - Bidirectional association
 - Multi-way relationship

Collection of Simple Types

```
public class Customer {
    Integer id;
    String name;
    String address;
    Set<String> phones;
}
```

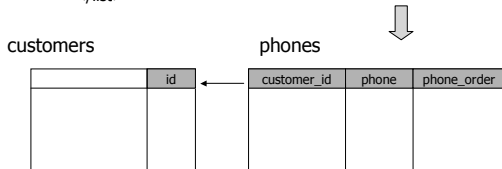
Map Set of Simple Types

```
<set name="phones" table="phones" lazy="true">
  <key column="customer_id"/>
  <element type="string" column="phone"/>
</set>
```



Map List of Simple Types

```
<list name="phones" table="phones" lazy="true">
  <key column="customer_id"/>
  <index column="phone_order"/>
  <element type="string" column="phone"/>
</list>
```

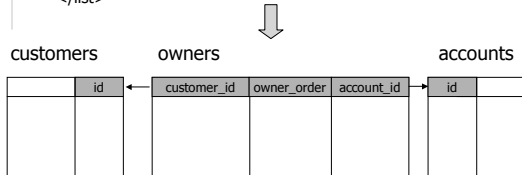


Collection of Object Types

```
public class Account {
    Integer id;
    BigDecimal balance;
    Date createdOn;
    List<Customer> owners;
}
```

Map List of Object Types

```
<list name="owners" table="owners" lazy="true">
  <key column="account_id"/>
  <index column="owner_order"/>
  <many-to-many class="Customer" column="customer_id"/>
</list>
```



Inheritance

```
public class CDAccount extends Account {
    Integer term;
}
```

◆ When do we want to create subclasses??

Map Subclass – Table Per Concrete Class

accounts

id	balance	created_on
----	---------	------------

cd_accounts

id	balance	created_on	term
----	---------	------------	------

Map Subclasses – Table Per Subclass

```
<joined-subclass name="CDAccount" table="cd_accounts">
  <key column="account_id"/>
  <property name="term"/>
</joined-subclass>
```



cd_accounts

id	term
----	------

accounts

id	balance	created_on
----	---------	------------

Map Subclasses – Table Per Hierarchy

```
<discriminator column="account_type" type="string"/>
<subclass name="CDAccount" discriminator-value="CD">
  <property name="term"/>
</subclass>
```



accounts

id	balance	created_on	term
----	---------	------------	------

Components

```
public class Address {
    String street, city, state, zip;
}

public class User {
    Integer id;
    String username, password;
    Address address;
}
```

Map Components

```
<component name="address" class="Address">
  <property name="street"/>
  <property name="city"/>
  <property name="state"/>
  <property name="zip"/>
</component>
```



users

id	...	street	city	state	zip	...
----	-----	--------	------	-------	-----	-----

Components Inside Collection

```
<list name="history" table="bibtex_history" lazy="true">
  <key column="bibtex_id" />
  <index column="bibtex_order" />
  <composite-element class="BibtexEntry">
    <property name="content" />
    <many-to-one name="editor" class="User" />
    <property name="lastModified" column="last_modified" />
  </composite-element>
</list>
```


Somewhat Unusual Mappings

- ◆ Bidirectional Association
 - Accounts and Owners
 - Item vs. Reviews, Ratings, and Tags
- ◆ Multi-way relationship
 - Tag

O/R Mapping vs. ER-Relational Conversion

<u>O/R Mapping</u>		<u>ER-Relational Conversion</u>
Class	↔	Entity Set
<property>	↔	Attribute
Association	↔	Relationship
Subclass		Subclass
• table per concrete class	↔	• OO method
• table per class hierarchy	↔	• NULL method
• table per subclass	↔	• ER method

Things We'll Talk Later (Or Not)

- ◆ Fine tune the schema
 - not-null, unique etc.
- ◆ Performance-related issues
 - Lazy-loading
- ◆ More about queries
 - Criteria queries
 - Native SQL queries

Conclusion?

- ◆ What does hibernate give us??

More Hibernate Resource

- ◆ *Hibernate in Action* by Christian Bauer and Gavin King
- ◆ Hibernate documentation at <http://www.hibernate.org>
 - Chapter 6-10
- ◆ DTDs at <http://sun.calstatela.edu/~cysun/documentation/DTDs/>

More Readings

- ◆ *Database Systems – The Complete Book* by Garcia-Molina, Ullman, and Widom
 - Chapter 2: ER Model
 - Chapter 3.2-3.3: ER to Relational Conversion
 - Chapter 4.1-4.4: OO Concepts in Databases
 - Chapter 9: OQL
 - Chapter 8.7: Transactions