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)
- 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 ) {
    …
    PreparedStatment p;
    p = connection.prepareStatment(
        "select * from employees where id = ?"
    );
    …
}
```

… Problems with CLI and Embedded SQL …

◈ Tedious translation between application objects and database tables

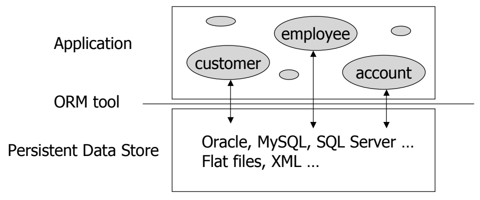
```
public Employee( Integer id ) {
    …
    ResultSet rs = p.executeQuery();
    if( rs.next() )
    {
        name = rs.getString("name");
        …
    }
}
```

… 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



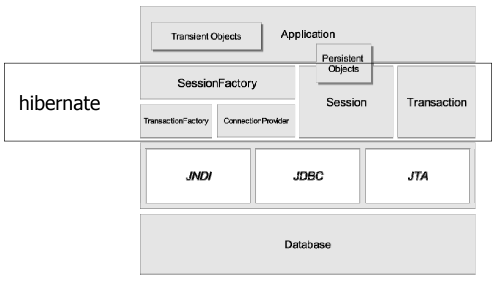
Advantages of ORM

- 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



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 has an identity field
  - Each persistent class implements the Serializable interface
  - Each persistent field has a pair of getter and setter, *which don't have to be public*

O/R Mapping Files

- Describe how class fields are mapped to table columns
- Three important types of elements in a 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 come with the downloaded Hibernate package

Access Persistent Objects

- `Session`
- `Query`
- `Transaction`
  - A transaction is required for updates
- http://www.hibernate.org/hib_docs/v3/api/org/hibernate/package-summary.html

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
  - *Insert()??*
    - *Save or update??*
  - Turn on `show_sql`
    - Caching and Isolation Levels

Caching in Hibernate

- Object cache
  - Caching Java objects
  - Simple and effective implementation
    - Hash objects using identifiers as key
- Query cache
  - Caching query results
  - No implementation that is both simple and effective

Cache Scopes

- Session
- Process
- Cluster

First-Level Cache

- Session scope
- Always on (and cannot be turned off)
- Ensure that there are no duplicate/inconsistent objects in the same session

Second-Level Cache

- Pluggable *Cache Providers*
  - Process cache
    - E.g. `EHCache`, `OSCache`
  - Cluster cache
    - E.g. `SwarmCache`, `JBossCache`
- Distinguished by
  - Cache scope
  - Concurrency policies

Isolation Example …

Sells

bar	beer	price
Joe's	Bud	2.50
Joe's	Miller	2.75
Sue's	Bud	2.50
Sue's	Miller	3.00

- Sue is querying `Sells` for the highest and lowest price Joe charges.
- Joe decides to stop selling Bud and Miller, but to sell only Heineken at $3.50

… Isolation Example

Sue's transaction:

```
-- MAX
SELECT MAX(price) FROM Sells WHERE bar='Joe''s';
-- MIN
SELECT MIN(price) FROM Sells WHERE bar='Joe''s';
COMMIT;
```

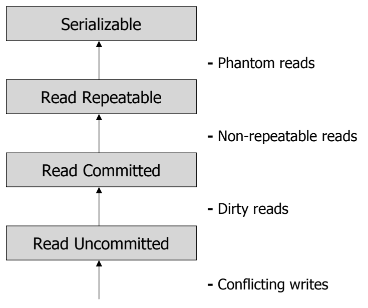
Joe's transaction:

```
-- DEL
DELETE FROM Sells WHERE bar='Joe''s';
-- INS
INSERT INTO Sells VALUES( 'Joe''s', 'Heineken', 3.50 );
COMMIT;
```

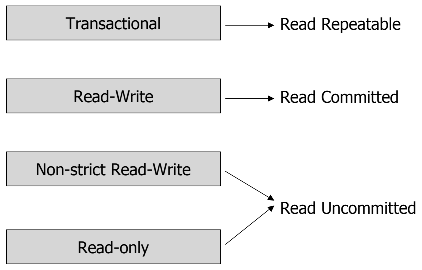
Potential Problems of Concurrent Transactions

- Caused by *interleaving operations*
- Caused by *aborted operations*
- For example:
  - MAX, DEL, MIN, INS
  - MAX, DEL, INS, MIN

Transaction Isolation Levels



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/hibernate-examples.ddl` – generated automatically by hbm2ddl

More About Mapping

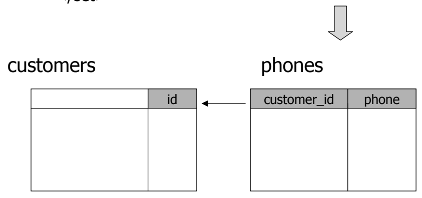
- Basic mapping
  - <id>
  - <property>
  - Association
    - many-to-one
    - one-to-many
    - one-to-one
    - many-to-many
- Collections
- Subclasses
- Components
- Other
  - Bidirectional association

Collection of Simple Types

```
public class Customer {

        Integer id;

        String name;
        String address;

        Set<String> phones;

}
```

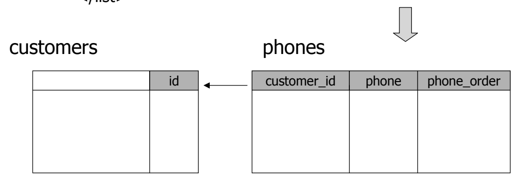
Map Set of Simple Types

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



Map List of Simple Types

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

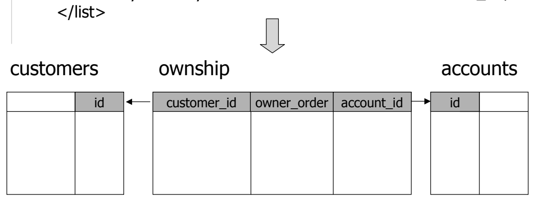


Collection of Object Types

```
public class Account {

        Integer id;

        Double balance;
        Date createdOn;

        List<Customer> owners;

}
```

Map List of Object Types

```
<list name="owners" table="ownship">
    <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;

}
```

Map Subclass – Table Per Concrete Class

accounts

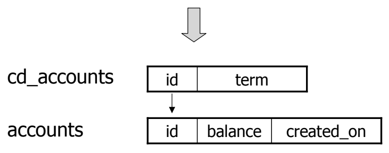
id	balance	created_on

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>
```



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">

    <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>
```

Bidirectional Association

```
public class Account {

        Integer id;

        Double balance;
        Date createdOn;

        List<Customer> owners;

}
```

```
public class Customer {

        Integer id;

        String name;
        String address;

        Set<String> phones;
        Set<Account> accounts;

}
```

Bidirectional Association Mapping

```
<class name="Customer" table="customers">

    …

    <set name="accounts" table="ownership" inverse="true">
        <key column="customer_id" />
        <many-to-many class="Account" column="account_id" />
    </set>

</class>
```

O/R Mapping vs. ER-Relational Conversion

O/R Mapping		ER-Relational Conversion
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

Tips for Hibernate Mapping

- Understand relational design
  - Know what the database schema should looks like before doing the mapping
- Understand OO design
  - Make sure the application design is object-oriented

OO Design and Hibernate Mapping Example

- User (student or instructor)
- A class section has a group of students and an instructor

Design #1

```
class User {                class Section {

        Integer id;                 Integer instructor;
        String name;                Set<User> students;
}                           }
```

Does the design pass the "English test"??

Design #2

```
class User {                class Section {

        Integer id;                 User instructor;
        String name;                Set<User> students;
}                           }
```

Does the design pass the "English test"??

Design #3

```
class User {                class Section {

        Integer id;                 User instructor;
        String name;                List<User> students;
}                           }
```

Lists or sets??

Design #4

```
class User {

        Integer id;
        String name;
        Set<Section> sectionsTakenOrTaught;
}

class Section {

        User instructor;
        Set<User> students;
}
```

Uni-direction or bi-direction association??

Lazy Loading

- Hibernate is "lazy" by default
  - `Account` -> `Customers` -> `Phones`
- But sometimes we want to be "eager"
  - Performance optimization, i.e. reduce the number of query requests
  - Disconnected clients

Fetch Strategies

- Select and subselect
- Batch size
- Join fetch

```
from Account a left join fetch a.owners
```

Hibernate Support in Spring

- HibernateTemplate
  - http://www.springframework.org/docs/api/org/springframework/orm/hibernate/HibernateTemplate.html
- CSNS source code under `src/csns/model/dao/hibernate`
- And much more (covered later in the lectures on Spring)

The Spring Advantage

Without Spring

```
Transaction tx = null;
try
{
   tx = s.beginTransaction();
   s.saveOrUpdate( e );
   tx.commit();
}
catch( Exception e )
{
   if( tx != null ) tx.rollback();
   e.printStackTrace();
}
```

With Spring

```
getHibernateTemplate()
.saveOrUpdate( user );
```

Hibernate Projects ...

- http://www.hibernate.org/
- Hibernate Core
- Hibernate Annotations
  - Use Java Annotations instead of XML to specify data mapping
- Hibernate EntityManager
  - For EJB
- Hibernate Shards
  - For using multiple databases at the same time

... Hibernate Projects

- Hibernate Validator
  - Enforces database integrity constraints both in database and in Java code using annotation
- Hibernate Search
  - Integrate Hibernate with full text search engines like Lucene
- Hibernate Tools
  - Generate Java code from database schema, Eclipse plugins, additional Ant tasks etc.
- NHibernate (Hibernate for .NET)

Readings

- *Java Persistence with Hibernate* by Christian Bauer and Gavin King (or *Hibernate in Action* by the same authors)
- Hibernate Core reference at http://www.hibernate.org
  - Chapter 3-10, 14

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