CS5220 Advanced Topics in Web Programming
Object-Relational Mapping with Hibernate and JPA (I)

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
  - Most reliable (ACID)
  - Standardized access (SQL)
  - 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_id   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

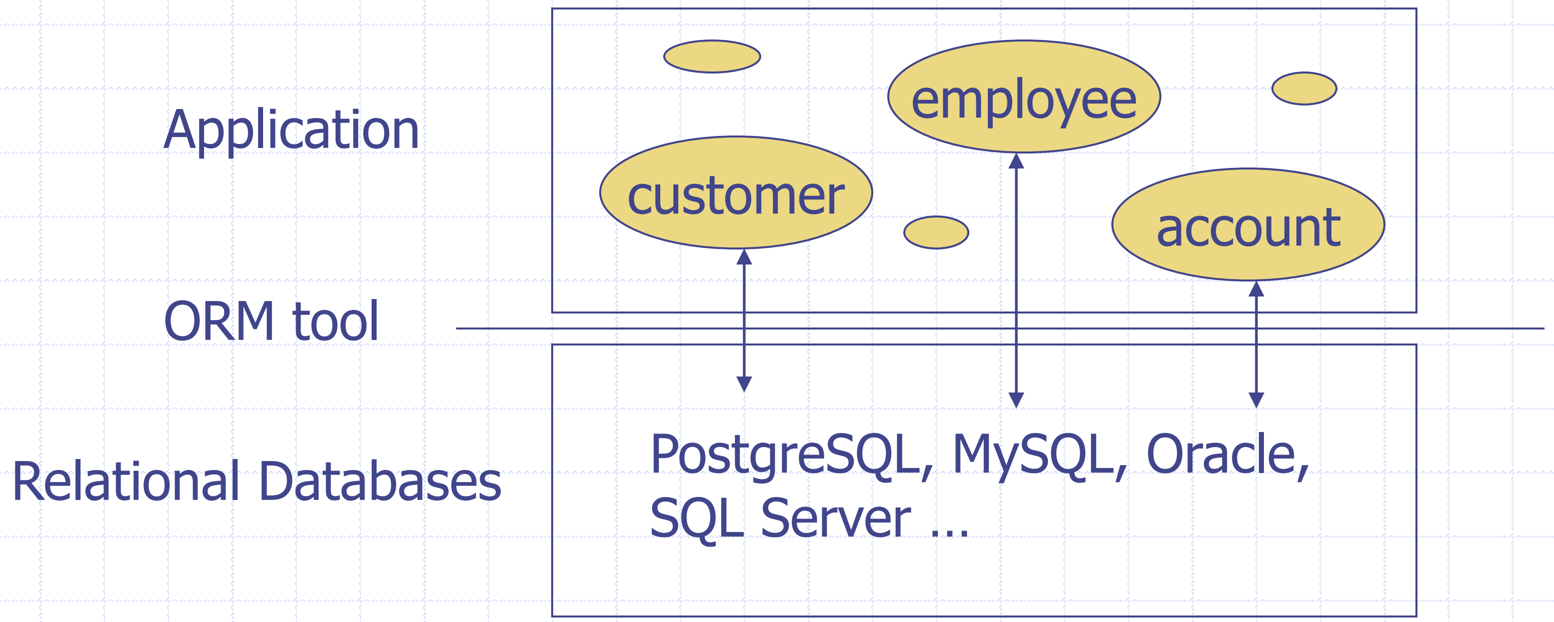
```
public Employee( Integer id ) {
    …
    ResultSet rs = p.executeQuery();
    if( rs.next() )
    {
        name = rs.getString("name");
        …
    }
}
```

… 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



Hibernate and JPA

- Java Persistence API (JPA)
  - Annotations for object-relational mapping
  - Data access API
  - An object-oriented query language JPQL
- Hibernate
  - The most popular Java ORM library
  - An implementation of JPA

Hibernate Usage

- Hibernate without JPA
  - API: `SessionFactory, Session, Query, Transaction`
  - More features
- Hibernate with JPA
  - API: `EntityManagerFactory, EntityManager, Query, Transaction`
  - Better portability
  - Behaviors are better defined and documented

A Hibernate Example

- Java class
  - `Employee.java`
- Code to access the database
  - `EmployeeTest.java`
- JPA configuration file
  - `persistence.xml`
- (Optional) Logging configuration file
  - `log4j.properties`

Persistent Class

- A class whose objects need to be saved (i.e. *persisted*) in a database
- Any Java model class can be a persistent class, though 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 Annotations

- Describe how Java classes are mapped to relational tables

@Entity	Persistent Java Class
@Id	Id field
@Basic (can be omitted)	Fields of simple types
@ManyToOne @OneToMany @ManyToMany @OneToOne	Fields of class types

persistence.xml

- <persistence-unit>
  - `name`
- <properties>
  - Database information
  - Provider-specific properties
- No need to specify persistent classes

Access Persistent Objects

- `EntityManagerFactory`
- **`EntityManager`**
- `Query` and `TypedQuery`
- `Transaction`
  - A transaction is required for updates

Some EntityManager Methods

- find(entityClass, primaryKey)
- merge(entity), persist(entity)
- getTransaction()
- createQuery(query, resultClass)

https://docs.jboss.org/hibernate/jpa/2.2/api/javax/persistence/EntityManager.html

States of Persistent Objects

```
entityManager = entityManagerFactory
     .createEntityManager();
```

```
Foo f1 = entityManager.find( Foo.class, 1 );
```

// f1 is a *managed* object

```
Foo f2 = new Foo();
```

// f2 is an *unmanaged* (i.e. new) object

```
entityManager.close();
```

// f1 become *detached*

ORM "magic" only works on managed objects

How merge() Works

```
Foo f2 = entityManager.merge(f1);
```

- If `f1` is a managed object, the returned `f2` is the same as `f1`
- If `f1` is an unmanaged or detached object, the returned `f2` is a *managed* object which is a *copy* of `f1`

Java Persistence Query Language (JPQL)

- A query language that looks like SQL, but for accessing *objects*
- Automatically translated to DB-specific SQL statements
- E.g. `select e from Employee e where supervisor = :supervisor`

See Chapter 4 of Java Persistence API, Version 2.1

Hibernate Query Language (HQL)

- A superset of JPQL
- http://docs.jboss.org/hibernate/orm/current/userguide/html_single/Hibernate_User_Guide.html#hql
- See DaoImpl code in CSNS2 for more examples

Benefits of ORM

- Remove the mismatch between OO design in application and relational design in database
- Simplify data access
  - Data is accessed as objects, i.e. no manual conversion between objects and rows/columns necessary
  - JPQL/HQL queries are usually simpler than SQL queries
  - Often times queries are automatically generated by the ORM tool, e.g. `e.getSupervisor().getSupervisor().getName()`
- Improve DBMS independency
- Object caching