CSE 530A
Lab 3
Washington University
Fall 2013
Table Definitions

• The table definitions for lab 3 are slightly different from those for lab 2
  – Serial ID columns have been added to all of the tables

Lab 2:
CREATE TABLE books (  
isbn TEXT PRIMARY KEY,  
title TEXT,  
author TEXT,  
price INTEGER
);

Lab 3:
CREATE TABLE books (  
book_id BIGSERIAL PRIMARY KEY,  
isbn TEXT NOT NULL,  
title TEXT NOT NULL,  
author TEXT NOT NULL,  
price INTEGER NOT NULL
);
Lab 2:
CREATE TABLE shopping_carts (  
  user_id BIGINT REFERENCES users(user_id) PRIMARY KEY,  
  modification_timestamp TIMESTAMP WITHOUT TIME ZONE
);

Lab 3:
CREATE TABLE shopping_carts (  
  cart_id BIGSERIAL PRIMARY KEY,  
  modification_timestamp TIMESTAMP WITHOUT TIME ZONE NOT NULL
);
Table Definitions

Lab 2:
CREATE TABLE users (  
   user_id BIGSERIAL PRIMARY KEY,  
   username TEXT UNIQUE,  
   password TEXT  
);

Lab 3:
CREATE TABLE users (  
   user_id BIGSERIAL PRIMARY KEY,  
   username TEXT UNIQUE NOT NULL,  
   password TEXT NOT NULL,  
   cart_id BIGINT REFERENCES shopping_carts(cart_id)  
);
Table Definitions

Lab 2:
CREATE TABLE shopping_cart_items (  
cart_id BIGINT REFERENCES shopping_carts(user_id),  
isbn TEXT REFERENCES books(isbn),  
quantity INTEGER,  
CONSTRAINT shopping_cart_items_key  
PRIMARY KEY (cart_id, isbn)  
);  

Lab 3:
CREATE TABLE shopping_cart_items (  
item_id BIGSERIAL PRIMARY KEY,  
cart_id BIGINT REFERENCES shopping_carts(cart_id) NOT NULL,  
book_id BIGINT REFERENCES books(book_id) NOT NULL,  
quantity INTEGER NOT NULL  
);
Entity Relationships

• The User class now has a reference to ShoppingCart
  – What is the type of the relationship between User and ShoppingCart?

```java
public class User implements Serializable {
    ...

    private ShoppingCart shoppingCart;
    ...

    public ShoppingCart getShoppingCart() {
        return shoppingCart;
    }
    ...
```
Entity Relationships

• User and ShoppingCart have a one-to-one relationship (Why?)

• Note that the ShoppingCart class does not have a reference to User
  – So the relationship is unidirectional

• What annotations need to be put on the ShoppingCart getter in User?
Entity Relationships

• Since the users table has a foreign key column for cart_id, we use a @JoinColumn annotation, specifying the column
  – What if shopping_carts had a foreign key column for user_id instead?

```java
@OneToOne
@JoinColumn(name = "cart_id")
public ShoppingCart getShoppingCart() {
    return shoppingCart;
}
```
Entity Relationships

- What if shopping_carts had a foreign key column for user_id (and ShoppingCart had a User field) instead?

```java
@OneToOne(mappedBy = "user")
public ShoppingCart getShoppingCart() {
    return shoppingCart;
}
```
Entity Relationships

• What if we wanted the relationship to be bidirectional?
  – That is, User has a ShoppingCart field and ShoppingCart has a User field

In User.java:

```java
@OneToOne
@JoinColumn(name = "cart_id")
public ShoppingCart getShoppingCart() {
    return shoppingCart;
}
```

In ShoppingCart.java:

```java
@OneToOne(mappedBy = "shoppingCart")
public User getUser() {
    return user;
}
```
Entity Relationships

• How do we attach a new ShoppingCart to a User (assuming a unidirectional relationship)?
Entity Relationships

• How do we attach a new ShoppingCart to a User (assuming a unidirectional relationship)?
  – Create a new ShoppingCart object
  – Persist (save) the ShoppingCart object
    • This will cause a id for the ShoppingCart to be generated
  – Get the User object
  – Set the shoppingCart field on the User object
  – Save the User object
    • This will persist the changes to the User object
Entity Relationships

• What if the User-ShoppingCart relationship is bidirectional?
  – Get the User object
  – Create the new ShoppingCart object
  – Set the user field on the ShoppingCart object
  – Save the ShoppingCart object

• Note that this does not set the shoppingCart field on the existing User object. However, if we get the User object from the database again then it will be set.
Entity Relationships

• What is the relationship between ShoppingCart and ShoppingCartItem?
Entity Relationships

• What is the relationship between ShoppingCart and ShoppingCartItem?
  – One-to-many
    • Each ShoppingCart can have many ShoppingCartItems but each ShoppingCartItem is related to one ShoppingCart
  – Bidirectional
    • As defined in the provided code, the relationship is bidirectional
      – Design decision
    • So ShoppingCart needs a @OneToMany annotation for items and ShoppingCartItem needs a @ManyToOne annotation for shoppingCart
Entity Relationships

- What is the relationship between ShoppingCartItem and Book?
Entity Relationships

• What is the relationship between ShoppingCartItem and Book?
  – Many-to-one
    • Multiple ShoppingCartItems can reference the same Book, but only one Book each
  – Unidirectional
    • ShoppingCartItem has a book field but Book does not have a List<ShoppingCartItem> field
      – Design decision
  – So ShoppingCartItem needs a @ManyToOne annotation for book
Transient Fields

• The Book class has a getter that does not correspond to a persistent field
  – getFormattedPrice()
  – Such getters (or fields) must have the @Transient annotation to avoid errors

```java
@Transient
public String getFormattedPrice() {
    ...
}
```
Entity Lifetime

• As discussed before, entity objects are only valid while in a Hibernate session
  – When the session is closed, the entity objects are invalid
  – This means we cannot store the User object in the session context like we did in lab 2

• Note that we are talking about two different "sessions" here
  – Hibernate Session
  – HttpSession
Entity Lifetime

- HttpSession
  - Created when a browser connects to our web app
  - Exists until explicitly invalidated (see the LogoutServlet class) or a timeout occurs
    - A timeout occurs if the browser does not send another request for some time
      - Configurable via tomcat setup
  - Can store things in the HttpSession that we want to exist across multiple requests
Entity Lifetime

• Hibernate Session
  – Generally use a session-per-request pattern
    • Wraps a database connection
  – Created at the beginning of request processing, closed at the end
  – Should not be shared by multiple requests
  – Entity objects are only valid while a Hibernate Session is open
Entity Lifetime

• Entity objects cannot be stored in the HttpSession
  – Objects would be invalid when accessed in subsequent requests
  • There actually is a way to detach an Entity object from a persistence session and reattach to another persistence session
    – Need to be careful about changes and conflicts
    – Should generally be avoided
Entity Lifetime

• Since we can't store the User object in the HttpSession in lab 3, we're storing the user ID instead
  – This means we need to re-fetch the User object from the database in subsequent requests
Entity Lifetime

• So the first part of all the Servlets follows this pattern

```java
private void process(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
    SessionFactory sessionFactory = (SessionFactory) request.getServletContext().getAttribute("hibernateSessionFactory");
    Session session = sessionFactory.openSession();
    Transaction tx = null;

    try {
        tx = session.beginTransaction();

        Long userId = (Long) request.getSession().getAttribute("userId");
        if (userId == null) {
            // not properly logged in
            request.getRequestDispatcher("WEB-INF/jsp/Login.jsp").forward(request, response);
            return;
        }

        User user = UserDao.retrieveUser(session, userId);
        if (user == null) {
            // not properly logged in
            request.getRequestDispatcher("WEB-INF/jsp/Login.jsp").forward(request, response);
            return;
        }
    }

    request.getRequestDispatcher("WEB-INF/jsp/Login.jsp").forward(request, response);
}
```