



ORM Concurrency

Enterprise Applications

- We want to create Enterprise Applications with Spring and Hibernate
 - The vast majority of these also use a web container
- Let's first review Java Web Containers and web dev.
 - And relate it to what we've learned so far

Web Containers

- We'll first discuss :
 - Web Container IoC
 - Servlets (beans)
 - Filters (interceptors)
- Then we'll look at JSP for view:
 - JSPs are all XML
 - JSTL tags for program control
 - EL inside statements and to print

Containers so Far

- We saw that Spring is a container:
 - IoC (creates objects)
 - DI (connects them)
 - AOP (proxies for extra functionality, discuss later)
- Hibernate is also a container:
 - Creates Objects (IoC)
 - Connects objects based on associations (DI)
 - Proxies to provide lazy loading (AOP)

Web Container

- We will see that a web container:
 - Creates Objects (IoC)
 - Can add proxies for extra functionality (Filters)
 - Does not connect objects together (no DI)
- Main difference, web containers work with:
 - Incoming Request objects
 - Outgoing Response objects

Not POJOs

- Another big difference is that the objects managed by web containers are not POJOs
 - To be a Servlet or Filter you have to extend or implement a Technology related class / interface
 - Web containers design is old
 - Before Rod Johnson's book about POJO containers

Comparing Terminology

- Servlet:
 - Object that the container creates and manages
 - What Spring called a Bean
- Web.xml
 - Configuration file that configures the container
 - What spring called applicationContext.xml
- Filter
 - Proxy for a Servlet
 - Somewhat similar to Interceptor in Spring MVC

web.xml

Inside project's
/WEB-INF/

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.4" xmlns="http://java.sun.com/xml/ns/j2ee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
http://java.sun.com/xml/ns/j2ee/web-app 2 4.xsd">

    <servlet>
        <servlet-name>Servlet Demo</servlet-name>
        <servlet-class>demo.ServletDemo</servlet-class>
    </servlet>

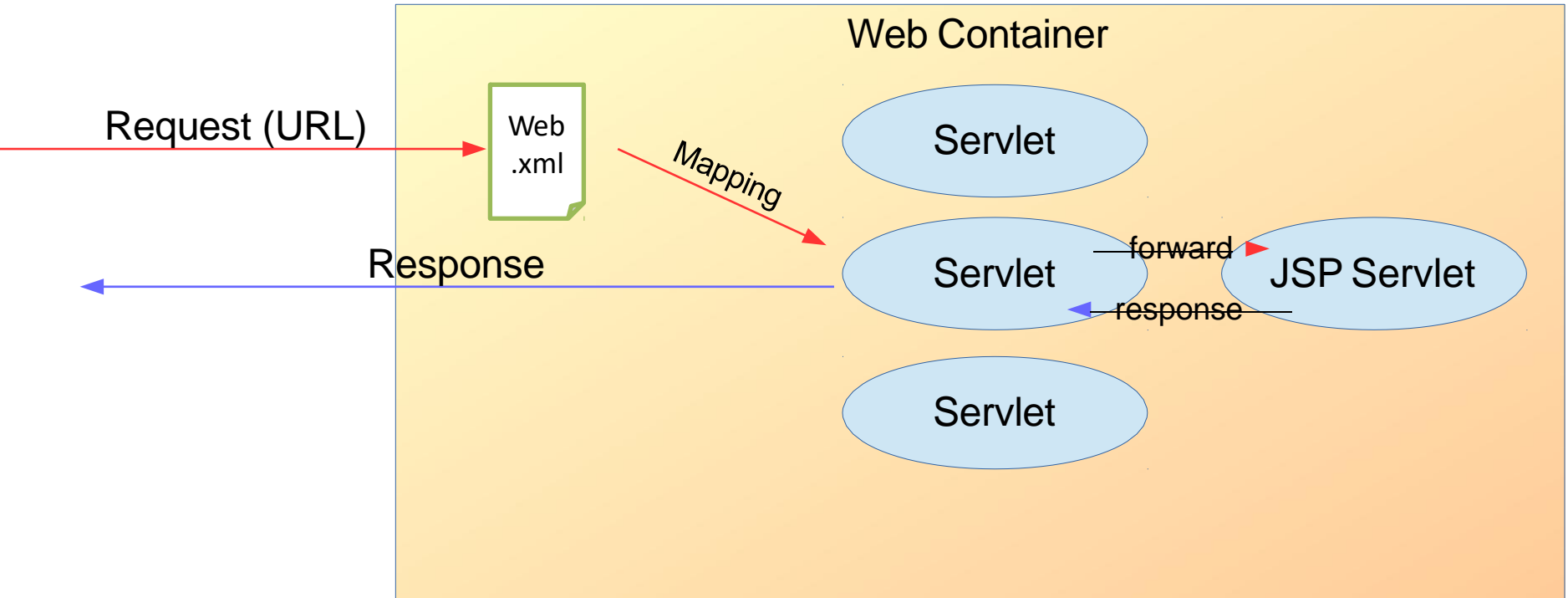
    <servlet-mapping>
        <servlet-name>Servlet Demo</servlet-name>
        <url-pattern>/servlet</url-pattern>
    </servlet-mapping>
</web-app>
```

Can also be done
@WebServlet

Request / Response

- **The container receives a request for a URL**
 - Looks at Servlet-Mappings to find a matching pattern
 - Passes request and a empty response to servlet
 - Request may contain additional key/value params
 - Servlet reads request, and fills in response
 - Optionally forwarding req/resp to other servlet for more
 - Response (text output) then printed to user

Visually



Requests

- HTTP (web) Requests have a type:
 - GET or POST for HTML
- May have key / value pair parameters:
 - GET in URL, POST as 'post data'
- Often also a Session ID cookie
 - Allows the server to find storage for this user

Servlet

```
@WebServlet(name = "Hello", urlPatterns = { "/Hello" })  
public class Hello extends HttpServlet {  
    private static final long serialVersionUID = 1L;
```

Not a POJO
Extends HttpServlet

```
    @Override  
    protected void doGet(HttpServletRequest request, HttpServletResponse response)  
        throws ServletException, IOException {
```

```
        request.setAttribute("now", new Date());  
        request.setAttribute("one", 1);  
        request.setAttribute("two", 2);
```

Method on HttpServlet
Lets you get container
(Context)

```
        ServletContext context = this.getServletContext();  
        String jsp = "/Hello.jsp";  
        RequestDispatcher dispatcher = context.getRequestDispatcher(jsp);  
        dispatcher.forward(request, response);
```

```
    }
```

```
}
```

Filter

```
@WebFilter(filterName = "OpenEntityManagerInView", urlPatterns = "/*")
public class EntityManagerInterceptor implements Filter {
    @Override
    public void destroy() { }
    @Override
    public void init(FilterConfig fc) throws ServletException { }
    @Override
    public void doFilter(ServletRequest req, ServletResponse res, FilterChain chain)
        throws IOException, ServletException {

        EntityManager em = EntityManagerHelper.getCurrent();
        try {
            em.getTransaction().begin();
            chain.doFilter(req, res);
            em.getTransaction().commit();
        } catch (RuntimeException e) {

            if (em != null && em.isOpen())
                em.getTransaction().rollback();

            throw e;

        } finally {
            em.close();
        }
    }
}
```

Not a POJO
Implements Filter

Web.xml for Filter

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.4" xmlns="http://java.sun.com/xml/ns/j2ee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">

...

<filter>
  <filter-name>OpenEntityManagerInView</filter-name>
  <filter-class>example.filter.OpenEntityManagerInView</filter-class>
</filter>

<filter-mapping>
  <filter-name>OpenEntityManagerInView</filter-name>
  <url-pattern>/*</url-pattern>
</filter-mapping>

...

</web-app>
```

Can also be done with
@WebFilter

Applications



CS544 EA

Web: JSP with EL & JSTL

View Technology

- JSP is a type of Servlet written in XML
 - Java's oldest web based view technology
- Write logic with XML tags (for loops etc)
 - JSTL: Java Standard Tag Library

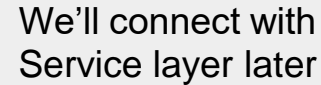
Pre-compiler transforms
XML into Java

Originally JSP had you write Java code inside HTML
This is now considered a bad practice. Deprecated!

Model View Control

- **Best practice:**

- Controller receives request
- Gets data from Service layer
- Forwards data to (JSP) View



We'll connect with
Service layer later

Servlet Controller

```
@WebServlet(name = "Hello", urlPatterns = { "/Hello" })
public class Hello extends HttpServlet {
    private static final long serialVersionUID = 1L;

    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {

        request.setAttribute("now", new Date());
        request.setAttribute("one", 1);
        request.setAttribute("two", 2);

        ServletContext context = this.getServletContext();
        String jsp = "/Hello.jsp";
        RequestDispatcher dispatcher = context.getRequestDispatcher(jsp);
        dispatcher.forward(request, response);

    }
}
```

Instead of retrieving data we'll just create some here and store inside request

Then forward to Hello.jsp

Hello.jsp with JSTL and EL

```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
```

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
```

```
<!DOCTYPE html>
```

```
<html>
```

```
  <head>
```

```
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
```

```
    <title>JSP Page</title>
```

```
  </head>
```

```
  <body>
```

```
    <h1>Hello JSP</h1>
```

```
    ${now}
```

EL can be used to print
or used in statement tag

```
  <div>
```

```
    <c:if test="${two gt one}">
```

```
      ${two}
```

Uses **now**, **one**, **two**
stored in request

```
    </c:if>
```

```
  </div>
```

<c:if> to create if
statement

```
  </body>
```

```
</html>
```

Browser shows:

Hello JSP

Tue Apr 30 12:52:13 CEST 2019
2

Expression Language

- `${expression}`
- `${object.value}`

Uses `.getValue()`
method

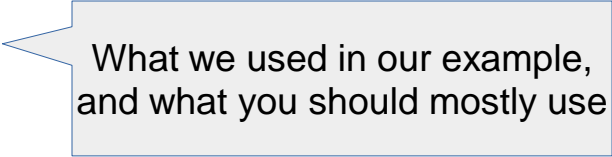
Easier for
non-Java people?

- Supported Operators

- Arithmetic: `+`, `-`, `*`, `/` (div), `%` (mod)
- Relational: `==` (eq), `!=` (ne), `<` (lt), `>` (gt), `<=` (lte), `>=` (gte)
- Logical: `&&` (and), `||` (or), `!` (not)
- Other: `()`, `empty`, `[]`

EL Scopes

- When using a variable in EL it uses the name as a key to find a value in the following scopes:
 - Page scope
 - Request scope
 - Session scope
 - Application scope



What we used in our example,
and what you should mostly use

First to Last



JSTL

- Example JSTL core tags:

Other libs can
also be loaded

```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
```

- Import the core tag lib with the prefix “c”

```
<c:if test="${two gt one}"> ... </c:if>
```

- Use the <c:if> tag to make an if statement

```
<c:forEach items="${customers}" var="customer"> ... </c:forEach>
```

- Use the <c:forEach> tag to make a loop

Tag	Description
<c:out>	Output result of evaluating an expression
<c:set>	Sets a key in one of the scopes
<c:remove>	Removes a key in one of the scopes
<c:catch>	Catches any java.lang.Throwable inside it
<c:if>	Executes content if test is true
<c:choose>	For a multi condition if (if/else)
<c:when>	Specify one (or more) conditions inside a <c:choose>
<c:otherwise>	Specify what should happen if none of the <c:when> are true
<c:import>	Ability to import data from a url (does http request)
<c:forEach>	Basic Foreach
<c:forTokens>	Iterates over tokens separated by the supplied delimiter
<c:redirect>	Redirects to a URL (HTTP 3xx)
<c:url>	Encodes a URL with optional parameters
<c:param>	To provide parameters to the url and import tags

JSTL Core Tags

<c:param>

Formatting Tags

```
<%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt" %>
```

Tag	Description
<fmt:message>	Gets message from bundle (for internationalization)
<fmt:param>	To give parameter to <fmt:message>
<fmt:bundle>	Specify a resource bundle to use
<fmt:setLocale>	Sets the locale
<fmt:requestEncoding>	Sets the request's character encoding
<fmt:timeZone>	Specifies the timezone for formatting
<fmt:setTimeZone>	Stores the specified timezone into a variable
<fmt:formatNumber>	Format a number based on the set locale
<fmt:parseNumber>	Parses a string of a (number, percentage, currency)
<fmt:formatDate>	Formats dates and times in a locale sensitive way
<fmt:parseDate>	Parses string representations of times and dates

Java is Depreciated inside JSP

- There were several ways to write Java in JSP
 - Pure Java code inside JSP (scriptlets)
- And elements that let you use Java:
 - JSP Expressions: `<%= user.getName() %>`
 - JSP Declarations: `<%! int a = 5 %>`
- These are all deprecated (**don't use them**)

Summary

- The web container is another container
 - Servlets (beans) take request and make response
 - Filters (like AOP) can intercept request / response
- JSP is type of Servlet written in XML
 - Uses JSTL tags for (if / loop) statements
 - Uses EL expressions inside statements or to print

Applications



CS544 EA

Hibernate Web Applications

Hibernate Web Applications

- Hibernate is generally only used in combination with Spring or a J2EE Server
 - To provide a deeper understanding of how it's integrated into an application
 - We're first going to **manually provide** some of the things Hibernate needs to run

Single Entity Manager Factory

- The Entity Manager Factory should start **once**
 - **Only one** for the entire application
 - Starts when the app starts
 - Closes when the app closes
- Good way to do this:
 - Make a **singleton** for it

Entity Manager & DAOs

- Repositories (DAOs) need to be able to get the **‘current entityManager’**
 - If each DAO method makes it's own EntityManager
 - We need multiple per web request
 - Each EntityManager has:
 - Its own DB connection, transaction, entity cache
 - All of which should be used for multiple operations!

EntityManager per Operation Anti-Pattern

- Using a EntityManager per operation is so bad it's considered an Anti-Pattern
- Also known as: “SessionPerOperation” Anti-Pattern

Never write a DAO like this!

```
public class BadCustomerDao {  
    private EntityManagerFactory emf;  
    public CustomerDao() {  
        EntityManagerFactory emf = EMF.get();  
    }  
  
    public Customer load(Long id) {  
        EntityManager em = emf.createEntityManager();  
        Customer c = em.find(Customer.class, id);  
        em.close();  
        return c;  
    }  
  
    public void save(Customer c) {  
        EntityManager em = emf.createEntityManager();  
        em.persist(c);  
        em.close();  
    }  
  
    public void update(Customer c) {  
        EntityManager em = emf.createEntityManager();  
        em.merge(c);  
        em.close();  
    }  
}
```

Entity Manager per Request

- We want one Entity Manager per (web) Request
 - Create it in the controller and pass it around as param?
 - ☐ Messy solution
- **Store it in the current thread**
 - Available to every method running in the thread
 - Known as “ThreadLocal”

EntityManager Helper

```
public class EntityManagerHelper {  
    private static final EntityManagerFactory emf,  
    private static final ThreadLocal<EntityManager> threadLocal,  
  
    static {  
        emf = Persistence.createEntityManagerFactory("cs544");  
        threadLocal = new ThreadLocal<EntityManager>();  
    }  
  
    public static EntityManager getCurrent() {  
        EntityManager em = threadLocal.get();  
        if (em == null) {  
            em = emf.createEntityManager();  
            threadLocal.set(em);  
        }  
        return em;  
    }  
  
    public static void closeEntityManagerFactory() {  
        emf.close();  
    }  
}
```

EntityManagerHelper provides:

.Singleton EntityManagerFactory

.ThreadLocal<EntityManager>

.getCurrent() method that can be called from any method

Based on: <https://stackoverflow.com/questions/15071238/entitymanager-threadlocal-pattern-with-jpa-in-jse>



EntityManager per Request DAO

- DAO's become thin wrappers:
 - Gets current EntityManager
 - Calls method

```
public class CustomerDao {  
  
    public Customer load(Long id) {  
        EntityManager em = EntityManagerHelper.getCurrent();  
        return em.find(Customer.class, id);  
    }  
  
    public void save(Customer c) {  
        EntityManager em = EntityManagerHelper.getCurrent();  
        em.persist(c);  
    }  
  
    public void update(Customer c) {  
        EntityManager em = EntityManagerHelper.getCurrent();  
        em.merge(c);  
    }  
}
```

Transaction

- Each **service method** should be **one transaction**
- Many Thread Local implementations close the EntityManager when the transaction commits
 - Means that all managed objects become detached
 - And automatic loading of related objects no longer works

Service Method

- ▶ Before an object is returned from a Service method:
 - ▶ Load any related objects needed by the recipient
 - ▶ Either have the DAO load all object into EM cache with query
 - ▶ Or have the Service follow references to 'force lazy loading'

```
public class CustomerService {  
    ...  
    public Customer getCustomer(Long id) {  
        EntityManager em = EntityManagerHelper.getCurrent();  
        em.getTransaction().begin();  
        Customer c = customerDao.load(id);  
        // follow references to ensure related objects are loaded  
        c.getAddress().getCity();  
        c.getCreditCard().getAddress().getCity();  
        // Then commit (may close entity manager)  
        em.getTransaction().commit();  
        // and return the 'object structure'  
        return c;  
    }  
}
```

Starts and stops the Transaction

Makes sure related objects are loaded

Applications



CS544 EA

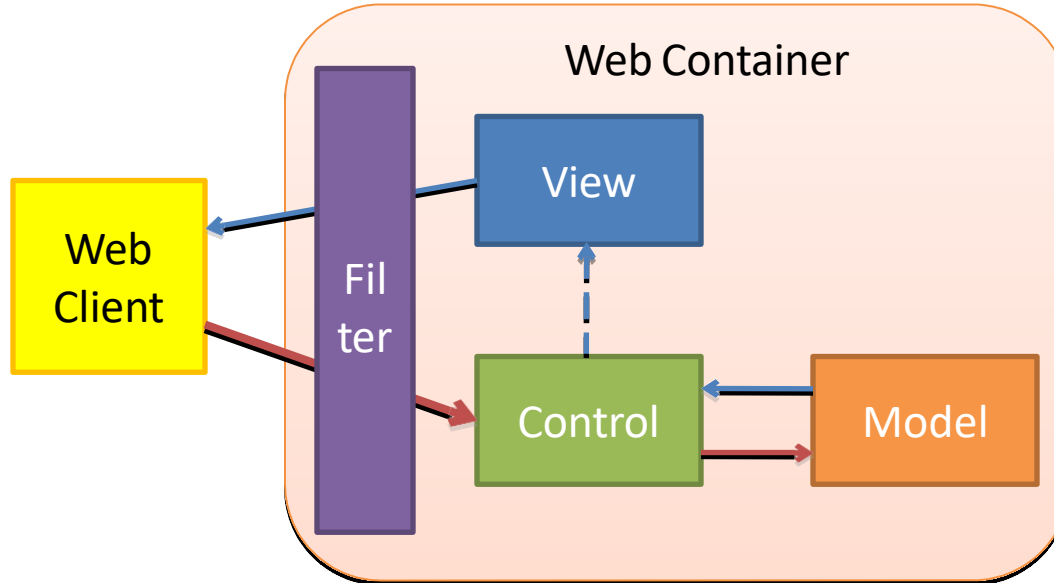
Hibernate Web: Open EntityManager in View

View

- Loading having all the entity objects loaded before they reach the view is a **best practice**
- Nevertheless, people find it frustrating that they cannot do lazy loading in the view
 - Solution: **OpenEntityManagerInView** pattern
 - Very popular / but some see it as an anti-pattern

OpenEntityManagerInView

- ▶ **Uses a filter** to open TX before controller
 - ▶ Closes TX after view is done



Open in View Filter

```
@WebFilter(filterName = "OpenEntityManagerInView", urlPatterns = "/*")
public class EntityManagerInterceptor implements Filter {
    @Override
    public void destroy() { }
    @Override
    public void init(FilterConfig fc) throws ServletException { }
    @Override
    public void doFilter(ServletRequest req, ServletResponse res, FilterChain chain)
        throws IOException, ServletException {

        EntityManager em = EntityManagerHelper.getCurrent();
        try {
            em.getTransaction().begin();
            chain.doFilter(req, res);
            em.getTransaction().commit();
        } catch (RuntimeException e) {

            if (em != null && em.isOpen())
                em.getTransaction().rollback();

            throw e;

        } finally {
            em.close();
        }
    }
}
```

Wraps all Servlets
(controller and view)
In one big transaction

Service Method

- **Just load first object**
 - Any related objects can be loaded by view

```
public class CustomerService {  
    ...  
    public Customer getCustomer(Long id) {  
        return customerDao.load(id);  
    }  
}
```

No longer has
Transaction code
(TX inside Filter)

No longer loads related
data (view can do so)

Criticism

No console app
or JavaFX app

- **Ties the Service Layer to Web Environment**
 - Only a web environment can lazy load (with filter)
- Can create Transaction that is too big
 - But each request usually calls only one service method
- Database activity happens during view
 - Some consider it a violation of layers
 - But it's not seen in code (just happens)

Hibernate App Summary

- The EMF should be a singleton
 - EM per operation is bad (for TX, cache, DB conn)
 - ThreadLocal used to give each request its own EM
- Load all related data before service method returns
 - Best practice, but there is a alternate option:
 - OpenEntityManagerInView filter so view can lazy load

Applications



CS544 EA

Concurrency

Concurrency

- Concurrent means **‘at the same time’**
 - Enterprise Apps need have many users (same time)
 - Concurrency problems center around Shared data
 - Data (state) only gets stored in DB
 - @Controller, @Service, and @Repository are stateless

Transactions

- The DB manages concurrency with Transactions
- A Transactions is a **unit of work** that is:
 - **ATOMIC**: The transaction is considered a single unit, either the entire transaction completes, or the entire transaction fails.
 - **CONSISTENT**: A transaction transforms the database from one consistent state to another consistent state
 - **ISOLATED**: Data inside a transaction can not be changed by another concurrent processes until the transaction has been committed
 - **DURABLE**: Once committed, the changes made by a transaction are persistent



Applications



CS544 EA

Concurrency: Isolation Level

Isolation Levels

- Proper isolation is expensive (takes lots of time) to produce in a multi-user environment
 - Isolation is often relaxed to increase DB speed
 - ANSI SQL defines 4 isolation levels

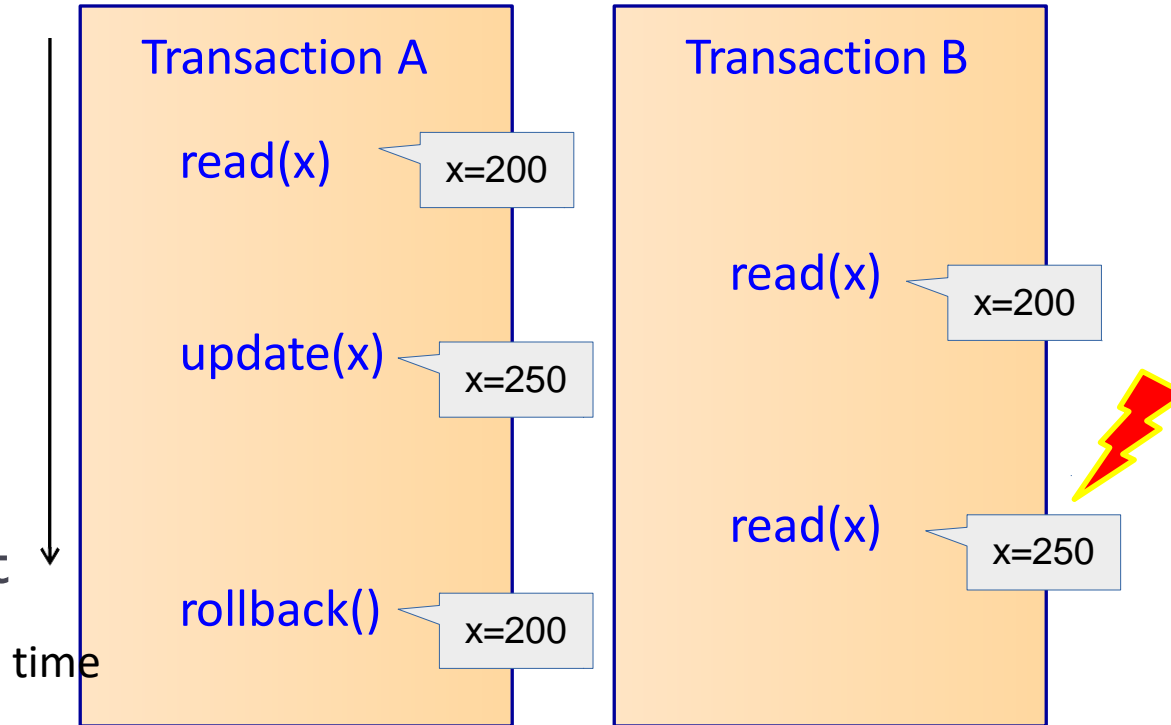
Read Uncommitted, Read Committed, Repeatable Read, Serializable

→
Weaker and Faster to Stronger and Slower

- Most Dbs default to Read Committed isolation
 - Only Serializable fully isolates a transaction from all concurrency issues

Read Uncommitted

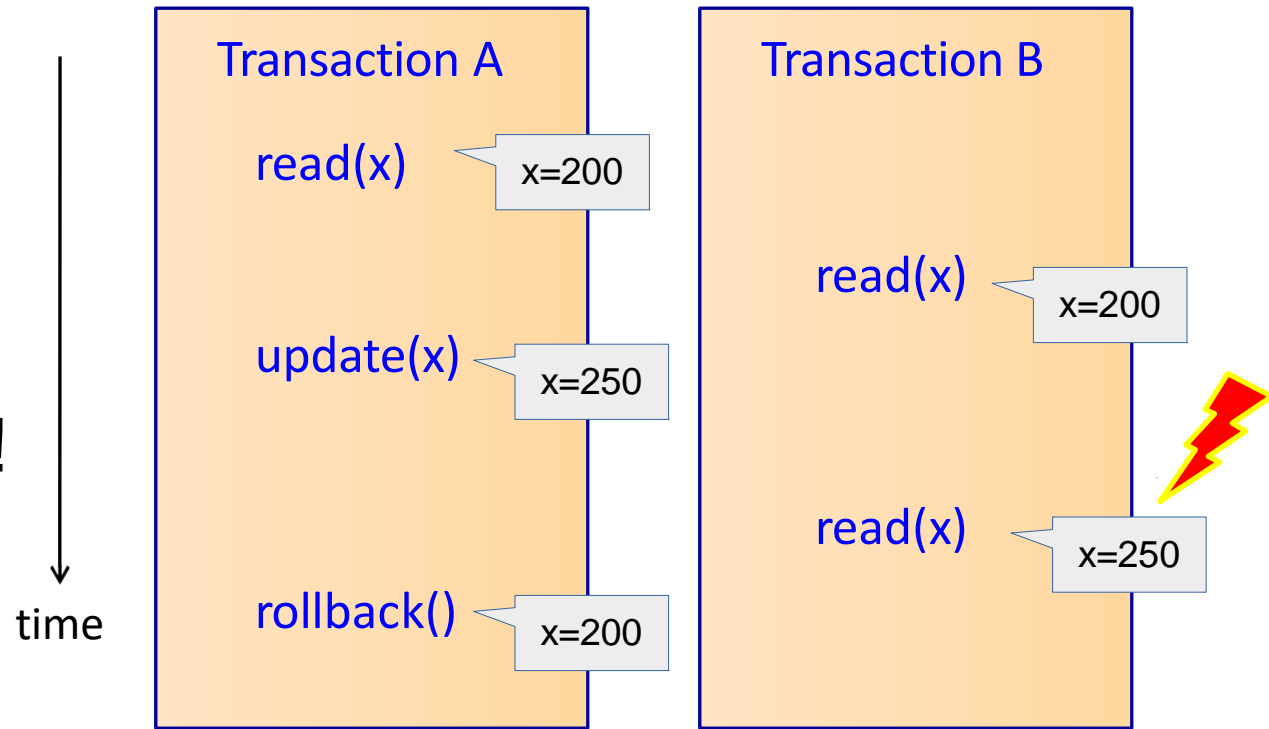
- ▶ TX A can read TX B's uncommitted updates
- ▶ No locks at all
- ▶ **Violates ACID**
- ▶ Not in Oracle
- ▶ Don't use in concurrent env!



Read Uncommitted

- TX A can read TX B's uncommitted updates

- No locks at all
- **Violates ACID**
- Not in Oracle
- Don't use in concurrent env!



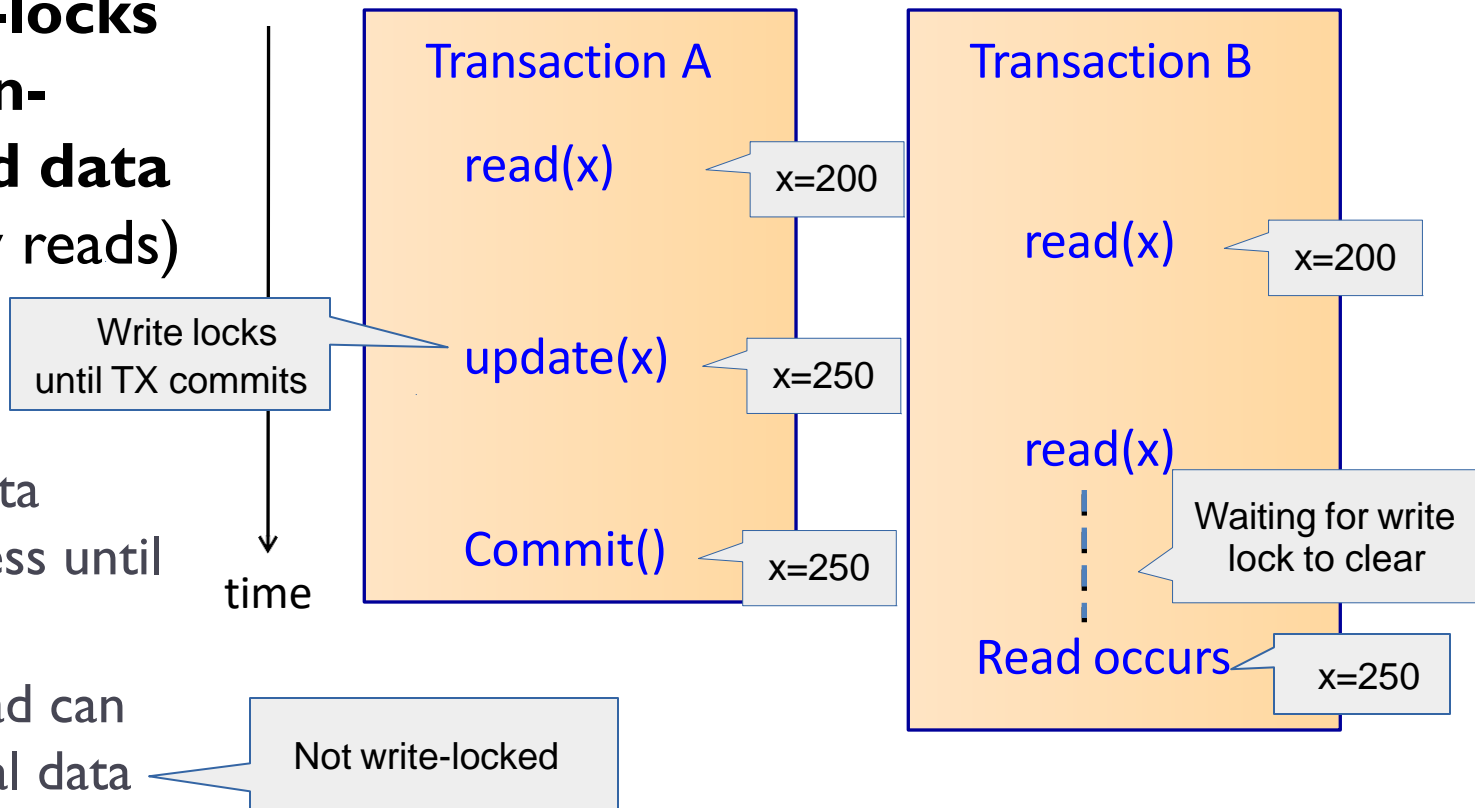
Concurrency Issues

- **Dirty reads:** a TX can read data that may never even get committed (useless)
- **Non-repeatable read:** a TX can read the same row twice and get two different values
- **Lost updates:** an update made by one TX silently disappears / overwritten (more on this later)
- **Phantom Read:** executing the same select twice may return more or less rows the second time

Read Committed

- ▶ Uses **write-locks** to hide non-committed data (solves dirty reads)

- ▶ Written data blocks access until commit
- ▶ Every thread can read normal data



Concurrency Issues

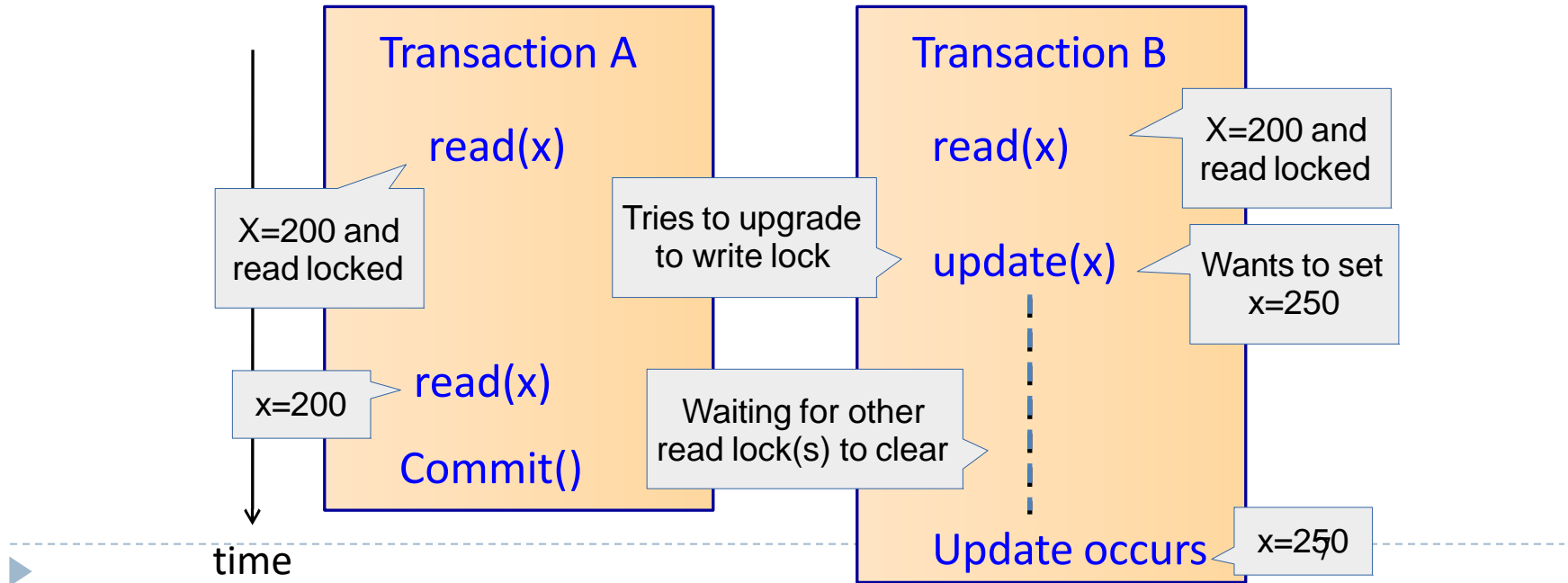
Not MySQL
it defaults to
Repeatable Read

- Read Committed is the **Default for many Dbs**
 - Write locks cause some delays, but not significant
 - **Speed more important** than fixing concurrency issues
- Do provide other ways of solving them:
 - Pessimistic locking (provided by most Dbs)
 - Optimistic concurrency (provided by JPA)

Not MySQL

Repeatable Read

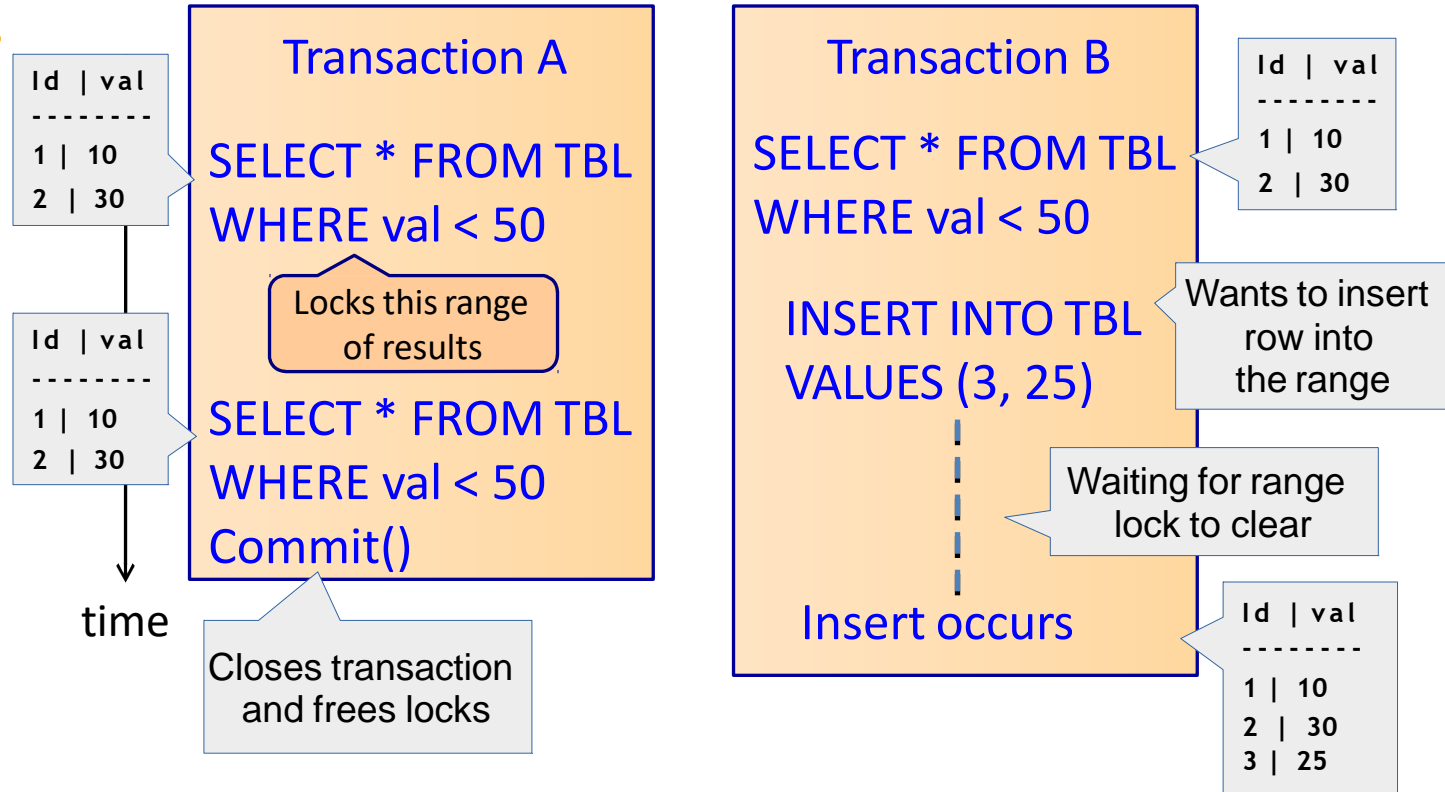
- ▶ **Uses read and write locks to solve** non-repeatable read and lost update problems
- ▶ Once read all future reads same value



Serializable

► Sets **range locks** to solve phantom read

- Lots of locks
- Slow
- Functionally similar to executing one after the other



Changing the default

- You can raise the default isolation level
 - **Everything will be slower**, less scalable
 - Even for transactions that don't need it

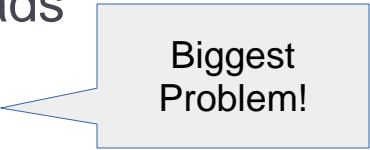
Not
Recommended

- Inside persistence.xml:

```
<property name="hibernate.connection.isolation" value="8" />
```

1 – Read Uncommitted
2 – Read Committed
4 – Repeatable Read
8 – Serializable

Using read-committed

- Because **speed is usually more important** most databases use read-committed
- This leaves DBs open to:
 - Non-repeatable reads
 - Phantom reads
 - Lost update Biggest Problem!
- We'll look at some ways to mitigate lost-update

Applications



CS544 EA

Concurrency: Optimistic Concurrency

Optimistic Concurrency

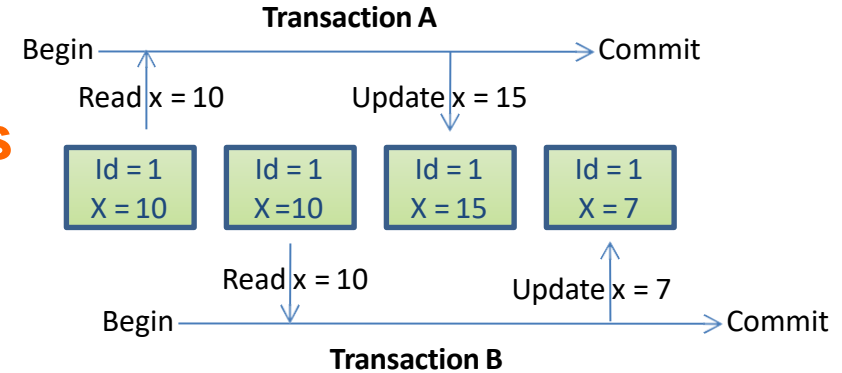
- Optimistic concurrency assumes that lost update conflicts **generally don't occur**
 - But keeps versions# so that it knows when they do
 - Uses read committed transaction level
 - Guarantees best performance and scalability
 - The default way to deal with concurrency



Lost Update Problem

▶ Read Committed Allows:

- ▶ **Last update to commit wins**
- ▶ First update lost



▶ Timeline:

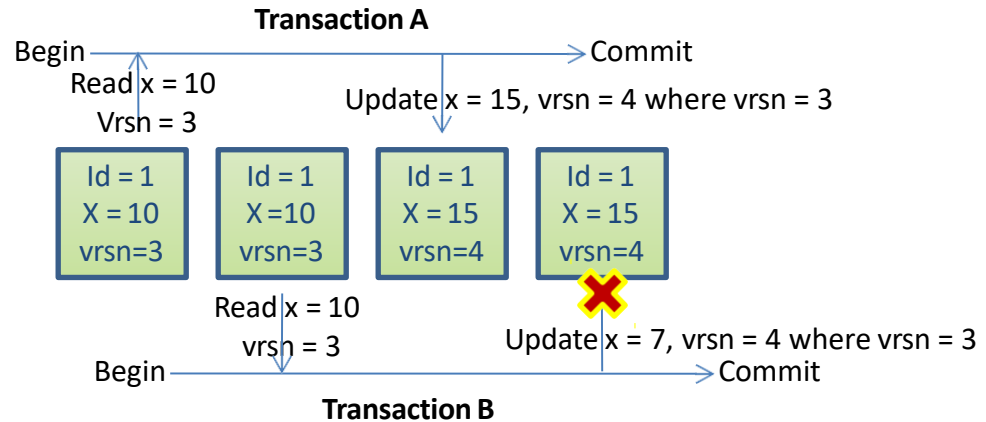
- ▶ Transactions A and B read $\text{id}=1, x=10$
- ▶ Transaction A changes x to $x=15$ (increment by 5)
- ▶ Transaction B wants to decrement by 3, sets $x=7$
- ▶ Neither A or B is aware that data was lost!

Believing
 $x = 10$



Versioning – First Update Wins

- ▶ Optimistic concurrency adds **a version column**
 - ▶ To track updates



▶ Last update fails

- ▶ UPDATE table SET $x = 15$ WHERE $id = 1$ AND $vrsn = 3$
- ▶ If other tx changed version, update does nothing
- ▶ JPA throws `OptimisticLockException` (in last TX)

OptimisticLockException

- When a version conflict occurs, JPA implementations throw a `OptimisticLockException`
 - Catching this exception allows you to **notify the user** about the conflict
 - The user can then reload the data and apply their updates against the latest data

Merging Conflicts

- ▶ If you have the time:
 - ▶ You can create a conflict merging page
 - ▶ Showing their the updates the other TX made
 - ▶ Showing the updates the user wanted to make
 - ▶ Allowing easy resolution
 - ▶ User may not always remember all details on error



Version Column

- The best way to enable versioning is with an additional **integer property** / column
 - Should have no semantic value (no meaning)
 - Should be updated by all apps using the table!

Uses @Version
no need for
getter / setter

```
@Entity
public class Customer {
    @Id
    @GeneratedValue
    private int id;
    private String firstname;
    private String lastname;

    @Version
    private int version;

    ...
}
```


Timestamp Column

- ▶ JPA also supports a **Timestamp column**
 - ▶ Not as good: may have business logic (can change)
 - ▶ Not every computer's time is exactly the same
 - ▶ But usually set by DB
 - ▶ Otherwise could give interesting bugs in finding who is first

```
@Entity
public class Customer {
    @Id
    @GeneratedValue
    private int id;
    private String firstname;
    private String lastname;

    @Version
    private Date timestamp;

    ...
}
```

@Version on
Date or Calendar

Without a column

- **Hibernate extension** – only works for objects that have not been detached
 - Checks if attributes are the same as when retrieved

```
@Entity
@org.hibernate.annotations.Entity(
    optimisticLock=OptimisticLockType.ALL,
    dynamicUpdate=true
)
public class Customer {
    @Id
    @GeneratedValue
    private int id;
    private String firstname;
    private String lastname;

    ...
}
```

Hibernate's version
of @Entity

Applications



CS544 EA

Concurrency: Application Transactions

Application Transactions

- ▶ Application Transactions are longer running conversations
 - ▶ Can be seen as a **Unit of Work** from the User Perspective
 - ▶ Spanning **two or more screens**
- ▶ The user expects these units of work to be
 - ▶ Atomic, Consistent, Isolated, and Durable
 - ▶ Submitting data after each screen would not allow us to roll back the entire unit of work (not Atomic)
 - ▶ Nor should you use a single database transaction across multiple screens
 - ▶ Keeping locks open during user think time



Checkout

- An online checkout is generally an Application Tx
 - Multiple screens (items, shipping, payment, etc)
 - Needs to be: Atomic, Consistent, Isolated, Durable
- Can be implemented with **optimistic concurrency**
 - Detach objects after first screen, store in session
 - Update objects on subsequent screens
 - Re-attach objects after final screen
 - Exception will be thrown if changes happened outside App TX

Pessimistic Locking



- ▶ For certain operations optimistic concurrency might not be enough
- ▶ Stricter isolation might be required to prevent the unrepeatable reads problem
- ▶ Hibernate can **request explicit database level locks** to provide increased isolation
- ▶ These locks will be released on commit

SELECT FOR UPDATE
on DBs that support it

```
Customer cust = (Customer)session.get(Customer.class, 1);  
em.lock(cust, LockModeType.UPGRADE);
```

```
Customer cust = em.find(Customer.class, 1, LockModeType.UPGRADE);
```

LockModeType

Lock Mode	Description	Throws PersistenceException on objects without version
None	Nothing	
OPTIMISTIC	Performs a version column check	
OPTIMISTIC_FORCE_INCREMENT	Ensures an increment on the version column before the transactions sommits	
PESSIMISTIC_FORCE_INCREMENT	Forces and increment on the version column right away and tries to get a write lock	
PESSIMISTIC_READ	Tries to get a read lock in the DB	
PESSIMISTIC_WRITE	Tries to get a write lock in the DB	
READ	Synonym of OPTIMISTIC	
WRITE	Synonym of OPTIMISTIC_FORCE_INCREMENT	

Newer apps should use
OPTIMISTIC versions
Not READ / Write

Concurrency Summary

- Databases often run in weaker isolation modes
- Optimistic concurrency uses a version column to prevent the Lost Update problem
- Pessimistic Locking can be used to force database locks or version column updates
- Application Transactions can be implemented using detached objects and version columns