Architecture of Enterprise Applications 7 Transaction

Haopeng Chen

***RE**liable, **IN**telligent and **S**calable Systems Group (**REINS**)*
Shanghai Jiao Tong University
Shanghai, China
http://reins.se.sjtu.edu.cn/~chenhp
e-mail: chen-hp@sjtu.edu.cn

Contents

- Transactions in Java EE Applications
  - What Is a Transaction?
  - Container-Managed Transactions
  - Bean-Managed Transactions
  - Transaction Timeouts
  - Isolation and Database Locking
  - Updating Multiple Databases

Transactions in Java EE Applications

- In a Java EE application, a transaction
  - is a series of actions that must all complete successfully, or else all the changes in each action are backed out. Transactions end in either a commit or a rollback.

- The Java Transaction API (JTA) allows applications
  - to access transactions in a manner that is independent of specific implementations

- The JTA defines the `UserTransaction` interface that applications use to start, commit, or roll back transactions.
  - Application components get a `UserTransaction` object through a JNDI lookup by using the name `java:comp/UserTransaction` or by requesting injection of a `UserTransaction` object.

What Is a Transaction?

- Pseudocode:

```
begin transaction
  debit checking account
  credit savings account
  update history log
commit transaction
```

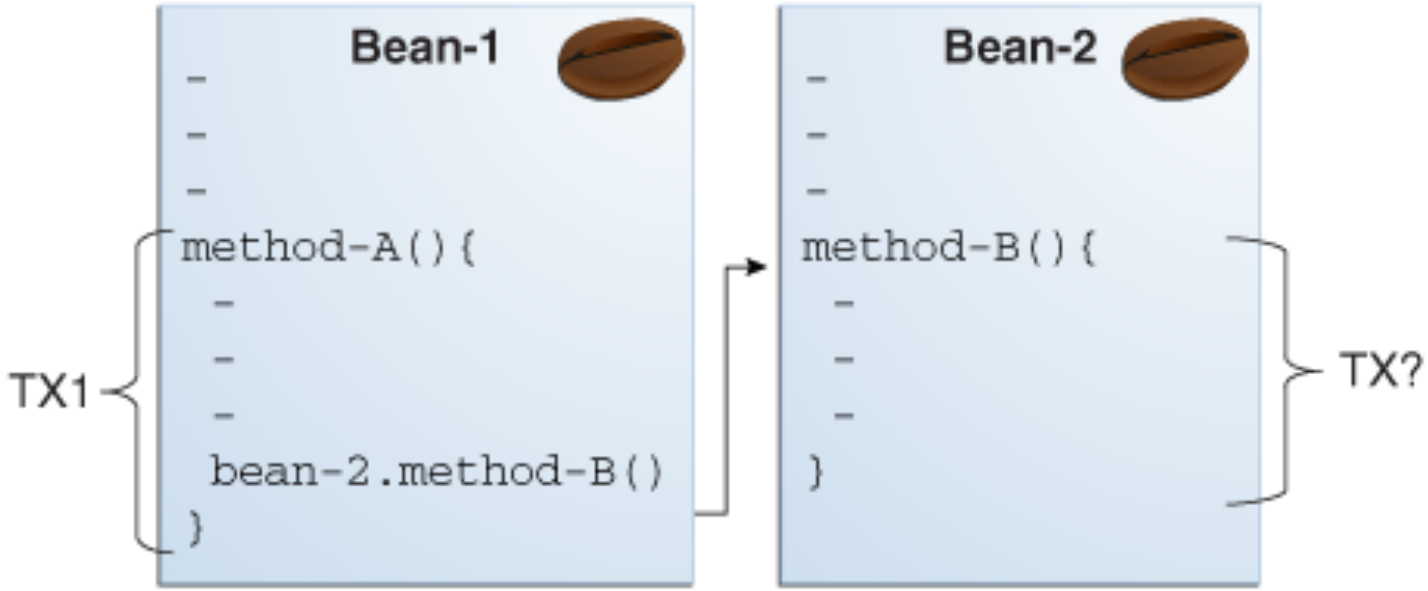
- A **transaction** is often defined as an indivisible unit of work
  - Either all or none of the three steps must complete.

- A transaction can end in two ways:
  - with a commit or with a rollback.

Container-Managed Transactions

REliable, INtelligent & Scalable Systems

- In an enterprise bean with container-managed transaction demarcation,
  - the EJB container sets the boundaries of the transactions.

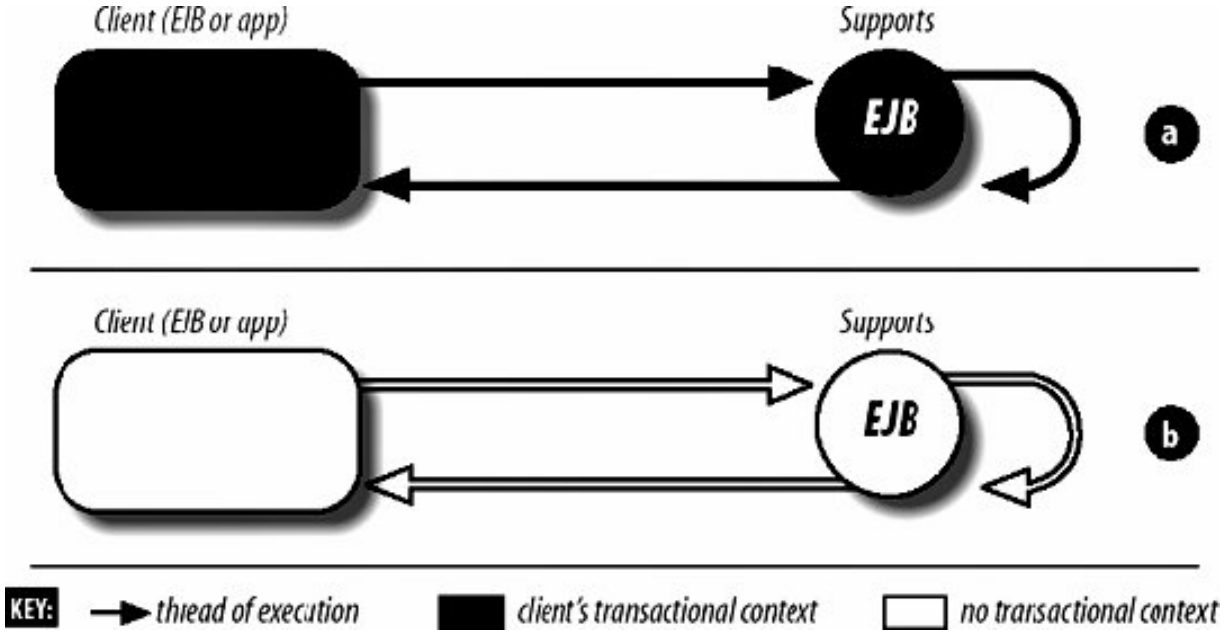


Transaction attributes defined

- NotSupported
  - Invoking a method on an EJB with this transaction attribute suspends the transaction until the method is completed.

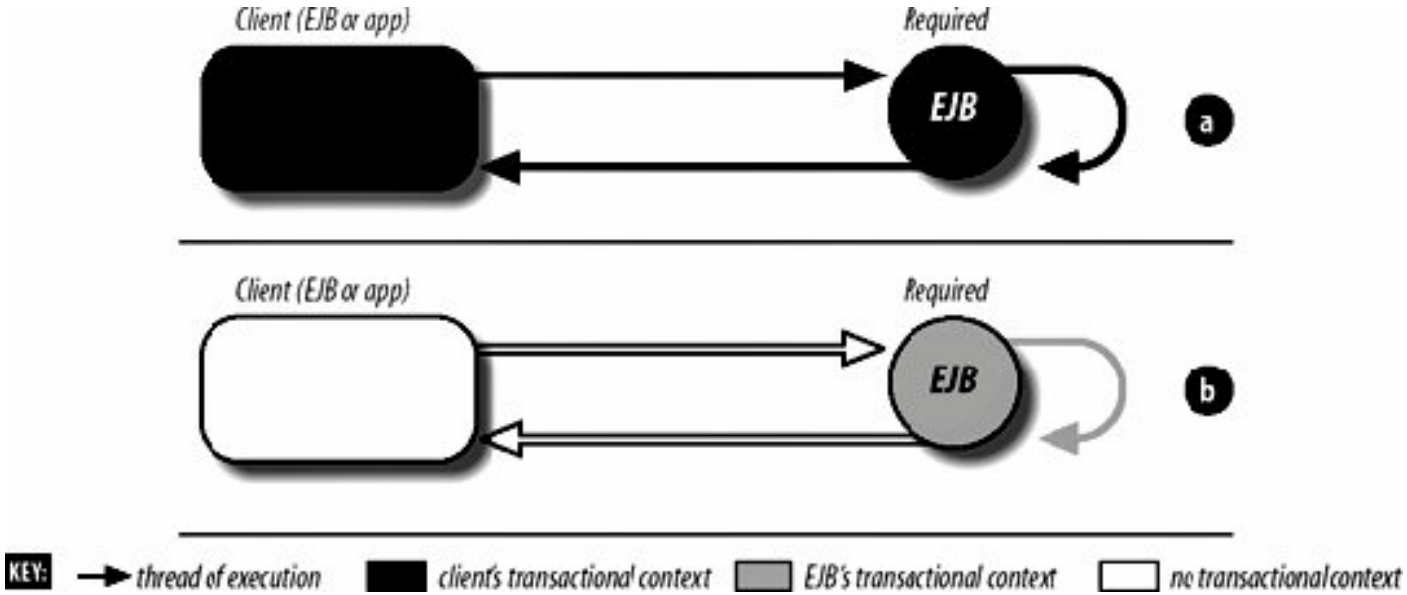
Transaction attributes defined

- Supports
  - This attribute means that the enterprise bean method will be included in the transaction scope if it is invoked within a transaction.



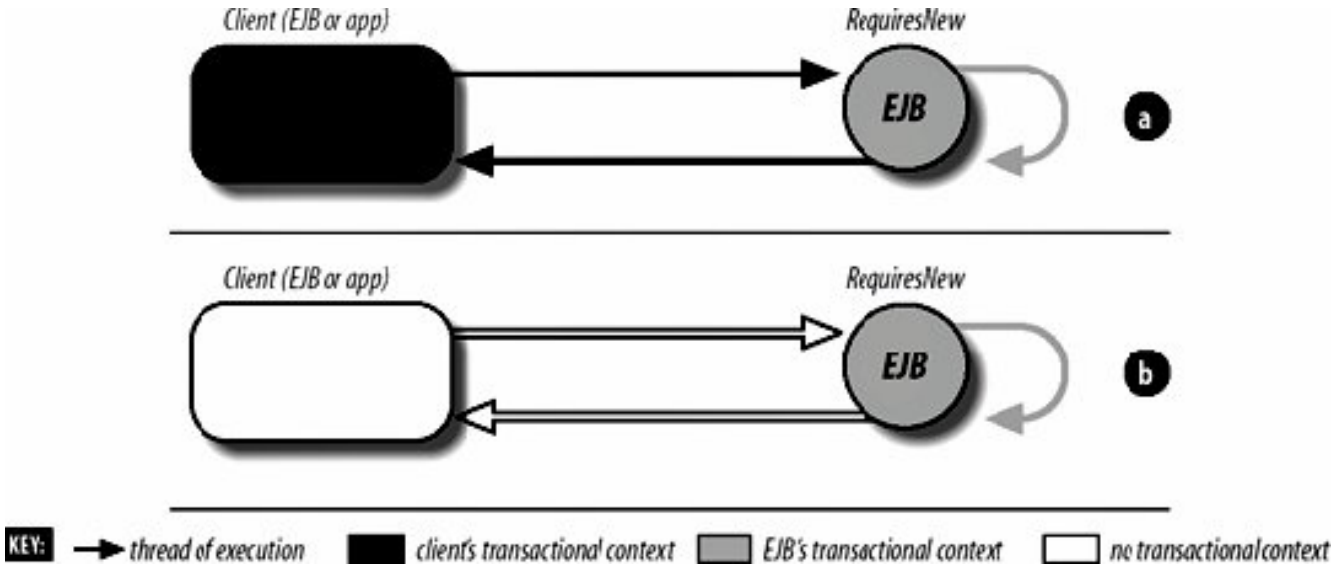
Transaction attributes defined

- Required
  - This attribute means that the enterprise bean method must be invoked within the scope of a transaction.



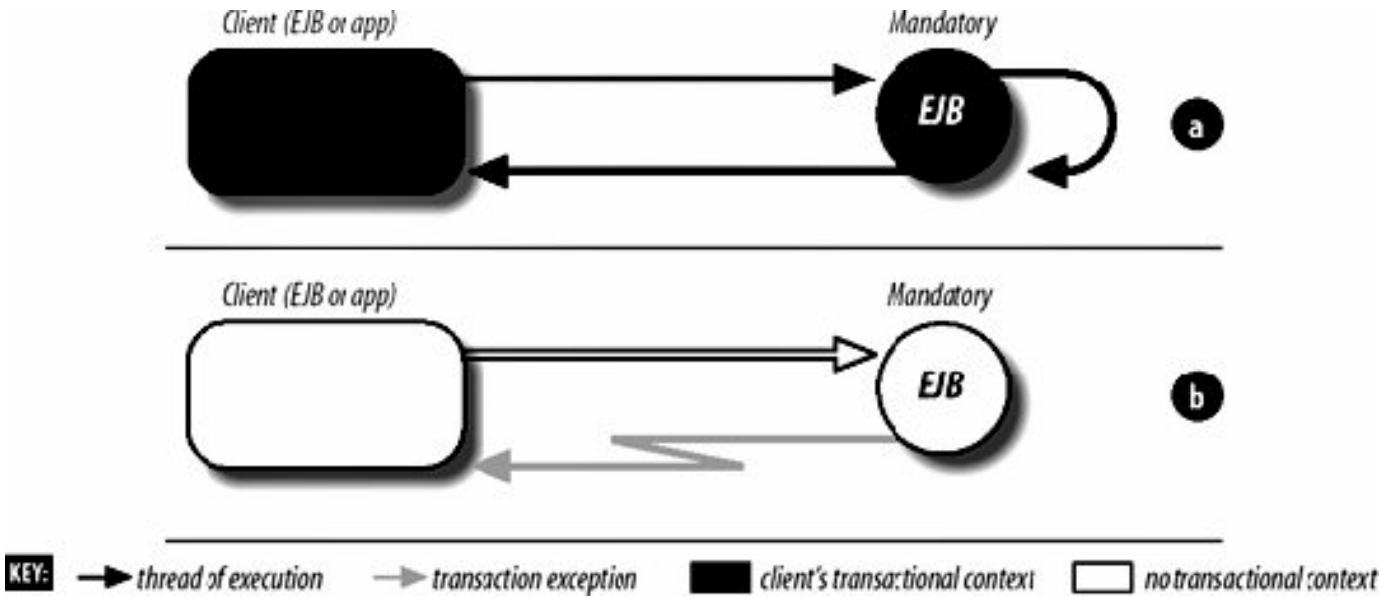
Transaction attributes defined

- RequiresNew
  - This attribute means that a new transaction is always started.



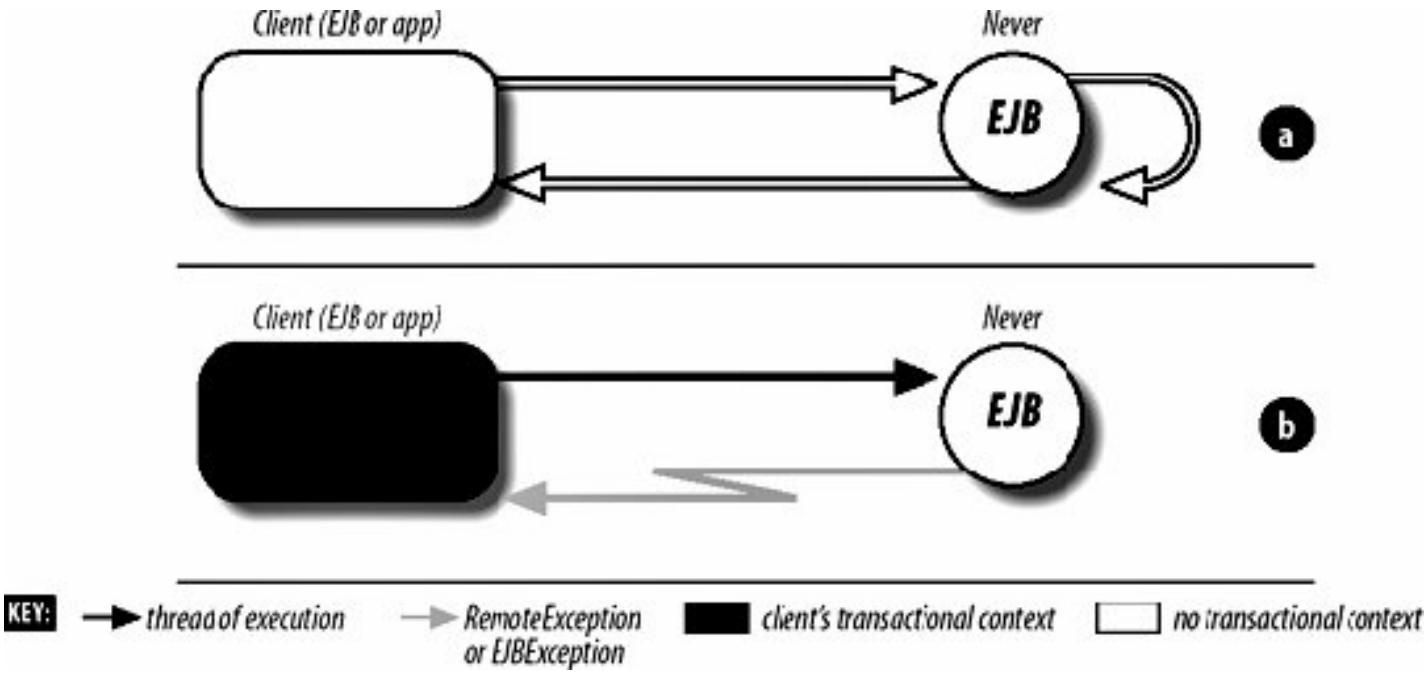
Transaction attributes defined

- Mandatory
  - This attribute means that the enterprise bean method must always be made part of the transaction scope of the calling client.



Transaction attributes defined

- Never
  - This attribute means that the enterprise bean method must not be invoked within the scope of a transaction.



Transaction Attributes and Scope

REliable, INtelligent & Scalable Systems

Transaction Attribute	Client's Transaction	Business Method's Transaction
Required	None	T2
Required	T1	T1
RequiresNew	None	T2
RequiresNew	T1	T2
Mandatory	None	Error
Mandatory	T1	T1
NotSupported	None	None
NotSupported	T1	None
Supports	None	None
Supports	T1	T1
Never	None	None
Never	T1	Error

Setting Transaction Attributes

- Transaction attributes are specified by
  - decorating the enterprise bean class or method with a `javax.ejb.TransactionAttribute` annotation
  - and setting it to one of the `javax.ejb.TransactionAttributeType` constants.

Transaction Attribute	TransactionAttributeType Constant
Required	TransactionAttributeType.REQUIRED
RequiresNew	TransactionAttributeType.REQUIRES_NEW
Mandatory	TransactionAttributeType.MANDATORY
NotSupported	TransactionAttributeType.NOT_SUPPORTED
Supports	TransactionAttributeType.SUPPORTS
Never	TransactionAttributeType.NEVER

Setting Transaction Attributes

- The following code snippet demonstrates how to use the `@TransactionAttribute` annotation:

```
@TransactionAttribute(NOT_SUPPORTED)
@Stateful
public class TransactionBean implements Transaction {
 ...
 @TransactionAttribute(REQUIRES_NEW)
 public void firstMethod() {...}

 @TransactionAttribute(REQUIRED)
 public void secondMethod() {...}
 public void thirdMethod() {...}
 public void fourthMethod() {...}
}
```

Rolling Back a CMT

- There are two ways to roll back a container-managed transaction.
  - First, if a system exception is thrown, the container will automatically roll back the transaction.
  - Second, by invoking the `setRollbackOnly` method of the `EJBContext` interface, the bean method instructs the container to roll back the transaction.
  - If the bean throws an application exception, the rollback is not automatic but can be initiated by a call to `setRollbackOnly`.

Synchronizing Instance Variables

- The `SessionSynchronization` interface, which is optional, allows stateful session bean instances to receive transaction synchronization notifications.
  - The container invokes the `SessionSynchronization` methods (`afterBegin, beforeCompletion, and afterCompletion`) at each of the main stages of a transaction.

```
afterBegin:
    oldvalue = value;

afterCompletion(boolean b):
    if (!b)
        value = oldvalue;
```

Methods Not Allowed in CMT

- The list of prohibited methods follows:
  - The `commit`, `setAutoCommit`, and `rollback` methods of `java.sql.Connection`
  - The `getUserTransaction` method of `javax.ejb.EJBContext`
  - Any method of `javax.transaction.UserTransaction`

Bean-Managed Transactions

- In bean-managed transaction demarcation,
  - the code in the session or message-driven bean explicitly marks the boundaries of the transaction.

- Pseudocode:

```
begin transaction
 ...
 update table-a
 ...
 if (condition-x)
     commit transaction
 else if (condition-y)
     update table-b
     commit transaction
 else rollback transaction
 begin transaction
     update table-c
 commit transaction
```

Transaction Timeouts

- For container-managed transactions,
  - you can use the Administration Console to configure the transaction timeout interval.
- For enterprise beans with bean-managed JTA transactions,
  - you invoke the `setTransactionTimeout` method of the `UserTransaction` interface.

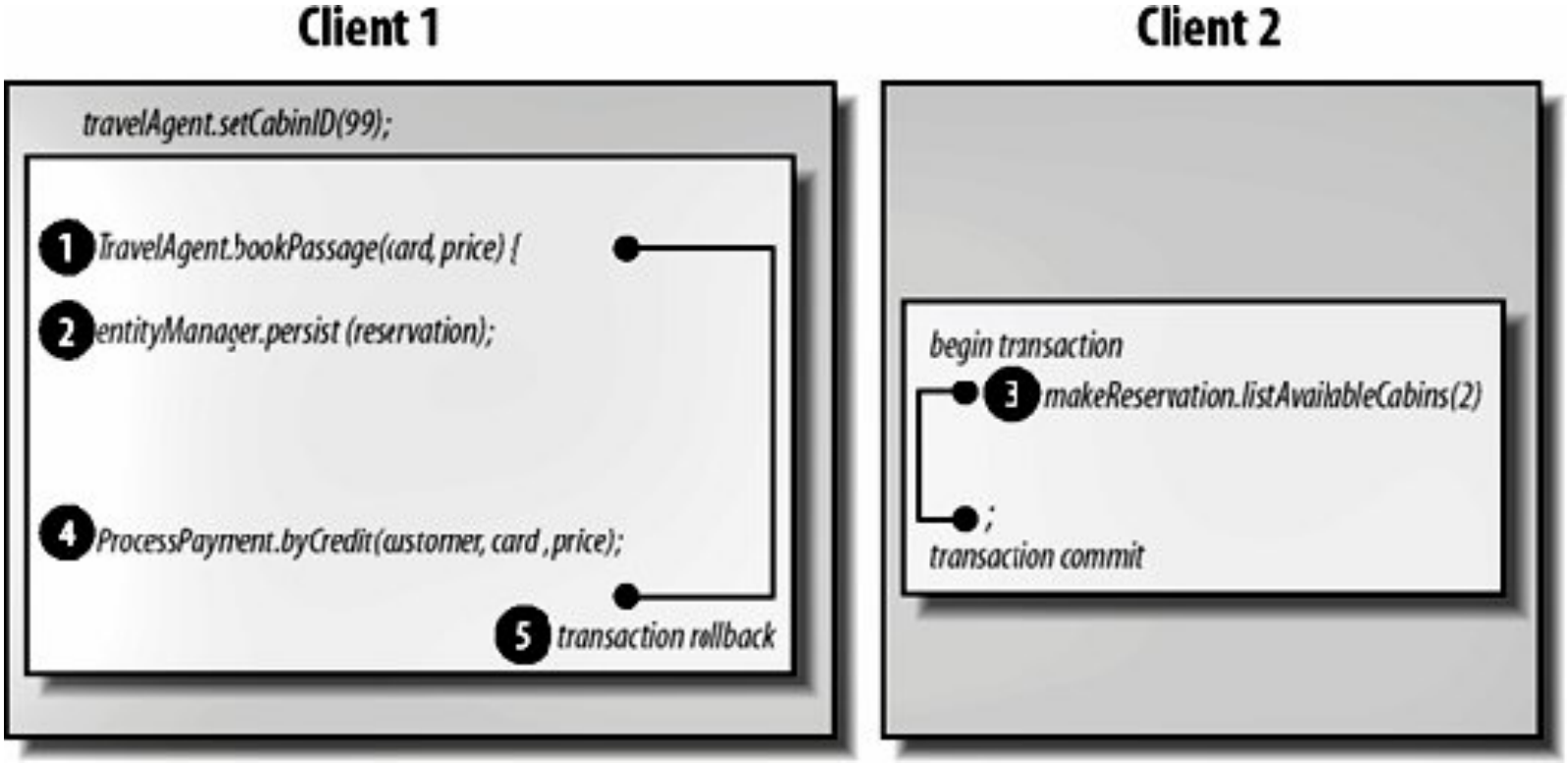
Isolation and Database Locking

- Let's think about two separate client applications accessing the same data specifically.

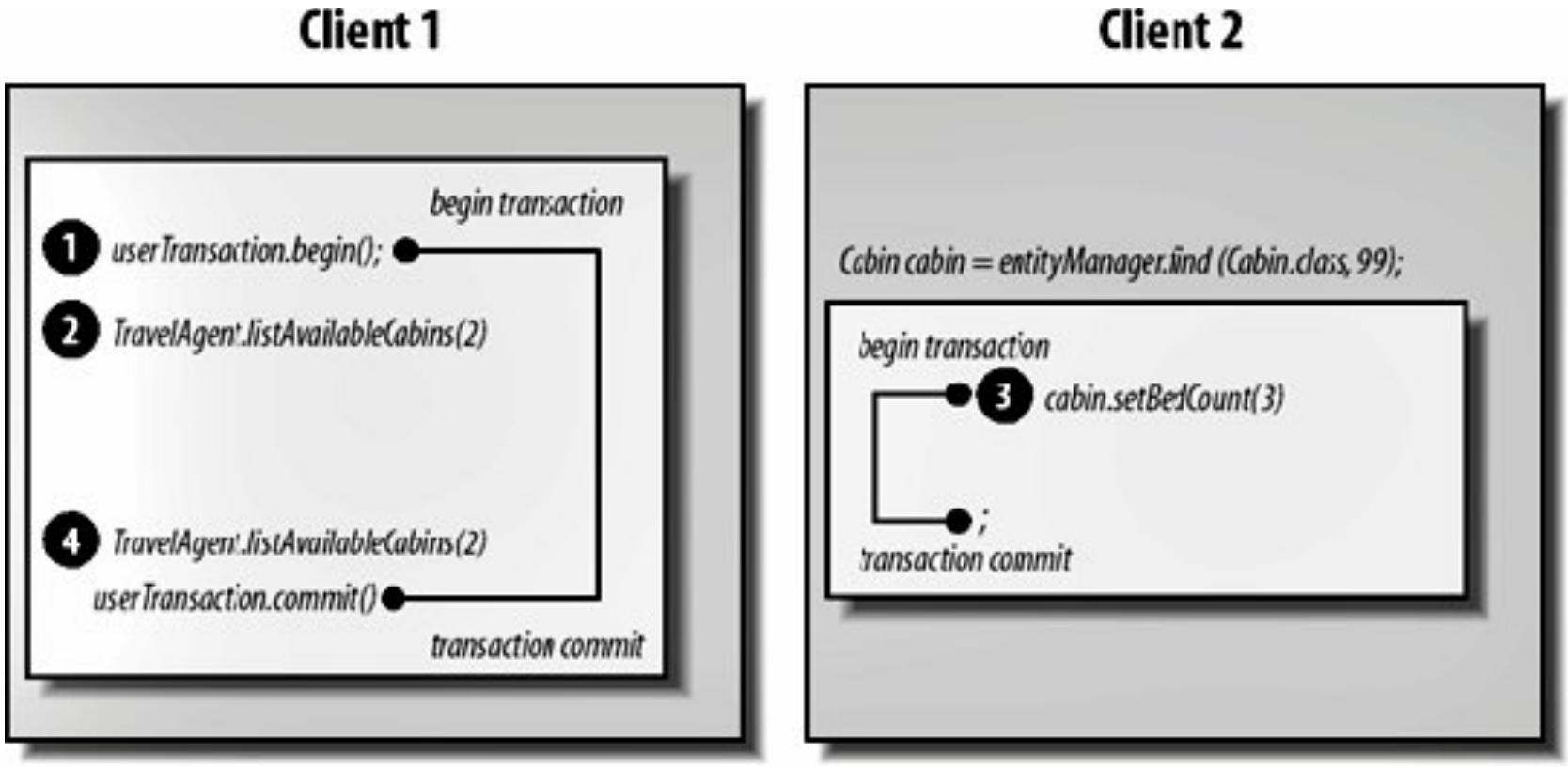
```
public List listAvailableCabins(int bedCount) {
    Query query = entityManager.createQuery(
                  "SELECT name FROM Cabin c
                   WHERE c.ship = :ship AND
                   c.bedCount = :beds AND NOT ANY (
                   SELECT cabin from Reservation res
                   WHERE res.cruise = :cruise");
  query.setParameter("ship", cruise.getShip( ));
  query.setParameter("beds", bedCount);
  query.setParameter("cruise", cruise);
  return query.getResultList( );
}
```

- For this example, assume that both methods have a transaction attribute of `Required` .

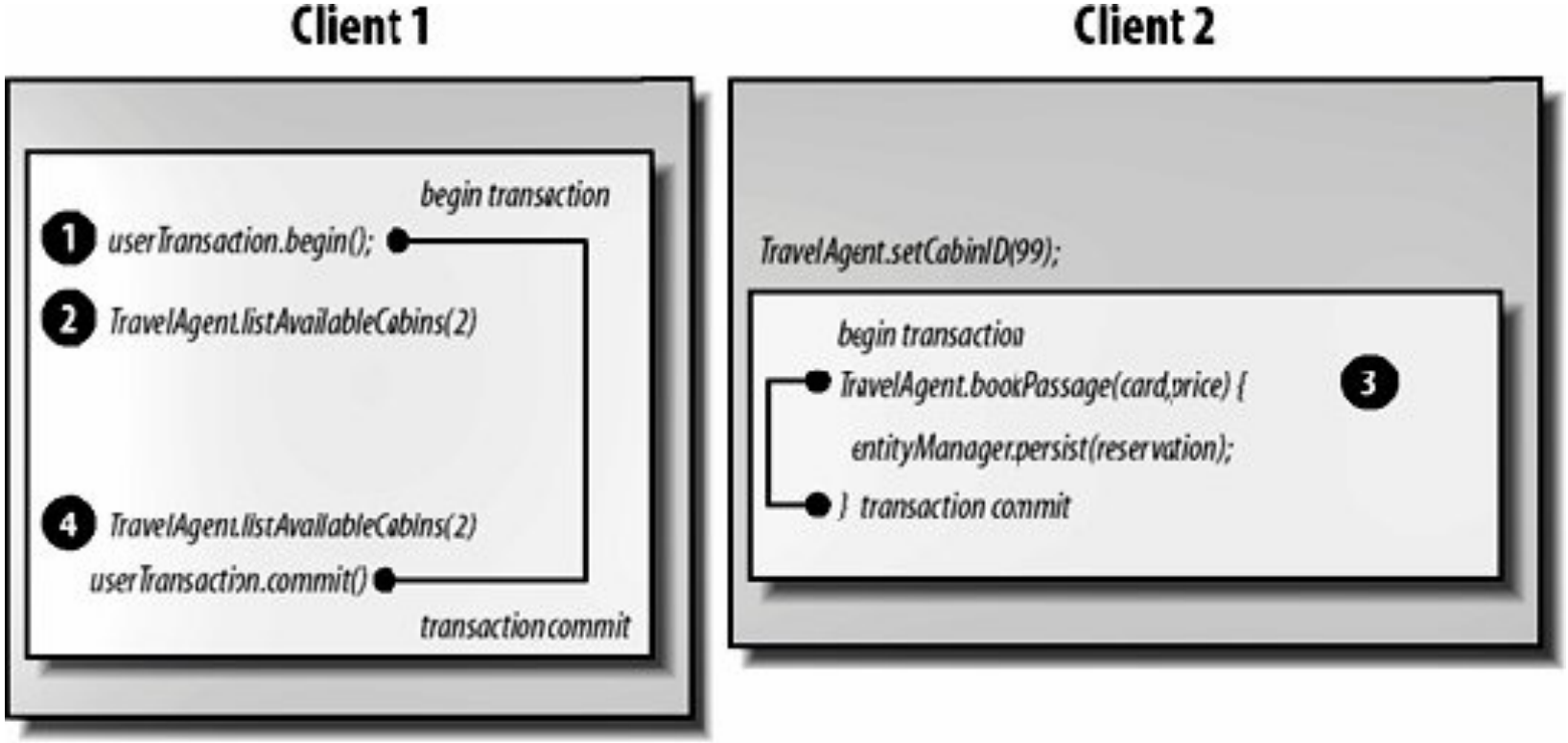
Dirty Reads


Client 1
travelAgent.setCabinID(99);
1 TravelAgent.bookPassage(card, price) {
2 entityManager.persist (reservation);
4 ProcessPayment.byCredit(customer, card , price);
5 transaction rollback
Client 2
begin transaction
3 makeReservation.listAvailableCabins(2)
;
transaction commit

Repeatable Reads



Phantom Reads



Database Locks

- Read locks
  - Read locks prevent other transactions from changing data read during a transaction until the transaction ends, thus preventing nonrepeatable reads.
  - Other transactions can read the data but not write to it. The current transaction is also prohibited from making changes.

- Write locks
  - Write locks are used for updates. A write lock prevents other transactions from changing the data until the current transaction is complete but allows dirty reads by other transactions and by the current transaction itself.
  - In other words, the transaction can read its own uncommitted changes.

Database Locks

- Exclusive write locks
  - Exclusive write locks are used for updates. An exclusive write lock prevents other transactions from reading or changing the data until the current transaction is complete.
  - It also prevents dirty reads by other transactions.

- Snapshots
  - A snapshot is a frozen view of the data that is taken when a transaction begins. Some databases get around locking by providing every transaction with its own snapshot.
  - Snapshots can prevent dirty reads, nonrepeatable reads, and phantom reads. They can be problematic because the data is not real-time data; it is old the instant the snapshot is taken.

Transaction Isolation Levels

- Read Uncommitted
  - The transaction can read uncommitted data (i.e., data changed by a different transaction that is still in progress).
  - Dirty reads, nonrepeatable reads, and phantom reads can occur.
  - Bean methods with this isolation level can read uncommitted changes.

- Read Committed
  - The transaction cannot read uncommitted data; data that is being changed by a different transaction cannot be read.
  - Dirty reads are prevented; nonrepeatable reads and phantom reads can occur.
  - Bean methods with this isolation level cannot read uncommitted data.

Transaction Isolation Levels

- Repeatable Read
  - The transaction cannot change data that is being read by a different transaction.
  - Dirty reads and nonrepeatable reads are prevented; phantom reads can occur.
  - Bean methods with this isolation level have the same restrictions as those in the Read Committed level and can execute only repeatable reads.

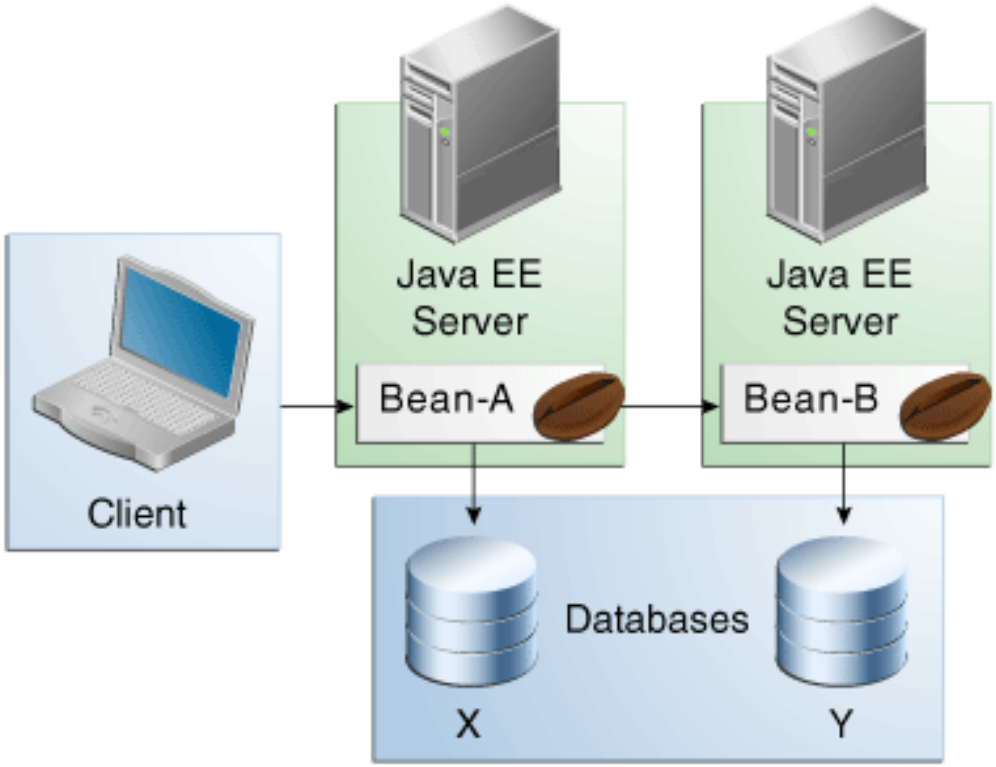
- Serializable
  - The transaction has exclusive read and update privileges; different transactions can neither read nor write to the same data.
  - Dirty reads, nonrepeatable reads, and phantom reads are prevented.
  - This isolation level is the most restrictive.

Controlling isolation levels

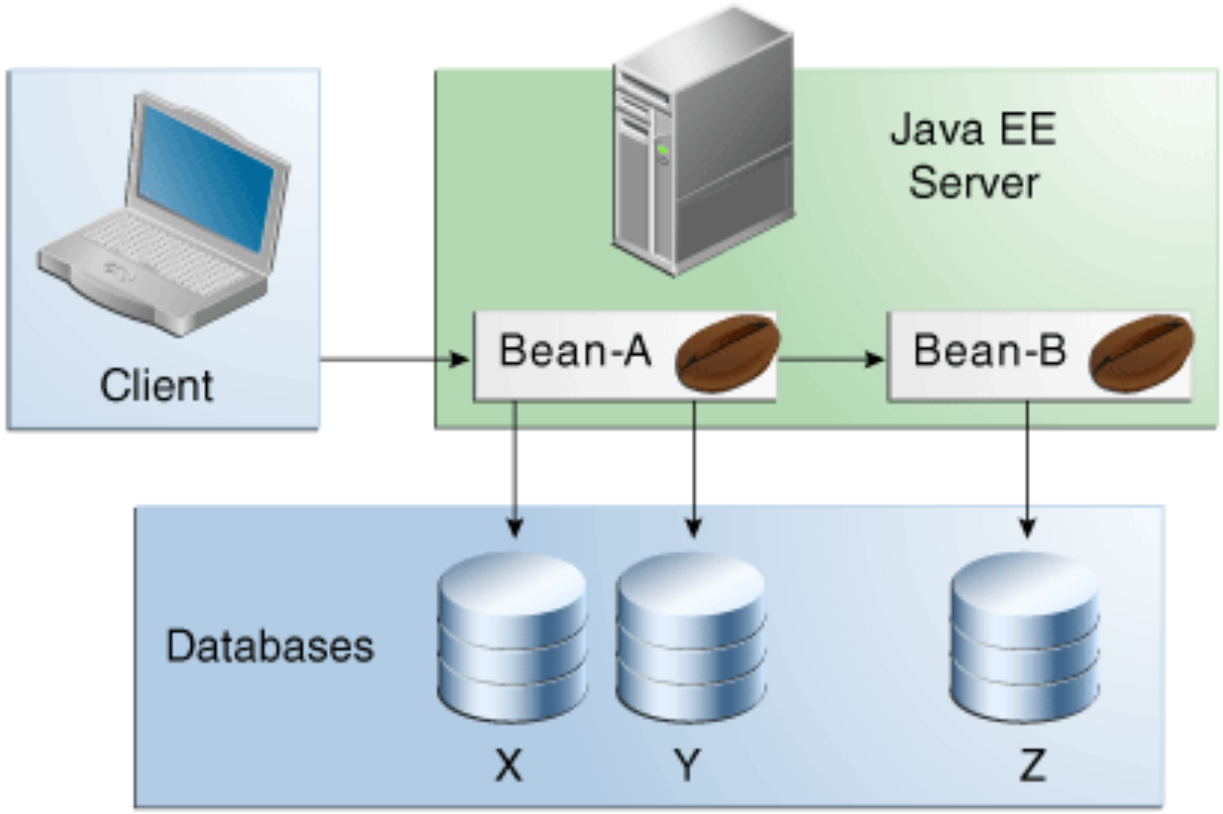
- You to specify the transaction isolation level using the database's API.
- For example:

```
DataSource source = (javax.sql.DataSource)

jndiCntxt.lookup("java:comp/env/jdbc/titanDB");
    Connection con = source.getConnection( );

con.setTransactionIsolation(Connection.TRANSACTION_SERIALIZABLE);
```

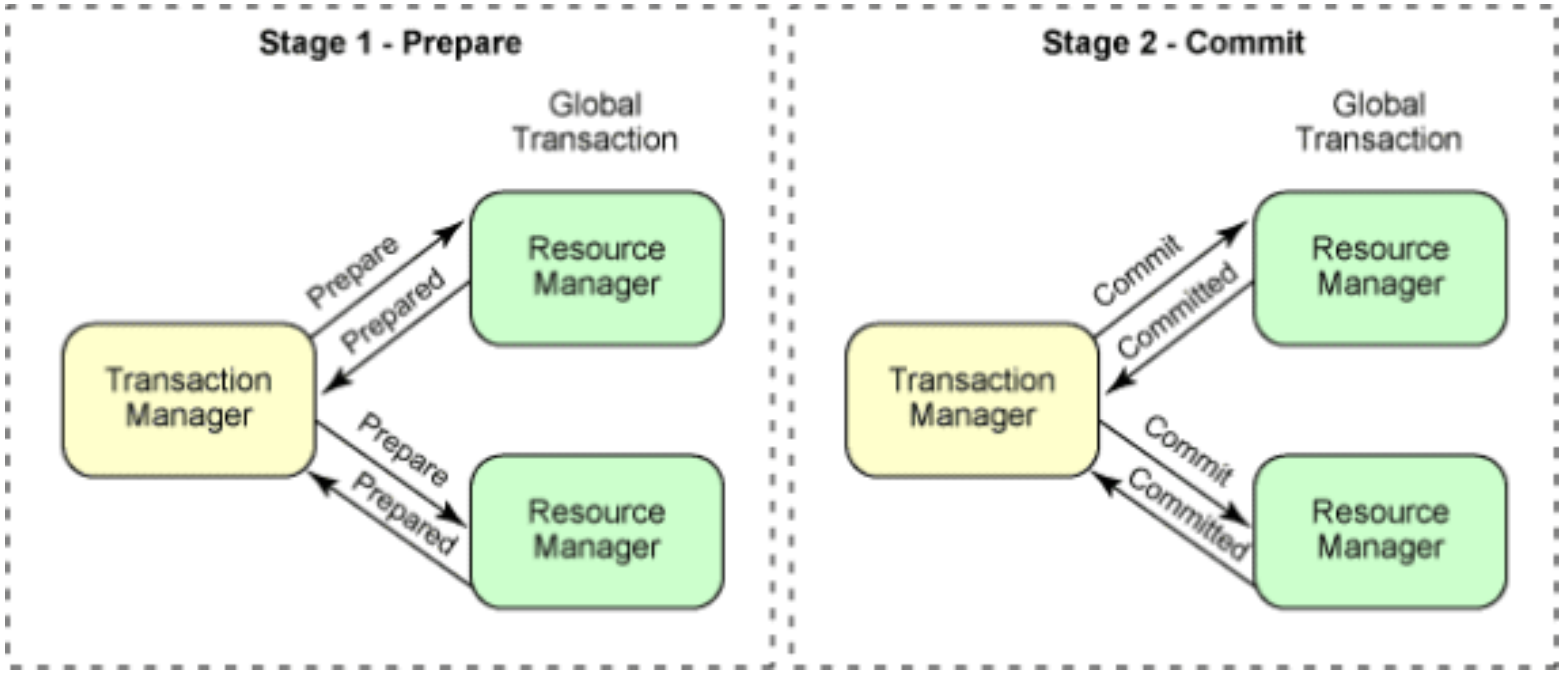
Updating Multiple Databases



Updating Multiple Databases



Two-phase commit



Project

- To improve the quality of your book store website, please add necessary transaction management into it, including:
  - To setup proper isolation level of your DB.
  - To implement data access with declarative transactions.
  - To manage the declarative transactions with at least 3 types.

References

- Web Applications: What are They? What of Them?
  - http://www.acunetix.com/websitesecurity/web-applications/
- The Java EE 7 Tutorial
  - http://docs.oracle.com/javaee/7/tutorial/doc/javaeetutorial7.pdf
- Mastering EJB 3
- IBM WebSphere Developer Technical Journal: Exploiting the J2EE Connector Architecture
  - https://www.ibm.com/developerworks/websphere/techjournal/0607_wakelin/0607_wakelin.html

Thank You!