

JEE - Java Enterprise Edition

Transactions continued



Java Enterprise Edition Transaction Isolation

- Transaction Isolation is the “I” in ACID
- Transaction Isolation is defined in terms of ***isolation conditions*** called
 - **dirty reads**
 - **repeatable reads**
 - **phantom reads**
- They describe what happens, when **two or more transactions** operate **on the same data**



Java Enterprise Edition Transaction Isolation

- To illustrate these conditions, think about two separate clients using their own instances of the TravelAgentEJB to access the same data, a cabin record with the primary key 99
- The examples resolve around the database table “RESERVATION”, accessed by the **bookPassage()** method and a new method **listAvailableCabins()** seen on the next slide



Java Enterprise Edition Transaction Isolation

List **all cabins**, that are **not part of an existing** Reservation

```
public List listAvailableCabins(int bedCount)
    throws IncompleteConversationalState {
    if (cruise == null)
        throw new IncompleteConversationalState( );

    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)");

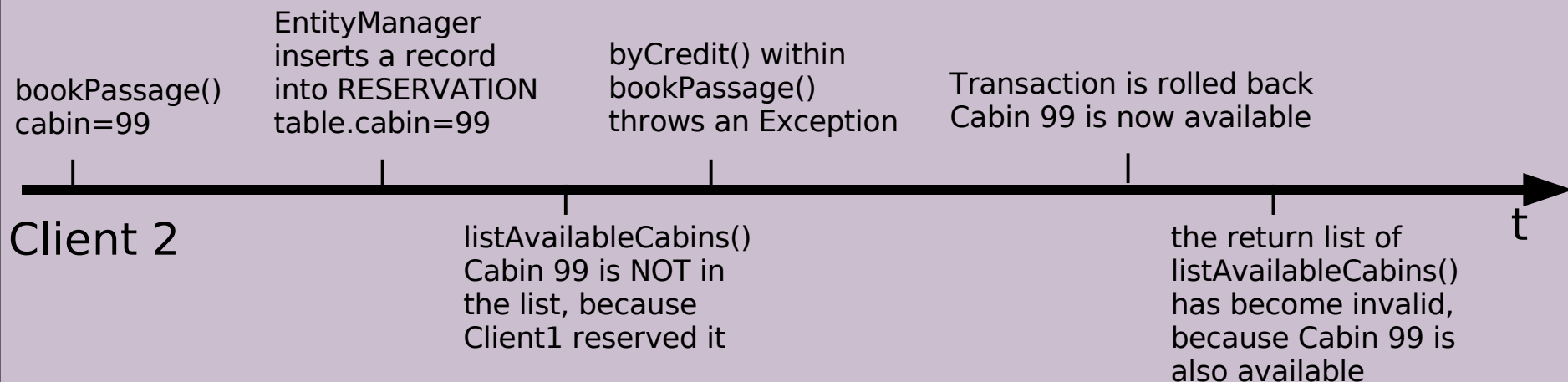
    query.setParameter("ship", cruise.getShip( ));
    query.setParameter("beds", bedCount);
    query.setParameter("cruise", cruise);

    return query.getResultList( );
}
```

Java Enterprise Edition Isolation Conditions

Client 1

Dirty Reads



Java Enterprise Edition

Dirty Reads

- A *dirty read* occurs when a transaction reads uncommitted changes made by a previous transaction
- If the first transaction is rolled back, the data read by the second transaction becomes invalid, because the rollback undoes the changes
- The second transaction will not be aware that the data it has read has become invalid



Java Enterprise Edition Isolation Conditions

Client 1

Repeatable Reads

Explicit
usertransaction
started

listAvailableCabins(2)
Cabin 99 is in the list
of available clients

listAvailableCabins(2)
Cabin 99 is still in the list

t

Client 2

changes bedCount
of Cabin 99 from 2
to 3.



Java Enterprise Edition

Repeatable Reads

- A *repeatable read* occurs when the data read is guaranteed to look the same if read again **during the same transaction**
- Two different ways:
 - The **data is locked against changes**. Data can not be changed by other transactions until the current transaction ends
 - The **data is a snapshot**, that doesn't reflect changes. Other transactions can change the data, but these changes will not be seen by this transaction if the read is repeated
- A *nonrepeatable read* occurs, when the data retrieved in a subsequent read can return different results

Java Enterprise Edition Isolation Conditions

Client 1

Phantom Reads

Explicit
usertransaction
started

listAvailableCabins(2)
Cabin 99 is in the list
of available clients

listAvailableCabins(2)
Cabin 99 is no longer in the list

t

Client 2

bookPassage()
creates a Reservation
reserving Cabin 99



Java Enterprise Edition Phantom Reads

- A phantom *read* occurs when new records added to the database are detectable by transactions that started prior to the insert
- Queries will include records added by other transactions after their transaction is started
- The inserted record is called *phantom record*, as it is not committed, yet



Java Enterprise Edition

Database Locks

- Relational databases use **different locking techniques**
- The most common are:
 - **Read locks**
 - **Write locks**
 - **Exclusive Write Locks**
- The mechanisms **control how transactions access data concurrently**
- Locking mechanisms **impact the read conditions** mentioned before
- Database **vendors implement the locks differently**

Java Enterprise Edition

Database Locks

- The three types of locks are:
 - **Read Locks** prevent other transactions from changing data read during a transaction. Other transactions can read the data, but not write it. The current transaction is also prohibited from making changes.
 - **Write Locks** are used for updates. They prevent other transactions from changing data until the current transaction is complete, but allow dirty reads from itself and other transactions



Java Enterprise Edition Database Locks

- The three types of locks are (continued):
 - **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. Some vendors do not allow transactions to read their own data, while it is exclusively locked



Java Enterprise Edition Transaction Isolation Levels

- Isolation Levels are commonly used in database systems to describe how locking is applied to data within a transaction
 - **Read Uncommitted:** The transaction can read uncommitted data. Dirty Reads, nonrepeatable reads and phantom reads can occur
 - **Read Committed:** The transaction cannot read uncommitted data. Dirty Reads are prevented, nonrepeatable reads and phantom reads can occur
 - **Repeatable Read:** The transaction cannot change data that is being read by a different transaction. Dirty Reads and nonrepeatable reads are prevented

Java Enterprise Edition Transaction Isolation Levels

- Isolation Levels (continued):
 - **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
- These **isolation levels are the same as those defined for JDBC**
- They **map to** the final **variables in the java.sql.Connection** class



Java Enterprise Edition Transaction Isolation Levels

- In EJB, the deployer sets transaction isolation levels in a vendor-specific way
- Choosing the correct isolation level requires some research about the database you are using and how it handles locking
- Different EJB servers allow different levels of granularity for isolation levels
- Most EJB servers and EntityManager implementations control the isolation level through JDBC



Java Enterprise Edition

Programmatic Locking

- The EntityManager interface has a specific lock() method for performing entity locks
- To use it, you pass in the entity object you want to lock and indicate whether you want a read or write lock

```
package javax.persistence;  
  
public enum LockModeType {  
    READ,  
    WRITE  
}
```

```
package javax.persistence;  
  
public interface EntityManager {  
    void lock(Object entity, LockModeType type);  
}
```



JEE - Java Enterprise Edition

Transactions continued

