



**Java™
Italian
Portal**



Le Transazioni nelle architetture J2EE

Fabrizio Marini
Principal Consultant



Fabrizio Marini

Nel 1996 è stato uno dei due fondatori della K-Tech di cui è stato socio fino al 2003. In K-Tech era il responsabile della formazione e dei clienti strategici dell' "Area Java". E' l'ideatore di JIP – Java Italian Portal (www.javaportal.it).

Ha iniziato a programmare in Java nel 1996 e da allora si è specializzato in questo settore, fornendo consulenze alle più prestigiose ditte del mercato italiano.

Tra il 2000 ed il 2001 ha tenuto numerosi corsi per la Selfin SpA (Centro di Istruzione IBM) inerenti la programmazione e l'analisi ad oggetti, Java, WebSphere ed UML.

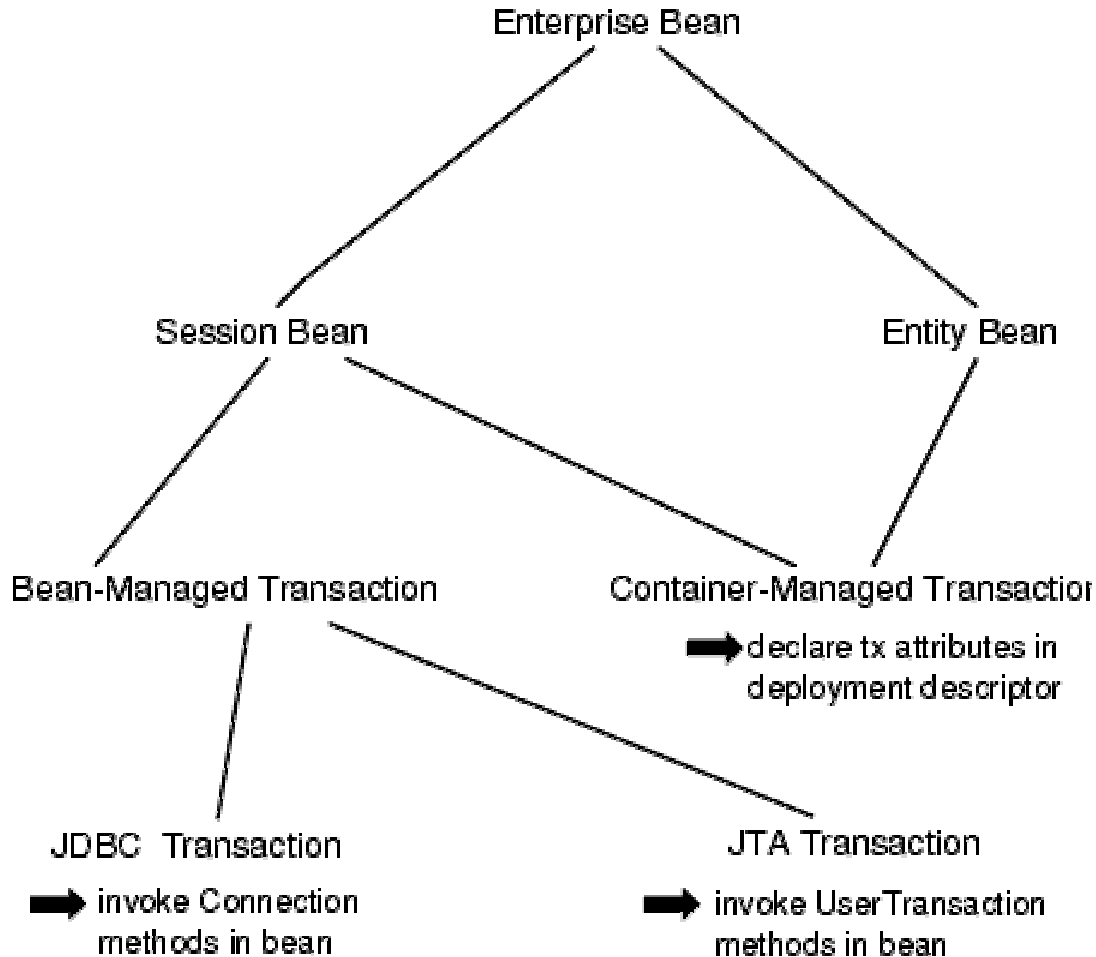
Da Gennaio 2001 ha iniziato ad insegnare per BEA Systems Italia specializzandosi sulla piattaforma WebLogic secondo lo standard J2EE1.2 & 1.3, in particolare ha tenuto corsi su WebLogic, EJB & JMS, Personalization & Commerce Server, Business Process Management - WebLogic Integration.

A Febbraio 2002 in collaborazione con SilverStream ha progettato ed aperto il primo UDDI in Italia. (www.webservicesportal.it)

Il 10.02.2003, lascia la K-Tech ed entra nei Professional Services di Bea Italia in qualità di Principal Consultant.



Summary of Transaction Options





Tipi di Transazioni

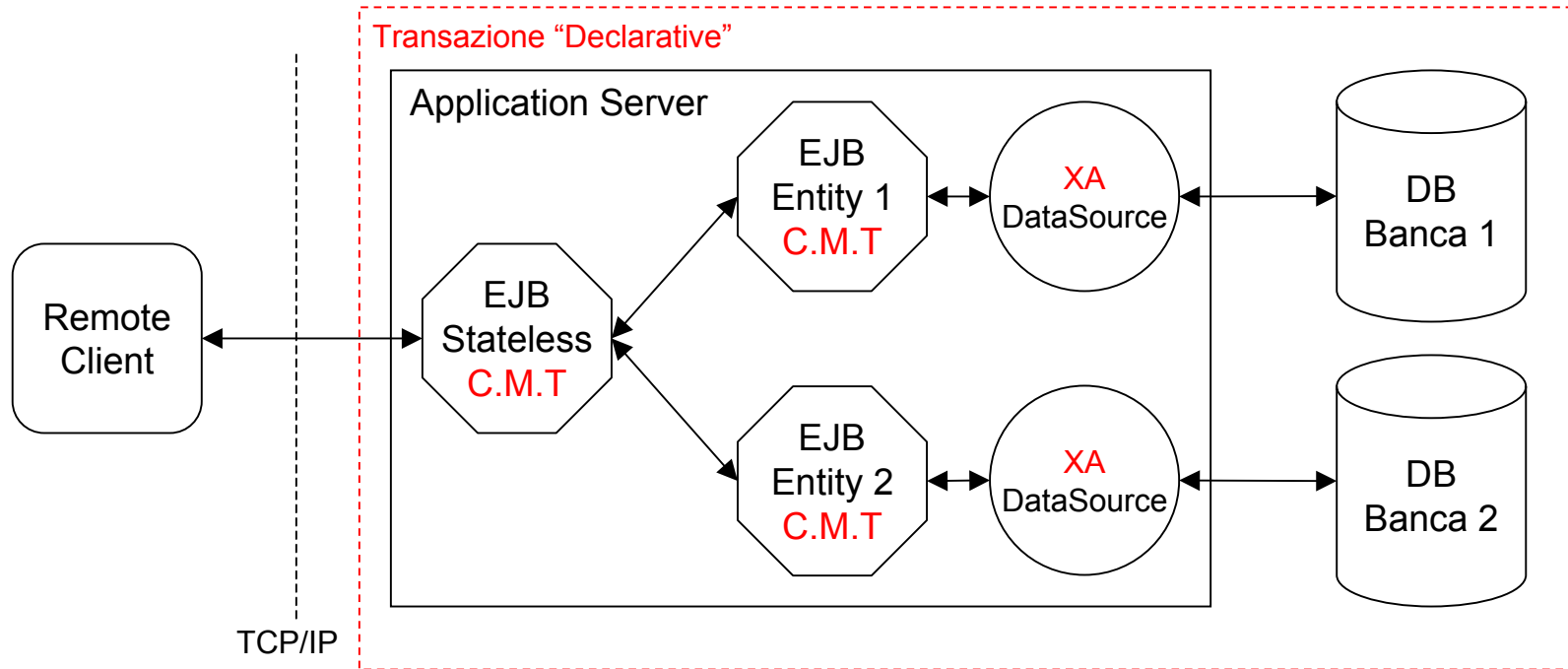
1 Declarative

2 Programmatic

3 Client Controlled



Transazione “Declarative”





Attributi Transazionali

- Required
- Supports
- Mandatory
- Not Supported
- Never
- Requires New
- Bean Managed [1.0]

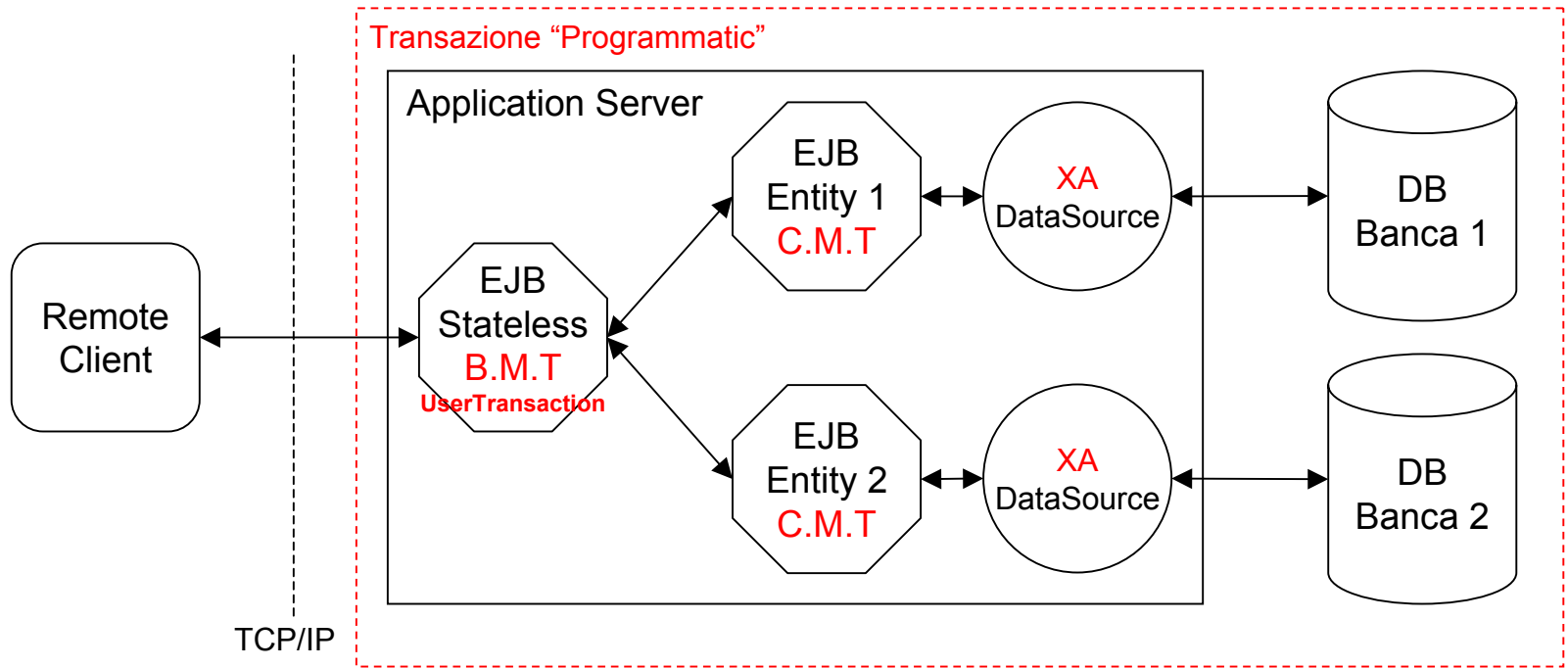


Transaction Attributes & Scope

Transaction Attribute	Client's Transaction	Business Method's Transaction
Required	none	T2
	T1	T1
RequiresNew	none	T2
	T1	T2
Mandatory	none	Error – Transaction Required Exception
	T1	T1
NotSupported	none	none
	T1	none
Supports	none	none
	T1	T1
Never	none	none
	T1	Error - Remote Exception



Transazione "Programmatic"



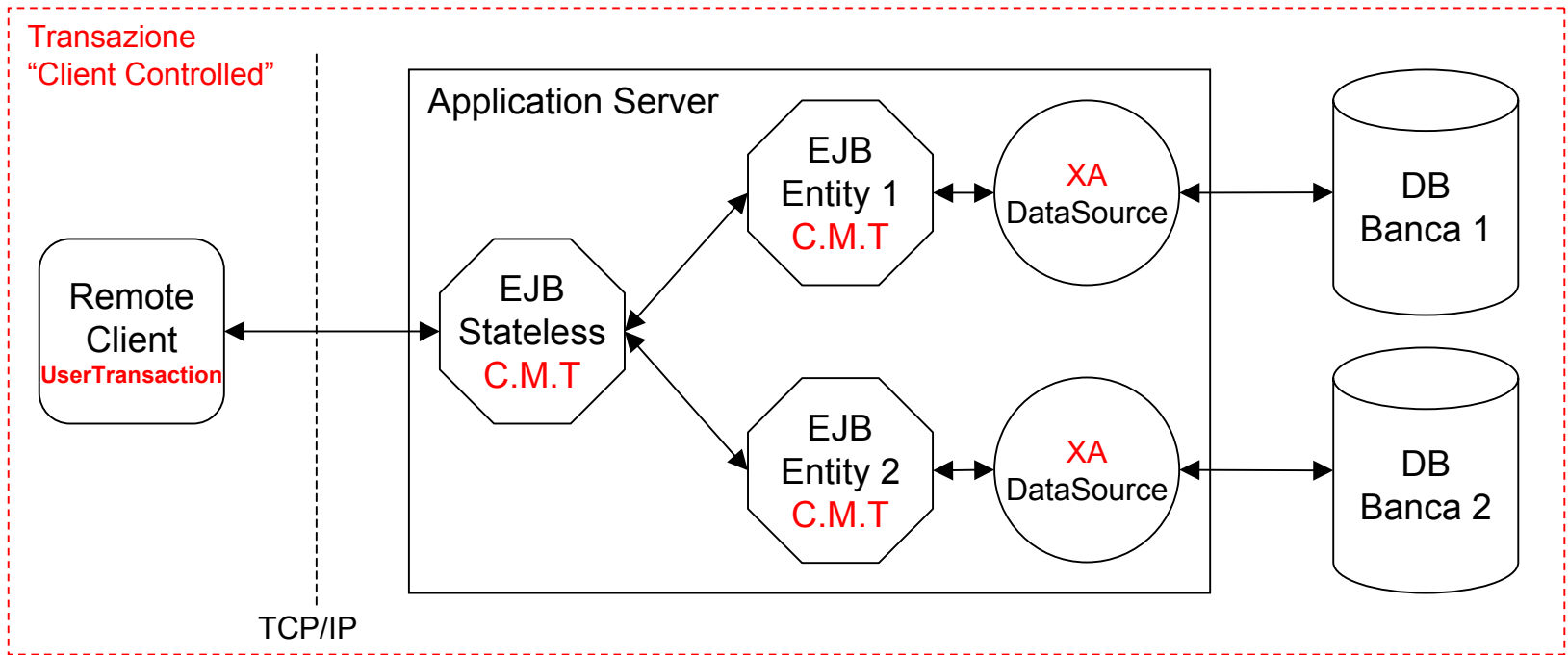


UserTransaction - Esempio

```
public class Atm{
    public static void main(String args[]){
        try{
            Hashtable env = new Hashtable();
            env.put(Context.INITIAL_CONTEXT_FACTORY, "weblogic.jndi.WLInitialContextFactory" );
            env.put(Context.PROVIDER_URL, "t3://localhost:7001");
            InitialContext ic = new InitialContext(env);
            UserTransaction ut = (UserTransaction)ic.lookup("javad.transaction.UserTransaction");
            ut.begin();
            Account1Home home1 = (Account1Home)PortableRemoteObject.narrow(
                ic.lookup("com.bea.edu.wls.Account1Bean"), Account1Home.class);
            Account1 account1 = home1.create("id1", 1000.00, "Elmer Fudd");
            account1.withdraw(50.00);
            Account2Home home2 = (Account2Home)PortableRemoteObject.narrow(
                ic.lookup("com.bea.edu.wls.Account2Bean"), Account2Home.class);
            Account2 account2 = home2.create("bean2", 2500.00, "Homer Simpson");
            account2.withdraw(99.99);
            ut.commit();
            System.out.println("Account 1: balance = " + account1.getBalance());
            System.out.println("Account 2: balance = " + account2.getBalance());
        } catch(Exception e) { System.out.println("Error " + e); }
    }
}
```



Transazione “Client Controlled”





Transactions in Applets and Application Client

The J2EE platform does not require transaction support in applets and application clients, though like distributed transactions, a J2EE product might choose to provide this capability for added value.

So, whether applets and application clients can directly access a UserTransaction object depends on the capabilities provided by the container.

To ensure portability, applets and application clients should delegate transactional work to enterprise beans.



Transaction Style

<i>Transaction Style</i>	<i>Utilizzo</i>
Programmatic	Uncommonly used Useful when you need multiple transactions per EJB method call
Declarative	Commonly used Useful when you need a single transaction per EJB method call
Client-Controlled	Rarely used Useful for systems involving non-EJB clients Client and server should be closely located



SessionSynchronization

- Motivation: A stateful session bean is a transactional resource too.
 - Has in-memory state that needs to rollback in case of failure.
- Can participate in transactions like a database by implementing SessionSynchronization:

```
public interface javax.ejb.SessionSynchronization
{
    public void afterBegin();
    public void beforeCompletion();
    public void afterCompletion(boolean);
}
```



Utilizzo di SessionSynchronization

- 2 uses of *SessionSynchronization*:
 - Alerts your bean to transaction failure
 - If transaction fails, you undo your in-memory state
 - Alerts you to transaction boundaries
 - Enables you to cache database data for performance
- Usage:

afterBegin()	Transaction just started	Read in database data (create your cache) Create a backup copy of your state
beforeCompletion()	Transaction about to end	Flush your cache
afterCompletion()	Transaction has ended	If transaction failed, revert to backup copy of your state

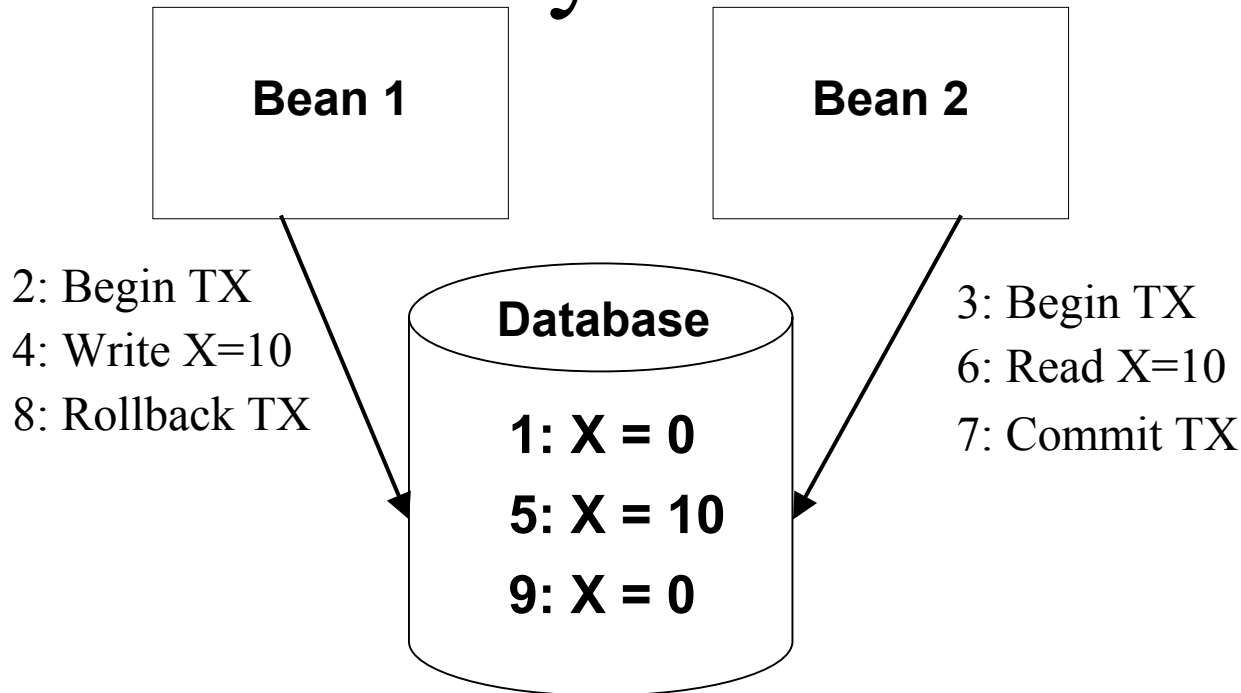


Transaction Isolation: I in “ACID”

- Problem: Need to share data safely
- Isolation is the tradeoff between concurrency and safety
 - EJB 1.0: EJB Isolation Levels put you in control
 - EJB 1.1 and beyond: No EJB isolation levels. Use JDBC.
- Let’s discuss how to properly use isolation
- We begin by investigating the types of transactional problems



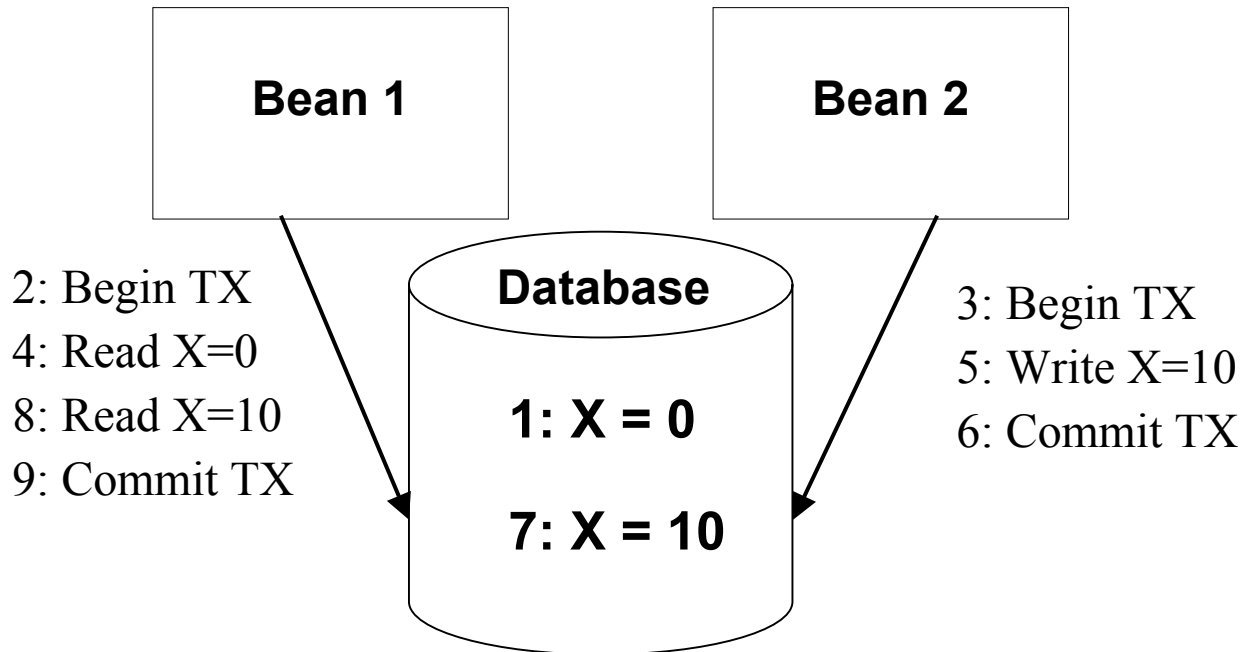
Dirty Read



- `TRANSACTION_READ_UNCOMMITTED` isolation level allows for this problem to happen
- `TRANSACTION_READ_COMMITTED` isolation level solves this problem



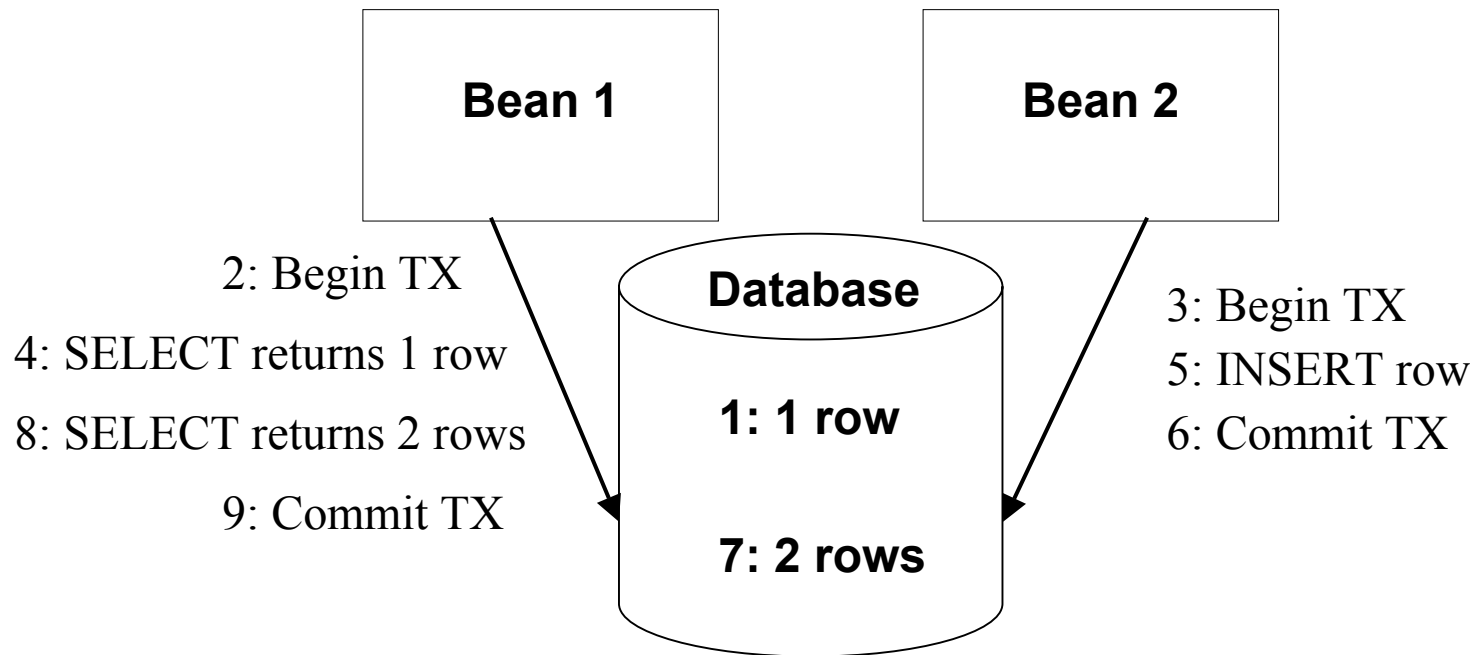
Unrepeatable Read



- `TRANSACTION_READ_COMMITTED` isolation level allows for this problem to happen
- `TRANSACTION_REPEATABLE_READ` isolation level solves this problem



Phantom Problem



- `TRANSACTION_REPEATABLE_READ` isolation level allows for this problem to happen
- `TRANSACTION_SERIALIZABLE` isolation level solves this problem



Transaction Isolation

Isolation Level	Dirty Reads?	Unrepeatable Reads?	Phantom Reads?
READ UNCOMMITTED	Yes	Yes	Yes
READ COMMITTED	No	Yes	Yes
REPEATABLE READ	No	No	Yes
SERIALIZABLE	No	No	No



Transaction Isolation

- `READ_UNCOMMITTED` (fastest)
 - You can see other transactions' uncommitted updates
 - No data sharing, non mission-critical apps.
- `READ_COMMITTED` (fast)
 - You can't see other transactions' uncommitted updates
 - Rough Reporting tools. Data should be committed. Only need a snapshot.
- `REPEATABLE_READ` (medium)
 - When you read a row, nobody can touch that row until you're done
 - Mission-critical, high data sharing. You have not performed a query, so phantoms are OK, so long as the data you're working with isn't modified.
- `SERIALIZABLE` (slow)
 - After you do a query, nobody can insert new rows that satisfies your query until you're done
 - Mission-critical, high data sharing. You have performed a query, and you'd like to know about any new phantom rows.



Ringraziamenti

KeyTECH
SOLUZIONI INFORMATICHE