Concurrent Access - Locking

Concurrent Access - Locking

- **Data concurrency**
  - If the application will have concurrent writers to the same objects, then a locking strategy is critical so that data corruption can be prevented.

- **Optimistic/Pessimistic Locking**
  - These are two strategies for preventing concurrent modification of the same object/row.

Concurrent Access - Locking

Optimistic Locking

Optimistic Locking

- Optimistic locking assumes that the data will not be modified between when you read the data until you write the data.
- This is the most common style of locking used and recommended in today's persistence solutions.
- This strategy involves checking that one or more values from the original object read are still the same when updating it.
- This verifies that the object is not changed by another user in between the read and the write.

Optimistic Locking - JPA

- **JPA supports using an optimistic locking version field that gets updated on each update.**

```
@Version
private long version;
```

- **This field can either be numeric or a timestamp value.**
  - A numeric value is recommended as a numeric value is more precise, portable, performant and easier to deal with than a timestamp.

Optimistic Locking - JPA

- **The `@Version` annotation or `<version>` element is used to define the optimistic lock version field.**
- **The annotation is defined on the version field or property for the object, similar to an Id mapping. The object must contain an attribute to store the version field.**

```
@Entity
public abstract class Employee{
   @Id
   private long id;
   @Version
   private long version;

  (…)
}
```

version field

Optimistic Locking - JPA

- **The object's version attribute is automatically updated by the JPA provider and should not be modified by the application.**

- **When a locking contention is detected an `OptimisticLockException` will be thrown.**

- **This could be wrapped insider a `RollbackException`, or other exceptions if using JTA**

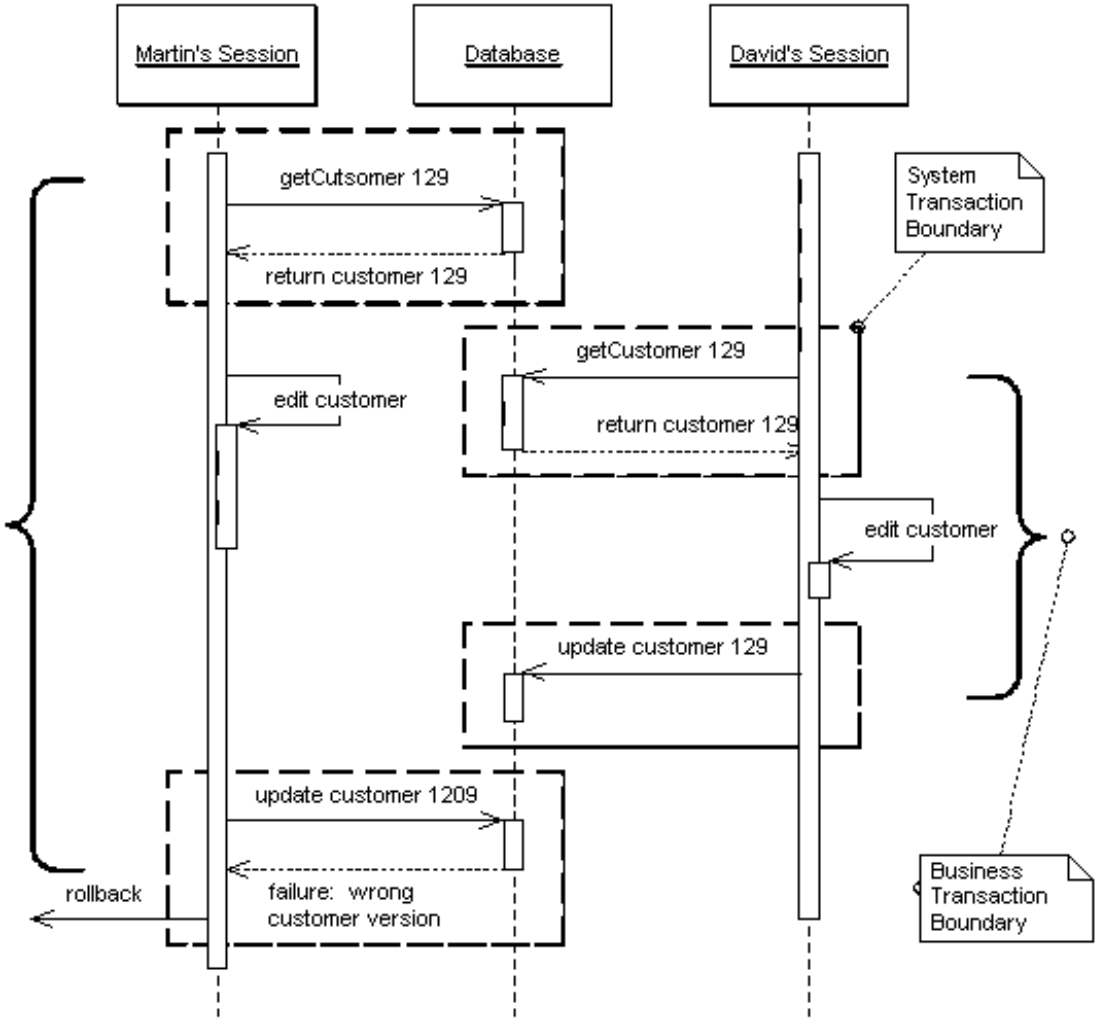
Optimistic Lock

Pattern

- **Prevents conflicts between concurrent business transactions by detecting a conflict and rolling back the transaction**

Optimistic Lock

Pattern



source: Patterns of Enterprise Application Architecture

Concurrent Access - Locking

Pessimistic Locking

Pessimistic Locking

- **Pessimistic locking means acquiring a lock on the object before you begin to edit the object, to ensure that no other users are editing the object;**
- **Pessimistic locking is typically implemented through using database row locks;**
  - such as through the SELECT ... FOR UPDATE SQL syntax.
- **The data is read and locked. When changes are made and the transaction is committed, it releases the lock.**

Pessimistic Locking

- **The main issues with pessimistic locking is it uses database resources, so require a database transaction and connection to be held open for the duration of the edit.**
- **This is typically not desirable for interactive web applications.**
- **The main advantages of pessimistic locking is that once the lock is obtained, it is fairly certain that the edit will be successful.**
  - This can be desirable in highly concurrent applications, where optimistic locking may cause too many optimistic locking errors.

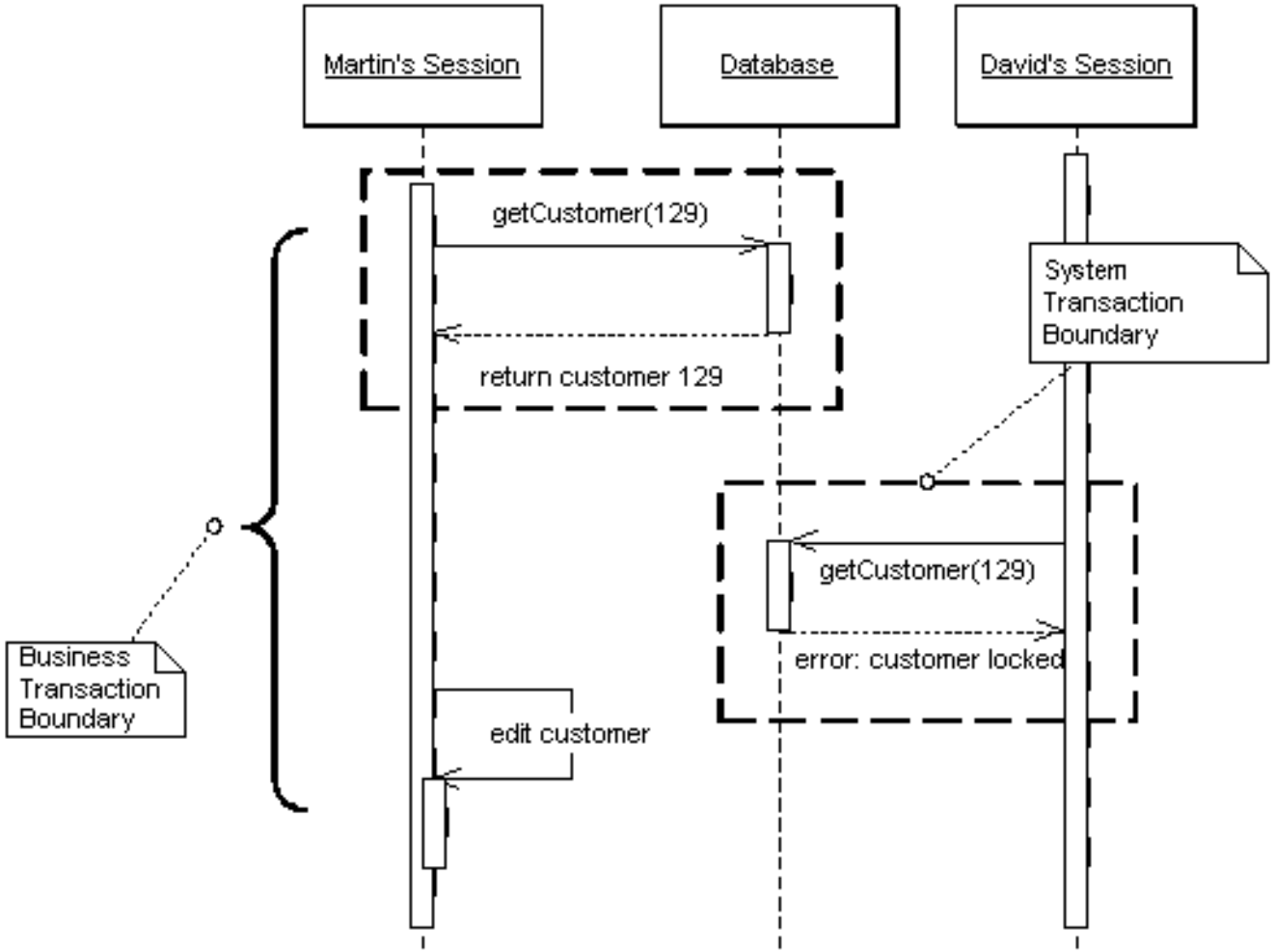
Pessimistic Lock

Pattern

- **Prevents conflicts between concurrent business transactions by allowing only one business transaction at a time to access data**

Pessimistic Lock

Pattern



source: Patterns of Enterprise Application Architecture

Pessimistic Locking in JPA 2

- **JPA 2 supports pessimistic locking, with three locking modes added to the existing optimistic locking such as READ and WRITE.**
- **These are all the locking modes available in JPA (the pessimistic locking modes are bold):**
  - READ (JPA1)
  - WRITE (JPA1)
  - OPTMISTIC (Synonymous to READ)
  - OPTIMISTIC_FORCE_INCREMENT (Synonymous to WRITE)
  - **PESSIMISTIC_READ (JPA2)**
  - **PESSIMISTIC_WRITE (JPA2)**
  - **PESSIMISTIC_FORCE_INCREMENT (JPA2)**
  - NONE

JPA 2 - PESSIMISTIC_READ

- **In this mode the Entity Manager holds the lock on an entity during read operations as soon as the transaction begins.**
- **The lock is not released until the transaction is completed.**
  - This is best used when you access data that is not frequently modified, as it allows other transactions to read the entity.

```
// Transaction t1 beginst1.begin();
// Employee entity is read from the DB
Employee e = manager.find(Employee.class, 1);
// Lock is performed after read
manager.lock(e, LockModeType.PESSIMISTIC_READ);
// Transaction is committed
t1.commit();
```

JPA 2 - PESSIMISTIC_WRITE

- **The PESSIMISTIC_WRITE mode generally represents an exclusive lock.**
- **In this mode the Entity Manager holds the lock on an entity as soon as the entity is updated in a transaction.**
  - This is best used in when there is a very high probability of update failure due to multiple transactions accessing the object.

```
// Transaction t1 begins
t1.begin();
// Employee entity is read and write lock is applied
Employee e = em.find(Employee.class, 1,
                     LockModeType.PESSIMISTIC_WRITE);
// Transaction is committed
t1.commit();
```

JPA 2 - PESSIMISTIC_FORCE_INCREMENT

- **In this mode the EntityManager holds the lock on an entity when a transaction reads the entity.**
- **The version number is incremented towards the end of the transaction, irrespective of whether the entity was updated or not.**

JPA 2 - PESSIMISTIC_FORCE_INCREMENT

```
t1.begin();
Employee e = em.find(Employee.class, 1);
// Get the department of Employee with Id 1. Returns department {Id='A'}
Department d = e.getDept();
em.lock(d, LockModeType.PESSIMISTIC_FORCE_INCREMENT);
em.flush();
// Make the Thread to sleep
Thread.sleep(20000);
t1.commit();
// Transaction t2 begins
t2.begin();
// Get an instance of Department entity with Id 'A'
Department d = em.find(Dep.class, 'A');
d.setDeptName("RESEARCH");
t2.commit();
```

At the same time, another transaction (t2) tries to update the same department instance, which results in an exception and the t2 transaction getting rolled back.

A lock is performed on the department entity and a flush method of the EntityManager is invoked so that the version number is incremented in the database even if the entity is not modified

Locking Demo

Demo