CS544 EA

Applications

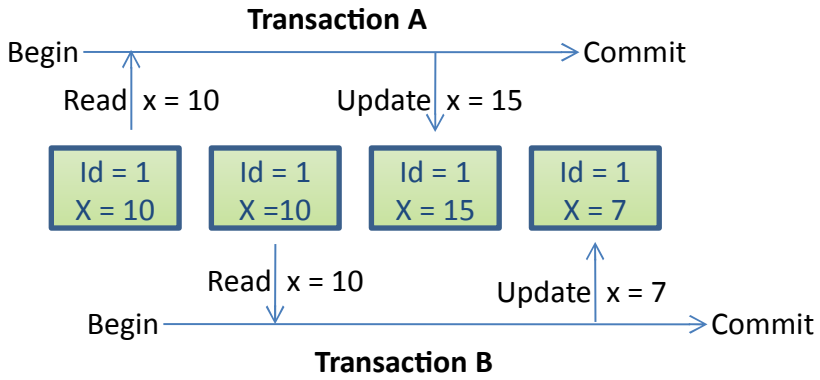
Concurrency: Optimistic Concurrency

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

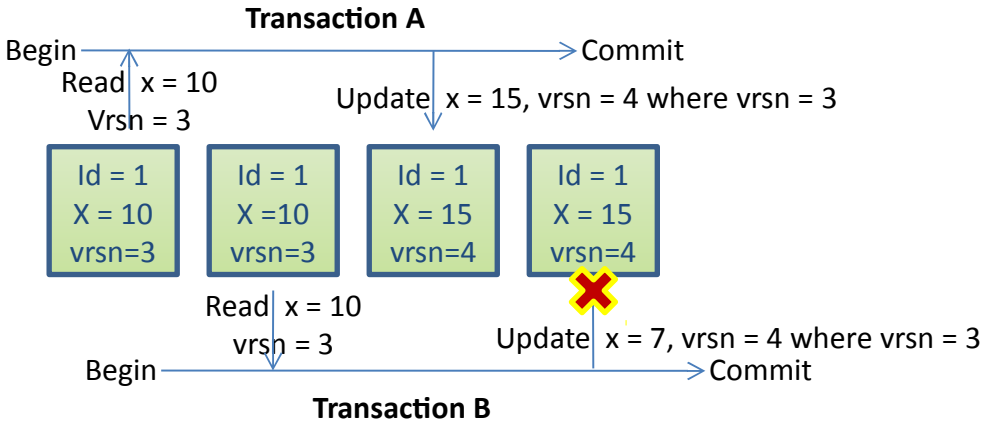
- Read Committed Allows:
  - **Last update to commit wins**
  - First update lost
- Timeline:
  - Transactions A and B read id=1, x=10
  - Transaction A changes x to x=15 (increment by 5)
  - Transaction B wants to decrement by 3, sets X=7
  - Neither A or B is aware that data was lost!



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