LOCKING

JPA supports two types of locking mechanisms: optimistic model and pessimistic model.

Let's consider the airline database as an example. The flights table stores information about flights, and tickets stores information about booked tickets. Each flight has its own capacity, which is stored in the flights.capacity column. Our application should control the number of tickets sold and should not allow purchasing a ticket for a fully filled flight.

To do this, when booking a ticket, we need to get the capacity of the flight and the number of tickets sold from the database, and if there are empty seats on the flight, sell the ticket, otherwise, inform the user that the seats have run out. If each user request is processed in a separate thread, data inconsistency may occur.

Suppose there is one empty seat on the flight and two users book tickets at the same time. In this case, two threads simultaneously read the number of tickets sold from the database, check that there is still a seat left, and sell the ticket to the client. In order to avoid such collisions, locks are applied.

Optimistic and pessimistic locking are two strategies for managing concurrent access to data in a database. They differ in how they prevent or resolve conflicts and errors when multiple transactions try to modify the same data.

Optimistic:

Optimistic locking is based on the assumption that conflicts are rare and that most transactions can complete without interference. With optimistic locking, transactions do not acquire locks on the data they read or write, but instead, they check for conflicts at the end of the transaction, before committing the changes. If a conflict is detected, such as another transaction modifying the same data, the transaction is aborted and restarted. To detect conflicts, optimistic locking uses either a timestamp or a version number that is associated with each record in the database.

Advantages of Optimistic Concurrency

- Does not require locks
- Offers support to scale applications
- Can serve multiple users at once
- No deadlock situations
- Does not affect performance

Disadvantages of Optimistic Concurrency

- Requires maintenance of versions or timestamps
- Requires manual implementation of concurrency handling logic

When to Use Optimistic Concurrency

Usually, the optimistic concurrency approach is suitable for applications with fewer conflicts. So, when you do not have many data conflicts, it will reduce the number of rollbacks required and the total cost of rollbacks.

It also works well with application scaling. So, if your application has requirements to scale or serve multiple users simultaneously, optimistic concurrency is the best option.

Pessimistic :

Pessimistic locking is based on the assumption that conflicts are likely and that transactions need to protect the data they access or modify. With pessimistic locking, transactions acquire locks on the data they read or write, and hold them until they commit or rollback the changes.

This prevents other transactions from accessing or modifying the same data until the lock is released. There are different types of locks, such as shared locks, exclusive locks, or row-level locks, that determine the level of access and isolation of the transactions

The pessimistic concurrency approach has three lock modes, and users will be able to read the locked record based on the applied lock.

- **Shared**

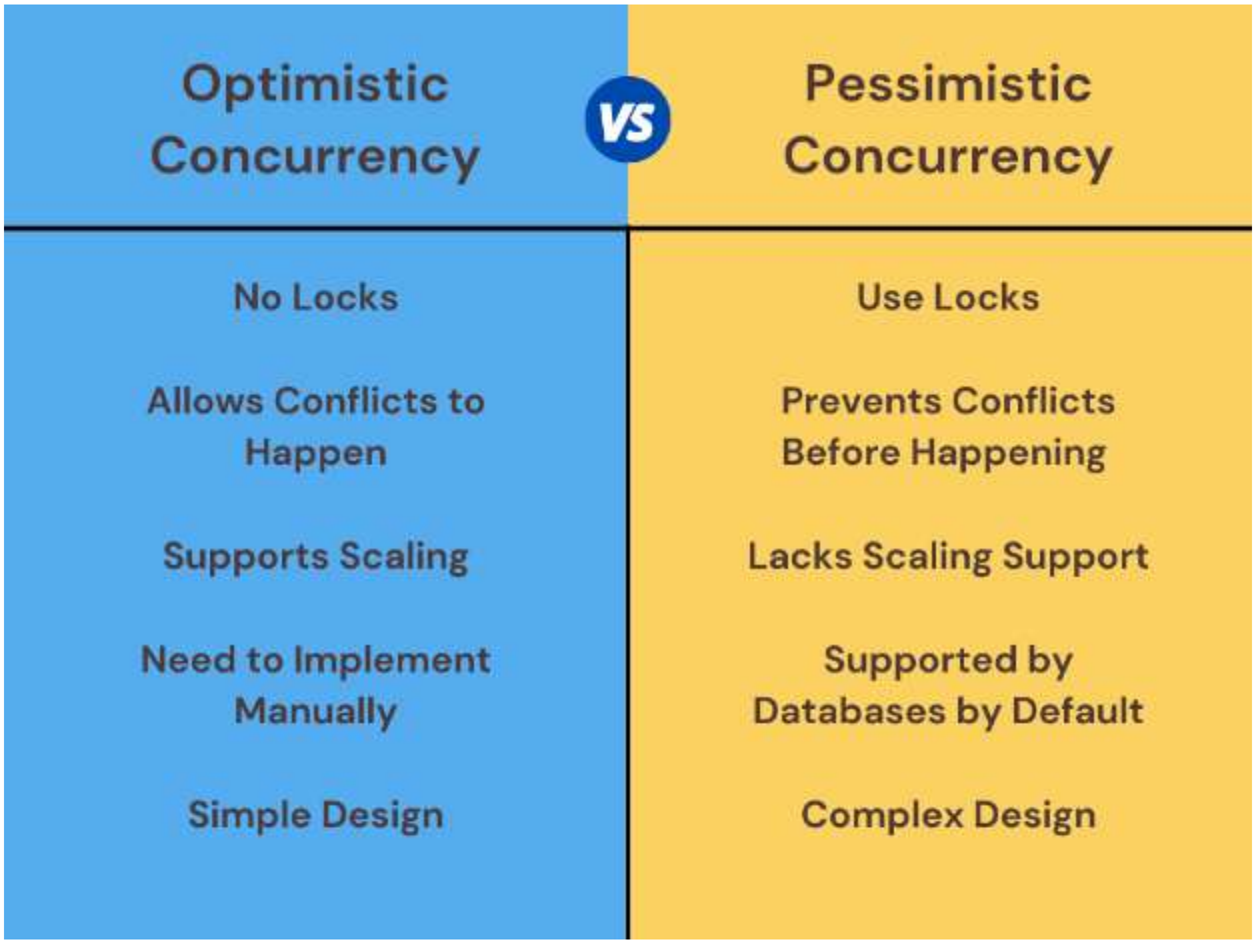
  - Allows other users to read the data record, but they can't update it.

- **Exclusive**

  - Only the user who applied the lock can read or update the data record. No other locks can be applied until the user releases the lock.

- **Update**

  - Similar to the exclusive approach, only the user who applied the lock can read or update the data record. However, users can apply update locks when another user already has a shared lock.



Optimistic Concurrency	VS	Pessimistic Concurrency
No Locks		Use Locks
Allows Conflicts to Happen		Prevents Conflicts Before Happening
Supports Scaling		Lacks Scaling Support
Need to Implement Manually		Supported by Databases by Default
Simple Design		Complex Design