
Database Locking in Spring Boot – Optimistic vs. Pessimistic

Purpose

Locking ensures **data consistency** when multiple users/processes try to update the same database record at the same time.

---

1. Optimistic Locking

What It Is:

- Assumes *no* conflicts will occur.
- Relies on a **@Version** field in the entity.
- If two updates happen concurrently, the second one **fails** if the version has changed.

How It Works:

1. Entity is fetched with a version, say v=0.
2. Another transaction fetches same entity with v=0.
3. First transaction updates and version becomes v=1.
4. Second transaction tries to update → **Version mismatch** → OptimisticLockingFailureException.

Usage in Spring Boot:

- Annotate your entity field:
- @Version
- private int version;
- Repository save() triggers version check.

Test Scenario:

- Open **two tabs or tools** like Postman.
- Send two requests to /products/optimistic/{id}:
  - One with qty=8, another with qty=18.
- If sent **at the same time**, one will fail with a version conflict.

---

2. Pessimistic Locking

What It Is:

- Assumes *conflicts will happen*.
- Acquires a **database lock** on the row so others must wait.
- Useful in **high-concurrency** scenarios.

How It Works:

- When one transaction fetches a record, others are **blocked** until it's released.
- Uses **@Lock(LockModeType.PESSIMISTIC_WRITE).**

Usage in Spring Boot:

In repository:

```
// Pessimistic lock method
   @Lock(LockModeType.PESSIMISTIC_WRITE)
   @Query("SELECT p FROM Product p WHERE p.id = :id")
   Product findByIdWithPessimisticLock(@Param("id") Long id);
```

In service:

```
@Transactional
   public void updateWithPessimisticLock(Long id, int newQty) {
        Product product =
productRepository.findByIdWithPessimisticLock(id);

        try {
           Thread.sleep(10000); // Simulate 10s DB operation
        } catch (InterruptedException e) {
           e.printStackTrace();
        }

        product.setQuantity(newQty);
        productRepository.save(product);
   }
```

Test Scenario:

1. Open **two Postman tabs**.
2. Hit /products/pessimistic/1?qty=8 in Tab 1 (waits 10s).
3. Quickly hit /products/pessimistic/1?qty=18 in Tab 2.
4. Tab 2 waits → updates after Tab 1 → **final value is 18**.

Summary Comparison:

Feature	Optimistic Locking	Pessimistic Locking
Conflict Strategy	Detects conflicts **after**	Prevents conflicts **before**
Locking Type	**No DB lock**, uses version	**DB-level row lock**
Exception on Conflict	Yes (OptimisticLockException)	No, other threads just wait
Performance	Better in low-contention	Safer in high-

		contention
Use Case	Many reads, few writes	Many concurrent writes

Real-world Examples:

- **Optimistic**: Shopping cart inventory updates in low-traffic apps.
- **Pessimistic**: Banking, ticketing systems with **high concurrency**.

Real-World Use Cases of Locking

1. Optimistic Locking – Best for Low Conflict Systems

E-commerce Inventory Management

- **Scenario**: Multiple users are shopping and may add items to cart or checkout around the same time.
- **What happens**:
  - When a user checks out, the system updates inventory (e.g., deducts 2 units of a product).
  - Optimistic locking ensures if **two users try to buy the last item**, only one will succeed.
- **Why Optimistic?**
  - Reads are frequent (browsing, cart viewing).
  - Actual writes (checkout) are less frequent.
  - We **optimistically** assume users won't conflict often.
  - If they do, one fails and retries.

---

2. Pessimistic Locking – Best for High Conflict Systems

Bank Account Transfer

- **Scenario**: Multiple services or users trying to debit the same account simultaneously.

- **What happens**:
  - A user transfers money from Account A to Account B.
  - System **locks Account A** to ensure no other transaction can withdraw at the same time.
- **Why Pessimistic?**
  - Accuracy is critical. You can't afford two withdrawals from the same balance.
  - Risk of race conditions is high.
  - So we **lock the row** until the transaction completes.

---

3. Ticket Booking System

Example: Booking concert tickets

- **With Optimistic Locking**:
  - Two users try booking the last 5 seats.
  - If both fetch seat data at same time, one's update will fail with version mismatch.
- **With Pessimistic Locking**:
  - When one user starts booking seats, the system **locks** those rows.
  - Others must wait until it's released (or get a timeout).

---

4. Employee Leave Management System

- **Optimistic**:
  - Multiple managers review leave balances.
  - One updates the balance (e.g., after approval).
  - If another also approves while balance was changed, optimistic locking will detect conflict.
- **Pessimistic**:
  - If approvals must happen in sequence, locking the row prevents concurrent approvals.

Choosing Between Them

Question	Optimistic	Pessimistic
Are updates rare but reads frequent?	Yes	Not ideal
Can a user retry after conflict?	Yes	Might not be possible
Is the cost of incorrect data high?	No	Yes
Is there high contention (many writers)?	Not common	Yes
Need to avoid deadlocks and performance issues?	Yes	Needs careful handling

---

Summary

System Type	Suggested Lock Type
Blog CMS (frequent reads, rare edits)	Optimistic
ATM Withdrawal/Banking System	Pessimistic
Online Shopping Cart	Optimistic
Railway/Flight Ticket Booking System	Pessimistic
Medical Record Updates	Pessimistic
Product Review/Commenting Systems	Optimistic
