
What is Locking in JPA?

Locking is used to **manage concurrent access** to the same data in a database — especially when multiple users or processes might try to **read or write** the same record at the same time. Without it, you might get data corruption or lost updates.

---

1. Optimistic Locking

Concept:

- Optimistic Locking **assumes** that **no one else is changing the data at the same time**.
- It allows multiple transactions to read the same data **without locking**.
- But before writing (updating), it checks whether the data was modified by someone else.

How it works:

- Add a **@Version** field (usually an integer or timestamp) to the entity.
- Every time the record is updated, the version number is incremented.
- When saving, Hibernate checks:
  - If the **version matches** → it updates.
  - If the **version has changed** → it throws an OptimisticLockException.

Best for:

- Low-contention environments (few conflicts).
- Applications where performance is important.

Example:

Two users read product id=1 with version 3.

- **User A** updates quantity → Hibernate checks version 3, matches → success → version becomes 4.
- **User B** tries to update with version 3, but DB has version 4 → fails with **OptimisticLockException**.

---

2. Pessimistic Locking

Concept:

- Pessimistic Locking **assumes conflicts are likely**.
- It **locks** the record when it is read, so no other transaction can read/write it until the lock is released.

How it works:

- Use **@Lock(LockModeType.PESSIMISTIC_WRITE)** on a query.
- The database places a **row-level lock.**
- Other users will have to **wait** or **fail** until the lock is released.

Best for:

- High-contention environments.
- When **data integrity** is more important than performance.

Example:

User A reads and locks product id=1.

- **User B tries to read/write** → must wait or gets blocked until A finishes.

Summary

Feature	Optimistic Locking	Pessimistic Locking
Locking type	No actual DB lock	Uses database-level row locks
When conflict detected	On save	On read
Performance	Faster	Slower (due to locking)
Use case	Low contention	High contention or critical data
Annotation used	@Version	@Lock(LockModeType.PESSIMISTIC_WRITE)

Which to Use?

- If your app rarely has concurrent updates → **Optimistic**.
- If your app must **strictly prevent** conflicts in real time (e.g., banking, ticket booking) → **Pessimistic**.