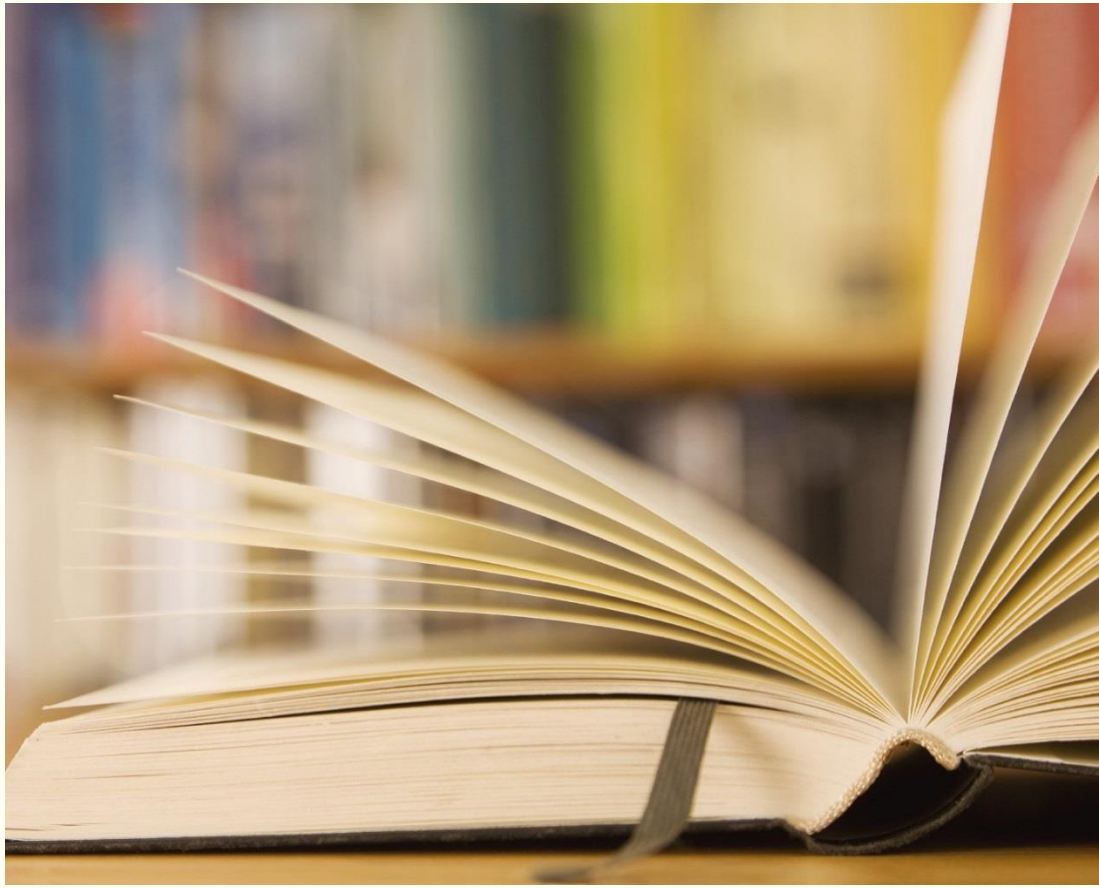
ENTERPRISE ARCHITECTURE

Najeeb Najeeb, PhD



PESSIMISTIC LOCKING

Pessimistic locking

- Hold a lock before you start working on the entity.
- Only the code that has the lock can access the entity.
- Is a synchronous operation.
- No chance for transaction failure due to concurrent changes.
- This is a choice when dealing with optimistic failure is hard.
- Limits application scalability.
- If you think you need pessimistic locking, think again.
- When concurrent writes are high, this may justify the cost of pessimistic locking.

Lock Modes

- PESSIMISTIC_READ
  - Shared lock, prevent update, or delete.
- PESSIMISTIC_WRITE
  - Exclusive lock, prevent read, update, or delete.
- PESSIMISTIC_FORCE_INCREMENT
  - Similar to PESSIMISTIC_WRITE and increments version attribute.

```
em.find(Student.class, 1l,
LockModeType.PESSIMISTIC_READ);

query.setLockMode(LockModeType.PESSIM
ISTIC_WRITE);

em.lock(student,
LockModeType.PESSIMISTIC_WRITE);

@NamedQuery(name="Student.FindAll",
query="SELECT s FROM Student s",
lockMode= PERSSIMISTIC_READ)
```

Scope and Timeout

- What do you lock beside the entity?
- PessimisticLockScope.NORMAL
  - Lock entity and its ancestors.
- PessimisticLockScope.EXTENDED
  - Lock entity and its ancestors and its relationships.

- How long to wait for a lock before throwing LockTimeoutException.
- Time in milliseconds.
- timeout = 0 "no wait" not supported by all DBs.

Main Point

- When the probability of collisions is high pessimistic locking becomes more effective. While pessimistic locking results in a performance hit, it eliminates the need for recovery from a data collision due to it never occurring.
- Maharishi talks about preventing the birth of an enemy is better than having to deal with them.