

Locking

*Nhiều vấn đề nó nằm ở tâm lý,
những cái định kiến sẽ là rào cản tâm lý.*



RONIN™
ENGINEER

3.1. Problem Context

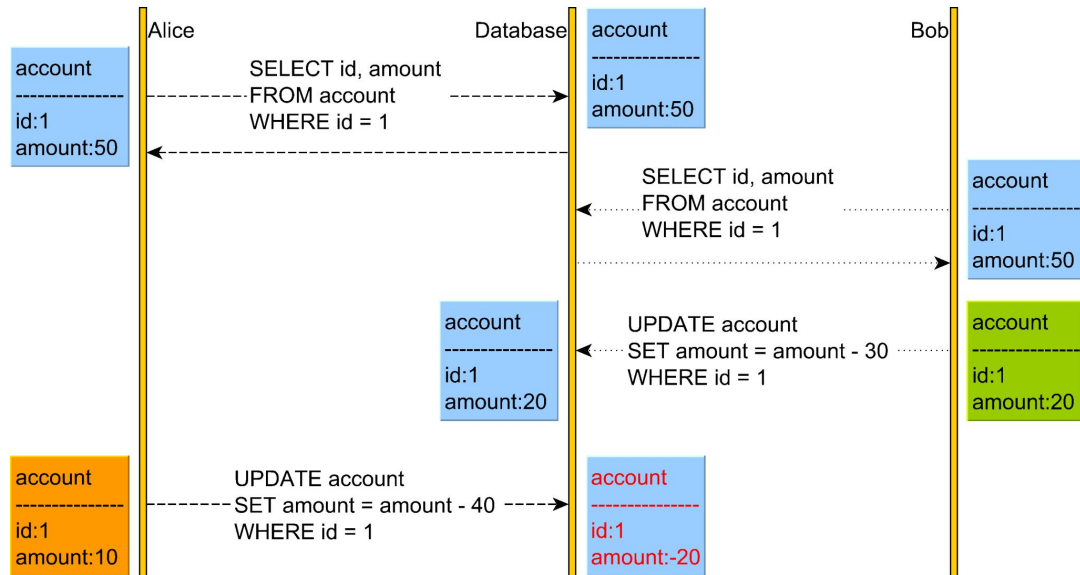
Context: Fund Transfer (write operation)

Problem: Race Condition

Ordering in Concurrency

- Balance of A: 50
- 2 requests at the same time
 - A → B: 30
 - A → C: 40
- $A: 50 - 30 = 20 \rightarrow 20 - 40 = -20$ (X)
- Valid balance of A ≥ 0





→ Locking



3.2.1. Types of Locks in MySQL

Classified based on operation-level:

- **Shared (Read):** Allow to read but lock to write
- **Exclusive (Update):** Lock to read and write

	Shared (S)	Exclusive (X)
Shared (S)		
Exclusive (X)		

Example 1:

1. Transaction A locks row X for read (S)
2. Transaction B can lock row X for read (S)

Example 2:

1. Transaction A locks row X for read (S)
2. Transaction B can't lock row X for update (X)

Example 3:

1. Transaction A locks row X for update (X)
2. Transaction B can't lock row X for read (S)

Question: What is operation type of lock used to solve the problem?

Question: What is operation type of lock
used to solve the problem?

Answer: Exclusive Lock (SELECT ... FOR UPDATE)

3.2.2. Types of Locks in MySQL

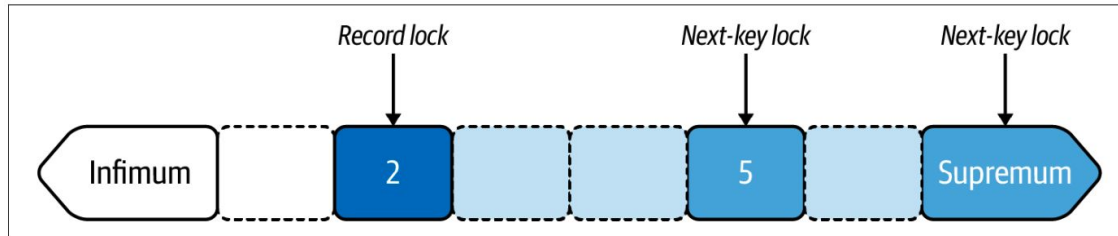
Classified based on row-level (MySQL):

- Global Lock
- Table Lock
 - Lock table. Example: lock tables <table_name> read
 - Intention Lock
 - AUTO-INC lock is a special table lock mechanism
 - Is improved in new versions
- **Row Level**
 - Record lock
 - Gap Lock
 - Next-Key Lock
 - Insert Intention Lock

3.3. Row Lock

Lock	Abbreviation	Locks gap	Description
Record lock	REC_NOT_GAP		Locks a single record
Gaplock	GAP	✓	Locks the gap before (less than) a record
Next-key lock		✓	Locks a single record and the gap before it
Insert intention lock	INSERT_INTENTION		Allows INSERT into gap

update account_balance set balance = balance + 10
where account_number between 2 and 5;



Question: What is type of row lock used to solve the problem?

Question: What is type of row lock
used to solve the problem?

Answer: Record lock
- locking a single record at a time.

3.4. Query to show lock info

```
SELECT thread_id, index_name, lock_type, lock_mode, lock_status, lock_data  
FROM   performance_schema.data_locks  
WHERE  object_name = '<table_name>';
```

3.6. Go on; Save Yourself

Why it locks redundant rows?

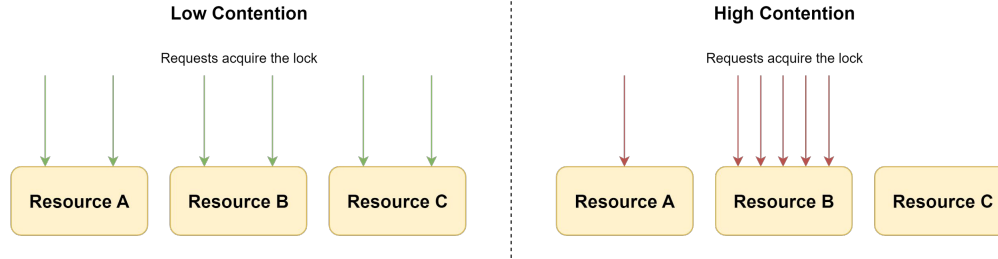
I hope that, as an engineer using MySQL, you never need to descend to this depth of InnoDB locking to achieve remarkable performance with MySQL. But I led us down here for two reasons. First, despite the illusions, the fundamentals of InnoDB row locking with respect to transaction isolation levels are tractable and applicable. You are now fantastically well prepared to handle every common InnoDB row locking issue—and more. Second, InnoDB made me do it because I stared too deeply for too long; and when it all blurred into one, I knew that I had fallen from the precipice and could never return. Don't ask why it locks the supremum pseudo-record beyond the table condition range. Don't ask why it has redundant gap locks. Don't ask why it converts implicit locks. Don't ask; else the questions never cease. Go on; save yourself.

[Efficient MySQL Performance - Daniel Nichter](#)

This is blackbox!

Brief answer: to ensure Repeatable Read isolation level

3.7. Lock Contention



3.8. Pessimistic Locking vs Optimistic Locking

(classified by implementation)

3.8.1. Pessimistic Locking

- Pessimistic Locking avoids conflicts by **using locking**
- Example:
 - `SELECT ... FOR SHARE;`
 - `SELECT ... FOR UPDATE;`
 - Keyword synchronized (heavy weight)
- Pros:
 - Data Integrity
- Cons:
 - **Heavyweight Lock: Concurrency bottlenecks in low contention**
 - Potential Deadlocks
- Use case: high concurrency + high contention



3.8.2. Optimistic Locking

- **No locks**, allows a conflict to occur, but it needs to **detect, resolve it at write time**.
- Implementation:
 - **Compare And Swap (CAS)**
 - **Version**
 - **Timestamp** (not recommended)



3.8.2. Optimistic Locking

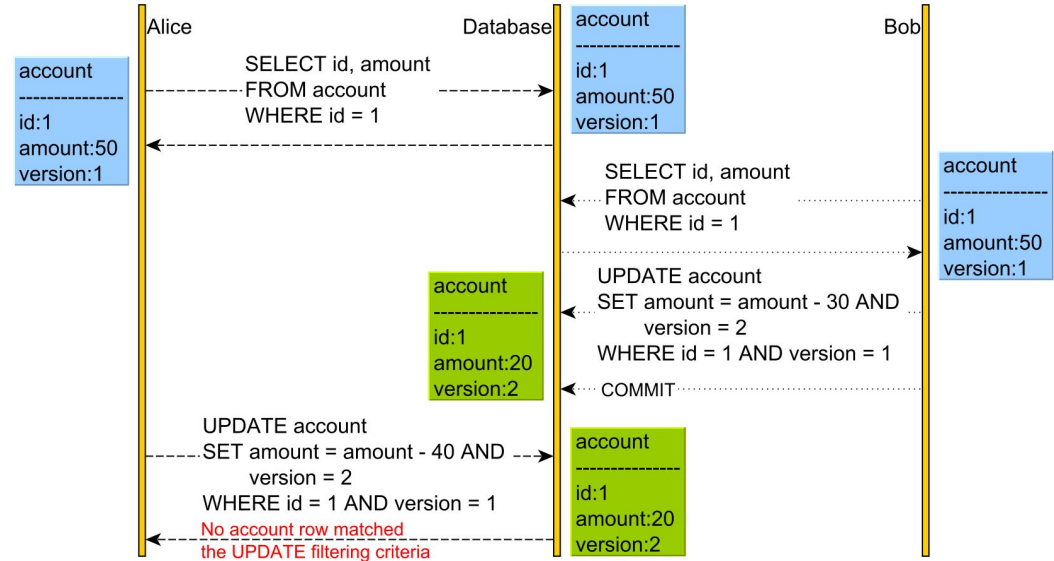
- Pros:
 - **Lightweight Lock:** Reduced Lock Overhead
→ Higher Concurrency
 - Avoid Deadlocks
- Cons:
 - **Risk of Conflicts.** If handling multiple requests concurrently, then **just one request will be processed successfully, others will be rejected.**
 - Conflict Resolution may be complex.
- **Use case: High concurrency + low contention**



4. Practices

4.1. Implement Optimistic Locking

- Add **version** column
- SELECT to get current info
- Update and increase version by 1 (version++)
- **SET amount = <new_value>, version = <new_version> WHERE id = <id> AND version = <old_version>**
- **Note: this acquires a lock for a row while updating the row only**



4.2. Notes

- Transaction and Locking are 2 different mechanism solving 2 different problems
- DB has its own locking mechanism != programming language's locking mechanism.
- ...

4.3. Best Practices

- **Subtract first, add later**
- Use record lock by leveraging primary key
- Avoid to use secondary index in UPDATE
- Examine row locks
- **Leverage Optimistic Locking**
- **Combine Distributed Locking + Optimistic Locking**
- ...

Recap

- When mention to ordering, remember queue and locking.

Locking solve the race condition problem.

- Low contention vs High contention.
- Pessimistic Locking are quite costly.

Leverage Optimistic Locking.

Homework

- Implement flight booking
 - Logic
 - Create a booking
 - Decrease the number of available seats in a flight
 - Using locking mechanisms
 - Basic:
 - Pessimistic Locking Only in DB
 - Advanced: combine
 - Optimistic Locking (version)
 - Distributed Lock
- **4 conditions for deadlock → simple way to avoid deadlock.**



References

- <https://blog.lawrencejones.dev/state-machines/index.html>
- <https://vladmihalcea.com/optimistic-vs-pessimistic-locking/>
- <https://faculty.kutztown.edu/schwesin/spring2022/csc343/lectures/Locking.pdf>

Thank you 🙏

