Database transaction concurrency control

locks, Isolation level, MVCC

10/01/2023

Concepts

Locks

- S(shared) lock, X(exclusive) lock
- Pessimistic lock, Optimistic lock
- table lock, row lock, page lock, record lock, gap lock, next-key lock

Isolation levels: Read uncommitted, Read Committed, Repeatable read, Serializable

And these names: Dirty Read/Write, Phantom Read, Non-repeatable Read

Also this concept: MVCC

Be pessimistic or optimistic

This is an idea, not an application.

Pessimistic lock, the idea is that before I want to touch this system resource (not have to be database records), it could be a file or network port, I want to **assume** that everyone else who uses this resource will affect my work. so I want to lock them to prevent others from making changes.

Optimistic lock is another way, which means I am okay to share the resource with other processes who need it, as long as not affecting my job, when conflict happens we can resolve it on the spot.

Shared/Lock or Exclusive

Just remember that **Shared** is for reading, **lock** it for writing, meaning that I allow another process to read this resource like I am doing, and let’s prevent whatever process from writing it.

Exclusive lock is easy to understand: I am using it, locking it, and no one else should be able to access(read or write) it.

Problems - Concurrency transaction

Dirty Read vs Phantom Read vs Non Repeatable Read

1) https://jennyttt.medium.com/dirty-read-non-repeatable-read-and-phantom-read-bd75dd69d03a
2) https://cloud.tencent.com/developer/article/1450773

Solutions — Isolation levels

Isolation Level	Dirty Read/Write	Non-Repeatable Reads	Phantom Reads
Read un-committed Isolation level	EXISTS	EXISTS	EXISTS
Read Committed isolation level	SOLVED	EXISTS	EXISTS
Repeatable Reads Isolation level	SOLVED	SOLVED	EXISTS
Serializable Isolation level	SOLVED	SOLVED	SOLVED

Isolation level=RC (Read committed)

the core ideas only 3 points:

- Make sure to only read committed data
- Lock whatever VISIBLE records when update

Note, only visible records, not anything else, by visible records, it means 2 conditions:

- The records exist in the database
- The records matched the “where” condition

e.g. `select * from the table where id > 2`
let’s say now in the database there are 3 records (1,2,4).

- visible record is 4 (3 is not included because it doesn't exist in the database)

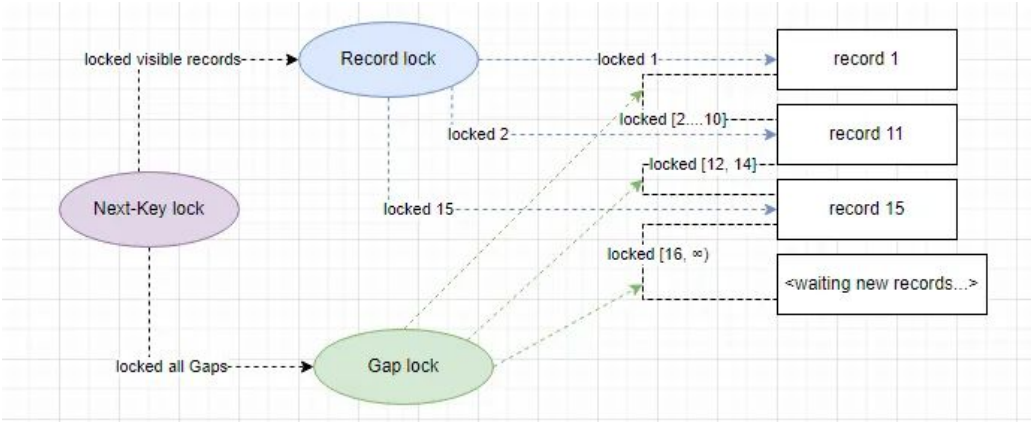
Isolation level=RR (Repeatable read)

when we use the RR (Repeatable read) Isolation level (which is the default in MySQL), the “gaps” are being locked (not just the visible records) as well.

In other words, a quick comparison RC VS RR:

- RC: Lock visible records in the database
- RR: Lock “Whatever records matched where condition” which could affect this transaction

Record lock and Gap lock



- Record lock: lock rows exist in the database;
- gap lock: lock “gaps” including infinity;
- Next-Key lock=record lock + gap lock

MVCC(Multi-version concurrency control)

https://devcenter.heroku.com/articles/postgresql-concurrency

https://vladmihalcea.com/how-does-mvcc-multi-version-concurrency-control-work/

MVCC vs Lock

Since we already have MVCC without any locking, why do you even want to have record-lock, and gaps lock again? MVCC can handle all cases with high performance.

No. To be “100% safe”, we still need next-key locks. example phantom-read (t1 select, t2 insert, t3 select again). again, MVCC can only “see” the database records and versioning the snapshots. we need to make sure “where condition matched records” are all safe, locks still need to be there.

Recap

Read Uncommitted: Dirty and problematic. don’t use.

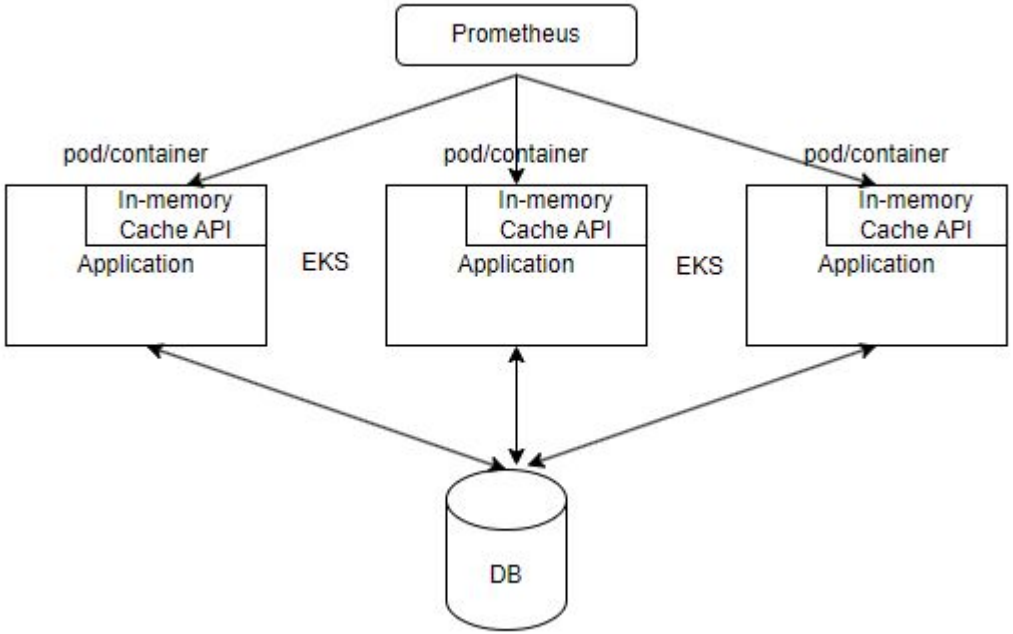
RC: only using record lock. lock visible records (and that's why it doesn't completely solve phantom read)

RR: use next-key lock=record lock+gaps lock. locks records matched with condition (even if it doesn’t exist in the database yet, as long as potentially could affect this transaction)

Serializable: Never experienced a use case about this. maybe in banking, there are some cases that can not be solved in RR. pls, comment if you know.

MVCC: maintain multiple snapshots of records using read-view to archive transaction concurrency safety.

Senario: How to sync in-memory cache



Senario: Create ElastiCache Service

Senario: Maintain Redis Pod

Senario: Use DB as sync center and sticky sessions