Dev notes

Notes about Android, Java, JavaScript, Go and other technical stuff

Home
GitHub
Google+
RSS

Tags:

- android
- angularjs
- ansible
- concurrency
- database
- devops
- docker
- git
- golang
- hugo
- java
- javascript
- linux
- python
- react
- windows

Linearizability, serializability, transaction isolation and consistency models

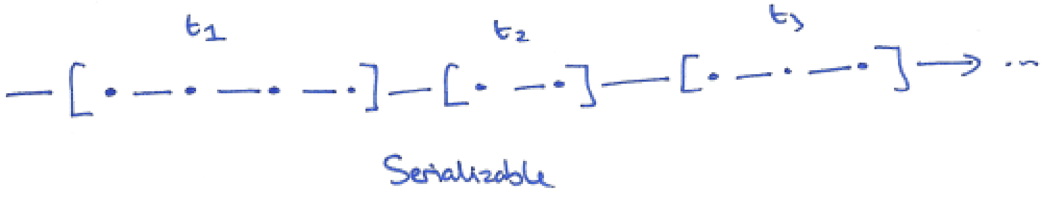
2016-03-17

Linearizability versus Serializability:

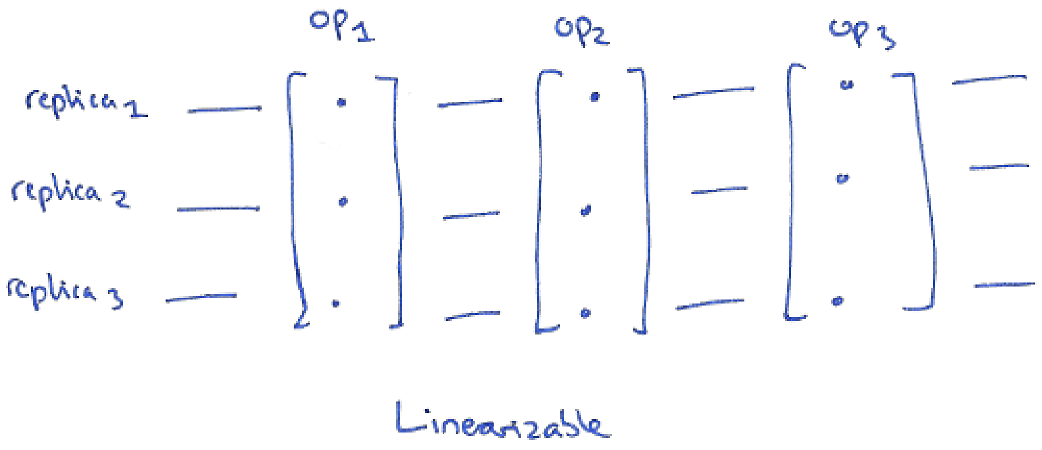
> **Linearizability** is a guarantee about single operations on single objects. It provides a real-time (i.e., wall-clock) guarantee on the behavior of a set of single operations (often reads and writes) on a single object (e.g., distributed register or data item).
>
> Linearizability for read and write operations is synonymous with the term "atomic consistency" and is the "C," or "consistency," in Gilbert and Lynch's proof of the CAP Theorem. We say linearizability is composable (or "local") because, if operations on each object in a system are linearizable, then all operations in the system are linearizable.
>
> **Serializability** is a guarantee about transactions, or groups of one or more operations over one or more objects. It guarantees that the execution of a set of transactions (usually containing read and write operations) over multiple items is equivalent to some serial execution (total ordering) of the transactions.
>
> Serializability is the traditional "I," or isolation, in ACID. If users' transactions each preserve application correctness ("C," or consistency, in ACID), a serializable execution also preserves correctness. Therefore, serializability is a mechanism for guaranteeing database correctness.

Distributed Consistency and Session Anomalies:

> In the database systems community, the gold standard is serializability. We've spent plenty of time looking at this in the last couple of days. Serializability concerns transactions that group multiple operations across potentially multiple objects. A serializable schedule is one that corresponds to some ordering of the transactions such that they happen one after the other in time (no concurrent / overlapping transactions). It's the highest form of isolation between transactions.



> In the distributed systems community, the gold standard is linearizability. Linearizability concerns single operations on single objects. A linearizable schedule is one where each operation appears to happen atomically at a single point in time. Once a write completes, all later reads (wall-clock time) should see the value of that write or the value of a later write. In a distributed context, we may have multiple replicas of an object's state, and in a linearizable schedule it is as if they were all updated at once at a single point in time.



> Also, small correction: the "C" in ACID is the *assumption* that the application program is correct (each transaction in isolation brings the database from a correct state to another correct state), whereas the AID properties are *guarantees* provided by the database. Together they imply serializability.

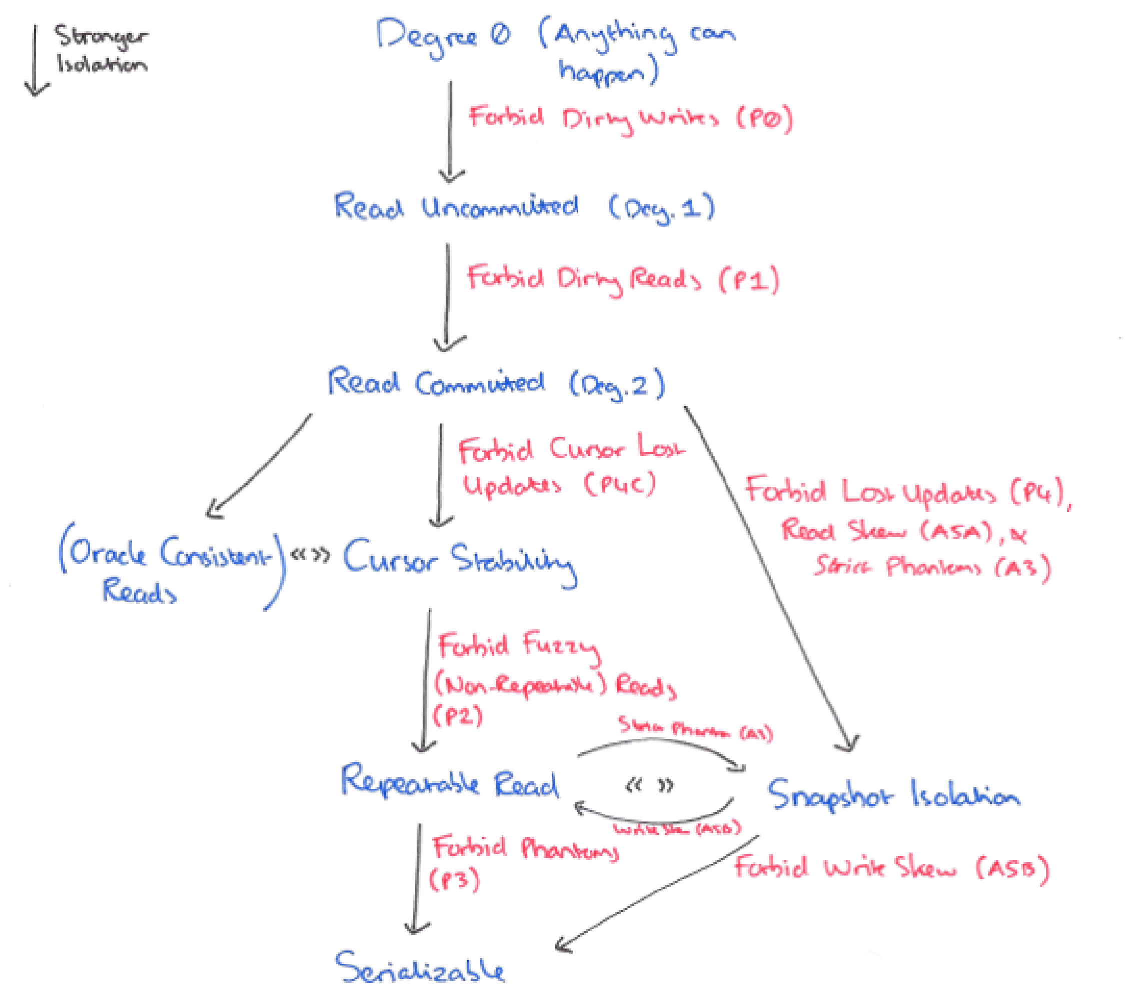
---

https://en.wikipedia.org/wiki/Consistency_model:

> **Strict consistency** is the strongest consistency model. It requires that if a process reads any memory location, the value returned by

> the read operation is the value written by the most recent write operation to that location.
>
> The **sequential consistency** model as defined by Lamport(1979) is a weaker memory model than strict consistency.
>
> **Linearizability** (also known as atomic consistency) can be defined as sequential consistency with the real-time constraint.
>
> **Causal consistency** can be considered a weakening model of sequential consistency by categorizing events into those causally related and those that are not. It defines that only write operations that are causally related must be seen in the same order by all processes.

A Critique of ANSI SQL Isolation Levels:


Stronger Isolation
Degree 0 (Anything can happen)
Forbid Dirty Writes (P0)
Read Uncommitted (Deg. 1)
Forbid Dirty Reads (P1)
Read Committed (Deg. 2)
Forbid Cursor Lost Updates (P4C)
Forbid Lost Updates (P4), Read Skew (A5A), & Strict Phantoms (A3)
(Oracle Consistent Reads) «» Cursor Stability
Forbid Fuzzy (Non-Repeatable) Reads (P2)
Strict Phantom (A3)
Repeatable Read « » Snapshot Isolation
Write Skew (A5B)
Forbid Phantoms (P3)
Forbid Write Skew (A5B)
Serializable

Other articles:

- Strong consistency models
- A beginner’s guide to database locking and the lost update phenomena
- Generalized Isolation Level Definitions
- PostgreSQL - 13.2. Transaction Isolation
- Забудьте CAP теорему как более не актуальную

Tags:
- Database
- Concurrency

0 Комментариев **dddpaul-github** 1 **Войти**

Рекомендовать **Поделиться** **Лучшее в начале**

Начать обсуждение...

ВОЙТИ С ПОМОЩЬЮ **ИЛИ ЧЕРЕЗ DISQUS** ?

Имя

Прокомментируйте первым.

ТАКЖЕ НА **DDDPAUL-GITHUB**

Дистанционное управление Motion с помощью Go

1 комментарий • 3 года назад

Pavel Derendyaev — Теперь осталось сделать ночной режим :)

Ansible: "defined" keyword

2 комментариев • 2 года назад

Pavel Derendyaev — Pavel, you are totally right. I've updated my post. Thanks.

Подписаться **Добавь Disqus на свой сайтДобавить DisqusДобавить**

Конфиденциальность