Database Transaction Isolation and Locking in Java

Kanstantsin Slisenka, Senior Software Engineer
EPAM Systems

Aug 25, 2016

About me

kanstantsin_slisenka@epam.com

github.com/kslisenko

www.slisenko.net

Kanstantsin Slisenka

EXPERIENCE

- Java backend developer
- Oracle Certified Java 7 Programmer, OCEWCD, OCEJPAD
- Speaker at Java tech talks

INTERESTED IN

- Java backend, SOA, databases, integration frameworks
- Solution architecture
- High load, fault-tolerant, distributed, scalable systems

DID YOU USE TX ISOLATION OR LOCKING AT YOUR PROJECTS?

Agenda

- Transaction phenomena and isolation levels
- Pessimistic and optimistic approaches
- Transaction isolation in MySQL
- Database-level locks in MySQL
- JPA features for locking

WHAT IS DATABASE TRANSACTION?

Database transactions and ACID

1. Atomicity
2. Consistency
3. Isolation
4. Durability

Transaction phenomena

PROBLEM

- Problem of concurrent updates made by parallel transactions
- No problem if no concurrent updates
- Databases have protection

PHENOMENA

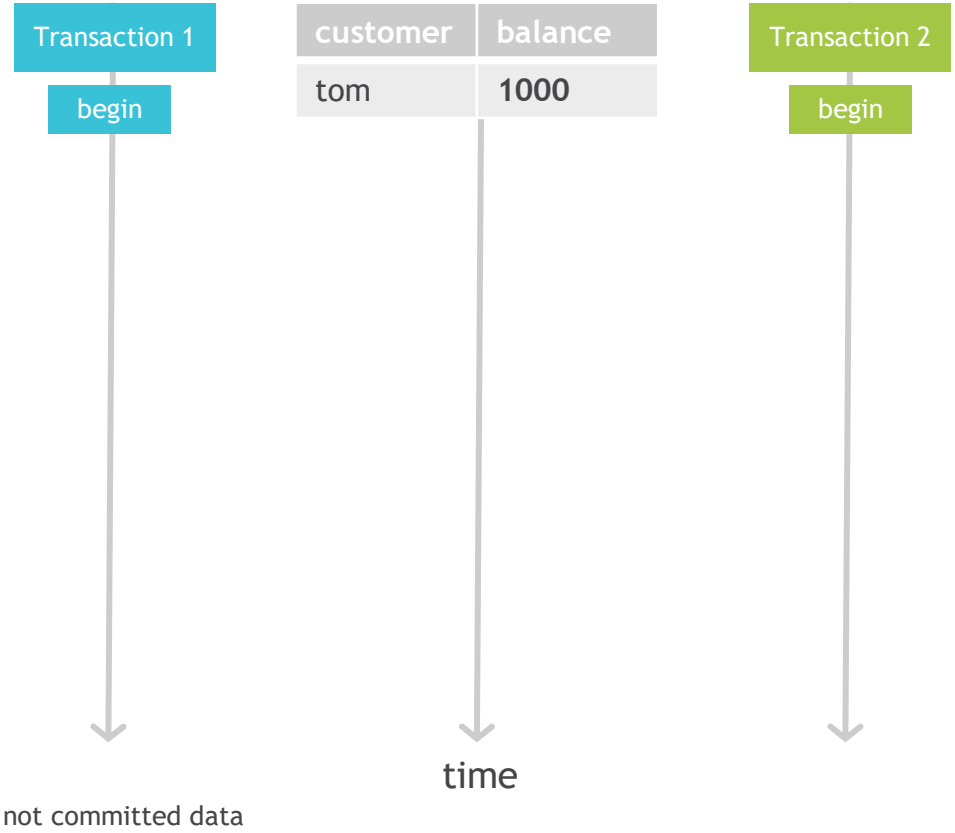
- Dirty read
- Non-repeatable read
- Phantom insert

Transaction phenomena: dirty read

PROBLEM

- Transactions can read not committed (dirty) data of each other
- Other transaction rollbacks, decision was made based on never existed data

DATABASES ARE PROTECTED AGAINST THIS IN REAL LIFE

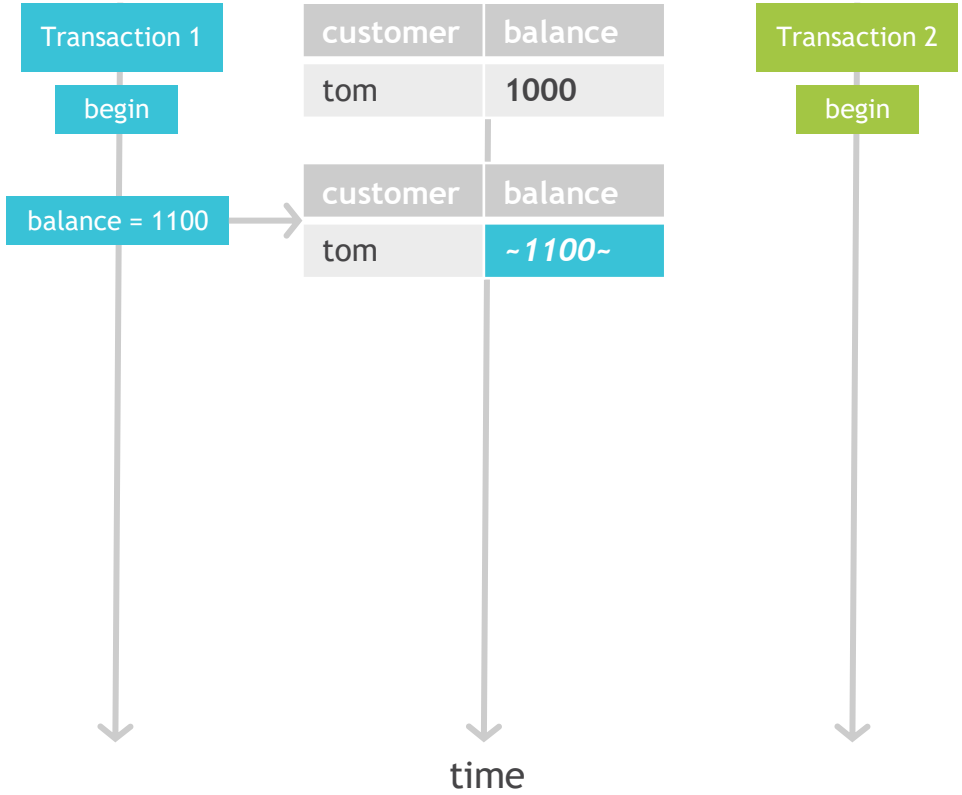


Transaction phenomena: dirty read

PROBLEM

- Transactions can read not committed (dirty) data of each other
- Other transaction rollbacks, decision was made based on never existed data

DATABASES ARE PROTECTED AGAINST THIS IN REAL LIFE



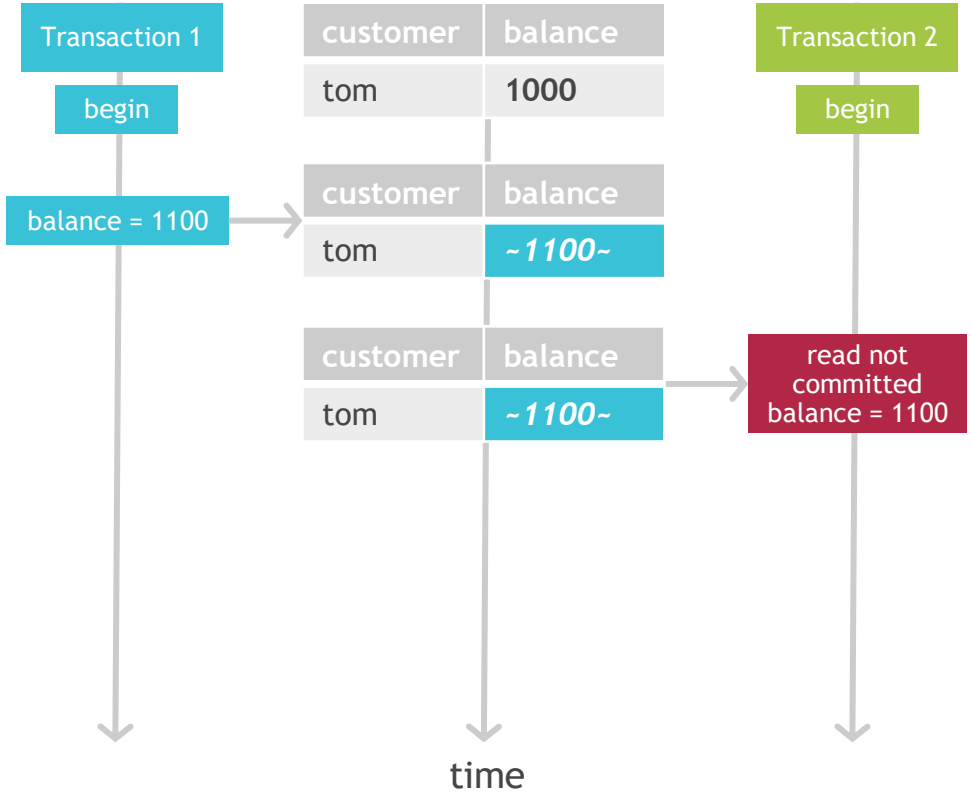
~ not committed data

Transaction phenomena: dirty read

PROBLEM

- Transactions can read not committed (dirty) data of each other
- Other transaction rollbacks, decision was made based on never existed data

DATABASES ARE PROTECTED AGAINST THIS IN REAL LIFE



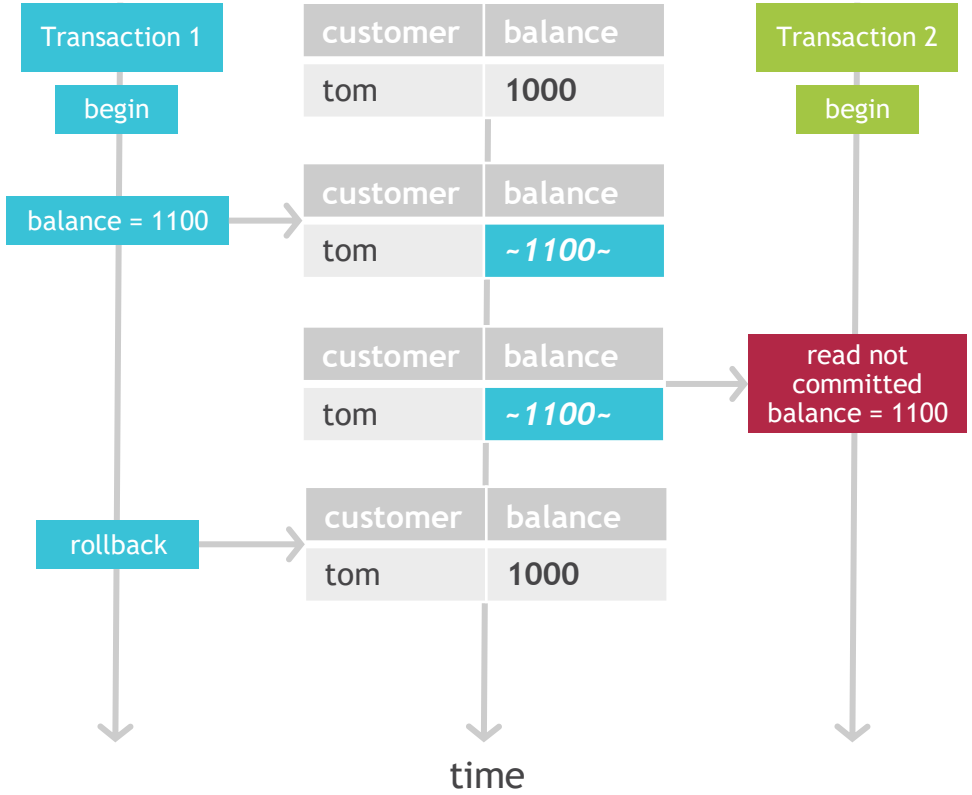
~ not committed data

Transaction phenomena: dirty read

PROBLEM

- Transactions can read not committed (dirty) data of each other
- Other transaction rollbacks, decision was made based on never existed data

DATABASES ARE PROTECTED AGAINST THIS IN REAL LIFE



~ not committed data

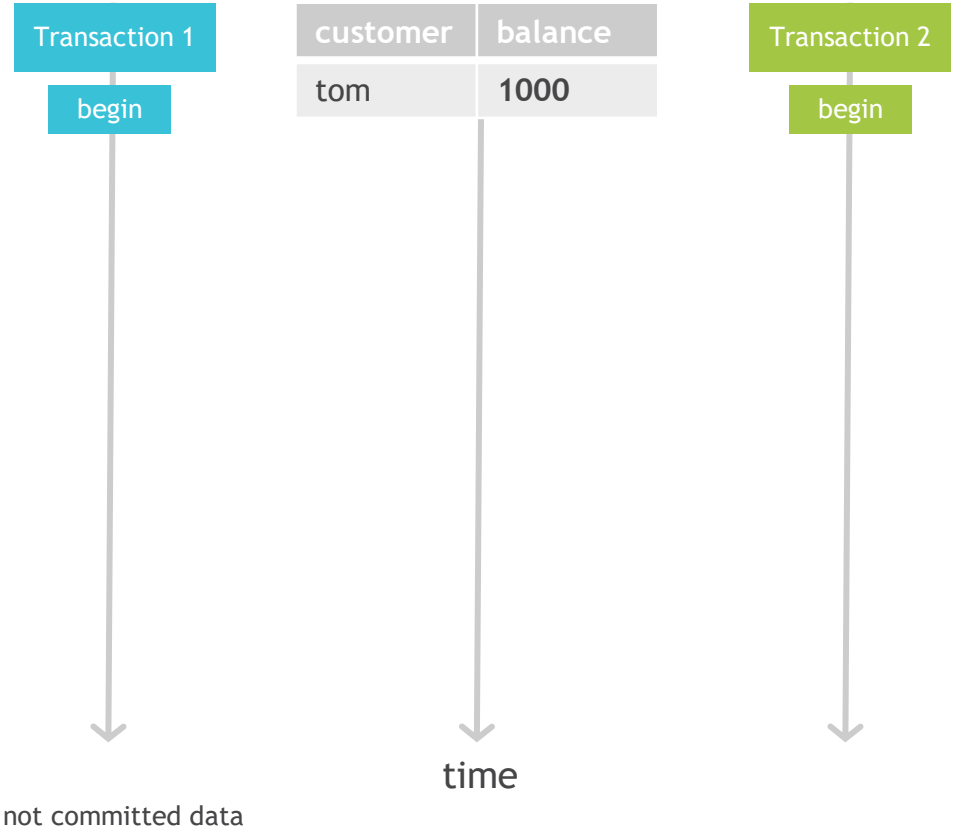
Transaction phenomena: non-repeatable read

PROBLEM

- One transaction updates data
- Other transaction reads data several times and get different results

WHEN WE CAN LIVE WITH THIS

- We are fine with not the most recent data



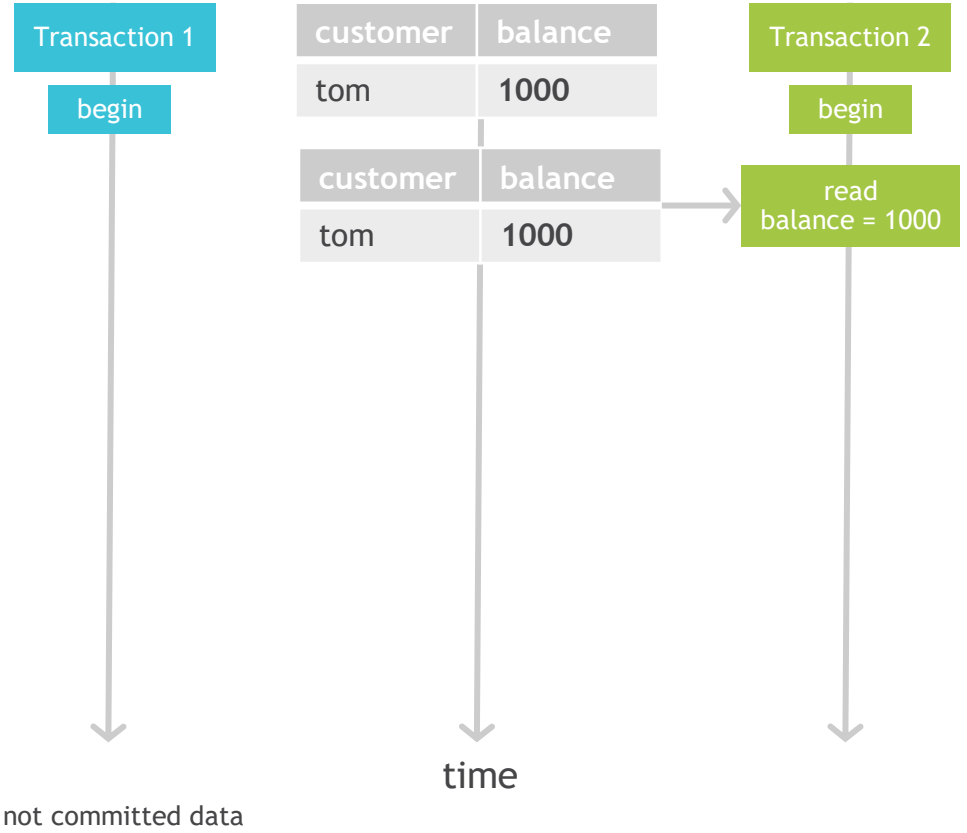
Transaction phenomena: non-repeatable read

PROBLEM

- One transaction updates data
- Other transaction reads data several times and get different results

WHEN WE CAN LIVE WITH THIS

- We are fine with not the most recent data



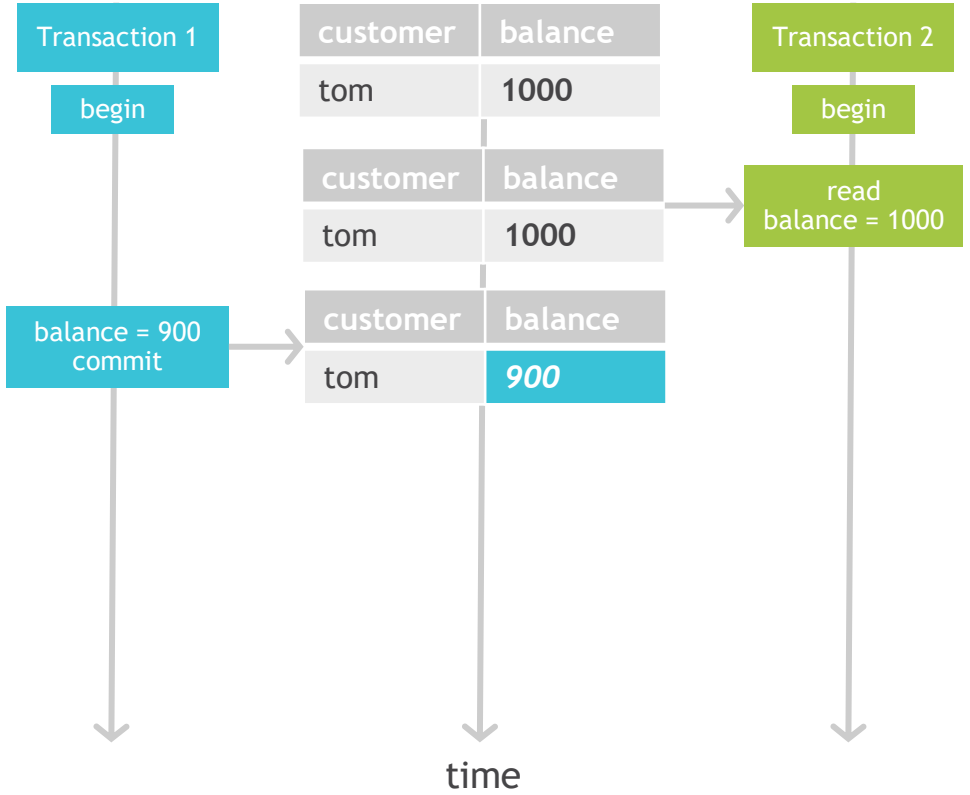
Transaction phenomena: non-repeatable read

PROBLEM

- One transaction updates data
- Other transaction reads data several times and get different results

WHEN WE CAN LIVE WITH THIS

- We are fine with not the most recent data



~ not committed data

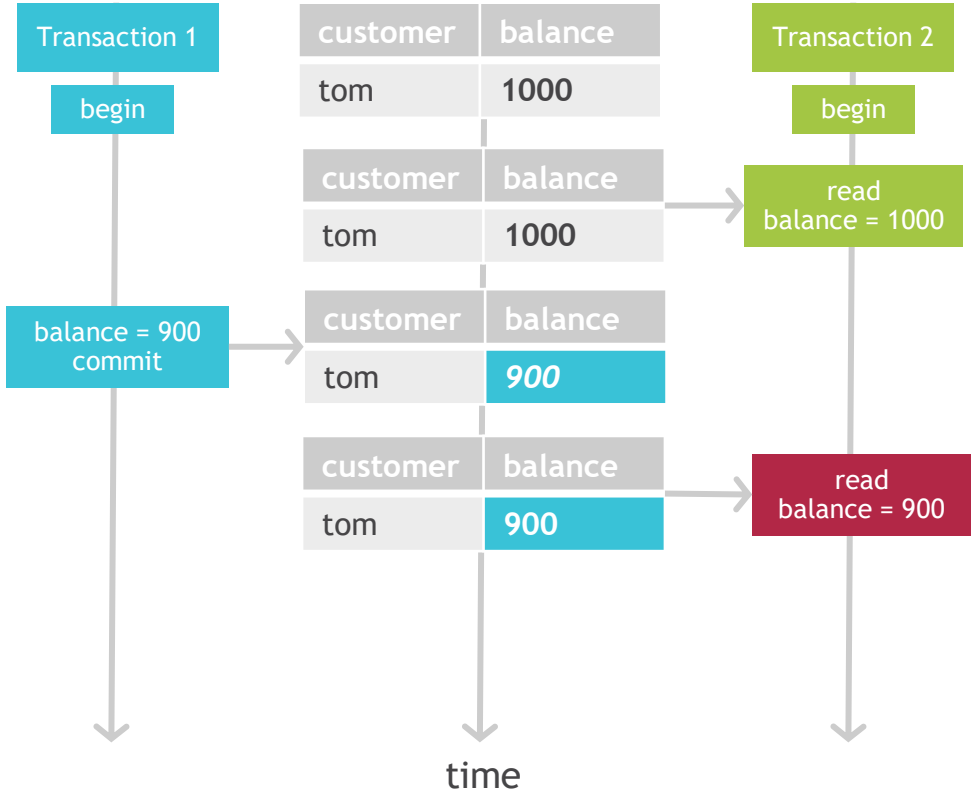
Transaction phenomena: non-repeatable read

PROBLEM

- One transaction updates data
- Other transaction reads data several times and get different results

WHEN WE CAN LIVE WITH THIS

- We are fine with not the most recent data



~ not committed data

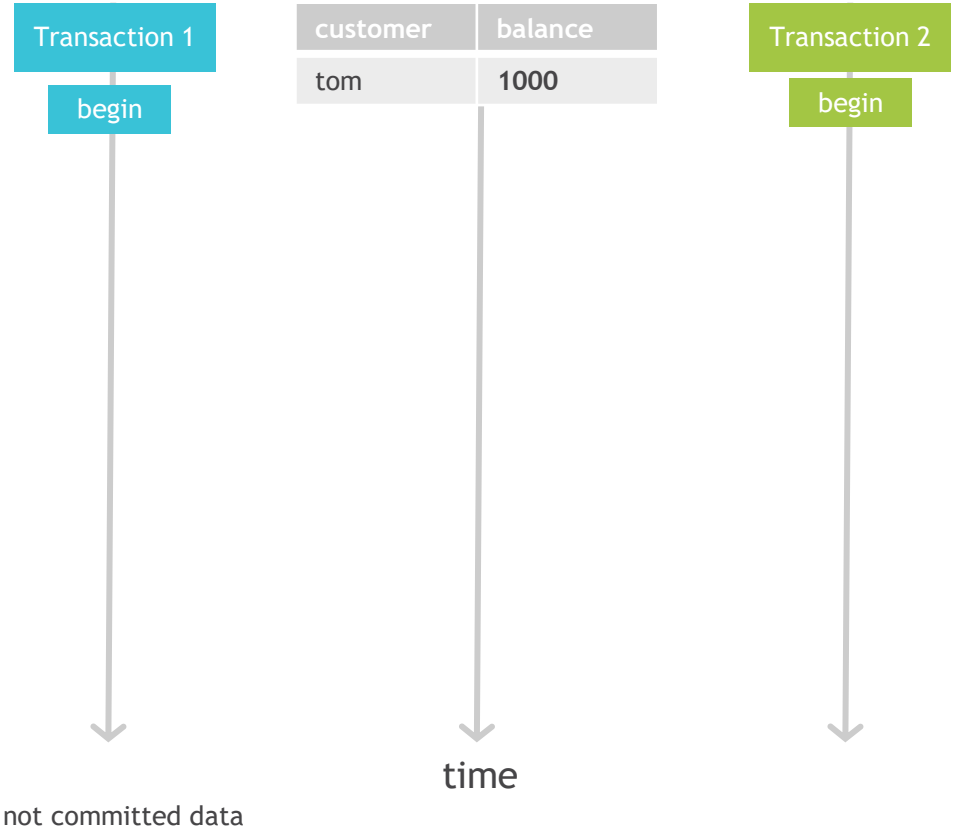
Transaction phenomena: phantom

PROBLEM

- One transaction inserts/deletes rows
- Other transaction reads several times and get different number of rows

WHEN WE CAN LIVE WITH THIS

- Read single rows, not ranges
- We are fine with not the most recent data



~ not committed data

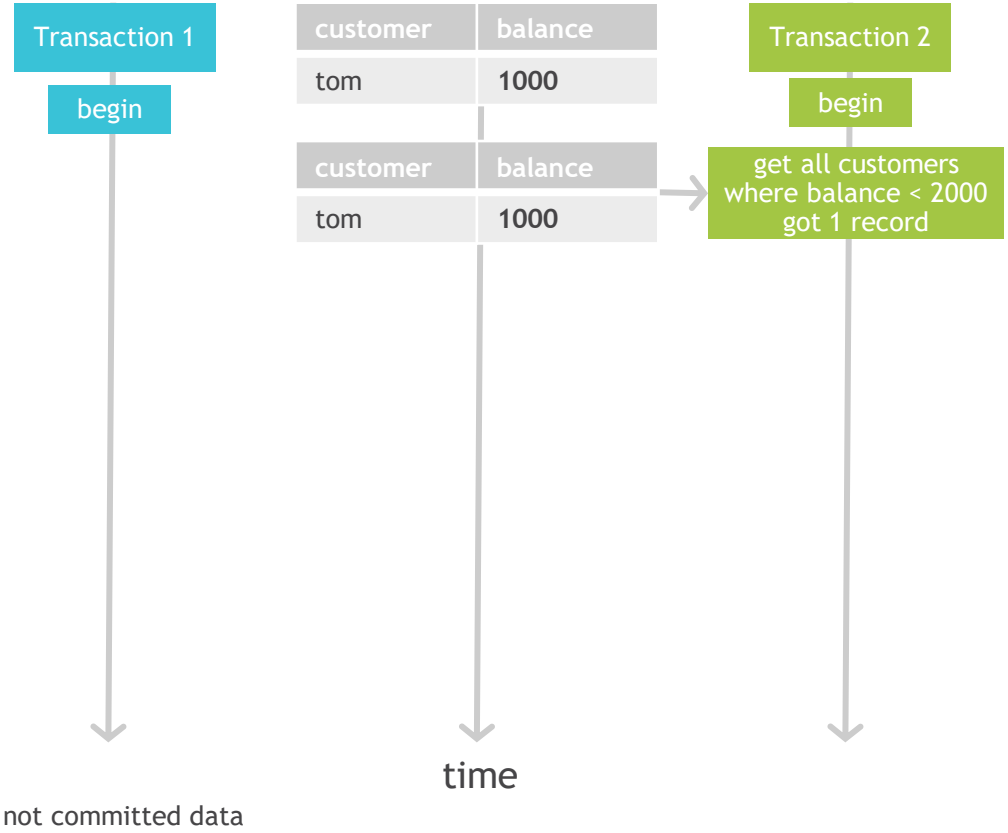
Transaction phenomena: phantom

PROBLEM

- One transaction inserts/deletes rows
- Other transaction reads several times and get different number of rows

WHEN WE CAN LIVE WITH THIS

- Read single rows, not ranges
- We are fine with not the most recent data



~ not committed data

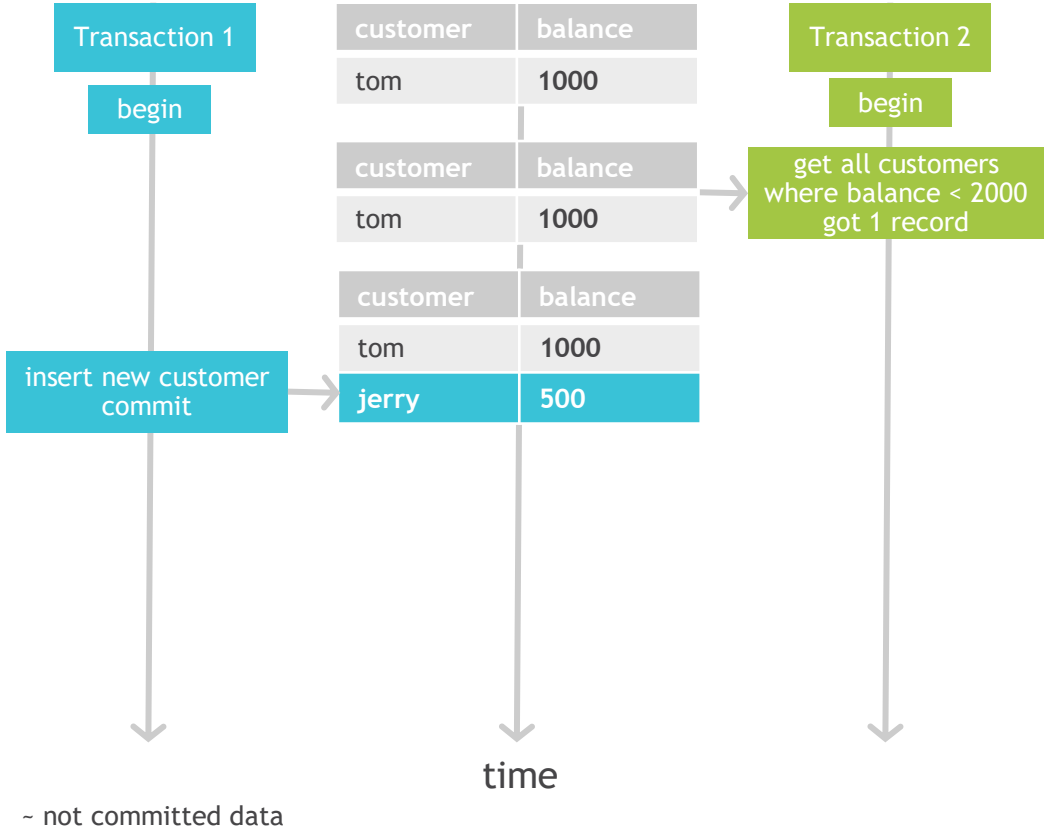
Transaction phenomena: phantom

PROBLEM

- One transaction inserts/deletes rows
- Other transaction reads several times and get different number of rows

WHEN WE CAN LIVE WITH THIS

- Read single rows, not ranges
- We are fine with not the most recent data



~ not committed data

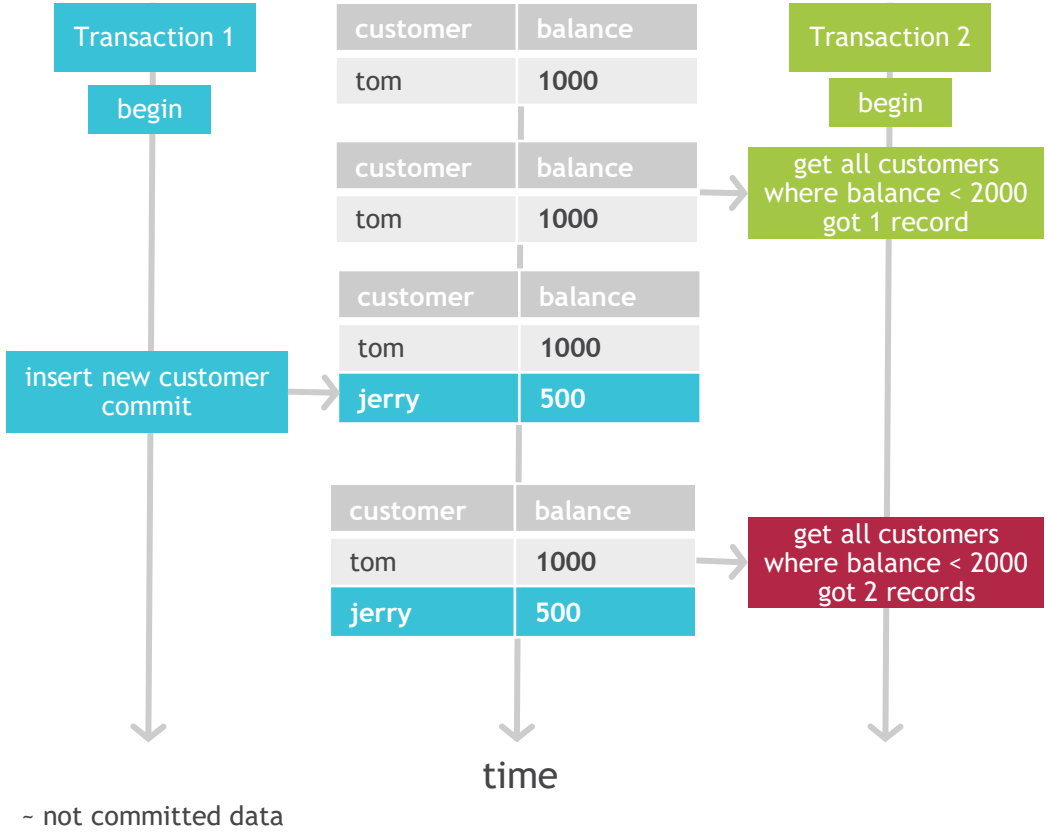
Transaction phenomena: phantom

PROBLEM

- One transaction inserts/deletes rows
- Other transaction reads several times and get different number of rows

WHEN WE CAN LIVE WITH THIS

- Read single rows, not ranges
- We are fine with not the most recent data



~ not committed data

HOW TO PROTECT?

Transaction isolation levels (standard)

	READ UNCOMMITED	READ COMMITED	REPEATABLE READ	SERIALIZABLE
Dirty read	YES	NO	NO	NO
Non-repeatable read	YES	YES	NO	NO
Phantom	YES	YES	YES	NO

- Defined in SQL92
- Trade-off between performance, scalability and data protection
- Same work performed with the same inputs may result in different answers, depending on isolation level
- **Implementation can be VERY DIFFERNT in different databases**

- ISO/IEC 9075:1992
- Information technology -- Database languages -- SQL

http://www.contrib.andrew.cmu.edu/~shadow/sql/sql1992.txt

HOW DOES IT WORK?

IN DIFFERENT DATABASES

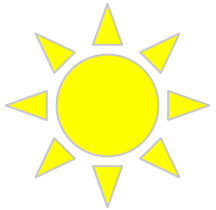
Optimistic and pessimistic approaches

PESSIMISTIC

- Locking rows or ranges
- Like ReadWriteLock/synchronized in Java
- Concurrent transactions wait until lock is released

OPTIMISTIC

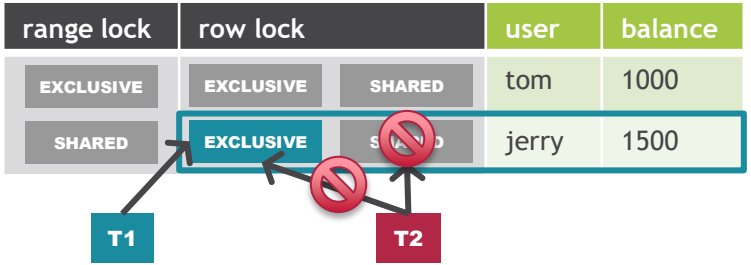
- Multi version concurrency control (MVCC)
- Doesn't lock anything
- Save all versions of data
- We work with data snapshots
- Like GIT/SVN

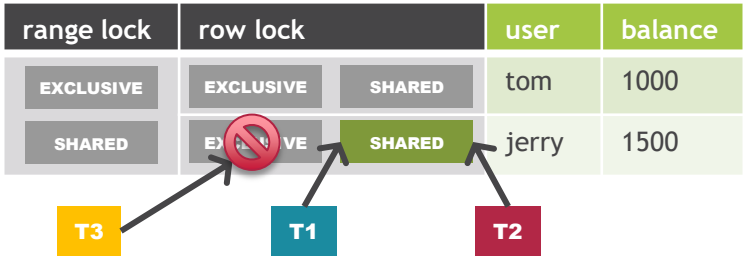


Pessimistic locking

BY OWNERSHIP

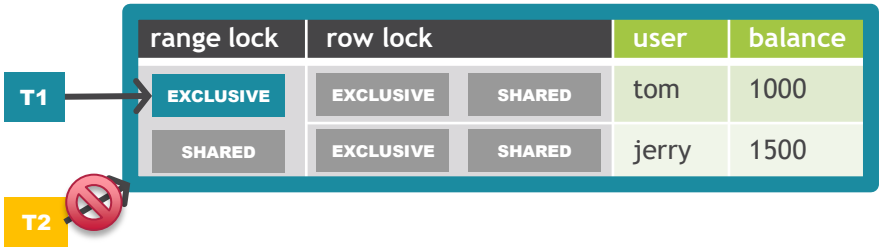
- Shared lock – read lock, many owners
- Exclusive lock – write lock, one owner

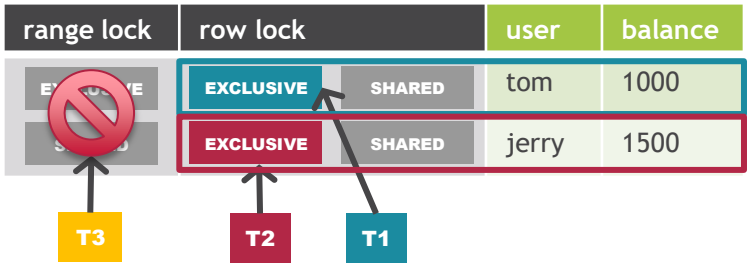




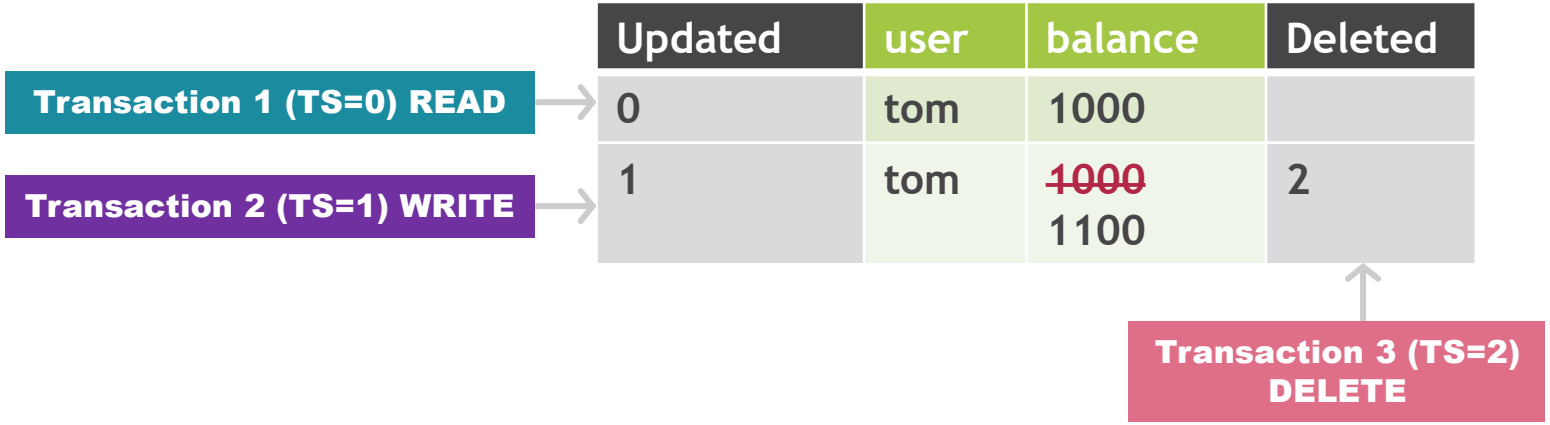
BY SCOPE

- Row - specific rows by index (if index exists)
- Range – all records by condition





Optimistic multi version concurrency control (MVCC)



HOW IT WORKS

- Transactions see the rows with version less or equal to transaction start time
- On update: new version is added
- On remove: deleted column is set of new version number

Optimistic MVCC vs pessimistic locks

	MVCC (OPTIMISTIC)	LOCKS (PESSIMISTIC)
Behavior	1. Each transaction works with it's own version 2. Concurrent transaction fails	1. Transaction which owns lock works with data 2. Concurrent transactions wait
Locks	NO	YES
Performance and scalability	GOOD	BAD
Deadlocks	NO	POSSIBLE
Guarantee of recent data version	NO	YES
Extra disk space needed	YES	NO
Durability	better (because of saved versions)	

Transaction isolation levels in different databases

DB	CONCEPT	READ UNCOMMITED	READ COMMITED	REPEATABLE READ	SERIALIZABLE	SPECIFICS
Oracle	MVCC	NOT SUPPORTED	DEFAULT return new snapshot each read	NOT SUPPORTED	returns snapshot of data at beginning of transaction	+ READ ONLY LEVEL transaction only sees data at the moment of start, writes not allowed always returns snapshots, transaction fail when concurrent update
MySQL (InnoDB)	MVCC		return new snapshot each time	DEFAULT - save snapshot at first read - return it for next reads	- locks ranges - transaction lifetime - Shared lock on select	
MSSQL	LOCKS	+ Double read phenomena: able to read same row twice while it is migrating to another place on disk	**SNAPSHOT (optimistic)** return new snapshot each time **LOCK (pessimistic)** DEFAULT - locks rows - statements lifetime	- locks rows - transaction lifetime	- locks ranges - transaction lifetime - selects: shared range lock - updates: exclusive lock	**+ SNAPSHOT LEVEL** - save snapshot at first read - return it for next reads - transactions fail in case of optimistic lock concurrent update
PostgreSQL	MVCC	NOT SUPPORTED	DEFAULT return new snapshot each read	- save snapshot at first read - return it for next reads	predicate locking (optimistic)	always returns snapshots, transaction fail when concurrent update

TRANSACTION ISOLATION IN MYSQL

LIVE DEMO

WANT TO OPTIMIZE?

LOCK SPECIFIC OBJECTS!

Pessimistic locking of specific rows/ranges (MySQL)

LOCKING SELECTS

- SELECT … LOCK IN SHARE MODE – shared (read) lock
- SELECT … FOR UPDATE – exclusive (write) lock

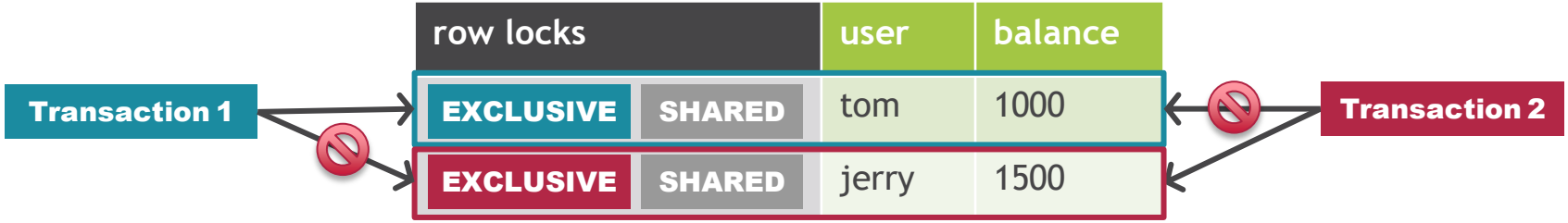
IDEA

- Increase isolation level for specific rows/ranges
- Other rows/ranges can have lower isolation level

MYSQL PESSIMISTIC LOCKING OF SPECIFIC OBJECTS

LIVE DEMO

Database deadlocks



Database deadlocks happen because of bad application architecture design

HOW TO PREVENT DEADLOCKS

- Take locks in same order in every transaction
- Use as small as possible transactions

WHAT ABOUT JAVA?

Transaction isolation: configuration in Java

JDBC

```
Connection c = DriverManager.getConnection("jdbc:mysql://localhost/testdb?user=test&password=pass");
c.setTransactionIsolation(level);
```

Hibernate

```
hibernate.connection.isolation=level
```

Connection.TRANSACTION_NONE	0
Connection.TRANSACTION_READ_UNCOMMITED	1
Connection.TRANSACTION_READ_COMMITED	2
Connection.TRANSACTION_REPEATABLE_READ	4
Connection.TRANSACTION_SERIALIZABLE	8

JPA features for locking

Enum LockModeType	
`PESSIMISTIC_READ`	Shared lock
`PESSIMISTIC_WRITE`	Exclusive lock
EntityManager	
`lock(Object entity, LockModeType lockMode)`	Makes additional locking select query just to lock entity
`find(Class<T> entityClass, Object primaryKey, LockModeType lockMode)`	Makes locking select when reading entity
`refresh(Object entity, LockModeType lockMode)`	Makes locking select when reloading entity
NamedQuery	
`@NamedQuery(name="myQuery", query="…", lockMode=LockModeType.PESSIMISTIC_READ)`	Allows to specify that we need locking select to any query

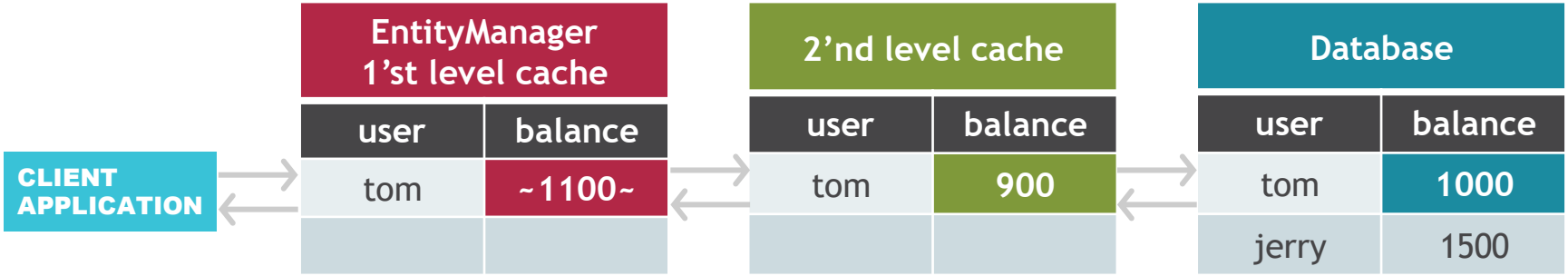
ADVANTAGES

- It is really simple
- Database specific things are hidden from developer
- Supports parent-child entities

DRAWBACKS

- Complex to manually lock entity relationships
- @NamedQuery is only way to specify query lock

Transaction isolation and locking with JPA



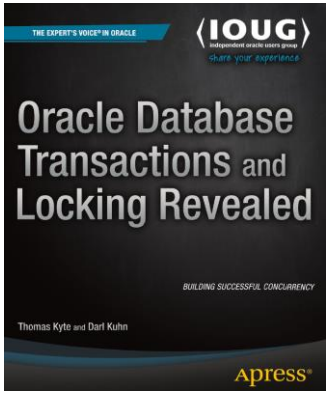
- Repeatable reads because of EntityManager's cache
- Requests do not always go to database

Behavior = EntityManager + 2'nd lvl cache + database

Conclusion

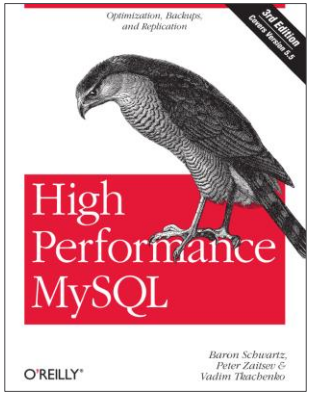
- Do you have problems because of concurrent updates?
  - Same as concurrent programming in Java
  - Sometimes we can allow phenomena
- Transaction Isolation is trade-off between data protection and performance
- Two main approaches in databases implementation:
  - Optimistic: no locks, data is versioned
  - Pessimistic: range and low locks
- JPA
  - Simplifies usage of pessimistic locking
  - Adds own specific behavior because of caches
- For better performance:
  - Prefer smaller transactions: they hold long time locks and can make deadlocks
  - Be careful with declarative transaction management it can make heavy transactions

References



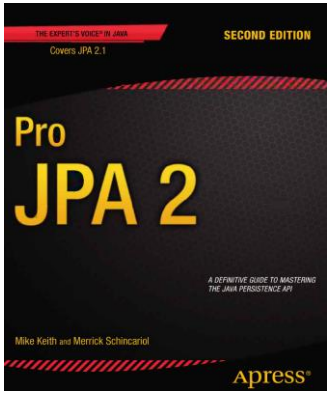
Oracle

Chapter 2-4



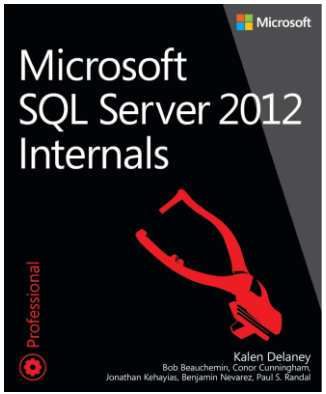
MySQL

Chapter 1



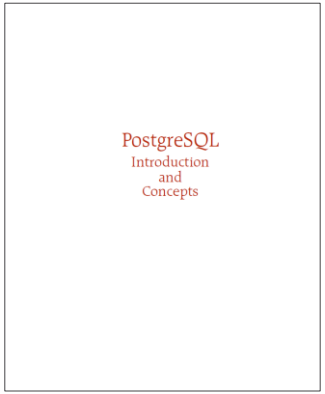
JPA

Chapter 12, Locking



MSSQL

Chapter 13



PostgreSQL

Chapter 10

Examples

https://github.com/kslisenko/tx-isolation

THANK YOU!

QUESTIONS?

kanstantsin_slisenka@epam.com