Husseinnasser.com
Updated-Oct/2021

ACID

Atomicity, Consistency, Isolation and Durability in Relational Database Systems

Agenda

- What is a Transaction?
- Atomicity
- Isolation
- Consistency
- Durability
- Quiz

What is a Transaction?

Transaction

- A collection of queries
- One unit of work
- E.g. Account deposit (SELECT, UPDATE, UPDATE)

Transaction Lifespan

- Transaction BEGIN
- Transaction COMMIT
- Transaction ROLLBACK
- Transaction unexpected ending = ROLLBACK (e.g. crash)

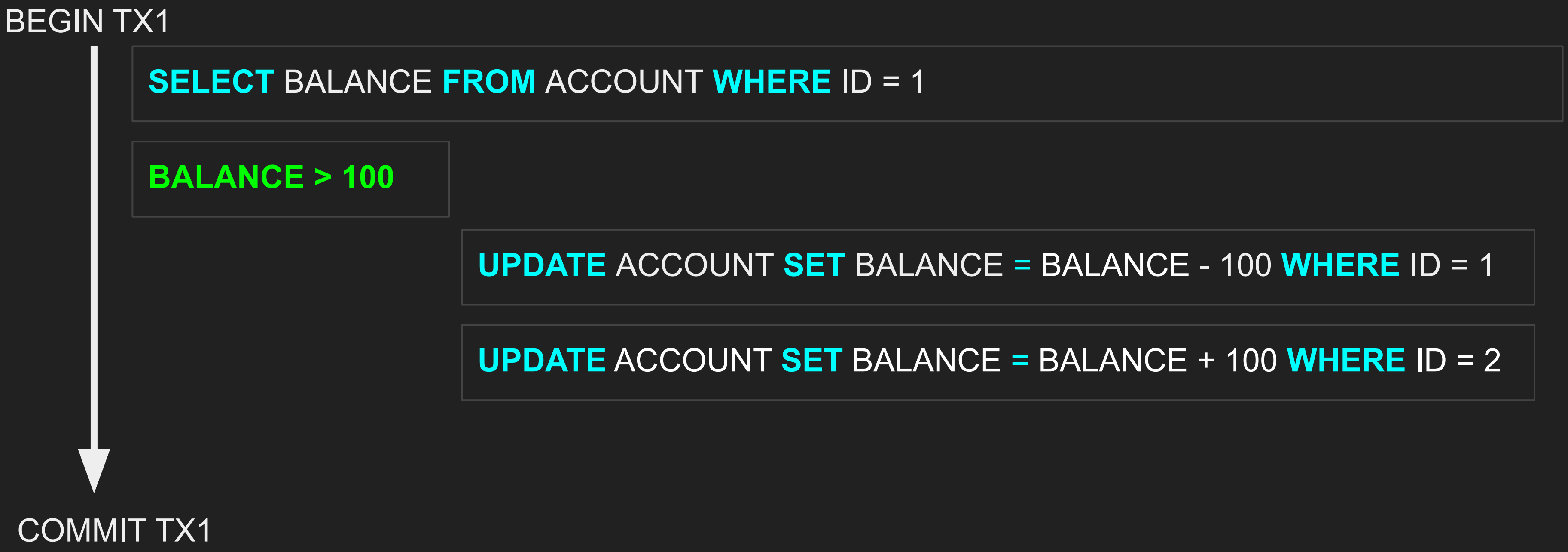
Nature of Transactions

- Usually Transactions are used to change and modify data
- However, it is perfectly normal to have a read only transaction
- Example, you want to generate a report and you want to get consistent snapshot based at the time of transaction
- We will learn more about this in the Isolation section

Transaction

ACCOUNT_ID	BALANCE
1	$900
2	$600

Send $100 From Account 1 to Account 2



BEGIN TX1

```
SELECT BALANCE FROM ACCOUNT WHERE ID = 1
```

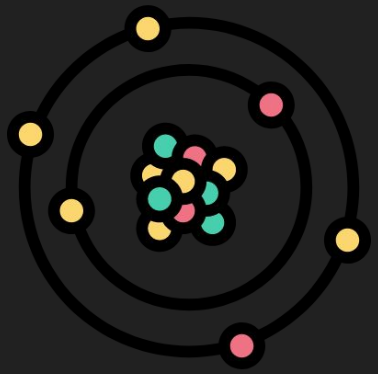
BALANCE > 100

```
UPDATE ACCOUNT SET BALANCE = BALANCE - 100 WHERE ID = 1
```

```
UPDATE ACCOUNT SET BALANCE = BALANCE + 100 WHERE ID = 2
```

COMMIT TX1

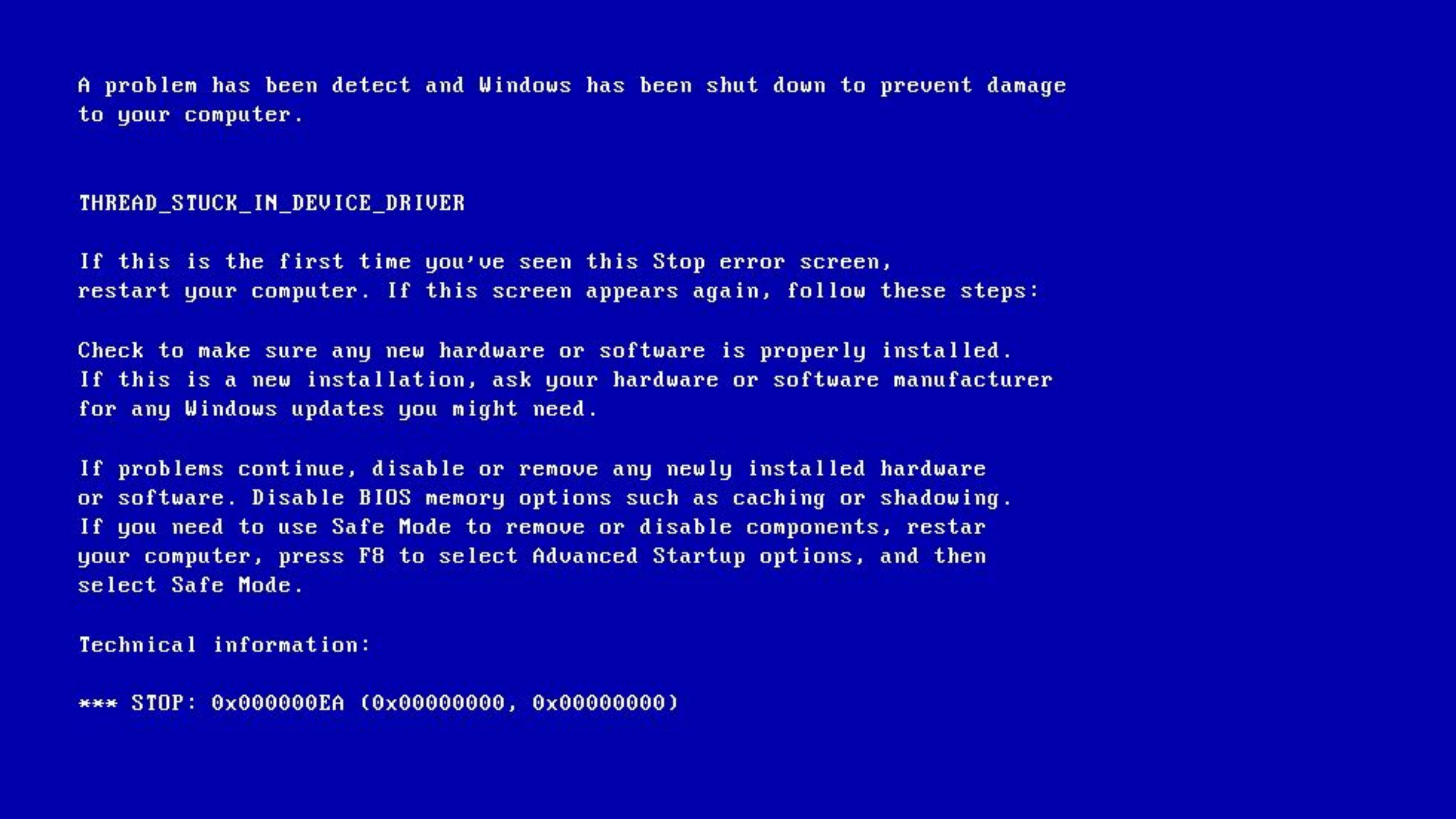
Summary
What is a Transaction?



Atomicity

Atomicity

- All queries in a transaction must succeed.
- If one query fails, all prior successful queries in the transaction should rollback.
- If the database went down prior to a commit of a transaction, all the successful queries in the transactions should rollback

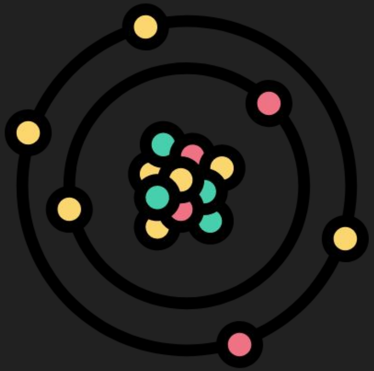


```
A problem has been detect and Windows has been shut down to prevent damage
to your computer.


THREAD_STUCK_IN_DEVICE_DRIVER

If this is the first time you've seen this Stop error screen,
restart your computer. If this screen appears again, follow these steps:

Check to make sure any new hardware or software is properly installed.
If this is a new installation, ask your hardware or software manufacturer
for any Windows updates you might need.

If problems continue, disable or remove any newly installed hardware
or software. Disable BIOS memory options such as caching or shadowing.
If you need to use Safe Mode to remove or disable components, restar
your computer, press F8 to select Advanced Startup options, and then
select Safe Mode.

Technical information:

*** STOP: 0x000000EA (0x00000000, 0x00000000)
```

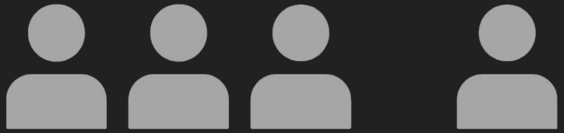
Atomicity

ACCOUNT_ID	BALANCE
1	$900
2	$500

- After we restarted the machine the first account has been debited but the other account has not been credited.
- This is really bad as we just lost data, and the information is inconsistent
- An atomic transaction is a transaction that will rollback all queries if one or more queries failed.
- The database should clean this up after restart.



Summary
Atomicity



Isolation

Isolation

- Can my inflight transaction see changes made by other transactions?
- Read phenomena
- Isolation Levels

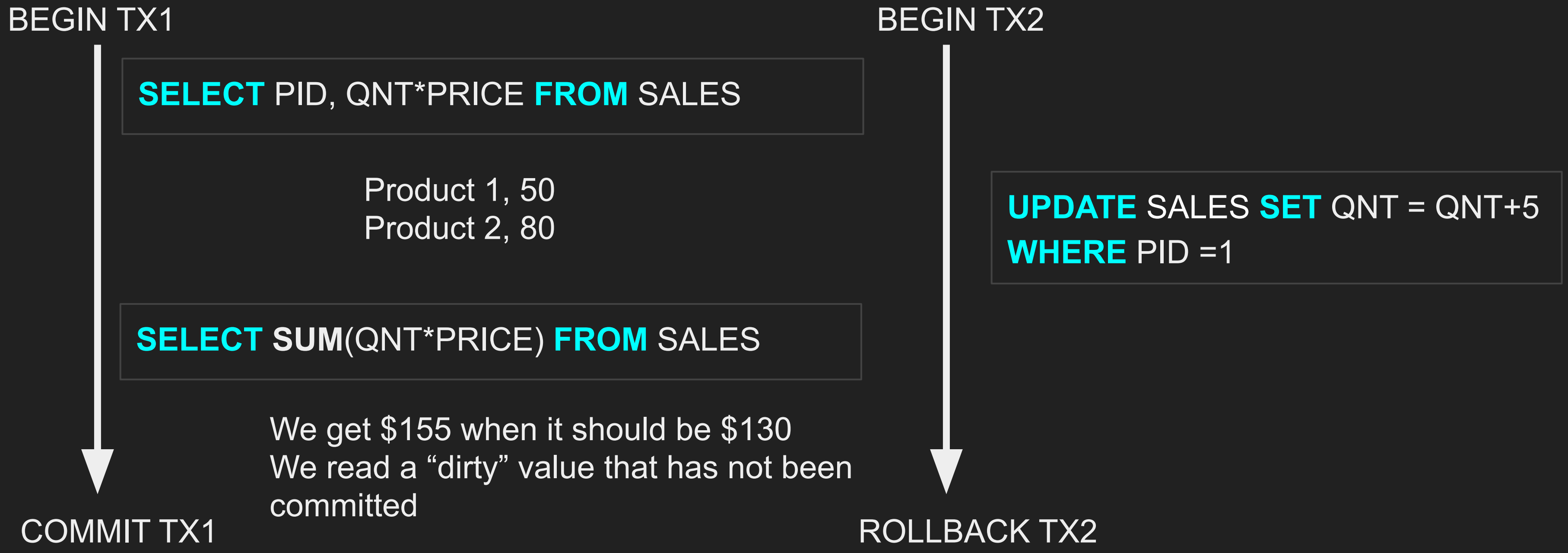
Isolation - Read phenomena

- Dirty reads
- Non-repeatable reads
- Phantom reads
- Lost updates

Dirty Reads

SALES

PID	QNT	PRICE
Product 1	10	$5
Product 2	20	$4



BEGIN TX1

SELECT PID, QNT*PRICE **FROM** SALES

Product 1, 50
Product 2, 80

SELECT SUM(QNT*PRICE) **FROM** SALES

We get $155 when it should be $130
We read a “dirty” value that has not been committed

COMMIT TX1

BEGIN TX2

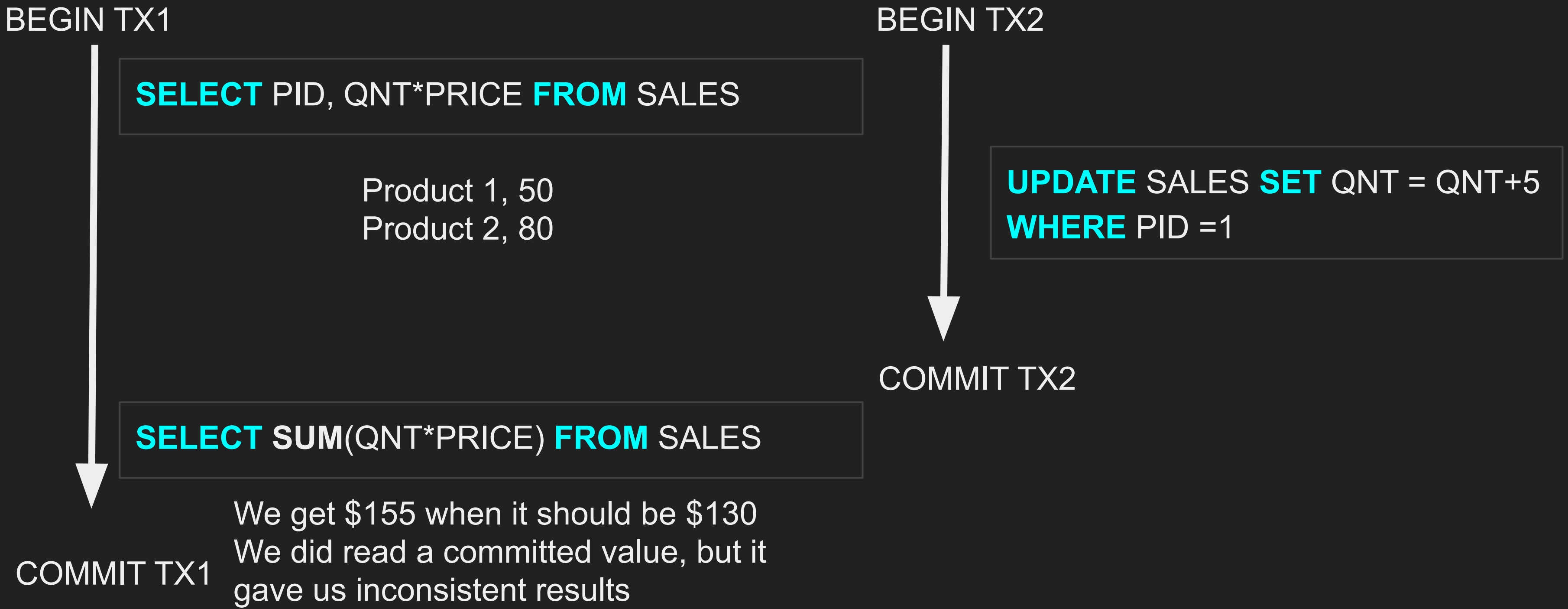
UPDATE SALES **SET** QNT = QNT+5
WHERE PID =1

ROLLBACK TX2

Non-repeatable read

SALES

PID	QNT	PRICE
Product 1	**15**	$5
Product 2	20	$4



BEGIN TX1

```
SELECT PID, QNT*PRICE FROM SALES
```

Product 1, 50
Product 2, 80

BEGIN TX2

```
UPDATE SALES SET QNT = QNT+5
WHERE PID =1
```

COMMIT TX2

```
SELECT SUM(QNT*PRICE) FROM SALES
```

COMMIT TX1

We get $155 when it should be $130
We did read a committed value, but it gave us inconsistent results

Phantom read

SALES

PID	QNT	PRICE
Product 1	10	$5
Product 2	20	$4
Product 3	10	$1

BEGIN TX1

```
SELECT PID, QNT*PRICE FROM SALES
```

```
Product 1, 50
Product 2, 80
```

BEGIN TX2

```
INSERT INTO SALES
VALUES ('Product 3', 10, 1)
```

COMMIT TX2

```
SELECT SUM(QNT*PRICE) FROM SALES
```

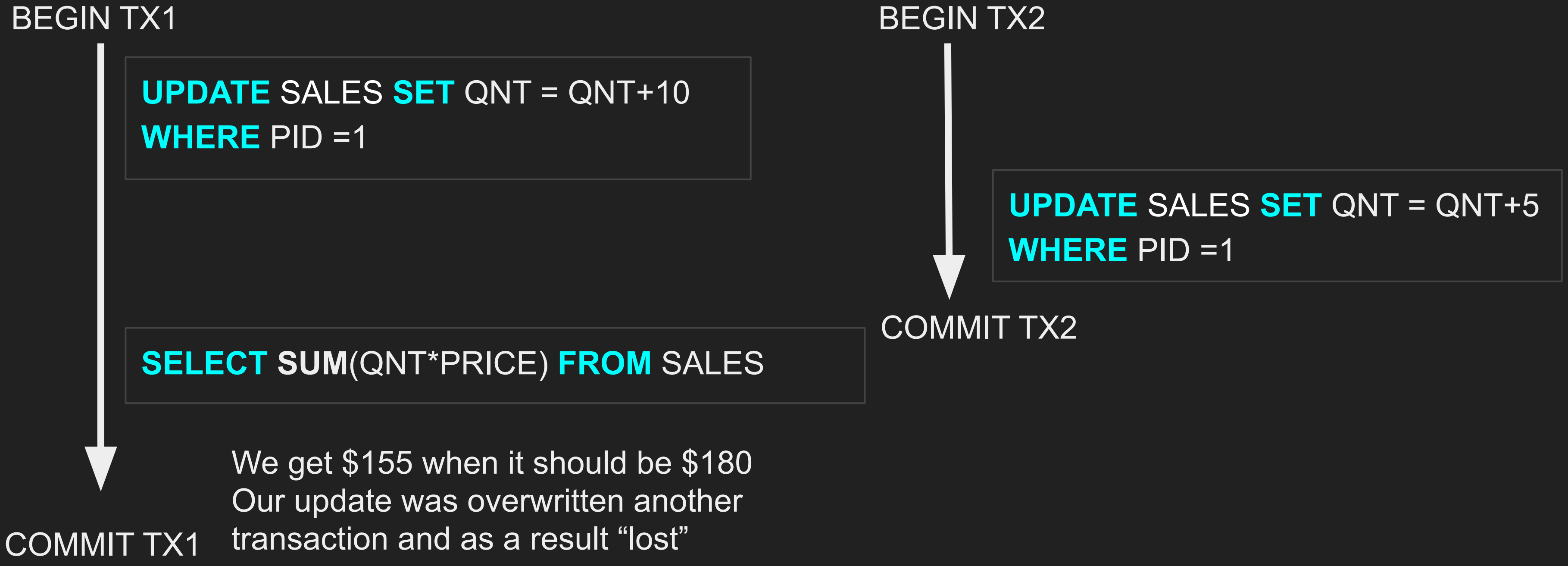
COMMIT TX1

We get $140 when it should be $130
We read a committed value that showed up in our range query

Lost updates

SALES

PID	QNT	PRICE
Product 1	15	$5
Product 2	20	$4



BEGIN TX1

```
UPDATE SALES SET QNT = QNT+10
WHERE PID =1
```

BEGIN TX2

```
UPDATE SALES SET QNT = QNT+5
WHERE PID =1
```

COMMIT TX2

```
SELECT SUM(QNT*PRICE) FROM SALES
```

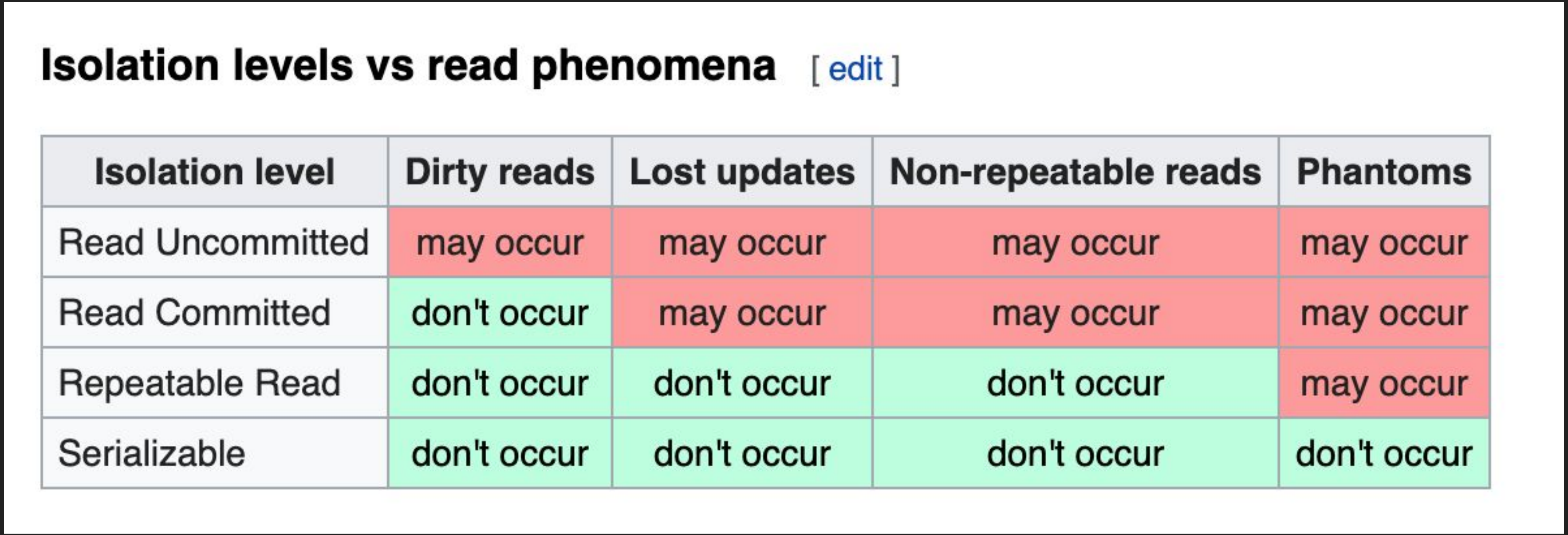
COMMIT TX1

We get $155 when it should be $180
Our update was overwritten another
transaction and as a result “lost”

Isolation - Isolation Levels for inflight transactions

- **Read uncommitted -** No Isolation, any change from the outside is visible to the transaction, committed or not.
- **Read committed -** Each query in a transaction only sees committed changes by other transactions
- **Repeatable Read -** The transaction will make sure that when a query reads a row, that row will remain unchanged while its running.
- **Snapshot -** Each query in a transaction only sees changes that have been committed up to the start of the transaction. It's like a snapshot version of the database at that moment.
- **Serializable -** Transactions are run as if they serialized one after the other.
- **Each DBMS implements Isolation level differently**

Isolation Levels vs read phenomena



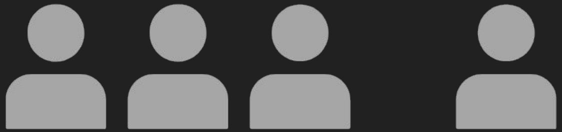
Isolation levels vs read phenomena [edit]

Isolation level	Dirty reads	Lost updates	Non-repeatable reads	Phantoms
Read Uncommitted	may occur	may occur	may occur	may occur
Read Committed	don't occur	may occur	may occur	may occur
Repeatable Read	don't occur	don't occur	don't occur	may occur
Serializable	don't occur	don't occur	don't occur	don't occur

https://en.wikipedia.org/wiki/Isolation_(database_systems)

Database Implementation of Isolation

- **Each DBMS implements Isolation level differently**
- **Pessimistic - Row level locks, table locks, page locks to avoid lost updates**
- **Optimistic - No locks, just track if things changed and fail the transaction if so**
- **Repeatable read “locks” the rows it reads but it could be expensive if you read a lot of rows, postgres implements RR as snapshot. That is why you don’t get phantom reads with postgres in repeatable read**
- **Serializable are usually implemented with optimistic concurrency control, you can implement it pessimistically with SELECT FOR UPDATE**



Summary
Isolation

Consistency

Consistency

- Consistency in Data
- Consistency in reads

Consistency in Data

- Defined by the user
- Referential integrity (foreign keys)
- Atomicity
- Isolation

Consistency in Data

Pictures

ID **(PK)**	BLOB	LIKES
1	xx	2
2	xx	1

Picture_Likes

USER **(PK)**	PICTURE_ID **(PK)(FK)**
Jon	1
Edmond	1
Jon	2

Spot inconsistency in this data

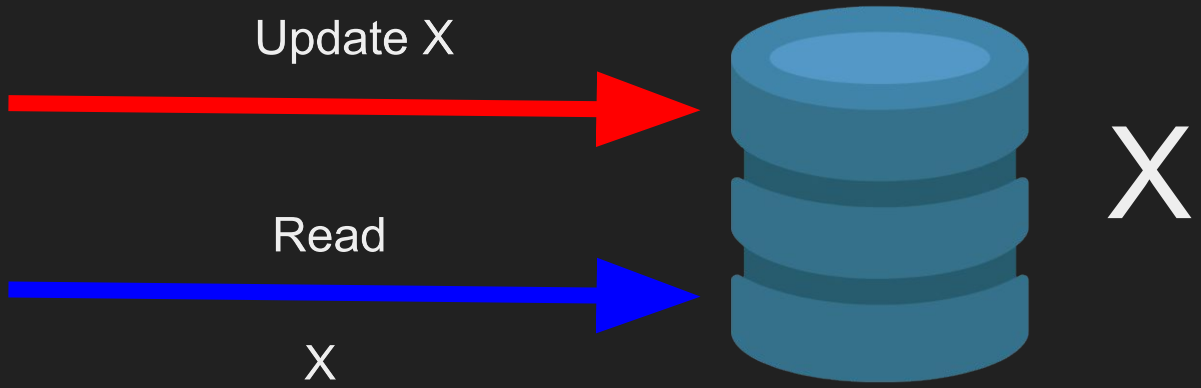
Pictures

ID **(PK)**	BLOB	LIKES
1	xx	5
2	xx	1

Picture_Likes

USER **(PK)**	PICTURE_ID **(PK)(FK)**
Jon	1
Edmond	1
Jon	2
Edmond	4

Consistency in reads



Consistency in reads

- If a transaction committed a change will a new transaction immediately see the change?
- Affects the system as a whole
- Relational and NoSQL databases suffer from this
- Eventual consistency

Summary
Consistency

Durability

Durability

- Changes made by committed transactions must be persisted in a durable non-volatile storage.
- Durability techniques
    - WAL - Write ahead log
    - Asynchronous snapshot
    - AOF

Durability - WAL

- Writing a lot of data to disk is expensive (indexes, data files, columns, rows, etc..)
- That is why DBMSs persist a compressed version of the changes known as WAL (write-ahead-log segments)

Durability - OS Cache

- A write request in OS usually goes to the OS cache
- When the writes go the OS cache, an OS crash, machine restart could lead to loss of data
- Fsync OS command forces writes to always go to disk
- fsync can be expensive and slows down commits

Summary
Durability

Summary

- What is a Transaction?
- Atomicity
- Isolation
- Consistency
- Durability