Assignment 6

# Concurrency Tuning

## Database Tuning

A4

Balint Peter, 12213073
Günter Lukas, 12125639
Ottino David, 51841010

June 16, 2025

### Notes

- You will need to run transactions concurrently using threads in Java. See `https://dbresearch.uni-salzburg.at/teaching/2020ss/dbt/account.zip` for an example.

## Experimental Setup

Describe your experimental setup in a few lines.

The experiment is started via a shell-script. The script creates the database and the table if it doesn't exist already and then executes each solution with a different amount of workers and different isolation levels. We decided to implement the database-access itself with a Python-script and the library psycopg2. The queries for both solutions are prepared according to the specification. If a query is executed and aborts, the program automatically rolls back the current transaction, waits a random time and tries executing it again until the last statement ist commited and the connection closed. Upon execution, the python script also measures execution time and correctness.

## Solution (a)

**Read Committed** Throughput and correctness for solution (a) with isolation level `READ COMMITTED`.

| #Concurrent Transactions | Throughput [transactions/sec] | Correctness |
|---|---|---|
| 1 | 22.29 | 1.0 |
| 2 | 65.68 | 0.96 |
| 3 | 87.98 | 0.89 |
| 4 | 95.01 | 0.85 |
| 5 | 101.07 | 0.72 |

**Serializable** Throughput and correctness for solution (a) with isolation level `SERIALIZABLE`.

| #Concurrent Transactions | Throughput [transactions/sec] | Correctness |
|---|---|---|
| 1 | 23.21 | 1.0 |
| 2 | 45.45 | 1.0 |
| 3 | 56.28 | 1.0 |
| 4 | 62.20 | 1.0 |
| 5 | 69.86 | 1.0 |

## Solution (b)

**Read Committed** Throughput and correctness for solution (b) with isolation level `READ COMMITTED`.

| #Concurrent Transactions | Throughput [transactions/sec] | Correctness |
|---|---|---|
| 1 | 30.15 | 1.0 |
| 2 | 74.56 | 1.0 |
| 3 | 92.94 | 1.0 |
| 4 | 96.26 | 1.0 |
| 5 | 98.23 | 1.0 |

**Serializable** Throughput and correctness for solution (b) with isolation level `SERIALIZABLE`.

| #Concurrent Transactions | Throughput [transactions/sec] | Correctness |
|---|---|---|
| 1 | 30.01 | 1.0 |
| 2 | 65.26 | 1.0 |
| 3 | 68.51 | 1.0 |
| 4 | 66.99 | 1.0 |
| 5 | 78.47 | 1.0 |

## Discussion

Discuss the outcomes and explain the difference between the isolation levels in PostgreSQL with respect to your experiment.

Explain **with your own words** how PostgreSQL deals with updates in the different isolation levels, within a transaction and within a single SQL command. Explicitly explain why you got the experimental results for solution (a) and (b), respectively.

### Solution (a)

Read Comitted is the default isolation level in Postgres. This means that for each statement within a transaction a snapshot is generated and mantained for said transaction. This means that the snapshop changes dynamically within a transaction and the transaction itself can already access that changed data, but other transactions only see the changes after the commit. If changes are commited, already existing snapshots for other transactions are not updated dynamically. If two transactions try to update the sama tuple, one of them has to wait until the other one commits or rolls back and then works with the changed data. This leads us to the result for solution (a) with isolation level Read committed. If only one worker is used, the correctness is still 1 since all the transactions are executed one after another. However, if the amount of workers is increased,

the solution leads to more and more errors. This is because two transactions will read the same data, but one commits before the other. However, the calculated new value for the second transaction is based on the data read before the commit of the first transaction. Therefore the changes made by the first transaction are overwritten and effectively lost.

If isolation level Serializable is used, this can't happen. Conceptually, this isolation level is equal to executing each transaction one ofter another. To understand how Postgres achieves this, we have to explain the isolation level Repeateable read first. In contrast to Read Committed, the snapshot used here is on transaction level and not for each statement. This means that each transaction has a consistent data base and commits from other transactions are not accessed within the transaction itself. If another transaction successfully commits, Postgres checks if the planned changes are still consistent with the original snapshot and if not the transaction returns an error. Serializable is basically the same, but additionally the system checks for so-called serialization anomalies. This means the result is not commited, if there is no serializable schedule that would lead to the same result. With this being the strictest isolation level in Postgres, our results lead to no errors occuring in the end result of the execution. But since this means more overhead for the database itself and more rollebacks/retries, the throughput is generally lower than for isolation level Read Committed.

### Solution (b)

Since solution b changes the executed statements from two to one for each account in each step, the aforementioned error case is no longer possible with isolation level Read Committed. Because the update is done in a single statement without reading the initial value first, the lock on the tuple is aquired immediately and not released until the statement is either commited or rolled back. If a second transaction were to be started while the first transaction is still in progress, the lock would lead to the second transaction having to wait with its execution and upon execution seeing only the already changed data. Therefore a correctness of 1.0 is achieved, even with a lower isolation level.

The isolation level Serializable is still correct in each case and the performance is closer to Read Committed than with solution a because fewer transactions get aborted and restarted but the additional overhead that comes with this isolation level still means that the throughput is lower. The experiment shows that for simple transactions like in solution b, the isolation level Read Committed still provies high enough correctness if the queries are thought out carefully.

For the sake of completeness we want to mention the theoretically lowest level of isolation in Postgres, Read Uncommitted. It is possible to define this level, but internally it is treated in the same way as Read Committed.

## Time Spent on this Assignment

Time in hours per person: **3**

## References

**Important:** Reference your information sources!
https://www.postgresql.org/docs/current/transaction-iso.html#XACT-READ-COMMITTED