

DAT602 assignment 3 REPORT

Lee Vartha



November 10, 2024

# Report: Comparison Of Concurrency Management in MySQL and .NET C#

Concurrency Management is designed in database to manage two processes that will get executed at the same time and ensure they don’t have conflicts between each other when executed.  
MySQL and .NET C# can provide different ways to manage this and in MySQL, concurrency is done through *transaction isolation levels* and .NET uses *Optimistic* and *Pessimistic* concurrency (through LINQ and Entity Framework)

**Concurrency Management in MySQL:**

Stored procedures in MySQL are used to showcase transactions through **START** **TRANSACTION**, **COMMIT** and **ROLLBACK**.

Concurrency management in MySQL is done through transaction isolation levels:

1. **Read Uncommitted**

Transactions can read uncommitted transactions, which leads to dirty reads. Dirty read refers to when a transaction reads data that was made by another uncommitted transaction – this increases complication to whether the read will end up being committed or rolled back. Overall, this is the weakest isolation level.

1. **Read Committed**

This means that dirty reads are prevented but non-repeatable – this may mean that data could change while the transaction is running. This level can also access updated data from different transactions.

1. **Repeatable Read**

This means that a transaction does read data, but phantom reads may still happen. Phantom reads is where a query can read rows that wasn’t previously in the transaction.

1. **Serializable**

This ensures that transactions are run sequentially – this is the best level to run as it prevents all types of concurrency anomalies.

**Concurrency management in .NET C#:**

**LINQ** = A way to query databases in C#  
**Entity Framework =** A tool used to connect C# to databases such as MySQL

Using concurrency management in .NET C# means that the application can run more than one operation at the same time. By utilizing concurrency, multiple objects can run independently, which would make performance nicer.

Concurrency management in .NET C# is done through optimistic and pessimistic controls:

**Optimistic** concurrency means that any conflicts between transactions don’t happen often – optimistic controls detect conflict and fixes it before changes are made to the database. From this, it means theres no negative impact on the performance in the application and it supports scalability.  
This is applied to both LINQ and Entity Frameworks.

**Pessimistic** concurrency locks data during ‘read or write’ operations – this means that transactions wont be able to access it all at the same time. Entity frameworks utilize this and ensures consistency in data, but this would negatively impact the performance in the application.

In comparison between MySQL and .NET C#, MySQL handles concurrency within the database using transaction isolation modes, which have levels that has different performance ranges. However, .NET C# manages concurrency within the application with LINQ or Entity Frameworks, utilizing either Optimistic or Pessimistic concurrency.

MySQL’s isolation levels showcase consistency with their usage, but it can impact performance. In .NET, Optimistic concurrency provides better performance for the application, whereas Pessimistic concurrency helps with consistency but causes issues with performance.

Overall, both MySQL and .NET C# each have comparable ways to manage concurrency, through the approaches are each different.  
MySQL works around concurrency using isolation levels in the database, while .NET C# uses Optimistic and Pessimistic concurrency strategies in the application.  
Each methods depends on whether the developers wish to focus on consistency or performance and based on what the needs are for the application.

Ackerland, J. (2022, November 27). *MySql Transaction Isolation Levels.* Medium.   
<https://medium.com/@jonackerland/mysql-transaction-isolation-levels-2876b0d8302d>