2.5 Concurrency

Concurrency of choice:

For our project we are using pessimistic concurrency control, because it is the best way of to evade from biggest concurrency issue of our project- lost update. Lost update in our project may occur when two people want to join the chat at the same time.

Places of interest regarding concurrency:

* Join chat
* Join chat with group (either all online members join or non)
* Join game

Concurrency problem:

Concurrency issue would happen like this- two users try to join chat where max number of places is 5 and 4 users are already inside, both user1 and user2 receive this info and update it according to their needs, when user1 finishes updating, chat shows that its full, but user2 continues calculations, when he finally finishes its calculation changes users online to 5 again. Both users have successfully joined, but number of users in chat is incorrect there are 6 users when only 5 could join.

Concurrency solution:

This issue is fixed with locks, after user starts joining chat he locks object and nobody else can modify it until he is finished with updates. After user finishes his task he releases lock and next person can do their task.



Above is image showing how we are locking the object, we chose to lock the specific objects, to maximize the speed, so that all users don’t have to stay in the same queue jut to join different chat (each chat has different queue).

ACID in our project:

We tried to ensure ACID properties in our project, we are ensuring atomicity- with transactions, consistency- by testing user inputs before sending them to database, isolation- by using locks and transaction isolation, durability by catching exceptions and giving inputs as parameters.

Isolation levels:



From: <https://docs.microsoft.com/en-us/sql/odbc/reference/develop-app/transaction-isolation-levels>

Isolation level of choice:

For our solution we decided to use repeatable read isolation level, because

possible deadlocks