|  |  |
| --- | --- |
| **Name** | Shubham Golwal |
| **UID no.** | 2020300015 |
| **Experiment No.** | 01 |

|  |  |
| --- | --- |
| **AIM:** | Concurrency control |
| **Program 1** | |
| **Problem Statement :** | Perform concurrency control in PostgreSQL Database with college database. |
| **Theory :** | **What are transactions?**  A transaction is a very small unit of a program and it may contain several lowlevel tasks. A transaction in a database system must maintain Atomicity, Consistency, Isolation, and Durability − commonly known as ACID properties − in order to ensure accuracy, completeness, and data integrity.  **What is concurrency control?**  Concurrency control concept comes under the Transaction in database management system (DBMS). It is a procedure in DBMS which helps us for the management of two simultaneous processes to execute without conflicts between each other, these conflicts occur in multi user systems.  Concurrency can simply be said to be executing multiple transactions at a time. It is required to increase time efficiency. If many transactions try to access the same data, then inconsistency arises. Concurrency control required to maintain consistency data.  For example, if we take ATM machines and do not use concurrency, multiple persons cannot draw money at a time in different places. This is where we need concurrency.  **The advantages of concurrency control are as follows −**  • Waiting time will be decreased.  • Response time will decrease.  • Resource utilization will increase.  • System performance & Efficiency is increased.  MOSS Concurrency Control Protocol (Distributed Locking in Database) -  GeeksforGeeks  *\*credits to gfg* |
| **Output** | Drop:    Create:    Insert:      Initial insert into database  Begin Transaction:      Started a new transaction and insert a new student into the college table:  **Commit:**    Made the changes become visible to other sessions (or users), committed the transaction by using the COMMIT TRANSACTION statement  Alternative: COMMIT WORK, COMMIT;  After executing the COMMIT statement, PostgreSQL also guarantees that the change will be durable if a crash happens.  ------x--------------------  **Update:**  Currently score at id = 2 is 75.      Changes is made to id =2 [Varun] and score is updated to 47.5  After committing changes are made visible.  **Rollback:**  Updated but not committed:    Score at id = 2 is 47.5.    After executing rollback score at id = 2 is reverted to 75  But if transaction is committed then we can’t revert back.    **Serializability:**  I’m gonna start 2 new transactions, then set their isolation level to serializable.    Get sum of all the scores of students        Serialized updates:  Window 1:    Window 2:    Error: could not serialize access due to read/write dependencies among transactions DETAIL:  Reason code: Cancelled on identification as a pivot, during commit attempt. HINT: The transaction might succeed if retried. SQL state: 40001  Why has it happened?  When we committed first update operation it worked fine. But when we committed second one on same tuple then it throws serializable error.    Checking serializable:  Window 2:  Is this equivalent to any serializable schedule?  Yes, this schedule is conflict serializable. Because we are performing write operation from both transactions at the same time. |
| **Conclusion:**   * I learnt meaning of serializability and concurrency in a transaction. * I also learnt that various Transaction language (TCL) commands are used for managing and controlling the transactions. * Executed serializability and under the conflict in serializability. | |