CS580 ONLY Project # 4 Fall 2021

Group Project

|  |  |
| --- | --- |
| Student Names | Antonio Zea |
| Section | CS 580 |

For this project you will be researching transaction processing and concurrency control techniques (Ch. 21-22 in both 6th & 7th edition of textbook). This project consists of both a small research summary of the issues of this topic as well as completing problems related to this topic (below). This is a group project, but be sure all know how to do the problems below as understanding of them will be on the final exam.

SUBMIT:

One word document called P4-LastNamesOfGroupMembers.

In the document include names of students in the group and provide the following:

* A 5 page write up summary of the issues and challenges of transactions and concurrency; and the methods used to handle these challenges. This write up should include both what are the theoretical challenges, but also a description of how such a scenario could occur in a real application.
* The answers to the problems below:

|  |
| --- |
| STUDENT NAME(s): |
| 1. Construct the serializability (or precedence) graph for the schedule specified bellow. Determine if the following schedule is (conflict) serializable. If it is, specify equivalent serial schedule(s).   r2(X); r3(X); w2(X); r1(X); w3(X) |
| 1. Construct the serializability (or precedence) graph for the schedule specified bellow. Determine if the following schedule is (conflict) serializable. If it is, specify equivalent serial schedule(s).   r1(X); r2(X); w2(X); w1(X); r3(X) |
| 1. Construct the serializability (or precedence) graph for the schedule specified bellow. Determine if the following schedule is (conflict) serializable. If it is, specify equivalent serial schedule(s).   r3(X); r2(X); w3(X); r1(X); w1(X), w2(X) |
| 1. Construct the serializability (or precedence) graph for the schedule specified bellow. Determine if the following schedule is (conflict) serializable. If it is, specify equivalent serial schedule(s).   r3(X); r2(X); w3(X); w1(X) |
| 1. Consider the three transactions T1, T2, and T3, and the schedule specified bellow. Construct the serializability (precedence) graph. Determine if the schedule is (conflict) serializable. If it is, specify equivalent serial schedule(s).   T1: r1(X); r1(Z); w1(X);  T2: r2(Z); r2(Y); w2(Z); w2(Y);  T3: r3(X); r3(Y); w3(Y);  SCHEDULE: r1(X); r2(Z); r1(Z); r3(X); r3(Y); w1(X); w3(Y); r2(Y); w2(Z); w2(Y); |

CH 21

* Introduction to Transaction Processing
  + Single-User versus Multiuser System
    - Database systems may be classified according to the number of users who can use the system concurrently. A DBMS is single-user if at most one user at a time can use the system and it is multiuser if many users can use the system concurrently. Single-user DBMSs are generally used in small applications that reside on personal computers (i.e. tinydb in mobile apps).
    - This functionality is supported through interleaved concurrency. Database operations do not actually happen at the exact same time. They are instead interleaved which means while one operation is working the other is waiting for its turn to work on the database.
  + Transactions, Database Items, Read and Write Operations and DBMS Buffers
    - A transaction is an executing program that forms a logical unit of database processing. One way to specify transaction boundaries
  + Why Concurrency Control is Needed
    - The Lost Update Problem
      * This problem occurs when two transactions that access the same database items their operations interleaved in a way that makes the value of some database items incorrect.
        + EXAMPLE HERE
    - The Temporary Update (or Dirty Read) Problem
      * This problem occurs when one transaction updates a database item and then the transaction fails for some reason. Meanwhile, the update item is accessed(read) by another transaction before it is changed back to its original value.
        + EXAMPLE HERE
    - The Incorrect Summary Problem
      * If one transaction is calculating an aggregate summary function on a number of database items while other transactions are updating some of these items, the aggregate function may calculate some values before they are updated and others after they are updated.
        + EXAMPLE HERE
    - The Unrepeatable Read Problem
      * Another problem that may occur is called unrepeatable read, where a transaction T reads the same item twice and the item is changed by another transaction T’ between the two reads. Hence T receives different values for its two reads of the same item.
        + EXAMPLE HERE
  + Why Recovery is Needed
    - Whenever a transaction is submitted to a DBMS for execution, the system is responsible for making sure that either all the operations in the transaction are completed successfully and their effect is recorded permanently in the database, or that the transaction does not have any effect on the database or on any other transactions.
    - Types of failure
      * A computer failure (system crash)
      * A transaction or system error
      * Local errors or exception conditions detected by the transaction
      * Concurrency control enforcement
      * Disk failure
      * Physical problems and catastrophes
* Transaction and System Concepts
  + Transaction States and Additional Operations
    - Begin\_Transaction
    - Read or Write
    - End\_Transaction
    - Commit\_Transaction
    - Rollback
  + The System Log
  + Commit Point of a Transaction
* Desirable Properties of Transactions
  + Atomicity
  + Consistency preservation
  + Isolation
  + Durability or permanency
* Characterizing Schedules Based on Recoverability\*\*\*\*\*
  + Schedule (Histories) of Transactions
  + Characterizing Schedules Based on Recoverability
* Characterizing Schedules Based on Serializability
  + Serial, Nonserial, and Conflict-Serializable Schedules
  + Testing for Conflict Serializability of a Schedule
  + How Serializability Is Used for Concurrency Control
  + View Equivalence and View Serializability
  + Other Types of Equivalence of Schedules

CH 22

* Two-Phase Locking Techniques for Concurrency Control
  + Types of Locks and System Lock Tables
  + Guaranteeing Serializability by Two-Phase Locking
  + Dealing with Deadlock and Starvation
* Concurrency Control Based on Timestamp Ordering
  + Timestamps
  + The Timestamps Ordering Algorithm
* Multiversion Concurrency Control Techniques
  + Multiversion Technique Based on Timestamp Ordering
  + Multiversion Two-Phase Locking Using Certify Locks
* Validation (Optimistic) Concurrency Control Techniques
* Granularity of Data Items and Multiple Granularity Locking
  + Granularity Level Considerations for Locking
  + Multiple Granularity Level Locking
* Using Locks for Concurrency Control in Indexes
* Other Concurrency Control Issues
  + Insertion, Deletion, and Phantom Records
  + Interactive Transactions
  + Latches

A database system may be classified according to the number of users who can use the system concurrently. A DBMS is single-user if at most one user at a time can use the system and it is multiuser if many users can use the system concurrently. Single-user DBMSs are generally used in small applications that reside on personal computers (i.e., TinyDB in mobile apps). Multiuser databases accomplish this functionality by using interleaved concurrency. Database operations do not actually happen at the exact same time. They are instead interleaved which means while one operation is working the other is waiting its turn to work on the database.

A transaction is an executing program that forms a logical unit of database processing. A transaction includes one or more database access operations which can include insertion, deletion, modification or retrieval operations.

One way of specifying the transaction boundaries is by specifying explicit begin transaction and end transaction statements in an application program. If the database operations in a transaction do not update the database but only retrieve data, the transaction is called read-only, otherwise it is known as a read-write transaction.

Several problems can occur when concurrent transactions execute in an uncontrolled manner. The Lost Update Problem occurs when two transactions that access the same database items their operations have their operations interleaved in a way that makes the value of some database items incorrect.

The Temporary Update (or Dirty Read) problem occurs when one transaction updates a database item and then the transaction fails for some reason. Meanwhile, the update item is accessed(read) by another transaction before it is changed back to its original value.

The Incorrect Summary problem occurs if one transaction is calculating an aggregate summary function on a number of database items while other transactions are updating some of these items, the aggregate function may calculate some values before they are updated and others after they are updated.

The Unrepeatable Read problem occurs when a transaction T reads the same item twice and the item is changed by another transaction T’ between the two reads. Hence T receives different values for its two reads of the same item.

Whenever a transaction is submitted to a DBMS for execution, the system is responsible for making sure that either all the operations in the transaction are completed successfully and their effect is recorded permanently in the database, or that the transaction does not have any effect on the database or on any other transactions.

Types of failure

A computer failure (system crash)

A transaction or system error

Local errors or exception conditions detected by the transaction

Concurrency control enforcement

Disk failure

Physical problems and catastrophes

A lock is variable associated with a data item that describes the status of the item with respect to possible operations that can be applied to it.

A binary lock can have two states or values: locked or unlocked. A binary lock enforces mutual exclusion on the data item. This system ends up being too restrictive, if every transaction is seeking to read the locked item, then there is not very good reason to lock the item since read operations from different transactions cannot conflict. Shared/Exclusive locks are favored as they allow any number of transactions to read the share-locked item while an exclusive-locked item is only accessible by one transaction that holds its lock.

The two-phase locking (2PL) protocol if all locking operations (read\_lock, write\_lock) precede the first unlock operation in the transaction. If every transaction in a schedule follows the two-phase locking protocol, the schedule is guaranteed to be serializable.

Conservative 2PL requires a transaction to lock all the items it accesses before the transaction begins execution. If any of the items needed cannot be locked, the transaction does not lock any item and instead waits until all items are available for locking.

In strict 2PL a transaction does not release any of its exclusive locks until after it commits or aborts. This leads to a strict schedule for recoverability although it is not deadlock-free.

Deadlock occurs when each transaction in a set of two or more transactions is waiting for some item that is locked by some other transaction in the set. One way to prevent deadlock is to use a deadlock prevention protocol such as wait-die or wound-wait. Although both techniques are deadlock-free both may cause some transactions to abort needlessly.

Wait-die allows the older transaction to wait until the resource is available for execution.

Starvation occurs when a transaction cannot proceed for an indefinite period of time while other transactions in the system continue normally. A fair waiting scheme such as FIFO can be used to mitigate starvation. Transactions are enabled to lock an item in the order in which they originally requested the lock.