Connor Brown

CS44800 Homework 5

**Question 1. (0.75 points)**

Discuss the four standard isolation levels and the anomalies each level prevents. Give one example for each anomaly.

1. Serializable - the highest level and most restrictive level that prevents other user from updating or inserting rows into the data set until the transaction completes.
   1. Prevents dirty reads, lost updates, non-repeatable reads, and phantom reads
2. Repeatable reads - the transaction locks the data that is used in the query and another transaction will be unable to update the data until the transaction is completed.
   1. Prevents dirty reads, lost updates, and non-repeatable reads
   2. A phantom read can occur when another transaction inserts a new row that match the current search conditions of the current transaction. The current transaction then retries the statement to retrieve the new rows, and results in a phantom read.
3. Committed read - a transaction can read only committed changes to the data and cannot read data that has been modified but not committed by other transactions.
   1. Prevents dirty reads and lost updates.
   2. Non-repeatable reads and phantom reads can occur when a transaction needs to read a row that has been modified by an incomplete transaction in another session.
4. Uncommitted read - allows a task to read data that can be in the middle of being modified by another transaction, uncommitted changes to the data in the database.
   1. Results in dirty reads.
   2. Values in the data can be changed and rows can appear or disappear in the data set before the end of the transaction, resulting in anomalies occurring.

**Question 2. (0.75 points)**

Compare optimistic concurrency control techniques and pessimistic concurrency control techniques. What are the advantages and disadvantages of each approach?

* Optimistic Concurrency Control
  + Advantages
    - Assumes conflicts to be rare and locks only when there are indications that two users try to update the same record at the same time.
    - It locks the record only when updating takes place.
    - Ensures changes to data are not performed between time of being read and being altered by using version numbers, timestamps, hashing, etc.
    - Minimizes the time for which a given resource is unavailable.
  + Disadvantages
    - Not useful in a program where there is a lot of updates as there is a higher possibility for error
* Pessimistic Concurrency Control
  + Advantages
    - Assumes that two or more users will want to update the same record at the same time and prevents the possibility by locking the record.
    - Locks the record as soon as it selects rows to update.
    - Guarantees the changes are made safely and consistently
  + Disadvantages
    - Resources are locked from the time it is first accessed in a transaction until the transaction is finished, making it inaccessible to the other transactions during that time
    - If most transactions just look at the resource and never change it, it may cause lock contention

**Question 3. (0.50 points)**

Consider the schedule of operation of three transactions that will be executed concurrently. R1(A), R2(B), R3(A), W1(A), R1(C), R2(C), W3(C), W2(A), R2(A), R1(D), W1(D)

1. Is this schedule serial?
   1. The schedule is not serial.
2. Give the dependency graph of this schedule.

|  |  |  |
| --- | --- | --- |
| T1 | T2 | T3 |
| R1(A) |  |  |
|  | R2(B) |  |
|  |  | R3(A) |
| W1(A) |  |  |
| R1(C) |  |  |
|  | R2(C) |  |
|  |  | W3(C) |
|  | W2(A) |  |
|  | R2(A) |  |
| R1(D) |  |  |
| W1(D) |  |  |

1. Is this schedule conflict serializable?
   1. Yes
2. If the answer is ‘yes’ to (c) provide the equivalent serial schedule. If you answer ‘no’, briefly explain why.

|  |  |  |
| --- | --- | --- |
| T1 | T2 | T3 |
| R1(A) |  |  |
| W1(A) |  |  |
| R1(C) |  |  |
| R1(D) |  |  |
| W1(D) |  |  |
|  | R2(B) |  |
|  | R2(C) |  |
|  | W2(A) |  |
|  | R2(A) |  |
|  |  | R3(A) |
|  |  | W3(C) |

**Question 4. (1.00 point)**

Consider the following schedules, where R, W, C stands for ‘Read’, ‘Write’ and ‘Commit’ respectively and subscripts refers to transactions (i.e., R1(X) means transaction T1 read X):

Schedule 1: R1(X), W1(X), R2(X), W2(X), R1(Y), W1(Y), C1, R2(Y), W2(Y) C2

Schedule 2: R2(X), W2(X), R3(Y), W3(Y), R3(Z), W3(Z), C3, R2(Z), W2(Z), C2, R1(X), W1(X), C1

Schedule 3: W1(A), W2(B), W3(C), R1(X), R2(X), R1(Y), W1(X), C1, W2(Y), C2, W3(Y), C3

For each schedule answer the following question:

1. Is this schedule conflict serializable? If yes, provide an equivalent serial schedule of transactions.
2. Does the schedule avoid cascading aborts? Explain your answer.
3. Is it possible under strict 2PL? Explain your answer. Remember that the 2PL protocol acquires an exclusive lock for writing and a shared lock for reading. If strict 2PL leads to a deadlock, give the wait-for graph.

* Schedule 1
  + It is conflict serializable

|  |  |
| --- | --- |
| T1 | T2 |
| R1(X) |  |
| W1(X) |  |
| R1(Y) |  |
| W1(Y) |  |
| C1 |  |
|  | R2(X) |
|  | W2(X) |
|  | R2(Y) |
|  | W2(Y) |
|  | C2 |

* + It avoids cascading aborts because there is no instance where the same transaction reads an uncommitted change.
  + It is possible under 2PL because it is able to write before the next transaction needs to read and so it does not cause a deadlock.
* Schedule 2
  + It is conflict serializable.

|  |  |  |
| --- | --- | --- |
| T1 | T2 | T3 |
|  |  | R3(Y) |
|  |  | W3(Y) |
|  |  | R3(Z) |
|  |  | W3(Z) |
|  |  | C3 |
|  | R2(X) |  |
|  | W2(X) |  |
|  | R2(Z) |  |
|  | W2(Z) |  |
|  | C2 |  |
| R1(X) |  |  |
| W1(X) |  |  |
| C1 |  |  |

* + It does avoid cascading aborts because each transaction commits before the next transaction needs to read.
  + It is possible under 2PL because it is able to write before the next transaction needs to read so it does not cause a deadlock.
* Schedule 3
  + It is not conflict serializable
  + It does avoid cascading aborts because there are no reads for a transaction that has been modified before a commit.
  + It is possible under 2PL because each transaction is able to write before another transaction needs to read.