1. Mardel Maduro
2. Student # 3349958
3. COMP 418
4. Assignment 3

## Part 1. Concepts and principles

### **Question 1**

**Define ACID properties, and briefly explain the methods used to deal with them in a DBMS.**

- The ACID properties (Atomicity, Consistency, Isolation, Durability) refer to the 4 most important properties of data transactions in terms of maintaining data when dealing with concurrent access and system failures. These properties are as follows:

1)Atomicity – Users should not have to worry about if actions were partially completed, they should either be done in full or not at all.

2)Consistency – Each transaction must preserve the consistency of the database

3)Isolation – Each transaction must be understandable as its own thing by the user

4)Durability – Each transaction’s effect must persist, even in the even of system failure

Atomicity & Durability are both handled by a component in the Database system called the recovery manager. Isolation is handled by a serializable schedule and consistency is handled by the user making sure that transactions that are entered leave the database in a consistent state when they are ran to completion.

### **Question 2**

**Why is concurrent execution of transactions important in a DBMS? Define the anomalies that might happen if concurrent execution of conflicting actions is not handled properly.**

- Concurrent execution of transactions are important in DBMS because it allows for better throughput and the ability to execute multiple transactions at the same time. This means we have better uses of the resources available to us.

The anomalies that might happen if concurrent execution of conflicting actions is not handled properly are:

1) Reading Uncommitted Data (WR Conflicts) – reading a value that was changed by a transaction before that transaction has committed the value.

2) Unrepeatable Reads (RW Conflicts) – changing values that have been read by another transaction before said transaction is completed.

3) Overwriting Uncommitted Data (WW Conflicts) – overwriting values currently being used by a transaction that isn’t completed

### **Question 3**

**Explain how timestamps are used for concurrency control and deadlock prevention. Also explain how does the Thomas write rule improve concurrency?**

-Timestamps are used for concurrency control and deadlock prevention by having it set up so that timestamps are added to every database object and they are used to keep the transactions in order by keeping track of the locks for each. This is used to ensure serializability. Timestamps can also be used to make sure that transactions don’t conflict with each other. This is done by giving priorities to processes, the lower the timestamp the higher the priority (older processes go first).

If a transaction Ti request a lock and transaction Tj holds a conflicting lock, the lock manager can use one of following policies:

1) Wait-die : If Ti has a higher priority , it is allowed to wait; otherwise it is aborted.

2) Wound-wait : If Ti has higher priority, abort Tj ; otherwise , Ti waits.

Thomas write rule can be used to improve concurrency by spending less time looking at old write requests, ignore them and only perform the most recent write since it was going to overwrite the outdated one/s anyway.

### **Question 4**

**Compare deadlock detection schemes and deadlock prevention schemes, and explain why detection schemes are more commonly used.**

-Deadlock detection schemes involve the use of Wait-For graphs and checking them for cycles. These schemes work by attempting to find and resolve actual deadlocks after they already occur. On the other hand, deadlock prevention schemes work by making sure that one of the four conditions needed for a deadlock to occur never happen, thus making sure the deadlock doesn’t occur.

The deadlock detection schemes are more commonly used since it is easier to check if a deadlock occurred and fix it than it is to predict the possibility of one occurring. As such since they are easier to implement, deadlock detection schemes are more commonly used.

### **Question 5**

Briefly explain the WAL protocol.

- The WAL (Write Ahead Log) protocol is used to ensure data atomicity and durability. The idea behind it is, we update the record log with a transaction before actually performing the transaction. This is done, so that in the event of a system crash, a recovery to the correct state can be performed. The log is used on recovery to check if the last transaction was fully completed, partially completed or not started and depending on which performs the necessary actions to put the system into the state that it should be in.

## Part 2. Design considerations and calculation

### **Question 1**

One morning the chief information officer (CIO) called an emergency meeting of the system administration team. As a database administrator, you are part of this team. The meeting was about a data inconsistency problem in a newly launched web application. The database in question deals with items and their prices. Your colleague in charge of the problematic database noticed that the phantom problem is happening while the application is following the strict 2PL protocol. He explained that while a first transaction (T1) has locked all existing items of type 1, a second transaction (T2) was able to add new items of type 1 at the same time, which caused the average price calculated by T1 to be incorrect when displayed to the user.

The CIO asked the rest of you to check whether the same problem is happening in the databases you are in charge of.

Can this Phantom problem occur in any of your databases where the set of database objects is fixed and only the values of objects can be changed? Explain.

- No this phantom problem can’t occur if the set of database objects are fixed and only the values of the objects can be changed. This is due to the fact that it wouldn’t be possible to add new objects into the database, we would only be able to modify values that are already there.

### **Question 2**

As a database administrator of a large manufacturing organization, you are in charge of their website database that handles suppliers, parts, catalog, customers, and payments of delivered items. The following relations are part of the large database:

Supplier(SID, Sname, Saddress, Stelephone)  
Customer(CID, Cname, Caddress, Ctelephone)  
Part(PID, Pname, Pmodel, Pcolor)  
Catalog(SID, PID, Price)

(The Catalog relation lists the prices charged for parts by Suppliers).

Because the application is web-based and the database is accessed by many users, you want to increase the concurrency usage without compromising the data quality. For each of the following transactions, state the minimum SQL isolation level you would use in order to avoid any conflict problems in them. Please explain your choice.

1. A transaction that adds a new part to a supplier’s catalog.

-Isolation Level: Read Uncommitted

Why: No need to lock the other rows since they aren’t needed for adding a new one. Dirty, unrepeatable and phantom reads aren’t going to be a problem since the part isn’t present in system for changes as yet.

1. A transaction that increases the price that a supplier charges for a part.

-Isolation Level: Repeatable Read

Why: this query would require that the value for the part’s price not be changed by any other transaction at the moment and as such we need it to be locked. We need to prevent dirty and unrepeatable reads, but phantom reads shouldn’t be a problem since this is an update.

1. A transaction that determines the total number of items for a given supplier.

-Isolation Level: Serializable

Why: To get the most accurate results we need for the system to prevent any changes from occurring with any of the relations that are needed for this query. We would need to prevent dirty, unrepeatable and phantom reads.

1. A transaction that shows, for each part, the supplier who has the part at the lowest price.

-Isolation Level: Serializable

Why: To get the most accurate results we need for the system to prevent any changes from occurring with any of the relations that are needed for this query. We would need to prevent dirty, unrepeatable and phantom reads.

### **Question 3**

A new application is being developed and will be using a database that includes a relation about items: Item (item\_id:integer, item\_name:string, color:string, price:real).

Both the purchasing department in charge of obtaining raw material and the manufacturing department in charge of manufacturing the items can change the price of manufactured items according to changes that may happen in the raw material cost or production cost. The two departments use different transactions to update the price of items. The new application uses the following sequences of actions, listed in the order they are submitted to the DBMS:

Sequence S1: T1:R(X), T2:W(X), T2:W(Y), T3:W(Y), T1:W(Y), T1:Commit, T2:Commit, T3:Commit

Sequence S2: T1:R(X), T2:W(Y), T2:W(X), T3:W(Y), T1:W(Y), T1:Commit, T2:Commit, T3:Commit

For each of the following concurrency control mechanisms, describe how they will handle each of the sequences (S1 & S2).

1. **Strict 2PL with timestamps used for deadlock prevention.**

Assuming the use of aWait-Die policy

- **S1:**

T1 gets shared lock on X

T2 aborts on request for X since it is of lower priority than T1

T2 is aborted and as such doesn’t make request for Y

T3 gets exclusive lock on Y

T1 waits for Y from T3 since T1 has a higher priority than T3

T3 completes transaction, commits and releases lock on Y, since Y is now released T1 wakes up

T1 get exclusive lock on Y, completes transaction, commits and releases locks on both X & Y

T2 restarts, completes itself, commits and releases X & Y

**-S2:**

T1 gets shared lock on X

T2 gets exclusive lock on Y

T2 requests exclusive lock on X, but is aborted since X is held by T1, and T2 has a lower priority than T1. Lock on Y is released

T3 gets exclusive lock on Y

T1 request exclusive lock on Y, but is made to wait since Y is held by T3, and T1 has a higher priority than T3

T3 completes transaction, commits and releases lock on Y. Release of Y wakes up T1

T1 completes transaction and releases locks on X & Y.

T2 restarts, runs to completion, commits and releases X & Y

1. Conservative (and Strict, i.e., with locks held until end-of-transaction) 2PL.

**-S1:**

T1 get shared lock on X and exclusive lock on Y

T1 completes transaction, commits and releases X & Y

T2 gets exclusive locks on X & Y

T2 completes transaction, commits and releases X & Y

T3 gets an exclusive lock on Y

T3 completes transaction, commits and releases Y

**-S2: (same as S1)**

T1 get shared lock on X and exclusive lock on Y

T1 completes transaction, commits and releases X & Y

T2 gets exclusive locks on X & Y

T2 completes transaction, commits and releases X & Y

T3 gets an exclusive lock on Y

T3 completes transaction, commits and releases Y

**Note:** Assume that the DBMS processes actions in the order shown. If a transaction is blocked, assume that all its actions are queued until it is resumed; the DBMS continues with the next action (according to the listed sequence) of an unblocked transaction. Write any other assumption you are using for your answer.

### **Question 4**

Using the following execution shown below, explain what is done in each of the ARIES recovery algorithm phases:

**Analysis** -

|  |  |  |
| --- | --- | --- |
| **LSN** | **LOG** |  |
| 00 | begin\_checkpoint | Indicates when checkpoint begins |
| 10 | end\_checkpoint | Contains current Transaction Table & Dirty Page Table |
| 20 | update: T1 writes P1 | Add (T1,20) to Transaction Table, Add (P1,20) to Dirty Page Table |
| 30 | update: T2 writes P2 | Add (T2,30) to Transaction Table, Add (P2,30) to Dirty Page Table |
| 40 | update: T3 writes P3 | Add (T3,40) to Transaction Table, Add (P3,40) to Dirty Page Table |
| 50 | T2 commit | Change status of T2 to C in Transaction Table |
| 60 | update: T3 writes P2 | Change (T3,40) to (T3,60) in Transaction Table |
| 70 | T2 end | Remove T2 from Transaction Table |
| 80 | update: T1 writes P5 | Change (T1,20) to (T1,70) in Transaction Table, Add (P5,70) to Dirty Page Table |
| 90 | T3 abort | No action |
|  | CRASH, RESTART |  |

In addition to the execution shown here, the system crashes during recovery after writing two log records to stable storage and again after writing another two log records.

1. After Analysis

|  |  |
| --- | --- |
| Transaction Table | Dirty Page Table |
| (T1,80), (T3,60) | (P1,20), (P2,30), (P3,40), (P5,80) |

**Redo Phase**

|  |  |
| --- | --- |
| LSN |  |
| 20 | No action (P1 > 10) |
| 30 | Redo change made in P2 |
| 40 | Redo change made in P3 |
| 50 | No action |
| 60 | Redo changes made in P2 |
| 70 | No action |
| 80 | Redo changes made in P5 |
| 90 | No action |

**Undo Phase**

-toUndo consists of (80,60)

LSN 80 -> Undo changes in P5, Appending a CLR : Undo T1 LSN 80, set undonextLSN = 20. Add 20 to toUndo.

-toUndo consists of (60,20)

LSN 60 -> Undo changes in P2, Append a CLR: Undo T3 LSN 40, set undonextLSN = 40. Add 40 to toUndo.

-toUndo consists of (40,20)

LSN 40 -> Undo changes in P3, Append a CLR: Undo T2 LSN 40, T3 end

-toUndo consists of (20)

LSN 20 -> Undo changes in P1, Append a CLR: Undo T1 LSN 20, T1 end