ST. XAVIER’S COLLEGE

**(Affiliated to Tribhuvan University)**

**Maitighar, Kathmandu**

****

**Database Management System**

**TheoryAssignment #8**

**SUBMITTED BY:**

**Utsav Luitel**

**013BSCCSIT046**

**SUBMITTED TO:**

|  |  |
| --- | --- |
| **Er. Sanjay Kr. Yadav**  **Lecturer** |  |
| **Department of Computer Science** | |

Database Concurrency Control.

1. Purpose of concurrency control
2. Two Phase locking
3. limitation of CCMs
4. Time-stamps-based-protocols
5. Commits protocols
6. Index Locking
7. Lock Granurality
8. Time stamp ordering multi-version concurrency control
9. DeadLock Handeling detection & Control.

Purpose of concurrency control

In information technology and computer science, especially in the fields of computer programming, operating systems, multiprocessors, and databases, **concurrency control** ensures that correct results for concurrent operations are generated, while getting those results as quickly as possible.

Concurrency control is a database management systems (DBMS) concept that is used to address conflicts with the simultaneous accessing or altering of data that can occur with a multi-user system. Concurrency control, when applied to a DBMS, is meant to coordinate simultaneous transactions while preserving data integrity. The Concurrency is about to control the multi-user access of Database.

Two Phase locking

Locking mechanisms are a way for databases to produce sequential data output without the sequential steps. The locks provide a method for securing the data that is being used so no anomalies can occur like lost data or additional data that can be added because of the loss of a transaction. Problems that Locks Solve Lost Update Problem- An update can get lost if two or more transactions try to update the same data. The two transactions are unaware of each other, and data can be overwritten. Temporary Update Problem- If one transaction updates the database and then fails, another transaction can read the incorrect value which lowers the integrity of the database. Incorrect Summary Problem- If one transaction is calculating an aggregate while another transaction is updating the same data the integrity would be compromised. Phantom Reads- When an insert or delete is performed on a row that is being used by another transaction the integrity of the data is compromised.

Limitation of CCMs

By implementing a few measures such as data sanitization and the limitation of executable queries and analyses. Data sanitization can be defined as the process…

Today’s level of ever evolving computer, database, and internet technology has enabled the collection and mining of data, as well as the utilization of that data on a level that was previously unimaginable. Each individual exists in a numerous databases around the world, from purchases made at the local supermarket to purchases made on-line to confidential medical records to credit information, etc. While individually the information that resides within the various databases listed above may not reveal much about a person, access to several databases may provide a detailed and possibly invasive amount of personal information. Inherently, there is a fundamental trade off between the functionality of a database or a database management system (DBMS) and the level of privacy given to the subjects of the database. While there are many benefits to advancements in database and DBMS technology, its advent has also created the possibility for significant abuses. Ideally, the design and implementation of a database would be constructed in a manner that will allow users to obtain and analyze information from a database(s) without allowing its users to access subjects’ private information. This problem can be dramatically reduced by implementing a few measures such as data sanitization and the limitation of executable queries and analyses. Data sanitization can be defined as the process of removing sensitive information from a document or other medium, so that it may be distributed to a broader audience. Sanitization attempts to reduce the personal content present in a database, while at the same time retaining enough functionality to supply the reader/user with the necessary information. The concept of the limitation of analyses is based on providing the user with either a limited amount of preset queries or query variables by which they must operate and/or limiting the number of queries and individual user may execute.

Time-stamps-based-protocols

Warrant – When data is entered into a data warehouse it is **time stamped**, or given a time ID, which can’t be changed. This promotes accurate back reports…

A timestamp is a unique identifier created by the DBMS to identify a transaction. Typically, timestamp values are assigned in the order in which the transactions are submitted to the system, so a timestamp can be thought of as the *transaction start time.*We will refer to the timestamp of transaction T as TS(T). Concurrency control techniques based on timestamp ordering do not use locks; hence, *deadlocks cannot occur.*

Generation of Timestamp:

Timestamps can be generated in several ways. One possibility is to use a counter that is incremented each time its value is assigned to a transaction. The transaction timestamps are numbered 1, 2, 3, . . . in this scheme. A computer counter has a finite maximum value, so the system must periodically reset the counter to zero when no transactions are executing for some short period of time. Another way to implement timestamps is to use the current date/time value of the system clock and ensure that no two timestamp values are generated during the same tick of the clock.

Commits protocol.

In computer networking and databases, the three-phase **commit protocol** (3PC) is a distributed algorithm which lets all nodes in a distributed system agree to **commit** a transaction.

Index locking

**Index locking** is a technique used to maintain **index** integrity. A portion of an **index** is locked during a database transaction when this portion is being accessed by the transaction as a result of attempt to access related user data.

Lock granularity

The optimistic concurrency model is handled using row versioning. The system automatically chooses the appropriate **lock granularity**. A row is the smallest resource that can be **locked**. The support of row-level **locking** includes both data rows and index entries

DeadLock Handeling detection & Control.

In a multi-process system, deadlock is an unwanted situation that arises in a shared resource environment, where a process indefinitely waits for a resource that is held by another process.

For example, assume a set of transactions {T0, T1, T2, ...,Tn}. T0 needs a resource X to complete its task. Resource X is held by T1, and T1 is waiting for a resource Y, which is held by T2. T2 is waiting for resource Z, which is held by T0. Thus, all the processes wait for each other to release resources. In this situation, none of the processes can finish their task. This situation is known as a deadlock.

Deadlocks are not healthy for a system. In case a system is stuck in a deadlock, the transactions involved in the deadlock are either rolled back or restarted.

Deadlock Prevention

To prevent any deadlock situation in the system, the DBMS aggressively inspects all the operations, where transactions are about to execute. The DBMS inspects the operations and analyzes if they can create a deadlock situation. If it finds that a deadlock situation might occur, then that transaction is never allowed to be executed.

There are deadlock prevention schemes that use timestamp ordering mechanism of transactions in order to predetermine a deadlock situation.

Wait-Die Scheme

In this scheme, if a transaction requests to lock a resource (data item), which is already held with a conflicting lock by another transaction, then one of the two possibilities may occur −

* If TS(Ti) < TS(Tj) − that is Ti, which is requesting a conflicting lock, is older than Tj − then Ti is allowed to wait until the data-item is available.
* If TS(Ti) > TS(tj) − that is Ti is younger than Tj − then Ti dies. Ti is restarted later with a random delay but with the same timestamp.

This scheme allows the older transaction to wait but kills the younger one.

Wound-Wait Scheme

In this scheme, if a transaction requests to lock a resource (data item), which is already held with conflicting lock by some another transaction, one of the two possibilities may occur −

* If TS(Ti) < TS(Tj), then Ti forces Tj to be rolled back − that is Ti wounds Tj. Tj is restarted later with a random delay but with the same timestamp.
* If TS(Ti) > TS(Tj), then Ti is forced to wait until the resource is available.

This scheme, allows the younger transaction to wait; but when an older transaction requests an item held by a younger one, the older transaction forces the younger one to abort and release the item.

In both the cases, the transaction that enters the system at a later stage is aborted.

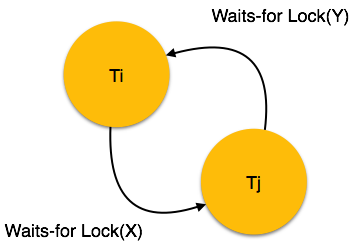
Deadlock Avoidance

Aborting a transaction is not always a practical approach. Instead, deadlock avoidance mechanisms can be used to detect any deadlock situation in advance. Methods like "wait-for graph" are available but they are suitable for only those systems where transactions are lightweight having fewer instances of resource. In a bulky system, deadlock prevention techniques may work well.

Wait-for Graph

This is a simple method available to track if any deadlock situation may arise. For each transaction entering into the system, a node is created. When a transaction Ti requests for a lock on an item, say X, which is held by some other transaction Tj, a directed edge is created from Ti to Tj. If Tj releases item X, the edge between them is dropped and Ti locks the data item.

The system maintains this wait-for graph for every transaction waiting for some data items held by others. The system keeps checking if there's any cycle in the graph.



Here, we can use any of the two following approaches −

* First, do not allow any request for an item, which is already locked by another transaction. This is not always feasible and may cause starvation, where a transaction indefinitely waits for a data item and can never acquire it.
* The second option is to roll back one of the transactions. It is not always feasible to roll back the younger transaction, as it may be important than the older one. With the help of some relative algorithm, a transaction is chosen, which is to be aborted. This transaction is known as the **victim** and the process is known as **victim selection**.