

Transactions Activity – Monday November 28

Problem 1: Transaction Isolation Levels

Consider a table **Football(college, league, rank)**. Write two transactions, X_1 and X_2 , consisting of SQL queries and/or updates over **Football**. Assume that transaction X_1 always executes with isolation level **Serializable**, and that both transactions always run to completion. Your transactions should be designed so that their possible behaviors are different for the four possible isolation levels for transaction X_2 :

- **Serializable**
- **Repeatable Read**
- **Read Committed**
- **Read Uncommitted**

In addition to specifying your two transactions, give an initial state of table **Football** so that the values retrieved in **Select** statements by one or both of the transactions, and/or the final state of table **Football**, may be different for each of the four isolation levels for transaction X_2 .

Problem 2: Implementing Transactions via Locking

Consider a locking protocol intended to enforce the various isolation levels offered to transactions. Assume each transaction T acquires the necessary locks just before each operation it performs, retains acquired locks for the duration of transaction T , then releases all of its locks when T either commits or rolls back. There are three types of locks:

- A **READ** lock on a tuple t – records that a transaction is reading tuple t
- A **WRITE** lock on a tuple t – records that a transaction is updating or deleting tuple t
- An **INSERT** lock on a table T – records that a transaction is inserting into table T

Here's the proposed protocol. In each group after the first one, the italicized line indicates what's changed from the previous group.

SERIALIZABLE transactions

- 1) To perform any operation on a table: no other **INSERT** lock
- 2) To read a tuple: no other **WRITE** lock, set **READ** lock
- 3) To update or delete a tuple: no other **READ** or **WRITE** lock, set **WRITE** lock
- 4) To insert a tuple: set **INSERT** lock

REPEATABLE READ transactions

- 1) *No global checking for **INSERT** locks*
- 2) To read a tuple: no other **WRITE** lock, set **READ** lock
- 3) To update or delete a tuple: no other **READ** or **WRITE** lock, set **WRITE** lock
- 4) To insert a tuple: set **INSERT** lock

READ COMMITTED transactions

- 1) No global checking for **INSERT** locks
- 2) *To read a tuple: no other **WRITE** lock*
- 3) To update or delete a tuple: no other **READ** or **WRITE** lock, set **WRITE** lock
- 4) To insert a tuple: set **INSERT** lock

READ UNCOMMITTED transactions

- 1) No global checking for **INSERT** locks
- 2) *To read a tuple: no checking or setting*
- 3) To update or delete a tuple: no other **READ** or **WRITE** lock, set **WRITE** lock
- 4) To insert a tuple: set **INSERT** lock

Unfortunately this implementation protocol does not work. Find as many bugs in the protocol as you can. A bug is demonstrated by showing two transactions X_1 and X_2 that follow the protocols for their isolation levels, but some possible behavior when the transactions are executed concurrently does not conform to the requirements of the isolation levels. Some notes:

- We don't consider the possibility of deadlock to be a bug.
- We're not concerned about the protocols being too conservative – i.e., imposing more restrictions than necessary – but rather when the protocols don't enforce the required consistency.
- Your examples may include transactions that commit and/or rollback, but try to find at least one bug that does not involve rollback.
- Your examples may include insertions, deletions, and/or updates, but try to find at least one bug that does not involve insertions. (Disclaimer: We haven't found one yet ourselves, but we don't know that there isn't one either!)