INM370 – Advanced Databases

Tutorial 4 – Relational Database Management Issues

1) Consider the following:

In SQL it is possible to set the isolation level of a transaction. More specifically, it is possible to specify to what degree a transaction can be isolated from the effects of other (concurrent) transactions. There are four types of isolation levels, specified in SQL standard, which[1] can be set for a certain transaction as defined below.
Please refer to week 3 slides, as well as the following resource – https://docs.microsoft.com/en-us/sql/t-sql/statements/set-transaction-isolation-level-transact-sql?view=sql-server-2017– for more information and examples. Please note that this resource considers the Isolation Levels available in the Microsoft SQL (MSSQL) DBMS. The Isolation Levels in MSSQL are more akin the ones specified in the SQL standard. Oracle, on the other hand, has its own interpretation and implementation of the Isolation Levels, as discussed in the lectures; for example see Section 22.5 from the Connolly and Begg book (6th Ed.) and/or the following URL from the original Oracle documentation: https://docs.oracle.com/en/database/oracle/oracle-database/21/cncpt/data-concurrency-and-consistency.html

- Isolation level READ UNCOMMITTED: A transaction can have access to data that is changed by other concurrent transaction(s) even if the other transaction(s) abort(s).

- Isolation level READ COMMITTED: This is usually the default isolation level (e.g. in MSSQL and Oracle). A transaction has access to committed data. Each operation in a transaction T only sees data that was committed before the operation started (and not the transaction). But non-repeatable read anomaly is possible.

- Isolation level REPEATABLE READ: A transaction can only access committed data with the guarantee that the data read cannot change for the duration of the transaction. However, this isolation level does not avoid *phantoms* – when a transaction re-executes a query returning a set of rows that satisfy a search condition, it finds that the set of rows satisfying the condition has changed due to another concurrent transaction that recently committed.

- Isolation level SERIALIZABLE: This case provides the highest level of isolation. In this case, serialisability is enforced at the transaction level.

  The isolation levels can be set in SQL using the following statement:

```
SET TRANSACTION
[ISOLATION LEVEL READ UNCOMMITTED |
READ COMMITTED | REPEATABLE READ | SERIALIZABLE]
```

Consider relation Item (ItemNum, Name, Price) with *primary key* ItemNum and the following initial values:

ItemNum	**Name**	**Price**

[1] There are non-standardised ones too, some such are in widespread use, e.g. Snapshot Isolation (SI)

I1	Pencil	20
I2	Pen	30

Suppose two concurrent transactions T1 and T2 below, with respective operations:
T1:

```
BEGIN Transaction;
O1: INSERT INTO Item VALUES (‘I3’, ’Notebook’, 40);
O2: UPDATE Item SET Price = Price + 30 WHERE Name=’Pencil’;
COMMIT;
```

T2[2]:

```
BEGIN Transaction;
O3: SELECT AVG (Price) AS a1 FROM Item;
O4: SELECT AVG (Price) AS a2 FROM Item;
COMMIT;
```

Assume that the individual operations O1, O2, O3 and O4 are executed atomically. Suppose that each transaction runs once and commits. Assume locking approach is used to implement concurrency control. Assume that transaction T1 always executes with isolation level SERIALIZABLE.

a) If transaction T2 is executed with isolation level SERIALIZABLE, state the possible schedules of operations allowed by the concurrency control mechanism, and the possible pairs of values for a1 and a2 that are returned by transaction T2?

b) If transaction T2 is executed with isolation level REPEATABLE READ, state the possible schedules of operations allowed by the concurrency control mechanism, and the possible pairs of values for a1 and a2 that are returned by transaction T2?

c) If transaction T2 is executed with isolation level READ COMMITTED, state the possible schedules of operations allowed by the concurrency control mechanism, and the possible pairs of values for a1 and a2 that are returned by transaction T2?

d) If transaction T2 is executed with isolation level READ UNCOMMITTED, state the possible schedules of operations allowed by the concurrency control mechanism, and the possible pairs of values for a1 and a2 that are returned by transaction T2?

You should use transaction BEGIN statements (e.g. B1 and B2), and transaction COMMIT statements (e.g. C1 and C2) when considering whether schedules are allowed, or not.

2) Consider the partial transaction schedules below. For each schedule: i) specify which transaction holds which lock (including what is the type of the lock), and which lock requests are denied, and ii) create the wait-for graph, label the arcs with the locks requested, and iii) state if the schedule is, or not, deadlock free. Consider this a shared/exclusive lock system. Assume that rigorous 2PL is used, and that the transactions are executing using serializable isolation level.

a) r1(A); r2(A); r3(B); r1(C); w1(A); r2(B); w2(B); w3(A);
b) r2(A); w2(B); r1(C); r1(B); r3(D); w3(C); w4(D)
c) w3(A); w1(B); r1(A); w2(C); r2(B); r3(C)

[2] The example is meant to help you understand the transaction isolation levels in a most effective way, irrespective of how realistic the given transactions might be in practice.

Assume the following:
w – a write operation;
r – a read operation
1, 2, 3, 4 – transactions T1, T2, T3 and T4, respectively
A, B, C, D – the data items being read or written by the respective transaction

3) Consider the following schedule of database operations from two transactions:

   W1(A); R2(A); W2(B); c2; R1(A); c1;

   Where:
   - R represents a Read command;
   - W represents a Write command;
   - 1, 2 represent respectively transactions T1 and T2;
   - A and B represent respectively data items A and B;
   - c1, c2 represent commits of transactions T1 and T2

   Answer the following questions. Justify your answers.
   a) Is it possible to execute this schedule using *rigorous two-phase locking*?
   b) Is it possible to execute this schedule using timestamp protocol for concurrency control assuming timestamp for T1 = 1 (older transaction) and timestamp for T2 = 2 (younger transaction)?

   Justify your answers.

4) Suppose you are the owner of relation `Employee (EmpNum, Salary, Department)`. Suppose you want to authorize user John to be able to retrieve, but not modify, employee information for those employees who earn less than £50,000 and work in a department with less than 10 people. Specify SQL statements that implement the above.

5) Consider a relation R (A, B, C). Suppose that Amy is the owner of this relation. Consider the following sequence of statements related to privileges on R[3]. The names in front of the statements represent the users issuing/withdrawing the privileges. What are Eve's privilege(s) on relation R, after this sequence of statements, if any?

Amy: `GRANT SELECT, DELETE on R to Bob WITH GRANT OPTION`
Amy: `GRANT SELECT, DELETE on R to Carol WITH GRANT OPTION`
Bob: `GRANT SELECT (A,B), DELETE on R to David WITH GRANT OPTION`
Carol: `GRANT SELECT (A,C) on R to David WITH GRANT OPTION`
David: `GRANT SELECT (A), DELETE on R to Eve`
Amy: `REVOKE SELECT, DELETE on R from Bob CASCADE`

[3] Assume that the *PrivilegeList* can include privileges for **specific** columns for SELECT, as well as INSERT and UPDATE SQL statements.