

CS143 Notes: TRANSACTION

Book Chapters

(5th) Chapters 15.1-4, 15.7-8

(6th) Chapters 14.1-5, 14.7-8, 14.10

(7th) Chapter 17.1-5, 17.7-8, 17.10

MOTIVATION FOR TRANSACTION

1. Crash recovery

- \langle eg, Transfer \$1M from Susan to Jane \rangle (example slide)
 - S_1 : UPDATE Account SET balance = balance - 1000000 WHERE owner = ‘Susan’
 - S_2 : Update Account SET balance = balance + 1000000 WHERE owner = ‘Jane’
 - System crashes after S_1 but before S_2 . What now?

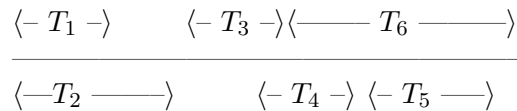
2. Concurrency

- We do not want to allow oncurrent access from multiple clients. We do not want to “lock out” the DBMS until one client finishes
 \langle explain with client/server diagram \rangle
- Can allow parallel execution while avoiding any potential problems from concurrency?
(we will see concurrency problem examples soon).

TRANSACTION AND “ACID” PROPERTY

- TRANSACTION: A sequence of SQL statements that are executed as a “unit”
- ACID PROPERTY OF TRANSACTION: Atomicity, Consistency, Isolation, Durability
 1. Atomicity: “ALL-OR-NOTHING”

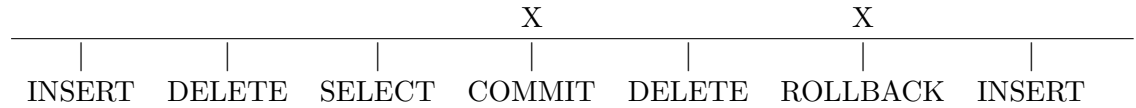
- Either ALL OR NONE of the operations in a transaction is executed.
 - If the system crashes in the middle of a transaction, all changes by the transaction are “undone” during recovery.
2. Durability
- After a balance transfer is “done”, the transfer silently “disappears” due to system crash. What will the customer think?
 - COMMIT: If a transaction “committed”, all its changes remain permanently even after system crash
 - * This guarantee may not be easy because some changes may be reflected only in memory for performance reasons
3. Isolation: Even if multiple transactions are executed concurrently, the result is the same as executing them in some sequential order.
- Each transaction is unaware of (is isolated from) other transaction running concurrently in the system
(explain by time line diagram)



4. Consistency: If the database is in a consistent state before a transaction, the database is in a consistent state after the transaction
- DBMS guarantees the ACID property for all transactions
 - With minor caveats that will be discussed later.
 - **Q:** How can the database system guarantee these? Any ideas?

DECLARING A TRANSACTION IN SQL

- Two important commands:
 - COMMIT: All changes made by the transaction is stored permanently
 - ROLLBACK: Undo all changes made by the transaction
- AUTOCOMMIT MODE
 1. With AUTOCOMMIT mode OFF
 - Transaction implicitly begins when any data in DB is read or written
 - All subsequent read/write is considered to be part of the same transaction
 - A transaction finishes when COMMIT or ROLLBACK statement is executed
(explain using time line diagram)



2. With AUTOCOMMIT mode ON

- Every SQL statement becomes one transaction
- Setting Autocommit mode:
 - In Oracle: SET AUTOCOMMIT ON/OFF (default is off)
 - In MS SQL Server: SET IMPLICIT_TRANSACTIONS OFF/ON (default is off)
 - * IMPLICIT_TRANSACTIONS OFF means AUTOCOMMIT ON in MS SQL Server
 - In MySQL: SET AUTOCOMMIT = {0|1} (default is on. InnoDB only)
 - In DB2: UPDATE COMMAND OPTIONS USING c ON/OFF (default is on)
 - In JDBC: connection.setAutoCommit(true/false) (default is on)
 - In Oracle, MS SQL Server, and MySQL, BEGIN temporarily disables autocommit mode until COMMIT or ROLLBACK

SQL ISOLATION LEVELS

- Motivation: In some cases, we may not need full ACID. We may want to allow some “bad” schedule to achieve more concurrency
 - SQL isolation levels allow a few “bad” scenarios for more concurrency
 - * dirty read, non-repeatable read, phantom
 - We go over three scenarios in which “relaxing” the strict ACID may be desirable for some applications

- ⟨explain the isolation levels through examples and fill in the table⟩

isolation level	dirty read	nonrepeatable read	phantom
read uncommitted			
read committed			
repeatable read			
serializable			

- DIRTY READ may be OK

- ⟨example⟩
 - * T_1 : UPDATE Employee SET salary = salary + 100
 - * T_2 : SELECT salary FROM Employee WHERE name = ‘John’
- **Q:** Under ACID, once T_1 update John’s salary, can T_2 read John’s salary?
 - * Sometimes, it may be okay for T_2 to proceed.
- DIRTY READ: a transaction reads uncommitted values
- “READ UNCOMMITTED” isolation level allows dirty read.
(Fill in the dirty read column)

- NON-REPEATABLE READ may be OK

- ⟨example⟩
 - * T_1 : UPDATE Employee SET salary = salary + 100 WHERE name = ‘John’
 - * T_2 : (S_1) SELECT salary FROM Employee WHERE name = ‘John’
 - ...
 - (S_2) SELECT salary FROM Employee WHERE name = ‘John’
- **Q:** Under ACID, can we get different values for S_1 and S_2 ?
 - * Sometimes it may be okay to get different values
- NON-REPEATABLE READ: When T_i reads the same row multiple times, T_i may get different values
- “READ UNCOMMITTED” or “READ COMMITTED” isolation levels allow NON-REPEATABLE READ.
(Fill in the non-repeatable read column)

- PHANTOM may be OK

- \langle example \rangle
 - * Initially, $\text{SUM}(\text{Employee.salary}) = \$100,000$
 - * T_1 : `INSERT INTO Employee (e1, 1000), (e2, 1000)`
 - * T_2 : `SELECT SUM(salary) FROM Employee`
- **Q:** Under ACID, what may T_2 return?

* Sometimes, it may be OK for T_2 to return \$101,000

- **Q:** Under REPEATABLE READ, what if T_2 is

`SELECT SUM(salary) FROM Employee`
`...`
`SELECT SUM(salary) FROM Employee`

What can T_2 return?

- PHANTOM: When new tuples are inserted, once some of them are seen by statements, or only some statements see the newly inserted tuples.
- Except for “SERIALIZABLE” isolation level, PHANTOM is always allowed.

- MIXED ISOLATION LEVELS

- \langle example on mixed isolation levels \rangle
 - * T_1 : `UPDATE Employee SET salary = salary + 100`
`ROLLBACK`
 - * T_2 : `SELECT salary FROM Employee WHERE name = 'John'`
- **Q:** T_1 - SERIALIZABLE, T_2 - SERIALIZABLE. What may T_2 return?

- **Q:** T_1 - SERIALIZABLE, T_2 - READ UNCOMMITTED. What may T_2 return?

- COMMENTS:
 - * Only when all transactions are serializable, we guarantee ACID.
 - * The isolation level is in the eye of the beholding transaction.

- READ ONLY TRANSACTION

- Many, many transactions are read only.
- By declaring a transaction as READ ONLY, we can help DBMS to optimize for more concurrency
- SQL ISOLATION LEVEL DECLARATION
 - SET TRANSACTION options
 - access mode: READ ONLY / READ WRITE (default: READ WRITE)
 - isolation level: ISOLATION LEVEL
 - * READ UNCOMMITTED
 - * READ COMMITTED (Oracle default)
 - * REPEATABLE READ (MySQL, DB2 default)
 - * SERIALIZABLE
 - e.g) SET TRANSACTION READ ONLY, ISOLATION LEVEL REPEATABLE READ
 - * READ UNCOMMITTED cannot be READ WRITE
 - * Needs to be declared before EVERY transaction for non-default settings