SQL: Transactions

CPS 196.3

Introduction to Database Systems

Transactions

- A transaction is a sequence of database operations with the following properties (ACID):
  - Atomic: Operations of a transaction are executed all-or-nothing, and are never left "half-done"
  - Consistency: Assume all database constraints are satisfied at the start of a transaction, they should remain satisfied at the end of the transaction
  - Isolation: Transactions must behave as if they were executed in complete isolation from each other
  - Durability: If the DBMS crashes after a transaction commits, all effects of the transaction must remain in the database when DBMS comes back up

SQL transactions

- A transaction is automatically started when a user executes an SQL statement
- Subsequent statements in the same session are executed as part of this transaction
  - These statements can see the changes made by earlier statements in this transaction
  - Statements in other concurrently running transactions should not see these changes
- `COMMIT` command commits the transaction
  - Its effects are made final and visible to subsequent transactions
- `ROLLBACK` command aborts the transaction
  - Its effects are undone

Fine prints

- Schema operations (e.g., `CREATE TABLE`) implicitly commit the current transaction
  - Because it is often difficult to undo a schema operation
- Sometime you need to turn off a feature called `AUTOCOMMIT`, which automatically commits every single statement
  - Example: Run DB2's `db2` command-line processor with the option `+c`
  - More examples to come when we cover database API's

Atomicity

- Partial effects of a transaction must be undone when
- How is atomicity achieved?
  - Logging

Isolation

- Transactions must appear to be executed in a serial schedule (with no interleaving operations)
- For performance, DBMS executes transactions using a serializable schedule
  - In this schedule, operations from different transactions can interleave and execute concurrently
  - But the schedule is guaranteed to produce the same effects as a serial schedule
- How is isolation achieved?
  - Locking, multi-version concurrency control, etc.

Consistency

- Consistency of the database is guaranteed by constraints and triggers declared in the database and/or transactions themselves
  - When inconsistency arises, abort the transaction or fix the inconsistency within the transaction

Durability

- Effects of committed transactions must survive DBMS crashes
- How is durability achieved?
  - Logging

SQL isolation levels

- Strongest isolation level: `SERIALIZABLE`
  - Complete isolation
  - SQL default
- Weaker isolation levels: `REPEATABLE READ`, `READ COMMITTED`, `READ UNCOMMITTED`
  - Increase performance by eliminating overhead and allowing higher degrees of concurrent
  - Trade-off: sometimes you get the "wrong" answer

Example schema

- CREATE TABLE Account
  (accno INTEGER NOT NULL PRIMARY KEY,
  name CHAR(30) NOT NULL,
  balance FLOAT NOT NULL CHECK(balance >= 0));

READ UNCOMMITTED

- Can read "dirty" data
  - A data item is dirty if it is written by an uncommitted transaction
- Problem:
- Example: wrong average

```
-- T1:                          -- T2:
UPDATE Account
SET balance = balance – 200
WHERE accno = 142857;           SELECT AVG(balance)
                                FROM Account;
ROLLBACK;
                                COMMIT;
```


READ COMMITTED

- No dirty reads, but non-repeatable reads possible
  - Reading the same data item twice can produce different results
- Example: different averages

```
-- T1:                          -- T2:
                                SELECT AVG(balance)
                                FROM Account;
UPDATE Account
SET balance = balance – 200
WHERE accno = 142857;
COMMIT;
                                SELECT AVG(balance)
                                FROM Account;
                                COMMIT;
```

REPEATABLE READ

- Reads are repeatable, but may see phantoms
- Example: different average (still!)
  - -- T1:

```
-- T1:                          -- T2:
                                SELECT AVG(balance)
                                FROM Account;
INSERT INTO Account
VALUES(428571, 1000);
COMMIT;
                                SELECT AVG(balance)
                                FROM Account;
                                COMMIT;
```


Summary of SQL isolation levels

Isolation level/anomaly	Dirty reads	Non-repeatable reads	Phantoms
READ UNCOMMITTED	Possible	Possible	Possible
READ COMMITTED	Impossible	Possible	Possible
REPEATABLE READ	Impossible	Impossible	Possible
SERIALIZABLE	Impossible	Impossible	Impossible

- Syntax: At the beginning of a transaction,
  `SET TRANSACTION ISOLATION LEVEL` *isolation_level* [`READ ONLY`|`READ WRITE`];
  - `READ UNCOMMITTED` can only be `READ ONLY`
- Criticized recently for definition in terms of anomalies
  - Berenson, Bernstein, Gray, et al. "A critique of ANSI SQL isolation levels," *SIGMOD* 1995