Transactions

Database Connectivity and Access

Transaction Handling

- Transactions
  - Units of work executed together
  - All or nothing
  - Ensure ACID compliance
- Important for any set of statements that needs to be executed as one operation
- Introduce complexity in applications
  - Contention!
  - Stale Data problems

When to use a transaction?

- Multiple operations that must happen together—YES
  - E.g. updating multiple tables with the shared / related information in separate steps
  - E.g. deleting records from both a child table and the parent table
- Single SQL statements—NO
  - Unnecessary overhead
- Multiple tables?
  - Can we use a JOIN – then NO transaction
  - Creating several temporary tables where source tables may change – YES need transaction

How to use a transaction

- Connect / Open database
- Start a transaction
- Execute database Select, Insert, Update, Delete commands AND any necessary code operations
- Commit or Rollback
- End Transaction
- Close database

Demo

AutoCommit

- What counts as a transaction when you're working at the console?

- Many DBMS have the concept of AutoCommit
  - If "on", every SQL operation is considered its own transaction and is committed
  - If "off", the entire session is one, single transaction

- But is AutoCommit on or off?
  - MySQL: AutoCommit is on by default
  - Oracle: AutoCommit is off by default

Executing Transactions

- Two Approaches
  - Manual
  - Object

Manual Transactions

- Manual Approach
  - Use execute() method and mimic console operation
    - Start transaction
    - Perform statements
    - Commit (or rollback) transaction

Executing Transactions (Manual)

```
// Oracle example

   conn.setAutoCommit(false);

// Any SQL or coding statements

   conn.commit();
```

Object Approach

- Object Approach
  - Create Transaction object
  - Set of statements
  - Execute
  - If everything is OK with transaction
    - Commit
  - else
    - Rollback

C# Transactions (Object)

```
cmd = new MySqlCommand();
cmd.Connection = conn;

MySqlTransaction trans = conn.BeginTransaction();
cmd.Transaction = trans;

    cmd.CommandText = something;
    cmd.ExecuteNonQuery();
    cmd.CommandText = something else;
    cmd.ExecuteNonQuery();

trans.Commit();
```

Java Transactions

- Autocommit is done at the connection level
- There is no Transaction object

```
        conn.setAutoCommit(false);   // start transaction
// a whole lot of SQL code
        if( everythingWorked )
                conn.commit();              // OK - commit
        else
                conn.rollback();            // Error - rollback
        conn.setAutoCommit(true);    // end transaction
```

Rollback

- Why roll back a transaction?
  - Intermediate values indicate a need to abort
  - Exception/Error thrown

- How to roll back a transaction?
  - conn.rollback() for Java

    OR
  - myTransObj.rollback() for C#

How NOT to use a transaction 1

1. Do NOT close the database within the transaction
   Once closed, cannot commit or rollback
2. Following a commit or rollback...
   Don't do a second commit or rollback
3. Following a catch that has a rollback, don't execute a commit (See #2 above)
4. The logic path should only have one possible commit or rollback, followed by end transaction
5. Setting the transaction back to true will do a commit, but letting it default is sloppy programming.
6. Sloppy programming is not accepted in this class or on the job

How NOT to use a transaction 2

7. Results from SELECT(s) obtained **before** a transaction should NOT be used within a transaction.
   → Dirty data
   ∴ Start the transaction before the SELECTs

Where Begin/End Tx?

- This is all pseudocode – NOT REAL CODE
- Where does begin tx, commit, rollback, end tx go?

1. Select MAX(codeId) from tableA
2. Get the resulting number into variable lastId
3. Add 1 to lastId
4. Insert into tableB (id, name) VALUES (lastId, nameX)
5. Add 1 to lastId
6. Insert into tableB (id, name) VALUES (lastId, nameY)
7. Catch SQL exceptions & any others