What Is A Transaction?

- A transaction maintains consistency of the database and the correspondence between the state of the database and the real-world
- ACID (Atomicity, Consistency, Isolation, and Durability) → essential properties of transactions

Transaction Support in SQL-92

- Each transaction has:
  - an access mode (read-only, read-write)
  - an isolation level

<table>
<thead>
<tr>
<th>Isolation Level</th>
<th>Dirty Read</th>
<th>Unrepeatable Read</th>
<th>Phantom Problem</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read uncommitted</td>
<td>Yes</td>
<td>May be</td>
<td>May be</td>
</tr>
<tr>
<td>Read committed</td>
<td>No</td>
<td>May be</td>
<td>May be</td>
</tr>
<tr>
<td>Repeatable Reads</td>
<td>No</td>
<td>No</td>
<td>May be</td>
</tr>
<tr>
<td>Serializable</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>

SQL Isolation Levels

- **Dirty Read**: Transaction T1 modifies a data item. Another transaction T2 then reads that data item before T1 performs a COMMIT or ROLLBACK. If T1 then performs a ROLLBACK, T2 has read a data item that was never committed and so never really existed.
- **Unrepeatable Read**: Transaction T1 reads a data item. Another transaction T2 then modifies or deletes that data item and commits. If T1 then attempts to re-read the data item, it receives a modified value or discovers that the data item has been deleted.
- **Phantom Problem**: Transaction T1 reads a set of data items satisfying some search condition. Transaction T2 then creates data items that satisfy T1’s search condition and commits. If T1 then repeats its read with the same search condition, it gets a set of data items different from the first read.
Transactions In JDBC

- **By default**, after each SQL statement is executed the changes are **automatically committed** to the database.
- In JDBC, one can **turn auto-commit off** to group two or more statements together into a transaction:
  - `Connection.setAutoCommit(false)`
- Guarantees that inserts / updates / deletes done inside a transaction are all or none (**Atomicity**)
- Call commit to permanently record the changes to the database after executing a group of statements:
  - `Connection.commit()`
- Call rollback if an error occurs:
  - `Connection.rollback()`

Transactions: Java Code

```java
Connection connection = DriverManager.getConnection(url, username, passwd);
connection.setAutoCommit(false);
try {
    statement.executeUpdate(...);
    statement.executeUpdate(...);
    connection.commit();
} catch (SQLException sqle) {
    try {
        connection.rollback();
    } catch (SQLException sqle1) {
        // report problem
    }
} finally {
    try {
        connection.close();
    } catch (SQLException sqle2) { }
}
```

either both or none will be executed

Transactions: Connection Methods

- `getAutoCommit() / setAutoCommit()`
  - By default, a connection is set to auto-commit
  - Retrieves or sets the auto-commit mode
- `commit()`
  - Force all changes since the last call to commit to become permanent
  - Any database locks currently held by this Connection object are released
- `rollback()`
  - Drops all changes since the previous call to commit
  - Releases any database locks held by this Connection object

Exception Handling- SQL Exceptions

- Nearly every JDBC method can throw a SQLException in response to a data access error
- If more than one error occurs, they are chained together
- SQL exceptions contain:
  - Description of the error: `getMessage()`
  - The SQLState (Open Group SQL specification) identifying the exception: `getSQLState()`
  - A vendor-specific integer, error code: `getErrorCode()`
  - A chain to the next SQLException: `getNextException()`
SQL Exceptions Example

```java
try {
    // JDBC statements
} catch (SQLException sqle) {
    while (sqle != null) {
        System.out.println("Message: " + sqle.getMessage());
        System.out.println("SQLState: " + sqle.getSQLState());
        System.out.println("Vendor Error: " + sqle.getErrorCode());
        sqle.printStackTrace(System.out);
        sqle = sqle.getNextException();
    }
}
```

Motivation (Concurency Control)

- A real world situation (Simplified case)
  Two users, A and B, want to register a web service. The id of each user should be unique. Assuming every time a new user is added, his/her id is the total number of existing users plus one.
- The above situation is modeled using table TEST. The registration information is stored in table TEST.
  - Create table TEST(id integer)

Motivation (Cont.)

- Possible Problems occur when two users try to register concurrently.
  - A: R(X) X=X+1 W(X)
  - B: R(X) X=X+1 W(X)

Try example
http://ugweb.cs.ualberta.ca/~c391/tutorial/examples/concur2.java

- Solution: One user can request an exclusive lock on the TEST table or on id, the other users can wait until the user who holds the lock commit or release the lock.

Locks in Oracle SQL

- We introduce two types of locks
  - Row Lock: lock the specific row and column in a table
    ```sql
    SELECT ...
    FOR UPDATE/DELETE/INSERT
    ```
  - Table Lock: Lock the whole table in exclusive mode
    ```sql
    LOCK TABLE ... IN EXCLUSIVE MODE
    ```

Try example
http://ugweb.cs.ualberta.ca/~c391/tutorial/examples/concur3.java

Note: You need Keyboard.java to run the example programs.
http://www-csfy.cs.ualberta.ca/~c114/W05/Tutorials/General/UofAC114/Keyboard.java
More tutorials on locks

- http://www.wisc.edu/drmt/oratips/sess004.html

Table locks

- http://dnaugler.cs.semo.edu/oracledocs/a96540/statements_914a.htm

Lab 7

Example(concur1.java)

```java
import java.io.*;
import java.sql.*;

public class concur1{
    public static void main(String args[]){
        String m_url = "jdbc:oracle:thin:@gwynne.cs.ualberta.ca:1521:CRS";
        String m_user = "****"; //replace this with your own id
        String m_pass = "****"; //replace this with your password
        String sql = "";
        Class drvClass = null;
        Connection m_con = null;
        Statement stmt = null;
        Statement stmt1 = null;
        ResultSet retset = null;
        /* create the connection to the underlying database*/
        try{
            drvClass = Class.forName("oracle.jdbc.driver.OracleDriver");
            m_con = DriverManager.getConnection(m_url,m_user,m_pass);
            catch(Exception ex){
                System.out.println("Connection Failed");
                System.out.println("Connection Error Message:"+ex.getMessage());
            }
            try{
                String m_user = "****"; //replace this with your own id
                String m_pass = "****"; //replace this with your password
                String sql = "";
                Class drvClass = null;
                Connection m_con = null;
                Statement stmt = null;
                Statement stmt1 = null;
                ResultSet retset = null;
                /* create the connection to the underlying database*/
                try{
                    drvClass = Class.forName("oracle.jdbc.driver.OracleDriver");
                    m_con = DriverManager.getConnection(m_url,m_user,m_pass);
                    catch(Exception ex){
                        System.out.println("Connection Failed");
                        System.out.println("Connection Error Message:"+ex.getMessage());
                    }
                    try{
                        int count = 0;
                        while(retset.next()) count++;
                        System.out.println("There are "+count+" rows in table TEST");
                        Keyboard.in.pause();
                        System.out.println("Now, I add one more row into table TEST");
                        stmt.executeUpdate("insert into TEST values(1++)");
                        m_con.commit();
                        stmt.close();
                        m_con.close();
                        }catch(SQLEXception ex) System.out.println("Update Message:"+ex.getMessage());
                    }
                }
            }
        }
    }
}
```

Example (concur3.java)

```java
public class concur3{
    public static void main(String args[]){
        // omitted code. Code is similar to concur1.java
        try{
            String m_user = "****"; //replace this with your own id
            String m_pass = "****"; //replace this with your password
            String sql = "";
            Class drvClass = null;
            Connection m_con = null;
            Statement stmt = null;
            Statement stmt1 = null;
            ResultSet retset = null;
            /* create the connection to the underlying database*/
            try{
               drvClass = Class.forName("oracle.jdbc.driver.OracleDriver");
                m_con = DriverManager.getConnection(m_url,m_user,m_pass);
                catch(Exception ex){
                    System.out.println("Connection Failed");
                    System.out.println("Connection Error Message:"+ex.getMessage());
                }
                try{
                    int count = 0;
                    retset.first();
                    count = retset.getInt(1);
                    Keyboard.in.pause();
                    System.out.println("Now, I update the id in table TEST");
                    stmt.executeUpdate("UPDATE test SET id=("+(++count)+")");
                    m_con.commit();
                    stmt.close();
                    m_con.close();
                    }catch(SQLEXception ex) System.out.println("Update Message:"+ex.getMessage());
                }
            }
        }
    }
}
```
Summary

- Transactions only guarantee all or none property of inserts / updates / deletes.
- Concurrency control is not always guaranteed by a transaction, need to consider read consistency.