Java Database Connectivity API

- Transactions
- PreparedStatement
- Examples

Transactions

A **transaction** is a sequence of operations performed as a single logical unit of work.

A logical unit of work must exhibit four properties, called the atomicity, consistency, isolation, and durability (ACID) properties, to qualify as a transaction.

Atomicity

A transaction must be an atomic unit of work; either all of its data modifications are performed, or none of them is performed.

Consistency

When completed, a transaction must leave all data in a consistent state.

Isolation

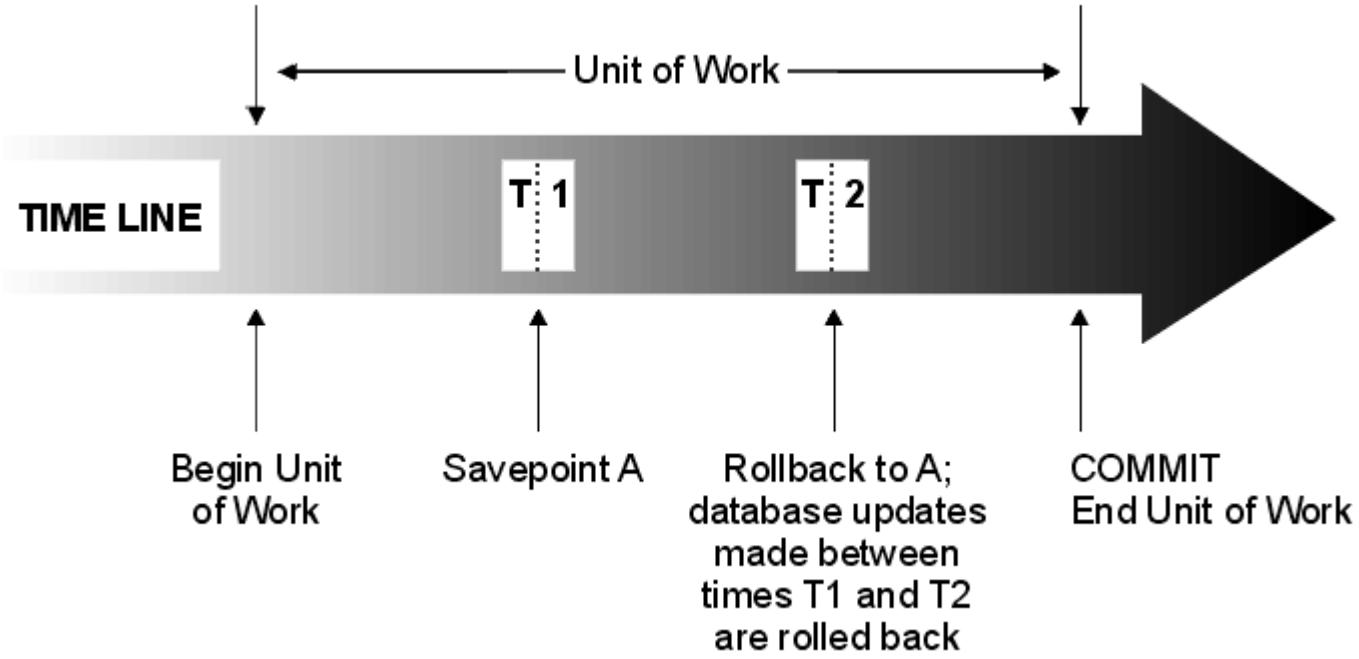
Modifications made by concurrent transactions must be isolated from the modifications made by any other concurrent transactions.

Durability

After a transaction has completed, its effects are permanently in place in the system. The modifications persist even in the event of a system failure.

Transactions are not feature of JDBC

Transaction mechanism must be supported by database



How to use transactions

JDBC supports transactions but by default they are turned-off

Changes applied to DB after execution of each statement

To turn-on transactions usage

connection.setAutoCommit(false);

To commit changes

connection.commit();

To rollback changes

connection.rollback();

```
public void execUpdate(Connection connection, String[] updates) {
        try {
                connection.setAutoCommit(false);
                for(String update: updates){
                        Statement stmt = connection.createStatement();
                        stmt.execute(update);
                        stmt.close();
                }
                connection.commit();
        } catch (SQLException e) {
                try {
                        connection.rollback();
                        connection.setAutoCommit(true);
                } catch (SQLException ignore) {}
        }
}
```

PreparedStatement

public interface PreparedStatement extends Statement {…}

Special type of precompiled Statement.

A SQL statement is precompiled and stored in a PreparedStatement object.

This object can then be used to efficiently execute this statement multiple times.

```
public void execUpdate(Connection connection, Map<Integer, String> idToName) {
    try{
        String update = "insert into users(id, user_name) values(?, ?)";
        PreparedStatement stmt = connection.prepareStatement(update);

        for(Integer id: idToName.keySet()){
            stmt.setInt(1, id);
            stmt.setString(2, idToName.get(id));
            stmt.executeUpdate();
        }
        connection.commit();
        stmt.close();
    } catch (SQLException e) {
        connection.rollback();
        e.printStackTrace();
    }
}
```