JDBC

Part 2

- Connection pool
- Transactions
- DAO

DataSource

- A DataSource object represents a particular DBMS or some other data source, such as a file
- The DataSource interface is implemented by a driver vendor.
- It can be implemented in three different ways:
  - A basic DataSource implementation produces standard Connection objects that are not pooled or used in a distributed transaction.
  - A DataSource implementation that supports connection pooling produces Connection objects that participate in connection pooling, that is, connections that can be recycled.
  - A DataSource implementation that supports distributed transactions produces Connection objects that can be used in a distributed transaction, that is, a transaction that accesses two or more DBMS servers.
- A DataSource class that supports distributed transactions typically also implements support for connection pooling.

Примеры

- http://dev.mysql.com/doc/refman/5.6/en/connector-j.html

Pitfalls of sharing a connection among threads

- Committing or rolling back a transaction closes all open *ResultSet* objects and currently executing *Statements*, unless you are using held cursors.
  - If one thread commits, it closes the *Statements* and *ResultSets* of all other threads using the same connection.
- Executing a *Statement* automatically closes any existing open *ResultSet* generated by an earlier execution of that *Statement*.
  - If threads share *Statements*, one thread could close another's *ResultSet*
- In many cases, it is easier to assign each thread to a distinct *Connection*. If thread *A* does database work that is not transactionally related to thread *B*, assign them to different *Connections*.
- Multiple threads are permitted to share a *Connection*, *Statement*, or *ResultSet*. However, the application programmer must ensure that one thread does not affect the behavior of the others.

Connection pooling

- Connection pooling is a technique of creating and managing a pool of connections that are ready for use by any thread that needs them
- Most applications only need a thread to have access to a JDBC connection when they are actively processing a transaction, which usually take only milliseconds to complete
- When a thread needs to do work against a MySQL or other database with JDBC, it requests a connection from the pool
- When the thread is finished using the connection, it returns it to the pool
- When the connection is "loaned out" from the pool, it is used exclusively by the thread that requested it.
- It is the same as if your thread called DriverManager.getConnection() every time it needed a JDBC connection

The main benefits to connection pooling

- Reduced connection creation time
- Simplified programming model
- Controlled resource usage
  - When using connection pooling, each individual thread can act as though it has created its own JDBC connection, allowing you to use straight-forward JDBC programming techniques.
  - http://www.roseindia.net/tutorial/java/jdbc/jdbcconnectionpooling.html

Transactions

- A transaction is a set of one or more statements that is executed as a unit, so either all of the statements are executed, or none of the statements is executed.
- **Disabling Auto-Commit Mode**
  - When a connection is created, it is in auto-commit mode.
  - The way to allow two or more statements to be grouped into a transaction is to disable the auto-commit mode.
    - con.setAutoCommit(false);

Transaction isolation level

- http://kek.ksu.ru/EOS/BD/SQL_transaction.html
- http://docs.oracle.com/javase/tutorial/jdbc/basics/transactions.html

DAO

- **DAO tutorial - the data layer**
  - http://balusc.blogspot.com/2008/07/dao-tutorial-data-layer.html