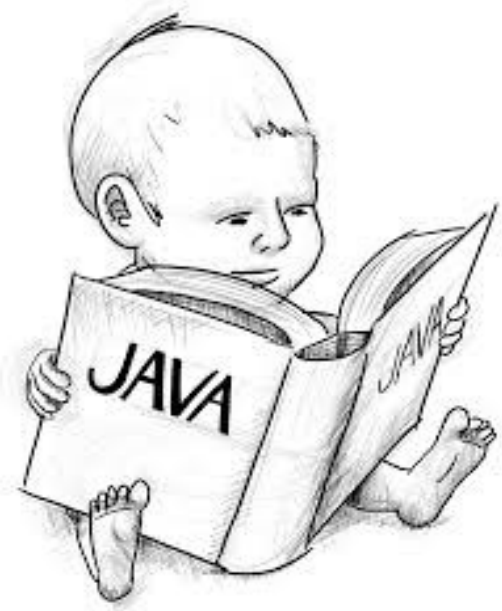
JDBC

Vivek Shah <bonii@diku.dk>
Frederik M. Madsen <fmma@diku.dk>
Ulrik T. Rasmussen <dolle@di.ku.dk>
Danil Annenkov <daan@di.ku.dk>

Marcos Vaz Salles <vmarcos@diku.dk>
Course Responsible

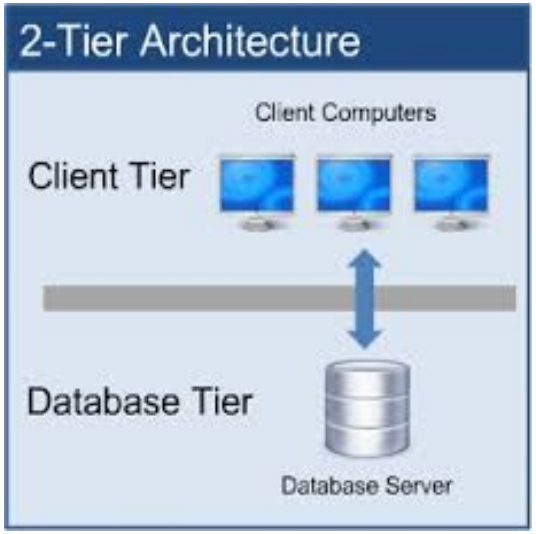


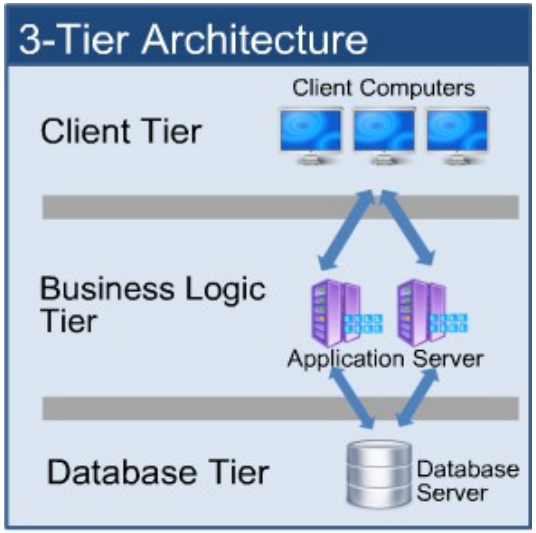
Tentative Plan

- JDBC
  - Introduction to JDBC.
  - Connecting to the database
  - Using SQL
  - Transactions
  - Mapping data types

Introduction to JDBC

- JDBC API provides access to tabular data (relational database).
- Can support both 2 or 3 tier architectures for data access.





Introduction to JDBC

JDBC enables us to do the following programming activities:

- Connect to a data-source like a database.
- Send queries and updates to data-source.
- Retrieve and process the received results.

The need for a DBMS

- First we need a DBMS. Let's get some suggestions.
- We need to install, configure and run it in the next 10 seconds without a database tutorial.
- And Apache Derby (Java DB) is the answer.
- Let's install, run and terminate a database server.
- Embedded mode versus ~~server mode~~. (bad naming)
- Embedded → Single user, single JVM access.
- Lifetime of JVM is the lifetime of database engine but not the data (persistent).

Derby

- Class.forName("org.apache.derby.jdbc.EmbeddedDriver").newInstance();

Connection url - jdbc:derby:databaseName;URLAttributes

- DriverManager.getConnection("jdbc:derby:example;create=true");
- DriverManager.getConnection("jdbc:derby:memory:example;create=true");
- DriverManager.getConnection("jdbc:derby:example;");
- DriverManager.getConnection("jdbc:derby:example;shutdown=true");
- DriverManager.getConnection("jdbc:derby:;shutdown=true");
- Does not create database if database exists.
- Shutdown throws exceptions (to signal success).
  - "08006" and "XJ015" are the only codes that indicate successful shutdown

Derby

Let's look at code examples

And we have a database, onward to JDBC

JDBC API to :

- Create a connection to the database.
- Send queries to the database and execute them.
- Retrieve results and process the received results.

Create a database connection

- A connection represents the link between the database and the Java application.
- Create connection using the DriverManager class and specifying the database connection URL.
- DriverManager.getConnection(connectionURL).
- DriverManager.getConnection(connectionURL, props) → supplies additional information wrapped in props to database.
- Use a DataSource. (And we will say no more).

Create statements (java.sql.Statement)

- Statement is used to implement SQL statements with no input or output parameters.
- PreparedStatement is used to pre-compile SQL statements that may contain input parameters.
  - Optimized version of statement to prevent interpretation of statements repetitively which different in arguments.
- CallableStatement is used to execute stored procedures which can contain input and output parameters.
  - Used to run scripts.
  - Stored procedures and functions.

Executing statements

- The statements that have been created need to be executed which can generate 0,1 or many ResultSet objects.
- A ResultSet object is a cursor on tabular data representing a database result set.
- Statement.executeUpdate – Used to run DDL and DML statements. Returns count of rows affected in case of DML statements or 0 if SQL statement returns nothing.
- Statement.executeQuery – Used to return single resultset. Commonly used for SELECT statements.
- Statement.execute – The most generic. Can return 0, 1 or many resultsets (use getResultSet(), getUpdateCount(), getMoreResults() methods to access result)

Processing the ResultSet

- Data in ResultSet is accessed through a cursor. Not the same as a database cursor.
- The cursor is like a “pointer” which points to a row in the resultset.
- Initially it points to before the first row.
- Can read only data from the row being pointed to by the cursor currently.
- The movement of cursor can be manipulated in various ways by methods in ResultSet class.
- Closing connection/statement before processing resultset causes error since there is nothing to read.

Closing connections, resultsets, statements

- Always remember to close statements, resultsets and connections when you no longer use them.
- The JVM can garbage collect the objects but not the database server.
- Not closing them form the most commonly found errors in applications using JDBC. Commonly known as connection and cursor leaks.

Using Transactions

- In some cases, we want groups of statements to take effect together or not at all.
- By default, the connection automatically commits after execution of each Statement.
- Disable auto-commit.
- Need to commit explicitly (for both reads and writes).
- Rollback to last commit or to specific savepoints (which can be manipulated).
- Can manipulate Transaction isolation levels (read committed, phantom reads, repeatable reads).

Batch Processing

Use batch processing if you insert/update large number of rows:

- Use *setAutoCommit(false)* to turn of auto-commit mode.
- Use *addBatch()* method of Statement or PreparedStatement class to add statements in batch.
- Execute whole query using *executeBatch()* method.
- Commit all changes explicitly by calling *Connection.commit().*

Mapping Data Types

- Data types in SQL and data types in Java do not match so one often needs to map.
- Different databases often differ in types as well.
- Fortunately, we just need to map JDBC type to Java types.
- Java.sql.* → Some common JDBC types (not all).
- CHAR, VARCHAR, LONGVARCHAR → String.
- INTEGER → int
- DATE → java.sql.Date
- TIMESTAMP → java.sql.Timestamp

Mapping Data Types

- For most Java types JDBC API has
  - *setXXX()/updateXXX()* methods in *PreparedStatment*
  - *getXXX()* methods in *ResultSet*

  that convert data according to the mapping (or throw SQLException).
- More types are available at

  http://db.apache.org/ojb/docu/guides/jdbc-types.html

Next steps

Explore the java.sql interface

JDBC: Summary

- How to run a database inside the JVM → Derby in embedded mode.
- JDBC API
  - To create connection to a database (DriverManger).
  - To generate and execute queries on database (Statement, PreparedStatement, CallableStatement).
  - To retrieve and process results (Resultsets).
- Notion and use of transaction in JDBC API.