Quick Start

This example runs the DAO Generator (EmpusaMB) and then runs a short example program that uses the DAOs. The DAO generator uses Apache Ant to run.

The full ready-to-run example with all its files is included in the `quick-start-example.zip` file.

This example is a standalone program that uses a “standalone data source”. With a few changes, this example can use a full blown data source and run on a web application server such as Tomcat, Weblogic, or WebSphere.

This example uses PostgreSQL database. With minimal changes it can also run on Oracle, Sybase, MySQL, DB2, and HyperSQL databases.

Step 1 - Create the database schema

Create the database schema in your RDBMS with the tables and views, PKs, FKs, unique constraints, etc.

```
create table account (
  id serial not null,
  current_balance integer not null,
  name varchar(100) not null,
  created_on date not null,
  primary key (id)
);

create table transaction (
  id serial not null,
  completed_at date not null,
  amount integer not null,
  account_id integer not null,
  primary key (id),
  constraint fk_tx_account foreign key (account_id)
    references account (id)
);

create view account_debit as
  select a.name, t.*
    from transaction t
    join account a on t.account_id = a.id
    where t.amount < 0;
```

Step 2 - Get the all the libraries

Download the MyBatis, EmpusaMB and JDBC driver libraries into the home dir of your project:

```
mybatis-3.3.0-jar
empusa-mybatis-2.3-runtime.jar
empusa-mybatis-2.3.jar
postgresql-9.4-1205.jdbc4.jar
```

Step 3 - Add the configuration files

Add the following files in the home dir of your project:

- File 1 - `build.xml`:

```
<?xml version="1.0"?>
<project name="dao-generator" basedir=".">

  <property name="daos.base.dir" value="src/main/java/com/company/daos" />
  <property name="mappers.base.dir" value="src/mybatis/mappers" />

  <target name="-init">
    <loadproperties srcfile="database.properties" />
    <mkdir dir="${daos.base.dir}/primitives" />
    <mkdir dir="${mappers.base.dir}/primitives" />
  </target>

  <target name="generate-daos" depends="-init">
    <delete includeemptydirs="true">
      <fileset dir="${daos.base.dir}" includes="**/*" />
      <fileset dir="${mappers.base.dir}" includes="**/*" />
    </delete>
    <antcall target="-init" />
    <taskdef name="empusamb" classname=
        "org.nocrala.tools.persistence.empusamb.ant.EmpusaMbAntTask">
      <classpath>
        <pathelement location="empusa-mybatis-2.3.jar" />
        <pathelement location="${driverclasspath}" />
      </classpath>
    </taskdef>
    <empusamb driver="${driverclass}"
              url="${url}"
              userid="${username}"
              password="${password}"
              division="${division}"
              configfile="dao-config.xml"
              display="list" />
  </target>

  <target name="run-example1" depends="-init">
    <mkdir dir="build" />
    <delete includeemptydirs="true">
      <fileset dir="build" includes="**/*" />
    </delete>
    <javac srcdir="src/main/java"
           destdir="build"
           includeantruntime="false"
           includes="**/*.java">
      <classpath>
        <pathelement location="mybatis-3.3.0.jar" />
        <pathelement location="empusa-mybatis-2.3-runtime.jar" />
      </classpath>
    </javac>
    <java classname="com.company.logic.Example1">
      <classpath>
        <pathelement path="build" />
        <pathelement path="src/mybatis" />
        <pathelement location="${driverclasspath}" />
        <pathelement location="mybatis-3.3.0.jar" />
        <pathelement location="empusa-mybatis-2.3-runtime.jar" />
      </classpath>
    </java>

  </target>

</project>
```

- File 2 - `database.properties`:

  Please replace all these properties, according to the RDBMS and your specific schema locations and credentials.

```
driverclasspath=postgresql-9.4-1205.jdbc4.jar
driverclass=org.postgresql.Driver
url=jdbc.url.to.your.dev.database
username=dev.database.username
password=dev.database.password
division=dev.schema
```

- File 3 - `dao-config.xml`:

```
<?xml version="1.0"?>
<!DOCTYPE empusa-mybatis SYSTEM "empusa-mybatis.dtd">

<empusa-mybatis>

  <layout>
    <daos gen-base-dir="src/main/java" package="com.company.daos" />
    <mappers gen-base-dir="src/mybatis" relative-dir="mappers" />
    <mybatis-configuration-template file="mybatis-template.xml" />
    <session-factory singleton-full-class-name=
      "com.company.sessionfactory.SessionFactory" />
    <select-generation temp-view-base-name="mybatis_temp_view" />
  </layout>

  <table name="account">
    <auto-generated-column name="id" />
  </table>

  <table name="transaction">
    <auto-generated-column name="id" />
  </table>
  <view name="account_debit" />

  <select name="big_deposit">
      <![CDATA[
    select a.name , t.*
      from account a, transaction t
      where t.account_id = a.id
        {* and t.amount >=
           #{minAmount,javaType=java.lang.Integer,jdbcType=NUMERIC} *}
      order by a.current_balance
      ]]>
  </select>

</empusa-mybatis>
```

- File 4 - `mybatis-template.xml`:

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE configuration
  PUBLIC "-//mybatis.org//DTD Config 3.0//EN"
  "http://mybatis.org/dtd/mybatis-3-config.dtd">
<configuration>

  <properties resource="../../database.properties" />

  <environments default="development">

    <environment id="development">

      <transactionManager type="JDBC" />

      <dataSource type="POOLED">
        <property name="driver" value="${driverclass}" />
        <property name="url" value="${url}" />
        <property name="username" value="${username}" />
        <property name="password" value="${password}" />
      </dataSource>

    </environment>

  </environments>

@@mappers@@

</configuration>
```

- File 5: `SessionFactory.java`

  Create the dir `src/main/java/com/company/sessionfactory` and place it on it.

```java
package com.company.sessionfactory;

import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.InputStream;
import java.sql.SQLException;

import org.apache.ibatis.session.SqlSessionFactory;
import org.apache.ibatis.session.SqlSessionFactoryBuilder;

public class SessionFactory {

  private static final String RESOURCE =
    "src/mybatis/mappers/primitives/mybatis-configuration.xml";

  private static SessionFactory instance = null;

  private SqlSessionFactory sqlSessionFactory = null;

  private SessionFactory() throws SQLException {
    try {
      File f = new File(RESOURCE);
      InputStream is = new FileInputStream(f);
      this.sqlSessionFactory = new SqlSessionFactoryBuilder().build(is);
    } catch (IOException e) {
      throw new SQLException("Could not read MyBatis configuration.", e);
    }
  }

  public static synchronized SessionFactory getInstance()
      throws SQLException {
    if (instance == null) {
      instance = new SessionFactory();
    }
    return instance;
  }

  public SqlSessionFactory getSqlSessionFactory() {
    return sqlSessionFactory;
  }

}
```

Step 4 - Generate the DAOs

With all four files created, just run the `generate-daos` Ant task.

```
$ ant generate-daos
```

You should see the task running and showing something like:

```
              ...
      [mkdir] Created dir: .../src/main/java/com/company/daos/primitives
      [mkdir] Created dir: .../src/mybatis/mappers/primitives
   [empusamb] Table account included.
   [empusamb] Table transaction included.
   [empusamb] View account_debit included.
   [empusamb] Select query big_tx included.
   [empusamb] Generating MyBatis DAOs for 2 tables, 1 view, and 1 select query...
   [empusamb] MyBatis DAOs generated.

BUILD SUCCESSFUL
Total time: 0 seconds
$
```

A short explanation

The output shows that DAOs were generated for two tables, one view, and one select query as defined in the `dao-config.xml` configuration file. Your database schema may have more tables and views, but only the ones mentioned in the configuration file are considered.

Once the ant task finishes all four DAOs are available under:

- `src/main/java/com/company/daos`
  ```
  AccountDAO.java
  AccountDebitDAO.java
  BigDepositDAO.java
  TransactionDAO.java
  ```
- `src/main/java/com/company/daos/primitives`
  ```
  AccountDAOPrimitives.java
  AccountDebitDAOPrimitives.java
  BigDepositDAOPrimitives.java
  TransactionDAOPrimitives.java
  ```

The first directory contains the main DAO java classes that you will normally use. You can also add domain logic in them if you wish. These DAOs are generated once and never overwritten.

The second directory contains the primitive DAO java classes. These ones are overwritten every time you regenerate the DAOs, to reflect the latest changes in the database. This is especially useful when columns, unique indexes, and foreign keys are added, removed, and/or renamed. Once you regenerate the DAOs, the primitive classes will have all the new changes readily available for you to use.

Step 5 - Run the example

Add the following example Java class that uses the DAOs to query the database. Create it as `src/main/java/com/company/logic/Example1.java`:

```
package com.company.logic;

import java.sql.Date;
import java.sql.SQLException;

import com.company.daos.AccountDAO;
import com.company.daos.AccountDebitDAO;
import com.company.daos.BigDepositDAO;
import com.company.daos.TransactionDAO;

public class Example1 {

  public static void main(final String[] args) throws SQLException {

    // Note: this simple example does NOT use database transactions.

    Date now = new Date(System.currentTimeMillis());

    AccountDAO a = new AccountDAO();
    a.setName("CHK1067");
    a.setCurrentBalance(100);
    a.setCreatedOn(now);
    a.insert();

    TransactionDAO t = new TransactionDAO();
    t.setAccountId(a.getId());
    t.setAmount(40);
    t.setCompletedAt(now);
    t.insert();
    int tx1 = t.getId();

    t.setAmount(500);
    t.insert();
    int tx2 = t.getId();

    t.setAmount(-440);
    t.insert();
    int tx3 = t.getId();

    System.out.println("Account " + a.getId() + " created, "
        + "with transactions " + tx1 + ", " + tx2 + ", and " + tx3
        + " also created.");

    AccountDebitDAO filter = new AccountDebitDAO();
    for (AccountDebitDAO d : filter.select()) {
      System.out.println("Debit found: tx " + d.getId()
          + " with amount $" + d.getAmount());
    }

    for (BigDepositDAO b : BigDepositDAO.select(300)) {
      System.out.println("Big deposit found: tx " + b.getId()
          + " with amount $" + b.getAmount());
    }

  }

}
```

Run the example program:

```
ant run-example1

     ...
     [java] Account 1 created, with transactions 1, 2, and 3 also created.
     [java] Debit found: tx 3 with amount $-440
     [java] Big deposit found: tx 2 with amount $500

BUILD SUCCESSFUL
Total time: 1 second
```

That's it! You can check the database schema to see the newly inserted rows. To use other available primitive methods and to use database transactions see the **Cheat Sheet** document.