# Hello World - MyBatis

This Hello World example describes how to run a simple Hello World one-class application.

It demonstrates an SQL insert into a table that uses a DAO class, and goes step by step showing the details of how to set up the application and run the example.

For simplicity this Hello World demo runs as a command line Java application but with minimal changes it can also run on a JEE web container such as Tomcat, JBoss, Weblogic, or WebSphere.

## Apache Ant

Apache Ant is used during the development of an application to automate all tasks related to the HotRod. Ant is not needed when the application is running or when deploying to a production environment.

This example uses Ant targets to run the database, to run the example, and to list database data. To learn more about Apache Ant see http://ant.apache.org. Ant was chosen since it can easily run java applications, and run other OS level tasks such as starting and stopping the database. All the Ant commands can be run from a terminal or within Eclipse IDE.

To run the tasks from Eclipse you'll need to open the Eclipse Ant View (Window -> Show View -> Ant), and then find the build.xml file for the Hello World example; drag and drop this file into the Ant View. Once there, expand it (small arrow icon on the left) to see all available tasks. To run any of them just double-click on it. The console window will be opened and will show the results of the execution.

To run the tasks from the command line you'll need to have Ant installed at the OS level and in your path. Then make sure to move to the home folder of the Hello World example before running any task. For example, if you need to run the tasks start-h2-database, of the Hello World example type:

cd examples/mybatis/helloworld

ant start-h2-database

## Setting up the database

This example uses a local H2 database. H2 was chosen since it's very easy to run locally without any installation; a simple command line launches it and it starts listening on a local network port for any JDBC client. To learn more about H2 database see http://www.h2database.com.

To start H2 run the Ant task start-h2-database. Type (or double click on it if using the Eclipse Ant View):

ant start-h2-database

The H2 database will start and you'll see an output like:

Buildfile: build.xml

start-h2-database:

-issue-h2-start:

[echo] [ H2 local database has been started ]

-h2-already-started:

BUILD SUCCESSFUL

Total time: 456 milliseconds

If you're unsure you started the H2 database you can run this command again. If it's already running it will find out and won't start it again.

To stop H2 run the Ant tasks stop-h2-database. Type (or double click on it if using the Eclipse Ant View):

ant stop-h2-database

The H2 database will be stopped and you'll see an output like:

Buildfile: build.xml

stop-h2-database:

-issue-h2-stop:

[java] Shutting down TCP Server at tcp://localhost:12345

[echo] [ H2 local database has been stopped ]

-h2-already-stopped:

BUILD SUCCESSFUL

Total time: 3 seconds

Please start H2 again and leave it running:

ant start-h2-database

Now we will initialize the database; we'll create a table and insert some data in it. We'll run the following SQL statements (all included in the prepare-database.sql file):

**drop** **table** if **exists** vehicle;

**create** **table** vehicle (

id **integer** identity **primary** **key** **not** **null**,

brand **varchar**(30) **not** **null**,

model **varchar**(30) **not** **null**,

used **boolean** **not** **null**,

current\_mileage **integer** **not** **null**,

purchased\_on **date**

);

**insert** **into** vehicle (brand, model, used, current\_mileage, purchased\_on)

**values** ('Kia', 'Soul', **true**, '28500', '2014-03-14');

**insert** **into** vehicle (brand, model, used, current\_mileage, purchased\_on)

**values** ('Toyota', 'Tercel', **false**, '26', '2017-01-28');

**insert** **into** vehicle (brand, model, used, current\_mileage, purchased\_on)

**values** ('DeLorean', 'DMC-12', **true**, '241689', '1982-11-17');

As you see the table will be created and initialized with three rows of data. To run these SQL statements run the Ant task initialize-database. Type (or double click on it if using the Eclipse Ant View):

ant initialize-database

All SQL statements will be executed and you'll see an output like:

Buildfile: build.xml

initialize-database:

[sql] Executing resource: prepare-database.sql

[sql] 0 rows affected

[sql] 0 rows affected

[sql] 1 rows affected

[sql] 1 rows affected

[sql] 1 rows affected

[sql] 5 of 5 SQL statements executed successfully

[echo] [ Database tables and data initialized ]

BUILD SUCCESSFUL

Total time: 894 milliseconds

Well, with that the database is now ready. If you want to make sure the table and data are there you can use your favorite tool to connect to it and inspect it. Or, for simplicity, this example provides an Ant task that lists the rows in the table. To list the data run the Ant task list-data. Type (or double click on it if using the Eclipse Ant View):

ant list-data

A simple SQL select will display all the data on the table, and you'll see an output like:

Buildfile: build.xml

list-data:

[sql] Executing commands

[sql] ID,BRAND,MODEL,USED,CURRENT\_MILEAGE,PURCHASED\_ON

[sql] 1,Kia,Soul,TRUE,28500,2014-03-14

[sql] 2,Toyota,Tercel,FALSE,26,2017-01-28

[sql] 3,DeLorean,DMC-12,TRUE,241689,1982-11-17

[sql] 0 rows affected

[sql] 1 of 1 SQL statements executed successfully

BUILD SUCCESSFUL

Total time: 789 milliseconds

The output is a little bit crude but is good enough for our purposes. You can see three rows of data including the Kia Soul, the Toyota Tercel, and the DeLorean DMC-12 in there. Congratulations! The database is now ready to be used.

## Generate the DAOs using HotRod

HotRod will generate a DAO class to interact with the vehicle table. To do it HotRod needs:

* A database running with the tables that will be generated.
* hotrod.xml - the HotRod configuration file.
* mybatis-template.xml - a template of the MyBatis main configuration file, ready to be filled with the list of mapper files.
* The Ant task in the build.xml file.
* The hotrod.jar library.

The hotrod.xml configuration file is in the home folder and looks like:

<?xml version=*"1.0"*?>

<!DOCTYPE hotrod SYSTEM "hotrod.dtd">

<hotrod>

<generators>

<mybatis>

<daos gen-base-dir=*"auto-generated/java"* dao-package=*"daos"* />

<mappers gen-base-dir=*"auto-generated/mappers"* relative-dir=*"persistence"* />

<mybatis-configuration-template file=*"mybatis-template.xml"* />

<session-factory singleton-full-class-name=*"sessionfactory.DatabaseSessionFactory"* />

<select-generation temp-view-base-name=*"hotrod\_temp\_view"* />

</mybatis>

</generators>

<table name=*"vehicle"*>

<auto-generated-column name=*"id"* />

</table>

</hotrod>

In this example the <generators> tag includes the <mybatis> tag. The latter indicates the generated files will be placed into:

* autogenerated/java - all Java classes.
* autogenerated/mappers - all MyBatis mappers.

Finally, the <table> tag tells HotRod to include the table VEHICLE. Also, it specifies that the ID column is auto-generated as an identity column.

To generate the DAOs, use the Ant task hotrod. Type (or double click on it if using the Eclipse Ant View):

ant hotrod

HotRod retrieves the structure of the table VEHICLE and will generate the DAOs. you'll see an output like:

Buildfile: build.xml

hotrod:

[delete] Deleting directory auto-generated

[mkdir] Created dir: auto-generated/java

[mkdir] Created dir: auto-generated/mappers

[hotrod] HotRod version 3.0.0 (build 20170128-221617)

[hotrod]

[hotrod] Database URL: jdbc:h2:tcp://localhost:12345/db001;IFEXISTS=TRUE

[hotrod] Database Name: H2 - version 1.3 (1.3.176 (2014-04-05))

[hotrod] JDBC Driver: H2 JDBC Driver - version 1.3 (1.3.176 (2014-04-05))

[hotrod]

[hotrod] HotRod Database Adapter: H2 Adapter

[hotrod] Database Catalog: DB001

[hotrod] Database Schema: PUBLIC

[hotrod]

[hotrod] Generating all facets.

[hotrod]

[hotrod] Table VEHICLE included.

[hotrod]

[hotrod] Generating MyBatis DAOs for 1 table, 0 views, 0 DAOs (0 sequences, 0 updates), and 0 select queries...

[hotrod]

[hotrod] MyBatis generation complete.

BUILD SUCCESSFUL

Total time: 887 milliseconds

HotRod generated the following files:

* auto-generated/java/daos/primitives/VehicleDAOPrimitives.java - Abstract Java class that includes all the predefined methods to insert, delete, update, and select data from the VEHICLE table. One file is generated for each table. Do not modify this class since it's automatically rewritten by HotRod every time it's run, to reflect the latest changes of the database.
* auto-generated/java/daos/VehicleDAO.java - Java class that extends from the VehicleDAOPrimitives class, so the developer can add custom logic to the DAO. One file is generated for each table. This source class is never rewritten by HotRod, to preserve all the custom logic.
* auto-generated/mappers/persistence/primitives/primitives-vehicle-dao.xml - MyBatis mapper file that includes all the SQL statements needed to interact with the database. One file is generated for each table. Do not modify this class since it's automatically rewritten by HotRod every time it's run, to reflect the latest changes of the database.
* auto-generated/mappers/persistence/primitives/mybatis-configuration.xml - MyBatis main configuration file. This is a single file in the application. It's based in the mybatis-template.xml file in the root of the example, but it now lists all the mapper files just generated. Do not modify this class since it's automatically rewritten by HotRod every time it's run. Modify the template instead if you need to.

With these files all is ready to run the Hello World application.

## Run the Hello World application

All the DAOs are ready and the Hello World application now needs to use them to insert a row into the database. The source code is in the java/examples/HelloWorld.java and includes the following lines of code:

VehicleDAO skoda = **new** VehicleDAO();

skoda.setBrand("Skoda");

skoda.setModel("Octavia");

skoda.setUsed(**false**);

skoda.setCurrentMileage(7);

skoda.setPurchasedOn(**new** Date(System.*currentTimeMillis*()));

skoda.insert();

System.***out***.println("New vehicle Skoda added, with id " + skoda.getId() + ".");

First, a DAO object is instantiated to hold the values we want to insert in the new table row. Then, we set each one of the values and finally we execute the insert() method to insert the row.

Note the ID property was not set. It's not needed since this column is auto-generated by the database. Since HotRod knows about it (as previously shown in the configuration file) its value is automatically retrieved upon insertion, and the ID property ends up with the newly created id value.

To run the Hello World example, use the Ant task helloworld. Type (or double click on it if using the Eclipse Ant View):

ant helloworld

The output looks like:

Buildfile: build.xml

helloworld:

[delete] Deleting directory build

[mkdir] Created dir: build

[javac] Compiling 4 source files to build

[echo]

[java] Example 1 - Hello World - Starting

[java] New vehicle Skoda added, with id 4.

[java] Example 1 - Hello World - Finished

BUILD SUCCESSFUL

Total time: 1 second

The new row was added! To verify it's there you can use the list-data Ant task discussed earlier. Type (or double click on it if using the Eclipse Ant View):

ant list-data

The table now looks like:

Buildfile: build.xml

list-data:

[sql] Executing commands

[sql] ID,BRAND,MODEL,USED,CURRENT\_MILEAGE,PURCHASED\_ON

[sql] 1,Kia,Soul,TRUE,28500,2014-03-14

[sql] 2,Toyota,Tercel,FALSE,26,2017-01-28

[sql] 3,DeLorean,DMC-12,TRUE,241689,1982-11-17

[sql] 4,Skoda,Octavia,FALSE,7,2017-01-28

[sql] 0 rows affected

[sql] 1 of 1 SQL statements executed successfully

BUILD SUCCESSFUL

Total time: 756 milliseconds

...and there it is! A fourth row is now shown with the Skoda Octavia vehicle. Incidentally, we can see its id actually has the value 4.

That's it! You have set up the HotRod application and configuration, you created a database with a table, then you generated the DAO classes using HotRod, and finally you ran the application. Congratulations!

Once done, don't forget to stop the H2 database! I always do.