

Working with databases

Working with databases

tapestry-model-hibernate (a submodule of tapestry-model) extensively uses [Hibernate](#) as its persistence layer. While you can provide a different "persistenceService" implementation, the only one provided by Tynamo is a `HibernatePersistenceService`, which as the name implies, implements a Hibernate based persistence mechanism. Hibernate is an ORM tool that maps your Java objects to a database schema.

By default, Tynamo uses in-memory [H2](#) database, which means it's super-simple to set up and the database only exists in-memory. H2 is unbelievably small and fast, evolves quickly and is nice to develop with, and even nicer for running your db integration tests against. Because the start-up cost is minimal, you always start from a clean slate and you know it works no matter what the environment is.

With Hibernate it's very well possible and easy to use more than one type of database since the switching cost is minimal - you just provide a different database configuration and off you go.

i However, one potential issue that you want to be careful with using reserved words in the database, because Hibernate maps your entities directly to a database schema.

A simple example of what doesn't work is a property named "date", i.e. having `getDate()` operation in one of your entities. These are typically easy to spot from the error logs when you start up your application after adding a new entity. The theory is that by supporting (and testing with) more databases you make your implementation stronger.

Typical scenarios for different database use cases are:

- **Development**, where it's nice to have a persistent database so you don't need to seed the data all the time and one that is as fast as possible on a small dataset
- **Integration testing** (often automatically launched from you build), where you typically set the database state for a testing and roll back the changes after each test
- **Production** (and production testing), where you obviously need a persistent storage, the set-up costs are often less relevant, it needs to work fast on a larger dataset; and you may require some other features from the database software, like backups, recoverability and clustering

Choosing the right database

This is not a generic database guide, but a few words about the choices. For development, your best bets are H2, HSQLDB, Derby or MySQL depending on your preferences and what you are used to. Most databases come with some type of command-line interface but H2 has a great, built-in web interface for looking at the raw database results and executing raw SQL queries. If you appreciate automating even the database setup for development, H2 and HSQLDB are easy to embed, and are especially useful when doing integration system with fully automated build systems.

Your production database of choice depends largely on your application type and your deployment environment. For a small-scale prototyping, free or open-source projects, HSQLDB, DERBY, MySQL and PostgreSQL are good choices. If the database choice isn't up to you, you should check that Hibernate first of all supports it and secondly, keep testing frequently that your domain model works with the particular database even if you don't use it during development. H2 is proven to work with millions of records without hick-ups (see [H2's own performance comparisons](#)).

Configuring Hibernate

Your typical hibernate.properties for development should look something like this:

```
hibernate.dialect=org.hibernate.dialect.H2Dialect
hibernate.show_sql=true
hibernate.hbm2ddl.auto=update
hibernate.connection.driver_class=org.h2.Driver
hibernate.connection.url=jdbc:h2:mem:test
hibernate.connection.username=sa
hibernate.connection.password=
```

For other database configurations (and required dependencies, check out our [Sample database configurations](#). Using *hibernate.hbm2ddl.auto=create-drop* causes the database to be created at the start-up and then deleted after use which might be advantageous in some cases, like for integration testing. Hibernate also allows you specify *hibernate.hbm2ddl.auto=validate*, which may make sense in a production environment, but notice that instantiating the Hibernate related services would fail if you use *validate* and Hibernate requires changes to be made in the database schema.

Back to [Developing in IDEs](#) or continue to [Customizing pages](#)