Useful Links

Download libraries: <https://sourceforge.net/projects/hibernate/files/hibernate-orm/>

Getting started: <http://docs.jboss.org/hibernate/orm/current/quickstart/html_single/>

Example files: <http://docs.jboss.org/hibernate/orm/current/quickstart/html_single/hibernate-tutorials.zip>

Key Terms

* **Pattern**: active record pattern
* **Entity**: class mapped to a table
* **Hibernate Query Language**: mini language for specific queries (not basic SELECT, INSERT, UPDATE, DELETE)

Setting up an Intellij project

1. Download/install intellij
2. Start new project
   1. Pick an SDK (JDK must be installed)
   2. Pick Hibernate
   3. Check “create default configuration file”
   4. Leave setting to download libraries
3. Check settings in hibernate.cfg.xml

|  |
| --- |
| <?xml version='1.0' encoding='utf-8'?>  <!DOCTYPE hibernate-configuration PUBLIC  "-//Hibernate/Hibernate Configuration DTD//EN"  "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">  <hibernate-configuration>  <session-factory>  <property name="connection.url">jdbc:mysql://jarcher.greenrivertech.net:3306/jarcher\_hibernate</property>  <property name="connection.driver\_class">com.mysql.jdbc.Driver</property>  <property name="connection.username">…</property>  <property name="connection.password">…</property>  <!-- DB schema will be updated if needed -->  <!-- <property name="hbm2ddl.auto">update</property> -->  <!-- JDBC connection pool (use the built-in) -->  <property name="connection.pool\_size">1</property>  <!-- SQL dialect -->  <property name="dialect">org.hibernate.dialect.MySQL5Dialect</property>  <!-- Echo all executed SQL to stdout -->  <property name="show\_sql">true</property>  <!-- Drop and re-create the database schema on startup -->  <property name="hbm2ddl.auto">update</property>  <!-- Names the annotated entity class -->  <mapping class="edu.greenriver.it.entities.MessagesEntity"/>  </session-factory>  </hibernate-configuration> |

1. White list IP on greenrivertech.net (under remote mysql)
2. Added a new bogus database to greenrivertech.net – jarcher\_hibernate
   1. Added a dummy table with records
3. Adding database to intellij project
   1. Opened Database panel (under menu: View 🡪 Tool Windows 🡪 Database)
   2. + 🡪 Database 🡪 MySql
   3. Entered connection details
      1. Host: jarcher.greenrivertech.net
      2. Database: jarcher\_hibernate
      3. User: jarcher\_user
      4. Password: …
      5. Url (supplied for me): jdbc:mysql://jarcher.greenrivertech.net:3306/jarcher\_hibernate
   4. Downloaded missing driver files (bottom of screen)
   5. Queried tables live in Intellij
4. Added mysql connector file project, then added through project settings
   1. Found here: <https://dev.mysql.com/downloads/connector/j/5.1.html>
   2. Menu: File 🡪 Project Structure 🡪 + 🡪 Java 🡪 Connector file
5. Opened persistence tool to create entities
   1. Menu: View 🡪 Tool Windows 🡪 Persistence
   2. Generated classes: Right click data source 🡪 Generate Persistence Mapping 🡪 Data Schema

CRUD with Hibernate

Test Entity:

|  |
| --- |
| package edu.greenriver.it.entities;  import org.hibernate.annotations.GeneratorType;  import org.hibernate.annotations.GenericGenerator;  import javax.persistence.\*;  import java.util.Date;  @Entity  @Table(name = "messages", schema = "jarcher\_hibernate", catalog = "")  public class MessagesEntity  {  private int id;  private String message;  private Date postedOn;  //default constructor  public MessagesEntity()  {  //do nothing, typically required for ORM  }  public MessagesEntity(String message, Date postedOn)  {  this.message = message;  this.postedOn = postedOn;  }  @Id  **@GeneratedValue(generator = "increment")**  **@GenericGenerator(name = "increment", strategy = "increment")**  @Column(name = "id")  public int getId()  {  return id;  }  @Basic  @Column(name = "message")  public String getMessage()  {  return message;  }  @Temporal(TemporalType.TIMESTAMP)  @Column(name = "posted")  public Date getPostedOn()  {  return postedOn;  }  public void setId(int id)  {  this.id = id;  }  public void setMessage(String message)  {  this.message = message;  }  public void setPostedOn(Date postedOn)  {  this.postedOn = postedOn;  }  @Override  public boolean equals(Object o)  {  if (this == o) return true;  if (o == null || getClass() != o.getClass()) return false;  MessagesEntity that = (MessagesEntity) o;  if (id != that.id) return false;  if (message != null ? !message.equals(that.message) : that.message != null) return false;  return true;  }  @Override  public int hashCode()  {  int result = id;  result = 31 \* result + (message != null ? message.hashCode() : 0);  return result;  }  @Override  public String toString()  {  return "Message: " + id + ", \"" + message + "\" (" + postedOn + ")";  }  } |

Saving a new record

|  |
| --- |
| public static void promptForMessage()  {  //get inputs  String message = promptString("Enter a message:");  String dateString = promptString("Enter a date (MM/DD/YYYY):");  //verify format  String[] dateParts = dateString.split("/");  while (dateParts.length != 3)  {  dateString = promptString("Please enter a date with the following format: (MM/DD/YYYY)");  }  int month = Integer.parseInt(dateParts[0]);  int day = Integer.parseInt(dateParts[1]);  int year = Integer.parseInt(dateParts[2]);  Date date = new GregorianCalendar(year, month - 1, day).getTime();  createMessage(message, date);  }  public static void createMessage(String message, Date date)  {  Session session = factory.openSession();  session.beginTransaction();  //create and save a new record  **MessagesEntity record = new MessagesEntity(message, date);**  **session.save(record);**  session.getTransaction().commit();  session.close();  }  private static String promptString(String prompt)  {  System.out.println(prompt);  return console.nextLine();  } |

Reading records

|  |
| --- |
| public static void readTest()  {  //session manages interactions with the database  Session session = factory.openSession();  session.beginTransaction();  //print table name  Set<EntityType<?>> entities = session.getSessionFactory().getMetamodel().getEntities();  Iterator<EntityType<?>> it = entities.iterator();  EntityType<?> messagesTableName = it.next();  System.out.println(messagesTableName.getName());  Query query = session.createQuery("from MessagesEntity");  List<MessagesEntity> messages = query.list();  for (MessagesEntity message : messages)  {  System.out.println(message.getId() + ": " + message.getMessage() + "(" +  message.getPostedOn() + ")");  }  //commit changes  session.getTransaction().commit();  session.close();  } |

Updating records

|  |
| --- |
| public static void updateTest()  {  //create and save an entity  Object[] parts = promptForMessage();  //update it!  updateRecord((String)parts[0], (Date)parts[1]);  }  public static void updateRecord(String message, Date date)  {  //perform the update  Session session = factory.openSession();  session.beginTransaction();  //save and show current record  MessagesEntity record = new MessagesEntity(message, date);  int newId = (Integer)session.save(record);  System.out.println("Current record: " + record.toString());  //update the record and persist!  String updatedMessage = promptString("Enter a new message: ");  record.setMessage(updatedMessage); //update the data!  session.update(record);  session.getTransaction().commit();  session.close();  } |

Deleting records

|  |
| --- |
| public static void deleteTest()  {  Session session = factory.openSession();  session.beginTransaction();  //get record by id  MessagesEntity message = session.get(MessagesEntity.class, 8);  System.out.println(message);  //delete it  session.delete(message);  session.getTransaction().commit();  session.close();  } |

Table Relationships

Adding the DB tables

* Referential integrity constraints: Managing foreign keys and their relationships between tables
* You must change the database engine for each table to Innodb to use this feature
  + Use the operations tab in phpmyadmin
* Click the relationships view under the referring table
* Add the constraint
* You can then follow the constraint using the foreign key as a link in phpmyadmin

Generating entities

* Generate the entities
* Check the “show default relationships” box
* Select foreign, primary keys and relationships from the tables
* Make sure the hibernate.cfg.xml is selected for the session factory
* Select generate JPA annotations
  + <mapping> tags will be added automatically to hibernate.cfg.xml

Hiccups

Not seeing your id update after persisting (saving) changes. Make sure that the your id is set in the entity with a generator. This does not happen automatically through Intellij.

|  |
| --- |
| @Id  **@GeneratedValue(generator = "increment")**  **@GenericGenerator(name = "increment", strategy = "increment")**  @Column(name = "id")  public int getId()  {  return id;  } |

Intellij errors with mapped table names (underlined names in red).

* Open persistance pane
* Right click persistance unit
* Assign data source
* Take hibernate.cfg.xml and choose the data source for the DB in the right column

Inellij errors with column name not being recognized in annotations

* Right click data source
* Click synchronize

Class Descriptions

**Session Factory**: A SessionFactory is an expensive-to-create, threadsafe object, intended to be shared by all application threads. It is created once, usually on application startup, from a Configuration instance.

**Session**: A Session is an inexpensive, non-threadsafe object that should be used once and then discarded for: a single request, a conversation or a single unit of work. A Session will not obtain a JDBC Connection, or a Datasource, unless it is needed. It will not consume any resources until used.

**How to use Session objects**:

*First, let's define a unit of work. A unit of work is a design pattern described by Martin Fowler as “[maintaining] a list of objects affected by a business transaction and coordinates the writing out of changes and the resolution of concurrency problems. ”[*[*PoEAA*](https://docs.jboss.org/hibernate/orm/3.3/reference/en-US/html/bi01.html#biblio-PoEAA)*] In other words, its a series of operations we wish to carry out against the database together. Basically, it is a transaction, though fulfilling a unit of work will often span multiple physical database transactions (see*[*Section 11.1.2, “Long conversations”*](https://docs.jboss.org/hibernate/orm/3.3/reference/en-US/html/transactions.html#transactions-basics-apptx)*). So really we are talking about a more abstract notion of a transaction. The term "business transaction" is also sometimes used in lieu of unit of work.*

*Do not use the*session-per-operation*antipattern: do not open and close a Session for every simple database call in a single thread. The same is true for database transactions. Database calls in an application are made using a planned sequence; they are grouped into atomic units of work. This also means that auto-commit after every single SQL statement is useless in an application as this mode is intended for ad-hoc SQL console work. Hibernate disables, or expects the application server to disable, auto-commit mode immediately. Database transactions are never optional. All communication with a database has to occur inside a transaction. Auto-commit behavior for reading data should be avoided, as many small transactions are unlikely to perform better than one clearly defined unit of work. The latter is also more maintainable and extensible.*

*The most common pattern in a multi-user client/server application is*session-per-request*. In this model, a request from the client is sent to the server, where the Hibernate persistence layer runs. A new HibernateSession is opened, and all database operations are executed in this unit of work. On completion of the work, and once the response for the client has been prepared, the session is flushed and closed. Use a single database transaction to serve the clients request, starting and committing it when you open and close theSession. The relationship between the two is one-to-one and this model is a perfect fit for many applications.*