What is Hibernate?

* A framework for persisting/ saving Java objects in a database.

🡪 🡪

Database

Hibernate

Java App

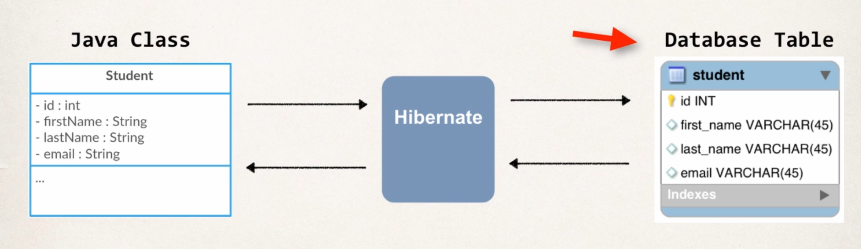
* 🡨

**Benefits of Hibernate**

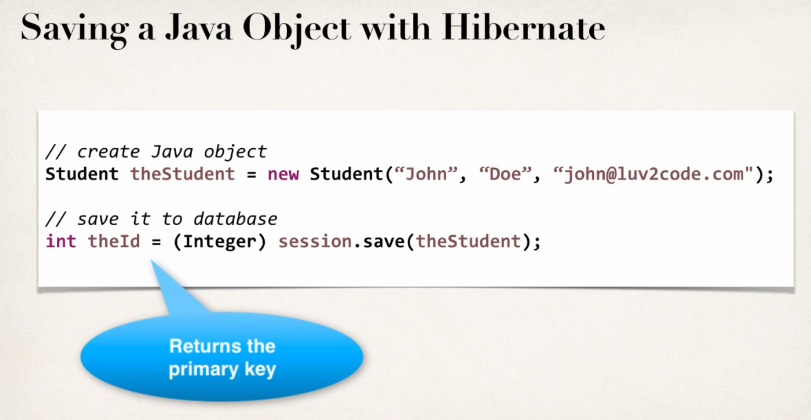
* Hibernate handles all of the low-level SQL
* Minimizes the amount of JDBC code we have to develop
* Hibernate provides the Object-to-Relational Mapping (ORM)

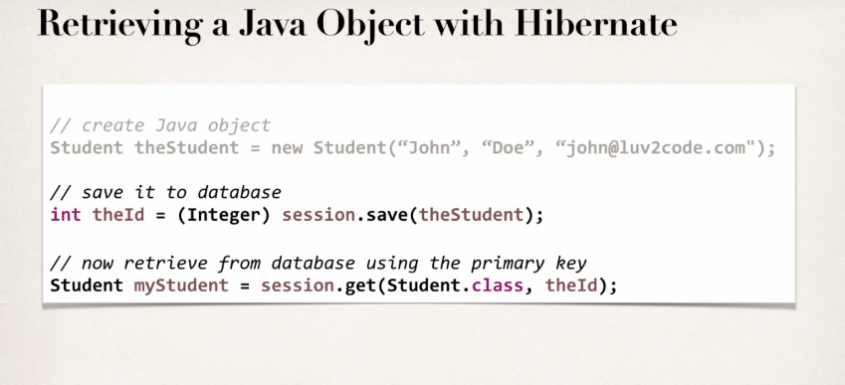
**Object-To-Relational Mapping (ORM)**

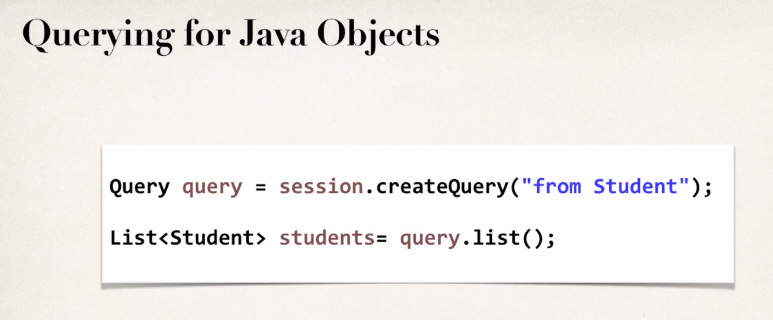
The developer defines mapping between Java class and database table.



All this mapping can be set up via a configuration file using XML or Java Annotations

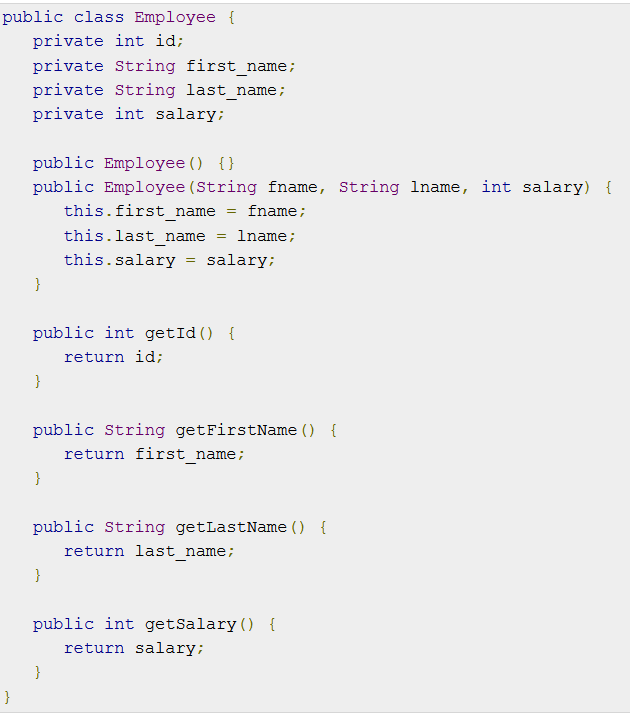




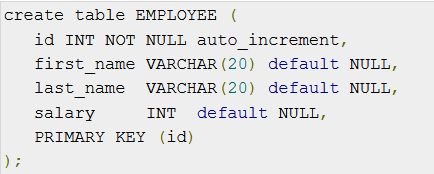


**Why Object Relational Mapping (ORM)?**

When we work with an object-oriented system, there is a mismatch between the object model and the relational database. RDBMSs represent data in a tabular format whereas object-oriented languages, such as Java or C# represent it as an interconnected graph of objects.



Consider the above objects are to be stored and retrieved into the following RDBMS table –



First problem, what if we need to modify the design of our database after having developed a few pages or our application? Second, loading and storing objects in a relational database exposes us to the following five mismatch problems –

|  |  |
| --- | --- |
| Sr.No. | Mismatch & Description |
| 1 | **Granularity**  Sometimes you will have an object model, which has more classes than the number of corresponding tables in the database. |
| 2 | **Inheritance**  RDBMSs do not define anything similar to Inheritance, which is a natural paradigm in object-oriented programming languages. |
| 3 | **Identity**  An RDBMS defines exactly one notion of 'sameness': the primary key. Java, however, defines both object identity (a==b) and object equality (a.equals(b)). |
| 4 | **Associations**  Object-oriented languages represent associations using object references whereas an RDBMS represents an association as a foreign key column. |
| 5 | **Navigation**  The ways you access objects in Java and in RDBMS are fundamentally different. |

The **O**bject-**R**elational **M**apping (ORM) is the solution to handle all the above impedance mismatches.

## What is ORM?

ORM stands for **O**bject-**R**elational **M**apping (ORM) is a programming technique for converting data between relational databases and object oriented programming languages such as Java, C#, etc.

An ORM system has the following advantages over plain JDBC −

|  |  |
| --- | --- |
| Sr.No. | Advantages |
| 1 | Let’s business code access objects rather than DB tables. |
| 2 | Hides details of SQL queries from OO logic. |
| 3 | Based on JDBC 'under the hood.' |
| 4 | No need to deal with the database implementation. |
| 5 | Entities based on business concepts rather than database structure. |
| 6 | Transaction management and automatic key generation. |
| 7 | Fast development of application. |

An**.** ORM solution consists of the following four entities −

|  |  |
| --- | --- |
| Sr.No | Solutions |
| 1 | An API to perform basic CRUD operations on objects of persistent classes. |
| 2 | A language or API to specify queries that refer to classes and properties of classes. |
| 3 | A configurable facility for specifying mapping metadata. |
| 4 | A technique to interact with transactional objects to perform dirty checking, lazy association fetching, and other optimization functions. |

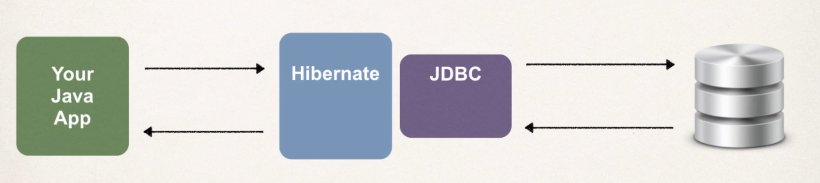
## Java ORM Frameworks

There are several persistent frameworks and ORM options in Java. A persistent framework is an ORM service that stores and retrieves objects into a relational database.

* Enterprise JavaBeans Entity Beans
* Java Data Objects
* Castor
* TopLink
* Spring DAO
* Hibernate

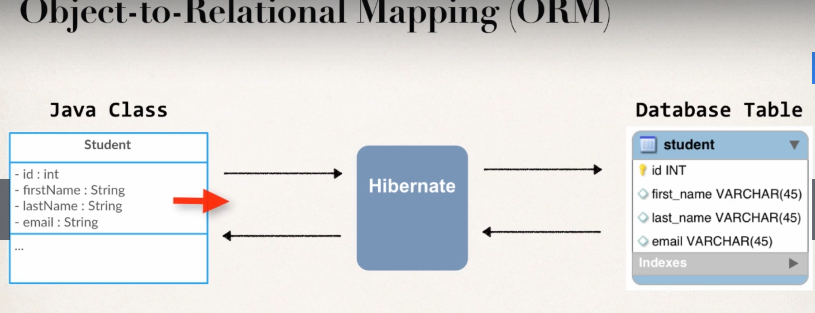
**Hibernate and JDBC**

Hibernate uses JDBC for all database communications. Hibernate uses JDBC in the background.



**Entity Class**

Java class that is mapped to a database table. Basically, a plain old java class with getters and setters, constructors. With annotations to help in mapping to the database.



**Java Annotations**

1. Map class to db table
2. Map fields to db columns.

**Hibernate Dev Process**

1. Add Hibernate Configuration file – XML Config file (legacy)
2. Annotate Java Class – Java Annotations (modern and preferred)
3. Develop Java Code to perform database operations.

**Two key players**

|  |  |
| --- | --- |
| Class | Description |
| SessionFactory | Reads the hibernate config file  Creates Session objects  Heavy -weight object  Only create once in the app |
| Session | Wraps a JDBC Connection  Main object used to save/retrieve objects  Short-lived object  Retrieved from SessionFactory |

1. **Hibernate Configuration File (hibernate.cfg.xml)**



1. Adding annotations to Entity Class





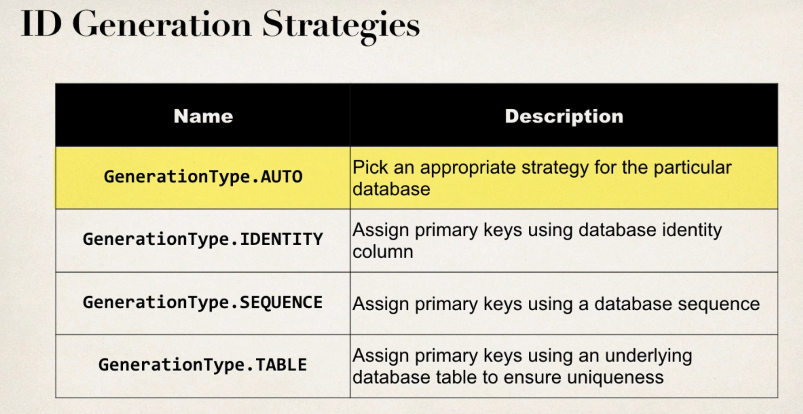
1. **Java Code to perform database operations**

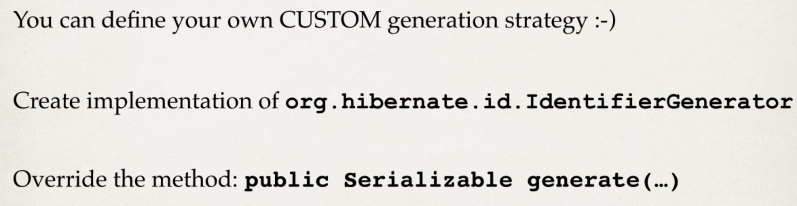
**CREATE**



**Primary Keys**

@GeneratedValue(strategy=GenerationType.”Name” 🡪 way of telling hibernate to the way of handling the particular Id explicitly.





Adding custom business logic in ***public Serializable generate(…)*** and always generate unique value, work in high-volume, multi-threaded environment, if using server clusters, always generate unique value.

**Entity Class**



**Creating 3 objects which will automatically map to the database via the *id***



**READ / RETRIEVE**





**Querying objects with Hibernate**

**Hibernate Query Language (HQL)**

* Query language for retrieving objects
* Similar to nature to SQL
  + where, like, order by, join, in, etc.

**Special Note about Deprecated Method in Hibernate 5.2**

If you are using Hibernate 5.2 or higher, then the Query list() method has been deprecated.In your code you should make the following update:

**Replace** - session.createQuery("from Student").list()

**With -** session.createQuery("from Student").getResultList()





How To View Hibernate SQL Parameter Values

FAQ: How To View Hibernate SQL Parameter Values

**Question:**

I see hibernate printing out the query parameters as ? in the console. Is it possible to printout the value that was actually queried on the database. Asking as this would help in the debugging purpose.

Answer:

When using Hibernate, if you log the Hibernate SQL statements, you will see this:

Hibernate: insert into student (email, first\_name, last\_name, id) values (?, ?, ?, ?)

However, for debugging your application, you want to see the actual parameter values in the Hibernate logs. Basically, you want to get rid of the question marks in the Hibernate logs.

You can view the actual parameters by viewing the low-level trace of the Hibernate logs. This is not set up by default. However, we can add log4j to allow us to see these low-level logs.

#### **Here is an overview of the process:**

1. Add log4j to your project classpath

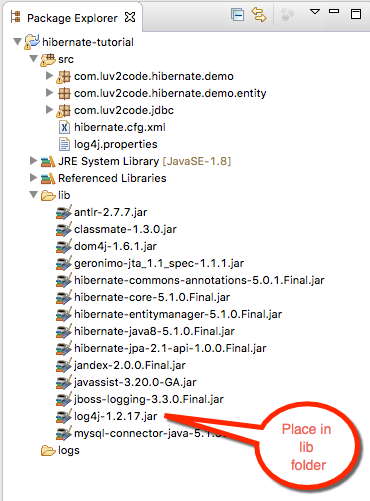
2. Add log4j.properties to your “src” directory

#### **Here are the detailed steps:**

**1. Add log4j to your project classpath**

1a. Download log4j v1.2.17 from this link: – <http://central.maven.org/maven2/log4j/log4j/1.2.17/log4j-1.2.17.jar>

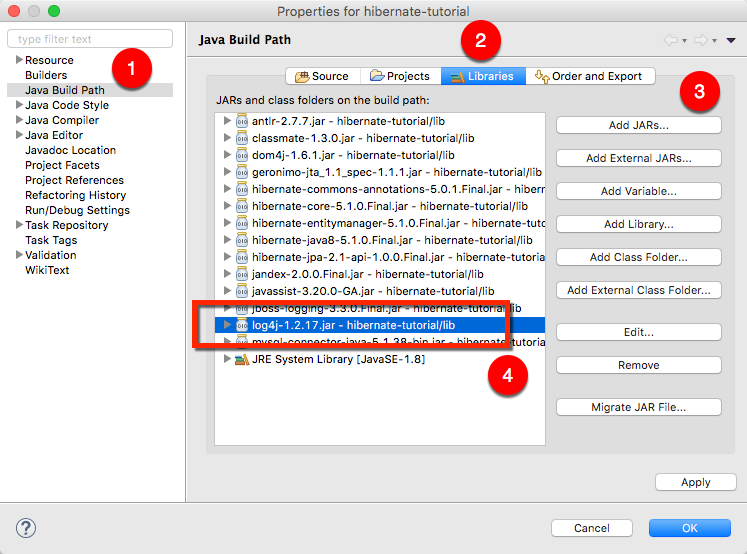
1b. Copy this file to your project’s lib directory



1c. Right-click your Eclipse project and select **Properties**

1d. Select **Build Path > Libraries > Add JARS…**

1e. Select the **log4j-1.2.17.jar** file from the **lib** directory

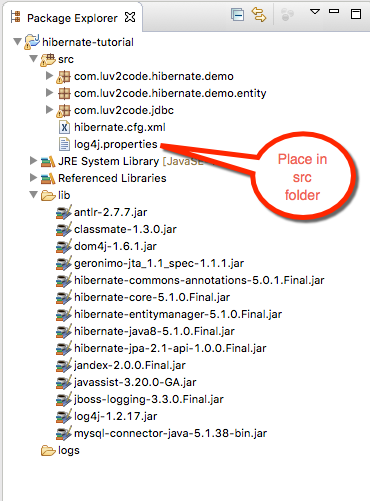


**2. Add log4j.properties to your “src” directory**

2a. Copy the text from below

1. # Root logger option
2. log4j.rootLogger=DEBUG, stdout
4. # Redirect log messages to console
5. log4j.appender.stdout=org.apache.log4j.ConsoleAppender
6. log4j.appender.stdout.Target=System.out
7. log4j.appender.stdout.layout=org.apache.log4j.PatternLayout
8. log4j.appender.stdout.layout.ConversionPattern=%d{yyyy-MM-dd HH:mm:ss} %-5p %c{1}:%L - %m%n
10. log4j.logger.org.hibernate=TRACE

2b. Save this file as "log4j.properties" in your “src” directory



Note: This file has an important setting:

log4j.logger.org.hibernate=TRACE

This allows you see a low-level trace of Hibernate and this allows you see the real SQL parameter values.

Now run your application. You will see a lot of low-level TRACE logs in the Eclipse Console window.

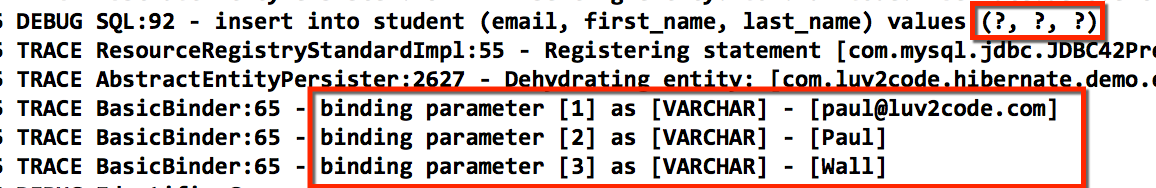
Right-click in the Eclipse Console window and select **Find/Replace…**

Search for: **binding parameter**

or search for: **extracted value**

(the search string changes depending on which version of Hibernate you are using)

You will see the logs with the real parameter values. Congrats!



**Update Objects using Hibernate**