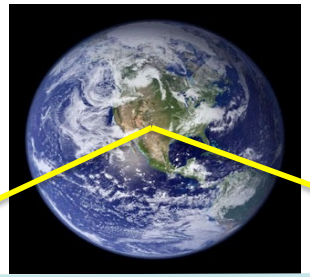
Persistency (JPA)

ORM - DAO



WORLD

MODEL

UML

ORM

ERA

ARCHITECTURE

DAO

DB

Actual storage

FS

platforms

Data

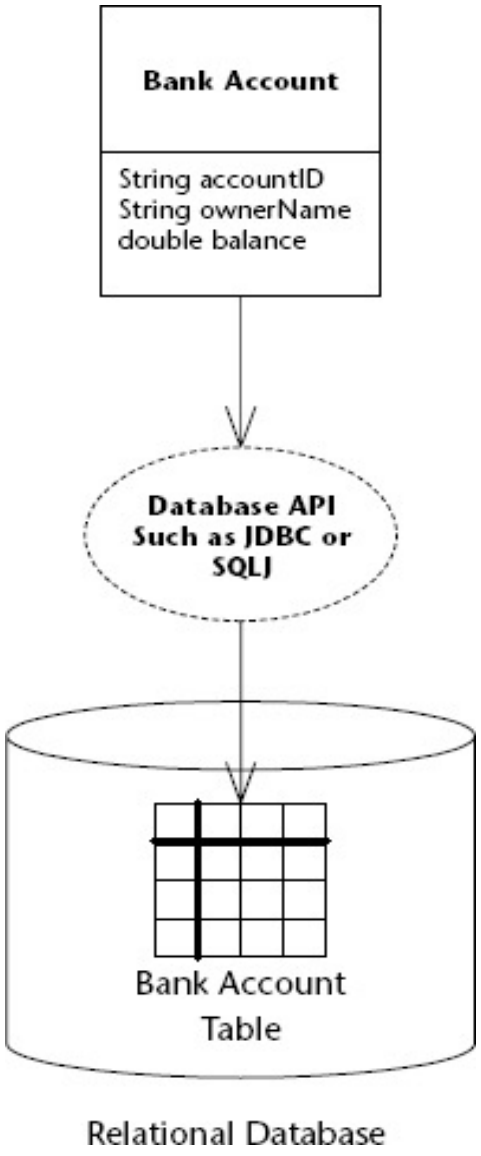
Object

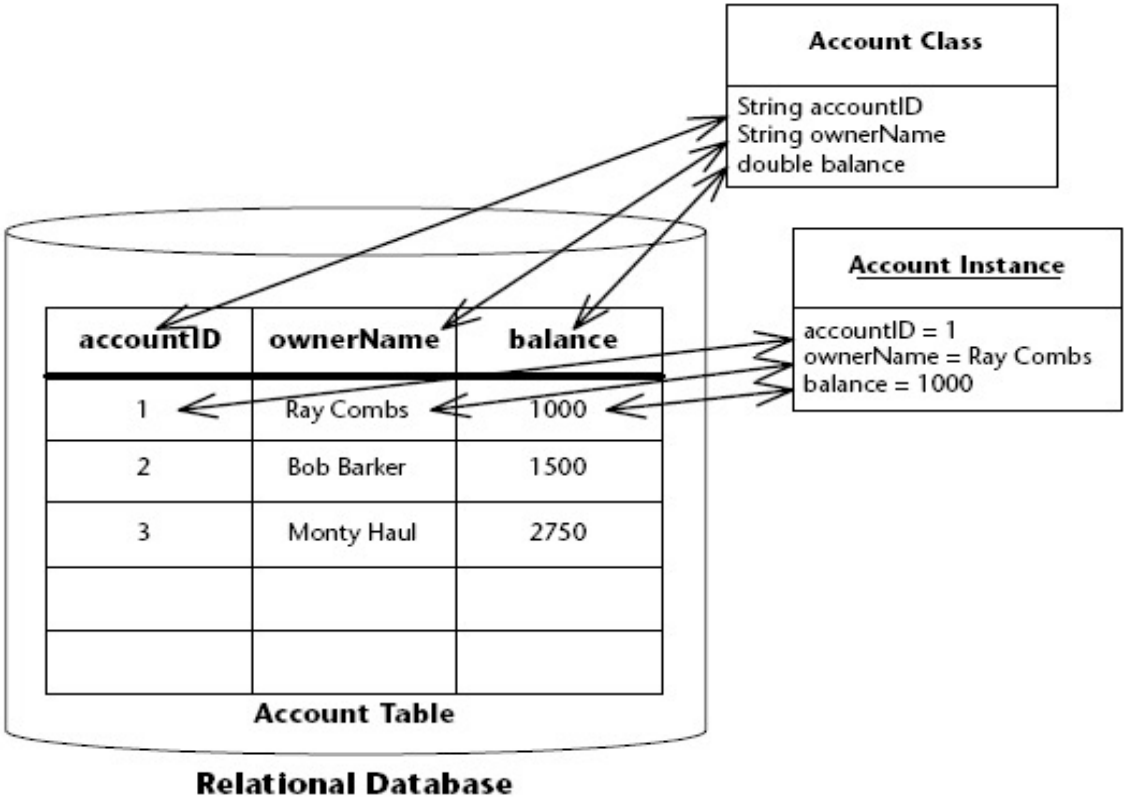
temp

tools

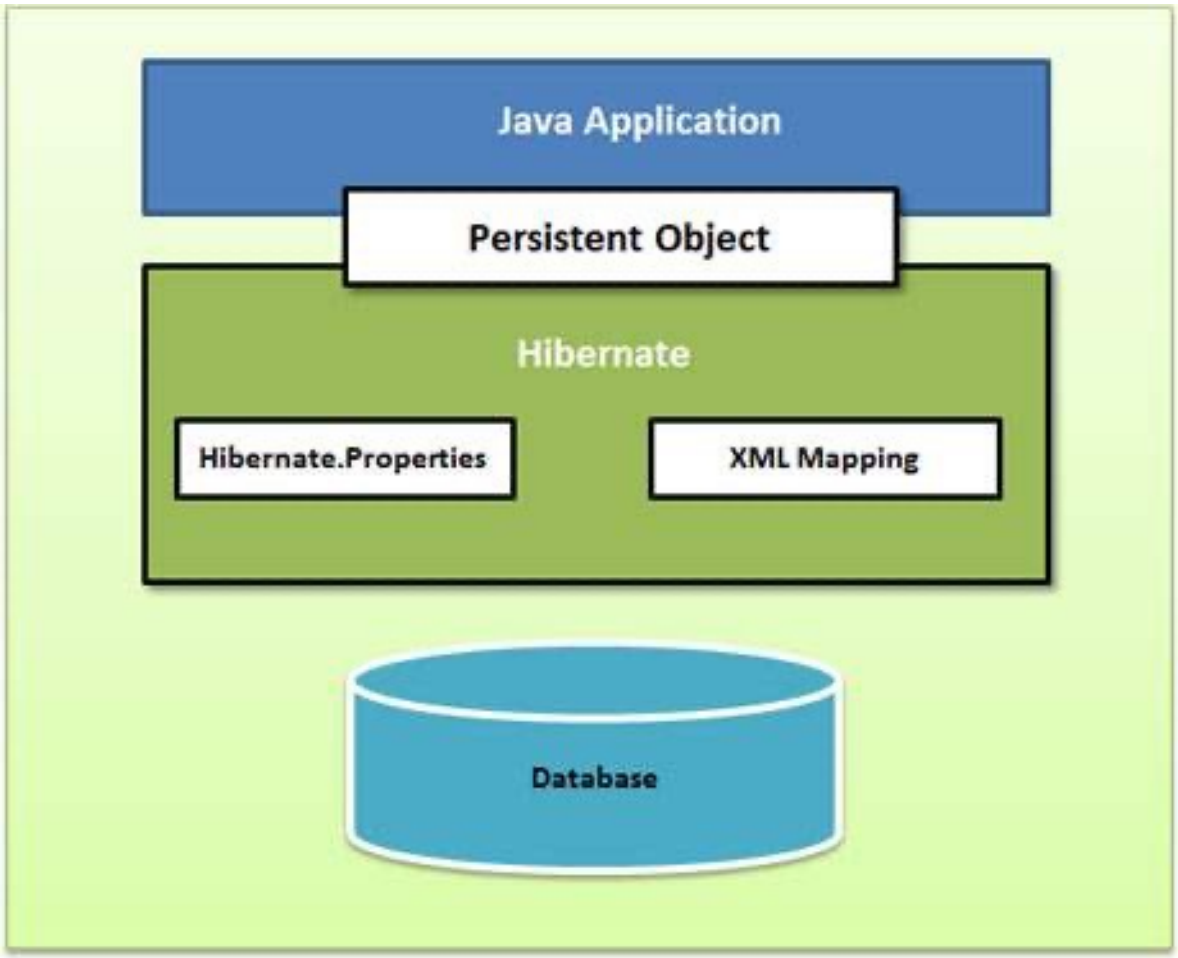
ORM

Object-Relational Mapping is NOT serialization!



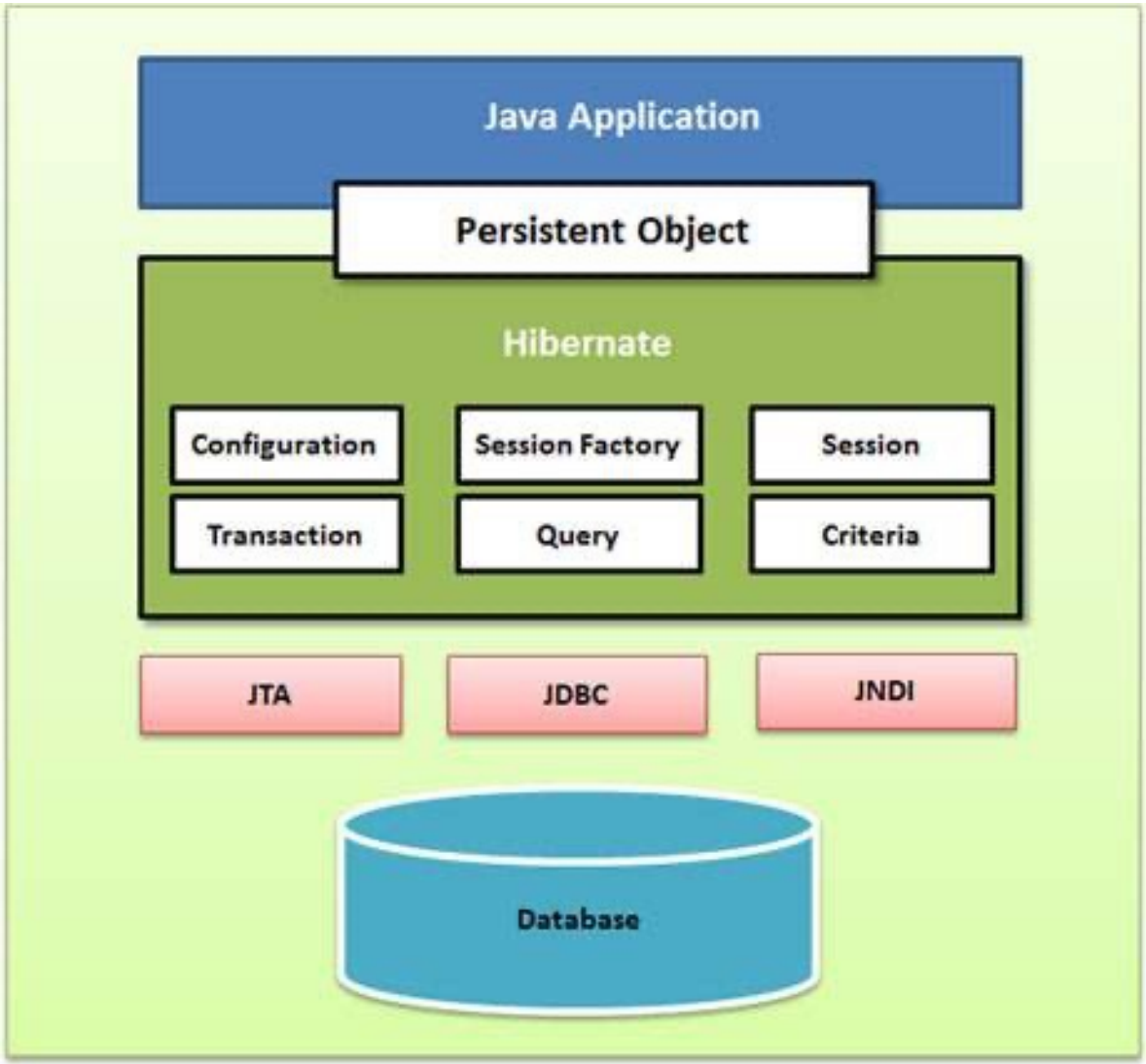


Hibernate



da https://www.tutorialspoint.com/hibernate/hibernate_architecture.htm

Hibernate


Java Application
Persistent Object
Hibernate
Configuration
Session Factory
Session
Transaction
Query
Criteria
JTA
JDBC
JNDI
Database

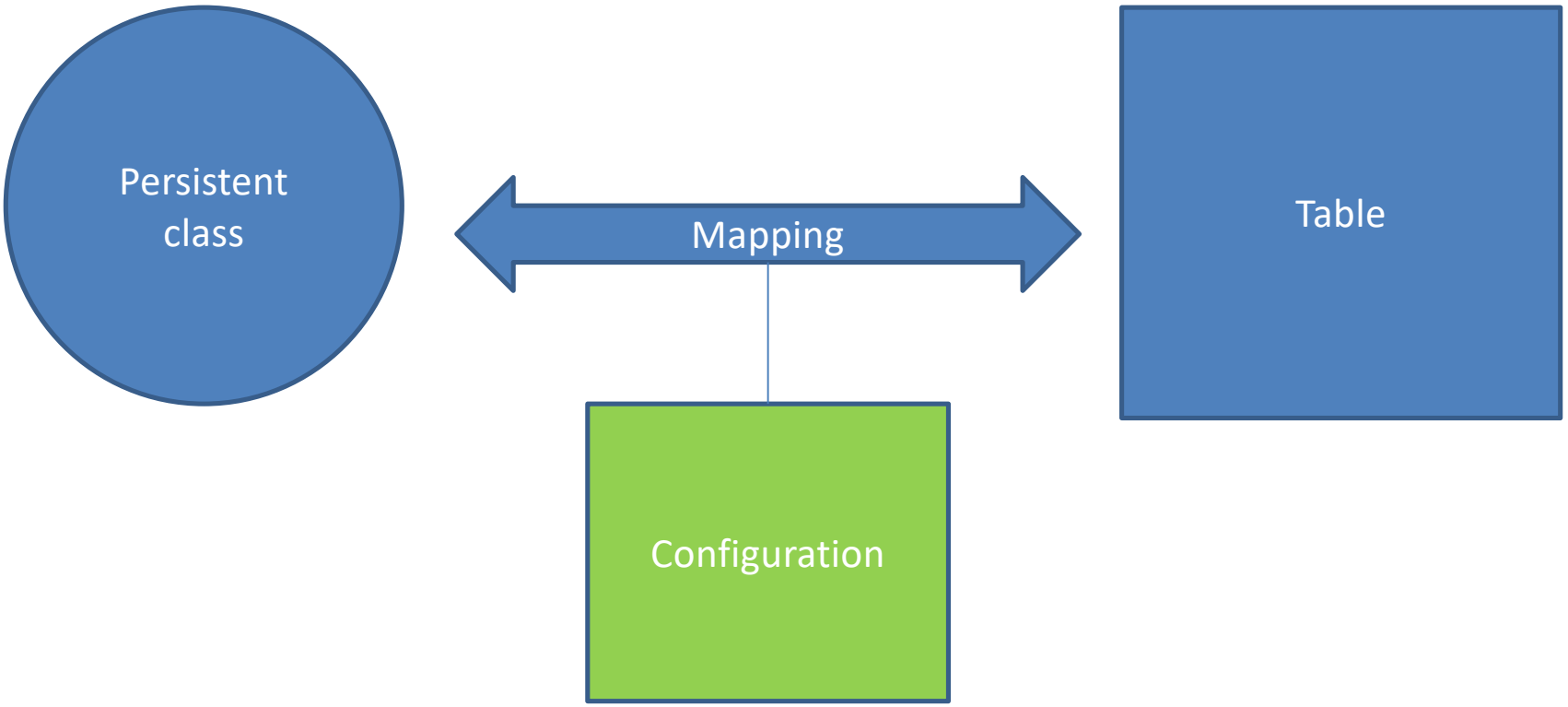
Hibernate properties

Sr. No.	Properties & Description
1	**hibernate.dialect** This property makes Hibernate generate the appropriate SQL for the chosen database.
2	**hibernate.connection.driver_class** The JDBC driver class.
3	**hibernate.connection.url** The JDBC URL to the database instance.
4	**hibernate.connection.username** The database username.
5	**hibernate.connection.password** The database password.
6	**hibernate.connection.pool_size** Limits the number of connections waiting in the Hibernate database connection pool.
7	**hibernate.connection.autocommit** Allows autocommit mode to be used for the JDBC connection.

Hibernate properties (AS version)

Sr. No.	Properties & Description
1	**hibernate.connection.datasource** The JNDI name defined in the application server context, which you are using for the application.
2	**hibernate.jndi.class** The InitialContext class for JNDI.
3	**hibernate.jndi.<JNDIpropertyname>** Passes any JNDI property you like to the JNDI *InitialContext*.
4	**hibernate.jndi.url** Provides the URL for JNDI.
5	**hibernate.connection.username** The database username.
6	**hibernate.connection.password** The database password.

Hibernate configuration



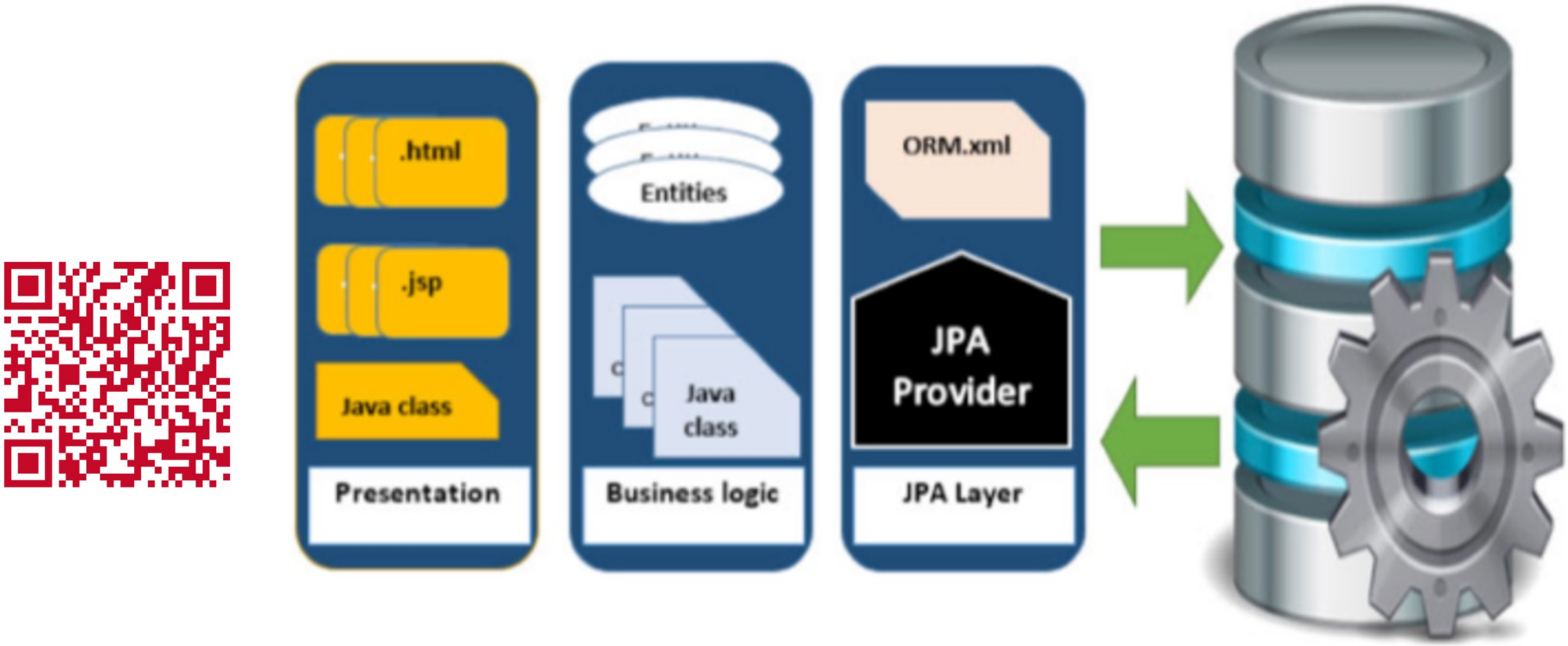
Hibernate Query Language

https://www.tutorialspoint.com/hibernate/index.htm

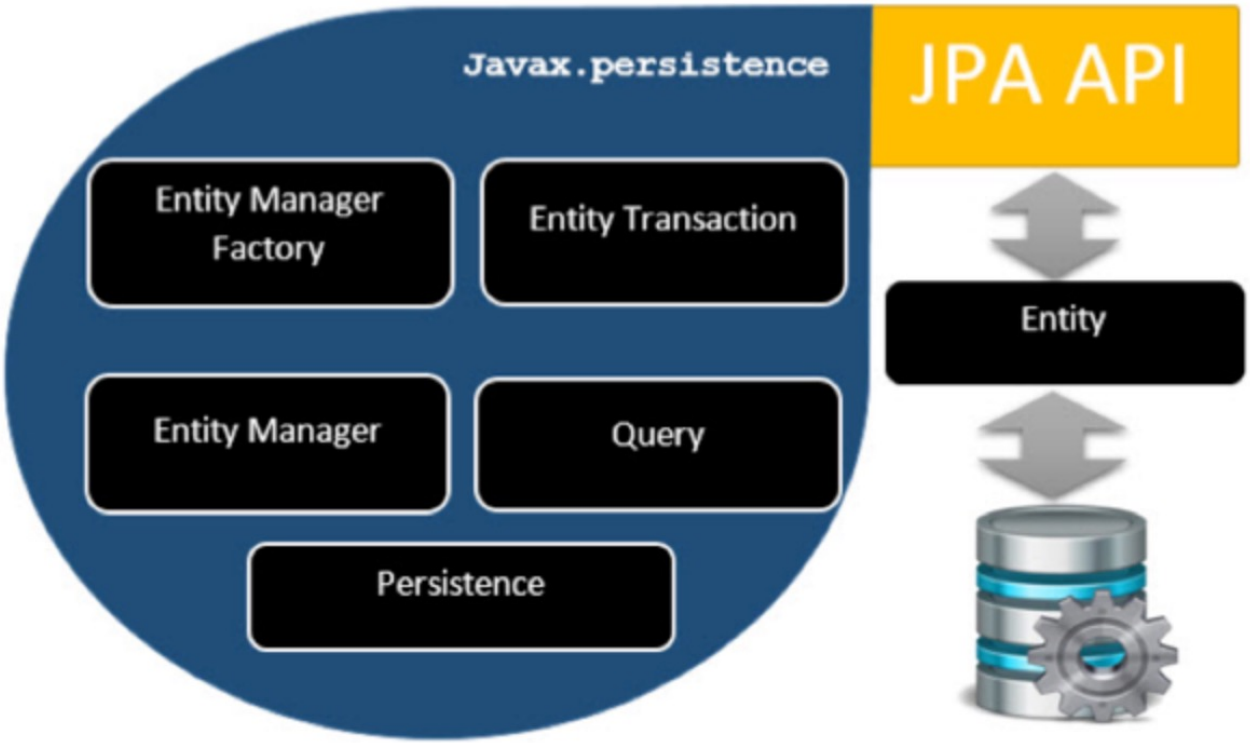
Hibernate tutorial

https://www.tutorialspoint.com/hibernate/index.htm

JPA in the architecture



JPA classes



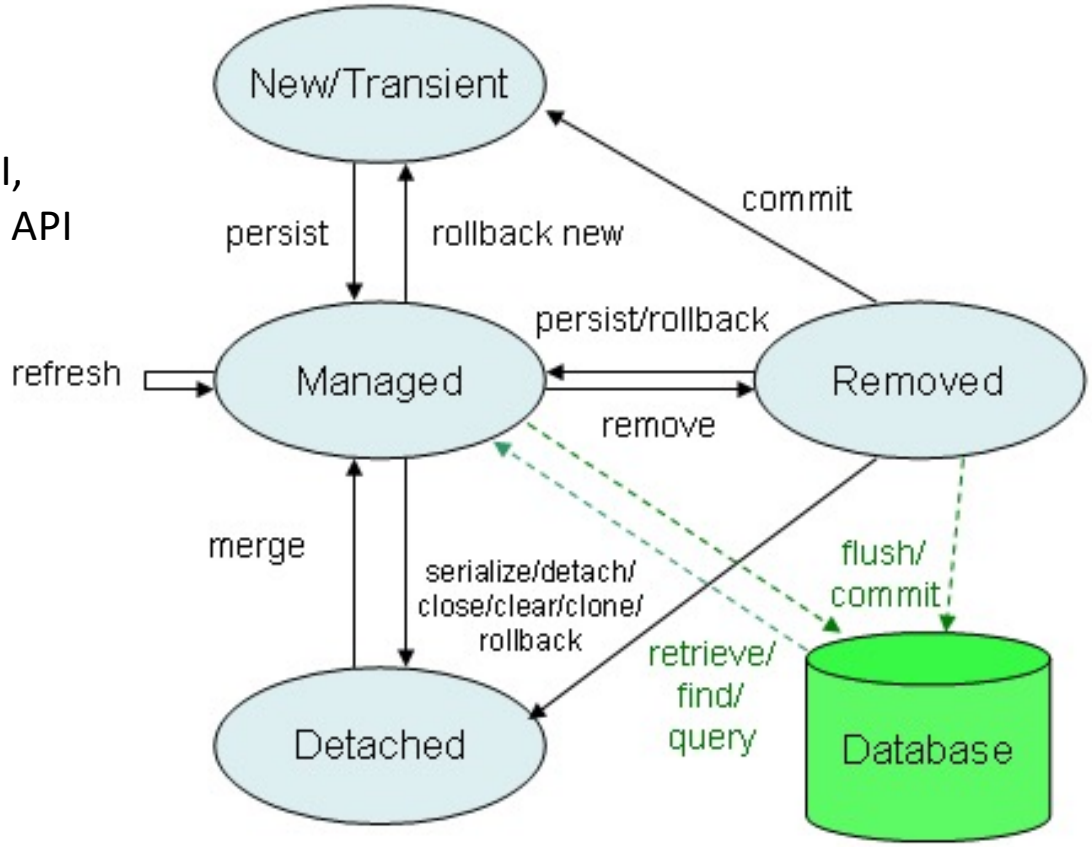
Units	Description
Persistence	This class contain static methods to obtain EntityManagerFactory instance.
EntityManagerFactory	This is a factory class of EntityManager. It creates and manages multiple EntityManager instances.
EntityManager	It is an Interface, it manages the persistence operations on objects. It works like factory for Query instance.
Entity	Entities are the persistence objects, stores as records in the database.
EntityTransaction	It has one-to-one relationship with EntityManager. For each EntityManager, operations are maintained by EntityTransaction class.
Query	This interface is implemented by each JPA vendor to obtain relational objects that meet the criteria.

JPA Providers

JPA is an open source API, therefore various enterprise vendors such as Oracle, Redhat, Eclipse, etc. provide new products by adding the JPA persistence flavor in them. Some of these products include: **Hibernate, Eclipselink, Toplink, Spring Data JPA, etc.**

Object states in JPA

JPA: Java Persistence API,
now Jakarta Persistence API



Persistent class

```
@Entity
@Table(name = "EMPLOYEE", schema = "APP", catalog = "")
public class EmployeeEntity {…}
```

Manager and persistence

```
EntityManagerFactory emf= Persistence.createEntityManagerFactory("default");
EntityManager em=emf.createEntityManager();

EmployeeEntity e1=new EmployeeEntity("Pippo","De Pippis");
em.persist(e1);

Query query =em.createQuery("FROM "+EmployeeEntity.class.getSimpleName()+" WHERE ID=0");
      list=query.getResultList();
      for (EmployeeEntity employee : list) {
         System.out.println(employee.getId()+" "+employee.getLastname());
      }
```

Some annotations

Annotation	Description
@Entity	Declares the class as an entity or a table.
@Table	Declares table name.
@Basic	Specifies non-constraint fields explicitly.
@Id	Specifies the property, use for identity (primary key of a table) of the class.
@GeneratedValue	Specifies how the identity attribute can be initialized such as automatic, manual, or value taken from a sequence table.
@Transient	Specifies the property that is not persistent, i.e., the value is never stored in the database.
@Column	Specifies the column attribute for the persistence property.

Annotation	Description
@GeneratedValue	Specifies how the identity attribute can be initialized such as automatic, manual, or value taken from a sequence table.
@SequenceGenerator	Specifies the value for the property that is specified in the @GeneratedValue annotation. It creates a sequence.
@TableGenerator	Specifies the value generator for the property specified in the @GeneratedValue annotation. It creates a table for value generation.
@NamedQuery	Specifies a Query using static name.

Queries: methods

List getResultList()
Object getSingleResult()
executeUpdate(): execute UPDATE o DELETE
setMaxResults(int max)
setFirstResult(int pos)

setParameter(…)

setFlushMode(FlushModeType flushmode)

How do you express queries in JPQL?

FROM:

```
String hql = "FROM com.hibernatebook.criteria.Employee";
Query query = session.createQuery(hql);
List<Employee> results = query. getResultList();
```

SELECT

```
String hql = "SELECT Employee.firstName FROM Employee";
Query query = session.createQuery(hql);
List results = query. getResultList();
```

AS: used to assign aliases to the classes in your HQL queries, especially when you have the long queries.

```
String hql = " SELECT E.firstName FROM Employee AS E";
Query query = session.createQuery(hql);
List results = query. getResultList();
```

The **AS** keyword is optional, hence also:

```
String hql = " SELECT E.firstName FROM Employee E";
Query query = session.createQuery(hql);
List results = query. getResultList();
```

How do you express queries?

WHERE

String hql = "**SELECT .firstName FROM Employee E WHERE E.Salary > 2000**";
Query query = session.createQuery(hql);
List results = query.list();

Conditional expressions:

c.name = 'books'
path_expression [NOT] IN (List_of_values)
string_value_path_expression [NOT] LIKE pattern_value

- *pattern values:*
  - *NULL,*
  - *EMPTY (for collections)*
  - *MEMBER OF collection*
  - *_ for single char,*
  - *% for multiple chars*

Aggregation expressions:

- *MAX*
- *MIN*
- *AVG*
- *COUNT*

How do you express queries?

GROUP BY:

```
SELECT a.categoria, COUNT(a.id) FROM Articolo a GROUP BY a.categoria
```

GROUP BY ... HAVING:

```
SELECT c.utente, COUNT(c.id) FROM Categoria c GROUP BY c.utente HAVING COUNT(c.id) > 5
```

ORDER BY:

```
SELECT c FROM Categoria c ORDER BY c.nome ASC
SELECT c FROM Categoria c ORDER BY c.nome ASC, c.titolo DESC
```

SUBQUERIES:

```
SELECT a FROM Articolo a WHERE a.utente IN
        (SELECT c.utente FROM Categoria c WHERE c.nome LIKE :nome)
```

How do you express queries?

UPDATE:

String hql = "**UPDATE Categoria c SET nome = “libri”, ... WHERE c.nome = “lettura”**";
Query query = session.createQuery(hql);
List<Employee> results = query. executeUpdate();

DELETE

String hql = "**DELETE Categoria c WHERE c.nome = “prova”**";
Query query = session.createQuery(hql);
List results = query. getResultList();

Installing
Derby DB and Hibernate

Hibernate quick installation

- download Hibernate from http://sourceforge.net/projects/hibernate/
- unzip it

or: use maven

Derby quick installation

- download Derby from http://db.apache.org/derby/ and unzip it
- For manual use:
  - in a shell, set DERBY_HOME and execute setEmbeddedCP

    DERBY_HOME=/*yourpathto*/db-derby-10.15.2-bin

    DERBY_HOME/bin/setEmbeddedCP
  - start Derby

    java –jar DERBY_HOME/lib/derbyrun.jar server start &

Using Derby and Hibernate with Intellij

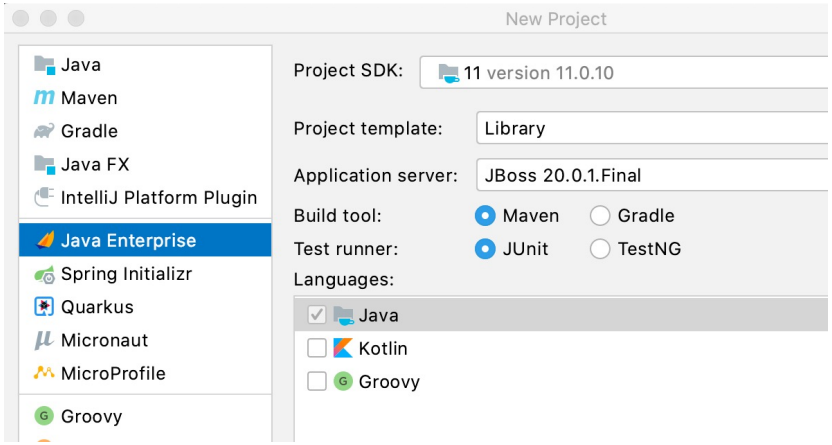
- Derby:
  - follow these instructions:

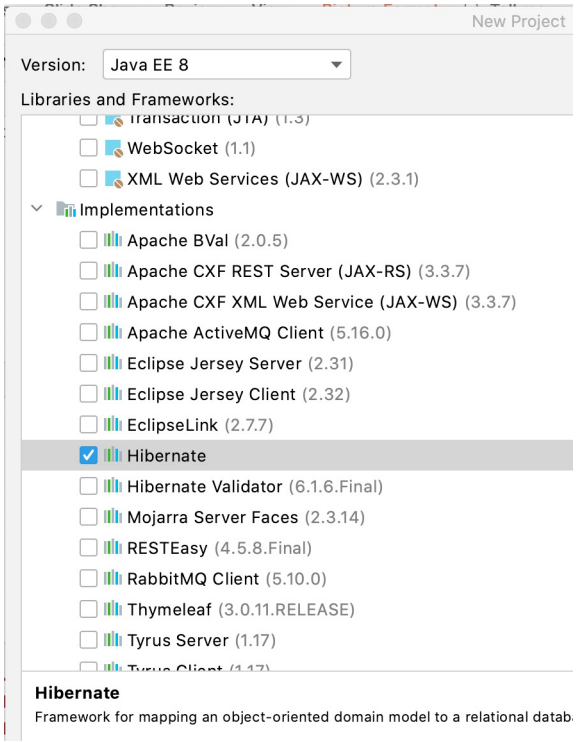
    https://www.jetbrains.com/help/idea/apache-derby.html
- Hibernate
  - follow these instructions:

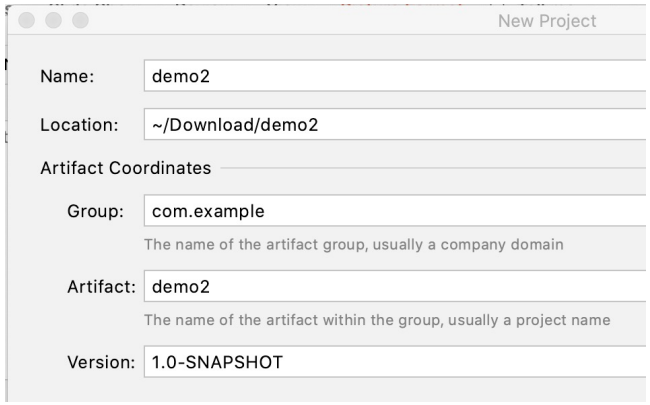
    https://www.jetbrains.com/help/idea/hibernate.html

Creating a JPA project in IntelliJ

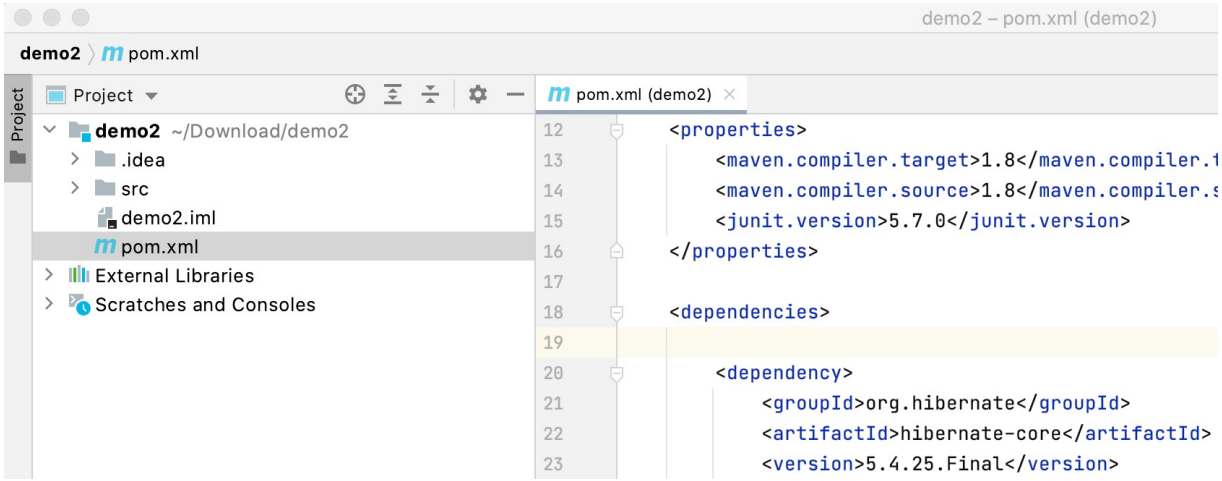
create an Hibernate project

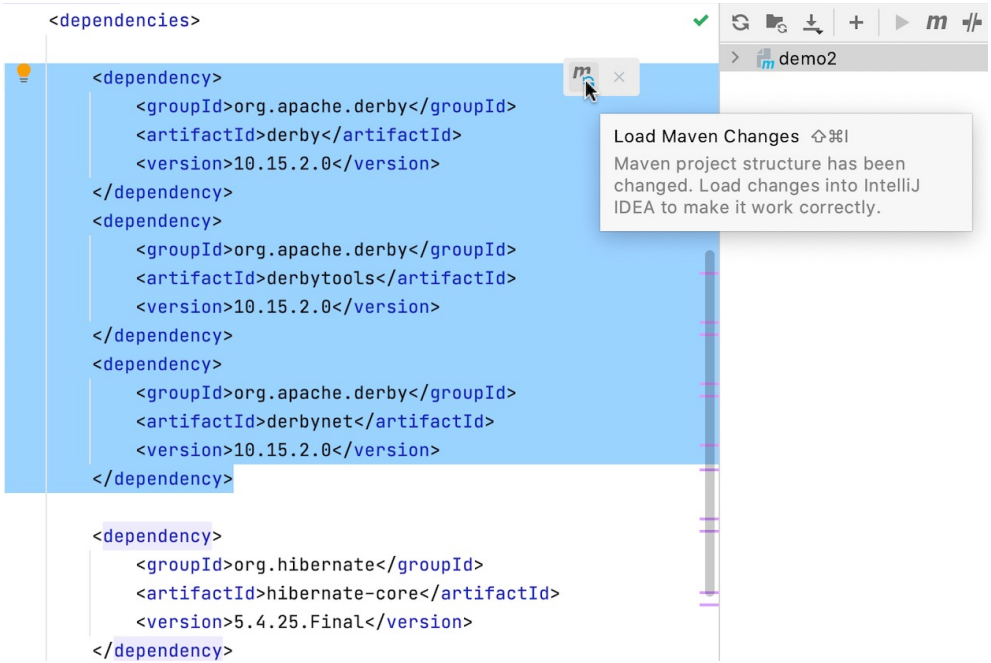


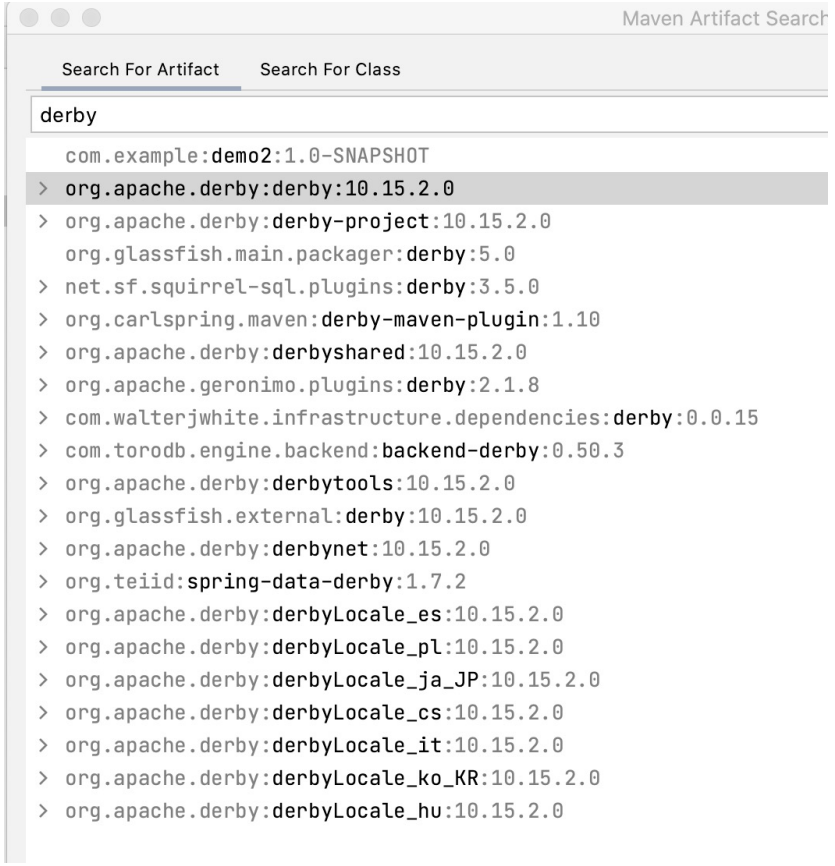




Adding Derby to project





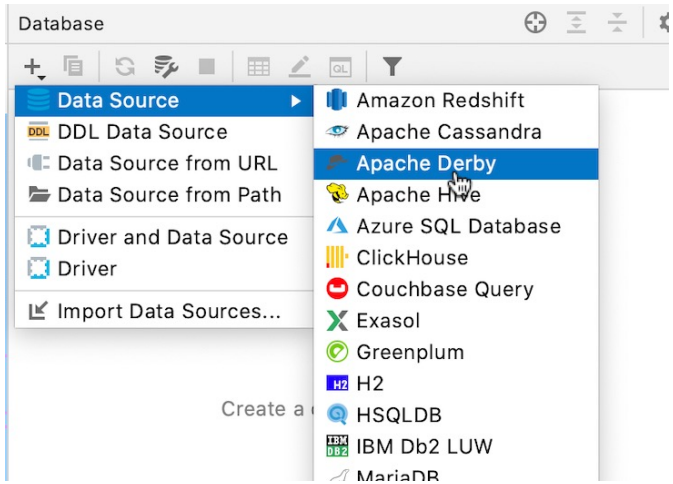


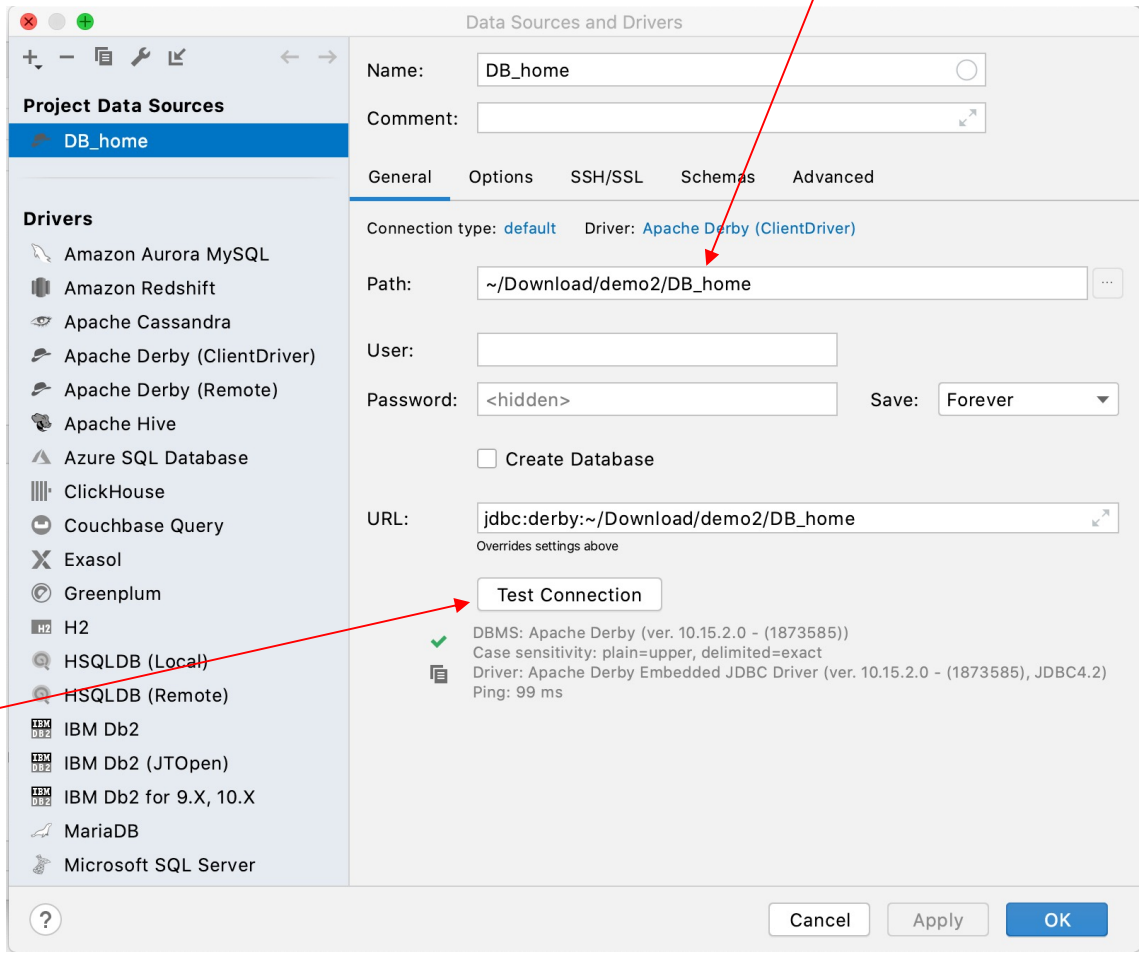
Adding Derby to project

```
MR-MacBookPro:Download ronchet$ export DERBY_HOME=/Users/ronchet/db-derby-10.15.
2.0-bin
MR-MacBookPro:Download ronchet$ echo $DERBY_HOME
/Users/ronchet/db-derby-10.15.2.0-bin
MR-MacBookPro:Download ronchet$
```

in this case:
in the project directory

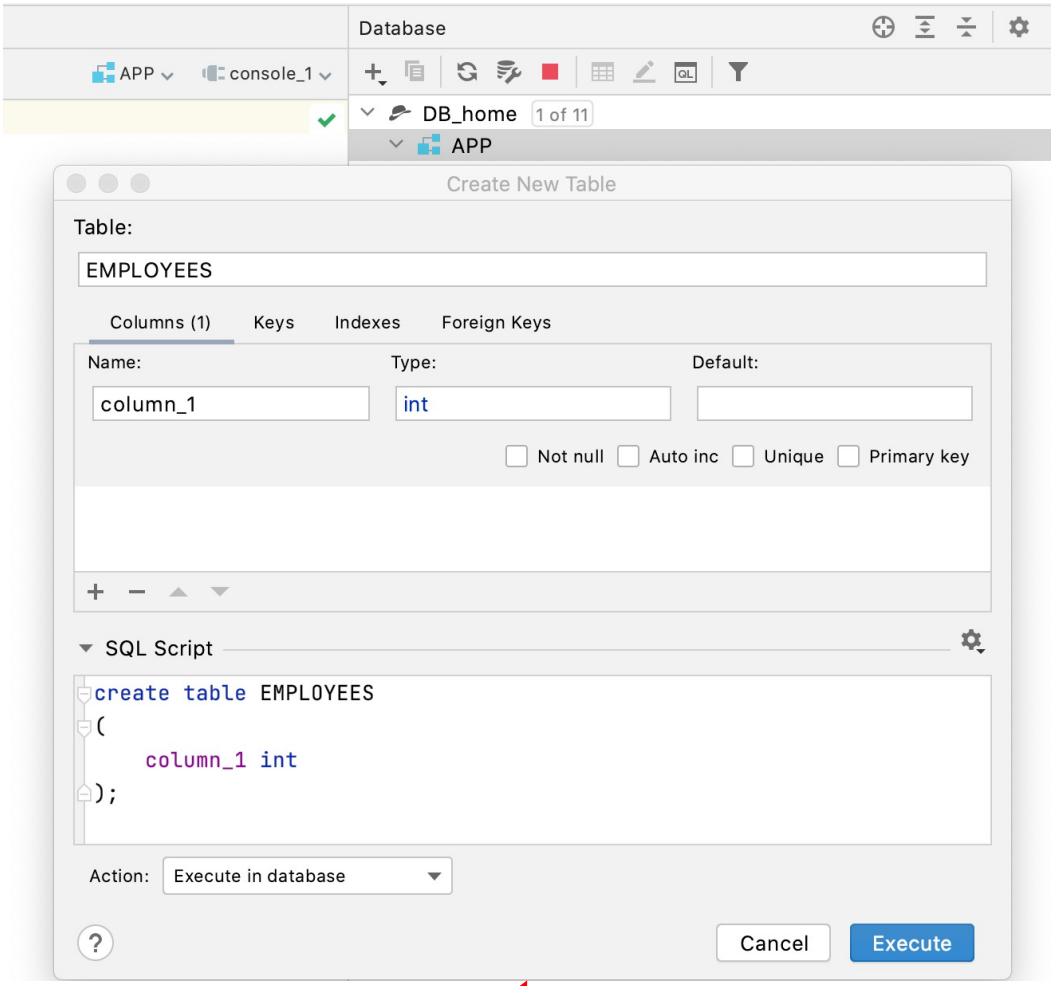
View- > Tool window -> Database





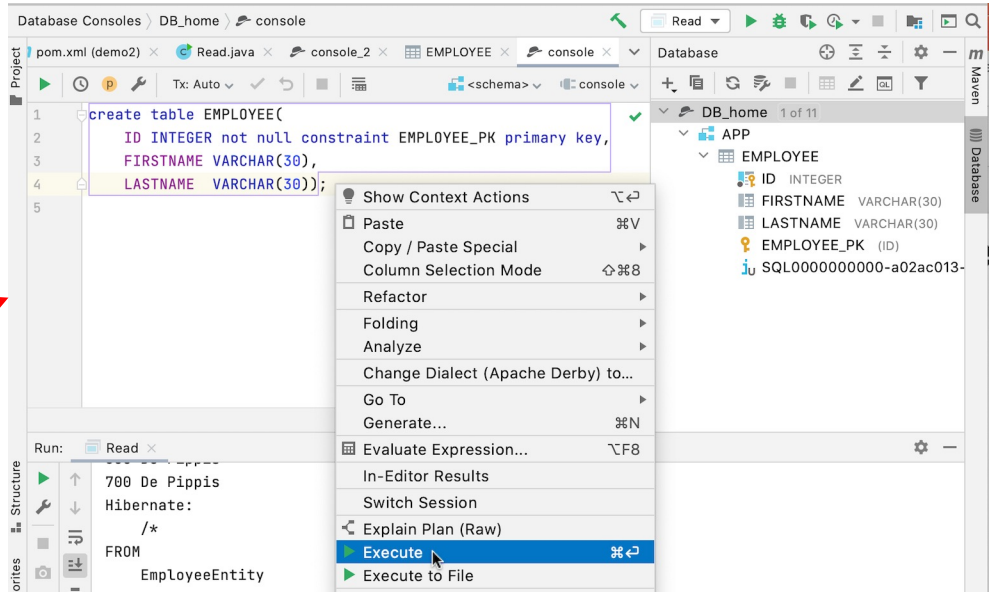
test

Create Table

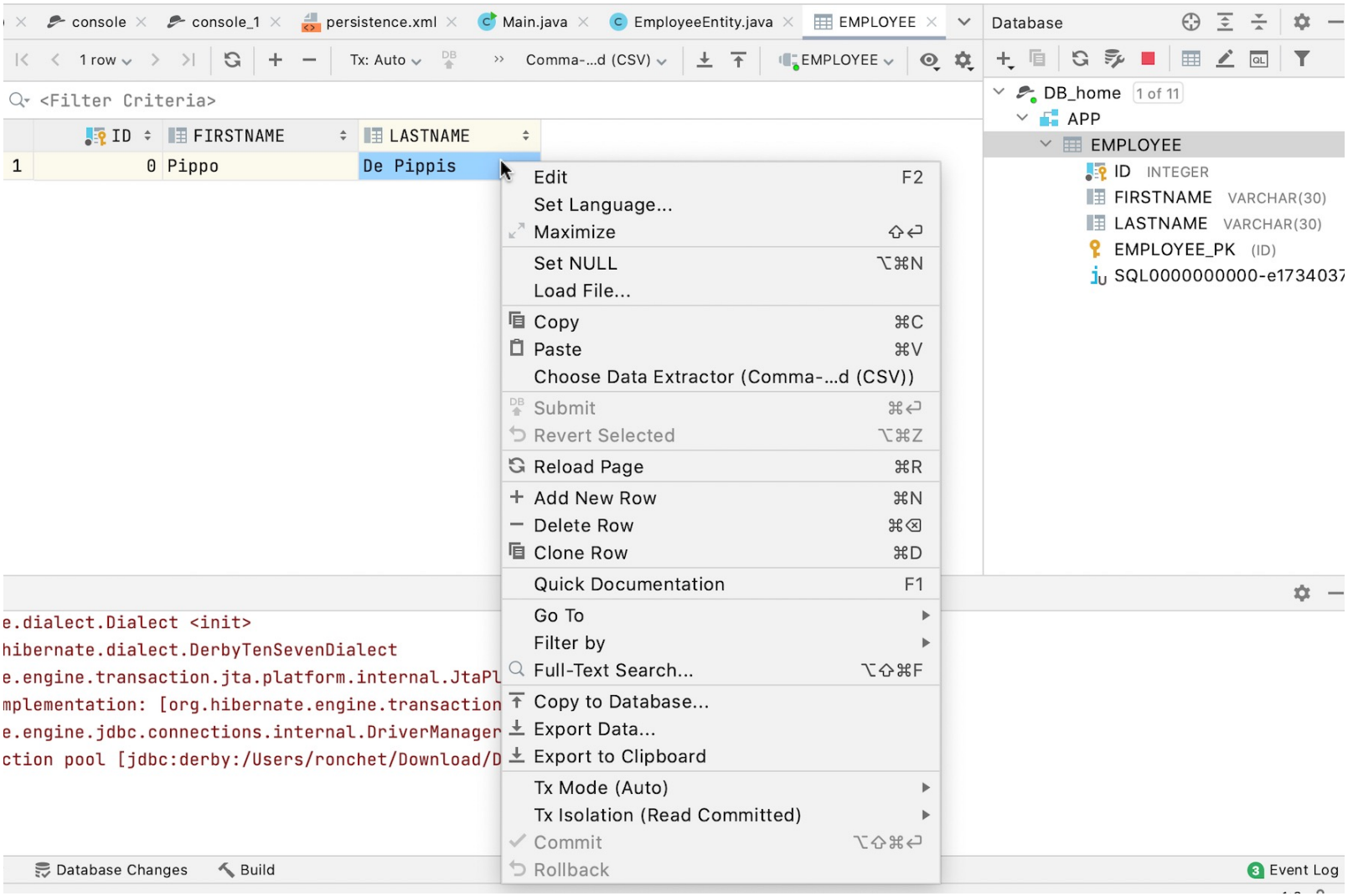


manually… or by SQL

```
create table EMPLOYEE(
    ID INTEGER not null constraint EMPLOYEE_PK primary key,
    FIRSTNAME VARCHAR(30),
    LASTNAME  VARCHAR(30));
```

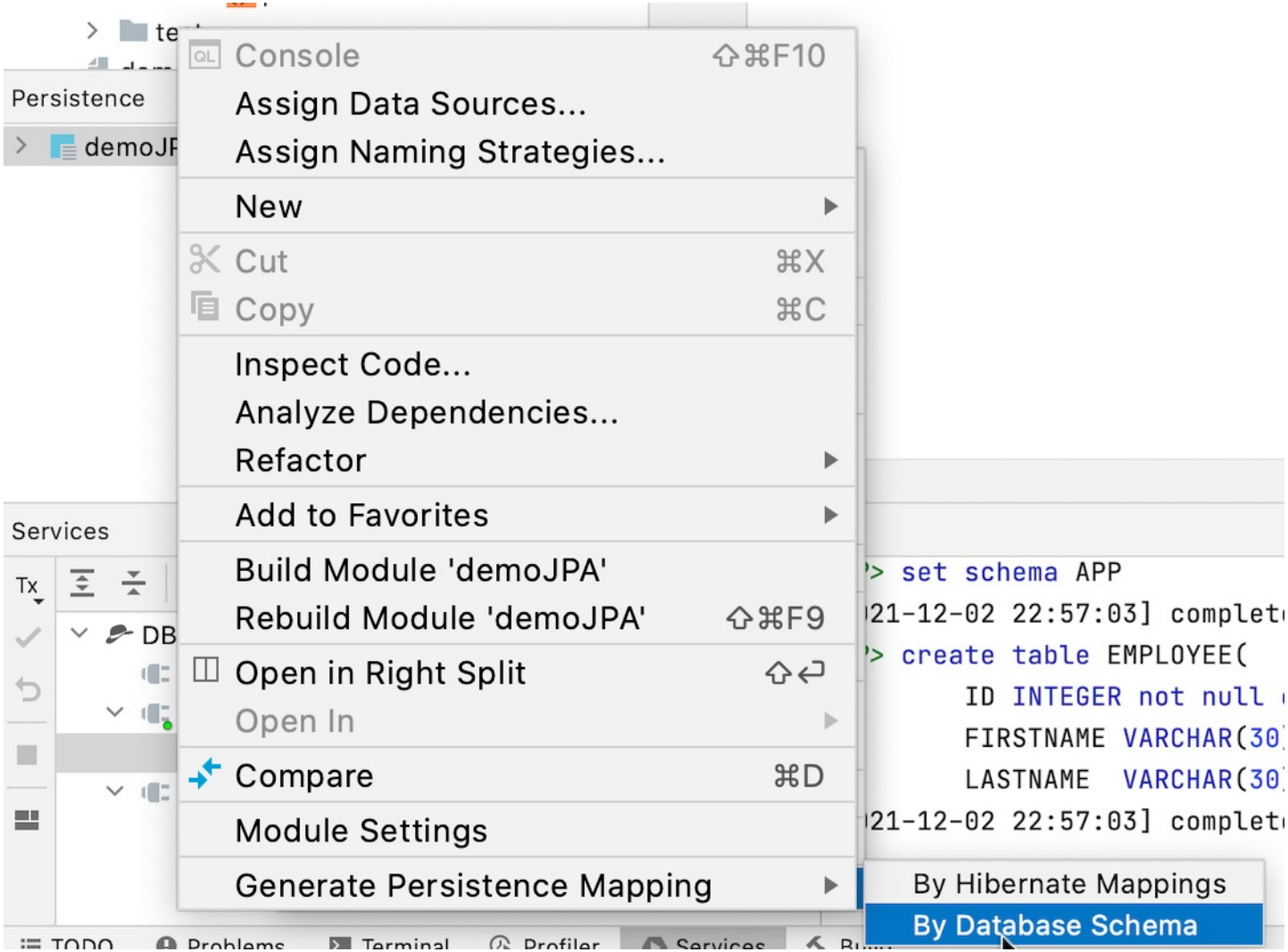


Edit DB data in IJ



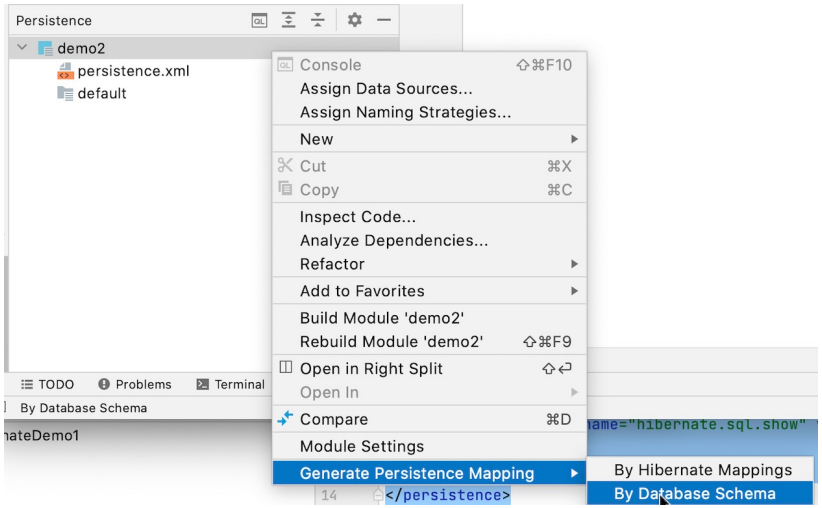
Generate mapping

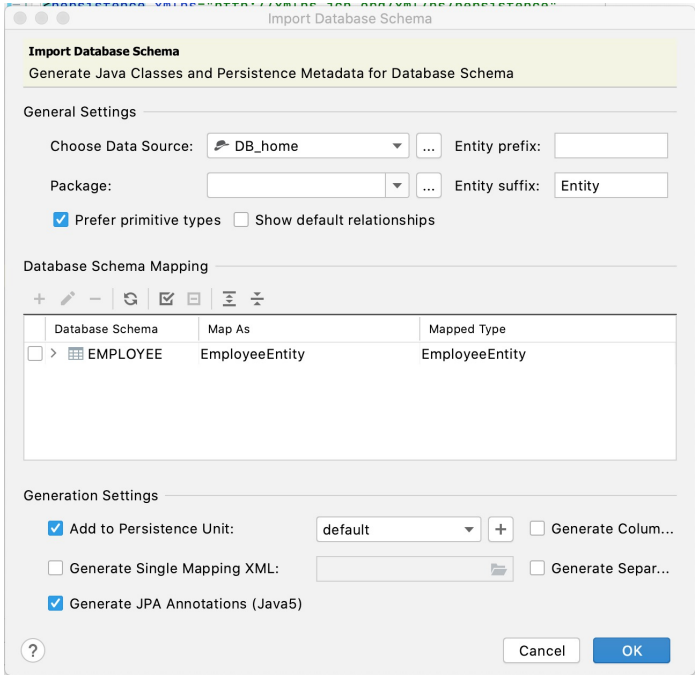
View- > Tool window -> Persistence

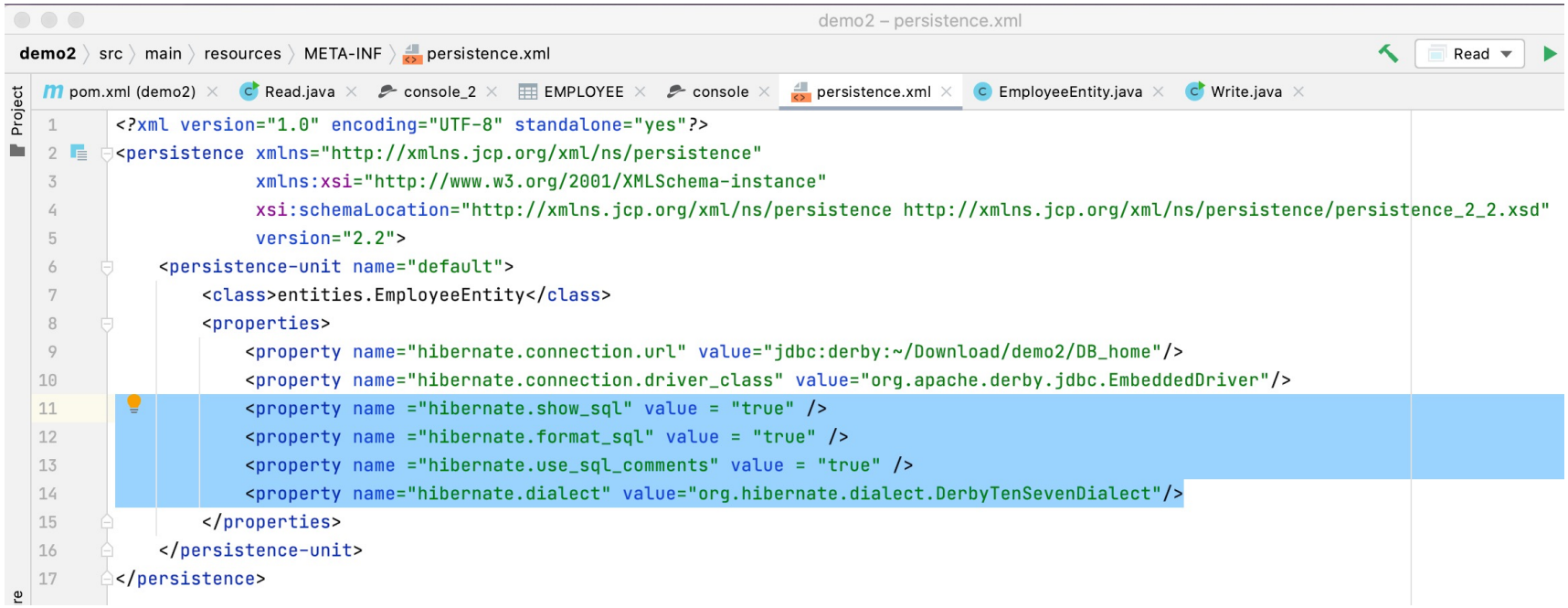


Update persistence.xml

View- > Tool window -> Persistence







add the blue lines

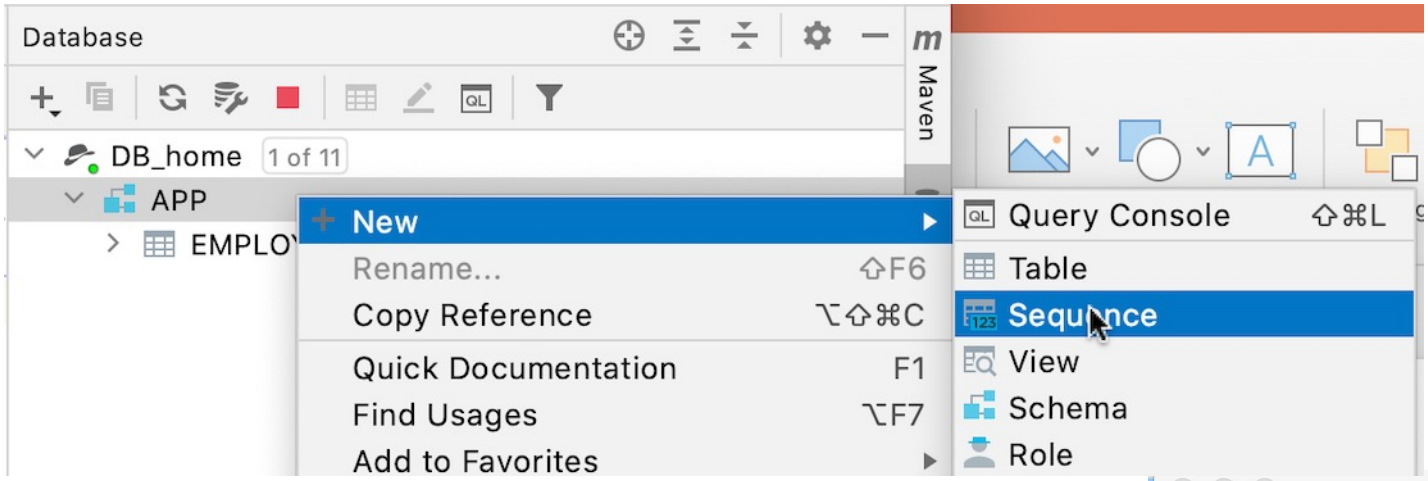
Configuration file?

If you are using Hibernate's proprietary API, you'll need the hibernate.cfg.xml.
If you are using JPA i.e. Hibernate EntityManager, you'll need
the persistence.xml.

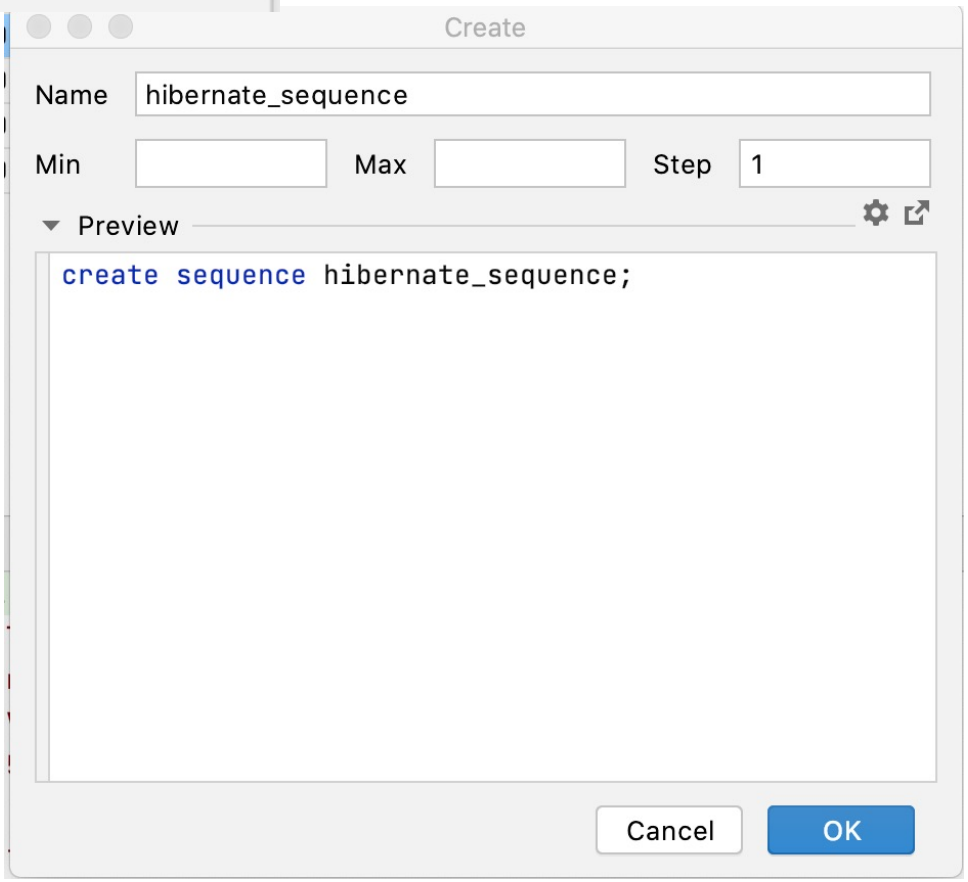
So you generally don't need both as you use **either** Hibernate proprietary API or JPA.

However, if you **were using** Hibernate Proprietary API and already have a hibernate.cfg.xml (and hbm.xml XML mapping files) but want to start using JPA, you can reuse the existing configuration files by referencing the hibernate.cfg.xml in the persistence.xml in the hibernate.ejb.cfg file property - and thus have both files.

Create sequence



name it
hibernate_sequence



EmployeeEntity

```
package entities;
import javax.persistence.*;

@Entity
@Table(name = "EMPLOYEE", schema = "APP", catalog = "")
public class EmployeeEntity {
    private int id;
    private String firstname;
    private String lastname;

    public EmployeeEntity() {
    }
    public EmployeeEntity(String firstname, String lastname) {
        this.firstname = firstname;
        this.lastname = lastname;
    }

    @Id
    @Column(name = "ID")
    @GeneratedValue(strategy=SEQUENCE)
    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }
```

add the red lines

EmployeeEntity

```
@Basic
@Column(name = "FIRSTNAME")
public String getFirstname() {
    return firstname;
}
public void setFirstname(String firstname) {
    this.firstname = firstname;
}

@Basic
@Column(name = "LASTNAME")
public String getLastname() {
    return lastname;
}
public void setLastname(String lastname) {
    this.lastname = lastname;
}
```

EmployeeEntity

```
    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;

        EmployeeEntity that = (EmployeeEntity) o;
        if (id != that.id) return false;
        if (firstname != null ? !firstname.equals(that.firstname) : that.firstname !=
null) return false;
        if (lastname != null ? !lastname.equals(that.lastname) : that.lastname != null)
return false;

        return true;
    }

    @Override
    public int hashCode() {
        int result = id;
        result = 31 * result + (firstname != null ? firstname.hashCode() : 0);
        result = 31 * result + (lastname != null ? lastname.hashCode() : 0);
        return result;
    }
}
```

Write

```
import entities.EmployeeEntity;

import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.EntityTransaction;
import javax.persistence.Persistence;

public class Write {
    public static void main(String[] args) {
        EntityManagerFactory emf= Persistence.createEntityManagerFactory("default");
        EntityManager em=emf.createEntityManager();
        EntityTransaction transaction=em.getTransaction();
        try {
            transaction.begin();
            EmployeeEntity e1=new EmployeeEntity("Pippo","De Pippis");
            em.persist(e1);
            transaction.commit();
        } finally {
            if (transaction.isActive())
                transaction.rollback();
            em.close();
            emf.close();
        }
    }
}
```

Read

```
import entities.EmployeeEntity;

import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.EntityTransaction;
import javax.persistence.Persistence;

public class Read {
    public static void main(String[] args) {
        EntityManagerFactory emf= Persistence.createEntityManagerFactory("default");
        EntityManager em=emf.createEntityManager();
        Query query = em.createQuery("FROM " + EmployeeEntity.class.getSimpleName());
        List<EmployeeEntity> list=query.getResultList();
        for (EmployeeEntity employee : list) {
            System.out.println(employee.getId()+" "+employee.getLastname());
        }
        query =em.createQuery("FROM "+EmployeeEntity.class.getSimpleName()+" WHERE ID=0");
        list=query.getResultList();
        for (EmployeeEntity employee : list) {
            System.out.println(employee.getId()+" "+employee.getLastname());
        }
        em.close();
        emf.close();
    }
```

first and second level caching

first level caching: it's associated with the current session and is used to reduce the number of SQL statements within the same transaction. It's on by default.

second level caching: it is used **across sessions**. Hibernate provides a flexible concept to exchange cache providers for the second-level cache. **It's not on by default** but it needs some configuration in your persistence.xml to tell Hibernate to turn on the cache.

The shared-cache-mode element has four possible values:
» **ALL**: Causes all entities and entity related state and data to be cached
» **NONE**: Causes caching to be disabled for the persistence unit and all caching is turned off for all entities
» **ENABLE_SELECTIVE**: Enables the cache and causes entities for which @Cacheable(true) is specified to be cached
» **DISABLE_SELECTIVE**: Enables the cache and causes all entities to be cached except those for which @Cacheable(false) is specified

http://www.masterthebоss.com/jboss-frameworks/hibernate-jpa/hibernate-cache/using-hibernate-second-level-cache-with-jboss-as-5-6-7?showall=

Introduction to Entities

Entities

- Entities have a client-visible, persistent *identity* (the primary key) that is distinct from their object reference.
- Entities have persistent, client-visible *state*.
- Entities are *not remotely accessible*.
- An entity's *lifetime* may be completely independent of an application's lifetime.
- Entities can be used in both Java EE and J2SE environments

Entities - example

This demo entity represents a Bank Account. The entity is not a remote object and can only be accessed locally by clients. However, it is made serializable so that instances can be passed by value to remote clients for local inspection. Access to persistent state is by direct field access.

```
package examples.entity.intro;
import java.io.Serializable;
import javax.persistence.Entity;
import javax.persistence.Id;
@Entity
public class Account implements Serializable {
    // The account number is the primary key
    @Id
    public int accountNumber;
    public int balance;
    private String ownerName;
    String getOwnerName() {return ownerName;}
    void setOwnerName(String s) {ownerName=s;}

    /** Entities must have a public no-arg constructor */
    public Account() {
        // our own simple primary key generation
        accountNumber = (int) System.nanoTime();
    }
```

Entities - example

```
    public void deposit(int amount) {
       balance += amount;
    }
    public int withdraw(int amount) {
        if (amount > balance) {
          return 0;
        } else {
          balance -= amount;
          return amount;
        }
    }
}
```

The entity can expose **business methods**, such as a method to decrease a bank account balance, to manipulate or access that data. Like a session bean class, an entity class can also declare some standard callback methods or a callback listener class. The persistence provider will call these methods appropriately to manage the entity.

Access to the entity's persistent state is by direct field access. An entity's state can also be accessed using JavaBean-style set and get methods.

The persistence provider can determine which access style is used by looking at how annotations are applied. In the discussed example the @Id annotation is applied to a field (as opposed to annotation applied to a method, so we have field access).

Access to the Entity

```
package examples.entity.intro;
import java.util.List;
import javax.ejb.Stateless;
import javax.ejb.Remote;
import javax.persistence.PersistenceContext;
import javax.persistence.EntityManager;
import javax.persistence.Query;
@Stateless
@Remote(Bank.class)
public class BankBean implements Bank {
    @PersistenceContext
    private EntityManager manager;
    public List<Account> listAccounts() {
        Query query = manager.createQuery ("SELECT a FROM Account a");
        return query.getResultList();
    }
    public Account openAccount(String ownerName) {
        Account account = new Account();
        account.ownerName = ownerName;
        manager.persist(account);
        return account;
    }
```

Access to the Entity

```
    public int getBalance(int accountNumber) {
        Account account = manager.find(Account.class, accountNumber);
        return account.balance;
    }
    public void deposit(int accountNumber, int amount) {
        Account account = manager.find(Account.class, accountNumber);
        account.deposit(amount);
    }
    public int withdraw(int accountNumber, int amount) {
        Account account = manager.find(Account.class, accountNumber);
        return account.withdraw(amount);
    }
    public void close(int accountNumber) {
        Account account = manager.find(Account.class, accountNumber);
        manager.remove(account);
    }
}
```

Persistence.xml

```
<?xml version= 1.0 encoding= UTF-8 ?>
<persistence xmlns= http://java.sun.com/xml/ns/persistence >
    <persistence-unit name= intro />
</persistence>
```

- A persistence unit is defined in a special descriptor file, the persistence.xml file, which is simply added to the META-INF directory of an arbitrary archive, such as an Ejb-jar, .ear, or .war file, or in a plain .jar file.

datasource configuration on Wildfly

http://www.masterthеboss.com/jboss-server/jboss-datasource/how-to-configure-a-datasource-with-jboss-7