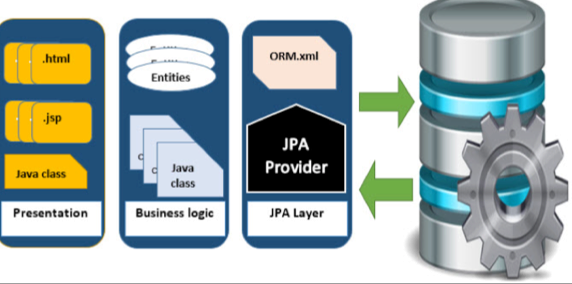
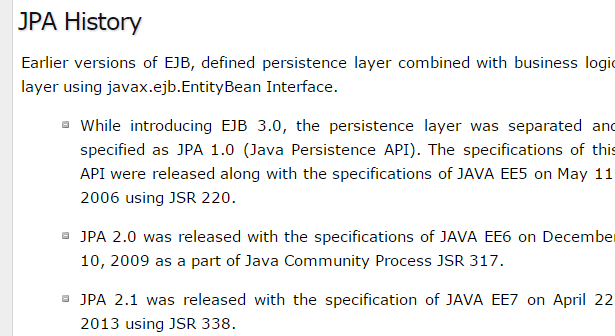
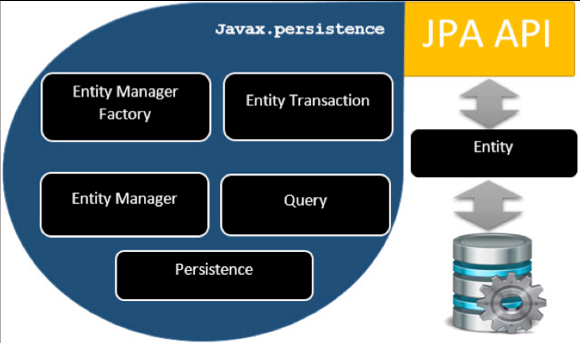
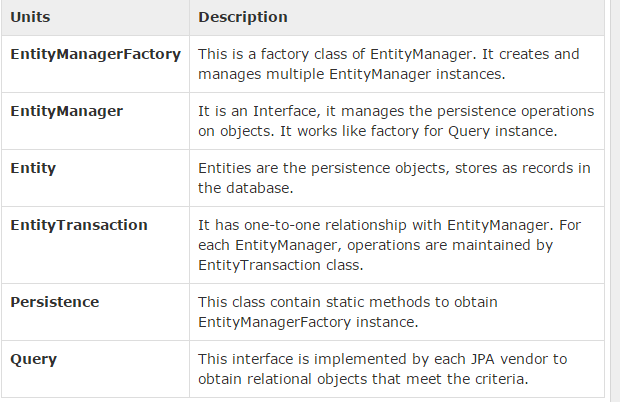
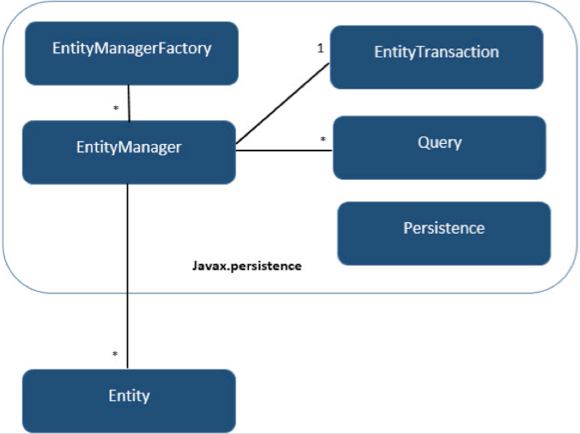
1. Java Persistence API is a collection of classes and methods to persistently store the vast amounts of data into a database, which is provided by the Oracle Corporation.
2. It forms a bridge between object models (Java program) and relational models (database program).
3. 
4. 
5. Class Level Architecture

(



## JPA Class Relationships

1. 

Now let’s see the above relationship in verbal

1. The relationship between EntityManagerFactory and EntityManager is **one-to-many**. It is a factory class to EntityManager instances.
2. The relationship between EntityManager and EntityTransaction is **one-to-one**. For each EntityManager operation, there is an EntityTransaction instance.
3. The relationship between EntityManager and Query is **one-to-many**. Many number of queries can execute using one EntityManager instance.
4. The relationship between EntityManager and Entity is **one-to-many**. One EntityManager instance can manage multiple Entities.
5. It means object database or object relational technologies are taking care of storing, retrieving, updating, and maintenance
6. It means object database or object relational technologies are taking care of storing, retrieving, updating, and maintenance
7. **As xml does not require compilation**, we can easily make changes to multiple data sources with less administration.
8. ORM is a programming ability to covert data from object type to relational type and vice versa.
9. The main feature of ORM is mapping or binding an object to its data in the database. While mapping we have to consider the data, type of data and its relations with its self-entity or entity in any other table.
10. only when the business component commit the data, it is stored into the database physically. Until then the modified data is stored in a cache memory as a grid format. Same is the process for obtaining data.

## ORM Architecture

1. We will see the ORM Architecture later

## Mapping.xml

1. The mapping.xml file is to instruct the JPA vendor for mapping the Entity classes with database tables.
2. Example lets as Consider an Entity class called Employee with the following fields 🡺eid, ename, salary, deg
3. The mapping file named **mapping.xml** is as follows:

Explanation:

1. **<entity-mappings>**🡺 tag defines the schema definition to allow entity tags into xml file.
2. **<entity>** : tag defines the entity class which you want to convert into table in a database. Attribute class defines the POJO entity class name.
3. **<table>** : tag defines the table name. **If you want to keep class name as table name then this tag is not necessary.**

And etc….

## Annotations

1. Is an alternative for Xml based configuration
2. list of annotations which we have not seen VTutorials

|  |  |
| --- | --- |
| **@SequenceGenerator** | This annotation is used to define the value for the property which is specified in @GeneratedValue annotation. It creates a sequence. |

|  |  |
| --- | --- |
| **@TableGenerator** | This annotation is used to specify the value generator for property specified in @GeneratedValue annotation. It creates a table for value generation. |
| **@UniqueConstraint** | This annotation is used to specify the field, unique constraint for primary or secondary table. |
| **@ColumnResult** | This annotation references the name of a column in the SQL query using select clause. |

## Bean Conventions

* Bean contains the default constructor or a file that contains serialized instance. Therefore, a bean can instantiate the bean.
* The properties of a bean can be segregated into Boolean properties and non-Boolean properties.
* Non-Boolean property contains **getter** and **setter** methods.
* **Boolean property contain setter and is method.**

1. **For Boolean property, is method to check if it is true or false. E.g. the Boolean property ‘empty’, the is method of this field is ‘isEmpty ()’**

# JPA - Entity Managers

1. Create Entity class
2. Create required database Schema
3. Persistence.xml

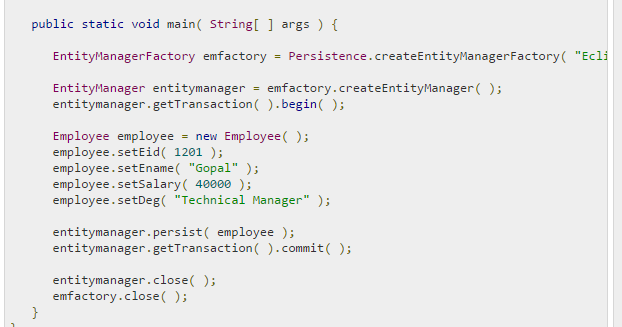


Explanation

* 1. In the above xml, <persistence-unit> tag is defined with specific name for JPA persistence 🡺ms this name is used in getting the EntityManagerFactory instance

1. Example for doing curd/crud operation in JPA

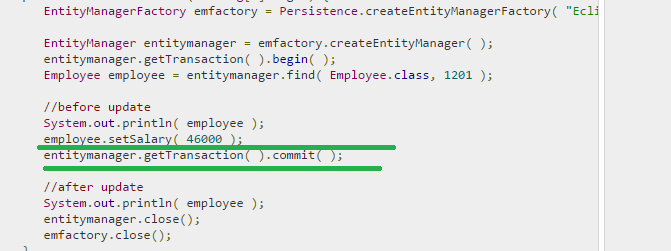
EntityManagerFactory emfactory = Persistence.createEntityManagerFactory( "**Eclipselink\_JPA"** );



Explanation

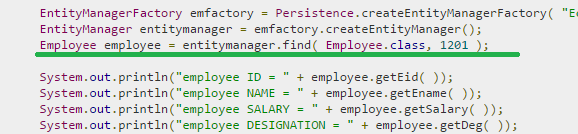
1. In the above code the **createEntityManagerFactory ()** creates a persistence unit by providing the same unique name which we provide for persistence-unit in persistent.xml file.
2. The entitymanagerfactory object will create the entitymanger instance by using **createEntityManager ()** method. The entitymanager object creates entitytransaction instance for transaction management. By using entitymanager object, we can persist entities into database.

## Not let’s see for Update Employee

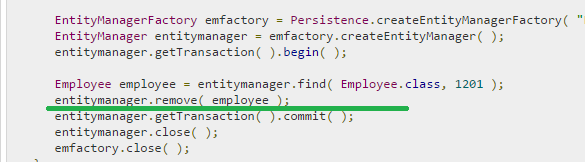
1. 

**Note 🡺 what is the difference between Entitymanager and Session why Hibernate uses Session and JPA uses EntityManager**

## Find Employee

1. 

## Deleting Employee

1. Note JPA uses remove() and hibernate uses delete(). In both Hibernate and JPA first find the Object and then do the delete operation
2. 

# JPA - JPQL

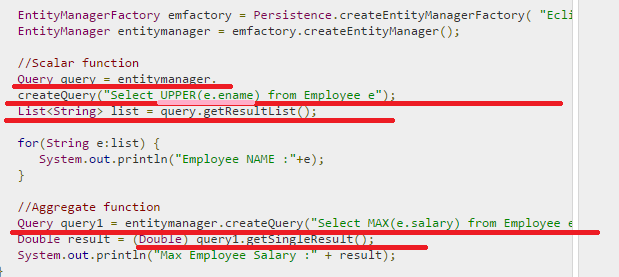
## JPQL 🡺 Java Persistence Query language

1. JPQL is Java Persistence Query Language defined in JPA specification
2. It is used to create queries against entities to store in a relational database. JPQL is developed based on SQL syntax
3. **EntityManager.createQuery()** API will support for querying language.

## Query Structure

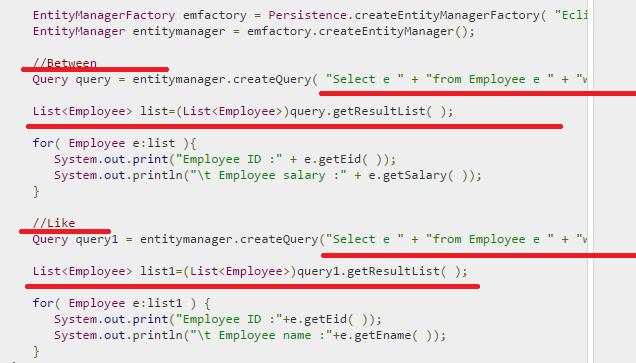
1. JPQL syntax is very similar to the syntax of SQL
2. SQL works directly against relational database tables, records and fields, whereas JPQL works with Java classes and instances.
3. For example,

## Scalar and Aggregate Functions

1. Scalar functions returns resultant values based on input values. Aggregate functions returns the resultant values by calculating the input values. 🡺 I think Scalar function is like String function aggregate function is like grouping function
2. 

query1 = entitymanager.createQuery("Select MAX(e.salary) from Employee e");

## Between, And, Like Keywords

1. **‘Between’, ‘And’, and ‘Like’** are the main keywords of JPQL. **These keywords are used after Where clause in a query**.
2. 

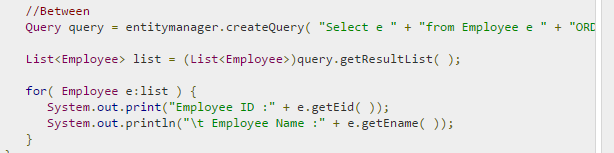
//Between

Query query = entitymanager.createQuery( "Select e " + "from Employee e " + "where e.salary " + "**Between 30000 and 40000"** )

//Like

Query query1 = entitymanager.createQuery("Select e " + "from Employee e " + "where e.ename **LIKE 'M%'**");

## Ordering

1. To Order the records in JPQL we use ORDER BY clause.

Query query = entitymanager.createQuery( "Select e " + "from Employee e " + **"ORDER BY** e.ename ASC" );

## Named Queries

1. Same as Hibernate

## Eager and Lazy Loading

1. See the Hibernate Tutorials

# JPA - Advanced Mappings

## Inheritance Strategies

1. JPA support three types of inheritance strategies such as SINGLE\_TABLE, JOINED\_TABLE, and TABLE\_PER\_CONCRETE\_CLASS. 🡺 and it is annotation based configuration

## JPA - Entity Relationships

1. The relationships between Entity classes are as follows:

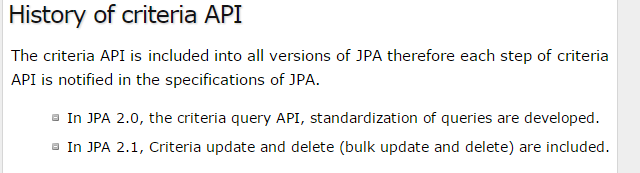
* @ManyToOne Relation
* @OneToMany Relation
* @OneToOne Relation
* @ManyToMany Relation

1. Here its annotation based Examples

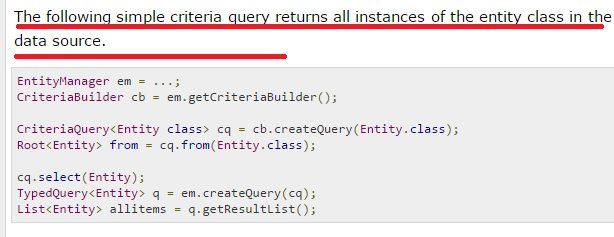
NOTE 🡺 IN HIBERNATE TO USE @ XML FILE NEEDS TO DO SOME MODIFICTION AS BY DEFAULT ITS OFF BUT IN JPA NO SUCH CHANGES I HAD SEEN

1. The Criteria API is a predefined API used to define queries for entities
2. It is the alternative way of defining a JPQL query.
3. These queries are **type-safe**, and portable and easy to modify by changing the syntax
4. **The major advantage of the criteria API is that errors can be detected earlier during compile time**

## History of criteria API

1. 

## Criteria Query Structure

1. It follows javax.persistence.criteria package to design a query.
2. 
3. Explanation for the above code is,

* **EntityManager instance is used to create a CriteriaBuilder object.**
* **CriteriaQuery instance is used to create a query object. This query object’s attributes will be modified with the details of the query.**
* **CriteriaQuery.from method is called to set the query root.**
* **CriteriaQuery.select is called to set the result list type.**
* **TypedQuery<T> instance is used to prepare a query for execution and specifying the type of the query result.**
* **getResultList method on the TypedQuery<T> object to execute a query. This query returns a collection of entities, the result is stored in a List.**

1. The persistence.xml file 🡺 No changes

Note🡺NEEDED TO LEARN MORE ON CRITERIA QUERY