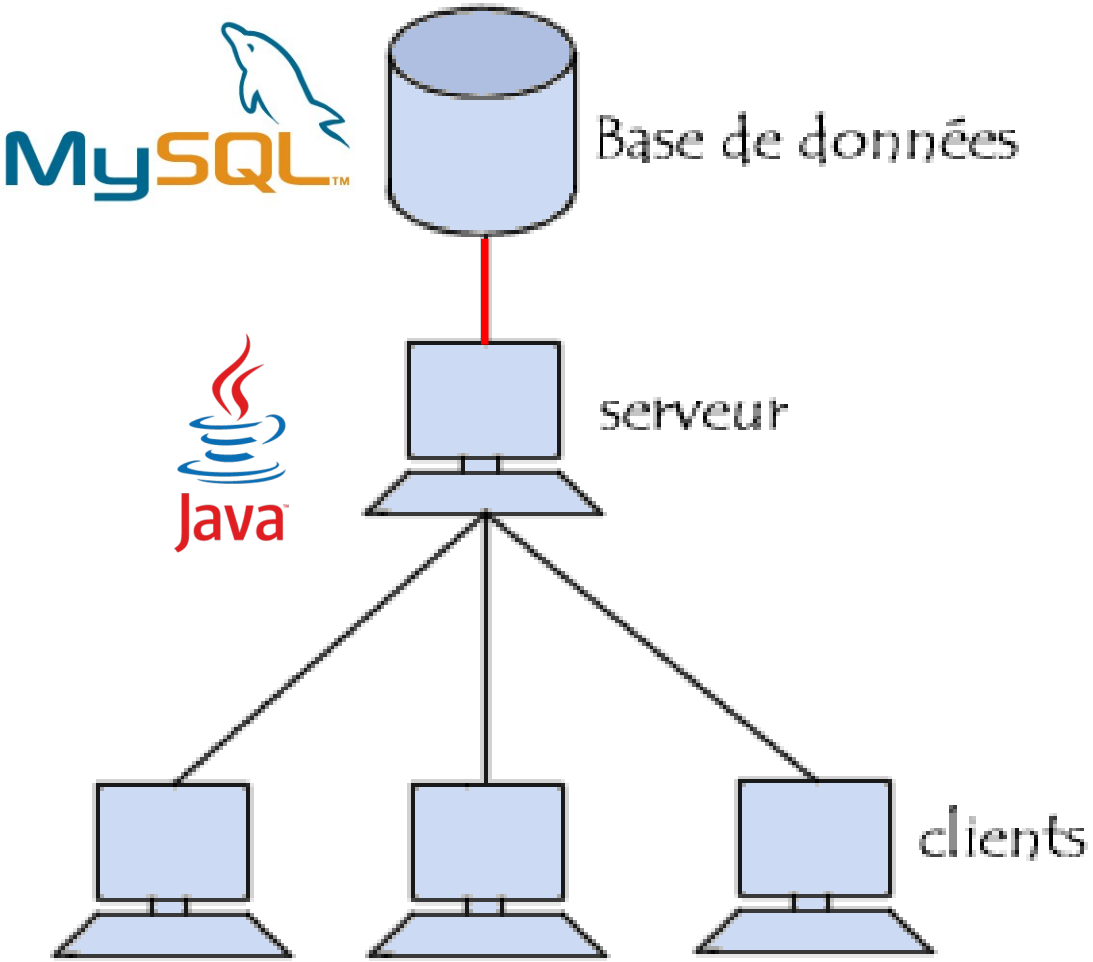
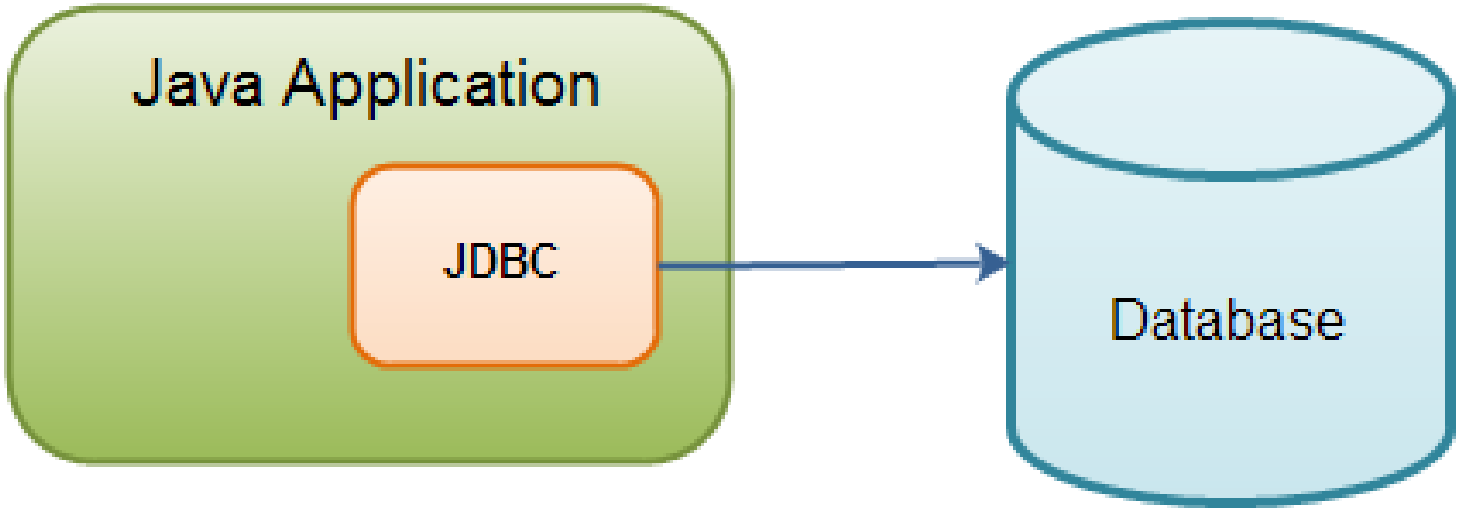
Persistance

Introduction



JDBC - Base



JDBC - Limites

```
select * from Client join Commande join Produit join ...
```

```
Produit produit = new Produit(rs.get(2), rs.get(3), …) ;
Commande commande = new Commande(rs.get(4), rs.get(5), …, produit) ;
Client client = new Client(rs.get(1), rs.get(2), …, commande) ;
```

ORM – Représentation

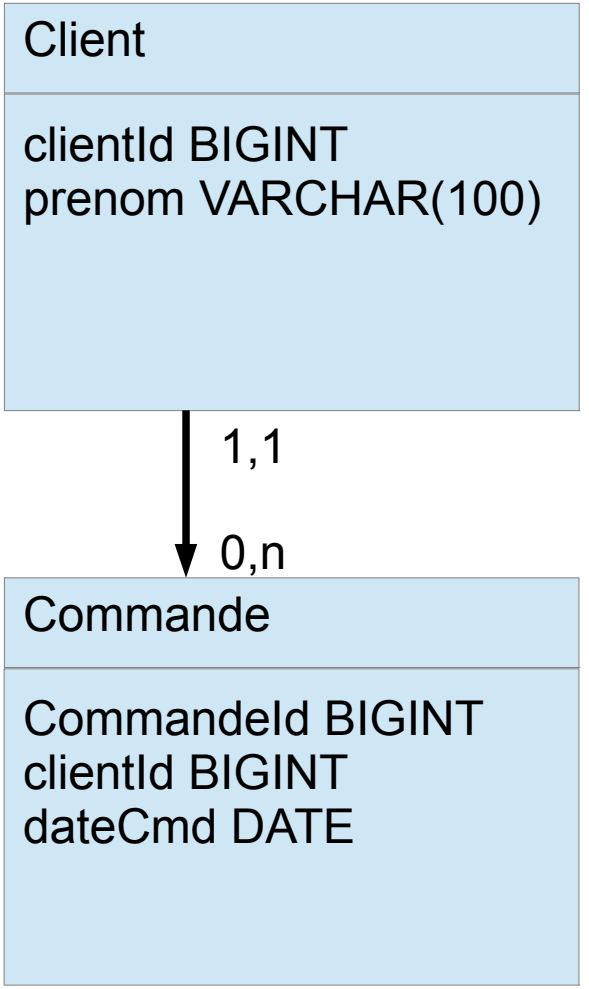
```
class Client{
    private Long clientId ;
    private String prenom ;
    //getters et setters
}
```

Client
clientId BIGINT prenom VARCHAR(100)

ORM – Représentation

```
class Client{
    private Long clientId ;
    private String prenom ;
    private List<Commande> commands ;
    //getters et setters
}
```

```
class Commande{
    private Long commandeId ;
    private Client client;
    private Date dateCmd ;
    //getters et setters
}
```



JPA – XML vs Annotations

XML

orm.xml

```
<entity-mappings>
  <entity class="Client">
  <table name="client"/>
    <attributes>
      <id name="clientId"/>
      <basic name="prenom">
        <column name="prenom"/>
      </basic>
    </attributes>
  </entity>
</entity-mappings>
```

Client.java

```
class Client{
    private Long clientId ;
    private String prenom ;
    //getters et setters
}
```

Annotations

Client.java

```
@Entity
@Table(name = "client")
class Client{

@Id
@Column(name = "client_id")
private Long clientId ;

@Column(name = "prenom")
private String prenom ;

//getters et setters
}
```

JPA - EntityManager

T find(T.class, id)

Boolean find(T)

TypedQuery<T> createQuery(query)

persist(T)

remove(T)

JPA – Implémentations

Que choisir ?

Hibernate – XML vs Annotations

XML

Client.hbm.xml

```
<hibernate-mapping>
    <class name="Client"
                table="client">
        <id name="clientId">
            <column name="client_id" />
        </id>

        <property name="prenom">
            <column name="prenom"/>
        </property>
    </class>
</hibernate-mapping>
```

Client.java

```
class Client{
    private Long clientId ;
    private String prenom ;
    //getters et setters
}
```

Annotations

Client.java

```
@Entity
@Table(name = "client")
class Client{

@Id
@Column(name = "client_id")
private Long clientId ;

@Column(name = "prenom")
private String prenom ;

//getters et setters
}
```

Hibernate – Opération sur les entités

Create :

```
Client c = new Client() ;
c.setPrenom("Test") ;
session.save(c) ;
```

Update :

```
Client c = session.get(Client.class, id) ;
c.setPrenom("TestModifié") ;
session.update(c) ;
```

Delete :

```
Client c = Session.get(Client.class, id) ;
Session.delete(c) ;
```

Hibernate – Opération génériques

```
public interface GeneralDAO { // from generic-dao

    public T find(ID id);

    public T[] find(ID... ids);

    public boolean save(T entity);

    public boolean[] save(T... entities);

    public boolean remove(T entity);

    public void remove(T... entities);

    public boolean removeById(ID id);

    public void removeByIds(ID... ids);

    public List<T> findAll();

    ...

}
```

Conclusion

Projets tutoriel :

- https://github.com/jeanbaptistedalle/jdbc

- https://github.com/jeanbaptistedalle/hibernate

Sources :

- Cours / exemples : http://www.jmdoudoux.fr/java/dej/chap-hibernate.htm

- Documentation :http://docs.jboss.org/hibernate/orm/4.3/manual/en-US/html/

- Exemples : http://www.tutorialspoint.com/hibernate/

- Exemples : http://www.mkyong.com/hibernate/

- Debugage : http://stackoverflow.com/

Annexe 1 : Hibernate – HQL vs Criteria

HQL

Initialisation :

```
session.createQuery(query)
```

List :

```
select c from Client c ;
```

Condition :

```
select c from Client c where
c.prenom=:prenom ;

query.set("prenom","test") ;
```

Condition avec jointure :

```
select c from Client c join
c.commandes as com where
com.dateCmd = :dateCmd ;

query.set("dateCmd",new
Date()) ;
```

Criteria

Initialisation :

```
session.createCriteria(Client.class)
```

List :

```
criteria.list() ;
```

Condition :

```
criteria.add(Restrictions.eq("prenom
", "test" )).list() ;
```

Condition avec jointure :

```
criteria.createAlias("c.commandes",
"com") ;

criteria.add(Restriction.eq(
"com.dateCmd", new Date()) ;

criteria.list() ;
```

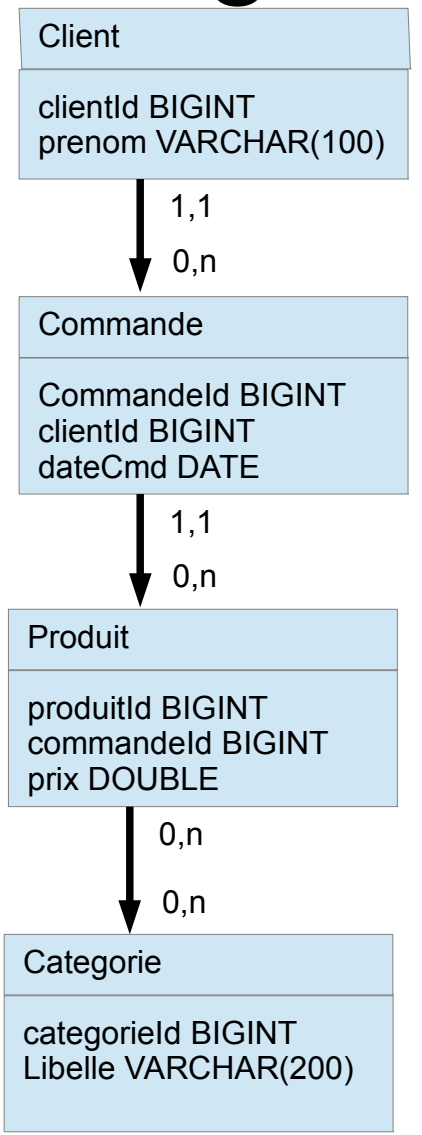
Annexe 2 : Hibernate – Lazy vs Eager

```
class Client{
    private Long clientId ;
    private String prenom ;
    private List<Commande> commands ;
}

class Commande{
    private Long commandeId ;
    private Date dateCmd ;
    private Client client;
    Private List<Produit> ;
}

class Produit{
    private Long produitId ;
    private Double prix ;
    private List<Categorie> categories ;
    private List<Commande> commandes;
}

class Categorie{
    private Long categorieId ;
    private String libelle;
    private List<Produit> produits;
}
```



Annexe 2 : Hibernate – Lazy vs Eager

Lazy mode :

On exécute une requête permettant d'obtenir le client1.

Si on désire obtenir la liste des commandes, une nouvelle requête est effectuée.

Si on désire ensuite obtenir la liste des produits d'une commande, une nouvelle requête est effectuée, etc.

On ne charge donc les données que si on les utilise.

Eager mode :

On exécute une requête permettant d'obtenir le client 1.

Lorsque le client 1 est chargé, on charge en même temps toutes les commandes qui lui sont liées, puis tous les produits des commandes, puis toutes les catégories des commandes, etc.

On effectue donc qu'une seule requête, mais très vite, on récupérera toute la base de données en une seule requête !