**(Java Persistence Api):**

La couche JPA permet de faire le pont entre le monde relationnel de la base de données et le monde objet manipulé par les programmes Java. Ce pont est fait par configuration et il y a deux façons de le faire :

* Avec des fichiers XML. C'était quasiment l'unique façon de faire jusqu'à l'avènement du JDK 1.5
* Avec des annotations Java depuis le JDK 1.5

**Les Annotations**

* **@Entity** : est la première annotation indispensable. Elle se place avant la ligne qui déclare la classe et indique que la classe en question doit être gérée par la couche de persistance JPA. En l'absence de cette annotation, toutes les autres annotations JPA seraient ignorées.
* **@Table :** désigne la table de la base de données dont la classe est une représentation.

Son principal argument est name qui désigne le nom de la table.

En l'absence de cet argument, la table portera le nom de la classe, ici [Personne]. Dans notre exemple, l'annotation @Table est donc superflue.

* **@Id :** sert à désigner le champ dans la classe qui est image de la clé primaire de la table. Cette annotation est obligatoire. Elle indique ici que le champ id de la ligne 11 est l'image de la clé primaire de la table.

* **@Column :**  sert à faire le lien entre un champ de la classe et la colonne de la table dont le champ est l'image. L'attribut name indique le nom de la colonne dans la table. En l'absence de cet attribut, la colonne porte le même nom que le champ.

C’est pour activer les proprièté des attributs d’une table

* **@GeneratedValue** : indique comment est générée la clé primaire lorsqu'elle est générée automatiquement par le SGBD. Cette annotation indique que la clé primaire est générée de façon automatique lors de l’insertion en base. Sans cette annotation, la valeur de l’identifiant de la clé primaire doit être affectée avant l’insertion en base.

Strategy : Permet de définir le mode de génération de la clé primaire

* Strategy = GenerationType.AUTO : La génération de la clé primaire est laissée à l’implémentation. C’est Hibernate qui s’en charge et qui crée une séquence unique sur tout le schéma via la table Hibernate séquence.
* Strategy = GenerationType. IDENTITY : La génération de la clé primaire se fera à partir d’une identité propre au SGBD. Il utilise un type de colonne spéciale à la base de données. Exemple pour MySQL, il s’agit d’un AUTO\_INCREMENT.

* Strategy = GenerationType. TABLE : La génération de la clé primaire se fera en utilisant une table dédiée Hibernate séquence qui stocke les noms et les valeurs des séquences.
* **Strategy = GenerationType. SEQUENCE**:

La clé primaire est générer de manière AUTOMATIQUE

La génération de la clé primaire se fera par une séquence définie dans le SGBD, auquel on ajoutera l’attribut generator. Cette stratégie doit être utilisée avec une autre annotation qui est @SequenceGenerator. Cette annotation possède l’attribut name pour le nom du generator, l’attribut sequenceName pour le nom de la séquence et enfin allocationSize qui est l’incrémentation de la valeur de la séquence.

Exemple: {{< highlight java "linenos=table" >}} @GeneratedValue (strategy = GenerationType.SEQUENCE, generator = "enerator\_client")

@SequenceGenerator (name = "generator\_client", sequenceName = "WINDEV\_SEQ", allocationSize = 1) {{< / highlight >}} Nous verrons ultérieurement comment utiliser d’autres annotations de mapping.

**Criteria Queries**

**CriteriaBuilder :** est le point de départ pour créer un CriteriaQuery. Il est obtenu à partir du EntityManager.

**CriteriaQuery :** est utilisé pour créer une requête qui renvoie un ensemble de résultats. Il représente une requête JPQL sous sa forme abstraite.

**Root:** est utilisé pour définir l'entité que vous interrogez. Il représente la racine de la requête, généralement une entité.

**Predicate : représente les conditions à inclure dans la clause WHERE d'une requête. Vous pouvez utiliser diverses méthodes de CriteriaBuilder pour créer des prédicats (par exemple, égal, différent, similaire, etc.).**