2.2.1.1. Définir la table

`@Table` est positionnée au niveau de la classe ; cela vous permet de définir le nom de la table, du catalogue et du schéma pour le mapping de votre entity bean. Si aucune `@Table` n'est définie les valeurs par défaut sont utilisées : le nom de la classe de l'entité (sans le nom de package).

```
@Entity
@Table(name="tbl_sky")
public class Sky implements Serializable {
...
```

L'élément `@Table` contient aussi un attribut `schema` et un attribut `catalog`, si vous avez besoin de les définir. Vous pouvez aussi définir des contraintes d'unicité sur la table en utilisant l'annotation `@UniqueConstraint` en conjonction avec `@Table` (pour une contrainte d'unicité n'impliquant qu'une seule colonne, référez-vous à `@Column`).

```
@Table(name="tbl_sky",
    uniqueConstraints = {@UniqueConstraint(columnNames={"month", "day"})}
)
```

Une contrainte d'unicité est appliquée au tuple {month, day}. Notez que le tableau `columnNames` fait référence aux noms logiques des colonnes.
Le nom logique d'une colonne est défini par l'implémentation de NamingStrategy d'Hibernate. La stratégie de nommage EJB3 par défaut utilise le nom de colonne physique comme nom de colonne logique. Notez qu'il peut être différent du nom de la propriété (si le nom de colonne est explicite). A moins que vous surchargiez la stratégie de nommage, vous ne devriez pas vous soucier de ça.

2.2.1.2. Versionner pour un contrôle de concurrence optimiste

Vous pouvez ajouter un contrôle de concurrence optimiste à un entity bean en utilisant l'annotation `@Version` :

```
@Entity
public class Flight implements Serializable {
...
    @Version
    @Column(name="OPTLOCK")
    public Integer getVersion() { ... }
}
```

La propriété de version sera mappée sur la colonne `OPTLOCK`, et le gestionnaire d'entités l'utilisera pour détecter des conflits lors des mises à jour (prévenant des pertes de données lors de mises à jours que vous pourriez voir avec la stratégie du last-commit-wins).

La colonne de version peut être un numérique (solution recommandée) ou un timestamp comme pour la spécification EJB3. Hibernate prend en charge n'importe quel type fourni que vous définissez et implémentez avec la classe `UserVersionType` appropriée.

2.2.2. Mapping de simples propriétés

2.2.2.1. Déclarer des mappings de propriétés élémentaires

Chaque propriété (champ ou méthode) non statique non transient d'un entity bean est considérée persistante, à

moins que vous l'annotiez comme `@Transient`. Ne pas avoir d'annotation pour votre propriété est équivalent à l'annotation `@Basic`. L'annotation `@Basic` vous permet de déclarer la stratégie de récupération pour une propriété :

```
public transient int counter; // propriété transient

private String firstname; // propriété persistante

@Transient
String getLengthInMeter() { ... } // propriété transient

String getName() {... } // propriété persistante

@Basic
int getLength() { ... } // propriété persistante

@Basic(fetch = FetchType.LAZY)
String getDetailedComment() { ... } // propriété persistante

@Temporal(TemporalType.TIME)
java.util.Date getDepartureTime() { ... } // propriété persistante

@Enumerated(EnumType.STRING)
Starred getNote() { ... } // enum persistée en tant que String dans la base de données
```

`counter`, un champ transient, et `lengthInMeter`, une méthode annotée comme `@Transient`, seront ignorés par le gestionnaire d'entités. Les propriétés `name`, `length`, et `firstname` sont mappées comme persistantes et à charger immédiatement (ce sont les valeurs par défaut pour les propriétés simples). La valeur de la propriété `detailedComment` sera chargée à partir de la base de données dès que la propriété de l'entité sera accédée pour la première fois. En général vous n'avez pas besoin de marquer de simples propriétés comme "à charger à la demande" (NdT: lazy) (à ne pas confondre avec la récupération d'association "lazy").

Note

Pour activer la récupération à la demande au niveau de la propriété, vos classes doivent être instrumentées : du bytecode est ajouté au code original pour activer cette fonctionnalité, veuillez vous référer à la documentation de référence d'Hibernate. Si vos classes ne sont pas instrumentées, le chargement à la demande au niveau de la propriété est silencieusement ignoré.

L'alternative recommandée est d'utiliser la capacité de projection de JPA-QL ou des requêtes Criteria.

EJB3 prend en charge le mapping de propriété de tous les types élémentaires pris en charge par Hibernate (tous les types de base Java, leur wrapper respectif et les classes sérialisables). Hibernate Annotations prend en charge le mapping des types Enum soit vers une colonne ordinale (en stockant le numéro ordinal de l'enum), soit vers une colonne de type chaîne de caractères (en stockant la chaîne de caractères représentant l'enum) : la représentation de la persistance, par défaut ordinale, peut être surchargée grâce à l'annotation `@Enumerated` comme montré avec la propriété `note` de l'exemple.

Dans les APIs core de Java, la précision temporelle n'est pas définie. Lors du traitement de données temporelles vous pourriez vouloir décrire la précision attendue dans la base de données. Les données temporelles peuvent avoir une précision de type `DATE`, `TIME`, ou `TIMESTAMP` (c'est-à-dire seulement la date, seulement l'heure, ou les deux). Utilisez l'annotation `@Temporal` pour ajuster cela.

`@Lob` indique que la propriété devrait être persistée dans un Blob ou un Clob selon son type : `java.sql.Clob`, `Character[]`, `char[]` et java.lang.`String` seront persistés dans un Clob. `java.sql.Blob`, `Byte[]`, `byte[]` et les types sérialisables seront persistés dans un Blob.

```
@Lob
```

```
public String getFullText() {
    return fullText;
}

@Lob
public byte[] getFullCode() {
    return fullCode;
}
```

Si le type de la propriété implémente `java.io.Serializable` et n'est pas un type de base, et si la propriété n'est pas annotée avec `@Lob`, alors le type Hibernate `serializable` est utilisé.

2.2.2.2. Déclarer des attributs de colonne

La(les) colonne(s) utilisée(s) pour mapper une propriété peuvent être définies en utilisant l'annotation `@Column`. Utilisez-la pour surcharger les valeurs par défaut (voir la spécification EJB3 pour plus d'informations sur les valeurs par défaut). Vous pouvez utilisez cette annotation au niveau de la propriété pour celles qui sont :

- pas du tout annotées
- annotées avec `@Basic`
- annotées avec `@Version`
- annotées avec `@Lob`
- annotées avec `@Temporal`
- annotées avec `@org.hibernate.annotations.CollectionOfElements` (pour Hibernate uniquement)

```
@Entity
public class Flight implements Serializable {
...
@Column(updatable = false, name = "flight_name", nullable = false, length=50)
public String getName() { ... }
```

La propriété `name` est mappée sur la colonne `flight_name`, laquelle ne peut pas avoir de valeur nulle, a une longueur de 50 et ne peut pas être mise à jour (rendant la propriété immuable).

Cette annotation peut être appliquée aux propriétés habituelles ainsi qu'aux propriétés `@Id` ou `@Version`.

```
@Column(
    name="columnName";                                  (1)
    boolean unique() default false;                     (2)
    boolean nullable() default true;                    (3)
    boolean insertable() default true;                  (4)
    boolean updatable() default true;                   (5)
    String columnDefinition() default "";               (6)
    String table() default "";                          (7)
    int length() default 255;                           (8)
    int precision() default 0; // decimal precision     (9)
    int scale() default 0; // decimal scale
```

(1) `name` (optionnel) : le nom de la colonne (par défaut le nom de la propriété)
(2) `unique` (optionnel) : indique si la colonne fait partie d'une contrainte d'unicité ou non (par défaut false)
(3) `nullable` (optionnel) : indique si la colonne peut avoir une valeur nulle (par défaut false).