HIBERNATE CHEAT SHEET

@DynamicInsert / @DynamicUpdate

Utilizamos estas anotaciones sobre una entidad cuando queremos que no se envíen en la query los valores nulos (@DynamicInsert) o los valores que no se han modificado desde la última persistencia (@DynamicUpdate)

```
@DynamicInsert
@DynamicUpdate
public class CustomerEntity {
```

@Where(clause = “condición SQL”)

Utilizamos esta anotación para realizar una selección de las entidades que se recuperan. Si la anotación se hace sobre la entidad se aplica sobre todos los objetos, si se hace sobre una relación (@OneToMany) afecta únicamente a la misma.

```
@Entity
@Table(name = "rental", schema = "public", catalog = "dvd2324")
@Where(clause = "return_date is null")
public class RentalEntity {
```

Ó

```
@OneToMany(mappedBy = "stock", fetch = FetchType.EAGER)
@Where(clause = "return_date is null")
public List<RentalEntity> getRentals() {
     return rentals;
}
```

@Transient

Mediante esta anotación podemos añadir funcionalidad a nuestras entidades (POJOS) que no será tenida en cuenta para la persistencia.

```
@Transient
public boolean canLend() {
 return getRentals().size() <
           BusinessData.MAX_RENTALS_ALLOWED;
}
```

fetch = FetchType.EAGER

@LazyCollection(LazyCollectionOption.FALSE)

Estos dos modificadores nos permiten indicar en qué momento se recuperan los datos de una subquery, si en el momento en que se soliciten (LAZY) o inmediatamente que se hace la query original (EAGER). Si no se especifica el tipo de fetch, es LAZY.

```
@OneToMany(mappedBy = "stock", fetch = FetchType.EAGER)
```

Ó

```
@OneToMany(mappedBy = "book")
@LazyCollection(LazyCollectionOption.FALSE)
public List<ReservationEntity> getReservedBy() {
```

@Filter y @FilterDef

Similar a la anotación @Where, pero permite activar y desactivar el filtro dentro de la sesión, además de incluir parámetros.

```
@FilterDef(name = "stockStoreFilter", parameters =
       @ParamDef(name = "storeNumber", type = "integer"))
@Filter(name = "stockStoreFilter",
        condition = "store_id = :storeNumber")
@Entity
@Table(name = "stock", schema = "public", catalog = "dvd2324")
public class StockEntity {
```

Queries JPA predefinidas

En Spring, podemos usar las queries predefinidas JPA para ampliar la funcionalidad de nuestros DAOs. Por ejemplo, si necesitáramos buscar a un empleado por departamento, y a lose podría ampliar la funcionalidad incorporando los siguientes métodos al DAO:

```
public interface IEmployeeEntityDAO extends
CrudRepository<EmployeeEntity, Integer> {
     findByDeptno(int deptno);
}
```

También podemos añadir operadores SQL habituales en el nombre:

```
findByDeptnoGreaterThan(int deptno);
```