What is a *FetchType*

The *FetchType* defines when Hibernate gets the related entities from the database, and it is one of the crucial elements for a fast persistence tier.

In general, you want to fetch the entities you use in your business tier as efficiently as possible. But that's not that easy. You either get all relationships with one query or you fetch only the root entity and initialize the relationships as soon as you need them.

Default *FetchTypes*

The default depends on the cardinality of the relationship. All to-one relationships use *FetchType.EAGER* and all to-many relationships *FetchType.LAZY*.

How to change a *FetchType*

You can change the default *FetchType* by providing your preferred *FetchType* to the relationship annotation as you can see in the following code snippet.

```
@Entity
@Table(name = "purchaseOrder")
public class Order implements Serializable {

  @OneToMany(mappedBy = "order", fetch = FetchType.EAGER)
  private Set<OrderItem> items = new HashSet<OrderItem>();

  ...

}
```

FetchType.EAGER - Fetch it so you'll have it when you need it

The *FetchType.EAGER* tells Hibernate to get all elements of a relationship when selecting the root entity.

```
OrderItem orderItem = em.find(OrderItem.class, 1L);

log.info("Fetched OrderItem: "+orderItem);

Assert.assertNotNull(orderItem.getProduct());
```

```
05:41:38,726 DEBUG SQL:92 - select orderitem0_.id as id1_0_0_,
orderitem0_.order_id as order_id4_0_0_, orderitem0_.product_id as
product_5_0_0_, orderitem0_.quantity as quantity2_0_0_,
orderitem0_.version as version3_0_0_, order1_.id as id1_2_1_,
order1_.orderNumber as orderNum2_2_1_, order1_.version as
version3_2_1_, product2_.id as id1_1_2_, product2_.name as
name2_1_2_, product2_.price as price3_1_2_, product2_.version as
version4_1_2_ from OrderItem orderitem0_ left outer join
purchaseOrder order1_ on orderitem0_.order_id=order1_.id left outer
join Product product2_ on orderitem0_.product_id=product2_.id where
orderitem0_.id=?

05:41:38,764  INFO FetchTypes:77 - Fetched OrderItem: OrderItem ,
quantity: 100
```

This seems to be very useful in the beginning. Joining the required entities and getting all of them in one query is very efficient.

But keep in mind, that Hibernate will ALWAYS fetch the Product entity for your OrderItem, even if you don't use it in your business code. If the related entity isn't too big, this is not an issue for to-one relationships. But it will most likely slow down your application if you use it for a to-many relationship that you don't need for your use case. Hibernate then has to fetch tens or even hundreds of additional entities which creates a significant overhead.

FetchType.LAZY - Fetch it when you need it

The *FetchType.LAZY* tells Hibernate to only fetch the related entities from the database when you use the relationship. This is a good idea in general because there's no reason to select entities you don't need for your uses case.

The used *FetchType* has no influence on the business code. You can call the getter method just as any other getter method.

```
Order newOrder = em.find(Order.class, 1L);

log.info("Fetched Order: "+newOrder);

Assert.assertEquals(2, newOrder.getItems().size());
```

Hibernate handles the lazy initialization transparently and fetches the *OrderItem* entities as soon as the getter method gets called.

```
05:03:01,504 DEBUG SQL:92 - select order0_.id as id1_2_0_,
order0_.orderNumber as orderNum2_2_0_, order0_.version as
version3_2_0_ from purchaseOrder order0_ where order0_.id=?

05:03:01,545  INFO FetchTypes:45 - Fetched Order: Order
orderNumber: order1

05:03:01,549 DEBUG SQL:92 - select items0_.order_id as
order_id4_0_0_, items0_.id as id1_0_0_, items0_.id as id1_0_1_,
items0_.order_id as order_id4_0_1_, items0_.product_id as
product_5_0_1_, items0_.quantity as quantity2_0_1_, items0_.version
as version3_0_1_, product1_.id as id1_1_2_, product1_.name as
name2_1_2_, product1_.price as price3_1_2_, product1_.version as
version4_1_2_ from OrderItem items0_ left outer join Product
product1_ on items0_.product_id=product1_.id where
items0_.order_id=?
```

This becomes a performance problem when you use it on a large list of entities. Hibernate then has to perform an additional SQL statement for each *Order* entity to fetch its *OrderItems*.