

Hibernate has few fetching strategies to optimize the Hibernate generated select statement, so that it can be as efficient as possible. The fetching strategy is declared in the mapping relationship to define how Hibernate fetch its related collections and entities.

## Fetching Strategies

There are four fetching strategies

1. fetch="join" = Disable the lazy loading, always load all the collections and entities.
2. fetch="select" (default) = Lazy load all the collections and entities.
3. batch-size="N" = Fetching up to 'N' collections or entities, \*Not record\*.
4. fetch="subselect" = Group its collection into a sub select statement.

For detail explanation, you can check on the [Hibernate documentation](#).

## Fetching strategies examples

Here's a "one-to-many relationship" example for the fetching strategies demonstration. A stock is belong to many stock daily records.

```
...
@Entity
@Table(name = "stock", catalog = "mkyong")
public class Stock implements Serializable{
    ...
    @OneToMany(fetch = FetchType.LAZY, mappedBy = "stock")
    @Cascade(CascadeType.ALL)
    @Fetch(FetchMode.SELECT)
    @BatchSize(size = 10)
    public Set getStockDailyRecords() {
        return this.stockDailyRecords;
    }
    ...
}
```

## 1. fetch="select" or @Fetch(FetchMode.SELECT)

This is the default fetching strategy. it enabled the lazy loading of all it's related collections. Let see the example...

```
//call select from stock
Stock stock = (Stock)session.get(Stock.class, 114);
Set sets = stock.getStockDailyRecords();

//call select from stock_daily_record
for ( Iterator iter = sets.iterator();iter.hasNext(); ) {
    StockDailyRecord sdr = (StockDailyRecord) iter.next();
    System.out.println(sdr.getDailyRecordId());
    System.out.println(sdr.getDate());
}
```

### Output

```
Hibernate:
select ...from mkyong.stock
where stock0_.STOCK_ID=?

Hibernate:
select ...from mkyong.stock_daily_record
where stockdaily0_.STOCK_ID=?
```

Hibernate generated two select statements

1. Select statement to retrieve the Stock records –**session.get(Stock.class, 114)**
2. Select its related collections – **sets.iterator()**

## 2. fetch="join" or @Fetch(FetchMode.JOIN)

The "join" fetching strategy will disabled the lazy loading of all it's related collections. Let see the example...

```
//call select from stock and stock_daily_record
Stock stock = (Stock)session.get(Stock.class, 114);
Set sets = stock.getStockDailyRecords();

//no extra select
for ( Iterator iter = sets.iterator();iter.hasNext(); ) {
    StockDailyRecord sdr = (StockDailyRecord) iter.next();
    System.out.println(sdr.getDailyRecordId());
    System.out.println(sdr.getDate());
}
```

### Output

```
Hibernate:
select ...
from
    mkyong.stock stock0_
left outer join
    mkyong.stock_daily_record stockdaily1_
        on stock0_.STOCK_ID=stockdaily1_.STOCK_ID
where
    stock0_.STOCK_ID=?
```

Hibernate generated only one select statement, it retrieve all its related collections when the Stock is initialized.

–**session.get(Stock.class, 114)**

1. Select statement to retrieve the Stock records and outer join its related collections.

### 3. batch-size="10" or @BatchSize(size = 10)

The batch-size fetching strategy is not define how many records inside in the collections are loaded. Instead, it defines how many collections should be loaded.

### 4. fetch="subselect" or @Fetch(FetchMode.SUBSELECT)

This fetching strategy is enable all its related collection in a sub select statement. Let see the same query again..

```
List list = session.createQuery("from Stock").list();

for(Stock stock : list){

    Set sets = stock.getStockDailyRecords();

    for ( Iterator iter = sets.iterator();iter.hasNext(); ) {
        StockDailyRecord sdr = (StockDailyRecord) iter.next();
        System.out.println(sdr.getDailyRecordId());
        System.out.println(sdr.getDate());
    }
}
```

#### Output

```
Hibernate:
select ...
from mkyong.stock stock0_

Hibernate:
select ...
from
mkyong.stock_daily_record stockdaily0_
where
stockdaily0_.STOCK_ID in (
select
stock0_.STOCK_ID
from
mkyong.stock stock0_
)
```

With "subselect" enabled, it will create two select statements.

1. Select statement to retrieve all the Stock records.
2. Select all its related collections in a sub select query.