inverse = “true” example and explanation

By mkyong | January 31, 2010 | Updated : April 26, 2011 | Viewed : 249,530 times

Always put inverse=”true” in your collection variable ?
There are many Hibernate articles try to explain the “inverse” with many Hibernate “official” jargon, which is very hard to understand (at least to me). In few articles, they even suggested that just forget about what is “inverse”, and always put **inverse=”true”** in the collection variable.

This statement is always true – “put inverse=true in collection variable”, but do not blindfold on it, try to understand the reason behind is essential to optimal your Hibernate performance.

What is “inverse” ?

This is the most confusing keyword in Hibernate, at least i took quite a long time to understand it. The “**inverse**” keyword is always declare in **one-to-many** and **many-to-many** relationship (many-to-one doesn’t has inverse keyword), it means which side is responsible to take care of the relationship.

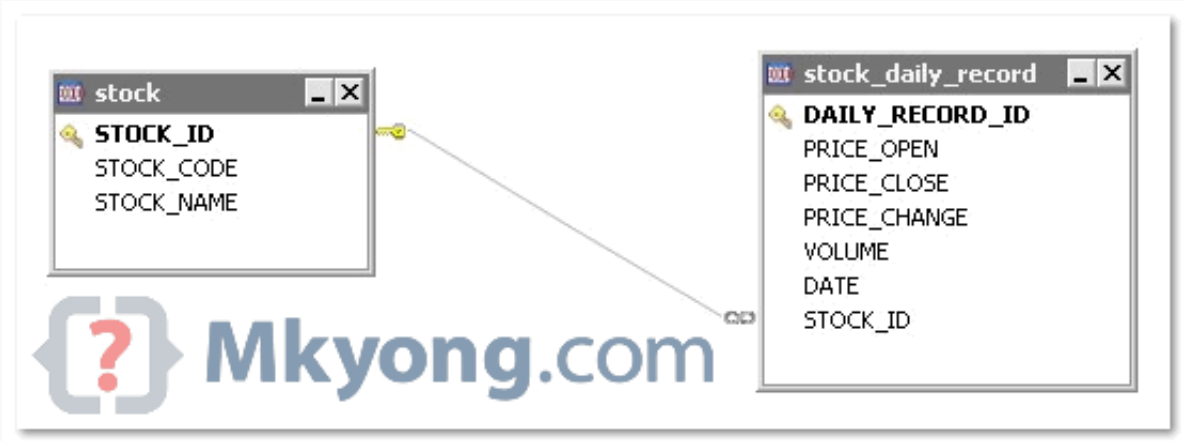
“inverse”, should change to “relationship owner”?

In Hibernate, only the “relationship owner” should maintain the relationship, and the “inverse” keyword is created to defines which side is the owner to maintain the relationship. However the “inverse” keyword itself is not verbose enough, I would suggest change the keyword to “**relationship_owner**“.

In short, inverse=”true” means this is the relationship owner, and inverse=”false” (default) means it’s not.

1. One to many Relationship

This is a **one-to-many** relationship table design, a STOCK table has many occurrences in STOCK_DAILY_RECORD table.



2. Hibernate Implementation

See the Hibernate implementation in XML mapping files.

File : Stock.java

Java

```
public class Stock implements java.io.Serializable {
   ...
   private Set<StockDailyRecord> stockDailyRecords =
                        new HashSet<StockDailyRecord>(0);
   ...
```

File : StockDailyRecord.java

Java

```
public class StockDailyRecord implements java.io.Serializable {
   ...
   private Stock stock;
   ...
```

File : Stock.hbm.xml

Markup

```
<hibernate-mapping>
    <class name="com.mkyong.common.Stock" table="stock" ...>
    ...
    <set name="stockDailyRecords" table="stock_daily_record" fetch="select">
        <key>
            <column name="STOCK_ID" not-null="true" />
        </key>
        <one-to-many class="com.mkyong.common.StockDailyRecord" />
    </set>
    ...
```

File : StockDailyRecord.hbm.xml

Markup

```
<hibernate-mapping>
  <class name="com.mkyong.common.StockDailyRecord" table="stock_daily_record" ...>
  ...
  <many-to-one name="stock" class="com.mkyong.common.Stock">
       <column name="STOCK_ID" not-null="true" />
  </many-to-one>
  ...
```

3. inverse = true / false

Inverse keyword is applied in one to many relationship. Here’s the question, if save or update operation perform in “Stock” object, should it update the “stockDailyRecords” relationship?

File : Stock.hbm.xml

Markup

```
    <class name="com.mkyong.common.Stock" table="stock" ...>
    ...
    <set name="stockDailyRecords" table="stock_daily_record" inverse="{true/false}" fetch="select">
        <key>
            <column name="STOCK_ID" not-null="true" />
        </key>
        <one-to-many class="com.mkyong.common.StockDailyRecord" />
    </set>
    ...
```

1. inverse=”true”

If inverse=”true” in the set variable, it means “stock_daily_record” is the relationship owner, so Stock will NOT UPDATE the relationship.

Markup

```
<class name="com.mkyong.common.Stock" table="stock" ...>
    ...
    <set name="stockDailyRecords" table="stock_daily_record" inverse="true" >
```

2. inverse=”false”

If inverse=”false” (default) in the set variable, it means “stock” is the relationship owner, and Stock will UPDATE the relationship.

Markup

```
<class name="com.mkyong.common.Stock" table="stock" ...>
    ...
    <set name="stockDailyRecords" table="stock_daily_record" inverse="false" >
```

See more examples below :

4. inverse=”false” Example

If keyword “inverse” is not define, the inverse = “false” will be used, which is

Markup

```
<!--Stock.hbm.xml-->
<class name="com.mkyong.common.Stock" table="stock" ...>
    ...
    <set name="stockDailyRecords" table="stock_daily_record" inverse="false">
```

It means “stock” is the relationship owner, and it will maintains the relationship.

Insert example …

When a “Stock” object is saved, Hibernate will generated three SQL statements, two inserts and one update.

Java

```
    session.beginTransaction();

    Stock stock = new Stock();
    stock.setStockCode("7052");
    stock.setStockName("PADINI");

    StockDailyRecord stockDailyRecords = new StockDailyRecord();
    stockDailyRecords.setPriceOpen(new Float("1.2"));
    stockDailyRecords.setPriceClose(new Float("1.1"));
    stockDailyRecords.setPriceChange(new Float("10.0"));
    stockDailyRecords.setVolume(3000000L);
    stockDailyRecords.setDate(new Date());

    stockDailyRecords.setStock(stock);
    stock.getStockDailyRecords().add(stockDailyRecords);

    session.save(stock);
    session.save(stockDailyRecords);
```

```
    session.getTransaction().commit();
```

Output…

SQL

```
Hibernate:
    insert
    into
        mkyongdb.stock
        (STOCK_CODE, STOCK_NAME)
    values
        (?, ?)
Hibernate:
    insert
    into
        mkyongdb.stock_daily_record
        (STOCK_ID, PRICE_OPEN, PRICE_CLOSE, PRICE_CHANGE, VOLUME, DATE)
    values
        (?, ?, ?, ?, ?, ?)
Hibernate:
    update
        mkyongdb.stock_daily_record
    set
        STOCK_ID=?
    where
        RECORD_ID=?
```

Stock will update the “**stock_daily_record.STOCK_ID**” through Set variable (stockDailyRecords), because Stock is the relationship owner.

Note
The third statement is really NOT necessary.

Update example …

When a “Stock” object is updated, Hibernate will generated two SQL statements, one inserts and one update.

Java

```
    session.beginTransaction();

    Stock stock = (Stock)session.get(Stock.class, 57);

    StockDailyRecord stockDailyRecords = new StockDailyRecord();
    stockDailyRecords.setPriceOpen(new Float("1.2"));
    stockDailyRecords.setPriceClose(new Float("1.1"));
    stockDailyRecords.setPriceChange(new Float("10.0"));
    stockDailyRecords.setVolume(3000000L);
    stockDailyRecords.setDate(new Date());

    stockDailyRecords.setStock(stock);
    stock.getStockDailyRecords().add(stockDailyRecords);

    session.save(stockDailyRecords);
    session.update(stock);

    session.getTransaction().commit();
```

Output…

SQL

```
Hibernate:
    insert
    into
        mkyongdb.stock_daily_record
        (STOCK_ID, PRICE_OPEN, PRICE_CLOSE, PRICE_CHANGE, VOLUME, DATE)
    values
        (?, ?, ?, ?, ?, ?)
```

```
Hibernate:
    update
        mkyongdb.stock_daily_record
    set
        STOCK_ID=?
    where
        RECORD_ID=?
```

Note
Again, the third statement is NOT necessary.

5. inverse=”true” Example

If keyword “inverse=true” is defined :

Markup

```
<!--Stock.hbm.xml-->
<class name="com.mkyong.common.Stock" table="stock" ...>
    ...
    <set name="stockDailyRecords" table="stock_daily_record" inverse="true">
```

Now, it means “**stockDailyRecords**” is the relationship owner, and “stock” will not maintains the relationship.

Insert example …

When a “Stock” object is saved, Hibernate will generated two SQL insert statements.

Java

```
    session.beginTransaction();

    Stock stock = new Stock();
    stock.setStockCode("7052");
```

```
    stock.setStockName("PADINI");

    StockDailyRecord stockDailyRecords = new StockDailyRecord();
    stockDailyRecords.setPriceOpen(new Float("1.2"));
    stockDailyRecords.setPriceClose(new Float("1.1"));
    stockDailyRecords.setPriceChange(new Float("10.0"));
    stockDailyRecords.setVolume(3000000L);
    stockDailyRecords.setDate(new Date());

    stockDailyRecords.setStock(stock);
    stock.getStockDailyRecords().add(stockDailyRecords);

    session.save(stock);
    session.save(stockDailyRecords);

    session.getTransaction().commit();
```

Output …

SQL

```
Hibernate:
    insert
    into
        mkyongdb.stock
        (STOCK_CODE, STOCK_NAME)
    values
        (?, ?)
Hibernate:
    insert
    into
        mkyongdb.stock_daily_record
        (STOCK_ID, PRICE_OPEN, PRICE_CLOSE, PRICE_CHANGE, VOLUME, DATE)
    values
        (?, ?, ?, ?, ?, ?)
```

Update example …

When a “Stock” object is updated, Hibernate will generated one SQL statement.

Java

```
    session.beginTransaction();

    Stock stock = (Stock)session.get(Stock.class, 57);

    StockDailyRecord stockDailyRecords = new StockDailyRecord();
    stockDailyRecords.setPriceOpen(new Float("1.2"));
    stockDailyRecords.setPriceClose(new Float("1.1"));
    stockDailyRecords.setPriceChange(new Float("10.0"));
    stockDailyRecords.setVolume(3000000L);
    stockDailyRecords.setDate(new Date());

    stockDailyRecords.setStock(stock);
    stock.getStockDailyRecords().add(stockDailyRecords);

    session.save(stockDailyRecords);
    session.update(stock);

    session.getTransaction().commit();
```

Output…

SQL

```
Hibernate:
    insert
    into
        mkyongdb.stock_daily_record
        (STOCK_ID, PRICE_OPEN, PRICE_CLOSE, PRICE_CHANGE, VOLUME, DATE)
    values
        (?, ?, ?, ?, ?, ?)
```

inverse vs cascade
Many people like to compare between inverse and cascade, but both are totally different notions, see the differential here.

Conclusion

Understanding the “inverse” is essential to optimize your Hibernate code, it helps to avoid many unnecessary update statements, like “insert and update example for inverse=false” above. At last, try to remember the inverse=”true” mean this is the relationship owner to handle the relationship.

Reference

1. http://simoes.org/docs/hibernate-2.1/155.html
2. http://docs.jboss.org/hibernate/stable/core/reference/en/html/example-parentchild.html
3. http://tadtech.blogspot.com/2007/02/hibernate-when-is-inversetrue-and-when.html