

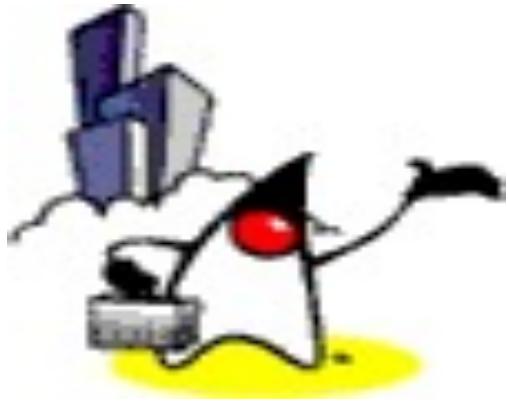
# Hibernate Fetch Strategies & N+1 Select Problem

**“Code with JPassion!”**



# Topics

- Fetching strategies
- Types of fetching strategies
  - How fetching is done
  - When fetching is done
- N+1 select problem
  - Factors that influence number of select statements, thus influence the overall performance
  - Various examples
- Recommendations



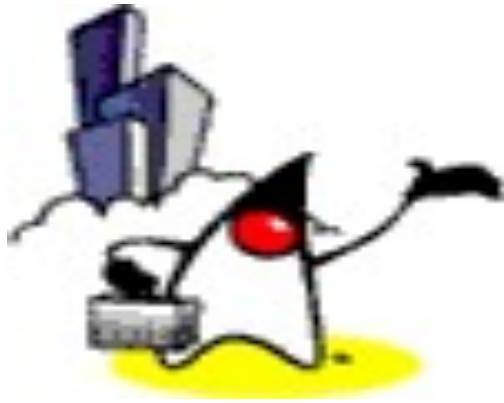
# Fetching Strategies

# What is a Fetching Strategy?

- A fetching strategy is the strategy Hibernate will use for **retrieving associated objects** (from the database server) if the application needs to navigate the association
- Fetching strategy will have performance impact
  - Number of select statements executed will be different depending on fetching strategy
- Fetch strategies may be declared in the mapping files, or over-ridden by a particular HQL or Criteria query

# Types of Fetching Strategies

- How fetching is done
  - Select (default)
  - Join
  - Subselect
  - Batch
- When fetching is done
  - Lazy (default)
  - Immediate



# Fetching Strategies: How Fetching is Done

# Select Fetching

- Select fetching is the default fetch mode
- Vulnerable to “N+1 selects problem”, which means
  - “N” number of SELECT's are executed to retrieve the associated entity or collection

# Join Fetching

- In “Join fetching”, Hibernate retrieves the associated instance or collection **in the same SELECT**, using an INNER JOIN

# Join Fetching in HQL

```
String hql = "from Product p join fetch p.supplier as s";
Query query = session.createQuery(hql);
List results = query.list();
```

# Join Fetching in Criteria API

```
Criteria crit = session.createCriteria(Product.class);
crit.setFetchMode("supplier", FetchMode.JOIN);
List results = crit.list();
```

# Subselect Fetching

- A second SELECT is used to retrieve the associated collections for all entities retrieved in a previous query or fetch.
- Unless you explicitly disable lazy fetching (by specifying *lazy="false"*), this second select will only be executed when you actually access the association.

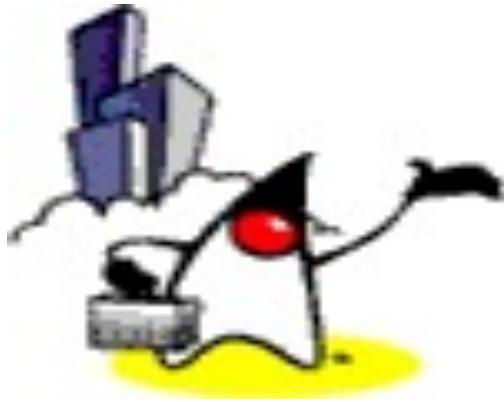
# Batch Fetching

- With batch-size="N" on a collection or an entity class mapping you tell Hibernate to optimize the second SELECT (either lazy or non-lazy) by fetching up to N other collections (or entity instances)

# How to set “fetch” strategy in Mapping file

- Select fetching (the default) is extremely vulnerable to “N+1 selects problem”, so you might want to enable join fetching in the mapping document:

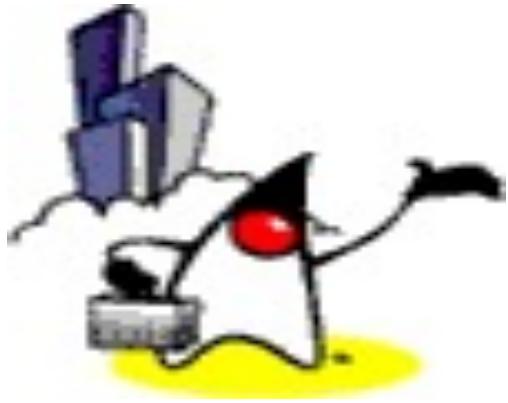
```
<set name="permissions"  
      fetch="join">  
  <key column="userId"/>  
  <one-to-many class="Permission"/>  
</set>
```



# Fetching Strategies: “When” Fetching is Done

# Lazy Fetching

- A collection is fetched when the application invokes an operation upon that collection
- Default



# N+1 Select Problem

# Example Scenario We are going to use

- *Supplier* with a one-to-many relationship with *Product*
  - One *Supplier* has (or supplies) many *Products*

# Example Data: Supplier & Product Tables

\*\*\*\*\* Table: Supplier \*\*\*\*\*

| ID | NAME            |
|----|-----------------|
| 1  | Supplier Name 1 |
| 2  | Supplier Name 2 |
| 3  | Supplier Name 3 |
| 4  | Supplier Name 4 |

\*\*\*\*\* Table: Product \*\*\*\*\*

| ID | NAME      | DESCRIPTION        | PRICE | SUPPLIERID |
|----|-----------|--------------------|-------|------------|
| 1  | Product 1 | Name for Product 1 | 2.0   | 1          |
| 2  | Product 2 | Name for Product 2 | 22.0  | 1          |
| 3  | Product 3 | Name for Product 3 | 30.0  | 2          |
| 4  | Product 4 | Name for Product 4 | 7.0   | 3          |

# Factors that influence “number of select statements”

- “When” fetching is done
  - *lazy* mode for *Supplier*
- “How” fetching is done
  - Fetch mode used for querying on *Product*
  - Select vs. Join
- Whether *Supplier* is “accessed” or not in the application (through *Product*)
- Caching
  - Whether *Supplier* is cached or not

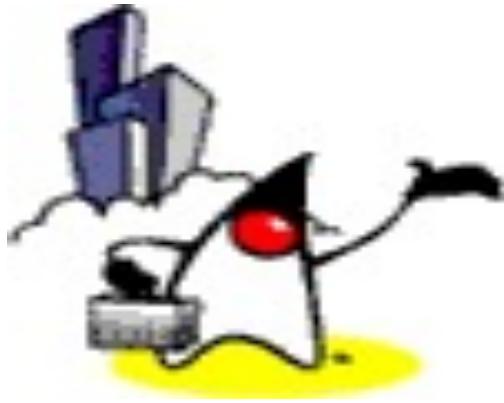
# Hands-on Lab Exercises

1. *LazyTrue-SelectFetch-SupplierNotAccessed*
2. *LazyTrue-SelectFetch-SupplierAccessed*
3. *LazyFalse-SelectFetch-SupplierNotAccessed*
4. *LazyFalse-SelectFetch-SupplierAccessed*
5. *LazyTrue-JoinFetch-SupplierNotAccessed*
6. *LazyTrue-JoinFetch-SupplierAccessed*
7. *LazyFalse-JoinFetch-SupplierNotAccessed*
8. *LazyFalse-JoinFetch-SupplierAccessed*

## Hands-on Lab Exercises

9. *LazyFalse-JoinFetchLeftOuter-SupplierAccessed*

10. *LazyFalse-JoinFetchRightOuter-SupplierAccessed*



# Example #1

## LazyTrue-SelectFetch- SupplierNotAccessed

# Example #1

- Factors
  - *lazy* mode for *Supplier* set to “true” (default)
  - Fetch mode used for querying on *Product* is Select fetch mode (default)
  - *Supplier* information is not accessed in the application
  - Caching does not play a role for the first time the *Supplier* is accessed
- Sample project in the hands-on lab
  - *01HibernateHQLQuery-LazyTrue-SelectFetch-SupplierNotAccessed*

# Supplier.hbm.xml: lazy="true"

```
<hibernate-mapping>
    <!-- lazy attribute is not set so it takes the default value of true -->
    <class name="Supplier">
        <id name="id" type="int">
            <generator class="increment"/>
        </id>

        <property name="name" type="string"/>
        <bag name="products" inverse="true" cascade="all,delete-orphan">
            <key column="supplierId"/>
            <one-to-many class="Product"/>
        </bag>

    </class>
</hibernate-mapping>
```

## Fetch mode is Select Fetch (default)

```
// It takes Select fetch mode as a default  
Query query = session.createQuery( "from Product p");  
List list = query.list();
```

```
// Supplier is not being accessed  
displayProductsListWithoutSupplierName(results);
```

# Supplier is not accessed

```
public static void displayProductsListWithoutSupplierName(List list){  
    Iterator iter = list.iterator();  
    if (!iter.hasNext()) {  
        System.out.println("No products to display.");  
        return;  
    }  
    while (iter.hasNext()) {  
        Product product = (Product) iter.next();  
        // Note that we are not accessing the Supplier  
        // String msg = product.getSupplier().getName() + "\t";  
        String msg = "\t";  
        msg += product.getName() + "\t";  
        msg += product.getPrice() + "\t";  
        msg += product.getDescription();  
        System.out.println(msg);  
    }  
}
```

# Number of Select Statements

select ... various field names ... from PRODUCT

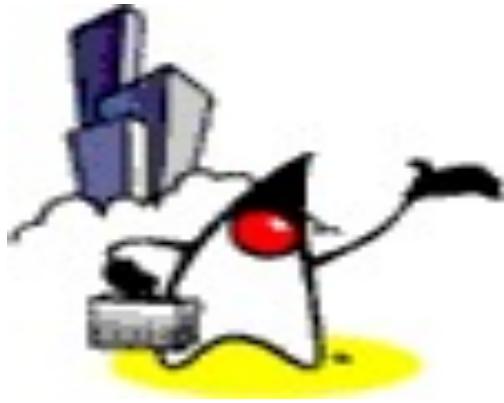
- 1 select statement for Product
- Delayed N+1 select problem, however, since Supplier is not accessed for now

# Lab:

Exercise1:

**01HibernateHQLQuery-LazyTrue-SelectFetch-SupplierNotAccessed  
3519\_hibernate\_joinfetch.zip**





## Example #2

### LazyTrue-SelectFetch-SupplierAccessed

## Example #2

- Factors
  - *lazy* mode for *Supplier* set to “true” (default)
  - Fetch mode used for querying on *Product* is Select fetch mode (default)
  - *Supplier* information is accessed (difference from Example #1)
  - Caching does not play a role for the first time the *Supplier* is accessed
- Sample project in the hands-on lab
  - *O2HibernateHQLQuery-LazyTrue-SelectFetch-SupplierAccessed*

## Fetch mode is Select Fetch (default)

```
// It takes Select fetch mode as a default  
Query query = session.createQuery( "from Product p");  
List list = query.list();
```

```
// Supplier is being accessed  
displayProductsListWithSupplierName(results);
```

# Supplier is accessed

```
public static void displayProductsListWithSupplierName(List list){  
    Iterator iter = list.iterator();  
    if (!iter.hasNext()) {  
        System.out.println("No products to display.");  
        return;  
    }  
    while (iter.hasNext()) {  
        Product product = (Product) iter.next();  
        String msg = product.getSupplier().getName() + "\t";  
        String msg = "\t";  
        msg += product.getName() + "\t";  
        msg += product.getPrice() + "\t";  
        msg += product.getDescription();  
        System.out.println(msg);  
    }  
}
```

# Select Statements Used

select ... various field names ... from PRODUCT

select ... various field names ... from SUPPLIER where SUPPLIER.id=?

select ... various field names ... from SUPPLIER where SUPPLIER.id=?

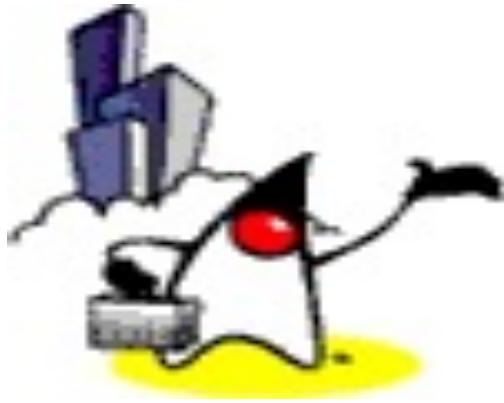
select ... various field names ... from SUPPLIER where SUPPLIER.id=?

- Result
  - 1 select statement for Product
  - N select statements for Supplier
- This is N+1 select problem!!!

# Lab:

**Exercise 2:  
02HibernateHQLQuery-LazyTrue-SelectFetch-SupplierAccessed  
3519\_hibernate\_joinfetch.zip**





## Example #3

### **LazyFalse-SelectFetch- SupplierNotAccessed**

# Example Scenario #3

- Factors
  - *lazy mode for Supplier* set to “**false**” (**non-default**)
  - Fetch mode used for querying on *Product* is **Select fetch mode (default)**
  - *Supplier* information is **not** accessed
  - Caching does not play a role for the first time the *Supplier* is accessed
- Sample project in the hands-on lab
  - *03HibernateHQLQuery-LazyFase-SelectFetch-SupplierNotAccessed*

# Supplier.hbm.xml: lazy="false"

```
<hibernate-mapping>
```

```
  <class name="Supplier" lazy="false">
    <id name="id" type="int">
      <generator class="increment"/>
    </id>

    <property name="name" type="string"/>
    <bag name="products" inverse="true" cascade="all,delete-orphan">
      <key column="supplierId"/>
      <one-to-many class="Product"/>
    </bag>

  </class>
</hibernate-mapping>
```

## Fetch mode is Select Fetch (default)

```
// It takes Select fetch mode as a default  
Query query = session.createQuery( "from Product p");  
List list = query.list();
```

```
// Supplier is not being accessed in our code  
displayProductsListWithoutSupplierName(results);
```

# Select Statements Used

select ... various field names ... from PRODUCT

select ... various field names ... from SUPPLIER where SUPPLIER.id=?

select ... various field names ... from SUPPLIER where SUPPLIER.id=?

select ... various field names ... from SUPPLIER where SUPPLIER.id=?

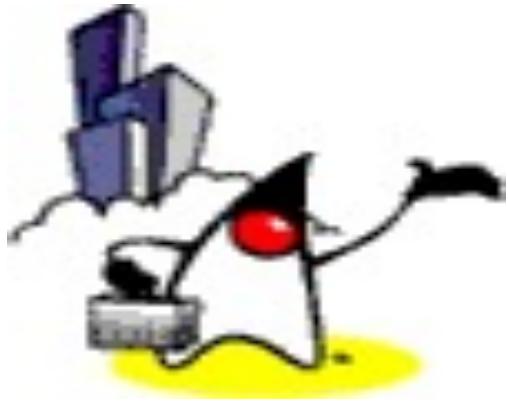
- Result
  - 1 select statement for Product
  - N select statements for Supplier
- This is N+1 select problem!
  - Even though Supplier is not accessed in our code, because **lazy** is set to **false**, N select statements are used to access Supplier

# Lab:

Exercise 3:

**03HibernateHQLQuery-LazyFalse-SelectFetch-SupplierNotAccessed**  
**3519\_hibernate\_joinfetch.zip**





## Example #4

### LazyFalse-SelectFetch-SupplierAccessed

## Example Scenario #4

- Factors
  - *lazy mode for Supplier set to “false”*
  - Fetch mode used for querying on *Product* is Select fetch mode (default)
  - *Supplier information is accessed*
  - Caching does not play a role for the first time the *Supplier* is accessed
- Sample project in the hands-on lab
  - *04HibernateHQLQuery-LazyFase-SelectFetch-SupplierAccessed*

# Select Statements Used

select ... various field names ... from PRODUCT

select ... various field names ... from SUPPLIER where SUPPLIER.id=?

select ... various field names ... from SUPPLIER where SUPPLIER.id=?

select ... various field names ... from SUPPLIER where SUPPLIER.id=?

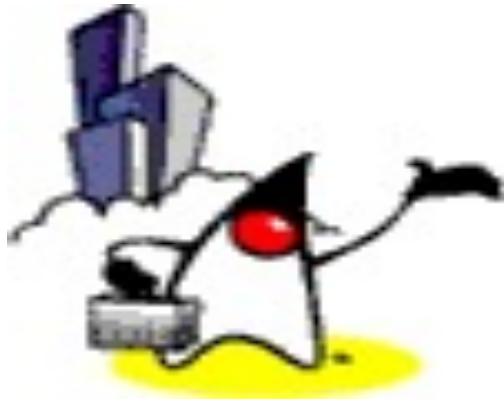
- Result
  - 1 select statement for Product
  - N select statements for Supplier
- **This is N+1 select problem!**
  - Regardless the explicit access to Supplier in our code, because lazy is set to false, N select statements are used to access Supplier

# Lab:

Exercise 4:

**04HibernateHQLQuery-LazyFalse-SelectFetch-SupplierAccessed  
3519\_hibernate\_joinfetch.zip**





# Example #5

## LazyTrue-JoinFetch- SupplierNotAccessed

## Example #5

- Factors
  - *lazy* mode for *Supplier* set to “true” (default)
  - Fetch mode used for querying on *Product* is **Join fetch mode**
  - *Supplier* information is **not** accessed
  - Caching does not play a role for the first time the *Supplier* is accessed
- Sample project in the hands-on lab
  - *05HibernateHQLQuery-LazyTrue-**JoinFetch**-SupplierNotAccessed*

## Fetch mode is Join Fetch

```
// Perform Join Fetch  
String hql = "from Product p join fetch p.supplier as s";  
Query query = session.createQuery(hql);  
List results = query.list();  
  
// Supplier is not being accessed  
displayProductsListWithoutSupplierName(results);
```

# Number of Select Statements

```
select ... various field names from Product product0_ inner join  
Supplier supplier1_ on product0_.supplierId=supplier1_.id
```

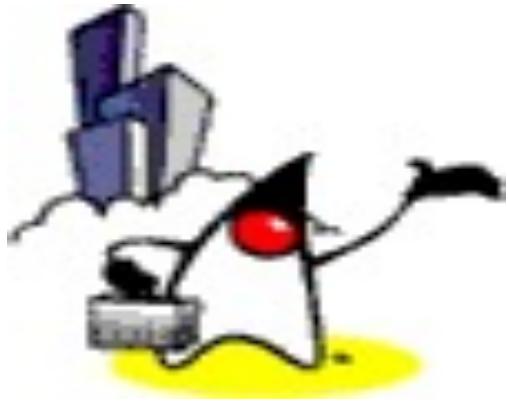
- 1 **inner join select statement**
- No N+1 select problem

# Lab:

Exercise 5:

05HibernateHQLQuery-LazyTrue-JoinFetch-SupplierNotAccessed  
3519\_hibernate\_joinfetch.zip





# Example #6

## LazyTrue-JoinFetch-SupplierAccessed

## Example #6

- Factors
  - *lazy* mode for *Supplier* set to “true” (default)
  - Fetch mode used for querying on *Product* is Join fetch mode
  - *Supplier* information **is accessed**
  - Caching does not play a role for the first time the *Supplier* is accessed
- Sample project in the hands-on lab
  - *06HibernateHQLQuery-LazyTrue-JoinFetch-SupplierAccessed*

## Fetch mode is Join Fetch

*// Perform Join Fetch*

```
String hql = "from Product p join fetch p.supplier as s";  
Query query = session.createQuery(hql);  
List results = query.list();
```

*// Supplier is being accessed*

```
displayProductsListWithSupplierName(results);
```

# Number of Select Statements

```
select ... various field names from Product product0_ inner join  
Supplier supplier1_ on product0_.supplierId=supplier1_.id
```

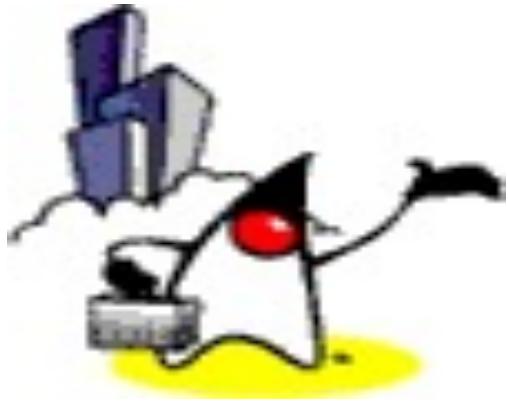
- 1 **inner join** select statement
- No N+1 select problem

# Lab:

Exercise 6:

**06HibernateHQLQuery-LazyTrue-JoinFetch-SupplierAccessed  
3519\_hibernate\_joinfetch.zip**





# Example #7

## **LazyFalse-JoinFetch-SupplierNotAccessed**

## Example #7

- Factors
  - *lazy* mode for *Supplier* set to “**false**”
  - Fetch mode used for querying on *Product* is **Join** fetch mode
  - *Supplier* information is **not** accessed
  - Caching does not play a role for the first time the *Supplier* is accessed
- Sample project in the hands-on lab
  - *07HibernateHQLQuery-LazyFalse-JoinFetch-SupplierNotAccessed*

# Number of Select Statements

```
select ... various field names from Product product0_ inner join  
Supplier supplier1_ on product0_.supplierId=supplier1_.id
```

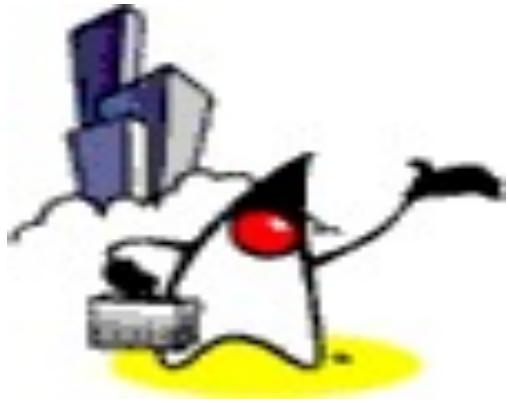
- 1 **inner join** select statement
- No N+1 select problem

# Lab:

Exercise 7:

**07HibernateHQLQuery-LazyFalse-JoinFetch-SupplierNotAccessed**  
**3519\_hibernate\_joinfetch.zip**





## Example #8

### LazyFalse-JoinFetch-SupplierAccessed

## Example #8

- Factors
  - *lazy mode for Supplier set to “false”*
  - Fetch mode used for querying on *Product* is Join fetch mode
  - *Supplier information is accessed*
  - Caching does not play a role for the first time the *Supplier* is accessed
- Sample project in the hands-on lab
  - *08HibernateHQLQuery-LazyFalse-JoinFetch-SupplierAccessed*

# Number of Select Statements

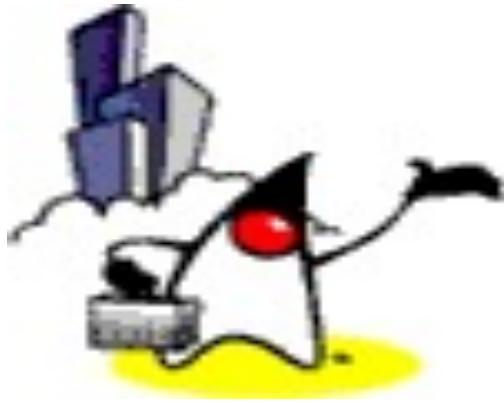
```
select ... various field names from Product product0_ inner join  
Supplier supplier1_ on product0_.supplierId=supplier1_.id
```

- 1 **inner join** select statement
- No N+1 select problem

# Lab:

**Exercise 8:  
08HibernateHQLQuery-LazyFalse-JoinFetch-SupplierAccessed  
3519\_hibernate\_joinfetch.zip**





# Example #9 #10

Use “Left Outer Join” &

Use “Right Outer Join”

# Lab:

Exercise 9:

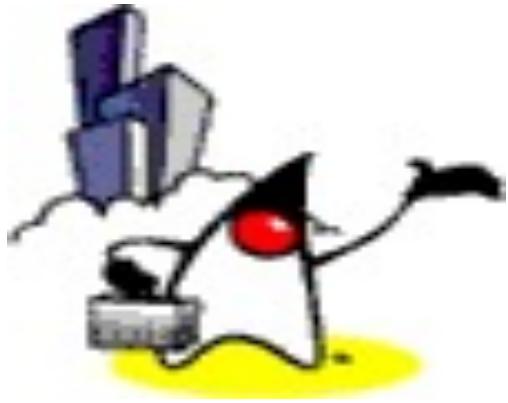
**09HibernateHQLQuery-LazyFalse-JoinFetchLeftOuter-SupplierAccessed**

Exercise 10:

**10HibernateHQLQuery-LazyFalse-JoinFetchRightOuter-SupplierAccessed**

**3519\_hibernate\_joinfetch.zip**





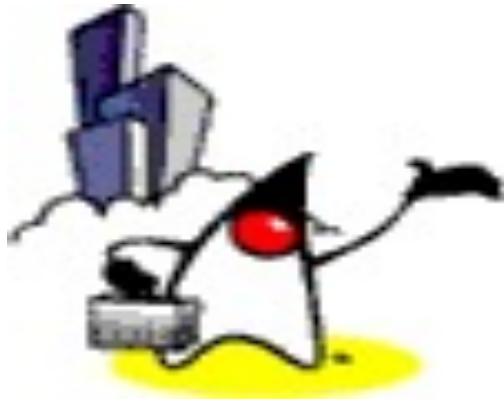
## **Example #10, #11, #12, #13**

### **Use Criteria Query**

# Lab:

**Exercise 11, 12, 13, 14:  
11HibernateCriteriaQuery-LazyTrue-SelectFetch-SupplierNotAccessed  
3519\_hibernate\_joinfetch.zip**





# Example #15

## LazyTrue-SelectFetch-SupplierAccessed-Caching

## Example #15

- Factors
  - *lazy mode for Supplier* set to “true” (default)
  - Fetch mode used for querying on *Product* is Select fetch mode (default)
  - *Supplier* information is accessed
  - Caching play a role the *Supplier* is accessed the second time
- Sample project in the hands-on lab
  - *HibernateHQLQuery-LazyTrue-SelectFetch-SupplierAccessed-Caching*

# Access Supplier Through Cache

```
System.out.println("\n--Performing HQL query with LazyTrue-SelectFetch-SupplierAccessed...");  
Query query = session.createQuery("from Product");  
List results = query.list();  
displayProductsListWithSupplierName(results);  
  
System.out.println("\n--Performing HQL query using Cache...");  
Query query = session.createQuery("from Product");  
List results = query.list();  
displayProductsListWithSupplierName(results);  
  
// Clear cache  
session.clear();  
System.out.println("\n--Performing HQL query after clearing Cache...");  
Query query = session.createQuery("from Product");  
List results = query.list();  
displayProductsListWithSupplierName(results);
```

# Select Statements Used

select ... various field names ... from PRODUCT

select ... various field names ... from SUPPLIER where SUPPLIER.id=?

select ... various field names ... from SUPPLIER where SUPPLIER.id=?

select ... various field names ... from SUPPLIER where SUPPLIER.id=?

select ... various field names ... from PRODUCT

select ... various field names ... from PRODUCT

select ... various field names ... from SUPPLIER where SUPPLIER.id=?

select ... various field names ... from SUPPLIER where SUPPLIER.id=?

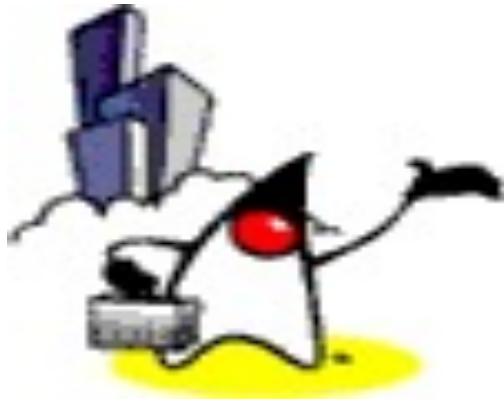
select ... various field names ... from SUPPLIER where SUPPLIER.id=?

# Lab:

Exercise 15:

15HibernateHQLQuery-LazyTrue-SelectFetch-SupplierAccessed-Caching  
3519\_hibernate\_joinfetch.zip





# Fetching Strategies: Recommendations

# Recommendation

- Join fetching is good for small collection of child objects often used with parent
- Large collections and/or not always used collections are better retrieved with lazy select fetching

# Code with Passion!

