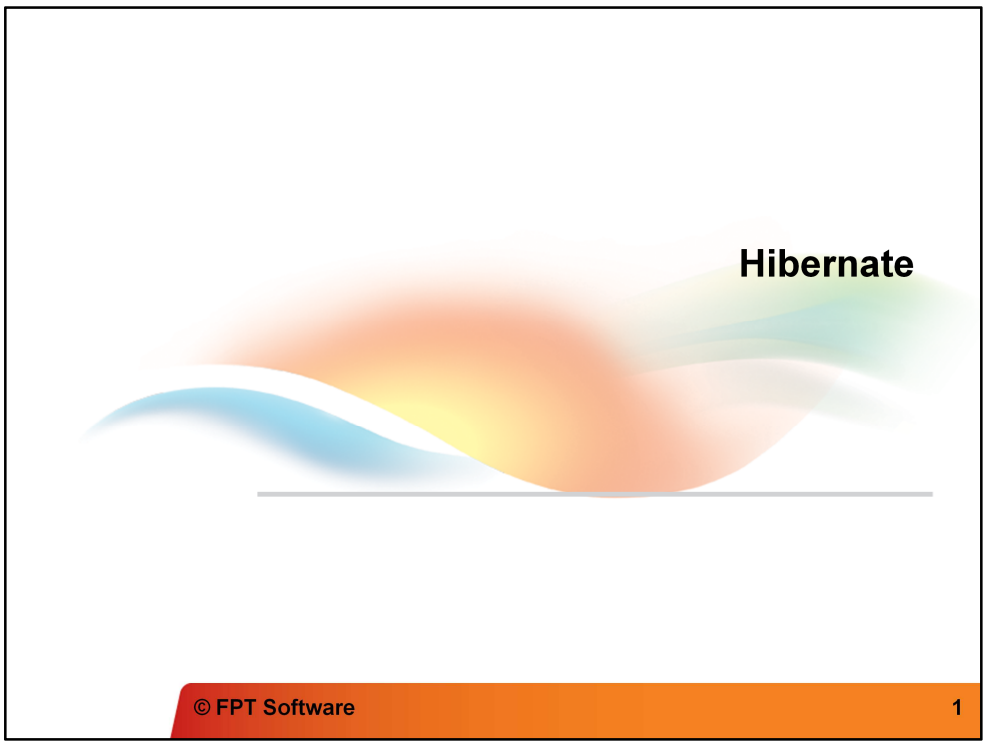
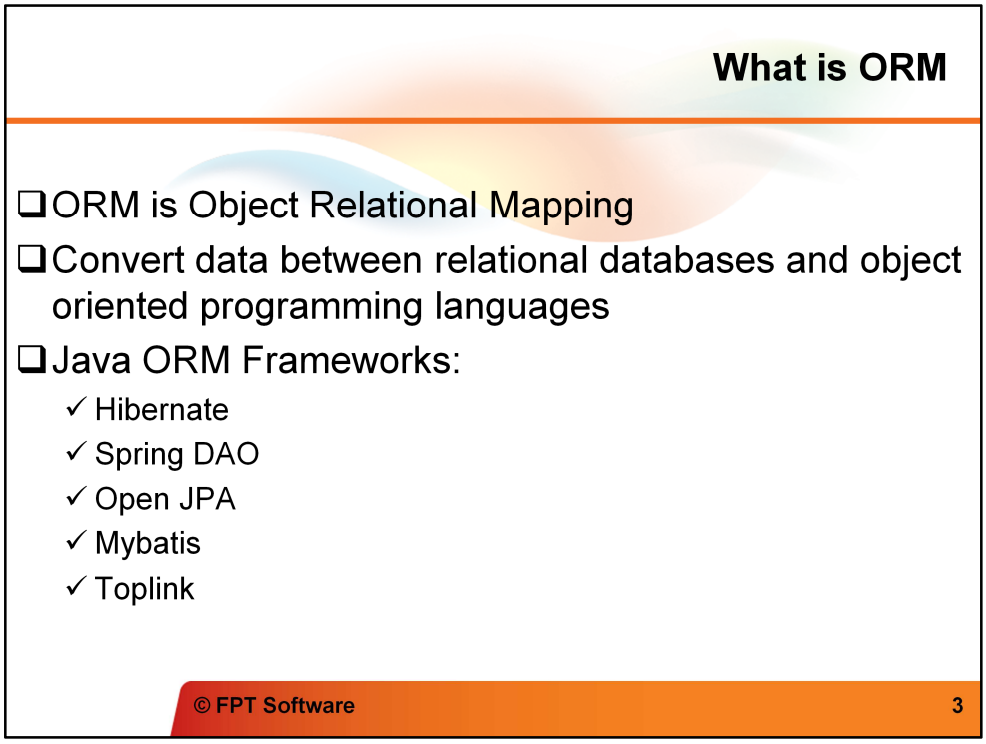

Hibernate
© FPT Software
1

Agenda

- ORM
- Hibernate Overview
- Hibernate Association & Collection mapping
- Hibernate Configuration
- Hibernate Query Language (HQL)
- Criteria Queries
- Native SQL
- Transaction
- Build a Hibernate Application

Overview: What is hibernate & high level architecture



Object - Relational mapping: Map object-oriented domain model to relational database

Persistence of associations and collections

Hibernate, open source ORM framework, widely used

Open JPA, Apache, open source, supports JPA API

Mybatis, free open source, formerly named iBATIS

Toplink by Oracle

What is Hibernate

- Hibernate is an Object-Relational mapping framework for object persistence

Object - Relational mapping: Map object-oriented domain model to relational database

Persistence of associations and collections

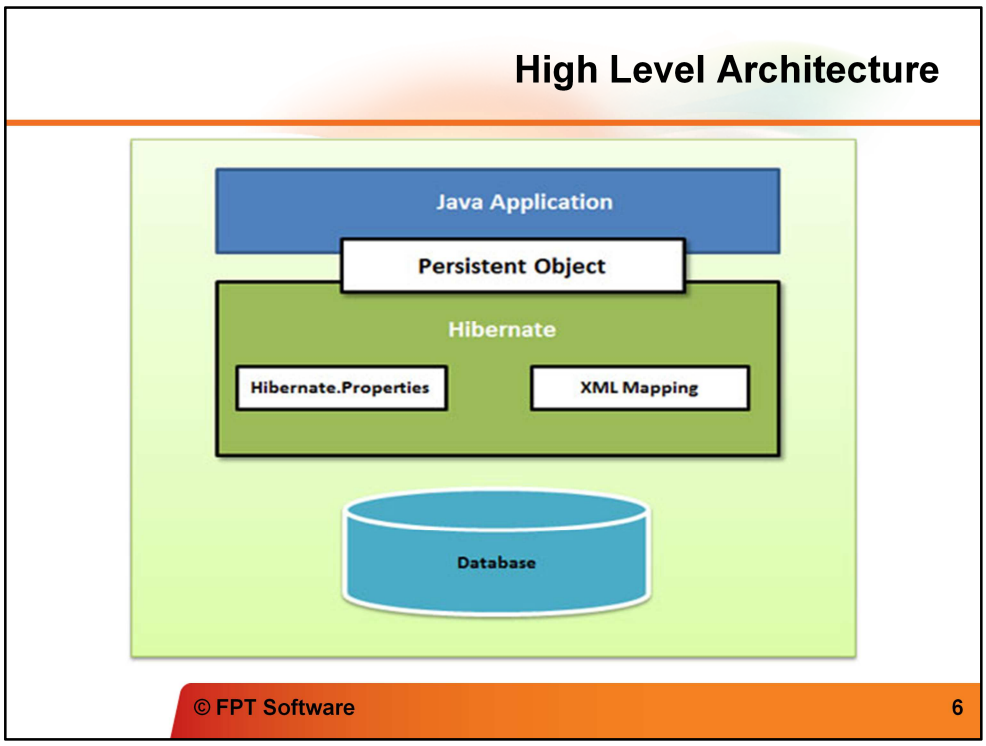
Hibernate advantages

- Mapping Java class to table
- Support for Query Language
- Database dependent Code, support multi databases: MySQL, MS SQL Server, Oracle…
- Optimize performance
- Automatic Versioning and Time Stamping

Mapping: Do not need to write code for mapping

Optimize Performance: Relational tuples are moved to this cache as a result of query. It improves performance if client application reads same data many times for same write.

Optimistic locking: Assured that the changes done by one person is not being roll backed by another one unintentionally



Persistent objects are the classes that in your program that has a representation in the database

Basic O/R Mapping - XML file

File : Stock.hbm.xml

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">

<hibernate-mapping>
    <class name="com.fsoft.stock.Stock" table="stock" catalog="training">
        <id name="stockId" type="java.lang.Integer">
            <column name="STOCK_ID" />
            <generator class="identity" />
        </id>
        <property name="stockCode" type="string">
            <column name="STOCK_CODE" length="10" not-null="true" unique="true" />
        </property>
        <property name="stockName" type="string">
            <column name="STOCK_NAME" length="20" not-null="true" unique="true" />
        </property>
    </class>
</hibernate-mapping>
```


Create the model class and mapping files are quite tedious in large application

With Hibernate tools, this can be generate automatically

Basic O/R Mapping - Annotations

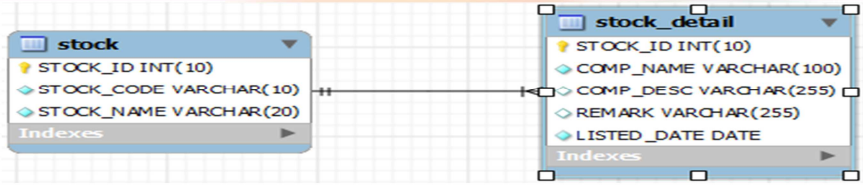
```
@Entity
@Table(name = "stock", catalog = "training", uniqueConstraints = {
        @UniqueConstraint(columnNames = "STOCK_NAME"),
        @UniqueConstraint(columnNames = "STOCK_CODE") })
public class Stock implements java.io.Serializable {
    private Integer stockId;
    private String stockCode;
    private String stockName;
    public Stock() {
    }
    @Id
    @GeneratedValue(strategy = IDENTITY)
    @Column(name = "STOCK_ID", unique = true, nullable = false)
    public Integer getStockId() {
        return this.stockId;
    }
    @Column(name = "STOCK_CODE", unique = true, nullable = false, length = 10)
    public String getStockCode() {
        return this.stockCode;
    }
    @Column(name = "STOCK_NAME", unique = true, nullable = false, length = 20)
    public String getStockName() {
        return this.stockName;
```


@Entity annotation marks this class as an entity bean

@**Table** annotation allows you to specify the details of the table that will be used to persist the entity in the database

@**Column** annotation is used to specify the details of the column to which a field or property will be mapped.

Association Mapping – One to One



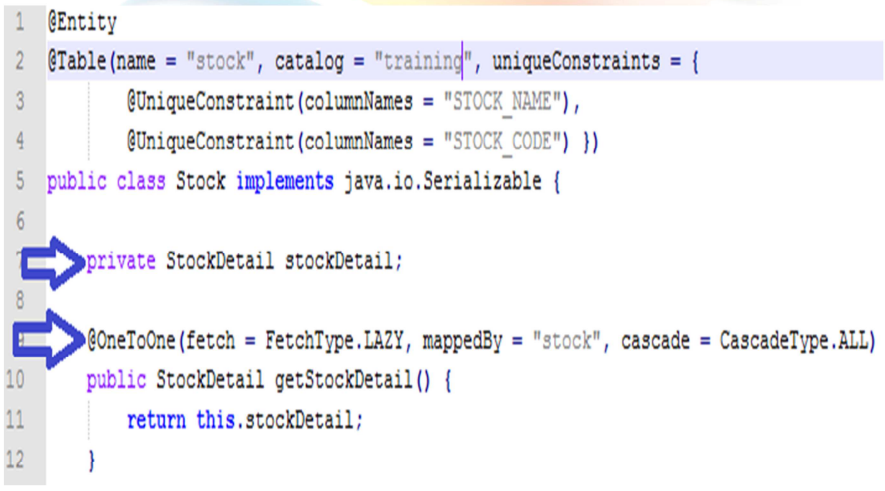

```
1  @Entity
2  @Table(name = "stock", catalog = "training", uniqueConstraints = {
3          @UniqueConstraint(columnNames = "STOCK_NAME"),
4          @UniqueConstraint(columnNames = "STOCK_CODE") })
5  public class Stock implements java.io.Serializable {
6
7      private StockDetail stockDetail;
8
9      @OneToOne(fetch = FetchType.LAZY, mappedBy = "stock", cascade = CascadeType.ALL)
10     public StockDetail getStockDetail() {
11         return this.stockDetail;
12     }
```


@Entity annotation marks this class as an entity bean

@**Table** annotation allows you to specify the details of the table that will be used to persist the entity in the database

@**Column** annotation is used to specify the details of the column to which a field or property will be mapped.

Association Mapping – One to One



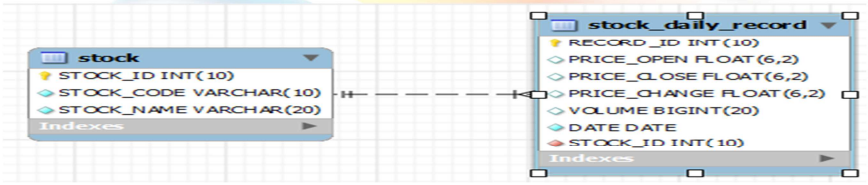
```
1  @Entity
2  @Table(name = "stock", catalog = "training", uniqueConstraints = {
3          @UniqueConstraint(columnNames = "STOCK_NAME"),
4          @UniqueConstraint(columnNames = "STOCK_CODE") })
5  public class Stock implements java.io.Serializable {
6
7      private StockDetail stockDetail;
8
9      @OneToOne(fetch = FetchType.LAZY, mappedBy = "stock", cascade = CascadeType.ALL)
10     public StockDetail getStockDetail() {
11         return this.stockDetail;
12     }
```


@Entity annotation marks this class as an entity bean

@**Table** annotation allows you to specify the details of the table that will be used to persist the entity in the database

@**Column** annotation is used to specify the details of the column to which a field or property will be mapped.

Association & Collection Mapping – One to Many




```
1  @Entity
2  @Table(name = "stock", catalog = "training", uniqueConstraints = {
3          @UniqueConstraint(columnNames = "STOCK_NAME"),
4          @UniqueConstraint(columnNames = "STOCK_CODE") })
5  public class Stock implements java.io.Serializable {
6
7      private Set<StockDailyRecord> stockDailyRecords = new HashSet<StockDailyRecord>(0);
8
9      @OneToMany(fetch = FetchType.LAZY, mappedBy = "stock")
0      public Set<StockDailyRecord> getStockDailyRecords() {
1          return this.stockDailyRecords;
2      }
```


@Entity annotation marks this class as an entity bean

@**Table** annotation allows you to specify the details of the table that will be used to persist the entity in the database

@**Column** annotation is used to specify the details of the column to which a field or property will be mapped.

Association & Collection Mapping – One to Many

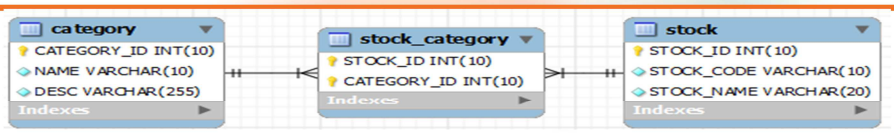
```
@Entity
@Table(name = "stock_daily_record", catalog = "training",
uniqueConstraints = @UniqueConstraint(columnNames = "DATE"))
public class StockDailyRecord implements java.io.Serializable {

    private Stock stock;

    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "STOCK_ID", nullable = false)
    public Stock getStock() {
        return this.stock;
    }
```


@Entity annotation marks this class as an entity bean

@**Table** annotation allows you to specify the details of the table that will be used to persist the entity in the database

@**Column** annotation is used to specify the details of the column to which a field or property will be mapped.

Association & Collection Mapping – Many to Many




```java
@Entity
@Table(name = "category", catalog = "training")
public class Category implements java.io.Serializable {

    private Set<Stock> stocks = new HashSet<Stock>(0);

    @ManyToMany(fetch = FetchType.LAZY, mappedBy = "categories")
    public Set<Stock> getStocks() {
        return this.stocks;
    }
```

Association Mapping – Many to Many

```
1  @Entity
2  @Table(name = "stock", catalog = "training", uniqueConstraints = {
3          @UniqueConstraint(columnNames = "STOCK_NAME"),
4          @UniqueConstraint(columnNames = "STOCK_CODE") })
5  public class Stock implements java.io.Serializable {
6
7      private Set<Category> categories = new HashSet<Category>(0);
8
9      @ManyToMany(fetch = FetchType.LAZY, cascade = CascadeType.ALL)
10     @JoinTable(name = "stock_category", catalog = "mkyongdb", joinColumns = {
11             @JoinColumn(name = "STOCK_ID", nullable = false, updatable = false) },
12             inverseJoinColumns = { @JoinColumn(name = "CATEGORY_ID",
13                     nullable = false, updatable = false) })
14     public Set<Category> getCategories() {
15         return this.categories;
16     }
```


@Entity annotation marks this class as an entity bean

@**Table** annotation allows you to specify the details of the table that will be used to persist the entity in the database

@**Column** annotation is used to specify the details of the column to which a field or property will be mapped.

Working with lazy associations

- Does not actually load all the children when loading the parent.
- Load children when requested to do it
- Lazy loading can help improve the performance

```
1  @Entity
2  @Table(name = "stock_daily_record", catalog = "training",
3  uniqueConstraints = @UniqueConstraint(columnNames = "DATE"))
4  public class StockDailyRecord implements java.io.Serializable {
5
6      private Stock stock;
7
8      @ManyToOne(fetch = FetchType.LAZY)
9      @JoinColumn(name = "STOCK_ID", nullable = false)
10     public Stock getStock() {
11         return this.stock;
12     }
```

Hibernate Configuration

- Hibernate configuration is managed by an instance of org.hibernate.cfg.Configuration
- Hibernate provides following types of configurations
  - ✓ hibernate.properties
  - ✓ hibernate.cfg.xml
  - ✓ Programatic configuration (API-based)

Hibernate Configuration - hibernate.properties

- A Java compliant property file which holds key value pair for different hibernate configuration strings

```
hibernate.connection.driver_class=com.mysql.jdbc.Driver
hibernate.connection.url= jdbc:mysql://localhost:3306/training
hibernate.connection.username=root
hibernate.connection.password=password
hibernate.connection.pool_size=1
```

Hibernate Configuration - hibernate.cfg.xml

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE hibernate-configuration PUBLIC
"-//Hibernate/Hibernate Configuration DTD 3.0//EN"
"http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
    <session-factory>
        <property name="hibernate.connection.driver_class">com.mysql.jdbc.Driver</property>
        <property name="hibernate.connection.url">jdbc:mysql://localhost:3306/training</property>
        <property name="hibernate.connection.username">root</property>
        <property name="hibernate.connection.password">password</property>
        <property name="hibernate.dialect">org.hibernate.dialect.MySQLDialect</property>
        <property name="show_sql">true</property>
        <mapping class="com.fsoft.stock.Stock" />c
        <mapping class="com.fsoft.stock.StockDetail" />
        <mapping class="com.fsoft.stock.StockDailyRecord" />
        <mapping class="com.fsoft.stock.Category" />
    </session-factory>
</hibernate-configuration>
```


Contains all the configuration parameters like database connection, class mappings

Hibernate Configuration - hibernate.cfg.xml

- All configurations in hibernate.cfg.xml will be loaded by using following API

```
SessionFactory sf =
    new  Configuration().configure(). buildSessionFactory();
```

- You can select a different XML configuration file using:

```
SessionFactory sf =
   new Configuration().configure("cust.cfg.xml").buildSessionFactory();
```


Both hibernate.cfg.xml and hibernate.properties files can be provided simultaneously in an application. In this case hibernate.cfg.xml gets precedence over hibernate.properties.

Hibernate Configuration - hibernate.cfg.xml

- All configurations in hibernate.cfg.xml will be loaded by using following API

```
SessionFactory sessionFactory = new  Configuration().configure(). buildSessionFactory();
```

- You can select a different XML configuration file using:

```
SessionFactory sf = new Configuration().configure("cust.cfg.xml").buildSessionFactory();
```


You can select a different XML configuration file using: SessionFactory sf = new Configuration().configure("mkyong.cfg.xml").buildSessionFactory();

Both hibernate.cfg.xml and hibernate.properties files can be provided simultaneously in an application. In this case hibernate.cfg.xml gets precedence over hibernate.properties.

Hibernate Configuration - Programmatic

- Instantiate Configuration directly and specify XML mapping document

```
Configuration cfg = new Configuration()
    .addResource("Stock").addResource("StockDetail")
    .addResource("Category").addResource("StockDailyRecord");
```

- Specify the mapped class

```
Configuration cfg = new Configuration()
    .addClass(Stock.class).addClass(StockDetail.class)
    .addClass(Category.class).addClass(StockDailyRecord.class);
```

Hibernate Query Language (HQL)

- Syntax is quite similar to database SQL language
- Uses class name instead of table name, and property names instead of column name

Retrieve a stock data where stock code is "7277".

```
Query query = session.createQuery("from Stock where stockCode = :code ");
query.setParameter("code", "7277");
List list = query.list();
```

```
Query query = session.createQuery("from Stock where stockCode = '7277' ");
List list = query.list();
```

HQL- Example

- Update a stock name to “DIALOG1″ where stock code is “7277″

```
Query query = session.createQuery("update Stock set stockName = :stockName" +
                                  " where stockCode = :stockCode");
query.setParameter("stockName", "DIALOG1");
query.setParameter("stockCode", "7277");
int result = query.executeUpdate();
```

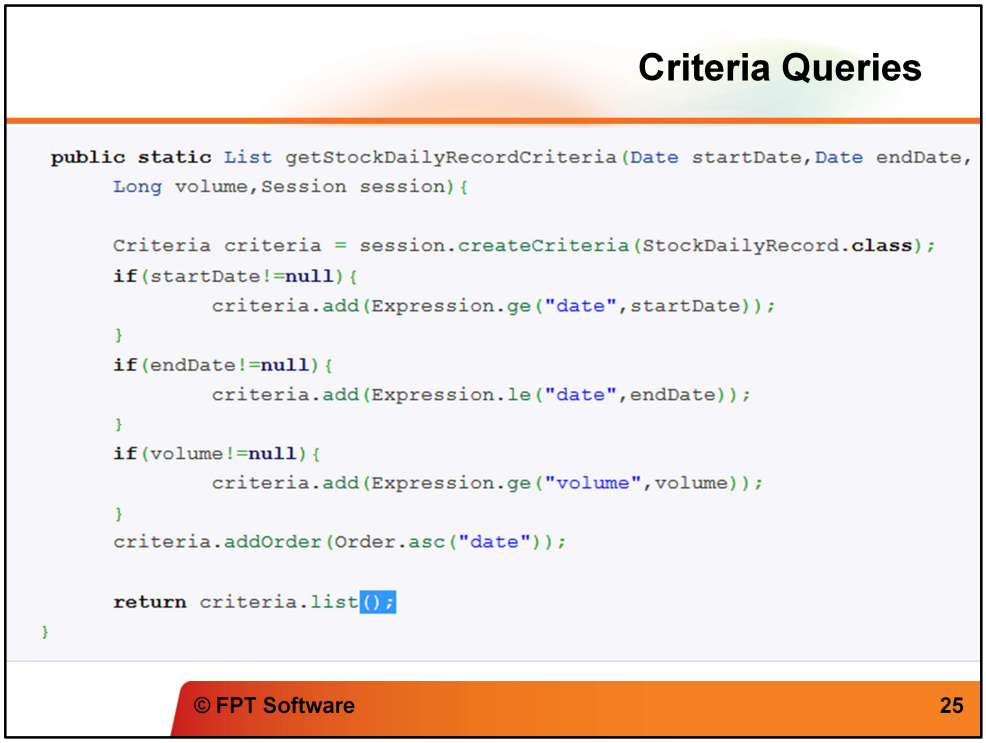
- Delete a stock where stock code is “7277″

```
Query query = session.createQuery("delete Stock where stockCode = :stockCode");
query.setParameter("stockCode", "7277");
int result = query.executeUpdate();
```

Criteria Queries

```
public static List getStockDailtRecord(Date startDate,Date endDate,
    Long volume,Session session){
    SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd");
    boolean isFirst = true;
    StringBuilder query = new StringBuilder("from StockDailyRecord ");
    if(startDate!=null){
     if(isFirst){
         query.append(" where date >= '" + sdf.format(startDate) + "'");
     }else{
         query.append(" and date >= '" + sdf.format(startDate) + "'");
     }
     isFirst = false;
    }
    if(endDate!=null){
     if(isFirst){
         query.append(" where date <= '" + sdf.format(endDate) + "'");
     }else{
         query.append(" and date <= '" + sdf.format(endDate) + "'");
     }
     isFirst = false;
    }
    if(volume!=null){
     if(isFirst){
         query.append(" where volume >= " + volume);
     }else{
         query.append(" and volume >= " + volume);
     }
     isFirst = false;
    }
    query.append(" order by date");
    Query result = session.createQuery(query.toString());
    return result.list();
}
```


Here's a case study to retrieve a list of **StockDailyRecord**, with optional search criteria – start date, end date and volume, order by date.



Java-based for API for building queries

Good for queries that are built up using lots of conditional logic; avoid messy string manipulation

This is Criteria example, you do not need to compare whether this is the first criteria to append the 'where' syntax, nor format the date. The line of code is reduce and everything is handled in a more elegant and object oriented way.

Criteria Queries

- **Restrictions.eq, lt, le, gt, ge**

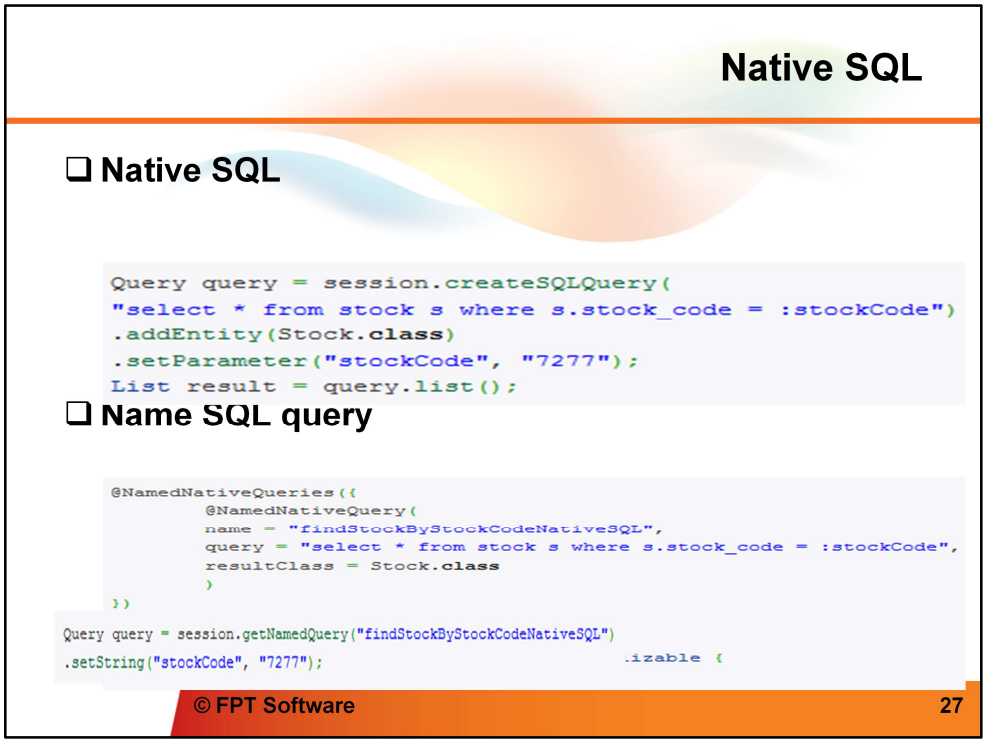
Make sure the volume is great than 10000.

```
Criteria criteria = session.createCriteria(StockDailyRecord.class)
    .add(Restrictions.gt("volume", 10000));
```

- **Restrictions.like, between, isNull, isNotNull**

Make sure the stock name is start with 'MKYONG' and follow by any characters.

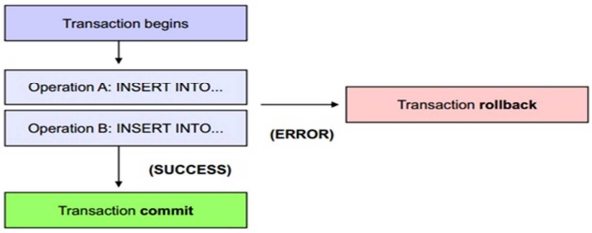
```
Criteria criteria = session.createCriteria(StockDailyRecord.class)
    .add(Restrictions.like("stockName", "MKYONG%"));
```



Hibernate provide a **createSQLQuery** method to let you call your native SQL statement directly.

Transactions

- A set of database operations which must be executed in entirely or not at all
- Should and either with a commit or a rollback
- All communication with a database has to occur inside a transaction



Transaction Handle

```
Session session = null;
Transaction tx = null;

try{
        session = HibernateUtil.getSessionFactory().openSession();
        tx = session.beginTransaction();
        tx.setTimeout(5);

        //doSomething(session);

        tx.commit();


}catch(RuntimeException e){
        try{
                tx.rollback();
        }catch(RuntimeException rbe){
                log.error("Couldn't roll back transaction", rbe);
        }
        throw e;
}finally{
        if(session!=null){
                session.close();
        }
}
```

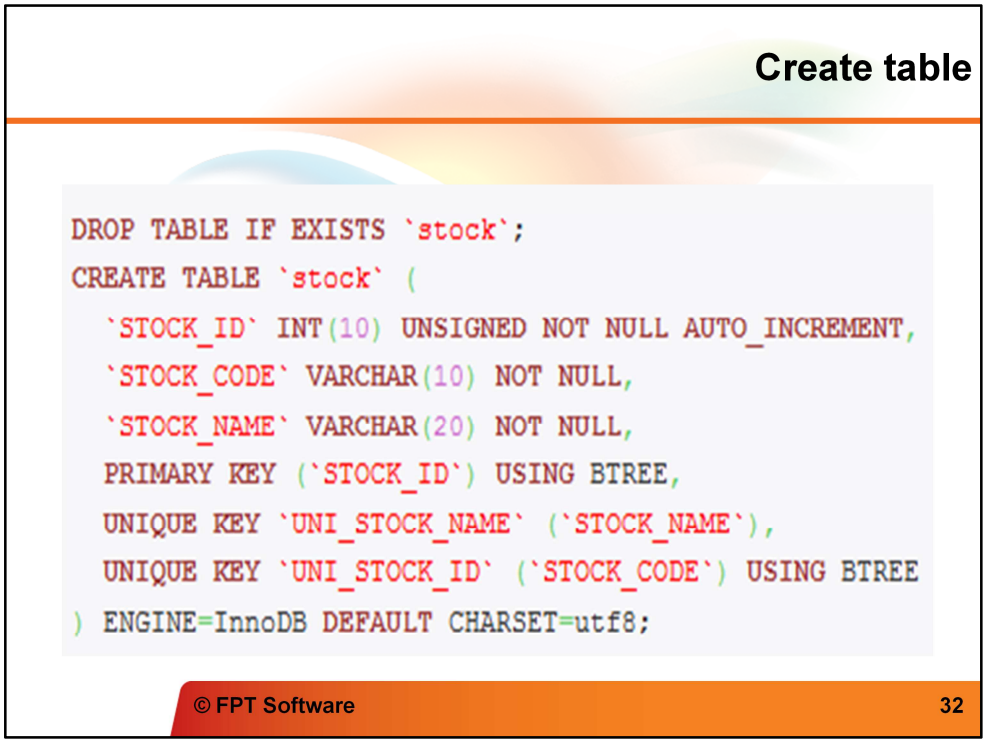

Any exceptions thrown by Hibernate are **FATAL**, you have to roll back the transaction and close the current session immediately

Building Hibernate Application

- Maven 3 + Hibernate 3 + MySQL 5.5 + JDK 1.6
- POM: Add Hibernate and MySQL dependency
- MySQL Script: Create table in MySQL
- Create the domain model with annotation
- Create Hibernate configuration file hibernate.cfg.xml
- Implement a HibernateUtil class
- Write main class

Add Hibernate and MySQL dependency

```
    <version>1.0</version>
    <name>HibernateExample</name>
    <url>http://maven.apache.org</url>
    <repositories>
        <repository>
            <id>JBoss repository</id>
            <url>http://repository.jboss.org/nexus/content/groups/public/</url>
        </repository>
    </repositories>
    <dependencies>
        <!-- MySQL database driver -->
        <dependency>
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-java</artifactId>
            <version>5.1.9</version>
        </dependency>
        <!-- Hibernate framework -->
        <dependency>
            <groupId>org.hibernate</groupId>
            <artifactId>hibernate-core</artifactId>
            <version>3.6.3.Final</version>
        </dependency>
        <dependency>
            <groupId>javassist</groupId>
            <artifactId>javassist</artifactId>
            <version>3.12.1.GA</version>
        </dependency>
    </dependencies>
</project>
```



MySQL script to create a “stock” table

Create the domain model with annotation

```
@Entity
@Table(name = "stock", catalog = "mkyong", uniqueConstraints = {
        @UniqueConstraint(columnNames = "STOCK_NAME"),
        @UniqueConstraint(columnNames = "STOCK_CODE") })
public class Stock implements java.io.Serializable {
    private Integer stockId;
    private String stockCode;
    private String stockName;

    public Stock() {
    }
    public Stock(String stockCode, String stockName) {
        this.stockCode = stockCode;
        this.stockName = stockName;
    }
    @Id
    @GeneratedValue(strategy = IDENTITY)
    @Column(name = "STOCK_ID", unique = true, nullable = false)
    public Integer getStockId() {
        return this.stockId;
    }
    public void setStockId(Integer stockId) {
        this.stockId = stockId;
    }
    @Column(name = "STOCK_CODE", unique = true, nullable = false, length = 10)
    public String getStockCode() {
        return this.stockCode;
    }
```

Create Hibernate Configuration file

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE hibernate-configuration PUBLIC
"-//Hibernate/Hibernate Configuration DTD 3.0//EN"
"http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
    <session-factory>
        <property name="hibernate.connection.driver_class">com.mysql.jdbc.Driver</property>
        <property name="hibernate.connection.url">jdbc:mysql://localhost:3306/training</property>
        <property name="hibernate.connection.username">root</property>
        <property name="hibernate.connection.password">password</property>
        <property name="hibernate.dialect">org.hibernate.dialect.MySQLDialect</property>
        <property name="show_sql">true</property>
        <mapping class="com.fsoft.stock.Stock" />c
        <mapping class="com.fsoft.stock.StockDetail" />
        <mapping class="com.fsoft.stock.StockDailyRecord" />
        <mapping class="com.fsoft.stock.Category" />
    </session-factory>
</hibernate-configuration>
```


Make Hibernate aware of the mapping classes : Stock

Create Hibernate Utility class

```java
public class HibernateUtil {
    private static final SessionFactory sessionFactory = buildSessionFactory();
    private static SessionFactory buildSessionFactory() {
        try {
            // Create the SessionFactory from hibernate.cfg.xml
            return new AnnotationConfiguration().configure().buildSessionFactory();
        }
        catch (Throwable ex) {
            // Make sure you log the exception, as it might be swallowed
            System.err.println("Initial SessionFactory creation failed." + ex);
            throw new ExceptionInInitializerError(ex);
        }
    }
    public static SessionFactory getSessionFactory() {
        return sessionFactory;
    }
    public static void shutdown() {
        getSessionFactory().close();
    }
}
```


Convenience class to handle building and obtaining the Hibernate SessionFactory

- Use recommended by the Hibernate org

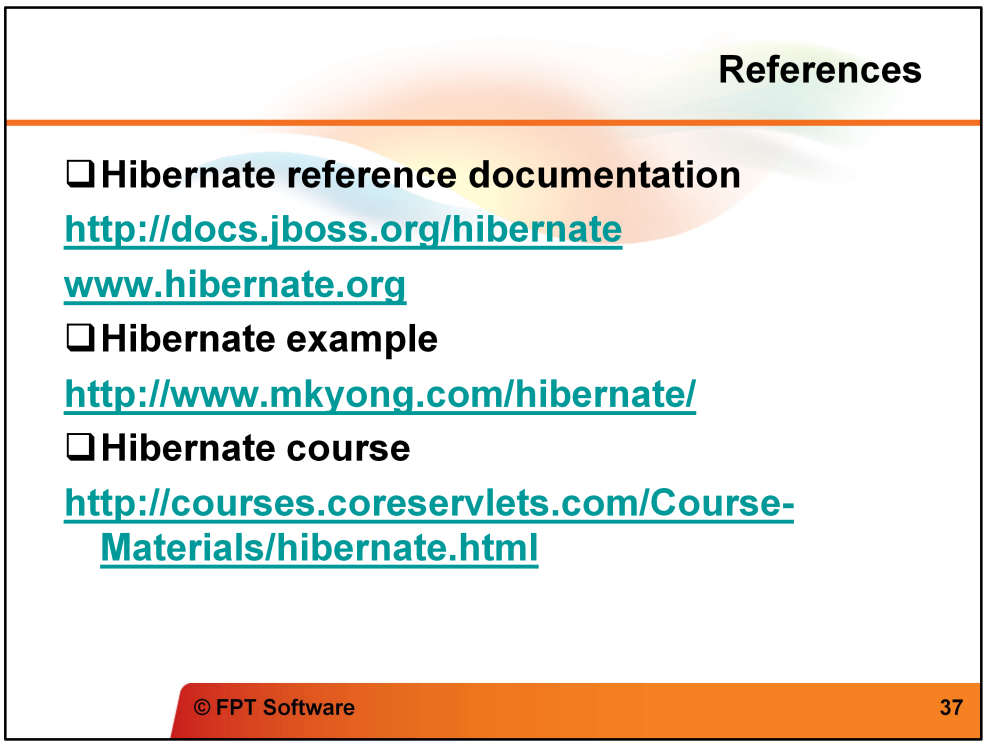
Used to build Hibernate 'Sessions'

Main class

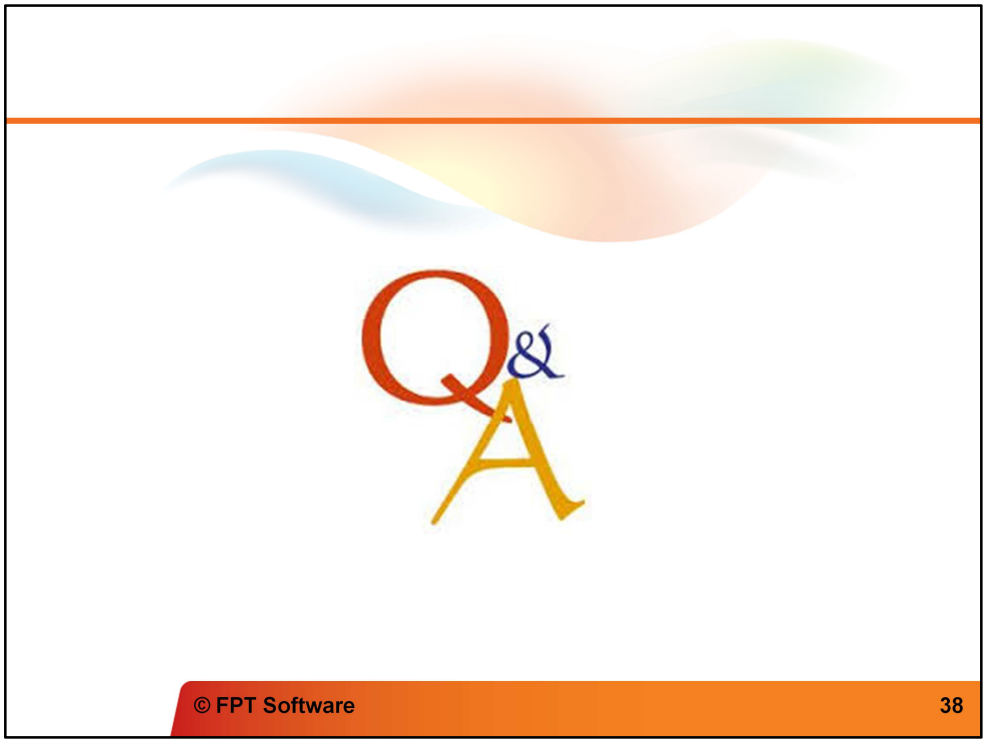
```
package com.fsoft.stock;

import org.hibernate.Session;
import com.fsoft.util.HibernateUtil;

public class App
{
    public static void main( String[] args )
    {
        System.out.println("Maven + Hibernate + MySQL");
        Session session = HibernateUtil.getSessionFactory().openSession();

        session.beginTransaction();
        Stock stock = new Stock();

        stock.setStockCode("4715");
        stock.setStockName("GENM");

        session.save(stock);
        session.getTransaction().commit();
    }
}
```


Run your App.java, it will insert a new record into “Stock” table.

Hibernate: insert into mkyong.stock (STOCK_CODE, STOCK_NAME) values (?, ?)



Một số vấn đề liên qua đến Hibernate.


Q&A