

# HIVE & WEBSERVICE TUTORIAL

---

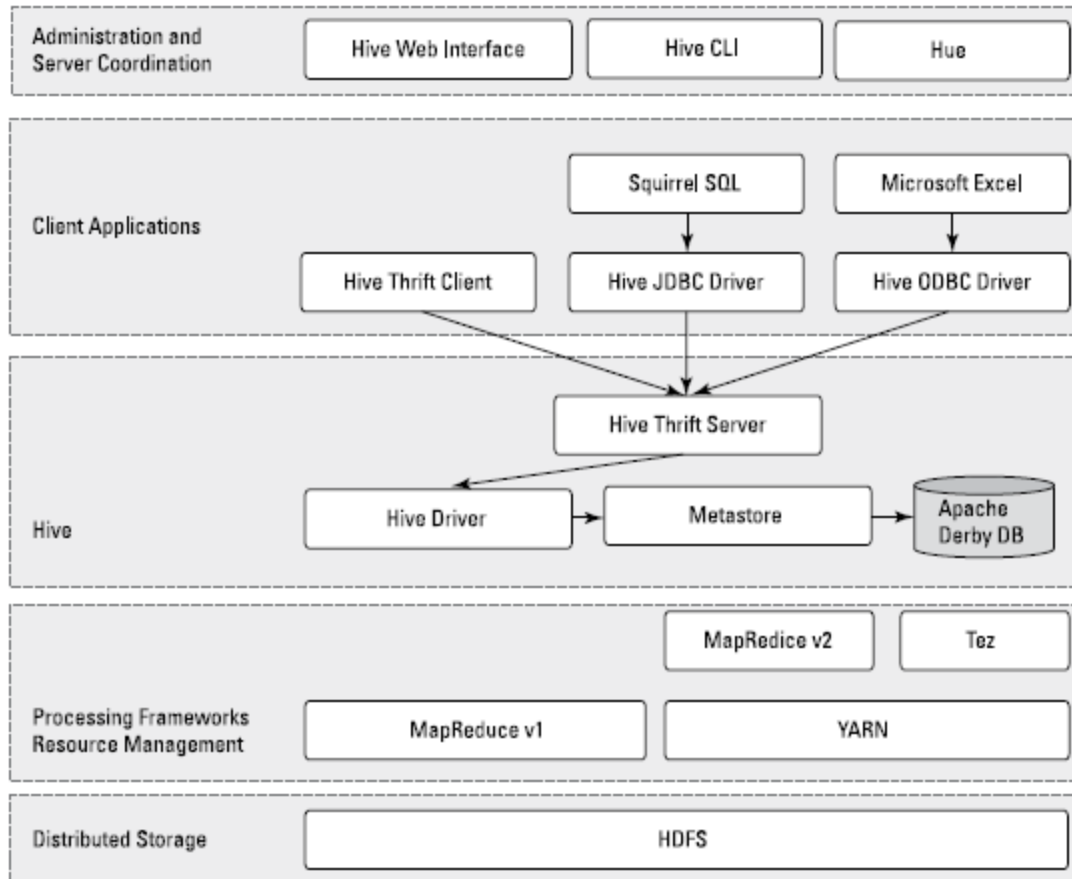
Hands-on Session

by Suchitra Jayaprakash  
suchitra@cmi.ac.in

# Apache HIVE

- HIVE hides the complexity of MapReduce. It provides SQL type script to perform MapReduce task.
- HIVE uses SQL dialect known as HIVE QUERY LANGUAGE (HiveQL).
- HIVE is data warehouse for managing and processing structured data.
- Hive supports "**READ Many WRITE Once**" pattern. Hive is "**Schema on READ only**".
- Apache Hive was created at Facebook by a team of engineers led by Jeff Hammerbacher.

# Apache Hive Architecture



(source: Hadoop for Dummies)

# Run HIVE

- **Start Cloudera server**

```
docker run --hostname=quickstart.cloudera --privileged=true -t -i --  
publish-all=true -p 8888:8888 -p 8080:80 -p 50070:50070 -p 8088:8088 -p  
50075:50075 -p 8032:8032 -p 8042:8042 -p 19888:19888 -p 10000:10000  
cloudera/quickstart /usr/bin/docker-quickstart
```

- **To get HIVE command prompt**
- Type hive and press enter to get hive command line interface.

# HIVE CLI

```
Logging initialized using configuration in fi
roperties
WARNING: Hive CLI is deprecated and migration
hive>
>
> CREATE SCHEMA user_db;
OK
Time taken: 3.927 seconds
hive> show databases;
OK
default
user_db
Time taken: 1.808 seconds, Fetched: 2 row(s)
hive>
```

- Create Database Statement:

**CREATE SCHEMA <database name>;**

It creates database in hive. Database is collection of table.

**SHOW DATABASES;**

It displays the list of databases in hive instance.

- Create Table Statement:

```
CREATE [TEMPORARY] [EXTERNAL] TABLE [IF NOT EXISTS] [db_name.] table_name
[(col_name data_type , ...)]
[COMMENT table_comment]
[ROW FORMAT row_format]
[STORED AS file_format]
```

# HiveQL

- Load OnlineStore dataset into HDFS using HIVE and perform few aggregation operations:

## Step 1 – Create Table

```
CREATE TABLE IF NOT EXISTS Online_Retail ( InvoiceNo STRING, StockCode STRING, Description  
STRING,Quantity INT, InvoiceDate TIMESTAMP, UnitPrice double ,CustomerID INT,Country STRING )  
COMMENT 'Online Retail Data Set'  
ROW FORMAT DELIMITED  
FIELDS TERMINATED BY ','  
LINES TERMINATED BY '\n'  
STORED AS TEXTFILE;
```

```
CREATE TABLE IF NOT EXISTS tmp(InvoiceNo STRING, StockCode STRING, Description STRING,  
Quantity INT, InvoiceDate STRING,UnitPrice double ,CustomerID STRING,Country STRING)  
ROW FORMAT DELIMITED  
FIELDS TERMINATED BY ','  
LINES TERMINATED BY '\n'  
STORED AS TEXTFILE;
```

```
hive> CREATE TABLE IF NOT EXISTS Online_Retail ( InvoiceNo STRING, StockCode STR  
ING, Description STRING,  
> Quantity INT, InvoiceDate TIMESTAMP,UnitPrice double ,CustomerID INT,Count  
ry STRING )  
> COMMENT 'Online Retail Data Set'  
> ROW FORMAT DELIMITED  
> FIELDS TERMINATED BY ','  
> LINES TERMINATED BY '\n'  
> STORED AS TEXTFILE;  
OK  
Time taken: 2.193 seconds  
hive> CREATE TABLE IF NOT EXISTS tmp(InvoiceNo STRING, StockCode STRING, Descrip  
tion STRING,  
> Quantity INT, InvoiceDate STRING,UnitPrice double ,CustomerID STRING,Count  
ry STRING)  
> ROW FORMAT DELIMITED  
> FIELDS TERMINATED BY ','  
> LINES TERMINATED BY '\n'  
> STORED AS TEXTFILE;  
OK  
Time taken: 0.417 seconds  
hive>
```

# HiveQL

## Step 2 – Load Data

- Copy Text file to docker container

```
docker cp E:/MSc_Datascience/BigDataHadoop/Slides/hive/Online_Retail.csv  
<containerid>:/tmp/Online_Retail.csv
```

- Load Hive tables

```
LOAD DATA LOCAL INPATH '/tmp/Online_Retail.csv'  
OVERWRITE INTO TABLE tmp;
```

```
INSERT INTO TABLE Online_Retail  
SELECT InvoiceNo, StockCode, Description, Quantity,  
from_unixtime(unix_timestamp(InvoiceDate, 'dd-MM-yyyy HH:mm')),  
UnitPrice, CustomerID, Country  
FROM tmp;
```

# HiveQL

```
Time taken: 0.117 seconds
hive> LOAD DATA LOCAL INPATH '/tmp/Online_Retail.csv'
> OVERWRITE INTO TABLE tmp;
Loading data to table default.tmp
Table default.tmp stats: [numFiles=1, numRows=0, totalSize=46123538, rawDataSize=0]
OK
Time taken: 8.551 seconds
hive>
```

```
hive> INSERT INTO TABLE Online_Retail
> SELECT InvoiceNo, StockCode, Description, Quantity,
> from_unixtime(unix_timestamp(InvoiceDate, 'dd-MM-yyyy HH:mm')),
> UnitPrice, CustomerID, Country
> FROM tmp;
Query ID = root_20200314175050_d13e5e65-f5bd-4c27-a85d-9fc45fe129a9
Total jobs = 3
Launching Job 1 out of 3
Number of reduce tasks is set to 0 since there's no reduce operator
Starting Job = job_1584207403190_0001, Tracking URL = http://quickstart.cloudera:8088/proxy/application_1584207403190_0001/
Kill Command = /usr/lib/hadoop/bin/hadoop job -kill job_1584207403190_0001
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 0
2020-03-14 18:00:16,234 Stage-1 map = 0%, reduce = 0%
2020-03-14 18:01:17,174 Stage-1 map = 0%, reduce = 0%, Cumulative CPU 15.73 sec
2020-03-14 18:02:17,926 Stage-1 map = 0%, reduce = 0%, Cumulative CPU 53.73 sec
2020-03-14 18:02:24,806 Stage-1 map = 100%, reduce = 0%, Cumulative CPU 58.01 sec
MapReduce Total cumulative CPU time: 58 seconds 10 msec
Ended Job = job_1584207403190_0001
Stage-4 is selected by condition resolver.
Stage-3 is filtered out by condition resolver.
Stage-5 is filtered out by condition resolver.
Moving data to: hdfs://quickstart.cloudera:8020/user/hive/warehouse/online_retail/.hive-staging_hive_2020-03-14_17-58-35_519_7469576875740326583-1/-ext-10000
Loading data to table default.online_retail
Table default.online_retail stats: [numFiles=1, numRows=541909, totalSize=47486522, rawDataSize=46944613]
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1 Cumulative CPU: 58.01 sec HDFS Read: 46128029 HDFS Write: 47486609 SUCCESS
Total MapReduce CPU Time Spent: 58 seconds 10 msec
OK
Time taken: 240.053 seconds
hive>
```



# HiveQL

## Step 3 – Select operation

```
hive> SELECT * from Online_Retail LIMIT 5;
OK
536365 85123A WHITE HANGING HEART T-LIGHT HOLDER      6      2010-12-01 08:26:00
:00    2.55 17850 United Kingdom
536365 71053  WHITE METAL LANTERN      6      2010-12-01 08:26:00    3.39 1
7850   United Kingdom
536365 84406B  CREAM CUPID HEARTS COAT HANGER  8      2010-12-01 08:26:00    2
.75   17850   United Kingdom
536365 84029G  KNITTED UNION FLAG HOT WATER BOTTLE 6      2010-12-01 08:26:00
:00    3.39 17850   United Kingdom
536365 84029E  RED WOOLLY HOTTIE WHITE HEART.    6      2010-12-01 08:26:00    3
.39   17850   United Kingdom
Time taken: 0.355 seconds, Fetched: 5 row(s)
hive>
```

**SELECT \* from Online\_Retail LIMIT 5;**

**SELECT InvoiceDate from Online\_Retail LIMIT 5;**

**SELECT Country,count(\*) FROM Online\_Retail GROUP BY Country;**

**SELECT \* FROM Online\_Retail WHERE UnitPrice>1000 AND Country = 'United Kingdom';**

Drop table

**DROP TABLE IF EXISTS tmp;**

# Java Client - Hive JDBC Example

- Create Java project using IDE like netbeans.
- Create a java client to execute hive queries.
- Connect to “jdbc:hive2://host:port/dbname” using Hive jdbc driver.
- Submit SQL query by creating a Statement object and using its executeQuery() method.
- Process the result set.

# JAVA Client : Code

```
package hiveclient;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
```

```
public class HiveClient {

    public static void main(String[] args) {

        System.out.println("Getting Data from Hadoop using Hive driver");

        try {
            String driverName = "org.apache.hive.jdbc.HiveDriver";
            Class.forName(driverName);
        } catch (ClassNotFoundException e) {

            e.printStackTrace();
            System.exit(1);
        }
    }
}
```

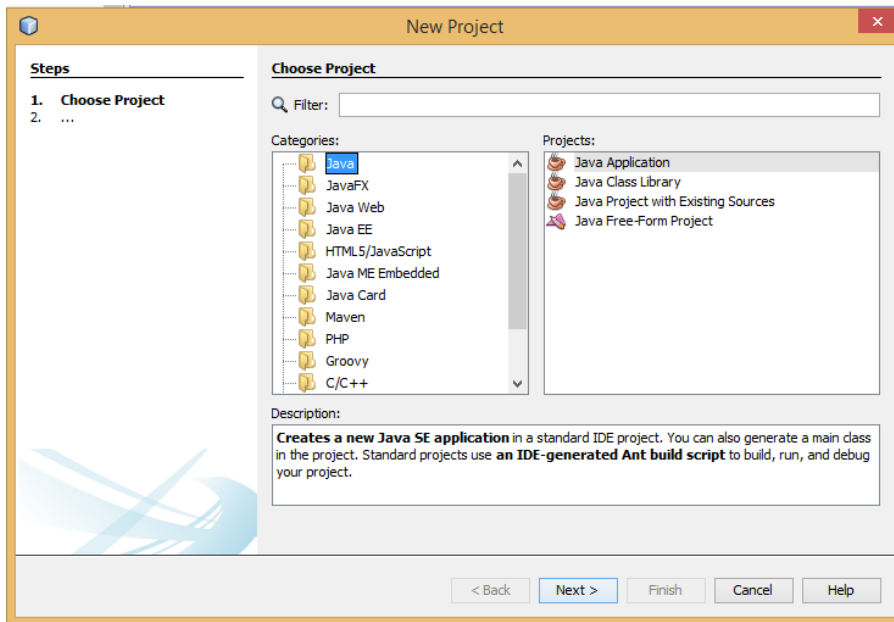
```
try {
    Connection con = DriverManager.getConnection("jdbc:hive2://192.168.99.100:10000/", "cloudera", "cloudera");

    Statement stmt = con.createStatement();
    String sql = "SELECT * FROM Online_Retail WHERE UnitPrice>1000 AND Country = 'United Kingdom'";
    System.out.println("Running: " + sql);
    ResultSet res = stmt.executeQuery(sql);
    while (res.next()) {
        System.out.println(res.getString(1) + "," + res.getString(2) + res.getString(3) + "," + res.getString(4) + res.getString(5)
    }
    res.close();

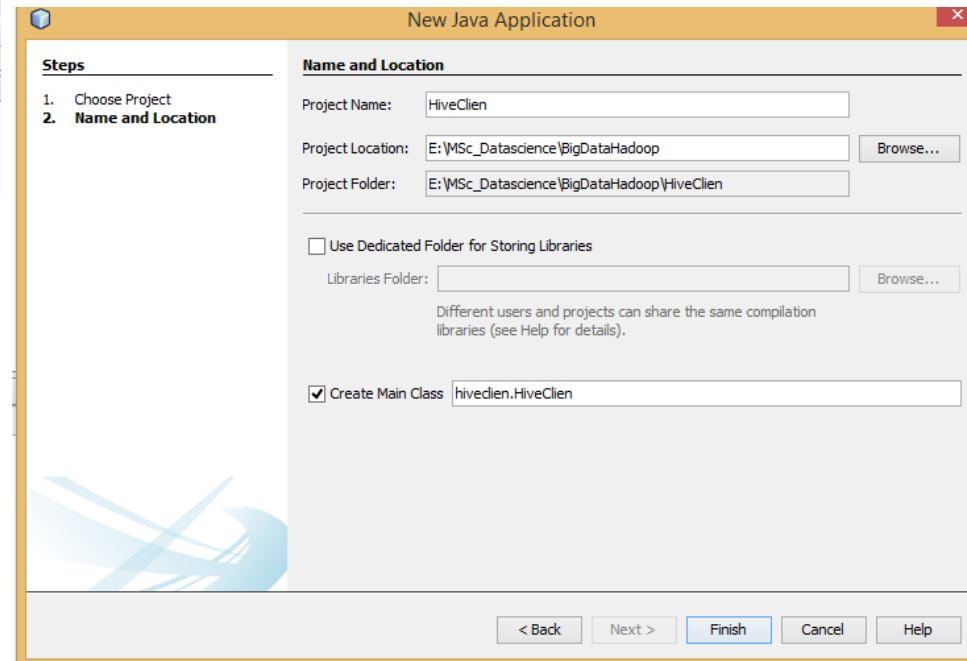
    System.out.println("Query execution complete");
} catch (SQLException e) {

    e.printStackTrace();
}
```

# JAVA Client : Set up



Create Java project HiveClient using IDE like eclipse



# JAVA Client : library files

- Copy HIVE Jar from Cloudera instance

1. Create folder in docker

```
mkdir /tmp/hivejar/
```

2. Create folder in docker

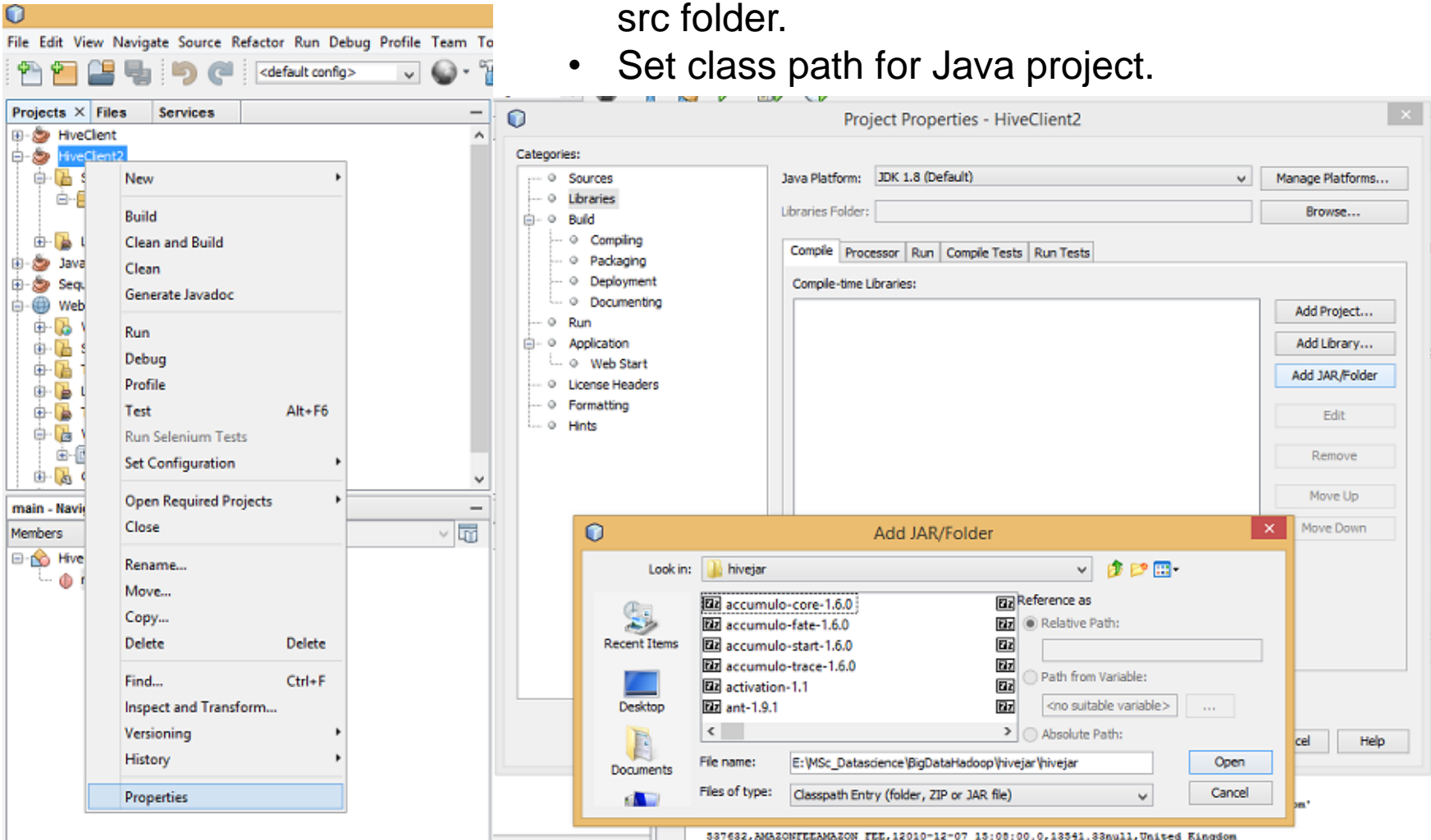
```
cp *.jar /tmp/hivejar/
```

3. Copy folder from docker to local drive

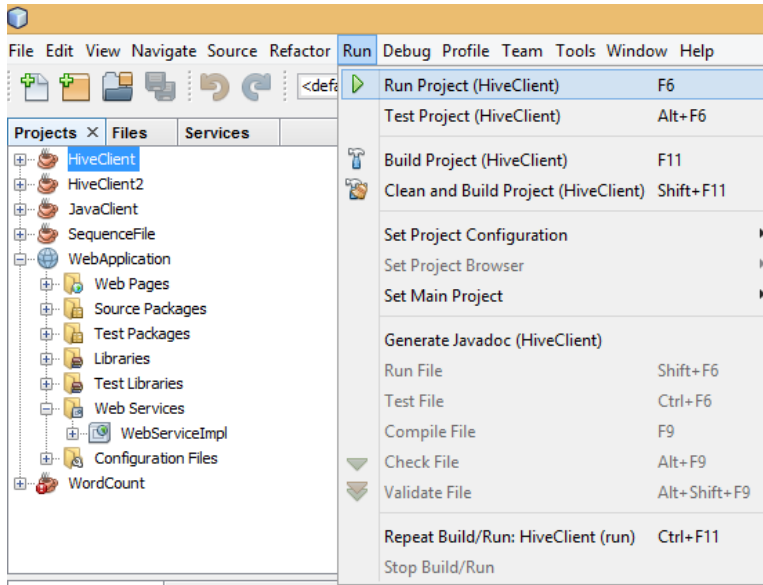
```
docker cp 26ffefdcdcf2f3:/tmp/hivejar  
E:/MSc_Datascience/BigDataHadoop/hivejar
```

# JAVA Client : Set up

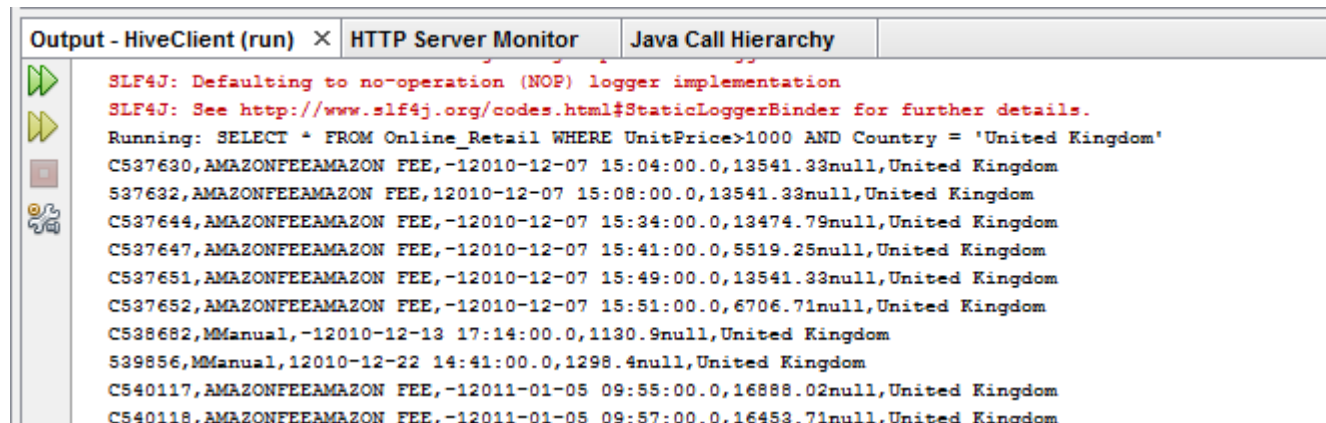
- Copy the HiveClient java class file into the src folder.
- Set class path for Java project.



# JAVA Client : Execute



- Click Run Project
- Output would be displayed in the output window.

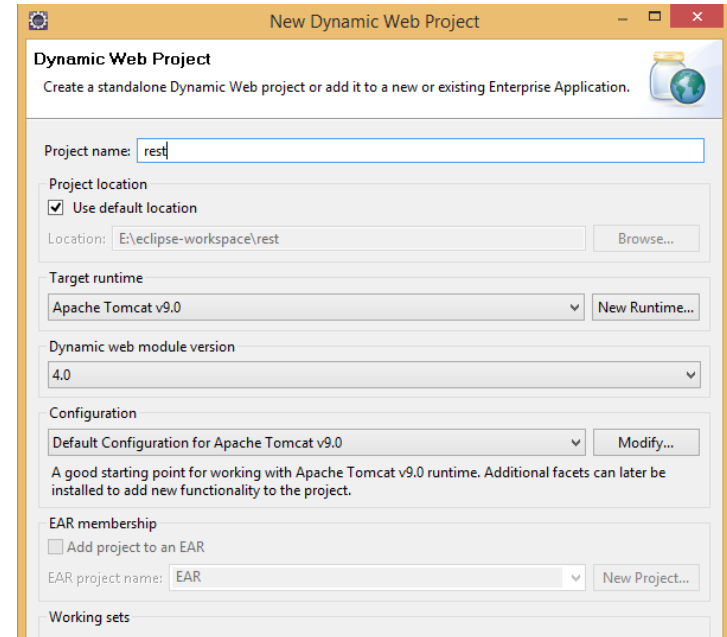
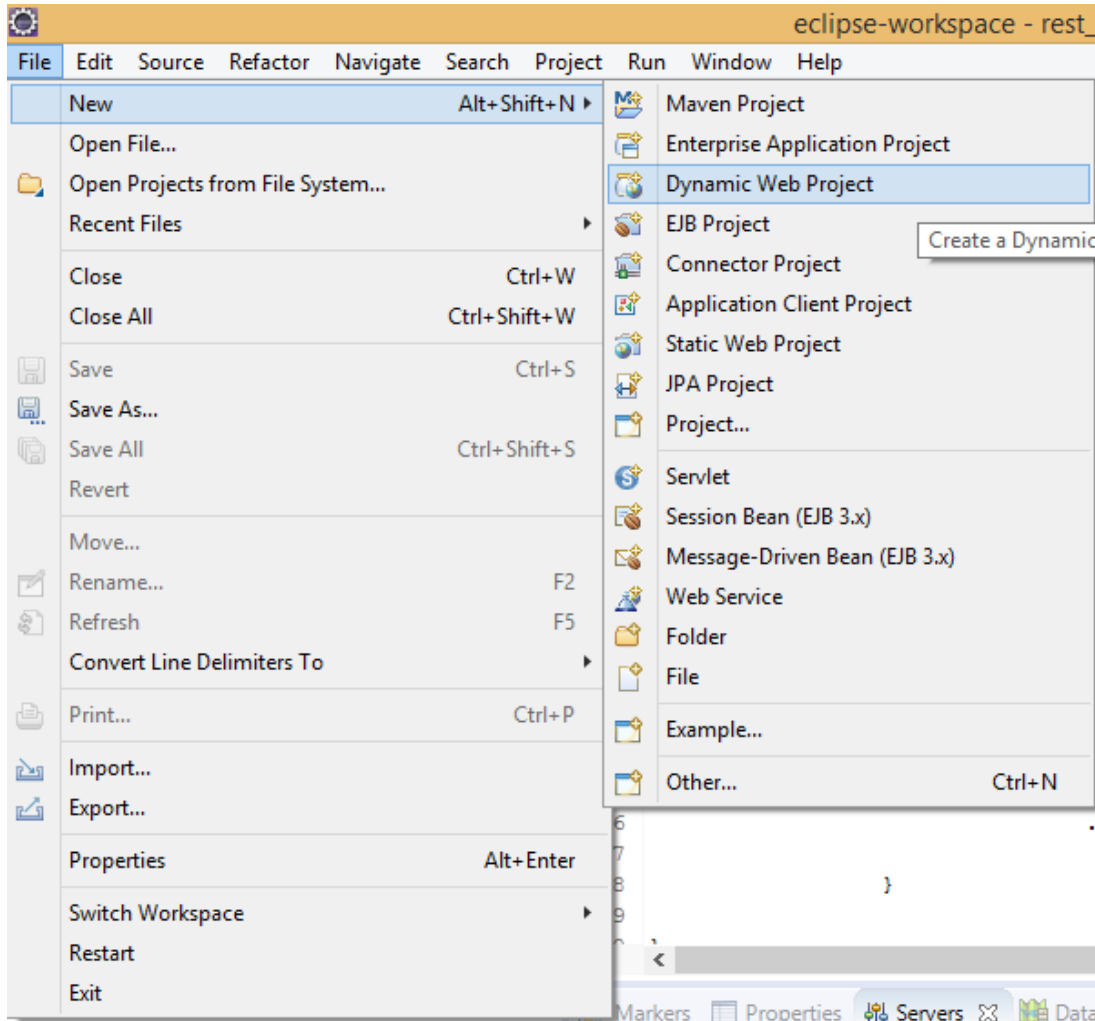


# REST API using JAX RS

- Download eclipse IDE.
- Download jaxrs-ri JAR 2.27 with all dependencies.
- Create a new dynamic web project in Eclipse.
- Copy jaxrs jars and hive jar to WEB-INF\lib directory of the dynamic web application.
- Create three java class for Rest API

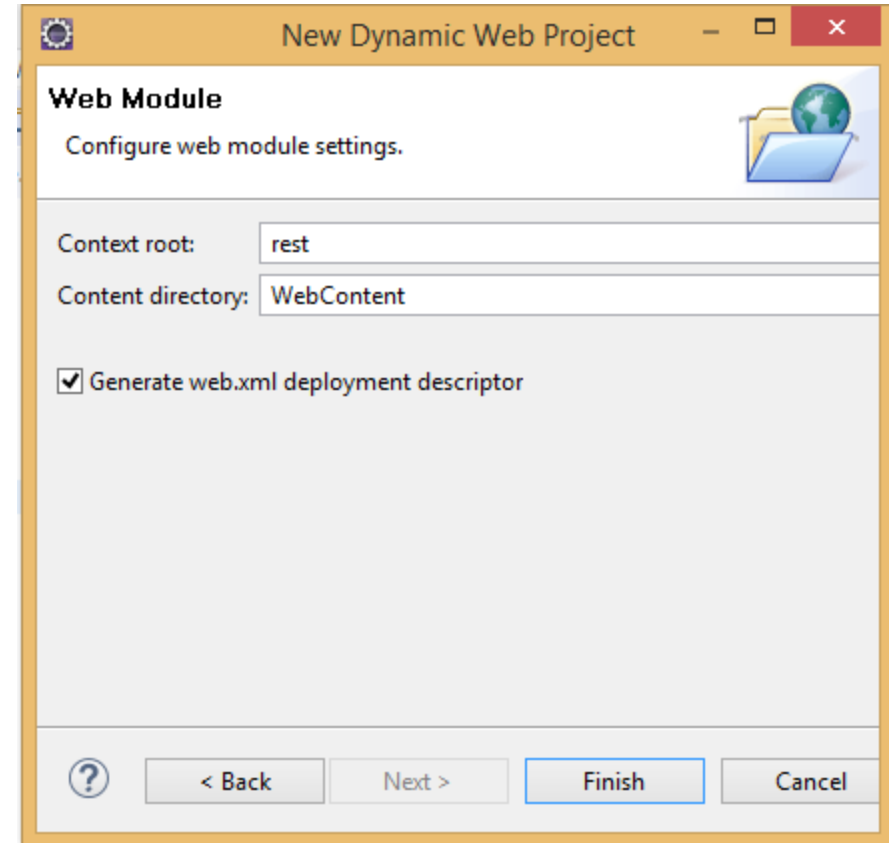
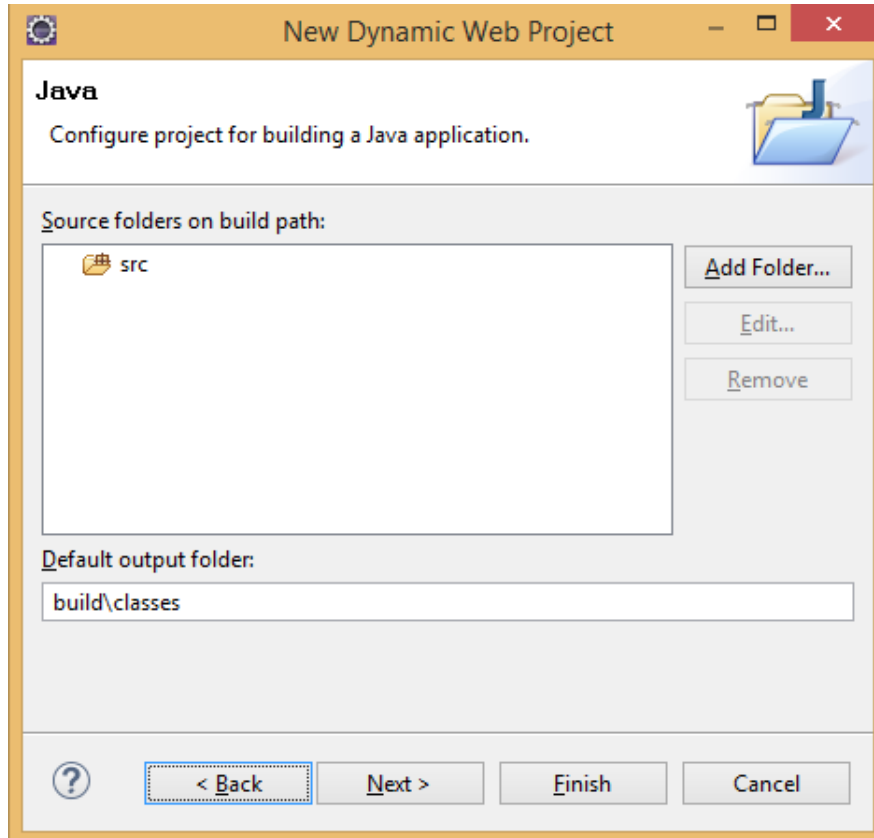


# Create dynamic web project



Using eclipse IDE, Create Dynamic Web application project.

# Create dynamic web project



Select Generate web.xml & click finish

# REST API – Java Classes

- Message.java
  - Data class to hold each column value.
  - Contains getter setter methods for all column.
  - @XmlElement annotation is required for Json response conversion.

```
1 package rest_api;
2 import javax.xml.bind.annotation.XmlRootElement;
3 @XmlElement
4 public class Message {
5
6     private String invoiceNo;
7
8     private String stockCode;
9
10    private String description;
11
12    private String quantity;
13
14    private String invoiceDate;
15
16    private String unitPrice;
17
18    private String customerId;
19    private String country;
20
21
22    public String getInvoiceNo() {
23        return invoiceNo;
24    }
25
```

```
    public String getInvoiceNo() {
        return invoiceNo;
    }
26
27    public void setInvoiceNo(String invoiceNo) {
28        this.invoiceNo = invoiceNo;
29    }
30
31    public String getStockCode() {
32        return stockCode;
33    }
34
35    public void setStockCode(String stockCode) {
36        this.stockCode = stockCode;
37    }
38
39    public String getDescription() {
40        return description;
41    }
42
43    public void setDescription(String description) {
44        this.description = description;
45    }
46
```

```
47
48    public String getUnitPrice() {
49        return unitPrice;
50    }
51
52    public void setUnitPrice(String unitPrice) {
53        this.unitPrice = unitPrice;
54    }
55
56    public String getCustomerId() {
57        return customerId;
58    }
59
60    public void setCustomerId(String customerId) {
61        this.customerId = customerId;
62    }
63
64    public String getCountry() {
65        return country;
66    }
67
68    public void setCountry(String country) {
69        this.country = country;
70    }
71
```

# REST API – Java Classes

- HiveDataService.java
  - It contains the Implementation of service ( In our case hive jdbc call)
  - @Path annotation to provide the context path.
  - Following annotation is used:

URI	HTTP Method	Response Type
@Path("/HiveData/fetchData")	@GET	@Produces(MediaType. <b>APPLICATION_JSON</b> )

```
package rest_api;

import javax.ws.rs.*;

@Path("/")

public class HiveDataService {
    @GET
    @Path("/HiveData/PurchaseDetail")
    @Produces(MediaType.APPLICATION_JSON)

    public Response fetchData() {
        List<Message> messages = new ArrayList<>();
        System.out.println("Getting Data from Hadoop using Hive driver")
```

```
try {
    Connection con = DriverManager.getConnection("jdbc:hive2://192.168.1.100:9000/default");

    Statement stmt = con.createStatement();
    String sql = "SELECT * FROM Online_Retail WHERE UnitPrice>1000 AND UnitPrice<2000";
    System.out.println("Running: " + sql);
    ResultSet res = stmt.executeQuery(sql);
    while (res.next()) {

        Message m = new Message();
        m.setInvoiceNo(res.getString(1));
        m.setStockCode(res.getString(2));
        m.setDescription(res.getString(3));
        m.setQuantity(res.getString(4));
        m.setInvoiceDate(res.getString(5));
        m.setUnitPrice(res.getDouble(6));
        m.setCustomerId(res.getString(7));
        m.setCountry(res.getString(8));

        messages.add(m);
    }
    res.close();

    System.out.println("Query execution complete");
} catch (SQLException e) {
    e.printStackTrace();
}

return Response
    .status(Response.Status.OK)
    .entity(messages)
    .build();
```

- Setting value to data object
- Converting data into Json response

# REST API – Java Classes

- HiveDataApplication.java
  - This class extends Jersey JAX-RS ResourceConfig with an @ApplicationPath annotation.
  - It acts as a hook between RESTful application and the web container.
  - When server is started, it will examine this class and look for JAX-RS annotated classes inside the package listed in the this class constructor.

```
package rest_api;

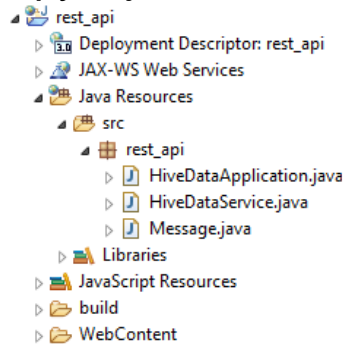
import javax.ws.rs.ApplicationPath;
import org.glassfish.jersey.server.ResourceConfig;

@ApplicationPath("/")
public class HiveDataApplication extends ResourceConfig {

    public HiveDataApplication() {
        packages("rest_api");
    }
}
```








# REST API – Project Setup

1) Copy 3 java file into src folder.

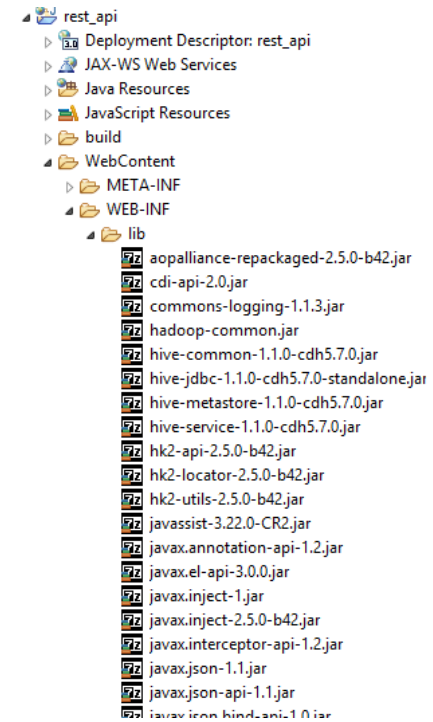


2) Download jaxrs-ri JAR 2.27 with all dependency. <https://jar-download.com/artifacts/org.glassfish.jersey.bundles/jaxrs-ri/2.27/source-code>

3) Copy following jars from downloaded hive jar.

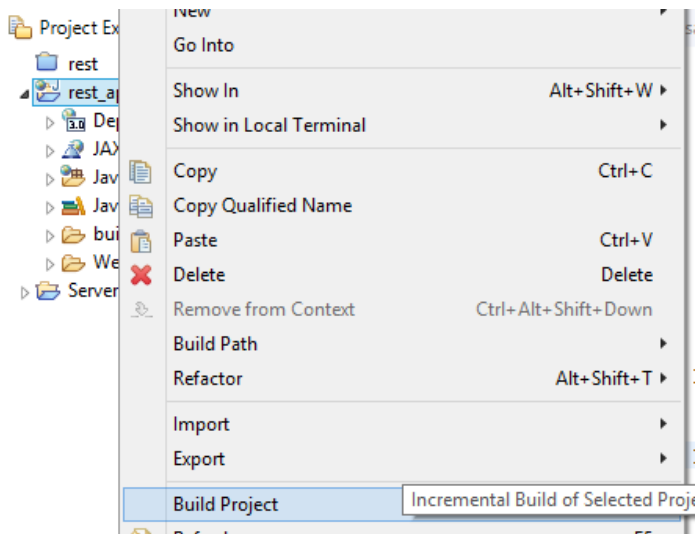
-  commons-logging-1.1.3
-  hadoop-common
-  hive-common-1.1.0-cdh5.7.0
-  hive-jdbc-1.1.0-cdh5.7.0-standalone
-  hive-metastore-1.1.0-cdh5.7.0
-  hive-service-1.1.0-cdh5.7.0
-  libthrift-0.9.2

4) Copy all Jar jaxrs jar and hive jar to WEB-INF lib folder.

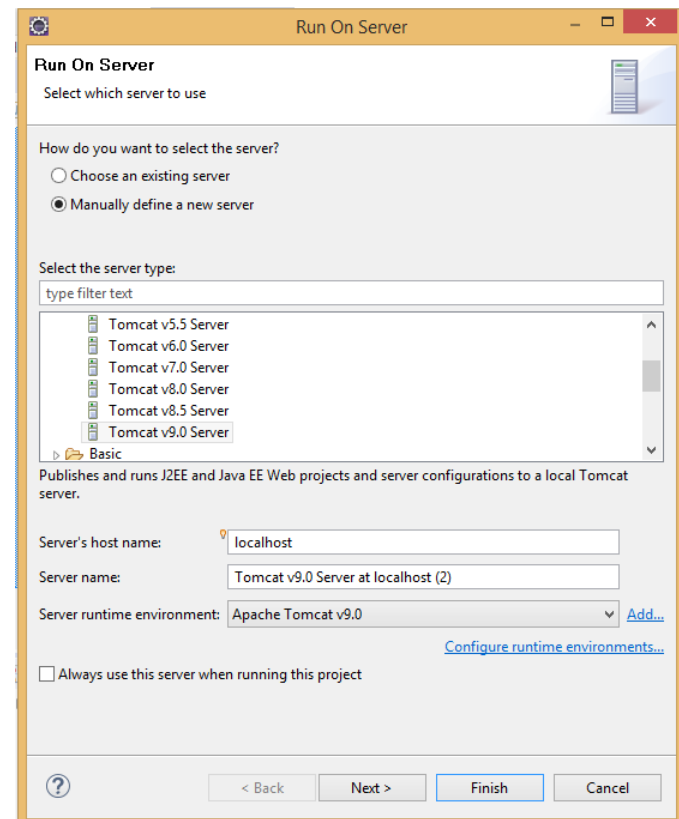


# REST API – Deployment

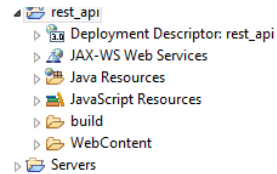
- 1) Right Click on project
- 2) Select Build project to create deployment war file



- 3) Right Click on project.
- 4) Click Run on server.
- 5) Configure Tomcat 9.0 server and run the project



# REST API - OUTPUT



```
3 import javax.ws.rs.*;
14
15 @Path("/")
16
17 public class HiveDataService {
18     @GET
19     @Path("/HiveData/fetchData")
20     @Produces(MediaType.APPLICATION_JSON)
21
22     public Response fetchData() {
23         List<Message> messages = new ArrayList<>();
24         System.out.println("Getting Data from Hadoop using");
25
26         try {
27             String driverName = "org.apache.hive.jdbc.HiveJdbcDriver";
28             Class.forName(driverName);
29         } catch (ClassNotFoundException e) {
30
31             e.printStackTrace();
32             System.exit(1);
33         }
34         try {
35             Connection con = DriverManager.getConnection("jdbc:hive2://localhost:9000/default;ssl=off");
36
37             Statement stmt = con.createStatement();
38             String query = "SELECT * FROM Online_Retail WHERE UnitPrice>1000 AND Country = 'United Kingdom'";
39             ResultSet rs = stmt.executeQuery(query);
40             while(rs.next()) {
41                 Message msg = new Message(rs.getString("Country"), rs.getString("Description"), rs.getDouble("UnitPrice"), rs.getString("InvoiceDate"));
42                 messages.add(msg);
43             }
44             con.close();
45         } catch (SQLException e) {
46             e.printStackTrace();
47             System.exit(1);
48         }
49         return Response.ok(messages).build();
50     }
51 }
```

Tomcat v9.0 Server at localhost [Apache Tomcat] C:\Java\jdk1.8.0\_101\bin\javaw.exe (23-Mar-2020, 12:55:59 pm)

```
Mar 23, 2020 1:11:38 PM org.apache.hive.jdbc.Utils parseURL
INFO: Supplied authorities: 192.168.99.100:10000
Mar 23, 2020 1:11:38 PM org.apache.hive.jdbc.Utils parseURL
INFO: Resolved authority: 192.168.99.100:10000
Running: SELECT * FROM Online_Retail WHERE UnitPrice>1000 AND Country = 'United Kingdom'
```

- 1) Web application would be deployed on server.
- 2) Open below url in browser [http://localhost:8080/rest\\_api/HiveData/PurchaseDetail](http://localhost:8080/rest_api/HiveData/PurchaseDetail)

```
[{"country": "United Kingdom", "description": "AMAZON FEE", "invoiceDate": "2010-12-07 15:04:00.0", "invoiceNo": "CS37630", "quantity": "-1", "stockCode": "AMAZONFEE", "unitPrice": "13541.33"}, {"country": "United Kingdom", "description": "AMAZON FEE", "invoiceDate": "2010-12-07 15:08:00.0", "invoiceNo": "537632", "quantity": "1", "stockCode": "AMAZONFEE", "unitPrice": "13541.33"}, {"country": "United Kingdom", "description": "AMAZON FEE", "invoiceDate": "2010-12-07 15:34:00.0", "invoiceNo": "CS37644", "quantity": "-1", "stockCode": "AMAZONFEE", "unitPrice": "5519.25"}, {"country": "United Kingdom", "description": "AMAZON FEE", "invoiceDate": "2010-12-07 15:41:00.0", "invoiceNo": "CS37647", "quantity": "-1", "stockCode": "AMAZONFEE", "unitPrice": "5519.25"}, {"country": "United Kingdom", "description": "AMAZON FEE", "invoiceDate": "2010-12-07 15:49:00.0", "invoiceNo": "CS37651", "quantity": "-1", "stockCode": "AMAZONFEE", "unitPrice": "16453.71"}, {"country": "United Kingdom", "description": "AMAZON FEE", "invoiceDate": "2010-12-13 17:14:00.0", "invoiceNo": "CS38682", "quantity": "-1", "stockCode": "M", "unitPrice": "1283.8"}, {"country": "United Kingdom", "description": "Manual", "invoiceDate": "2010-12-22 14:41:00.0", "invoiceNo": "539856", "quantity": "1", "stockCode": "M", "unitPrice": "16888.02"}, {"country": "United Kingdom", "description": "AMAZON FEE", "invoiceDate": "2011-01-05 09:57:00.0", "invoiceNo": "CS40118", "quantity": "-1", "stockCode": "AMAZONFEE", "unitPrice": "16453.71"}, {"country": "United Kingdom", "description": "Manual", "invoiceDate": "2011-01-20 11:48:00.0", "invoiceNo": "CS41651", "quantity": "-1", "stockCode": "M", "unitPrice": "1283.8"}, {"country": "United Kingdom", "description": "Bank Charges", "invoiceDate": "2011-01-20 11:50:00.0", "invoiceNo": "CS41653", "quantity": "1", "stockCode": "BANK CHARGES", "unitPrice": "1486.12"}, {"country": "United Kingdom", "description": "Manual", "invoiceDate": "2011-02-15 12:36:00.0", "invoiceNo": "CS44047", "quantity": "-1", "stockCode": "M", "unitPrice": "5575.28"}, {"country": "United Kingdom", "description": "AMAZON FEE", "invoiceDate": "2011-02-21 15:11:00.0", "invoiceNo": "CS44589", "quantity": "-1", "stockCode": "AMAZONFEE", "unitPrice": "5258.77"}, {"country": "United Kingdom", "description": "AMAZON FEE", "invoiceDate": "2011-03-18 12:56:00.0", "invoiceNo": "CS46987", "quantity": "-1", "stockCode": "AMAZONFEE", "unitPrice": "5258.77"}, {"country": "United Kingdom", "description": "AMAZON FEE", "invoiceDate": "2011-03-18 12:59:00.0", "invoiceNo": "CS46989", "quantity": "-1", "stockCode": "AMAZONFEE", "unitPrice": "5258.77"}, {"country": "United Kingdom", "description": "AMAZON FEE", "invoiceDate": "2011-03-18 12:59:00.0", "invoiceNo": "CS46989", "quantity": "-1", "stockCode": "AMAZONFEE", "unitPrice": "5258.77"}, {"country": "United Kingdom", "description": "Manual", "invoiceDate": "2011-03-28 11:51:00.0", "invoiceNo": "CS47899", "quantity": "-1", "stockCode": "M", "unitPrice": "1486.12"}, {"country": "United Kingdom", "description": "Manual", "invoiceDate": "2011-03-28 11:51:00.0", "invoiceNo": "CS47899", "quantity": "-1", "stockCode": "M", "unitPrice": "1486.12"}, {"country": "United Kingdom", "description": "Manual", "invoiceDate": "2011-03-28 11:51:00.0", "invoiceNo": "CS47899", "quantity": "-1", "stockCode": "M", "unitPrice": "1486.12"}]
```



THANK YOU