POLITECNICO DI MILANO

POLO REGIONALE DI COMO
Master of Science in Computer Engineering

Design and Implementation of a Graph-based Database Using HyperGraphDB

Mahsa Teimourikia

1. Introduction

This report is prepared for database systems II project assignment. The technology we used for this project is called HyperGraphDB which is an open source graph-based database.
For working with the classes of HyperGraphDB there is a need to have Java 5+ and an IDE. There is a IDE designed on HyperGraphDB called Seco which is a software scripting, prototyping environment for the Java platform. However because it works with beansell scripting language we used Eclipse instead to be able to work with Java programming language.

A graph database is a database that uses graph structures with nodes, edges, and properties to represent and store information. Nodes represent entities such as people, businesses, accounts, or any other item you might want to keep track of. Edges are the lines that connect nodes to nodes or nodes to properties and they represent the relationship between the two. Most of the important information is really stored in the edges. Meaningful patterns emerge when one examines the connections and interconnections of nodes, properties, and edges [6].

HyperGraphDB is a general purpose, extensible, portable, distributed, embeddable, open-source data storage mechanism. It is a graph database designed specifically for artificial intelligence and semantic web projects, it can also be used as an embedded object-oriented database for projects of all sizes[1].

2. Assumptions

We selected graph-databases as our database technology. And we chose these two requirements to implement:

- Management of the chain of resellers, customers affiliated to the loyalty program (through the resellers), relations among resellers, and so on
- Sales statistics (by date, period, location, user segment,)

For implementing a sample database we assumed to have a big chain supermarket with headquarters in Milano, Roma, Verona and Genova. There are two Suppliers who provide the products of these supermarkets. The main marketing division which is located in Milano decides about sending products to each headquarter. Headquarters keep the received products in their warehouses and then they distribute these products between different resellers.
Also some resellers provide the products of other resellers. It means that resellers can get the products from headquarter warehouses or from other resellers.

We assumed to have 4 product groups:

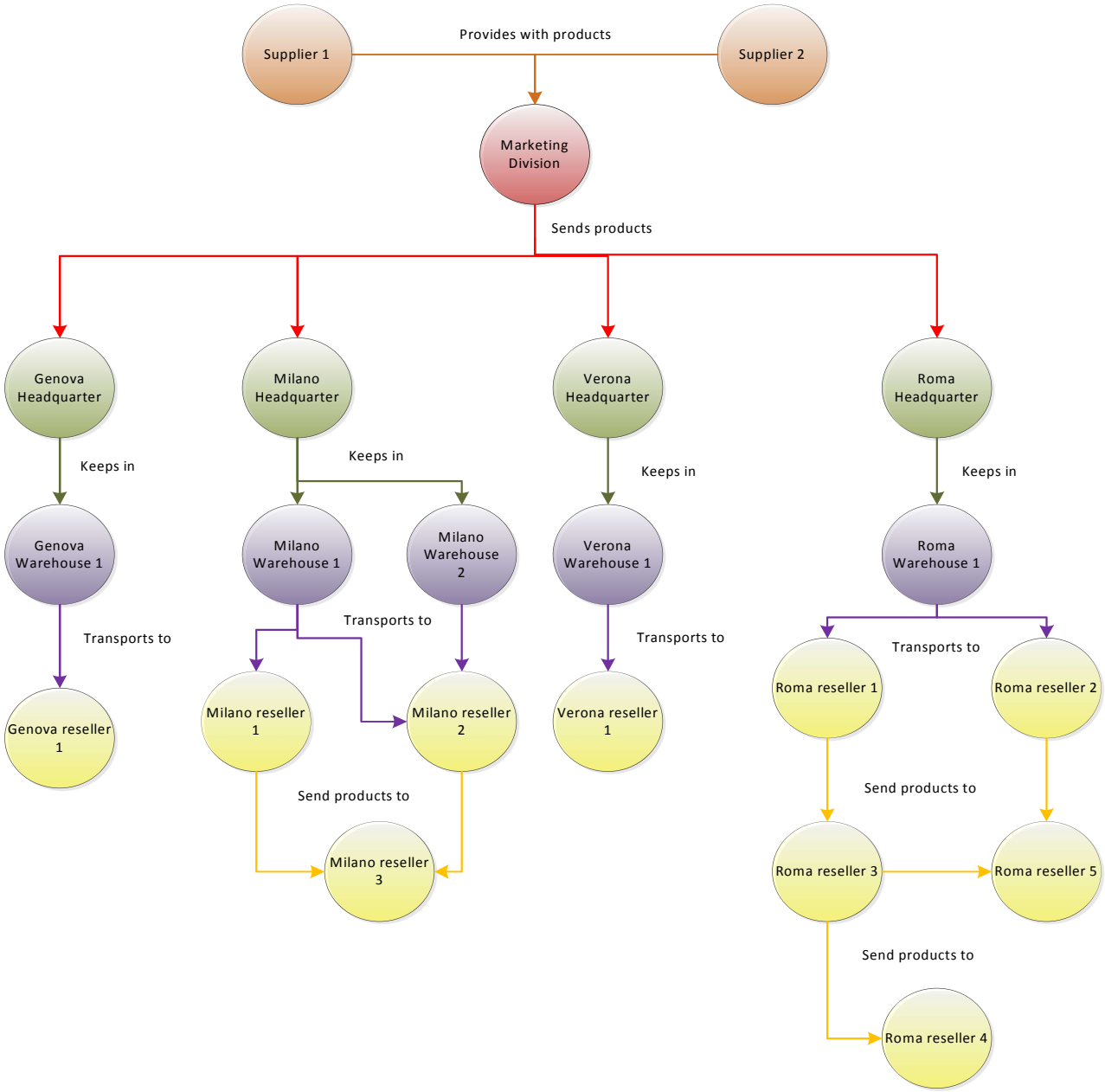
- Product group 1: "Food", "Liquor", "Animal Food"
- Product group 2: "Notebook", "Book", "Writing Accessories"
- Product group 3: "Cosmetics", "Shampoo", "Cream", "Lotion"
- Product group 4: "Utensils", "Detergent", "Maintenance Supplies".

Also we have considered different groups for customers.

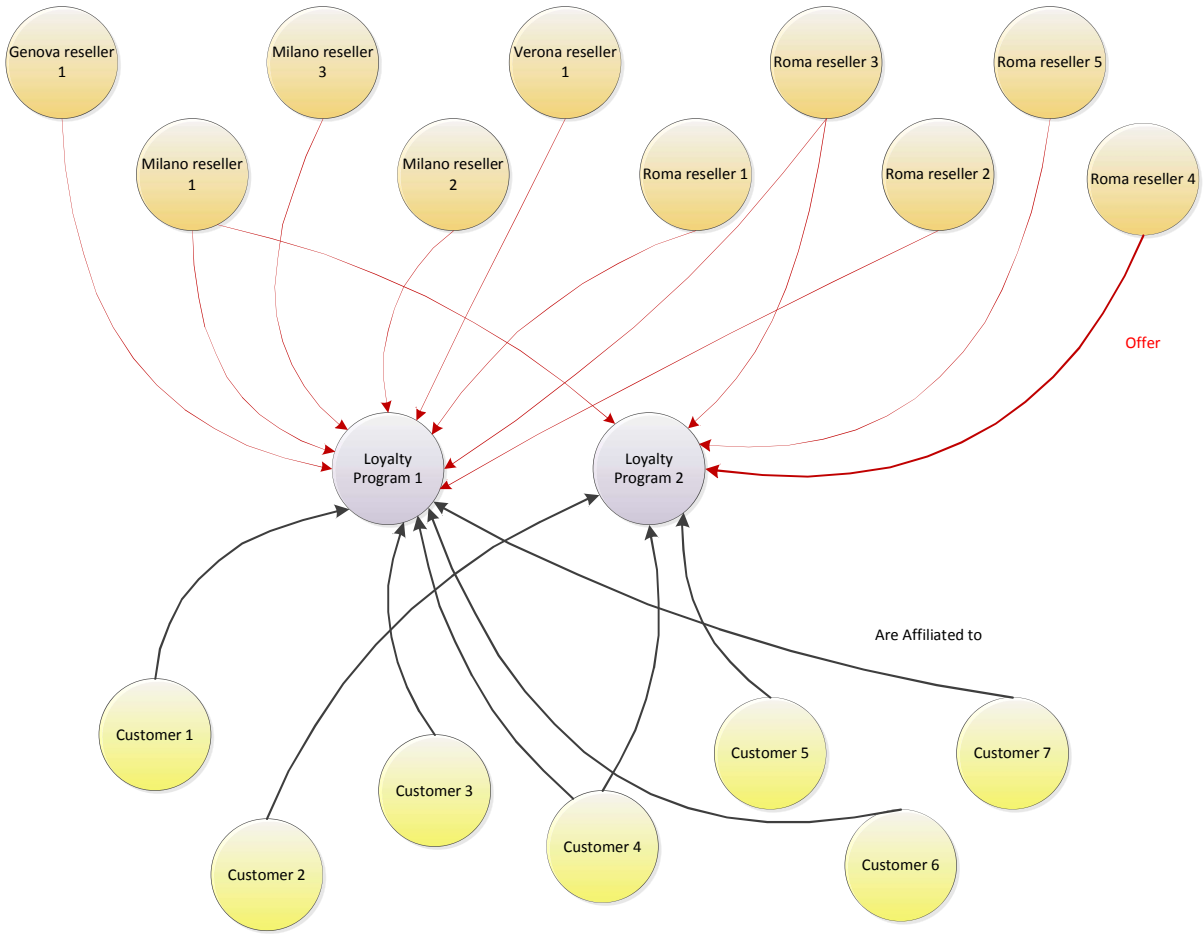
3. Analysis and Design

For making the graphs easier to understand we designed each part separately. However, in the actual implementation we merged them together.

- **Graph design for management of resellers' chain and relations among resellers:**

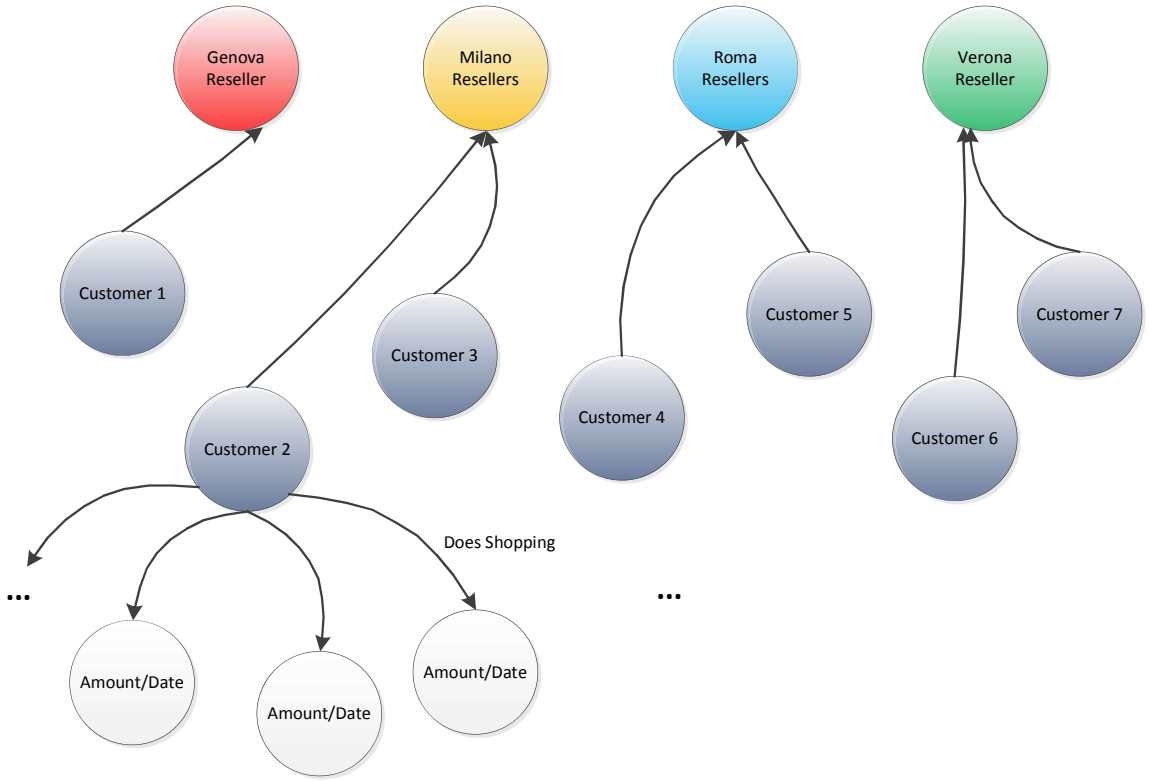


- **Graph design for customers affiliated to loyalty programs through resellers:**



- **Graph design for sales statistics:**

This graph is simplified in order the make it more understandable. Each reseller sells some of the product groups. In this graph we grouped the resellers upon location to have a less crowded graph. Also for each customer who does the shopping a node will be added to the graph which shows the date of shopping and the amount of payment for a product group. These nodes will be later used for Sales Statistics and to find the customers who are affiliated to a loyalty program.



- ## Implementations of nodes and relationships:

In HyperGraphDB the nodes and relationships are called *atoms.* Each atom is a Java object containing the properties and values of each node or relationship. Therefore we implemented each node as a Java class. Below there is an example of these classes:

```
public class Supplier {
	private String name;
	private String location;

	Supplier(){
		this.name = "";
		this.location = "";}

	Supplier(String name, String location){
		this.name = name;
		this.location = location;}

	public String getName(){return name;}
	public String getLocation(){return location;}
	public void setName(String name){this.name = name;}
	public void setLocation(String location){this.location = location;}
}
```

Creating the graph database and adding nodes:

```
String location = "D:/Documents/Polimi/Database Systems 2/Project/Project";
HyperGraph resellerChainGraph = HGEnvironment.get(location);
```

```
Supplier Supplier1 = new Supplier();
HGHandle SupplierHandle = resellerChainGraph.add(Supplier1);
```

Defining relationships between nodes:

```
HGValueLink Supplier_MarketingDivision = new HGValueLink("Provides With
Products", SupplierHandle2, MarketingDivisionHandel1);
```

Updating database entries:

```
Supplier1.setName("New Name");
Supplier1.setLocation("New Location");
resellerChainGraph.update(Supplier1);
```

Deleting database entries:

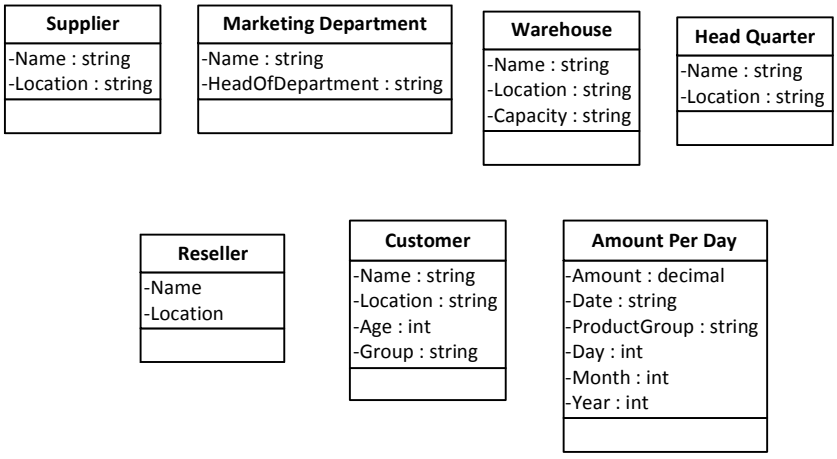
```
resellerChainGraph.remove(SupplierHandle);
```

Getting Queries:

```
HGQueryCondition condition = new And(
 new AtomTypeCondition(Supplier.class),
 new AtomPartCondition(new String[] {"name"}, "Supp1", ComparisonOperator.EQ)
);
HGSearchResult rs = resellerChainGraph.find(condition);
while (rs.hasNext()) {
      HGHandle current = (HGHandle) rs.next();
      Supplier test = resellerChainGraph.get(current);
      System.out.println(test.getName());
      System.out.println(test.getLocation());
}
```

- ## Class diagrams:

For making classes simpler we only show here classes and their attributes. We do not mention class methods and operations here.

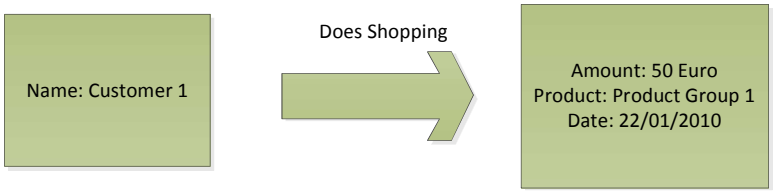


- ## Update Queries:

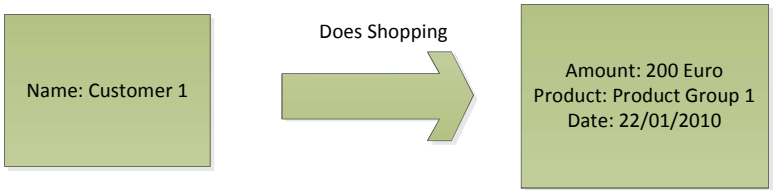
1. Update the graph-database and set new values for *Amount of purchase* of products of *Product Group 1*, by *Customer 1* on *22/01/2010*.

```
//First we should find the Customer 1
HGQueryCondition FindCustomer1 =
   new And( new AtomTypeCondition(Customer.class),
            new AtomPartCondition(new String[]{"name"}, "Customer 1",
ComparisonOperator.EQ));

HGSearchResult rs1 = resellerChainGraph.find(FindCustomer1);
Customer C1 = new Customer();
while (rs1.hasNext()) {
      HGHandle current = (HGHandle) rs1.next();
      C1 = resellerChainGraph.get(current);}
rs1.close();
//Then We have to find how much this customer paid on 22/01/2010 for ProductGroup1
AmountPerDay customerPaymentOfDay = new AmountPerDay();
List links =(List)hg.getAll(resellerChainGraph,
       hg.incident(resellerChainGraph.getHandle(C1)));

for(int i=0; i<links.size(); i++){
   HGValueLink l1 = (HGValueLink)links.get(i);
   HGQueryCondition FindCustomerPaymentsOfDay = new And(
      new AtomTypeCondition(AmountPerDay.class),
      new AtomPartCondition(new String[]{"date"}, "22/01/2010",
ComparisonOperator.EQ),
      new AtomPartCondition(new String[]{"productGroup"}, "Product Group 1",
ComparisonOperator.EQ),
      hg.target(resellerChainGraph.getHandle(l1)));
   HGSearchResult rs3 = resellerChainGraph.find(FindCustomerPaymentsOfDay);
   while (rs3.hasNext()) {
      HGHandle current = (HGHandle) rs3.next();
      AmountPerDay x = resellerChainGraph.get(current);
      if(x != null){ customerPaymentOfDay = x;}
   }
}
//Now we Update the amount of payment
customerPaymentOfDay.setAmount(200);
resellerChainGraph.update(customerPaymentOfDay);
```

- ### Before Update:



- **After Update:**



2. Update the graph database and Group for customers older than 30 who had purchased product group 3 in Milano, set them as Group 10.

```
HGQueryCondition FindCustomers30Milano = new And(
      new AtomTypeCondition(Customer.class),
      new AtomPartCondition(new String[]{"age"}, 30, ComparisonOperator.GTE),
      new AtomPartCondition(new String[]{"location"}, "Milano",
ComparisonOperator.EQ)
                  );
      HGSearchResult rs12 = resellerChainGraph.find(FindCustomers30Milano);
      ArrayList Customers30Milano = new ArrayList<Customer>();
      while (rs12.hasNext()) {
            HGHandle current = (HGHandle) rs12.next();
            Customers30Milano.add((Customer)
resellerChainGraph.get(current));
      }
      //For each customer who is younger than 25 years old:
      for(int i=0; i<Customers30Milano.size(); i++)
      {
          Customer TempCustomer30Milano = (Customer)Customers30Milano.get(i);
          List allLinksCustomers30Milano =
(List)hg.getAll(resellerChainGraph,

      hg.incident(resellerChainGraph.getHandle(Customers30Milano.get(i))));
         //List of payments for the current customer
         ArrayList Customer30MilanoPayments = new ArrayList<AmountPerDay>();
         for(int j=0; j<allLinksCustomers30Milano.size(); j++){

             HGValueLink LinkCustomer25 =
(HGValueLink)allLinksCustomers30Milano.get(j);

             if(LinkCustomer25.getValue() == "Does Shopping"){
             HGQueryCondition PaymentsOfCustomer25Query = new And(
                  new AtomTypeCondition(AmountPerDay.class),
                  new AtomPartCondition(new String[]{"productGroup"},
"Product Group 3", ComparisonOperator.EQ),

      hg.target(resellerChainGraph.getHandle(LinkCustomer25)));
             HGSearchResult rs13 =
resellerChainGraph.find(PaymentsOfCustomer25Query);
             while(rs13.hasNext())
             {
                  HGHandle current = (HGHandle)rs13.next();

      Customer30MilanoPayments.add((AmountPerDay)resellerChainGraph.get(curre
nt));
             }
         }
```

```
          if(Customer30MilanoPayments.size()>0){
      TempCustomer30Milano.setGroup("Group 10");
            resellerChainGraph.update(TempCustomer30Milano);
                              }
            }
      }
      HGQueryCondition FindCustomersGroup10 = new And(
            new AtomTypeCondition(Customer.class),
            new AtomPartCondition(new String[]{"group"}, "Group 10",
ComparisonOperator.EQ));
            HGSearchResult rs14 =
resellerChainGraph.find(FindCustomers30Milano);
            while (rs14.hasNext()) {
                  HGHandle current = (HGHandle) rs14.next();
                  Customer temp1 = (Customer)
resellerChainGraph.get(current);
                  System.out.print("Customer Name: "+ temp1.getName()+ " ,
location: " + temp1.getLocation());
                  System.out.println(" , Group: "+ temp1.getGroup()+ " , Age:
"+ temp1.getAge());
}
```

- **The Output:**

```
Customer Name: Customer 3 , location: Milano , Group: Group 10 , Age: 33
Customer Name: Customer 12 , location: Milano , Group: Group 10 , Age: 38
Customer Name: Customer 2 , location: Milano , Group: Group 10 , Age: 30
```

- **Queries:**

1. Find the list of customers who were affiliated to the loyalty program 2 on January:

```
//1- Find the list of all customers
List AllCustomers = (List)hg.getAll(resellerChainGraph,new
AtomTypeCondition(Customer.class));

Customer C2 = new Customer();
HGValueLink l2 = new HGValueLink();

//2- For each Customer find the list of payments that customer has //spent on
Jan
for(int i=0; i < AllCustomers.size(); i++){
   double sumOfPayment = 0;
   C2 = (Customer)AllCustomers.get(i);
   List links =(List)hg.getAll(resellerChainGraph,
                  hg.incident(resellerChainGraph.getHandle(C2)));
   for(int j = 0; j < links.size(); j++){
      l2 = (HGValueLink)links.get(j);
```

```
        HGQueryCondition CustomerPayments = new And(
        new AtomTypeCondition(AmountPerDay.class),
        new AtomPartCondition(new String[]{"month"}, 1, ComparisonOperator.EQ),
        hg.target(resellerChainGraph.getHandle(l2)));

        HGSearchResult rs4 = resellerChainGraph.find(CustomerPayments);
        while(rs4.hasNext())
        {
              HGHandle current = (HGHandle)rs4.next();
              AmountPerDay x1 = resellerChainGraph.get(current);
              if(x1 != null){
                    sumOfPayment += x1.getAmount();
              }
        }
    }
//The amount of payment in one month should be more than 200 euro for
//customers to be considered as affiliated to the loyalty program 2
    if(sumOfPayment>=200.0){
        System.out.print(C2.getName());
        System.out.print(" , ");
        System.out.println(sumOfPayment);
        System.out.println("----------------------------------------");
     }
}
```

- **The output:**

```
Customer Name,  Amount of Payment on January

Customer 14 ,   1390.0
------------------------------------------------
Customer 1  ,   660.0
------------------------------------------------
Customer 10 ,   310.0
------------------------------------------------
Customer 3  ,   450.0
------------------------------------------------
Customer 9  ,   960.0
------------------------------------------------
Customer 13 ,   430.0
------------------------------------------------
Customer 11 ,   730.0
------------------------------------------------
Customer 12 ,   570.0
------------------------------------------------
Customer 8  ,   330.0
------------------------------------------------
Customer 6  ,   860.0
------------------------------------------------
```

2. Find the resellers who send products to the other resellers and are located in Milano, find the number of other resellers they send product to and list them:

```
HGQueryCondition AllRomaResellersQuery = new And(
   new AtomTypeCondition(Reseller.class),
   new AtomPartCondition(new String[]{"location"}, "Roma",
ComparisonOperator.EQ)
                  );
List AllRomaResellers = (List)hg.getAll(resellerChainGraph,
AllRomaResellersQuery);
//For Each Reseller
for(int i=0; i<AllRomaResellers.size(); i++)
{
    Reseller r1 = (Reseller)AllRomaResellers.get(i);
    Set ResellerLinks =
(Set)resellerChainGraph.getIncidenceSet(resellerChainGraph.getHandle(r1));
    Iterator linksIt = ResellerLinks.iterator();
    int arity = 0;
    List resellerList = null;
    while(linksIt.hasNext())
    {
    HGHandle RHandle = (HGHandle)linksIt.next();
    HGValueLink x1 = resellerChainGraph.get(RHandle);
    if(x1.getValue()=="Send Products To")
    {
      arity++;
      HGQueryCondition FindAllTargetsQuery = new And(
      new AtomTypeCondition(Reseller.class),
      new Not(new AtomPartCondition(new String[] {"name"}, r1.getName(),
ComparisonOperator.EQ)),

      hg.target(resellerChainGraph.getHandle(x1)));
      HGSearchResult rs5 = resellerChainGraph.find(FindAllTargetsQuery);
      while(rs5.hasNext())
      {
            resellerList.add(rs5.next());
      }

      rs5.close();
   }
}
```

- **The Output:**

```
Reseller Name: Roma Reseller 5 , Roma
Arity: 2
List of resellers who sends products to:
Roma Reseller 3 , Roma ,
Roma Reseller 2 , Roma ,
-----------------------------------------
Reseller Name: Roma Reseller 1 , Roma
Arity: 1
List of resellers who sends products to:
```

```
Roma Reseller 3 , Roma ,
----------------------------------------
Reseller Name: Roma Reseller 4 , Roma
Arity: 1
List of resellers who sends products to:
Roma Reseller 3 , Roma ,
----------------------------------------
Reseller Name: Roma Reseller 2 , Roma
Arity: 1
List of resellers who sends products to:
Roma Reseller 5 , Roma ,
----------------------------------------
Reseller Name: Roma Reseller 3 , Roma
Arity: 3
List of resellers who sends products to:
Roma Reseller 5 , Roma ,
Roma Reseller 1 , Roma ,
Roma Reseller 4 , Roma ,
----------------------------------------
```

3. Sales Statistics: Find the sales of product group 1 on January 2010 which were bought by customers of group 1 who are older than 22 years old.

```
HGQueryCondition FindCustomerOfGroup1 = new And(
      new AtomTypeCondition(Customer.class),
      new AtomPartCondition(new String[]{"group"}, "Group 1",
ComparisonOperator.EQ),
      new AtomPartCondition(new String[]{"age"}, 22,
ComparisonOperator.GTE));
HGSearchResult rs6 = resellerChainGraph.find(FindCustomerOfGroup1);
ArrayList CustomerGroup1 = new ArrayList<Customer>();
while (rs6.hasNext()) {
      HGHandle current = (HGHandle) rs6.next();
      CustomerGroup1.add((Customer) resellerChainGraph.get(current));
                  }
//For each customer of group1 above the 22 years old:
      for(int i=0; i<CustomerGroup1.size(); i++)
      {  List AllLinksCustomerGroup1 = (List)hg.getAll(resellerChainGraph,
hg.incident(resellerChainGraph.getHandle(CustomerGroup1.get(i))));
      //List of payments for the current customer
      ArrayList CustomerGroup1Payments = new ArrayList<AmountPerDay>();

      for(int j=0; j<AllLinksCustomerGroup1.size(); j++){

            HGValueLink LinkCustomerGroup1 =
(HGValueLink)AllLinksCustomerGroup1.get(j);

            if(LinkCustomerGroup1.getValue() == "Does Shopping"){
            HGQueryCondition PaymentsOfCustomerGroup1Query = new And(
               new AtomTypeCondition(AmountPerDay.class),
               new AtomPartCondition(new String[]{"month"}, 1,
ComparisonOperator.EQ),
               new AtomPartCondition(new String[]{"productGroup"}, "Product
```

```
Group 1", ComparisonOperator.EQ),

hg.target(resellerChainGraph.getHandle(LinkCustomerGroup1)));

            HGSearchResult rs7 =
resellerChainGraph.find(PaymentsOfCustomerGroup1Query);
            while(rs7.hasNext())
            {
               HGHandle current = (HGHandle)rs7.next();

CustomerGroup1Payments.add((AmountPerDay)resellerChainGraph.get(current));

            }
      }
   }
}
```

- **The Output:**

```
Customer Name: Customer 13 , location: Roma , Group: Group 1 , Age: 27
Payments:
Amount: 240.0 , Product Group: Product Group 1 , Date: 15/01/2010
Payments:
Amount: 90.0 , Product Group: Product Group 1 , Date: 09/01/2010
Payments:
Amount: 50.0 , Product Group: Product Group 1 , Date: 22/01/2010
-------------------------------------------------
Customer Name: Customer 11 , location: Verona , Group: Group 1 , Age: 26
Payments:
Amount: 240.0 , Product Group: Product Group 1 , Date: 15/01/2010
Payments:
Amount: 50.0 , Product Group: Product Group 1 , Date: 22/01/2010
Payments:
Amount: 90.0 , Product Group: Product Group 1 , Date: 09/01/2010
-------------------------------------------------
Customer Name: Customer 1 , location: Genova , Group: Group 1 , Age: 24
Payments:
Amount: 50.0 , Product Group: Product Group 1 , Date: 22/01/2010
Payments:
Amount: 250.0 , Product Group: Product Group 1 , Date: 13/01/2010
Payments:
Amount: 90.0 , Product Group: Product Group 1 , Date: 09/01/2010
-------------------------------------------------
Customer Name: Customer 2 , location: Milano , Group: Group 1 , Age: 30
-------------------------------------------------
Customer Name: Customer 12 , location: Milano , Group: Group 1 , Age: 38
Payments:
Amount: 50.0 , Product Group: Product Group 1 , Date: 22/01/2010
Payments:
Amount: 180.0 , Product Group: Product Group 1 , Date: 06/01/2010
Payments:
Amount: 90.0 , Product Group: Product Group 1 , Date: 09/01/2010
-------------------------------------------------
Customer Name: Customer 10 , location: Roma , Group: Group 1 , Age: 38
```

```
Payments:
Amount: 240.0 , Product Group: Product Group 1 , Date: 15/01/2010
-------------------------------------------------
```

4. Find the total sales of product group 1 in Milano in year 2010:

```
double TotalSales = 0.0;
HGQueryCondition FindMilanoCustomers = new And(
      new AtomTypeCondition(Customer.class),
      new AtomPartCondition(new String[]{"location"}, "Milano",
ComparisonOperator.EQ));
HGSearchResult rs8 = resellerChainGraph.find(FindMilanoCustomers);
ArrayList MilanoCustomers = new ArrayList<Customer>();

while (rs8.hasNext()) {
   HGHandle current = (HGHandle) rs8.next();
   MilanoCustomers.add((Customer) resellerChainGraph.get(current));
}
//For each customer in Milano:
for(int i=0; i<MilanoCustomers.size(); i++)
{
   List allLinksMilanoCustomers = (List)hg.getAll(resellerChainGraph,
   hg.incident(resellerChainGraph.getHandle(MilanoCustomers.get(i))));
   //List of payments for the current customer
   ArrayList MilanoCustomerPayments = new ArrayList<AmountPerDay>();

   for(int j=0; j<allLinksMilanoCustomers.size(); j++){

   HGValueLink LinkMilanoCustomer =
(HGValueLink)allLinksMilanoCustomers.get(j);

   if(LinkMilanoCustomer.getValue() == "Does Shopping"){
   HGQueryCondition PaymentsOfMilanoCustomerQuery = new And(
      new AtomTypeCondition(AmountPerDay.class),
      new AtomPartCondition(new String[]{"productGroup"}, "Product Group 1",
ComparisonOperator.EQ),

      hg.target(resellerChainGraph.getHandle(LinkMilanoCustomer)));
      HGSearchResult rs9 =
resellerChainGraph.find(PaymentsOfMilanoCustomerQuery);
      while(rs9.hasNext())
      {
         HGHandle current = (HGHandle)rs9.next();

      MilanoCustomerPayments.add((AmountPerDay)resellerChainGraph.get(current
));
      }
   }
}

      for(int z = 0; z< MilanoCustomerPayments.size(); z++)
      {
         AmountPerDay currentPayment1 = (AmountPerDay)
      MilanoCustomerPayments.get(z);
         TotalSales += currentPayment1.getAmount();
```

```
	}
}
System.out.println("Total Sales of Product Group 1 in year 2010 in Milano: "
+ TotalSales);
```

- **The Output:**

```
Total Sales of Product Group 1 in year 2010 in Milano: 3835.0 Euro
```

5. Find customers under the age 25 who have spent more than 200 Euro on Product Group1 in January 2010.

```
HGQueryCondition FindCustomers25 = new And(
	new AtomTypeCondition(Customer.class),
	new AtomPartCondition(new String[]{"age"}, 25,
ComparisonOperator.LTE));
	HGSearchResult rs10 = resellerChainGraph.find(FindCustomers25);
	ArrayList Customers25 = new ArrayList<Customer>();
	while (rs10.hasNext()) {
		HGHandle current = (HGHandle) rs10.next();
		Customers25.add((Customer) resellerChainGraph.get(current));
	}
	//For each customer who is younger than 25 years old:
	for(int i=0; i<Customers25.size(); i++)
	{
	   double TotalCustomer5Payments =0.0;
	   List allLinksCustomers25 = (List)hg.getAll(resellerChainGraph,
	  hg.incident(resellerChainGraph.getHandle(Customers25.get(i))));
	   //List of payments for the current customer
	   ArrayList Customer25Payments = new ArrayList<AmountPerDay>();
	   for(int j=0; j<allLinksCustomers25.size(); j++){

		HGValueLink LinkCustomer25 =
(HGValueLink)allLinksCustomers25.get(j);

		if(LinkCustomer25.getValue() == "Does Shopping"){
			HGQueryCondition PaymentsOfCustomer25Query = new And(
			new AtomTypeCondition(AmountPerDay.class),
			new AtomPartCondition(new String[]{"productGroup"},
"Product Group 1", ComparisonOperator.EQ),
			new AtomPartCondition(new String[]{"month"}, 1,
ComparisonOperator.EQ),

	hg.target(resellerChainGraph.getHandle(LinkCustomer25)));
	HGSearchResult rs11 =
resellerChainGraph.find(PaymentsOfCustomer25Query);
	while(rs11.hasNext())
	{
		HGHandle current = (HGHandle)rs11.next();

	Customer25Payments.add((AmountPerDay)resellerChainGraph.get(current));

	}
```

```
    }
}
for(int z = 0; z< Customer25Payments.size(); z++)
{
      AmountPerDay currentPayment1 = (AmountPerDay)
Customer25Payments.get(z);
      TotalCustomer5Payments += currentPayment1.getAmount();
}

if(TotalCustomer5Payments >= 200.0){
      Customer currentCustomer5 = (Customer)Customers25.get(i);
      System.out.print("Customer Name: "+ currentCustomer5.getName()+ " ,
location: " + currentCustomer5.getLocation());
      System.out.println(" , Group: "+ currentCustomer5.getGroup()+ " , Age:
"+ currentCustomer5.getAge());
      System.out.println("Total Payment For Product Group 1 on January 2010:
"+ TotalCustomer5Payments);
      System.out.println("---------------------------------");
      }
}
```

- **The Output:**

```
Customer Name: Customer 14 , location: Milano , Group: Group 2 , Age: 15
Total Payment For Product Group 1 on January 2010: 860.0
-----------------------------------------------------------
Customer Name: Customer 1 , location: Genova , Group: Group 1 , Age: 24
Total Payment For Product Group 1 on January 2010: 390.0
-----------------------------------------------------------
```

4. References:

1. HyperGraphDB Tutorials by Borislav Lordanov: http://hypergraphdb.org/learn
2. Eclipse Documentations: http://www.eclipse.org/resources/
3. Graph Databases, by Emil Eifrem: http://www.youtube.com/watch?v=2ElGO1P8v0c, http://www.infoq.com/presentations/emil-eifrem-neo4j
4. Customer Loyalty Programs, by Prof. Ran Kivetz: http://www.youtube.com/watch?v=HK-dDtRcO4M
5. Supply Chain Management, Arizona State University, W. P. Cary School of Bussiness: http://www.youtube.com/watch?v=Mi1QBxVjZAw
6. Graph databases: http://en.wikipedia.org/wiki/Graph_database

- **Application Download Links:**
  - HyperGraphDB version 1.1 for Windows: http://hypergraphdb.googlecode.com/files/hypergraphdb-1.1.zip

- Java 5+ : http://www.java.com/en/download/chrome.jsp?locale=en&host=www.java.com
- Eclipse IDE for Java Developers: http://www.eclipse.org/downloads/
- SWT Plugin for Eclipse: http://www.eclipse.org/swt/
- Seco: http://www.kobrix.com/seco.jsp