Design and Implementation of a Graph-based Database Using HyperGraphDB

Mahsa Teimourikia

Graph-Based databases

- A graph database is a database that uses graph structures with nodes, and edges 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 Edges to other Edges, 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, and edges [6].

Our Technology

- The technology we used for this project is called HyperGraphDB which is an open source graph-based database.

- IDEs for using HGDB:
  - There is a IDE designed on HyperGraphDB called Seco which is a software scripting, prototyping environment for the Java platform.
  - Eclipse.

HyperGraphDB

- HyperGraphDB is a general purpose, extensible, portable, distributed, embeddable, open-source data storage mechanism.
- It is written in Java
- 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.
  - a graph database.
  - a (non-SQL) relational database.

Project Assumptions

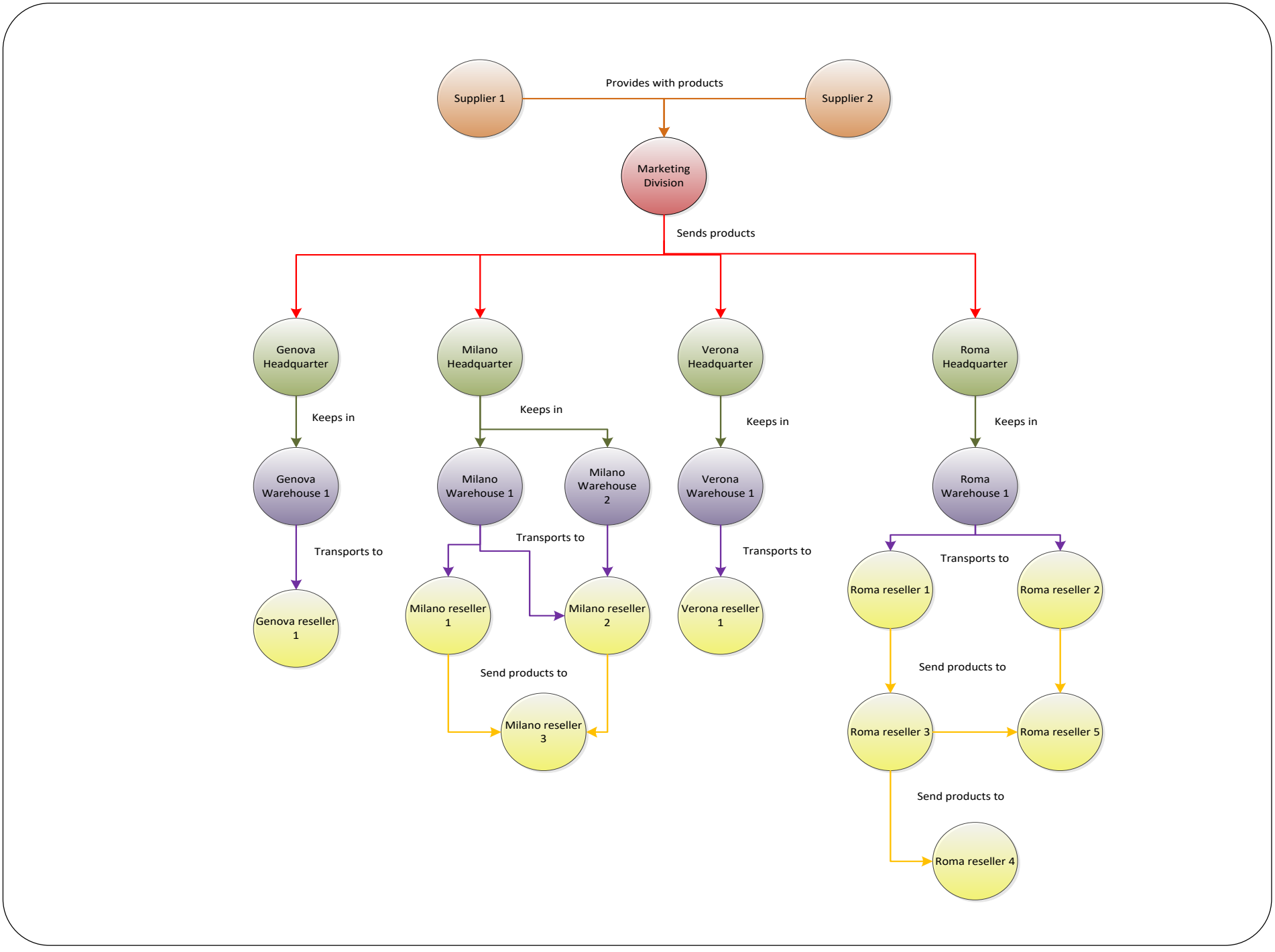
- We chose these two requirements to implement:
  - Management of the chain of resellers, customers affiliated to the loyalty program (through the resellers), relations among resellers, and so on
  - Sales statistics (by date, period, location, user segment,)

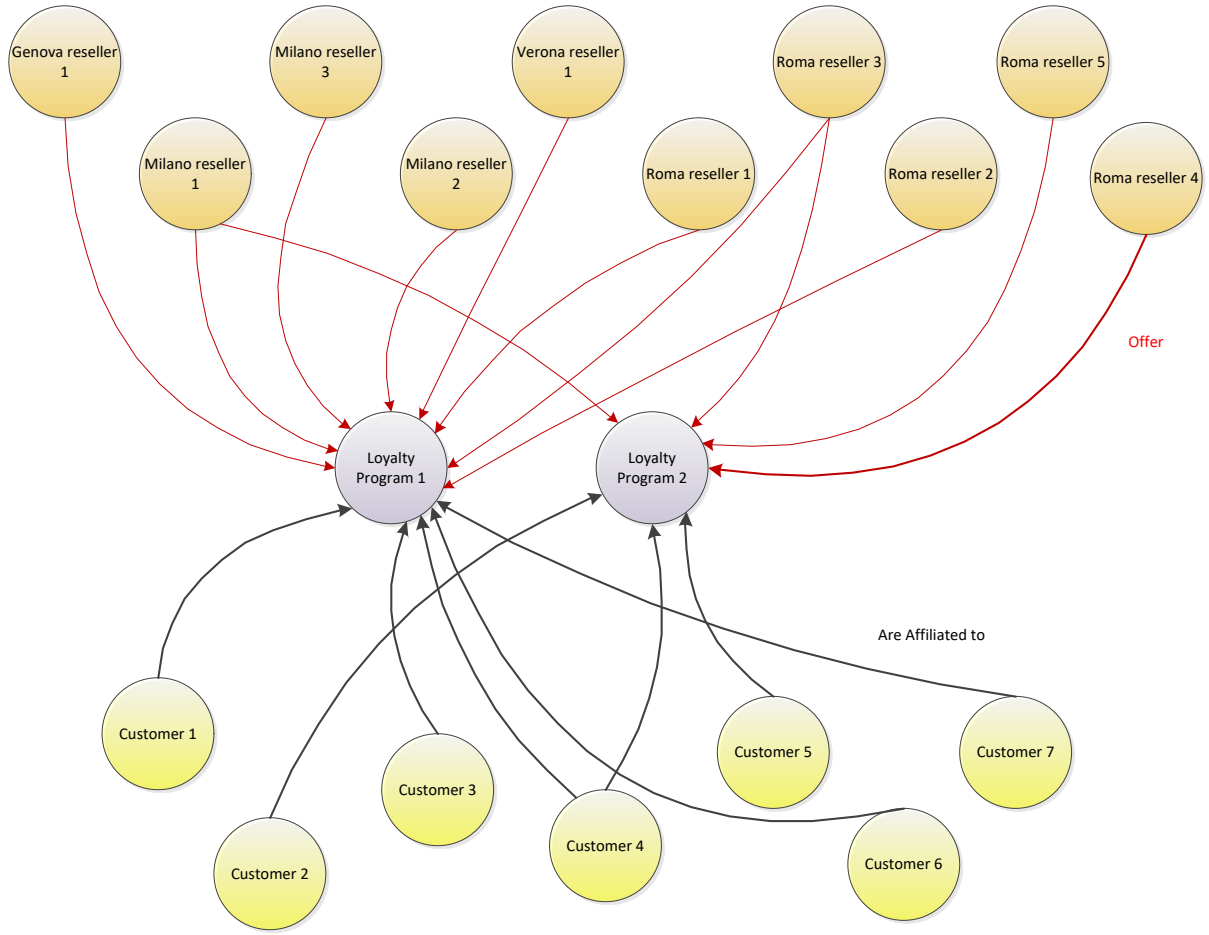
Assumptions

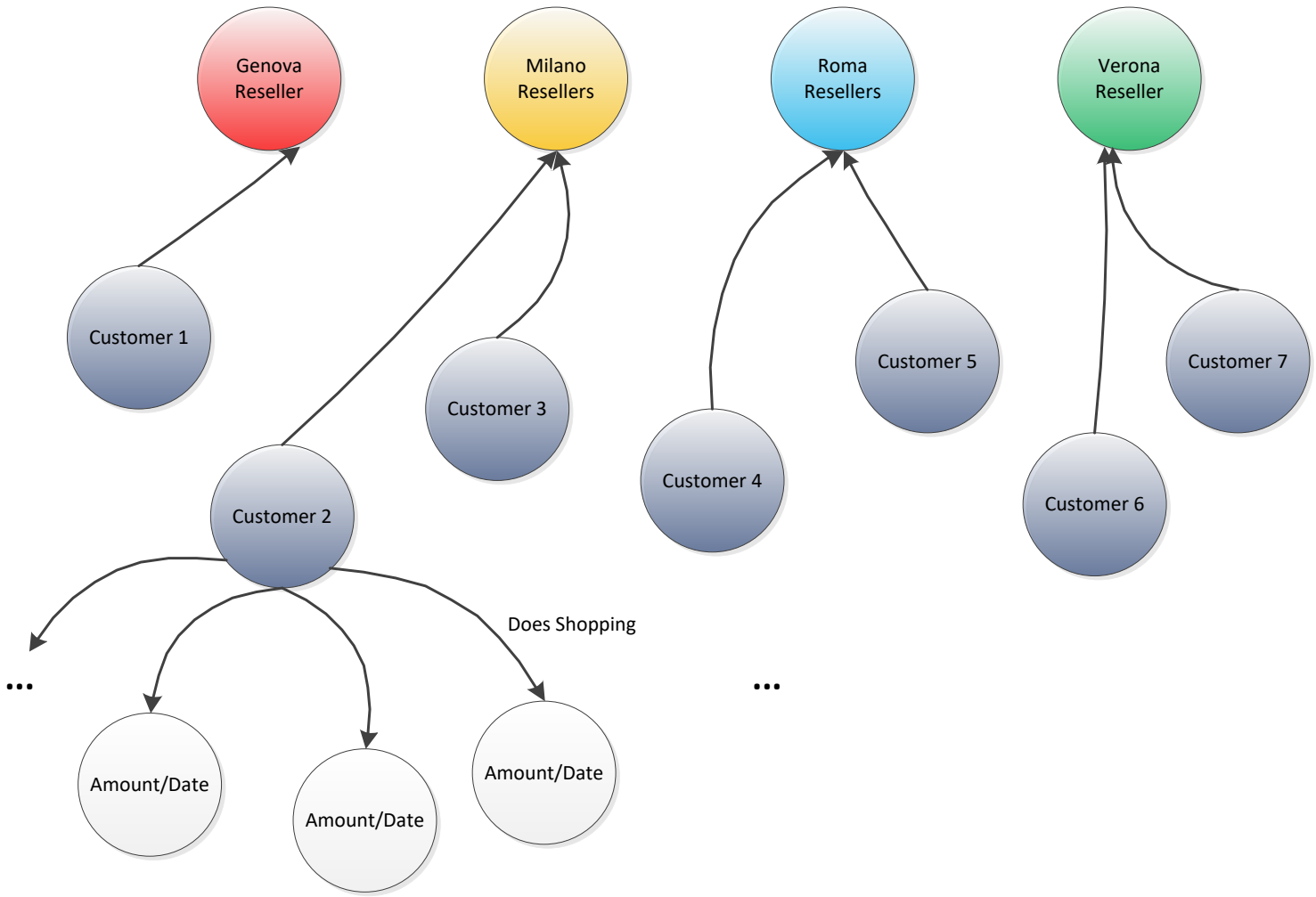
- We have a big chain supermarket with headquarters in Milano, Roma, Verona and Genova.
- There are two Suppliers who provide the products of these supermarkets.
- The main marketing division which is located in Milano decides about sending products to each headquarter.
- Headquarters keep the received 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.

Assumptions

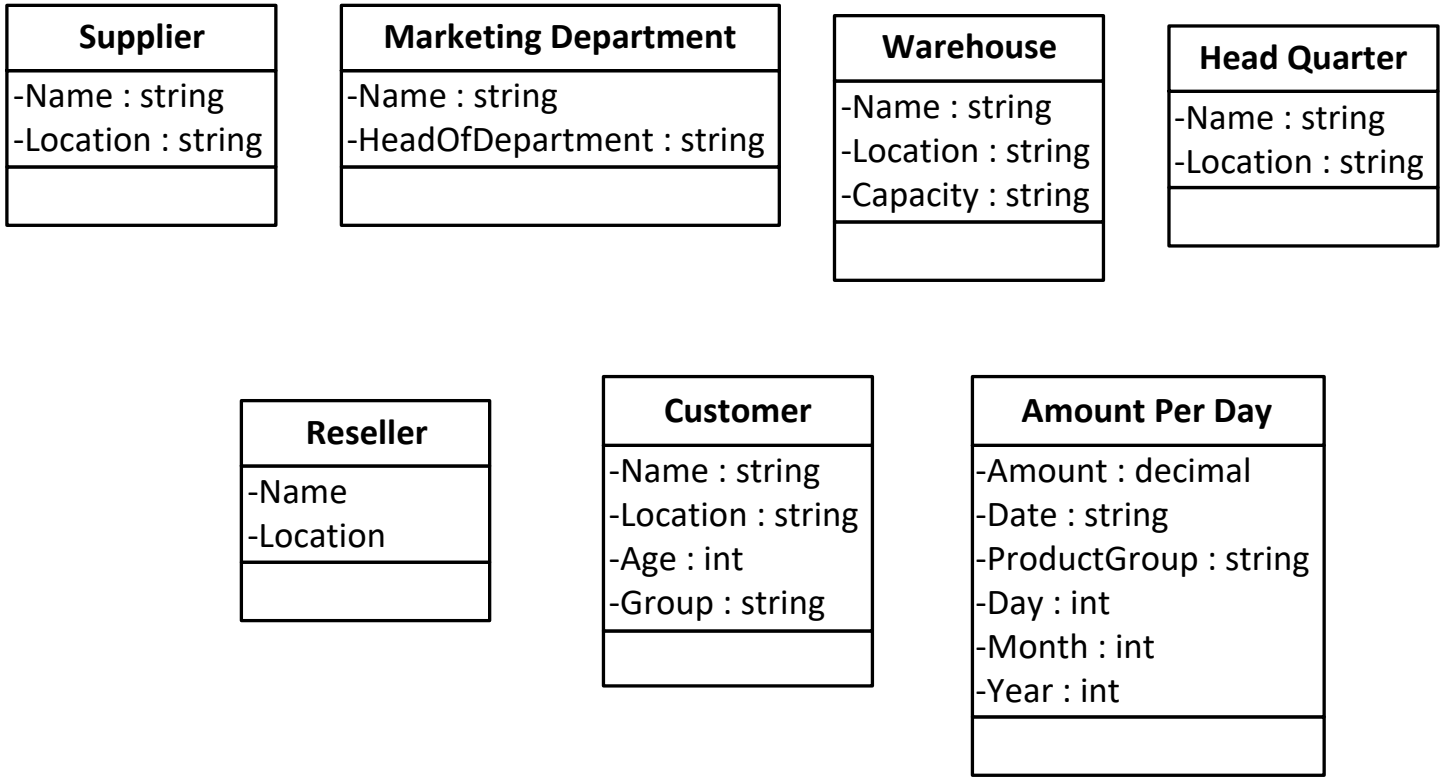
- We assumed to have 4 product groups:
- Product group 1: "Food", "Liquor", "Animal Food"
- Product group 2: "Notebook", "Book", "Writing Accessories"
- Product group 3: "Cosmetics", "Shampoo", "Cream", "Lotion"
- Product group 4: "Utensils", "Detergent", "Maintenance Supplies".
- Also we have considered different groups for customers.


Supplier 1
Provides with products
Supplier 2
Marketing Division
Sends products
Genova Headquarter
Keeps in
Genova Warehouse 1
Transports to
Genova reseller 1
Milano Headquarter
Keeps in
Milano Warehouse 1
Milano Warehouse 2
Transports to
Milano reseller 1
Milano reseller 2
Send products to
Milano reseller 3
Verona Headquarter
Keeps in
Verona Warehouse 1
Transports to
Verona reseller 1
Roma Headquarter
Keeps in
Roma Warehouse 1
Transports to
Roma reseller 1
Roma reseller 2
Send products to
Roma reseller 3
Roma reseller 5
Send products to
Roma reseller 4


Genova reseller 1
Milano reseller 3
Verona reseller 1
Roma reseller 3
Roma reseller 5
Milano reseller 1
Milano reseller 2
Roma reseller 1
Roma reseller 2
Roma reseller 4
Offer
Loyalty Program 1
Loyalty Program 2
Are Affiliated to
Customer 1
Customer 3
Customer 5
Customer 7
Customer 2
Customer 4
Customer 6


Genova Reseller
Milano Resellers
Roma Resellers
Verona Reseller
Customer 1
Customer 3
Customer 5
Customer 7
Customer 4
Customer 6
Customer 2
Does Shopping
...
...
Amount/Date
Amount/Date
Amount/Date

Class Diagrams



Implementing Nodes As Java 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:

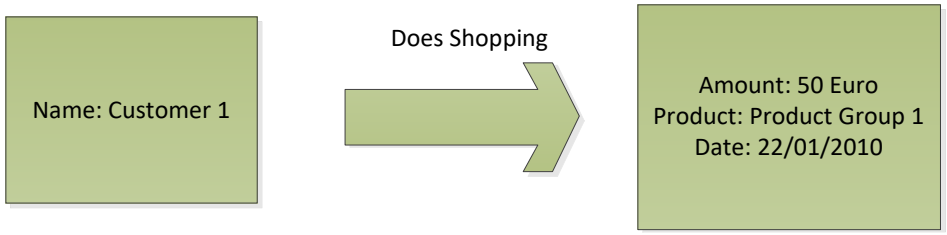
- resellerChainGraph.remove(SupplierHandle);

Getting Queries:

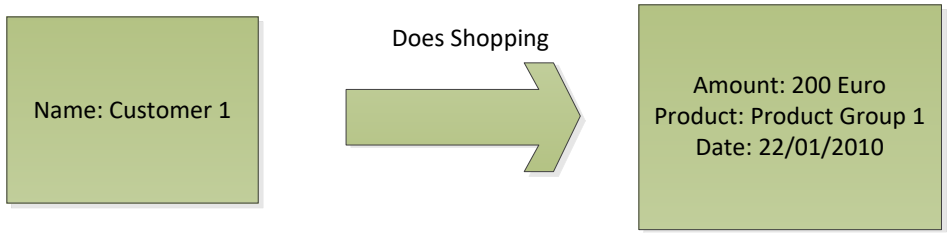
- HGQueryCondition condition = **new** And(
- **new** AtomTypeCondition(Supplier.**class**),
- **new** AtomPartCondition(**new** String[] {"name"}, "Supp1", ComparisonOperator.*EQ*)
-);
- HGSearchResult rs = resellerChainGraph.find(condition);
- **while** (rs.hasNext()) {
- HGHandle current = (HGHandle)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

Sample Update Queries:

- 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*.
- Before Update:



- **After Update:**



```
//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);
```

Sample Queries

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

- We have 2 Loyalty Programs that the resellers offer:
  - Loyalty Program 1: Customers that spent more than 100 euro per week in different resellers of the supermarket.
  - Loyalty Program 2: Customers that spent more than 200 euro per month in different resellers of the supermarket.

Query:

```
//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();
            }
            }
   }
```

Sample Query

- 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.

Query

```
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));
                                        }
                    }
  }
}
```

```
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
-------------------------------------------------
```

References:

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

THANKYOU FOR LISTENING