Introduction

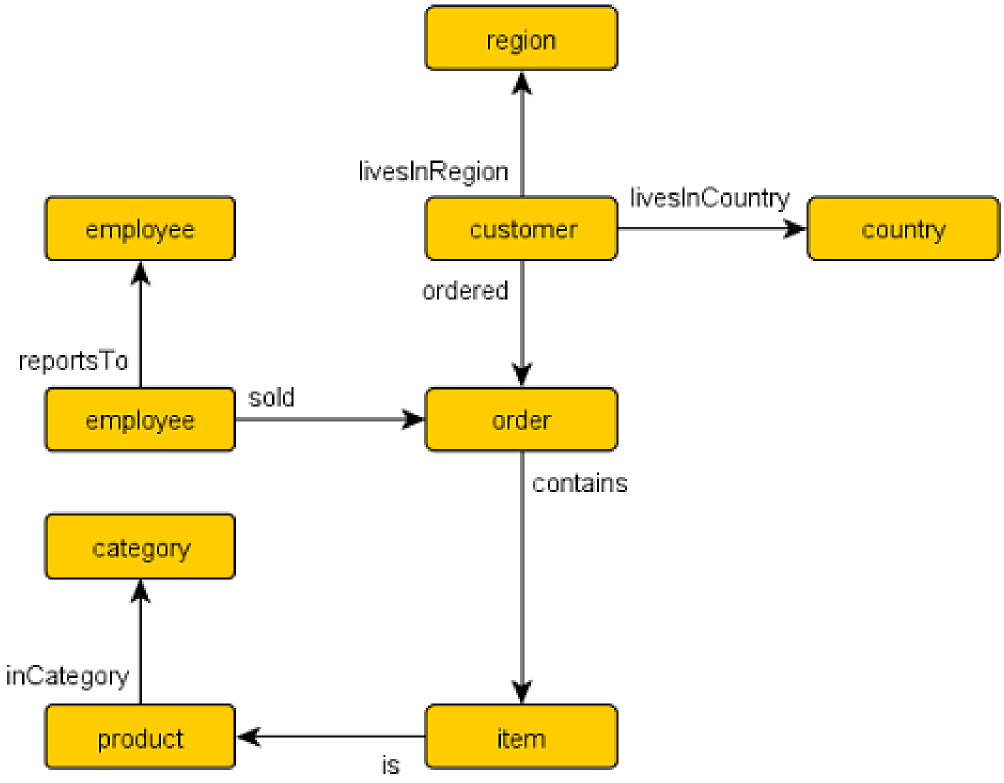
SQL2Gremlin teaches Apache TinkerPop's Gremlin graph traversal language using typical patterns found when querying data with SQL. The format of the Gremlin results will not necessarily match the format of the SQL results. While SQL can only provide results in a tabular form, Gremlin provides various ways to structure a result set. Next, the Gremlin queries demonstrated are for elucidatory purposes and may not be the optimal way to retrieve the desired data. If a particular query runs slow and an optimal solution is desired, please do not hesitate to ask for help on the Gremlin-users mailing list. Finally, the SQL examples presented make use of T-SQL syntax. MySQL users may not know some of the expressions (e.g. paging), but should be able to understand the purpose of the query.

If you would like to see other SQL2Gremlin translations using the Northwind dataset, please provide a ticket on the SQL2Gremlin issue tracker.

Acknowledgement

Gremlin artwork by Ketrina Yim — "safety first."

Northwind Graph Model



Getting Started

To get started download the latest version of the Gremlin shell from www.tinkerpop.com and extract it. Then download the file northwind.groovy and start your Gremlin shell:

```
# find the latest Gremlin shell download
LATEST_URL=`curl -s http://tinkerpop.incubator.apache.org/ | gre
LATEST_FILENAME=`echo ${LATEST_URL} | grep -o '[^/]*$'`
LATEST_DIRECTORY=`echo ${LATEST_FILENAME} | sed 's/-bin\.zip//'`

# download and extract the Gremlin shell
wget -q ${LATEST_URL}
unzip -q ${LATEST_FILENAME}

# start the Gremlin shell
wget -q http://sql2gremlin.com/assets/northwind.groovy -O /tmp/n
${LATEST_DIRECTORY}/bin/gremlin.sh /tmp/northwind.groovy
```

In your Gremlin shell create the Northwind graph, a graph traversal source and you're ready to go:

```
gremlin> graph = NorthwindFactory.createGraph()
```

```
==>tinkergraph[vertices:3209 edges:6177]
gremlin> g = graph.traversal()
==>graphtraversalsource[tinkergraph[vertices:3209 edges:6177], s
```

The graph is now filled with vertices and edges. Vertices have a number of properties, depending on what they represent. The following properties are globally indexed for fast lookups:

VertexLabel	Property	Description
`region`	`name`	The code or name for a specific region.
`country`	`name`	The code or name for a specific country.
`category`	`name`	The name of a specific category.
`customer`	`customerId`	The well-known Northwind customer identifier (e.g. ALFKI).

Select

Select all

This sample shows how to query all categories.

SQL

```
SELECT *
  FROM Categories
```

Gremlin

```
gremlin> g.V().hasLabel("category").valueMap()
==>[name:[Beverages], description:[Soft drinks, coffees, teas, b
==>[name:[Condiments], description:[Sweet and savory sauces, rel
==>[name:[Confections], description:[Desserts, candies, and swee
==>[name:[Dairy Products], description:[Cheeses]]
==>[name:[Grains/Cereals], description:[Breads, crackers, pasta,
==>[name:[Meat/Poultry], description:[Prepared meats]]
```

```
==>[name:[Produce], description:[Dried fruit and bean curd]]
==>[name:[Seafood], description:[Seaweed and fish]]
```

References:

- Has Step
- ValueMap Step

Select single column

This sample shows how to query the names of all categories.

SQL

```
SELECT CategoryName
  FROM Categories
```

Gremlin

```
gremlin> g.V().hasLabel("category").values("name")
==>Beverages
==>Condiments
==>Confections
==>Dairy Products
==>Grains/Cereals
==>Meat/Poultry
==>Produce
==>Seafood
```

References:

- Has Step

Select multiple columns

This sample shows how to query the names and descriptions of all categories.

SQL

```
SELECT CategoryName, Description
  FROM Categories
```

Gremlin

```
gremlin> g.V().hasLabel("category").valueMap("name", "descriptic
==>[name:[Beverages], description:[Soft drinks, coffees, teas, b
==>[name:[Condiments], description:[Sweet and savory sauces, rel
==>[name:[Confections], description:[Desserts, candies, and swee
==>[name:[Dairy Products], description:[Cheeses]]
==>[name:[Grains/Cereals], description:[Breads, crackers, pasta,
==>[name:[Meat/Poultry], description:[Prepared meats]]
==>[name:[Produce], description:[Dried fruit and bean curd]]
==>[name:[Seafood], description:[Seaweed and fish]]
```

References:

- Has Step
- ValueMap Step

Select calculated column

This sample shows how to query the length of the name of all categories.

SQL

```
SELECT LENGTH(CategoryName)
  FROM Categories
```

Gremlin

```
gremlin> g.V().hasLabel("category").values("name").
               map {it.get().length()}
==>9
==>10
==>11
==>14
==>14
==>12
==>7
==>7
```

References:

- Has Step
- Lambda Steps
- String::length()

Select distinct values

This sample shows how to query all distinct lengths of category names.

SQL

```
SELECT DISTINCT LENGTH(CategoryName)
  FROM Categories
```

Gremlin

```
gremlin> g.V().hasLabel("category").values("name").
               map {it.get().length()}.dedup()
==>9
==>10
==>11
==>14
==>12
==>7
```

References:

- Dedup Step
- Has Step
- Lambda Steps
- String::length()

Select scalar value

This sample shows how to query the length of the longest category name.

SQL

```
SELECT MAX(LENGTH(CategoryName))
  FROM Categories
```

Gremlin

```
gremlin> g.V().hasLabel("category").values("name").
               map {it.get().length()}.max()
==>14
```

References:

- Has Step
- Lambda Steps
- Max Step
- String::length()

Filtering

Filter by equality

This sample shows how to query all products having no unit in stock.

SQL

```
SELECT ProductName, UnitsInStock
  FROM Products
 WHERE UnitsInStock = 0
```

Gremlin

```
gremlin> g.V().has("product", "unitsInStock", 0).valueMap("name'
==>[unitsInStock:[0], name:[Chef Anton's Gumbo Mix]]
==>[unitsInStock:[0], name:[Alice Mutton]]
==>[unitsInStock:[0], name:[Thüringer Rostbratwurst]]
==>[unitsInStock:[0], name:[Gorgonzola Telino]]
==>[unitsInStock:[0], name:[Perth Pasties]]
```

References:

- Has Step

- ValueMap Step

Filter by inequality

This sample shows how to query all products with a unit price not exceeding 10.

SQL

```
SELECT ProductName, UnitsOnOrder
  FROM Products
 WHERE NOT(UnitsOnOrder = 0)
```

Gremlin

```
gremlin> g.V().has("product", "unitsOnOrder", neq(0)).
               valueMap("name", "unitsOnOrder")
==>[name:[Chang], unitsOnOrder:[40]]
==>[name:[Aniseed Syrup], unitsOnOrder:[70]]
==>[name:[Queso Cabrales], unitsOnOrder:[30]]
==>[name:[Sir Rodney's Scones], unitsOnOrder:[40]]
==>[name:[Gorgonzola Telino], unitsOnOrder:[70]]
==>[name:[Mascarpone Fabioli], unitsOnOrder:[40]]
==>[name:[Gravad lax], unitsOnOrder:[50]]
==>[name:[Ipoh Coffee], unitsOnOrder:[10]]
==>[name:[Rogede sild], unitsOnOrder:[70]]
==>[name:[Chocolade], unitsOnOrder:[70]]
...
```

References:

- Has Step
- ValueMap Step
- A Note on Predicates

Filter by value range

This sample shows how to query all products with a minimum price of 5 and maximum price below 10.

SQL

```
SELECT ProductName, UnitPrice
  FROM Products
 WHERE UnitPrice >= 5 AND UnitPrice < 10
```

Gremlin

```
gremlin> g.V().has("product", "unitPrice", between(5f, 10f)).
               valueMap("name", "unitPrice")
==>[unitPrice:[6.0], name:[Konbu]]
==>[unitPrice:[9.2], name:[Teatime Chocolate Biscuits]]
==>[unitPrice:[9.0], name:[Tunnbröd]]
==>[unitPrice:[9.65], name:[Jack's New England Clam Chowder]]
==>[unitPrice:[9.5], name:[Rogede sild]]
==>[unitPrice:[9.5], name:[Zaanse koeken]]
==>[unitPrice:[7.0], name:[Filo Mix]]
==>[unitPrice:[7.45], name:[Tourtière]]
==>[unitPrice:[7.75], name:[Rhönbräu Klosterbier]]
```

References:

- Has Step
- ValueMap Step
- A Note on Predicates

Multiple filter conditions

This sample shows how to query all discontinued products that are still not out of stock.

SQL

```
SELECT ProductName, UnitsInStock
  FROM Products
 WHERE Discontinued = 1
   AND UnitsInStock <> 0
```

Gremlin

```
gremlin> g.V().has("product", "discontinued", true).has("unitsIn
              valueMap("name", "unitsInStock")
==>[unitsInStock:[29], name:[Mishi Kobe Niku]]
==>[unitsInStock:[20], name:[Guaraná Fantástica]]
```

```
==>[unitsInStock:[26], name:[Rössle Sauerkraut]]
==>[unitsInStock:[26], name:[Singaporean Hokkien Fried Mee]]
```

References:

- Has Step
- ValueMap Step
- A Note on Predicates

Ordering

Order by value ascending

This sample shows how to query all products ordered by unit price.

SQL

```
  SELECT ProductName, UnitPrice
    FROM Products
ORDER BY UnitPrice ASC
```

Gremlin

```
gremlin> g.V().hasLabel("product").order().by("unitPrice", incr)
               valueMap("name", "unitPrice")
==>[unitPrice:[2.5], name:[Geitost]]
==>[unitPrice:[4.5], name:[Guaraná Fantástica]]
==>[unitPrice:[6.0], name:[Konbu]]
==>[unitPrice:[7.0], name:[Filo Mix]]
==>[unitPrice:[7.45], name:[Tourtière]]
==>[unitPrice:[7.75], name:[Rhönbräu Klosterbier]]
==>[unitPrice:[9.0], name:[Tunnbröd]]
==>[unitPrice:[9.2], name:[Teatime Chocolate Biscuits]]
==>[unitPrice:[9.5], name:[Rogede sild]]
==>[unitPrice:[9.5], name:[Zaanse koeken]]
...
```

References:

- Has Step
- Order Step
- ValueMap Step

Order by value descending

This sample shows how to query all products ordered by descending unit price.

SQL

```
  SELECT ProductName, UnitPrice
    FROM Products
ORDER BY UnitPrice DESC
```

Gremlin

```
gremlin> g.V().hasLabel("product").order().by("unitPrice", decr)
               valueMap("name", "unitPrice")
==>[unitPrice:[263.5], name:[Côte de Blaye]]
==>[unitPrice:[123.79], name:[Thüringer Rostbratwurst]]
==>[unitPrice:[97.0], name:[Mishi Kobe Niku]]
==>[unitPrice:[81.0], name:[Sir Rodney's Marmalade]]
==>[unitPrice:[62.5], name:[Carnarvon Tigers]]
==>[unitPrice:[55.0], name:[Raclette Courdavault]]
==>[unitPrice:[53.0], name:[Manjimup Dried Apples]]
==>[unitPrice:[49.3], name:[Tarte au sucre]]
==>[unitPrice:[46.0], name:[Ipoh Coffee]]
==>[unitPrice:[45.6], name:[Rössle Sauerkraut]]
...
```

References:

- Has Step
- Order Step
- ValueMap Step

Paging

Limit number of results

This sample shows how to query the first 5 products ordered by unit price.

SQL

```
  SELECT TOP (5) ProductName, UnitPrice
    FROM Products
ORDER BY UnitPrice
```

Gremlin

```
gremlin> g.V().hasLabel("product").order().by("unitPrice", incr)
               valueMap("name", "unitPrice")
==>[unitPrice:[2.5], name:[Geitost]]
==>[unitPrice:[4.5], name:[Guaraná Fantástica]]
==>[unitPrice:[6.0], name:[Konbu]]
==>[unitPrice:[7.0], name:[Filo Mix]]
==>[unitPrice:[7.45], name:[Tourtière]]
```

References:

- Has Step
- Limit Step
- Order Step
- ValueMap Step

Paged result set

This sample shows how to query the next 5 products (page 2) ordered by unit price.

SQL

```
   SELECT Products.ProductName, Products.UnitPrice
     FROM (SELECT ROW_NUMBER()
                    OVER (
                      ORDER BY UnitPrice) AS [ROW_NUMBER],
                  ProductID
             FROM Products) AS SortedProducts
       INNER JOIN Products
               ON Products.ProductID = SortedProducts.ProductID
    WHERE [ROW_NUMBER] BETWEEN 6 AND 10
 ORDER BY [ROW_NUMBER]
```

Gremlin

```
gremlin> g.V().hasLabel("product").order().by("unitPrice", incr)
               valueMap("name", "unitPrice")
==>[unitPrice:[7.75], name:[Rhönbräu Klosterbier]]
==>[unitPrice:[9.0], name:[Tunnbröd]]
==>[unitPrice:[9.2], name:[Teatime Chocolate Biscuits]]
==>[unitPrice:[9.5], name:[Rogede sild]]
==>[unitPrice:[9.5], name:[Zaanse koeken]]
```

References:

- Has Step
- Range Step
- Order Step
- ValueMap Step

Grouping

Group by value

This sample shows how to determine the most used unit price.

SQL

```
  SELECT TOP(1) UnitPrice
    FROM (SELECT Products.UnitPrice,
                 COUNT(*) AS [Count]
            FROM Products
        GROUP BY Products.UnitPrice) AS T
ORDER BY [Count] DESC
```

Gremlin

```
gremlin> g.V().hasLabel("product").groupCount().by("unitPrice").
```

```
               order(local).by(valueDecr).mapKeys().limit(1)
==>18.0
```

References:

- Has Step
- GroupCount Step
- Limit Step
- MapKeys Step
- Order Step
- ValueMap Step

Joining

Inner join

This sample shows how to query all products from a specific category.

SQL

```
    SELECT Products.ProductName
      FROM Products
INNER JOIN Categories
        ON Categories.CategoryID = Products.CategoryID
     WHERE Categories.CategoryName = 'Beverages'
```

Gremlin

```
gremlin> g.V().has("name","Beverages").in("inCategory").values('
==>Chai
==>Rhönbräu Klosterbier
==>Chartreuse verte
==>Chang
==>Lakkalikööri
==>Ipoh Coffee
==>Guaraná Fantástica
==>Côte de Blaye
==>Steeleye Stout
```

```
==>Outback Lager
...
```

References:

- Has Step
- Vertex Steps

Left join

This sample shows how to count the number of orders for each customer.

SQL

```
   SELECT Customers.CustomerID, COUNT(Orders.OrderID)
     FROM Customers
LEFT JOIN Orders
       ON Orders.CustomerID = Customers.CustomerID
 GROUP BY Customers.CustomerID
```

Gremlin

```
gremlin> g.V().hasLabel("customer").match(
           __.as("c").values("customerId").as("customerId"),
           __.as("c").out("ordered").count().as("orders")
gremlin> ).select("customerId", "orders")
==>[customerId:ALFKI, orders:6]
==>[customerId:ANATR, orders:4]
==>[customerId:ANTON, orders:7]
==>[customerId:AROUT, orders:13]
==>[customerId:BERGS, orders:18]
==>[customerId:BLAUS, orders:7]
==>[customerId:BLONP, orders:11]
==>[customerId:BOLID, orders:3]
==>[customerId:BONAP, orders:17]
==>[customerId:BOTTM, orders:14]
...
```

References:

- As Step

- Count Step
- Has Step
- Match Step
- Select Step
- Vertex Steps

Miscellaneous

Concatenate

This sample shows how to concatenate two result sets (customers whos company name starts with A and customers whos company name starts with E).

SQL

```
SELECT [customer].[CompanyName]
  FROM [Customers] AS [customer]
 WHERE [customer].[CompanyName] LIKE 'A%'
 UNION ALL
SELECT [customer].[CompanyName]
  FROM [Customers] AS [customer]
 WHERE [customer].[CompanyName] LIKE 'E%'
```

Gremlin

```
gremlin> g.V().hasLabel("customer").union(
           filter {it.get().value("company")[0] == "A"},
           filter {it.get().value("company")[0] == "E"}).values(
==>Alfreds Futterkiste
==>Ana Trujillo Emparedados y helados
==>Antonio Moreno Taquería
==>Around the Horn
==>Eastern Connection
==>Ernst Handel
```

References:

- Has Step

Create, Update and Delete

This sample shows how to create new vertices and edges, how to update them and finally how to delete them.

SQL

```
INSERT INTO [Categories] ([CategoryName], [Description])
     VALUES (N'Merchandising', N'Cool products to promote Gremli

INSERT INTO [Products] ([ProductName], [CategoryID])
     SELECT TOP (1) N'Red Gremlin Jacket', [CategoryID]
       FROM [Categories]
      WHERE [CategoryName] = N'Merchandising'

UPDATE [Products]
   SET [Products].[ProductName] = N'Green Gremlin Jacket'
 WHERE [Products].[ProductName] = N'Red Gremlin Jacket'

DELETE FROM [Products]
 WHERE [Products].[ProductName] = N'Green Gremlin Jacket'

DELETE FROM [Categories]
 WHERE [Categories].[CategoryName] = N'Merchandising'
```

Gremlin

```
gremlin> c = graph.addVertex(label, "category",
                   "name", "Merchandising",
                   "description", "Cool products to promote Grem
==>v[0]
gremlin>
gremlin> p = graph.addVertex(label, "product",
                   "ame", "Red Gremlin Jacket")
==>v[3]
gremlin>
gremlin> p.addEdge("inCategory", c)
==>e[5][3-inCategory->0]
```

```
gremlin>
gremlin> g.V().has("product", "name", "Red Gremlin Jacket").
               property("name", "Green Gremlin Jacket").iterate(
gremlin>
gremlin> p.remove()
==>null
gremlin> g.V().has("category", "name", "Merchandising").drop()
```

References:

- Mutating the Graph
- Has Step
- Drop Step

CTE

Recursive query

This sample shows how to query all employees, their supervisors and their hierarchy level depending on where the employee is located in the supervisor chain.

SQL

```
WITH EmployeeHierarchy (EmployeeID,
                        LastName,
                        FirstName,
                        ReportsTo,
                        HierarchyLevel) AS
(
    SELECT EmployeeID
         , LastName
         , FirstName
         , ReportsTo
         , 1 as HierarchyLevel
      FROM Employees
     WHERE ReportsTo IS NULL

     UNION ALL

    SELECT e.EmployeeID
```

```
         , e.LastName
         , e.FirstName
         , e.ReportsTo
         , eh.HierarchyLevel + 1 AS HierarchyLevel
      FROM Employees e
INNER JOIN EmployeeHierarchy eh
        ON e.ReportsTo = eh.EmployeeID
)
  SELECT *
    FROM EmployeeHierarchy
ORDER BY HierarchyLevel, LastName, FirstName
```

Gremlin (hierarchical)

```
gremlin> g.V().hasLabel("employee").where(__.not(out("reportsTo'
               repeat(__.in("reportsTo")).emit().tree().by(map {
                 def employee = it.get()
                 employee.value("firstName") + " " + employee.va
               }).next()
==>Andrew Fuller={Margaret Peacock={}, Janet Leverling={}, Nancy
```

You can also produce the same tabular result that's produced by SQL.

Gremlin (tabular)

```
gremlin> g.V().hasLabel("employee").where(__.not(out("reportsTo'
               repeat(__.as("reportsTo").in("reportsTo").as("emp
               select(last, "reportsTo", "employee").by(map {
                 def employee = it.get()
                 employee.value("firstName") + " " + employee.va
               })
==>[reportsTo:Andrew Fuller, employee:Nancy Davolio]
==>[reportsTo:Andrew Fuller, employee:Janet Leverling]
==>[reportsTo:Andrew Fuller, employee:Margaret Peacock]
==>[reportsTo:Andrew Fuller, employee:Steven Buchanan]
==>[reportsTo:Andrew Fuller, employee:Laura Callahan]
==>[reportsTo:Steven Buchanan, employee:Robert King]
==>[reportsTo:Steven Buchanan, employee:Anne Dodsworth]
==>[reportsTo:Steven Buchanan, employee:Michael Suyama]
```

References:

- As Step
- Has Step
- Lambda Steps
- Repeat Step
- Select Step
- Vertex Steps
- Tree Step
- Where Step

Complex

Pivots

This sample shows how to determine the average total order value per month for each customer.

SQL

```
SELECT Customers.CompanyName,
       COALESCE([1], 0)  AS [Jan],
       COALESCE([2], 0)  AS [Feb],
       COALESCE([3], 0)  AS [Mar],
       COALESCE([4], 0)  AS [Apr],
       COALESCE([5], 0)  AS [May],
       COALESCE([6], 0)  AS [Jun],
       COALESCE([7], 0)  AS [Jul],
       COALESCE([8], 0)  AS [Aug],
       COALESCE([9], 0)  AS [Sep],
       COALESCE([10], 0) AS [Oct],
       COALESCE([11], 0) AS [Nov],
       COALESCE([12], 0) AS [Dec]
  FROM (SELECT Orders.CustomerID,
               MONTH(Orders.OrderDate)
               SUM([Order Details].UnitPrice * [Order Detai]
          FROM Orders
    INNER JOIN [Order Details]
            ON [Order Details].OrderID = Orders.OrderID
      GROUP BY Orders.CustomerID,
                   MONTH(Orders.OrderDate)) o
     PIVOT (AVG(Total) FOR [Month] IN ([1],
                                       [2],
                                       [3],
                                       [4],
                                       [5],
                                       [6],
                                       [7],
                                       [8],
                                       [9],
                                       [10],
                                       [11],
                                       [12])) AS [Pivot]
INNER JOIN Customers
        ON Customers.CustomerID = [Pivot].CustomerID
  ORDER BY Customers.CompanyName
```

Gremlin

```
gremlin> months = new java.text.DateFormatSymbols().getShortMont
gremlin> monthMap = (0..11).collectEntries {[months[it], []]}; [
gremlin> rowTotal = {it.get().value("unitPrice") * it.get().valu
gremlin>
gremlin> g.V().hasLabel("customer").order().by("customerId", inc
           where(out("ordered")).as("customer").
           map {
             g.withSideEffect("m", monthMap.clone()).V(it.get()
               group("m").by {months[new Date(it.value("orderDat
                          by(out('contains').map(rowTotal).sum(
                          by(sum(local)).cap("m").next().sort {
           }.as("totals").select("customer", "totals").by(id).b
==>[customer:8, totals:[Jan:851.0, Feb:0.0, Mar:491.2, Apr:960.
==>[customer:9, totals:[Jan:851.0, Feb:0.0, Mar:1005.59999999999
==>[customer:10, totals:[Jan:1511.0, Feb:0.0, Mar:1005.599999999
==>[customer:11, totals:[Jan:1511.0, Feb:735.0, Mar:6070.6, Apr:
==>[customer:12, totals:[Jan:5395.95, Feb:4132.700000000001, Mar
==>[customer:13, totals:[Jan:6020.95, Feb:4132.700000000001, Mar
==>[customer:14, totals:[Jan:6750.95, Feb:8181.700000000001, Mar
==>[customer:15, totals:[Jan:6750.95, Feb:8181.700000000001, Mar
==>[customer:16, totals:[Jan:7593.95, Feb:11182.1, Mar:13168.4,
==>[customer:17, totals:[Jan:12127.45, Feb:11182.1, Mar:22391.0,
...
```

References:

- As Step
- Group Step
- Has Step
- Order Step
- Select Step
- Sum Step
- Where Step
- Vertex Steps
- Transform Collection to a Map with collectEntries
- DateFormatSymbols::getShortMonths()

Recommendation

This sample shows how to recommend 5 products for a specific customer. The products are chosen as follows:

- determine what the customer has already ordered
- determine who else ordered the same products
- determine what others also ordered
- determine products which were not already ordered by the initial customer, but ordered by the others
- rank products by occurence in other orders

SQL

```
SELECT TOP (5) [t14].[ProductName]
  FROM (SELECT COUNT(*) AS [value],
               [t13].[ProductName]
          FROM [customers] AS [t0]
   CROSS APPLY (SELECT [t9].[ProductName]
                  FROM [orders] AS [t1]
            CROSS JOIN [order details] AS [t2]
            INNER JOIN [products] AS [t3]
                    ON [t3].[ProductID] = [t2].[ProductID]
            CROSS JOIN [order details] AS [t4]
            INNER JOIN [orders] AS [t5]
```

```
                      ON [t5].[OrderID] = [t4].[OrderID]
               LEFT JOIN [customers] AS [t6]
                      ON [t6].[CustomerID] = [t5].[CustomerID]
              CROSS JOIN ([orders] AS [t7]
                          CROSS JOIN [order details] AS [t8]
                          INNER JOIN [products] AS [t9]
                                  ON [t9].[ProductID] = [t8].[Pr
                   WHERE NOT EXISTS(SELECT NULL AS [EMPTY]
                                      FROM [orders] AS [t10]
                                CROSS JOIN [order details] AS [t
                                INNER JOIN [products] AS [t12]
                                        ON [t12].[ProductID] = [
                                     WHERE [t9].[ProductID] = [t
                                       AND [t10].[CustomerID] =
                                       AND [t11].[OrderID] = [t1
                     AND [t6].[CustomerID] <> [t0].[CustomerID]
                     AND [t1].[CustomerID] = [t0].[CustomerID]
                     AND [t2].[OrderID] = [t1].[OrderID]
                     AND [t4].[ProductID] = [t3].[ProductID]
                     AND [t7].[CustomerID] = [t6].[CustomerID]
                     AND [t8].[OrderID] = [t7].[OrderID]) AS [t1
           WHERE [t0].[CustomerID] = N'ALFKI'
        GROUP BY [t13].[ProductName]) AS [t14]
ORDER BY [t14].[value] DESC
```

Gremlin

```
gremlin> g.V().has("customerId", "ALFKI").as("customer").
               out("ordered").out("contains").out("is").aggregat
               in("is").in("contains").in("ordered").where(neq('
               out("ordered").out("contains").out("is").where(wi
               groupCount().order(local).by(valueDecr).mapKeys()
               values("name")
==>Gorgonzola Telino
==>Guaraná Fantástica
==>Camembert Pierrot
==>Chang
==>Jack's New England Clam Chowder
```

References:

- Aggregate Step

Last updated 2015-10-23 11:56:19 CEST