Mapping Queries to Visualizations: Advanced Database Topics and Practice for Business Intelligence Students

Michel Mitri
James Madison University, Harrisonburg, VA, United States., mitrimx@jmu.edu

Mapping Queries to Visualizations: Advanced Database Topics and Practice for Business Intelligence Students

Michel Mitri
James Madison University
mitrimx@jmu.edu

ABSTRACT

This paper describes a teaching methodology for associating SQL queries with appropriately corresponding data visualizations, along with a software tool for automatically mapping SQL queries to Google's library of Image and Interactive Charts. The methodology is appropriate for an audience of advanced database students, and in particular for use in business intelligence (BI) classes during coverage of topics such as data visualization, dashboards, scorecards, and business performance management (BPM).

Keywords

SQL, aggregation queries, hierarchical queries, data visualization, dashboards, business intelligence (BI), balanced scorecard methodology (BSC).

INTRODUCTION

A major trend in IT today is increased development and use of business intelligence (BI) systems, involving enterprise databases, data warehouses, and applications that include online analytical processing (OLAP), data mining, business performance management (BPM), and advanced data visualization techniques. Data visualizations often take the form of dashboards and scorecards, and include graphical components ranging from the traditional bar, pie, and line charts, to geographical images and maps, to hierarchical organizational charts and tree maps, to non-hierarchical conceptual networks, to meters and gauges, to time-varying motion charts. Each of these visualizations is appropriate for illustrating certain types of information; some are naturally suited for conveying quantitative information, some represent conceptual relationships, others depict thresholds and/or key performance indicators (KPI) measured against strategic goals, and still others represent changing patterns over time. Visualizations come with a variety of interactive capabilities, and often allow users to drill down from aggregate summary information to more specific details. Visualizations support strategic decision making by enabling users to view pictorial representations that provide rich, multidimensional information through the assignment of meaning to visual features such as size, color, or spatial relationships. Often dashboard applications are developed based on a particular quality control/management framework such as Six Sigma or Balanced Scorecard (BSC) (Kaplan and Norton 1992), and the framework itself may guide the choices developers make regarding which types of visualizations to use and how to query the underlying data sources.

Because visualizations are intended for conveying certain types of information, it follows that the data sources for these visualizations will be required to fit specified formats. This implies, for data coming from relational databases, that there are certain patterns of SQL statements that will be most well suited to a particular visualization; or vice versa, that for a given class of SQL query, you will expect to visually represent it in a particular type of visualization. Recognizing the appropriate query-to-visualization mapping is a subtle art, one that is important to master for an advanced practitioner in DSS-motivated database querying.

This paper presents a method for teaching students in a lab-oriented business intelligence class about techniques for crafting database queries with the intent to display the results in meaningful graphical visualizations. The course is an elective offered to senior-level CIS students who have completed prerequisite coursework in database design and computer programming. By the time students reach the data visualization component of the course, they have done extensive SQL queries including joins, aggregations with grouping, sub-queries, and T-SQL extensions.

Throughout the entire BI course, we use Microsoft's AdventureWorks database and associated data warehouse as the major platform (http://msdn.microsoft.com/en-us/library/ms124501(v=sql.100).aspx). AdventureWorks is very useful for a business-oriented information technology course because it consists of a large number of interrelated tables comprising a realistic set of business functions and processes of a typical organization. The major schemas in the AdventureWorks

database are tied to human resources, sales and customer service, production and manufacturing, and supply chain, and include many table attributes related to financial/accounting information such as product prices, purchasing costs, employee salaries, production costs, etc. This gives rise to a rich variety of potential queries that could be used in order to present end users (knowledge workers and managers) with meaningful, strategically significant visualizations. Of course, this database and its associated data warehouse are also used extensively in other sections of the class, including OLAP and data mining. But the purpose of this paper is to describe AdventureWorks' use for teaching data visualization.

In this class, students learn data visualization using Google's free visualization API (http://code.google.com/apis/chart/interactive/docs/index.html), which is subdivided into two separate open-source libraries. The first, commonly known as Image Charts (http://code.google.com/apis/chart/image/), is a web service that takes REST queries (parameterized URLs) and returns static image files (PNG format). The second, called Interactive Charts, is a JavaScript library of visualization classes, which include interactive event-handling features. For both of these visualization libraries, users are expected to provide a specific data format in order to retrieve the desired visualization. The challenge presented to students is to produce queries to the database that result in the data formats that are appropriate for a given visualization. As mentioned above, this query-to-visualization mapping is an important skill for anyone hoping to succeed with dashboard development.

To help facilitate this, I developed software that enables students to (a) connect to a data source and submit SQL and T-SQL queries, (b) select a Google visualization to use, and (c) generate dashboards from these SQL-to-Visualization mappings. This SQL-to-Visualization mapping software algorithmically converts the metadata and data results from user-generated SQL queries into the visualization-generating strings required for the Image Chart (REST query) or Interactive Chart (JavaScript code), and then automatically creates and displays visualizations for these queries. Consider that dashboard displays typically produce summary information from aggregate queries; thus the SQL-to-Visualization software provides a drilldown feature, allowing for progressively disaggregated database queries to produce more narrowly focused visual displays, all through the automated SQL-to-visualization mapping algorithms. Figure 1 illustrates this process.

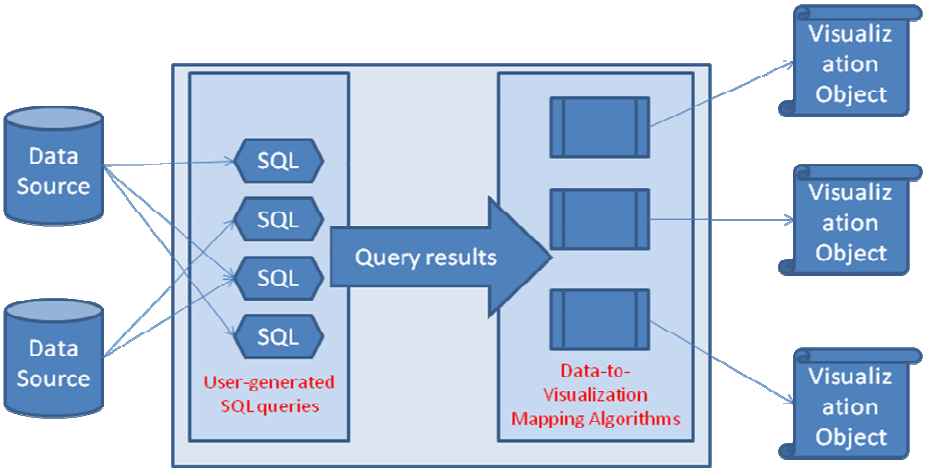

Figure 1: The overall architecture of SQL-to-Visualization mapping software

The remainder of this paper describes the methodology of mapping SQL queries to visualizations, and the software tool that is used to assist in teaching this method to students. We start with a section describing the mapping of an aggregate query to a Google Image Chart, along with a technique for automatically generating drilldown capabilities for successive disaggregation and display. In the following section, we describe the mapping of a unary join query to a hierarchical Google Interactive Chart, along with simple event handling. Then, we describe a more complex SQL-to-Visualization mapping involving both hierarchical and aggregation elements, along with a method for drilling down from the lowest level in the aggregation hierarchy; again, this example involved Google Interactive Charts. We end with some pedagogical considerations.

EXAMPLE 1

Google Image Charts are static images generated via parameterized URL strings (REST queries) sent to Google's chart web service: http://chart.apis.google.com/chart. The REST parameters allow you to establish data and label values, chart type and title, color settings, image size, and a variety of other controls. The web service returns a PNG image. Figure 2 shows an example REST query and the resulting image.

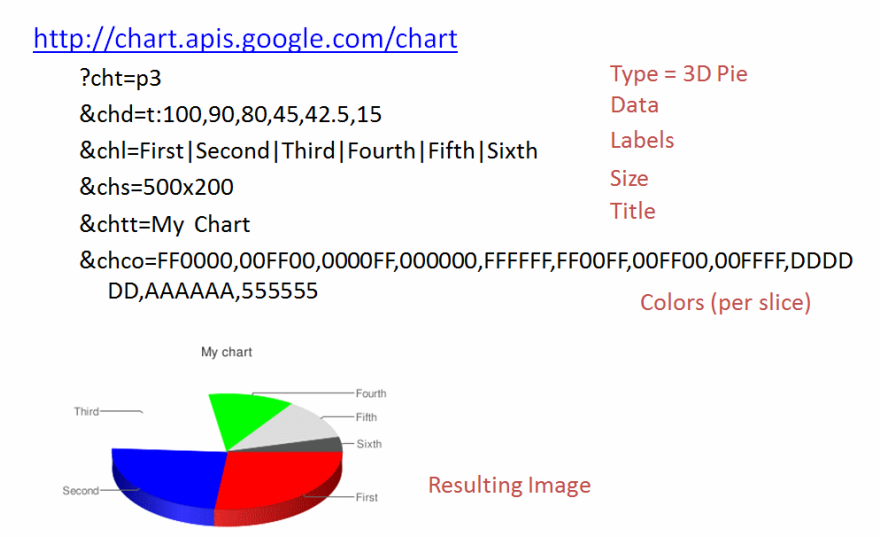

Figure 2: A REST query to the Google Image Charts web service and resulting image

For the REST query in figure 2, **cht=p3** indicates that the chart type is a 3-dimensional pie. The **chd** parameter gives the numeric data and the **chl** parameter gives the labels. A full description of the API is found at http://code.google.com/apis/chart/image/docs/making_charts.html.

As you can see, the two main query elements required for this visualization are (a) list of text values that will be used as labels for the chart (**chl** parameter) and (b) an equal length list of numeric values that will be used for determining the sizes of each pie wedge (**chd** parameter). Thus, for this visualization, students can use an aggregate query with grouping returning two columns, such as the one shown in figure 3.

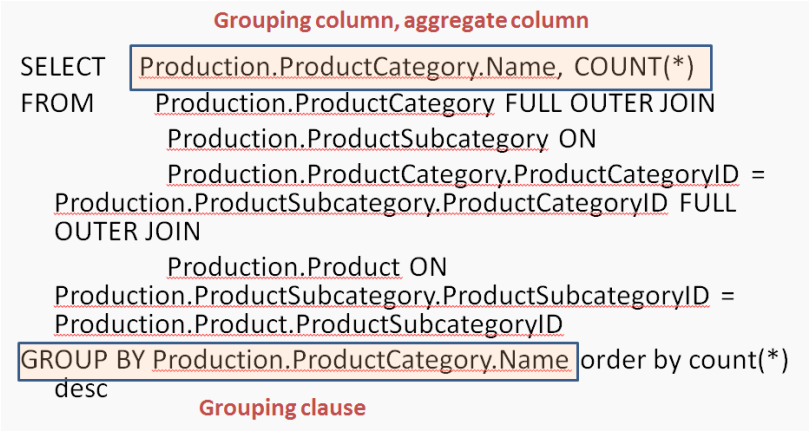

Figure 3: a sample aggregate query for the AdventureWorks database showing product categories and the count of products in these categories

Figure 4 shows the resulting mapping and display that occurs when this query is input to SQL-to-Visualization software and the user selects a bar chart for the visualization.

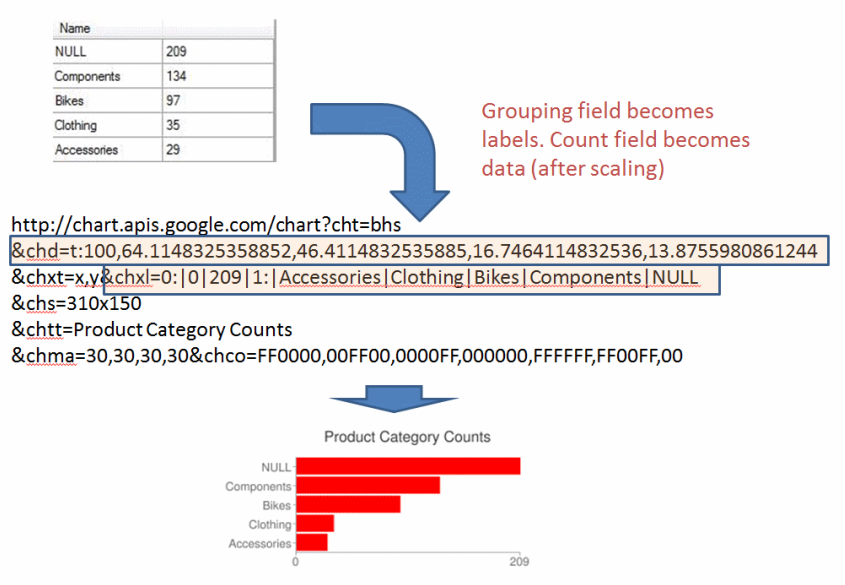

Figure 4: Query results, with automatically generated Image Chart web service request and resulting image

In the AdventureWorks database, product categories are subdivided into product subcategories, which are further divided into the products themselves, as shown in figure 5.

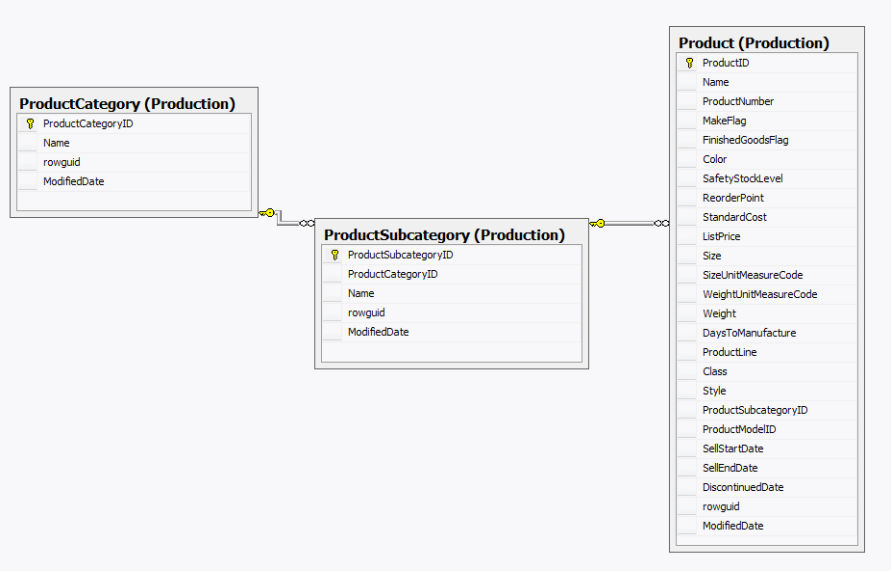

Figure 5: AdventureWorks database centered on categories, subcategories, and products

This introduces the possibility of introducing drilldown via progressive disaggregation. For a given category, the natural drilldown query would be to break out the product count by subcategory. Therefore, the SQL-to-Visualization mapping software include the capacity for a series of *drilldown fields*, enabling a user to specify how the disaggregation will progress. For the query in figure 3 on the AdventureWorks database, a drilldown field would be the name of the subcategories. In such a case, automatically generating drilldown queries would provide a mapping like shown in figure 6.

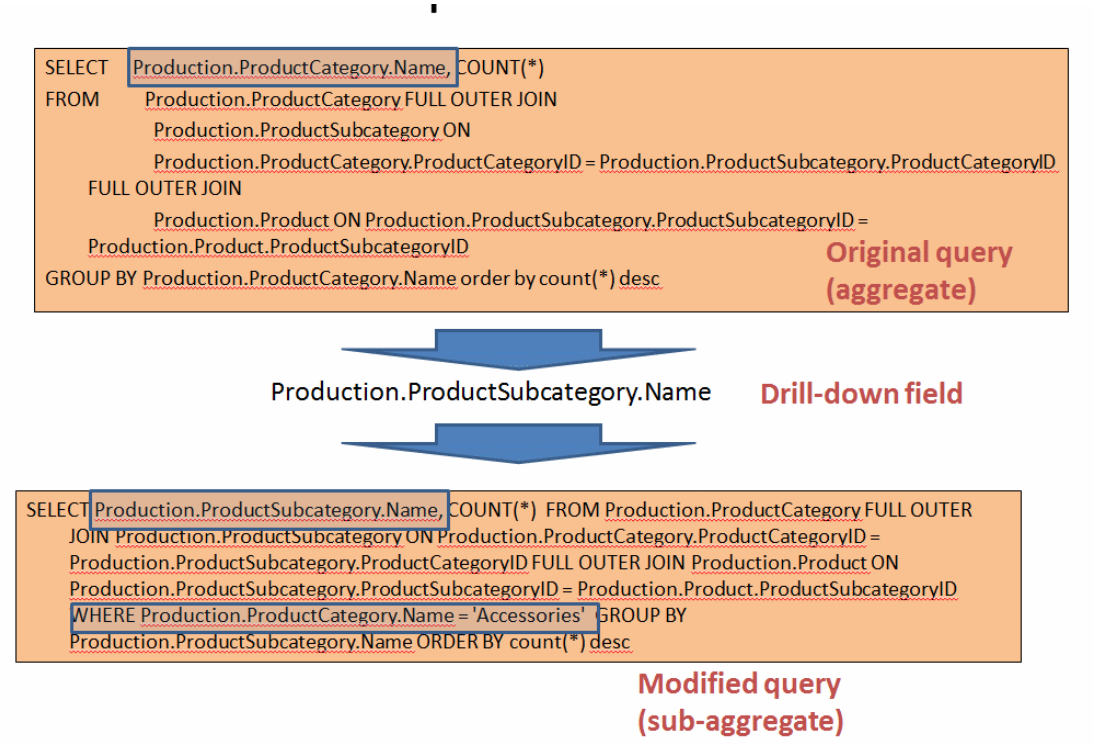

Figure 6: Automatic generation of drilldown queries in SQL-to-Visualization mapping software

Note that in this case, the grouping field from the original query (a) is replaced by a user-defined *drilldown field*, and (b) becomes a *filtering field* that is used in a WHERE clause to focus on a particular subcategory. In this way, the SQL-to-Visualization software enables automated drilldown functionality. The resulting drilldown visualization display looks like figure 7 In general, each level of drilldown introduces a new drilldown field for the grouping and creates a new condition in the WHERE clause involving filtering on the previous drilldown field.

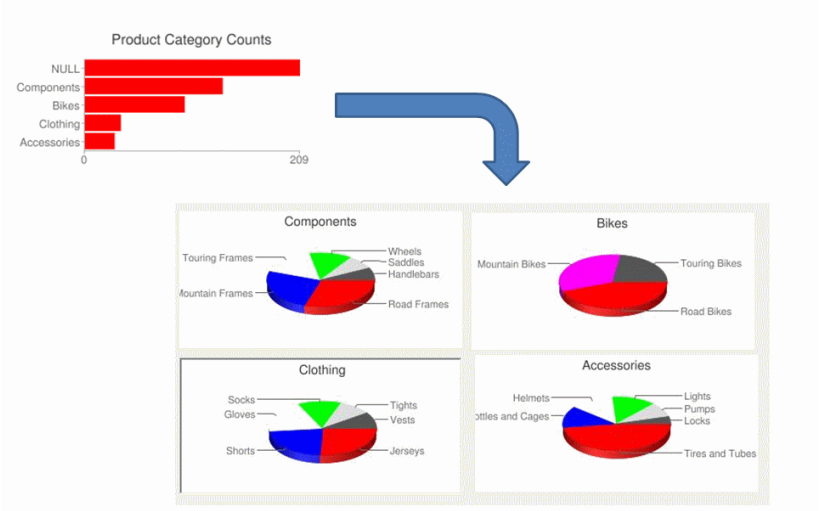

Figure 7: Visual results of drilldown query in SQL-to-Visualization software

In this section, we see how SQL-to-Visualization mapping can be applied to Google Image Charts. But these charts are limited in their functionality; specifically they do not provide interactive event-handling capabilities. For interactive charting we need to move to the second type of Google Visualization.

EXAMPLE 2

The previous example described SQL mappings to Google Image Charts, which are static images produced via REST queries submitted to Google's Image chart web service. Google also provides a much more interactive JavaScript-based library of visualization objects. In this section we consider a SQL-to-Visualization mapping that produces an organization chart based on a self-join query involving a unary relationship of a single table. This query produces a Google Interactive OrgChart that maps the supervisor-subordinate hierarchy of AdventureWorks' employees.

The Google Interactive Chart API (JavaScript class library) is documented at http://code.google.com/apis/chart/interactive/docs/index.html. Each interactive chart requires a particular data format. For the Organization Chart visualization, the requirement is a table of two mandatory columns and one optional column. These are listed below:

- Column 0 - The node ID. It should be unique among all nodes, and can include any characters, including spaces. This is shown on the node. You can specify a formatted value to show on the chart instead, but the unformatted value is still used as the ID.
- Column 1 - [optional] The ID of the parent node. This should be the unformatted value from column 0 of another row. Leave unspecified for a root node.
- Column 2 - [optional] Tool-tip text to show, when a user hovers over this node.

In general, the type of query that would be ideally suited for organization charts are queries involving unary joins. In AdventureWorks, one table with such unary relationships is the Employees table in the Human Resources schema. Another potential suitable source in the AdventrureWorks database is the BillOfMaterials table in the Production schema. In this example, we explore a mapping of a unary join query on the Employees table (along with joins to other tables for obtaining extra information) which can be used to generate an org-chart visualization via the SQL-to-Visualization mapping software. organizational hierarchy, focused on employees in the Research and Development department. As displayed in figure 8, a tool tip will appear giving an employee's job title and email address when the user hovers a mouse over that employee's node.

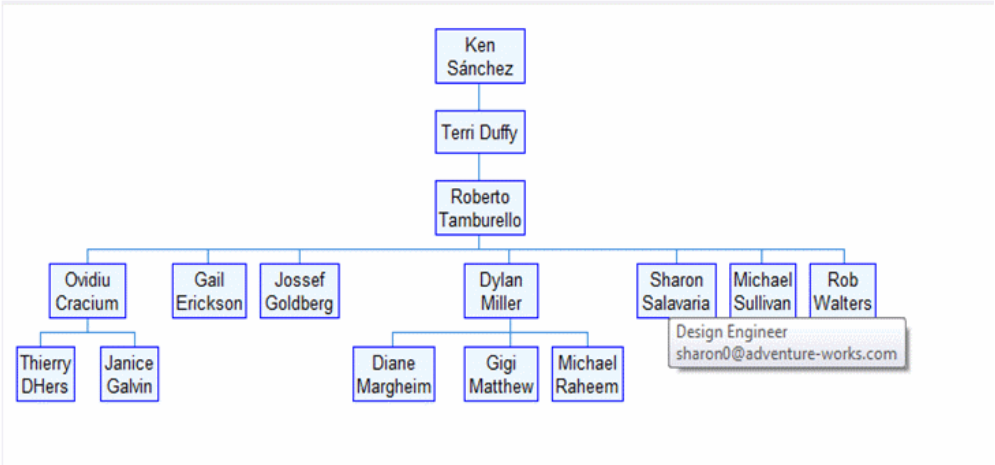

Figure 8: An OrgChart Google visualization based on AdventureWorks query of employee hierarchy.

The AdventureWorks (2005 version) query used to produce this chart is shown below.

```
SELECT    Person.Contact.FirstName + ' ' + Person.Contact.LastName AS Employee,
    Contact_1.FirstName + ' ' + Contact_1.LastName AS Manager,
                Employee.Title + '\n' + Person.Contact.EmailAddress as Info

FROM      HumanResources.Employee INNER JOIN
              HumanResources.Employee AS Employee_1 ON
    HumanResources.Employee.ManagerID = Employee_1.EmployeeID INNER JOIN
              Person.Contact ON HumanResources.Employee.ContactID =
    Person.Contact.ContactID INNER JOIN
              Person.Contact AS Contact_1 ON Employee_1.ContactID = Contact_1.ContactID
    INNER JOIN
              HumanResources.EmployeeDepartmentHistory ON
    HumanResources.Employee.EmployeeID =
    HumanResources.EmployeeDepartmentHistory.EmployeeID AND
              Employee.EmployeeID =
    HumanResources.EmployeeDepartmentHistory.EmployeeID INNER JOIN
              HumanResources.Department ON
    HumanResources.EmployeeDepartmentHistory.DepartmentID =
    HumanResources.Department.DepartmentID AND
              HumanResources.EmployeeDepartmentHistory.DepartmentID =
    HumanResources.Department.DepartmentID
where Department.GroupName = 'Research and Development'
Order by Person.Contact.LastName
```

Figure 9: AdventureWorks unary join query for producing organization chart.

This query produces the following results. Note that the first column is an employee's name (node label), the second is the manager's name (parent label), and the third contains the remaining information that appears in the tool tip.

	Employee	Manager	Info
1	Ovidiu Cracium	Roberto Tamburello	Senior Tool Designer\novidiu0@adventure-works.com
2	Thierry D'Hers	Ovidiu Cracium	Tool Designer\nthierry0@adventure-works.com
3	Terri Duffy	Ken Sánchez	Vice President of Engineering\nterri0@adventure-works.com
4	Gail Erickson	Roberto Tamburello	Design Engineer\ngail0@adventure-works.com
5	Janice Galvin	Ovidiu Cracium	Tool Designer\njanice0@adventure-works.com
6	Jossef Goldberg	Roberto Tamburello	Design Engineer\njossef0@adventure-works.com
7	Diane Margheim	Dylan Miller	Research and Development Engineer\ndiane1@adventure-works.com
8	Gigi Matthew	Dylan Miller	Research and Development Engineer\ngigi0@adventure-works.com
9	Dylan Miller	Roberto Tamburello	Research and Development Manager\ndylan0@adventure-works.com
10	Michael Raheem	Dylan Miller	Research and Development Manager\nmichael6@adventure-works.com
11	Sharon Salavaria	Roberto Tamburello	Design Engineer\nsharon0@adventure-works.com
12	Michael Sullivan	Roberto Tamburello	Senior Design Engineer\nmichael8@adventure-works.com
13	Roberto Tamburello	Terri Duffy	Engineering Manager\nroberto0@adventure-works.com
14	Rob Walters	Roberto Tamburello	Senior Tool Designer\nrob0@adventure-works.com
15	Rob Walters	Roberto Tamburello	Senior Tool Designer\nrob0@adventure-works.com

Figure 10: Result set from AdventureWorks unary join query

When this query is applied to the Google Interactive OrgChart user selection in the SQL-to-Visualization mapper, the following HTML/JavaScript code is generated:

```
<html>
  <head>
    <script src="http://www.google.com/jsapi?key=YOUR_KEY_HERE" type="text/javascript"></script>
    <script type="text/javascript">
        google.load('visualization',1, {packages:['orgchart']});
        google.setOnLoadCallback(drawVisualization);
        function drawVisualization() {
            var data = new google.visualization.DataTable();
            data.addColumn('string','Employee');
            data.addColumn('string','Manager');
            data.addColumn('string','Info');
            data.addRows(1);
            data.setCell(0,0,'Ovidiu Cracium');
            data.setCell(0,1,'Roberto Tamburello');
            data.setCell(0,2,'Senior Tool Designer\novidiu0@adventure-works.com');
            data.addRows(1);

            ,.....................

            data.setCell(14,0,'Rob Walters');
            data.setCell(14,1,'Roberto Tamburello');
            data.setCell(14,2,'Senior Tool Designer\nrob0@adventure-works.com');
            var visualization;
            visualization = new google.visualization.OrgChart(document.getElementById('visualization_div'));
            visualization.draw(data);
        }
    </script>
  </head>
  <body>
    <div id="visualization_div" style="width: 700px; height: 350;"></div>
  </body>
</html>
```

Figure 11: JavaScript generated from result set metadata and data

The SQL-to-Visualization software generates a full HTML page with JavaScript code instantiating a Google DataTable object, defining its metadata and populating its content via the result set of the query, and ultimately placing the associated OrgChart visualization in a tag on the page. This script is also available for users to copy and modify; thus a side benefit of this software is that it gives students the opportunity to make refinements and learn a bit of JavaScript.

EXAMPLE 3

Once students master aggregation and hierarchical queries, they can move on to more complex queries for use in more sophisticated interactive charts, such as Google's TreeMap. The TreeMap is hierarchical, like an OrgChart, but it also include visual representations for quantitative data (represented by size and/or color). In addition, it includes an inherent event-handling mechanism that allows drill-down through its hierarchical structure, which allows for successive disaggregation similar to EXAMPLE 1. In addition, the SQL-to-Visualization mapping software provides an extra level of drilldown by allowing users to create a drilldown query which can be associated with a second visualization. In this example we see the application of these features for creating a multi-level TreeMap with an associated BarChart by utilizing Google visualization event-handling features.

The TreeMap data format requires three columns with one additional optional column: (1) node label, (2) parent label, (3) a numeric value for size, and (4) an optional numeric value for color. From the AdventureWorks database the union query of figure 12 will produce an arbitrary root node (necessary for the TreeMap), then a second level of product category nodes, and finally a third level of product subcategory nodes.

```
select top 1 'All' as SubCategory, null as Category, 0 as TotProducts , 0 as AvgPrice from Production.ProductCategory

union

select Production.ProductCategory.Name as SubCategory, 'All' as Category, 0 as TotProducts, 0 as AvgPrice from Production.ProductCategory

union

SELECT     Production.ProductSubcategory.Name AS SubCategory,
           Production.ProductCategory.Name AS Category, COUNT(*) AS TotProducts, AVG(ListPrice) as AvgPrice
FROM        Production.ProductCategory FULL OUTER JOIN
                  Production.ProductSubcategory ON
                  Production.ProductCategory.ProductCategoryID = Production.ProductSubcategory.ProductCategoryID FULL OUTER JOIN
                  Production.Product ON Production.ProductSubcategory.ProductSubcategoryID = Production.Product.ProductSubcategoryID
GROUP BY Production.ProductCategory.Name, Production.ProductSubcategory.Name, Production.ProductCategory.ProductCategoryID
HAVING      (NOT (Production.ProductCategory.ProductCategoryID IS NULL))
```

Figure 12: Union query producing three levels of data for a TreeMap

When the metadata and data produced from this query are applied to the SQL-to-Visualization mapper, the resulting JavaScript produces an image like figure 13. In this case size represents number of products in a category or subcategory, and color indicates the average price of these products. The behavior of the TreeMap is such that clicking on a non-leaf node will display that node's children, and this behavior continues through successive disaggregations until reaching the bottom level of the hierarchy (see figure 14 for a view of Component's child nodes.

Figure 13: Top level of TreeMap produced by union query

The SQL-to-Visualization mapper supports an extra drilldown query as shown in figure 14. In this case the drilldown query produces ten most expensive products and their list prices in a selected product category. The screenshot of the SQL-to-Visualization mapper shows a second-level view of the TreeMap and a bar chart that displays when the user selects the Wheels node. The JavaScript is available for users to copy and modify if desired.

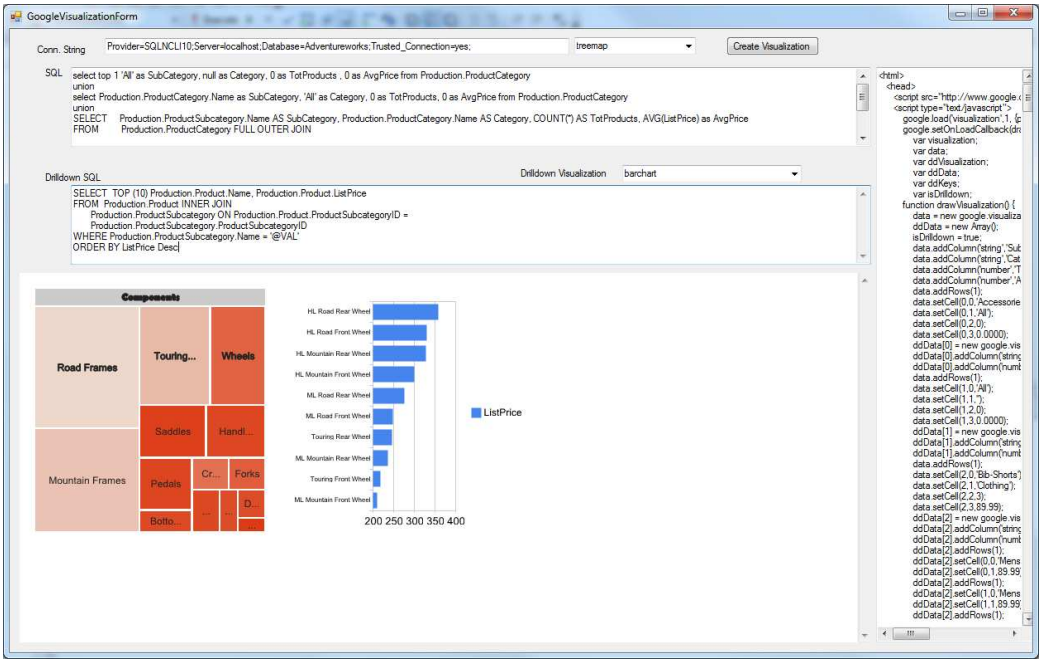

Figure 14: SQL-to-Visualization mapper with anchor and root queries creating associated visualizations

PEDAGOGICAL CONSIDERATIONS

Through a pedagogical process involving rigorous SQL query practice, conceptually relating query patterns to visualization types, and utilizing real-world databases and open-source visualization libraries, a lab-oriented business intelligence course can provide a vital and robust active learning experience for information systems students. When placed in the context of a broader BI curriculum that includes business performance management (BPM), data integration and warehousing, online analytical processing (OLAP), and data mining, the data visualization skills learned by students offer a powerful foundation for their professional development, In addition to mastering the technical issues (SQL, visualization mappings, etc.), students can apply this process to established best BPM practices (Turban et al 2011). For example, utilizing the methodology and software tool described in this paper, students can create dashboards generated from queries that address the four main perspectives of the balanced scorecard (BSC) approach, shown in figure 15.

	Strategy Map: Linked Objectives	Balanced Scorecard: Measures and Targets		Strategic Initiatives: Action Plan
Financial	*Increase Net Income*	Net income gorwth	Increase 25%	
Customer	*Increase Customer Retention*	Maintenance retention rate	Increase 15%	Change licensing and maintenance contracts
Process	*Improve Call Center Performance*	Issue turnaround time	Improve 30%	Standardize call center processes
Learn and Growth	*Reduce Employee Turnover*	Voluntary turnover rate	Reduce 25%	Salary and bonus upgrade

Figure 15: Balanced scorecard perspectives, strategy map, measures/targets, and action plan (Turban et al 2011)

Dashboards built via the BSC approach will include measures and targets related to (a) financial considerations, (b) customer demographics, trends, and satisfaction levels, (c) internal business processes, and (d) considerations of learning and growth. The AdventureWorks database provides a repository of BSC-relevant data for all four of these perspectives. By utilizing the AdventureWorks database and the SQL-to-Visualization mapping software, students can gain practical experience applying realistic data to a well-established methodology for strategic planning and assessment. This combination of technology skills and business acumen is exactly what students in a rigorous BI curriculum should be learning.

REFERENCES

1. Google Inc. (checked 5/4/2012) Google chart tools, available at http://code.google.com/apis/chart/interactive/docs/index.html.
2. Google Inc. (checked 5/4/2012) Google chart tools: image charts, available at http://code.google.com/apis/chart/image.
3. Kaplan, R. and Norton, D. (1992) The balanced scorecard – measures that drive performance, Harvard Business Review, 70, 1, 71-79.
4. Microsoft, Inc. (checked 5/4/2012) Adventureworks sample databases, available at http://msdn.microsoft.com/en-us/library/ms124501(v=sql.100).aspx.
5. Turban, E., Sharda, R., Delen, D., King, D. (2011), Business Intelligence: A Managerial Approach, Prentice Hall, Upper Saddle River, NJ.