A system for query and graphical analysis of abstract relational databases

Alina OBJELEAN
Coventry University, United Kingdom
alina.objelean@gmail.com

***Abstract* — Many organisations, both large and small, understand the competitive advantage that comes from leveraging the knowledge locked in their databases. The digital economy is largely driven by the effectiveness of use of this information in achieving their objectives. This creates significant demand to provide custom reporting features to be able to explore and analyse the data gathered or generated by software systems. This paper presents a software solution that enables users to gain real-time access to their data with fully customizable graphical and statistical analysis. The novel approach of this web based solution is offering these powerful features on potentially any arbitrary database located in the cloud. Featuring an intuitive "drag and drop" interface the software aims to provide the ability to build complex queries and translate them into insightful visual representations for users of various levels of technical expertise.**

***Index Terms* —dashboard, data analysis, relational databases, SQL queries**

I. INTRODUCTION

Over the past years, many organizations, both large and small, have implemented resource planning systems or enterprise systems. The motivation behind these investments is to improve organizational efficiency, effectiveness, and ultimately performance. Companies have gathered immense amounts of data based on customer profiles, transaction records, phone calls and business data stored across multiple databases. The number of information stored is increasing constantly at a "terrific annual compound rate of 60%" [1]. The abundance of data now available is a resource, but on their own, resources and technologies are neither good nor bad; it depends on how they are used [2].

Organizations need the ability to measure and act on key indicators and events in real time. The solutions that are currently available range from predefined set exports to very sophisticated and costly business intelligence applications. The former solution may not offer the required insights and understanding of the data and trends and the latter may prove too sophisticated and expensive to provide the necessary return on investment.

The solution described in this paper arose from a practical problem when different organizations using one of our data management systems required an ever growing number of custom reports. The implementation of these individual reports was adding a lot of development and management overhead, which lead to the idea of building a dashboard interface that would allow users to create their own bespoke reports. This paper presents the design and implementation details of a web based system that aims to solve these challenges by offering customizable data exploration and visual analysis of abstract relational databases.

II. SYSTEM OVERVIEW

The software has been designed to support the interactive exploration of multiple multidimensional relational databases by extracting statistical information and analysing and visualizing data and trends. One of the key areas in data analysis focuses on presenting graphical metaphors that allow people to discover data trends and patterns. There are 5 steps to this data analysis:

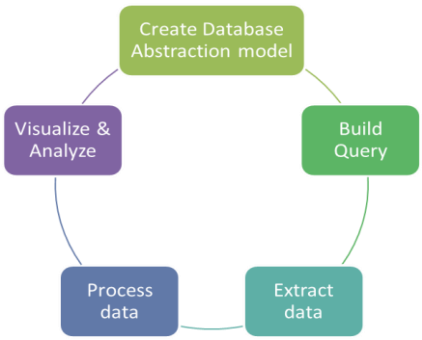


Figure 1-Process flow steps

1. CREATE DATABASE ABSTRACTION MODEL

Relational databases organize data into tables that mostly correspond to real life business entities. For example, business entities such as products and transactions may correspond to similar tables where each row contains information about a product or a sale or another business object or fact. Each column on that table will refer to a property of the entity. For example a product will have attributes like: name, description, price, quantity, etc.

The first step in the database exploration is to identify all the relevant business objects that will be used in data analysis such as **entities** (tables) and **attributes** (columns). This is achieved by querying the database schema and creating an abstraction model.

```
SELECT TABLE_SCHEMA, TABLE_NAME,
COLUMN_NAME, DATA_TYPE FROM
INFORMATION_SCHEMA.columns
WHERE TABLE_NAME <> 'sysdiagrams'
AND TABLE_NAME IN
      (SELECT TABLE_NAME
      FROM INFORMATION_SCHEMA.TABLES
      WHERE TABLE_TYPE = 'BASE TABLE')
ORDER BY TABLE_SCHEMA, TABLE_NAME,
COLUMN_NAME
```

A representation of this abstraction model is then stored in the application's database and is used to build the queries and create reports specified by the user. The following figure shows the schema of the tables associated with the storage of the queried database model.

Figure 2-Internal database schema

For each attribute stored in the DBAttribute table the application will also store what type of data it represents in one of the following formats: Numeric, Date/Time or Text.

This abstraction gives a very simple representation of the data. But to be able to create more complex reports we need to know exactly how these entities are connected. This is achieved by analysing the way tables are linked between them through foreign keys. A foreign key is a field in a relational table (the child) that matches the primary key column of another table (the parent) and is used to cross-reference tables.

Foreign keys are stored in the DBLink table which contains information about how various tables are connected. This information is crucial for building queries involving multiple entities since a query can contain data from two or more different entities only if those entities are connected.

In terms of the database structure, this means that we can select data from different tables only if a foreign key exists to allow joining the tables.

If the database schema is represented as an oriented graph where tables are vertices and the edges linking the tables are foreign keys, the problem becomes to find if there is a path between any two given vertices.

Floyd–Warshall algorithm is a classic algorithm designed to find the least-expensive paths between all vertices in a graph [3]. The algorithm compares all possible paths through the graph between each pair of vertices. It does so by incrementally improving an estimate on the shortest path between two vertices, until the estimate is optimal [3].

When this routine finishes the entries in all positions of the weights matrix represent the lowest-cost traversal between the row-vertex and column-vertex. The matrix also contains intermediate nodes that give the shortest path between any two entities. These results are then stored as comma separated values in the database.

2. BUILD QUERY

The figure below represents the chart designer that builds dimensions on the two axes:

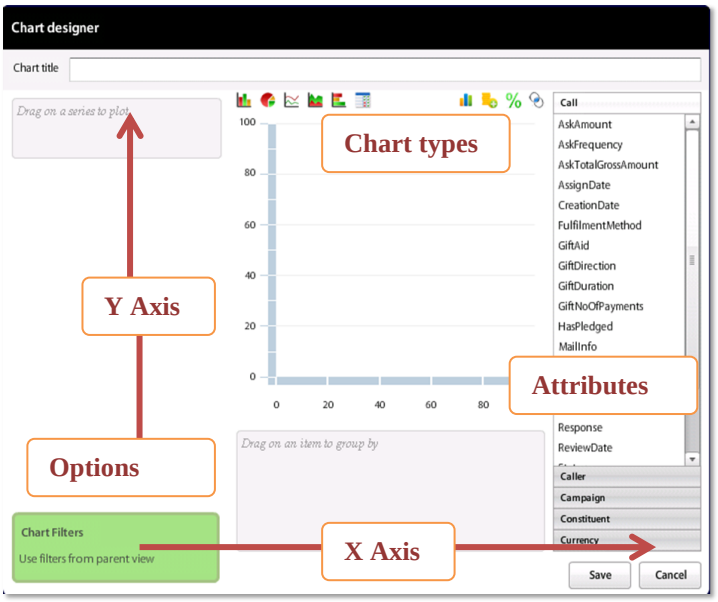


Figure 3 Chart designer

Users generate graphs by visually specifying the following elements:

- **Attributes** - each series is built around an attribute. A chart contains one attribute for X Axis and up to 4 series on Y axis
- **Aggregate functions** – defines how Y series values are grouped by items on X axis
- **Grouping** – optional, allows to classify and categorize items
- **Filter, Order by, Limit** – optional chart parameters

Based on the visual specification above the chart description is stored in an XML format. An example of an XML-formatted query is presented below:

```
<?xml version="1.0" encoding="utf-8"?>
<Query id="1">
  <xAxis groupId="0"
         Entity="User"
         Attribute="Name"
         AttributeType="1"
         Operation=""
         FilterItems=""
         SeriesName="Caller.Name" />
  <yAxis Entity="CampaignCall"
         Attribute="TotalGrossPledged"
```

```
        AttributeType="2"
        Operation="Sum"
        FilterOverall=""
        groupId="0:Response.Name:1"
        SeriesName="Sum of
TotalGrossPledged">
    <FilterItems type="1">
      <Filter Entity="Campaign"
              Attribute="Title"
              AttributeType="1"
              Operation=""
              FilterItems="1:[Campaign1]" />
    </FilterItems>
  </yAxis>
</Query>
```

The above XML file is used to interpret and build SQL queries on the server side by a component called QueryProcessor. Each query is run independently in a separate thread that is monitored for performance purposes. If a thread exceeds its maximum allowable time, the query is terminated to avoid low responsiveness generated by complex queries.
QueryProcessor does not determine the appearance of the visualization, it only computes the data points based on the specification. Results are then sent back to the client side which is responsible for rendering the graph image using the charting component available within client's framework.
The following section describes a formal mechanism of mapping the above XML file structure to an SQL query used to retrieve the data.

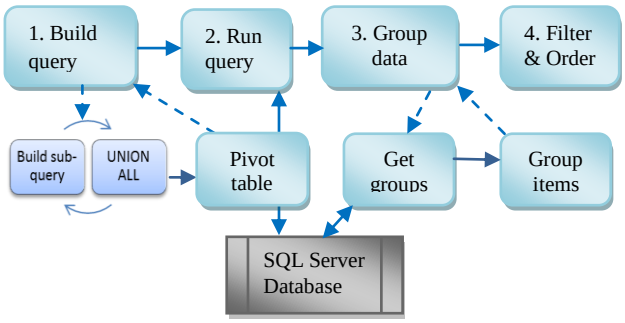


Figure 4-Query processor steps

The query generates a table configuration consisting of 2 separate expressions: one for the X axis and one for the Y axis using the following SQL statement:

```
SELECT [EntityX].[AttributeX] AS x,
Operator([EntityY1].[AttributeY1]) as
y1
FROM [EntityX]
{JOIN [EntityY1]…}
WHERE {Filters…}
GROUP BY [EntityX].[AttributeX]
{HAVING [Overall filters]}
{ORDER BY [N] ASC/DESC}
```

The query will produce a result in the following format:

X	Y1
a_1	b_1
a_2	b_2
a_3	b_3
…	…
a_n	b_n

Figure 6- Simple result format

For date/time type attributes a nested SQL statement is used to automatically group by required grouping type (year, month, day, or hour)

```
SELECT  dateadd( [grouping], q.x, 0 )
AS x,  Operator ( q.y1 ) as y1
FROM(
      SELECT  datediff([grouping], 0,
      [EntityX].[AttributeX]) AS x,
      [EntityY1].[AttributeY1] as y1
      FROM [EntityX] {JOIN [EntityY1]…}
      WHERE {Filters})
      ) q
GROUP BY x
```

The inner select statement makes sure only the necessary part of the date time value is selected. For example if items are grouped by year, it will only select the year part of the date, ignoring the rest. The outer select statement then aggregates the results grouping them by x values.
If a query contains more than one Y series, the SQL query is broken down into sub-queries very similar to the one described above. The only difference in this case is that three expression statements are used: one for x series values, one for y series values and one to indicate the name of the series. So the SQL sub -query will become:

```
SELECT [EntityX].[AttributeX] AS x,'y1'
AS col, Operator
([EntityY1].[AttributeY1]) as value
FROM [EntityX]
{JOIN [EntityY1]…}
WHERE {Filters…}
GROUP BY [EntityX].[AttributeX]
```

Each sub-query will produce a result in the following format

X	Col	value
a_1	Y1	b_1
a_2	Y1	b_2
a_3	Y1	b_3
a_4	Y1	b_4
…	…	…
a_n	Y1	b_n

Figure 5-Result format for multiple series

Result-sets are then combined using UNION ALL operator. Each SELECT statement within the UNION must have the same number of columns. The columns must also have similar data types and must be in the same order. This is

why a third column is required to identify the series the values belong to.
This is how data will look after combining result sets:

X	col	value
a_1	Y1	b_1
a_2	Y1	b_2
...	...	...
a_n	Y1	b_n
a_{n+1}	Y2	c_1
a_{n+2}	Y2	c_2
...	...	...
a_m	Y2	c_m
a_{t+1}	Y3	d_1
a_{t+2}	Y3	d_2
...	...	...
a_t	Y3	d_m
a_{u+1}	Y4	e_1
a_{u+2}	Y4	e_2
...	...	...
a_u	Y4	e_m

Figure 7-Combined result format for multiple series

To obtain the data in the required format we need to rotate the 'col' expression in the resulted table by turning the unique values from that column into multiple columns in the output.
Microsoft SQL Server provides a relational operator PIVOT that rotates a table-valued expression by turning the unique values from one column in the expression into multiple columns in the output, and performs aggregations where they are required on any remaining column values that are wanted in the final output.
By pivoting the table in the previous figure the results are obtained in the following format:

X	Y1	Y2	Y3	Y4
a_1	b_1	c_1	d_1	e_1
a_2	b_2	c_2	d_2	e_2
a_3	b_3	c_3	d_3	e_3
a_4	b_4	c_4	d_4	e_4
...	...	...	...	...
a_n	b_n	c_n	d_n	e_n

Figure 8-Multiple series final result format

3. PROCESS DATA

Based on the requirements specified in the XML file, the data may be necessary to be grouped and filtered before passing on to the client side.

a. GROUPING

There are 2 types of groupings available in the system:

A. **Groups** – each group consists of a number of discrete elements. It is used to categorize data. For example to place users into teams or products into different categories.
B. **Bands** – a set of continuous intervals. It is used to specify ranges. For example for date attributes - grouping by months or years; for numeric attributes it might be used to define age groups (e.g. young = 0 – 29, mid = 30 – 50, senior over 50, etc.).

A group's definition is stored in the database in DBAttributeData table. Since a group can have a virtually unlimited number of parts, to minimize the amount of data stored on the database we use an encoding convention to store the group structure in the database as a string.
The convention uses the following format to store the definition:

groupingType%groupingName%group1^item1^…^itemN!...!groupN^item1^…^itemN!

groupingType – can be groups or bands; must be followed by '%' sign to mark the separation between the type and the next element.
groupingName – this is the name of the classification as defined by the user; must be followed by a '%' sign as a separator.
group1..groupN – names of the groups as defined by the user. Groups' definitions are separated by a '!' sign.
Item1…itemN – the actual items that make up the group separated by a '^' sign.

A definition for bands follows almost the same pattern as above except we don't need to store the individual items, but it is enough to remember the lower limit of a group. The upper limit will be defined by the next group's lower limit. The '!' sign is used as a separator to store these data.

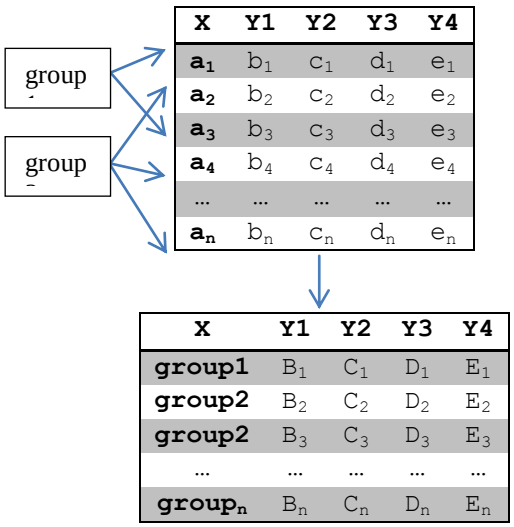
An example of a group definition:

E.g. we have 2 categories of products: beverages: coffee, tea, soda; and bakery: croissants and cakes. Using the above convention this grouping would be stored in the following format:
groups%Product categories%Beverages^ coffee ^tea! Bakery ^ croissants ^ cakes

An example of bands definition for age groups (young = 0 – 29, mid = 30 – 50, senior over 50) would be stored in the following format:

bands%Age groups%young!0!mid!30!senior!50

This notation relies on the fact that groups are stored in ascending order and as such they need to be ordered before formatting the data.
The procedure will first identify groups' structure and then place items into the groups they belong to:

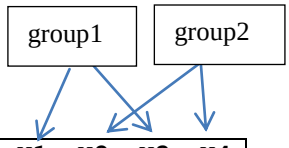


X	Y1	Y2	Y3	Y4
a_1	b_1	c_1	d_1	e_1
a_2	b_2	c_2	d_2	e_2
a_3	b_3	c_3	d_3	e_3
a_4	b_4	c_4	d_4	e_4
...	...	...	...	...
a_n	b_n	c_n	d_n	e_n

X	Y1	Y2	Y3	Y4
group1	B_1	C_1	D_1	E_1
group2	B_2	C_2	D_2	E_2
group2	B_3	C_3	D_3	E_3
...	...	...	...	...
group$_n$	B_n	C_n	D_n	E_n

Figure 9-Grouping values

Similarly to grouping items on the X Axis described in the above table, the application features the ability to group Y series:



X	Y1	Y2	Y3	Y4
a_1	b_1	c_1	d_1	e_1
a_2	b_2	c_2	d_2	e_2
a_3	b_3	c_3	d_3	e_3
a_4	b_4	c_4	d_4	e_4
...	...	...	...	...
a_n	b_n	c_n	d_n	e_n

The resulted table will be in the following format:

X	group1	group2	...	groupN
a_1	B_1	C_1	...	E_1
a_2	B_2	C_2	...	E_2
a_3	B_3	C_3	...	E_3
a_4	...	...	...	...
...	B_n	C_n	...	E_n
a_n	B_1	C_1	...	E_1

Figure 10-Grouping series

To achieve this result the procedure of grouping the items on X Axis is reused, but to apply it to the Y axis the table is transposed before and after the procedure.

b. Filtering and Sorting

To apply filters and order data is user DataTable.Select() method. The DataTable Select method accepts a filter and sort argument to return an array of DataRow objects that conform to the criteria in a FilterExpression and to the specified sort order:

```
DataRow[] rows = t.Select(filter, order);
```

c. Formatting Results

The results from the processing thread are formatted as an XML document and sent back to the client side which is then responsible of creating a visual representation based on these results. An example of a XML result is displayed below:

```
<Query id="44" Title="Title of the report"
ViewId="7">
    <QueryInfo>
       <x seriesName="Product name"></x>
       <y1 seriesName="Total Sales for category =
"Beverages"></y1>
       <y2 seriesName="Total Sales for category =
"Bakery"></y2>
        ............
      <y5 seriesName="Total Sales for category =
"Other category"></y5>
     </QueryInfo>
     <QueryData>
       <Item x="Product 1" y1="0" y2="0" y3="100"
y4="0" y5="0"></Item>
       <Item x="Product 3" y1="0" y2="0" y3="112"
y4="0" y5="0"></Item>
        ............
        <Item x="Product N" y1="0" y2="7"
y3="100" y4="7" y5="1"></Item>
     </QueryData>
</Query>
```

Query node contains the query id and report title attributes. Query id is used to identify and map results to the object making the request. The child nodes are split into two sections. The first node "QueryInfo" contains information about series; it is used to draw the chart's legend and identify series by name. The second child "QueryData" contains the actual data that are used to draw the points on the chart.

III. Results

We present the end results and the capabilities of the software by considering its application in a system that serves telephone fundraising campaigns and is currently used by some of the most prestigious universities in the UK. The implementation of the dashboard solution was required to enable managers and users to access real-time campaign statistics and see exactly how callers and pledges were doing minute by minute with ability to create charts and reports from any of the information contained within the database [7][8].

The first step is to setup the connection details to the database containing campaign data.

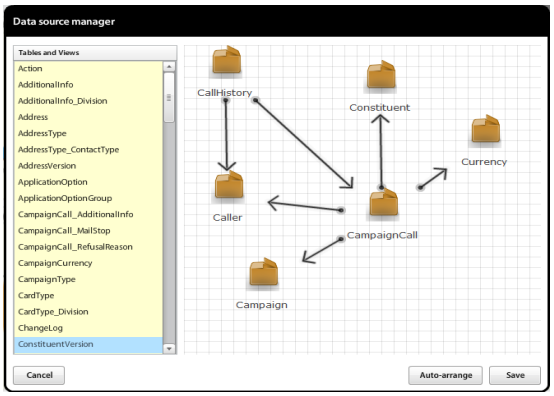


Figure 11-Data source manager

Once the connection is established successfully, the database structure is mapped to corresponding entities and attributes.
Users can then do various adjustments to the model by selecting only the entities and their corresponding attributes required for analysis.

The next step is generating visual graphs using the chart designer component displayed below. Using an intuitive drag and drop interface, this module allows users to create charts by setting the attributes along the two axes. It also presents a rich set of features including the ability to group, filter, sort, and specify the visual representation of the data

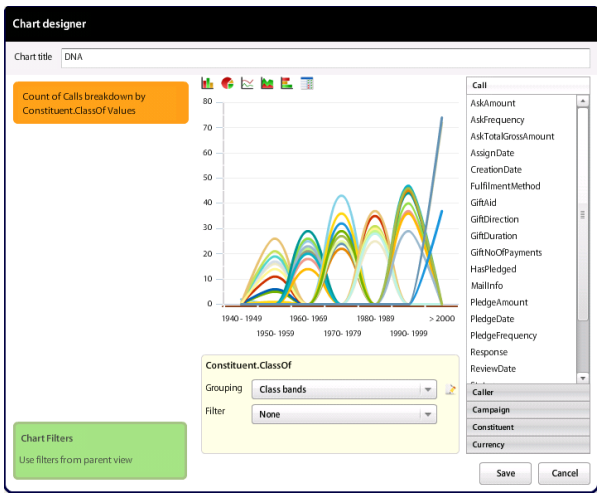


Figure 12-Building charts with chart designer

The final views constitute fully customizable sets of charts enabling users to choose what data to see and how to display it, featuring the ability to drill down, analyse views and trends and export the data.

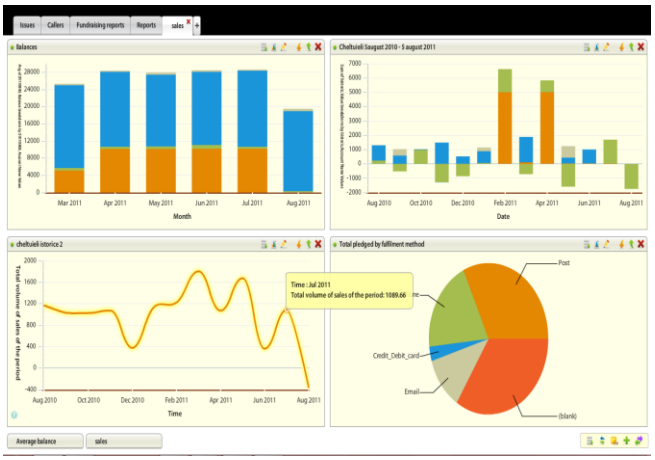


Figure 13-Graphical representation of the data

IV. CONCLUSIONS AND FUTURE WORK

This paper presented a solution for the analysis and exploration of distributed multidimensional databases.
The first contribution of this system is a visual specification for graphically describing the structure of a relational database offering the ability to easily understand and manipulate the entities which reports are based on.

The second contribution of this system is to dynamically build and display relational query results using a rich, expressive set of graphical views. The software features an ample set of operations and filters that can be applied to the data to support this exploratory process through its visual interface. It offers the ability to view overall trends or categorize data and drill-down into areas of interest. There are many plans for future work including: addition of data mining algorithms and techniques, development for various database management systems and support for data stored in different file formats such as: Excel, CVS, etc.

ACKNOWLEDGMENTS

I would like to thank my supervisor Dr Xiang Fei for his support and reviews of this paper. I would also like to thank my colleagues at Exasoft and Bit10 who have been involved in the development and testing of this system and especially our technical team leader Nick Barker.

REFERENCES

[1] The Economist A Special Report on Managing Information: All Too Much | The Economist [online] available from <http://www.economist.com/specialreports/displaystory.cfm?story_id=15557421>

[2] Grigori, D., Casati, F., Castellanos, M., Dayal, U., Sayal, M., and Shan, M. (2004) 'Business Process Intelligence'. Computers in Industry 53 (3), 321-343

[3] Wikipedia Floyd-Warshall algorithm [online] < https://secure.wikimedia.org/wikipedia/en/wiki/Floyd-Warshall_algorithm>

[4] Rosow, E. and Adam, J. (2004) 'Real-Time Executive Dashboards and Virtual Instrumentation: Solutions for Health Care Systems'. in Clinical Engineering Handbook. ed. by Joseph F Dyro. Burlington: Academic Press, 476-483
Loshin, D. (2003) 'The Value of Business Intelligence'. In Business Intelligence. ed. by Anon San Francisco: Morgan Kaufmann, 11-25

[5] Mohamed Z. Elbashir, Philip A. Collier, Michael J. Davern (2008) 'Measuring the effects of business intelligence systems: The relationship between business process and organizational performance', International Journal of Accounting Information Systems, Volume 9, Issue 3, Eighth International Research Symposium on Accounting Information Systems (IRSAIS), Pages 135-153, ISSN 1467-0895

[6] Unknown Big Data is Less about Size, and More about Freedom [online] available from <http://techcrunch.com/2010/03/16/big-data-freedom/> [29/08/2011]

[7] Fundraising Fundamentals Dashboard | Exasoft PLC [online] available from <http://explc.com/products/fundraising-fundamentals/dashboard> [29/08/2011]

[8] Improving Telephone Fundraising for University Alumni | bit10 ltd [online] available from <http://www.bit10.net/fundraising-fundamentals> [29/08/2011]