BrioQuery 6 – Level III
DISCLAIMER:

• All of the results in this manual are based on using the Training instance of the Employee Appointment data model.

• You will need to create a training OCE to access the training instance.

• If you have problems, contact the WAI Business Analyst at 49-49943.
# Table of Contents

## ABOUT THIS TRAINING

- TRAINING OBJECTIVES ................................................................. 5
  - STANDARD CONVENTIONS FOR DOCUMENTATION ......................... 6
  - KEYBOARD CONVENTIONS ............................................................ 6
  - MOUSE CONVENTIONS ................................................................ 6
- WHAT IS BRIQUER? ....................................................................... 7
- BENEFITS OF BRIQUER ................................................................ 7
- BRIQUER SECTIONS (SECTION PANE) ................................................ 7

## BRIQUER WINDOW FEATURES AND TOOLS

- MENU BAR .................................................................................. 8
  - REQUEST LINE .......................................................................... 8
  - LIMIT LINE ................................................................................ 8
  - SORT LINE ................................................................................ 9
  - TOOLBARS ............................................................................... 9
- SECTION PANE ........................................................................... 11
- CATALOG PANE ........................................................................... 11
- CONTENTS PANE ......................................................................... 11
- STATUS BAR ............................................................................... 12

## USING THE QUERY SECTION ......................................................... 13

- USING AGGREGATE FUNCTIONS .................................................. 13
  - USING THE SUM FUNCTION ...................................................... 13
  - EXERCISE: USE THE SUM FUNCTION ......................................... 14
  - USING THE COUNT DISTINCT FUNCTION ..................................... 17
  - EXERCISE: USE THE COUNT DISTINCT FUNCTION ..................... 17

## NOTES .......................................................................................... 19

- USING STRING FUNCTIONS ........................................................ 21
  - USING THE INITCAP FUNCTION .................................................. 21
  - EXERCISE: USE THE INITCAP FUNCTION ..................................... 22
  - USING THE CONCATENATE FUNCTION ......................................... 28
  - EXERCISE: CONCATENATE THE NAME FIELDS ......................... 28
  - USING THE SUB STRING FUNCTION .......................................... 32
  - EXERCISE: USE THE SUBSTR FUNCTION ................................... 32

## NOTES .......................................................................................... 38

- USING CONDITIONAL FUNCTIONS .............................................. 39
About This Training

This Advanced BrioQuery training assists the learner in the use of the BrioQuery software. The training is intended for an intermediate or advanced user with prior experience in using this type of software and relational databases. Prior knowledge of the use of the Microsoft Windows 2000 operating system is required.

Training Objectives

This training will cover advanced topics in the Query and Results section of the BrioQuery Tool.

In this training you will learn:

- Basics of the BrioQuery Window
- How to create advanced queries using Aggregate functions
- How to create advanced queries using String functions
- How to create computed items within the Results section to build grouping items and if/else statements
- How to query against Excel
- How to build multiple queries using the same bqy file
- How to build multiple queries using different data sources in the same bqy file
- BrioQuery Tips
**Standard Conventions for Documentation**

- Actions are **bolded** and CAPITALIZED
- Special notes are *italicized*
- Button and menu names are set to **14 point** font size and underlined.
- File names, paths or directories are printed in Courier New

**Keyboard Conventions**

- Names of keys that you press during hands-on exercises are in small capital letters, for example, TAB and SHIFT.
- A plus sign (+) between two key names means those keys must be pressed at the same time. For example, “Press ALT+TAB” means to hold down ALT while pressing TAB.
- A comma (,) between two or more key names means that you must press each of the keys consecutively, not together. For example, “Press ALT, T, X” means to press and release each key in sequence. “Press ALT+W, L” means to first press ALT and W together, release them, and then press L.

**Mouse Conventions**

- **Click** means to point to an object and then press and release the left mouse button. The word “click” is used for selecting command buttons, option buttons, and check boxes.
- **Drag** means hold down the mouse button while moving the mouse.
- **Double Click** means to rapidly press and release the mouse button twice.
- **Right Click** means to point to an object and then press and release the right mouse button. Clicking the right mouse button opens a shortcut menu that provides easy access to commands associated with the current action.
What is BrioQuery?
BrioQuery is a query and reporting tool for accessing databases from the Windows desktop. It allows the user to access data that is stored on database servers without understanding the complexity of query programming language. BrioQuery allows the user to create a query (or question) of the database. The information is retrieved in a spreadsheet-like format, and various reports can be created from that information. BrioQuery filters large quantities of data to select only what is desired. It also quickly formats data so results can be evaluated and understood.

Benefits of BrioQuery
➢ User-friendly Tool
➢ Predefined Data Models, Queries, and Reports stored in the repository
➢ User may create queries
➢ User may create reports that help answer management decisions with a reasonable response time
➢ User may focus on information without worrying about file structure
➢ User may save queries for later use or modifications

BrioQuery Sections (Section Pane)
➢ **Query** - Specifies the database fields that are to be retrieved and optional limit and sort conditions on the data values. Multiple queries can be created in a single *.bqy* file.
➢ **DataModel** – A visual representation of an actual database.
➢ **Results** - Displays the data that matches the criteria in a query using a table format.
➢ **Pivot** - Constructs pivot reports summarizing query result data by various categories.
➢ **Report** - Formats reports that display and group the query results.
➢ **Table** - Displays a columnar representation of the data. The table section functions like the Results section in that Pivots and Charts can be based on a table’s dataset.
➢ **Chart** – Creates a 3-D graphic representation of data results.
➢ **Executive Information System (EIS)** – Allows users to build and deploy analytic applications. The EIS section is a pushbutton approach to querying a database. It is a document front-end that makes it easy for end-users to access information.
BrioQuery Window Features and Tools

There are a number of features and tools included in the BrioQuery screen display which are very useful. These include: the Standard Toolbar, the Request Line, Section Panes, and the Status Bar.

The following graphics show menus, toolbars and lines which appears when BrioQuery is launched. This includes the BrioQuery menu bar, the Standard Toolbar, and the Request Line.

**Menu Bar**
The menu bar shown is from the Query section and will be the menu bar shown at startup. Each section of BrioQuery has its own menu bar. While the File and Edit menus are standard across all sections, other menus will appear depending on which section is active.

**Request Line**
The Request Line is the area where named items (columns) are placed to return data from the database.

**Limit Line**
The Limit Line is the area where "limits" or criteria are placed on named items (columns) so that a smaller, more specific, subset of data is returned from the database. To view the Limit Line, CLICK on the **Limits** button in the Section Title Bar.
**Sort Line**
The Sort Line is the area where sort conditions on data fields are placed so that the data is returned in a specific order. To view the Sort Line, CLICK on the Sort button in the Section Title Bar.

In BrioQuery, typically there are at least three methods for performing most activities, a button on a Toolbar, a menu option, or a drag and drop method. If you choose not to use the Standard Toolbar, you have the option to use various menus to perform the same functions. Features from some of the menus are duplicated in the Standard Toolbar.

**Toolbars**
The Standard Toolbar can be opened using the View menu. Whenever a query is created or opened this toolbar will be displayed. You can choose not to display this toolbar; however it is extremely useful. When you choose not to display the toolbar, it will not reappear when the tool is opened again.

In BrioQuery, typically there are at least three methods for performing most activities, a button on a Toolbar, a menu option, or a drag and drop method. If you choose not to use the Standard Toolbar, you have the option to use various menus to perform the same functions. Features from some of the menus are duplicated in the Standard Toolbar.
The functions available via the buttons in the Standard Toolbar are:

<table>
<thead>
<tr>
<th>Icon</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>☐</td>
<td>New Document</td>
</tr>
<tr>
<td>☐</td>
<td>Open BrioQuery Document</td>
</tr>
<tr>
<td>☐</td>
<td>Save BrioQuery Document</td>
</tr>
<tr>
<td>☐</td>
<td>Print</td>
</tr>
<tr>
<td>☐</td>
<td>Print Preview</td>
</tr>
<tr>
<td>☐</td>
<td>Remove</td>
</tr>
<tr>
<td>☐</td>
<td>Show Section/Catalog</td>
</tr>
<tr>
<td>☐</td>
<td>Insert New Section</td>
</tr>
<tr>
<td>☐</td>
<td>Properties</td>
</tr>
<tr>
<td>☐</td>
<td>Limit</td>
</tr>
<tr>
<td>☐</td>
<td>Sort Ascending</td>
</tr>
<tr>
<td>☐</td>
<td>Sort Descending</td>
</tr>
<tr>
<td>☐</td>
<td>Apply Sum</td>
</tr>
<tr>
<td>☐</td>
<td>Group Labels</td>
</tr>
<tr>
<td>☐</td>
<td>Process</td>
</tr>
<tr>
<td>☐</td>
<td>Connection Manager</td>
</tr>
<tr>
<td>☐</td>
<td>Back</td>
</tr>
<tr>
<td>☐</td>
<td>Forward</td>
</tr>
<tr>
<td>☐</td>
<td>EIS Home</td>
</tr>
<tr>
<td>☐</td>
<td>Help</td>
</tr>
</tbody>
</table>

The Formatting toolbar is available in the View Menu. This toolbar will allow you to customize your reports to highlight important information and polish the look.

The Formatting toolbar is a graphical representation of the functions supported by the various options in the Format menu. The Format menu is useful for making changes in all the sections: the Results section, the Pivot section, the Table section, and the Report section.

The Section toolbar is only available in specific sections and provides commands to use in those sections.
**Section Pane**
The **Section Pane** is located to the left of the **Contents Pane**. The **Section Pane** allows quick access to every section of the BrioQuery window: the **Query** section, the **Results** section, the **Pivot** section, the **Chart** section, and the **Report** section. There can be multiple Pivots, Charts and/or Reports, and therefore multiple sections. **Pivot**, **Report** and **Chart** do not automatically appear in the **Section Pane**. To create them, you must insert them using the **Insert** menu. To view the different sections, **CLICK** on each section and view it in the **Contents Pane** or you can use the forward and back arrows in the **Section Title bar**.

**Catalog Pane**
The **Catalog Pane** is also located to the left of the **Contents Pane**. The **Catalog Pane** contains the data and objects that are used to build Pivot Reports, Charts, Tables and Reports. The **Catalog Pane** changes depending on the section you are working in. To use the **Catalog Pane**, drag an object from the **Catalog Pane** to the **Contents Pane** or to the **Outliner**.

**Contents Pane**
The **Contents Pane** is the lower right area of the window. The **Contents Pane** provides a view of the section you are using. It will show a Data Model, a Query, a Report, a Pivot or a Chart.
Status Bar
At the very bottom of the BrioQuery window is the Status Bar. This feature gives you information about the number of rows returned by a query, the number of columns selected for the query, a graphic to show whether the database connection is active, and an information area that provides help information when you move the cursor to different areas of the window. The status bar will provide status statements during connection and processing.
Using the Query Section

The Query Section is the foundation of any BrioQuery file. In this section you can connect to a database server through database connection files. Once you are connected to a database server, the query section is used to load pre-built data models or queries. In addition, the query section allows you to build your own data models using the Table Catalog, or build your own queries by applying limits and sorts and processing your request for data. You must build a query first before any other section can be inserted (i.e. report, pivot, chart). All of the exercises in this manual will be done using a Training datamodel, with the host dssqa. For detailed instructions on creating a training OCE, see the Open Catalog Extensions section in the Brio Query 6 Training Guide.

#### Using Aggregate Functions

Aggregate functions are used to compute aggregate values that summarize data. Examples of aggregate functions are averages, maximums, counts and sums. You can use these functions to aggregate or summarize data from the server before it is returned to the Results section.

The aggregate functions below are Sum and Count Distinct. You can apply these functions to existing data items to build a computed item. Similar to other fields on the request line, computed items can be used in pivots and reports or re-used to calculate other data.

#### Using the Sum Function

The sum function allows numeric fields to be "summed" or aggregated. This can help reduce the number of rows retrieved thus reducing query performance time. This is especially helpful with large queries. Note that the more fields that are on the request line the less helpful this function is. An example of the Sum function would be:

```
SUM (Transaction_Summary_Dref.Amount )
```
Exercise: Use the Sum Function

Use the Sum function to show the total Full time annual rate by Fiscal Year for the Civil Engineering department (department number 1284). First process the query without using the sum function, then process the query using the sum function so you can see the difference.

TO USE THE SUM FUNCTION:

1. **OPEN** the Employee Appointment data model from the Training folder and log on, if necessary.

2. **CLICK** on the Data Model section and **DOUBLE CLICK** on all the tables to open them.

3. **CLICK** back on the Query section.

4. From the Organization Unit table, **ADD** Dept to the Request Line.

5. From the Employee Appointment table, **ADD** Fisc Yr and Full Time Annual Rate to the Request Line.

6. From the Employee Appointment table, **LIMIT** Dept to “1284”.

7. **PROCESS** the query. The query should return 45 rows.
8. Now let’s process the query using the sum function to see the difference. CLICK back on the Query section.

9. RIGHT CLICK on the Full Time Annual Rate field on the Request Line and SELECT Data Functions from the shortcut menu.

10. CLICK Sum.

The Request Line will look like this:

11. PROCESS the query. This time the query should return 10 rows.

12. Notice that the Full Time Annual Rate field is summed by Fiscal Year.

13. After verifying the results, CLOSE the query.

The final results should look like this:
<table>
<thead>
<tr>
<th>Dept</th>
<th>Fiscal Yr</th>
<th>Full Time Annual Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>1204</td>
<td>1997-98</td>
<td>41,250</td>
</tr>
<tr>
<td>1204</td>
<td>1998-99</td>
<td>42,500</td>
</tr>
<tr>
<td>1204</td>
<td>1999-00</td>
<td>46,240</td>
</tr>
<tr>
<td>1204</td>
<td>1999-01</td>
<td>45,240</td>
</tr>
<tr>
<td>1204</td>
<td>1999-02</td>
<td>49,240</td>
</tr>
<tr>
<td>1204</td>
<td>1999-03</td>
<td>60,525</td>
</tr>
<tr>
<td>1204</td>
<td>1999-04</td>
<td>92,320</td>
</tr>
<tr>
<td>1204</td>
<td>1999-05</td>
<td>64,750</td>
</tr>
<tr>
<td>1204</td>
<td>1999-06</td>
<td>222,060</td>
</tr>
<tr>
<td>1204</td>
<td>1999-07</td>
<td>506,074</td>
</tr>
</tbody>
</table>

Advanced BrioQuery - Query & Results
Using the Count Distinct Function

To retrieve a count of unique occurrences of a data item, use the count distinct aggregate function. This function returns the number of rows in the query where the values are distinct. This function could be used to count document numbers or the number of people. An example of this is:

\[
\text{COUNT (DISTINCT (Transaction.Doc_Nbr))}
\]

Exercise: Use the Count Distinct Function

Use the Count Distinct function to show the total number of staff in each department. The results should list the Department titles and the number of staff in each department. There may be multiple instances of employees’ social security numbers if the employee has more than one appointment at the University.

To use the COUNT DISTINCT function:

1. OPEN the Employee Appointment data model from the Training folder and log on, if necessary.
2. CLICK on the Data Model section and DOUBLE CLICK on all the tables to open them.
3. CLICK back on the Query section.
4. From the Organization Unit table, ADD Dept Title to the Request Line.
5. From the Employee table, ADD Ssn to the Request Line.
6. From the Employee Appointment table, LIMIT Status to “CURRENT”.
7. RIGHT CLICK on the Ssn field on the Request Line and SELECT Data Functions from the shortcut menu. CLICK Count Distinct.

The Request Line will look like this:

8. PROCESS the query. The query should return 69 rows.
9. Notice that the Results show a count of the Social Security numbers in each Department. *This field can now be used in a pivot or report.*

10. After verifying the results, leave the query open to use it with the next exercise.

The final results should look like this:
Using String Functions

The string functions below require the creation of a computed item in BrioQuery. A computed item is a data item that does not actually exist in the database. It is created by building equations to compute data items. You can also apply functions to existing data items to build a computed item. Similar to other fields on your request line, computed items can be used in pivots and reports or reused to calculate other data.

Some of the string functions include: concatenate function, initcap function and the substring function.

Using the InitCap Function

One of the string functions is the Initcap function. Initcap will capitalize the first letter of each word. This function is often used with names. The Initcap function returns a string of characters with the first letter of each word capitalized and the remaining characters in lower case.

In the example below, the initcap function will return the employee’s last name from the employee table with the first letter in uppercase and the remaining letters in lower case (i.e. Doe).
Exercise: Use the Initcap Function

* Use the Initcap function to capitalize only the first character in the employees’ last names and first names. The query should be limited to bring back only the current employees in department 1284, Civil Engineering. Build a report using the employees’ names, staff type, and years of service. Group the employees by department title.

**TO USE THE INITCAP FUNCTION:**

In this exercise, because we are using the same datamodel, we are going to re-build the query rather than loading the datamodel from the repository. If you change datamodels, you must load the new one from the repository.

1. **CLICK** on the Query section of the query used in the previous exercise.

2. **CLICK** on the word Request in the Request Line. *This will select all the fields on the Request Line.*

3. **CLICK** the Remove button. *(Tip: You can also RIGHT CLICK and SELECT Remove.)*

4. **CLICK** the Remove button in the Remove Query Column dialog box.

5. **CLICK** on the word Limit in the Limit Line. *This will select all the fields on the Limit Line.*

6. **CLICK** the Remove button. *(Tip: You can also RIGHT CLICK and SELECT Remove.)*

7. From the Employee table, **ADD** Name Last, and Yrs Service to the Request Line.

8. From the Organization Unit table, **ADD** Department Title to the Request Line.

9. From the Position Code table, **ADD** Staff Type to the Request Line.

10. From the Employee Appointment table, **LIMIT** Status to “CURRENT”.

11. From the Employee Appointment table, **LIMIT** Dept to “1284”.

12. **DOUBLE CLICK** on the Name Last field on the Request Line.
13. In the Item Properties dialog box, INSERT the cursor before the data field in the Definition box. TYPE in Initcap and a space and INSERT parentheses before and after the data field using the buttons at the bottom of the Item Properties dialog box.

The Item Properties dialog box will look like this:

14. CLICK OK.

15. There is another way to perform the same action. For the First Name, we’ll use the Function and Reference buttons to complete the steps. RIGHT CLICK on the Request Line and SELECT Add Computed Item… from the shortcut menu.

16. In the Modify Item dialog box, CHANGE the Name to “First Name”.
17. **CLICK** the **Functions** button to choose what type of string function you wish to use.

18. **SELECT** **String Functions** in the **Functions Categories** box.

19. **SELECT** **Initcap** in the **Functions** box.

20. **CLICK** the **Reference** button.

21. In the **Reference** dialog box, **SELECT** **Employee** in the **Topic** box and **SELECT** **Name First** in the **Item** box. *The Topic box contains the*
available tables and the Item box contains the fields in the table that have been selected.

22. **CLICK OK.** The Functions dialog box reflects the options chosen.

23. **CLICK OK.**

The Modify Item dialog box will look like this:
24. CLICK OK.

25. PROCESS the query. The query should return 7 rows.


27. DRAG Dept Title from the Catalog Pane to the Report Group 1 section of the Outliner.

28. DRAG Name Last, First Name, Staff Type and Yrs Service to the Table Dimensions section of the Outliner.

29. SELECT the Name Last column and CLICK the Sort Ascending button on the Standard toolbar. Notice that both the last name and first name fields only have the first character capitalized. If we had not used the Initcap function, the entire last name and first name would appear in uppercase letters because that is how these fields are stored in the database.

30. After verifying the results, CLOSE the query.

The final report should look like this:
Using the Concatenate Function

The concatenate function is also a string function. It allows you to combine data stored in multiple fields in your database into one string of characters. This is frequently used when the user wants to combine the last name and the first name into one field.

In the example below, employees’ names are stored in the database in three separate fields, first name, middle name and last name. The concatenate function used in the query will return an employee’s last name (Employee.Name_Last) from the employee table and concatenate it with the employee’s First Name (Employee.Name_First) separated by a comma. It will also concatenate the middle initial and separate it from the first name using a comma (i.e. Doe, John, S). The double pipes || are used to concatenate fields. Also notice that the commas are enclosed in single quotes. The single quotes allow you to insert your own text, spacing or punctuation to be concatenated with the data that is returned from the database. The final result is the employee’s name in ‘Last, First, Middle’ format.

The SQL statement in the Definition box should be:

Employee.Name_Last || ', ' || Employee.Name_First || ', ' || Employee.Name_Middle

Exercise:  Concatenate the Name Fields

Concatenate the Name Last and Name First fields and separate them by a comma. The query should be limited to bring back only the current employees in department 1155, Botany
and Plant Pathology, and should return the following fields: Name Last, Name First, Staff Type, Dept Title, FTE and Full Time Annual Rate.

**TO CONCATENATE FIELDS:**

1. **OPEN** the file, Name.bqy.

2. **RIGHT CLICK** on the Request Line and **SELECT** Add Computed Item from the shortcut menu.

3. In the Modify Item dialog box, **CHANGE** the Name to “Full Name”.

4. **CLICK** the Reference button.

5. In the Reference dialog box, **SELECT** Employee in the Topic: box and **SELECT** Name Last in the Item: box.

6. **CLICK OK**.

7. **INSERT** your cursor after Employee.Name Last, **TYPE** a space, and **TYPE** in the double pipes ||. *(The “pipe” key is located above the Enter key on the keyboard.)*

8. **TYPE** another space, single quote, comma, space and single quote, space, double pipes || and space.

9. **CLICK** the Reference button.

10. In the Reference dialog box, **SELECT** Employee in the Topic: box and **SELECT** Name First in the Item: box.

11. **CLICK OK**.

The SQL statement in the Definition box should be:

Employee.Name_Last || ', ' || Employee.Name_First
12. **CLICK OK.**

13. **PROCESS** the query. *The query should return 5 rows.*

   The computed column labeled **Full Name** should return Last Name, First Name (i.e. HUNTER, HOLLY). *You can now add this computed column to a Pivot or Report just like you would any other data field.*

   The final results should look like this:
Note: You can also use the Function and Reference buttons to set up the concat function. For example, Concat (s1, s2) returns text strings s1 and s2 as a concatenated field. Concat ("brio","query") = brioquery
Using the Sub String Function

The sub string function is used to return a specified portion of a string of data. You can use this function when you want to return only a specific part of a longer string of data. One example to look at is phone numbers.

If phone numbers are stored in 123/456-7890 format and we only want to view the format 456-7890, the sub string function is the perfect function to use. In the following example, we only want the part (sub string) of the string that starts at position 4 and ends 8 positions later. To return only those characters, you would set up your sub string function to read:

Substr(“123/456-7890”, 4, 8)

This example returns: 456-7890

In this example, there are a few ideas that need to be generalized in order to apply this formula in other situations. The generic function reads: Substr(s, n, m).

- S = The entire string
- N = The numeric position in the string at which to start the sub string
- M = The number of characters to extract starting at N

The sub string function can also be applied on fields that are returned in a query. An example where this would be used is to apply this formatting to an entire table of phone numbers. You still use the same substr(s, n, m) function, but you use a column name for the variable “s” in the string. For example, Substr(Employee.Phone_Number, 4, 8), will return all of the employees’ phone numbers in the Phone_Number column with the area codes omitted.

Exercise: Use the Substr Function

Use the Substr function to bring back campus phone numbers with only the last 5 digits of the phone number. Use the query that you just built in the example above.

To Use the Substr Function:

1. CLICK on the Query section.
2. RIGHT CLICK on the Request Line.
3. **SELECT** Add Computed Item from the shortcut menu.

4. In the **Name** box of the **Modify Item** dialog box, **TYPE** in “Campus Phone”.

5. **CLICK** on the **Functions** button to choose what type of string function you wish to use.

6. **SELECT** String Functions in the **Functions Categories:** box.

7. **SELECT** Substr in the **Functions:** box.
8. **CLICK** the Reference… button.

9. **SELECT** **Employee** in the **Topic:** box (this is the table).

10. **SELECT** **Office Phone** in the **Item:** box (this is the field).

11. **CLICK OK**.

12. In the **Starting Position** box, **TYPE** in 4. *Remember, this is the numeric position in the string at which to start the sub string.*

13. In the **Number of chars to extract** box, **TYPE** in 5. *Remember, this is the number of characters to extract.*
14. **CLICK OK.**

The Modify Item dialog box should look like this:

15. **CLICK OK.** A new field will appear on your Request Line, titled SUBSTR (Campus Phone).

16. **PROCESS** the query. *The query should return 5 rows.*
17. From the **Insert** menu, **SELECT** New Report.

18. **DRAG** Dept Title from the **Catalog Pane** to the **Report Group1** section of the **Outliner**.

19. **DRAG** Full Name, Staff Type and Campus Phone from the **Catalog Pane** to the **Table Dimensions** section of the **Outliner**.

   Notice that because the substr function was used, the **Campus Phone** field contains only the last 5 digits of the phone number. If we had not used the substr function, the entire phone number would have been returned.

20. **RIGHT ALIGN** the Campus Phone label and the data items in this column by **SELECTING** the column and **CLICKING** the Justify Right button on the **Formatting** toolbar.

21. **RESIZE** the Dept Title field to make all the text viewable.

22. After verifying the results, leave the query open to use it with the next exercise.

*The final report should look like this:*
Note: You can also type the statement with the substr function directly into the Definition box of the Modify Items dialog box instead of using the Function and Reference buttons.
Using Conditional Functions

Conditional functions are used in BrioQuery to dynamically change the output of a query based on conditions that are established. Based upon those conditions, BrioQuery can change the way that the query results are presented in order to make the data more useful. For example, Purdue University employees may be familiar with building codes, but people outside of the University are not. Someone from outside of the University probably does not know the code, FREH, and which building this code refers to. It would waste data storage space to enter the building’s full name to ease name recognition. Instead, in BrioQuery, the decode statement can be used to replace FREH with Freehafer Hall in a query’s output.

Using the Decode Function

The decode function is a conditional function that can be used in BrioQuery to display different values than the actual ones that are returned from a query. Decode (c, exp, val, def) compares the value of item (c) to one or more expressions (exp), and returns the value (val) matched to each expression, or a default (def).

In the query presented next, the location code field from the database is compared to a building code (the expression) and if it is equal, the long name of the building (the value) is returned. If it is not equal, the default value is returned. In this case, the default value is a blank space.
An example of the decode statement:

```
DECODE (Employee.Office_Location, 'KRAN', 'Krannert Building', '')
```

Another way to use the decode statement is to compare two fields within the database. In this case, two values are returned, tested, and based on the result of the comparison; a Y or N is displayed. The Home Department (Employee.Home_Dept) is being retrieved from the Employee table. At the same time the Appointment Department (Employee_Appointment.Dept) is being retrieved from the Employee Appointment table. The values from each table are tested against each other for equality. If the values are equal, the decode function returns a “Y”. If the values are not equal, the decode function returns an “N”. In this example, “N” is the default value. Remember to use single quotes for the text you want to display if the comparison is equal or not equal.

An example of the decode statement:

```
DECODE (Employee.Home_Dept., Employee_Appointment.Dept, ‘Y’, ‘N’)
```
Exercise: Use the Decode Function

Create a query that returns current employees in department 1288. In the query, retrieve the following data from the database: employees’ last names, first names, department number, annual rate and full time annual rate. Create a computed item and use the decode function to compare the employees’ annual rates with their full time annual rates. If the two fields are equal, display a “Y” in the newly created column. If the two fields are not equal, display an “N” in the new column.

To Use the Decode Function:

In this exercise, because we are using the same datamodel, we are going to re-build the query rather than loading the datamodel from the repository. If you change datamodels, you must load the new one from the repository.

1. CLICK on the Query section of the query used in the previous exercise.

2. CLICK on the word Request in the Request Line. This will select all the fields on the Request Line.

3. CLICK the Remove button. (Tip: You can also RIGHT CLICK and SELECT Remove.)

4. CLICK the Remove button in the Remove Query Column dialog box.

5. CLICK on the word Limit in the Limit Line. This will select all the fields on the Limit Line.

6. CLICK the Remove button. (Tip: You can also RIGHT CLICK and SELECT Remove.)

7. From the Employee table, ADD Name Last, and Name First to the Request Line.

8. From the Employee Appointment table, ADD Dept, Annual Rate and Full Time Annual Rate to the Request Line.

9. From the Employee Appointment table, LIMIT Status to “CURRENT”.

10. From the Employee Appointment table, LIMIT Dept to “1288”.
11. **RIGHT CLICK** on the Request Line and **SELECT Add Computed Item**… from the shortcut menu.

12. In the **Name** box of the **Modify Item** dialog box, **TYPE** in “Annual rate = to FTAR?”

13. **INSERT** your cursor in the **Definition** box.

14. **TYPE** in “decode” (no quotation marks) and left parenthesis.

15. **CLICK** the **Reference** button to choose the fields you want to compare.

16. In the **Reference** dialog box, **SELECT Employee Appointment** in the **Topic:** box and **SELECT Annual Rate** in the **Item:** box.
17. **CLICK OK**.

18. In the decode expression in the **Modify Item** dialog box, **TYPE** a comma and a space after (Employee_Appointment.Annual_Rate.

19. **CLICK** the **Reference** button again to choose the next field for the comparison.

20. In the **Reference** dialog box, **SELECT** Employee_Appointment in the **Topic:** box and **SELECT** Full Time Annual Rate in the **Item:** box.

21. **CLICK OK**.

22. **INSERT** your cursor at the end of the decode expression in the **Modify Item** dialog box. **TYPE** a comma, a space, a single quotation mark, Y, single quotation mark, comma, space, single quotation mark, N, single quotation mark and right parenthesis. *See the example.*
23. **CLICK** on the **Options** button.

24. **SELECT** **String** for the **Datatype**. **Datatype is located at the bottom of the Item Properties or Modify Items dialog box.** The String Datatype is used in this example because the decode function is returning text, and the datatype of the fields are numbers. Because of this, you must select “String” for the datatype.

![Item Properties dialog box](image)

25. **CLICK** **OK**. **This will place a decode field on the Request Line.**

26. **PROCESS** the query. **The query should return 6 rows.**

27. If the **Annual Rate** field is equal to the **Full Time Annual Rate** field, a “Y” is returned in the newly created column. If the two fields are not equal, an “N” is returned.

28. After verifying the results, **CLOSE** the query.
The final results should look like this:

![Table Image]

**Note:** You may have to select the Options button in the Modify Item dialog box and change the datatype depending on what you want the decode function to accomplish. This was shown in the example above. Also see the BrioQuery Help for descriptions of different datatypes. Search on “Adjusting data types”.
Class Exercise

Create a query that returns the last names, first names, staff Ids and campus names for all the current employees in department 1155, 1284 and 1288. Use the concatenate function to return the City, State and Zip code fields as one field with a comma in between each item. Use the initcap function on the campus name field to make it appear in upper and lowercase. Add the campus phone numbers using the substring function to only return the last 5 digits of the phone number.

1. **OPEN** the Repository by **SELECTING** the correct OCE.
2. **LOG** onto the database and **LOAD** the Employee Appointment data model.
3. From the Employee table, **ADD** Name Last, Name First, and Staff ID to the Request Line.
4. **RIGHT CLICK** on the Request Line and **SELECT** Add Computed Item.
5. In the Name box, **TYPE** “Campus Name”.
6. **CLICK** the Functions… button.
7. In the Functions Categories: box, **SELECT** String Functions.
8. In the Functions: box, **SELECT** Initcap.
9. **CLICK** on the Reference… button.
10. In the Topic: box, **SELECT** Organization Unit.
11. In the Item: box, **SELECT** Campus Name.
12. **CLICK OK**.
13. **CLICK OK**
14. **CLICK OK** again. *This will place a computed item on your Request Line.*
15. **RIGHT CLICK** on the Request Line again, and **SELECT** Add Computed Item.
16. In the Name box, **TYPE** “City, State, Zip”.
17. **CLICK** on the Reference… button.
18. In the **Topic:** box, **SELECT Employee**, if necessary.

19. In the **Item:** box, **SELECT Home City**.

20. **CLICK OK**.

21. **INSERT** the cursor after the **Employee.Home_City** field and **TYPE** a space, double pipes || space, single quote, comma, space, single quote, space, double pipes || and a space.

22. **CLICK** on the **Reference...** button.

23. In the **Topic:** box, **SELECT Employee**, if necessary.

24. In the **Item:** box, **SELECT Home State**.

25. **CLICK OK**.

26. **INSERT** the cursor after the **Employee.Home_State** field and **TYPE** a space, double pipes || space, single quote, comma, space, single quote, space, double pipes || and a space.

27. **CLICK** on the **Reference...** button.

28. In the **Topic:** box, **SELECT Employee**, if necessary.

29. In the **Item:** box, **SELECT Home Zip**.

30. **CLICK OK**. *Your Modify Item dialog box should look like this:*

![Modify Item dialog box](image)

31. **CLICK OK**. *This will place another computed item on your Request Line.*
32. **RIGHT CLICK** on the Request Line.

33. **SELECT** Add Computed Item from the shortcut menu.

34. In the **Name** box of the Modify Item dialog box, **TYPE** in “Campus Phone”.

35. **CLICK** the Functions… button.

36. **SELECT** String Functions in the Functions Categories: box.

37. **SELECT** Substr in the Functions: box.

38. **CLICK** the Reference… button.

39. In the **Topic:** box, **SELECT** Employee, if necessary.

40. In the **Item:** box, **SELECT** Office Phone.

41. **CLICK** OK.

42. In the **Starting Position** box, **TYPE** in 4.

43. In the **Number of chars to extract** box, **TYPE** in 5.

44. **CLICK** OK.

45. **CLICK** OK again. *Another new field will appear on your Request Line.*

46. From the Employee Appointment table, **LIMIT** Status to “CURRENT”.

47. From the Employee Appointment table, **LIMIT** Dept to “1155, 1284, 1288”. *Be sure all three department numbers are selected.*

48. **PROCESS** the query. *The query should return 18 rows.*

49. **RESIZE** the columns so all the data is viewable.
The results should look like this:

![Query Result](image)

50. After verifying the results, leave the query open to use it with the next exercise.
Notes
Using the Results Section

When you process a query, BrioQuery displays the data retrieved in the Results section. The data set is placed in columns and rows in this section. Although the data retrieved may be from several different tables, the results are displayed in a single, consolidated table.

Pivots, Charts, Tables and Reports that are created in BrioQuery are all based on the subset of data retrieved to the Results section.

In the Results section, changes can be made to the data set to aid in creating reports. Also, limits can be applied in the Results section to further refine the data set, and you can calculate or group data items.

➢ Grouping Columns

To create a new level or group of data that does not exist in the database, use the Grouping columns feature to merge labels into new groupings. This will allow you to aggregate the associated data. This function operates on the same principle as the Group feature in the Pivot Section.

The difference between grouping columns in the Results section and the Group feature in the Pivot section is the level at which they are implemented. The Group feature in the Pivot section (the paperclip button) enables you to group labels temporarily to help analyze the data within a particular section. Grouping columns, on the other hand, are new items added to your Results section which can then be used in the Report section.

Exercise: Use the Grouping Columns Function

Create a query that returns the current department numbers, department titles and campus names. Once the results are returned, use the Grouping Columns feature to group all the West Lafayette campus department numbers and label them “Main campus – WL”. Then group the remaining campuses (Calumet, PU Fort Wayne, North Central) and label them “Regional campuses”. Create a report that shows the department titles and department numbers and group them by the new campus field (i.e. main campus - WL or regional campuses).
TO USE THE GROUPING COLUMNS FEATURE:

In this exercise, because we are using the same datamodel, we are going to re-build the query rather than loading the datamodel from the repository. If you change datamodels, you must load the new one from the repository.

1. CLICK on the Query section of the query used in the previous exercise.

2. CLICK on the word Request in the Request Line. This will select all the fields on the Request Line.

3. CLICK the Remove button. (Tip: You can also RIGHT CLICK and SELECT Remove.)

4. CLICK the Remove button in the Remove Query Column dialog box.

5. CLICK on the word Limit in the Limit Line. This will select all the fields on the Limit Line.

6. CLICK the Remove button. (Tip: You can also RIGHT CLICK and SELECT Remove.)

7. From the Organization Unit table, ADD Dept Title, Dept, and Campus Name to the Request Line.

8. From the Employee Appointment table, LIMIT Status to “CURRENT”.

9. PROCESS the query. The query should return 122 rows.

10. In the Results section, SELECT the Campus Name column. RIGHT CLICK on this column, and SELECT Add Grouping Column… from the shortcut menu. (You can also SELECT Add Grouping Column… from the Results menu.)

11. In the Grouped Column dialog box, CHANGE the Column Name to “Campus”.

12. **CLICK** the **New Group** button.

13. In the **New Group** box, **TYPE** “Main Campus - WL”.

14. **CLICK** **OK**.

15. **SELECT** “West Lafayette” from the **Available Values** box and **CLICK** the left arrow button to move it to the **Items in Group** box.
16. **CLICK** the New Group button.

17. In the New Group box, **TYPE** “Regional Campuses”.

![New Group window](image1)

18. **CLICK OK**.

19. **SELECT** “CALUMET”, “NORTH CENTRAL”, and “PU FORT WAYNE” from Available Values: box and **CLICK** the left arrow button to move them to the Items in Group: box. You can use the CTRL or SHIFT keys to select multiple items in the Available Values: box.

![Grouped Column: Campus Name window](image2)

20. **CLICK OK**. You should now have an additional column in your Results section titled “Campus”. Scroll down to view the differences in the new grouping. Also note that the new group shows up in blue in the Outliner section.
18. From the **Insert** menu, **SELECT** **New Report**.

19. **DRAG** the **Campus** field to the **Report Group1** section in the **Outliner**.

20. **DRAG** the **Dept Title** and **Dept** fields to the **Table Dimensions** section in the **Outliner**.

21. **RESIZE** the columns so all the data is viewable.

22. **SELECT** the **Dept Title** column and **CLICK** the **Sort Ascending** button on the **Standard** toolbar. This will sort the Department Titles alphabetically.

23. After verifying the results, **CLOSE** the query.
The final report should look like this:

![Advanced BrioQuery - Query & Results](image)

**Note:** You can specify how ungrouped values appear if you choose not to include them in a new grouping. Click the Options button in the Group Column dialog box. Click Null to leave the values ungrouped. Click Default and type in a default name to assign all ungrouped values a default name (i.e. misc for miscellaneous). Click Individual Group to assign each ungrouped values the name originally assigned to it. Any new values that do not meet the defined groupings will not be grouped. You may consider using an if/else statement instead.

**Note:** To modify a grouping column, or to add new groups, select the grouped column in the Results section, and right click on it to enable the shortcut menu. Select Modify Column, and make changes.
If/Else Statements

The If/else statement is a conditional statement. It will execute when a specified condition is true. When a value equals the specified condition (or the statement is true), the statement will perform a certain set of operations. When the specified condition is false, a different set of operations will be executed. This is the “else” part of the statement. If/else statements can include nested if statements. These are called Compound If/else statements. Braces must be used to enclose multiple statements.

Example of If/else statement:
If ( x = ###) {y} else {z}

The brackets {} indicate the action to be performed.

Exercise: Use the If/Else Statement

Create a computed item in the results section. Use an If/Else statement to calculate a 3% raise for any employee making less than $65,000. If an employee is making equal to or greater than $65,000, display “0” in the computed item field. Label the computed item “Raise Amount”.

TO USE THE IF/ELSE STATEMENT:

1. **OPEN** the file, if_else.bqy and log on, if necessary.
2. **CLICK** on the Results section, if necessary.
3. **RIGHT CLICK** in the Contents Pane in the Results section and **SELECT Add Computed Item…** from the shortcut menu.
4. In the **Computed Item** dialog box, **TYPE** “Raise Amount” in the **Name**: box.
5. **CLICK** the if button at the bottom of the **Computed Item** dialog box. *The if statement with parentheses and brackets will be placed in the Definition box.*

6. **INSERT** the cursor between the parentheses and **CLICK** the **Reference** button.

7. **SELECT** Full Time Annual Rate in the **Item:** box

8. **CLICK** **OK**.

9. **INSERT** the cursor after Full Time Annual Rate and before the right parentheses, and **CLICK** the **greater than or equal to** button at the bottom of the **Computed Item** dialog box. **TYPE** 65000.

10. **INSERT** the cursor between the brackets. **TYPE** 0.
11. **INSERT** the cursor after the last bracket and **TYPE** else (Full_Time_Annual_Rate * .03 ). Remember to **USE** the buttons at the bottom of the dialog box to add parentheses and the multiplication symbol and **USE** the **Reference** button when adding the **Full Time Annual Rate** field.

12. **CLICK OK.** You should have a new column in the Results section labeled “Raise Amount” that reflects the specified conditions in the if/else statement. **This field can be used in a pivot or report.**

13. After verifying the results, **CLOSE** the query.
The final results should look like this:

<table>
<thead>
<tr>
<th>Name Last</th>
<th>Name First</th>
<th>Dept</th>
<th>Adjunct</th>
<th>Full Time Annual Pay</th>
<th>Rank Annual Pay</th>
</tr>
</thead>
<tbody>
<tr>
<td>King</td>
<td>Kevin</td>
<td>EDUC</td>
<td></td>
<td>40,000</td>
<td>5,000</td>
</tr>
<tr>
<td>Bond</td>
<td>Sonny</td>
<td>ORG</td>
<td></td>
<td>45,000</td>
<td>5,300</td>
</tr>
<tr>
<td>Cooper</td>
<td>Alice</td>
<td>SOC</td>
<td></td>
<td>46,000</td>
<td>4,400</td>
</tr>
<tr>
<td>Murphy</td>
<td>Eddie</td>
<td>HIST</td>
<td></td>
<td>44,150</td>
<td>3,341</td>
</tr>
<tr>
<td>Carter</td>
<td>Jim</td>
<td>REST</td>
<td></td>
<td>50,120</td>
<td>2,020</td>
</tr>
<tr>
<td>Duncan</td>
<td>Sandy</td>
<td>EDUC</td>
<td></td>
<td>50,849</td>
<td>1,500</td>
</tr>
<tr>
<td>Daphne</td>
<td>Opal</td>
<td>ENVI</td>
<td></td>
<td>58,780</td>
<td>1,821</td>
</tr>
<tr>
<td>Curtis</td>
<td>Tony</td>
<td>AGR</td>
<td></td>
<td>58,500</td>
<td>1,685</td>
</tr>
<tr>
<td>Burnett</td>
<td>Carol</td>
<td>COM</td>
<td></td>
<td>61,500</td>
<td>0</td>
</tr>
<tr>
<td>Dixon</td>
<td>Celeste</td>
<td>PSY</td>
<td></td>
<td>59,020</td>
<td>1,700</td>
</tr>
<tr>
<td>Johnson</td>
<td>Drora</td>
<td>MEC</td>
<td></td>
<td>108,120</td>
<td>0</td>
</tr>
<tr>
<td>Connery</td>
<td>Ben</td>
<td>LIBR</td>
<td></td>
<td>60,000</td>
<td>1,800</td>
</tr>
<tr>
<td>Ford</td>
<td>Harrison</td>
<td>HORT</td>
<td></td>
<td>59,370</td>
<td>0</td>
</tr>
<tr>
<td>Bush</td>
<td>Raymond</td>
<td>BOTY</td>
<td></td>
<td>56,730</td>
<td>0</td>
</tr>
<tr>
<td>Cobb</td>
<td>Bill</td>
<td>CURR</td>
<td></td>
<td>18,000</td>
<td>540</td>
</tr>
<tr>
<td>Pippin</td>
<td>Scotty</td>
<td>CHEM</td>
<td></td>
<td>18,240</td>
<td>487</td>
</tr>
<tr>
<td>Eastwood</td>
<td>Curt</td>
<td>CDAP</td>
<td></td>
<td>18,458</td>
<td>593</td>
</tr>
<tr>
<td>Newman</td>
<td>Paul</td>
<td>SCHO</td>
<td></td>
<td>17,971</td>
<td>539</td>
</tr>
<tr>
<td>Columbus</td>
<td>Christopher</td>
<td>CONT</td>
<td></td>
<td>20,809</td>
<td>934</td>
</tr>
<tr>
<td>Bond</td>
<td>James</td>
<td>OFC</td>
<td></td>
<td>23,029</td>
<td>697</td>
</tr>
<tr>
<td>Mathieu</td>
<td>Walter</td>
<td>SCHO</td>
<td></td>
<td>23,029</td>
<td>1,001</td>
</tr>
<tr>
<td>Brandt</td>
<td>Marlion</td>
<td>FORG</td>
<td></td>
<td>23,009</td>
<td>690</td>
</tr>
<tr>
<td>Curtis</td>
<td>Jamie</td>
<td>SCHO</td>
<td></td>
<td>54,883</td>
<td>1,681</td>
</tr>
<tr>
<td>Sawyer</td>
<td>Diane</td>
<td>ADMA</td>
<td></td>
<td>20,000</td>
<td>816</td>
</tr>
<tr>
<td>Carlos</td>
<td>Duc</td>
<td>LIBU</td>
<td></td>
<td>25,050</td>
<td>676</td>
</tr>
<tr>
<td>Lande</td>
<td>Jessica</td>
<td>CEN</td>
<td></td>
<td>35,970</td>
<td>919</td>
</tr>
<tr>
<td>Craver</td>
<td>Tim</td>
<td>ELEC</td>
<td></td>
<td>25,008</td>
<td>763</td>
</tr>
<tr>
<td>Osborn</td>
<td>Neil</td>
<td>PHYS</td>
<td></td>
<td>44,096</td>
<td>1,501</td>
</tr>
</tbody>
</table>
Class Exercise

Exercise 1

You will be working with two existing files in the following exercises. The first query contains current employees’ names, department numbers and full time annual rates. The query is limited to only return employees from regional campuses. Create a grouping column that includes two groups: “Greater than 40,000” and “Less than 40,000” based on full time annual rates. (Hint: Remember this should be done in the Results section after the query has been processed.)

1. **OPEN** the file, groupingexer.bqy.
2. **PROCESS** the query, if necessary.
3. **SELECT** the Results section, if necessary.
4. **SELECT** the “Full Time Annual Rate” column.
5. **RIGHT CLICK** on the “Full Time Annual Rate column and **SELECT Add Grouping Column…** from the shortcut menu.
6. In the Grouped Column dialog box, **TYPE** “Salary Groups” in the Column Name: box.
7. **CLICK** the New Group button.
8. **TYPE** “Less than $40,000”.

![New Group dialog box](Image)

9. **CLICK OK**.

10. From the **Available Values**: box, **SELECT** all the values under $40,000. Use the **SHIFT** key to **SELECT** multiple values at once.

11. **CLICK** the left arrow button to move the values into the **Items in Group**: box.

12. **CLICK** the **New Group** button.

13. **TYPE** “Greater than $40,000”.

![New Group dialog box](Image)

14. **CLICK OK**.

15. From the **Available Values**: box, **SELECT** all the values greater than $40,000. Use the **SHIFT** key to **SELECT** multiple values at once. *(Hint: *If you cannot see the values, be sure to use the scroll bar to scroll to the top.)*

16. **CLICK** the left arrow button to move the values into the **Items in Group**: box. **You should now have two groups listed in the Groups**: box.


17. **CLICK OK.** A new column should appear in your Results section titled Salary Groups. This column will indicate whether an employee’s Full Time Annual Rate is greater or less than $40,000.

The results should look like this:

18. Now insert a report and group the employees by those with Full Time Annual Rates greater than $40,000 and those with Full Time Annual Rates less than $40,000. From the **Insert** menu, **SELECT New Report.**
19. **DRAG** the **Salary Groups** field to the **Report Group1** section in the **Outliner**.

20. **DRAG** the **Name Last** and **Name First** fields to the **Table Dimensions** section in the **Outliner**.

21. **DRAG** the **Full Time Annual Rate** field to the **Table Facts** section in the **Outliner**.

22. **FORMAT** the **Full Time Annual Rate** Column to be currency with no decimals.

23. **RIGHT CLICK** and **DESELECT** “Show Column Total”, if necessary, so the columns do not subtotal.

24. After verifying the results, **CLOSE** the query.
The final report should look like this:

**EXERCISE 2**

The second query contains current employees from the Calumet and Fort Wayne campuses, along with their department numbers and FTE. Create an if/else statement labeled “Full Time?” that displays the text “Full Time” if the employees have an FTE equal to 1. If the employees do not have an FTE equal to 1, display the text “Part Time”. Remember the if/else statement should be created in the Results section as a computed item.

1. **OPEN** the file, `ifelseexer.bqy`.

2. **CLICK** the Results section, if necessary.

3. **RIGHT CLICK** in the Results section, and SELECT Add Computed Item… from the shortcut menu.
4. In the **Computed Item** dialog box, **TYPE** “Full Time?” in the **Name**: box.

5. In the **Definition** box, **TYPE** in the following if/else statement:
   
   ```
   if (Fte == 1) {'Full Time'} else {'Part Time'}
   ```
   
   (Hint: *Use the Reference button when adding the field Fte to your if/else statement. Use the buttons at the bottom of the dialog box to add the equal symbol (= =).*

6. **CLICK** **OK**. You should have a new column in the Results section labeled “Full Time?” that reflects the specified conditions in the if/else statement. This field can be used in a pivot or report.

7. After verifying the results, **CLOSE** the query.
The results should look like this:

![Query Results Table]

**Note:** For Compound If/else statements see the BrioQuery tips web site listed in the BrioQuery Resources section at the back of this manual.

**Warning:** If you type your own statement in the Definition box, you must use == for equal, or use the button.
Using the Table Section

Tables are reports that display a columnar representation of your database tables and can be used as building blocks in any reporting section.

In many ways, the Table section operates similarly to the Results section. Pivots, Charts, and other reports can be created based on a table's dataset in the same way that these reports are derived from data sets returned in the Results section. When a query is processed, the returned data items are automatically loaded into the Outliner of the Results section. You can then immediately view the data grid.

When a new Table section is inserted while in the Results section, the new Table is attached to the Results section dataset. Any changes made to the dataset of the Results are immediately propagated to the Table section.

Creating Limits in the Table Section

Two primary benefits of the Table section are limits and customized views. Limits in the table section are used to limit the data that is viewed. In the Results section, all the data returned from the database will appear. In the Table section, however, you can limit the data to only see subsets of the results. Unlike the Results section, the Table section will allow you to apply limits to computed items created in the Results section.

Customized views of reports can be created in the Table section. This is helpful because you can create multiple tables with different limits, rather than creating separate queries. For example, you can create a table with employees who have less than 10 years of service, and another table with employees who have greater than 10 years of service within the same query.

Exercise: Use the Table Section

Create several tables to keep track of years of service. The first table should include employees who have between zero and 10 years of service. The second table should include employees who have between 10 and 20 years of service. The third table should include employees who have greater than 20 years of service. Because you cannot aggregate data in
the Table section, you will need to build a Pivot report for each Table that will summarize the data.
TO USE THE TABLE SECTION:

1. **OPEN** the file, Table section.bqy and log on, if necessary.

2. **PROCESS** the query, if necessary.

3. **CLICK** on the **Results** section, if necessary.

4. From the **Insert** menu, **SELECT** New Table.

5. **DRAG** the **Name** and **Yrs Service** fields from the **Catalog Pane** to the **Outliner**.

6. **DRAG** **Yrs Service** from the **Catalog Pane** to the **Limit Line**.

7. In the **Limit** dialog box, **CLICK** the **Show Values** button.

8. In the **Limit: Yrs Service** dialog box, **SELECT** “0.48”, **HOLD DOWN** the **SHIFT** key, scroll down and **SELECT** “9.46”.
9. **CLICK OK.**

10. From the Insert menu, **SELECT New Pivot.** Since the Table Section will not aggregate data, you can move it into the Pivot section to do this.

11. **DRAG** the Name field from the Catalog Pane to the Side Labels section of the Outliner.

12. **DRAG** the Yrs Service field from the Catalog Pane to the Facts section of the Outliner.

13. **RESIZE** the Name column so all the names are viewable.

14. **CLICK** on the tab beneath the Name column, and **SELECT** the Sum or Grand Total button on the Standard toolbar.

15. **DOUBLE CLICK** on the Table Section and **RENAME** it to “Table 0 to 10 yrs”.

16. **CLICK OK.**

17. **CLICK** on the Results section. *It is necessary to go back to the Results section to work with the complete set of data. Or you can duplicate the Table section and change the limits.*

18. From the Insert menu, **SELECT New Table.**

19. **DRAG** the Name and Yrs Service fields from the Catalog Pane to the Outliner.
20. **DRAG** Yrs Service from the **Catalog Pane** to the **Limit Line**.

21. In the **Limit** dialog box, **CLICK** on the **Show Values** button.

22. In the **Limit: Yrs Service** dialog box, **SELECT** “10.89”, **HOLD DOWN** the **SHIFT** key, scroll down and **SELECT** “19.08”.

23. **CLICK OK**.

24. From the **Insert** menu, **SELECT** **New Pivot**. *Since the Table Section will not aggregate data, you can move it into the Pivot section to do this.*

25. **DRAG** the **Name** field from the **Catalog Pane** to the **Side Labels** section of the **Outliner**.

26. **DRAG** the **Yrs Service** field from the **Catalog Pane** to the **Facts** section of the **Outliner**.

27. **RESIZE** the **Name** column so all the names are viewable.

28. **CLICK** on the tab beneath the **Name** column, and **SELECT** the **Sum** or **Grand Total** button on the **Standard** toolbar.

29. **DOUBLE CLICK** on the **Table** Section and **RENAME** it to “Table 10 to 20 yrs”.

30. **CLICK OK**.
31. This time, we’ll duplicate the Table Section and change the limits. **CLICK** on the **Edit** menu and **SELECT** Duplicate Section.

32. **DOUBLE CLICK** on Yrs Service on the Limit Line.

33. In the **Limit:Yrs Service** dialog box, **SELECT** “21.92”, **HOLD DOWN** the **SHIFT** key, scroll down and **SELECT** “24.75”.

34. **CLICK** OK.

35. From the **Insert** menu, **SELECT** New Pivot. *Since the Table Section will not aggregate data, you can move it into the Pivot section to do this.*

36. **DRAG** the Name field from the **Catalog Pane** to the **Side Labels** section of the **Outliner**.

37. **DRAG** the Yrs Service field from the **Catalog Pane** to the **Facts** section of the **Outliner**.

38. **RESIZE** the Name column so all the names are viewable.

39. **CLICK** on the tab beneath the Name column, and **SELECT** the Sum or Grand Total button on the **Standard** toolbar.

40. **DOUBLE CLICK** on the Table Section and **RENAME** it to “Table Greater than 20 yrs”. *The data from each table section can be added to a report.*

*The final pivots should look like this:*
41. After verifying the results, CLOSE the query.

Class Exercise
The query should contain the current employees from the regional campuses along with their department numbers and the split appointment flag. Create two tables, one that limits the employees to only those with a split appointment and one that shows all employees that do not have a split appointment. Under each table, create a pivot report to better display the data.

1. **OPEN** the file, `tableexer.bqy`.
2. **PROCESS** the query and **CLICK** the Results section, if necessary.
3. **INSERT** a new Table.
4. **ADD** “Name Last”, “Name First”, “Dept” and “Split Appt” to the Outliner.
5. **LIMIT** the “Split Appt” field to “N”.
6. **RENAME** Table 1 to “Non Split Appointments”.
7. **SELECT** the Insert menu and **SELECT** New Pivot.
8. **ADD** “Name Last”, “Name First”, and “Dept” to the Side Labels of the Outliner.
9. **CLICK** back on the Results section to create another table.
10. **SELECT** the Insert menu and **SELECT** New Table.
11. **ADD** “Name Last”, “Name First”, “Dept” and “Split Appt” to the Outliner.
12. **LIMIT** the “Split Appt” field to “Y”.
13. **RENAME** Table 2 to “Split Appointments”.
14. **INSERT** a new Pivot.
15. **ADD** “Name Last”, “Name First”, and “Dept” to the Side Labels of the Outliner.
The two Pivot reports should look like this:

**Pivot 1**

![Pivot 1](image1)

**Pivot 2**

![Pivot 2](image2)

16. After verifying the results, leave the query open to use it with the next exercise.
Using Multiple Queries in a Single File

A single BrioQuery file can contain multiple query sections. Each query can be associated with the same data model or different data models. The connection file is identified within each query. In addition, each query will have its own Results section.

You can use multiple queries to combine results from two separate databases. Multiple results in a single file will then allow you to create reports that integrate data from multiple data sources.

Two Queries using the Same Data Source

BrioQuery allows the creation of more than one query in one BQY file. Additionally, you are able to store two queries in the same BQY file that access either the same data model or use two different data sources. In this exercise we will look at two queries that use the same data model. Once the two queries are created you will have two queries that act independently but are stored in the same file. You will be able to create two separate queries with different limits, and be able to display the results together in a single report.

Exercise: Create Two Queries in one BQY

Create the first query and have it return current employees from the West Lafayette campus who have split appointments. Create the second query and return current employees from the Calumet campus who have split appointments. Insert a new report and add the data for West Lafayette in one table and the data for Calumet in another table. Place the tables side by side. Add fields to label the tables.

TO CREATE TWO QUERIES IN ONE BQY:

In this exercise, because we are using the same datamodel, we are going to re-build the query rather than loading the datamodel from the repository. If you change datamodels, you must load the new one from the repository.
1. CLICK on the **Query** section of the query used in the previous exercise.

2. CLICK on the word **Request** in the **Request Line**. *This will select all the fields on the Request Line.*

3. CLICK the **Remove** button. *(Tip: *You can also RIGHT CLICK and SELECT Remove.)*

4. CLICK the **Remove** button in the **Remove Query Column** dialog box.

5. CLICK the **Remove** button until you no longer get the **Report Refresh** dialog box.

6. CLICK on the word **Limit** in the **Limit Line**. *This will select all the fields on the Limit Line.*

7. CLICK the **Remove** button. *(Tip: *You can also RIGHT CLICK and SELECT Remove.)*

8. From the **Employee** table, **ADD** **Name Last** and **Name First** to the **Request Line**.

9. From the **Employee Appointment** table, **ADD** **Dept** to the **Request Line**.

10. From the **Organization Unit** table, **ADD** **Campus Name** to the **Request Line**.

11. From the **Employee Appointment** table, **LIMIT** **Status** to “CURRENT”.

12. From the **Employee Appointment** table, **LIMIT** **Split Appt** to “Y”.

13. From the **Organization Unit** table, **LIMIT** **Campus Name** to “West Lafayette”. *(Hint: Remember you can use the Show Values button since Organization Unit table is a dimension table.)*

14. PROCESS the query. *The query should return 23 rows.*

15. RIGHT CLICK on the **Results** section and RENAME it to “Results – WL”.

16. Now create your next query. From the **Insert** menu, SELECT **New Query**.

17. When prompted, SELECT **Yes** to link the new query to the master Datamodel.
18. A new Query section has been added. **ADD** Name Last and Name First from the Employee table to the Request Line.

19. From the Employee Appointment table, **ADD** Dept to the Request Line.

20. From the Organization Unit table, **ADD** Campus Name to the Request Line.

21. From the Employee Appointment table, **LIMIT** Status to “CURRENT”.

22. From the Employee Appointment table, **LIMIT** Split Appt to “Y”.

23. From the Organization Unit table, **LIMIT** Campus Name to “Calumet”. (Hint: Remember you can use the Show Values button since Organization Unit table is a dimension table.)

24. **PROCESS** the query. *The query should return 2 rows.*

25. **RIGHT CLICK** on the Results section and **RENAME** it to “Results – Calumet”. *The two queries that have been created are independent of each other. However, in the report section, you can now display the results of the two queries together.*

27. DRAG “Name Last”, “Name First”, and “Dept” from Results-WL from the Catalog Pane to the Table Dimensions in the Outliner.

28. RESIZE the columns to make them smaller.

29. DOUBLE CLICK on Query2 in the Catalog Pane.

30. DOUBLE CLICK on Results - Calumet in the Catalog Pane.


32. RIGHT CLICK and SELECT Insert Table from the shortcut menu. *This will insert another table for the Calumet data.*

33. DRAG and DRAW the table to the right of the existing table.
34. **DRAG** “Name Last”, “Name First”, and “Dept” from Results - Calumet from the Catalog Pane to the Table2 Dimensions in the Outliner.

35. **RESIZE** the columns to make them smaller so all the data is viewable.

36. **ARRANGE** the two tables so they are side by side.

37. **MOVE** the tables down to allow space above them to insert a field.

38. From the Report menu, **SELECT** Insert Field. **CLICK** above the first table to insert the Empty field.

39. **SELECT** the Empty field.

40. In the Expression bar, delete the Empty field and **TYPE** “Split Appointments on the WL Campus”. Be sure to use quotation marks.

41. **PRESS** the Enter key.

42. **SELECT** the new field and **CHANGE** the font to size 10 and **BOLD** it. **RESIZE** the field so all the text is viewable.

43. From the Report menu, **SELECT** Insert Field to insert another field. **CLICK** above the second table to insert the Empty field.

44. **SELECT** the Empty field.

45. In the Expression bar, delete the Empty field and **TYPE** “Split Appointments
on the Calumet Campus”. Be sure to use quotation marks.

46. **PRESS** the **Enter** key.

47. **SELECT** the new field and **CHANGE** the font to size 10 and **BOLD** it. **RESIZE** the field so all the text is viewable.

48. After verifying the results, **CLOSE** the query.

*The final report should look like this:*
Querying an Excel File

BrioQuery can be used to query data sources other than databases, for example, Microsoft Excel. Microsoft Excel can store data in rows and columns similar to the way data is stored in a database. By setting up a Microsoft Excel file with rows and columns, BrioQuery can be used to access the data stored in the spreadsheet.

There are four main steps involved in the process of querying an Excel spreadsheet. First of all, you must prepare the Excel spreadsheet by formatting your data in tables. Next, a driver must be created that will serve as the bridge between BrioQuery and Excel. Third, in BrioQuery, you create an OCE that points to the Excel file. From there, the data in Excel can be used in the same ways as the data from a database. Finally, the fourth step is to create the query using the data stored in the Excel spreadsheet.

Exercise: Query an Excel File

Follow the four steps to connect to and query an Excel file. Have the query return the following fields: Dept, staff Id, Name Last, Name First and Email.

STEP 1: PREPARE THE SPREADSHEET

1. OPEN the file, email addresses.xls in Excel.

2. SELECT the cells, A1:E15 to include in the table.
   In order for BrioQuery to recognize the data properly, the data must be formatted in columns with the first row selected indicating the field name.

3. From the Insert menu, SELECT Name and SELECT Define.

4. In the Define Name dialog box, TYPE the name desired for the table. TYPE “Email_Addresses”.


5. **CLICK** the **Add** button.

6. **CLICK OK**.

7. **CLOSE** Excel and **CLICK YES** to save changes to the spreadsheet.

---

**STEP 2: CREATE THE ODBC DRIVER FOR EXCEL**

1. **OPEN** the Control Panel by **CLICKING** on the Start Menu and **SELECTING** Settings then Control Panel. *This may be different depending on the configuration of your workstation.*

2. **DOUBLE-CLICK** Data Sources (ODBC) Icon.

   *If you do not see Data Sources (ODBC), you may need to **DOUBLE-CLICK** the Administrative Tools icon first.*

3. If you are a Windows 2000 user, **SELECT** the **User DSN** tab. *The tab may not
have any system data sources listed.

If you are a Windows NT user, SELECT the System DSN tab. The tab may not have any system data sources listed.

4. SELECT Excel Files.

5. CLICK the Add button to add an Excel driver.

6. LOCATE the driver, “Microsoft Excel Driver (*.xls)”. 
7. **CLICK** Finish.

8. **TYPE** in the desired name of this driver. **TYPE** “Excel email” in the **Data Source Name** and the **Description** box. *Make sure to name it something that relates to the document you are going to point to.*

![ODBC Microsoft Excel Setup](image1)

9. **CLICK** the **Select Workbook** button.

10. **LOCATE** the desired file.

![Select Workbook](image2)

11. **CLICK** **OK**.

12. **CLICK** **OK** again.

13. **CLICK** **OK** again.

14. **CLOSE** the **Control Panel**.
STEP 3: CREATE THE OCE THAT POINTS TO YOUR EXCEL FILE

1. In BrioQuery, CLOSE all open queries, and SELECT New from the File menu.

2. SELECT A New Database Connection File.

3. CLICK OK.

4. SELECT ODBC from both drop-down menus.

5. CLICK Next.
6. In the **Host** box, **SELECT** “Excel email” from the drop down box. *This is the name of the driver you created in the top portion of these instructions. A username and password is not necessary unless your spreadsheet is password protected.*

7. **SELECT** **Next**.

8. **SELECT** **Finish**.

9. **SELECT** **Yes** to save your OCE.
10. **ENTER** the desired name of your OCE. **TYPE** Excelemail and **CLICK** Save.

**STEP 4: CREATE THE QUERY**

If Step 4: Create the Query is done immediately after Step 3: Create the OCE, you can skip to #7. If Step 4: Create the Query is not done immediately after Step 3: Create the OCE, you will need to begin with #1.

1. In BrioQuery, **CLOSE** all open queries, and **SELECT** New from the File menu, if necessary.

2. **SELECT** Recent Database Connection Files. If the OCE does not display in the box, **SELECT** the Browse button.

3. **SELECT** the OCE, Excelemail.occ.
4. **CLICK OK.**

5. **CLICK OK.** You only need to enter a **Host User** and **Host Password** if the file is password protected.

6. If necessary, **CLICK** on the **Query** section.

7. **DOUBLE-CLICK** on **Tables** in the **Catalog Pane.** A list of the tables that you defined in Excel should appear.

8. **SELECT** the desired table(s) and **DRAG** them onto the **Contents Pane.**
   **DRAG** the **Email_Addresses** table into the **Contents Pane.**

9. Now you can **CREATE** the desired query. **CLICK** on the title bar of the table,
Email Addresses, to SELECT the table.

10. **RIGHT CLICK** and SELECT **Add Selected Items** from the shortcut menu. This will add all of the data fields in the table to the **Request Line**.

11. **PROCESS** the query. *The query should return 14 rows.*

*The final results should look like this:*

```
<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
</table>
| 1 | 1155 | 00010986 | 4 | HARR
| 2 | 1155 | 00014919 | 2 | CROW
| 3 | 1155 | 00021240 | 2 | HARR
| 4 | 1155 | 00026890 | 4 | HARR
| 5 | 1264 | 00019959 | 7 | EDDARDS
| 6 | 1264 | 00024529 | 3 | HARR
| 7 | 1264 | 00034692 | 4 | HARR
| 8 | 1264 | 00041114 | 6 | HARR
| 9 | 1264 | 00050919 | 3 | HARR
| 10 | 1264 | 00051405 | 2 | HARR
| 11 | 1264 | 00051856 | 5 | HARR
| 12 | 1264 | 00080502 | 1 | HARR
| 13 | 1264 | 00013995 | 8 | HARR
| 14 | 1264 | 00091830 | 7 | HARR
```

12. After verifying the results, leave the query open to use it with the next exercise.
Two Queries in One BQY File

BrioQuery has the capability to store multiple queries in one file. You may want to create two queries in the same BQY file that access different data sources (i.e. a Microsoft Excel file and an Oracle database). In this case, both queries can be created in the same BQY file, independently of each other. A third query must then be created to combine the results of the two previously created queries. You can then use these combined results to produce reports, pivots, and charts. When creating multiple queries from different data sources, you cannot use locked data models. You must use the individual tables to create your queries. When using individual tables, the tables may automatically be joined. You must know your data well enough to be able to delete any joins that are not needed.

Exercise: Query an Excel File and an Oracle Database

Create two queries, one using an Excel spreadsheet and the other using the Employee Appointment training database. Since email addresses are not stored in the Employee Appointment training database, we need to obtain this data from our Excel spreadsheet. Once the two queries are processed, create a third query with no connection to combine the results. Create a report and add the following fields to the Table Dimensions in the Outliner: Name Last, Name First, Office Location, Office Phone, and Email.

CREATE TWO QUERIES IN ONE BQY USING DIFFERENT DATA SOURCES:

1. CREATE the first query in Excel. This was done in the section above.

2. CREATE a query that accesses an Oracle database. From the Insert menu, SELECT New Query.

The following message appears.

3. CLICK No.

4. SELECT Training.oce.
5. **CLICK OK**.

6. **TYPE** in the Host User and Host Password.

7. **CLICK OK**.

8. **DOUBLE CLICK** on the Tables in the Catalog Pane.

9. **DRAG** the Employee table and the Employee Appointment table to the Contents Pane.

10. From the Employee table, **ADD** Staff ID, Name Last, Name First, Office Location and Office Phone to the Request Line.

11. From the Employee Appointment table, **LIMIT** Status to “CURRENT”.

12. From the Employee Appointment table, **LIMIT** Dept to “1155”, “1284” and “1288”. *Be sure all three departments are selected.*
13. **PROCESS** the query. *The query should return 18 rows.*

*The results should look like this:*

```
14. From the **Insert** Menu, **SELECT New Query**.  
   *This is the query that will be used to combine the results of the two queries that were created in steps 2 and 3.*

15. **CLICK No** to the following message:

```
Auto Logon

Automatically Logon using an existing connection? [Yes] [No]

Excelemail.occ

16. **CLICK No Connection**.
17. **CLICK OK.**

18. **RIGHT CLICK** on the **Tables** in the **Catalog Pane** and **SELECT Local Results.**

19. **DOUBLE CLICK** on **Local Results** (or **CLICK** on the + sign) in the **Catalog Pane.**

20. **DRAG** **Results** and **Results2** into the **Contents Pane.**
21. **LINK** the two tables by **CLICKING** on the **Staff ID** field in the **Results2** table and **DRAGGING** it over the **Staff ID** field in the **Results** table. The join will appear as a line between the two tables.

22. From the **Results2** table, **ADD** all the fields to the **Request Line**.
23. From the Results table, **ADD** the Email field to the Request Line.

![Query3](image)

24. **PROCESS** the query. *The query should return 18 rows.*

Results3 will combine the results of the two queries and should look like this:

![Results3](image)

25. You can now create reports, pivots, and charts with the combined results. **CLICK** on the Insert menu and **SELECT** New Report.

26. In the Catalog Pane, **DOUBLE CLICK** on Query3 (or **CLICK** on the + sign) and then **DOUBLE CLICK** on Results3 to view the fields that can be added to the report.

27. **ADD** the Name Last, Name First, Office Location, Office Phone and Email fields to the Table Dimensions section in the Outliner.

28. **RESIZE** the columns in the table so that all the data is viewable. After verifying the results, **CLOSE** the query.
The final report should look like this:
Class Exercise

Create two queries in the same bqy file using the same database. In the first query, use the Employee Appointment database to return current clerical and service employees on the West Lafayette campus who have an hourly rate that is less than or equal to $10. In the second query, use the Employee Appointment database to return current clerical and service employees on the West Lafayette campus who have an hourly rate that is greater than $10. Create a report and insert the results of both queries into separate tables. Create a label for each table titled, “Staff with Hourly Rates <= $10” and “Staff with Hourly Rates > $10”.

1. OPEN the Repository and SELECT the Training OCE.

2. LOG onto the database and LOAD the Employee Appointment data model.

3. ADD the Name Last and Name First fields to the Request Line.

4. ADD the Hour Rate, Pos Code and Dept fields to the Request Line.

5. From the Organization Unit table, LIMIT Campus Name to “West Lafayette”.  
   Remember, you can use the Show Values button since Organization Unit is a dimension table.

6. LIMIT Status to “CURRENT”.

7. LIMIT Hour Rate to less than or = to “10”.

   ![Limit Hour Rate dialog box](image)
8. **LIMIT** Staff Type to “Clerical” and “Service”. Be sure they are both selected. *Remember, you can use the Show Values button since Position Code is a dimension table.*

9. **PROCESS** the query. *The query should return 10 rows.*

   The first query’s results should look like this:

10. Now create the next query. From the **Insert** menu, **SELECT New Query**.
11. When prompted, **SELECT Yes** to link the new query to the master Datamodel.

![BrioQuery dialog box](image1.png)

12. A new **Query** section has been added. **ADD** the **Name Last** and **Name First** fields to the **Request Line**.

13. **ADD** the **Hour Rate**, **Pos Code** and **Dept** fields to the **Request Line**.

14. **LIMIT** **Campus Name** to “West Lafayette”.

15. **LIMIT** **Status** to “CURRENT”.

16. **LIMIT** **Hour Rate** to greater than “10”.

![Limit: Hour Rate dialog box](image2.png)
17. **LIMIT** Staff Type to “Clerical” and “Service”. Be sure they are both selected.

![Limit: Staff Type dialog box]

18. **PROCESS** the query. *The query should return 6 rows.*

*The second query’s results should look like this:*

![Results dialog box with sample data]

19. **INSERT** a new report.

20. **ADD** Name Last, Name First, Dept and Hour Rate from Results to the Table Dimensions in the Outliner.
21. **DOUBLE CLICK** on **Query2** (or **CLICK** on the + sign) in the **Catalog Pane**.

22. **DOUBLE CLICK** on **Results2** (or **CLICK** on the + sign) in the **Catalog Pane**.

23. **CLICK** in the **Body** of the **Report** and **RESIZE** it to make it larger.

24. **RIGHT CLICK** and **SELECT** **Insert Table** from the shortcut menu. *This will insert another table for the Results2 data.*

25. **DRAG** and **DRAW** the table under the existing table.

26. **ADD** **Name Last**, **Name First**, **Dept** and **Hour Rate** from **Results2** to the **Table Dimensions2** in the **Outliner**.

27. From the **Report** menu, **SELECT** **Insert Field**, **CLICK** above the first table to insert the **Empty field**. You may have to drag the tables down to insert the empty field above the first table.

28. **SELECT** the **Empty field**.

29. In the **Expression** bar, delete the **Empty field** and **TYPE** “Staff with Hourly Rates <= $10”. Be sure to use quotation marks.

30. **PRESS** the **Enter** key.
31. From the Report menu, SELECT Insert Field again. CLICK above the second table and insert an Empty field.

32. SELECT the Empty field.

33. In the Expression bar, delete the Empty field and TYPE “Staff with Hourly Rates >$10”. Be sure to use quotation marks.

34. PRESS the Enter key.

35. SELECT each of the new fields and CHANGE the font to size 12 and BOLD the fields.

36. RESIZE the fields so all the text is viewable.

37. FORMAT the numbers in the Hour Rate columns to Currency.

38. RIGHT ALIGN the Hour Rate labels.

The final report should look like this:
# BrioQuery Resources

## What Function Should I Use?

<table>
<thead>
<tr>
<th>Section</th>
<th>Function</th>
<th>Benefits</th>
<th>Tradeoffs?</th>
<th>Level of Difficulty</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Query</strong></td>
<td>Computed items and aggregates</td>
<td>Can reduce the number of rows returned which can improve query performance</td>
<td>Some computed items can be pretty advanced so more knowledge is needed (SQL).</td>
<td>Beginner to Advanced</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Can be used in all sections of Brio</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Results</strong></td>
<td>Grouping Column</td>
<td>Can be used in Pivot, Report or Table sections</td>
<td></td>
<td>Beginner</td>
</tr>
<tr>
<td></td>
<td>If/else statements</td>
<td>Can be used in Pivot, Report or Table sections</td>
<td></td>
<td>Medium</td>
</tr>
<tr>
<td><strong>Pivot</strong></td>
<td>Group Data Labels (paper clip)</td>
<td>Easy to use</td>
<td>Can only be used in Pivot section</td>
<td>Beginner</td>
</tr>
<tr>
<td><strong>Table</strong></td>
<td>Computed Items</td>
<td>Helpful for manipulating data to then use in different reports. For example if you wanted a separate report for each fiscal year, you could set up a table section to focus on one year and then build reports off of the respective table. Helpful in limiting multiple queries with similar information. When you run the query one time, it can update all sections.</td>
<td></td>
<td>Medium</td>
</tr>
</tbody>
</table>
## Limit Table

<table>
<thead>
<tr>
<th>Limit Type</th>
<th>Operation Performed to Select Data</th>
<th>Data on which this comparison is valid</th>
</tr>
</thead>
<tbody>
<tr>
<td>Equal =</td>
<td>Only data where the requested item(s) is equal to the specified value(s) will be retrieved.</td>
<td>text, numbers, dates</td>
</tr>
<tr>
<td>Not Equal &lt;&gt;</td>
<td>Only data where the requested item(s) is NOT equal to the specified value(s) will be retrieved.</td>
<td>text, numbers, dates</td>
</tr>
<tr>
<td>Less Than &lt;</td>
<td>Only data where the requested item(s) is less than the specified value(s) will be retrieved.</td>
<td>text, numbers, dates</td>
</tr>
<tr>
<td>Less or Equal &lt;=</td>
<td>Only data where the requested item(s) is less than or equal to the specified value(s) will be retrieved.</td>
<td>text, numbers, dates</td>
</tr>
<tr>
<td>Greater Than &gt;</td>
<td>Only data where the requested item(s) is greater than the specified value(s) will be retrieved.</td>
<td>text, numbers, dates</td>
</tr>
<tr>
<td>Greater or Equal &gt;=</td>
<td>Only data where the requested item(s) is greater than or equal to the specified value(s) will be retrieved.</td>
<td>text, numbers, dates</td>
</tr>
<tr>
<td>Begins With</td>
<td>Only data where the requested item(s) begin with the specified value(s) will be retrieved.</td>
<td>text</td>
</tr>
<tr>
<td>Contains</td>
<td>Only data where the specified value(s) is somewhere in the requested item(s) will be retrieved.</td>
<td>text</td>
</tr>
<tr>
<td>Ends width</td>
<td>Only data where the requested item(s) ends with the specified value(s) will be retrieved</td>
<td>text</td>
</tr>
<tr>
<td>Limit Type</td>
<td>Operation Performed to Select Data</td>
<td>Data on which this comparison is valid</td>
</tr>
<tr>
<td>------------</td>
<td>-----------------------------------</td>
<td>---------------------------------------</td>
</tr>
</tbody>
</table>
| Like       | Similar to contains; this matches the SQL LIKE command and requires that you use wild card characters to specify where arbitrary characters can be inserted; for most SQL databases such as Oracle, the wild card characters are % (any additional characters to the end of the string) and _ (any single character) when entering the specified value. Additional wild card characters may be available and the wild card characters may vary for different databases. In addition, special characters may need to be entered in a specific fashion.  

  e.g. Job_title like ‘T%’  
would retrieve records for jobs like Testing, Typist, Truckdriver, etc.  

  e.g. Part_number like “F_ABC%”  
would retrieve items with part number F1ABC999 and FKABCn3646-S. | text |
| Between    | Only data where the requested item(s) is between the specified value(s) will be retrieved. Note that data retrieved can also be equal to the specified beginning and ending values. | numbers, text, dates |
| Is Null    | Only data where the requested item(s) has not been set will be retrieved. Blank strings (i.e. spaces) and zero ARE NOT null values. Only items that have not been set to any value are considered to be null.  

  In DSS, no text is set to Null; numbers or dates are set to Null when no value has been collected. For instance SAT score would be Null if the student had not taken the test. | text, numbers, dates |
## Standard Data Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Section Used in:</th>
<th>Operation</th>
<th>SQL Function Generated</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>Query</td>
<td>Returns unaggregated values as stored in the database. This is the default in Query.</td>
<td></td>
</tr>
<tr>
<td>Sum</td>
<td>All</td>
<td>Calculates the sum of the column in each group (numerical data only)</td>
<td>SUM(column definition)</td>
</tr>
<tr>
<td>Average</td>
<td>All</td>
<td>Calculates the average of the column for each group (numerical data only)</td>
<td>AVERAGE(column definition)</td>
</tr>
<tr>
<td>Non-Null Average</td>
<td>Pivot Chart Report</td>
<td>Returns average of underlying values; null values excluded.</td>
<td></td>
</tr>
<tr>
<td>Minimum</td>
<td>All</td>
<td>Returns the smallest entry in each group (numerical data only). You must set the datatype to include text and dates.</td>
<td>MIN(column definition)</td>
</tr>
<tr>
<td>Maximum</td>
<td>All</td>
<td>Returns the largest entry in each group (numerical data only). You must set the datatype to include text and dates.</td>
<td>MAX(column definition)</td>
</tr>
<tr>
<td>Count</td>
<td>All</td>
<td>Counts the number of appearances in each group of each unique value.</td>
<td>COUNT(column definition)</td>
</tr>
<tr>
<td>Count Distinct</td>
<td>Query</td>
<td>Same as Count but excludes identical rows</td>
<td>COUNT(column definition)</td>
</tr>
<tr>
<td>Null Count</td>
<td>Pivot Chart Report</td>
<td>Returns number of nulls among underlying values.</td>
<td>STDDEV(column definition)</td>
</tr>
<tr>
<td>Non-Null Count</td>
<td>Pivot Chart Report</td>
<td>Returns number of underlying values; null values excluded.</td>
<td></td>
</tr>
<tr>
<td>Standard Deviation</td>
<td>Query</td>
<td>Returns standard deviation of values.</td>
<td></td>
</tr>
<tr>
<td>Variance</td>
<td>Query</td>
<td>Calculates the variance of the selected measure evaluated over the specified dataset.</td>
<td>VARIANCE(column definition)</td>
</tr>
<tr>
<td>Weight</td>
<td>Query</td>
<td>Used to compute weighted values in Pivot reports.</td>
<td>WEIGHT(column definition)</td>
</tr>
<tr>
<td>Rank</td>
<td>Pivot</td>
<td>Returns the rank of a number in a column of numbers.</td>
<td>RANK (numbers, break_col)</td>
</tr>
<tr>
<td>Function</td>
<td>Section Used in:</td>
<td>Operation</td>
<td>SQL Function Generated</td>
</tr>
<tr>
<td>-------------------</td>
<td>------------------</td>
<td>---------------------------------------------------------------------------</td>
<td>------------------------</td>
</tr>
<tr>
<td>% of Column</td>
<td>Pivot</td>
<td>Returns sum of all underlying values as a percentage of their respective surface column.</td>
<td></td>
</tr>
<tr>
<td>% of Row</td>
<td>Pivot</td>
<td>Returns sum of underlying values as a percentage of their respective surface row.</td>
<td></td>
</tr>
<tr>
<td>% of Grand</td>
<td>Pivot</td>
<td>Returns sum of underlying values as a percentage of all surface values in the report.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Chart</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Increase</td>
<td>Pivot</td>
<td>Calculates the increase between the previous two rows or columns.</td>
<td></td>
</tr>
<tr>
<td>% Increase</td>
<td>Pivot</td>
<td>Calculates the percentage increase between the previous two rows or columns.</td>
<td></td>
</tr>
<tr>
<td>Title</td>
<td>Report</td>
<td>Returns column names.</td>
<td></td>
</tr>
</tbody>
</table>
# Data Types

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Automatic</td>
<td>BrioQuery will automatically determine the data type returned from the database. This is the default and should be the normal choice.</td>
</tr>
<tr>
<td>Byte</td>
<td>A single byte of computer storage. Bytes can take numeric values from 0 to 255 and can also be used to store a single text character.</td>
</tr>
<tr>
<td>Date</td>
<td>Calendar date. This will be stored in the format selected, typically mm/dd/yy.</td>
</tr>
<tr>
<td>Integer (16 bit)</td>
<td>A 16-bit value (2 bytes). A 16-bit integer can take values from 0 to 65,536. If the integer is signed, it can take values ranging from +32,768 to -32,768.</td>
</tr>
<tr>
<td>Integer (32 bit)</td>
<td>A 32-bit value (4 bytes). A 32-bit integer can take values from 0 to 16,777,216. If the integer is signed, it can take values ranging from +8,388,608 to -8,388,608.</td>
</tr>
<tr>
<td>Long Text</td>
<td>Very long text fields. Maximum length is defined by the database and connection API.</td>
</tr>
<tr>
<td>Real</td>
<td>Numbers with decimal points. Up to 5 positions are allowed to the right of the decimal. The range of values depends on the number of decimal points.</td>
</tr>
<tr>
<td>String</td>
<td>Text characters. Maximum length is 256 characters.</td>
</tr>
<tr>
<td>Time</td>
<td>Time. The format used for times is set by user preference.</td>
</tr>
<tr>
<td>TimeStamp</td>
<td>Date/time combination. The format used is set by user preference.</td>
</tr>
</tbody>
</table>


**World Wide Web URLs**

The following URLs offer assistance when using DSS and BrioQuery.

**BrioQuery Tips Page**
http://www.adpc.purdue.edu/WAI/Brio/brio_tips.htm

**Querying ODS and DSS Using One bqy File**
http://www.adpc.purdue.edu/WAI/Brio/odsfin&dssfin.htm

**DSS Homepage**
http://www.adpc.purdue.edu/WAI/DSS/DSS.htm

**WAI Homepage**
http://www.adpc.purdue.edu/WAI/