# Contents

**Introduction** .................................................................................................................. 4
  - Who is this book for?........................................................................................................ 4
  - What is this book not for?.................................................................................................. 4
  - What is in a recipe?......................................................................................................... 4
  - Difficulty criteria............................................................................................................. 5

**Combination of data sets** ............................................................................................... 7
  - Linking to find all possible values of a column.......................................................... 7
  - Combining rows of multiple tables............................................................................. 8
  - Finding missing values by linking tables.................................................................. 10
  - Adding source information to the description of columns........................................ 12
  - Checking for common column names among tables.............................................. 15
  - Linking columns of tables on closest match............................................................ 16
  - Determining if two tables match............................................................................... 17

**Data summarizations and aggregations** ...................................................................... 19
  - Determining the most highly correlated pairs......................................................... 19
  - Parameterizing column names................................................................................... 21
  - Dividing data into buckets......................................................................................... 24
  - Calculating unique counts with different criteria.................................................... 25
  - Creating pairs without duplication.......................................................................... 25

**Integer and string manipulations** .................................................................................. 28
  - Converting names to initials....................................................................................... 28
  - Transforming dates from text to integers.................................................................. 29
  - Determining days between entries.......................................................................... 31

**Data cleansing** ............................................................................................................. 33
  - Removing duplicate rows from a table................................................................. 33
  - Removing values $n$ standard deviations from the mean....................................... 34

**QuickApps** .................................................................................................................... 36
  - Layout attributes........................................................................................................ 36
    - Creating layouts with multiple tab panels......................................................... 36
    - Creating layouts with nested splitters.............................................................. 38
    - Creating layouts with accordion or collapsible panels................................... 39
  - Configuring a QuickApp for render targets......................................................... 42
  - Creating a table with clickable columns and rows............................................ 48
  - Formatting data in a grid......................................................................................... 50

**Analytical Solutions** ..................................................................................................... 54
  - Performing a time comparison analysis................................................................. 54
  - Performing an affinity analysis............................................................................... 55
  - Extracting loadings/eigenvectors from your PCA................................................. 57
Introduction

The 1010data Macro Language is the most powerful and direct way to interact with 1010data's platform. The Macro Language consists of three different sets of operations. At the lowest level are the 1010data core operations. These operations are responsible for transforming data for analytical purposes. Another level higher, are the <block> operations, which provide programmatic functionality such as conditionals and loops. Finally, the <dynamic> and <loop> operations provide the user with the ability to build applications on top of data and give data consumers the ability to guide their analyses.

When developing queries, specific problems can arise that are not easily solved by consulting the 1010data Reference Manual or 1010data User's Guide. This Cookbook was created as a compilation of problems and solutions intended to assist 1010data users with issues that may arise when developing queries. Each recipe contains tips and best practices to solve a variety of problems including combining data sets, aggregating data, manipulating strings, cleansing data, creating QuickApps and executing analyses.

Although this book includes a variety of recipes to aid in the development of queries, it is not all inclusive. It will continue to grow and change to accommodate new solutions and new features of the system. If you have an objective that you would like to see included in this Cookbook or you have found a way to solve a specific problem, please email the code to support@1010data.com. In addition, contact support@1010data.com with any other questions or suggestions.

Who is this book for?

This Cookbook is made for users of varying experience levels. Whether you are an expert or you are just starting out, 1010data's Cookbook can assist you with problems you might come across while using our online software. Each problem should help you to better understand the functions and operations applied, as well give a discussion to how they operate. However keep in mind that some coding experience is needed in order to understand the concepts in this book.

What is this book not for?

This book is not a complete guide to all operations, functions and expressions used in 1010data's software and will not touch on everything needed to execute queries. For a more detailed guide please see our 1010data Reference Manual or 1010data User's Guide.

What is in a recipe?

Each recipe in this Cookbook, with the exception of the Debugging Guide, contains the same basic elements. This page describes the contents of each section contained in a recipe and the elements that make a Debugging Recipe different.

General recipes

The following sections may be present in each recipe:

Difficulty

This section contains a difficulty gauge ranging from 1 to 5. See Difficulty criteria on page 5 for the characteristics used to rate each recipe.

Objective
This section describes the task the user is trying to accomplish and any problems that arise while attempting to complete it. It might also describe relevant examples in order to help better understand the issue.

**Solution**

This section provides the Macro Language that accomplishes the task set forth in the objective.

**Alternate Solution**

Sometimes there are multiple ways to complete the same task. If there is an alternate way to solve the problem described in the objective, the code will be provided here. However, it should be noted that the preferred method is given in the main solution.

**Discussion**

This section describes the methodology behind the solution, and the alternate solution if there is one present. It also discuses the usefulness of the topic and possible areas where it might be applied.

**Common Errors**

This section includes errors that a user might receive when developing and executing the query given in the solution. However, it is not a complete list of all possible errors. For each error listed, you will see the direct message received from the system, why you are receiving this error and how you can correct it.

**Further reading**

Due to the fact that this Cookbook is not a complete guide to all functions and operation available for use in 1010data's platform, this section includes links to the Reference Manual for further discussion on the main components of the recipe.

**Debugging recipes**

Different from a general recipe, a recipe within the Debugging Guide describes ways to locate the origin of your error and ways to resolve it. Each of these recipes also contain an objective with a general description of the problem and a specific use case. The solution then contains multiple code blocks to show the process of resolving the error. Similarly to a general recipe, the discussion describes the steps taken in the solution and gives insight into the reasoning behind decisions.

**Difficulty criteria**

Each recipe is equipped with a difficulty measure based on the criteria discussed in this section. A recipe holding a specific difficulty rating can contain one more of the elements in the description of that level or a lower level.

**Level 1**

- Uses mainly basic core operations
- May use additional functions where the inputs are direct values and/or column names

**Level 2**

- Uses string manipulations
- Passes values from one function to another
Level 3

- Uses block code with programmatic operations, such as `<for>`, `<foreach>`, and `<if>`... etc.
- Uses basic QuickApp operations such as various layouts and basic widgets

Level 4

- Uses a `<loop>` operation
- Uses multiple nested operations
- QuickApp that:
  - Introduces basic widget interaction
  - Creates charts or tables

Level 5

- Uses modeling and analysis based `g_functions`
- Uses nested `<loop>` operations
- QuickApp that:
  - Passes variables between widgets
  - Uses nest widgets
  - Uses `<do>` statements that set variables for later use
Combination of data sets

Includes recipes on types of links, expansions, merges, creating additional data, and more.

Linking to find all possible values of a column

By duplicating each row in the base table, you can accommodate all matches in the foreign table when linking the two tables together on a common column.

Difficulty

Objective

You want to perform a tabulation to obtain a summary of results for different groups, but some of the groups may not have any data. However, you would like there still to be a result in the tabulation for every group. For example, you want to calculate the sum of sales per store for a specific date, but some stores might not have had sales on that particular day.

Solution

```xml
<base table="pub.doc.retail.altseg.stores"/>
<link table2="pub.doc.retail.altseg.sales_detail_transid" col="store" col2="store" expand="1">
  <sel value="trans_date=20151010"/>
</link>
<tabu label="Tabulation on Store Master" breaks="store,city">
  <tcol source="xsales" fun="sum" label="Sum of Sales" format="type:currency"/>
</tabu>
```

Discussion

Knowing that an item contains no values for a specific date can be useful. However, if you perform a tabulation and no values exist for a certain field, that field will simply not be present in the tabulation. If you instead link in a table that contains all possible values for that field and specify that expand="1", performing the same tabulation will create a row for that field.

Instead of tabulating on the Sales Detail table, this solution uses the Store Master table as a base and links in the Sales Detail table. Within the <link>, you can use a selection to only obtain transactions from October 10, 2010. Even though no transactions were made in store 186 on that date, a row is still created due to the expand attribute.

Expand="1" will duplicate every row in the Store Master table to accommodate every match in the Sales Detail table. Since there are no matches for store 186 in the Sales Detail table, it will not be expanded, but that row will still be kept in the final table and it will show that there was no sales. After the tabulation, the Sum of Sales tabulation shows that store 186 had a total of $0.00 sales.

Further reading

If you would like to learn more about the functions and operations discussed in this recipe, click on the links below:
Combining rows of multiple tables

You can efficiently combine the rows of multiple tables that are located in the same folder and have the same columns using `<loop>`. The result will be a single worksheet containing the rows from each table.

### Difficulty

![Difficulty Level](image)

### Objective

You have a folder containing multiple tables that have the same columns but which have data from different years. You would like to combine the rows of each table into a master table, and you would like each row to indicate which year it is from. While it is possible to simply perform a `<merge>` operation for each table you'd like to include, new tables added to the folder in the future will not be accounted for, and your code could become lengthy. Instead, utilizing the `<loop>` operation allows you to perform the same `<merge>` operation on every table in the folder. This method will also allow you to include any tables that might be added in the future without having to modify the code.

### Solution

```xml
<loop with="files">
  <outer>
    <directory folder="pub.demo.weather"/>
    <color cols="path"/>
    <col name="path" format="width:50"/>
    <sel value="contains(path;'pub.demo.weather.hourly')"/>
    <transpose/>
  </outer>
  <inner>
    <base table="{@files.m0}"/>
    <letseq tables="{@files}" keys="{pkg_names(@tables)}">
      <tabu label="Tabulation on Hourly U.S. Weather" breaks="id,date">
        <tcol source="prec" fun="avg" label="Average`Hourly`Precip`(.01 inch)"/>
        <tcol source="snow" fun="avg" label="Average`Snow`Depth`(inches)"/>
        <tcol source="temp" fun="avg" label="Average`Dry`Bulb Temp`(Celsius)"/>
      </tabu>
      <foreach table_year="{@keys}" tally_="@i">
        <if test="{@i>1}"
          <then>
            <merge table2="{@files.{@table_year}}" match="pad">
              <tabu label="Tabulation on Hourly U.S. Weather" breaks="id,date">
                <tcol source="prec" fun="avg" label="Average`Hourly`Precip`(.01 inch)"/>
                <tcol source="snow" fun="avg" label="Average`Snow`Depth`(inches)"/>
                <tcol source="temp" fun="avg" label="Average`Dry`Bulb Temp`(Celsius)"/>
              </tabu>
            </merge>
          </then>
        </if>
      </foreach>
    </letseq>
  </inner>
</loop>
```
Alternate solution

```xml
<loop with_="files">
  <outer>
    <directory folder="pub.demo.weather"/>
    <colord cols="path"/>
    <col name="path" format="width:50"/>
    <sel value="contains(path;'pub.demo.weather.hourly')"/>
    <transpose/>
  </outer>
  <inner>
    <base table="default.lonely"/>
    <letseq tables="{@files}" keys="{pkg_names(@tables)}">
      <foreach table_year="{@keys}" tally_="@i">
        <merge table2="{@files.{@table_year}}" match="pad">
          <tabu label="Tabulation on Hourly U.S. Weather" breaks="id,date">
            <tcol source="prec" fun="avg" label="Average Hourly Precip \(.01\) inch"/>
            <tcol source="snow" fun="avg" label="Average Snow Depth \(\) inches"/>
            <tcol source="temp" fun="avg" label="Average Dry Bulb Temp \(\) (Celsius)"/>
          </tabu>
        </merge>
      </foreach>
    </letseq>
    <sel value="date<>NA"/>
    <colord hide="c1"/>
  </inner>
</loop>
```

Discussion

The `<loop>` operation is a useful way to perform the same actions repeatedly without the need to duplicate code. This recipe uses `<loop>` in conjunction with the `<merge>` operation to merge several tables in the same folder. In this way, you can combine numerous tables without needing to use multiple `<merge>` operations.

Both of the above solutions use the `<directory>` operation to first create a table containing the information for all of the tables within a specific folder. Once that information is obtained, the `<colord>` operation is used to reduce the worksheet’s contents to a single column of table names. Since the folder contains additional tables that should not be included in the merged table, the `<sel>` operation is used to select only the rows that contain the names of the desired tables. Then, the `<transpose>` operation converts the worksheet from a single column to a single row, and the values are placed in a package by the `<outer>` part of the `<loop>` in the variable specified by the `with_` attribute.

In the main solution, the `<inner>` loop then specifies a base table using the first item in the package `({@files.m0})`, and the `<foreach>` operation merges each additional item. In order to avoid merging the base table to itself, you can use an `<if>` statement to only merge after the first iteration. An initial
Tabulation is performed to aggregate the data into the desired format, and the same tabulation is then performed on each subsequent table before the final results are merged.

Alternatively, you can use an empty table, default.lonely, as the base table to which all other tables are merged. This will create an empty row at the top of the data table and an empty column labeled c1. However, these can be easily removed by using the <sel> and <colord> operations after the merge.

**Further reading**

If you would like to learn more about the functions and operations discussed in this recipe, click on the links below:

<loop>
<merge>
<foreach>

**Finding missing values by linking tables**

When merging tables, sometimes they do not have equal amounts of information. When using match="pad", the columns that do not have matches are populated with NA values for the foreign table. To find the missing values that are added when tables that have unmatched columns, you can create a reference or lookup table.

**Difficulty**

![Difficulty scale](image)

**Objective**

You wanted to merge several tables together, but some tables contained more columns than others, so you used the <merge> operation with match="pad". The merged table now contains N/A values in its columns for those rows from the foreign tables that did not have a matching column. Based on common information between the tables that were merged together, you can determine the missing values by creating a reference table. This reference table can then be linked to your merged table to create a lookup column for the missing values. For example, if the foreign tables you are merging contained columns for the city and zip code, but your base table only contained a column for the city, you could fill in the missing zip code information based on the data in the foreign tables.

**Solution**

```xml
<base table="pub.doc.gov.area.x2003"/>
<willbe name="year" label="Year" value="2003" format="type:nocommas"/>
<merge table2="pub.doc.gov.area.x2004" match="pad">
    <willbe name="year" label="Year" value="2004" format="type:nocommas"/>
</merge>
<link table2="pub.doc.gov.area.x2003" col="state,county"
    col2="state,county" suffix="_lookup">
    <colord cols="state,county,countyname"/>
</link>
<willbe name="county_name"
    value="if(countyname=NA;countyname_lookup;countyname)"/>
<colord hide="countyname,countyname_lookup"/>
```
Alternate solution

```xml
<base table="pub.doc.gov.county.x2002"/>
<merge table2="pub.doc.gov.county.x2003" match="pad"/>
<link table2="pub.doc.gov.county" col="state" col2="state" suffix="_lookup">
  <loop with_="alphnum">
    <outer>
      <directory folder="pub.doc.gov.county"/>
      <colord cols="path"/>
      <col name="path" format="width:50"/>
      <transpose/>
    </outer>
    <inner>
      <base table="{@alphnum.m2}"/>
      <letseq tables="{@alphnum}" keys="{pkg_names(@tables)}">
        <foreach table_year="{@keys}"
          tally_="@i">
          <if test="{@i>3}">
            <then>
              <merge table2="{@alphnum.{@table_year}}"/>
            </then>
            <else/>
            </if>
        </foreach>
      </letseq>
    </inner>
  </loop>
  <colord cols="state,state_alpha"/>
  <sel value="g_first1(state;;)"/>
</link>
<willbe name="statealpha" value="if(state_alpha=NA;state_alpha_lookup;state_alpha)"/>
<colord hide="state_alpha,state_alpha_lookup"/>
```

Discussion

Often when merging tables, one table might have more columns, and therefore, more information, than another table. Thus, when you merge the tables, there will be rows that have missing values. This information can be found if the tables share other information that correlates to the missing values.

For example, in the first solution, tables containing two different years of area rent data are merged together. One table contains a column for the county number as well as a column for the county name, but the other table only contains a column for the county number. In the merged table, the column for the county name will contain N/A values for those rows from the table that did not have any county name information. By creating a reference column that shows how each number correlates with each name, the missing county names can be determined.

The first solution merges the tables using the `<merge>` operation with `match="pad"` to make sure that if the table columns don't match, all are still included in the merge. The `<colord>` operation is used to display only those columns that uniquely identify each county name in a new worksheet, which is then linked into the merged worksheet. By setting `col` and `col2` equal to these identifying columns, a `countynamlookup` column is created in the merged worksheet. A new column is then created that contains a full list of county names, by using the original name if it existed, or the lookup value if it was N/A.

If you are sure that the table you are using to create a reference column includes all possible values, then the first solution in this recipe works and uses a minimal amount of code. However, if the values are not all inclusive, it is a better idea to create a reference table using the information from multiple tables. In the recipe `Combining rows of multiple tables` on page 8, the `<loop>` operation was used to merge multiple tables together. Using this same idea, a reference table can be created by looping through the information of multiple tables within the `<link>` operation.
The second solution merges rows from two tables of county rent data. One table contains a column for the state number and the state abbreviation, while the other table only contains the state number. To make sure there is a reference for every possible state, this solution loops through all of the years of county rent data to create an all-inclusive reference table. As in the first solution, the reference table is then linked into the merged table to find the missing values.

**Further reading**

If you would like to learn more about the functions and operations discussed in this recipe, click on the links below:

<merge>
<link>
if(C1;R1;C2;R2;...;D)

---

**Adding source information to the description of columns**

By using `<loop>`, you can add the path of the originating table to the description of each linked column, when combining the columns of multiple tables.

**Difficulty**

![Difficulty scale](image)

**Objective**

You want to combine the columns of multiple tables, but you want to be able to distinguish which table each linked column originated from. You can do this by adding the path of the originating table to the description of each linked column.

**Solution**

```xml
<block ltable="pub.doc.retail.altseg.products" lcol="sku,dept,group"
    lcol2="sku,dept,group">
  <base table="pub.doc.retail.altseg.sales_detail_transid"/>
  <set linklist="(str_to_lst(@lcol2;','))"/>
  <link table2="{@ltable}" col="{@lcol}" col2="{@lcol2}"/>
  <loop with_="columns">
    <outer>
      <base table="{@ltable}"/>
      <columns/>
      <colord cols="name"/>
      <transpose/>
    </outer>
    <inner>
      <loop with_="description">
        <outer>
          <base table="{@ltable}"/>
          <columns/>
          <colord cols="desc"/>
          <transpose/>
        </outer>
        <inner>
          <letseq desc="{@description}" keys="{pkg_names(@desc)}">
```
<letseq names="{@columns}" keys="{pkg_names(@names)}">
  <foreach text="{@keys}" colname="{@keys}" tally="@i">
    <if test="{@columns.{@colname}=@linklist.1|
                @columns.{@colname}=@linklist.2|
                @columns.{@colname}=@linklist.3}">
      <col name="{@columns.{@colname}}"
           desc="{@description.{@text}} (column linked on)"/>
    </then>
    <else>
      <col name="{@columns.{@colname}}"
           desc="{@description.{@text}} (originating table: {@ltable})"/>
    </else>
  </foreach>
</letseq>
</letseq>
</inner>
</loop>

Alternate solution

<dynamic base="">
  <do onchange="@base" value1="@table_names" row1="1" col1="1" value2="@column_names" row2="1" col2="2" value3="@desc_text" row3="1" col3="3">
    <loop with="tables">
      <outer>
        <directory folder="{@base}"/>
        <colord cols="path"/>
        <transpose/>
      </outer>
      <inner>
        <base table="{@tables.m0}"/>
        <columns/>
        <willbe name="colnames" value="g_splice(;;;name;',';')"/>
        <willbe name="descriptions" value="g_splice(;;;desc;',';')"/>
        <willbe name="tablenames" value="'{@tables.m0}''"/>
        <colord cols="tablenames,colnames,descriptions"/>
        <sel value="i =1"/>
        <letseq names="{@tables}" keys="{pkg_names(@names)}">
          <foreach tablename="{@keys}" tally="@i">
            <if test="[@i]>1">
              <merge table2="{@tables.{@tablename}}" match="pad">
                <columns/>
                <willbe name="colnames" value="g_splice(;;;name;',';')"/>
                <willbe name="descriptions" value="g_splice(;;;desc;',';')"/>
                <willbe name="tablenames" value="'{@tables.{@tablename}}''"/>
                <colord cols="tablenames,colnames,descriptions"/>
                <sel value="i =1"/>
              </merge>
            </if>
          </foreach>
        </letseq>
      </inner>
    </loop>
  </do>
</dynamic>
<willbe name="tables" value="g_splice(;;;tablenames;'|';")"/>
<willbe name="columns" value="g_splice(;;;colnames;'|';")"/>
<willbe name="descs" value="g_splice(;;;descriptions;'|';")"/>
<colord cols="tables,columns,descs"/>
</do>
<do onchange_="@base">
  <set tablebase="{str_to_lst(@table_names;'|')}"/>
  <set columnlists="{str_to_lst(@column_names;'|')}"/>
  <set desclists="{str_to_lst(@desc_text;'|')}"/>
</do>
<widget class_="browser" value_="@base" filter_="folder" accept_="folder" typevalue_="@base_type" label_="Base Path" dropdownwidth_="550"/>
<widget class_="text" text_="{@tablebase}"/>
<widget class_="text" text_="{@columnlists}"/>
<widget class_="text" text_="{@desclists}"/>
</dynamic>

**Discussion**

When combining columns of multiple tables, one way to be able to identify the originating table of each column is by adding a suffix to the column name. However, you can also change column metadata, such as the column description or label, using the `<col>` operation. This first solution in this recipe adds the path of the originating table to the description of each linked column using `<col>`. In the first solution, two nested loops are used to create two packages, one for column names and one for the description of each column. Using `<letseq>` and `<foreach>` in the innermost loop, the path of the originating table is added to the description of each column. However, in the `<if>` statement, if the current column name in the package matches one of the linking columns (specified by `col2`), the phrase "(column linked on)" is amended to the description instead.

The second solution uses a browser widget in a QuickApp (within a `<dynamic>`) to allow the end user to select a directory folder and then generates lists of column names and descriptions for each table in the selected folder. Again, a loop is used to create a package, which contains a path for each table in the folder. In order to generate the strings of column names and descriptions, `g_splice(G;S;O;X;D;N)` is used on the `name` and `desc` columns in each table. The resulting columns are joined together using `g_splice(G;S;O;X;D;N)` to create a master string.

The first `<do onchange_="@base">` construct allows you to assign the value of a particular cell (denoted by a given row and column) to a dynamic variable. Finally the strings are turned into lists using `str_to_lst(X;Y)`. Text widgets display the lists to ensure everything was properly executed. These lists can then be used to change the column metadata in any of the tables you decide to link from the selected folder.

**Common errors**

Non-list component in compound @variable name "lcol2.1"

In the first solution, if you simply try to use `@lcol2.1` in your `<if>` statement to denote the first column name designated to link on, you will receive this error.

Even though the column names you entered in the variable `lcol2` are separated by commas, the variable as a whole is considered as a string by the system and not a list. In order to access each item in the list using an index, you must first use the `str_to_lst(X;Y)` function to transform the string into a list-value. Setting this to a new variable, you can now use an index to access each item (i.e., `@linklist.1`).

**Further reading**

If you would like to learn more about the functions and operations discussed in this recipe, click on the links below:
Checking for common column names among tables

Using `<loop>` makes it easy to perform multiple identical operations. Instead of previewing multiple tables to decide which column to link on, you can determine which tables contain columns with matching names by looping through a directory folder.

**Difficulty**

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Objective**

You want to determine all the tables in one or more folders that contain the same column. For instance, you may want to combine all of the tables within a folder, but you do not want to open every table to check if it contains the specific column you want to link on.

**Solution**

```xml
<loop with_="all_tabs" mode_="merge">
  <outer>
    <directory folder="pub.demo.baseball" depth="0"/>
    <merge table2="default.lonely">
      <directory folder="pub.doc.retail.altseg" depth="0"/>
      <sel value="ttype='table'"/>
    </merge>
    <sel value="(type='tab')"/>
  </outer>
  <inner>
    <base table="{@all_tabs.path}"/>
    <columns/>
    <willbe name="path" value="'{@all_tabs.path}'"/>
    <sel value="contains(name;'div' 'Id')"/>
    <colord cols="name,path"/>
    <col name="name" format="width:20"/>
    <col name="path" format="width:45"/>
  </inner>
</loop>
```

**Discussion**

Creating a block that can check for common column names within tables can be particularly useful if you have folders that contain a large number of tables. This recipe determines if there are any tables in the folders `pub.demo.baseball` and `pub.demo.retail` that have column names containing the substrings "div" or "id".

The solution uses a `<loop>` where the `<outer>` creates a package of table paths and the `<inner>` merges the column names of each table. A column is created to hold the table path for each column, and a
selection statement using the `contains(X;L)` function selects only the columns that contain the specified substrings in their names.

A `<sort>` operation is performed at the end so that the common columns are grouped together and the table paths in each group are sorted in ascending order.

Further reading

If you would like to learn more about the functions and operations discussed in this recipe, click on the links below:

`<loop>`
`<merge>`
`contains(X;L)`

Linking columns of tables on closest match

Sometimes it is necessary to combine tables on a common column that do not contain exact matching information. You can combine tables using an `asof` link that matches rows with their nearest value.

Difficulty

Objective

You want to combine the columns of two tables, linking based on a common column. However, the values in this common column are not an exact match from table to table. For example, you have a table containing sales information and another table containing inventory data, and you would like to combine the columns of the tables based on date. While your sales table contains daily entries, your inventory table contains weekly entries, therefore you want to match the nearest week of inventory to each entry in your sales table. This can be done in 1010data by performing an as-of link.

Solution

```xml
<base table="pub.doc.retail.altseg.sales_detail_transid"/>
<sel value="sku=211611 & store=85"/>
<tabu label="Tabulation on Sales Detail" breaks="trans_date">
    <tcol source="qty" fun="sum" name="sum_sold" label="Sum of Qty Sold"/>
</tabu>
<link table2="pub.doc.retail.altseg.inventory_store" col="trans_date"
    col2="date" suffix="_inv" type="asof">
    <sel value="sku=211611 & store=85"/>
    <colord cols="date,quantityOnHand"/>
    <sort col="date" dir="up"/>
</link>
```

Alternate solution

```xml
<base table="pub.doc.retail.altseg.sales_detail_transid"/>
<sel value="sku=211611 & store=85"/>
<tabu label="Tabulation on Sales Detail" breaks="sku,trans_date">
    <tcol source="qty" fun="sum" name="sum_sold" label="Sum of Qty Sold"/>
</tabu>
```
Discussion

Using Sales Detail for your base table, the desired item SKU and store numbers are selected, and a tabulation is performed to obtain the number of units sold for each day. Within the <link> operation, the same selections must be done on the Inventory table. To effectively preform the asof link, you must then sort the foreign table by date. Your base table must also be sorted by date in the same direction. The base table used in this recipe is already sorted by date in the ascending direction, so the foreign table is sorted to match. With the tables sorted correctly, performing an asof link will bring in the inventory data for the week closest to each date in the Sales Detail table.

Further reading

If you would like to learn more about the functions and operations discussed in this recipe, click on the links below:

<link>

As-of Links

Determining if two tables match

You can determine if two tables contain the same information by comparing the columns of one table to the columns of a second table.

Difficulty

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
</table>

Objective

You have two tables that appear to be the same, but you want to determine if they are indeed exact matches. You can do this by creating an md5 hash of the information across all columns in both tables and then compare the two hashes.

Solution

<meta>empty</meta>
<base table="pub.doc.retail.altseg.stores"/>
<willbe name="md5_hash" value="r_md5sum(store division subdivision zip divisiondesc subdivisiondesc manager sqft format city state dc compDate;)"/>
<colord cols="md5_hash"/>
<link table2="pub.doc.retail.altseg.stores" col="md5_hash" col2="md5_hash" type="exclude">
  <willbe name="md5_hash" value="r_md5sum(store division subdivision zip divisiondesc subdivisiondesc manager sqft format city state dc compDate;)"/>
</link>
Alternate solution

```xml
<base table="pub.doc.retail.altseg.stores"/>
<wilname="md5_hash" value="r_md5sum(store division subdivision zip divisiondesc subdivisiondesc manager sqft format city state dc compeffdate pycompeffdate compDate;)"/>
<colord cols="md5_hash"/>
<link table2="pub.doc.retail.altseg.stores" col="md5_hash" col2="md5_hash" type="exclude">
  <wilname="md5_hash" value="r_md5sum(store division subdivision zip divisiondesc subdivisiondesc manager sqft format city state dc compeffdate pycompeffdate compDate;)"/>
  <wilname="match" value="1"/>
  <colord cols="md5_hash,match"/>
</link>
```

Discussion

Sometimes two tables can look similar but it's difficult to determine if they are an exact match just by previewing both tables. Instead, you can create an md5 hash of the information contained in both tables and use this hash to compare the tables.

In both solutions shown, a column is created that contains the md5 hash in both the base table and the foreign table. This is created by using the function, `r_md5sum(C;S)`, which hashes the data across a given set of columns. For this recipe, the hash includes data from all columns, but if you want to compare only a subset of the columns from each table, you can adjust accordingly.

The first solution performs a link and exclude with the foreign table, which will result in the selection of rows from the base table that do not exist in the foreign table. In this case, no rows are selected because all rows from the base table have a match in the foreign table.

   **Note:** `<meta> empty </meta>` is present at the beginning of the query. This allows no rows to be selected as the result of your query. If this was absent from your query, you would receive the error message **No rows selected**.

The second solution creates an additional column, `match`, that contains the value 1. When a regular link is performed between the base table and the foreign table on the `md5_hash` column, `match` will display a 1 if the row matches the foreign table and a 0 otherwise.

Further reading

If you would like to learn more about the functions and operations discussed in this recipe, click on the links below:

- `r_md5sum(C;S)`
- `<meta>`
- `<link>`
Data summarizations and aggregations

Includes recipes utilizing tabulations, g functions, selections, and performing sorts and various mathematical operations.

Determining the most highly correlated pairs

You can calculate the correlation coefficient of two items using $g_{\text{cor}}(G;S;X;Y)$, but sometimes it is necessary to compute the correlation coefficient for multiple pairs and determine which items are the most highly correlated.

Difficulty

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Objective

You want to identify the relationship between multiple pairs of items determine which items are most closely related. For example, you want to know how a day’s weather condition correlates to the products sold that day. Then you want to select the most correlated product for each weather condition.

Solution

```xml
<library>
  <block name="correlate_weather_condition" condition="">
    <base table="pub.demo.weather.wunderground.forecast"/>
    <.sel value="year(datefor)=2016"/>
    <sel value="period<>1" />
    <willbe name="conditions2" value="if((contains(conditions;'Snow'));'Snow';(contains(conditions;'Rain'));'Rain';(contains(conditions;'Cloudy'));'Overcast';(contains(conditions;'Thunderstorm'));'Thunderstorm';conditions)="/">
      <link table2="pub.doc.retail.altseg.sales_detail_transid" col="zipcode, dateof" col2="zip_txt,trans_date" type="select" expand="1">
        <sel value="year(trans_date)=2016"/>
        <link table2="pub.doc.retail.altseg.stores" col="store" col2="store" type="select">
          <sel value="state='CA'"/>
        </link>
        <willbe name="zip_txt" value="strsubst(padleft(string(zip);5);' ';0;'0')"/>
        <link table2="pub.demo.weather.wunderground.forecast" col="trans_date,zip_txt" col2="dateof,zipcode" type="include"/>
        <tabu label="Tabulation on Sales Detail" breaks="zip_txt, sku,trans_date">
          <tcol fun="sum" name="vol" source="qty" label="Volume"/>
        </tabu>
        <willbe name="relative_vol" value="vol/g_avg(zip_txt;;vol)" label="Volume Relative to Avg for Zip"/>
        <color cols="zip_txt,trans_date,sku,relative_vol"/>
      </link>
    </willbe>
  </sel>
</block>
```
Discussion

The use of blocks is helpful when you want to perform the same set of operations multiple times. There are two ways to reuse a query contained in a <block>. One way is to put the block inside a <library>. The other way is to use a <defblock>. With both methods, when you want to use the block, you need to use either the <insert> or <call> operation. Additionally, when using <library>, your library either needs to be present in your current query, or if it is saved elsewhere, it needs to be imported using the <import> operation.

For this recipe, the <block> is contained in a <library>, which is present at the beginning of the query. The block is structured so that you can correlate to any condition. This is done by substituting a variable, condition, which is initialized in the block, for every instance in the query that requires a named condition. Therefore, when you insert the <block>, you also state which condition you want to correlate to.

Before starting the analysis, the Forecast table is reduced by selecting only future weather forecasts, i.e. where the date of is not equal to the date for, or the period is not 1, and also by selecting forecasts for the year 2016. Then, you want to compare every forecast to every item bought on that day. A <link> is performed to combine the current Forecast worksheet to the Sales Detail table. However, in order to compare all combinations, expand="1" needs to be specified. This attribute will duplicate each entry in the Forecast worksheet to allow for multiple matches in the foreign table.

Within the link to Sales Detail, multiple operations are used to reduce the table. Similar to the reductions on the Forecast table, only transactions occurring in 2016 are selected. A link and select to the Store Master table, causes only the transactions from stores in California to be retained. In order to make a
further reduction, and strictly include transactions from stores in locations that there is forecast data for, a link and include is performed with the Forecast table. Now that the table is reduced, a tabulation is performed. This tabulation finds the total amount of each SKU sold on each date for each store location.

After the worksheets are linked, you are left with a record of the relative volume sold for each product, location, date combination. To start calculating the correlation coefficients, you need to first flag the weather condition you want to correlate to. Then you count how many times that condition will appear within the forecast horizon and determine the percentage.

The correlation between the relative volume sold and the percentage of days the specified weather condition appears in the forecast horizon is determined with $g_{cor}(G;S;X;Y)$. However, this function requires the $G$ argument to contain the column(s) on which the table is segmented, and currently the worksheet is segmented by zipcode. Therefore, an empty merge, `<merge/>`, is performed to condense the table to a single segment.

**Note:** You should only perform an empty merge on tables smaller than ten million rows, or the system will have difficulty completing the query.

After the table is condensed into a single segment, you can either leave the $G$ argument blank, or use your desired grouping columns.

To make sure you are including results calculated from a suitable amount of data points, a selection is done to retain SKU/period combinations that had greater than 100 observations. Additional selections are done to select rows where the correlation coefficient isn't NA, where the SKU/period combination is the first occurrence, and where the correlation coefficient is the highest for that SKU. After these selections, if there is still more than one occurrence of each SKU, then the occurrence with the longest horizon, or highest period is selected.

These correlations are ranked using $g_{rank}(G;S;O;X)$, and the top ten correlated products are selected. To obtain more information about the highest correlated products, the Product Master table is linked in and the description column is added to the worksheet.

Within a second block, an `<insert>` statement with the condition variable initialized to rain will give you the top ten correlated products for rainy weather. Then, with the addition of the `<foreach>` loop, you can obtain the top ten correlated products for every weather condition by inserting the block for each condition and merging the results together. Lastly, a `<sort>` combined with a `<sel>` statement will create a table that includes the most highly correlated product for each weather condition.

**Further reading**

If you would like to learn more about the functions and operations discussed in this recipe, click on the links below:

```block
<insert>
g_cor(G;S;X;Y)
g_rank(G;S;O;X)
```

**Parameterizing column names**

Sometimes it is necessary to dynamically determine column names for computed columns based on the information contained in a variable.
Objective
You want to create several computed columns but know that later you will want the same computations
done on different columns. One solution is to change all of the information in your code dealing with the old
columns to information about the new columns. However, this can be tedious and time consuming, not to
mention that you could miss instances of the old column and not replace the outdated information. Instead
you can wrap your code in a block where a variable containing the desired column names is defined. The
rest of your code then extracts necessary information based on the information contained in the variable
instead of a specific column name.

Solution
<block name="computation" colname="xsales,qty,cost" seg="transid customer">
    <base table="pub.doc.retail.altseg.sales_detail_transid"/>
    <loop with="columns">
        <outer>
            <base table="pub.doc.retail.altseg.sales_detail_transid"/>
            <columns/>
            <color col="label"/>
            <set collist="{str_to_lst(@colname;',')}"/>
            <set collist="'{lst_to_str(@collist;'\'\'')}'"/>
            <sel value="name=@collist"/>
            <transpose/>
        </outer>
        <inner>
            <letseq names="{@columns}" keys="{pkg_names(@names)}">
                <foreach var="{@colname}" col_label="{@keys}" tally_="@i">
                    <willbe name="sum_of_{@var}" label="Sum of
                    {columns.{@col_label}}" value="g_sum({@seg};;{@var})"/>
                </foreach>
            </letseq>
        </inner>
    </loop>
    <sel value="g_first1({@seg};;)"/>
</block>

Alternate solution
<block name="computation" colname="xsales,qty,cost" seg="transid"
    function="sum">
    <base table="pub.doc.retail.altseg.sales_detail_transid"/>
    <loop with="columns">
        <outer>
            <base table="pub.doc.retail.altseg.sales_detail_transid"/>
            <columns/>
            <color col="label"/>
            <set collist="{str_to_lst(@colname;',')}"/>
            <set collist="'{lst_to_str(@collist;'\'\'')}'"/>
            <sel value="name=@collist"/>
            <transpose/>
        </outer>
        <inner>
            <letseq names="{@columns}" keys="{pkg_names(@names)}">
                <func function="sum">
                    <base table="pub.doc.retail.altseg.sales_detail_transid"/>
                    <loop with="columns">
                        <outer>
                            <base table="pub.doc.retail.altseg.sales_detail_transid"/>
                            <columns/>
                            <color col="label"/>
                            <set collist="{str_to_lst(@colname;',')}"/>
                            <set collist="'{lst_to_str(@collist;'\'\'')}'"/>
                            <sel value="name=@collist"/>
                            <transpose/>
                        </outer>
                        <inner>
                            <letseq names="{@columns}" keys="{pkg_names(@names)}">
Discussion

Changing your code to perform the same operations with different values can be problematic. Especially if the code is lengthy, making a mistake in your editing can cause an error that might be difficult to locate. Additionally, it can be a challenge to make sure each instance of the old information is replaced with the new. Running the same query with different data can be simplified by creating one or more variables in a <block> that hold the values of the information you would like to change in the future.

In this solution two variables are defined, one to store column names, colname, and one to store the segmentation of the data table, seg. The column names contained in the colname variable will be used to create tabulated columns containing the sum of the values of each column. These values are calculated using the g_sum(G;S;O;X) function. The seg variable is passed to the g_function to clarify how the table is segmented. To properly label the computed columns, the labels from the source columns are packaged with each column name in the outer loop. Because the column names entered in colname are not stored in a selection friendly format, you must use str_to_lst(X;D) and lst_to_str(X;D) to change the delimiter to a space and add single quotes around each column name.

Alternatively, you can perform a tabulation instead of creating a computed column. This will give you the added option of changing the function by creating a third variable, function. However, using a g_function is preferred due to the table's segmentation. Performing a tabulation will not work on tables with over 3 billion rows.

Common errors

Unequal length iterators in <foreach>

If you do not transpose your table in the outer loop, you will receive this error.

This is because the package created from the outer loop connects each column name to the last value in the column. Without transposing the table, you only have one column. This then tells the <foreach> operation that there is one value in the keys variable and three values in the colname variable.

Further reading

If you would like to learn more about the functions and operations discussed in this recipe, click on the links below:

str_to_lst(X;D)
lst_to_str(X;D)
Dividing data into buckets

Dividing data into buckets can make it easier to perform analyses and gather information about the data, especially if the data set is large. If the location of the items are not important, you can randomly divide the data into a predefined number of equal groups.

Difficulty

![Difficulty Scale]

Objective

You want to divide your data set evenly into a chosen number of buckets (or groups), where each bucket contains roughly the same number of values. Using characteristics of the data to create the buckets could lead to the groups being uneven. For example, if you're dividing records on people, using age or gender as a selection basis may lead to unevenly sized groups. In order to ensure that the groups are even, you can randomly distribute the data by using a random number generator.

Solution

```xml
<base table="pub.doc.retail.altseg.customers"/>
<willbe name="picker" value="draw(12345;10)"/>
<tabu label="Tabulation on Customer Master" breaks="picker">
    <tcol source="cust_age" fun="avg" label="Average age"/>
    <tcol source="cust_age" fun="cnt" label="# Records"/>
</tabu>
```

Discussion

To divide your data into an equal number of buckets, you need to create a new column that assigns each record a number to determine their bucket. Say you wanted 10 buckets, then each value in your data set would need to be assigned a number 1 through 10. In order to make these assignments random, you can use the `draw(X;Y)` function, where the value given to `Y` is the number of buckets desired.

Additionally, this solution uses tabulations to make sure each bucket created contains roughly the same number of values and that the average of each bucket is comparable.

Common errors

The query failed after 0.0 seconds with the message: Computed column picker cannot be parsed: Inappropriate argument for "draw"

With the `draw(X;Y)` function, `X` is used as a seed, which is used to generate the random numbers before each is assigned a value from `Y`. This number can be any integer, however if it exceeds 2,147,483,646 you will receive this error.

Further reading

If you would like to learn more about the functions and operations discussed in this recipe, click on the links below:

`draw(X;Y)`
Calculating unique counts with different criteria

Calculating the unique count of a specific column can be accomplished with a tabulation. However, calculating that unique count for two different situations requires more manipulation.

**Difficulty**

**Objective**

You want to calculate the number of stores each customer visited during a specified week and during a specified month. Performing a unique count tabulation, using Store as the source column and grouping on Customer, will result in the number of stores each customer visited during the given time period. However, you need this number broken up into the number of stores per week and per month.

**Solution**

```xml
<base table="pub.doc.retail.altseg.sales_detail_transid"/>
<willbe name="store_week" value="if(between(trans_date;20160403;20160409); store;NA)"/>
<willbe name="store_month" value="if(between(trans_date;20160403;20160430); store;NA)"/>
<sel value="store_month<>NA"/>
<tabu label="Tabulation" breaks="customer">
  <break col="customer" sort="up"/>
  <tcol fun="ucnt" name="ucnt_store_week" source="store_week"
       label="Unique Count Store for Week"/>
  <tcol fun="ucnt" name="ucnt_store_month" source="store_month"
       label="Unique Count Store for Month"/>
</tabu>
```

**Discussion**

Before performing the tabulation, two new source columns are created, one for the specified week, and one for the specified month. If the store visit took place during the specified time period, the store number is recorded in the column, if not, an NA value is given. Then a selection is performed to only include the rows that occur in the given time period.

You can then perform a tabulation using `store_week` and `store_month` as your source columns and `ucnt` as your function.

**Further reading**

If you would like to learn more about the functions and operations discussed in this recipe, click on the links below:

```xml
<tabu>
  if(C1;R1;C2;R2;...;D)
</tabu>
```

**Creating pairs without duplication**

You can create a list of every possible combination of two lists of items by performing a link and expand. However, with this method, you will get duplicate pairs, where the items are simply listed in a different
order (i.e., reciprocals). Using computed columns and g_functions, you can create a list of pairs without any reciprocals.

**Difficulty**

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Objective**

You want to compute the affinity of every pair of items bought together. To complete this analysis, you need to create the pairs within each basket from a transaction table. However, this table is already large, so performing a link and expand to create the pairs will create an even larger worksheet, as well as duplicate pairs. To avoid these inefficiencies, you should only expand the table as much as necessary to allow for the creation of all pairs of items without creating duplicates.

**Solution**

```xml
<block name="expand_prep">
  <sel value="year(trans_date)=2016 & month(trans_date)=03"/>
  <sel value="g_first1(transid sku;;)"/>
  <willbe name="order" value="g_cumcnt(transid;;)"/>
  <willbe name="num" value="g_ucnt(transid;;sku)"/>
  <willbe name="expand" value="num-order"/>
</block>
<base table="pub.doc.retail.altseg.sales_detail_transid"/>
<insert block="expand_prep"/>
<sel value="expand" expand="1"/>
<willbe name="linker" value="order+ii_1()"/>
<link table2="pub.doc.retail.altseg.sales_detail_transid" col="transid,linker" col2="transid,order">
  <insert block="expand_prep"/>
  <colord cols="transid,sku,order"/>
</link>
<colord cols="transid,sku,c1"/>
```

**Discussion**

Before beginning any expansions, a selection is done on the table in order to limit the results to a specific time period. This will help prevent the table from becoming unnecessarily large. Additionally, only the first instance of each transaction/SKU is selected because if a specific item is bought twice in one transaction, you don’t need to compare it to itself.

Now that the table is reduced to only the necessary data points, you can start creating the preparatory columns for the expansion. First, you need a column that specifies the order of products within each transaction. This can be done by performing a cumulative count for each transaction. The order of the products is important because each product is expanded in order to accommodate the products that appear after it, therefore a definitive order is needed. Then, you need a column that indicates how many products are in each transaction, which is calculated with \( g_{ucnt}(G;S;X) \). By subtracting the order of each product from the number of products contained in the transaction, you can determine how many times to expand each row.

Each item in the transaction will be expanded to accommodate the number of unique products that appear thereafter in the same transaction. Using this method, the first item will be expanded once for every item in the transaction excluding itself. The next item will be expanded one time less than the first and so on, until the last item is actually not expanded at all, but rather removed from the table, as it will have appeared in every prior pair.
Once the table is expanded, it is linked to itself using the `linker` column and the `order` column. The column, `linker`, is created by adding the enumeration of each SKU in each transaction to that SKU's order. For example, SKU 112741 in transaction -2081370256 is expanded three times, and it's the first SKU listed in the transaction. Therefore, in the `linker` column, the first instance of the SKU will have a value of 2, \((1+1)\), the second instance will have a value of 3, \((1+2)\), and the third instance will have a value of 4, \((1+3)\). Linking this column to the `order` column from the same table, will create the unique pairs of items.

**Note:** This method only works if the order of the items in each pair does not matter. For example, if you need to compare apples and oranges but it doesn't matter if you're comparing apples to oranges or oranges to apples, this methodology will work. However, if you need to know how apples compare to oranges and how oranges compare to apples, you should perform a link and expand which will give you these reciprocal pairs.

### Common errors

*There was a problem getting a worksheet for a link (this can be caused by the worksheet being too large). The trapped error was: Too much data to get from table at subprocesses; hide some columns or select fewer rows*

If the table that you are working with is larger than 10,000,000 rows after the expansion, it will be too large to link to itself due to worksheet size limitations. In order to avoid this error, you can make further selections on your original table in order to decrease the worksheet size.

### Further reading

If you would like to learn more about the functions and operations discussed in this recipe, click on the links below:

- `<sel>`
- `<link>`
- `g_cumcnt(G;S;O)`
- `g_ucnt(G;S;X)`
### Integer and string manipulations

Includes recipes that manipulate strings and integers by extracting, separating, ordering, and converting characters, among other operations including date manipulations and regular expressions.

### Converting names to initials

Using string manipulations, you can convert a column containing names to a column containing initials.

#### Difficulty

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
</table>

#### Objective

You have a column containing the first and last names of all the people in a particular group. You would like to create a column containing just the initials of each person.

#### Solution

```xml
<base table="pub.doc.retail.altseg.stores"/>
<willbe name="first" value="taketo(manager;' ')"/>
<willbe name="second" value="dropto(manager;' ')"/>
<willbe name="first_initial" value="first(first;1)"/>
<willbe name="second_initial" value="first(second;1)"/>
<willbe name="initials" label="Initials"
   value="splice(first_initial second_initial;'')"/>
<colord hide="first,second,first_initial,second_initial"/>
```

#### Alternate solution

```xml
<base table="pub.doc.retail.altseg.stores"/>
<willbe name="initials" label="Initials"
   value="splice(first(manager;1) first(dropto(manager;' ');1);'')"/>
```

#### Discussion

When it is not necessary to retain a complete string within a column, converting to initials can be useful and simplify your data table. In this solution, string functions are used to extract the first letter from the first and last name of a person. These letters are then combined to create a column containing just the person's initials.

In the main solution, the first and last names are initially split into two separate columns, and then two additional columns are created to hold the first letter from those columns. The `initials` column is then created using the `splice(X;Y)` function, which combines the columns containing the first and second initials. Finally, the `<colord>` operation hides the intermediate columns that were used in the transformation.

The alternate solution uses the same functions but combines them into one line to simplify the code and avoid the need to create unnecessary columns. This solution also avoids using `taketo(X;Y)` by simply using the first letter in the original column that contained the name.
Further reading
If you would like to learn more about the functions and operations discussed in this recipe, click on the links below:

- taketo(X;Y)
- dropto(X;Y)
- splice(X;Y)

Transforming dates from text to integers

Sometimes dates are stored in a variety of text formats for display purposes, but in order to operate on these dates using 1010data functions, they need to be transformed to 1010data's standard YYYYMMDD integer format.

Difficulty

Objective

You have a column containing the date for every row in your table, but the dates are stored as text instead of integers. You want to transform each date into 1010data's standard integer format (YYYYMMDD).

Solution

Transform a date stored as a string in the form MMDDYYYY (e.g., 01042015)

```xml
<willbe name="new_date" value="int(splice(strextract(date_column;5;4) strextract(date_column;1;2) strextract(date_column;3;2);''))" format="type:date"/>
```

Transform a date stored as a string in the form MM-DD-YYYY (e.g., 01-04-2015)

```xml
<willbe name="new_date" value="int(splice(strpick(date_column;'-';3) strpick(date_column;'-';1) strpick(date_column;'-';2);''))" format="type:date"/>
```

Transform a date stored as a string in the form MM/DD/YYYY (e.g., 01/04/2015)

```xml
<willbe name="new_date" value="int(splice(strpick(date_column;'/';3) strpick(date_column;'/';1) strpick(date_column;'/';2);''))" format="type:date"/>
```

Transform a date stored as a string in the form MM/DD/YY (e.g., 01/04/15)

```xml
<willbe name="new_date" value="int(splice('20' strpick(date_column;'/';3);'') strpick(date_column;'/';1) strpick(date_column;'/';2);''))" format="type:date"/>
```

Transform a date stored as a string in the form M/D/YYYY (e.g., 1/4/2015)

```xml
<willbe name="new_date" value="int(splice(strpick(date_column;'/';3);''))" format="type:date"/>
```
if(width(strpick(date_column;'/';1))=1;  
splice('0' strpick(date_column;'/';1);'');  
strpick(date_column;'/';1))  
if(width(strpick(date_column;'/';2))=1;  
splice('0' strpick(date_column;'/';2);'');  
strpick(date_column;'/';2));''))"  
format="type:date"/>

Transform a date stored as a string in the form M/D/YY (e.g., 1/4/15)

<willbe name="new_date"  
value="int(splice(splice('20' strpick(date_column;'/';3);'')  
if(width(strpick(date_column;'/';1))=1;  
splice('0' strpick(date_column;'/';1);'');  
strpick(date_column;'/';1))  
if(width(strpick(date_column;'/';2))=1;  
splice('0' strpick(date_column;'/';2);'');  
strpick(date_column;'/';2));''))"  
format="type:date"/>

Transform a date stored as a string in the form DD-Mon-YY (e.g., 04-Jan-14)

<willbe name="date_reorder"  
value="splice(strdrop(strtake(date_column;'-';2);'-';1)  
strtake(date_column;'-';1) strdrop(date_column;'-';2);'-')="/  
<willbe name="date_replace" value="repstr(date_reorder;  
'Jan' '01' 'Feb' '02' 'Mar' '03' 'Apr' '04' 'May' '05' 'Jun' '06'  
'Jul' '07' 'Aug' '08' 'Sept' '09' 'Oct' '10' 'Nov' '11' 'Dec' '12')="/  
<willbe name="new_date"  
value="int(splice(splice('20' strpick(date_replace;'-';3);'')  
strpick(date_replace;'-';1) strpick(date_replace;'-';2);''))"  
format="type:date"/>

Discussion

In 1010data, dates are stored internally as 8-digit integers with the form YYYYMMDD. For instance, January 4, 2015, would be stored in 1010data as 20150104. Though storing dates this way is more efficient in terms of processing speeds, dates are not commonly presented in this way. 1010data uses display formats to show dates in more familiar forms. For example, 20150104 could be displayed as 01/04/15 using the date format, or as 01/04/2015 using the date4y format. Regardless of how this date is displayed, it is still stored as the integer 20150104.

However, sometimes dates are not loaded into 1010data as integers. Sometimes, dates are loaded as text values, and the forms that these strings can take may vary. Some examples are:

- 1/4/15
- 01042015
- 01-04-2015
- 1/4/2015

Dates stored as text are not recognized by 1010data as dates and can’t be used in any of the date-related functions. For instance, the function year(X) returns the year portion of a date value. However, if you try to provide this function with a date stored as text, you’ll get an error.

Note: You can find out the data type of the column containing your date values by clicking the Show Information icon at the top of the column in the 1010data Trillion-Row Spreadsheet.

This solution provides some simple manipulations to transform various text formats into the proper integer type so that they can be recognized and processed correctly by 1010data. All of the transformations assume that the text column in which the dates are stored is named date_column, although that name is arbitrary and purely for example.
Common errors

```
eslc{} An error was trapped at subprocess (nj3023/13015). The error message was: While evaluating computed column new_date the following error occurred: Internal (type) error
```

Remember that all of these date formats are stored as text values. If you try to change dates saved as integers, you will get this error. To avoid this error, double check the data type of the values in the column by clicking the Show Information icon ( ) at the top of the column.

Further reading

If you would like to learn more about the functions and operations discussed in this recipe, click on the links below:

- `splice(X;Y)
- `strextract(X;P;N)
- `strpick(X;Y;I)
- `strtake(X;Y;I)
- `if(C1;R1;C2;R2;...;D)

Determining days between entries

Using a combination of `days and `g_rshift, you can determine how many days have elapsed between the previous occurrence of a particular value and the current entry.

Difficulty

```
Difficulty
```

Objective

You want to determine how many days have passed from the last occurrence of a particular value to the current entry. For example, you might want to know how long it has been since the adjusted closing price of a stock was over a certain amount or how many days have passed since a customer last visited a particular store.

Solution

```
<base table="pub.doc.retail.altseg.sales_detail_transid"/>
<sel value="customer='2768ac88'"/>
<tabu label="Visit Dates" breaks="customer,store,trans_date">
  <tcol source="trans_date" fun="first" name="visit_date" label="Date of Visit"/>
</tabu>
<colord hide="date"/>
<willbe name="store103" value="store=103" label="Visited Store 103"/>
<willbe name="days_since" label="Days Since Last Visit" value="days(g_rshift(customer;store103;date;date;-1);trans_date)"/>
```

Discussion

This recipe determines the number of days between a customers most recent two visits.
In order to use the appropriate `G` argument later in the query, you need to perform a tabulation because the table is segmented by transaction ID and not by customer. You can use `fun="first"` to obtain the first instance of every date.

A reference column named `store103` to identify the rows in which the customer visited store 103. The function `g_rshift(G;S;O;X;N)` finds the previous occurrence within that group of rows in relation to the current row, and the `days(X;Y)` function determines the number of days between them.

**Further reading**

If you would like to learn more about the functions and operations discussed in this recipe, click on the links below:

- `g_rshift(G;S;O;X;N)`
- `days(X;Y)`
Data cleansing

Includes recipes conveying how to denormalize, deduplicate, and overall cleanse data sets.

Removing duplicate rows from a table

Sometimes a table can contain multiple copies of the same information. Whether these duplicates were original to the table, or they occurred after combining data sets, you can easily locate and remove all of the duplicate rows from a table.

Difficulty

![Difficulty Chart]

Objective

You merged two tables together and noticed that there are now some rows that contain the same information. You want to remove all of these duplicate rows from your worksheet.

Solution

```xml
<base table="pub.doc.gov.area.x2003"/>
<merge table2="pub.doc.gov.county.x2003"/>
<tableu label="Tabulation on Worksheet" breaks="msa,countyname,rent50_0">
    <tcol source="rent50_0" fun="first" label="First Rent"/>
</tableu>
<link table2="pub.doc.gov.area.x2003" col="msa,countyname,rent50_0" col2="msa,countyname,rent50_0" type="select"/>
</link>
<merge table2="pub.doc.gov.county.x2003"/>
```

Alternate solution

```xml
<base table="pub.doc.gov.area.x2003"/>
<merge table2="pub.doc.gov.county.x2003"/>
<willbe name="test" value="g_cnt(msa,countyname,rent50_0);>1"/>
<sel value="g_first1(msa,countyname,rent50_0);"/>
```

Discussion

Sometimes tables contain duplicate information. Additionally, when you merge tables, the resulting worksheet can contain duplicate information. This can cause problems when running analyses and could skew your results. This recipe discusses two ways to locate and remove duplicate information from two merged tables.

When finding duplicates, you need to establish the columns that uniquely identify each row. For example, a table containing sales information could have rows that can be uniquely identified by the transaction ID and the item SKU. In this recipe, rows are uniquely identified by the columns msa, countyname, and rent50_0.

The first solution uses a tabulation where the aforementioned columns are used for grouping and the first entry in each group is selected. Then the original merged worksheet is linked in and all of the duplicate rows are no longer present.
Alternatively, you can use $g\_cnt(G;S)$ to discover if there are any duplicates present in the table using the unique identifiers and then use $g\_first1(G;S;O)$ to select only the first instance. This method is particularly useful if your table is segmented.

**Further reading**

If you would like to learn more about the functions and operations discussed in this recipe, click on the links below:

- $g\_cnt(G;S)$
- $g\_first1(G;S;O)$

**Removing values $n$ standard deviations from the mean**

You can determine outliers by identifying values in specific columns that fall $n$ standard deviations outside of the mean for that column in a given data set. Rows that fall outside the desired range can then be eliminated.

**Difficulty**

![Difficulty scale](image)

**Objective**

You are working with a table that contains outlier values which could skew your analysis. You want to identify all of the values that fall outside $n$ standard deviations from the mean and eliminate them from your data set to get a more accurate picture.

**Solution**

```xml
<base name="pub.demo.weather.wunderground.observed_hourly"/>
<willbe name="avg_tempi" value="g\_avg(zipcode;;tempi)"/>
<willbe name="std_tempi" value="g\_std(zipcode;;tempi)"/>
<willbe name="lower\_limit" value="avg_tempi-(1*std_tempi)"/>
<willbe name="upper\_limit" value="avg_tempi+(1*std_tempi)"/>
<sel value="between(tempi;lower\_limit;upper\_limit)"/>
```

**Discussion**

Having outlier values in your data set can cause your analyses and aggregations to be skewed. Creating a range of acceptable values to select on will help improve results. This recipe selects values lying within one standard deviation of the mean, but you can change this number to create a different selection range.

This solution uses two $g\_functions$: one to determine the average of the values in the data, $g\_avg(G;S;X)$, and one to determine the standard deviation, $g\_std(G;S;X)$. Upper and lower limits are created by adding and subtracting $n$ (in this case 1) multiplied by the standard deviation from the average temperature. Then, to eliminate the values further than one standard deviation away, only the values within this range are selected.
Further reading

If you would like to learn more about the functions and operations discussed in this recipe, click on the links below:

- $g_{avg}(G;S;X)$
- $g_{std}(G;S;X)$
- $between(X;Y;Z)$
QuickApps

Includes recipes to help develop and customize QuickApps.

Layout attributes
Recipes that explain how to change the layout of your QuickApp.

Creating layouts with multiple tab panels
You can effectively organize your QuickApp and its widgets by creating multiple tab panels.

Difficulty

Objective
You have created several widgets but want a more organized way to view them. By creating multiple tab panels, you are able to view each widget separately but still maintain easy access to all widgets.

Solution

```xml
<dynamic>
  <layout type="tabpanel" margin="5px 0px 0px 12" background="#F3F1F0" height="480" width="1120">
    <layout label="Sales by Store" height="420" background="#F3F1F0">
      <widget class="graphics" width="1098" height="420">
        base="pub.doc.retail.altseg.sales_detail_transid">
          <tabu label="Tabulation on Sales" breaks="store">
            <tcol source="xsales" fun="sum" label="Sum by Store"/>
          </tabu>
          <graphspec>
            <chart type="pie" title="Sales by Store"/>
          </graphspec>
        </widget>
      </layout>
    </layout>
    <layout label="Sales by Date" height="420" background="#F3F1F0">
      <widget class="graphics" width="1000" height="420">
        base="pub.doc.retail.altseg.sales_detail_transid">
          <sel value="year(trans_date)=2015"/>
          <col name="trans_date" format="type:date"/>
          <tabu label="Tabulation on Sales" breaks="trans_date">
            <tcol source="xsales" fun="sum" label="Sum of Sales"/>
          </tabu>
          <graphspec width="888" height="480">
            <chart type="line" title="Sum of Sales Over Time" samples="100000">
              <data x="trans_date" y="t0"/>
              <grid hide="1"/>
              <axes xlabel="Date" ylabel="Sum of Sales"/>
              <ticks xrot="45" yrot="0"/>
            </chart>
          </graphspec>
        </widget>
      </layout>
    </layout>
  </layout>
</dynamic>
```
Discussion

Tabpanels are simply one type of layout available for organizing your QuickApps. You use the `type_` attribute to declare which layout you would like to use. This recipe declares the type to be `tabpanel` in the main layout. In this solution, three individual tab panels, labeled here as Sales By Date, Sales By Store, and Data, are created as sub-layouts within the main tabpanel layout. These labels were specified by entering the label values in the `label_` attribute. Then, within each tab layout, you can create the desired widget that will be shown within that tabpanel.

The image below shows the QuickApp in this solution with each widget on a separate tab panel.

![Sales by Store](image)

Common errors

Only `<layout>` tags are permitted in `<layout type_="tabpanel">`. Tabpanel is limited in regards to nested layouts. Once you have used the tabpanel in each tab, you can only incorporate images or widgets. The tabpanel prevents the user from adding additional layouts, for instance, another tabpanel within one of the original tabs. If you attempt to add another tabpanel (i.e., `<layout type_="tabpanel">`), you will get this error.

Further reading

If you would like to learn more about the functions and operations discussed in this recipe, click on the links below:

`<layout>`
Creating layouts with nested splitters

You can divide your QuickApp view into individual sections by using the layout type `splitter`.

**Difficulty**

![Difficulty Scale](image)

**Objective**

You want to create multiple sections in your QuickApp that can hold additional layouts and/or widgets. By wrapping your code in a `<layout>` of `type`="splitter", you can create different sections by nesting additional `<layout>` elements within the main splitter layout and declaring each nested layout's percentage of the total width.

**Solution**

```xml
<dynamic>
  <layout type="splitter" width="100%" height="100%" arrange="h">
    <layout width="10%">
      <widget class="text" text="1st Pane"/>
    </layout>
    <layout width="30%">
      <layout>
        <widget class="text" text="2nd Pane"/>
      </layout>
      <layout type="tabpanel">
        <layout label="Tab One">
          <widget class="button"/>
        </layout>
        <layout label="Tab Two">
          <widget class="button"/>
        </layout>
      </layout>
    </layout>
  </layout>
</layout>

<layout>
  <layout type="splitter" width="100%" height="100%" arrange="v">
    <layout height="10%">
      <widget class="text" text="3rd Pane"/>
    </layout>
    <layout arrange="v">
      <layout>
        <widget class="text" text="4th Pane"/>
      </layout>
      <layout type="accordion" width="265" margin="10px 0px 0px 10">
        <layout label="Panel One" height="200" width="243">
          <widget class="text" text="Tab1"/>
        </layout>
        <layout label="Panel Two" height="200" width="243">
          <widget class="text" text="Tab2"/>
        </layout>
      </layout>
    </layout>
  </layout>
</layout>
</layout>
</layout>
</dynamic>
```
Discussion

The splitter layout allows you to divide your QuickApp into different sections. Thus, the end user is able to see multiple widgets simultaneously. This solution creates four different sections, and the size of each can be designated by the `width_` and `height_` attributes. The height and width of each panel of the splitter layout can be specified by providing a percentage of the total width and/or height to the `width_` and `height_` attributes of each nested `<layout>`.

The image below shows the splitter layout described in this recipe.

![Splitter Layout Example](image)

Common errors

**Invalid empty layout in `<layout>`**

- Not populating one of the layouts with a widget will result in this error.
- To avoid this error, make sure each section of your splitter sections contain at least one widget.

Further reading

If you would like to learn more about the functions and operations discussed in this recipe, click on the links below:

* `<layout>`
  - Creating layouts with accordion or collapsible panels

Using the layout type `accordion` (or `collapsible`), creates multiple panels in which additional widgets and layouts can be nested.

Difficulty

![Difficulty Scale]

**Objective**

You want to allow the user to make multiple selections but want to display each selection widget in a separate panel. By nesting each panel within a main layout of type `accordion` (or `collapsible`), each
widget is displayed in a separate area, making it easier for users to keep track of which selections they've already made.

Solution

```xml
<dynamic store_num="19" dept_num="55" year="2015" tab_value=""/>
<layout arrange="h">
  <layout type="accordion" width="265" initopen="0"
    margin="10px 0px 0px 10" openvalue="@tab_value">
    <layout label="Select Store" height="200" width="243">
      <widget name="store_num" value="@store_num" class="dropdown"
        base="pub.doc.retail.altseg.sales_detail_transid">
        <col name="trans_date" format="type:date4y"/>
        <tabu breaks="store">
          <tcol source="xsales" fun="sum"/>
        </tabu>
        <sel value="g_first1(store;;)"/>
        <color cols="store"/>
        <sort col="store" dir="up"/>
      </widget>
    </layout>
    <layout label="Select Department" height="300" width="243">
      <widget name="dept_num" class="dropdown" value="@dept_num"
        base="pub.doc.retail.altseg.sales_detail_transid">
        <col name="trans_date" format="type:date4y"/>
        <tabu breaks="dept">
          <tcol source="xsales" fun="sum"/>
        </tabu>
        <sel value="g_first1(dept;;)"/>
        <color cols="dept"/>
        <sort col="dept" dir="up"/>
      </widget>
    </layout>
    <layout label="Select Year" height="300" width="243">
      <widget name="year_sel" inputwidth="217"
        margin="8px 0px 0px 8" value="@year">
        <table>2014;2015</table>
      </widget>
    </layout>
  </layout>
  <layout type="tabpanel" width="900">
    <layout label="Sales">
      <widget name="xsales_num" class="graphics" label="Sales"
        base="pub.doc.retail.altseg.sales_detail_transid" width="800">
        <sel value="store={@store_num}"/>
        <sel value="dept={@dept_num}"/>
        <sel value="year(trans_date)={@year}"/>
        <col name="trans_date" format="type:date4y"/>
        <tabu breaks="trans_date">
          <tcol name="xsales_sum_total" source="xsales" fun="sum"/>
        </tabu>
      </widget>
      <graphspec>
        <chart type="line" title="Sum of Sales Over Time"
          samples="100000">
          <data x="trans_date" y="xsales_sum_total"/>
          <grid hide="1"/>
          <axes xlabel="Date" ylabel="Sum of Sales"/>
          <ticks xrot="45" yrot="0"/>
        </chart>
      </graphspec>
    </layout>
  </layout>
</layout>
```
Accordion and collapsible panels allow you to organize your widgets but still keep them all accessible to the user. These layouts are particularly useful if you want the end user to be able to make a series of selections using multiple widgets.

In this recipe, three selection widgets are created: two drop-down lists and a checklist. The selections are used to determine data displayed in two different charts. To allow the user to go through each selection more efficiently, they are all placed in different sections using type_="accordion" in the main layout. With this type, only one section can be open at a time. To allow the user to open more than one section at a time, you can replace the layout type accordion with the type collapsible.

Note that even though accordion and collapsible can be used interchangeably, there are scenarios where it is better to use one or the other. For example, if there are a plethora of different sections where opening all of them would cause the user to scroll, it is best to use accordion.

In the image below, you can see the solution using the accordion layout, and the last section is open.
Further reading

If you would like to learn more about the functions and operations discussed in this recipe, click on the links below:

<layout>

Configuring a QuickApp for render targets

This recipe demonstrates how to create an interface that gives the end user different options for downloading data grids and charts from a QuickApp.

Difficulty

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
</table>

Objective

You want to give the end user the option to export the information created in your QuickApp to an Excel worksheet or a CSV file based on the user’s inputs. These inputs specify which widgets the user would like to include in the export, the name of the file that is to be downloaded, and if the user desires an Excel output, whether or not the information should be exported to one worksheet or multiple worksheets. By creating additional widgets where class_="button" and type_="render", you can specify attributes to give the end user control of what information is downloaded and where it is saved.

Solution

<dynamic startdate="20150101" enddate="20160101" storenum="19" deptnum="36" groupnum="468" target_selection="xlsx" widget_selection="" downloadname="" oneshheet="" many sheets="0" graph="1" grid="1" chart="1">
  <do onchange_="@widget_selection">
    <if test="{contains(@widget_selection;"retail_data")=1}">
      <set grid="1"/>
    </then>
    <else>
      <set grid="0"/>
    </else>
  </do>
</layout>
<if test="{contains(@widget_selection;'sales_graph')=1}">
    <set graph="1"/>
</if>
<if test="{contains(@widget_selection;'store_perf')=1}">
    <set chart="1"/>
</if>
<if test="{@widget_selection=''}">
    <set chart="1"/>
    <set graph="1"/>
    <set grid="1"/>
</if>
<layout arrange_="h">
    <layout type_="tabpanel">
        <layout label_="Data Grid">
            <widget name="retail_data" class_="grid" require_="{@grid}"
                invmode_="hide" base_="pub.doc.retail.altseg.sales_detail_transid">
            <sel value="between(trans_date;{@startdate};{@enddate})"/>
            <sel value="store={@storenum}"/>
            <sel value="dept={@deptnum}"/>
            <sel value="group={@groupnum}"/>
        </widget>
    </layout>
    <layout label_="Sales Over Time">
        <widget name="sales_graph" class_="graphics" require_="{@graph}"
            invmode_="hide" base_="pub.doc.retail.altseg.sales_detail_transid">
        <sel value="between(trans_date;{@startdate};{@enddate})"/>
        <col name="trans_date" format="type:date4y"/>
        <tabu breaks="trans_date">
            <tcol name="sum_of_sales" source="xsales" fun="sum"/>
        </tabu>
        <colord cols="sum_of_sales,trans_date"/>
        <graphspec>
            <chart type="line" title="Sum of Sales Over Time"
                samples="100000">
            <data x="trans_date" y="sum_of_sales"/>
            <grid hide="1"/>
            <axes xlabel="Date" ylabel="Sum of Sales"/>
            <ticks xrot="45" yrot="0"/>
        </chart>
        </graphspec>
    </layout>
    <layout label_="Store Performace">
        <widget name="store_perf" class_="graphics" require_="{@chart}"
            invmode_="hide" base_="pub.doc.retail.altseg.sales_detail_transid">
        <sel value="between(trans_date;{@startdate};{@enddate})"/>
        <sort col="store" dir="up"/>
    </layout>
</layout>
<tabu breaks="store">
  <ttcol name="sum_of_sales_by_store" source="xsales" fun="sum"/>
</tabu>
<graphspec>
  <chart type="pie">
    <data x="store" y="sum_of_sales_by_store"/>
  </chart>
</graphspec>
</widget>
</layout>

<layout>
  <widget class_="dropdown" label_="Select a Target"
    value_="@target_selection">
    <table>xlsx,data;Excel,CSV
  </table>
  <transpose/>
</widget>
  <widget class_="dropdownlist" label_="Select Widgets to Render"
    value_="@widget_selection">
    <if test="{"@target_selection='data'}">
      <then>
        <table>retail_data,Data Grid
      </table>
    </then>
    <else>
      <table>retail_data,sales_graph,store_perf;Data Grid,
        Sales Over Time,Store Performance
    </table>
    <transpose />
    </else>
  </if>
</widget>
  <widget class_="field" label_="Name of File:" value="@downloadname"/>
  <widget class_="button" type="render" target_="{"@target_selection}" 
    text="Click to Export on One Sheet" include_="{"@widget_selection}" 
    filename="{"@downloadname}" require_="{"@target_selection='xlsx'}"
    invmode_="hide"/>
  <widget class_="button" type="render" target_="{"@target_selection}" 
    text="Click to Export on Multiple Sheets" 
    filename="{"@downloadname}" require_="{"@target_selection='xlsx'}"
    invmode_="hide"/>
  <widget class_="button" type="render" target_="{"@target_selection}" 
    text="Click to Export" include_="retail_data"
    filename="{"@downloadname}" require_="{"@target_selection='data'}"
    invmode_="hide"/>
</layout>
</dynamic>

**Discussion**

Using the render functionality is a convenient way to get certain QuickApp widgets into various file formats. As a QuickApp developer, it is usually a good idea to give your users options. Since both Excel and CSV file formats are supported by render, it makes sense to give the user the option to select which file format they want to download. You can provide further flexibility by allowing them to decide which of the available widgets should be included in the render as well.

In this recipe, the user is provided this functionality by parametrizing a few values. First, create a variable based on the file format selection that will be made available to the render button. However, you must consider that not all render targets support the same widgets. For instance, the Excel render target
supports charts, while the comma-separated list target (data) does not. So while you have given the user the ability to select the target type, you also want to avoid providing options that aren't applicable. Therefore, it makes sense to only allow the selection of a widget if the render target selection supports that type.

As you can see from the tab panels in this example, there are only two types of widgets: a grid and two charts. If the user selects the Excel target, you want to give them the option to select any number of the widgets in the tab panels. If the user selects the CSV target, only one widget in this example is supported by that target. Therefore, you only give the user the option to select the grid widget when CSV is selected.

This can all be accomplished with two variables, which are initialized in the opening tag of the <dynamic> element. The target_selection variable holds the value that is passed to the target_attribute in the render button to convey which type of file should be created. The widget_selection variable holds the list of widgets that the user wants to include in the render. By using a <do> statement that begins when a widget selection is made, you can assign values of 0 or 1 to each widget based on the widget_selection contents to determine which widgets should be included or excluded from the downloaded file. The require_attribute is used to ensure that the widget selection drop-down list is only available when the target_selection variable holds the value xlsx, and the individual widgets are only available when the user selects them. And finally, the invmode_attribute is used to hide each widget when the require_attribute is false.

A third variable, filename, can be included to give the user the option to name the output file. The widget with attribute class="field" creates a text box that stores the user's input in the created variable. If no filename is entered, it will automatically be named download.xlsx or download.csv, depending on the render target chosen.

If an Excel sheet is needed by the user, you can create further adaptability by generating two different render buttons, one for exporting all widgets to a single worksheet and one to render each widget on a different worksheet. The button to render on a single worksheet uses the include_attribute to include all of the widgets stored in the widget_selection variable. Not including this attribute in the button to render to multiple worksheets causes all of the widgets selected to be exported to their own worksheet. This is the default behavior.

After the user makes their selections, clicking the render button will download a file in the specified format with the supported widgets selected by the user. If the Excel target is chosen, all selected widgets will appear in the specified number of worksheets. If CSV is chosen for the target, only the information from the grid will appear in the file because charts are not supported in that format.
Option 1: Single Excel worksheet

![Excel Sheet 1]

Option 2: Multiple Excel worksheets

![Excel Sheet 2]

Option 3: CSV file

![CSV File]

Common errors

Unknown target_ for render

While initializing the `target_selection` variable and the `widget_selection` variable in `<dynamic>`, a default value is required for `target_selection`. If one is not provided, you will receive this error.

This occurs because render cannot function without a specified target, while if nothing is stated for `include_`, all widgets will simply be included. Solving this error is as simple as providing a default of either `xlsx` or `data` when the variable is declared. However, keep in mind that if the default is `data`, the widget selector will be hidden due to the `invmode_="hide"` attribute.

Rendering Workbook failed (Too many rows and/or columns).

Another possible error occurs when you attempt to download a table that exceeds the allowable size limit. When this occurs, this message will pop up.

To avoid this error, be mindful of the size of your data and try to use selections to only obtain the necessary information.

Further reading

If you would like to learn more about the functions and operations discussed in this recipe, click on the links below:

```
type_="render"
class_="dropdown"
```
Creating a table with clickable columns and rows

Using the clickable attribute in a QuickApp allows the user to interact with the data table and select specific columns and rows. With this attribute, you can create a clickable grid that displays the selected values in a chart.

Difficulty

Objective

You want to give the end user the option to select the data they want to display in a chart by clicking on rows and columns in a table.

Solution

```xml
<dynamic pkgcolval="{pkg();}" go="0" back="" colvar="zipcode" valvar="">
    <do onchange="pkgcolval">
        <if test="{'{pkg_get(@pkgcolval;'name')}'<>''}">
            <set go="1"/>
            <set valvar="28117"/>
        </if>
    </do>
    <layout arrange="h">
        <layout arrange="v">
            <widget class="grid" type="scroll"
                clickablecolhead="meantempi,meanpressurem,meanwspdi"
                pkgcolvalue="@pkgcolval"
                height="350" base="pub.demo.weather.wunderground.observed_daily"
                clickable="zipcode" value="@valvar">
                <sel value="date>20130600"/>
                <colord cols="zipcode,date,meantempi,meanpressurem,meanwspdi"/>
            </widget>
            <widget class="graphics" invmode="hide" require="{@go}"
                base="pub.demo.weather.wunderground.observed_daily">
                <sel value="date>20130600"/>
                <sel value="{@colvar}={@valvar}"/>
                <graphspec width="600" height="375">
                    <chart type="line" samples="1000"
                        title="{pkg_get(@pkgcolval;'label')} over time">
                        <data x="date" y="{pkg_get(@pkgcolval;'name')}/">
                            <legend hide="1"/>
                            <axes xlabel="Date" ylabel="{pkg_get(@pkgcolval;'label')}/"></axes>
                            <ticks xrot="25"/>
                        </data>
                    </chart>
                </graphspec>
            </widget>
        </layout>
    </layout>
    <widget class="button" text="Go Back" invmode="hide"
        require="{@go}" type="set" value="@go" newvalue="0"/>
</dynamic>
```
Discussion
Creating a clickable grid allows the end user to have more interaction with the table. If a chart is created using the information in the table, the user is then able to select which information should be displayed. This creates a versatile widget where the user has the ability to view different data combinations in a chart.

This solution uses three variables, pkgcolval, colvar, and valvar, to define the data that is displayed in the chart. colvar contains the column name of which values will become clickable using the clickable attribute in the grid widget. valvar holds the value that the user selects from this column in the table. Column headers can also be made clickable by passing column names to the clickablecolhead attribute. When a column header is selected by the user, a package is created with all of the information about the column and is stored in the pkgcolval variable. This makes it so the column name and label can be accessed by the chart.

After the user clicks on the desired data, the QuickApp should look like the following image.

Common errors
Undefined @variable referred to in: zipcode={@valvar}

Upon selecting a column header, if a row value is not selected first, you will see this error.
Simply initializing the `valvar` variable in the `<dynamic>` will not fix this error. Instead, when a column header is selected a `<do onchange_="@pkgcolval">` operation occurs to temporarily assign a value to `valvar` until the user makes a different selection.

Further reading

If you would like to learn more about the functions and operations discussed in this recipe, click on the links below:

```html
<chart>
class_="grid"
pkg_names(X)
```

Formatting data in a grid

You can use a slider widget to select data in a grid where the values are formatted based on the selection.

### Difficulty

![Difficulty scale](image)

### Objective

You want to create a chart driven by conditionally formatted data selected using a slider. The slider ranges from 0% to 100%, and the user can move both ends to select their desired percentage of data. When data is selected, you want the values to be formatted black in the table, and if the data is not selected you want it to appear red.

### Solution

```html
<meta>empty</meta>
<defblock name="repeat">
  <base table="pub.fin.econ.rates.libor"/>
  <willbe name="month" value="month(date)"/>
  <willbe name="year" value="year(date)"/>
  <sel value="g_first1(month,year;;)"/>
  <sel value="libor6m>0|libor12m>0"/>
  <sel value="date>19890101"/>
  <willbe name="tot_rows" value="/n "/>
  <willbe name="ind_rows" value="/i "/>
  <willbe name="sixmonthhigh" value="g_hi(;;libor6m)"/>
  <willbe name="twelvemonthhigh" value="g_hi(;;libor12m)"/>
  <willbe name="ninetofive" value="95"/>
</defblock>
<dynamic six_factor_left="0" six_factor_right="100" twelve_factor_left="0" twelve_factor_right="100" num_rows="" max_found="0" b_6_l="0" b_6_r="10"
b_12_l="0" b_12_r="10">
  <do on="init" value_1_="" num_rows_1_="" col_1_="" row_1_="" value_2_="" b_6_r_="" col_2_="" sixmonthhigh_2_="" row_2_="" value_3_="" b_12_r_="" col_3_="" twelvemonthhigh_3_="" row_3_="" value_4_="" max_found_="" col_4_="" twelvemonthhigh_4_="" row_4_="" value_5_="" six_factor_right_="" col_5_="" ninetofive_5_="" row_5_="" value_6_="" twelve_factor_right_="" col_6_="" ninetofive_6_=""/>
</dynamic>
```
<insert block="repeat"/>
</do>
<do onchange="six_factor_left" value="b_6_l" col="libor6m">
  <insert block="repeat"/>
  <sort col="libor6m" dir="up"/>
  <sel value="ind_rows={floor(@six_factor_left*@num_rows/100)}+1"/>
</do>
<do onchange="six_factor_right" value="b_6_r" col="libor6m">
  <insert block="repeat"/>
  <sort col="libor6m" dir="up"/>
  <sel value="ind_rows={floor(@six_factor_right*@num_rows/100)}-1"/>
</do>
<do onchange="twelve_factor_left" value="b_12_l" col="libor12m">
  <insert block="repeat"/>
  <sort col="libor12m" dir="up"/>
  <sel value="ind_rows={floor(@twelve_factor_left*@num_rows/100)}+1"/>
</do>
<do onchange="twelve_factor_right" value="b_12_r" col="libor12m">
  <insert block="repeat"/>
  <sort col="libor12m" dir="up"/>
  <sel value="ind_rows={floor(@twelve_factor_right*@num_rows/100)}-1"/>
</do>
<layout arrange_="v">
  <layout arrange_="h">
    <widget class_="grid" type_="scroll" label_="LIBOR">
      base_="pub.fin.econ.rates.libor" colstwtfbgcolor_="libor6m,libor12m"
      fgcolorforcol_="libor6m_color,libor12m_color" rowcol_="0">
        <willbe name="month" value="month(date)"/>
        <willbe name="year" value="year(date)"/>
        <sel value="g_first1(month,year;;)"/>
        <sel value="libor6m<>0|libor12m<>0"/>
        <sel value="date>19890101"/>
        <sel value="libor6m<>10.68"/>
        <willbe name="libor6m_color" value="if(between(libor6m;{@b_6_l};{@b_6_r});'black';'red')"/>
        <willbe name="libor12m_color" value="if(between(libor12m;{@b_12_l};{@b_12_r});'black';'red')"/>
        <colord cols="date,libor6m,libor12m"/>
    </widget>
  </layout>
  <layout arrange_="v">
    <widget class_="graphics" label_="LIBOR Rates Over Time">
      base_="pub.fin.econ.rates.libor"
      colord cols="date,libor6m,libor12m"/>
      <willbe name="index" value="n"/>
      <willbe name="ref6" value="between(libor6m;{@b_6_l};{@b_6_r})"/>
      <willbe name="ref12" value="between(libor12m;{@b_12_l};{@b_12_r})"/>
      <willbe name="16" value="if(ref6;libor6m:0)"/>
      <willbe name="112" value="if(ref12;libor12m:0)"/>
      <graphspec>
        <chart type="line" title="LIBOR Rates Over Time">
          <data x="date" y="16"/>
          <data x="date" y="112"/>
          <axes xlabel="Date" ylabel="LIBOR Rates" ymax="[@max_found]"/>
          <legend hide="0"/>
          <ticks xrot="20"/>
        </chart>
      </graphspec>
    </widget>
  </layout>
</layout>
Discussion

Conditional formatting in a grid widget is a useful way to provide visual feedback to an end user based on their interactions with the QuickApp. This solution uses values selected on a slider widget to determine the formatting of a grid, but input from any widget can be used.

This solution bases the selection of data on the values contained in certain percentiles. These percentiles and the rows that contain values within them are found using four <do> operations that occur when each slider is moved. An additional <do> operation, that occurs when the QuickApp is started, initializes all variables used within the code based on information contained in the data table. These variables include a left and right value for each LIBOR rate slider and the max value for each rate.

The objective of this recipe is to display values in the grid that meet a certain condition as a different color than the values that do not meet the condition. To do this, you create a series of reference columns that will contain the colors you want to apply to the values in the grid. The values in these columns will be recalculated each time the selection from the slider widget is changed. You can use the if and between functions to conditionalize which values are assigned to each row for each computed column so that if any value in a selected column lies between the left and right slider values. If the value for the given data point is between the selected values, it should be formatted in black font, otherwise, it should be formatted in red.

The screenshot below shows the final product. Notice the initial slider is set to select values between 0 and 95%, therefore all LIBOR rates contained in this range are formatted as black when the QuickApp is initialized.
Common errors

**min** attribute must be an number between -0i and 0i

The `min` and `max` attributes can take an integer or a variable that contains an integer. If you do not provide a valid entry you will get this error.

To resolve this error, make sure an integer is provided for each value. Additionally, if a variable is being utilized, double check the value of the variable and make sure you are using the correct syntax when referencing the variable (i.e., `min="(@var)"`).

Further reading

If you would like to learn more about the functions and operations discussed in this recipe, click on the links below:

```class="slider"
<do>```
Analytical Solutions

Includes recipes that demonstrate how to complete specific analysis.

Performing a time comparison analysis

Comparing aggregated data for two different time periods can help to identify patterns and changes in data.

Difficulty

Objective

You want to perform an analysis that compares the data from one time period to a previous time period. For instance, retailers often want to compare this week’s sales to last week’s sales. Another example is comparing the average temperature of July this year to the average temperature of July last year. In order to do this, the values for each time period must be in the same row but in different columns. Often, especially in time-ordered data sets, the values you want to compare are located in the same column but not in the same row. One solution is to use the 1010data function \( g_{\text{rshift}}(G;S;O;X;N) \) to create a new column that contains the values shifted by the time period required by the analysis. Alternatively, you can compute a weighted sum in a tabulation using reference columns, one for each time period, to place the desired values in new columns in the same row.

Solution

```xml
<base table="pub.doc.retail.altseg.sales_detail_transid"/>
<willbe name="month" label="Month" value="month(trans_date)"
    format="type:nomomas"/>
<tabu label="Tabulation" breaks="year,month">
    <break col="year" sort="up"/>
    <break col="month" sort="up"/>
    <tcol source="xsales" fun="sum" name="sum" label="Sum of Sales"/>
</tabu>
<willbe name="last_year" label="Last Years Sum"
    value="g_{\text{rshift}}(month;;;sum;-1)"
    format="dec:2"/>
<willbe name="yoy" label="YOY" value="sum-last_year"
    format="dec:2"/>
<willbe name="yoy_perc" label="YOY Percentage" value="yoy/last_year"
    format="type:pct;dec:2"/>
```

Alternate solution

```xml
<base table="pub.demo.weather.wunderground.observed_daily"/>
<sel value="meantempi<150"/>
<sel value="meantempi>-40"/>
<colord cols="zipcode,date,meantempm,meantempi"/>
<sel value="year(date)=2014 2015"/>
<willbe name="month" label="Month" value="month(date)"/>
<willbe name="year" label="Year" value="year(date)"
    format="type:nomomas"/>
<willbe name="mean" value="meantempi"
    format="type:num"/>
<tabu label="Tabulation on Observed Daily" breaks="month,year"/>
```
Performing a time comparison analysis allows you to examine how your data changes from one time period to another. Additionally, the summarized values for each time period should be displayed in adjacent columns to allow for easy comparison between the two.

This recipe presents two methods. The most efficient way is to use the 1010data function \texttt{g\_rshift(G;S;O;X;N)} to shift rows according to time period. In the primary solution, this function is used to create a new column that, along with additional columns, can then be used to determine the percent loss or gain from one year to the next. The second solution utilizes the dot-product (or weighted sum) method to calculate the year-over-year change for each month in the data set.

A key component in either method is having a date column with dates stored in the standard integer format of \texttt{YYYYMMDD} that 1010data uses. If the dates are not stored in this format or if they are stored as text values, it is essential to transform them into the correct format. For assistance in performing this transformation, see the recipe titled \textit{Transforming dates from text to integers} on page 29.

In the first solution, two new columns representing the month and year are used in a tabulation over each month/year combination. (Note that the sample data set, \texttt{pub.doc.retail.salesdetail}, only contains information for January of 2014 and 2015.) After the tabulation, the \texttt{g\_rshift(G;S;O;X;N)} function is used to shift the rows in the worksheet so that the same month from both years in the analysis are in adjacent rows. Once the aggregations are in the same row, calculating the difference between them, and ultimately the percentage change, is simple.

The second solution, which uses the weighted sum approach is similar, but instead of shifting rows, a column containing a boolean flag is created for each year in the analysis. After the monthly values are calculated, this flag column is then used as the reference column, which acts as a weight for the dot product in the final tabulation. Again, with the values now in adjacent rows, computing the percent loss or gain from one year to the next is straightforward.

**Further reading**

If you would like to learn more about the functions and operations discussed in this recipe, click on the links below:

\texttt{g\_rshift(G;S;O;X;N)}

**Performing an affinity analysis**

You can determine the relationship of two items based on the amount of times the items were bought in the same transaction, the number of transactions that contained each item individually and the amount of transaction over all.
Objective

You want to determine the affinity, or the relationship, between two items. Let's say, you are analyzing retail transactions and you want to know how likely it is for a certain item to be purchased with a different item. For example, hot dogs and hot dog buns have a high affinity. Or you are the owner if an ice cream shop and you want to know which flavor and which topping are most commonly ordered together.

Solution

```
<base table="pub.doc.retail.altseg.sales_detail_transid"/>
<sel value="between(trans_date;20150101;20160101)"/>
<sel value="xsales>0"/>
<note>Flag Primary Products</note>
<willbe name="sel" value="group=65"/>
<sel value="g_or(customer trans_date transid; ;sel)"/>
<willbe name="unique_trans" value="g_last1(customer trans_date transid;)"/>
<tabu breaks="sku" label="Tabulation">
  <tcol source="transid" fun="ucnte" name="bskts_both" label="# Baskets with Both"/>
  <tcol source="unique_trans" fun="sum" name="total_trans"/>
</tabu>
<willbe name="bskts_prod" value="g_sum(;;total_trans)" label="# Baskets with Product"/>
<willbe name="bskts_both_pct" value="bskts_both/bskts_prod*100" label="% of Baskets with Both" format="dec:2"/>
<note>Total Baskets of Co-Purchase Product</note>
<link table2="pub.doc.retail.altseg.sales_detail_transid" col="sku" col2="sku">
  <sel value="between(trans_date;20150101;20160101)"/>
  <sel value="xsales>0"/>
  <tabu breaks="sku" label="Tabulation">
    <tcol source="transid" fun="ucnte" name="bskts_co" label="# Baskets with Co-Purchase Product"/>
  </tabu>
</link>
<willbe name="linker" value="1"/>
<note>All Baskets in time/store range selection</note>
<link table2="pub.doc.retail.altseg.sales_detail_transid" col="linker" col2="linker">
  <sel value="between(trans_date;20150101;20160101)"/>
  <sel value="xsales>0"/>
  <tabu label="Tabulation">
    <tcol source="transid" fun="ucnte" name="bskts_all" label="Total # Baskets in Sample"/>
  </tabu>
  <willbe name="linker" value="1"/>
</link>
<willbe name="bskt_aff" label="Affinity" value="min(9999;int(bskts_both_pct/(bskts_co/bskts_all*100)*100))"/>
<sel value="bskts_both_pct>=.1"/>
<sel value="(bskts_both_pct<>100)&(bskts_co<>bskts_both)"/>
<sort col="bskt_aff" dir="down"/>
```
Discussion

Determining the relationship between items is very useful, especially in retail. If you can figure out what items are commonly bought together you can use these insights to make more effective decisions. This recipe uses demo retail sales data to determine the relationship between an item bought within group 65 to every other item bought in that basket. Below is the formula used to calculate affinity and the basis for our actions in this recipe.

\[
Affinity = \left( \frac{\text{# of Baskets with Both Products}}{\text{# of Baskets with CoPurchased Product}} \right) \times 100
\]

The solution first flags every row that contains an item from group 65 and then, using \( g_{or}(G;S;X) \), selects all baskets that contain at least one of these items. A tabulation is completed by breaking on the item SKU to determine the number of unique transactions and the number of baskets that contain both the current item and an item from group 65. By calculating the sum of unique transactions we can then determine how many total baskets there were that contained an item from group 65. Thus, the percentage of baskets containing both items can then be obtained.

From here, we need to link in additional tables. Within each link, we can perform additional calculations. These include tabulations to determine the total number of baskets in our sample and the number of baskets that contain each additional item purchased with the item in group 65. A final column uses all of our computed information to determine the affinity score for each item pair.

Further reading

If you would like to learn more about the functions and operations discussed in this recipe, click on the links below:

\( g_{or}(G;S;X) \)

Extracting loadings/eigenvectors from your PCA

Using 1010data’s \( g_{\text{function}} \), \( g_{\text{pca}}(G;S;XX;Z) \), you can create a model that corresponds to the principal component analysis of one or more variables. However, in order to determine the loadings/eigenvectors you need to use \( \text{param}(M;'evecs';J \, I) \).
Objective

You have completed a principal component analysis using \( g\_pca(G;S;XX;Z) \). Now you are interested in determining the loadings, otherwise known as eigenvectors. Using the \( \text{param} \) function with the value 'evecs' as the argument for the \( P \) parameter, you can calculate the individual elements of each eigenvector, but you want to create a matrix containing all of the loadings.

Solution

```xml
<block name="eigenvector_table"
  model_vars="age,duration,previous,empvrate,hsng,h_unk,def">
  <base table="pub.demo.mleg.uci.bankmarketing"/>
  <willbe name="yy" value="y='yes'"/>
  <willbe name="hsng" value="housing='yes'"/>
  <willbe name="h_unk" value="housing='unknown'"/>
  <willbe name="def" value="default='yes'"/>
  <willbe name="model_pca" value="g\_pca(;;{@model_vars};'method''corr')"/>
  <sel value="i_<=csl\_len('{@model_vars}')"/>
  <willbe name="row" value="i_()"/>
  <for i="1" to="{csl\_len('{@model_vars}')}">
    <willbe name="evec_{@i}" value="param(model\_pca;'evecs';{@i} row)"/>
  </for>
</block>
```

Discussion

The first part of this solution is discussed in the Principal Component Analysis tutorial, which can be accessed from the Further reading links below. Once you have your model, you can use the \( \text{param} \) function to determine the weights of the components. However, with this function you need to calculate each element individually, specifying the column and element for each eigenvector.

Storing your model variables or column names in a block variable allows you to systematically calculate all loadings. You can count the number of elements in your block variable, \( \text{model\_vars} \), using \( \text{csl\_len}(X) \), and therefore determine the number of variables included in your model. There will only be as many eigenvectors and elements within as there are variables in your model, therefore, a selection is performed to limit the number of rows to this number. You then create a column that denotes the row number.

Using a \(<\text{for}>\) loop, you can create a column for each eigenvector. \( \text{param}(M;'\text{evecs'};J I) \) takes two numbers in the last argument. The first is \( J \), for the \( J \)th eigenvector, and \( I \), for the \( I \)th element in the \( J \)th eigenvector. Therefore, you can use the current iteration number, \( {@i} \), for \( J \) and the current row number, \( \text{row} \), for \( I \).

After a \(<\text{colord}>\) statement, you are left with the matrix containing the loadings/eigenvectors for the model produced by \( g\_pca(G;S;XX;Z) \).

**Note:** Notice that you can use \( \text{evec}_* \) instead of writing out each of the column names and it will include any column name that starts with "evec_".

Further reading

If you would like to learn more about the functions and operations discussed in this recipe, click on the links below:

- \( \text{param}(M;P;I) \)
- \( g\_pca(G;S;XX;Z) \)
- Principal Component Analysis
Debugging guide

Includes recipes to help determine the origin of and fix error messages.

Determining the value of a variable

You can use a `<signal>` element to print out the value of a variable at a specified point in your query.

Objective

The query you are writing keeps giving you an error at the same point each time you run it. You suspect the problem is that one if the variables is not receiving the expected value at that point in the query, and you need to find what value the variable is actually holding.

In this example a `<foreach>` loop performs multiple tabulations using two different variables. Upon executing the query, you receive the error `Unequal length iterators in <foreach>`. You aren't sure how long the iterators are or how to make them the same length.

Solution

Here is the original code that is producing the error:

```xml
<block name="computation" colname="xsales,qty,cost" seg="transid">
  <base table="pub.doc.retail.altseg.sales_detail_transid"/>
  <loop with_="columns">
    <outer>
      <base table="pub.doc.retail.altseg.sales_detail_transid"/>
      <columns/>
      <color cols="label"/>
      <set collist="{str_to_lst(@colname;',')}"/>
      <set collist="'{lst_to_str(@collist;'
      \n')}'"/>
      <sel value="name={@collist}"/>
    </outer>
    <inner>
      <letseq names="{@columns}" keys="{pkg_names(@names)}">
        <foreach var="{@colname}" col_label="{@keys}" tally_="@i">
          <willbe name="sum_of_{@var}"
            label="Sum of {@columns.{@col_label}}"
            value="g_sum({@seg};;{@var})"/>
        </foreach>
      </letseq>
    </inner>
  </loop>
  <sel value="g_first1({@seg};;)"/>
</block>
```

First, you figure out what how many iterators are held in the variable `colname`.

```xml
...<inner>
  <signal msg="{@colname}"/>
  <letseq names="{@columns}" keys="{pkg_names(@names)}">
...```

Then, in a separate query, run the code in the outer to find out how many iterators are in the package `columns`.

```xml
<block name="computation" colname="xsales,qty,cost" seg="transid">
  <base table="pub.doc.retail.altseg.sales_detail_transid"/>
  <columns/>
```
<colord cols="label"/>
<set collist="'{str_to_lst(@colname;'','')}'/>
<set collist="'{lst_to_str(@collist;'\'\')}''/>
<sel value="name={@collist}'"/>
</block>

And the final code with improvements is:

<block name="computation" colname="xsales,qty,cost" seg="transid">
    <base table="pub.doc.retail.altseg.sales_detail_transid"/>
    <loop with_="columns">
        <base table="pub.doc.retail.altseg.sales_detail_transid"/>
        <columns/>
        <colord cols="label"/>
        <set collist="'{str_to_lst(@colname;'','')}'/>
        <set collist="'{lst_to_str(@collist;'\'\')}''/>
        <sel value="name={@collist}'"/>
        <transpose/>
    </loop>
    <inner>
        <letseq names="{@columns}" keys="{pkg_names(@names)}">
            <foreach var="{@colname}" col_label="{@keys}" tally_="@i">
                <willbe name="sum_of_{@var}" label="Sum of  
                    '{@columns.{@col_label}}'" value="g_sum({@seg};;{@var})'/>
            </foreach>
        </letseq>
    </inner>
</block>

Discussion

To fix an error that has to do with the value(s) being held in a variable(s), it is helpful to use a <signal> message. Setting the message equal to the variable in question will return the value of that variable at the point in the code where the signal is placed. You know from the error message received that the problem lies with the values of the iterators in the <foreach> operation, colname and columns. From setting colname equal to the message, you will discover that there are three elements and therefore three iterators within the variable.
However, you can’t use the same method to discover the values held in columns because a signal message cannot display the value of a package. Instead, copy the code from the outer loop which is used to create the package and run it in a separate query. You also need to include the block so that you have the values of the variables used in this section of the code. After executing this code, you obtain the table that generates the package.

The outer loop creates a package where each column name in the table is paired with the last item in the column. Therefore, this table generates a one item package. In order to create a three item package including each element in the table, you need to <transpose> this table.

Now columns is a package that contains three pairs of values and when used with <letseq> in a <foreach> loop, creates three iterations.

Further reading
If you would like to learn more about the functions and operations discussed in this recipe, click on the links below:

pkg (K; V)
Determining the value held by a variable

Within a QuickApp, you can use text widgets to determine the value being held by a variable at any given point in time.

Objective

Upon executing your query, a widget in your QuickApp doesn’t function as expected or is invalidated. Because this widget depends on the values selected and assigned to variables earlier in the query, you think the problem lies with the assignment. You need to determine what value the variable is holding at different points in time.

In this recipe, a nested widget is used to create several drop-down lists from a `<foreach>` loop. The values selected by these drop-down lists are then used to validate and select rows in a grid widget. Even after a value is selected from each drop-down list, the grid widget is still not being validated. You need to determine what value is being held by each of these variables.

Solution

Original code with invalidated widget:

```xml
<dynamic store="19,25,55" selection1="" selection2="" selection3="">
  <layout arrange_="h">
    <layout>
      <widget class_="nest">
        <dynamic selection1="" selection2="" selection3="">
          <layout>
            <foreach num="{@store}" tally_="@i">
              <widget class_="dropdown" label_="Store {@num}" value_="@selection{@i}" num="{@num}" base_="pub.doc.retail.altseg.sales_detail_transid">
                <sel value="store={@num}"/>
                <colord cols="dept"/>
                <tabu breaks="dept">
                  <tcol name="first" source="dept" fun="first"/>
                  </tabu>
                </foreach>
            </layout>
          </dynamic>
        </layout>
      </widget>
    </layout>
  </layout>
</dynamic>
</layout>
<widget class_="grid" require_="{"@selection1}" "&@selection2}" "&@selection3}"">
  <layout>
    <widget class_="nest">
      <dynamic>
        <foreach num="{@store}" tally_="@i">
          <widget class_="dropdown" label_="Store {@num}" value_="@selection{@i}" num="{@num}" base_="pub.doc.retail.altseg.sales_detail_transid">
            <sel value="store={@num}"/>
            <colord cols="dept"/>
            <tabu breaks="dept">
              <tcol name="first" source="dept" fun="first"/>
              </tabu>
            </foreach>
        </dynamic>
      </layout>
    </widget>
  </layout>
</widget>
</layout>
</dynamic>
```

First, you put a text widget after the drop-down lists:

```xml
...  </widget>
```
Then, you put the same text widget before the grid widget:

```xml
<widget class_="text"
text_="Values= {@selection1} {@selection2} {@selection3}"/>
</layout>
</dynamic>

Once you have identified the issue and implemented a fix, you can remove the text widgets. For this example, the final code is:

```xml
<dynamic store="19,25,55" selection1="" selection2="" selection3="">
<layout arrange_="h">
  <note>Below is the change from the original code, using the bind_ and to_ attributes</note>
  <widget class_="nest" bind_="selection1,selection2,selection3"
to_="selection1,selection2,selection3">
    <dynamic selection1="" selection2="" selection3="">
      <foreach num_="{@store}" tally_="@i">
        <widget class_="dropdown" label_="Store
{num}" value_="@selection{@i}"
num_="{num}" base_="pub.doc.retail.altseg.sales_detail_transid">
          <sel value_="store={@num} & dept={@selection1}"
            colord cols_="dept"></colord>
          <tabu breaks_="dept">
            <tcol name_="first" source_="dept" fun_="first"/>
          </tabu>
          <sort col_="first" dir_="up"/>
        </widget>
      </foreach>
    </dynamic>
  </widget>
</layout>
</dynamic>
</widget>
</layout>
</dynamic>
```
Discussion

To determine the value held by a variable in a QuickApp, it is helpful to use a text widget. Similar to a print statement, a text widget will display the value held by that variable at the point in the query it is placed.

In this solution, you first place a text widget after the drop-down lists to verify that each list is assigning a value to each of the variables.

You can see that the variables are holding the values selected in the drop-down lists, but the grid widget is still not being validated. Therefore, the variables must be holding a different value before being passed to the parameters in the grid. Then, you create the same text widget and place it before the grid.

With this text widget, you can see that the variables are not holding any value before being passed to the grid widget. Thus, something occurs between these two text widgets that causes the change in each variable's value. Upon further examination, you realize that the values are being assigned within an inner <dynamic> and consequently are not available outside of that tag. However, by using the bind_ and to_ attributes of the nested widget, you can assign the selected values to a variable outside the inner <dynamic> making them available to the grid widget.

Further reading

If you would like to learn more about the functions and operations discussed in this recipe, click on the links below:

class_="nest"
Analogues

1010data for Excel users

1010data can solve the same problems that you are used to solving in Excel in one or more ways. Sometimes there are direct translations between the two, and sometimes a unique solution is needed in order to arrive at the same result.

Introduction

Many people use Excel to analyze and visualize their data. This section describes how to solve the same problems and create the same visualizations using 1010data.

There are two main parts of this section, how to perform similar calculations and how to obtain similar visualizations. You can perform similar analyses in 1010data using both the GUI and the Macro Language. Some analogue solutions shown in this section will have both a GUI solution and a Macro Language solution, while others might only have a Macro Language solution. Additionally, some analogues might have multiple solutions that show how to solve the problem in different ways, keep in mind that the first solution listed is often the most efficient solution and therefore is preferred.

Visualizing your data in 1010data can be done in many different ways. Most simply, you can use the chart builder in the GUI to build multiple different types of charts. Additionally, you can build QuickApps to visualize different aspects of your data and allow end users to have control over what they see.

1010data allows you to analyze large amounts of data with more efficiency and faster speeds.

Function comparisons

You know how to use certain functions in Excel, but you want to know how to complete the same tasks using 1010data’s functions. While some Excel functions have a direct match to a function in 1010data, others match to a combination of functions/operations. The following sections show some of the analogous functions between Excel and 1010data.

Date and Time Functions

<table>
<thead>
<tr>
<th>Excel Function</th>
<th>1010data Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NOW()</td>
<td>@now_</td>
<td>Returns the current date and time.</td>
</tr>
<tr>
<td>HOUR(serial_number)</td>
<td>hour(X)</td>
<td>Returns the hour portion of a time-related value.</td>
</tr>
<tr>
<td>MINUTE(serial_number)</td>
<td>minute(X)</td>
<td>Returns the minutes portion of a time-related value.</td>
</tr>
<tr>
<td>SECOND(serial_number)</td>
<td>second(X)</td>
<td>Returns the seconds portion of a time-related value.</td>
</tr>
<tr>
<td>DAY(serial_number)</td>
<td>day(X)</td>
<td>Returns the day portion of a date-related value.</td>
</tr>
<tr>
<td>MONTH(serial_number)</td>
<td>month(X)</td>
<td>Returns the month portion of a date-related value.</td>
</tr>
<tr>
<td>YEAR(serial_number)</td>
<td>year(X)</td>
<td>Returns the year portion of a date-related value.</td>
</tr>
<tr>
<td>Excel Function</td>
<td>1010data Function</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------------------------</td>
<td>-------------------</td>
<td>-------------------------------------------------------</td>
</tr>
<tr>
<td>WEEKDAY(serial_number, [return_type])</td>
<td>dayofwk(X)</td>
<td>Returns the integer value corresponding to the day of the week of a date-related value.</td>
</tr>
<tr>
<td>DAYS(start_date,end_date)</td>
<td>days(X;Y)</td>
<td>Returns the number of days between two date-related values.</td>
</tr>
</tbody>
</table>

**Logical Functions**

<table>
<thead>
<tr>
<th>Excel Function</th>
<th>1010data Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IF(logical_test, value_if_true, value_if_false)</td>
<td>if(C1;R1;C2;R2;...;D)</td>
<td>Returns the value corresponding to the first condition that evaluates to true; if no conditions are true, returns the specified default value.</td>
</tr>
<tr>
<td>IFNA(value, value_if_na )</td>
<td>ifnull(X;Y)</td>
<td>Returns a specified value when the value in a given column is N/A.</td>
</tr>
<tr>
<td>AND(logical_test1, [logical_test2], ...)</td>
<td>g_and(G;S;X)</td>
<td>For Excel, it returns TRUE if all given conditions are true for a specified value or FALSE otherwise. For 1010data, it returns a boolean value indicating whether all values within a given group are true.</td>
</tr>
<tr>
<td>OR( logical_test1, [logical_test2], ...)</td>
<td>g_or(G;S;X)</td>
<td>For Excel, it returns TRUE if any one of a set given conditions is true for a specified value or FALSE none are true. For 1010data, it returns a boolean value indicating whether any value within a given group is true.</td>
</tr>
</tbody>
</table>

**Math and Trig Functions**

<table>
<thead>
<tr>
<th>Excel Function</th>
<th>1010data Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABS(number)</td>
<td>abs(X)</td>
<td>Returns the absolute value of a given value.</td>
</tr>
<tr>
<td>SQRT(number)</td>
<td>sqrt(X)</td>
<td>Returns the square root of the given value.</td>
</tr>
<tr>
<td>POWER(number,power)</td>
<td>exp(X;Y)</td>
<td>Returns the first given value raised to the power of the second given value.</td>
</tr>
<tr>
<td>MROUND(number, multiple)</td>
<td>round(X;Y)</td>
<td>Returns the result of rounding the first given value to the nearest multiple of the second given value.</td>
</tr>
<tr>
<td>Excel Function</td>
<td>1010data Function</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>-------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>FLOOR(number,significance)</td>
<td>floor(X)</td>
<td>Returns the largest integer less than or equal to the given value, expressed as a decimal number.</td>
</tr>
<tr>
<td>CEILING(number,significance)</td>
<td>ceil(X)</td>
<td>Returns the smallest integer greater than or equal to the given value, expressed as a decimal number.</td>
</tr>
<tr>
<td>COS(number)</td>
<td>cos(X)</td>
<td>Returns the cosine of the given value.</td>
</tr>
<tr>
<td>SIN(number)</td>
<td>sin(X)</td>
<td>Returns the sine of the given value.</td>
</tr>
<tr>
<td>TAN(number)</td>
<td>tan(X)</td>
<td>Returns the tangent of the given value.</td>
</tr>
<tr>
<td>SUM(number1, [number2],...)</td>
<td></td>
<td>For Excel, it returns the sum of the specified numbers. For 1010data, it returns the sum of values within a given group.</td>
</tr>
<tr>
<td>PRODUCT(number1, [number2],...)</td>
<td></td>
<td>For Excel, it returns the product of the specified numbers. For 1010data, it returns the product of values within a given group.</td>
</tr>
</tbody>
</table>

**Text Functions**

<table>
<thead>
<tr>
<th>Excel Function</th>
<th>1010data Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TEXT(value,format_text)</td>
<td>string(X)</td>
<td>Returns the input value expressed as a text string.</td>
</tr>
<tr>
<td>VALUE(text)</td>
<td>int(X)</td>
<td>For Excel, it converts the selected string value into a numeric value.</td>
</tr>
<tr>
<td>Excel Function</td>
<td>1010data Function</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------</td>
<td>------------------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>real(X)</td>
<td></td>
<td>For Excel, it converts the selected string value into a numeric value. For 1010data, it returns the given value expressed as a decimal number.</td>
</tr>
<tr>
<td>LOWER(text)</td>
<td>lowercase(X)</td>
<td>Returns a given string converted to all lowercase.</td>
</tr>
<tr>
<td>UPPER(text)</td>
<td>uppercase(X)</td>
<td>Returns a given string converted to all uppercase.</td>
</tr>
<tr>
<td>REPLACE(old_text,start_num,num_chars,new_text)</td>
<td>num_data(replacement_text)</td>
<td>Returns a given string, with a specified number of characters starting at a certain position replaced by a given substring.</td>
</tr>
<tr>
<td>SUBSTITUTE(text,old_text,new_text,[instance_num])</td>
<td>strsubst(X;Y;I;Z)</td>
<td>Returns the given string with a particular occurrence (or all occurrences) of a given substring replaced with a different substring.</td>
</tr>
<tr>
<td>CONCATENATE(text1,[text2],...)</td>
<td>colunion(X;Y)</td>
<td>Returns a string consisting of the union of values from a given set of columns concatenated together using the specified separator, sorted in ascending order.</td>
</tr>
<tr>
<td>LEFT(text,[num_chars])</td>
<td>padright(X;Y)</td>
<td>Returns the given string with a certain number of characters, adding blanks at the end if necessary.</td>
</tr>
<tr>
<td>RIGHT(text,[num_chars])</td>
<td>padleft(X;Y)</td>
<td>Returns the given string with a certain number of characters, adding blanks at the beginning if necessary.</td>
</tr>
<tr>
<td>LEN(text)</td>
<td>strlen(X)</td>
<td>Returns the number of characters in a given string.</td>
</tr>
<tr>
<td>FIND(find_text,within_text,[start_num])</td>
<td>strfind(X;Y;I)</td>
<td>Returns the position within a given string of a particular occurrence of a given substring.</td>
</tr>
</tbody>
</table>
Calculations

Finding duplicates

Sometimes tables contain duplicate entries and it can be necessary to locate and remove these duplicates.

Difficulty

![Difficulty Scale]

Objective

You merged two tables containing similar information, but now there appears to be duplicates in your new table. You would like to locate and remove these duplicates. You know how to do this in Excel by utilizing the *Remove Duplicates* dialog, and you would like to perform the same operations using 1010data.

Excel solution

Excel has multiple ways to locate and remove duplicates. One way is to use the *Remove Duplicates* dialog, located on the *Data* tab in the *Data Tools* group. Using the *Remove Duplicates* dialog, you can determine the duplicates in your table using multiple criteria. For this analogue, duplicates in the table are determined if they share the same county name, state name and 50 percentile 0 bedroom rent. The image below shows the dialog in Excel with these columns selected.

![Remove Duplicates Dialog]

Upon selecting **OK**, all duplicates will be removed and you will be left with the following table.
Although Excel offers a simple tool to remove duplicates, 1010data can process more data at a faster speed with similar ease.

1010data Macro Language solution

```
<base table="pub.doc.gov.area.x2003"/>
<merge table2="pub.doc.gov.county.x2003" />
<tableu label="Tabulation on Worksheet" breaks="countynam,e statename, rent50_0">
  <tcol source="rent50_0" fun="first" label="First Rent"/>
</tableu>

<link table2="pub.doc.gov.area.x2003" col="countynam,e statename, rent50_0"
  col2="countynam,e statename, rent50_0" type="select"> 
  <merge table2="pub.doc.gov.county.x2003"/>
</link>
```

When finding duplicates, you need to establish the columns that uniquely identify each row. For example, a table containing sales information could have rows that can be uniquely identified by the transaction ID and the item SKU. In this recipe, rows are uniquely identified by the columns msa, countynam,e, and rent50_0.

This solution uses a tabulation where the aforementioned columns are used for grouping and the first entry in each group is selected. Then the original merged worksheet is linked in and all of the duplicate rows are no longer present, as can be seen below.

Alternate 1010data Macro Language solution

```
<base table="pub.doc.gov.area.x2003"/>
<merge table2="pub.doc.gov.county.x2003">

<willbe name="test" value="g_cnt(countynam,e statename rent50_0)>1"/>

<sel value="g_first1(countynam,e statename rent50_0;;)"/>
```
Alternatively, you can use \( g\text{\_}cnt(G;S) \) to discover if there are any duplicates present in the table using the unique identifiers described in the previous solution, and then use \( g\text{\_}first1(G;S;O) \) to select only the first instance. This method is particularly useful if your table is segmented. The final table without duplicates can be seen below.

Further reading

If you would like to learn more about the functions and operations discussed in this recipe, click on the links below:

- \( g\text{\_}cnt(G;S) \)
- \( g\text{\_}first1(G;S;O) \)

Calculating the maximum and minimum values

A single tabulation in 1010data can calculate the maximum and minimum values of a table.

Difficulty

Objective

You have a table consisting of daily weather observations and you would like to find the maximum and minimum temperatures for a specific zipcode. You know how to do this in Excel using the \( \text{MAX} \) and \( \text{MIN} \) functions, and you would like to produce the same results using 1010data.

Excel solution

With Excel’s \( \text{MAX} \) and \( \text{MIN} \) functions, you can enter a range of numbers and the functions will output which number in the range is the highest and lowest respectively. These functions can be seen in the equations below.

\[
\text{MAX}(F:F)
\]

\[
\text{MIN}(F:F)
\]

Although Excel has no problem finding the maximum and minimum values for this table, a larger table could produce issues. Excel is limited in the size of data tables it can process, and there also might be
longer processing times with larger tables. 1010data can calculate the same results on larger tables and at faster speeds.

**1010data GUI solution**

After selecting the desired zipcode, for this example, 01001, you can create computed columns that contain the maximum and minimum temperature values.

Within the 1010data menu, click **Columns > Create Computed Column** to open the following dialog.

Within the Value Expression field, you can use the \( g_{hi}(G;S;X) \) and \( g_{lo}(G;S;X) \) to find the max and min temperatures respectively.

Two columns will be added to your table, with each row populated by the max and min temperature.
Alternate 1010data GUI solution

After selecting the desired zipcode, for this example, 01001, you can use 1010data's GUI to perform a tabulation and find the maximum and minimum temperature values.

Within the 1010data menu, click **Analysis > Tabulation** to open the following dialog.

Using the drop-down menus, you can calculate two summaries on the **Mean Temp (F)** column, one for the highest number and one for the lowest number in the column. Selecting these options will create the results shown in the image below.
1010data Macro Language solution

```
<base table="pub.demo.weather.wunderground.observed_daily"/>
<sel value="zipcode='01001'"/>
<willbe name="max_temp" value="g_hi(zipcode;;meantempi)"
label="Max Temperature"/>
<willbe name="min_temp" value="g_lo(zipcode;;meantempi)"
label="Min Temperature"/>
```

When able to, using g_functions is most often the preferred solution. They take advantage of the table's segmentation and therefore are more efficient than tabulations. $g_{hi}(G;S;X)$ and $g_{lo}(G;S;X)$ will find the highest and lowest temperatures in each group. Since there is only one zipcode in the current worksheet, there is only one group, and these functions will just return the values for zipcode 01001.

Alternate 1010data Macro Language solution

The following macro code will produce the same results as using 1010data's GUI in the alternate solution and can be entered in the Edit Actions dialog.

```
<base table="pub.demo.weather.wunderground.observed_daily"/>
<sel value="zipcode='01001'"/>
<tabu label="Max and Min of Mean Temperature">
  <tcol source="meantempi" fun="hi" label="Highest\' Mean Temp\' (F)"/>
  <tcol source="meantempi" fun="lo" label="Lowest\' Mean Temp\' (F)"/>
</tabu>
```

Further reading

If you would like to learn more about the functions and operations discussed in this recipe, click on the links below:

- $g_{hi}(G;S;X)$
- $g_{lo}(G;S;X)$

Returning values based on location

G_functions can return a selection of data based on the location of a specified value, similar to using MATCH and OFFSET in Excel.

Difficulty

![Difficulty Scale]

Objective

You have a data table and you would like to return a collection of values surrounding a specific data point. For example, you have a record of daily snowfall and would like to know the entries for the week prior and
the week after the largest recorded snowfall. You know how to do this in Excel by using the MAX, MATCH, and OFFSET functions, and you would like to know how to perform the same operations using 1010data.

**Excel solution**

The OFFSET function returns a cell or number of cells surrounding a specified position. To obtain all of the arguments for OFFSET, MATCH is used in combination with MAX to find the position of the cell that contains the highest snowfall for the recorded data. Then the following array formula is used to find the information for the surrounding days.

```
=OFFSET(D1,MATCH(MAX(D:D),D:D,0)-4,-2,7,3)
```

Your Excel worksheet should look similar to the image below.

Because the formula is an array formula, you need to select multiple cells in which to enter the formula in, because a single cell can only display one value. This can be difficult when selecting large amounts of data. Using 1010data, a single command can select data of varying size.

**1010data Macro Language solution**

```
<base table="pub.demo.weather.wunderground.observed_daily"/>
<sel value="zipcode=01031"/>
<willem name="test" value="g_hi(zipcode;;snowfalli)"/>
<sel value="between(i_;g_position(zipcode;;;snowfalli;test)-7;
g_position(zipcode;;;snowfalli;test)+7)"/>
<colord cols="zipcode,date,snow,snowfalli"/>
```

In order to select the data that surrounds the data point with highest snowfall, you first need to find the highest snowfall value, using g_hi(G;S;X), and populate a computed column with that value. This computed column is needed because the function g_position(G;S;O;X;Y) compares values in specified columns to find the row containing that value.

**i_** is a system variable that denotes a specific row in the table. Using the function between(X;Y;Z) you can select rows that are contained within a range based on the row position of the max snowfall, plus or minus seven rows. This combination of operations produces the worksheet seen below.
Further reading

If you would like to learn more about the functions and operations discussed in this recipe, click on the links below:

- `g_hi(G;S;X)`
- `g_position(G;S;O;X;Y)`
- `between(X;Y;Z)`

Finding a value corresponding to the nth element

Using the `g_cumcnt` function, you can find corresponding information related to a specific occurrence of a value, similar to using `COUNTIF` and `VLOOKUP` in Excel.

Difficulty

 Objective

You want to find out information for the specific occurrence of a value in your data table. For example, a store has a list of customers and you want to know the customer number of the 5th Aaron on the list. The `VLOOKUP` function in Excel returns the information in a specified column for a given value. However, this function will only return the information for the first instance. By combining the use of `COUNTIF` and `VLOOKUP` you can find the information for a specific occurrence of the value. Using 1010data, you can use `g_cumcnt` and a selection statement to determine the same information.

Excel solution

`VLOOKUP` finds the position of the specified value and returns the corresponding information in a different column. Since this function will only return the information for the first instance of the value, you need to create a new column using `COUNTIF` to distinguish between each occurrence.
This new column is now used in the \texttt{VLOOKUP} function. To determine the customer number of the 5th Aaron in the list you type the following equation in Excel:

\[ \texttt{=VLOOKUP("AARON5",A:I,5,FALSE)} \]

This equation returns 0375421a.

Although this set of operations finds the desired information, it does not do so in an efficient manner. It should be noted that in order to use the \texttt{VLOOKUP} function to find a value, the column that contains the value must be listed first in the column order. Additionally, after executing the \texttt{COUNTIF} formula on the first cell, you must then apply it to each additional row in the table. This can be done with a simple click of the mouse, but will take several minutes to actually complete. The same results can be found using 1010data's platform in a fraction of the time.

\textbf{1010data Macro Language solution}

\begin{verbatim}
<base table="pub.doc.retail.altseg.customers"/>
<willbe name="occurrence" value="splice(firstName,g_cumcnt(firstName;;);'')"/>
<sel value="occurrence='AARON5'"/>
<colord cols="customer"/>
\end{verbatim}

Similar to the Excel method, an additional column needs to be created to distinguish the occurrences of each name. This can be done using \texttt{splice} to combine the first name with it's occurrence in the table, found using \texttt{g_cumcnt(G;S;O)}.

Once this column is created, you can simply select the value in the column that is equal to the first name combined with the desired occurrence. Using \texttt{colord} to only show the \texttt{customer} column gives a similar look to the output of the Excel function.

\textbf{Further reading}

If you would like to learn more about the functions and operations discussed in this recipe, click on the links below:

\texttt{g_cumcnt(G;S;O)}
Finding corresponding data in different tables

You can use a link and select to find information on a specified value contained in a separate table. This operation is similar to using `LOOKUP` and `VLOOKUP` to find corresponding information in a different worksheet.

**Difficulty**

![Difficulty Scale]

**Objective**

You have a table containing the sum of sales for every department in each store. You would like to determine the store with the highest total sales and then find the city in which that store is located. However, information about each store is contained in a separate table. This can be done in Excel by using the `MAX`, `LOOKUP`, and `VLOOKUP` functions, but you would like to know how to perform the same operations in 1010data.

**Excel solution**

The table containing the summary of sales is created by using a pivot table, as can be seen in the Creating a Simple Pivot Table analogue. To find the store that had the highest sum of sales from this table, you first use the `MAX` function to determine which sum is the highest.

\[=\text{MAX}(Q5:Q22)\]

The `LOOKUP` function then uses the results of the `MAX` function to find the store number that is associated with the highest sales.

\[=\text{LOOKUP}(C18,Q5:Q22,A5:A22)\]

`VLOOKUP` uses the resultant store number to find the name of the city where the store is based, from the table located in a separate worksheet.

\[=\text{VLOOKUP}(C19,\text{Sheet1}!A1:O11,10,\text{FALSE})\]

Three functions is all it takes to find the location of the store with the highest sales, however there are some drawbacks to these functions. In order to use the `LOOKUP` function you must have your table sorted in ascending order in relation to total sales and your look-up column must be the last in your table.
Contrary to LOOKUP, the look-up column for VLOOKUP must be the first in the table. If 1010data is used to find the store information, these sorting constraints are not an issue.

**1010data GUI solution**

Once the Sum of Sales tables is complete (see the analogue, *Creating a Simple Pivot Table*), click **Rows > Select Rows (Advanced)** to open the following dialog.

![Select Rows (Advanced) dialog](image)

To find the store with the highest total sales, you need to enter `g_hi(;;t0)`, in the **Expression** field. Then from the same dialog; clicking on **Link** opens the dialog below.

![Link in Another Table dialog](image)

After selecting the desired table, in this case, **Store Master**, you again use the drop-down menus and select the columns on which to link. These columns will be used to combine the two tables on a common data point. Since the previous selection left the table with only one row, when the Store Master table is linked, only the information about the top selling store is selected.

![Store Master table](image)

**1010data Macro Language solution**

The same results can be found by entering the following macro code into the **Edit Actions** dialog.

```
<base table="pub.doc.retail.altseg.sales_detail_transid"/>
<sel value="trans_date=20150114"/>
```
The first part of the code creates a pivot-like table that can be found in the *Creating a Simple Pivot Table* analogue. The selection statement is used to find the highest value in the column t0, which is a computed column that contains the total sales, and select only the row containing that value. By using the `<link>` operation we are able to bring in the information pertaining to the selected store from the *Store Master* table.

**Further reading**

If you would like to learn more about the functions and operations discussed in this recipe, click on the links below:

**Summing every other value**

Using the `mod` function makes it simple to sum every other value both in Excel and 1010data.

**Difficulty**

Excel solution

Using the functions `SUM, IF, MOD, ROW, INDIRECT, and COUNT`, Excel is able to divide the row numbers by 2 in order to determine if that row should be included in the sum, and then sum the selected values using the equation below.

```
{=SUM(IF(MOD(ROW(INDIRECT("1:"&COUNT(F2:F100001))))-1,2)=0,F2:F100001,""))}
```

The worksheet including the solution, can be seen in the image below.
However, Excel cannot process the entire Sales Detail table because it is too large. Therefore, these operations are only done on a subset of the table containing 100,000 rows. 1010data on the other hand can perform these operations on the entire table at a much faster speed.

### 1010data Macro Language solution

```xml
<base table="pub.doc.retail.altseg.sales_detail_transid"/>
<sel value="i_<100001"/>
<willbe name="every_other" value="if(mod(i();2);xsales;0)"/>
<tabu label="Tabulation on Sales Detail">
  <tcol source="every_other" fun="sum" label="Sum of every_other"/>
</tabu>
```

The selection done in the second line of this solution is used in order to produce the same results as the Excel solution, since Excel cannot process the table in its entirety. However, 1010data can perform the same operations on the entire table, without any additional time constraint.

To sum every other row, first a column called every_other, is created using the `<willbe>` operation. Within the value attribute of this column, the function `mod(X;Y)` is used to determine which rows are even and which are odd by returning a 0 if the row number is divisible by 2 and 1 if it is not divisible by 2. `i_` is the system variable for row number, and in this case, the sticky version, `i_()`, is used so that the row numbers don't change as the table changes. Then the `if` function fills the column with the extended sales value if `mod` returns 1, and 0 otherwise.

After this column is created, you can then perform a tabulation to determine the sum of every other row. Within the `<tcol>` operation, the source column is `every_other`, and the function is `fun="sum"`. The final table can be seen in the image below.

### Further reading

If you would like to learn more about the functions and operations discussed in this recipe, click on the links below:

- `mod(X;Y)`
**Summing over multiple conditions**

G functions allow you to calculate multiple sums based on data meeting certain criteria, similar to **SUMIFS** in Excel.

**Difficulty**

![Difficulty Scale]

**Objective**

You would like to sum several different sets of values based on various criteria. However, these sets are contained all in one table and the different values are distributed throughout the table instead of being organized by groups. In Excel you can use the **SUMIFS** function to set conditions for which data you would like to sum. With 1010data, you can solve this issue by using the **if** function in tandem with the **g_sum** function, or you can use the **g_sum** function along with a flag column to denote the values you would like to include in your summation.

**Excel solution**

The **SUMIFS** function in Excel, sums the values in a column based on corresponding columns containing specified values. In this recipe, the **Observed Daily Weather** data is used to sum the total number of rainy, foggy and snowy days based on a specified zip code. For example, if you wanted to calculate these sums for the zip code 01002, you would enter the following formulas in an Excel worksheet.

```
=SUMIFS($C$2:$C$101,$A$2:$A$101,1002,$C$2:$C$101,1)
=SUMIFS($D$2:$D$101,$A$2:$A$101,1002,$D$2:$D$101,1)
=SUMIFS($E$1:$E$100,$A$1:$A$100,1002,$E$1:$E$100,1)
```

The first condition corresponds to the column, or the data, you want summed, while the following conditions depict where to look for the criteria each value must meet and what the criteria is, respectively. With this function, you can have an unlimited number of criteria conditions. The image below shows the data in Excel along with the cells containing the calculated sums.
For each sum you want to compute, if it is based on a different set of criteria, you need to repeat this formula in another cell with the conditions changed. However, with 1010data's platform you can perform several sums each based on different criteria, in the same equation. Additionally, Excel is limited to the amount of data it can process, while 1010data can process larger amounts of data than Excel and at a faster speed.

### 1010data Macro Language solution

```xml
<base table="pub.demo.weather.wunderground.observed_daily"/>
<sel value="year(date)=2015 & day(date)=01"/>
<sel value="i<_101"/>
<colord cols="zipcode,date,fog,rain,snow,sum"/>
<willbe name="sum" value="if(zipcode=01002&rain=1; splice('rain',g_sum(zipcode;;rain);'-');zipcode=01002&fog=1; splice('fog',g_sum(zipcode;;fog);'-');zipcode=01002&snow=1; splice('snow',g_sum(zipcode;;snow);'-');NA)="/>
```

The `if(C1;R1;C2;R2;...;D)` function allows for multiple conditions and also provides an option in the case that none of these conditions are met. The first condition relates what information you want to match, and the second condition describes what to do if this criteria is met.

For this analogue, the transaction sum is calculated if the store is equal to 15 and if the department is 36. You calculate the sum using `g_sum(G;S;X)`, where `G` is the list of columns to denote which values are in the same group, and `X` is the column that you want summed.

Although the `if` function is not the most efficient function, it allows us to calculate multiple sums with one continuous line of code and have all of the results appear in one column.

### Alternate 1010data Macro Language solution

```xml
<base table="pub.demo.weather.wunderground.observed_daily"/>
<sel value="year(date)=2015 & day(date)=01"/>
<sel value="i<_101"/>
<colord cols="zipcode,date,fog,rain,snow,sum"/>
<willbe name="flagr" value="zipcode=01002&rain=1"/>
<willbe name="flagf" value="zipcode=01002&fog=1"/>
<willbe name="flags" value="zipcode=01002&snow=1"/>
<willbe name="rsum" label="Sum of Rainy Days" value="g_sum(zipcode;flagr;rain)="/>
```
Alternatively, you can utilize the S parameter in the g_sum function which denotes a flag, or selection column. This column will contain either a 1 or a 0 to denote whether or not that row of data should be included in the final calculation. You create this reference column(s) using a willbe operation where the value is equal to the criteria. This solution is more efficient but creates six columns instead of just one, as in the previous solution.

Further reading

If you would like to learn more about the functions and operations discussed in this recipe, click on the links below:

- `if(C1;R1;C2;R2;...;D)`
- `g_sum(G;S;X)`
- `splice(X;Y)`

Splitting and concatenating strings

Using string manipulations can convert a column containing names to a column containing initials.

**Difficulty**

- **0**
- **1**
- **2**
- **3**
- **4**
- **5**

**Objective**

You have a column containing the first and last name of a group of people. Because you do not require the full name of each person, you would like to create a column containing just the first and last initial of each person. You know how to do this using the LEFT, RIGHT, and MID functions in Excel, and you would like to achieve the same results using 1010data.

**Excel solution**

The LEFT, RIGHT, and MID functions in Excel extract characters from a specified string based on given inputs. To extract the initials from a first and last name contained in the same column, you need to use a combination of all three functions.

Obtaining the first initial is done by using the LEFT function and selecting only the first character starting on the left hand side of the string. The second initial is more difficult to obtain. First, the FIND function is used to find the position of the space that separates the first and last name, then MID is used to extract only the space and the character that follows the space. Finally, you can use the RIGHT function to choose only the first character, starting on the right end of the string.

```
=LEFT(G2,1) & RIGHT(MID(G2,FIND(" ",G2,1),2),1)
```
While this combination of functions is sufficient in obtaining the initials, 1010data offers a larger range of string manipulation functions to perform similar tasks.

**1010data Macro Language solution**

```
<base table="pub.doc.retail.altseg.stores"/>
<willbe name="first" value="taketo(manager; ' ')"/>
<willbe name="second" value="dropto(manager; ' ')"/>
<willbe name="first_initial" value="first(first; 1)"/>
<willbe name="second_initial" value="first(second; 1)"/>
<willbe name="initials" label="Initials"
  value="splice(first_initial second_initial; '')"/>
<colord hide="first,second,first_initial,second_initial"/>
```

When it is not necessary to retain a complete string within a column, converting to initials can be useful and simplify your data table. In this solution we utilize several string functions to extract the first letter from each name and create a column containing just the initials.

Four columns are created to separate the first and last names, and then to select the leading letter in each name. Then, the `splice` function is used to combine the individual letters and create the "Initials" column. Using `colord` you can hide the columns you created to aid in your transformation.

**Alternate 1010data Macro Language solution**

```
<base table="pub.doc.retail.altseg.stores"/>
<willbe name="initials" label="Initials"
  value="splice(first(manager; 1) first(dropto(manager; ' '); 1); '');"/>
```

The alternate solution uses the same functions but combines them into one line to simplify the code and avoid the need to create unnecessary columns. This solution also avoids using `taketo` by simply using the first letter in the name column.

**Further reading**

If you would like to learn more about the functions and operations discussed in this recipe, click on the links below:

- `taketo(X;Y)`
- `dropto(X;Y)`
- `splice(X;Y)`

**Determining if a data set contains a specified value**

There are various ways to determine if a specific value is contained within a data set, similar to using `IF` and `OR` in Excel.
Objective

You have a data set and you want to determine if the data contains a particular value. For example, you have a table containing sales transaction data and you want to know if a certain item was sold based on the SKU number. You know how to do this in Excel using the `IF` and `OR` functions, and you would like to perform the same operations in 1010data.

Excel solution

There are multiple ways to determine if your data set contains a specified value using Excel. One versatile method uses the functions `IF` and `OR`. The `OR` function serves as the logic test for the `IF` function. It compares a single value to a column or an array of values to determine if there is any match. In this case, you enter a SKU number into the cell Q5 and the following equation returns, `Found`, if it is present in the `Sku` column, and `Not Found`, if it is not.

```excel
=IF(OR(Q5=(E2:E100001)),"Found","Not Found")
```

The image below shows the Sales Detail data imported from 1010data and the results of the above equation.

However, using Excel has its drawbacks. If you try to import the Sales Detail table from 1010data in its entirety, you will receive the following error message: Query results in 4,481,814 rows, only 1,048,576 rows available in worksheet at specified result destination. Using 1010data, you can search for a specific value using the whole table instead of just a subset.

1010data GUI solution

Similar to Excel, there are multiple ways to determine if a data set contains a specific value using 1010data's GUI. Mirroring the solution from Excel, you can populate a cell, or in 1010data's case, a column of cells, with text stating if the value is "Found" or "Not Found." Clicking on Columns > Create Computed Column, will open the following dialog.
You can name the column by populating the **Column Name** and **Column Heading** text fields, and then within the **Value Expression** text field, the formula to determine if the value is present in the table is entered. The \( \text{if}(C1;R1;C2;R2;\ldots;D) \) function is used so that if the SKU is present, the new column is populated with "Found" and otherwise, "Not Found."

**Alternate 1010data GUI solution**

Another solution that is perhaps the simplest and fastest way is to perform a selection. However, this solution will transform the table to display only the data containing the selected value. To open the following dialog, click **Rows > Select Rows** from the 1010data menu.

Using the drop-down menus in the dialog, you select **SKU** and **has the value(s)**. Then, you enter the SKU you would like to search for in the text box. If the SKU is found in the table, a message will appear that displays how many rows contain that value. If the value is not found, you will receive a message that displays **No matches.**, as can be seen in the dialog below.
To obtain a worksheet that looks similar to the Excel solution, using the Macro Language, you can create a column that will be conditionally populated with text. `<willbe>` creates the column and in the `value` attribute, the function `if(C1;R1;C2;R2;...;D)` is used to supply the system with the two options, “Found” and “Not Found” based on the entered SKU.

Alternate 1010data Macro Language solution

A selection statement can be used within the Macro Language to produce the same results as the alternate GUI solution. If the SKU you enter is present in the table, it will select only the rows that contain that SKU. If it is not present, a similar message will appear, stating `No rows selected`.

Further reading

If you would like to learn more about the functions and operations discussed in this recipe, click on the links below:

`if(C1;R1;C2;R2;...;D)`

Filtering and sorting a table

Filtering and sorting the data in a table based on specified criteria can be useful for both completing aggregations and viewing the data.
Objective

You would like to filter your table to only include data with specific qualities. Additionally, you would like to then sort the remaining data by a chosen column. You know how to do this in Excel by using the filter and sort tools, and you would like to produce the same results with 1010data.

Excel solution

Within the Editing portion of Excel's toolbar, there is a Sort & Filter section. Upon highlighting the spreadsheet in its entirety and selecting filter, you then have the option to filter by each column. For this analogue, you want to show data containing department 13 and groups 501 and 196. Therefore, by using the drop-down on the Department and Group columns, you use these selections as the filters. Then to sort, you chose the desired column and select either Sort A to Z or Sort Z to A. This analogue uses the former. The final spreadsheet is shown below.

Although Excel offers an easy solution to this problem, 1010data offers an equally simple solution. Additionally, 1010data can process more data at a faster speed, making it ideal for larger data sets.

1010data GUI solution

1010data's GUI offers a similar feel to the format of Excel. Within the Select Rows dialog, you are able to filter the table based on multiple criteria. For this analogue, you want to select data that contains department 13 and group 501 or 196. The image below shows these options selected from the drop-down menus and entered in the appropriate text fields. Upon clicking Select, the filters will be applied to the table.
As can be seen in the next image, each column in your table contains sort icons. These allow you to sort the table based on the selected column. In this analogue, the table is sorted by item description from A to Z, therefore the **Sort Ascending** icon is selected.

The final table with all filters and sorting applied can be seen below.

### 1010data Macro Language solution

```xml
<base table="pub.doc.retail.altseg.products"/>
<sel value="(dept=13) & (group=196 501)"/>
<sort col="description" dir="up"/>
```

By entering this macro code into the **Edit Actions** dialog, you can achieve the same results as the 1010data GUI solution. The `<sel>` operation allows you to filter the data, and multiple criteria can be
combined by using & or |. Then, the \texttt{<sort>} operation sorts the specified column by the supplied direction, up or down.

**Determining the differences between two data sets**

You can use 1010data to determine the number of differences between two tables, similar to using \texttt{COUNTIF} in Excel.

**Difficulty**

You have two tables and you would like to know how many points differ between the two data sets. You know how to do this in Excel by using \texttt{COUNTIF} and \texttt{SUM}, and you would like to produce the same results using 1010data. For this analogue, the number of differences between the players in the \texttt{Batting} table and the \texttt{Fielding} table are calculated.

**Excel solution**

After importing the two tables with all duplicates removed into Excel, the two columns of player ID’s are placed side by side in the same worksheet. This allows the comparisons to take place. Then the \texttt{COUNTIF} function is used to compare if any single entry is contained in both columns. \texttt{SUM} then adds these results to determine the number of similarities between the tables.

\[
=\text{COUNTIF}($B:$B,$A2) \\
=\text{SUM}(C:C)
\]

Another \texttt{COUNTIF} function is used to count the total number of players.

\[
=\text{COUNTIF}(A:B,"*")-2
\]

Then finally, the number of differences are calculated by subtracting twice the similarities from the total count.

\[
=E2-2*D2
\]

An image of the Excel worksheet can be seen below.
Calculating these results in Excel is not ideal. Pre-adjustments to the tables are required before you can begin work in Excel. In addition, the time to complete the comparison between the two ranges is lengthy. With 1010data, you can complete all parts of the calculation in one place and at a faster speed.

1010data Macro Language solution

```xml
<base table="pub.demo.baseball.batting"/>
<willbe name="first" value="g_first1(lahmanid;;lahmanid)"/>
<sel value="first=1"/>
<colord cols="lahmanid"/>
<merge table2="pub.demo.baseball.fielding" type="intersect">
  <willbe name="first" value="g_first1(lahmanid;;lahmanid)"/>
  <sel value="first=1"/>
  <colord cols="lahmanid"/>
</merge>
<tabu label="Number of Rows">
  <tcol source="lahmanid" fun="cnt" label="Count"/>
</tabu>
<merge table2="pub.demo.baseball.batting">
  <willbe name="first" value="g_first1(lahmanid;;lahmanid)"/>
  <sel value="first=1"/>
  <merge table2="pub.demo.baseball.fielding">
    <willbe name="first" value="g_first1(lahmanid;;lahmanid)"/>
    <sel value="first=1"/>
  </merge>
  <tabu label="Number of Rows">
    <tcol source="lahmanid" fun="cnt" label="Count"/>
  </tabu>
</merge>
</transpose>
<willbe name="differences" value="m1-2*m0"/>
```

With 1010data, tabulations are utilized to determine the number of rows, and therefore the number of players.

In the **Batting** table, a column is created using `<willbe>` to distinguish the first occurrence of every player with the function `g_first1(G;S;O)`. A selection statement then selects only this first occurrence to eliminate duplicates. Since you are only interested in the differences of players, `<colord>` is used to show only the **lahmanid** column.
Then, the **Fielding** table is merged with the **Batting** table, with the same transformations applied. A key step here is using `type="intersect"` with the merge, therefore, only players that appear in both tables will appear in the new merged table. Now that you have a table that contains only the players the two tables have in common, you can count them by performing a tabulation and using `fun="cnt"`.

The total number of players contained in both tables is needed to then determine the number of differences. This is done by again applying the same transformations to each individual table and merging them together. Then the count tabulation is again used to count the total number of players. Additionally, all of this is done inside another merge, so the results can be combined with the results of the previous tabulation.

Due to the nature of the table, a `<transpose/>` operation is needed to allow both results to be visible. Finally, another `<willbe>` column is created where its value is equal to the total number of players less twice the similarities. The similarities need to be doubled to account for their presence in both tables.

Below is an image of your final table.

### Transposed table

For: first=1  
Row 1 of 1

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>m0</td>
<td>5107</td>
<td></td>
</tr>
<tr>
<td>ml</td>
<td>2032</td>
<td></td>
</tr>
<tr>
<td>differences</td>
<td>10,114</td>
<td></td>
</tr>
</tbody>
</table>

### Further reading

If you would like to learn more about the functions and operations discussed in this recipe, click on the links below:

- `g_first1(G;S;O)`
- `<transpose/>`
- `<link>`

### Computing multiple cumulative sums

Cumulative sums can be useful to distinguish individual sums of several different groups.

### Difficulty

![Difficulty scale](image)

### Objective

You have a table consisting of sales transactions and you would like to compute the cumulative sum for each customer's purchase. You know how to do this in Excel using the `SUMIF` function, and you would like to perform the same operation using 1010data.

### Excel solution

In Excel, you can compute a cumulative sum for each desired group using some manipulation of the `SUMIF` function. The below formula will produce a cumulative sum for each customer's purchase.

```excel
=SUMIF(A$2:A2,A2,F$2:F2)
```

And your resulting table should look like the following image.
A simple cumulative sum can be calculated using the **SUM** function, and using the **Subtotal** tool, you can obtain a sales subtotal for each transaction. However, in order to obtain a cumulative subtotal for each transaction, you need to manipulate the **SUMIF** function to compute an individual cumulative sum for each unique transaction number. 1010data's cumulative sum function \( g_{\text{cumsum}}(G;S;O;X) \), allows you to calculate each transaction cumulative sum with one simple expression.

Additionally, with Excel you can only compute the cumulative sums for a subset of the transactions because the sales detail table in its entirety is too large for Excel. With 1010data, you can perform these operations on the entire table at a much faster speed.

### 1010data GUI solution

From the 1010data menu bar click **Columns > Create Computed Column** to open the dialog shown below.

In the text box that corresponds to **Value Expression**, enter the \( g_{\text{cumsum}}(G;S;O;X) \) function. The note section within the dialog relates that `customer` and `transid` must be used for the `G` argument. The `S` and `O` arguments are left blank and `xsales` is used for the `X` argument. Upon clicking **Submit**, the **Sales Detail** table will look like the worksheet seen below.
The same results can be obtained by entering the following macro code into the Edit Actions dialog.

```
<base table="pub.doc.retail.altseg.sales_detail_transid"/>
<willbe name="csum" label="Cumulative Sum" value="g_cumsum(transid;;;xsales)"/>
```

### Visualizations

#### Creating a simple pivot table

Performing a cross tabulation can produce a table that mimics a pivot table in Excel.

### Difficulty

You would like to aggregate a large amount of data and have the results shown over two metrics in a new table. For example, you have a data set consisting of completed sales transactions for a given time period and you would like a table that shows the sum of sales for each individual store and each department within each store. You know how to create a pivot table in Excel and you want to obtain the same results using 1010data.

### Excel solution

Pivot tables in Excel are useful because they allow you to summarize large amounts of data and view it across multiple metrics. Additionally, Excel's user-friendly GUI allows you to easily manipulate the data to properly fit your desired table.

After selecting a cell inside your data set, you can insert a pivot table based on the data associated with that cell.
Your pivot table will be created in a new worksheet, or the existing worksheet, based on your selection. You can then use the GUI to drag which data you want to be displayed as columns and rows, and which data you want to be summarized.

Your completed pivot table should look like the sample below.
Although the process to create a pivot table in Excel is relatively simple, it has its limitations. The original table containing sales transactions is too large for Excel to process, therefore these calculations are only done on a subset of the data.

Instead, if you use 1010data's platform to perform the same operation, there is a much larger limit to the size of your data. Additionally, 1010data also has an easy to use GUI to produce an Excel like pivot table using a cross tabulation.

### 1010data GUI solution

Upon opening the Sales Detail table, click **Analysis > Cross Tabulation** to open the dialog shown below.

![Cross Tabulation Dialog](image)

Here you can see some similarities to Excel's GUI. However, instead of dragging the desired data, you use the drop-down menus, to select the data for the rows and columns of the cross tabulation or "pivot table." You also select the data for which to summarize, and the type of summary you wish to complete. There are additional options you can utilize such as the sort direction and adding a reference column.

For this analogue, **Store** is selected from the first **Rows of Result** drop-down menu and **Department** from **Columns of Result**. You want to compute the sum sales over these metrics, therefore you select **Extended Sales** for the column that contains the summary data and **sum** for the **Type of Summary**. After selecting these inputs and clicking **Submit**, you obtain the table below.

### 1010data Macro Language solution

You can obtain the same result in 1010data's platform by entering the macro code into the **Edit Actions** dialog instead of using the GUI. Below is the macro code that creates the pivot table-like results.

```xml
<table="pub.doc.retail.altseg.sales_detail_transid"/>
<sel value="trans_date=20150114"/>
<tabu label="Sum of Sales Per Department and Store"/>
```
The selection of data is completed in order to effectively mimic the Excel pivot table, which can only compute a subset of the data due to size limitations. However, 1010data can complete the aggregation on the entire data set. The \texttt{<tabu>} operation contains attributes that establish the title of the new table, as well as what data to group the rows and columns by. Within this operation, child elements are contained. \texttt{<break>} specifies the sort direction for a grouping column, and \texttt{<tcol>} specifies the source column to summarize and the summary function that will be used on that column. The resultant table can be seen below.

Further reading

If you would like to learn more about the functions and operations discussed in this recipe, click on the links below:

\texttt{<tabu>}

Creating multidimensional pivot tables

Cross tabulations allow you to view multiple metrics across two dimensions, which is similar to a pivot table in Excel.

Difficulty

![Difficulty level](image)

Objective

You would like to aggregate different types of data across multiple dimensions. For instance, you might want to view summaries for each hour of every day. In this analogue, we use weather data to complete a tabulation of the average temperature and humidity across these dimensions. Another example would be viewing sales and units sold across day and store. You know how to do this by creating a pivot table in Excel, but you want to recreate this operation using 1010data.
Excel solution

Pivot tables in Excel are useful because they allow you to summarize large amounts of data and view it across multiple metrics. Additionally, their user-friendly GUI allows you to easily manipulate the data to properly fit your desired table.

After selecting a cell inside your data set, you can insert a pivot table based on the data associated with that cell.

Your pivot table will be created in a new worksheet, or the existing worksheet, based on your selection. You can then use the GUI to drag which data you want to be displayed as columns and rows, and which data you want to be summarized.
Your completed pivot table should look like the sample below.

Although the process to create a pivot table in Excel is relatively simple, it has its limitations. In order to create the desired table for this example, the data that was loaded into Excel had to have the pre-calculated averages for each date instead of the multiple entries for each data point. Since the original data set consists of 1,841,200,126 rows, it is too large for Excel. Therefore, the daily averages are calculated before loading the data into Excel and the number of rows is reduced to 25,824.

Instead, if you use 1010data’s platform to perform the same operation, there is a much larger limit to the size of your data. The following solution shows how to create a "pivot table" using 1010data.
1010data Macro Language solution

```xml
<base table="pub.demo.weather.wunderground.observed_hourly"/>
<willbe name="hour" value="hour(time)"/>
<willbe name="tempi<>NA"/>
<tabu xtab="0" breaks="date" cbreaks="hour" label="Cross Tabulation">
  <tcol source="tempi" fun="avg"/>
  <tcol source="hum" fun="avg"/>
</tabu>
<sel value="2" expand="1"/>
<willbe name="ii" value="ii_(1)"/>
<willbe name="meas" value="if(ii=1;'Temp';'Hum')" label="Measure"/>
<willbe name="value" value="if(ii=1;t0;t1)"/>
</tabu>
```

Creating a worksheet in 1010data which mimics a pivot table requires some additional thinking beyond simply using the Excel GUI. Multiple cross tabulations are done to produce a table with multi-metric tabs. The first cross tabulation calculates the desired information, both averages for every hour of every day, but does not display it in a way that's easy to view. A row is created for each hour of each day and the averages are in two separate columns in their respective rows.

In order to transform this to a more user friendly view, you first use `expand="1"` to duplicate every row in the table. Then a reference column is created to flag whether it is the first or second entry. Two additional columns are created, one containing 'Temp' if the reference column contains a '1' and 'Hum' otherwise, and the second containing the corresponding average values.

Performing a second cross tabulation where `hour` and the reference column, `meas`, are used as column breaks, you obtain a table that resembles a pivot table.

Below is an image of your resulting worksheet.

![Worksheet Image](image-url)

Further reading

If you would like to learn more about the functions and operations discussed in this recipe, click on the links below:

```xml
<tabu>
expand="1"
```
## Using conditional formatting

Conditional formatting allows you to easily distinguish data in a table or chart.

### Difficulty

![Difficulty](chart.png)

### Objective

You have a table of total sales for 18 different stores and you would like to use conditional formatting to differentiate between the highest and the lowest sales. You know how to use the Conditional Formatting tool in Excel to create a three color scale that is applied to the table, and you would like to obtain the same visualization using 1010data.

### Excel solution

Excel's Conditional Formatting tool has many different options to visualize aspects of your data table. After highlighting the data you wish to format, all that is required is to select from the menu which option you desire. For this analogue, a three color scale where red shows the lowest sales and green shows the highest sales, is chosen. Below is an image of the formatted table.

![Excel Table](excel_table.png)

### 1010data Macro Language solution

```xml
<widget class="grid" type="scroll" width="400" height="600"
label="Sum of Sales Per Store"
base="pub.doc.retail.altseg.sales_detail_transid"
colstwbgcolor="sum_xsales" bgcolorforcol="sum_xsales_color">
  <tabu label="Sum of Sales Per Store" breaks="store">
    <tcol source="xsales" fun="sum" name="sum_xsales"
      label="Sum of Extended Sales"/>
  </tabu>
  <willbe name="sum_xsales_color"
    value="if(between(sum_xsales;0;3000000);"#f7363d";
        between(sum_xsales;3000000;7000000);"#fa532e";
        between(sum_xsales;7000000;9000000);"#fd8535";
        between(sum_xsales;9000000;11000000);"#feb367";
        between(sum_xsales;11000000;13000000);"#fed167";
    ">
```
A similar looking table to the one created in Excel, can be obtained by using 1010data. In order to conditionally format the table, you need to create a Quickapp using the `<dynamic>` operation. Within these tags, a widget with class_="grid" is used as the display for the table. The given tabulation is used to determine the sum of sales per store. In order to apply the desired formatting, two attributes are used within the widget to specify which column is going to be receiving the formatting and which column will supply the colors to be used in the formatted column. These attributes are `colstwtbgcolor` and `bgcolorforcol`, respectively.

The column that supplies the colors is created using the `<willbe>` operation where the value is determined by an `if(C1;R1;C2;R2;...;D)` statement. For each range that you wish to hold a different color, a `between(X;Y;Z)` statement is used for the condition and the desired color is used for the result. These colors are determined by using the color-hex code. In this analogue, the ranges and colors are selected to mimic the solution created in Excel. The completed result can be seen below.

As you can see from the supplied Macro Language for this solution, using 1010data is not exactly easier than using Excel when it comes to conditionally formatting tables. All that is required in Excel is for the user to select an option from a menu. With 1010data, the user must determine the ranges to hold each color and also find the desired color's code, all while immersing these decisions in 9 lines of code. While this process does allow the user more flexibility in their formatting, it can be a much lengthier process.

However, adding conditional formatting to an Excel table can cause the program to run at a much slower speed. While using a table of 19 rows won't slow the operations of Excel, a table with several thousand rows will certainly create lag time. Therefore, although creating a table with Excel is significantly easier, it is not always the better choice, especially when dealing with large tables.

Further reading

If you would like to learn more about the functions and operations discussed in this recipe, click on the links below:

<dynamic>
<widget>
<tabu>
1010data for SQL users

1010data users have backgrounds using different programming languages, databases, and analytical tools. One of the most widely used tools for data is SQL. This section of the 1010data Cookbook was created to help the users with SQL background learn how to complete tasks in 1010data. Each analogue will describe a problem that the user knows how to complete in SQL, and then show how to solve the problem in 1010data.

Introduction

SQL and 1010data store their data in very different ways, therefore the way you treat your data in each language needs to be different as well. SQL is a relational database, and the data is stored in rows, allowing for data to be easily added and removed. Contrastingly, 1010data is an analysis-first database, and the data is stored in columns, allowing for various analyses to be executed over large amounts of data, more efficiently in many cases.

When looking at each analogue, you might notice that there are multiple 1010data solutions for one SQL solution. Each analogue attempts to illustrate a direct translation from SQL to 1010data, however sometimes this direct translation might not be the most efficient solution for 1010data's platform. Therefore, the most efficient solution will be listed as the main solution, while the alternate solutions shown contain a different approach to solve the problem. It is also important to note that each SQL solution is created using MySQL, and therefore there might be alternative ways to solve the problem using different functions and operations that are not available in MySQL.

When using SQL, you need to think about what you want your end result to be, and which steps will get you there. Most often, you will have to create your query in such a way that all commands are nested within each other, using a series of JOIN's and/or UNION's, or you will need to create intermediate tables containing your results at each point in your query in order to use these results in the next step. You cannot simply enter in commands as you progress, building on the query as you go. And if you decide to make a change in your query at any point in time, you need to resubmit your whole query from the beginning with the change added. With 1010data, you're able to build on your query as you go. This is because 1010data preserves the state of your session and stores the results of your queries in your cache. Also, this eliminates the need to materialize intermediate tables to use in your query.

Workflow

As mentioned in the introduction, the way you think about your data while using SQL and 1010data needs to be different, the same goes for the way you think about your work flow. Let's look at a simple problem and discuss how you would go about solving the problem in both SQL and 1010data.

The Problem

You are in charge of engaging customers at a grocery store and you need to find out some information on the customers that visit your store. First, you want to know when was the first time each customer shopped at your store and how many days are in between each customer's first visit and their next visit. You also want to know how much they spend on each visit.

SQL Work Flow

Before beginning work, you need to think of what you want your end result to be and how you are going to get there. For this problem, you know that you want to end up with one entry for each customer, with one column displaying the date of their first visit to the store and then another displaying the next sequential visit. Each of these entries should also contain a summary of the amount the customer spent during each visit, and there should be a column to communicate how many days have passed since the first visit.

Since you want to compute a summary of sales for each customer by visit date, your first calculation should compute a sum of sales for each customer and each date. Your code might look something like the following:
From these results, you need to find the earliest date for each customer, as well as the next sequential date. However, you can’t simply enter a new query that uses these results unless you created a new table to store them. Instead, you need to combine your existing query with additional queries in order to obtain the next round of information. In this particular case, you need to combine two additional queries to obtain both the first and second date.

```
SELECT customer,MIN(first_date),sum_sales
FROM (SELECT customer,sum_sales,MIN(date) AS first_date
      FROM (SELECT customer,date,SUM(xsales) AS sum_sales
            FROM sales_item_detail
            GROUP BY customer,date)
      GROUP BY customer)
LEFT JOIN (SELECT customer,MIN(date) AS second_date,SUM(xsales) AS sum_sales
            FROM sales_item_detail
            GROUP BY customer,date)
ON first_results.customer = sum_results2.customer
WHERE first_results.first_date <> sum_results2.second_date
GROUP BY customer;
```

Then, you can join these results to the results from an additional query where you can find the next sequential date that each customer shopped at the store. However, if a customer did not shop at the store a second time, that customer’s record is not included in the resultant table.

After you have the dates of the first and second visits of each customer, you need to add an additional column to your previous query that determines the number of days between these visits.

```
SELECT first_results.customer,MIN(first_date),first_results.sum_sales,
      MIN(second_date),sum_results2.sum_sales,DATEDIFF(second_date,first_date) AS days_between
FROM (SELECT customer,sum_sales,MIN(date) AS first_date
      FROM (SELECT customer,date,SUM(xsales) AS sum_sales
            FROM sales_item_detail
            GROUP BY customer,date)
      GROUP BY customer)
LEFT JOIN (SELECT customer,MIN(date) AS second_date,SUM(xsales) AS sum_sales
            FROM sales_item_detail
            GROUP BY customer,date)
ON first_results.customer = sum_results2.customer
WHERE first_results.first_date <> sum_results2.second_date
GROUP BY customer;
```
FROM sales_item_detail
GROUP BY customer, date
) AS sum_results2
ON first_results.customer = sum_results2.customer
WHERE first_results.first_date <> sum_results2.second_date
GROUP BY customer;

Your resultant table should look like the following image.

![Resultant Table Image]

As you can tell by this example, you not only need to know how you would like your results to look, but you also need to determine how to connect all of the different parts of your query in order to obtain this result.

**1010data Work Flow**

The work flow for solving the same problem in 1010data is very different. First, to obtain the sum of sales for each customer and each date, you can perform a tabulation with customer and date used as the break columns.

```
<tabu label="Sum of Sales per Customer and Date" breaks="customer,trans_date">
  <tcol source="xsales" fun="sum" name="sum_sales" label="Sum of Extended Sales"/>
</tabu>
```

You can determine the first date that a customer shopped at the store with a simple select statement, but before that is done, you want to find the next sequential date that the customer shopped after their first visit.

```
<willbe name="next_date" label="Date of Next Visit" value="g_rshift(customer;;trans_date;trans_date;1)"/>
```

Using the same logic, you can create a column that contains the sales for this second visit.

```
<willbe name="next_date_sales" label="Sales of Next Visit" value="g_rshift(customer;;trans_date;sum_sales;1)"/>
```

And an additional column that calculates the days in between these visits.

```
<willbe name="days_between" label="Days in Between Visits" value="days(trans_date;next_date)"/>
```

Lastly, you just need to select the rows where the Date column contains the customer’s first visit.

```
<sel value="g_first1(customer;;trans_date)"/>
```

This table contains records for all customers, those that did not shop at the store a second time just have NA values in the respective columns.
With this particular example, the same order of steps was not followed using SQL and 1010data. In the SQL work flow, the first date each customer shopped at the store was determined second, after the sum of sales was calculated. However, in 1010data, the first date was not selected till the very end. If you had selected only the first date for each customer at the beginning using 1010data, you would have had to follow a similar approach to SQL- linking tables. Waiting to perform this selection allows you to gather all other necessary information first and eliminates the need to link back in the same information.

When using 1010data, each time you add on to your query, you don't have to rerun the previous steps. Additionally, you don't have to nest and join your queries together in order to produce your desired end result. Each new step uses the results of the previous steps as it's base without needing to create a temporary table to store the results. It is important to consider this work flow while constructing your 1010data queries, both in order to take advantage of the system and to create efficient queries.

**Data Types**

The data types offered in both SQL and 1010data differ. It is important to know how these data types should be converted in between systems.

**Table 1: Data Type Equivalents**

<table>
<thead>
<tr>
<th>SQL Data Type</th>
<th>1010data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Integers</td>
<td>Integer (i)</td>
</tr>
<tr>
<td>Byteints</td>
<td></td>
</tr>
<tr>
<td>Smallints</td>
<td></td>
</tr>
<tr>
<td>Booleans</td>
<td></td>
</tr>
<tr>
<td>Dates</td>
<td></td>
</tr>
<tr>
<td>Times</td>
<td></td>
</tr>
<tr>
<td>Timestamps (date + time)</td>
<td>Decimals /Floats (f)</td>
</tr>
</tbody>
</table>
### SQL Data Type

<table>
<thead>
<tr>
<th>SQL Data Type</th>
<th>1010data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bigints (where the max absolute value is less than 8,000,000,000,000)</td>
<td></td>
</tr>
<tr>
<td>Numerics</td>
<td>String/Alphanumeric (a)</td>
</tr>
<tr>
<td>Bigints (where the max absolute value is greater than 8,000,000,000,000)</td>
<td></td>
</tr>
<tr>
<td>Char</td>
<td></td>
</tr>
<tr>
<td>Varchar</td>
<td></td>
</tr>
</tbody>
</table>

### Function comparisons

**Date and Time Functions**

<table>
<thead>
<tr>
<th>SQL Function</th>
<th>1010data Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NOW()</td>
<td>@now_</td>
<td>Returns the current date and time.</td>
</tr>
<tr>
<td>CURDATE()</td>
<td>@today_</td>
<td>Returns the current date.</td>
</tr>
<tr>
<td>DATE(date)</td>
<td>date(X)</td>
<td>Returns the date portion of a date +time value.</td>
</tr>
<tr>
<td>year(X)</td>
<td></td>
<td>Returns the year portion of a date-related value.</td>
</tr>
<tr>
<td>month(X)</td>
<td></td>
<td>Returns the month portion of a date-related value.</td>
</tr>
<tr>
<td>day(X)</td>
<td></td>
<td>Returns the day portion of a date-related value.</td>
</tr>
<tr>
<td>quarter(X)</td>
<td></td>
<td>Returns the quarter corresponding to a date-related value.</td>
</tr>
<tr>
<td>yrmo(X)</td>
<td></td>
<td>Returns the month form of a date-related value.</td>
</tr>
<tr>
<td>yrqtr(X)</td>
<td></td>
<td>Returns the quarter form of a date-related value.</td>
</tr>
<tr>
<td>hour(X)</td>
<td></td>
<td>Returns the hour portion of a time-related value.</td>
</tr>
<tr>
<td>minute(X)</td>
<td></td>
<td>Returns the minutes portion of a time-related value.</td>
</tr>
<tr>
<td>second(X)</td>
<td></td>
<td>Returns the seconds portion of a time-related value.</td>
</tr>
<tr>
<td>seconds(X)</td>
<td></td>
<td>Returns the number of seconds from midnight to a time-related value.</td>
</tr>
<tr>
<td>EXTRACT(unit FROM date)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DATE_ADD(date, INTERVAL expr type)</td>
<td>shift(X;Y)</td>
<td>Returns the date-related value shifted by the amount specified.</td>
</tr>
</tbody>
</table>
### SQL Function | 1010data Function | Description
---|---|---
| | shiftmonths(X;Y;E) | Returns the date-related value shifted by the number of months specified.
| DATEDIFF(date1,date2) | days(X;Y) | Returns the number of days between two date-related values.
| DAYOFWEEK(date) | dayofwk(X) | Returns the integer value corresponding to the day of the week of a date-related value.

### Mathematical Functions

<table>
<thead>
<tr>
<th>SQL Function</th>
<th>1010data Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AVG()</td>
<td>g_avg(G;S;X)</td>
<td>For SQL, the function returns the average value. For 1010data, the average of the values within a given group is returned.</td>
</tr>
<tr>
<td>ABS()</td>
<td>abs(x)</td>
<td>Returns the absolute value of a given value.</td>
</tr>
<tr>
<td>ACOS()</td>
<td>acos(X)</td>
<td>Returns the arccosine (in radians) of the given value.</td>
</tr>
<tr>
<td>ASIN()</td>
<td>asin(X)</td>
<td>Returns the arcsine (in radians) of the given value.</td>
</tr>
<tr>
<td>ATAN()</td>
<td>atan(X)</td>
<td>Returns the arctangent (in radians) of the given value.</td>
</tr>
<tr>
<td>BIT_AND()</td>
<td>bitand(X;Y)</td>
<td>Returns the bitwise AND of two 32-bit integers.</td>
</tr>
<tr>
<td>BIT_OR()</td>
<td>bitor(X;Y)</td>
<td>Returns the bitwise OR of two 32-bit integers.</td>
</tr>
<tr>
<td>BIT_XOR()</td>
<td>bitxor(X;Y)</td>
<td>Returns the bitwise XOR (exclusive-OR) of two 32-bit integers.</td>
</tr>
<tr>
<td>CAST(expr AS type)</td>
<td>int(X)</td>
<td>Returns the largest integer less than or equal to the given value, expressed as an integer number.</td>
</tr>
<tr>
<td></td>
<td>real(X)</td>
<td>Returns the given value expressed as a decimal number.</td>
</tr>
<tr>
<td>CEIL()</td>
<td>ceil(X)</td>
<td>Returns the smallest integer greater than or equal to the given value, expressed as a decimal number.</td>
</tr>
<tr>
<td>COS()</td>
<td>cos(X)</td>
<td>Returns the cosine of the given value.</td>
</tr>
</tbody>
</table>
| COUNT() | g_cnt(G;S) | For SQL, the function returns the number of rows. For 1010data
<table>
<thead>
<tr>
<th>SQL Function</th>
<th>1010data Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>COUNT(DISTINCT expr)</td>
<td>g_ucnt(G;S;X)</td>
<td>Returns the number of distinct values within a given group.</td>
</tr>
<tr>
<td>FIRST()</td>
<td>g_first(G;S;O;X)</td>
<td>For SQL, the function returns the first value in the designated column. For 1010data, the first non-N/A value within a given group is returned.</td>
</tr>
<tr>
<td></td>
<td>g_first1(G;S;O)</td>
<td>For SQL, the function returns the first value in the designated column. For 1010data, a boolean value indicating whether this is the first row in a given group is returned.</td>
</tr>
<tr>
<td>LAST()</td>
<td>g_last(G;S;O;X)</td>
<td>For SQL, the function returns the last value in the designated column. For 1010data, the last non-N/A value within a given group is returned.</td>
</tr>
<tr>
<td></td>
<td>g_last1(G;S;O)</td>
<td>For SQL, the function returns the last value in the designated column. For 1010data, a boolean value indicating whether this is the last row in a given group is returned.</td>
</tr>
<tr>
<td>LOG()</td>
<td>loge(X)</td>
<td>Returns the natural logarithm of the given value.</td>
</tr>
<tr>
<td>LOG10()</td>
<td>log(X;Y)</td>
<td>For SQL, the base-10 logarithm is returned. For 1010data, the logarithm of the first given value to the base of the second given value is returned.</td>
</tr>
<tr>
<td>LOG2()</td>
<td>loge(X)</td>
<td>Returns the natural logarithm of the given value.</td>
</tr>
<tr>
<td>MAX()</td>
<td>max(X;Y)</td>
<td>For SQL, the function returns the largest value in the designated column. For 1010data, the larger of two given values is returned.</td>
</tr>
<tr>
<td></td>
<td>g_hi(G;S;X)</td>
<td>For SQL, the function returns the largest value in the designated column. For 1010data, the highest value within a given group is returned.</td>
</tr>
<tr>
<td>SQL Function</td>
<td>1010data Function</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>MIN()</td>
<td>min(X;Y)</td>
<td>For SQL, the function returns the smallest value in the designated column. For 1010data, the smaller of two given values is returned.</td>
</tr>
<tr>
<td>MIN()</td>
<td>g_lo(G;S;X)</td>
<td>For SQL, the function returns the smallest value in the designated column. For 1010data, the lowest value within a given group is returned.</td>
</tr>
<tr>
<td>MOD()</td>
<td>mod(X;Y)</td>
<td>Returns the modulo of two given values.</td>
</tr>
<tr>
<td>RAND()</td>
<td>draw(X;Y)</td>
<td>For SQL, returns a random floating point value. For 1010data, using X as a seed, random numbers are drawn between 0 and Y-1. ( _ should be used when selects and links with expansions are sued)</td>
</tr>
<tr>
<td>STD()</td>
<td>g_std(G;S;X)</td>
<td>For SQL, the function returns the standard deviation of the population. For 1010data, the population standard deviation of values within a given group is returned.</td>
</tr>
<tr>
<td>STD()</td>
<td>r_std(C;S)</td>
<td>For SQL, the function returns the standard deviation of the population. For 1010data, the standard deviation of values across a given set of columns is returned.</td>
</tr>
<tr>
<td>SUM()</td>
<td>g_sum(G;S;X)</td>
<td>For SQL, the function returns the sum of the values within a column. For 1010data, the sum of values within a given group is returned.</td>
</tr>
<tr>
<td>VARIANCE()</td>
<td>g_var(G;S;X)</td>
<td>For SQL, the function returns the variance a population. For 1010data, the variance of values within a given group is returned.</td>
</tr>
</tbody>
</table>

**String Functions**

<table>
<thead>
<tr>
<th>SQL Function</th>
<th>1010data Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CAST(expr AS type)</td>
<td>string(X)</td>
<td>Returns the input value expressed as a text string.</td>
</tr>
<tr>
<td>CHAR_LENGTH(str)</td>
<td>strlen(X)</td>
<td>Returns the number of characters in a given string.</td>
</tr>
<tr>
<td>SQL Function</td>
<td>1010data Function</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------</td>
<td>-------------------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>CONCAT_WS()</td>
<td>splice(X;Y)</td>
<td>Returns a string consisting of the list of given values concatenated together using the specified separator.</td>
</tr>
<tr>
<td>colunion(X;Y)</td>
<td></td>
<td>Returns a string consisting of the union of values from a given set of columns concatenated together using the specified separator, sorted in ascending order.</td>
</tr>
<tr>
<td>INSERT(str, pos, len, newstr)</td>
<td>stredit(X;P;N;Y)</td>
<td>Returns a given string, with a specified number of characters starting at a certain position replaced by a given substring.</td>
</tr>
<tr>
<td>INSTR(str, subst)</td>
<td></td>
<td>For SQL, the position of the first occurrence of the substring in the given string is returned. For 1010data, the position within a given string of a particular occurrence of a given substring is returned.</td>
</tr>
<tr>
<td>LOCATE(substr, str)</td>
<td>strfind(X;Y;I)</td>
<td>Returns the specified number of characters, starting at a certain position, extracted from a given string.</td>
</tr>
<tr>
<td>LEFT(str, len)</td>
<td>first(X;Y)</td>
<td>Returns a string consisting of the first number of characters of a given string.</td>
</tr>
<tr>
<td>LOWER(str)</td>
<td>strdowncase(X)</td>
<td>Returns the given string with all uppercase characters converted to lowercase.</td>
</tr>
<tr>
<td></td>
<td>lowercase(X)</td>
<td>Returns a given string converted to all lowercase.</td>
</tr>
<tr>
<td>MID(str, pos, len)</td>
<td>strextract(X;P;N)</td>
<td>Returns the specified number of characters, starting at a certain position, extracted from a given string.</td>
</tr>
<tr>
<td>SUBSTRING(str, pos, len)</td>
<td></td>
<td>Return the given string repeated a certain number of times.</td>
</tr>
<tr>
<td>REPLACE(str, from_str, to_str)</td>
<td>strrepeat(X;N)</td>
<td>For SQL, all occurrences of the substring, from_str, within the specified string, are replaced with to_str. For 1010data, the given string with a particular occurrence (or all occurrences) of a given substring replaced with a different substring is returned.</td>
</tr>
<tr>
<td></td>
<td>strsubstit(X;Y;I;Z)</td>
<td>For SQL, all occurrences of the substring, from_str, within the specified string, are replaced with to_str. For 1010data, the given string with substrings replaced</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SQL Function</th>
<th>1010data Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>repstr(X;L)</td>
<td>For SQL, all occurrences of the substring, from_str, within the specified string, are replaced with to_str. For 1010data, the given string with substrings replaced</td>
</tr>
</tbody>
</table>
### SQL Function | 1010data Function | Description
--- | --- | ---
| | | based on a given list of string pairs is returned.
| RIGHT(str,len) | last(X;Y) | Returns a string consisting of the last number of characters of a given string.
| TRIM([BOOTH|LEADING|TRAILING] remstr FROM str) | strtrim(X;Y;C;D) | Returns the given string with certain characters trimmed from either or both ends.
| UPPERCASE() | strupcase(X) | Returns the given string with all lowercase characters converted to uppercase.
| | uppercase(X) | Returns a given string converted to all uppercase.

#### Logic Functions

<table>
<thead>
<tr>
<th>SQL Function</th>
<th>1010data Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IF(expr1,expr2,expr3)</td>
<td>if(C1;R1;C2;R2;...;D)</td>
<td>For SQL, if expr1 is true, then expr2 is returned, otherwise expr3 is returned. For 1010data, the value corresponding to the first condition that evaluates to true is returned; if no conditions are true, the specified default value is returned.</td>
</tr>
<tr>
<td>case(X;V1;R1;V2;R2;...;D)</td>
<td>For SQL, if expr1 is true, then expr2 is returned, otherwise expr3 is returned. For 1010data, the value corresponding to the case that a given value matches is returned; if no cases match, the specified default value is returned.</td>
<td></td>
</tr>
<tr>
<td>IFNULL(expr1,expr2)</td>
<td>ifnull(X;Y)</td>
<td>For SQL, if expr1 is NULL, then expr2 is returned, otherwise expr1 is returned. For 1010data, a specified value is returned when the value in a given column is N/A.</td>
</tr>
<tr>
<td>NULLIF(expr1,expr2)</td>
<td>nullif(X;Y)</td>
<td>For SQL, if expr1 equals expr2, NULL is returned, otherwise expr1 is returned. For 1010data, if the two given values match, N/A is returned.</td>
</tr>
</tbody>
</table>

### Counting occurrences

Based on specified groups in your data, you can determine how many times pairs of values occur together.
Objective
You want to know how many times two different products were purchased in the same transaction. You
know how to do this in SQL by using SELECT and JOIN, and you would like to produce the same results
with 1010data.

SQL solution

```sql
SELECT COUNT(*)
FROM (SELECT DISTINCT sku,transid FROM sales_item_detail WHERE sku=406444) AS t1
JOIN (SELECT DISTINCT sku,transid FROM sales_item_detail WHERE sku=407358) AS t2
ON t1.transid=t2.transid;
```

To find the number of transactions that contain both items, you need to perform two different selection
statements to get the transaction ID's where each SKU is bought, and join them together on matching ID's.
Then, you count the total number of rows to obtain 691 transactions in which both items were bought.

Using 1010data, you can calculate the number of transactions without having to join together multiple
results. Additionally, completing this calculation in 1010data is much faster than SQL, especially for larger
data sets and more complicated queries.

1010data Macro Language solution

```xml
<base table="pub.doc.retail.altseg.sales_detail_transid"/>
<willbe name="first_sku" value="sku=406444"/>
<willbe name="second_sku" value="sku=407358"/>
<sel value="g_or(transid;;first_sku)&g_or(transid;;second_sku)"/>
<tabu label="Tabulation on Sales Detail">
  <tcol source="transid" fun="ucnt" name="num_trans" label="Number of Transactions"/>
</tabu>
```

To determine if any one transaction holds either of the desired items, two flag columns are created, called
first_sku and second_sku. These columns hold either a 1, if the specified SKU is present, or a 0, if
the SKU is not present. The function \( g_{or}(G;S;X) \), uses these columns to determine if each transaction
contained at least one of these items. Combining two of these functions in a single select statement allows
you determine if both items were bought in a single transaction. G_functions are unique to 1010data, and
they allow you to perform calculations much faster because they utilize the unique structure of 1010data
which recognizes order. Finally, a tabulation is used to count the number of unique transactions ID's in
order to determine how many transactions contained both items.

Alternate 1010data Macro Language solution

```xml
<base table="pub.doc.retail.altseg.sales_detail_transid"/>
<sel value="g_first1(transid sku;;)"/>
<willbe name="sku_present" value="sku=406444 407358"/>
<sel value="sku_present=1"/>
<tabu label="Tabulation on Sales Detail" breaks="transid">
  <tcol source="sku_present" fun="cnt" name="sum_skus" label="Sum of SKU’s Present"/>
</tabu>
```
There is also a way to produce the same results in 1010data without using g functions. Again, a flag column is created, called sku_present, which holds a 1 if the SKU is present and a 0 otherwise. By performing a tabulation that sums the flag column for each transaction, you can determine if both items were bought together if the sum is 2 or greater. Selecting only these rows and completing a count tabulation results in the number of transactions that contain both specified items.

**Further reading**

If you would like to learn more about the functions and operations discussed in this recipe, click on the links below:

- `g_or(G;S;X)`

**Counting based on conditions**

You can determine how many times a specified value occurs in a table based on predetermined and conditional requirements.

**Difficulty**

![Difficulty Scale](image)

**Objective**

You want to determine how many customers bought a specific item during a promotion and then repurchased that item at a later date. You know how to do this in SQL using `SELECT`, `DISTINCT`, and `JOIN`, and you would like to produce the same results using 1010data.

**SQL solution**

```sql
SELECT DISTINCT bght_on_pr.customer FROM
  (SELECT customer,transid,sku FROM sales_item_detail WHERE date=20150105 AND sku=406444) AS bght_on_pr
JOIN
  (SELECT customer,transid,sku FROM sales_item_detail WHERE date>20150105 AND sku=406444) AS bght_after
ON bght_on_pr.customer=bght_after.customer;
```

Using two different `SELECT` statements combined with an `AS`, you can retrieve temporary store results from the `Sales Detail` table, in two temporary tables. One table contains customers that purchased the item on the promotion date, and the other contains customers that purchased the item after the promotion date. Using `JOIN` and matching up customer ID's will produce a list of customers that bought the item both on the promotion and again afterwards. However, you must use `DISTINCT` to obtain a list of unique customer ID's.

With 1010data, you can obtain the same results without the use of temporary tables.
1010data Macro Language solution

```xml
<base table="pub.doc.retail.altseg.sales_detail_transid"/>
<sel value="(sku=406444) & (trans_date>=20140105)"/>
<willbe name="bght_on_pr" value="trans_date=20150105"/>
<willbe name="bght_after" value="trans_date>20150105"/>
<merge/>
<sel value="g_or(customer;;bght_on_pr) & g_or(customer;;bght_after)"/>
<sel value="g_first1(customer;;)"/>
```

With 1010data, you can use g_functions to produce the same results. This is the preferred method because g_functions take advantage of how the data is stored. However, your table needs to be properly segmented in order to use g_functions. The Sales Item Detail table is segmented by customer and transaction ID. In order to determine the number of customers that bought the item both on the promotion and afterwards, the table would need to be only segmented by customer.

In order to work around this restriction you perform an empty merge, `<merge/>` on the table. This will condense the table into a single segment, and then you can use the necessary groupings within each g_function. However, you should only use this method on tables smaller than ten million rows, or the system will have difficulty completing the query.

Two g_or(G;S;X) functions are used to determine if any customer bought the item on the promotion and then bought it again afterwards. g_first1(G;S;O) then selects only the first instance of every customer to avoid duplicates and to obtain a list of customers that meet your requirements.

Alternate 1010data Macro Language solution

```xml
<base table="pub.doc.retail.altseg.sales_detail_transid"/>
<sel value="(sku=406444) & (trans_date>=20140105)"/>
<willbe name="bght_on_pr" value="trans_date=20150105"/>
<willbe name="bght_after" value="trans_date>20150105"/>
<tabu label="Tabulation on Sales Detail" breaks="customer">
    <tcol source="bght_on_pr" fun="sum" name="sum_on_pr" label="Bought on Promotion"/>
    <tcol source="bght_after" fun="sum" name="sum_after_pr" label="Bought After Promotion"/>
</tabu>
<sel value="(sum_on_pr<>0) & (sum_after_pr<>0)"/>
```

If your table is not properly segmented, and therefore you are unable to use g_functions, there is another approach which instead performs a tabulation. First, the data is limited to the transactions that occurred on or after the promotion date and include the specified item, in this example the promotion occurred on January 5th, 2015 and the SKU of the item is 406444. Two flag columns are created, one to determine if the customer bought the item on the date of the promotion and one to determine if the customer purchased the same item again afterwards.

Performing a tabulation which breaks on customer, will group together the same customers and display how many times they bought the item on the promotion date and after the promotion date. Finally, you select only the customers that bought the item both on the promotion date and at least one time afterwards.

**Further reading**

If you would like to learn more about the functions and operations discussed in this recipe, click on the links below:

- `g_or(G;S;X)`
- `g_first1(G;S;O)`
- `<tabu>`
Calculating group specific averages

Instead of calculating one overall average, you can calculate smaller averages based on criteria specific groups.

Difficulty

| 0 | 1 | 2 | 3 | 4 | 5 |

Objective

You would like to calculate the median price for each product sold by a store during a specified time period. You know how to do this in SQL by using the `AVG` and `GROUP BY` commands, and you would like to produce the same results using 1010data.

SQL solution

In SQL, you can execute multiple operations in a single line of code. The following line calculates the average weekly sales for each SKU sold in each store.

```
SELECT sku, store, week(date), AVG(xsales)
FROM sales_item_detail
GROUP BY sku, store, WEEK(date);
```

Although it is relatively simple to complete this calculation, SQL has certain drawbacks. For example, you can't easily save query results and you can't build on you queries without nesting `SELECT` statements.

1010data Macro Language solution

```
<base table="pub.doc.retail.altseg.sales_detail_transid"/>
<link table2="pub.doc.retail.calendar" col="trans_date" col2="trans_date"/>
<tabu label="Average Weekly Sales" breaks="sku,store,fw,fy">
  <break col="sku" sort="up"/>
  <tcol source="xsales" fun="avg" name="avg_sales"
  label="Average Weekly Sales"/>
</tabu>
```

For this example, the `Date` column from each table is used to link the sales detail table and the fiscal calendar, so you can correctly add the fiscal week to each transaction.

The `Average Weekly Sales` tabulation uses the columns `Sku`, `Store`, `Fiscal Week`, and `Fiscal Year` as break columns. This is similar to the columns given to `GROUP BY` in SQL. It is important to include both the fiscal week and fiscal year because if a table contains multiple years, just grouping by fiscal week can improperly group values. The `<tcol>` operation creates a column containing the average extended sales for each specified group.

For the break column `sku`, an additional operation, `<break>` is used to sort the column in order to mimic the results in SQL and illustrate that they are indeed the same.

Further reading

If you would like to learn more about the functions and operations discussed in this recipe, click on the links below:

```
<tabu>
<link>
```
Selecting based on rank

You can select values based on certain criteria, for example if their rank puts them in a certain range.

**Objective**

You want to know which ten stores sold the most of a specific product during a given time period. You know how to do this in SQL by using commands and functions such as `SUM`, `WHERE`, and `GROUP BY`, and you would like to know how to obtain the same results with 1010data.

**SQL solution**

To find the ten stores that had the highest sales of item 406444 during 2015, you would execute the following query.

```sql
SELECT store, sku, ROUND(SUM(qty)) AS `Sum of Sales`
FROM sales_item_detail
WHERE sku=406444 AND YEAR(date)=2015
GROUP BY store
ORDER BY SUM(qty) DESC
LIMIT 10;
```

Using `SELECT` you can select the store number, SKU, and the sum of sales from the `Sales Item Detail` table. Additionally, you can select only the rows that contain the specified SKU and that occurred during the given time period. Grouping by the store and ordering by the sum of sales, you can obtain a list of the stores and their sales, ranked by the highest sum of sales. Using `LIMIT`, only the top ten stores are shown in the result.

With 1010data, these same results can be produced and additionally the results can be easily saved to a table or used in future queries. To use these results for additional analysis in SQL, you would have to nest this select statement within another query.

**1010data Macro Language solution**

```
<base table="pub.doc.retail.altseg.sales_detail_transid"/>
<sel value="year(trans_date)=2015&sku=406444"/>
<tabu label="Sum of Product Sales" breaks="store,sku">
  <tcol source="xsales" fun="sum" name="sum_xsales"
    label="Sum of Extended Sales"/>
</tabu>
<sel value="g_rank(;;;sum_xsales)<=10"/>
<sort col="sum_xsales" dir="down"/>
```

This analogue finds the ten stores that had the highest sales of the product with SKU 406444 during the year 2015. Therefore, you first use a selection statement to obtain only the transactions made in 2015 for SKU 4016444. A tabulation is then done to obtain the sum of sales for each store.

After the sum for each store is calculated, you can use the `g_rank(G;S;O;X)` to select the top ten stores with the highest sales. The `<sort>` operation is then used to mimic the results produced by SQL and order the stores by highest to lowest sales.

**Alternate 1010data Macro Language solution**

```
<base table="pub.doc.retail.altseg.sales_detail_transid"/>
```
Alternatively, you can sort the Sum of Extended Sales column in descending order after the tabulation, and then use a selection statement to manually select the top ten stores.

Further reading
If you would like to learn more about the functions and operations discussed in this recipe, click on the links below:

\[ g_{\text{rank}}(G;S;O;X) \]

Time comparison analysis by groups
You can compare aggregated data for two different time periods, even if that data is contained in the same column.

Difficulty

Objective
You want to perform an analysis that compares the data from one time period to a previous time period for multiple different categories. For instance, retailers often want to compare this year’s sales to last year’s sales. Additionally, you want these sales to be further broken down by group. Often, especially in time-ordered data sets, the values you want to compare are located in the same column but not in the same row. You know how to do this in SQl by using **JOIN** and **UNION**, and you would like to produce the same results using 1010data.

SQL solution

```sql
SELECT results.month_2015 AS `Month`,results.group_2015 AS `Group`,
results.sum_sales_2015 AS `2015 Sum of Sales`,
results.sum_sales_2016 AS `2016 Sum of Sales`, results.sum_sales_2016 -
results.sum_sales_2015 AS YOY,
(results.sum_sales_2016 - results.sum_sales_2015)/results.sum_sales_2015 AS `YOY Perc` FROM
(SELECT t1.`MONTH(date)` AS month_2015 ,t1.group AS group_2015,t1.`ROUND(SUM(xsales),2)` AS sum_sales_2015,IFNULL(t2.`MONTH(date)`,
 t1.`MONTH(date)` ) AS month_2016,IFNULL(t2.group,t1.group) AS group_2016,IFNULL(t2.`ROUND(SUM(xsales),2),0)` AS sum_sales_2016 FROM
(SELECT `MONTH(date)`,YEAR(date),`group`, `ROUND(SUM(xsales),2)` FROM sales_item_detail WHERE YEAR(date)=2015 GROUP BY `MONTH(date),YEAR(date),`group`) AS t1
```
Using SQL, in order to utilize results from executed select statements in additional select statements, you need to nest the statements into one query. Additionally, each of these nested select statements need to be labeled as a temporary table, using `AS`. With 1010data, you can execute queries consecutively, using the results of the previous query as a starting point for the next one, without having to save your results or nest your statements.

Another drawback of using SQL to complete this problem is that when you join tables together using `LEFT JOIN` and `RIGHT JOIN`, values from the joining table that do not have a match, are given a value of `NULL`. With 1010data, you use a `<link>` instead of a `JOIN` to combine tables in this way and values that do not match are given a value of `0`. This makes it possible to still use these values in aggregations, where as aggregate functions in SQL ignore `NULL` values.

**Note:** This solution was created using MySQL where completing a `FULL OUTER JOIN` is not possible. Therefore, a `LEFT JOIN` and a `RIGHT JOIN` are combined with a `UNION` to produce the same affect.

### 1010data Macro Language solution

```xml
<base table="pub.doc.retail.altseg.sales_detail_transid"/>
<willbe name="month" label="Month" value="month(trans_date)"
      format="type:nocommas"/>
<willbe name="year" label="Year" value="year(trans_date)"
      format="type:nocommas"/>
<tabu label="Tabulation on Sales Detail" breaks="group,month,year">
    <break col="group" sort="up"/>
    <break col="month" sort="up"/>
    <break col="year" sort="up"/>
    <tcol source="xsales" fun="sum" name="sum"
          label="Sum of `Extended`Sales">
</tabu>
<willbe name="last_year" label="Last Years Sum"
       value="g_rshift(month group;;;;sum;-1)"
       format="dec:2"/>
<willbe name="yoy" label="YOY" value="sum-last_year"
       format="dec:2"/>
```
Performing a time comparison analysis allows you to examine how your data changes from one time period to another. Additionally, the summarized values for each time period should be displayed in adjacent columns to allow for easy comparison between the two.

This analogue presents two methods. The most efficient way is to use the 1010data function $g_rshift(G;S;O;X;N)$ to shift rows according to time period. This function is used to create a new column that, along with additional columns, can then be used to determine the percent loss or gain from one year to the next.

However, first two new columns representing the month and year are created to be used along with Group as break columns in a tabulation over each month/year/group combination. (Note that the sample data set, pub.doc.retail.salesdetail, only contains information for January of 2014 and 2015.) After the tabulation, the $g_rshift(G;S;O;X;N)$ function is used to shift the rows in the worksheet so that the same month from both years in the analysis are in adjacent rows. Once the aggregations are in the same row, calculating the difference between them, and ultimately the percentage change, is simple.

**Alternate 1010data Macro Language solution**

```xml
<base table="pub.doc.retail.altseg.sales_detail_transid"/>
<willbe name="month" label="Month" value="month(date)"/>
<willbe name="fy2015" label="FY2014" value="year(date)=2015"/>
<willbe name="fy2016" label="FY2015" value="year(date)=2016"/>
<tabu label="YOY Monthly Group Sales" breaks="month,group">
  <tcol source="xsales" fun="sumwd" name="sum2015" weight="fy2015" label="2015 Sum of Monthly Group Sales"/>
  <tcol source="xsales" fun="sumwd" name="sum2016" weight="fy2016" label="2016 Sum of Monthly Group Sales"/>
</tabu>
<willbe name="yoy" label="YOY" value="sum2016-sum2015" format="dec:2"/>
<willbe name="yoy_perc" label="YOY Percentage" value="yoy/sum2015" format="type:pct;dec:2"/>
<sort col="group" dir="up"/>
```

Another solution is to compute a weighted sum in a tabulation using reference columns, one for each time period, to place the desired values in new columns in the same row. Before completing a tabulation, you must create the reference columns. One column is created to distinguish the month and two additional columns are created to distinguish which fiscal year each transaction belongs to.

A tabulation then uses the Month column along with Group to break on and group the records by. The fiscal year flag columns are then used as weights for the dot product in the calculation. Once the aggregations are in the same row, calculating the difference between them, and ultimately the percentage change, is simple.

**Further reading**

If you would like to learn more about the functions and operations discussed in this recipe, click on the links below:

```xml
<tabu>
```

**Performing an inner join**

An INNER JOIN in SQL is similar to a link and select in 1010data.
**Objective**

You want to perform an "INNER JOIN" in 1010data, that is you want to combine the columns of two tables and only retain the entries that match.

**SQL solution**

```sql
SELECT *
FROM sales_item_detail
JOIN product_master
ON sales_item_detail.sku=product_master.sku
WHERE YEAR(date)=2016 AND groupdesc='FRUIT';
```

An INNER JOIN in SQL is the same as a regular JOIN. When two tables are joined, only the rows that match the criteria specified with ON are included in the result.

**1010data Macro Language solution**

```xml
<base table="pub.doc.retail.altseg.sales_detail_transid"/>
<sel value="year(trans_date)=2016"/>
<link table2="pub.doc.retail.product" col="sku" col2="sku" type="select">
  <sel value="groupdesc='FRUIT'"/>
</link>
```

Performing a JOIN or an INNER JOIN between two tables in SQL will result in a worksheet that contains only the matching rows between both of the tables. You can obtain the same result in 1010data by performing a link and select.

By specifying type="select", only the rows from the base table that have a match in the foreign table are retained. As shown here, you can nest a query inside the `<link>` that will be applied to the foreign table before the link occurs. Here, only groups that contain "FRUIT" in the description are selected from the foreign table, and this is used as the selection basis in the link.

However, unlike SQL, you need to be aware of which table is your base table and which is your foreign table. If the smaller table is your base table and the foreign table has multiple entries for one entry in your base table, only the first entry will be included in the results. You need to make sure either that the larger table is your base table, or that you use `expand="1"` if the larger table is your foreign table.

**Further reading**

If you would like to learn more about the functions and operations discussed in this recipe, click on the links below:

`<link>`

**Performing a left or right join**

With some manipulation, a simple link in 1010data can produce the same results as a LEFT JOIN or a RIGHT JOIN in SQL.
Objective
You want to perform a "LEFT JOIN" or a "RIGHT JOIN" in 1010data, that is you want to match all of the rows in your base table with the appropriate row in the foreign table, and if there is not a match, you want the result from the foreign table to be NULL.

SQL Solution
SELECT store_master.store,sum_sales,division,subdivision,manager,city,state
FROM (SELECT store,sum(xsales) as sum_sales
     FROM sales_item_detail
     WHERE date=20150621
     GROUP BY store)
     AS sales
LEFT JOIN store_master
     ON sales.store=store_master.store;

Performing a LEFT JOIN in SQL combines all of the columns from the left table to those of the right table. If there is an exact match between a row in both tables, then the corresponding data is joined in, if there is not a match to the right table, the columns from the right table for that row are populated with NULL values.

SELECT store_master.store,sum_sales,division,subdivision,manager,city,state
FROM (SELECT store,sum(xsales) as sum_sales
     FROM sales_item_detail
     WHERE date=20150621
     GROUP BY store)
     AS sales
RIGHT JOIN store_master
     ON sales.store=store_master.store;

A RIGHT JOIN is the opposite of a LEFT JOIN, where all information from the right table is kept and if there is not a match in the left table, the row is populated with NULL values in each of the columns.

1010data Macro Language solution
<base table="pub.doc.retail.altseg.sales_detail_transid"/>
<sel value="trans_date=20150621"/>
<tabu breaks="store" label="Tabulation">
    <break col="store" sort="up"/>
    <tcol fun="sum" source="xsales" name="sum_sales" format="dec:2"/>
</tabu>
<link table2="pub.doc.retail.altseg.stores" col="store" col2="store" cols="division,subdivision,manager,city,state"/>

<<base table="pub.doc.retail.altseg.stores"/>
<colord cols="store,division,subdivision,manager,city,state"/>
<sort col="store" dir="up"/>
<link table2="pub.doc.retail.altseg.sales_detail_transid" col="store" col2="store">
    <sel value="trans_date=20150621"/>
</link>
The difference of simulating a LEFT JOIN or a RIGHT JOIN in 1010data is the table that is designated as your base table. If you say that `pub.doc.retail.altseg.sales_detail_transid` is your "left table" and `pub.doc.retail.altseg.stores` is your "right table," then simulating a LEFT JOIN would require you to use the Sales Item Detail table as your base and then perform a simple link on `store` to the Store Master table.

Similarly, simulating a "RIGHT JOIN" requires you to use Store Master as your base table, then link to Sales Item Detail on `store`. As you can imagine, the results of these two "joins" differ from each other. Due to the selection of a specific date, some stores do not have entries in the Sales Item Detail table, because they did not have transactions on that day. Therefore, similar to the LEFT JOIN in SQL, only the store information for the stores that had sales on that day is brought in when using the Sales Item Detail table as a base. With the Store Master as the base, you obtain the same result as a RIGHT JOIN - the sum of sales is brought in for every store. Therefore, if a store had no sales on that day, the result is a NULL value.

In this example, there is only one row for each store, therefore the only difference between a "left join" or a "right join" is whether or not there is an entry for store 186, which had no sales on the date selected, June 21, 2016. However, if you did not tabulate to find the sum of sales and instead just wanted to bring in the store information for each individual transaction, the table that is used as your base is very important.

Joins in SQL will bring in all possible matches from the foreign table to the base table. Therefore, if you perform the RIGHT JOIN in the SQL example above, and use the whole Sales Item Detail instead of calculating the sum of sales per store, the information in the Store Master table will be duplicated to accommodate every entry in the sales table.

Regular links in 1010data, only bring in the first entry from the foreign table if there are multiple matches to the base table. Again starting with the Store Master and linking in the whole Sales Item Detail table, the result will be the same size as the store table and only the first transaction for each store will be linked in.

Thus, you should think carefully about which table you should use as your base. In general, your foreign table should contain the higher number of unique values. In this example, the transaction table does not contain any unique values in the store column, meaning that there is at least two entries for every store. However the store table only contains one entry for every store, therefore it has the higher number of unique values. So when performing a link between these two tables in their entirety, you should use the transaction table as your base.

### Further reading

If you would like to learn more about the functions and operations discussed in this recipe, click on the links below:

<link>

### Performing a union

A `UNION` in SQL combines the rows from one table with the rows from another, pending both tables contain the same number of columns with matching data types, similar to a merge in 1010data. During this union, all duplicates are removed.
Objective

You want to combine the rows of two tables that contain weather observations for different years. You know how to do this in SQL by performing a **UNION** of the two tables, and you would like to obtain the same results using 1010data.

**SQL solution**

```sql
SELECT * 
FROM hourly_weather_90 
UNION 
SELECT * 
FROM hourly_weather_91;
```

**1010data Macro Language solution**

```xml
<base table="pub.demo.weather.hourly90"/>
<merge table2="pub.demo.weather.hourly91" type="union"/>
```

With SQL, the default behaviour of **UNION** is to remove all duplicates from the resultant worksheet. The default behaviour for 1010data is to retain all rows, including duplicates. In order to obtain a worksheet with no duplicates, you must specify **type="union"** in the **<merge>** operation.

**Further reading**

If you would like to learn more about the functions and operations discussed in this recipe, click on the links below:

`<merge>`

**Performing a union all**

A **UNION ALL** in SQL combines the rows from one table with the rows from another, pending both tables contain the same number of columns with matching data types, similar to a **merge** in 1010data. With this variation of **UNION ALL** rows, including duplicates, are retained in the final worksheet.

**Difficulty**

![Difficulty Scale](image)

**Objective**

You want to combine the rows of two tables that contain weather observations for different years. Additionally, if there are duplicate rows between the two tables, you want to retain both copies. You know how to do this in SQL by performing a **UNION ALL** with the two tables, and you would like to obtain the same results using 1010data.

**SQL solution**

```sql
SELECT * 
FROM hourly_weather_90 
UNION ALL 
SELECT * 
FROM hourly_weather_91;
```

**1010data Macro Language solution**

```xml
<base table="pub.demo.weather.hourly90"/>
```
With SQL, `UNION ALL` will combine all rows from both tables and will retain duplicates. To obtain the same worksheet with 1010data, you can specify `type="union"` in the `<merge>` operation. However, the default behaviour for 1010data is to retain all rows, so not specifying a type will produce the same results.

### Further reading

If you would like to learn more about the functions and operations discussed in this recipe, click on the links below:

<merge>

### Denormalizing tables

In SQL it is very common to have smaller tables that are used as reference tables to a larger data set. This allows the data to be stored easier and accessed faster. However, if you want to perform any calculations that involve these reference tables in addition to your main table, you must first combine them, or denormalize them.

#### Difficulty

![Difficulty scale](image)

#### Objective

You would like to know the sum of sales of fruit during the summer of 2015 for each of your stores. Additionally, you would like to know the location of each of these stores. However, the necessary information to complete this analysis is contained in three different tables. Therefore, you need to combine or denormalize the tables first.

#### SQL solution

```sql
SELECT * 
FROM sales_item_detail 
LEFT JOIN product_master 
  ON sales_item_detail.sku=product_master.sku 
LEFT JOIN store_master 
  ON sales_item_detail.store=store_master.store;
```

Denormalizing the tables so that all information relevant to the problem is in one table is relatively simple. You just need two `JOIN`'s to combine all three tables. However, if you then want to perform an analysis on this denormalized table it gets more complicated. You could either create a temporary table from this query to then use in your next query, or you could add additional logic to the current query to get your desired results.

```sql
SELECT sales_item_detail.store, 
       city, 
       state, 
       SUM(xsales) AS sum_sales 
FROM sales_item_detail 
  LEFT JOIN product_master 
    ON sales_item_detail.sku=product_master.sku 
  LEFT JOIN store_master 
    ON sales_item_detail.store=store_master.store 
WHERE groupdesc='FRUIT' AND date<=20150621 AND date>=20150922 
GROUP BY sales_item_detail.store;
```
In the above query, additional logic is added to the prior query in order to obtain the desired results presented in the objective. However, when you submit this new query, the tables need to be recombined and the results you already obtained are not utilized. With 1010data, you can denormalize your tables and then run additional queries without resubmitting previous work.

1010data Macro Language solution

First a selection is made to include only those transactions that occur in the summer of 2015. In order to further select the products from the "Fruit Group," you need to link in the Product Master table. You could simply link in this table and make a selection afterwards, but for this solution the selection is done within the link operation and then type="include" is specified. This link type will not bring in any additional columns from the Product Master, but it will perform a selection on the base table based on the data present in the foreign table.

Then, you can perform a tabulation to obtain the sum of sales for fruit during the summer of 2015 by store, and link in the Store Master table to add location information to the worksheet.

Alternate 1010data Macro Language solution

This solution denormalizes the tables first, and then performs the necessary operations to obtain the desired calculation. The important thing to remember is that with either solution, the state of your session from the previous query is used going forward, so that operations do not need to be resubmitted.

Further reading

If you would like to learn more about the functions and operations discussed in this recipe, click on the links below:

Transposing a table

In order to transpose a table using SQL, multiple operations are required. Using 1010data, you can transpose a table with one basic operation.
**Difficulty**

![Difficulty Chart](image)

**Objective**

You have determined the top 5 selling products based on the sum sales for transactions occurring in 2015. Now that you have a column containing the top five SKUs, you want to transpose the table so that each SKU is in its own column.

**SQL solution**

```sql
SET @row_num=0;
SELECT MAX (CASE WHEN rank=1 THEN sku END) AS first,
       MAX (CASE WHEN rank=2 THEN sku END) AS second,
       MAX (CASE WHEN rank=3 THEN sku END) AS third,
       MAX (CASE WHEN rank=4 THEN sku END) AS fourth,
       MAX (CASE WHEN rank=5 THEN sku END) AS fifth
FROM (SELECT sku,
       @row_num := @row_num + 1 AS rank
    FROM (SELECT sku,
           sum(xsales) AS sum_sales
    FROM sales_detail
    WHERE YEAR(trans_date)=2015
    GROUP BY sku
    ORDER BY sum_sales DESC
    LIMIT 5
  ) sales_by_sku
) top_ranked;
```

In order to transpose a table in SQL, you have to create each new column in your SELECT statement and specify which values you want in each. In this example, it is not too intensive to create five additional columns containing only one type of information, however this becomes more difficult when transposing a larger, more complex table. Additionally, in order to transpose only the `sku` column, you need to create a column containing the row index, which is `rank` in this example.

**1010data Macro Language solution**

```xml
<base table="pub.doc.retail.altseg.sales_detail_sku"/>
<sel value="year(trans_date)=2015"/>
<willbe name="sales_per_sku" value="g_sum(sku;xsales)" format="dec:2"/>
<sel value="g_first1(sku;)">
<sort col="sales_per_sku" dir="down"/>
<sel value="i_()<=5"/>
<color cols="sku"/>
<willbe name="names"
  value="case(i_();1;'first';2;'second';3;'third';4;'fourth';'fifth')"/>
<transpose names="names"/>
```

With 1010data, one operation transposes your table- `<transpose/>`. This operation offers many attributes that give you the ability to customize your transposed table. In this analogue, the `names` attribute is used in order to name each column in the transposed table. Any column containing data formatted as text, can be used with this attribute. Additionally, there are attributes that allow you to label the columns of your new table, `labels`, and attributes that allow you to have data of different types transposed into one column, `promote`, among others.
Further reading

If you would like to learn more about the functions and operations discussed in this recipe, click on the links below:

<transpose>