For assistance with building queries using Query Manager outside of this class, please contact the Technology Support Center (TSC).

E-mail: tsc@bgsu.edu
Phone: 419-372-0999
## Table of Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduction</td>
<td>1</td>
</tr>
<tr>
<td>Create a Simple Query</td>
<td>12</td>
</tr>
<tr>
<td>Question 1A</td>
<td>15</td>
</tr>
<tr>
<td>Question 1B</td>
<td>16</td>
</tr>
<tr>
<td>Edit Fields – Select Fields to Display</td>
<td>17</td>
</tr>
<tr>
<td>Question 2</td>
<td>20</td>
</tr>
<tr>
<td>Edit Fields – Display Order and Sort Order</td>
<td>21</td>
</tr>
<tr>
<td>Question 3</td>
<td>26</td>
</tr>
<tr>
<td>Edit Fields – Heading and Translate Values</td>
<td>27</td>
</tr>
<tr>
<td>Question 4</td>
<td>32</td>
</tr>
<tr>
<td>Edit Criteria – Part I</td>
<td>33</td>
</tr>
<tr>
<td>Question 5A</td>
<td>43</td>
</tr>
<tr>
<td>Question 5B</td>
<td>44</td>
</tr>
<tr>
<td>Edit Criteria – Part II</td>
<td>45</td>
</tr>
<tr>
<td>Question 6</td>
<td>49</td>
</tr>
<tr>
<td>Working with Prompts</td>
<td>50</td>
</tr>
<tr>
<td>Question 7</td>
<td>52</td>
</tr>
<tr>
<td>Writing Expressions</td>
<td>53</td>
</tr>
<tr>
<td>Question 8</td>
<td>60</td>
</tr>
<tr>
<td>Using a Prompt with Expressions</td>
<td>61</td>
</tr>
<tr>
<td>Use Aggregate Functions</td>
<td>63</td>
</tr>
<tr>
<td>Question 10</td>
<td>66</td>
</tr>
<tr>
<td>Joins – Introduction</td>
<td>67</td>
</tr>
<tr>
<td>Joins – Record Hierarchy</td>
<td>72</td>
</tr>
<tr>
<td>Question 11</td>
<td>76</td>
</tr>
<tr>
<td>Joins – Related Record</td>
<td>77</td>
</tr>
<tr>
<td>Question 12</td>
<td>81</td>
</tr>
<tr>
<td>Joins – Any Record</td>
<td>82</td>
</tr>
<tr>
<td>Question 13</td>
<td>86</td>
</tr>
<tr>
<td>Outer Joins</td>
<td>87</td>
</tr>
<tr>
<td>Question 14</td>
<td>96</td>
</tr>
<tr>
<td>Subqueries – Checking for Existence</td>
<td>97</td>
</tr>
<tr>
<td>Subqueries – Single Value</td>
<td>104</td>
</tr>
<tr>
<td>Question 16</td>
<td>109</td>
</tr>
<tr>
<td>Unions</td>
<td>110</td>
</tr>
<tr>
<td>Scheduling Queries</td>
<td>115</td>
</tr>
<tr>
<td>Supplemental Material</td>
<td>Page</td>
</tr>
<tr>
<td>--------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>Finding Records and Fields</td>
<td>120</td>
</tr>
<tr>
<td>Effective Date, Effective Sequence, and Effective Status</td>
<td>121</td>
</tr>
<tr>
<td>Basics of “Effective” Data</td>
<td>121</td>
</tr>
<tr>
<td>Using Effective Date in Queries</td>
<td>123</td>
</tr>
<tr>
<td>Query Organization</td>
<td>124</td>
</tr>
<tr>
<td>Copy a Query to a User</td>
<td>125</td>
</tr>
<tr>
<td>Delete a Query</td>
<td>127</td>
</tr>
<tr>
<td>Move a Query to a Folder</td>
<td>128</td>
</tr>
<tr>
<td>Rename a Query</td>
<td>130</td>
</tr>
<tr>
<td>Save a Query with a New Name</td>
<td>131</td>
</tr>
<tr>
<td>Grouping Criteria and the OR operator</td>
<td>133</td>
</tr>
<tr>
<td>Choosing Logical Operators</td>
<td>133</td>
</tr>
<tr>
<td>Grouping Criteria</td>
<td>134</td>
</tr>
<tr>
<td>Wildcards</td>
<td>137</td>
</tr>
<tr>
<td>Criteria and Case-Sensitive Data</td>
<td>138</td>
</tr>
<tr>
<td>Key Fields</td>
<td>139</td>
</tr>
<tr>
<td>Primary Keys</td>
<td>139</td>
</tr>
<tr>
<td>Foreign Keys</td>
<td>140</td>
</tr>
<tr>
<td>Keys with Effective Dates</td>
<td>141</td>
</tr>
<tr>
<td>What is a View?</td>
<td>142</td>
</tr>
<tr>
<td>Expressions</td>
<td>142</td>
</tr>
<tr>
<td>String Functions</td>
<td>143</td>
</tr>
<tr>
<td>Numeric Functions</td>
<td>144</td>
</tr>
<tr>
<td>Date Functions</td>
<td>145</td>
</tr>
<tr>
<td>Conversion Functions</td>
<td>145</td>
</tr>
<tr>
<td>Condition Functions</td>
<td>148</td>
</tr>
<tr>
<td>CASE Expression</td>
<td>149</td>
</tr>
<tr>
<td>What Else is There?</td>
<td>151</td>
</tr>
<tr>
<td>Blank and Unknown Values</td>
<td>151</td>
</tr>
<tr>
<td>Structured Query Language (SQL)</td>
<td>155</td>
</tr>
<tr>
<td>Basic Query</td>
<td>155</td>
</tr>
<tr>
<td>Outer Joins</td>
<td>157</td>
</tr>
<tr>
<td>Aggregate Query</td>
<td>158</td>
</tr>
<tr>
<td>Query Manager and SQL</td>
<td>159</td>
</tr>
<tr>
<td>Troubleshooting</td>
<td>160</td>
</tr>
</tbody>
</table>
INTRODUCTION

You have completed the first two classes of the BGSU PeopleSoft Query course: Fundamentals of Database Structure and Query Viewer. You are now ready for the third class, Query Manager. Query Manager is a hands-on, two-full day class, providing useful query concepts. Query concepts include:

- Creating queries
- Edit Fields
- Edit Criteria
- Working with Prompts
- Writing Expressions
- Using Prompts & Expressions
- Using Aggregate Functions
- Inner & Outer Joins
- Subqueries
- Unions

Query Manager provides access to search for, view, run, create, modify, and save queries.

What is a Query?

A query is a question or request for information. For example, your manager might ask for a list of laboratory classes occurring during the summer 2013 semester. He would like to see the course ID, course offer number, session code, class section, subject, catalog number, academic career, and description fields. In order to obtain the results, several records can be joined together and saved as a query.

When using Query Viewer the queries have already been created by a user that has access to Query Manager. In order to know what results are correct, you first have to understand the basic terminology of a query as well as how a query is structured.
If you recall from Query class I – Fundamentals of Database Structure the PeopleSoft query terms include:

| Record: | Stores information about many objects of interest of the same type (laboratory classes). |
| Fields: | Individual characteristics for that object of interest (course ID, Offer Nbr, etc). |
| Rows: | All of the data that represents that object of interest. The entire row contains all of the information on the laboratory class. |

Let’s take a deeper look at how a query is structured.

There is more to a query than records, fields, and rows. There is a process that assists users in creating a query.

| Records: | Search for records to add to your query (CLASS_TBL) |
| Fields: | Edit or change column headings, column order, and ordering results for the selected fields. |
| Expressions: | Formulas used to calculate a result from data in fields or other values. |
| Prompts: | Add, change, and delete the values the user is asked to enter when the query is run. |
| Criteria: | Set the conditions that determine which rows will appear in the results, based on the data in each row. |
| Run: | Display of the query results in a browser window. |

* denotes optional task

Now that you know what a query is and the process of creating a query let’s move forward.
**Basic Search**

A Basic search can be used when some information about the Query name is known.

The navigation menu that appears may look very different from your navigation menu; a user’s security access determines the CSS modules that are available to them.

Once signed into the system, click the arrow next to **Reporting Tools**. Next, click the arrow beside **Query**. Finally, **click** the **Query Manager** link.

The system takes you to the Basic search page. As you learned in Query class II – View a Query; located on the Basic search page is a Search By field with a drop down menu. The drop down menu has criteria that will assist in further defining a search. Only one criterion at a time can be used when performing a Basic search.

The system defaults to Search By: Query Name. There is a high volume of public queries that exist in the system. In order to identify custom queries (queries that are created by BGSU for BGSU) a naming convention has been created and is encouraged to be followed by users creating queries.
The following naming conventions apply to BGSU Custom *public* queries.

<table>
<thead>
<tr>
<th>Name Component</th>
<th>R</th>
<th>Description</th>
<th>Valid Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>FACILITY</td>
<td>R</td>
<td>“BG” for BGSU</td>
<td>BG</td>
</tr>
<tr>
<td>UNDERSCORE</td>
<td>R</td>
<td>Separation</td>
<td>_</td>
</tr>
<tr>
<td>PRODUCT SUFFIX</td>
<td>R</td>
<td>Two character Product Abbreviation</td>
<td>See Table- Product Suffixes</td>
</tr>
<tr>
<td>UNDERSCORE</td>
<td>R</td>
<td>Separation</td>
<td>_</td>
</tr>
<tr>
<td>QUERY NAME</td>
<td>R</td>
<td>Descriptive name for the query</td>
<td>Developer Defined</td>
</tr>
</tbody>
</table>

### Product Suffix (CSS)

<table>
<thead>
<tr>
<th>Product Suffix (CSS)</th>
</tr>
</thead>
<tbody>
<tr>
<td>AD</td>
</tr>
<tr>
<td>CC</td>
</tr>
<tr>
<td>FA</td>
</tr>
<tr>
<td>HO</td>
</tr>
<tr>
<td>SF</td>
</tr>
<tr>
<td>SI</td>
</tr>
<tr>
<td>SR</td>
</tr>
</tbody>
</table>

### Product Suffix (FMS)

<table>
<thead>
<tr>
<th>Product Suffix (FMS)</th>
</tr>
</thead>
<tbody>
<tr>
<td>AP</td>
</tr>
<tr>
<td>GL</td>
</tr>
<tr>
<td>GM</td>
</tr>
<tr>
<td>PO</td>
</tr>
</tbody>
</table>

### Product Suffix (HCM)

<table>
<thead>
<tr>
<th>Product Suffix (HCM)</th>
</tr>
</thead>
<tbody>
<tr>
<td>BA</td>
</tr>
<tr>
<td>BN</td>
</tr>
<tr>
<td>HR</td>
</tr>
<tr>
<td>PY</td>
</tr>
<tr>
<td>TL</td>
</tr>
</tbody>
</table>

### EXAMPLES:

<table>
<thead>
<tr>
<th>EXAMPLES:</th>
</tr>
</thead>
<tbody>
<tr>
<td>BG_SR_ALL_CLASS_RM_INFO</td>
</tr>
<tr>
<td>BG_FA_PKG_PLAN_SETUP</td>
</tr>
<tr>
<td>BG_HR_ALLDEPTMENTS</td>
</tr>
</tbody>
</table>
As Query Names appear in the system:

What if you don’t know the Query Name? Then, a different criterion can be used to assist in your search. For example, you do not know the query name yet you know the Folder Name.

A Basic search will only search by the “begins with” condition type. When a query is placed in a public folder, often times the query owner will share the Folder Name.

Below is an example of using the criterion Folder Name with a “begins with” condition of “Student”. As the example shows, all folders that begin with Student appear in the Search Results.

![Example image of Query Manager with Folder Name search](image-url)
**Exercise 1**

Practice using the Basic search method. Use the following criteria to record the number of queries that exist.

<table>
<thead>
<tr>
<th>Search By</th>
<th>Begins With</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>Query Name</td>
<td>BGFA</td>
<td>________</td>
</tr>
<tr>
<td></td>
<td>BG_FIRE</td>
<td>________</td>
</tr>
<tr>
<td></td>
<td>BG_CC</td>
<td>________</td>
</tr>
<tr>
<td></td>
<td>BG_AD</td>
<td>________</td>
</tr>
<tr>
<td>Folder Name</td>
<td>Student</td>
<td>________</td>
</tr>
<tr>
<td></td>
<td>Admissions</td>
<td>________</td>
</tr>
<tr>
<td></td>
<td>Training</td>
<td>________</td>
</tr>
<tr>
<td>Description</td>
<td>ISIR</td>
<td>________</td>
</tr>
<tr>
<td></td>
<td>Class</td>
<td>________</td>
</tr>
<tr>
<td></td>
<td>Course</td>
<td>________</td>
</tr>
<tr>
<td></td>
<td>Grade</td>
<td>________</td>
</tr>
<tr>
<td></td>
<td>Academic</td>
<td>________</td>
</tr>
</tbody>
</table>
**Advanced Search**

Using the Basic search function is easy when you know what the query, folder, or description *begins with*. The Advanced search method allows several criteria to be used as well as a selection of conditions to place on the criteria.

Click the **Advanced Search** link

The system takes you to the Advanced search page.

Notice the fields on the Advanced search page are the same options that you find on the Basic search page in the Search By dropdown. The Advanced search page offers the use of several different criteria and condition types. Condition types are used with search criteria to limit the results you want to see when searching for a query.

<table>
<thead>
<tr>
<th>Condition Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>begins with</em></td>
<td>applies to the start of <code>&lt;The folder name begins with Student&gt;</code></td>
</tr>
<tr>
<td><em>between</em></td>
<td>find cases in which the criterion is between two values <code>&lt;amount owed is between $250.99 and $999.99&gt;</code></td>
</tr>
<tr>
<td><em>contains</em></td>
<td>have within <code>&lt;the description contains the word “term”&gt;</code></td>
</tr>
<tr>
<td><em>in</em></td>
<td>the criterion is in a list of values <code>&lt;is in the list ACAD_PROG, ACAD_PLAN, DESCR&gt;</code></td>
</tr>
<tr>
<td><em>not=</em></td>
<td>the value is not equal to what you enter <code>&lt;Term not= 2095&gt;</code></td>
</tr>
</tbody>
</table>
Below is an example of using different criteria and different condition types to search for results.

**Query Manager**

Enter any information you have and click Search. Leave fields blank for a list of all values.

ADVANCED SEARCH ALLOWS MULTIPLE CRITERIA AND MULTIPLE CONDITION TYPES TO BE USED

When using the IN or BETWEEN operators, enter comma separated values without quotes. i.e. JOB,EMPLOYEE,IRNL,IN.

<table>
<thead>
<tr>
<th>Query</th>
<th>Description</th>
<th>Owner</th>
<th>Folder</th>
<th>Edit</th>
<th>Raw in</th>
<th>Raw out</th>
<th>Raw XML</th>
<th>Schedule</th>
</tr>
</thead>
<tbody>
<tr>
<td>BG_SR</td>
<td>ACAD_PLAN_COUNTS_ALL</td>
<td>Student Records</td>
<td>Student</td>
<td>STUDENT</td>
<td>Records</td>
<td>Edit</td>
<td>HTML</td>
<td>Excel</td>
</tr>
<tr>
<td>BG_SR</td>
<td>ACAD_PLAN_COUNTS_W_DUPLICATES</td>
<td>Student Records</td>
<td>Student</td>
<td>STUDENT</td>
<td>Records</td>
<td>Edit</td>
<td>HTML</td>
<td>Excel</td>
</tr>
<tr>
<td>BG_SR</td>
<td>ACAD_STANDING_ALL</td>
<td>Student Records</td>
<td>Student</td>
<td>STUDENT</td>
<td>Records</td>
<td>Edit</td>
<td>HTML</td>
<td>Excel</td>
</tr>
<tr>
<td>BG_SR</td>
<td>ACAD_STANDING_LIST</td>
<td>Student Records</td>
<td>Student</td>
<td>STUDENT</td>
<td>Records</td>
<td>Edit</td>
<td>HTML</td>
<td>Excel</td>
</tr>
</tbody>
</table>
**Exercise 2**

This exercise will allow you to use multiple criteria and specific condition types selected for each criterion. When there are two criteria be sure to enter both of them before clicking the Search button.

<table>
<thead>
<tr>
<th>Criteria</th>
<th>Condition Type</th>
<th>Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>Query Name</td>
<td>contains</td>
<td>Campus</td>
</tr>
<tr>
<td>Description</td>
<td>contains</td>
<td>Location</td>
</tr>
<tr>
<td>Query Name</td>
<td>contains</td>
<td>Schedule</td>
</tr>
<tr>
<td>Description</td>
<td>contains</td>
<td>Athletes</td>
</tr>
<tr>
<td>Folder Name</td>
<td>contains</td>
<td>Records</td>
</tr>
<tr>
<td>Query Name</td>
<td>contains</td>
<td>Class</td>
</tr>
<tr>
<td>Description</td>
<td>contains</td>
<td>Term</td>
</tr>
<tr>
<td><strong>Uses Field Name</strong></td>
<td><strong>begins with</strong></td>
<td><strong>Acad</strong></td>
</tr>
<tr>
<td><strong>Uses Field Name</strong></td>
<td><strong>begins with</strong></td>
<td><strong>Session</strong></td>
</tr>
<tr>
<td><strong>Uses Field Name</strong></td>
<td><strong>begins with</strong></td>
<td><strong>INSTR</strong></td>
</tr>
</tbody>
</table>
Retrieving Queries

Now that you are familiar with the different search methods and naming conventions, let’s retrieve the query that we are searching for. For this example, the Basic Search method is used. The Search By field criteria is Folder Name and the search is on Career.

There are several results that meet the search criterion. Click the Edit link. The system places you on the Fields page. Additional query building pages are accessed by clicking the desired tab.
Page Description:

- **Records** - Search for records to add to your query, view a list of fields in a record, and add a record to your query.

- **Query** - Select fields to be displayed in the results. You can also join additional records to your query here and delete records.

- **Expressions** - Formulas used to calculate a result from data in fields and other values.

- **Prompts** - Add, change, and delete the values the user is asked to enter when the query is run.

- **Fields** - Edit or change column headings, column positions, and ordering results for the fields selected.

- **Criteria** - Set the conditions that determine which rows will appear in the results, based on the data in each row.

- **Having** - Set the conditions that determine which rows will appear in the results, based on the data in a group of rows.

- **View SQL** - View SQL statement generated whenever the query is modified.

- **Run** - Display query results in the browser window.
**Create a Simple Query**

In this exercise you are asked to get the basic information about all campuses recorded in PeopleSoft. Let’s create a query using the `CAMPUS_TBL` record to demonstrate the default criteria, Check All button, and the Effective Date criteria.

Creating a query isn’t as hard as we think it is. Creating queries becomes challenging when you have to locate the records and know what data resides in them. For purposes of training, you will be provided with all of the record and field names.

Starting in Query Manager, **click Create New Query**.

The Records page is displayed with default search criterion as Record Name. **Enter CAMPUS_TBL** then click the **Search** button.

**Click** the **Add Record** link.

**Click OK** to acknowledge that an effective date criteria has been automatically added.
Click the Check All button to add all fields.

Click the Run tab.

That’s it! You have results!

Now, let’s save it. The only page that does not have the save option is Run. Click the Records tab then click Save As.

During training, the same naming convention for saving will be used. Enter `TRNG_QM##_E1` for the query name. To help you remember what this query is about, you may also wish to enter a description.

Finally, click the OK button.
The query name and description each have a maximum length of 30 characters. You may need to abbreviate some terms.

A meaningful description can help you identify a query. The description appears in the list of queries when you search. You can also search for queries by using words in the description.
**Question 1A**

The first query you will create is a query to list all academic programs. Show all fields from the academic programs record in the results.

- Academic programs are in the `ACAD_PROG_TBL` record
- Save and Save often. Save this query with the name `TRNG_QM##_Q1A`

What is the code corresponding to the College of Business Administration?

What does the code SPEC represent?

![Academic Programs Table]

| Institution | Acad Prog | Eff Date | Status | Descr | Short Descr | Career | Calendar | Adver Edit | Level Rate | Acad Group | Acad Plan | Campusc | First Term | CP ExecRate | DocExt Only | Princacy |
|-------------|-----------|----------|--------|-------|------------|--------|----------|------------|------------|------------|-----------|----------|---------|-----------|-------------|-------------|-----------|---------|
| 1 BGSUN | ACEN | 05/29/2013 | A | Academic Enhancement | ACEN | UGRD | USEM | P | UGRD | AEFC | MAIN | 0000 | Y | 20 |
| 2 BGSUN | MAST | 08/25/2014 | A | Graduate College - Master | MAST | GRAD | GSEM | P | GRAD | GRCLG | MAIN | 0000 | Y | 10 |
| 3 BGSUN | ARTSC | 05/29/2013 | A | College of Arts and Sciences | ARTSC | UGRD | USEM | P | UGRD | ASCLG | MAIN | 0000 | Y | 20 |
| 4 BGSUN | BUSN | 05/29/2013 | A | College of Business Admin | BUSN | UGRD | USEM | P | UGRD | BACLG | MAIN | 0000 | Y | 20 |
| 5 BGSUN | CERT | 05/29/2013 | A | Graduate College - Certificate | CERT | GRAD | GSEM | P | GRAD | GRCLG | MAIN | 0000 | Y | 10 |
| 6 BGSUN | DOCT | 05/29/2013 | A | Graduate College - Doctoral | DOCT | GRAD | GSEM | P | GRAD | GRCLG | MAIN | 0000 | Y | 10 |
| 7 BGSUN | EDUC | 05/29/2013 | A | Education & Human Development | EDUC | UGRD | USEM | P | UGRD | EDCLG | MAIN | 0000 | Y | 20 |
| 8 BGSUN | FIRE | 05/29/2013 | A | Firelands College | FIRE | UGRD | USEM | P | UGRD | FCCLG | FIRE | 0000 | Y | 20 |
| 9 BGSUN | GSTNA | 05/29/2013 | A | Guest Student | GSTNA | UGRD | USEM | P | USGT | RROFC | 0000 | Y | 0 |
| 10 BGSUN | HTH | 05/29/2013 | A | Health and Human Services | HTH | UGRD | USEM | P | UGRD | HHCLG | MAIN | 0000 | Y | 20 |
| 11 BGSUN | MUSC | 05/29/2013 | A | College of Musical Arts | MUSC | UGRD | USEM | P | UGRD | MOCLG | MAIN | 0000 | Y | 20 |
| 12 BGSUN | NDGNA | 05/29/2013 | A | Graduate Student - Non Degree | NDGNA | GRAD | GSEM | P | SYRGR | GRCLG | 0000 | Y | 0 |
| 13 BGSUN | SPEC | 05/29/2013 | A | Graduate College - Specialist | SPEC | GRAD | GSEM | P | GRAD | GRCLG | MAIN | 0000 | Y | 10 |
| 14 BGSUN | TECH | 08/29/2013 | A | Cell of Tech, Arch & Appl Engr | TECH | UGRD | USEM | P | UGRD | TECLG | MAIN | 0000 | Y | 20 |
| 15 BGSUN | NDGA | 05/19/2014 | A | Graduate Student - Non Degree | NDGA | GRAD | GSEM | P | SYRGR | GRCLG | MAIN | 0000 | Y | 90 |
| 16 BGSUN | GSTFA | 05/19/2014 | A | Guest Student | GSTFA | UGRD | USEM | P | SYRUG | RROFC | 0000 | Y | 90 |
**Question 1B**

This question being asked is to create a query that lists all of the known departments. Show every field of each department.

- Departments are stored in the **DEPT_TBL** record
- There will be several thousand rows in the results. Use the **View All** link and your browser’s **Find** feature to locate the information you need
- Save and Save often. **Save As TRNG_QM##_Q1B**

![Department List](image)

What is the department ID for Office of Student Retention?  
______________

Which department has the ID 303000?  
______________

---

**MEMO**
**Edit Fields – Select Fields to Display**

As you have seen, selecting all of the fields can result in dozens of columns. Often times we will want to select the fields we want to display in the results. If you have noticed, short descriptions are provided next to the fields, on the Query page. Let’s see.

We will get a list of all countries known to PeopleSoft. Rather than showing all of the fields of each country, let’s show only the country, description, and two-character code.

- Create a query on **COUNTRY_TBL**
- Show the following fields:
  - **COUNTRY**
  - **DESCR**
  - **COUNTRY_2CHAR**

![Field Selection Screen]

- **Click Run** tab

Your results show three fields: Country, Description, and 2-Character Code.

<table>
<thead>
<tr>
<th>Country</th>
<th>Description</th>
<th>2-Char Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>KOS</td>
<td>Kosovo</td>
<td>X</td>
</tr>
<tr>
<td>NAP</td>
<td>Not Applicable</td>
<td>N</td>
</tr>
<tr>
<td>NRE</td>
<td>Not Reported</td>
<td>R</td>
</tr>
<tr>
<td>ABW</td>
<td>Aruba</td>
<td>AW</td>
</tr>
<tr>
<td>AFG</td>
<td>Afghanistan</td>
<td>F</td>
</tr>
<tr>
<td>AGO</td>
<td>Angola</td>
<td>A</td>
</tr>
<tr>
<td>AIA</td>
<td>Anguilla</td>
<td>A</td>
</tr>
<tr>
<td>ALB</td>
<td>Albania</td>
<td>AL</td>
</tr>
<tr>
<td>AND</td>
<td>Andorra</td>
<td>AD</td>
</tr>
<tr>
<td>ANT</td>
<td>Netherlands Antilles</td>
<td>N</td>
</tr>
</tbody>
</table>

- **Save and Save often. Save As** TRNG_QM##_E2A
Next, assume that you do not need all of the information about each campus in the query you created earlier. Let’s remove three of the fields from the results.

Go back to the Query Manager Basic search page. Search for the TRNG_QM##_E1 query you built earlier.

- Click on the Edit link. This will place you on the Fields page.
- You will be creating a copy of the original query then editing the copy. Click the Save As link and save this query with the name TRNG_QM##_E2B. (It is important to use Save As so you don’t overwrite the original query.)
- Click the Query tab
Earlier we used this example to “check all” fields. Now we are going to uncheck fields.
Uncheck the following fields:

- **INSTITUTION**
- **DESCRSHORT**
- **FACILITY_CONFLICT**

Save the query again then run the query. Notice anything with the results? The amount of rows remains the same. Three fields have been removed from the results.

The fields selected in the Query tab affect only which fields are shown in the results. Consider this example: your neighbor has 20 books on his bookshelf. If you ask him about the books that are on his bookshelf, he can tell you only the title of the book, the title and author of each book, or the title, author, and publisher of each book, but he will always tell you about 20 books.
Question 2

For this question, create a query that lists all aid disbursement plans at BGSU. Show only the aid year, career, disbursement plan code, and description of each plan.

- Use the `DISB_PLAN_TBL` record
- Show only the `AID_YEAR`, `ACAD_CAREER`, `DISBURSEMENT_PLAN`, and `DESCR` fields
- Save and Save often. Save this query as `TRNG_QM##_Q2`
**Edit Fields – Display Order and Sort Order**

Sometimes the results of the query do not look “organized”. The order of columns from left to right may not be ideal for your purposes. The results may have no particular order, with values in primary fields going from low to high and back to low throughout. You can use the Reorder/Sort feature in the Fields tab to set the order in which columns are displayed from left to right and the ordering of rows in the results.

Let’s get a list of job codes in PeopleSoft. Show only the job code, effective date, description, job function, and grade for each job code. We will then make the description be the left-most column in the results and order the rows so that they are listed by ascending job code and descending grade.

- Use the following record **JOBCODE_TBL** (job codes)
- Show the following fields:
  - **JOBCODE**
  - **EFFDT**
  - **DESCR**
  - **JOB_FUNCTION**
  - **GRADE**

- **Save** the query as **TRNG_QM##_E3**.
- **Click Run** tab. Notice that the fields are in the same order as the fields listed in the Query tab.
To better organize the structure of the results, let’s place the DESCR field as the left-most column. Notice on the Fields page there are short descriptions provided for the field names.

- **Click Fields** tab. This will allow us to reorder fields and sort rows
- **Click Reorder / Sort** button

![Fields page with DESCR field moved to the first column]

- Enter a **1** in the **New Column** field of the DESCR row to move DESCR to the first column
- **Click OK**

The Fields page appears. DESCR is now the first column.

![Fields page showing DESCR as the first column]

**Save** the query.
Click the Run tab. View the results. Notice the DESCR field is the left-most column.

Next, go back to the Edit Field Ordering screen and sort the results by job code.

- Click Fields tab
- Click Reorder / Sort button
- Place a “1” in the New Order By column in the A.JOBCODE row
- Click OK
Click Run tab to view results

Results show that the rows are ordered by job code in ascending order.

Now, go back to the Edit Field Ordering screen and sort the results by descending grade first and job code second.

- Click the Fields tab
- Click the Reorder / Sort button
- Sort the GRADE field in descending order
  - Place a “1” in the New Order By column.
  - Check the Descending box
- Place a 2 in the New Order By column of the A.JOBCODE row
- Click OK

Look at the Fields page. Observe the “2” in the A.JOBCODE row and the “1D” in the A.GRADE row.
Results show that the Grade rows are in descending order. The rows ordered by Job Code are in ascending order within each Grade.

It can be easy to confuse the ordering of columns with the ordering of rows. It may help to keep the following guidelines in mind:

**Columns:**
- How fields appear in the results from *left* to *right*
- Change their position by entering numbers in the New Column column
- Look for key phrases in the query request such as “order the *fields,*” “order the *columns,*” and “show <some field> *first.*” Also look for the order in which fields are named in the request.

**Rows:**
- How data is ordered from *top* to *bottom*
- Change the ordering by entering numbers in the New Order By column
- Look for key phrases in the query request such as “sort the *data by,*” “order the *data,*” “sort the *results,*” “display the data *in order by,*” “from highest to lowest,” etc. Also look for words as “sort,” “order,” “ascending,” and “descending.”

**IMPORTANT:** if you do not specify any row ordering, the results will be sorted in an arbitrary order! You should always have a specific sort order in your queries so the results will be consistently displayed in the same order. This also helps organize the rows in a meaningful way.

(In this course we won’t always set a row ordering – but only so you can concentrate on the topic at hand!)
**Question 3**

Create a query that lists the academic programs defined for the University. Show only the academic career, academic program, description, and campus. Display the description column first. Sort the results by career first in descending order (so UGRD comes before GRAD) and academic program second.

- Use the `ACAD_PROG_TBL` record.
- Select the `ACAD_PROG`, `DESCR`, `ACAD_CAREER`, and `CAMPUS` fields.
- Move the `DESCR` field to be the left-most displayed.
- Sort by `ACAD_CAREER` descending and `ACAD_PROG` ascending.
- Save the query with the name `TRNG_QM##_Q3`.

<table>
<thead>
<tr>
<th>#</th>
<th>Academic Enhancement</th>
<th>ACEN</th>
<th>UGRD</th>
<th>MAIN</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>College of Arts and Sciences</td>
<td>ARTSC</td>
<td>UGRD</td>
<td>MAIN</td>
</tr>
<tr>
<td>3</td>
<td>College of Business Admin</td>
<td>BUSN</td>
<td>UGRD</td>
<td>MAIN</td>
</tr>
<tr>
<td>4</td>
<td>Education &amp; Human Development</td>
<td>EDUC</td>
<td>UGRD</td>
<td>MAIN</td>
</tr>
<tr>
<td>5</td>
<td>Firelands College</td>
<td>FIRE</td>
<td>UGRD</td>
<td>FIRE</td>
</tr>
<tr>
<td>6</td>
<td>Guest Student</td>
<td>GSTFA</td>
<td>UGRD</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>Guest Student</td>
<td>GSTNA</td>
<td>UGRD</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>Health and Human Services</td>
<td>HLTH</td>
<td>UGRD</td>
<td>MAIN</td>
</tr>
<tr>
<td>9</td>
<td>College of Musical Arts</td>
<td>MUSIC</td>
<td>UGRD</td>
<td>MAIN</td>
</tr>
<tr>
<td>10</td>
<td>Coll of Tech, Arch &amp; Appl Engr</td>
<td>TECH</td>
<td>UGRD</td>
<td>MAIN</td>
</tr>
<tr>
<td>11</td>
<td>Graduate College - Certificate</td>
<td>CERT</td>
<td>GRAD</td>
<td>MAIN</td>
</tr>
<tr>
<td>12</td>
<td>Graduate College - Doctoral</td>
<td>DOCT</td>
<td>GRAD</td>
<td>MAIN</td>
</tr>
<tr>
<td>13</td>
<td>Graduate College - Master</td>
<td>MAST</td>
<td>GRAD</td>
<td>MAIN</td>
</tr>
<tr>
<td>14</td>
<td>Graduate Student - Not Degree</td>
<td>NGDFA</td>
<td>GRAD</td>
<td>MAIN</td>
</tr>
<tr>
<td>15</td>
<td>Graduate Student - Not Degree</td>
<td>NGDNA</td>
<td>GRAD</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>Graduate College - Specialist</td>
<td>SPEC</td>
<td>GRAD</td>
<td>MAIN</td>
</tr>
</tbody>
</table>
**Edit Fields – Heading and Translate Values**

The headings of columns can appear in short or long description format. PeopleSoft automatically places the short description on field headings. PeopleSoft does give us the option to place a long description on the field headings. Also, many values are stored and displayed as codes and abbreviations rather than words, such as “A” for “active” and “NWD” for “not withdrawn.” Sometimes it is preferable to show words instead of these codes, called *translate values* in PeopleSoft. You can show a short or long description instead of a translate value in the results of a query. The next exercise will demonstrate how to change a heading to a long description and how to translate the value to something that is easily understood.

In the next exercise, you will create a query to list item types. The query will show the set ID, item type, effective date, effective status, description, item type code, and enrolled flag of each item type. You will then change the description field heading and show the meaning of each item type code.

Create a new query on the **ITEM_TYPE_TBL** record. Show the following fields:

- SETID
- ITEM_TYPE
- EFFDT
- EFF_STATUS
- DESCR
- ITEM_TYPE_CD
- ENROL_FLG

![Diagram of query configuration options]

In the next exercise, you will create a query to list item types. The query will show the set ID, item type, effective date, effective status, description, item type code, and enrolled flag of each item type. You will then change the description field heading and show the meaning of each item type code.
Click the Run tab

Observe the heading “Descr”. Observe the Item Code “C”.

Click the Fields tab

Save and save often. Save as TRNG_QM##_E4

Click the Edit button in the DESCR row.

The Edit Field Properties page appears. The Field Name category shows the name of the field and the short and long description for the field. Four options appear in the Heading section. None, Text (enter any text), RFT Short (defined short name), and RFT Long (defined long name). Click on RFT Short or RFT Long to use the predefined short or long heading text; click on Text to enter your own heading text.
If the Heading is switched from RFT Short to Text, the change in the Heading Text will not be shown until you click OK. This occurs if the Heading value is changed, no matter what its original value and new value are.

Click RFT Long to display a long description for the field heading.

- Click OK. The Heading Text for DESCR is now “Description” instead of “Descr”.

- Click OK. The Heading Text for DESCR is now “Description” instead of “Descr”.

- Save your query
- Click the Run tab

The Descr field heading now appears as Description.

Earlier, Item Code values were pointed out. Take the opportunity to scroll through the results to see if other values appear. (Use the View All link to display all of the rows to see more item codes.) Item Code values C, D, F, P, R, W, and X appear in the results. What do they mean?

- Click Fields tab
- Click Edit in the ITEM_TYPE_CD row
There is a set of meanings defined for the possible values of ITEM_TYPE_CD. Query Manager is aware of this and provides the option to show the meaning in place of the code in the results. Notice the Edit Fields Properties page has changed. A Translate Value section appears. The Translate Value section has three options: None (currently selected), Short, and Long.

- **Click Short**
- **Click OK**

You are returned to the **Fields** tab. The XLAT column in the ITEM_TYPE_CD row now shows “S” to indicate the short translate value will be displayed in the results.

- **Save** your query
- **Click** the **Run** tab
What are the meanings of the item type codes C, D, F, P, R, W, and X?
Question 4

Create a new query on academic plans. Show the following fields: academic plan, effective date, description, academic plan type, and academic program. Make the academic program the left-most column and order the results by academic programs in descending order followed by the academic plan in ascending order. Change the heading of academic programs to “Program” and show the short translation of academic plan type.

- Use the record ACAD_PLAN_TBL
- Show the following fields:
  - ACAD_PLAN
  - EFFDT
  - DESCR
  - ACAD_PLAN_TYPE
  - ACAD_PROG
- Move ACAD_PROG to be the first column
- Sort by ACAD_PROG in descending order and ACAD_PLAN in ascending order
- Change the heading of ACAD_PROG to “Program”
- Change the translate value type of ACAD_PLAN_TYPE to Short

Save and Save often. Save as TRNG_QM##_Q4
\textit{Edit Criteria – Part I}

Maybe you have to write a lot of letters to residents of Canada and you need a list of the Canadian provinces and their abbreviations. When you have much more data than you need or want you must add specific criteria to the query to retrieve only the desired data. Selection criteria apply to all data examined by the query and the system will only retrieve the rows with the specified criteria.

Create a query to show all state codes and names. Limit the results to Canadian provinces.

1. Start a new query on \texttt{STATE_TBL} (state codes and names)
2. Show all fields
3. Click Run tab

\begin{center}
\begin{tabular}{|c|c|c|}
\hline
\textbf{Country} & \textbf{State} & Description \\
\hline
1 & GBR & Durham \\
2 & GBR & Dyfed \\
3 & GBR & E. YORK & East Riding of Yorkshire \\
4 & GBR & E. SUSX & East Sussex \\
5 & GBR & ESSEX & Essex \\
6 & GBR & FER & Fermanagh \\
7 & GBR & RFE & Fife Region \\
8 & GBR & GLOUFS & Gloucestershire \\
9 & GBR & GRAM & Grampian Region \\
10 & GBR & GTRON & Greater London \\
\hline
\end{tabular}
\end{center}

States and provinces in PeopleSoft belong to countries; the country the state or province belongs to is in the COUNTRY field, which is shown in the Country column of the results.

We want to find the Canadian provinces. Use your browser’s Find feature to locate all provinces that have a country of “CAN”, the value representing Canada.

1. Click View All link to show all results
2. Use the browser Find feature and search for “CAN”
3. Save and Save often. Save as \texttt{TRNG_QM##_E5}

It would be easier to restrict the query results to just the Canadian provinces. To do this we must add criteria to the query. \textit{Criteria} are conditions added to a query that restrict the rows that are included in the results. Criteria are almost always a comparison between two expressions, which are sets of symbols, field names, and values that produce results. An expression can be a single field or values or can be a complex calculation. Most criteria will involve comparing the value in a field to a value in another field or to a constant, a number, work, or phrase that you enter. Comparisons between expressions usually check whether two expressions are equal or if one is greater than another.

1. Click Fields tab
We have used the Reorder / Sort button and the Edit button; the only one left is the Add Criteria funnel 🔗. The Edit Criteria Properties page appears.

- **Click** 🔗 in the COUNTRY row

In the Edit Criteria Properties page, you must identify two expressions, Expression 1 and Expression 2. For each expression you must choose its type and define the expression. For Expression 1, you can set Expression 1 type to either Field to use a field from a record or Expression to use a new or already defined expression. Most of the time, you will select Field for the type. For Expression 2, you have three more Type options: Constant (enter a value), Prompt, and Subquery. The prompt and subquery options are discussed later in this course.

To obtain a list of Canadian provinces and their abbreviations, we started by building a query that listed all states and provinces known to the system. Now we are adding a criterion that COUNTRY field must have a value of “CAN,” the country code representing Canada.

The first part of the criterion will refer to the COUNTRY field. **Set** the **Expression 1 Type** option to **Field**. **Select** A.COUNTRY for the Record Alias.Fieldname option; use the magnifying glass to look up records and fields available to the query. (If you click the Add Criteria funnel button while on the Query page, the Record Alias.Fieldname option is automatically filled in.)

We want only rows in which COUNTRY is equal to “CAN”. **Select** “equal to” for the **Condition Type** option.

Now complete the second part of the criterion. We are looking for a specific value, so **set** the **Expression 2 Type** to **Constant**. In the Constant field, either enter CAN or use the magnifying glass to summon a list from which you can select it.

- **Click** the 🔍 in the Expression 2 Define Constant box. The system will prompt for a Country abbreviation.
The system takes you to a “prompt.” The system is prompting for a Country to identify as a single fixed value. We want our results to show the Canadian provinces. Use the to look up Country abbreviations.

- **Place** Country abbreviation in Constant box
- **Click OK**

The system takes you back to the Edit Criteria Properties page.

- **Click OK** to return to the Fields tab

Next, let’s change this to retrieve states of Mexico instead of provinces of Canada.

- **Go** to the **Criteria** tab
- **Click Edit**

The same criteria information will be used for Mexico as was used for Canada.

- **Click OK**
- **Modify** criteria – change **CAN** to **MEX**

- **Save** the query
- **Click the Run tab**
Next, let’s list the states and provinces in North America (Canada, USA, and Mexico). For this we will need three criteria.

- Click **Fields** tab
- Click in the **COUNTRY** row

Observe the Expression 2 column; Canada is placed as the second criteria.

- Add **CAN** as the COUNTRY, again
- Click **OK**
- Click the **Criteria** tab

Look at the Logical column. The word “and” appears in the CAN row. The “and” in the logical column indicates that both criteria must be met for the state to be in the result set.

- Click **Run** tab

There are NO matching rows!
Go back to the Criteria tab

The Criteria as defined are that A.COUNTRY is equal to MEX and A.COUNTRY is equal to CAN. A state can’t be in Canada and Mexico at the same time, so it is impossible to meet both criteria!

Change the “and” in the CAN row to “or”

The “OR” operator is used whenever one of the criteria must be met to include a row in the result set. What we really want are states in which the country is MEX or CAN.

Click the Run tab. Now the results include Canadian provinces and Mexican states!

Now let’s add a criterion that country is equal to USA. This time we will add the criterion from the Criteria tab instead of the Fields tab.

Go to Criteria tab

Click Add Criteria button
Use the **Search** to look up the selected fields for the query.

- **Click** in the Expression 1 Choose Record and Field box
- **Click** A.COUNTRY

When selecting a field, you must click the link corresponding to that field. It may be tempting to click the Cancel button, since it’s a prominent button at the bottom of the screen.

- **Place USA** in the Expression 2 Constant box
- **Click** the OK button

**Remember:** If you do not know the abbreviation click the **Search**. This will take you to the prompt with a **Search** to look up the Country abbreviations.

- **Change** “AND” to “OR”
- **Save** your query
- **Click** the Run tab
There are several different ways you can add criteria when building a query.

1. On the Query tab, click the Use as Criteria button. This takes you to the Edit Criteria Properties page; the field in the same row as the button is automatically selected as Expression 1.
2. On the Fields tab, click the Add Criteria button. This takes you to the Edit Criteria Properties page; the field in the same row as the button is automatically selected as Expression 1.
3. On the Criteria tab, click the Add Criteria button.

4. On the Expressions tab, click the Add Criteria button. This takes you to the Edit Criteria Properties page; Expression 1 Type is set to Expression instead of Field and the expression in the same row as the button is automatically selected as Expression 1. (Expressions are covered later in this class.)
**Question 5A**

Create a new query to list academic plans (majors, minors, etc.). For each plan, show the institution, academic plan, effective date, effective status, description, academic program, degree, and academic career. Limit the results to plans in the College of Arts and Sciences (academic program ARTSC). You need to add one criterion to the query. (A criterion on effective date will be added automatically.) To which field will you apply it?

- Use the **ACAD_PLAN_TBL** record
- Show the following fields:
  - **INSTITUTION**
  - **ACAD_PLAN**
  - **EFFDT**
  - **EFF_STATUS**
  - **DESCR**
  - **ACAD_PROG**
  - **DEGREE**
  - **ACAD_CAREER**
- Add a criterion that the academic program is equal to ARTSC.
- Save and Save often. Save as **TRNG_QM##_Q5A**
**Question 5B**

Obtain a list of laboratory classes that occurred in the Summer 2013 semester. Display the course ID, course offer number, session code, subject, class section, catalog number, academic career, and description fields.

- Use the `CLASS_TBL` record
- Show the following fields:
  - `CRSE_ID`
  - `CRSE_OFFER_NBR`
  - `SESSION_CODE`
  - `CLASS_SECTION`
  - `SUBJECT`
  - `CATALOG_NBR`
  - `ACAD_CAREER`
  - `DESCR`
- A `STRM` value of "2135" is used for the Summer 2013 semester
- An `SSR_COMPONENT` value of "LAB" indicates a laboratory class. There is no list of predetermined values, LAB must be entered manually
- Save and Save often. Save as `TRNG_QM##_Q5B`
**Edit Criteria - Part II**

Not every query will involve matching exact values. Part two will show us when to use other operators besides equal to.

In the last set of exercises, all of the criteria used the Condition Type “equal to” since each of them checked if a field was equal to a particular value. There are many other condition types that you can use, allowing you to create criteria that check if a field is greater than or less than a value or between two values. Here are some useful condition types and their equivalents in mathematical operators:

- **between** - is Expression 1 between two numbers, two words, etc. (inclusive)?
  - low-value <= Expr1 <= high value
- **equal to** - is Expression 1 equal to Expression 2?
  - Expr1 = Expr2
- **greater than** - is Expression 1 greater than Expression 2?
  - Expr1 > Expr2
- **less than** - is Expression 1 less than Expression 2?
  - Expr1 < Expr2
- **not between** - is Expression 1 outside a range given by two values?
  - Expr1 < low value or Expr1 > high value
- **not equal to** - is Expression 1 not equal to Expression 2?
  - Expr1 <> Expr2
- **not greater than** - is Expression 1 less than or equal to Expression 2? Expr1 <= Expr2
- **not less than** - is Expression 1 greater than or equal to Expression 2?
  - Expr1 >= Expr2

Note that the common concept of “greater than or equal to” is defined in PeopleSoft Query as “not less than” and that “less than or equal to” is defined as “not greater than.”

Create a new query to list all classes being held in Spring 2015 with a waitlist capacity of 40 or more.

- Start a query on **CLASS_TBL**
- Select the **SUBJECT, CATALOG_NBR, CLASS_SECTION, DESCR, and WAIT_CAP** fields
- Click the **Use as Criteria** button in the **STRM** row
- Set Condition Type to **equal to**
- Set constant to **2152**
- Click OK
- Click the **Use as Criteria** button in the **WAIT_CAP** row
- Set Condition Type to **greater than**
- Set constant to **40**
- Click OK
- Click the **Criteria** tab to review the criteria
Save the query as TRNG_QM##_E6A
Click the Run tab.

Are these the results we are looking for? We want to result to yield classes with waitlist capacities of 40 or more. We chose “greater than” for the condition; this means classes with a waitlist capacity of exactly 40 will be missed. We need to change the condition so that it is effectively “greater than or equal to,” which in Query Manager is referred to as “not less than.”

Click the Criteria tab
Click Edit in the WAIT_CAP row

Change Condition Type to “not less than”

Click OK.

Save the query
Click Run tab
Now classes with a waitlist capacity of exactly 40 are included in the results
<table>
<thead>
<tr>
<th>Section</th>
<th>Subject</th>
<th>Catalog</th>
<th>Course Title</th>
<th>Wait</th>
<th>Cap</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>BIOL</td>
<td>1040</td>
<td>Intro To Biology</td>
<td>40</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>PHIL</td>
<td>1010</td>
<td>Intro To Philosophy</td>
<td>60</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>PHIL</td>
<td>1020</td>
<td>Ethics</td>
<td>60</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>BIOL</td>
<td>1010</td>
<td>Environment Of Life</td>
<td>60</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>MUCT</td>
<td>1010</td>
<td>Exploring Music</td>
<td>40</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>BIOL</td>
<td>1040</td>
<td>Intro To Biology</td>
<td>40</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>BIOL</td>
<td>1040</td>
<td>Intro To Biology</td>
<td>40</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>BIOL</td>
<td>3310</td>
<td>Human Anat/Phys I</td>
<td>50</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>ATH</td>
<td>2100</td>
<td>Care Prev Spots Injuries</td>
<td>40</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>ARTH</td>
<td>1450</td>
<td>Western Art</td>
<td>50</td>
<td></td>
</tr>
</tbody>
</table>
Using Wildcards

There are many uses for wildcards (see Supplemental Material); use “student%” to find anything starting with student and use “%student” to find anything ending with student. Let’s obtain a list of item types that contain “Fine” in the description. We will use the wildcard symbol (%) in front of and behind the word Fine.

- Start a query on ITEM_TYPE_TBL
- Select fields:
  - SETID
  - ITEM_TYPE
  - EFFDT
  - EFF_STATUS
  - DESCR
- Save and save often. Save as TRNG_QM##_E6B

- Click Fields tab
- Click % in the DESCR row
- Set Condition Type to “like”
- Place %Fine% in the Constant box
- Click OK
- Save the query
- Click the Run tab

Important note: “like” is case sensitive! If you change “Fine” to “fine” you’ll get NO results. Refer to the “Criteria and Case Sensitive Data” Segment of the Supplemental Material.
**Question 6**

For this question, find the term codes, description, and term beginning date for a specified amount of time. Create a new query to get the term code, description, and term beginning date for all fall terms from 2010 through 2019 for undergraduate students. Use the term description to determine whether or not a term is in the fall.

- Use the `TERM_TBL` record
- Show the following fields:
  - `STRM`
  - `DESCR`
  - `TERM_BEGIN_DT`
- An `ACAD_CAREER` value of “UGRD” indicates undergraduate
- Add Criteria to `TERM_BEGIN_DT` for terms starting between 01/01/2010 and 12/31/2019
- How will you set up a criterion to look for “Fall” at the beginning of the term description? Examine the `DESCR` field to determine if a term is in the Fall semester. What condition type will you use? Where will you place wildcards?
- Save and save often. Save as `TRNG_QM##_Q6`

![Term Code Table](image-url)
Working with Prompts

Prompts allow a user to enter different values into a query each time it runs. For instance, you might look for transactions over $100 in one instance and transactions over $2000 in another instance. Rather than writing two different queries, you can write one query that prompts for a value. This makes the query more flexible and reduces duplicate queries that do almost the same thing.

Let’s create a query that lists all the states belonging to a country that the user selects at run time.

1. Use the following record **STATE_TBL**
2. Click the **Check All** button
3. Save and save often. Save as **TRNG_QM##.E7**
4. Click in the **COUNTRY** row
5. Set Choose Expression 2 Type to “Prompt”
6. Click **New Prompt** link for Expression 2

If the prompt is based on an existing field (COUNTRY), the prompt’s characteristics (such as Type, Format, and Length) default to those of the field. In the case of COUNTRY, Type is already set to Character, Format to Upper, and Length to 3).

A prompt can restrict legal values to those from a table, a list of translate values, or Boolean values (yes and no). The Edit Type field is used for this. Observe the **Edit Type** is set to **Prompt Table**. Prompt Table is set to **COUNTRY_TBL**. This means the user can only select a country known to the system.

1. Click **OK** to return to the Edit Criteria Properties dialog
2. Click **OK** to return to the Query tab

The Expression 2 prompt is set to :1. Upon running the query :1 will be replaced by the value entered by the user (a country code). To see this in the criterion **click** the **Criteria** tab.
o **Click** the Prompts tab to confirm the addition of the new prompt

![Prompt Window](image.jpg)

o **Save** the query
o **Click Run** tab

A prompt window appears in which you are asked to enter a country.

o **Enter JPN**
o **Click OK**

![Prompt Values](image2.jpg)

It is not possible to set a value to appear by default in the prompt window. Numeric prompts always default to zero and text and date prompts always default to blank.

If you create a prompt but do not reference it in an expression or in criteria, the prompt will not appear when you run the query.
**Question 7**

Let’s use an existing query to find classes with a particular component in Summer 2013. Modify the query you wrote earlier to display the laboratory sections offered in Summer 2013 so that it prompts the user for a component. This question offers a great learning opportunity on editing criteria and adding criteria.

Run the query twice. First, enter a component of LAB to demonstrate that the query still returns the same results as before adding the prompt. Second, enter a component of LEC to test it out with a different component.

- Use existing query **TRNG_QM##_Q5B**
- Save the query with a new name **TRNG_QM##_Q7**
- Change the criterion that **SSR_COMPONENT** is equal to LAB to be equal to a prompt
- Run query using component **LAB**
- Rerun query using component **LEC** (lecture)
- Save again when finished

![Component = LAB](image1)

![Component = LEC](image2)
Writing Expressions

Expressions are ways to perform calculations and conversions and to edit or combine text strings. They can be displayed as a column in the result set and can be used in criteria.

Expressions are comprised of fields, values, operators, and functions. Operators generally perform a calculation on the fields and values preceding and following them. For instance, in “5 + 3”, the “+” is the operator which performs a calculation on 5 and 3 to produce 8. Functions take a set of input values and produce an output value. The square root function is SQRT; SQRT (4) in an expression has a result of 2.

For the next exercise, we’ll perform a “what-if” analysis. We’ll examine what would happen if Student Financial Aid offers 2.5% more aid for each award type and if this exceeds the current budget for each award type. This would help us determine if more money needs to be allocated for that award.

Our analysis will be for the 2014 Aid Year and will only consider awards for which aid was offered in that year.

- Start a query on ITEM_TYPE_FISCL
- Select fields:
  - ITEM_TYPE
  - MAX_OFR_BUDGT
  - OFR_GROSS
- Click the Use as Criteria button for SETID.
- Set the Constant to ‘BGSUN’.
- Click the OK button.
- Click the Use as Criteria button for AID_YEAR.
- Set the Constant to 2014.
- Click the OK button.
- Click the Use as Criteria button for OFR_GROSS.
- Set the Condition Type to “greater than”.
- Set the Constant to 0.
- Click the OK button.
- Save and save often. Save as TRNG_QM##_E8A
- Click the Run tab to make sure the query runs

<table>
<thead>
<tr>
<th>Item Type</th>
<th>Budgeted</th>
<th>Gross</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>200000.00</td>
<td>552.50</td>
</tr>
<tr>
<td>2</td>
<td>200000.00</td>
<td>60771.00</td>
</tr>
<tr>
<td>3</td>
<td>200000.00</td>
<td>60771.00</td>
</tr>
<tr>
<td>4</td>
<td>200000.00</td>
<td>60771.00</td>
</tr>
<tr>
<td>5</td>
<td>200000.00</td>
<td>60771.00</td>
</tr>
<tr>
<td>6</td>
<td>200000.00</td>
<td>60771.00</td>
</tr>
<tr>
<td>7</td>
<td>200000.00</td>
<td>60771.00</td>
</tr>
<tr>
<td>8</td>
<td>200000.00</td>
<td>60771.00</td>
</tr>
<tr>
<td>9</td>
<td>200000.00</td>
<td>60771.00</td>
</tr>
<tr>
<td>10</td>
<td>200000.00</td>
<td>60771.00</td>
</tr>
</tbody>
</table>

- Click Expressions tab
- Click Add Expression button
We must choose the type of expression, indicating what sort of result it gives. Will the result be a string (character), a number, a date, or something else? In this case, select **Signed Number** from the Expression Type drop-down.

We must also set the size of the result. If the result was a string, the length would be the maximum number of characters in the string. For a number, we set the number of length in digits, including both before and after the decimal point. We also separately specify the number of digits after the decimal.

- **Set** Expression Type to **Signed Number**
- **Set Length** to **20**
- **Set Decimals** to **3** (this means there will be 17 digits before the decimal)
- **Click** the **Add Field** link and select **A.OFR_GROSS**
- **In The Expression Text box**, enter “**1.025**” (without the quotes) after A.OFR_GROSS (the finished expression should be “A.OFR_GROSS * 1.025”)
- **Click** **OK**

You are returned to the Expressions tab. The new expression is displayed in the Expressions List.
The *Use as Field* link is used when the expression is going to be used as a field. We want to create a column that provides us the gross offer amount including the 2.5% increase.

- **Click** the *Use as Field* link in the row for A.OFR_GROSS * 1.025

The expression A.OFR_GROSS * 1.025 appears on the Fields page.

![Fields page](image)

Note that the heading of the column containing the expression defaults to the expression itself. It’s best to change this to a meaningful column heading.

- **Click** the *Edit* button in the A.OFR_GROSS * 1.025 row
- **Observe** that the heading type is *Text*; this is needed to allow a custom column heading
- **Change** the Heading Text to “**Increased Offer**”
- **Click OK**
- **Save** the query
- **Click Run** tab

![Edit Field Properties](image)

Now we want to see all the awards that will have a higher gross offer amount than budgeted amount *after* the increase to the offers.

![View All](image)
We’ll use the current query as a starting point. Click the **Criteria** tab. Click the **Save As** link. Set the new name of the query to **TRNG_QM##_E8B** then click the **OK** button.

Next, perform the following steps:

- **Click** the **Add Criteria** button
- **Set** Expression 1 Type to **Expression**
- **Click** in Expression 1, Define Expression box
- **Click** **A.OFR_GROSS * 1.025**
- **Set** Condition Type to “greater than”
- **Set** Expression 2 Type to **Field**
- **Click** in Expression 2, Choose Record and Field box
- **Select** **A.MAX_OFR_BUDGET**
- **Click** **OK**

You should see a new criterion for **A.OFR_GROSS * 1.025** being greater than **A.MAX_OFR_BUDGT** listed on the Criteria tab.

This exercise has shown that you can not only show the results of calculations in the rows returned by the query, you can also use the results of calculations to control which rows are returned.

- **Save** your query
- **Click** **Run** tab
Let’s return to the query you wrote to list the Fall terms starting between 2010 and 2019. Search for and edit the TRNG_QM##_Q6 query. Save this query as TRNG_QM##_E8C.

The convention used at BGSU is that the last digit of a value in the STRM field indicates the semester in which the term occurs – 2 for Spring, 5 for Summer, and 8 for Fall.

It is preferable to use values that are consistent and defined, such as the last digit of the term, rather than a description or name. Such strings are more likely to have misspellings or issues of case, either in the data or in your criteria. For instance, you might accidentally use “fall” or “faLL” instead of “Fall” in your criteria on DESCR of the University might decide to replace “Fall” with “Autumn,” so your query would no longer find any rows.

We will replace the criterion that uses the DESCR field with one that examines the STRM field. However, we cannot use STRM as itself in the criterion, as only the last digit indicates the semester. We will build an expression that uses the SUBSTR (substring) function to extract part of the term then use this expression in the criterion. Click the Criteria tab.
Click the button in the criteria row for **DESCR**

Click the **Expressions** tab

Click the **Add Expression** button

Set **Expression Type** to “character” and **Length** to 1. The semester indicator is only one character long

In the **Expression Text** area, enter the following: `SUBSTR(A.STRM,4,1)`. This means the expression will be the substring of **STRM** that starts with the fourth character and has a length of one character.

Click the **OK** button

After clicking the **OK** button, you are returned to the **Expressions** tab. The new expression will be in the **Expressions list**.

We need to restrict the results to only Fall terms; we are doing this through a criterion that the fourth digit of the term code is 8. To do this, we need to add a criterion that uses the new expression.
In the row of the Expressions List for the **SUBSTR** expression, click the **Add Criteria** button.

The Edit Criteria Properties screen appears. Observe that the **SUBSTR** expression has already been selected.

- The Condition Type is already “equal to.” In the **Constant** field for Expression 2, enter 8.
- Click the **OK** button.

Now let’s see how the new criterion looks. Click the **Criteria** tab. Observe that the bottom row shows “**SUBSTR(A.STRM, 4, 1) equal to 8**”.

- **Save** your changes.
- **Click** the **Run** tab to view the results. They should be the same as in the original query.
Question 8

Obtain a list of undergraduate students active in Fall 2012 at BGSU. Show each student’s ID number, primary academic program, and academic load in units. In addition, add a “Load Type” column in the results that has a value based on the academic load as follows:

- If academic load is “F,” the load type is “FULL”
- If academic load is “T,” the load type is “3QTR”
- If academic load is “H,” the load type is “HALF”
- If academic load is anything other than F or H the load type is “LESS”

Sort the results by student number in the left-most column.

- Use the STDNT_CAR_TERM record
- Display the EMPLID, ACAD_PROG_PRIMARY, and ACADEMIC_LOAD fields
- An undergraduate is indicated by an ACAD_CAREER value of ‘UGRD’
- Make sure to include a criterion that INSTITUTION is equal to ‘BGSUN’
- The value of STRM for Fall 2012 is ‘2128’
- A student active in a term is one that has not withdrawn; WITHDRAW_CODE should be equal to ‘NWD’
- Create an expression that uses the value of ACADEMIC_LOAD to display either ‘FULL’, ‘3QTR’, ‘HALF’, or ‘LESS’. Use CASE to generate the value to be displayed. Consult the Supplemental Material for how to use CASE.
- Use the expression you created as a field. Give the field a heading of “Load Type”
- Save the query as TRNG_QM##_Q8
Using a Prompt with Expressions

It is possible to use a prompt inside an expression, allowing the user great flexibility in determining what calculations are performed. We will use the previous query on salary grades and change this query to accept a percentage.

- Find the existing query TRNG_QM##_E8B
- Save as TRNG_QM##_E9
- Click the Prompts tab
- Click the Add Prompt button
- Set Type to Number
- Set Heading Text to Increase (%)
- Set Length to “5”
- Set Decimals to “2”
- Click OK

Recall from the Working with Prompts lesson that the prompt indicator :1 will be replaced with the value that you enter when the query is run. When you run the query and enter “5.00” for the increase amount, the expression becomes A.OFR_GROSS * (1 + 5.00 / 100), which evaluates to A.OFR_GROSS * 1.05. By including a prompt in the expression, you can enter a different increase amount each time the query is run.
- **Save** the query
- **Click** the Run tab
- **Enter** 5.00 for the increase percentage

```
<table>
<thead>
<tr>
<th>Item Type</th>
<th>Budgeted</th>
<th>Gross</th>
<th>Increased Gross</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>220900.00</td>
<td>228324.500</td>
<td>236740.725</td>
</tr>
<tr>
<td>2</td>
<td>2084.00</td>
<td>3084.00</td>
<td>3238.200</td>
</tr>
<tr>
<td>3</td>
<td>1607.00</td>
<td>3000.00</td>
<td>3150.000</td>
</tr>
<tr>
<td>4</td>
<td>2080.00</td>
<td>2000.00</td>
<td>2100.000</td>
</tr>
<tr>
<td>5</td>
<td>1094.00</td>
<td>1000.00</td>
<td>1050.000</td>
</tr>
<tr>
<td>6</td>
<td>1250.00</td>
<td>1250.00</td>
<td>1312.500</td>
</tr>
<tr>
<td>7</td>
<td>3300.00</td>
<td>3000.00</td>
<td>3150.000</td>
</tr>
<tr>
<td>8</td>
<td>20900.00</td>
<td>20000.00</td>
<td>21000.000</td>
</tr>
<tr>
<td>9</td>
<td>757.00</td>
<td>750.00</td>
<td>787.500</td>
</tr>
</tbody>
</table>
```
Use Aggregate Functions

All queries built so far operate on one row at a time. It is common to want to collect information on groups of rows, such as a count, a sum, a minimum, a maximum, and an average. There are aggregate functions that are used to compile this information on either all rows in a query or broken down by certain fields. Let’s get a count of academic plans.

- Start a query on **ACAD_PLAN_TBL**
- **Select** INSTITUTION
- **Click** Fields tab
- **Edit** INSTITUTION row
- **Set** Aggregate to **Count**

The Aggregate column now shows Count.

- **Click** OK
- **Save** and Save often. Save as **TRNG_QM##_E10A**

Once the aggregate function has been placed at Count, in the row for INSTITUTION, the Agg (Aggregate) column shows Count and the Heading Text on the Fields page shows Count Institution.

- **Click** Run tab

Note that this is not a count of institutions; it is a count of all academic plans. The heading is Count Institution because we set the aggregate function on the Institution field.

Let’s get a count of plans per program.

- **Click** Query tab
- **Save** the query as **TRNG_QM##_E10B**
- **Select** the **ACAD_PROG** field for display
- **Click** Fields tab

The two fields that will be displayed are the count and ACAD_PROG. By convention counts are on the right-hand side of the results, so move ACAD_PROG to the left.

- **Click** the **Reorder / Sort** button
- **Move** ACAD_PROG to the left-most column
- **Enter** the value 1 in the **New Column** field for A.ACAD_PROG
An important consideration when using aggregate functions is how the rows are grouped together. As seen in this example, you can obtain a count of all rows returned by a query and you can obtain a count of rows within groups by values in a field such as **ACAD_PROG**.

When you have a field that has an aggregate function, the rows are grouped by *all* fields being displayed that are *not* being aggregated.

In the first part of the exercise, the only field being selected was **INSTITUTION**, and this had the Count aggregate applied to it. Since there were no fields that did not have an aggregate, the only group was all rows returned by the query. Therefore, the result was the count of plans returned by the query.

In the second part of the exercise, both **ACAD_PROG** and **INSTITUTION** were selected. Only **INSTITUTION** had an aggregate function. **ACAD_PROG** did not have an aggregate, so the rows were grouped by **ACAD_PROG**. Therefore, the result was the count of plans *for each program*. 

<table>
<thead>
<tr>
<th>Acct Prog</th>
<th>Count Institution</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>235</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>277</td>
</tr>
<tr>
<td>4</td>
<td>152</td>
</tr>
<tr>
<td>5</td>
<td>30</td>
</tr>
<tr>
<td>6</td>
<td>31</td>
</tr>
<tr>
<td>7</td>
<td>209</td>
</tr>
<tr>
<td>8</td>
<td>114</td>
</tr>
<tr>
<td>9</td>
<td>46</td>
</tr>
<tr>
<td>10</td>
<td>30</td>
</tr>
</tbody>
</table>
The Count, Min, and Max aggregate functions can be applied to any field. For numeric fields, Min and Max return the lowest and highest numeric values in the group, respectively. For text fields, they return the earliest and latest values in alphabetical order in the group, respectively. (When dealing with mixed-case data, uppercase is considered to be "less than" lowercase and shorter strings are "less than" longer strings.) The Sum and Average functions can only be used on numeric fields.

If you want to have criteria that use the results of an aggregate, such as restricting the results to those rows with a count greater then 50, you must add these in the Having tab instead of the Criteria tab. (If you click the button for a field that has an aggregate function, you will automatically add a having criterion instead of a regular criterion.)

You can use aggregate functions in expressions. The names of the functions are SUM, COUNT, MIN, MAX, and AVG; each of these accepts a field name as an argument. You must check the Aggregate Function checkbox in the Edit Expressions Properties page to use one of these functions in an expression.
**Question 10**

Find the average pay rate of active, administrative, salaried employees by department. Start a new query on JOB. Select department ID and annual rate. Change the aggregate function on annual rate to average. Add criteria that employee status is equal to A (active), full part time is equal to F (full-time), employee type is equal to S (salaried), and salary administration plan is ADM (administration).

- Use the JOB record
- Show fields for **Department ID** and **Annual Rate**
- Obtain an average on **Annual Rate**
- Add Criteria:
  - **EMPL_STATUS** is equal to A
  - **FULL_PART_TIME** is equal to F
  - **EMPL_TYPE** is equal to S
  - **SAL_ADMIN_PLAN** is ADM

Save and save often. Save as **TRNG_QM##_Q10A**

Next modify the query to find the average salary for individual job codes within each department.

- Display the **Job Code** field
- Move **Job Code** to be after the department ID but before the average annual rate

Save this query as **TRNG_QM##_Q10B**.
Joins – Introduction

Most queries that you will write will involve retrieving information from several records. The PeopleSoft data is stored in a relational database; this means information about something is recorded in many different tables, with each table holding data only about one aspect of that thing. For instance, a student has names, addresses, and classes; the names are stored in one table, the addresses in another table, and the class schedule in a third table. This is done to better organize the data and to reduce duplication, such as by ensuring names are all in one table and making it unnecessary to replicate those names in other tables.

Each table in a relational database has a primary key, which is one or more fields that uniquely identify each row in that table. For example, a person in PeopleSoft is uniquely identified by his or her EMPLID, since no two people can have the same EMPLID. Tables that contain data that belong to a person or relate to a person will therefore have an EMPLID field. Since a person can have several addresses, such as a home address and a work address, the addresses table might have a primary key of both the EMPLID and the address type (home or work) to uniquely identify an address. (Fields that are in the primary key are shown with a key icon next to them on the Query tab.)

To use information from several tables in your query, you must join tables together so that data related to the same thing is presented in the same row in the results. If you want to show a student’s name alongside that student’s address, you would join the names and addresses tables together. You must specify in your query which fields to use to relate the tables to each other. In the case, you would join on the EMPLID field, since the student ID is what uniquely identifies a student; rows in the names table and rows in the addresses table that have the same EMPLID belong to the same student.

There are two main types of joins: an inner join, in which only rows in which there is matching data in both tables are returned, and an outer join, in which rows from one table can still be returned even if there is not a match in the other table. Outer joins are discussed later in this course.

An example of an inner join will best demonstrate how a join operates.

Consider these excerpts from the countries table (COUNTRY_TBL) and states table (STATE_TBL).

<table>
<thead>
<tr>
<th>COUNTRY_TBL (A)</th>
<th>STATE_TBL (B)</th>
</tr>
</thead>
<tbody>
<tr>
<td>COUNTRY</td>
<td>DESCRIPT</td>
</tr>
<tr>
<td>CAN</td>
<td>Canada</td>
</tr>
<tr>
<td>MEX</td>
<td>Mexico</td>
</tr>
<tr>
<td>USA</td>
<td>United States</td>
</tr>
<tr>
<td>COUNTRY</td>
<td>STATE</td>
</tr>
<tr>
<td>CAN</td>
<td>AB</td>
</tr>
<tr>
<td>CAN</td>
<td>BC</td>
</tr>
<tr>
<td>CAN</td>
<td>ON</td>
</tr>
<tr>
<td>CAN</td>
<td>QC</td>
</tr>
<tr>
<td>MEX</td>
<td>DG</td>
</tr>
<tr>
<td>MEX</td>
<td>JA</td>
</tr>
<tr>
<td>MEX</td>
<td>YU</td>
</tr>
<tr>
<td>USA</td>
<td>IN</td>
</tr>
<tr>
<td>USA</td>
<td>MI</td>
</tr>
<tr>
<td>USA</td>
<td>OH</td>
</tr>
</tbody>
</table>

They will be joined together so the countries and the states and provinces within them can be listed in query results. They will be joined on the COUNTRY field since this uniquely identifies a country. For instance, a row in STATE_TBL with a COUNTRY equal to “USA” represents a state in the United States.
When an inner join takes place, every row in the left table is compared against every row in the right table. When the values in the join field or fields match, the rows are combined and added as one row to the results. When the values in the join field or fields do not match, no row is added to the results.

In this example, the first comparison is of the Canada row in COUNTRY_TBL and the Alberta row in STATE_TBL. Since the join field COUNTRY has the same value CAN in both tables, a row combining the Canada row with the Alberta row is added to the results.

```
<table>
<thead>
<tr>
<th>COUNTRY_TBL (A)</th>
<th>STATE_TBL (B)</th>
<th>RESULTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>CAN Canada</td>
<td>CAN AB Alberta</td>
<td></td>
</tr>
<tr>
<td>MEX Mexico</td>
<td>CAN BC British Columbia</td>
<td></td>
</tr>
<tr>
<td>USA United States</td>
<td>CAN ON Ontario</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CAN QC Quebec</td>
<td></td>
</tr>
<tr>
<td></td>
<td>MEX DG Durango</td>
<td></td>
</tr>
<tr>
<td></td>
<td>MEX YUC Yucatan</td>
<td></td>
</tr>
<tr>
<td></td>
<td>USA IN Indiana</td>
<td></td>
</tr>
<tr>
<td></td>
<td>USA MI Michigan</td>
<td></td>
</tr>
<tr>
<td></td>
<td>USA OH Ohio</td>
<td></td>
</tr>
</tbody>
</table>
```

Next, the Canada row in COUNTRY_TBL is compared against the British Columbia row in STATE_TBL. Since the join field is COUNTRY and the COUNTRY field in both COUNTRY_TBL and STATE_TBL has the value CAN, there is a match. A new row is added to the results with a combination of the Canada row and the British Columbia row.

```
<table>
<thead>
<tr>
<th>COUNTRY_TBL (A)</th>
<th>STATE_TBL (B)</th>
<th>RESULTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>CAN Canada</td>
<td>CAN AB Alberta</td>
<td></td>
</tr>
<tr>
<td>MEX Mexico</td>
<td>CAN BC British Columbia</td>
<td></td>
</tr>
<tr>
<td>USA United States</td>
<td>CAN ON Ontario</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CAN QC Quebec</td>
<td></td>
</tr>
<tr>
<td></td>
<td>MEX DG Durango</td>
<td></td>
</tr>
<tr>
<td></td>
<td>MEX YUC Yucatan</td>
<td></td>
</tr>
<tr>
<td></td>
<td>USA IN Indiana</td>
<td></td>
</tr>
<tr>
<td></td>
<td>USA MI Michigan</td>
<td></td>
</tr>
<tr>
<td></td>
<td>USA OH Ohio</td>
<td></td>
</tr>
</tbody>
</table>
```

The comparisons continue with the Ontario and Quebec rows in STATE_TBL. These are both combined with Canada and added to the results.

```
<table>
<thead>
<tr>
<th>COUNTRY_TBL (A)</th>
<th>STATE_TBL (B)</th>
<th>RESULTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>CAN Canada</td>
<td>CAN AB Alberta</td>
<td></td>
</tr>
<tr>
<td>MEX Mexico</td>
<td>CAN BC British Columbia</td>
<td></td>
</tr>
<tr>
<td>USA United States</td>
<td>CAN ON Ontario</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CAN QC Quebec</td>
<td></td>
</tr>
<tr>
<td></td>
<td>MEX DG Durango</td>
<td></td>
</tr>
<tr>
<td></td>
<td>MEX YUC Yucatan</td>
<td></td>
</tr>
<tr>
<td></td>
<td>USA IN Indiana</td>
<td></td>
</tr>
<tr>
<td></td>
<td>USA MI Michigan</td>
<td></td>
</tr>
<tr>
<td></td>
<td>USA OH Ohio</td>
<td></td>
</tr>
</tbody>
</table>
```

The next comparison is between the Canada row in COUNTRY_TBL and the Durango row in STATE_TBL. Since COUNTRY is CAN in the row of COUNTRY_TBL and MEX in the row of STATE_TBL, there is no match; nothing is added to the results.
The same occurs as the Canada row in COUNTRY_TBL is compared with the remaining rows in STATE_TBL for Mexico and the United States. Since the COUNTRY fields do not have matching values, no more rows are added to the results.

After the Canada row has been compared against all the rows of STATE_TBL, processing moves on to the Mexico row, which will also be compared against every row in STATE_TBL. In the first comparison, the Mexico row is compared to the Alberta row. Since the COUNTRY values differ, nothing is added to the results.

The Mexico row is then compared against the British Columbia, Ontario, and Quebec rows, with no matches on the COUNTRY field. Next, the Mexico row is compared to the Durango row; since COUNTRY is MEX in both the row from COUNTRY_TBL and the row from STATE_TBL, a new row is added to the results, combining data from COUNTRY_TBL and STATE_TBL.

Jalisco and Yucatan are checked next and both are added to the results since there is a match on the COUNTRY value of MEX. When the Mexico row is then compared to the Indiana row, since MEX does not equal USA, Indiana is not added to the results.
After the Mexico row of COUNTRY_TBL has been compared to every row in STATE_TBL, processing continues with the United States row of COUNTRY_TBL. The United States row is first compared to the Alberta row of STATES_TBL; since USA and CAN do not match, nothing is added to the results.

Scanning continues through the rows of STATE_TBL. No matches are found among all the rows in STATE_TBL in which COUNTRY is CAN or MEX. Once the processing reaches the Indiana row in STATE_TBL, the COUNTRY value is USA in the rows being compared in both COUNTRY_TBL and STATE_TBL, so a row is added to the results.

Michigan and Ohio are added to the results as well since COUNTRY is USA in the rows of STATE_TBL for those states.

At the end, the results contain every row from COUNTRY_TBL combined with each row in STATE_TBL in which the COUNTRY fields of COUNTRY_TBL and STATE_TBL match each other.
Criteria can be added to further restrict the results if desired. For instance, one could add a condition that the name of the state contains “an”, expressed as B.STATE is like “%an%”. The results would then be the following:

### RESULTS

<table>
<thead>
<tr>
<th>A_COUNTRY</th>
<th>A_DESCR</th>
<th>B_COUNTRY</th>
<th>B_STATE</th>
<th>B_DESCR</th>
</tr>
</thead>
<tbody>
<tr>
<td>MEX</td>
<td>Mexico</td>
<td>MEX</td>
<td>Durango</td>
<td></td>
</tr>
<tr>
<td>USA</td>
<td>United States</td>
<td>USA</td>
<td>IN</td>
<td>Indiana</td>
</tr>
<tr>
<td>USA</td>
<td>United States</td>
<td>USA</td>
<td>MI</td>
<td>Michigan</td>
</tr>
</tbody>
</table>

In PeopleSoft Query, inner joins are further classified as one of three types:

- **Record Hierarchy** – all data in one table represents details about or children of data in another table
- **Related Record** – data in a table gives more information, such as descriptions and meanings about something referenced in another table
- **Any Record** – manually join two tables on any fields

The distinctions between these types and how to construct inner joins are discussed in the following sections.
Joins – Record Hierarchy

The hierarchy join is used to get information about something at two or more levels; this is sometimes referred to as a master/detail or parent/child relationship. One example of this is an invoice; the invoice itself is one level and the line items are another level.

For the next exercise, we will get a list of currently defined service indicators (holds). These have codes (in SRVC_IND_CD_TBL) and reasons (in SRVC_IN_RSN_TBL). If we want to show the descriptions of both the code and the reason, we need to join the tables. For a parent/child relationship, the join is on all the key fields (* = key field).

- Start a query on SRVC_IND_CD_TBL (this table provides codes)
- Select fields:
  - INSTITUTION*
  - SRVC_IND_CD*
  - EFFDT*
  - EFF_STATUS
  - DESCR
- Save and save often. Save as TRNG_QM##_E11
- Click Hierarchy Join link

The hierarchy join window shows the parent/child relationship. The relationship is joined on all key fields of the parent table. The primary keys in the parent table are INSTITUTION, SRVC_IND_CD, and EFFDT. The primary key of the child table, SRVC_IN_RSN_TBL, contains all fields of the primary key of its parent, as well as the reason code, SRVC_IND_REASON. The “Select record for hierarchy join” window allows you to select a child record and automatically add the criteria for the join.

- Click SRVC_IN_RSN_TBL (this record provides reasons).
- Show fields:
  - INSTITUTION
  - SRVC_IND_CD
  - EFFDT
  - SRVC_IND_REASON
  - DESCR
Notice that there is an additional record that has been added to the Chosen Records section. The second alias record is `B.SRVC_IN_RSN_TBL – Service Indicator Reason joined with A`.

- Save the query.
- Click Run tab

Notice the Institution, Srv Ind Cd and Eff Date fields are the same; the Descr fields are different because the first Descr field is for codes (A. table) and the second Descr field is for reason (B. table).

- Click Query tab
- Remove fields from `SRVC_IN_RSN_TBL`:
  - INSTITUTION
  - SRVC_IND_CD
  - EFFDT

- Save the query.
- Click Run tab
The results are the same but the redundant fields are gone.

Let’s limit the results to only the REG (Registration Hold) codes and reasons.

- Click the **Fields** tab
- Click 🏷️ in the **INSTITUTION** row
- Enter “BGSUN” in for the Constant
- Click OK to finish the new criterion
- Click 🏷️ in the **SRVC_IND_CD** row
- Enter “REG” in for the Constant
- Click OK to finish the new criterion

- Save the query
- Click the **Run** tab
Criteria can still be added even when there are multiple records in a query. What about sorting fields and rows? Yes, fields can still be reordered/sorted using the Fields tab. The fields from both A. record and B. record appear on the Fields page.

Let’s sort the list by the description of the reason.

- **Click Fields** tab
- **Click** the **Reorder / Sort** button
- **Set New Order By** to 1 for **B.DESC**. (A is the alias for SRVC_IND_CD_TBL and B is the alias for SRVC_IN_RSN_TBL. Since we want the description of the reason, we choose B.DESC instead of A.DESC.)
- **Click OK**
- **Save** the query
- **Click Run** tab
Question 11

Obtain a list of students who earned degrees with honors using the ACAD_DEGR record. Show the field employee ID, degree, academic career, completion term, and degree conferred date. Join the record with ACAD_DEGR_HONS and show the honors code and honors award date. Limit the results to those conferred on or after 07/01/2012 and order the results by degree first and ascending and EMPLID second.

- Use the ACAD_DEGR record
- Show Fields for:
  - EMPLID
  - DEGREE
  - ACAD_CAREER
  - COMPLETION_TERM
  - DEGR_CONFER_DT
- Join with ACAD_DEGR_HONS
- Show Fields:
  - HONORS_CODE
  - HONORS_AWARD_DT
- Set criteria that degree conferred date is on or after 07/01/2012
- Show the results sorted first by Degree and next by EMPLID, both in ascending order
- Display the results with Degree as the first column and EMPLID as the second column
- Save and save often. Save as TRNG_QM##_Q11
Joins – Related Record

This type of join is used to get information from a prompt table (sometimes called setup table). In a relational database, duplication of information is generally avoided. For instance, data on academic program is stored in one row of ACAD_PROG_TBL, with a unique identifier (ACAD_PROG) used to refer to it. Other records that make use of academic programs will have an ACAD_PROG field but not the description, academic group, and grading scheme fields. ACAD_PROG is the “key” used to get data from the other table.

Let’s get a list of classes and their instructors. When showing the term, rather than showing the abbreviation (2128), show the description (Fall 2012).

- Use the following record CLASS_INSTR
- Show the following fields:
  - CRSE_ID
  - CRSE_OFFER_NBR
  - STRM
  - SESSION_CODE
  - CLASS_SECTION
  - EMPLID
  - INSTR_ROLE
- Add criteria that INSTR_ROLE must be PI (primary instructor) or SI (secondary instructor)
- Save and save often. Save as TRNG_QM##_E12

**NOTICE:** There is not a Condition Type of “or”. The “in list” condition type is used when the value in the selected record field matches one of the comparison values in a list. The list we will use is PI and SI.

- From the Condition Type drop down select “in list”
- Click in Expression 2 box
Click Add Value button in the PI row
Click Add Value button in the SI row
Click OK to finish adding values
Click OK to complete the criterion

Now we will join a table containing descriptions of the terms so that the results can show the description instead of the abbreviation.

Click Query tab
Click “Join TERM_TBL – Term Definition Table” link

The “Select join type” dialog box appears. There are two options: “Join to filter and get additional fields (Standard Join)” and “Join to get additional fields only (Left outer join)”. We want to filter and get additional fields (the system defaults to a Standard Join).

Click OK (note that the OK button is the rightmost button instead of the leftmost)

Next, choose the fields from the joined table to display in the results. Select the DESCR field from B.TERM_TBL- Term Definition Table joined with A.STRM – Term.
The results seem to have duplicated, why? Terms are defined separately for undergraduate and graduate careers in PeopleSoft, though BGSU does not have different schedules for each career. To eliminate the duplicates, restrict the results to only the terms for undergraduate careers. (This will not restrict the results to undergraduate classes.)

- **Add criteria** that **ACAD_CAREER** is **UGRD**. (Which record is **ACAD_CAREER** in?)
- **Save** and **save often**

Now that the term description is included, there is no need for the **STRM** field.

- **Click Fields** tab
  - **Click** [ ] in the **A.STRM** row
  - **Click** Edit next to **B.DESCR**
  - **Set Heading** to **Text**
  - **Type Term** in the Heading Text field
  - **Click OK**
  - **Save** the query
  - **Click Run** tab
<table>
<thead>
<tr>
<th>Course ID</th>
<th>Offer No</th>
<th>Session</th>
<th>Section</th>
<th>ID</th>
<th>Role</th>
<th>Term</th>
</tr>
</thead>
<tbody>
<tr>
<td>007300</td>
<td>1</td>
<td>8W1</td>
<td>1001</td>
<td>585</td>
<td>Pi</td>
<td>Summer Semester 2009</td>
</tr>
<tr>
<td>009547</td>
<td>1</td>
<td>8W2</td>
<td>1001</td>
<td>995</td>
<td>Pi</td>
<td>Summer Semester 2009</td>
</tr>
<tr>
<td>004189</td>
<td>1</td>
<td>8W1</td>
<td>1001</td>
<td>551</td>
<td>Pi</td>
<td>Summer Semester 2009</td>
</tr>
<tr>
<td>004189</td>
<td>1</td>
<td>8W2</td>
<td>1002</td>
<td>548</td>
<td>Pi</td>
<td>Summer Semester 2009</td>
</tr>
<tr>
<td>008537</td>
<td>1</td>
<td>8W3</td>
<td>0001</td>
<td>183</td>
<td>Pi</td>
<td>Summer Semester 2009</td>
</tr>
<tr>
<td>004190</td>
<td>1</td>
<td>8W2</td>
<td>1002</td>
<td>291</td>
<td>Pi</td>
<td>Summer Semester 2009</td>
</tr>
<tr>
<td>004200</td>
<td>1</td>
<td>8W2</td>
<td>1001</td>
<td>291</td>
<td>Pi</td>
<td>Summer Semester 2009</td>
</tr>
<tr>
<td>004291</td>
<td>1</td>
<td>8W2</td>
<td>1001</td>
<td>545</td>
<td>Pi</td>
<td>Summer Semester 2009</td>
</tr>
<tr>
<td>014445</td>
<td>1</td>
<td>3W</td>
<td>0001</td>
<td>959</td>
<td>Pi</td>
<td>Summer Semester 2009</td>
</tr>
<tr>
<td>004294</td>
<td>1</td>
<td>8W2</td>
<td>1002</td>
<td>665</td>
<td>Pi</td>
<td>Summer Semester 2009</td>
</tr>
</tbody>
</table>
**Question 12**

Create a query on academic programs, showing the institution, program (both abbreviated and description), effective date, career, and group for each program. In addition, obtain the description of each group from the academic groups record. Finally, change the headings of the two description columns to “Program Description” and “Group Description.”

- Use the `ACAD_PROG_TBL` record
- Show Fields for
  - `INSTITUTION`
  - `ACAD_PROG`
  - `EFFDT`
  - `DESCR`
  - `ACAD_CAREER`
  - `ACAD_GROUP`
- Join with `ACAD_GROUP_TBL`
- Show field for `Description`
- Change the headings of the academic program description to “Program Description” and academic group description to “Group Description”
- Save and save often. Save as `TRNG_QM##_Q12`
Joins – Any Record

On some occasions you will want to get information from records that do not have a parent/child relationship and where one is not defined as the prompt table for another. Query Manager allows you to build a query such that any fields are joined together, but this must be done manually and with care.

Fields to be joined together will likely have the same name or similar names and must have data that represents the same thing. It would make no sense to join EMPLID in one table to STRM in another, as these are completely different; either no row will be returned because the fields can never match or the ones that are returned are mistakes because the rows aren't really related.

Let’s get a list of financial aid awards for the 2012 aid year in which the accepted amount is greater than $3500. Include the description of the award by joining to the item type table.

- Start a query on **STDNT_AWARDS**
- Select fields:
  - EMPLID
  - INSTITUTION
  - AID_YEAR
  - ITEM_TYPE
  - ACAD_CAREER
- Add criteria:
  - AID_YEAR is equal to 2012
  - ACCEPT_AMOUNT is greater than $3500.
- Save and save often. Save the query as **TRNG_QM##_E13**.
- Click the Run tab to test what you have built so far.

The results show the awards in which more than $3500 was accepted, but it is not clear what those awards are. To get the description of the item type, we will join **STDNT_AWARDS** to **ITEM_TYPE_TBL**.
Click the Query tab.
Click the Hierarchy Join link.

There is NO parent/child relationship between awards and item types. This is because awards don't own item types (the types are independent and used for things other than awards) and item types don't extend information for an award.

Click the Cancel button

Also observe that there is no Related Record join for ITEM_TYPE.

Click the Records tab
Search for ITEM_TYPE_TBL
Click the Join Record link in the ITEM_TYPE_TBL row

A "select join type and then record to join" page appears. You are asked to first choose which type of join to use. In this case you will select the default option, "Join to filter and get additional fields". This is known as a standard join or inner join. (Note that the Hierarchy Join and Related Record joins from previous sections are also inner joins.) You then click the link corresponding to the record to which you want to join the new record. In this case you are joining ITEM_TYPE_TBL to STDNT_AWARDS.

Click A = STDNT_AWARDS – Student Awards
Next, Query Manager tries to detect fields in common on which the join will be built, usually on key fields and fields with the same name. These automatic joins are listed on the Auto Join Criteria page. For each automatic join that you accept, a criterion will be built. You can remove these later if you find they are unnecessary and add some later in the Criteria tab. Query Manager won’t always be correct – for instance, it will miss cases in which the join needs to be on two fields with different names – but it usually provides a good start.

For this exercise, leave both criteria selected and click the Add Criteria button.

The Query page appears and ITEM_TYPE_TBL has been added to the chosen records.

To see that the join criteria were automatically created, click the Criteria tab. The join criteria on ITEM_TYPE and SETID, selected on the Auto Join Criteria page, were added to the query.
Go back to the **Query** tab and select the **DESCR** field for display.

- **Save** your query
- **Click** the **Run** tab. The results now include the description of each item type.

![Screenshot of BGSU PeopleSoft Query Course Query Manager](image)
**Question 13**

Let’s find active employees related to specific job codes and the positions they are in. Create a query using the Job table. Display the employee ID, employee record, effective date, effective sequence, and department ID. Create a prompt, with NO Table Edit, using the JOBCODE field. Join the record to POSITION_DATA to include a position description.

- Start a query on the **JOB** table
- Select fields:
  - **EMPLID**
  - **EMPL_RCD**
  - **EFFDT**
  - **EFFSEQ**
  - **POSITION_NBR**
- Add criteria that **JOBCODE** is equal to a value entered through a prompt. Set the Edit Type of the prompt to ‘No Table Edit’.
- Add criteria that **A.HR_STATUS** is equal to ‘A’
- Join **POSITION_DATA**
  - Select field **DESCR** (agree to the automatic join B.POSITION_NBR = A.POSITION_NBR)
- Save and save often. Save as **TRNG_QM##_Q13**
- Run the query using active job code:
  - 900013
  - 800023

---

**Job Code = 900013**

<table>
<thead>
<tr>
<th>ID</th>
<th>Empl Rcd</th>
<th>Eff Date</th>
<th>Sequence</th>
<th>Position</th>
<th>Descr</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>123</td>
<td>09/01/2014</td>
<td>0</td>
<td>000004000</td>
<td>Instructor</td>
</tr>
<tr>
<td>2</td>
<td>332</td>
<td>09/01/2014</td>
<td>0</td>
<td>000007336</td>
<td>Instructor</td>
</tr>
<tr>
<td>3</td>
<td>224</td>
<td>09/01/2014</td>
<td>0</td>
<td>000008072</td>
<td>Instructor</td>
</tr>
<tr>
<td>4</td>
<td>507</td>
<td>09/01/2014</td>
<td>1</td>
<td>000006005</td>
<td>Instructor</td>
</tr>
<tr>
<td>5</td>
<td>728</td>
<td>09/01/2014</td>
<td>0</td>
<td>000008923</td>
<td>Instructor</td>
</tr>
<tr>
<td>6</td>
<td>086</td>
<td>09/01/2014</td>
<td>1</td>
<td>000006040</td>
<td>Instructor/Graduate Coordinator</td>
</tr>
<tr>
<td>7</td>
<td>385</td>
<td>09/01/2014</td>
<td>0</td>
<td>000003014</td>
<td>Instructor</td>
</tr>
<tr>
<td>8</td>
<td>664</td>
<td>09/01/2014</td>
<td>1</td>
<td>000006090</td>
<td>Instructor, Associate Director</td>
</tr>
<tr>
<td>9</td>
<td>782</td>
<td>09/01/2014</td>
<td>1</td>
<td>000007255</td>
<td>Speech Language Pathologist</td>
</tr>
<tr>
<td>10</td>
<td>302</td>
<td>09/01/2014</td>
<td>1</td>
<td>000002493</td>
<td>Instructor</td>
</tr>
</tbody>
</table>

**Job Code = 800023**

<table>
<thead>
<tr>
<th>ID</th>
<th>Empl Rcd</th>
<th>Eff Date</th>
<th>Sequence</th>
<th>Position</th>
<th>Descr</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>035</td>
<td>09/01/2014</td>
<td>0</td>
<td>0000055430</td>
<td>Assistant Director</td>
</tr>
<tr>
<td>2</td>
<td>721</td>
<td>09/01/2014</td>
<td>0</td>
<td>000007839</td>
<td>Assistant Director, Operations</td>
</tr>
<tr>
<td>3</td>
<td>926</td>
<td>09/01/2014</td>
<td>0</td>
<td>000001314</td>
<td>Assistant Director, Pgm Mgmt</td>
</tr>
<tr>
<td>4</td>
<td>146</td>
<td>09/01/2014</td>
<td>0</td>
<td>000004570</td>
<td>Assistant Director, Student</td>
</tr>
<tr>
<td>5</td>
<td>495</td>
<td>09/01/2014</td>
<td>0</td>
<td>000001393</td>
<td>Assistant Director</td>
</tr>
<tr>
<td>6</td>
<td>481</td>
<td>09/01/2014</td>
<td>0</td>
<td>000008007</td>
<td>Assistant Director, SES</td>
</tr>
<tr>
<td>7</td>
<td>419</td>
<td>09/01/2014</td>
<td>0</td>
<td>000006333</td>
<td>Asst Director, Athletic Dir</td>
</tr>
<tr>
<td>8</td>
<td>492</td>
<td>09/01/2014</td>
<td>0</td>
<td>000001555</td>
<td>Asst DI, Athletic Comm</td>
</tr>
<tr>
<td>9</td>
<td>773</td>
<td>09/01/2014</td>
<td>0</td>
<td>000001586</td>
<td>Asst DI, Financial Affairs</td>
</tr>
<tr>
<td>10</td>
<td>431</td>
<td>09/01/2014</td>
<td>0</td>
<td>000001577</td>
<td>Assistant Director</td>
</tr>
</tbody>
</table>
**Outer Joins**

When you’re joining two tables, the first table (the one on the left) may have rows that don’t have matching counterparts in the second table (the one on the right). The table on the right may have rows that don’t have matching counterparts in the table on the left. If you perform an inner (standard) join on those tables, all the unmatched rows are excluded from the output. Outer joins do not exclude the unmatched rows. Outer joins come in three types: the left outer join, the right outer join, and the full outer join. Query Manager supports only the left outer join.

Consider these excerpts from the countries table (COUNTRY_TBL) and states table (STATE_TBL).

<table>
<thead>
<tr>
<th>COUNTRY_TBL (A)</th>
<th>STATE_TBL (B)</th>
</tr>
</thead>
<tbody>
<tr>
<td>COUNTRY</td>
<td>DESCRIPT</td>
</tr>
<tr>
<td>CAN</td>
<td>Canada</td>
</tr>
<tr>
<td>NOR</td>
<td>Norway</td>
</tr>
<tr>
<td>PAN</td>
<td>Panama</td>
</tr>
<tr>
<td>USA</td>
<td>United States</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

They will be joined together so the countries and the states and provinces within them can be listed in query results. They will be joined on the COUNTRY field since this uniquely identifies a country. For instance, a row in STATE_TBL with a COUNTRY equal to “USA” represents a state in the United States.

When a left outer join takes place, every row in the left table is compared against every row in the right table. When the values in the join field or fields match, the rows are combined and added as one row to the results. When the row of the left table has no matches with any row in the right table, a row is added to the results containing all fields from the left table and empty values where values from the right table would normally appear.

In this example, the first comparison is of the Canada row in COUNTRY_TBL and the Alberta row in STATE_TBL. Since the join field COUNTRY has the same value CAN in both tables, a row combining the Canada row with the Alberta row is added to the results.

As processing continues, the Canada row from COUNTRY_TBL is compared against the British Columbia, Ontario, and Quebec rows from STATE_TBL. Since the values of the join field COUNTRY match in all cases, new rows are added to the results for each province.
The next comparison is between the Canada row of COUNTRY_TBL and the Indiana row of STATE_TBL. Since the COUNTRY values of CAN and USA do not match, a new row is not added to the results.

```
COUNTRY_TBL (A)  STATE_TBL (B)  RESULTS
---  ---  ---
CAN  Canada  CAN  AB  Alberta
NOR  Norway  CAN  BC  British Columbia
MNN  Panama  CAN  ON  Ontario
USA  United States  CAN  QC  Quebec

not a match

not a match
```

Similarly, no rows are added when the Canada row is compared against the Michigan and Ohio rows.

Next, the Norway row in COUNTRY_TBL is compared to the Alberta row of STATE_TBL. Since the COUNTRY values of NOR and CAN do not match, nothing is added to the results.

```
COUNTRY_TBL (A)  STATE_TBL (B)  RESULTS
---  ---  ---
CAN  Canada  CAN  AB  Alberta
NOR  Norway  CAN  BC  British Columbia
MNN  Panama  CAN  ON  Ontario
USA  United States  CAN  QC  Quebec

not a match

not a match
```

Processing continues with each of the rows in STATE_TBL compared to the Norway row of COUNTRY_TBL. When processing reaches a comparison of the Ohio row, nothing has been added to the results for Norway.

```
COUNTRY_TBL (A)  STATE_TBL (B)  RESULTS
---  ---  ---
CAN  Canada  CAN  AB  Alberta
NOR  Norway  CAN  BC  British Columbia
MNN  Panama  CAN  ON  Ontario
USA  United States  CAN  QC  Quebec

not a match

not a match
```

Since there have been no matches for Norway and this is a left outer join, a row is added to the results containing all fields from the left table (COUNTRY_TBL) and empty values in place of all fields from the right table (STATE_TBL). In this case, A.COUNTRY and A.DESCRIPT are filled with NOR and Norway while B.COUNTRY, B.STATE, and B.DESCRIPT are filled with empty values.

```
COUNTRY_TBL (A)  STATE_TBL (B)  RESULTS
---  ---  ---
CAN  Canada  CAN  AB  Alberta
NOR  Norway  CAN  BC  British Columbia
MNN  Panama  CAN  ON  Ontario
USA  United States  CAN  QC  Quebec

not a match

not a match
```

Processing follows with the Panama row of COUNTRY_TBL being matched against each row in STATE_TBL. Since there are no rows in STATE_TBL with a COUNTRY value of PAN, nothing will be added to the results.
After the last row of `STATE_TBL` is checked and no matches have been found, a row is added to results containing the fields of the Panama row of `COUNTRY_TBL` and empty values filling in the fields of `STATE_TBL`.

Finally, the United States row of `COUNTRY_TBL` is compared to each row of `STATE_TBL`. The rows for Indiana, Michigan, and Ohio have a matching `COUNTRY` value of USA, so rows will be added to the results for each of these states. When the outer join is complete, the results will be the following:

Below is a comparison of the results of using an inner join on `COUNTRY_TBL` and `STATE_TBL` and an outer join on these two tables.
Now let us build a query that utilizes an outer join. We will get a list of Theatre and Film classes from the Spring 2013 semester. In cases in which there is an exam date and time recorded for the class, display it. If there is no exam date and time make sure the class is still shown.

Class data is in CLASS_TBL and exam data is in CLASS_EXAM. We will need to join CLASS_TBL and CLASS_EXAM, but will use an outer join because if there is no row in CLASS_EXAM matching the row in CLASS_TBL (which will be common because if a class has multiple components, such as lecture, lab, and recitation, only one will have an exam), we still want the data from CLASS_TBL to be in the results.

- Start a query on CLASS_TBL
- **Select** fields:
  - CRSE_ID
  - CRSE_OFFER_NBR
  - STRM
  - SESSION_CODE
  - SUBJECT
  - CLASS_SECTION
  - CATALOG_NBR
  - DESCR
  - SSR_COMPONENT
- **Add criteria:**
  - STRM equal to 2132
  - SUBJECT equal to THFM
- Go to the **Fields** tab
- Click the **Reorder/Sort** button
- **Set** the **New Order By** of:
  - SUBJECT to 1
  - CATALOG_NBR to 2
  - CLASS_SECTION to 3
- **Set** the **New Column** value of CLASS_SECTION to be immediately after CATALOG_NBR
- Click the **OK** button

- Save and save often. Save as **TRNG_QM#E14**.
- Run the query to test what you have built so far.
We now have a list of Theatre and Film classes from the Spring 2013 semester. Next we will join these results to CLASS_EXAM to get the exam times for each class.

- Click the Records tab
- Search for and select the CLASS_EXAM record
- Click the Join Record link
- In the “Select join type” page, select the “Join to get additional fields only (Left outer join)” option
- Click the A = CLASS_TBL – Class Table link

As when you did the Any Record join in the previous section, Query Manager attempts to find the proper join conditions and offers to create criteria for them.

- Accept the default join conditions by leaving them all checked
- Click the Add Criteria button

The Query tab is shown. The CLASS_EXAM record is added to the query and the description indicates it has been outer joined to CLASS_TBL (referring to it by its alias of “A”).
Select the following fields of CLASS_EXAM:
- CLASS_EXAM_SEQ
- EXAM_DT
- EXAM_START_TIME
- EXAM_END_TIME

Save the query
Click the Run tab

Now there are exams shown for classes that have exam dates and times defined in the system and the classes that do not have exams defined are still displayed.
There is an important factor to consider when adding criteria to a query that has an outer join. You must make sure that the criteria “belong” to the correct clause of the query. This determines whether the criteria is applied before the join is made (limiting the rows that could be joined) or after the join is made (limiting the results of the join).

Click the **Criteria** tab to show all the criteria in the query.

Since there is an outer join in the query, the **Belongs to** column is displayed. This indicates whether a criterion belongs to the main query (the value is blank), so that it is applied after the join, or to a specific record (the value is an alias letter – B in this case), so that it is applied before the join.
Click the **Edit** button in the row for **STRM**. In the Edit Criteria Properties page, there is a section named “This criteria belongs to”. The value shown is “WHERE clause”. This indicates that the criterion is applied after the join. (“WHERE clause” refers to the SQL generated by Query Manager that is sent to the database to execute the query.)

Click the **Cancel** button.

Now click the **Edit** button in the row for **CRSE_ID**. In the Edit Criteria Properties page, the section “This criteria belongs to” shows “ON clause of outer join B”. This indicates that the criterion is applied to record B (CLASS_EXAM) before the join. (“ON clause” also refers to the generated SQL.)

Click the **Cancel** button.

When Query Manager added the join criteria from the Auto Join Criteria step, it set them to belong to the ON clause of outer join B because we need to limit the rows being joined to those having matching values in record A (CLASS_TBL).

The two criteria we added before the outer join – those on STRM and SUBJECT – belong to the WHERE clause because term and subject are not in CLASS_EXAM table and have no effect on whether there is a matching row in CLASS_EXAM.

What if we wanted to show only exams on particular days? You would add a criterion that belongs to the ON clause because you’d want to check only rows in CLASS_EXAM that have those days before joining. If you did it after the join, all the classes with no exam would be thrown out because the condition that EXAM_DT has the desired value would be applied to a rows with an empty EXAM_DT.
In general:

Select “ON clause of outer join” if:
- this criterion is part of the join condition (such as A.CRSE_ID is equal to B.CRSE_ID)
  
  **OR**
  
- this criterion restricts rows that are in the record being outer joined (such as B.EXAM_DT is 5/2/2013)

Select “WHERE clause” option if:
- this criterion is related to the main record or records in the query (such as A.STRM is equal to “2132”)


Question 14

Create a query to list offers of awards of less than $750 of financial aid that were accepted. Limit the results to the 2013 aid year and to undergraduates. In cases in which there is a message for the award, show the message; if there is no such message, the award must still be displayed in the results. Order the results by student ID and item type. Make the award message code and descriptions the rightmost two columns in the results.

- Use the `STDNT_AWARDS` and `AWD_MESSAGE_TBL` records
- Consider carefully whether to use an inner join or an outer join. What should happen if there is no row in `AWD_MESSAGE_TBL` matching the row in `STDNT_AWARDS`?
- Show the following fields from `STDNT_AWARDS`:
  - EMPLID
  - ITEM_TYPE
  - DISBURSEMENT_PLAN
  - SPLIT_CODE
  - AWARD_MSG_CD
  - OFFER_AMOUNT
  - ACCEPT_AMOUNT
- Save this query as `TRNG_QM##_Q14`
- Show the `DESCRLONG` field from `AWD_MESSAGE_TBL`
- Add the following criteria:
  - INSTITUTION is “BGSUN”
  - AID_YEAR is “2013” (aid years are specified as calendar years, not term codes)
  - ACAD_CAREER is “UGRD”
  - AWARD_STATUS is “A” (accepted)
  - OFFER_AMOUNT is less than $750
- Position AWARD_MSG_CD as the second-to-last column in the results
- Position DESCRLONG as the last (rightmost) column in the results
- Sort the results by EMPLID and ITEM_TYPE
- Save the query again
**Subqueries – Checking for Existence**

Many times, to answer one question, you need to ask several other questions. There may be cases in which you have to run one query to collect information to be used in another query. Rather than write and run two separate queries, you can use a subquery within your main query to collect that information. The result of the subquery can then be used like a field or expression in your query criteria.

To answer some questions, you may need to collect specific information, but to answer others, you may only need to find out whether something – usually a child row or a related row – exists. In a query, this is done by including an *exists* subquery or a *does not exist* subquery in your main query.

For each row returned by the main query, the subquery will report whether or not there is a matching row matching the conditions in the subquery. This is similar to an outer join, which reports the data from a second record regardless of whether there is a match, but no data is actually returned.

Consider again the excerpts from the countries (COUNTRY_TBL) and states (STATE_TBL) tables.

Let us ask the question of which countries have states or provinces. We do not need to know what those states or provinces are, so it is not necessary to join COUNTRY_TBL to STATE_TBL. In this case we would use an *exists* subquery. For each country in COUNTRY_TBL, the subquery would check STATE_TBL for any rows with the same value in COUNTRY as the country being examined.

Processing starts with the Canada row of COUNTRY_TBL. The subquery examines the states table for the existence of a row with the same value in COUNTRY – CAN. Since one is found, the Canada row is added to the results.

Processing continues with the Norway row. The subquery checks for the existence of a row in STATE_TBL with NOR in the COUNTRY field. Since there is no such row, nothing is added to the results.
The Panama row is examined next. Since there are no rows in STATE_TBL with a COUNTRY value of PAN, no rows are added to the results.

<table>
<thead>
<tr>
<th>COUNTRY_TBL (A)</th>
<th>STATE_TBL (B)</th>
<th>RESULTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>CAN</td>
<td>Canada</td>
<td>CAN</td>
</tr>
<tr>
<td>NOR</td>
<td>Norway</td>
<td>NOR</td>
</tr>
<tr>
<td>PAN</td>
<td>Panama</td>
<td>PAN</td>
</tr>
<tr>
<td>USA</td>
<td>United States</td>
<td>USA</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Lastly, the United States row is examined. The subquery finds a row in STATE_TBL that has a COUNTRY value of USA, matching the COUNTRY_VALUE in the United States row; the United States row is therefore added to the results.

<table>
<thead>
<tr>
<th>COUNTRY_TBL (A)</th>
<th>STATE_TBL (B)</th>
<th>RESULTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>CAN</td>
<td>Canada</td>
<td>CAN</td>
</tr>
<tr>
<td>NOR</td>
<td>Norway</td>
<td>NOR</td>
</tr>
<tr>
<td>PAN</td>
<td>Panama</td>
<td>PAN</td>
</tr>
<tr>
<td>USA</td>
<td>United States</td>
<td>USA</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

If we wanted to instead know which countries did not have any states or provinces, we would use a *not exists* subquery. For every row country in COUNTRY_TBL, the subquery would check if a row with a matching COUNTRY exists in STATES_TBL. Only if there is *not* a matching row in STATE_TBL will the row in COUNTRY_TBL be added to the results. This is the opposite of an *exists* subquery.

Using the same excerpts of COUNTRY_TBL and STATE_TBL from the previous example, here is how the countries that do not have states or provinces would be found using a not exists subquery:
<table>
<thead>
<tr>
<th>COUNTRY_TBL (A)</th>
<th>STATE_TBL (B)</th>
<th>RESULTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>COUNTRY</td>
<td>DESCRIPTOR</td>
<td></td>
</tr>
<tr>
<td>CAN</td>
<td>Canada</td>
<td></td>
</tr>
<tr>
<td>NOR</td>
<td>Norway</td>
<td></td>
</tr>
<tr>
<td>PAN</td>
<td>Panama</td>
<td></td>
</tr>
<tr>
<td>USA</td>
<td>United States</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>COUNTRY</th>
<th>STATE</th>
<th>DESCRIPTOR</th>
</tr>
</thead>
<tbody>
<tr>
<td>CAN</td>
<td>AB</td>
<td>Alberta</td>
</tr>
<tr>
<td>CAN</td>
<td>BC</td>
<td>British Columbia</td>
</tr>
<tr>
<td>CAN</td>
<td>ON</td>
<td>Ontario</td>
</tr>
<tr>
<td>CAN</td>
<td>QC</td>
<td>Quebec</td>
</tr>
<tr>
<td>USA</td>
<td>IN</td>
<td>Indiana</td>
</tr>
<tr>
<td>USA</td>
<td>MI</td>
<td>Michigan</td>
</tr>
<tr>
<td>USA</td>
<td>OH</td>
<td>Ohio</td>
</tr>
</tbody>
</table>

This space intentionally left blank.
Next we will build an exists subquery in Query Manager. Consider the following exercise. Write a query to find undergraduate classes offered in Fall 2013 that had graduate students enrolled in them. You can think of this as a two-step process: collect all undergraduate classes from Fall 2013, then for each class, check if there were any graduate students enrolled. The main query will list classes and the subquery will check for the existence of any graduate students in each class.

- Start a query on **CLASS_TBL**
- Select fields:
  - CRSE_ID
  - SESSION_CODE
  - CLASS_SECTION
  - SUBJECT
  - CATALOG_NBR
  - DESCR
  - CLASS_NBR
- Add Criteria:
  - STRM is equal to **2138** (Fall 2013)
  - ACAD_CAREER is equal to **UGRD**

Save and save often. **Save** as **TRNG_QM##_E15**.

**Run** the query to see a list of all undergraduate classes offered in Fall 2013.

Now we will add the limitation that the results will show only classes that had a graduate student enrolled.

- Go to **Criteria tab**
- Click **Add Criteria**
- Change Condition Type to “exists”. The Expression 1 section disappears and the Expression 2 section shows only the Define/Edit Subquery link.
- Click the **Define/Edit Subquery** link
You are taken to the Records page. Below the Query Name field you will see “Working on selection: Subquery for exists”. This informs you that you are now building the subquery that will be used to report information back to the main query.

Now we will build the subquery that checks for graduate student enrollment in the classes being processed in the main query.

- **Search** for and add **STDNT_ENRL_VW**. (Observe that the link is titled Add Record instead of Join Record. A subquery is not a join.)

- **Select** the **EMPLID** field. (For subqueries you can select only one field. For exists and not exists subqueries it does not matter which field you choose.)
- **Save** the query

After selecting the field you are taken to the Fields tab. The field that you selected is displayed in the Fields list.
Next the subquery must be tied back to the main query so that only students from the classes in the main query are examined. It must also be restricted to enrolled graduate students.

- Click the Criteria tab
- Add Criteria:
  - A.CLASS_NBR equal to B.CLASS_NBR
  - A.INSTITUTION equal to B.INSTITUTION
  - A.STRM equal to B.STRM
  - B.ACAD_CAREER equal to GRAD
  - B.STDNT_ENRL_STATUS equal to E (enrolled)

- Save the query
- Click the Run tab
Note: if you need to work on different parts of the query, click the Subquery/Union Navigation link near the top of every Query Manager page except for Run.

You will be taken to a page that lists the main part of the query, all subqueries, and all unions (which are described in the next section). Click the link corresponding to the part of the query you want to work on.
**Subqueries – Single Value**

In the previous segment we used a subquery to find out whether or not there was a graduate student enrolled in an undergraduate class. The subquery did not need to return any information about the student – just that a student existed. In this segment we will use a subquery to get information about what is being examined in the main query, then use that information to restrict the rows being kept in the results.

Let’s find undergraduate student-athletes who were enrolled for more than 17 credit hours in Spring 2014. List the student’s program of study and the sports in which the students are participating. Show the students in order by student ID, program, and sport.

As we build this query, keep the idea of the subquery in mind. For each student, we will ask how many credit hours that student is taking in the selected term. This will be done with a subquery. The result of that subquery – the number of credit hours – will then be used in the criteria of the main query.

Note that this question will be asked for every student; the credit hours may be different for different students. This is why a subquery is useful and appropriate – without one, we’d have to get the enrollment manually for every student-athlete!

- Start a query on **ACAD_PROG**
- **Select** fields:
  - **EMPLID**
  - **ACAD_PROG**
- **Add** criteria:
  - **ACAD_CAREER** is equal to **UGRD**
  - **INSTITUTION** is equal to **BGSUN**
  - **PROG_STATUS** is equal to **AC**
- Save and save often. Save as **TRNG_QM##_E16**.
- Run the query. The results list each active undergraduate student and their programs of study.

```
EMPLID | ACAD_PROG
-------|----------
333    | CSTFA
312    | CSTFA
334    | CSTFA
725    | FIRE
911    | FIRE
885    | ACEN
996    | ACEN
372    | ACEN
```

Now we will limit the results to just student-athletes and retrieve the sports they are in by joining to the **ATHL_PART_STAT** record.

- **Click** the **Records** tab
- Search for and select the **ATHL_PART_STAT** record
Join **ATHL_PART_STAT** to **ACAD_PROG**
- Accept the default join criteria by clicking the **Add Criteria** button
- **Select** the **SPORT** field
- **Add** criteria on **ATHL_PART_STAT** as follows:
  - **CUR_PARTICIPANT** is equal to **Y**
- **Click** the **Fields** tab
- **Click** the **Reorder/Sort** button
- Set the **New Order By** values as follows:
  - **EMPLID** is **1**
  - **ACAD_PROG** is **2**
  - **SPORT** is **3**

**Save** the query and run it again. Now the results contain just current students who are also participating in at least one sport. (Students who are in more than one sport will be listed multiple times, once for each sport.)

Next we need to add a criterion to the query to limit results to those in which the student is enrolled for more than 17 credit hours in Spring 2014. This will be done with a subquery.

- **Click** the **Criteria** tab
- **Click** the **Add Criteria** button
- **Select Expression** for **Expression 1 Type**
- **Click** the **New Expression** link
Create the expression:
- Select **Number** for Expression Type
- Set **Length** to 2
- Set **Expression Text** to 17
- **Click OK**

(To compare a constant to the result of a subquery it is necessary to create an expression; even though there is a Constant option for Expression 2 Type, there is no Subquery option for Expression 1 Type.)

- **Set Condition Type** to less than
- **Set Expression 2 Type** to "Subquery"
- **Click Define/Edit Subquery**

You are taken to the Records page. Near the top of the page the message "Working on selection: Subquery for 17" is displayed. This indicates you are building a subquery rather than working on the main query.

- **Select** and add **STDNT_CAR_TERM** to the subquery

After clicking on the Add Record link, the Query page appears. Fields of STDNT_CAR_TERM are displayed, but instead of each field having a checkbox, they each have a Select link. This is because a subquery can return only one value. In this case we want the UNT_TAKEN_PRGRSS.
Click the Select link in the **UNT_TAKEN_PRGRSS** row.

You are taken to the **Fields** tab. **C.UNT_TAKEN_PRGRSS** is listed as the single field for this subquery.

Save the query.

Now we need to limit the subquery to the correct student and to the desired term. We will do this by adding criteria to the subquery.

Go to either the **Query** or **Criteria** tab – making sure that "Working on selection: Subquery for 17" is still shown – then add the following criteria:

- **A.EMPLID** equal to **C.EMPLID**
- **A.ACAD_CAREER** equal to **C.ACAD_CAREER**
- **A.INSTITUTION** equal to **C.INSTITUTION**
- (these three criteria make sure the subquery provides results on the same student that the main query is processing)
- **C.STRM** is equal to **2142**
- **Save** the query
- **Click** the **Run** tab

These results list all students who are active student-athletes who, as indicated through a subquery, are taking more than 17 credit hours in the Spring 2014 semester.
Question 16

Find classes offered in Spring 2014 that are for the BG Perspective general education program. For each class, show the section, subject, catalog number, and description, as well as the primary name of the primary instructor of each class. This will require joining CLASS_TBL to CLASS_INSTR and CLASS_INSTR to NAMES. It will also require a subquery criterion on CLASS_TBL to check for the existence of rows in CLASS_ATTRIBUTE with a CRSE_ATTR of “BGE” (despite the name of the program being “BG Perspective” the code is “BGE” rather than “BGP”).

- Show the following fields from CLASS_TBL:
  - CLASS_SECTION
  - SUBJECT
  - CATALOG_NBR
  - DESCR

- Add a criterion that STRM is equal to 2142

- Add an exists subquery using the CLASS_ATTRIBUTE record

- Add criteria to the subquery:
  - B.CRSE_ID is equal to A.CRSE_ID
  - B.CRSE_OFFER_NBR is equal to A.CRSE_OFFER_NBR
  - B.STRM is equal to A.STRM
  - B.SESSION_CODE is equal to A.SESSION_CODE
  - B.CLASS_SECTION is equal to A.CLASS_SECTION
  - B.CRSE_ATTR is equal to BGE (BG Perspective)

- Join the CLASS_INSTR record to CLASS_TBL

- Add a criterion that INSTR_ROLE is equal to PI (primary instructor)

- Join the NAMES record to CLASS_INSTR

- Show the NAME field from NAMES

- Add a criterion that NAME_TYPE is equal to PRI (primary)

- Save as TRNG_QM##_Q16

<table>
<thead>
<tr>
<th>Section</th>
<th>Subject</th>
<th>Catalog</th>
<th>Descr</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>ARTH</td>
<td>1450</td>
<td>Western Art</td>
<td>Durham, Rachel Kathleen</td>
</tr>
<tr>
<td>2</td>
<td>BA</td>
<td>1500</td>
<td>Overview Business Admin</td>
<td>White, Ruth C</td>
</tr>
<tr>
<td>3</td>
<td>BIOL</td>
<td>1010</td>
<td>Environment Of Life</td>
<td>McLean, Jaclyn Elizabeth</td>
</tr>
<tr>
<td>4</td>
<td>BIOL</td>
<td>1010</td>
<td>Environment Of Life</td>
<td>McLean, Jaclyn Elizabeth</td>
</tr>
<tr>
<td>5</td>
<td>GSW</td>
<td>1120</td>
<td>Academic Writing</td>
<td>Zlot, Jessica Dawn</td>
</tr>
<tr>
<td>6</td>
<td>GRW</td>
<td>1120</td>
<td>Academic Writing</td>
<td>Stolar, Stacey Andrew</td>
</tr>
<tr>
<td>7</td>
<td>GSW</td>
<td>1120</td>
<td>Academic Writing</td>
<td>Raczewski, Frank Daniel</td>
</tr>
<tr>
<td>8</td>
<td>PHYS</td>
<td>2620</td>
<td>College Physics II</td>
<td>Williams, Jonathan E</td>
</tr>
<tr>
<td>9</td>
<td>PHYS</td>
<td>2620</td>
<td>College Physics II</td>
<td>Williams, Jonathan E</td>
</tr>
<tr>
<td>10</td>
<td>PHYS</td>
<td>2620</td>
<td>College Physics II</td>
<td>Williams, Jonathan E</td>
</tr>
</tbody>
</table>
**Unions**

A union is the combination of results from two or more queries. If two queries are combined in a union, the result is all of the rows from the first query and all of the rows from the second query. Duplicate rows are discarded.

In order for two queries to be combined in a union, the queries must have the same number of columns and each column from the first query must have the same data type (date, name, or string) as the corresponding column from the second query. For instance, if the third column in the first query is a date, the third column in the second query must also be a date.

For the next exercise, you have an important communication for undergraduate students in the Music program, both current and recently admitted (for Fall 2013 or later). Current students are listed in the `ACAD_PROG` table but admitted students are listed in `ADM_APPL_PROG`. Instead of building two queries you will use a union to collect the current and admitted students into the same set of results. Since the communication will be sent by e-mail you will join to `EMAIL_ADDRESSES` to obtain e-mail addresses for each student.

- Start a new query. Add the `ACAD_PROG` record
- Select the `EMPLID` field
- Add the following criteria:
  - `ACAD_CAREER` is equal to `UGRD`
  - `INSTITUTION` is equal to `BGSUN`
  - `ACAD_PROG` is equal to `MUSIC`
  - `PROG_STATUS` is in the list `AC, LA`
- Save and save often. Save as `TRNG_QM##_E17`

The Criteria page should look like this.

- **Run** the query

The results should look like this.
Now that we have the current Music students, we need to obtain the admitted students. We will do this by building a second query that retrieves the students and combining them in a union with the first query.

- Click on any tab except for Run. **Click the New Union link** located at the bottom of the page.

You are taken to the Records page. The indicator “Working on selection: Union 1” shows that you are working on the second part of this query.

- Search for and add the **ADM_APPL_PROG** record.
- Select the **EMPLID** field. (Since this is a union, rather than a subquery, you can select multiple fields. However, remember that the number of fields in each union must be the same.)
- **Add** the following criteria:
  - **ACAD_CAREER** is equal to **UGRD**
  - **INSTITUTION** is equal to **BGSUN**
  - **ACAD_PROG** is equal to **MUSIC**
  - **PROG_ACTION** is in the list **ADMT**, **MATR**
  - **ADMIT_TERM** is not less than **2138**
Now the results list students who are currently in the Music program and those admitted to it for Fall 2013 and later.

In order to send an e-mail to these students, we must obtain the e-mail addresses of the students. E-mail addresses are stored in the EMAIL_ADDRESSES table. You will join EMAIL_ADDRESSES to both ACAD_PROG and to ADM_APPL_PROG.

First we will work on the part of the query dealing with current students. When you are editing a query that contains a union, the Subquery/Union Navigation link is shown on all tabs except for the Run tab. Follow this link to get a list of all parts of the query then follow the link corresponding to the part of the query you want to work on.

- **Click** the Records tab
- **Click** the Subquery/Union Navigation link
All parts of the query that are connected by a union are displayed.

In this exercise, **Top Level of Query** corresponds to the part getting current students (from `ACAD_PROG`) and **Union 1** corresponds to the part getting admitted students (from `ADM_APPL_PROG`).

- **Click** the **Top Level of Query** link

The Query tab for the first part of the query is shown. Now we need to join **EMAIL_ADDRESSES**.

- **Click** the **Records** tab.
- **Search for and join the** EMAIL_ADDRESSES record to **ACAD_PROG**.
- **Accept the default join criteria by clicking the Add Criteria button.**
- **Select** the **EMAIL_ADDR** field.
- **Add** the following criteria:
  - E_ADDR_TYPE is equal to CAMP.

- **Save** the query

When two or more parts of a query are connected by a union, they must select the same number of fields and the field types must match. (For instance, if the first column in one part is a date, the first column in all other parts must also be a date.)

We still need to add the e-mail address to the second part of the query.

- **Click** the **Records** tab
- **Click** the **Subquery/Union Navigation** link
- **Click** the **Union 1** link

The Query tab for the second part of the query is shown. Now we will join **EMAIL_ADDRESSES**. (Keep in mind that you joined EMAIL_ADDRESSES to **ACAD_PROG** in the first part of the query, but the second part is completely separate from the first part.)

- **Click** the **Records** tab.
- **Search for and join the** EMAIL_ADDRESSES record to **ADM_APPL_PROG**.
Accept the default join criteria by clicking the **Add Criteria** button.

Select the **EMAIL_ADDR** field.

Add the following criteria:

- **E_ADDR_TYPE** is equal to **CAMP**.

Now the query results include both current and admitted students and the e-mail address of each student.

Duplicate rows are automatically discarded when parts are connected in a union. A duplicate row in this context is based on all fields in the row. In addition, the results are automatically sorted by first field, second field, and so on, but this order can be overridden if desired.
**Scheduling Queries**

Some queries can be so complex that it can take several minutes to get results; during the time the query is running it is usually not possible to perform other tasks in PeopleSoft unless another PeopleSoft window is open. There are also cases in which a query needs to be run on a regular basis, such as once a day or once a week, for reporting and monitoring purposes.

Scheduling a query to run at a particular time can address both of these items. When a query is scheduled, this allows it to be run without user intervention. The user can work in other pages in PeopleSoft then pick up the results when the query has finished running. If the query is scheduled to run under a recurrence, the query will be run at the days and times defined for that recurrence, automatically being rescheduled to run again at the appropriate time.

To schedule a query, search for it in the Query Manager search page, then click the **Schedule** link.

You will then be asked to select or create a **run control** to identify the process submitted to the PeopleSoft Process Monitor for running your query on a schedule. If you have no run controls that run the selected query you will automatically be put on the **Add a New Value** tab for creating a new run control; otherwise you will be taken to the **Find an Existing Value** tab from which you can select a run control you have already made for scheduling the query.

This example will involve creating a new run control by clicking the Add a New Value tab. The Private Query and Query Name fields are filled in automatically. Enter a Run Control ID to assign to this run control then click the **Add** button. If the query has any prompts you will be asked to enter values for each prompt.
You will then be asked to enter a description for this scheduled run. Enter this then click the OK button.

The Process Scheduler Request page appears. To pick a specific date and time at which the query should be run, enter the date and time into the Run Date and Run Time fields. To instead have the query run repeatedly at the same time every day or every week, select a predefined recurrence from the Recurrence dropdown.

In the Process List section you will see one process named PSQUERY; this is the internal process that will run the query. In the Format dropdown there are several different formats for the output of the query, including:

- HTM: web page, similar to the output of a query when run from Query Manager
- PDF: a document in the Adobe PDF format
- TXT: a comma-separated variable (CSV) text file
- XLS: an Excel workbook
Select the format that you want for the output. When you are ready to submit, click the OK button. You will be returned to the Query Manager search page.

To see the results of the query navigate to Reporting Tools > Report Manager then click the Administration tab. Your recently generated reports are displayed.

Click the link in the Details column corresponding to the scheduled query for which you want to view results. The details of that report are displayed. In the File List section, find the link with the name of the query (not the run control ID) that you scheduled.

Click the file link to open the file directly or right-click it and choose Save Target As to save it to your computer.

If you selected the TXT format to generate a CSV file and you are running a computer with Excel, make sure to save the file to your computer instead of opening directly and to save it with an extension of CSV instead of the default of XLS. This will prevent Excel from giving a warning that the file is in a different format than that specified by the extension. In addition, if there are values with leading zeroes such asEMPLIDs and ZIP codes in the results, create a new workbook then import the CSV file using Data > From Text and set the data type of those columns to Text; this will prevent Excel from dropping leading zeroes from those values. If you do not need to import the data into a database or other program you could change the format to XLS when submitting the query to get the data as a true Excel workbook.
In Campus Solutions (CSS) and Human Capital Management (HCM) there are many defined recurrences to choose from. The name of the recurrence indicates the days, times, and frequency of that recurrence. Some examples of the naming convention are as follows:

- DLY5CN0700A – daily (DLY) on weekdays (5) at 7:00 AM (0700A); schedule when the current request starts (C) and schedule missed runs (N)
- DLY7PN0400P – daily (DLY) every day of the week (7) at 4:00 PM (0400P); schedule when the previous recurrence finishes (P) and schedule missed runs (N)
- WKYFPN0915P – weekly (WKY) on Fridays (F) at 9:15 PM (0915P); schedule when the previous request finishes (P) and schedule missed runs (N)
- WKYMWPY0630A – weekly (WKY) on Mondays and Wednesdays (MW) at 6:30 AM (0630A); schedule when the previous request finishes (P) and do not schedule missed runs (Y)

The letters U and R are used to indicate Sunday and Thursday, respectively.

In Financial Management Solutions (FMS) there are also many defined recurrences. The naming convention is less strict than in CSS. Most recurrences are just named after the days and times on which they run; examples include “M-F at 5pm” and “T – S 8:00 AM”.

...
Supplemental Material

Finding Records and Fields

Normally a request for a query will be in the form of a question or a request for certain information. It will usually not come with the fields, records, and criteria given to you directly. Instead, it will be like a story problem. You’ll be responsible for translating the request into a query that PeopleSoft understands.

What if you don’t already know where to find the information?

- Use the Advanced Search capabilities. You can search by record name, description, and fields used in a record, among others. For instance, if you know a query is about programs, you can search for records having “program” in the description or PROGRAM in the name. If this does not provide enough options or any correct ones, try abbreviations such as PGM and PROG. You could also search for records having a field with PROG in the name.
- Look at the descriptions. When you search for records, the description of each record is given along with the name of the record. When you show fields in a record, the description of each field is displayed.
- Ask the requestor to show you where the underlying data is in PeopleSoft. If he or she can show you some pages, you can get some field labels that may appear in the descriptions of fields. If the requestor shows you a field labeled “Contact Person,” then you can search for records that have fields with descriptions containing “contact” or “person” or names containing “PERS”.
- Use references that others in your department have created.
- Ask your coworkers. 😊
Effective Date, Effective Sequence, and Effective Status

Frequently there is a need to keep track of the history of changes to something in a database. The status of a student will change as he applies to a program, matriculates, possibly adds minors, and completes that program. There may also be a need to retain information about something, such as its name, flags, and amounts. In addition, to aid in planning ahead, there may be a need to store something that will come into effect in the future.

PeopleSoft uses special fields in many records to enable having data effective only at certain times – Effective Date (EFFDT), Effective Sequence (EFFSEQ), and Status as of Effective Date (EFF_STATUS).

Basics of “Effective” Data

Effective Date is the most commonly used of these three fields; the other two fields will not be in a record without an EFFDT field. It indicates that the record is effective as of a certain date.

This concept may be easiest to understand using an example. Consider the following rows of ACAD_PROG_TBL, which contains information about academic programs.

<table>
<thead>
<tr>
<th>INSTITUTION</th>
<th>ACAD_PROG</th>
<th>EFFDT</th>
<th>EFF_STATUS</th>
<th>DESCR</th>
<th>ACAD_CAREER</th>
<th>ACADCALENDAR_ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>BGSUN</td>
<td>MUSIC</td>
<td>8/29/1982</td>
<td>A</td>
<td>College of Musical Arts</td>
<td>UGRD</td>
<td>USEM</td>
</tr>
<tr>
<td>BGSUN</td>
<td>MUSIC</td>
<td>9/21/1976</td>
<td>A</td>
<td>College of Musical Arts</td>
<td>UGRD</td>
<td>UQTR</td>
</tr>
<tr>
<td>BGSUN</td>
<td>MUSIC</td>
<td>9/24/1968</td>
<td>A</td>
<td>School of Music</td>
<td>UGRD</td>
<td>UQTR</td>
</tr>
<tr>
<td>BGSUN</td>
<td>MUSIC</td>
<td>1/1/1910</td>
<td>A</td>
<td>School of Music</td>
<td>UGRD</td>
<td>USEM</td>
</tr>
</tbody>
</table>

The row with the highest EFFDT that is not in the future contains the information current for today. In the example above, since August 29, 1982, the name of the music program has been “College of Musical Arts” and the academic calendar was based on semesters (USEM). From September 21, 1976 through August 28, 1982, the name was the same, but the calendar was based on quarters (UQTR).

If there is an Effective Date field in a record, the Effective Date is always part of the key, and it is the last part of the key unless there is also an Effective Sequence. The Effective Date applies to all of the key fields preceding it. In the example, EFFDT applies to the combination of INSTITUTION and ACAD_PROG. The Effective Date for program MUSIC does not indicate when data for effective for program ARTSC, program BUSN, etc.

By convention, the Effective Date used for the first instance of something that is effective dated is 1/1/1900 for base PeopleSoft data and 1/1/1910 for data added by BGSU.
If it is likely for there to be several changes to something on the same day, an Effective Sequence field is included in the record. This is a number that starts at 0 on a particular day and increases by 1 for each change to that something (identified by key) on that day. The most current information has the highest non-future EFFDT and the highest EFFSEQ for that EFFDT.

Again, an example can more clearly demonstrate this concept.

<table>
<thead>
<tr>
<th>EMPID</th>
<th>EMP_RID</th>
<th>EFFDT</th>
<th>EFFSEQ</th>
<th>DEPTID</th>
<th>JOBCODE</th>
<th>SUPERVISOR_ID</th>
<th>HR_STATUS</th>
<th>ACTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>9106</td>
<td>0</td>
<td>4/20/08</td>
<td>0</td>
<td>310200</td>
<td>600001</td>
<td>2164</td>
<td>I</td>
<td>TER</td>
</tr>
<tr>
<td>9106</td>
<td>0</td>
<td>7/29/07</td>
<td>1</td>
<td>310200</td>
<td>600001</td>
<td>2164</td>
<td>A</td>
<td>DTA</td>
</tr>
<tr>
<td>9106</td>
<td>0</td>
<td>7/29/07</td>
<td>0</td>
<td>071100</td>
<td>600001</td>
<td>1316</td>
<td>A</td>
<td>HIR</td>
</tr>
</tbody>
</table>

Employee 9106 was hired on 7/29/2007, into department 071100 under supervisor 1316; this is shown in the row with EFFDT of 7/29/2007 and EFFSEQ of 0. Later that day, the employee was transferred to department 310200 with supervisor 2164; this is shown in the row with EFFDT of 7/29/2007 and EFFSEQ of 1. This assignment was effective until the employee’s termination on 4/20/2008.

There may be cases in which something is deactivated or will be deactivated or is added to the database before it will become effective. These cases are handled by using a Status as of Effective Date field. Consider these two examples from ACAD_PLAN_TBL, which contains information on academic plans (majors).

<table>
<thead>
<tr>
<th>INSTITUTION</th>
<th>ACAD_PLAN</th>
<th>EFFDT</th>
<th>EFF_STATUS</th>
<th>DESCR</th>
<th>ACAD_PLAN_TYPE</th>
<th>ACAD_PROG</th>
<th>DEGREE</th>
</tr>
</thead>
<tbody>
<tr>
<td>BGSUN</td>
<td>AERO-BSTC</td>
<td>8/25/25</td>
<td>I</td>
<td>Aerotechnology</td>
<td>MAJ</td>
<td>TECH</td>
<td>BSTC</td>
</tr>
<tr>
<td>BGSUN</td>
<td>AERO-BSTC</td>
<td>1/1/1910</td>
<td>A</td>
<td>Aerotechnology</td>
<td>MAJ</td>
<td>TECH</td>
<td>BSTC</td>
</tr>
</tbody>
</table>

The above rows indicate that the Aerotechnology major, with a Bachelor of Science in Technology degree (DEGREE = ‘BSTC’) upon graduation, is active (EFF_STATUS = ‘A’) until 8/24/2025. On 8/25/2025, it will become inactive (EFF_STATUS = ‘I’). This means that AERO-BSTC will be inactive in the future but is active now. (This is an example and does not necessarily reflect the fate of the Aerotechnology major.)

<table>
<thead>
<tr>
<th>INSTITUTION</th>
<th>ACAD_PLAN</th>
<th>EFFDT</th>
<th>EFF_STATUS</th>
<th>DESCR</th>
<th>ACAD_PLAN_TYPE</th>
<th>ACAD_PROG</th>
<th>DEGREE</th>
</tr>
</thead>
<tbody>
<tr>
<td>BGSUN</td>
<td>ENVHLTH-BS</td>
<td>6/18/11</td>
<td>I</td>
<td>Environmental Health</td>
<td>MAJ</td>
<td>ARTSC</td>
<td>BS</td>
</tr>
<tr>
<td>BGSUN</td>
<td>ENVHLTH-BS</td>
<td>1/1/1915</td>
<td>A</td>
<td>Environmental Health</td>
<td>MAJ</td>
<td>ARTSC</td>
<td>BS</td>
</tr>
</tbody>
</table>

The second example shows that plan ENVHLTH-BS – a Bachelor of Science for Environmental Health in the College of Arts and Sciences – was active until 6/17/2011 and was discontinued on 6/18/2011, placing it in an inactive status. This means that ENVHLTH-BS is currently inactive. (There is an equivalent ENVHBSENVH under program HLTH that is still active.)
Using Effective Data in Queries

The most common use of effective date logic in queries is to report only active data current at the time the query is run. This may be referred to as the “maximum non-future effective date.”

Writing effective date logic manually involves adding a subquery on the same record as in the main query, joining on all key fields except for the date. This can be cumbersome for records with many fields in the key.

Query Manager makes this unnecessary! When you add a record that has an Effective Date field to a query, an effective date criterion is automatically added to the query. A message stating this appears as soon as you add that record. You can see this criterion in the Criteria tab as “Eff Date <= Current Date.” You can change the criterion to look at the date in a field, expression, or a specific date of your choice. You can also show rows having the first effective date or last effective date regardless of whether the effective date is in the future.

If the record also has an Effective Sequence field, the criterion will be “Eff Date <= Current Date (EffSeq = Last),” indicating that only the last row created on the effective date will be included in the results. You can edit this criterion to use the first row instead of the last or to show all rows regardless of Effective Sequence.

If you accidentally delete a criterion on Effective Date, you can add one manually. If you choose EFFDT as a field in the criterion, you can choose from special Condition Types that apply only to effective dates:

- Eff Date < – effective date is less than the selected date
- Eff Date <= – effective date is less than or equal to the selected date
- Eff Date > – effective date is greater than the selected date
- Eff Date >= – effective date is greater than or equal to the selected date
- First Eff Date – effective date is the earliest for the key
- Last Eff Date – effective date is the latest for the key

The selected date can be the current date, a constant, a value in a field, or the result of an expression. Recall that effective dates are tied to keys, so “First Eff Date” refers to the row having the earliest effective date for all rows having the same values in their key fields (except Effective Date and Effective Sequence).

Query Manager does not automatically add criteria on Status of Effective Date. If you are interested in only active or only inactive rows, you must manually add the criterion on EFF_STATUS to your query. There are just two possible values of EFF_STATUS: ‘A’ for active and ‘I’ for inactive.
Query Organization

As the number of queries that you work with grows, you may want to group them by purpose, department, or other characteristics. You might also want to send them to other users, rename them, or delete old ones.

Query Manager enables you to organize your queries in a manner similar to how you may organize your files with Windows XP, Windows 7, and Mac OS X. However, there are some important differences.

Files in a file system are uniquely identified by an internal ID. Two or more files can have the same name, though such files usually must be stored in different folders. Queries are uniquely identified by name in a storage area; two queries in the same user's storage space cannot share the same name even if they are in different folders. Two queries can only have the same name if they are in two different users’ private storage. No private query can have the same name as a public query.

Queries may be grouped into folders. This can aid in finding queries in a large institution, since you can search for only those queries that are in a particular folder. This is similar to storing files in different folders or directories in a file system. However, unlike with file systems, you cannot store a folder inside another folder. With Query Manager, there are only two levels of folders: inside a folder and not inside a folder.

With a file system, you must create new folders and delete unneeded folders manually. Query Manager implicitly creates a folder when you save or move a query to a folder that does not already exist. Query Manager also automatically deletes a folder when you remove the last query from that folder.
Copy a Query to a User

It may be useful to send a query to another user if that user needs to make minor modifications before running it, to see how it was built, or to run the query with different row-level security than that of the first user. Query Manager enables you to copy a query to another user’s storage space.

To copy a query to another user, do the following:

- From the Query Manager search page, search for the query to be copied.

In the row for the query in the Search Results area, check the Select checkbox. (You may copy multiple queries by checking the checkbox corresponding to each query to be copied.)

- From the Action dropdown, choose Copy to User.
- Click the Go button.

- Enter the User ID of the user who will receive the query.
- Click the OK button.

If the copy is successful, you will receive a message confirming this.
If the receiving user does not have access to the records in the copied query, that query will not appear when the receiving user searches for queries.

A public query cannot be copied to another user because this would result in a private query having the same name as a public query. If you want to copy a public query, the receiving user must edit the query and save it to his or her private storage. (See the Save a Query with a New Name segment.)

A query cannot be copied to another user if that user has a query with the same name; you cannot overwrite an existing query by copying one.
Delete a Query

Deleting unneeded queries, such as those that are out of date, or created as a test, can be done easily through Query Manager. To delete a query, do the following:

- From the Query Manager search page, search for the query to be deleted.

In the row for the query in the Search Results area, check the Select checkbox. (You may delete multiple queries by checking the checkbox corresponding to each query to be deleted.)

- From the Action dropdown, choose Delete Selected.
- Click the Go button.

When asked to confirm the deletion, click the Yes button.

If the last query is deleted from a folder, that folder is also deleted.

It is possible for you to delete a public query. Be very careful when selecting queries to delete since you may accidentally delete a query someone else needs!
Move a Query to a Folder

It can be useful to group queries with a similar purpose or for the same department or college into a folder. Both Query Viewer and Query Manager allow you to search for queries by a folder name and filter the results of a search for queries by a folder name. To move a query to another folder, do the following:

- From the Query Manager search page, search for the query to be moved.

In the row for the query in the Search Results area, check the Select checkbox. (You may move multiple queries by checking the checkbox corresponding to each query to be moved.)

- From the Action dropdown, choose Move to Folder.
- Click the Go button.

A Move to Folder page appears.

- To move the selected query or queries to an existing folder, click the “Select an existing folder to move to” radio button and choose a folder from the dropdown.

- To move the selected query or queries to a new folder, click the “OR enter a folder name to move to” radio button and enter a folder name in the text box. A folder name may be at most 18 characters long.

*Make sure to both click a radio button and choose a folder

- Click the OK button.
*Query Manager will take the action indicated by the radio button. If you enter a new folder name but do not click the second radio button, the query will be moved to whatever existing folder is showing in the dropdown!

If the last query in a folder is moved out of that folder, the folder is deleted. If a new folder name is entered and a query is moved to it, a folder with that name is automatically created.

To move a query so that it is not in any folder, move it to a folder with a blank name.

Private queries remain private after being moved. Similarly, public queries remain public after being moved.

It is possible for you to move a public query to another folder. Be very careful when selecting queries to move since you may accidentally move a query someone else needs!
Rename a Query

On occasion, you may want to change the name of a query to make it easier to identify, correct a spelling mistake, or reduce confusion with another query. To rename a query, do the following:

1. From the Query Manager search page, search for the query to be renamed.
2. In the row for the query in the Search Results area, check the Select checkbox. (You may rename multiple queries by checking the checkbox corresponding to each query to be renamed.)
3. From the Action dropdown, choose Rename Selected.
4. Click the Go button.
5. Enter a new name for each selected query next to the old name of each query.
6. Click the OK button.

Renaming a query does not change who owns the query or the folder in which it is stored.

You cannot rename a query such that it has the same name as a private query in your storage area or a public query.

It is possible to rename a public query. Be very careful when selecting queries to rename since you may accidentally rename a query someone else needs.
Save a Query with a New Name

There are instances in which you may want to give a query a new name while keeping the existing query intact instead of renaming it. These include testing changes to a query and retaining historical versions of a query. To save a query with a new name, do the following:

- From the Query Manager search page, search for the query to be saved with a new name.
- In the row for the query in the Search Results area, **click** the *Edit* link.

At the bottom of any tab page except Run, **click** the *Save As* link.

- **Enter** a new name in the Query text box.
- **Enter** a new description or folder name or change the Owner between Private and Public if desired.
- **Click** the *OK* button.

You can overwrite an existing query by saving a query with the same name as an existing query. If you attempt this, you will be told a query with that name exists and asked if you want to continue. **Click Yes** to overwrite or **No** to abandon the save.

You cannot save a query to your private storage such that the query has the same name as a public query.

No change is made to the original query.
It is possible for you to overwrite a public query by saving a query with the same name as another public query. Be very careful when saving a public query since you may accidentally overwrite a query someone else needs.
Grouping Criteria and the OR operator

- On some occasions, you will need to write a query that returns rows that meet some criteria, but not all of them. For instance, you may be asked to limit the results to those for which the program status is either ‘AC’ or ‘LA’. This could be implemented using the criterion “PROG_STATUS in list ‘AC’, ‘LA’”, but if the criteria involve multiple fields or cannot be enumerated in a list, you will need to create multiple criteria and link them with the OR operator.

- When you add criteria to a query, by default they are linked with the AND operator. This means that in order for a row to be included in the results, criterion 1 and criterion 2 and criterion 3 etc. must all be met. With an OR operator, if criterion 1 is true or criterion 2 is true, then the row will be included (assuming the other criteria are also met). Note that if both criterion 1 and criterion 2 are true, the row will still be in the results; it is only required that one of the criteria be met.

Choosing Logical Operators

- To change the operator linking two criteria, go to the Criteria tab and select the operator from the Logical column. For the first criterion, you can only select NOT to negate that criterion. For the other criteria, you can choose AND, AND NOT, OR, and OR NOT. Typically you will use this to change from AND to OR.

- In the screen shot below, the AND operator links the criterion above (A.EFFDT <= Current Date) and the criterion in the same row (A.INSTITUTION equal to BGSUN).

- The operators are called “Logical” because they operate on values that are either true or false. Each criterion has a comparison that produces a true result or a false result. In a row in which INSTITUTION is “BGSUN,” the criterion A.INSTITUTION equal to BGSUN evaluates to true.
Grouping Criteria

- There is an important consideration when using both the AND and OR operators in a query. As in mathematics, there is a defined order of operations. The AND operators are given higher priority than OR operators. The AND operator is roughly equivalent to multiplication and the OR operator is roughly equivalent to addition in this sense. You can use parentheses to affect how the criteria are grouped, usually with the goal of combining the criteria linked with OR operators.

- Consider an example in which you are asked to list the academic program (ACAD_PROG record) information about students who are at BGSU ('BGSUN') in the Arts and Sciences program (ACAD_PROG = 'ARTSC') and were either admitted in Fall 2012 (ADMIT_TERM = '2128') or completed the program in Spring 2012 (COMPLETION_TERM = '2122'). Note that academic program data is effective dated.

- Here is a first attempt at building these criteria in our query:

  ![Criteria Table]

  Below is an excerpt from the results returned by the query:

  ![Result Table]

  - Row 201 matches what was expected since the program is ARTSC and the admit term is Fall 2012. However, row 202 doesn’t match what was intended; while the completion term is Spring 2012, the program is HLTH. This row was included because it meets the COMPLETION_TERM equal to 2122 criterion. Recall that in a case of criteria linked by an OR operator, either what is before the OR or after the OR must be true to be included in the results. There are four criteria linked by AND operators before the OR, so either these four criteria in combination must be true or the one criterion after the OR have to be met for the row to appear among the results.
In order to have the query worked as intended – that the results are all in the ARTSC program and that either the admit term is Fall 2012 or the completion term is Spring 2012 – parentheses are needed. To add parentheses to a query, go to the Criteria tab and click the Group Criteria button. The Edit Criteria Grouping page appears.

In the column between Logical and Expression 1, you enter an opening parenthesis "(" before the first criterion in the group that you want to create. In the column to the right of Expression 2, you enter a closing parenthesis ")" after the last criterion in the group.

In this case, we are creating a group containing the criterion on ADMIT_TERM and the criterion on COMPLETION_TERM. Parentheses are added are indicated in the screen shot below:
• Click OK to confirm the changes to the grouping. The parentheses are displayed on the Criteria page.

• This shows that the two criteria on ADMIT_TERM and COMPLETION_TERM are now to be evaluated first. If either ADMIT_TERM is equal to 2128 or COMPLETION_TERM is equal to 2122, the result of the OR operation will be true. If neither of these are true, the result of the OR operation will be false, which will cause the row to be excluded.

• Running the modified query produces results like the following:

• Observe that all of the rows in the result set have either the admit term of Fall 2012 or the completion term of Spring 2012, and that no matter which of the two terms match the criteria, the academic program is ARTSC.

• In general, if you are going to use the OR operator in a query, you will likely need to group criteria together using parentheses.
**Wildcards**

Wildcards are symbols that substitute for other characters in search strings. They act much like "wild" cards that can be used as if they were any other card in a card game.

The primary use of wildcards in queries is to find rows in which a text field *contains* a string rather than *equals* a string. You may want to find instances in which that text has a particular word or a set of consecutive characters.

In the *Edit Criteria – Part 2* segment, you learned about the "like" condition type, which allows you to find rows in which a text field is "like" a word or phrase, meaning the field contains that word or phrase. To use like in this manner, you must employ wildcards. (If you have no wildcards in your constant, then like is the same as equal!)

There are two wildcards that you can use in queries. The percent sign (%) is used to substitute for *zero or more* characters. The underscore (_) is used to substitute for *any single character*.

The use of wildcards is best demonstrated through examples. Consider the following search strings that use wildcards and some string that would match them.

<table>
<thead>
<tr>
<th>Search String</th>
<th>Value in Field</th>
<th>Match?</th>
<th>Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>abc%</td>
<td>abc</td>
<td>Yes</td>
<td>Starts with &quot;abc&quot; and is followed by zero characters</td>
</tr>
<tr>
<td></td>
<td>abcd</td>
<td>Yes</td>
<td>Starts with &quot;abc&quot; and is followed by one character</td>
</tr>
<tr>
<td></td>
<td>abcdedefgh</td>
<td>Yes</td>
<td>Starts with &quot;abc&quot; and is followed by many characters</td>
</tr>
<tr>
<td></td>
<td>ab</td>
<td>No</td>
<td>Does not start with &quot;abc&quot;</td>
</tr>
<tr>
<td></td>
<td>abz</td>
<td>No</td>
<td>Does not start with &quot;abc&quot;</td>
</tr>
<tr>
<td></td>
<td>aabc</td>
<td>No</td>
<td>Does not start with &quot;abc&quot; (even though &quot;abc&quot; is in the value)</td>
</tr>
<tr>
<td>%def</td>
<td>def</td>
<td>Yes</td>
<td>Begins with no characters and ends with &quot;def&quot;</td>
</tr>
<tr>
<td></td>
<td>cdef</td>
<td>Yes</td>
<td>Begins with one character and ends with &quot;def&quot;</td>
</tr>
<tr>
<td></td>
<td>abcd</td>
<td>Yes</td>
<td>Begins with many characters and ends with &quot;def&quot;</td>
</tr>
<tr>
<td></td>
<td>ef</td>
<td>No</td>
<td>Does not end with &quot;def&quot;</td>
</tr>
<tr>
<td></td>
<td>ref</td>
<td>No</td>
<td>Does not end with &quot;ref&quot;</td>
</tr>
<tr>
<td></td>
<td>deff</td>
<td>No</td>
<td>Does not end with &quot;def&quot; (even though &quot;def&quot; is in the value)</td>
</tr>
<tr>
<td>%ghi%</td>
<td>ghi</td>
<td>Yes</td>
<td>Begins with no characters, followed by &quot;ghi,&quot; followed by no characters</td>
</tr>
<tr>
<td></td>
<td>efgi</td>
<td>Yes</td>
<td>Begins with several characters, followed by &quot;ghi,&quot; followed by no characters</td>
</tr>
<tr>
<td></td>
<td>ghijk</td>
<td>Yes</td>
<td>Begins with no characters, followed by &quot;ghi,&quot; followed by no characters</td>
</tr>
<tr>
<td></td>
<td>efgij</td>
<td>Yes</td>
<td>Begins with several characters, followed by &quot;ghi,&quot; followed by no characters</td>
</tr>
<tr>
<td></td>
<td>efgijhjk</td>
<td>Yes</td>
<td>Begins with &quot;efg,&quot; followed by &quot;ghi,&quot; ending with &quot;ijk&quot;</td>
</tr>
<tr>
<td></td>
<td>efgjk</td>
<td>No</td>
<td>Does not contain &quot;ghi&quot;</td>
</tr>
<tr>
<td></td>
<td>efghi</td>
<td>No</td>
<td>Does not contain &quot;ghi&quot;</td>
</tr>
<tr>
<td></td>
<td>gh</td>
<td>No</td>
<td>Does not contain &quot;ghi&quot;</td>
</tr>
<tr>
<td>a_c</td>
<td>abc</td>
<td>Yes</td>
<td>&quot;a&quot; followed by one character followed by &quot;c&quot;</td>
</tr>
<tr>
<td></td>
<td>a7c</td>
<td>Yes</td>
<td>&quot;a&quot; followed by one character followed by &quot;c&quot;</td>
</tr>
<tr>
<td></td>
<td>ac</td>
<td>No</td>
<td>No characters between &quot;a&quot; and &quot;c&quot;</td>
</tr>
<tr>
<td></td>
<td>abbc</td>
<td>No</td>
<td>Too many characters between &quot;a&quot; and &quot;c&quot;</td>
</tr>
</tbody>
</table>
Criteria and Case-Sensitive Data

Criteria that involve textual data use case-sensitive comparisons. This means that “hello” and “Hello” are two distinct values that are not considered to be equal. If your query has a criterion that DESCR is equal to “Psychology department” but the row for this department has a DESCR of “Psychology Department” (note the capital D), the row will not be included in the results.

By convention, codes such as those used for statuses and types are fully uppercase, to prevent issues of case sensitivity from arising and from having codes that are the same except for case. (It would be confusing to have “AC,” “Ac,” and “ac” all as valid options having different meanings for the same field!) In queries that use criteria involving codes, make sure to enter the code using all uppercase letters. (If the field is defined in PeopleSoft as allowing only uppercase letters, then your value will be transformed to uppercase. However, not every field that has codes is defined in this manner.)

Consider exercise 6B, in which you are obtaining a list of item types that have “Fine” in the description. The current data is set up such that “Fine” is always capitalized. What if this was not the case? How might you work around this?

One alternative is to have two criteria, one which checks if DESCR contains “Fine” and one which checks if DESCR contains “fine.”

```
Criteria | Expression 1 | Expression 2 | Condition | Expressions
---|---|---|---|---
  | A.DESCR - Description  | like | %Fine%  | Edit
```

Note that because of the other criteria in the query, it is necessary to group the two criteria on DESCR in parentheses, and use the OR operator instead of the AND operator on those criteria. (We want rows in which either the description contains “Fine” or the description contains “fine.”)

Another alternative is to use an expression to convert DESCR to a known case – either uppercase or lowercase – and compare that against a constant in the same case.

```
Criteria | Expression 1 | Expression 2 | Condition | Expressions
---|---|---|---|---
  | UPPER(A.DESCR)  | like | %Fine%  | Edit
```

In this example, we check if the uppercase version of DESCR, returned by the expression UPPER(A.DESCR), contains the text “FINE,” which is also in uppercase. Cases of “Fine” and “fine” will be matched since the comparison will be against their uppercase equivalent, which is “FINE” for both.
Key Fields

An important aspect of relational database structure is that information about an object is separated into multiple tables, with each table having data about a certain set of characteristics. For instance, in PeopleSoft, the PERSON record is the basis of personal information, the NAMES record contains the names of each person, and the ADDRESSES record has the postal and physical addresses of each person. The data about one person is spread in PERSON, NAMES, ADDRESSES, and other records.

In order to collect the data about a person, the relevant records must be joined together in a query. The database management system needs to know how to find the data in one record based on data in another. This is done through the use of key fields.

Primary Keys

A primary key is a field or set of fields that uniquely identifies an object. No other object can have the same primary key as another. No two cars have the same vehicle identification number (VIN). No two dollar bills have the same serial number. In PeopleSoft, no two people can have the same employee ID number, so the primary key of the PERSON record is EMPLID.

A primary key may contain multiple fields. For instance, the primary key of STATE_TBL is the combination of COUNTRY and STATE. The state abbreviation or code alone cannot uniquely identify a state or province; the code “MI” represents the province of Misiones in Argentina, the province of Milano in Italy, the state of Michigan in the United States, and the state of Miranda in Venezuela. However, all countries have a unique code, and no country has two states with the same code, so the country and state codes together can uniquely identify a state. The province of Misiones is identified by country code ARG and state code MI whereas Michigan is identified by country code USA and state code MI.

When tables have a parent-child relationship, the primary key of the parent is contained within the primary key of the child. This enables all children rows of a parent row to be found by joining the tables on the primary key fields. A query for gathering information from both COUNTRY_TBL and STATE_TBL would join these two tables on COUNTRY; the COUNTRY field is the primary key of COUNTRY_TBL and is part of the key of STATE_TBL, the child table of COUNTRY_TBL. When using a hierarchy join in Query Manager, the parent and child records are automatically joined on the fields their primary keys have in common.

The primary key is generally the first field or fields in the list of fields of a record. In Query Manager, the primary key is indicated on the Query tab by a key icon.
Foreign Keys

A foreign key is a field or set of fields that refer to the primary key of another table. These are typically used for restricting data in a field to only allowable values and to reduce data duplication. For example, in the ADDRESSES record, there is both a COUNTRY and a STATE field; these in combination are a foreign key to STATE_TBL. It will not be possible to enter a country/state combination that does not exist in STATE_TBL into an address.

In order to get information about a state, such as its full name, one would have to join a record to STATE_TBL using both COUNTRY and STATE. A related record join in Query Manager automatically joins the foreign key of one record to the primary key of another record.

Keep in mind that when you are manually joining two records by adding criteria that you will likely need to add them for every field in the foreign key. In the exercise in which DEPT_TBL was joined to PERSON_NAME, you had to add a criterion that requires that MANAGER_ID of DEPT_TBL was the same as EMPLID of PERSON_NAME; EMPLID is the primary key of PERSON_NAME.
Keys with Effective Dates

The PeopleSoft database design includes the concept of effective data (see Effective Date, Effective Sequence, and Effective Status elsewhere in this manual), in which information about an object is recorded at different points in time, and the history about that object can be retrieved.

When effective data fields are in a record, they are considered to be part of the primary key. This is because the date (and sequence, if used) helps uniquely identify the information about an object at a point in time.

Consider the following rows of ACAD_PROG, which contains data about the programs in which a student is participating and has participated.

<table>
<thead>
<tr>
<th>EMPLID</th>
<th>ACAD_CAREER</th>
<th>STDNT_CAR_NBR</th>
<th>EFFDT</th>
<th>EFFSEQ</th>
<th>INSTITUTION</th>
<th>ACAD_PROG</th>
<th>PROG_STATUS</th>
<th>PROG_ACTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>1379</td>
<td>UGRD</td>
<td>0</td>
<td>1/12/2012</td>
<td>12</td>
<td>BGSUN</td>
<td>EDUC</td>
<td>AC</td>
<td>PLNC</td>
</tr>
<tr>
<td>1379</td>
<td>UGRD</td>
<td>0</td>
<td>1/7/2011</td>
<td>0</td>
<td>BGSUN</td>
<td>EDUC</td>
<td>AC</td>
<td>PRGC</td>
</tr>
<tr>
<td>1379</td>
<td>UGRD</td>
<td>0</td>
<td>8/20/2010</td>
<td>0</td>
<td>BGSUN</td>
<td>ACEN</td>
<td>AC</td>
<td>ACTV</td>
</tr>
</tbody>
</table>

Student 1379 entered the Academic Enhancement program on 8/20/2010, then switched to Education & Human Development on 1/7/2011, then changed a plan within that program on 1/12/2012. All three of the rows are about the same student’s participation, but represent different states of that participation by date. The EFFDT and EFFSEQ fields help uniquely identify the state of participation – by using these fields, one can find out which program a student was in at a given point in time.

The full primary key of ACAD_PROG is the combination of EMPLID, ACAD_CAREER, STDNT_CAR_NBR, EFFDT, and EFFSEQ.

When joining records that have effective data fields, one must be careful to consider whether or not it is appropriate to join on EFFDT (and EFFSEQ, if present). If a parent has effective data fields and it is being joined to a child record, then the join must include EFFDT. For example, the service indicator code record SRVC_IND_CD_TBL has a key of INSTITUTION, SRVC_IND_CD, and EFFDT. The reasons belonging to service indicators are in SRVC_IN_RSN_TBL and have a key of INSTITUTION, SRVC_IND_CD, EFFDT, and SRVC_IND_REASON. To join information on codes and reasons together, one must join on INSTITUTION, SRVC_IND_CD, and EFFDT, to ensure that the reason data is linked to the proper code.

However, when there is not a parent-child relationship, then EFFDT will not be part of the join. Data about academic plans is in ACAD_PLAN_TBL and the key is INSTITUTION, ACAD_PLAN, and EFFDT. Data about subplans is in ACAD_SUBPLN_TBL and the key is INSTITUTION, ACAD_PLAN, ACAD_SUB_PLAN, and EFFDT. The key of ACAD_PLAN_TBL is not contained in the key of ACAD_SUBPLN_TBL; the effective date in ACAD_PLAN_TBL is
about the plan whereas the effective date in ACAD_SUBPLN_TBL is about the subplan. To join these two records, use INSTITUTION and ACAD_PLAN alone.

Query Manager automatically takes care of these considerations. When performing a hierarchy join, the EFFDT (and EFFSEQ, if needed) field used as part of the join. When performing a related record join, EFFDT will be left out of the join. With a manual join (such as between ACAD_PLAN_TBL and ACAD_SUBPLN_TBL), Query Manager will not detect EFFDT as a possible field for the join criteria that can be automatically added.

**What is a View?**

A “view” is a special kind of query. It is a query that acts like a table in queries. Many views are delivered with PeopleSoft and developers can add their own. (However, this cannot be done through Query Manager.) Views are created for many purposes:

- Save a commonly-used subquery so that it does not need to be rewritten in several queries
- Reduce the number of fields returned from a table
- Return only current rows so that effective date logic need not be added to the main query
- Join data from many tables together into one convenient result set that can be used as if it were a table

In Query Manager, tables and views are both considered records, so you will see no difference in how they are used. By convention, views in PeopleSoft often have names ending with “VW,” though some have names just containing “VW.” You may find it beneficial to use views when they are available, as they can simplify your queries by hiding some details like fields, joins, and effective date logic.

**Expressions**

Recall from the Query Manager class that expressions are calculations, usually performed on fields from the records in your query, which produce a result. The result can be text, a date, or a number, and may be displayed and used in criteria as if it were a real field.

Reference fields in expressions by using their actual name, not their description and preceding the name with their alias and a period. For example, if you have SAL_GRADE_TBL aliased as “A” and want to use field MID_RT_ANNUAL in your expression, refer to the field as “A.MID_RT_ANNUAL” (without the quotes). Reference prompts by using a colon followed by the prompt number, such as “:1” (without the quotes).

One type of calculation that can be performed is mathematical. You can add, subtract, multiply and divide numeric values. The mathematical operators are:

- + (addition)
- - (subtraction)
- * (multiplication)
- / (division)
Standard mathematical order of operations (multiplication and division are done before addition and subtraction; computation goes from left to right) is used. Parentheses can be used to group calculations together and force them to be done out of this order.

The only operator that applies to strings is concatenation, which is appending one string to the end of another string. The concatenation operator is the double pipe, ||. (The pipe character can usually be entered by holding down the Shift key while pressing the backslash \ key, which is normally above the Enter key.) For example, if A.STR1 contains ‘ABC’ and A.STR2 contains ‘DEFG’, then A.STR1 || A.STR2 results in ‘ABCDEFG’.

There are not operators that perform calculations on two dates. You can add to or subtract from a date. The numeric value is translated into a number of days. If A.START_DT is 2/13/2012 then the expression A.START_DT + 7 results in 2/20/2012. If the numeric value is a decimal amount, then the time will also be affected. If A.START_DT is 2/13/2012 10:00 AM then the expression A.START_DT + 0.5 results in 2/13/2012 10:00 PM.

Expressions can also contain functions, which take some input values and return an output value. Functions operate similar to how they do in mathematics. The familiar square root operation can be considered a function; it takes an input value and returns the value that, when multiplied by itself, produces the original value. Some common mathematical functions are even written similar to how functions appear in queries; the sine function in trigonometry is written as \( \sin(x) \), indicating that it accepts one value.

Functions in expressions are of the form FUNCTION_NAME(arg1, arg2, ..., argN), where FUNCTION_NAME is the name of the function and arg1, arg2, etc. are its arguments, which are the input values. When using a function in an expression, the name must be spelled exactly as expected, the argument list must be enclosed in parentheses, all arguments must be separated by commas, and all required arguments must be given. (There are sometimes optional arguments. In documentation these are frequently indicated by enclosing the argument in [square brackets].)

*** Note on functions: the functions listed below are for use in Oracle databases version 11 and higher; BGSU is currently using Oracle 11g. Other databases, such as DB/2, Informix, and Sybase, may support different functions.

**String Functions**

Here are some functions that can be used on strings:

**CONCAT(str1, str2)**

The CONCAT function concatenates str1 and str2, returning str1 followed immediately by str2. Equivalent to str1 || str2.

**LENGTH(str1)**

LENGTH returns the length of str1 in number of characters. For example, LENGTH('ABC DEF') returns 7 (the space is included).
LOWER(str1)

LOWER returns a copy of str1 with all uppercase letters changed to lowercase.

SUBSTR(str1, start[, how_many])

SUBSTR returns the substring – a part of a string – of str1, starting at position start. If how_many is not provided, all characters from start through the end of the string are returned. If how_many is provided, the substring beginning at position start and having a length of how_many are returned.

Examples:
- SUBSTR('ABCDEFG', 1, 3) returns 'ABC'
- SUBSTR('ABCDEFG', 3, 4) returns 'CDEF'
- SUBSTR('ABCDEFG', 4) returns 'DEFG'

UPPER(str1)

UPPER returns a copy of str1 with all lowercase letters changed to uppercase.

**Numeric Functions**

Here are some functions that can be used on numeric values:

ABS(n)

ABS returns the absolute value of n. If n is positive or zero, the result is n. If n is negative, the result is n without the negative sign. For example, ABS(-5) returns 5.

CEIL(n)

CEIL returns the ceiling of n, which is the smallest integer equal to or greater than n. For example, CEIL(5.2) is 6, since 5 is not equal to or greater than 5.2, but 6 is greater than 5.2. Note that CEIL(-4.4) is -4. CEIL is equivalent to rounding up.

FLOOR(n)

FLOOR returns the floor of n, which is the largest integer equal to or less than n. For example, FLOOR(8.7) is 8, since 9 is not equal to or less than 8.7, but 8 is less than 8.7. Note that FLOOR(-3.2) is -4. FLOOR is equivalent to rounding down.

ROUND(n[, dec])

ROUND returns n rounded to dec decimal places, or to the nearest integer if dec is not provided. If dec is negative, the value is rounded to dec powers of 10 (-1 to the nearest 10, -2 to the nearest 100, etc.).

Examples:
- ROUND(12.3) returns 12
• ROUND(12.8) returns 13
• ROUND(-12.3) returns -12
• ROUND(-12.8) returns -13
• ROUND(12345.6789, 2) returns 12345.68
• ROUND(12345.6789, 1) returns 12345.7
• ROUND(12345.6789, 0) returns 12346
• ROUND(12345.6789, -1) returns 12350
• ROUND(12345.6789, -2) returns 12300

**SQRT(n)**

SQRT returns the square root of n. The square root is the number, which multiplied by itself, results in n.

**TRUNC(n[, dec])**

TRUNC returns n truncated to dec decimal places. If dec is zero or not provided, all digits after the decimal are dropped. If dec is negative, ABS(dec) digits to the left of the decimal are replaced by zero, and all digits after the decimal are dropped.

Examples:

• TRUNC(12.3) returns 12
• TRUNC(12.8) returns 12
• TRUNC(-12.3) returns -12
• TRUNC(-12.8) returns -12
• TRUNC(12345.6789, 2) returns 12345.67
• TRUNC(12345.6789, 1) returns 12345.6
• TRUNC(12345.6789, 0) returns 12345
• TRUNC(12345.6789, -1) returns 12340
• TRUNC(12345.6789, -2) returns 12300

**Date Functions**

Here are some functions that can be used on dates:

**ADD_MONTHS(dt, m)**

ADD_MONTHS returns the date that is m months in the future of dt.

**SYSDATE**

SYSDATE returns the current system date and time. Note that it does not take any arguments.

**Conversion Functions**

Here are some functions that can be used to convert values from one type (date, number, string) to another:
TO_CHAR(dt[, fmt])

TO_CHAR converts a date or part of a date into a string. If fmt is not provided, the date is returned in the default format for the system, usually 'DD-MON-YY'. (In this format, 2/13/2009 would be '02-FEB-09'.)

The fmt argument is a string containing codes that instruct TO_CHAR how to format the date. There are many different format codes and several can be used at one time. Some of the format codes are:

- AM – the AM/PM indicator
- DD – day of the month
- HH – hour of the day (1-12)
- HH24 – hour of the day (0-23)
- MI – minute of the hour (0-59)
- MM – month (1-12, with 1 = January)
- MON – three-character abbreviation of the month
- SS – second of the minute (0-59)
- YY – two-digit year
- YYYY – four-digit year

Punctuation such as dashes, colons, and slashes are included in the result string in the positions given in the fmt.

Assume the current date is March 15, 2009 and the time is 4:25 PM. Here are several examples of how TO_CHAR would display this date:

- TO_CHAR(SYSDATE, 'MM/DD/YYYY') returns 03/15/2009
- TO_CHAR(SYSDATE, 'DD/MM/YYYY') returns 15/03/2009
- TO_CHAR(SYSDATE, 'YYYYMMDD') returns 20090315
- TO_CHAR(SYSDATE, 'HH:MI') returns 04:25
- TO_CHAR(SYSDATE, 'HH24:MI') returns 16:25
- TO_CHAR(SYSDATE, 'MM/DD/YYYY HH24:MI:SS') returns 03/15/2009 16:25:00
- TO_CHAR(SYSDATE, 'MM') returns 03 (this could be used for grouping in order to aggregate results by month!)

TO_CHAR(n[, fmt])

TO_CHAR can also convert a number to a string. If fmt is not provided, a default format will be used. This function is typically used to format numbers with commas, periods, currency symbols, and leading zeroes.

The fmt argument is a string containing codes that instruct TO_CHAR how to format the number. There are many different format codes and several can be used at one time. Some of the format codes are:

- $ – display a dollar sign
• 0 – display a leading zero
• 9 – display a digit
• , – display a comma
• . – display a period
• FM – disable leading and trailing spaces

Here are some examples of using TO_CHAR to format a number:

• TO_CHAR('1234567.89') returns 1234567.89
• TO_CHAR('1234567.89', '9,999,999') returns 1,234,568 (note the automatic rounding!); there is a leading space in front of the number to leave room for a negative sign
• TO_CHAR('123', '099999') returns 000123; there is a leading space in front of the number to leave room for a negative sign
• TO_CHAR('123', 'FM099999') returns 000123; there is no leading space
• TO_CHAR('12345.60', '$999,999.99') returns $12,345.60; there are two leading spaces, one to leave room for the sign and one because there is no hundred-thousands digit
Condition Functions

These are functions that return different values based on a set of conditions that you specify.

**DECODE(x, val1, [val2, result2, ..., valN, result] [, default])**

DECODE is used to produce a result that is based on the result of x. The x, val1, result1, etc. arguments can be numbers or strings. Up to 127 comparisons (value/result pairs) are allowed if there is not default value and up to 136 comparisons are allowed if there is a default.

The value of x is compared against val1, val2, etc. through valN, one at a time. When a match occurs, the corresponding result argument is returned. If x matches val1, DECODE returns result1, if x matches val2, DECODE returns result2, and so on. If there is no match, the default argument is returned, provided it is given; otherwise the special value NULL is returned.

Consider the following examples:

- **DECODE(A.COLOR, 'R', 'Red', 'G', 'Green', 'B', 'Blue')**
  - If A.COLOR is "R" then DECODE returns "Red"
  - If A.COLOR is "G" then DECODE returns "Green"
  - If A.COLOR is "B" then DECODE returns "Blue"
  - If A.COLOR is "Y" then DECODE returns NULL

- **DECODE(A.DIRECTION, 0, 'North', 90, 'East', 180, 'South', 270, 'West', 'Unknown')**
  - If A.DIRECTION is 0 then DECODE returns "North"
  - If A.DIRECTION is 90 then DECODE returns "East"
  - If A.DIRECTION is 180 then DECODE returns "South"
  - If A.DIRECTION is 270 then DECODE returns "West"
  - If A.DIRECTION is 360 then DECODE returns "Unknown"

You can also use the DECODE function to substitute a value when a field is blank. For date fields, an empty value is the special value NULL. To replace NULL values from the field A.DATE_FIELD with a default of date of 5/22/2012, use DECODE(A.DATE_FIELD, NULL, TO_DATE('05/22/2012', 'MM/DD/YYYY'), A.DATE_FIELD). Note the second A.DATE_FIELD as the last argument; this indicates that the original value of DATE_FIELD whenever DATE_FIELD is not NULL.

For text fields, an empty value is a single space. To replace a single space from the field A.TEXT_FIELD with the default string "N/A," use DECODE(A.TEXT_FIELD, ' ', 'N/A', A.TEXT_FIELD). As with the previous example, the original value of the field is used when the field is not empty.
**CASE Expression**

A CASE expression is a powerful tool for producing one of several possible results based on conditions. It is more flexible than DECODE, which just maps input values to output values. CASE expressions allow ranges to be mapped to output values and for multiple input fields to influence the output.

CASE expressions have the following format:

```
CASE
  WHEN condition1 THEN result1
  [WHEN condition2 THEN result2]
  ...
  [ELSE default_result]
END
```

The CASE and END keywords are required. Each possible case is designated by the WHEN keyword, followed by a condition, the THEN keyword, and the result. The result must be a value or an expression that evaluates to a single value. The condition can be simple or complex, with many Boolean operators (such as AND and OR), as long as it evaluates to a result of true or false.

At least one WHEN clause must be given. All other WHEN clauses and the ELSE clause are optional. However, it is good practice to provide a default case with the ELSE clause in case unexpected values are encountered.

The conditions are examined in order from the first WHEN clause to the last WHEN clause. If condition1 is true, then the result of the CASE expression is set to result1. If condition1 is false, then condition2 is examined (if given); if condition2 is true, then the result of the CASE expression is set to result2. Each condition is examined until one that is true is encountered. If none of the conditions are true, the result of the CASE expression is set to default_result if an ELSE clause is provided; otherwise, the expression is set to NULL.

Consider this example of determining if a state is a Great Lakes state:

```
CASE
  WHEN A.STATE IN ('OH', 'MI', 'IN', 'IL', 'WI', 'MN', 'NY', 'PA') THEN 'Y'
  ELSE 'N'
END
```

If the STATE field contains one out of several state codes (OH, MI, etc.) then the CASE expression evaluates to “Y”. Otherwise, it evaluates to “N”. 
Next, consider a version of this expression that takes into account that the Canadian province of Ontario is considered part of the Great Lakes region:

```
CASE
  WHEN A.COUNTRY = 'CAN' AND A.STATE = 'ON' THEN 'Y'
  WHEN A.COUNTRY = 'USA' AND A.STATE IN ('OH', 'MI', 'IN', 'IL', 'WI', 'MN', 'NY', 'PA') THEN 'Y'
  ELSE 'N'
END
```

Each of the conditions includes an AND operator and examines data in two fields, COUNTRY and STATE. The conditions can include many conditions combined together.

The above could also be expressed as the following:

```
CASE
  WHEN
    (A.COUNTRY = 'CAN' AND A.STATE = 'ON') OR
    (A.COUNTRY = 'USA' AND A.STATE IN ('OH', 'MI', 'IN', 'IL', 'WI', 'MN', 'NY', 'PA'))
  THEN 'Y'
  ELSE 'N'
END
```

Here is an example with many conditions. This CASE expression translates a temperature in Fahrenheit into a description.

```
CASE
  WHEN A.TEMPERATURE < 30 THEN 'Very cold'
  WHEN A.TEMPERATURE BETWEEN 30 AND 44 THEN 'Cold'
  WHEN A.TEMPERATURE BETWEEN 45 AND 59 THEN 'Cool'
  WHEN A.TEMPERATURE BETWEEN 60 AND 69 THEN 'Mild'
  WHEN A.TEMPERATURE BETWEEN 70 AND 79 THEN 'Warm'
  WHEN A.TEMPERATURE BETWEEN 80 AND 89 THEN 'Hot'
  ELSE 'Very hot'
END
```

Assume that the value in A.TEMPERATURE is 53. The first condition, A.TEMPERATURE < 30, is examined and found to be false. The second condition, A.TEMPERATURE BETWEEN 30 AND 44, is examined and is also false. The third condition, A.TEMPERATURE BETWEEN 45 AND 59, is true. The result of the expression is “Cool”.

If the value of A.TEMPERATURE is above 89, none of the conditions in the WHEN clauses will be true, so the default value “Very hot” will be the result.

Note that since the order of the clauses is significant, the following CASE expression is equivalent:

```
CASE
  WHEN A.TEMPERATURE < 30 THEN 'Very cold'
  WHEN A.TEMPERATURE <= 44 THEN 'Cold'
  WHEN A.TEMPERATURE <= 59 THEN 'Cool'
  WHEN A.TEMPERATURE <= 69 THEN 'Mild'
END
```
WHEN A.TEMPERATURE <= 79 THEN 'Warm'
WHEN A.TEMPERATURE <= 89 THEN 'Hot'
ELSE 'Very hot'
END

What Else is There?

There are many more functions and formats beyond those listed here. The Oracle Database SQL Language Reference (http://docs.oracle.com/cd/E11882_01/server.112/e26088/toc.htm) is comprehensive, but may be difficult to read for the non-technical user. If you plan to frequently use functions in your queries, you may want to find a book on the Oracle dialect of SQL. There are plenty of Oracle SQL books, both hardcopy and electronic, available at the BGSU Jerome Library and through OhioLINK.

Blank and Unknown Values

In many instances, not every field in a row will have a value. This may occur because a value is not required and one was not available to be entered or the user chose not to enter one. The special keyword NULL is used in SQL to designate that a field has no value or an unknown value.

However, in PeopleSoft the convention for blank and unknown values is different. If a field is optional, then a blank field has a different representation based on its data type: date fields use NULL, numeric fields use zero, and text fields use a single space. This has the disadvantage of being unable to distinguish between a truly unknown value and a default value; for instance, if the value of a numeric field is zero, there is no way to know whether the value is actually zero or if the value is not known.

Outer joins introduce further complexity in dealing with blank fields. When two tables are connected in a left outer join, if there is no match in the right table for the join fields in the left table, then all fields selected from the right table are given a value of NULL. This occurs because the join takes place at the database level, outside of PeopleSoft, whereas indicators of unknown values like a single space and zero are applied within the PeopleSoft application. This means that the results of the outer join might have cases of both a NULL and a space in the same field in two different rows; the NULL indicates no matching row was found while a space indicates a match was found but the field value is unknown or empty.

Consider a case in which COUNTRY_TBL is outer-joined to STATE_TBL on the COUNTRY field. The COUNTRY, DESCR, and EU_MEMBER_STATE fields of COUNTRY and the STATE, DESCR, and NUMERIC_CD fields of STATE_TBL are displayed. The results will show all defined countries, along with each country’s states or provinces; if there are no states or provinces, the base information about the country is still shown.
To demonstrate that a text field that is empty from a matched row has a single space but a text field from an unmatched row has NULL, an expression is built using CASE and this expression is used as a field for display.
The CASE expression will substitute "<null>" whenever the NUMERIC_CD field is NULL (from an unmatched row) and "<space>" whenever the NUMERIC_CD field is a single space (from an empty field of a matched row).

The screen shot below is an excerpt from the results of the query.
Note how Bahamas has the value “<null>” in the Numeric Cd Edit field. This indicates that no rows were found in STATE_TBL meeting the outer join conditions A.COUNTRY = B.COUNTRY.

The Netherlands region of Rossum has the value “<space>” in the Numeric Cd Edit field. This indicates that there is a row in STATE_TBL meeting the outer join conditions, but that the NUMERIC_CD field has no data.

The Canadian province of Nunavut has an actual value in the Numeric Cd Edit field because there is a matching row in STATE_TBL and the NUMERIC_CD field is not blank.

The following lists summarize the handling of blank and unknown values in PeopleSoft:

- When you see an empty cell in the results of a query
  - If the field is a date or a number, the value is the default NULL assigned in an outer join
  - If the field is text, the value is either the default NULL assigned in an outer join or the single space that represents an empty text value
- To substitute alternate values when a field is empty or NULL, create an expression using CASE; see the Expressions section of the Supplemental Material for examples.
- To write criteria to find blank/unknown values
  - For dates, use the Condition Type “is null.”
  - For numbers, use the Condition Type “equal to,” select Constant for Expression 2 Type, and enter the number 0 in the Constant text box. (There is no way to distinguish a true zero from a missing value.)
  - For text, use the Condition Type “equal to,” select Constant for Expression 2 Type, and enter nothing in the Constant text box. When you see this criterion in the Criteria page, it will display Expression 2 as ' '.
- To write criteria to find rows from the left table of an outer join that have no match in the right table, use any field from the right table and the Condition Type “is null.” Place this criterion on the WHERE clause, not the ON clause, even though it involves the table being outer-joined. This is because the criterion is intended to find results in which a field has the NULL value, not rows in the table being outer-joined in which that field has the NULL value.
- The “is null” Condition Type can also be used to find rows in which the result of a DECODE function is NULL because none of the values in it were matched; see the DECODE function in the Expressions section of the Supplemental Material for more information.
Make sure that you use the special Condition Type “is null” rather than “equal to” the value NULL. If you do the latter, you are telling the database system you want all rows in which the field is the word “NULL” rather than the indicator NULL.

**Structured Query Language (SQL)**

The standard language for writing queries to retrieve and manipulate the data in a relational database is the Structured Query Language, abbreviated SQL. When you use Query Manager to construct a query, a SQL statement is built automatically, and it is this statement that the database management system executes to return data to you.

While it is not necessary to be able to read or write SQL statements to use Query Manager, a basic understanding of SQL can help with troubleshooting queries by revealing the details that Query Manager hides.

**Basic Query**

A basic query in SQL has the following form:

```sql
SELECT <display field/expression list>
FROM <table/record list>
WHERE <criteria>
ORDER BY <sort field/expression list>
```

The keywords in all capital letters indicate the general structure of a query. You select the data that you want to retrieve from one or more tables where certain conditions are met and order the results by particular fields.

Consider the following example query that retrieves the codes and descriptions of all countries and states in which the country is a member of the European Union and sorts the results by country name and state name.

```sql
SELECT A.COUNTRY, A.DESCR, B.STATE, B.DESCR
FROM PS_COUNTRY_TBL A, PS_STATE_TBL B
WHERE A.COUNTRY = B.COUNTRY
AND A.EU_MEMBER_STATE = 'Y'
ORDER BY A.DESCR, B.DESCR
```

The FROM clause lists the tables (corresponding to records in PeopleSoft terminology) from which the data is being retrieved. The two tables in this case are PS_COUNTRY_TBL and PS_STATE_TBL. These have been given aliases of A and B, respectively. The tables are referred to by alias throughout the rest of the query. Note that all tables in the FROM clause are joined together, with the join conditions given in the WHERE clause. The joins are inner joins unless otherwise specified; this is explained below.

Most tables in PeopleSoft have the same name as the corresponding record, but the name is preceded by “PS_.” Thus, the table corresponding to ADDRESSES is PS_ADDRESSES. Tables that contain internal PeopleSoft information do not necessarily follow this convention.
Aliases do not have to be single letters (A, B, C, etc.). They can be abbreviations. Query Manager uses single letters; you are therefore limited to a maximum of 26 records in a query.

The list of fields being displayed is given as a series of field names, separated by commas, in the SELECT clause. The field names are fully qualified because they are preceded by a table alias (A or B) and a period. For instance, B.STATE indicates that STATE should be taken from table PS_STATE_TBL, which has the alias B. Using fully qualified names is not required except when a field being referenced exists in multiple tables used within the query, in which case it is needed to distinguish between such fields. In the query above, a reference to COUNTRY must be fully qualified so that the database system knows whether to take it from PS_COUNTRY_TBL (A.COUNTRY) or PS_STATE_TBL (B.COUNTRY).

The WHERE clause contains the criteria used to limit results in the query. In this example there are two criteria. The first, A.COUNTRY = B.COUNTRY, is the join criterion between PS_COUNTRY_TBL and PS_STATE_TBL on the COUNTRY field. The second, A.EU_MEMBER_STATE = 'Y,' is what restricts the results to those countries that are members of the European Union.

The ORDER BY clause designates how the results are sorted. The first field given is the primary sort field; the second field given is the secondary sort field, and so on. To have the rows sorted by a field in descending order, follow the field name with the DESC keyword. In the example, an ORDER BY clause of A.DESCR, B.DESCR directs the database server to return the results in order by first by country name, then by state name within each country.

You can use numbers instead of fields and expressions in the ORDER BY clause. Each number corresponds to the position of a field or expression in the SELECT clause, with 1 representing the first field or expression. This is not recommended for manually-written SQL since if you change the order of items in the SELECT clause; you will have to change the position numbers as well. In the given example, the ORDER BY clause could be rewritten as ORDER BY 2, 4. Note that the queries generated by Query Manager use position numbers.

Whitespace is ignored; tabs, spaces, and carriage returns are legal except in the middle of field names and keywords (except for multi-word keywords such as ORDER BY). When using a fully-qualified name, there cannot be a space between the alias and the period and the period and the field name. “A.DESC” is allowed but “A . DESCR” is not. However, “A.COUNTRY=B.COUNTRY” and “A.COUNTRY = B.COUNTRY” are the same.
Outer Joins

The previous example involves two tables connected using an inner join. The syntax for an outer join is a little more complex, involving giving the join conditions within the FROM clause. The following example obtains a list of Theatre and Film classes held in Spring 2013 and their exam times; when an exam time is not known, the class is still listed. (This is the SQL equivalent of exercise TRNG_QM##_E14 as generated by Query Manager.)

SELECT A.CRSE_ID, A.CRSE_OFFER_NBR, A.STRM, A.SESSION_CODE, A.SUBJECT,
       A.CATALOG_NBR, A.CLASS_SECTION, A.DESCR, A.SSR_COMPONENT, B.CLASS_EXAM_SEQ,
       TO_CHAR(B.EXAM_DT,'YYYY-MM-DD'),
       TO_CHAR(CAST((B.EXAM_START_TIME) AS TIMESTAMP),'HH24.MI.SS.FF'),
       TO_CHAR(CAST((B.EXAM_END_TIME) AS TIMESTAMP),'HH24.MI.SS.FF')
FROM (PS_CLASS_TBL A LEFT OUTER JOIN PS_CLASS_EXAM B ON A.CRSE_ID = B.CRSE_ID AND
       A.CRSE_OFFER_NBR = B.CRSE_OFFER_NBR AND A.STRM = B.STRM AND
       A.SESSION_CODE = B.SESSION_CODE AND A.CLASS_SECTION = B.CLASS_SECTION)
WHERE A.STRM = '2132'
   AND A.SUBJECT = 'THFM'
ORDER BY 5, 6, 7

In the FROM clause, the description of the outer join is enclosed in parentheses. The LEFT OUTER JOIN keyword indicates that an outer join will be used to combine data from the two tables. The ON clause designates the criteria that are used to perform the outer join; this is built by Query Manager when “this criteria belongs to” is set to “ON clause of outer join B.”

There is alternate outer join syntax for various database systems. For instance, with Oracle 11g, the table names can be a regular list in the FROM clause, and the criteria that would normally be in the ON clause would have the symbol “(+)” following them. The following Oracle-specific query is equivalent to the above example:

SELECT A.CRSE_ID, A.CRSE_OFFER_NBR, A.STRM, A.SESSION_CODE, A.SUBJECT,
       A.CATALOG_NBR, A.CLASS_SECTION, A.DESCR, A.SSR_COMPONENT, B.CLASS_EXAM_SEQ,
       TO_CHAR(B.EXAM_DT,'YYYY-MM-DD'),
       TO_CHAR(CAST((B.EXAM_START_TIME) AS TIMESTAMP),'HH24.MI.SS.FF'),
       TO_CHAR(CAST((B.EXAM_END_TIME) AS TIMESTAMP),'HH24.MI.SS.FF')
FROM PS_CLASS_TBL A, PS_CLASS_EXAM B
WHERE A.CRSE_ID = B.CRSE_ID(+) AND
       A.CRSE_OFFER_NBR = B.CRSE_OFFER_NBR(+) AND
       A.STRM = B.STRM(+) AND
       A.SESSION_CODE = B.SESSION_CODE(+) AND
       A.CLASS_SECTION = B.CLASS_SECTION(+)
       AND A.STRM = '2132'
       AND A.SUBJECT = 'THFM'
ORDER BY 5, 6, 7
Aggregate Query

A query that involves aggregate functions has a couple of additional clauses:

```
SELECT <display field/expression list>
FROM <table/record list>
WHERE <criteria>
GROUP BY <grouping field/expression list>
HAVING <having criteria>
ORDER BY <sort field/expression list>
```

Consider the following example in which the query gets a count of active plans per program, ordered by program code, showing only those programs for which there are more than 50 plans.

```
SELECT A.ACAD_PROG, COUNT(*)
FROM PS_ACAD_PLAN_TBL A
WHERE A.EFFDT = (SELECT MAX(A_ED.EFFDT)
                   FROM PS_ACAD_PLAN_TBL A_ED
                   WHERE A.INSTITUTION = A_ED.INSTITUTION
                   AND A.ACAD_PLAN = A_ED.ACAD_PLAN
                   AND A_ED.EFFDT <= SYSDATE)
       AND A.EFF_STATUS = 'A'
GROUP BY A.ACAD_PROG
HAVING COUNT(*) > 50
ORDER BY A.ACAD_PROG
```

The GROUP BY clause lists the fields that are used to group rows together for the aggregate function to work upon. In the above example, the GROUP BY clause has the field A.ACAD_PROG, so the counts are of rows having the same academic program code.

If the results will be limited based on the result of the aggregate function, there must be a HAVING clause. In this clause, there are criteria that use the aggregate function to restrict the groups that are shown. The above example has one criterion in its HAVING clause – COUNT(*) > 50 – which instructs the database server to only return groups having a count of rows greater than 50. The results will therefore be limited to programs that have more than 50 plans.

The asterisk * is a shortcut representing all fields. When used in the SELECT clause, it represents all fields from all tables in the query if it is not preceded by an alias (SELECT *) or all fields from one table if it is preceded by an alias (A.*). When used in the COUNT(*) function, in represents all fields in the GROUP BY clause.
Query Manager and SQL

The tabs in Query Manager correspond roughly to the clauses of a query in SQL as follows:

- **Records** – no equivalent, since this is where tables are searched for and added or joined
- **Query** – the FROM clause
- **Expressions** – no equivalent; can appear where fields and values are allowed
- **Prompts** – no equivalent; can appear where fields and values are allowed
- **Fields**
  - The SELECT clause since each field listed in this tab is displayed in the results
  - The ORDER BY clause if the sort order is changed using the Edit Field Ordering page (Reorder/Sort button)
  - The GROUP BY clause since if there is a field with an aggregate function, those fields *not* being aggregated comprise the groupings
- **Criteria**
  - The WHERE clause of the main query
  - The FROM clause when “this criteria belongs to” is set to “ON clause of outer join X,” where X is the alias of the record on the right side of the join
- **Having** – the HAVING clause
- **View SQL** – no equivalent, but shows the SQL generated to execute the query
- **Run** – no equivalent, since it shows the results of the query

Note that the SQL that Query Manager generates may not exactly match the SQL that you or someone else writes. There are multiple ways to express the same query in SQL, just as there are multiple ways to express the same concept or instructions in human languages. This does not mean that your query is incorrect or that the query from Query Manager is incorrect; they may be equivalent.
Troubleshooting

Suggestions of what to check when a query produces incorrect results and/or errors:

- If the query produces many more rows than are expected
  - Are there any criteria missing?
    - Were you asked to limit the results to a particular term but there is no criteria on STRM?
    - Should you be showing only "active" rows, such as those with EFF_STATUS = 'A', PROG_STATUS = 'AC', etc.?
    - Are there criteria on the appropriate key fields, such as INSTITUTION = BGSUN', SETID = 'BGSUN' or 'BGHCM', ACAD_CAREER = 'UGRD' or 'GRAD', etc.?
  - Are the joins between records correct?
    - If a join is missing, each row from the first table is combined with each row from the second table.
    - If you added the join yourself by creating criteria, did you join on all appropriate key fields? For example, if you are querying on SRVC_IND_DATA and join to SRVC_IND_RSN_TBL to get the description of the reason code, did you join on INSTITUTION and SRVC_IND_CD as well as SRVC_IND_REASON?
    - Did you use a left outer join instead of a standard join? Left outer joins are used only when you need rows from the first table to be shown regardless of whether there is a match in the second table. If a match is required, use a standard join.

- If the query produces many fewer rows than are expected
  - Did you create a criterion on the wrong field? For example, to find a class by course ID, did you put a criterion on CATALOG_NBR instead of CRSE_ID?
  - Did you compare against the wrong value? For instance, if you want active programs, did you check for PROG_STATUS = 'AC' or PROG_STATUS = 'ACTV'? (In this case, the former is correct; 'ACTV' is a valid value for PROG_ACTION.)
  - Did you manually join two records on the wrong fields? The values in each field are not like each other, so there is unlikely to be a match, and if there is one, it will be by coincidence. For instance, the field ACAD_PLAN should not be joined to the field ACAD_PROG since they represent different things. (Hint: fields that can be joined will likely have identical or similar names.)
  - Is it impossible to meet the conditions? If you wrote a query to find cases where PROG_STATUS is either 'AC' or 'LA', did you use PROG_STATUS = 'AC' AND PROG_STATUS = 'LA' or PROG_STATUS = 'AC' OR PROG_STATUS = 'LA'? Since it is not possible for a field to have two values that are the same time, a query with X = <value1> AND X = <value2> will always return no rows.
• Problems with expressions
  o “Invalid identifier” error
    ▪ Field names referenced in an expression must be spelled correctly
      (SRVC_IND_RSN is incorrect; SRVC_IND_REASON is correct)
    ▪ Function names must also be spelled correctly (TRUNCATE is incorrect; TRUNC is correct)
    ▪ Strings (literal text included in the expression) must be enclosed in single quotes
  o Value displayed has too few characters or digits
    ▪ Check the Length property of the expression.
      • Character: If Length is 10 but the result of the expression is 20 characters long, only the first 10 characters are shown.
      • Number: Values are shortened by trimming digits off the end of the integer portion of the number (1234567.890 when displayed with a Length property of 5 is shown as 12345).
  o Missing parentheses
    ▪ Each opening parenthesis ( must have a corresponding closing parenthesis )
    ▪ Errors can vary depending on how the query engine interprets the generated SQL
  o Functions
    ▪ Did you include all required arguments? For example, SUBSTR takes two or three arguments, and the first two are required. Using SUBSTR('ABC') results in a “not enough arguments for function” error.
    ▪ Did you include too many arguments? For example, LOWER takes only one argument. Using LOWER('ABC', 5) results in a “too many arguments for function” error.
    ▪ Did you use the right types of arguments? MONTHS_BETWEEN takes two date arguments results in an “inconsistent datatypes” error. If a date is expected, a date must be given, if a number is expected, a number (or string that can be converted to a number) must be given, etc.
    ▪ Did you use the correct function? CEIL, FLOOR, ROUND, and TRUNC all round numbers, but only ROUND and TRUNC can round to multiples other than 1, and all four round slightly differently (CEIL rounds up, FLOOR rounds down, ROUND follows regular rounding rules, and TRUNC drops the digits after the decimal point).
• Problems with unions
  o Each query that is part of a union must have the same number of fields. If there is a difference, this error is displayed: “A UNION requires the same number of fields for each SELECT. Correct and retry.”
• Problems with prompts
  o A prompt that uses a prompt table may show no values when using the lookup feature. If the prompt table has a key with more than one field, there must be prompts for all fields before that field in the field order. For instance, if there is a prompt for Program (ACAD_PROG) using ACAD_PROG_TBL as a
prompt table, since the key of ACAD_PROG_TBL is INSTITUTION + ACAD_PROG + EFFDT, there must also be a prompt for INSTITUTION. In addition, the prompts must be specified in the order of the fields of the key; in the example, the prompt on INSTITUTION must come before the prompt on ACAD_PROG.

- Problems with aggregates
  - You cannot use fields that are being aggregated as regular criteria. If you apply the Count aggregate function to STRM, you cannot also add the criterion STRM = '2095'. When you aggregate, you are no longer dealing with individual rows, but groups. (You can still add criteria on the fields that are not being aggregated. In the example on STRM, perhaps you could aggregate on ACAD_CAREER or ACAD_PROG.)
  - Remember that once you add an aggregate to a displayed field, the aggregation is done on groups defined by the other fields. For example, if you want to count how many students there are per program, don’t include EMPLID in the fields being displayed, or else the count will be per combination of program and EMPLID.
  - If you put an aggregate function into an expression, you must check the Aggregate Function checkbox, or you will receive the error “not a single-group group function.” This is indicating that you are trying to use an aggregate function on results that are not being aggregated.

- Unable to find a query
  - Make sure you are on the Query Manager search page (titled “Query Manager”) rather than the Records tab (titled “Find an Existing Record”). Since both have search capabilities, one can mistake one of these search pages for the other page.
  - If you tried searching for the full name of the query, you may have misspelled the name either when searching for it or saving it. Try entering less of the name. For instance, search for “TRNG_QM##” or even “TRNG” instead of “TRNG_QM##_Q15.” You may get several extraneous results but have a better chance of getting the one that you need.

- Learn SQL!
  - The query is not checked for errors before it reaches the database. The error messages reported by Query Manager are native to the database and refer to the generated SQL. To debug the query, it is sometimes easiest to look at the generated SQL. This will show which records are included, how they are joined (even if you didn’t create the join conditions yourself); subqueries, effective date logic, and so on, as a legal SQL query. It can also show a little more clearly which criteria are for which part of the query in the case of large queries.