The supplemental material is a continuation of the topics covered in the Query Manager workbook. The supplemental material was created to be used as a reference when creating queries.

Queries can be difficult to create and manage. This material provides instructions on how to manage and organize queries to the importance of effective date usage and logical operators when adding criteria. The expressions section of this material provides step-by-step instructions on how to add multiplication and date format strings when adding expressions to specific fields. The section on Outer joins is worth looking at to better understand the "where" clause.

Topics that are expanded on include (but not limited to):

- Effective Data and most common use for effective date logic in queries.
- Query organization importance including:
  - Copy a query to a User
  - Delete queries
  - Move queries to a folder
  - Rename queries
- Grouping criteria and the use of logical operators (and & or)
- Expressions and instructions on adding:
  - Mathematical
  - String Functions
  - Numeric Functions
  - Date Functions
  - Conversion Functions
  - Conditional Functions
  - Case Functions
- Outer Joins and the "where" clause.
- Aggregate queries
- Troubleshooting

We hope that you find this documentation a helpful tool. However, there is nothing more helpful than finding a group of people that you can network with to discuss queries and the problems that you incur. Help is always available by emailing bgat100@bgsu.edu.

We always look forward to suggestion on how to improve this training material as well as how you use this documentation to assist in improving you process of creating queries.

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 have created. Lori Beeman created several lists of records and their descriptions as hierarchies, so you not only get the internal names of the records, but see which records are related to each other as parents and children.
- 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.

INSTITUTION	ACAD_PROG	EFFDT	EFF_STATUS	DESCR	ACAD_CAREER	ACAD_CALENDAR_ID
BGSUN	ARTSC	8/29/1982	A	College of Arts and Sciences	UGRD	USEM
BGSUN	ARTSC	9/26/1971	A	College of Arts and Sciences	UGRD	UQTR
BGSUN	ARTSC	9/24/1968	A	College of Liberal Arts	UGRD	UQTR
BGSUN	ARTSC	1/1/1910	A	College of Liberal Arts	UGRD	USEM

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 arts and sciences program has been "College of Arts and Sciences" and the academic calendar was based on semesters (USEM). From September 26, 1971 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 ARTSC does not indicate when data for effective for program BUSN, program MUSIC, 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.

EMPLID	EMPL_RCD	EFFDT	EFFSEQ	DEPTID	JOBCODE	SUPERVISOR_ID	HR_STATUS	ACTION
9106	0	4/20/2008	0	310200	600001	2164	I	TER
9106	0	7/29/2007	1	310200	600001	2164	A	DTA
9106	0	7/29/2007	0	071100	600001	1316	A	HIR

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).

INSTITUTION	ACAD_PLAN	EFFDT	EFF_STATUS	DESCR	ACAD_PLAN_TYPE	ACAD_PROG	DEGREE
BGSUN	AERO-BSTC	8/25/2015	I	Aerotechnology	MAJ	TECH	BSTC
BGSUN	AERO-BSTC	1/1/1910	A	Aerotechnology	MAJ	TECH	BSTC

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 8/24/2015. On 8/25/2015, it will become inactive (EFF_STATUS = 'I'). This means that AERO-BSTC will be inactive *in the future* but is active now.

INSTITUTION	ACAD_PLAN	EFFDT	EFF_STATUS	DESCR	ACAD_PLAN_TYPE	ACAD_PROG	DEGREE
BGSUN	THEA-BSJ	5/18/2008	I	Theatre	MAJ	ARTSC	BSJ
BGSUN	THEA-BSJ	1/1/1910	A	Theatre	MAJ	ARTSC	BSJ

The second example shows that plan THEA-BSJ – a Bachelor of Science in Journalism (BSJ) for Theatre – was active until 5/17/2008 and was discontinued on 5/18/2008, placing it in an inactive status. This means that THEA-BSJ is *currently* inactive.

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 Explorer or My Computer in Windows. 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

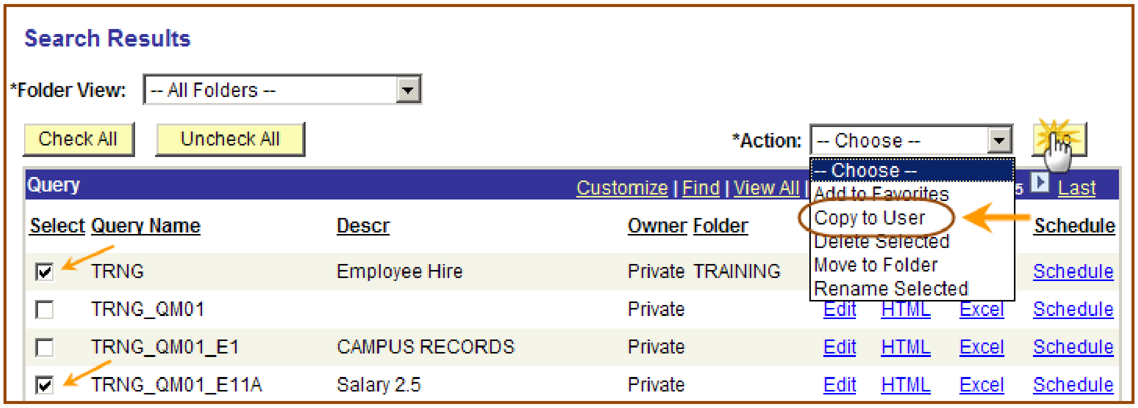
It may be useful to send a query to another user if that user needs to make minor modifications before running it, to see how it was built, or to run the query with different row-level security than that of the first user. Query Manager enables you to copy a query to another user's storage space.

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.



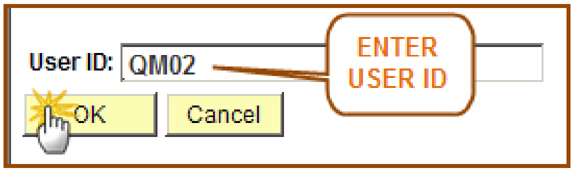
Search Results

*Folder View: -- All Folders --

*Action: -- Choose --

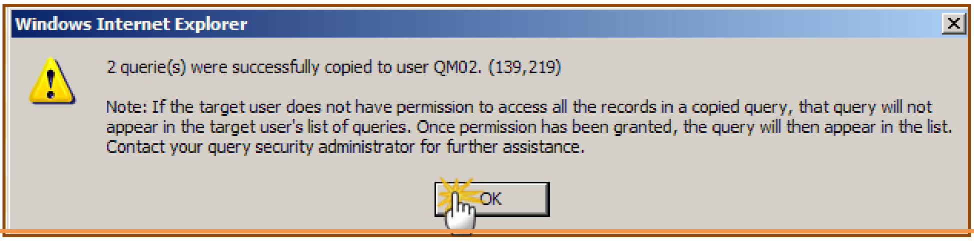
Select	Query Name	Descr	Owner	Folder				Schedule
☑	TRNG	Employee Hire	Private	TRAINING				Schedule
☐	TRNG_QM01		Private		Edit	HTML	Excel	Schedule
☐	TRNG_QM01_E1	CAMPUS RECORDS	Private		Edit	HTML	Excel	Schedule
☑	TRNG_QM01_E11A	Salary 2.5	Private		Edit	HTML	Excel	Schedule

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



User ID: QM02

If the copy is successful, you will receive a message confirming this.



Windows Internet Explorer

2 querie(s) were successfully copied to user QM02. (139,219)

Note: If the target user does not have permission to access all the records in a copied query, that query will not appear in the target user's list of queries. Once permission has been granted, the query will then appear in the list. Contact your query security administrator for further assistance.

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:

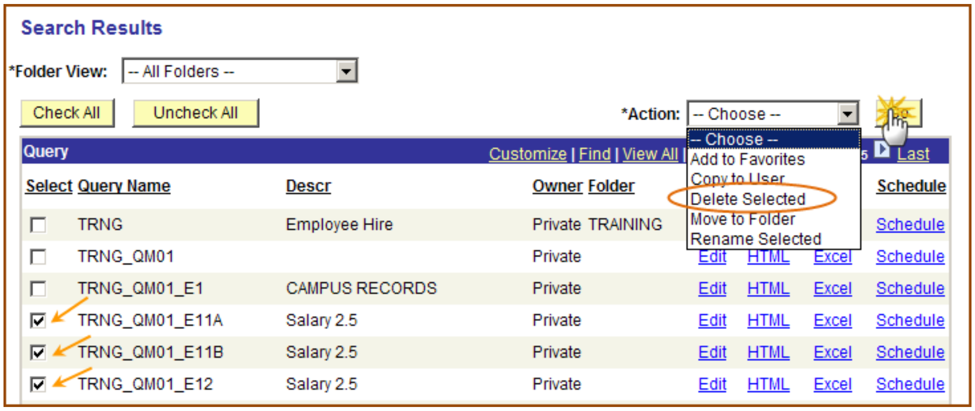
- From the Query Manager search page, search for the query to be deleted.

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.



Select	Query Name	Descr	Owner	Folder				Schedule
☐	TRNG	Employee Hire	Private	TRAINING				Schedule
☐	TRNG_QM01		Private		Edit	HTML	Excel	Schedule
☐	TRNG_QM01_E1	CAMPUS RECORDS	Private		Edit	HTML	Excel	Schedule
☑	TRNG_QM01_E11A	Salary 2.5	Private		Edit	HTML	Excel	Schedule
☑	TRNG_QM01_E11B	Salary 2.5	Private		Edit	HTML	Excel	Schedule
☑	TRNG_QM01_E12	Salary 2.5	Private		Edit	HTML	Excel	Schedule

delete

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!

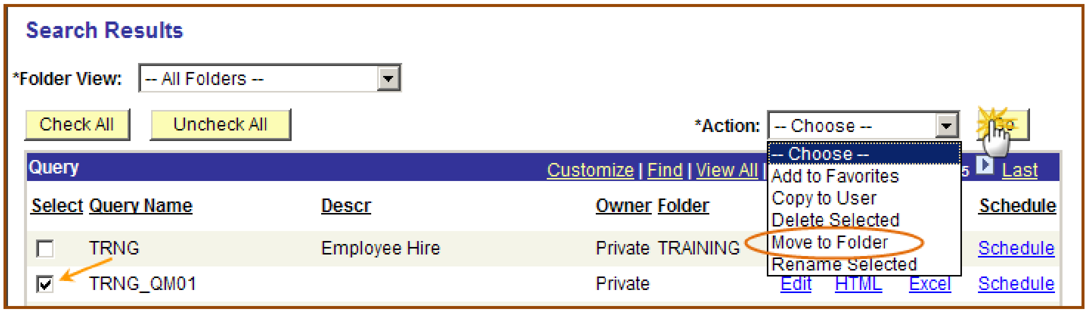
Move a Query to a Folder

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:

- o 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.)

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



Search Results

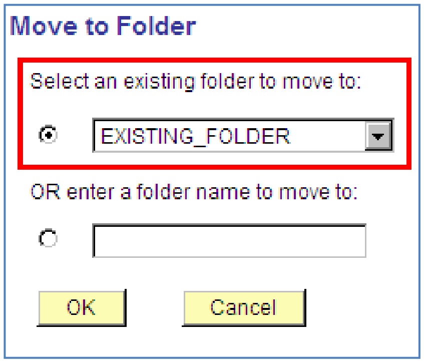
*Folder View: -- All Folders --

*Action: -- Choose --

Select	Query Name	Descr	Owner	Folder		Schedule
□	TRNG	Employee Hire	Private	TRAINING		Schedule
☑	TRNG_QM01		Private		Edit HTML Excel	Schedule

A Move to Folder page appears.

- o 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.



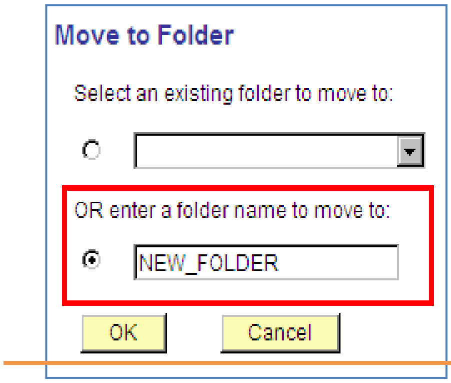
Move to Folder

Select an existing folder to move to:

● EXISTING_FOLDER

OR enter a folder name to move to:

○



Move to Folder

Select an existing folder to move to:

○

OR enter a folder name to move to:

● NEW_FOLDER

- o 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

- o **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!

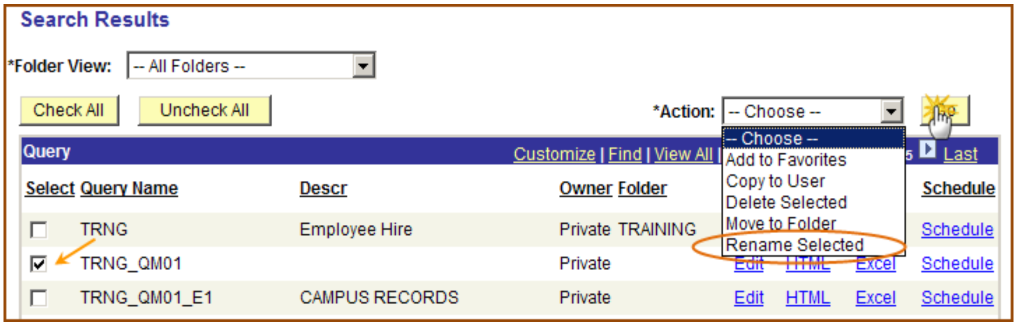
Rename a Query

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:

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

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.)

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



Search Results

*Folder View: -- All Folders --

Check All | Uncheck All

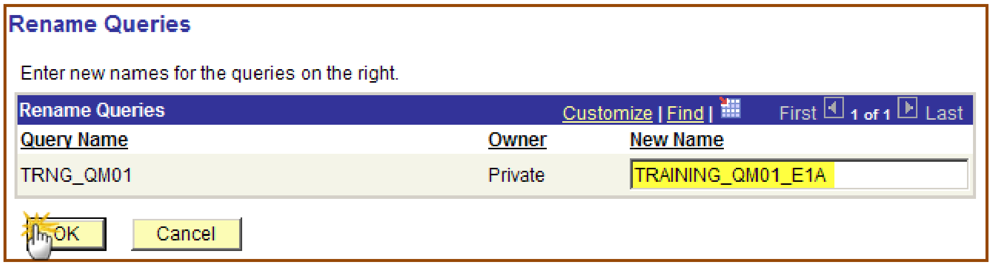
*Action: -- Choose -- | Go

-- Choose --
Add to Favorites
Copy to User
Delete Selected
Move to Folder
Rename Selected

Query — Customize | Find | View All | ... 5 Last

Select	Query Name	Descr	Owner	Folder				Schedule
□	TRNG	Employee Hire	Private	TRAINING				Schedule
☑	TRNG_QM01		Private		Edit	HTML	Excel	Schedule
□	TRNG_QM01_E1	CAMPUS RECORDS	Private		Edit	HTML	Excel	Schedule

- **Ente**r a **new name** for each selected query next to the old name of each query.
- **Click** the **OK** button.



Rename Queries

Enter new names for the queries on the right.

Rename Queries — Customize | Find | First 1 of 1 Last

Query Name	Owner	New Name
TRNG_QM01	Private	TRAINING_QM01_E1A

OK | Cancel

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.

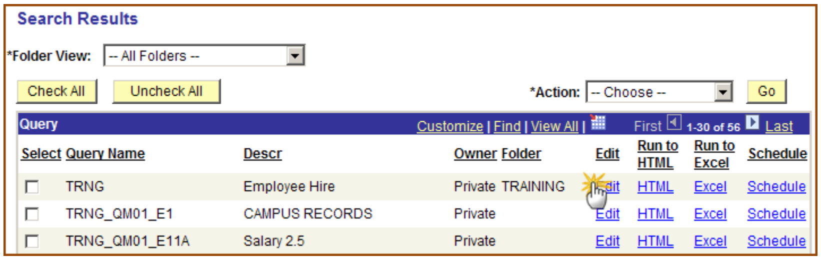
Save a Query with a New Name

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:

- o 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.



Search Results

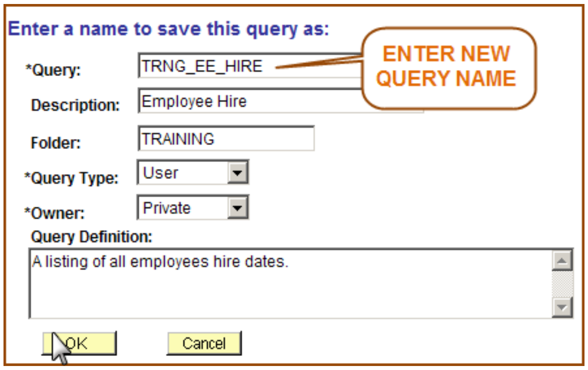
*Folder View: -- All Folders --

Check All | Uncheck All | *Action: -- Choose -- | Go

Query — Customize | Find | View All | First 1-30 of 56 Last

Select	Query Name	Descr	Owner	Folder	Edit	Run to HTML	Run to Excel	Schedule
☐	TRNG	Employee Hire	Private	TRAINING	Edit	HTML	Excel	Schedule
☐	TRNG_QM01_E1	CAMPUS RECORDS	Private		Edit	HTML	Excel	Schedule
☐	TRNG_QM01_E11A	Salary 2.5	Private		Edit	HTML	Excel	Schedule

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



Enter a name to save this query as:

*Query: TRNG_EE_HIRE

Description: Employee Hire

Folder: TRAINING

*Query Type: User

*Owner: Private

Query Definition: A listing of all employees hire dates.

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

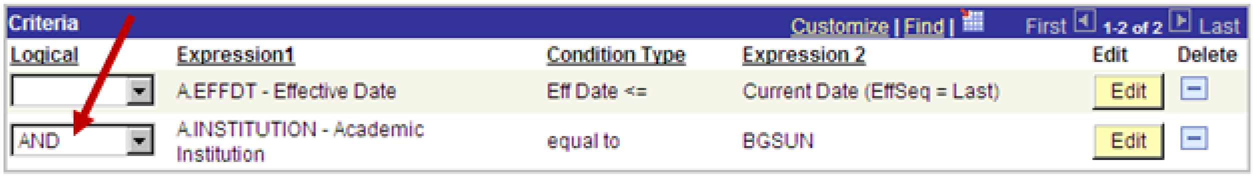
- On some occasions, you will need to write a query that returns rows that meet some criteria, but not all of them. For instance, you may be asked to limit the results to those for which the program status is either ‘AC’ or ‘LA’. This could be implemented using the criterion “PROG_STATUS in list ‘AC’, ‘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).



Criteria — Customize | Find | — First 1-2 of 2 Last

Logical	Expression1	Condition Type	Expression 2	Edit	Delete
	A.EFFDT - Effective Date	Eff Date <=	Current Date (EffSeq = Last)	Edit	−
AND	A.INSTITUTION - Academic Institution	equal to	BGSUN	Edit	−

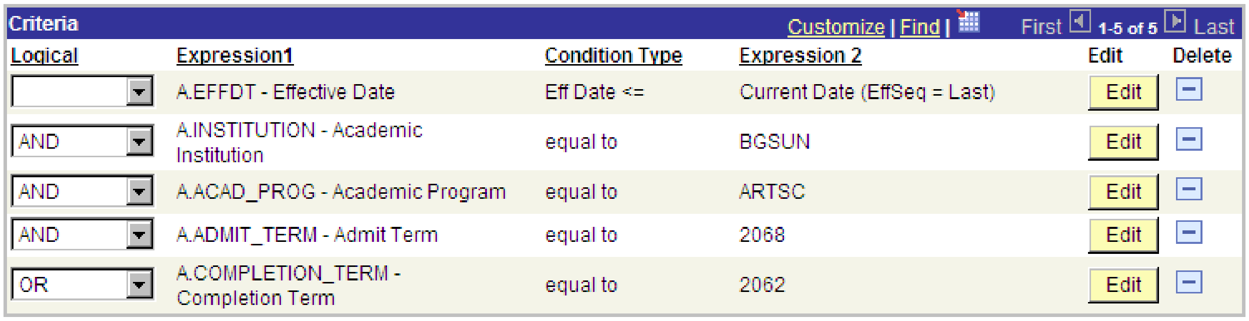
- The operators are called “Logical” because they operate on values that are either *true* or *false*. Each criterion has a comparison that produces a true result or a false result. In a row in which INSTITUTION is “BGSUN,” the criterion A.INSTITUTION equal to BGSUN evaluates to *true*.

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 (INSTITUTION = 'BGSUN') in the Arts and Sciences program (ACAD_PROG = 'ARTSC') and were either admitted in Fall 2006 (ADMIT_TERM = '2068') *or* completed the program in Spring 2006 (COMPLETION_TERM = '2062'). Note that academic program data is effective dated.

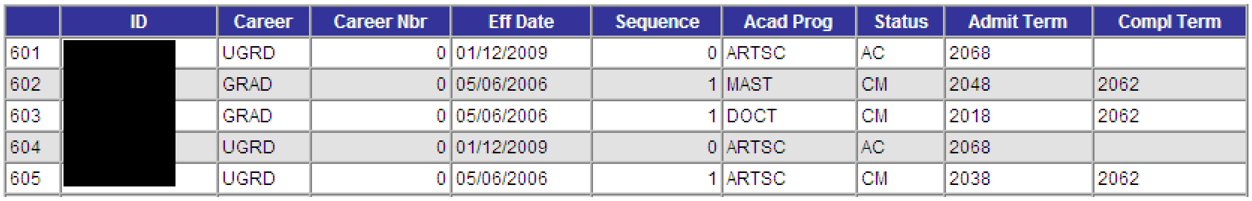
- Here is a first attempt at building these criteria in our query:



Criteria — Customize | Find | — First 1-5 of 5 Last

Logical	Expression1	Condition Type	Expression 2	Edit	Delete
	A.EFFDT - Effective Date	Eff Date <=	Current Date (EffSeq = Last)	Edit	−
AND	A.INSTITUTION - Academic Institution	equal to	BGSUN	Edit	−
AND	A.ACAD_PROG - Academic Program	equal to	ARTSC	Edit	−
AND	A.ADMIT_TERM - Admit Term	equal to	2068	Edit	−
OR	A.COMPLETION_TERM - Completion Term	equal to	2062	Edit	−

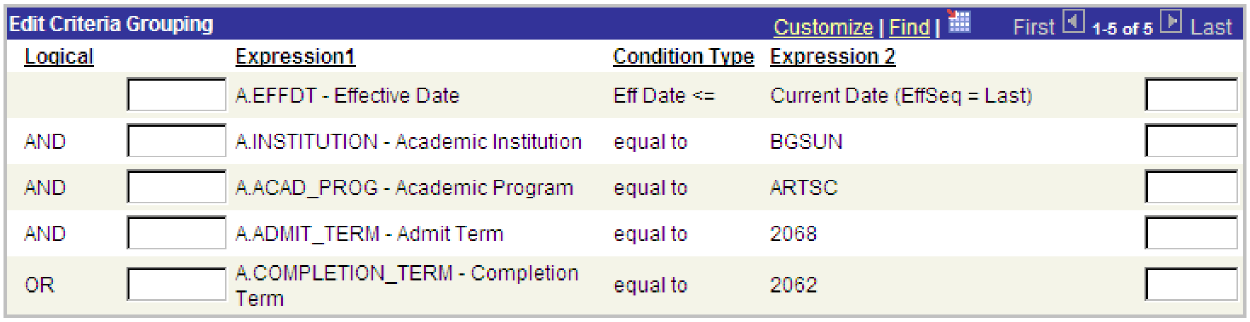
Below is an excerpt from the results returned by the query:



	ID	Career	Career Nbr	Eff Date	Sequence	Acad Prog	Status	Admit Term	Compl Term
601	[redacted]	UGRD	0	01/12/2009	0	ARTSC	AC	2068	
602	[redacted]	GRAD	0	05/06/2006	1	MAST	CM	2048	2062
603	[redacted]	GRAD	0	05/06/2006	1	DOCT	CM	2018	2062
604	[redacted]	UGRD	0	01/12/2009	0	ARTSC	AC	2068	
605	[redacted]	UGRD	0	05/06/2006	1	ARTSC	CM	2038	2062

- Row 601 matches what was expected since the program is ARTSC and the admit term is Fall 2006. However, row 602 doesn't match what was intended; while the completion term is Spring 2006, the program is MAST. This row was included because it meets the COMPLETION_TERM equal to 2062 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 2006 or the completion term is Spring 2006 – 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.

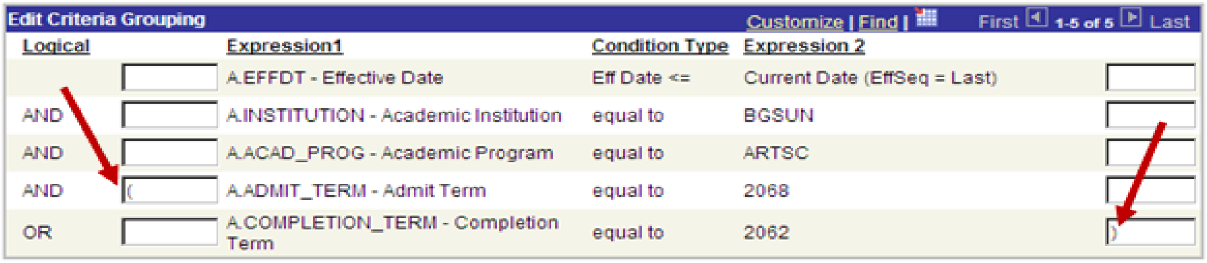


Edit Criteria Grouping

Logical		Expression1	Condition Type	Expression 2	
		A.EFFDT - Effective Date	Eff Date <=	Current Date (EffSeq = Last)	
AND		A.INSTITUTION - Academic Institution	equal to	BGSUN	
AND		A.ACAD_PROG - Academic Program	equal to	ARTSC	
AND		A.ADMIT_TERM - Admit Term	equal to	2068	
OR		A.COMPLETION_TERM - Completion Term	equal to	2062	

- 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.

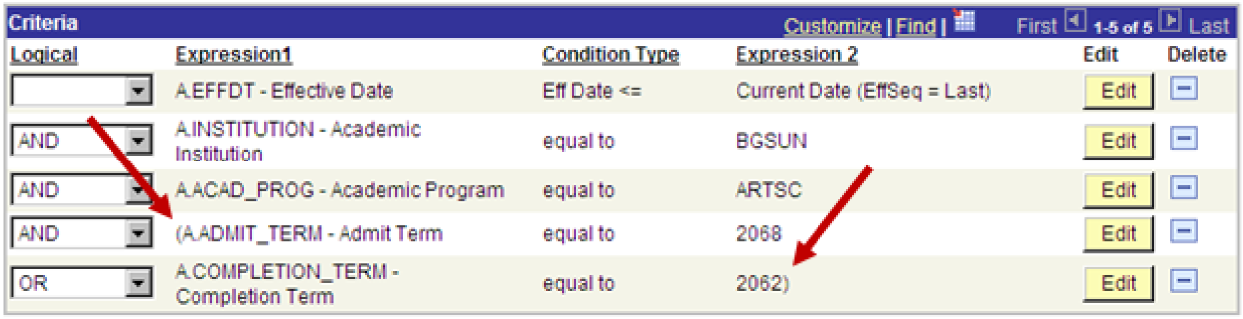
- In this case, we are creating a group containing the criterion on ADMIT_TERM and the criterion on COMPLETION_TERM. Parentheses are added are indicated in the screen shot below:



Edit Criteria Grouping

Logical		Expression1	Condition Type	Expression 2	
		A.EFFDT - Effective Date	Eff Date <=	Current Date (EffSeq = Last)	
AND		A.INSTITUTION - Academic Institution	equal to	BGSUN	
AND		A.ACAD_PROG - Academic Program	equal to	ARTSC	
AND	(	A.ADMIT_TERM - Admit Term	equal to	2068	
OR		A.COMPLETION_TERM - Completion Term	equal to	2062	)

- Click OK to confirm the changes to the grouping. The parentheses are displayed on the Criteria page.

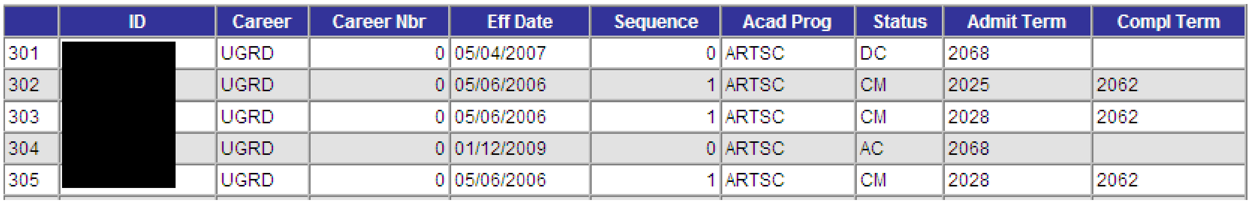


Criteria

Logical	Expression1	Condition Type	Expression 2	Edit	Delete
	A.EFFDT - Effective Date	Eff Date <=	Current Date (EffSeq = Last)	Edit	⊟
AND	A.INSTITUTION - Academic Institution	equal to	BGSUN	Edit	⊟
AND	A.ACAD_PROG - Academic Program	equal to	ARTSC	Edit	⊟
AND	(A.ADMIT_TERM - Admit Term	equal to	2068	Edit	⊟
OR	A.COMPLETION_TERM - Completion Term	equal to	2062)	Edit	⊟

- 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 2068 or COMPLETION_TERM is equal to 2062, 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:



	ID	Career	Career Nbr	Eff Date	Sequence	Acad Prog	Status	Admit Term	Compl Term
301	[redacted]	UGRD	0	05/04/2007	0	ARTSC	DC	2068	
302	[redacted]	UGRD	0	05/06/2006	1	ARTSC	CM	2025	2062
303	[redacted]	UGRD	0	05/06/2006	1	ARTSC	CM	2028	2062
304	[redacted]	UGRD	0	01/12/2009	0	ARTSC	AC	2068	
305	[redacted]	UGRD	0	05/06/2006	1	ARTSC	CM	2028	2062

- Observe that all of the rows in the result set have either the admit term of Fall 2006 or the completion term of Spring 2006, 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.

Search String	Value in Field	Match?	Reason
abc%	abc	Yes	Starts with “abc” and is followed by zero characters
	abcd	Yes	Starts with “abc” and is followed by one character
	abcdefgh	Yes	Starts with “abc” and is followed by many characters
	ab	No	Does not start with “abc”
	abz	No	Does not start with “abc”
	aabc	No	Does not start with “abc” (even though “abc” is in the value)
%def	def	Yes	Begins with no characters and ends with “def”
	cdef	Yes	Begins with one character and ends with “def”
	abcdef	Yes	Begins with many characters and ends with “def”
	ef	No	Does not end with “def”
	ref	No	Does not end with “ref”
	deff	No	Does not end with “def” (even though “def” is in the value)
%ghi%	ghi	Yes	Begins with no characters, followed by “ghi,” followed by no characters
	efghi	Yes	Begins with several characters, followed by “ghi,” followed by no characters
	ghijk	Yes	Begins with no characters, followed by “ghi,” followed by no characters
	efghijk	Yes	Begins with several characters, followed by “ghi,” followed by no characters
	efgghiijk	Yes	Begins with “efg,” followed by “ghi,” ending with “ijk”
	efgjk	No	Does not contain “ghi”
	efgpijk	No	Does not contain “ghi”
	gh	No	Does not contain “ghi”
a_c	abc	Yes	“a” followed by one character followed by “c”
	a7c	Yes	“a” followed by one character followed by “c”
	ac	No	No characters between “a” and “c”

Search String	Value in Field	Match?	Reason
	abbc	No	Too many characters between “a” and “c”

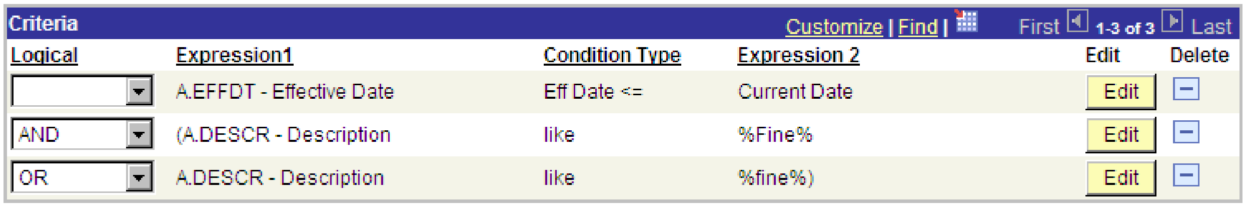
Criteria and Case-Sensitive Data

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.”

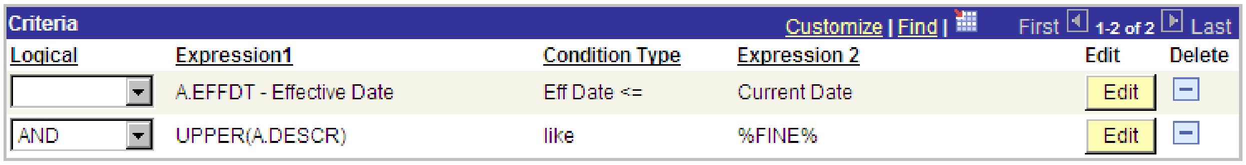


Criteria — Customize | Find | — First 1-3 of 3 Last

Logical	Expression1	Condition Type	Expression 2	Edit	Delete
	A.EFFDT - Effective Date	Eff Date <=	Current Date	Edit	−
AND	(A.DESCR - Description	like	%Fine%	Edit	−
OR	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 — Customize | Find | — First 1-2 of 2 Last

Logical	Expression1	Condition Type	Expression 2	Edit	Delete
	A.EFFDT - Effective Date	Eff Date <=	Current Date	Edit	−
AND	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*.

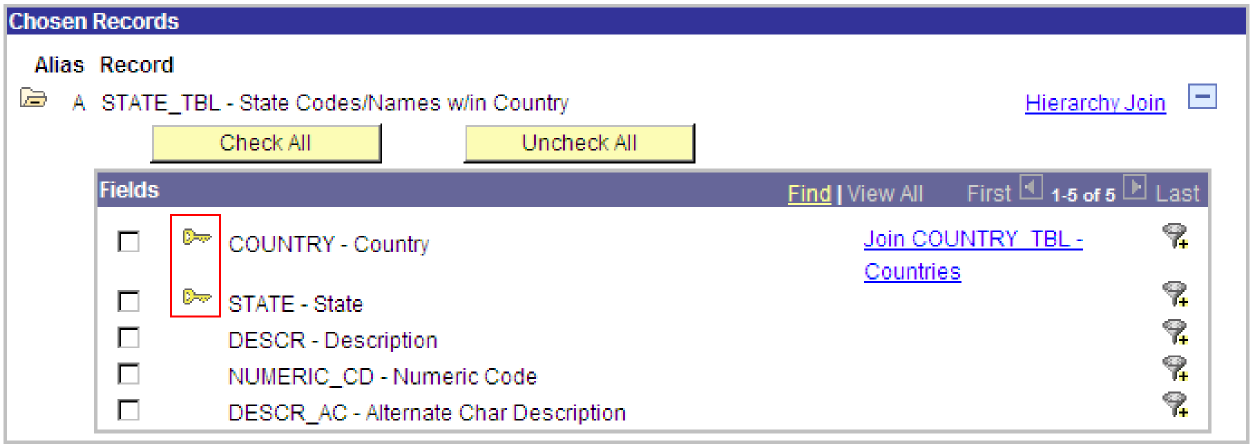
Primary Keys

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.



Alias Record

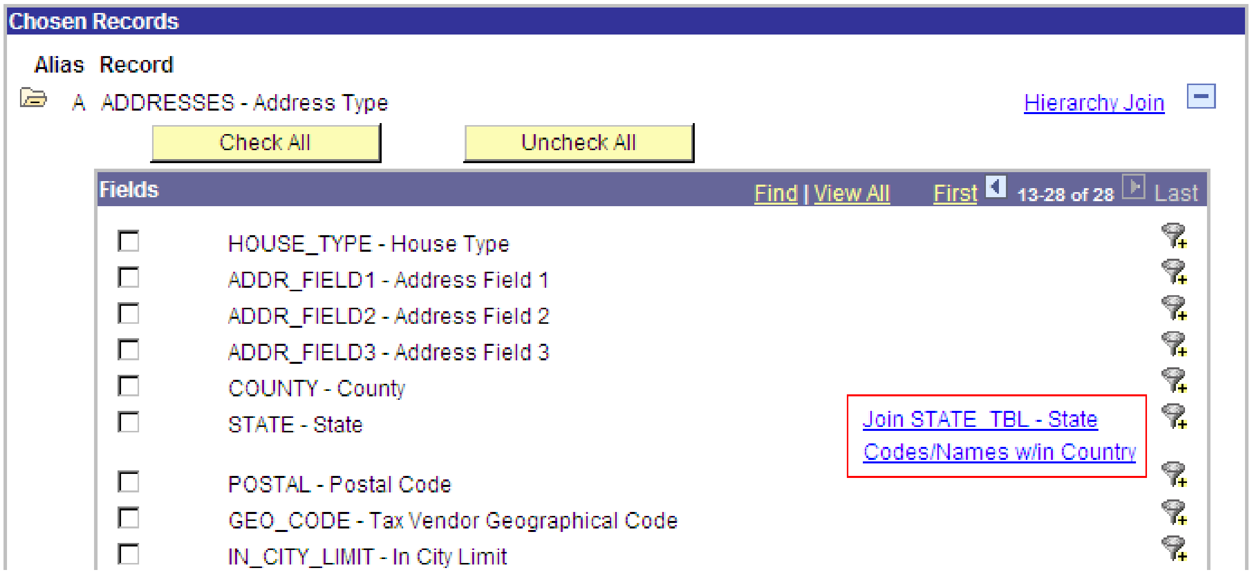
A STATE_TBL - State Codes/Names w/in Country

□ COUNTRY - Country	Join COUNTRY_TBL - Countries
□ STATE - State	
□ DESCR - Description	
□ NUMERIC_CD - Numeric Code	
□ DESCR_AC - Alternate Char Description	

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.



Chosen Records

Alias Record

A ADDRESSES - Address Type

Hierarchy Join

Fields Find | View All First 13-28 of 28 Last

☐ HOUSE_TYPE - House Type
☐ ADDR_FIELD1 - Address Field 1
☐ ADDR_FIELD2 - Address Field 2
☐ ADDR_FIELD3 - Address Field 3
☐ COUNTY - County
☐ STATE - State Join STATE_TBL - State Codes/Names w/in Country
☐ POSTAL - Postal Code
☐ GEO_CODE - Tax Vendor Geographical Code
☐ IN_CITY_LIMIT - In City Limit

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.

EMPLID	ACAD_ CAREER	STDNT_ CAR_NBR	EFFDT	EFFSEQ	INSTITUTION	ACAD_PROG	PROG_STATUS	PROG_ACTION
1379	UGRD	0	1/12/2009	0	BGSUN	EDUC	AC	PLNC

EMPLID	ACAD_ CAREER	STDNT_ CAR_NBR	EFFDT	EFFSEQ	INSTITUTION	ACAD_PROG	PROG_STATUS	PROG_ACTION
1379	UGRD	0	1/7/2008	0	BGSUN	EDUC	AC	PRGC
1379	UGRD	0	8/20/2007	0	BGSUN	ACEN	AC	ACTV

Student 1379 entered the Academic Enhancement program on 8/20/2007, then switched to Education & Human Development on 1/7/2008, then changed a plan within that program on 1/12/2009. 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 2/13/2009 then the expression A.START_DT + 7 results in 2/20/2009. If the numeric value is a decimal amount, then the time will also be affected. If A.START_DT is 2/13/2009 10:00 AM then the expression A.START_DT + 0.5 results in 2/13/2009 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 10 and higher; BGSU is currently using Oracle 10g. 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 5/22/2009, use DECODE(A.DATE_FIELD, NULL, TO_DATE('05/22/2009', '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'
  WHEN A.TEMPERATURE <= 79 THEN 'Warm'
  WHEN A.TEMPERATURE <= 89 THEN 'Hot'
  ELSE 'Very hot'
END
```

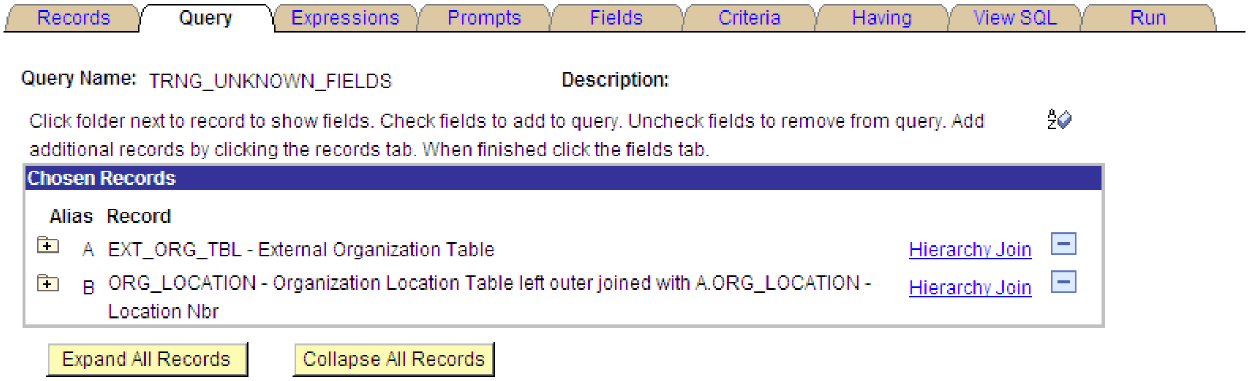
Blank and Unknown Values

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 EXT_ORG_TBL is outer-joined to ORG_LOCATION on the ORG_LOCATION field. The EXT_ORG_ID, EFFDT, DESCR, and ORG_LOCATION fields of EXT_ORG_TBL and the DESCR, ADDRESS1, and ADDRESS2 fields of ORG_LOCATION are displayed. There is a criterion on EFF_STATUS such that it must be equal to 'A'. The results will show all defined external organizations, along with the organization's locations; if there are no locations, the base information about the organization is still shown.

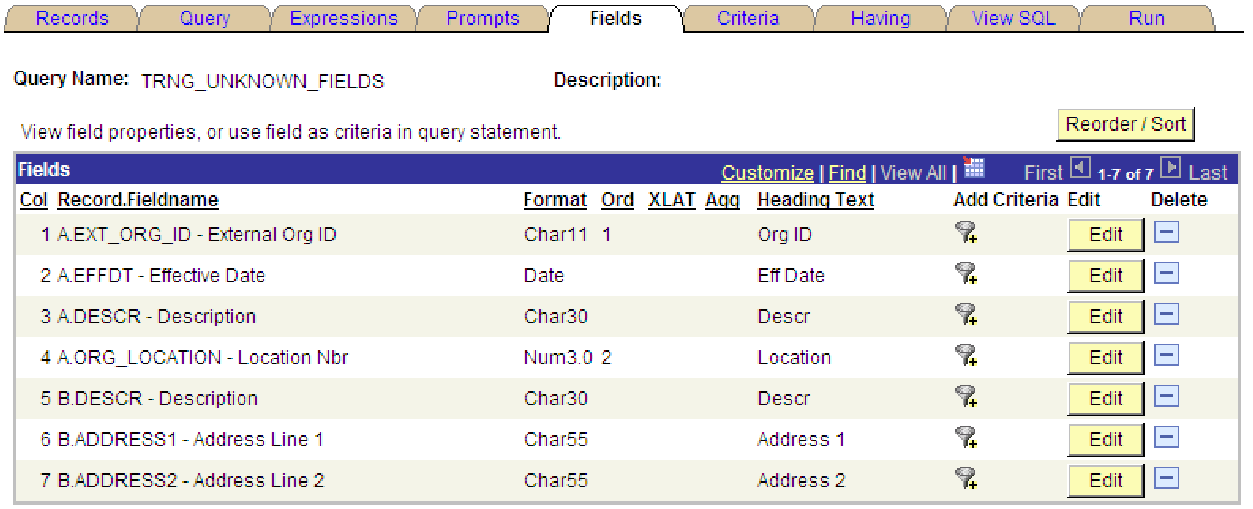


Query Name: TRNG_UNKNOWN_FIELDS   Description:

Click folder next to record to show fields. Check fields to add to query. Uncheck fields to remove from query. Add additional records by clicking the records tab. When finished click the fields tab.

Chosen Records

Alias	Record		
A	EXT_ORG_TBL - External Organization Table	Hierarchy Join	⊟
B	ORG_LOCATION - Organization Location Table left outer joined with A.ORG_LOCATION - Location Nbr	Hierarchy Join	⊟

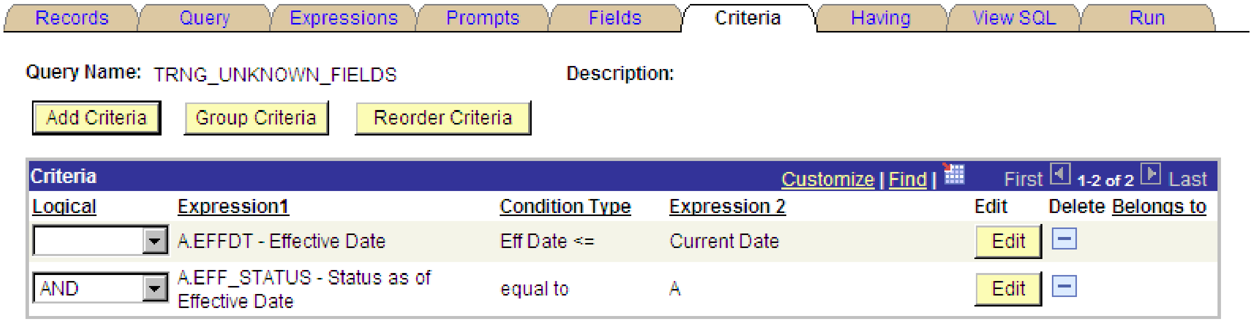


Query Name: TRNG_UNKNOWN_FIELDS Description:

View field properties, or use field as criteria in query statement.

Fields

Col	Record.Fieldname	Format	Ord	XLAT	Agg	Heading Text	Add Criteria	Edit	Delete
1	A.EXT_ORG_ID - External Org ID	Char11	1			Org ID	▽	Edit	⊟
2	A.EFFDT - Effective Date	Date				Eff Date	▽	Edit	⊟
3	A.DESCR - Description	Char30				Descr	▽	Edit	⊟
4	A.ORG_LOCATION - Location Nbr	Num3.0	2			Location	▽	Edit	⊟
5	B.DESCR - Description	Char30				Descr	▽	Edit	⊟
6	B.ADDRESS1 - Address Line 1	Char55				Address 1	▽	Edit	⊟
7	B.ADDRESS2 - Address Line 2	Char55				Address 2	▽	Edit	⊟

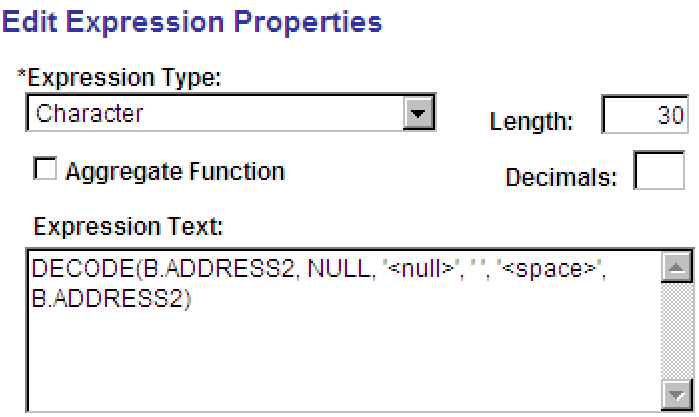


Query Name: TRNG_UNKNOWN_FIELDS Description:

Criteria

Logical	Expression1	Condition Type	Expression 2	Edit	Delete	Belongs to
	A.EFFDT - Effective Date	Eff Date <=	Current Date	Edit	⊟	
AND	A.EFF_STATUS - Status as of Effective Date	equal to	A	Edit	⊟	

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 DECODE and this expression is used as a field for display.



Edit Expression Properties

*Expression Type: Character

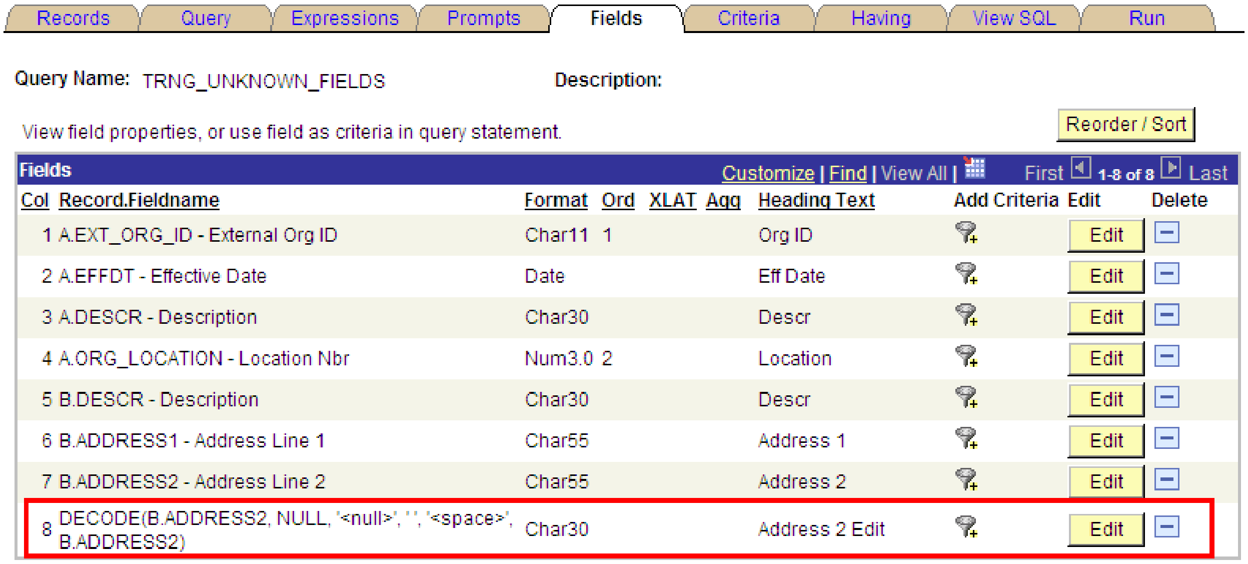
Length: 30

☐ Aggregate Function

Decimals:

Expression Text:

```
DECODE(B.ADDRESS2, NULL, '<null>', ' ', '<space>',
B.ADDRESS2)
```



Query Name: TRNG_UNKNOWN_FIELDS **Description:**

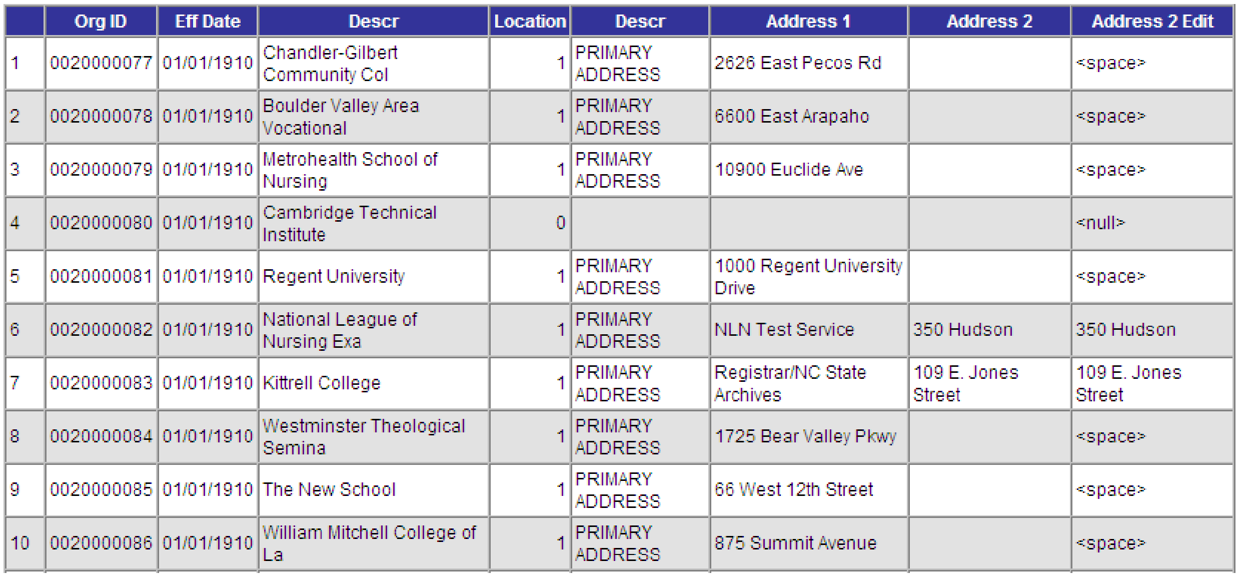
View field properties, or use field as criteria in query statement.

Fields

Col	Record.Fieldname	Format	Ord	XLAT	Agg	Heading Text	Add Criteria	Edit	Delete
1	A.EXT_ORG_ID - External Org ID	Char11	1			Org ID		Edit	−
2	A.EFFDT - Effective Date	Date				Eff Date		Edit	−
3	A.DESCR - Description	Char30				Descr		Edit	−
4	A.ORG_LOCATION - Location Nbr	Num3.0	2			Location		Edit	−
5	B.DESCR - Description	Char30				Descr		Edit	−
6	B.ADDRESS1 - Address Line 1	Char55				Address 1		Edit	−
7	B.ADDRESS2 - Address Line 2	Char55				Address 2		Edit	−
8	DECODE(B.ADDRESS2, NULL, '<null>', ' ', '<space>', B.ADDRESS2)	Char30				Address 2 Edit		Edit	−

The DECODE statement will substitute "<null>" whenever the ADDRESS2 field is NULL (from an unmatched row) and "<space>" whenever the ADDRESS2 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.



	Org ID	Eff Date	Descr	Location	Descr	Address 1	Address 2	Address 2 Edit
1	0020000077	01/01/1910	Chandler-Gilbert Community Col	1	PRIMARY ADDRESS	2626 East Pecos Rd		<space>
2	0020000078	01/01/1910	Boulder Valley Area Vocational	1	PRIMARY ADDRESS	6600 East Arapaho		<space>
3	0020000079	01/01/1910	Metrohealth School of Nursing	1	PRIMARY ADDRESS	10900 Euclide Ave		<space>
4	0020000080	01/01/1910	Cambridge Technical Institute	0				<null>
5	0020000081	01/01/1910	Regent University	1	PRIMARY ADDRESS	1000 Regent University Drive		<space>
6	0020000082	01/01/1910	National League of Nursing Exa	1	PRIMARY ADDRESS	NLN Test Service	350 Hudson	350 Hudson
7	0020000083	01/01/1910	Kittrell College	1	PRIMARY ADDRESS	Registrar/NC State Archives	109 E. Jones Street	109 E. Jones Street
8	0020000084	01/01/1910	Westminster Theological Semina	1	PRIMARY ADDRESS	1725 Bear Valley Pkwy		<space>
9	0020000085	01/01/1910	The New School	1	PRIMARY ADDRESS	66 West 12th Street		<space>
10	0020000086	01/01/1910	William Mitchell College of La	1	PRIMARY ADDRESS	875 Summit Avenue		<space>

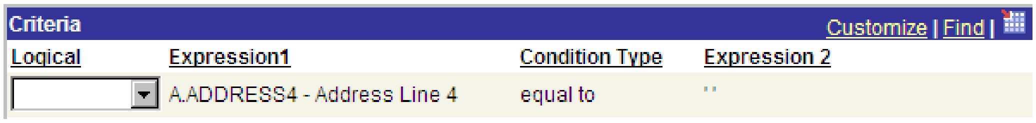
Note how organization 0020000080 has the value "<null>" in the Address 2 Edit field. This indicates that no rows were found in ORG_LOCATION meeting the outer join conditions A.EXT_ORG_ID = B.EXT_ORG_ID and A.ORG_LOCATION = B.ORG_LOCATION.

Organization 0020000081 has the value "<space>" in the Address 2 Edit field. This indicates that there is a row in ORG_LOCATION meeting the outer join conditions, but that the ADDRESS2 field has no data.

Organizations 0020000082 and 0020000083 have actual values in the Address 2 Edit field because there is a matching row in ORG_LOCATION and the ADDRESS2 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 a function such as DECODE; 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 ' '.



Criteria

Logical	Expression1	Condition Type	Expression 2
	A.ADDRESS4 - Address Line 4	equal to	''

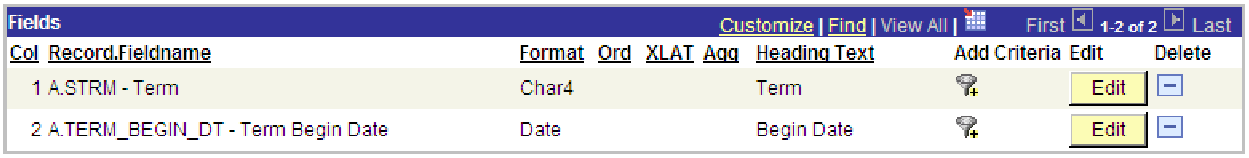
- To write criteria to find rows from the left table of an outer join that have no match in the right table, use any field from the right table and the Condition Type "is null." Place this criterion on the WHERE clause, *not* the ON clause, even though it involves the table being outer-joined. This is 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.

Using Dates in Expressions

Special considerations must be taken when using date fields in expressions. Query Manager automatically converts all values from date fields to text in its processing. For instance, if A.TERM_BEGIN_DT contains the *date* August 24, 2009, Query Manager turns this into the *string* “08/24/2009”.

This can be confirmed by looking at the SQL generated for the query. Consider a query containing a date field, such as the following example:



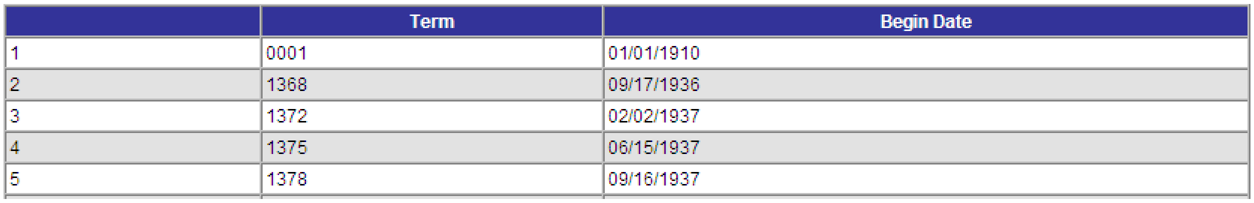
Fields — Customize | Find | View All | First 1-2 of 2 Last

Col	Record.Fieldname	Format	Ord	XLAT	Agg	Heading Text	Add Criteria	Edit	Delete
1	A.STRM - Term	Char4				Term		Edit	⊟
2	A.TERM_BEGIN_DT - Term Begin Date	Date				Begin Date		Edit	⊟

The query made by Query Manager is:

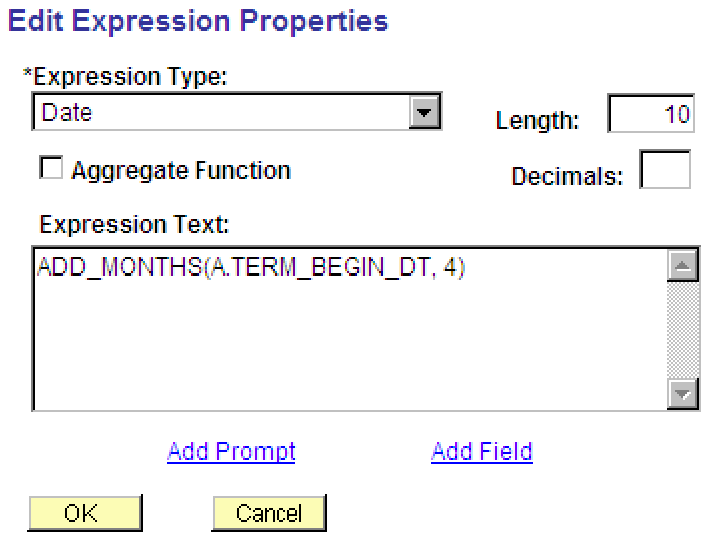
```
SELECT A.STRM, TO_CHAR(A.TERM_BEGIN_DT,'YYYY-MM-DD')
  FROM PS_TERM_TBL A
  WHERE A.ACAD_CAREER = 'UGRD'
```

In the SELECT clause, the value in the TERM_BEGIN_DT field is passed to the TO_CHAR function. Note also that it is converted to a format such that the components appear in the order year, month, and day (ex. “2009-08-04”). This is then converted again when the query results are displayed.



	Term	Begin Date
1	0001	01/01/1910
2	1368	09/17/1936
3	1372	02/02/1937
4	1375	06/15/1937
5	1378	09/16/1937

The consequence of the date being converted to a string is that functions that expect a date value will not work properly. This is because the query engine will not know how to derive a valid date from the string. For example, consider a modified version of the above query to which an expression is added; this expression finds the date four months after the beginning of the term.



Edit Expression Properties

*Expression Type: Date

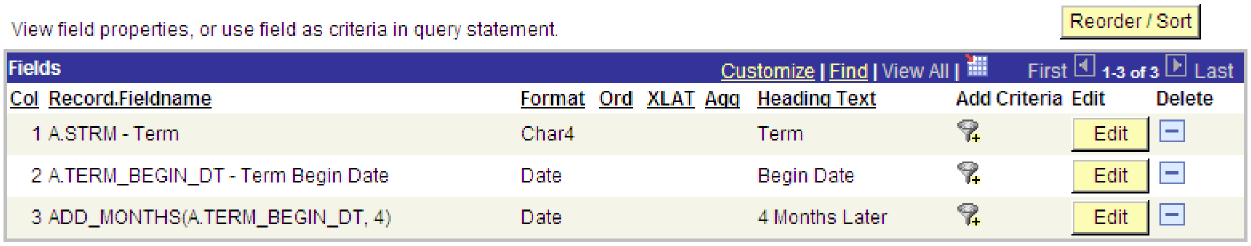
Length: 10

□ Aggregate Function

Decimals:

Expression Text:

```
ADD_MONTHS(A.TERM_BEGIN_DT, 4)
```



View field properties, or use field as criteria in query statement.

Col	Record.Fieldname	Format	Ord	XLAT	Agg	Heading Text	Add Criteria	Edit	Delete
1	A.STRM - Term	Char4				Term		Edit	⊟
2	A.TERM_BEGIN_DT - Term Begin Date	Date				Begin Date		Edit	⊟
3	ADD_MONTHS(A.TERM_BEGIN_DT, 4)	Date				4 Months Later		Edit	⊟

Upon running the revised query, this error is reported: “ORA-01861: literal does not match format string (50,380)”. This error is shown when a string to be converted or displayed in a certain manner is incompatible with the instructions of how to convert or display that string; these instructions are the “format string” referred to in the message.

Viewing the SQL generated for the query can help show why the query fails:

```
SELECT A.STRM, TO_CHAR(A.TERM_BEGIN_DT,'YYYY-MM-DD'),
  ADD_MONTHS( TO_CHAR(A.TERM_BEGIN_DT,'YYYY-MM-DD'), 4)
  FROM PS_TERM_TBL A
  WHERE A.ACAD_CAREER = 'UGRD'
```

The TO_CHAR function marked in red converts the date to a string. If A.TERM_BEGIN_DT contains the date January 11, 2010, TO_CHAR(A.TERM_BEGIN_DT, 'YYYY-MM-DD') produces the *string* “2010-01-11”. Next, ADD_MONTHS, marked in blue, attempts to add four months to the first value provided to it. However, the first parameter is receiving text instead of a date; in this instance, the function call is effectively ADD_MONTHS('2010-01-11', 4). The string “2010-01-11” cannot be automatically converted to a date, so this results in an error.

To work around this aspect of Query Manager, the date field that is implicitly converted to a string must be converted *back* to a date. This can be done by using the TO_CHAR function to transform the string into a date. (See the *Conversion Functions* segment elsewhere in this section.) The following table lists the various types of date fields and the conversion that must be performed to use them in an expression.

Field Type	Convert the Query Manager-converted date to an actual date
Date	TO_DATE(*date field*, 'YYYY-MM-DD')
Datetime	TO_DATE (*datetime field*,'YYYY-MM-DD-HH24.MI.SS. ' || CHR(34) || '000000' || CHR(34))
Time	TO_DATE(*time field*, 'HH24:MI:SS.' || CHR(34) || '000000' || CHR(34))

The ADD_MONTHS expression above would have to be the following in order to work properly in Query Manager: ADD_MONTHS(TO_DATE(A.TERM_BEGIN_DT, 'YYYY-MM-DD'), 4). This will be converted by Query Manager to ADD_MONTHS(TO_DATE(TO_CHAR(A.TERM_BEGIN_DT,'YYYY-MM-DD'), 'YYYY-MM-DD'), 4) when the SQL is built; the string produced by TO_CHAR will be converted to a date by TO_DATE before being given to ADD_MONTHS, satisfying the requirement that ADD_MONTHS receives a date.

Note that the TO_DATE function will produce actual date values and the expressions that use them will *not* be reformatted by Query Manager. Instead, they will be displayed in Oracle's native format, which is a two-digit year, a three-letter month abbreviation, and a two-digit day, separated by dashes (ex. "24-AUG-09"). To display such a date in the same format Query Manager normally uses, the result of the expression must be converted into a string. The following table shows how TO_CHAR can be used to convert the whole expression to a string that matches Query Manager's display format.

Field Type	Convert date expression to string having Query Manager display format	Minimum Length
Date	TO_CHAR(*expression*, 'MM/DD/YYYY')	10
Datetime	TO_CHAR(*expression*, 'MM/DD/YYYY fmHH12fm:MI:SSAM') Note: this format does not exactly match that of Query Manager; the hour is not preceded by an extra space if the hour is less than 10. If the "fm" format modifiers are removed, the format will have the correct length, but the hour will be zero-padded, so the format will still not match exactly. The format model used above is as close to the Query Manager display format as possible without resorting to extremely elaborate conversions.	21

Field Type	**Convert date expression to string having** **Query Manager display format**	**Minimum Length**
Time	TO_CHAR(*expression*, 'fmHH12fm:MI:SS.' || CHR(34) || '000000' || CHR(34) || 'AM')	17

In summary, when using dates in expressions:

1. Query Manager automatically converts the date to a string. Convert it back to a date using the TO_DATE function.
2. Perform other desired operations on the date, such as date arithmetic and using functions such as ADD_MONTHS.
3. If the expression will be used as a field, pass the whole expression as the first argument to TO_CHAR and the desired format model as the second argument. To match the default Query Manager display format, consult the above table.
4. If the expression will be used as a field and in criteria, make two expressions, one that just converts the string to a date, and one that converts the string to a date then back to a string of the desired format.

This repeated conversion between dates and strings is only needed to work around the automatic conversion of date fields to strings in Query Manager. This is not required when running a SQL statement directly.

What Else is There?

There are many more functions and formats beyond those listed here. The Oracle Database SQL Reference (http://www.oracle.com/pls/db10g/portal.portal_demo3?selected=1) 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.

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:

```
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.

```
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 or the period and the field name. “A.DESCR” 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 departments and the names of their managers; when a manager is not known, the department is still listed. (This is the SQL equivalent of exercise TRNG_QM##_E14B.)

```
SELECT A.DEPTID, A.DESCR, A.MANAGER_ID, B.NAME
  FROM (PS_DEPT_TBL A LEFT OUTER JOIN PS_PERSON_NAME B ON A.MANAGER_ID = B.EMPLID)
 WHERE A.EFFDT = (SELECT MAX(A_ED.EFFDT)
                    FROM PS_DEPT_TBL A_ED
                   WHERE A.SETID = A_ED.SETID
                     AND A.DEPTID = A_ED.DEPTID
                     AND A_ED.EFFDT <= SYSDATE)
   AND A.SETID = 'BGSUN'
   AND A.EFF_STATUS = 'A'
```

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 10g, 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.DEPTID, A.DESCR, A.MANAGER_ID, B.NAME
  FROM PS_DEPT_TBL A, PS_PERSON_NAME B
 WHERE A.MANAGER_ID = B.EMPLID(+)
   AND A.EFFDT = (SELECT MAX(A_ED.EFFDT)
                    FROM PS_DEPT_TBL A_ED
                   WHERE A.SETID = A_ED.SETID
                     AND A.DEPTID = A_ED.DEPTID
                     AND A_ED.EFFDT <= SYSDATE)
   AND A.SETID = 'BGSUN'
   AND A.EFF_STATUS = 'A'
```

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 the same time, a query with X = <value1> AND X = <value2> will always return no rows.

- Problems with expressions
    - "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
    - 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).
    - Missing parentheses
        - Each opening parenthesis (must have a corresponding closing parenthesis)
        - Errors can vary depending on how the query engine interprets the generated SQL
    - 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
  - 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
  - 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.