GE VERNOVA

Asset Performance Management
On-Premises APM
V5.2.2.0.0

Queries

Contents

Chapter 3: Query Sources, Fields, and Joins   23

Chapter 4: Query Expressions, Clauses, and Prompts   29

Chapter 5: Query Settings   39

Chapter 6: Workflow   43

Chapter 7: Reference   47

Copyright Digital, part of GE Vernova

Chapter

1

Query Creation

Topics:

About Creating Queries

When you initiate the query creation process, you can choose to select the query sources (entity and relationship families) and query columns (family fields) using either of the following options:

- **Design** workspace: Provides a visual representation of a query, and lets you manually add sources, criteria, and links.
- **SQL** workspace: Provides a workspace into which you can directly enter SQL code to build a query. The SQL View is intended for more advanced Query users.

About Running Queries

Queries are available in many places throughout APM. In some cases, you might create a query manually and save it for future use. In other cases, you might run a query that another user constructed or that is delivered with the baseline content.

When you run a query, you will see the results in the **Results** workspace, which will make it obvious that you are looking at query results. For example, if you open a query from the Catalog or select a dashboard hyperlink that references a stored query, the results will be displayed on the **Results** workspace.

In other cases, however, you might run a query and see the results in a different format on a different screen. In these cases, you might not realize that you are looking at query results. Inspection Management, for example, provides a customized workflow that allows you to execute queries that return specific information, such as equipment that can have bundle inspections. In addition, queries provide a customized form where you can query the database for records in families that meet specific criteria.

When a query is created, it can be configured to return raw data stored in the database or reformatted data. This means that you might run different queries that return the same data in different formats. In addition, queries can be configured to prompt you for information before returning any results. It is a good practice to use parameters instead of literal values. This helps in both performance and reuse.

About Saving Queries

After you create a query, you can save it so that you or other users can access it later. Queries are saved as Catalog items to the APM Catalog.

Details

After a query has been saved to the Catalog, whenever you make changes to it, you will need to re-save the query to retain your changes.

Saving a query is similar to saving any other item to the Catalog, but consider the following query-specific considerations:

- If you try to save a query with invalid SQL syntax, an error occurs.
- Query names must not contain / or \.

About SQL Code

If you are familiar with SQL syntax, instead of designing the query, you can enter SQL code directly and run the query to view the results.

SQL Details

APM supports the use of Oracle and SQL server databases, all of which can be queried using SQL statements. While the same basic SQL code can be used to query any type of database, there are some differences in the syntax that is supported by each database server. Therefore, APM uses a proprietary version of SQL that is constructed automatically and translated at runtime by the system into the SQL syntax that is appropriate for the type of database you are using. We call this form of SQL Meta-SQL.

In most cases, Meta-SQL syntax is the same as standard SQL syntax. This means that in most cases, you can type the SQL syntax that you are familiar with. If, however, you use functions that are specific to one database server (e.g., Oracle), when you run the query on a different database server (e.g., SQL Server), an error appears, and you will be unable to run the query and view its results until you correct the SQL code. It is a good practice to use Meta-SQL whenever possible.

In other words, when you type SQL code directly and select , the following events occur:

- APM reads the syntax and determines whether or not it is valid Meta-SQL.
- APM translates the Meta-SQL into SQL that can be interpreted by the type of database you are using.
- The database executes that translated code and returns results to APM.
- The query results appear.

Throughout the APM documentation, we use the term SQL when referring to the SQL code that appears when you select **SQL** in the content header.

All tasks that you can perform when designing a query write Meta-SQL code that can be viewed when you select the **SQL** tab. Not all SQL code that you enter directly, however, can be interpreted by the **Design** workspace. This means that in some cases, you can write SQL code that will cause the **Design** workspace to be unavailable. If you write a query using SQL code and want other users to be able to modify the query design, first try to access the **Design** workspace to make sure that it is available before saving the query.

Specific instructions for writing SQL code are beyond the scope of this documentation. Where appropriate, we provide guidelines and suggestions for how to write SQL expressions and use SQL functions, but this documentation does not contain comprehensive SQL code explanations.

Important: You must always add aliases for all columns. In standard SQL columns, aliases are optional, but they are required in queries in Meta SQL.

Access the Query Page

Procedure

In the **Applications** menu, navigate to the **TOOLS** section, and then select **Queries**.

The **Query** page appears, displaying a list of queries.

Query

⚲

VIEW QUERY	PATH
Query for Entity Keys Collection	Public\A Temporary Folder to Contain Catalog Items Moved from the Public Fol
Automation_Job_Status_By_FileName	Public\Meridium\Modules\APM Connect\Queries\Automation_Job_Status_By_F
Automation SAP WorkHistory	Public\Meridium\Modules\APM Connect\Queries\Automation SAP WorkHistory
Query to Compare the Latest and Previous AHI Values	Public\Asset Content\Utilities\Queries\Query to Compare the Latest and Previc
Query to Compare the Latest and Previous AHI Values - Date	Public\Asset Content\Utilities\Queries\Query to Compare the Latest and Previc
TEST_CATALOG_READ_MT	Public\Meridium\Queries\TEST_CATALOG_READ_MT
Automation SAP Work History Details	Public\Meridium\Modules\APM Connect\Queries\Automation SAP Work Histor
FluidClassAssets	Public\Public\FluidClassAssets
pVClassAssets	Public\Public\pVClassAssets
Fluid Class Filter	Public\Fluid Class Filter
BoilerClassAssets	Public\Public\BoilerClassAssets
Automation WMI Task Calibration	Public\Meridium\Modules\APM Connect\Queries\Automation WMI Task Calibra
Automation WMI Task Inspection	Public\Meridium\Modules\APM Connect\Queries\Automation WMI Task Inspec
ComplianceStrategy	Public\ComplianceStrategy
RBI DM Query	Public\Meridium\Modules\Risk Based Inspection\Queries\RBI DM Query
Configured Risk Matrix Query	Public\Meridium\Modules\Risk Based Inspection\Queries\Configured Risk Matri

Tip: You can select a link in the **View Query** column to access a results-only view of a query in a new page. You can select the links in the **Path** column to access the full, modifiable view of a query in a new page. You can also access the modifiable view from the results-only view.

To export a query to a file, you can use the **Export to a File** (⤴) button.

Note: Only queries created or accessed via the **Query** page will appear in the list. The 25 most recently accessed queries will appear in the list.

Access the Design Workspace

Procedure

1. To access the **Design** workspace for a new query, perform the following steps.

   a) Access the **Queries** page.
   b) In the upper-right corner, select **Create New**.

      The **Design** workspace appears, and the **Select a Family or Query** window is active, where you can add sources to the design canvas. After adding a source, you can use the design canvas to add fields to the grid in the **Conditions** section and to add criteria to the query.
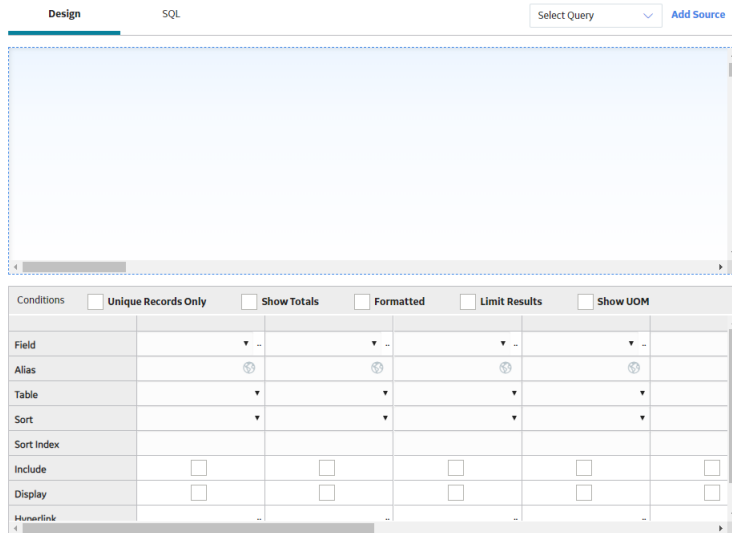
2. To access the **Design** workspace for an existing query, perform the following steps.

   a) Access the **Queries** page.
   b) Select **Browse**.

   The **Select a query from the catalog** window appears.

   c) Navigate to the folder that contains the query you want to view. Select the query, and then select **Open**.

   The **Results** workspace appears.

   d) In the page heading, select **Design**.

   The **Design** workspace appears, where you can add sources to the design canvas. Use the design canvas to add fields to the grid in the **Conditions** section. Use the **Conditions** section to add criteria to the query.

   Note: You cannot use the term Interval as an alias in queries in Postgres.

   Next Steps

   - Add a source

Access the SQL Workspace

Procedure

1. To access the **SQL** workspace for a new query:

   a) Access the **Queries** page.
   b) In the upper-right corner, select **Create New**.

   The **Design** workspace appears.

   c) In the page heading, select **SQL**.

   The **SQL** workspace appears, where you can modify the SQL code directly.

2. To access the **SQL** workspace for an existing query:

   a) Access the **Queries** page.

b) In the upper-right corner, select **Browse**.

   The **Select a query from the catalog** window appears.

c) On the left side of the window, navigate to the folder that contains the query you want to view. Select the query, and then select **Open**.

   The **Results** workspace appears.

d) In the page heading, select **SQL**.

   The **SQL** workspace appears, where you can modify the SQL code directly.

Modify a Query

Procedure

1. Access the **Design** workspace for an existing query, or access the **SQL** workspace for an existing query.
2. Modify the query as needed, and then save the query.
   The query is modified.

Modify the Query Type

About This Task

This topic describes how to select the type of query that you want to create or modify the type of an existing query.

Note: You should not create a Delete query from a Select query that contains relationship families. If you create a Delete query from a Select query, and then you select the relationship family as the target source on the **Target Query Source** window, an error message will appear, and the values in the design canvas and **Conditions** section will be removed.

Procedure

1. Access the **Design** workspace.
2. In the page heading, select the drop-down list box, and then select one of the following query types: Select Query; Crosstab Query; Delete Query; Update Query; Append Query.
   The **Conditions** section changes to display rows that are appropriate for the selected query.

Include or Exclude a Field in the Query Results

Before You Begin

When you create a query, all the fields that you added are selected by default to be included in the query results and displayed in the **Results** workspace. In some cases, you might want to add a field to the grid in the **Conditions** section so that you can define criteria for that field without including it in the query results.

For example, you might want to configure a query that returns all Pumps installed on a certain date. You could add the Pump ID field and the Asset Installation Date to the query, define

criteria to limit the Asset Installation Date to a certain date, and then exclude the Asset Installation Date.

Tip: You can choose to return a field in the query results but not display it in the **Results** workspace. To do so, select the **Include** check box, and clear the **Display** check box.

About This Task

This topic describes how to include or exclude a field from the query results for a Select query.

Procedure

1. Access the **Design** workspace.
2. For the field that you do not want to include in the query results, in the **Include** cell, clear the check box.
   When you run the query, the field will not appear in the results.

Results

- When you clear the **Include** check box for a field, the **Display** check box for that field is automatically cleared. Fields can be displayed in the results only if they are also included in the query results.
- If you exclude a field from the query results, you must define content in the **Criteria** cell, the **Sort** cell, or the **Total** cell in that column for the field to be saved with the query code. In other words, if the field is not included in the query results, it must be included in SQL in some other way to be saved with the query. Otherwise, when you open or run the saved query, the excluded field will not be displayed in the **Design**, **Results**, or **SQL** workspaces.

Create a Crosstab Query

Procedure

1. Access the **Design** workspace.
2. In the page heading, select the drop-down list box, and then select **Crosstab Query**.

   The **Conditions** section is updated to include the following rows:

   - Field
   - Alias
   - Table
   - Total
   - Crosstab
   - Sort
   - Sort Index
   - Criteria
   - Or
3. For the fields that you want to use as column headings in the **Results** workspace, in the **Crosstab** cells, in the drop-down lists, select **Column Heading**.

   Important: The **Total** cell that corresponds to at least one field that you want to use as a column heading must be set to **Group By**.
4. For the fields that you want to use as row headings in the **Results** workspace, in the **Crosstab** cells, in the drop-down lists, select **Row Heading**.

Important: The **Total** cell that corresponds to at least one field that you want to use as a row heading must be set to **Group By**.

5. For the field that you want to use as the aggregate (i.e., the intersection of the row and column), in the **Crosstab** cell, in the drop-down list, select **Value**.

   Important: For the field that you selected as the aggregate, the **Total** cell cannot be set to **Group By**, **Expression**, or **Where**.

6. After you have made your selections, in the page heading, select ⊳ to run the query and confirm that it returns the appropriate results, and then save the query.

Create an Update Query

Procedure

1. Access the **Design** workspace.
2. In the page heading, select the drop-down list box, and then select **Update Query**.
   If your query contains one query source, the source is selected as the target source.

   -or-

   If your query contains more than one query source, the **Target Query Source** window appears.

3. On the **Target Query Source** window, select a source from the sources on the canvas, and then select **Add**. If the query contains more than one query source, the query sources that you do not select are removed automatically from the design canvas.
   The **Conditions** section is updated to include the following rows:

   - Field
   - Table
   - Update To
   - Criteria
   - Or

4. For the fields that you want to modify, in the **Update To** cells, create expressions to specify criteria for the record update. Any record that meets the specified criteria will be updated according to the expression. For example, if you want to update Motor records for which the manufacturer is WEST to the manufacturer WEST&LONG, you can create a criteria expression in the **Update To** cell for the Asset Manufacturer field in the Motor family.

5. In the page heading, select ⊳.
   A dialog box appears, indicating how many records will be updated by the query.

6. To perform the update, select **Yes**, or, to stop the update, select **No**.
   Depending on your selection, the update is performed or stopped.

   Note: When you run an Update query that will update a large number of records, an error message may appear. If this occurs, adjust the query criteria to reduce the number of records that will be updated at one time.

Create an Append Query

Procedure

1. Access the **Design** workspace.
2. In the page heading, select the drop-down list box, and then select **Append Query**.
   The **Append Query - Target Source** window appears.
3. In the **Families** list, select the target family or the family to which you want to append records, and then select **Add**.
   The **Target Source** link appears in the page heading. You can select this link to modify the target family.

   The **Conditions** section is updated to include the following rows:

   - Field
   - Alias
   - Table
   - Sort
   - Sort Index
   - Append To
   - Criteria
   - Or
4. Add one or more sources to the design canvas, and then add one or more fields from those sources to the grid in the **Conditions** section.
5. For the fields that contain records that you want to append to fields in the target family, in the **Append To** cells, in the drop-down lists, select the desired fields in the target family. In the **Criteria** cells, create expressions to specify criteria. Any record that meets the specified criteria will be appended to the specified field in the target family.
6. To run the query, in the page heading, select ▶. Confirm that it returns the appropriate results, and then save the query.

Create a Delete Query

Procedure

1. Access the **Design** workspace.
2. In the page heading, select the **Select Query** box, and then select **Delete Query**.
   If your query contains one query source, the source is selected as the target source.

   -or-

   If your query contains more than one query source, the **Target Query Source** window appears.
3. On the **Target Query Source** window, select a source from the sources on the canvas, and then select **Add**. If the query contains more than one query source, the query sources that you do not select are removed automatically from the design canvas.

   The **Conditions** section is updated to include the following rows:

   - Field
   - Table

- Criteria
- Or

4. For the fields that you want to delete, in the **Criteria** cells, create expressions to specify criteria for the record deletion. Any record that meets the specified criteria will be deleted. For example, if you want to delete Motor records for which the manufacturer is WESTINGHOUSE, you can create the expression 'WESTINGHOUSE' in the **Criteria** cell for the Manufacturer field in the Motor family.

5. If there are relationships that must be deleted when the entity is deleted, select the **Force Delete** check box. Otherwise, an error occurs.

6. In the page heading, select ⊳.
   A dialog box appears, indicating how many records will be deleted by the query.

7. To perform the deletion, select **Yes**, or, to stop the deletion, select **No**.

   Depending on your selection, the deletion is performed or stopped.

Run a Query

Procedure

1. Access the **Design** workspace for the Select query whose results you want to view.

2. In the workspace heading, select ⊳.
   The query results appear in the **Results** workspace.

Next Steps

- Save a query.

Save a Query

Procedure

1. Access the **Design** workspace.

2. In the page heading, select 💾. If the query has already been saved, any changes that you have made will be saved to the same Catalog folder with the existing Catalog item properties. If the query has not yet been saved, the **Save As** window will appear. The following instructions assume that you are saving the query for the first time. If you are viewing a previously saved query and you want to save a copy of the current query with a different name or to a different location, you can select 💾, and then use the following instructions to complete that task. The process is the same as saving a query for the first time.

3. In the folder hierarchy, navigate to the folder in which you want to save the query.

4. In the **Name** box, enter a name for the query. The name is required and must be unique to the Catalog folder in which you are saving the query.
   The **Caption** box is populated automatically with the value that you entered in the **Name** box.

5. In the **Description** box, you can enter a description for the query. This is not required to save the query.
   The **Of type** box is populated automatically with the Catalog item type.

6. Select **Save**.
   The query is saved to the Catalog.

Chapter
2

Query Results

Topics:

About Query Results

The following topics describe available actions related to query results.

Access the Results Workspace

Procedure

1. Access the **Query** page.
2. Select **Browse**.

   The **Select a query from the catalog** window appears.
3. Navigate to the folder that contains the query you want to view, select the query, and then select **Open**.

   The **Results** workspace appears.

Sort the Query Results

Before You Begin
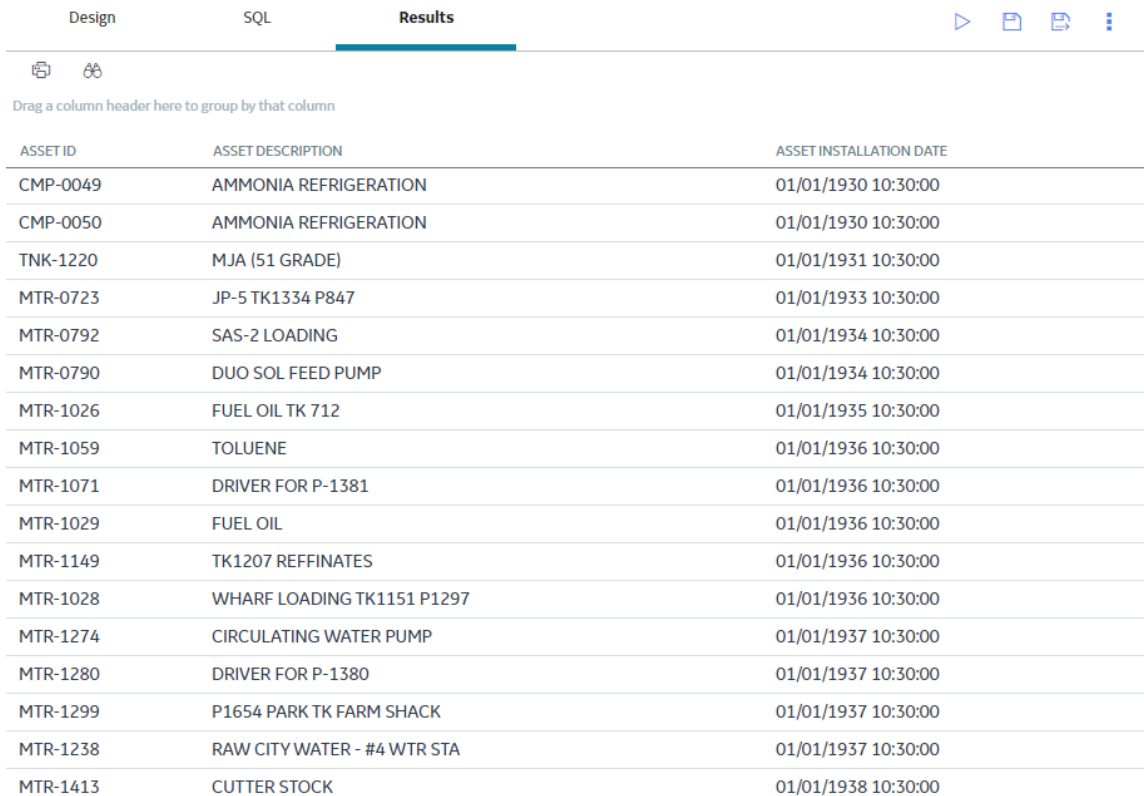
When defining sort criteria for a query, note that:

- The sort criteria determines the default sort order for query results. After the results are displayed, users can modify the sort order in the results grid.
- If no sort criteria has been defined, the query results will not be sorted in any particular order by default. Users can still modify the sort order in the results grid.
- Any sort criteria that is defined for numeric columns in a query will be applied to stored values (vs. displayed values) regardless of whether the query is running in formatted or unformatted mode.
- Unlike the sorting that you can apply directly to the results, sorting preferences that you define in the **Design** workspace are stored with the query itself and are applied each time you run the query.

For example, according to the following image, the Asset Installation Date will be sorted in Ascending order:

Field	[Asset ID]			[Asset Description]			[Asset Installation C	
Alias	Asset ID			Asset Description			Asset Installation Date	
Table	Pump			Pump			Pump	
Sort	None			None			Ascending	
Sort Index	0			0			1	

These sort options will affect the query results, as shown in the following image:

ASSET ID	ASSET DESCRIPTION	ASSET INSTALLATION DATE
CMP-0049	AMMONIA REFRIGERATION	01/01/1930 10:30:00
CMP-0050	AMMONIA REFRIGERATION	01/01/1930 10:30:00
TNK-1220	MJA (51 GRADE)	01/01/1931 10:30:00
MTR-0723	JP-5 TK1334 P847	01/01/1933 10:30:00
MTR-0792	SAS-2 LOADING	01/01/1934 10:30:00
MTR-0790	DUO SOL FEED PUMP	01/01/1934 10:30:00
MTR-1026	FUEL OIL TK 712	01/01/1935 10:30:00
MTR-1059	TOLUENE	01/01/1936 10:30:00
MTR-1071	DRIVER FOR P-1381	01/01/1936 10:30:00
MTR-1029	FUEL OIL	01/01/1936 10:30:00
MTR-1149	TK1207 REFFINATES	01/01/1936 10:30:00
MTR-1028	WHARF LOADING TK1151 P1297	01/01/1936 10:30:00
MTR-1274	CIRCULATING WATER PUMP	01/01/1937 10:30:00
MTR-1280	DRIVER FOR P-1380	01/01/1937 10:30:00
MTR-1299	P1654 PARK TK FARM SHACK	01/01/1937 10:30:00
MTR-1238	RAW CITY WATER - #4 WTR STA	01/01/1937 10:30:00
MTR-1413	CUTTER STOCK	01/01/1938 10:30:00

Note: The sort order that you specify within the query definition will be saved as the default layout for the results.

About This Task

For any query, you can define criteria to determine how the results will be sorted by default.

Procedure

1. For the query whose results you want to sort, access the **Design** workspace.
2. In the **Conditions** section, in the **Sort** cell for the field that you want to sort, select **Ascending** or **Descending**. By default, the **Sort Index** cell for the first field that you sort displays a 1, indicating that this field will be used as the primary sort value.
3. If you want to set a secondary sort value, in the **Sort** cell of the appropriate field, select **Ascending** or **Descending**. By default, the **Sort Index** cell for the second field that you sort displays a 2, indicating that this field will be used as the secondary sort value.
4. Repeat these steps until you have defined all the sort values you want.
   After you run the query, the columns for which you defined a sorting preference will be sorted in the specific order.

Sort Column Values in the Results Workspace

About This Task
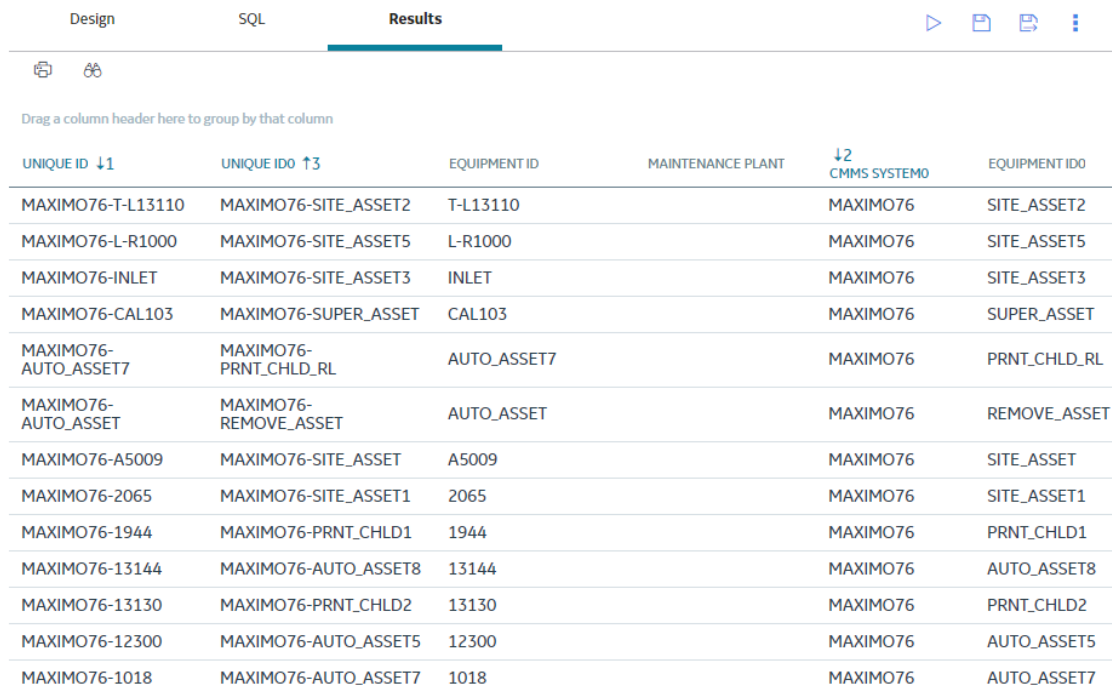
You can sort values in one or more columns in ascending or descending order.

Note: These settings are not saved when you access the **Results** workspace again.

Procedure

1. .
2. To sort the values in a column in ascending or descending order, select the column header.
   The values in the selected column are sorted in ascending or descending order.
3. To sort the values in multiple columns, perform the following steps:

   a) For the column whose values you want to sort, right-click the column header, and then select **Sort Ascending** or **Sort Descending**.
   b) For each additional column whose values you want to sort, right-click the column header, and then select the sort order.

   The values in selected columns are sorted in ascending or descending order.

Design	SQL	**Results**		▷ 🖫 🖺 ⋮	

Drag a column header here to group by that column

UNIQUE ID ↓1	UNIQUE ID0 ↑3	EQUIPMENT ID	MAINTENANCE PLANT	↓2 CMMS SYSTEM0	EQUIPMENT ID0
MAXIMO76-T-L13110	MAXIMO76-SITE_ASSET2	T-L13110		MAXIMO76	SITE_ASSET2
MAXIMO76-L-R1000	MAXIMO76-SITE_ASSET5	L-R1000		MAXIMO76	SITE_ASSET5
MAXIMO76-INLET	MAXIMO76-SITE_ASSET3	INLET		MAXIMO76	SITE_ASSET3
MAXIMO76-CAL103	MAXIMO76-SUPER_ASSET	CAL103		MAXIMO76	SUPER_ASSET
MAXIMO76-AUTO_ASSET7	MAXIMO76-PRNT_CHLD_RL	AUTO_ASSET7		MAXIMO76	PRNT_CHLD_RL
MAXIMO76-AUTO_ASSET	MAXIMO76-REMOVE_ASSET	AUTO_ASSET		MAXIMO76	REMOVE_ASSET
MAXIMO76-A5009	MAXIMO76-SITE_ASSET	A5009		MAXIMO76	SITE_ASSET
MAXIMO76-2065	MAXIMO76-SITE_ASSET1	2065		MAXIMO76	SITE_ASSET1
MAXIMO76-1944	MAXIMO76-PRNT_CHLD1	1944		MAXIMO76	PRNT_CHLD1
MAXIMO76-13144	MAXIMO76-AUTO_ASSET8	13144		MAXIMO76	AUTO_ASSET8
MAXIMO76-13130	MAXIMO76-PRNT_CHLD2	13130		MAXIMO76	PRNT_CHLD2
MAXIMO76-12300	MAXIMO76-AUTO_ASSET5	12300		MAXIMO76	AUTO_ASSET5
MAXIMO76-1018	MAXIMO76-AUTO_ASSET7	1018		MAXIMO76	AUTO_ASSET7

Note: The columns whose values are sorted in ascending order are indicated by ↑ in the column headers, and the columns whose values are sorted in descending order are indicated by ↓ in the column headers. The numbers appearing in the column headers denote the sequence in which you have sorted the columns.

Tip: To clear the sorting from a column, right-click the column header, and then select **Clear Sorting**.

Group by Column Values in the Results Workspace

About This Task

You can group query results based on the values in a column.

Note: These settings are not saved when you access the **Results** workspace again.

Procedure

1. Access the **Results** workspace.
2. To group the results based on the values in a column, perform the following steps:

    a) Right-click the column header and select **Group by This Column**.

    Tip: You can also drag and drop a column header on an area outside the result grid to create a group.

    The results are grouped based on the values in the selected column.

    b) To create sub-groups, right-click additional column headers and select **Group by This Column**.
    A sub-group is created based on the values in the selected column.

Note: The groups are automatically sorted in ascending order. You can change the order by selecting the grouped column name.

Tip: To remove a group, perform the following steps:

- Right-click the grouped column name and select **Ungroup**. You can also drag and drop a grouped column name on an area within the result grid to remove the group.
- To remove all groups, right-click any grouped column name or column header, and then select **Ungroup All**.

Filter the Query Results
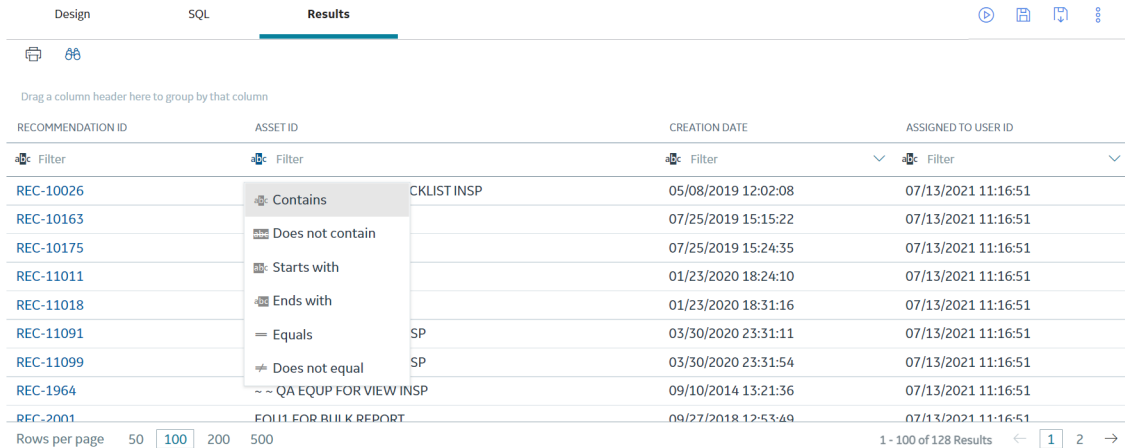
About This Task

You can filter the values in the query results by setting a filter in one or more columns.

Note: These settings are not saved when you access the **Results** workspace again.

Procedure

1. Access the **Results** workspace.

2. Select 👓 .
   A Filter row is added below each column name.

3. Select ⓐ below the column name in which you want to set the filter criterion.
   A window with a list of filter options appears.

Design	SQL	**Results**		▷ 🖫 🖫 ⋮

🖨 👓

Drag a column header here to group by that column

RECOMMENDATION ID	ASSET ID		CREATION DATE	ASSIGNED TO USER ID	
ⓐ Filter	ⓐ Filter		ⓐ Filter ⌄	ⓐ Filter	⌄
REC-10026		CKLIST INSP	05/08/2019 12:02:08	07/13/2021 11:16:51	
REC-10163	ⓐ Contains		07/25/2019 15:15:22	07/13/2021 11:16:51	
REC-10175	ⓐ Does not contain		07/25/2019 15:24:35	07/13/2021 11:16:51	
REC-11011	ⓐ Starts with		01/23/2020 18:24:10	07/13/2021 11:16:51	
REC-11018	ⓐ Ends with		01/23/2020 18:31:16	07/13/2021 11:16:51	
REC-11091	= Equals	SP	03/30/2020 23:31:11	07/13/2021 11:16:51	
REC-11099	≠ Does not equal	SP	03/30/2020 23:31:54	07/13/2021 11:16:51	
REC-1964	~ ~ QA EQUP FOR VIEW INSP		09/10/2014 13:21:36	07/13/2021 11:16:51	
REC-2001	EQU1 FOR BULK REPORT		09/27/2018 12:53:49	07/13/2021 11:16:51	

| Rows per page | 50 | 100 | 200 | 500 | | 1 - 100 of 128 Results | ← | 1 | 2 | → |

4. Select the option for filtering the values in the column.

5. Enter the value for the filter in the filter cell of the column.
   The values are filtered as per the criterion set in the column.

Export a Query Result Set to a File

About This Task

Note that, when you run a query in unformatted mode, the results will still display formatted date values. Date values will always be displayed in the local time for the user; however, if you export the unformatted query result set to a file or dataset, the exported date values will be unformatted. Formatted queries and queries with a large number of results take a longer time to export.

Therefore, long running queries should be exported in the background.

Note: When running the query during export, you can specify the page size to be used by the query engine.

Procedure

1. For the Select query for which you want to export, access the **Design** workspace.

2. In the upper-right corner of the workspace, select ⋮ , and then select **Export to a File**.

   The **Export to a File** window appears.

3. In the **Please provide a File Name** box, provide the file name.

4. In the drop-down list, select the desired file type.

5. Select the page size to be used by query execution engine during export to file. By default, this value is set to 50000.

6. If this is a long running query, select the **Run in background** check box.

7. Select **Export**.

   - If you choose to run the export immediately, the exported file will download using your browser. Save the file to the desired location.
   - If you choose to run the export in the background, you will get a message that it has been submitted.
     From the queries overview page, select **Query Export**. A list of exported query results will be displayed.

     - If the status is **Submitted**, your export has not started.
     - If the status is **Running**, your export is currently running.
     - If the status is **Finished**, you may download the results by selecting the row and selected the download icon.
     - If the status is **Failed**, your export failed, and the error message is in the **Errors** column.

8. After downloading, you may delete the export by selecting the row, and then selecting the **Delete** icon.

   Note: If the query that is being exported contains a lot of pages, the export process is slow. For queries which results in less number of records, there is no difference in export time. For queries with large number of results, if you set a higher page size value, the time required to export can be reduced.

Export a Query Result Set to a Dataset

About This Task

A query can be exported as a dataset, a fixed set of information that is stored as a Catalog item.

Note: When you run a query in unformatted mode, the results will still display formatted date values. Date values will always be displayed in the local time for the user; however, if you export the unformatted query result set to a file or dataset, the exported date values will be unformatted.

Procedure

1. For the Select query for which you want to export, access the **Design** workspace.

2. In the upper-right corner of the workspace, select ⋮ , and then select **Export to a Dataset**.
   The **Save As** window appears.

3. In the folder hierarchy, navigate to the folder in which you want to save the query.

4. In the **Name** box, enter a name for the dataset. If you select the same name and location as an existing dataset, that existing dataset will be overwritten when you save.
   The **Caption** box is populated automatically with the value that you entered in the **Name** box.

5. In the **Description** box, you can enter a description for the dataset. This is not required to save the query.

6. The **Of type** box is populated automatically with the Catalog item type.

7. Select **Save**.
   The query result set has been exported to a dataset.

Modify the Value in the Field Cell

Procedure

1. Access the **Design** workspace.
2. In the **Conditions** section, in the grid, in the **Field** cell for the column whose value you want to define, select the gray button in the right side of the cell.

   The **Advanced** section of the **Expression Builder** window appears.
3. In the text box, enter the text you want to appear in the results, surrounded in single quotation marks.
4. Select **Done**.
5. In the page heading, select ▷.

Remove the Limit on the Number of Results

Procedure

1. For the query for which you want to remove the limit on the number of results that are returned, access the **Design** workspace.
2. In the heading of the **Conditions** section, clear the **Limit Results** check box.
3. The **Limit Results to Top** box is disabled, and the number that appeared in it is removed.

4. In the content heading, select ▷.
   The query results are displayed, and the number of results that are returned is no longer limited.

Modify the Output Mode of a Select Query

Procedure

1. For the Select query for which you want to modify the output mode, access the **Design** workspace.
2. In the grid in the **Conditions** section, depending on which mode you want to use, select or clear the **Formatted** check box as needed.

3. In the content header, select ▷.
   The query is run in the selected mode, and the results appear.

   Note: When you run a query in unformatted mode, the results will still display formatted date values. Date values will always be displayed in the local time for the user.

   Tip: You can save the query to save the selected mode. The next time you run the query, it will use the mode that was selected when it was last saved.

Display Unique Records Only

Procedure

1. For the query for which you want to display unique records only, access the **Design** workspace.
2. In the grid in the **Conditions** section, select the **Unique Records Only** check box.
3. The **Limit Results to Top** box is disabled, and the number that appeared in it is removed.
4. In the content heading, select ▷.
   The query is run, and the results will include only unique records.

Limit the Number of Results

Procedure

1. For the query for which you want to define the number of results that are returned, access the **Design** workspace.
2. In the **Conditions** section heading, select the **Limit Results** check box.
3. In the **Limit Results to Top** box, enter the number of results you want the query to return.
4. In the content header, select ▷.
   The query results are displayed according to your selection in the **Records** box. For example, if you enter 20 in the **Records** box, the query will return only the first 20 records that meet the other query criteria.

Aggregate Query Results

Before You Begin

Ensure that all the fields for which you want to aggregate results have a total type selected (for example, group by, count).

Procedure

1. For the query for which you want to aggregate the query results, access the **Design** workspace.
2. Select the **Show totals** check box.
3. In the content header, select ▷.
   The query results are aggregated.

Show the Units of Measurement

Procedure

1. For the query for which you want to see the units of measurement in the query results, access the **Design** workspace.

2. Select the **Show UOM** check box.

3. In the content header, select ⊳.
   The units of measurement appear in the query results.

Create a Hyperlink

Procedure

1. To create an internal URL:

   a) Access the **Design** workspace of the query in which you want to create hyperlinks.
   b) In the grid in the **Conditions** section, for the field on which you want to create the

      hyperlink, in the **Hyperlink** cell, select .
      The **URL Builder** window appears.

      Note: Do not create a hyperlink on a date field. Doing so will convert the value in the date field to the local time of the user creating it, and the value will be incorrect for users in different time zones.

   c) In the **Select URL Format** section, select **Meridium APM**, and then select **Next**.
   d) In the **Select URL Feature Area** section, in the list, select the area of APM to which you want the URL to point, and then select **Next**.
   e) In the **Select URL Type** section, in the list, select the specific location to which you want the URL to point, and then select **Next**.
   f) In the **Specify Parameters** section, specify the Entity Key and any other values necessary to populate the URL, and then select **Finish**.

      Note: If the parameter value you specify includes a special character, you must enter the URL-encoded value. For example, the manual hyperlink to open a record in record manager using a datasheet with the ID `TEST?&%` must be entered as `#record-manager/{1}?datasheetid= TEST%3F%26%25`, assuming that the entity key of the record appears in column 1 of the query.

      The **Expression Builder** window closes, and then, in the grid in the **Conditions** section, <expr> appears in the field on which you created the prompt.

2. To create an external URL:

   a) Access the **Design** workspace of the query in which you want to create hyperlinks.
   b) In the grid in the **Conditions** section for the field on which you want to create the

      hyperlink, in the **Hyperlink** cell, select . Do not create a hyperlink on a date field. Doing so will convert the value in the date field to the local time of the user creating it, and the value will be incorrect for users in different time zones.
      The **URL Builder** window appears.

   c) In the **Select URL Format** section, select **Manual**.
   d) In the **Specify Parameters** section, enter the URL to which you want the hyperlink to correspond, and then select **Finish**.

      Note: If the parameter value you specify includes a special character, you must enter the URL-encoded value. For example, the manual hyperlink to open a record in record manager using a datasheet with the ID `TEST?&%` must be entered as `#record-manager/{1}?datasheetid= TEST%3F%26%25`, assuming that the entity key of the record appears in column 1 of the query.

The **Expression Builder** window closes, and in the grid in the **Conditions** section, <expr> appears in the field on which you created the expression.

Results

Note: If you clear the **Included** check box for the System field, and then run the query in formatted mode, selecting a hyperlink in the query results may cause the following error message to appear: An entity with the key '{<X>}' could not be found. Please close the tab. If this occurs, select the **Included** check box for the System field, and then rerun the query.

- You cannot modify a hyperlink using the **URL Builder** window. The URL Builder interface does not store settings for existing hyperlinks.
- If you want to use the **URL Builder** window to modify a hyperlink, you can recreate the hyperlink and include your modifications in the **URL Builder** window.

Delete a Hyperlink

Procedure

1. Access the **Design** workspace.
2. In the grid in the **Conditions** section, for the cell in which you want to delete a hyperlink, select ✕.
   The hyperlink is removed from the query.

Chapter
3

Query Sources, Fields, and Joins

Topics:

About Query Sources, Fields, and Joins

The following topics describe available actions related to query sources, fields, and joins.

Add a Source

Procedure

1. To add a source to a new query:

   a) Access the **Design** workspace.
   b) Depending on the type of query source that you want to use, on the **Select a Family or Query** window, select either the **Families** or **Queries** tab.
      Depending on your selection, the list of entity families or Catalog folders to which you have at least View privileges appears.

      Note: If all sources have been removed from the **Design** workspace, in the page heading, select **Add Source** to open the **Select a Family or Query** window.
   c) In the **Families** section, in the list, select an entity family. Or, in the **Queries** section, navigate through the folder hierarchy and select a query.
   d) Select **Add**.
      The **Select a Family or Query** window closes, and your selected query source appears as a node on the design canvas.
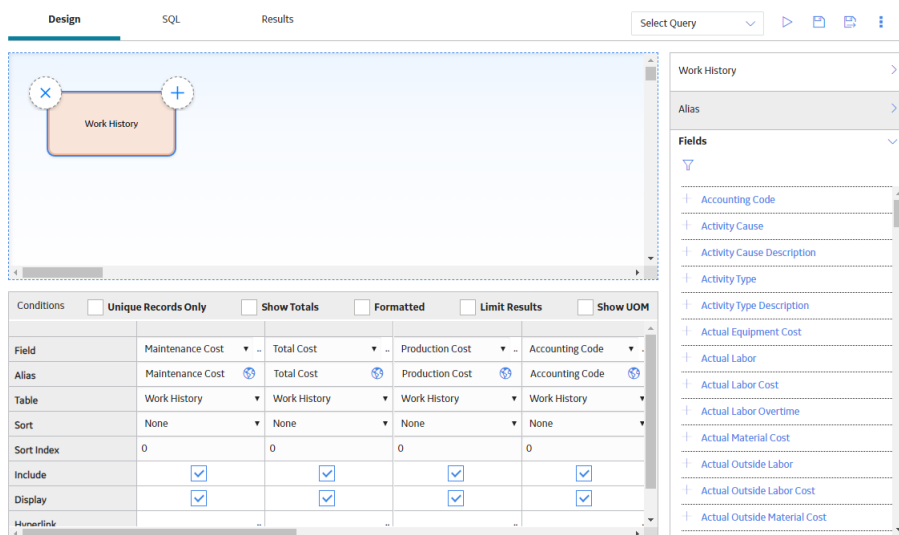
2. To add a related source:

   a) Access the **Design** workspace.
   b) On the design canvas, select the source to which you want to add a related source.
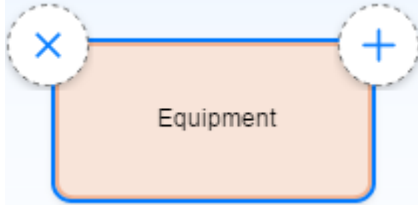
c) Select the plus symbol, and then drag the pointer to another location on the design canvas.
A blank node appears, and, in the direction that you drag the pointer, a line connecting the two nodes is drawn automatically.
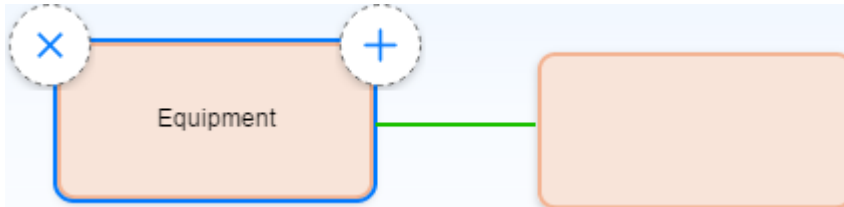
d) In the location where you want to place the related source, release the pointer.

The **Add Related Source** window appears.

e) Depending on the type of related source you want to use, select one of the following tabs. Select the **Add Related Family** tab to access the list of entity families to which you have at least View privileges, as well as the corresponding APM relationship families that link those entity families to the source you selected on the design canvas. Select the **Add Manual Join** tab to access options that allow you to manually link a field in the predecessor source (the source you selected) to a field in a successor family or query. Once you select a successor source, you can then define the type of join you want to use.

f) In the **Add Related Family** section, select the desired entity family and relationship family combination. Alternatively, add a manual join via the **Add Manual Join** section.

g) Select **Add**.

The design canvas appears, and the new source node is populated with the selected source. The line connecting the two nodes changes to represent one of the following types of relationships.

Join Type	Line Style
APM Inner Join	
APM Outer Left Join	
APM Outer Right Join	
Manual Inner Join	
Manual Outer Left Join	
Manual Outer Right Join	

Next Steps

- Add a field to a query.

Add a Field to a Query

Procedure

1. Access the **Design** workspace.
2. In the design canvas, select the source that contains the field you want to add to the conditions grid.
   The **[<Source Name>]** window appears, where **[<Source Name>]** is the name that appears on the query source node in the design canvas.
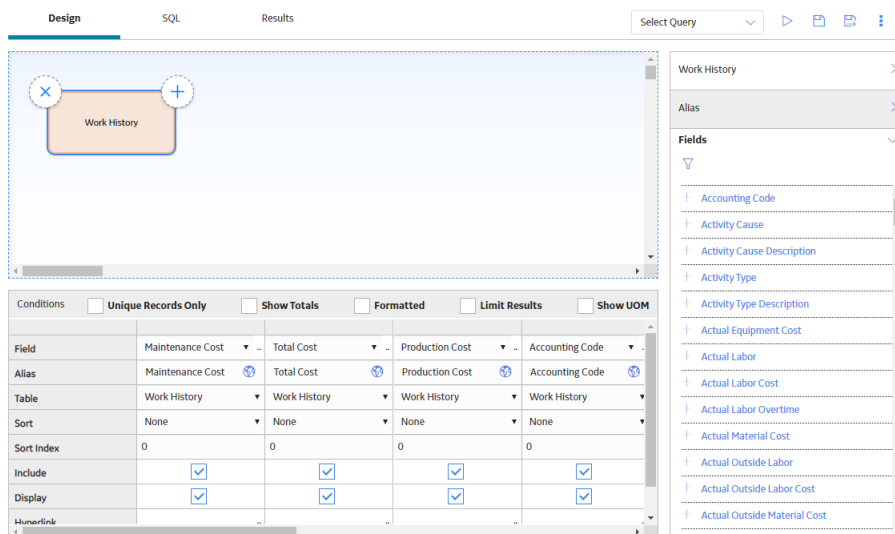
3. In the **[<Source Name>]** window, in the **Fields** section, select the field that you want to add to the query.
   The field appears in the grid in the **Conditions** section.

   Note: If you add the same field from one source to the grid in the **Conditions** section twice, you must specify a unique value in the **Alias** cell for each field. If you do not specify a unique alias, APM will add a unique alias automatically.

Results

- The **Field** cell contains a drop-down list, where you can modify your field selection.

Next Steps

- Run the query.

Arrange Columns

Procedure

1. Access the **Design** workspace.
2. In the **Conditions** section, select the heading of the column that you want to move, drag the pointer to a different location within the grid, and then release the pointer.
   The column is moved to the specified location.

Modify the Properties of a Join

Procedure

1. To modify a join:

   a) Access the **Design** workspace.
   b) On the design canvas, select the line that connects two query sources.
      The window for the selected relationship family appears.
   c) Select **Join Properties**.
      The **Join Properties** window appears.
   d) On the **Join Properties** window, modify the join type to one of the following options, and then select **Done**:

Option	Description
Inner join	Represented by the **Only include rows that are linked through '<Relationship Family>'** option.
Left outer join	Represented by the **Include ALL rows from '[<predecessor source>]' and only those rows from '[<successor source>]' where the joined fields are equal** option.
Right outer join	Represented by the **Include ALL rows from '<Successor Family>' and only those rows from '<Predecessor Family>' that are linked through '<Relationship Family>'** option.

2. To modify a manual join:

   a) Access the **Design** workspace.
   b) On the design canvas, select the line that connects two query sources.
      The window for the selected relationship family appears.
   c) Select **Join Properties**.
      The **Join Properties** window appears.
   d) On the **Join Properties** window, modify the join type to one of the following options, and then select **Done**:

   - Inner join, represented by the **Only include rows that are linked through '<Relationship Family>'** option.
   - Left outer join, represented by the **Include ALL rows from '[<predecessor source>]' and only those rows from '[<successor source>]' where the joined fields are equal** option.
   - Right outer join, represented by the **Include ALL rows from '<Successor Family>' and only those rows from '<Predecessor Family>' that are linked through '<Relationship Family>'** option.

   The properties of the join are modified.

Delete a Join

Procedure

1. Access the **Design** workspace for the query from which you want to delete a join.

2. On the design canvas, delete a source that is directly connected to the join that you want to delete.
The join is deleted.

Display the System and Inactive Fields for a Query Source

Procedure

1. Access the **Design** workspace.
2. Add one or more sources to the design canvas.
3. Select the source node for which you want to display system or inactive fields.
The window for the selected source node appears.

4. In the window, in the **Fields** section, select ⏷. Then, in the drop-down list, select either one or both of the **Show System Fields** and **Show Inactive Fields** check boxes.
Depending on your selection, the system and/or inactive fields for the family appear in the **Fields** section, at the end of the list. System fields are identified by their field IDs, and inactive fields are identified by their field captions appearing within square brackets.

Remove a Query Source

Procedure

1. Access the **Design** workspace for the query from which you want to remove a source.
2. On the design canvas, select the source node that you want to remove.
The option buttons appear just above the source node.

3. Select the ✕ button.

The source and all connected joins are removed from the query.

Chapter
4

Query Expressions, Clauses, and Prompts

Topics:

About Query Expressions, Clauses, and Prompts

Create an Expression

Before You Begin

When you construct expressions in the **Expression Builder** window, the system does not check whether your syntax is correct until you save it.

About This Task

There are two sections on the **Expression Builder** window: the **Simple** section and the **Advanced** section. The steps that you follow to create an expression on a field depend on which cell that you select in the grid in the **Conditions** section, and which type of expression you want to create.

Procedure

1. To create a simple expression:

   a) For the query in which you want to create a simple expression, access the **Design workspace**.
   b) In the grid in the **Conditions** section, for the field on which you want to create the expression, in either the **Criteria** cell or the **Or** cell, select the button in the right side of the cell.
      The **Simple** section of the **Expression Builder** window appears.
   c) Using the fields provided, construct your expression.
   d) Select **Done**.

      The **Expression Builder** window closes, and in the grid in the **Conditions** section, <expr> appears in the field on which you created the expression. You can modify the expression directly in the **Advanced** section of the **Expression Builder** window.
2. To create an advanced expression:

   a) For the query in which you want to create an advanced expression, access the **Design workspace**.
   b) In the grid in the **Conditions** section, in the appropriate **Field**, **Criteria**, or **Or** cell, select the gray button in the right side of the cell. If the **Simple** section is currently displayed, then select **Advanced**.
      The **Advanced** section of the **Expression Builder** window appears.
   c) Using the fields provided, construct your expression.
   d) Select **Done**.
      The **Expression Builder** window closes, and in the grid in the **Conditions** section, <expr> appears in the field on which you created the expression.

      Tip: If you re-access the **Expression Builder** window via any cell, you can modify the expression directly in the **Advanced** section of the **Expression Builder** window, or you can modify previously made selections to update the expression.

Create a WHERE Clause

Procedure

1. Access the **Design** workspace for the query within which you want to create a WHERE clause.
2. For an aggregate query, select **Where** in the **Total** cell.
3. Access the **Expression Builder** window from the **Criteria** or **Or** cells, enter the necessary parameters, and then select **Done**.
4. For an aggregate query, clear the **Include** and **Display** check boxes in the column in which you want to create a WHERE clause.

Results

- The WHERE clause is generated in the SQL code.
- If you want to modify the clause, you can do so in the **Expression Builder** window, or you can modify the SQL code directly.

Create a HAVING Clause

Procedure

1. Access the **Design** workspace for the query within which you want to create a HAVING clause.
2. In the **Conditions** section, make sure the **Show Totals** check box is selected.
3. In the **Total** cell of the desired field, in the drop-down list, select any option other than **Where**.
4. Access the **Expression Builder** window from the **Criteria** or **Or** cells, enter the necessary parameters, and then select **Done**.

Results

- The HAVING clause is generated in the SQL code.
- If you want to modify the clause, you can do so in the **Expression Builder** window, or you can modify the SQL code directly.

Delete an Expression

Procedure

1. Access the **Design** workspace.
2. In the grid in the **Conditions** section, for the cell in which you want to delete an expression, select ✕.
   The expression is removed from the query.

Access the Prompt Settings Section

1. Access the **Advanced** section of the **Expression Builder** window for the field on which you want to create a prompt.
2. Select ? .
   The **Prompt Settings** section appears.
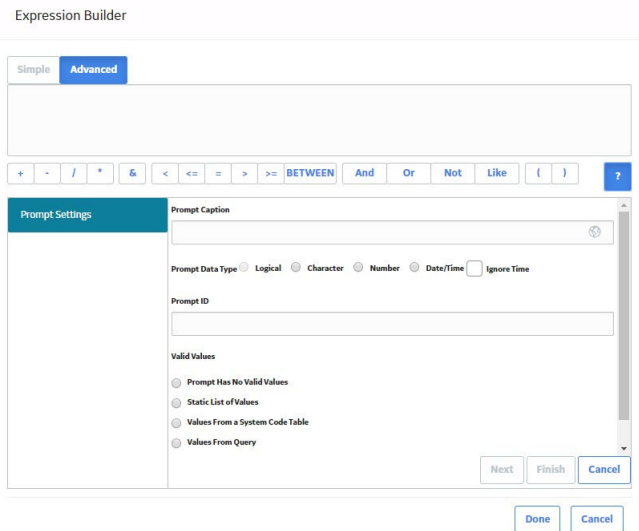
Note: Ensure that you always define a Prompt ID. This value is used internally by APM to identify the prompt, and if not specified, it can cause unexpected behaviours.

Create a Prompt with No List of Valid Values

Procedure

1. Access the **Prompt Settings** section of the **Expression Builder** window.
2. Define the prompt settings as needed.
3. In the **Valid Values** list, select **Prompt Has No Valid Values**, and then select **Next**.
   The **Configure Default Prompt Value** section appears.
4. In the **Default Prompt Value** box, you can enter a default value for the prompt that will appear when a user runs the query, then select **Finish**.
   The prompt appears in the **Expression Builder** window.
5. Select **Done**.
   The **Expression Builder** window closes, and then, in the grid in the **Conditions** section, <expr> appears in the field on which you created the prompt.

Results

• When a user runs the query, the prompt will appear. If you specified a default prompt value, that value will appear in a modifiable text box.

- If the user enters a value that does not exist in the limiting field, the query will not return any results. For example, if two asset types exist in the results, such as Rotating Pumps and Centrifugal Pumps, and the user enters `Reciprocating Pump`, the query results will be empty.

Create a Prompt with a Static List of Valid Values

Procedure

1. Access the **Prompt Settings** section of the **Expression Builder** window.
2. Define the prompt settings as needed.
3. In the **Valid Values** list, select **Static List of Values**, and then select **Next**.
   The **Static List of Valid Values** section appears.
4. In the **Enter List of Values** box, enter the first value that you want to appear in the list, and then select ✛. Repeat this step to continue adding values as needed. To delete a value from the list, select the corresponding 🗑. If you are defining a list of values for a numeric field and you enter a non-numeric character in the list, then the **Next** button will be disabled. You will not be able to proceed or close the Prompt Builder until you replace the non-numeric character with a numeric value.
5. Select the **Values are Exclusive** check box if you want users to be able to select only from the list of defined values in the prompt window. If you do not select this check box, users will be able to enter alternate values.
6. Select the **Allow Multiple Selections** check box if you want users to be able to select multiple values by which to filter the query results. The **Allow Multiple Selections** check box is enabled only if you select the **Values are Exclusive** check box.
7. Select **Next**.
   The **Configure Default Prompt Selection** section appears.
8. If you want a particular value to be selected by default, then select the row containing the necessary value, and then select **Finish**.

   Note: You can select multiple default prompt values only if the Prompt Data Type is set to Character, and you selected the **Allow Multiple Selections** check box in the **Static List of Valid Values** section.

   The prompt appears in the **Expression Builder** window.
9. Select **Done**.
   The **Expression Builder** window closes, and then, in the grid in the **Conditions** section, <expr> appears in the field on which you created the prompt.

Results

- When a user runs the query, the prompt will display the list of predefined values. The user will need to select a value to view the query results.

Create a Prompt with a List of System Codes

Procedure

1. Access the **Prompt Settings** section of the **Expression Builder** window.
2. Define the prompt settings as needed.

3. In the **Valid Values** list, select **Values From a System Code Table**, and then select **Next**.
   The **Values From a System Code Table** section appears.
4. In the **System Code Table** list, select the System Code Table whose values you want to display in the prompt.
5. Select the **Use Reference System Code** check box if you want to specify a referenced System Code Table (i.e., a System Code Table that is referenced within the original table in the Configuration Manager). If you select this check box, then select values in the **Reference Table** and **Reference System Code** lists.
6. Select the **Values are Exclusive** check box if you want users to be able to choose only from the list of presented values in the prompt window. If you do not select this check box, users will be able to enter an alternate value for the prompt.
7. Select the **Allow Multiple Selections** check box if you want users to be able to select multiple values by which to filter the query results. The **Allow Multiple Selections** check box is enabled only if you select the **Values are Exclusive** check box.
8. Select **Next**.
   The **Configure Default Prompt Selection** section appears.
9. If you want a particular value to be selected by default, then select the row containing the necessary value, and then select **Finish**.

   Note: You can select multiple default prompt values only if the Prompt Data Type is set to Character, and you selected the **Allow Multiple Selections** check box in the **Values From a System Code Table** section.

   The prompt appears in the **Expression Builder** window.
10. Select **Done**.
    The **Expression Builder** window closes, and then, in the grid in the **Conditions** section, <expr> appears in the field on which you created the prompt.

Results

- When a user runs the query, the prompt will contain a list of values as defined in the associated System Code Table.

Create a Prompt with a List of Query Results

Before You Begin

You can use only a Select query for generating a list of prompt values. Note that:

- If the selected query contains only one column, the values return in that column will be displayed in the list of available prompt values. For example, if the query contains the Equipment ID column, the prompt will display a list of Equipment IDs.
- If the selected query contains more than one column, the list of prompt values will contain a concatenated list of values from the second column and each subsequent column. For example, if the query results look like this table:

Equipment ID	Manufacturer	Description	Status
123	Alco	Tank	Active
456	Whitlock	Pump	Inactive
789	Delta	Pressure Vessel	Active

... the list of prompt values will look like this:

- Alco Tank Active
- Whitlock Pump Inactive
- Delta Pressure Vessel Active

About This Task

When you create a prompt that presents a list of results from another query, the prompt selection dialog box will display a drop-down list of values retrieved by running that query.

Tip: You can use a prompt with a list of query results to filter a dependent prompt's values.

Procedure

1. Access the **Prompt Settings** section of the **Expression Builder** window.
2. Define the prompt settings as needed.
3. In the **Valid Values** list, select **Values From Query**, and then select **Next**.
   The **Values From a Query** section appears.
4. In the **Enter query text or click Browse button to select an existing query** box, enter SQL code directly. If you completed this task, then proceed to Step 6. Otherwise, select **Browse**, and then proceed to step 5.
   The **Select a query from the catalog** window appears.
5. Select a query from the folder hierarchy, and then select **Open**.
6. Select the **Values are Exclusive** check box if you want users to be able to choose only from the list of presented values in the prompt window. If you do not select this check box, users will be able to enter an alternate value for the prompt.
7. Select the **Allow Multiple Selections** check box if you want users to be able to select multiple values by which to filter the query results.

   Note: The **Allow Multiple Selections** check box is enabled only if you select the **Values are Exclusive** check box.
8. Select **Next**.
   The **Configure Default Prompt Selection** section appears.
9. If you want a particular value to be selected by default, then select the row containing the necessary value, and then select **Finish**.
   The prompt appears in the **Expression Builder** window.

   Note: You can select multiple default prompt values only if the Prompt Data Type is set to Character and you selected the **Allow Multiple Selections** check box in the **Values From Query** section.
10. Select **Done**.
    The **Expression Builder** window closes, and then, in the grid in the **Conditions** section, <expr> appears in the field on which you created the prompt.

Create a Prompt with a List of Values from a Record

About This Task

When you build a prompt that presents a list of fields from a table, the prompt will display a list of values pulled from the specified field of all records in a given family. For example, you could build a prompt that contains values pulled from the Taxonomy Type field of all records in the Equipment family.

Tip: You can use a prompt with a list of values from a record to filter a dependent prompt's values.

Note: For a family with a large number of records, it will take a long time to populate this list.

Procedure

1. Access the **Prompt Settings** section of the **Expression Builder** window.
2. Define the prompt settings as needed.
3. In the **Valid Values** list, select **Distinct List of Values From [X]**, where **X** is the name of the field for which you are defining the prompt criteria, and then select **Next**.
   The **Values From a Query** section appears. And the **SQL** box is populated with a query to get the distinct values.
4. Select the **Values are Exclusive** check box if you want users to be able to choose only from the list of presented values in the prompt window. If you do not select this check box, users will be able to enter an alternate value for the prompt.
5. Select the **Allow Multiple Selections** check box if you want users to be able to select multiple values by which to filter the query results. The **Allow Multiple Selections** check box is enabled only if you select the **Values are Exclusive** check box.
6. Select **Next**.
   The **Configure Default Prompt Selection** section appears.
7. If you want a particular value to be selected by default, then select the row containing the necessary value, and then select **Finish**. You can select multiple default prompt values only if the Prompt Data Type is set to Character and you selected the **Allow Multiple Selections** check box in the **Values From Query** section.

   Note: You can select multiple default prompt values only if the Prompt Data Type is set to Character and you selected the **Allow Multiple Selections** check box in the **Values From Query** section..

   The prompt appears in the **Expression Builder** window.
8. Select **Done**.
   The **Expression Builder** window closes, and then, in the grid in the **Conditions** section, <expr> appears in the field on which you created the prompt.

Results

- When the user runs the query, the prompt will display a list of values pulled from the selected field for all records in the selected family.

Create a Prompt on a Logical Field

Procedure

1. Access the **Prompt Settings** section of the **Expression Builder** window.
2. Define the prompt settings as needed.
3. In the **Prompt Data Type** section, select **Logical**.
   All the options in the **Valid Values** section are disabled.
4. Select **Next**.
   The **Configure Default Prompt Value** section appears.
5. In the **Default Prompt Value** list, select the value that you want to be selected by default. You can select True or False, or you can select All, which returns both True and False values. Then, select **Finish**.
   The prompt appears in the **Expression Builder** window.
6. Select **Done**.

The **Expression Builder** window closes, and then, in the grid in the **Conditions** section, <expr> appears in the field on which you created the prompt.

Results

When a user runs the query, the prompt displays the values **All**, **True**, and **False**.

Filter Prompt Values Based on Previous Prompt Selections

Before You Begin

- These instructions assume that you have already created a Select query for a family that contains at least two fields, and that you have added those fields to your query.
- The first prompt should exist in a column that appears to the left of the column on which you will build the second prompt. If you do not order the fields this way, the values in the first prompt will still filter the values in the second prompt, but the second prompt will appear first when you run the query, which does not follow the appropriate workflow.

About This Task

You can configure multiple prompts where the value in one prompt filters the values available in another prompt, which would further refine the records that are returned.

Procedure

1. Access the **Prompt Settings** section of the **Expression Builder** window for the field on which you want to build the first prompt.
2. Define the prompt settings as needed for the first prompt.
3. In the **Prompt ID** box, specify a unique ID for the prompt. You will use this prompt ID in the second prompt.
4. In the **Valid Values** list, select one of the following prompt types: No list of valid values; Static list of valid values; List of query results; List of values from a record. Note that, if you select No list of valid values, when you run the query, you must enter the exact value stored in the database to populate the dependent prompt with the appropriate values.
5. Depending on the type of prompt you selected, finish defining the prompt settings, and then select **Finish** in the **Prompt Settings** section. Then, select **Done** on the **Expression Builder** window.
   The **Expression Builder** window closes, and then, in the grid in the **Conditions** section, <expr> appears in the field on which you created the prompt.
6. Access the **Prompt Settings** section of the **Expression Builder** window from the **Criteria** cell for the field on which you want to build the second prompt.
7. Define the prompt settings as needed for the second prompt.
8. In the **Valid Values** list, select one of the following prompt types: List of query results; List of values from a record.
9. Depending on the type of prompt that you selected, finish defining the prompt settings, and then select **Finish**.

   The **Expression Builder** window returns to focus. The following instructions assume that you created the second prompt with a list of values from a record.
10. On the **Expression Builder** window, in the text box, directly before the ORDER BY clause, enter a WHERE clause to indicate the field on which the prompt is based and the prompt ID associated with that field. The WHERE clause should look like this: `WHERE [<FieldID 1>] = (? :s :id=<PromptID 1>)`. In this example, `<FieldID 1>` represents the field

ID of the field on which you built the first prompt, and `<PromptID 1>` represents the prompt ID that you defined for the first prompt in Step 3.

11. Select **Done**.

Results

When you run the query, the first prompt will display a list of values according to how you specified the valid values (e.g., static list of valid values). The second prompt will display only values that apply to the first prompt's value.

Modify an Existing Prompt

Before You Begin

- You cannot modify a prompt using the **Prompt Settings** section. The Prompt Settings interface does not store settings for existing prompts.
- If you want to use the **Prompt Settings** section to modify a prompt, you can recreate the prompt and include your modifications in the **Prompt Settings** section.

Procedure

1. Access the **SQL** workspace, or Access the **Expression Builder** window.
2. Modify the prompt as needed.

Delete a Prompt

Procedure

1. Access the **Design** workspace.
2. In the grid in the **Conditions** section, for the cell in which you want to delete a prompt, select ✕.
The prompt is removed from the query.

Chapter

5

Query Settings

Topics:

Access the Query Settings Page

About This Task

From the Query Settings page, you can set the query timeouts, purge the saved exports, or modify the filtering options in result grid. To access the Query Settings page, complete the following steps:

Procedure

In the **Applications** menu, navigate to **ADMIN** > **Operations Manager** > **Query Settings**. The **Query Settings** page appears.

Operations Manager

Query Settings

Query Timeout Limit (in seconds) for Restricted Users

0

Day of week to run export purge

Sunday

Number of days to keep query exports

90

☐ Filter Result Grid Case Insensitive

About Query Timeouts

Using Query Timeouts, you can specify the query time-out limit. The query time-out limit is the amount of time in seconds that the APM system will allow a new or modified query to attempt to return results before timing out.

This setting allows you to control the performance of the queries in your system by enforcing a requirement that they meet a specific performance goal. This setting is used in the query design when a Security User whose query privilege setting is **Restricted By Timeout Limit** tries to save a new or modified query.

Before a Security User whose query privilege is **Restricted By Timeout Limit** can save a new or modified query, they will have to run the query so that the APM system can determine if it runs within the time-out limit. Otherwise, the save options will remain disabled. If a query time-out limit has been specified in that database, when the query runs, the APM system will allow the query to run until the specified query time-out limit has been met. After the time-out limit is reached:

- If the query has not returned results, a message will appear, indicating that the query cannot be saved, and the save options will remain disabled.
- If the query has returned results, the save options will be enabled, and the Security User can save the query.

Specify the Limit for Query Timeout

About This Task

The following instructions provide details on specifying the query time-out limit for Security Users who are restricted by query time-outs. These instructions do not provide a recommendation on choosing the time-out value itself. The value that you choose should match the amount of time that your organization has defined as a reasonable amount of time for a query to run before returning results. The value you enter must be a whole number (i.e., not a decimal value).

Procedure

1. Access the **Query Timeout Limits** page.
2. In the **Query Timeout Limit (in minutes) for Restricted Users** box, enter the number that represents amount of time to which you want to limit query runs.
3. Select 💾.
   A confirmation message appears, indicating that the query timeout is created.

About Purging Saved Exports

There is a scheduled job to purge old background exported queries. You can configure the day this job runs and for how many days to keep the query exports on the **Query Settings** page.

Set Purge Export Frequency

Procedure

1. In the **Applications** menu, navigate to **ADMIN** > **Operations Manager** > **Query Settings**.
   The **Query Settings** page appears.
2. In the **Day of week to run export purge** box, enter the day you want to run the scheduled job. It can be set to any day of the week and runs at midnight UTC on that day. By default, it is set to Sunday.
3. In the **Number of days to keep query exports** box, enter the number of days that you want to keep the query exports. Keeping this number low uses less database disk space and keep the scheduled job from consuming too many resources. By default, it is set to 90 days.

About Case Insensitive Filtering

Filtering in result grid is case sensitive. Case-insensitive filtering can be enabled using the **Filter Result Grid Case Insensitive** setting. This feature only affects the filtering in result grid.

Set Case Insensitive Filtering

Procedure

1. In the **Applications** menu, navigate to **ADMIN** > **Operations Manager** > **Query Settings**.
   The **Query Settings** page appears.
2. Select the **Filter Result Grid Case Insensitive** check box.

Chapter
6

Workflow

Topics:

Core Analysis: Query Analysis Workflow

Core Analysis processes leverage APM data to identify opportunities for business improvement. When the process identifies such an opportunity, a Performance Recommendation is raised to communicate the need and track the required work. You can use the Query Analysis Workflow process to identify opportunities or needs for improvement based on generated queries that identify bad actors or high-impact failures.

In the following workflow diagram, the blue text in a shape indicates that the corresponding description has been provided in the sections that follow the diagram. For more information, refer to the Interpreting the Workflow Diagrams topic in the APM Product Workflows documentation.

Note: For information on the personas associated with a APM module, refer to the APM Product Workflows documentation.
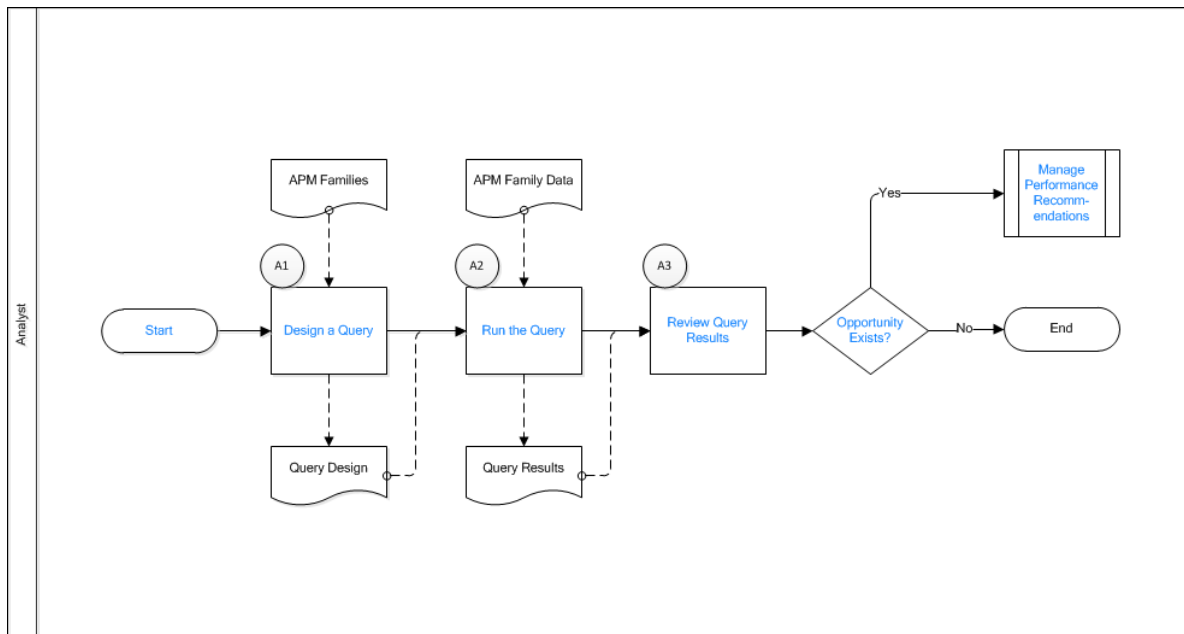
1. Start on page 44
2. Design a Query on page 45
3. Run the Query on page 45
4. Review Query Results on page 45
5. Opportunity Exists? on page 45
6. Manage Performance Recommendations on page 45

Start

Persona: Analyst

To satisfy a specific business problem, a APM User initiates a core analysis that applies standard data analysis techniques.

Design a Query

Persona: Analyst

To satisfy a specific business need, design a query to select specific data from APM entities. Multiple entities can be included in the query design to extract related records.

Run the Query

Persona: Analyst

Execute the query to produce results. The user modifies the query design to achieve the desired result.

Review Query Results

Persona: Analyst

Analyze the query results. The query can identify bad actors or high impact failures for further analysis. In the analysis of query records, consult other forms of APM data.

Opportunity Exists?

Persona: Analyst

If a APM User identifies an opportunity or need for improvement, then a Performance Recommendation is raised to communicate the need and track the required work. Otherwise, the workflow ends.

Manage Performance Recommendations

Persona: Analyst

If a APM User identifies an opportunity or need for improvement, then a Performance Recommendation is raised to communicate the need and track the required work.

For more information, please consult the Manage Performance Recommendations documentation.

Queries Workflow

This workflow provides the basic, high-level steps for using this module. The steps in this workflow do not reference every possible procedure.

Procedure

1. Access the **Design** workspace.

   Note: Interaction with diagramming and design canvases is not available on touch-screen devices.

2. Add at least one query source.
3. Add at least one field from that query source.
4. Run the query to make sure it returns the expected results.
5. Save the query.

The process of creating a more complicated query might include the following additional steps:

6. Add criteria using expressions.
7. Add prompts.
8. Add hyperlinks.
9. Define the results.

Next Steps

After you have initiated the process of creating a Select query using either of these options, when the **Design** workspace appears, you can modify the query type to create any of the following types of queries based on that Select query:

- Crosstab query
- Delete query
- Update query
- Append query

Chapter

7

Reference

Topics:

Reference Information: Query Types

About Select Queries

A Select query returns a list of records that belong to one or more specified families and match your query criteria.

The results of a Select query can provide you with a comprehensive or custom view of the data that exists in the families in your database, allowing you to see only the records and fields that you need to see. Creating a Select query is also often the first step in a process that involves using the query results to perform a certain task. For example, you might create a Select query and then use those results to build a report or create a graph.

About Crosstab Queries

A Crosstab query lets you group data into categories, where a category is determined by a value that exists in multiple fields across multiple families in the database.

In the results of a Select query, each field appears in a column, providing a simple list of data. The results of a Crosstab query appear in a grid. In other words, a Crosstab query presents the same information as a Select query, but in a different format. The format that you choose will depend on the type of information that is returned by the query, and how you want to view it.

Example: Select vs. Crosstab Query Results

Suppose you have a family called Pumps, which stores data on Pump Location, Pump Manufacturer, and Pump Failures. If you queried the family and included the location, manufacturer, and failures fields:

- A Select query would display the results as shown in the following table.

Pump Location	Pump Manufacturer	Pump Failures
Zone 1	ACME	3
Zone 1	SUPER	5
Zone 2	ACME	4

...where each field appears as a separate column of information.
- A Crosstab query would display the results as shown in the following table.

Zone	ACME	SUPER
Zone 1	3	5
Zone 2	4	NULL

...where locations appear as rows, and manufacturers appear as columns.

In this example, you can see that ACME is the manufacturer of multiple pumps. The manufacturer, therefore, represents the category by which you

---

want to display the remaining data (pump location). Each column in the results grid represents a separate value within the same category. So, in this example, ACME and SUPER are different types of manufacturers within the manufacturer category.

Example: Crosstab Query

When management personnel request that work be performed on a piece of equipment, the work results in some amount of downtime for the piece of equipment. Some work activities result in longer amounts of downtime than others. In your company, work requests are recorded in Work Request records, which contain the following fields:

- **Work Request ID** Identifies the work request with a unique value.
- **Work Activity** Indicates the type of work that should be performed (e.g., repair).
- **Downtime** Indicates the total amount of time that the equipment was out of service while work was being performed on it.

If you were to create a Select query to view information about work requests that have been completed, the results might look something like those shown in the following image.

WORK REQUEST ID	WORK ACTIVITY	DOWNTIME
WR1	Repair	1
WR1	Adjust	2
WR1	Clean	1

In these results, you can see each work request ID, the corresponding work activity, and the total amount of downtime per request. The format of these results, however, does not display the work requests grouped by activity type. While this result set is small, which allows you to visually determine how many work requests fall into each activity type, more typical query results will contain enough rows of data that it will be difficult to divide it into categories by visually comparing the data. This is especially true when the results span multiple pages.

To group the results such that you can see at a glance how many work requests fall into each activity type, you could decide to make the query an aggregate query. If you use the COUNT function on the Work Request ID field and the SUM function on the Downtime field, the query results would look similar to those shown in the following image.

WORK REQUEST ID	WORK ACTIVITY	DOWNTIME
1	Adjust	2
1	Clean	1
1	Repair	1

In these results, you can see that three work requests asked for an adjustment, three requests asked for something to be cleaned, and four work requests asked for a repair. You can also see that the total amount of downtime for all adjustments was four days, the total amount of downtime for all cleaning tasks was six days, and the total amount of downtime for all repairs was 20 hours.

While the stored data is interesting when viewed in this format, you might be more interested in determining which work requests resulted in a downtime over a certain number of days. For instance, suppose that you expect repairs to take over seven days, but cleaning tasks that take more than seven days are unacceptable to management personnel. You might want to construct the query such that it groups the raw data into two categories: downtime and work activity. You then want to determine how many work requests in each type of activity resulted in downtime between one and seven days, and how many resulted in downtime over seven days.

The results of a Select query cannot present the data in this format. To format the data such that it provides the desired information, you must create a Crosstab query, where you can:

- Convert the stored downtime values into categories: 1 to 7 Days and Over 7 Days.
- Determine the total amount of downtime per work activity.
- Divide the total amount of downtime per work activity into the predefined categories of 1 to 7 Days and Over 7 Days.

The Crosstab query will contain the same fields as the Select query: Work Request ID, Work Activity, and Downtime.

To convert the stored downtime values into categories, however, you will need to add another column that includes a DECODE statement that uses the SIGN function.

The DECODE statement would look like this:

```
                    Decode(SIGN(([Work Request].
[Downtime] - 7)), -1, '1-7 Days', 0, '1-7 Days', 'Over 7
Days')
```

This statement indicates that:

- First, the value seven should be subtracted from the actual downtime values. Because the Downtime field has a unit of measure of Days, and there are seven days in a week, each downtime value will be greater than or equal to zero and less than or equal to seven. After subtracting the value seven from these stored downtime values, the calculated result will be either a negative number, zero, or a positive number.
- After subtracting seven from the actual downtime value:

  ○ If the value is negative or zero, the record should be grouped into the category 1 to 7 Days. This means that any work request with the following downtime values will be grouped into the 1 to 7 Days category: 0, 1, 2, 3, 4, 5, 6.
  ○ If the value is anything other than zero or a negative number, the record should be grouped into the category Over 7 Days. This means that any work request with a downtime value greater than or equal to eight will be grouped into the category Over 7 Days.

In the grid in the **Conditions** section, the Work Activity field will be the row heading, and the column with the DECODE statement will be the column heading. The Work Request ID field will be the intersecting field, or the **Value**, and a COUNT function will be defined in the **Total** cell for the Work

Request ID field. This means that the intersecting cell in the results will contain a number instead of a Work Request ID. The number will indicate the number of work requests that fall into the category defined by the intersection of the row and the column (e.g., the number of repair work requests that resulted in a downtime of over seven days).

In addition, the SUM function will be defined in the **Total** cell for the Downtime field. This will ensure that the results contain only one row representing each work activity instead of multiple rows containing the same work activity. For example, if there are three repair requests, because the SUM function is defined on the Downtime field, the results will contain only one row representing the repair work type (displaying the total number of work requests of that type) instead of three rows representing the repair work type (displaying only one work request of that type for each row).

The results will be grouped as shown in the following table.

Work Activity	Downtime Category (1 to 7 Days)	Downtime Category (Over 7 Days)
Work Activity (Repair)	# of Repairs with a Downtime of 1 to 7 Days	# of Repairs with a Downtime Over 7 Days
Work Activity (Clean)	# of Cleaning Tasks with a Downtime of 1 to 7 Days	# of Cleaning Tasks with a Downtime Over 7 Days
Work Activity (Adjust)	# of Adjustments with a Downtime of 1 to 7 Days	# of Adjustments with a Downtime Over 7 Days

About Update Queries

An Update query modifies the records that match the criteria that you have specified in the query. An Update query makes global changes to a group of records belonging to one or more families.

An Update query is similar to a Select query in that it retrieves records from the database that match the criteria defined within the query. The difference is that instead of displaying the results, the results are modified according to the criteria defined in the query.

For example, if an equipment manufacturer name has changed, you would need to update all the existing records for equipment made by that manufacturer so that the value in the Manufacturer field is the new name of the manufacturer. Instead of manually modifying these values, you could create an Update query to modify all the records for this family.

Only Super Users and members of the MI Power User Role Security Group can create and run the Update queries, provided that they have been granted View permissions to the Catalog folder in which they are stored. To update the records returned by the query, a user must have Update permissions to the family to which those records belong.

Note: The recommended workflow to create an Update query is to first design a Select query and review the results. After you have confirmed that the Select query returns the required records correctly, in the Design workspace, change the query type to Update, and then set the update value. This ensures that the Update query does not alter the records that you did not intend to change.

Note: Modifications made by an Update query cannot be undone automatically. If you need to undo your changes, you will need to create another Update query or modify the records individually.

About Append Queries

An Append query adds a group of records from one or more families to another family.

An Append query lets you:

- Use a predefined Select query to find records in one or more families that you want to add to another family.
- Take the records that are returned and add them to other families, thereby creating new records in those families.

For example, you might create a set of records in one family that are the same or similar to a set of records that you also need to create in another family. You can also consolidate records that currently exist in multiple families into a single family. In either of these cases, you can use an Append query to find the records that currently exist in one family and add them to another family. In cases where you are using an Append query to move records, you might also want to use a Delete query to delete the records from the original family after they have been moved.

In an Append query, you will define field mappings to map fields in the source family to fields in the target family. Only fields that are included in the Select query can be mapped to the target family, so you will want to make sure that the query includes all the columns containing values that you want to use to populate new records. Any fields that are not mapped will not be populated in the new records. The process of defining mappings is facilitated by the **Append To** cell, which appears in the grid in the **Conditions** section when you create an Append query, and which allows you to select the target field to which each source field corresponds.

Only Super Users and members of the MI Power User Role Security Group can create and run the Append queries, provided that they have been granted View permissions to the Catalog folder in which they are stored. To create the records returned by the query, a user must have Insert permissions to the family in which the records are being created.

About Delete Queries

A Delete query deletes records that meet the criteria in the query from the database. Delete queries delete entire records, not just individual fields in records.

If you need to delete many records at once, this type of query can save you time. For example, suppose you created various records in a family and later determined that those records were invalid. You could create a Delete query to delete those records from the family based on the desired criteria, such as a value in the record, the creation date, or the user who created them.

Only Super Users and members of the MI Power User Role Security Group can create and run the Delete queries, provided that they have been granted View permissions to the Catalog folder in which they are stored. To delete the records returned by the query, a user must have Delete permissions on the family to which those records belong.

Note: Deletions performed by a Delete query cannot be undone.

Note: The recommended workflow to create a Delete query is to first design a Select query and review the results. After you have confirmed that the Select query returns the required records correctly, in the Design workspace, change the query type to Delete. This ensures that the Delete query does not delete records that the you did not intend to remove

Reference Information: Query Results

About Displaying Custom Text Instead of Field Values

By default, the values displayed in each column in the results are the values in the fields of records included in the results. You can modify the default behavior, however, if you want all cells in a column to display the same text.

To modify the values displayed in a column, you will need to modify the value in the **Field** cell for the appropriate column in the grid in the **Conditions** section of the **Design** workspace.

Tip: If you modify the value in the **Field** cell for a column that contains Group By in the **Total** cell, an error message will appear when you run the query. If you receive this error, try changing Group By to Min, Max, or Expression.

> **Custom Text in the Installation Date Column**
>
> If your query returns the Asset Installation Date for all Pump records, the values in the Asset Installation Date column will be the installation dates that are stored in the Pump records in your database. You could, instead, make the column display custom text instead of the stored dates. For example, the installation dates might fall within a given time period, such as January 2005 and December 2005, so you might want the text to read **Installed in 2005** to indicate the time frame as opposed to the actual date.

> **Hyperlinks**
>
> If your query results contain hyperlinks, you will probably want to customize the text of your hyperlinks. For example, if your query returns all Pump records, you might include the Asset ID, Asset Description, and Asset Installation Date in the results. By default, the columns will display the value stored in the Asset ID, Asset Description, and Asset Installation fields of each record.
>
> You might decide to add a hyperlink to the Asset ID field that will open each record in the results in the Record Manager. Suppose you want the hyperlinks to display some text other than the Asset ID. For example, you might want the hyperlink to display the text Open in Record Manager.

About Formatted and Unformatted Mode

You can run Select queries in two different modes: formatted and unformatted.

Details

- **Formatted Mode**: Causes the results to display formatted values rather than stored values. In other words, the values displayed in the results will be formatted based on any format rules or criteria defined for the fields included in the query.
- **Unformatted Mode**: Causes the results to display stored values rather than formatted values. In other words, any format rules that have been defined will not be applied, and the results will display values exactly as they exist in the database.

Note: When you run a query in unformatted mode, the results will still display formatted date values. Date values will always be displayed in the local time for the user; however, if you export the unformatted query result set to a dataset, the exported date values will be unformatted.

Formatted Mode

Consider an example where the System Code Table Priority contains the System Codes listed in the following table and is configured to display descriptions only.

ID	Description
1	Very High
2	High
3	Medium
4	Low
5	Very Low

Suppose that your database contains the Task family, which contains a Priority field, and also assume that a Valid Values rule has been applied to this field so that it displays a list of values from the Priority System Code Table. Because this System Code Table is set up to display descriptions only, the available values for this field will be Very High, High, Medium, Low, and Very Low. When a user selects one of these values, the corresponding numeric ID will be stored in the field.

When you create a Select query on the Task family, you can choose to run it in formatted or unformatted mode. If you create a query that includes the Task ID, Task Type, and Priority fields and run it in unformatted mode, the results might look something like the following image.

TASK ID	TASK TYPE	PRIORITY
987453	INTERNAL_VISUAL	1
986574	INTERNAL_VISUAL	4
984537	EXTERNAL_VISUAL	3

Notice in this image that numeric values are displayed in the **Priority** column. These are the System Code ID values that are stored in the Priority fields of these records. If you run the same query in formatted mode, the results would look like the following image.

54

TASK ID	TASK TYPE	PRIORITY
987453	INTERNAL_VISUAL	High
986574	INTERNAL_VISUAL	High
984537	EXTERNAL_VISUAL	Very Low

Notice that the **Priority** column now displays the System Code descriptions that correspond to the stored IDs. In other words, the results now show the formatted values rather than the stored values.

By default, newly created Select queries will run in unformatted mode. You can change the mode and save it with the query so that the next time you run the query, it will use the mode that you last saved. Note that the **Formatted** check box is available only for Select queries.

Note: If you select the **Show Totals** option, the query behaves as if it were running in unformatted mode even if you select to run it in formatted mode.

About Defining an Alias

The alias for a field specifies how it will be labeled in the query results and in any report or graph that is created from the query. When you add fields to the grid in the **Conditions** section, the alias is set automatically to the field caption. You can modify the alias.

Details

When defining aliases, keep in mind the following considerations:

- Within a given query, each alias must be unique. If you specify an alias that has already been defined for another column in the same query, a number will be appended to the end of each duplicate alias to distinguish them from the other aliases in the query. For example, if you specify Asset as the alias in four columns, the aliases would be changed to:
  - Asset
  - Asset0
  - Asset1
  - Asset2

  Each duplicate alias will be numbered in this way using the next available number (i.e., 3, 4, 5, and so on).
- The alias cannot exceed 27 characters in length (including spaces). You will not be able to enter more than 27 characters into any **Alias** cell.

About Displaying Unique Records Only

You can specify that results of a query return only unique records. For example, if you design a query to return all types of Air Cooled Heat Exchangers, you might get 10 results, but there might be only two different types of Air Cooled Heat Exchangers within those results, such as Air Cooled Heat Exchanger and Cooling Tower. If you specify that the query return only records where the asset type is unique, you will see only two results: one Air Cooled Heat Exchanger result and one Cooling Tower result.

Uniqueness is defined at the query level, not the field level. This means that if the same query that returns two results with different asset types is also designed to return the asset status, and you specify that the query return only unique records, you will see records where the

asset type and status are unique. You could see results where there are active Air Cooled Heat Exchangers, Inactive Air Cooled Heat Exchangers, and active Cooling Towers in the database.

About Query Performance

Query performance is influenced by a number of factors, including hardware efficiency (e.g., how fast the Server machines are), the system's workload at the time you run the query (e.g., how many users are logged in and making requests), and the efficiency of the database (e.g., how well the database is being maintained). As you can see, many of the factors that influence query performance are beyond your control. For example, you cannot control how many other users are currently using the system.

Query performance is also affected by a number of factors that you can control, including how you construct the query and the options that you choose for running it. The following Details sections discuss things that you can do to maximize the performance of queries.

Note: The suggestions provided here may not improve the performance of all queries. In addition, other factors (e.g., deficiencies in hardware resources) may negate any improvement that would otherwise be achieved by implementing these suggestions.

Limiting the Size of the Query

Generally, the more data that you attempt to retrieve with a query, the longer the query will take to run. For example, a query that retrieves data for all fields in the Recommendation family (which has many subfamilies) will take longer to run than a query that returns only a few fields from the APM General Recommendation family (A subfamily of the Recommendation family).

Knowing this, you can improve query performance by limiting the amount of data that you return in the query results. For example:

- Always query on the lowest-level family possible. For example, instead of querying on a family, query on individual subfamilies. If you need to query on more than one subfamily, you might want to create more than one query.
- Only include in your query results the fields that you actually need. Instead of returning the information for all fields in a given family, limit your results to only the specific fields that you want to analyze.
- Use criteria to limit results to a specific set of records. For example, if you are interested only in the pumps associated with a given manufacturer, limit your query results by applying criteria to the Manufacturer field.

Note: Performance improvement from limiting the size of the query will be seen only when comparing queries that are otherwise similar.

Running Select Queries in Unformatted Mode

APM provides two output modes for running Select queries: formatted mode and unformatted mode.

Because unformatted mode returns results without taking the extra step of applying formatting before displaying results, running a query in unformatted mode can be more efficient than running the same query in formatted mode. This performance improvement is particularly significant when you run a query that includes many formatted fields.

Note: Unformatted mode is the default output format for new Select queries.

Other Recommendations

- Avoid outer joins because they can impact the performance.
- Avoid using OR. Instead use UNION for better performance.
- If using Like with a leading wildcard character on an indexed field, the index will not be used.
- Avoid unnecessary joins. If no fields are used from a family, do not add the family to the query.
- Avoid Distinct when possible.
- When writing aggregate queries, use Where and Having correctly; Where is used to filter results before calculating the aggregate, whereas Having is used to filter out the results based on the aggregation.

About Limiting the Number of Results

You might want to limit the number of results that are returned for certain queries such that only the first n number of records are returned based on a specific set of criteria.

Note: This feature can be used only with Select queries.

Limit the Number of Results

Suppose you want to view the 20 most expensive equipment items according to total cost, where the value in the Breakdown Indicator field is set to True. In this case, you would create the query on the Equipment and Work History fields, joined via the Has Work History relationship, and add the Equipment ID, Equipment Short Description, and Total Cost fields to the query.

The Total Cost field is included, and the query is configured to show the sum of all total cost values, grouped by Equipment ID. In this case, rather than showing two separate rows in the results for each Work History record that is linked to this Equipment record, the results will contain one row for this Equipment record, and the **Total Cost** column will display the sum of the individual values in the Total Cost field in each Work History record. Additionally, the query is configured to show the results in descending order according to total cost.

Based on this query configuration, if you were to run the query at this point, you would see all Equipment records that meet the query criteria.

In this scenario, however, you want to view only the 20 most expensive pieces of equipment. To refine this query to suit your needs, in the **Conditions** section heading, set the limit to 20. When you run the query again, only those 20 Equipment records will be returned by the query.

SQL Servers and Duplicate Values

If you are using a SQL Server database, sort the query by a field, and then configure the query to return a limited number of records, if more than one record contains the same value in the field by which you sorted the query, those records will be displayed in a random order relative to one another. For instance, if two Equipment records are linked to Work History records with the same total cost, each time you ran the query, those Equipment records would be displayed in a random order relative to one another. In other words, the first time you ran the query, the Equipment record with the

---

ID 000000000001060839 might appear above the Equipment record with the ID 000000000001060840. The second time you ran the query, however, those Equipment records might appear in the reverse order. This limitation applies to SQL Server databases only.

Query URLs

There are two URL routes associated with queries: **query** and **qdetail**. The following table describes the various paths that build on the route, and the elements that you can specify for each.

Element	Description	Accepted Value(s)	Syntax
query : Displays the **Query** page.			
query/<EntityKey>/<WorkspaceName> : Displays the **<QueryName>** or **New Query** page.			
Catalog Item Key	Specifies the Catalog Item Key of the query that you want to open in the **Query** tool.	Numeric Catalog item key	#query/**Catalog Item Key**
Catalog Item Path	Specifies the path and name of the query that you want to open in the **Query** tool.	Catalog item path	#query?path=**Catalog Path \Query name**
p0, p1, p2 etc. (specifying a literal value)	Specifies a literal value that will be passed into a query containing a prompt.	Any value that is acceptable for the prompt type (e.g., numeric values for numeric prompts)	#query/**Catalog Item Key? p0=Literal Value** #query?path=**Catalog Path \Query name&p0=Literal Value**
p0, p1, p2 etc. (specifying a variable value)	Specifies a variable value from a specified column key in a query that will be passed from a query into a query containing a prompt.	Any value that is acceptable for the prompt type (e.g., numeric values for numeric prompts)	#query/**Catalog Item Key? p0={Column Key}** #query?path=**Catalog Path \Query name&p0={Column Key}**
qdetail/<Catalog Item Key>: Displays the query in a new, view-only page.			
Parameter Name = **Parameter Value**	Specifies the Parameter Name and Parameter Value of the query whose results you want to open in a new, view-only page.	Parameter Names and Parameter Values	#qdetail/**Catalog Item Key?Parameter Name**=**Parameter Value**

Examples: Query URLs

Example URL	Destination
#query/**3223198**	Opens the query with the Catalog Item Key of 3223198 in the **Query** tool.
#query?path=**Public \Meridium\Modules\Core \Queries\APM Query**	Opens the query named 'APM Query' in the **Query** tool that is found in the specified Catalog folder.
#query/**3223198?p0=Literal Value**	Opens the query with the Catalog Item Key of 3223198 in the **Query** tool and passes the specified literal value into the first prompt in the query.
#query?path=**Public \Meridium\Modules\Core \Queries\APM Query&p0=Literal Value**	Opens the query named 'APM Query' in the **Query** tool that is found in the specified Catalog folder and passes the specified literal value into the first prompt in the query.
#query/**3223198?p0={1}**	Opens the query with the Catalog Item Key of 3223198 in the **Query** tool and passes the specified variable from an existing query into the first prompt of the query.
#query?path=**Public \Meridium\Modules\Core \Queries\APM Query&p0={1}**	Opens the query named 'APM Query' in the **Query** tool that is found in the specified Catalog folder and passes the specified variable value from an existing query into the first prompt of the query.
#qdetail/**900000003707? Manufacturer=UNITED %2BPUMPS**	Opens the results of the query with the Catalog Item Key 900000003707, Parameter Name Manufacturer, and Parameter Value UNITED %2BPUMPS in a new, view-only page.

Reference Information: Query Joins, Functions, and Hyperlinks

About APM Inner Joins

When you use two related entity families as query sources, APM creates an inner join between the two entity families by default. This inner join causes the query results to include only the records that are linked through the specified relationship.

If two entity families are not related through a relationship family, you can manually create an inner join to connect one or more fields in one family to one or more fields in the other family.

The following table shows the line style that appears in the design canvas to represent APM inner joins.

Join Type	Line Style
APM Inner Join	————————————

Example: Inner Join

Suppose you add the Equipment entity family, Work History entity family, and Has Work History relationship family as query sources. Assuming that the Has Work History family relates the Equipment family to the Work History family, APM will create an inner join between the Equipment and Work History families by default. As a result, the query results will return only Equipment records that are linked to a Work History record through the Has Work History relationship. The query results will not include any Equipment records that are not linked to a Work History record or any Work History records that are not linked to an Equipment record.

About APM Outer Joins

An outer join allows you to return records that satisfy the join conditions and records from one family for which there are no matching records in the other family.

You can modify the default inner join to manually create an outer join for families that are related through a relationship family.

The following table shows the line styles that appear in the design canvas to represent APM outer joins.

Join Type	Line Style
APM Outer Left Join	◄ - - - - - - - - - - - - - -
APM Outer Right Join	- - - - - - - - - - - - - - ►

Example 1: Creating a APM Outer Join

If you use two unrelated families as query sources, such as the Centrifugal Pump family and the Rotary Pump family, you can create an outer join to view all Centrifugal Pumps and Rotating Pumps whose manufacturer is the same and all Centrifugal Pumps with a different manufacturer. If the Centrifugal Pump family was added as a query source as Table #1 and the

Rotary Pump family was added as Table #2, this would create a left join, which would be indicated in the SQL code.

On the other hand, if you wanted to view all Rotary Pumps and Centrifugal Pumps whose manufacturer is the same and all Rotary Pumps with a different manufacturer, you would create a right join, which would be indicated in the SQL code.

Example 2: Creating a APM Outer Join

You can modify a relationship-driven inner join to create an outer join. For example, suppose that you use the Centrifugal Pump entity family, the Failure entity family, and the Asset Has Failure relationship family as query sources. If your system is configured such that the Asset Has Failure family relates the Centrifugal Pump family to the Failure family, APM would create an inner join between the two entity families automatically.

If you ran the query using the default join, the results would include all Centrifugal Pump records that are linked to a Failure record. If you wanted to modify those results, however, to view all Centrifugal Pumps with their linked Failure records and all Centrifugal Pumps that did not have linked Failure records, you would need to modify the inner join to create an outer join. This would return Centrifugal Pumps with and without linked Failure records, but not Failure records without linked Centrifugal Pump records. For example, you could configure a query to show all the Centrifugal Pump records, where only three are linked to Failure records.

This type of join is considered a left join, as indicated by the following SQL code:

```
                    SELECT [Centrifugal Pump].
 [ASSET_ID_CHR] "Asset ID", [Failure].[EFAIL_ASSETID_CHR]
 "Failure ID"

                         FROM [Centrifugal Pump]RIGHT
 JOIN SUCC [Failure] ON {Asset Has Failure}
```

You could also decide to return all Failure records with their linked Centrifugal Pump records and all Failure records that do not have linked Centrifugal Pump records.

This type of join is considered a right join, as indicated by the following SQL code:

```
                    SELECT [Centrifugal Pump].
 [ASSET_ID_CHR] "Asset ID", [Failure].[EFAIL_ASSETID_CHR]
 "Failure ID"

                         FROM [Centrifugal Pump]RIGHT
 JOIN SUCC [Failure] ON {Asset Has Failure}
```

About Manual Joins

In many cases, you will want to query the database for information that exists in more than one family. When you do so, you may also want to join a field in a family with a similar field in another family, which will create an ad hoc association between the two families. By joining a field in one family to a field in another family, you create a query join.

Details

There are two types of manual joins in APM:

- **Manual Inner Join:** Returns a row for every record where the values in the joined fields in both families are equal. An inner join returns only those records that satisfy the join conditions, so any unmatched records are dropped from the result set.
- **Manual Outer Join:** Returns a row for every record in one family, and a row for every record where the values in the joined fields in both families are equal. An outer join can be either a left outer join or a right outer join. When you add query sources to the grid in the **Conditions** section and join their fields, the family that you added first is referred to as Table #1, and the family that you added second is referred to as Table #2.
    - A left outer join returns every record in Table #1, regardless of whether each record is linked to a record from Table #2. The results also include the records in Table #2 in which the value in the joined field is equal to that in Table #1.
    - A right outer join returns every record in Table #2, regardless of whether each record is linked to a record in Table #1. The results also include the records in Table #1 in which the value in the joined field is equal to that in Table #2.

The following table shows the line styles that appear in the design canvas to represent manual joins.

Join Type	Line Style
Manual Inner Join	
Manual Outer Left Join	
Manual Outer Right Join	

Example: Manual Inner Join

Suppose that you notice that two pieces of equipment, such as a centrifugal pump and an air cooled heat exchanger, have been malfunctioning recently. You suspect that the failures could be caused by human error during installation. You decide to query the database to find instances where these pieces of equipment were installed on the same day by the same person.

To do so, you would need to add the Centrifugal Pump family and the Air Cooled Heat Exchanger family as query sources. You would then need to join the Asset Installation Date fields and the Responsible Installer fields between the two families via an inner join. Doing so would return only records where a centrifugal pump was installed by the same person on the same date as an air cooled heat exchanger.

What is a Function?

A function is a SQL component that manipulates data and returns a value that is not stored in the database, but is derived from calculating or reformatting values. Functions can be used to calculate or reformat values:

- Stored in the database.
- Based on static data (e.g., the current date).

In SQL code, a function can be included as part of the SELECT statement, WHERE clause, the GROUP BY clause, or the HAVING clause, or it can exist outside of these SQL components. You can write functions in the SQL code, or in the Expression Builder.

You can use the following functions in APM:

- Aggregate Functions
- Character Functions
- Conversion Functions
- Number/Mathematical Functions
- The DECODE Function
- Date Functions

Tip: For more information on functions, refer to MetaSQL Functions on page 80.

Note: Throughout this documentation, functions are grouped into categories according to how they are grouped in the Expression Builder.

Example 1: SUM Function Contained within the SELECT Statement

In the following SQL code, the SUM function is displayed in bold text.

```
SELECT [Asset].[ASSET_ID_CHR] "Asset ID", Sum([Failure].
[EFAIL_TOTCST_FROM]) "Total Failure Cost"
 FROM [Asset] JOIN SUCC [Failure] ON {Asset Has Failure}
 WHERE [Failure].[EFAIL TOTCST FRM] > 50000
GROUP BY [Asset].[ASSET ID CHR]
```

In this example, the SUM function is contained within the SELECT statement.

Example 2: SUM Function Contained within the SELECT Statement and in the HAVING Clause

In the following SQL code, the SUM function is displayed in bold text and appears twice: once in the SELECT statement and once in the HAVING clause.

```
SELECT [Asset].[ASSET_ID_CHR] "Asset ID", Sum([Failure].
[EFAIL_TOTCST_FROM]) "Total Failure Cost"
FROM [Asset] JOIN SUCC [Failure] ON {Asset Has Failure}
GROUP BY [Asset].[ASSET ID CHR]
HAVING Sum([Failure].[EFAIL TOTCST FROM]) > 5000
```

About the GROUP BY Clause

A GROUP BY clause is used to group query results.

The GROUP BY Clause

Suppose that you want to see the total number of failures your equipment has experienced per equipment manufacturer.

You would add the Asset Manufacturer field and the Failure ID field to the grid in the **Conditions** section. The **Total** row indicates that you want to group the results by manufacturer and display a total count of failures for each manufacturer that is returned. The sort preference indicates that you want to sort the results in descending order according to the failure count.

In the results, each row represents a different manufacturer and the total count of failures for each manufacturer that is returned.

In this example, the query results are grouped by one field only, so each manufacturer appears only one time. You can, however, group query results by more than one field. When you group a query by multiple rows, the query determines all possible combinations of results and returns each distinct combination. Therefore, the more fields you group by, the more results you will see.

Continuing with this example, if you add the Asset ID field to the query, the results will contain many more rows because there are more combinations to display. In this case, one manufacturer may be displayed twice if that manufacturer manufactures multiple equipment items.

About Aggregate and Analytic Functions

Aggregate functions perform a calculation on a set of values and return a single value. Analytic functions compute an aggregate value based on a set of values, and, unlike aggregate functions, can return multiple rows for each set of values. Throughout this documentation, we refer to queries that contain aggregate functions as aggregate queries, and queries that contain analytic functions as analytic queries.

If you want to run a query using only aggregate functions, the query will return one row with a column for each field. If you want to run a query using an aggregate function in conjunction with the GROUP BY clause, the query will return one row for each value found in the grouped field.

Aggregate functions are used in conjunction with the GROUP BY clause, which specifies how the query results will be grouped and displayed. In other words, if you use any aggregate functions on fields in a query, then all remaining fields must appear in the GROUP BY clause.

Aggregate functions and analytic functions can be selected in the **Total** cell of the grid in the **Conditions** section.

The SUM and AVERAGE functions can be used only on numeric query fields. The MIN, MAX, and COUNT functions can be used on date, numeric, or character fields.

Tip: For more information on functions, refer to MetaSQL Functions on page 80.

64

Analytic functions compute an aggregate value based on a set of values, and, unlike aggregate functions, can return multiple rows for each set of values. Use analytic functions to compute moving averages, running totals, percentages, or top-N results within a group.

If you want to run a query using an analytic function, the query will return one row for each field in the defined range of fields used to perform the calculations.

Note: In the following table, items in the special font (that is, `field`, `field1`) represent user-supplied parameters. Items contained within brackets are optional. Do not enter the brackets themselves.

Analytic Function	Format of Code Using the Function
AVG	AVG(`field`) OVER([PARTITION BY `field1[, field2, ...]]` ORDER BY `field1[, field2, ...])`
CUME_DIST	CUME_DIST() OVER([PARTITION BY `field1[, field2, ...]]` ORDER BY`field1[, field2, ...])`
FIRST_VALUE	FIRST_VALUE(`field`) OVER([PARTITION BY `field1[,field2, ...]]` ORDER BY `field1[, field2, ...])`
LAST_VALUE	LAST_VALUE(`field`) OVER([PARTITION BY `field1[,field2, ...]]` ORDER BY `field1[, field2, ...])`
MAX	MAX(`field`) OVER([PARTITION BY `field1[,field2, ...]]` ORDER BY `field1[, field2, ...])`
MIN	MIN(`field`) OVER([PARTITION BY `field1[, field2, ...]]` ORDER BY `field1[, field2, ...])`
NTILE	NTILE(`number`) OVER([PARTITION BY `field1[,field2, ...]]` ORDER BY `field1[, field2, ...])`
ROW	ROW_NUMBER() OVER([PARTITION BY`field1[, field2, ...]]` ORDER BY `field1[, field2, ...])`
SUM	SUM(`field`) OVER([PARTITION BY `field1[,field2, ...]]` ORDER BY `field1[,field2, ...])`

About Character Functions

Character functions are used to manipulate values returned on character fields.

Tip: For more information on functions, refer to MetaSQL Functions on page 80.

Example: REPLACE Function

Suppose that you want to view the Failure ID of the failure associated with each piece of equipment or location in your database. Failure IDs are stored with a dash in the syntax, and you want to display them with a double colon instead of the dash.

In this case, you might configure a query on the Asset and Failure families, joined via the Asset Has Failure relationship, and add the Asset ID and Failure ID fields to the query.

In this example, to return the Failure IDs and replace the dash with a double colon, you would configure an expression using the REPLACE function. You would also configure the alias to ensure that the column displays the text Failure ID.

In this case, the expression syntax is:

```
REPLACE([Failure].[Failure ID], '-', '::')
```

The syntax indicates that you want to replace the dash (-) with a double colon (::).

More Examples of Character Functions

The following table lists more examples of using Character functions:

Function	Description	Example	Stored Value	Result
CONCAT	Concatenates two field values and displays the result.	**CONCAT** ([Asset)ID], [Asset Type])	Asset ID: T-101 Asset Type: Tank	T-101Tank
&	Concatenates two or more field values and adds delimiters between the values.	[Asset ID] &':' & [Asset Type]	Asset ID: T-101 Asset Type: Tank	T-101:Tank
LOWER	Displays the value in all lowercase letters.	**LOWER** ([Asset Type])	Tank	Tank
UPPER	Displays the value in all uppercase letters.	**UPPER** ([Asset Type])	Tank	Tank
SUBSTR	Displays a specific number of characters depending on the starting point you specify and the number of characters that you specify should be returned.	**SUBSTR** ([Asset ID],0,3) Zero (0) specifies the starting point (from left to right) and three (3) specifies the number of characters after the starting point that you want to display.	PMP-101	PMP
IsNull	Displays a specified value when a null value is found.	**IsNull** ([Asset Type],'No Value Listed')	Asset Type field contains a null value	No Value Listed

Function	Description	Example	Stored Value	Result
LTRIM	Displays the value with the specified characters removed from the beginning (left) of the string.	**LTRIM** ([PhoneNumber], '(540)-')	(540) 344-9205	344-9205
RTRIM	Displays the value with the specified characters removed from the end (right) of the string.	**RTRIM** ([Failure ID], '-0123456789')	FAIL-1234	FAIL
STRING_AGG	Concatenates the string values and places separator values between them. The separator is not added at the end of string. If this function is used with *group by*, it returns one row per grouping, else returns one row. You can also control the order in which the values are concatenated. The SQL server has a 8,000 character limit. Oracle and Postgres return CLOB and TEXT respectively.	SELECT String_Agg([MI_EQUIP000]. [MI_EQUIP000_EQUIP_ID_C], ';') "Equipment" FROM [MI_FNCLOC00] JOIN_SUCC [MI_EQUIP000] ON {MIR_FLHSEQ} GROUP BY [MI_FNCLOC00]. [MI_FNCLOC00_FNC_LOC_C]	0000000000010567 81 0000000000100000 68	0000000000010567 81,00000000001000 0068
JSON_VALUE	Gets a single value from a JSON string. Always returns as a character.	JSON_VALUE([FAMILY].[JSON field], ` $.property')	{"property": 1}	"1"

About Conversion Functions

A Conversion function is used to modify query results by reformatting the data. You can use this type of function if you want to reformat the data to simplify it (e.g., you could remove unnecessary zeros from dates, where 01/07/2007 could be converted to 1/7/2007) or if you want to reformat the data so it appears as a different data type completely (e.g., you could spell out a month instead of representing the month with a number, where 01/07/2007 could be converted to January 7, 2007).

Description	Function
Convert a number or date to a character	CastChar
Convert a character to a date	CastDate
Convert a character to a number	CastNum

About Date Functions

Date functions are used to manipulate values returned on date fields.

Tip: For more information on functions, refer to MetaSQL Functions on page 80.

Example: MI_DateAdd

Suppose that all air cooled heat exchangers in your facility require inspection every six months. You might want to run a query to see the last inspection date of all air cooled heat exchangers and the date that is six months after that date. If each Air Cooled Heat Exchanger record in your database contains an Asset Inspection Date value, you could do so using the MI DateAdd function.

Note: Oracle databases do not support date math by year ('yy') or month ('mm') with time stamps enabled; therefore, the DateAdd function for Oracle databases will use 365.25 days for years and 30 days for months.

You might configure a query on the Air Cooled Heat Exchanger family, and add the Asset ID, Asset Description, Asset Inspection Date, and Next Inspection Date fields to the query.

In this example, to return the next inspection date, you would configure an expression using the MI DateAdd function. You would also configure the alias of this column to indicate that the column returns the next inspection date.

The expression syntax is:

```
MI DateAdd('mm', 6, [Air Cooled Heat Exchanger].[Asset Inspection
Date])
```

The following table lists more examples of how you can use the MI DateAdd function.

Parameter	Example	Description	Stored Value	Result
yy	MI DateAdd(' **yy** ', **1**, [Air Cooled Heat Exchanger].[Asset Installation Date])	Adds one year to the date on the Asset Installation Date field in Air Cooled Heat Exchanger records.	11/01/2009	11/01/2010
mm	MI DateAdd(' **mm** ', **6**, [Air Cooled Heat Exchanger].[Asset Installation Date])	Add six months to the date on the Asset Installation Date field in Air Cooled Heat Exchanger records.	11/01/2009	05/01/2010

Parameter	Example	Description	Stored Value	Result
dd	MI DateAdd(' **dd** ', **4**, [Air Cooled Heat Exchanger].[Asset Installation Date])	Adds four days to the date on the Asset Installation Date field in Air Cooled Heat Exchanger records.	11/01/2009	11/05/2009
hh	MI DateAdd(' **hh** ', **4**, [Air Cooled Heat Exchanger].[Asset Installation Date])	Adds four hours to the date on the Asset Installation Date field in Air Cooled Heat Exchanger records.	11/01/2009 4:00:00	11/01/2009 8:00:00
mi	MI DateAdd(' **mi** ', **4**, [Air Cooled Heat Exchanger].[Asset Installation Date])	Adds four minutes to the date on the Asset Installation Date field in Air Cooled Heat Exchanger records.	11/01/2009 4:00:00	11/01/2009 4:04:00
mi	MI DateAdd(' **mi** ', **-4**, [Air Cooled Heat Exchanger].[Asset Installation Date])	Subtracts four minutes from the date on the Asset Installation Date field in Air Cooled Heat Exchanger records.	11/01/2009 4:00:00	11/01/2009 3:56:00
ss	MI DateAdd(' **ss** ', **4**, [Air Cooled Heat Exchanger].[Asset Installation Date])	Adds four seconds to the date on the Asset Installation Date field in Air Cooled Heat Exchanger records.	11/01/2009 4:00:00	11/01/2009 4:00:04

Example: MI_DatePart

Suppose that you want to view only the month in which the shell and tube heat exchangers were installed in your facility. If all Shell and Tube Heat Exchanger records in your database contain an Asset Installation Date field, and the values are stored in the format mm/dd/yyyy, you could do so using the MI DatePart function.

You might configure a query on the Shell and Tube Heat Exchanger family, and add the Asset ID, Asset Description, and Installation Month fields to the query.

In this example, to return the month in which each shell and tube heat exchanger was installed, you would configure an expression using the MI DatePart function. You would also configure the alias to indicate that the column returns the year the piece of equipment was installed.

The expression syntax is:

```
MI DatePart('mm', [Shell and Tube Heat Exchanger].[Asset Installation
Date])
```

Using this syntax, instead of displaying the installation date in the stored format of mm/dd/yyyy, the query results will display only the value representing the month.

The following table lists more examples of how you can use the MI DatePart function.

Parameter	Example	Description	Stored Value	Result
yy	MI DatePart('**yy**', [Shell and Tube Heat Exchanger].[Asset Installation Date])	Displays only the year from a stored date value.	11/01/2009	2009
yyyy	MI DatePart('**yyyy**', [Shell and Tube Heat Exchanger].[Asset Installation Date])			
year	MI DatePart('**year**', [Shell and Tube Heat Exchanger].[Asset Installation Date])			
q	MI DatePart('**q**', [Shell and Tube Heat Exchanger].[Asset Installation Date])	Displays a number representing the quarter of the year based on a stored date value.	11/01/2009	4
qq	MI DatePart('**qq**', [Shell and Tube Heat Exchanger].[Asset Installation Date])			
quarter	MI DatePart('**quarter**', [Shell and Tube Heat Exchanger].[Asset Installation Date])			
m	MI DatePart('**m**', [Shell and Tube Heat Exchanger].[Asset Installation Date])	Displays only the month from a stored date value.	11/01/2009	11
mm	MI DatePart('**mm**', [Shell and Tube Heat Exchanger].[Asset Installation Date])			
month	MI DatePart('**month**', [Shell and Tube Heat Exchanger].[Asset Installation Date])			

Parameter	Example	Description	Stored Value	Result
wk	MI DatePart('**wk**', [Shell and Tube Heat Exchanger].[Asset Installation Date])	Displays a number representing the week of the month based on a stored date value.	11/01/2009	1
weekofmonth	MI DatePart('**weekofmonth**', [Shell and Tube Heat Exchanger].[Asset Installation Date])			
d	MI DatePart('**d**', [Shell and Tube Heat Exchanger].[Asset Installation Date])	Displays only the day from a stored date value.	11/01/2009	01
dd	MI DatePart('**dd**', [Shell and Tube Heat Exchanger].[Asset Installation Date])			
day	MI DatePart('**day**', [Shell and Tube Heat Exchanger].[Asset Installation Date])			
dw	MI DatePart('**dw**', [Shell and Tube Heat Exchanger].[Asset Installation Date])	Displays a number representing the day of the week based on a stored date value (where 1 = Sunday, 2 = Monday, etc.).	11/01/2009	1
weekday	MI DatePart('**weekday**', [Shell and Tube Heat Exchanger].[Asset Installation Date])			
y	MI DatePart('**y**', [Shell and Tube Heat Exchanger].[Asset Installation Date])	Displays a number representing the day of the year based on a stored date value.	11/01/2009	305
dy	MI DatePart('**dy**', [Shell and Tube Heat Exchanger].[Asset Installation Date])			
dayofyear	MI DatePart('**dayofyear**', [Shell and Tube Heat Exchanger].[Asset Installation Date])			

Parameter	Example	Description	Stored Value	Result
hh	MI DatePart('**hh**', [Shell and Tube Heat Exchanger].[Asset Installation Date])	Displays only the hour from a stored date value.	11/01/2009 4:00:00	4
hour	MI DatePart('**hour**', [Shell and Tube Heat Exchanger].[Asset Installation Date])			
n	MI DatePart('**n**', [Shell and Tube Heat Exchanger].[Asset Installation Date])	Displays only the minutes from a stored date value.	11/01/2009 4:00:00	00
mi	MI DatePart('**mi**', [Shell and Tube Heat Exchanger].[Asset Installation Date])			
minute	MI DatePart('**minute**', [Shell and Tube Heat Exchanger].[Asset Installation Date])			
s	MI DatePart('**s**', [Shell and Tube Heat Exchanger].[Asset Installation Date])	Displays only the seconds from a stored date value.	11/01/2009 4:00:00	00
ss	MI DatePart('**ss**', [Shell and Tube Heat Exchanger].[Asset Installation Date])			
second	MI DatePart('**second**', [Shell and Tube Heat Exchanger].[Asset Installation Date])			

Example: NOW

Suppose that you want to see the tasks that are assigned to a specific user and due in the next thirty days. If Task records contain the Next Date field that stores the date on which the task is due, you could do so using the NOW function.

You might configure a query on the Task family, and add the Task ID, Next Date, and Task Assigned To fields to the query.

In this example, you would configure an expression on the Next Date field using the NOW function. Additionally, you would configure a prompt to prompt the user for the name of the user whose Task records you want to view.

The expression syntax is:

```
(>= Now() AND <= (Now() + 30))
```

The syntax indicates that you want to view the tasks that are due today and within the next thirty days. For example, if a task is due thirty-one days after today's date, it will not appear in the query results.

The following table lists more examples of how you can use the NOW function.

Example Expression	Description	Current Date	Result
NOW(0) or NOW()	Displays the current date.	11/01/2009	11/01/2009
NOW() - 3	Displays the date that is three days prior to the current date.	11/01/2009	10/29/2009
NOW() + 3	Displays the date that is three days later than the current date.	11/01/2009	11/04/2009

Example: LastDate

Suppose that you want to see all the air cooled heat exchanger failures in your facility that occurred in a given month. If each Air Cooled Heat Exchanger record is linked to a Failure record, and each Failure record contains the Failure Date/Time field that stores that date and time on which a failure occurred, you could do so using the LAST DATE function.

You might configure a query on the Air Cooled Heat Exchanger and Failure families, joined via the Asset Has Failure relationship, and add the Asset ID, Failure ID, Failure Date/Time, and Last Day fields to the query.

In this example, you would configure an expression using the LAST DATE function. Additionally, you would create a prompt that prompts you to select the last day of the month whose failures you want to view. You would configure the alias to indicate that the column displays the last day of the month.

The expression syntax is:

```
LastDate([Failure].[Failure Date/Time])
```

The prompt syntax is:

```
((? :d :caption='Last Day' :id=Last Day))
```

When you run the query, a prompt appears, where you can select 11/30/2009 12:00:00 A.M. to indicate that you want to view failures that occurred during the month of November.

Example: ISNULL

Suppose that you want to see the installation date of the shell and tube heat exchangers in your facility even if the Asset Installation Date field in the Shell and Tube Heat Exchanger records is empty, and if the Asset Installation Date field is empty, you want to show the value Not Installed.

In this case, you might configure a query on the Shell and Tube Heat Exchanger family, and add the Asset ID, Asset Description, and Installation Dates (ALL) fields to the query.

In this example, you would configure an expression using the ISNULL function. You would also configure the alias to indicate that the column returns the installation dates.

In this case, the expression syntax is:

```
IsNull([Shell and Tube Heat Exchanger].[Asset Installation Date], 'Not
Installed')
```

...where **Not Installed** is the value that will appear where a null value is found on the Asset Installation Date field in the Shell and Tube Heat Exchanger records.

Example: ROUND

Suppose that you want to see the values on the As Left fields for the analyzer instruments in your facility rounded to two decimal places. If all Analyzer Calibration records contain the As Left field, you can do so using the ROUND function.

You might configure a query on the Analyzer Calibration family, and add the Analyzer ID, Calibration Date, and As Left (rounded) fields to the query.

In this example, you would configure an expression using the ROUND function. You would also configure the alias to indicate that the column displays the rounded as left values.

The expression syntax is:

```
ROUND([Analyzer Calibration].[As Left], '2')
```

...where **2** is the number of decimal places to which the values will be rounded in the results.

Example: DATENAME

Suppose that you want to see the name of the month in which the shell and tube heat exchangers were installed. Instead of 11, you want to see November, and you do not want to return the year or day in the query results. If all Shell and Tube Heat Exchanger records in your database contain an Asset Installation Date field, and the values are stored in the format mm/dd/yyyy (e.g., 11/01/2009), you could do so using the DATENAME function.

You might configure a query on the Shell and Tube Heat Exchanger family, and add the Asset ID, Asset Description, and Installed by Month fields to the query.

In this example, you would configure an expression using the DATENAME function. You would also configure the alias to indicate that the results display the month.

The expression syntax is:

```
DATENAME('mm', [Shell and Tube Heat Exchanger].[Asset Installation
Date])
```

...where mm indicates the part of the date (i.e., month) that you want to convert to its character format.

The following table lists more examples of how you can use the DATENAME function.

Parameter	Description	Example	Stored Value	Result
yy year	Displays a number representing the year from a stored date value.	DATENAME('**year**', [Shell and Tube Heat Exchanger].[Asset Installation Date])	11/01/2009	2009
qq quarter	Displays a number representing the quarter of the year based on a stored date value.	DATENAME('**qq**', [Shell and Tube Heat Exchanger].[Asset Installation Date])	11/01/2009	4
mm month	Displays the month from a stored date value.	DATENAME('**mm**', [Shell and Tube Heat Exchanger].[Asset Installation Date])	11/01/2009	November
ww	Displays a number representing the week of the month based on a stored date value.	DATENAME('**ww**', [Shell and Tube Heat Exchanger].[Asset Installation Date])	11/01/2009	1
dd day	Displays a number representing the day from a stored date value.	DATENAME('**day**', [Shell and Tube Heat Exchanger].[Asset Installation Date])	11/01/2009	01
dw	Displays the day of the week based on a stored date value.	DATENAME('**dw**', [Shell and Tube Heat Exchanger].[Asset Installation Date])	11/01/2009	Sunday
dy dayofyear	Displays a number representing the day of the year based on a stored date value.	DATENAME('**dayofyear**', [Shell and Tube Heat Exchanger].[Asset Installation Date])	11/01/2009	305

Example: DATEPART

Suppose that you want to see the year in which the shell and tube heat exchangers were installed in your facility. If all Shell and Tube Heat Exchanger records in your database contain an Asset Installation Date field, and the values are stored in the format mm/dd/yyyy (e.g., 11/01/2009), you could do so using the DATEPART function.

You might configure a query on the Shell and Tube Heat Exchanger family, and add the Asset ID, Asset Description, and Installation Year fields to the query.

In this example, you would configure an expression using the DATEPART function. You would also configure the alias to indicate that the column displays the installation year.

The expression syntax is:

```
DATEPART('year', [Shell and Tube Heat Exchanger].[Asset Installation
Date])
```

...where `year` is the portion of the date value that you want to view.

Parameter	Example	Description	Stored Value	Result
yy	DATEPART('**yy**', [Shell and Tube Heat Exchanger].[Asset Installation Date])	Displays a number representing the year from a stored date value.	11/01/2009	2009
yyyy	DATEPART('**yy**', [Shell and Tube Heat Exchanger].[Asset Installation Date])			
year	DATEPART('**year**', [Shell and Tube Heat Exchanger].[Asset Installation Date])			
q	DATEPART('**q**', [Shell and Tube Heat Exchanger].[Asset Installation Date])	Displays a number representing the quarter of the year based on a stored date value.	11/01/2009	4
qq	DATEPART('**qq**', [Shell and Tube Heat Exchanger].[Asset Installation Date])			
quarter	DATEPART('**quarter**', [Shell and Tube Heat Exchanger].[Asset Installation Date])			
m	DATEPART('**m**', [Shell and Tube Heat Exchanger].[Asset Installation Date])	Displays a number representing the month from a stored date value.	11/01/2009	11
mm	DATEPART('**mm**', [Shell and Tube Heat Exchanger].[Asset Installation Date])			
month	DATEPART('**month**', [Shell and Tube Heat Exchanger].[Asset Installation Date])			

Parameter	Example	Description	Stored Value	Result
wk	DATEPART('**wk**', [Shell and Tube Heat Exchanger].[Asset Installation Date])	Displays a number representing the week of the month based on a stored date value.	11/01/2009	1
ww	DATEPART('**ww**', [Shell and Tube Heat Exchanger].[Asset Installation Date])			
week	DATEPART('**week**', [Shell and Tube Heat Exchanger].[Asset Installation Date])			
d	DATEPART('**d**', [Shell and Tube Heat Exchanger].[Asset Installation Date])	Displays a number representing the day from a stored date value.	11/01/2009	1
dd	DATEPART('**dd**', [Shell and Tube Heat Exchanger].[Asset Installation Date])			
day	DATEPART('**day**', [Shell and Tube Heat Exchanger].[Asset Installation Date])			
dw	DATEPART('**dw**', [Shell and Tube Heat Exchanger].[Asset Installation Date])	Displays a number representing the day of the week based on a stored date value.	11/01/2009	1 (i.e., Sunday)
weekday	DATEPART('**weekday**', [Shell and Tube Heat Exchanger]. [Asset Installation Date])			
y	DATEPART('**y**', [Shell and Tube Heat Exchanger].[Asset Installation Date])	Displays a number representing the day of the year based on a stored date value.	11/01/2009	305
dy	DATEPART('**dy**', [Shell and Tube Heat Exchanger].[Asset Installation Date])			
dayofyear	DATEPART('**dayofyear**', [Shell and Tube Heat Exchanger]. [Asset Installation Date])			

Parameter	Example	Description	Stored Value	Result
hh	DATEPART ('**hh**', [Shell and Tube Heat Exchanger].[Asset Installation Date])	Displays only the hour from a stored date value.	11/01/2009 4:00:00	4
hour	DATEPART ('**hour**', [Shell and Tube Heat Exchanger].[Asset Installation Date])			
n	DATEPART ('**n**', [Shell and Tube Heat Exchanger].[Asset Installation Date])	Displays only the minutes from a stored date value.	11/01/2009 4:00:00	00
mi	DATEPART ('**mi**', [Shell and Tube Heat Exchanger].[Asset Installation Date])			
minute	DATEPART ('**minute**', [Shell and Tube Heat Exchanger].[Asset Installation Date])			
s	DATEPART ('**s**', [Shell and Tube Heat Exchanger].[Asset Installation Date])	Displays only the seconds from a stored date value.	11/01/2009 4:00:00	00
ss	DATEPART ('**ss**', [Shell and Tube Heat Exchanger].[Asset Installation Date])			
second	DATEPART ('**second**', [Shell and Tube Heat Exchanger].[Asset Installation Date])			

About Number/Mathematical Functions

A number/mathematical function is used to manipulate values returned on numeric fields.

Tip: For more information on functions, refer to MetaSQL Functions on page 80.

About the DECODE Function

A DECODE function is used to evaluate a value in the query results and, based on that evaluation, trigger an action or return a different value.

Example: DECODE Function

Suppose that, in your organization, you have open and closed work history events. Each Work History record contains an Order System Status field,

which is used to record the status of that work history event. You want to see how many open work history events you have and how many closed work history events you have. Instead of the stored values of CLSD TECO or OPEN, however, you want to see the values Closed or Open in your query results.

In addition, you are concerned that not all Work History records contain a value in the Order System Status field. You also want to see in your query results which records do not have a value in this field so that you can update those records.

In this case, you might configure a query on the Work History family, and then add the Expr and Order System Status fields to the query.

In this example, to return the status of each work history event as Closed, Open, or No Status (meaning that the record does not contain a value in the Order System Status field), you can configure an expression using the DECODE function.

In this case, the expression syntax is:

```
                    Decode([Work History].[Order System
Status], 'CLSD TECO', 'Closed', 'OPEN', 'Open', 'No
Status')
```

This syntax indicates that for Work History records that contain the value CLSD TECO in the Order System Status field, you want to return the value Closed. For those records with the value OPEN in the Order System Status field, you want to return the value Open. For records where the Order System Status field is empty, you want to return the value No Status.

You would also add a COUNT function on the Order System Status field so that you can see the number of work history events that fall into each category.

Other possible uses include:

- You want to review the types of maintenance activities that are being performed in your plant and the total number of failures that resulted in each type of activity. Users can enter any value they choose into a field that tracks maintenance activities, so you know that your users are using different terminology to mean the same thing. For example, to indicate that they replaced broken components, different users might enter replace, replacing, or replacement. You can write a DECODE statement to indicate that the values replace, replacing, and replacement should return the value Replace in the query results.
- Members of management want to investigate failures that resulted in a failure cost of $50,000 or more. You can write a DECODE statement to indicate that failures with a failure cost greater than or equal to $50,000 should return the value Please Investigate in the query results. Other failures should return the value OK in the query results.

MetaSQL Functions

Aggregate Functions

Aggregate functions perform a calculation on a set of values and return a single value. Use of the OVER clause converts the function into an Analytic Function.

- AVG
- COUNT
- MAX
- MIN
- SUM

Meridium Functions

Meridium functions are supported regardless of what database management system you use. The MetaSQL compiler will automatically convert the expression to a native SQL expression.

Function Syntax:

```
<meridium_function> ::=
{
  function_id ( [ { <expression> [,...n] } | * ] )
}
```

...where function_id is the name of the function you are executing.

The following functions are supported:

UserKey

Gets the key of the current logged-in user.

```
SELECT UserKey()
FROM <source>
```

Str

Converts the designated field to a VARCHAR value.

```
-- Syntax
SELECT Str(<expression> [, <size>]) "Expr" FROM <source>

-- Simple Example
SELECT [MI_ACTION].ENTY_KEY "ENTY_KEY"
, Str([MI_ACTION].ENTY_KEY) "Str Enty Key"
FROM [MI_ACTION]

-- Specify the size
SELECT [MI_ACTION].ENTY_KEY "ENTY_KEY"
, Str([MI_ACTION].ENTY_KEY, 20) "Str Enty Key"
FROM [MI_ACTION]
```

IsNull

Evaluates an expression. If the value is null, evaluates the expression in the second argument and returns its result.

Note: Both arguments must return the same data type.

```
-- Syntax
SELECT IsNull(<expression>, <expression>) "Expr" FROM <source>

-- Example (Note the conversion of number to string ensuring
consistent data types)
SELECT [MI_ACTION].[MI_ACTION_ID_C] "Action ID"
, [MI_ACTION].[MI_ACTION_SHORT_DESC_C] "Name"
, IsNull(Str([MI_ACTION].[MI_ACTION_RESOURCE_COST_N]), 'No Cost
Data') "Cost"
FROM [MI_ACTION]
```

Now

Returns the DBMS system date.

```
-- Syntax
SELECT Now()
FROM <source>
```

LocalizedCaption

Returns a localized Family Caption for a given Family Key and User Key.

```
-- Syntax
SELECT LocalizedCaption(<family_key>, <user_key>)
FROM <source>

-- Example
SELECT mi_families.FMLY_CAPTION_TX "Caption"
, LocalizedCaption(mi_families.FMLY_KEY, 64251708261) "Localized
Caption"
FROM mi_families
```

Decode

Returns a given output based on one or more possible inputs.

```
-- Syntax
SELECT DECODE(<field_id>, <input_value>, <output_value> [ ,
<input_value>, <output_value> ], <default_value> )
FROM <source>

-- Example
SELECT [MI_ACTION].[MI_ACTION_ID_C] "Action ID"
, [MI_ACTION].[MI_ACTION_SHORT_DESC_C] "Name"
, [MI_ACTION].[MI_ACTION_RESOURCE_COST_N] "Cost"
, Decode([MI_ACTION].[MI_ACTION_RESOURCE_COST_N], 100, 'Cheap',
10000, 'Moderately Expensive', 100000, 'Expensive', 'Misc') "Cost
Category"
FROM [MI_ACTION]
ORDER BY "Cost" Desc
```

DatePart

Returns the specified part of a date/time value. DatePart and MI_DatePart are equivalent. The return is a number.

```
--Syntax
SELECT DatePart( <date_part>, <expression> )
FROM <source>

-- Example
SELECT [MI_ACTION].[MI_ACTION_ID_C] "Action ID"
, [MI_ACTION].[MI_ACTION_SHORT_DESC_C] "Name"
, [MI_ACTION].[MI_ACTION_TARGE_COMPL_DATE_CHR] "Target Completion
Date"
, DatePart('yy', [MI_ACTION].[MI_ACTION_TARGE_COMPL_DATE_CHR])
"Year"
, DatePart('mm', [MI_ACTION].[MI_ACTION_TARGE_COMPL_DATE_CHR])
"Month"
, DatePart('dd', [MI_ACTION].[MI_ACTION_TARGE_COMPL_DATE_CHR]) "Day"
, DatePart('hh', [MI_ACTION].[MI_ACTION_TARGE_COMPL_DATE_CHR])
"Hour"
, DatePart('mi', [MI_ACTION].[MI_ACTION_TARGE_COMPL_DATE_CHR])
"Minute"
, DatePart('ss', [MI_ACTION].[MI_ACTION_TARGE_COMPL_DATE_CHR])
"Second"
, DatePart('dw', [MI_ACTION].[MI_ACTION_TARGE_COMPL_DATE_CHR]) "Day
of the week"
, DatePart('qq', [MI_ACTION].[MI_ACTION_TARGE_COMPL_DATE_CHR])
"Quarter"
, DatePart('dy', [MI_ACTION].[MI_ACTION_TARGE_COMPL_DATE_CHR]) "Day
of the year"
, DatePart('ww', [MI_ACTION].[MI_ACTION_TARGE_COMPL_DATE_CHR])
"Week"
FROM [MI_ACTION]
WHERE [MI_ACTION].[MI_ACTION_TARGE_COMPL_DATE_CHR] IS NOT NULL
```

The date_part can contain any of the following values:

- 'quarter', 'qq', 'q' - Quarter of the year, 1 - 4
- 'year', 'yyyy', 'yy' - Year, 4 digits (always)
- 'month', 'mm', 'm' - Month of the year, 1 - 12
- 'day', 'dd', 'd' - Day of the month, 1 - 31
- 'hour', 'hh', 'h' - Hour of the day, 0 - 23
- 'minute','mi', 'n' - Minute of the hour, 0 - 59
- 'second', 'ss', 's' - Second of the minute, 0 - 59
- 'dayofyear', 'dy', 'y' - Day of the year
- 'dayofweek', 'weekday', 'dw' - Day of the week, 1 - 7, Sunday is 1
- 'weekofyear', 'week', 'ww' - Week of the year
- 'weekofmonth', 'wk' - Week of month

DateAdd

Adds the specified number of units to a given date/time and returns a new date/time value. DateAdd and MI_DateAdd are equivalent. On Oracle databases, adding years and months is not supported.

```
-- Syntax
SELECT DateAdd( <date_part>, <num_const>, <expression> )
```

```
FROM <source>

-- Example
SELECT [MI_ACTION].[MI_ACTION_ID_C] "Action ID"
, [MI_ACTION].[MI_ACTION_SHORT_DESC_C] "Name"
, [MI_ACTION].[MI_ACTION_TARGE_COMPL_DATE_CHR] "Target Completion
Date"
, DateAdd('yy', 10, [MI_ACTION].[MI_ACTION_TARGE_COMPL_DATE_CHR])
"Add 10 Years"
, DateAdd('mm', 10, [MI_ACTION].[MI_ACTION_TARGE_COMPL_DATE_CHR])
"Add 10 Months"
, DateAdd('dd', 10, [MI_ACTION].[MI_ACTION_TARGE_COMPL_DATE_CHR])
"Add 10 Days"
, DateAdd('hh', 10, [MI_ACTION].[MI_ACTION_TARGE_COMPL_DATE_CHR])
"Add 10 Hours"
, DateAdd('mi', 10, [MI_ACTION].[MI_ACTION_TARGE_COMPL_DATE_CHR])
"Add 10 Minutes"
, DateAdd('ss', 10, [MI_ACTION].[MI_ACTION_TARGE_COMPL_DATE_CHR])
"Add 10 Seconds"
FROM [MI_ACTION]
WHERE [MI_ACTION].[MI_ACTION_TARGE_COMPL_DATE_CHR] IS NOT NULL
```

The date_part can contain one of the following values:

- 'year', 'yyyy', 'yy' - Year
- 'month', 'mm', 'm' - Month
- 'day', 'dd', 'd' - Day
- 'hh', 'h' - Hour
- 'minute', 'mi', 'n' - Minute
- 'second', 'ss', 's' - Second

The num_const is the number of units that will be added to the provided date.

DateFormat

Returns a date/time value formatted as a string.

```
--Syntax
SELECT DateFormat( <date_format>, <expression> )
FROM <source>

-- Example
SELECT [MI_ACTION].[MI_ACTION_ID_C] "Action ID"
, [MI_ACTION].[MI_ACTION_SHORT_DESC_C] "Name"
, [MI_ACTION].[MI_ACTION_TARGE_COMPL_DATE_CHR] "Target Completion
Date"
, DateFormat('yyyy-mm-dd', [MI_ACTION].
[MI_ACTION_TARGE_COMPL_DATE_CHR]) "Date"
FROM [MI_ACTION]
WHERE [MI_ACTION].[MI_ACTION_TARGE_COMPL_DATE_CHR] IS NOT NULL
```

The date_format can contain any reasonable combination of the following values:

- 'yyyy' - four digit year
- 'yyy' - last 3 digits of the year
- 'yy' - last 2 digits of the year
- 'y' - last digit of the year
- 'mm' - month of the year, 01 - 12

- 'mon' - abbrreviated month name
- 'dd' - Day of the month, 01 - 31
- 'ddd' - Day of the year, 001 - 366
- 'd' - Day of the week, 1 - 7, Sunday is 1
- 'dy' - abbrreviated day name
- 'hh' - Hour of the day, 01 - 12
- 'hh12' - Hour of the day, 01 - 12
- 'hh24' - Hour of the day, 00 - 23
- 'mi' - Minute of the hour, 00 - 59
- 'ss' - Second of the minute, 00 - 59
- 'ms' - Millisecond, 000 - 999
- 'am' - (or pm) meridiem indicator

Note:

- Oracle does not support fractional seconds.
- Microsoft does not support day of the year.

DateName

Returns the month or day name for date/time value. Note that this value will not be localized.

```
--Syntax
SELECT DateName( <date_format>, <expression> )
FROM <source>

-- Example
SELECT [MI_ACTION].[MI_ACTION_ID_C] "Action ID"
, [MI_ACTION].[MI_ACTION_SHORT_DESC_C] "Name"
, [MI_ACTION].[MI_ACTION_TARGE_COMPL_DATE_CHR] "Target Completion
Date"
, DateFormat('day', [MI_ACTION].[MI_ACTION_TARGE_COMPL_DATE_CHR])
"Day"
FROM [MI_ACTION]
WHERE [MI_ACTION].[MI_ACTION_TARGE_COMPL_DATE_CHR] IS NOT NULL
```

The date_format can contain any of the following values:

- 'month', 'mm', 'm' - Month of the year
- 'day', 'dd', 'weekday', 'dw' - Day of the week

UTC

Returns the formatted string equivalent of the argument date/time value. The return will be formatted 'yyyy-mm-dd hh:mi:ss'.

```
--Syntax
SELECT UTC(<expression> )
FROM <source>

-- Example
SELECT [MI_ACTION].[MI_ACTION_ID_C] "Action ID"
, [MI_ACTION].[MI_ACTION_TARGE_COMPL_DATE_CHR] "Target Completion
Date"
, UTC([MI_ACTION].[MI_ACTION_TARGE_COMPL_DATE_CHR]) "UTC Date"
```

```
FROM [MI_ACTION]
WHERE [MI_ACTION].[MI_ACTION_TARGE_COMPL_DATE_CHR] IS NOT NULL
```

DateDiff

Returns the difference between to date/time values as number. The argument determines the unit of measure returned. The expression1 argument is subtracted from expression2.

```
--Syntax
SELECT DateDiff( <date_part>, <expression1>, <expression2>)
FROM <source>

-- Example
SELECT [MI_ACTION].[MI_ACTION_ID_C] "Action ID"
, [MI_ACTION].[MI_ACTION_SHORT_DESC_C] "Name"
, [MI_ACTION].[MI_ACTION_TARGE_COMPL_DATE_CHR] "Target Completion
Date"
, DateDiff('day', [MI_ACTION].[MI_ACTION_TARGE_COMPL_DATE_CHR],
[MI_ACTION].LAST_UPDT_DT) "Days"
FROM [MI_ACTION]
WHERE [MI_ACTION].[MI_ACTION_TARGE_COMPL_DATE_CHR] IS NOT NULL
```

The date_PART can contain any of the following values:

- 'day', 'dd', 'd' - Days
- 'hour', 'hh', 'h' - Hours
- 'minute', 'mi', 'n' - Minutes
- 'second', 'ss', 's' - Seconds

Length

Returns the length of the specified expression.

```
-- Syntax
SELECT Length( <expression> )
FROM <source>

-- Example
, [MI_ACTION].[MI_ACTION_SHORT_DESC_C] "Name"
, Length([MI_ACTION].[MI_ACTION_SHORT_DESC_C]) "Name Length"
, [MI_ACTION].[MI_ACTION_TARGE_COMPL_DATE_CHR] "Target Completion
Date"
, Str([MI_ACTION].[MI_ACTION_TARGE_COMPL_DATE_CHR]) "Target
Completion Date String"
, Length(Str([MI_ACTION].[MI_ACTION_TARGE_COMPL_DATE_CHR])) "Target
Completion Date Length"
, (1000000 + 2689432) "Test"
, Length((1000000 + 2689432)) "Test Length"
FROM [MI_ACTION]
```

Note: When determining string length of a Date/Time value, the results may not be as expected. To ensure predictable results, cast the Date/Time to a string with a known format prior to passing it into the Length() function.

CastChar, CastNu,, CastDate

Takes an expression and attempts to return a value cast to the desired type.

```
-- Syntax
SELECT CastChar(<expression>, <size>)
```

```
FROM <source>

SELECT CastNum(<expression>)
FROM <source>

SELECT CastDate(<expression>, <date_format_string>)
FROM <source>

-- Example
SELECT TOP 1 CastChar(Now()) "Now string"
, CastChar(1234567, 7) "Number string"
, CastNum('1234567') "Number from string"
, CastDate('2009-01-01', 'YYYY-MM-DD') "Date from string"
FROM [MI_ACTION]
```

The size accepts a <num_const> representing the size of the VARCHAR.

The date_format_string is a format string for the date/time value (i.e. YYYY-MM-DD). This string format needs to be compatible with the underlying DBMS.

IndexOf

Returns the index (1-based) of a given character expression within another character expression.

Note: This function is case-sensitive for Oracle and PostgreSQL, but case-insensitive for SQL.

```
-- Syntax
SELECT IndexOf( <char_const_to_search>, <char_const_to_find> [,
<start_position>] )
FROM <source>

-- Example
SELECT [MI_ACTION].[MI_ACTION_ID_C] "Action ID"
, [MI_ACTION].[MI_ACTION_SHORT_DESC_C] "Name"
, INDEXOF([MI_ACTION].[MI_ACTION_SHORT_DESC_C], 'c') "Index of C"
FROM [MI_ACTION]
```

Modulus

Returns the modulus (remainder) of exp1 divided by exp2.

```
-- Syntax
SELECT Modulus( <exp1>, <exp2>)
FROM <source>
```

Substring

```
-- Syntax
SELECT Substring( <expression>, <start_pos> [, <end_pos>] )
FROM <source>

-- Example
SELECT [MI_ACTION].[MI_ACTION_ID_C] "Action ID"
, [MI_ACTION].[MI_ACTION_SHORT_DESC_C] "Name"
, Substring([MI_ACTION].[MI_ACTION_SHORT_DESC_C], 2) "Name
substring from pos 2"
, Substring([MI_ACTION].[MI_ACTION_SHORT_DESC_C], 2, 6) "Name
```

```
substring from pos 2-6"
FROM [MI_ACTION]
```

Upper, Lower

Return the argument string expression in upper or lower case.

```
-- Syntax
SELECT Upper(<expression>)
FROM <source>

SELECT Lower(<expression>)
FROM <source>

-- Example
SELECT [MI_ACTION].[MI_ACTION_ID_C] "Action ID"
, [MI_ACTION].[MI_ACTION_SHORT_DESC_C] "Name"
, Upper([MI_ACTION].[MI_ACTION_SHORT_DESC_C]) "Upper"
, Lower([MI_ACTION].[MI_ACTION_SHORT_DESC_C]) "Lower"
FROM [MI_ACTION]
```

LTrim, RTrim, Trim

Return a character expression after removing the leading or trailing white space (or both, for Trim).

```
-- Syntax
SELECT LTrim(<expression>)
FROM <source>

SELECT RTrim(<expression>)
FROM <source>

-- Example
SELECT [MI_ACTION].[MI_ACTION_ID_C] "Action ID"
, [MI_ACTION].[MI_ACTION_SHORT_DESC_C] "Name"
, ('----|' & '     ' & [MI_ACTION].[MI_ACTION_SHORT_DESC_C] & '
' & '|----') "Untrimmed Name"
, ('----|' & LTRIM(('     ' & [MI_ACTION].[MI_ACTION_SHORT_DESC_C]
& '     ')) & '|----') "LTrimmed Name"
, ('----|' & RTRIM(('     ' & [MI_ACTION].[MI_ACTION_SHORT_DESC_C]
& '     ')) & '|----') "RTrimmed Name"
FROM [MI_ACTION]
```

LPad, RPad

Return a character expression of a specified length after padding with a specified character. A space is the default padchar value.

```
-- Syntax
SELECT LPad(<expression>, <length> [, <padchar>])
FROM <source>

SELECT RPad(<expression>, <length> [, <padchar>])
FROM <source>

-- Example
SELECT [MI_ACTION].[MI_ACTION_ID_C] "Action ID"
, LPAD(CASTCHAR(DatePart('d', [MI_ACTION].
```

```
[MI_ACTION_TARGE_COMPL_DATE_CHR])), 2, ' ') "Day"
FROM [MI_ACTION]
```

Concat

Concatenates the arguments returning a string. The argument list is a params argument (multiple arguments are supported). At least two arguments are required.

```
-- Syntax
SELECT Concat(<exp1>, <exp2> [,<expn>])
FROM <source>

SELECT Concat(<exp1>, <exp2>)
FROM <source>

-- Example
SELECT Concat('foo-', [MI_ACTION].[MI_ACTION_ID_C], '-bar') "Action
ID"
FROM [MI_ACTION]
```

Year, Month, Day

Return a part of the specified date expression. For more control or granularity when retrieving date parts, see the DatePart function.

```
-- Syntax
SELECT Year(<date_expression>)
FROM <source>

SELECT Month(<date_expression>)
FROM <source>

SELECT Day(<date_expression>)
FROM <source>

-- Example
SELECT [MI_ACTION].[MI_ACTION_ID_C] "Action ID"
, [MI_ACTION].[MI_ACTION_TARGE_COMPL_DATE_CHR] "Target Completion
Date"
, YEAR([MI_ACTION].[MI_ACTION_TARGE_COMPL_DATE_CHR]) "Target
Completion Year"
, MONTH([MI_ACTION].[MI_ACTION_TARGE_COMPL_DATE_CHR]) "Target
Completion Month"
, DAY([MI_ACTION].[MI_ACTION_TARGE_COMPL_DATE_CHR]) "Target
Completion Day"
FROM [MI_ACTION]
```

SysDate

Retrieves the system date.

```
-- Syntax
SELECT SysDate()
FROM <source>

-- Example
SELECT [MI_ACTION].[MI_ACTION_ID_C] "Action ID"
, SYSDATE() "System Date"
FROM [MI_ACTION]
```

LastDate

Returns a date representing the last day of the month in which a given date occurs.

```
-- Syntax
SELECT LastDate(<const_char>)
FROM <source>

-- Example
SELECT [MI_ACTION].[MI_ACTION_ID_C] "Action ID"
, LASTDATE('2016-02-01') "Last Day of Feb, 2016"
, LASTDATE('2017-02-01') "Last Day of Feb, 2017"
, LASTDATE('2018-02-01') "Last Day of Feb, 2018"
, LASTDATE('2019-02-01') "Last Day of Feb, 2019"
, LASTDATE('2020-02-01') "Last Day of Feb, 2020"
FROM [MI_ACTION]
```

cost_char must be formatted as YYYY-MM-DD.

Case

Performs a series of evaluations and returns the first result that evaluates to True. The simple syntax works well for an EQUALS comparison. However, if more complexity is required, you can use the general CASE expression to perform various types of comparisons.

```
-- Syntax (simple CASE expression)
SELECT CASE <field_id>
  WHEN <expression> THEN <expression>
  [ ELSE <expression> ]
END
FROM <source>

-- Example
SELECT [MI_ACTION].[MI_ACTION_ID_C] "Action ID"
, [MI_ACTION].[MI_ACTION_DESCRIPTION_T] "Description"
, CASE [MI_ACTION].[MI_ACTION_DESCRIPTION_T]
    WHEN 'Perform check of lubricant, add or change oil when needed'
THEN 'Check'
    WHEN 'Perform changeout of lubricant' THEN 'Changeout'
ELSE 'Other'
END "Lubricant Action"
FROM [MI_ACTION]
WHERE [MI_ACTION].[MI_ACTION_DESCRIPTION_T] LIKE '%Lubricant%'


-- Syntax (general CASE expression)
SELECT CASE
  WHEN <boolean_expression> THEN <expression>
  [ ELSE <expression> ]
END
FROM <source>

-- Example
SELECT [MI_ACTION].[MI_ACTION_ID_C] "Action ID"
, [MI_ACTION].[MI_ACTION_DESCRIPTION_T] "Description"
, CASE
    WHEN [MI_ACTION].[MI_ACTION_DESCRIPTION_T] LIKE 'Perform check
of lubricant%' THEN 'Check'
    WHEN [MI_ACTION].[MI_ACTION_DESCRIPTION_T] LIKE 'Perform
```

```
changeout%' THEN 'Changeout'
ELSE 'Other'
END "Lubricant Action"
FROM [MI_ACTION]
WHERE [MI_ACTION].[MI_ACTION_DESCRIPTION_T] LIKE '%Lubricant%'
```

Sign

Returns the sign of the numeric argument (-1, 0, +1).

```
-- Syntax
SELECT Sign(<exp1>)
FROM <source>
```

Abs

Returns the absolute value of the numeric the argument.

```
-- Syntax
SELECT abs(<exp1>)
FROM <source>
```

Reverse

Returns the reverse order of a string value.

```
-- Syntax
SELECT Reverse(<exp1>)
FROM <source>
```

Floor

Returns the largest integer less than or equal to the argument numeric expression.

```
-- Syntax
SELECT Floor(<exp1>)
FROM <source>
```

Round

Returns the nearest integer for the argument numeric expression. Rounds to the precision, if specified.

```
-- Syntax
SELECT Round(<exp1> [, <precision>])
FROM <source>
```

Analytic Functions

Analytic functions are used for aggregating data on a row-by-row basis. They work similar to aggregate functions, except they can return multiple rows of results for each group. They are useful for calculating running totals, moving percentages, and so on.

Analytic functions work the same way, regardless of the database management system. For more information on analytic functions, refer to https://learn.microsoft.com/en-us/sql/t-sql/functions/analytic-functions-transact-sql?view=sql-server-2017.

Syntax:

```
<analytic_function> ::= function_id() [OVER ( [PARTITION BY field_id]
ORDER BY <order_by_expression> )]
```

...where function_id is the name of the function you are executing, field_id is the field used to group results, and <order_by_expression> specifies which fields must be used to sort the results in the current partition.

The following functions are supported:

SUM

```
SELECT SUM([FIELD]) OVER(PARTITION BY [FIELD], [FIELD]... ORDER BY
[FIELD] ASC, [FIELD] DESC... ROWS BETWEEN CURRENT ROW AND 1
FOLLOWING) "Cost" FROM [FAMILY]
```

MAX

```
SELECT MAX([FIELD]) OVER(PARTITION BY [FIELD], [FIELD]... ORDER BY
[FIELD] ASC, [FIELD] DESC... ROWS BETWEEN CURRENT ROW AND 1
FOLLOWING) "Cost" FROM [FAMILY]
```

AVG

```
SELECT AVG([FIELD]) OVER(PARTITION BY [FIELD], [FIELD]... ORDER BY
[FIELD] ASC, [FIELD] DESC... ROWS BETWEEN CURRENT ROW AND 1
FOLLOWING) "Expr" FROM [FAMILY]
```

COUNT

```
SELECT COUNT([FIELD]) OVER(PARTITION BY [FIELD], [FIELD]... ORDER
BY [FIELD] ASC, [FIELD] DESC... ROWS BETWEEN CURRENT ROW AND 1
FOLLOWING) "Expr" FROM [FAMILY]
```

ROW_NUMBER

```
SELECT ROW_NUMBER() OVER(PARTITION BY [FIELD], [FIELD]... ORDER BY
[FIELD] ASC, [FIELD] DESC...) "Row #" FROM [FAMILY]
```

CUME_DIST

```
SELECT CUME_DIST() OVER ([PARTITION BY [FIELD], [FIELD]...] ORDER
BY [FIELD] ASC, [FIELD] DESC...) "Expr" FROM [FAMILY]
```

PERCENT_RANK

```
SELECT PERCENT_RANK([FIELD]) OVER(PARTITION BY [FIELD], [FIELD]...
ORDER BY [FIELD] ASC, [FIELD] DESC... ROWS BETWEEN CURRENT ROW AND
1 FOLLOWING) "Expr" FROM [FAMILY]
```

PERCENTILE_DISC

```
SELECT PERCENTILE_DISC([FIELD]) OVER(PARTITION BY [FIELD],
[FIELD]... ORDER BY [FIELD] ASC, [FIELD] DESC... ROWS BETWEEN
CURRENT ROW AND 1 FOLLOWING) "Expr" FROM [FAMILY]
```

PERCENTILE_CONT

```
SELECT PERCENTILE_CONT([FIELD]) OVER(PARTITION BY [FIELD],
[FIELD]... ORDER BY [FIELD] ASC, [FIELD] DESC... ROWS BETWEEN
CURRENT ROW AND 1 FOLLOWING) "Expr" FROM [FAMILY]
```

NTILE

```
SELECT NTILE(num_const) OVER(PARTITION BY [FIELD], [FIELD]... ORDER
BY [FIELD] DESC) "Expr" FROM [FAMILY]
```

num_const is a positive integer representing the number of groups that will be created in the result

---

Using Max, Min, Avg, Count, and Sum

```
SELECT DISTINCT [MI_MRBIANAL].[MI_CRITANAL_RSK_CAT_C]
"Risk Category"
, Min([MI_MRBIANAL].[MI_CRITANAL_PROB_OF_FAIL_UP_C])OVER
( PARTITION BY [MI_MRBIANAL].[MI_CRITANAL_RSK_CAT_C] )
"Min Prob of Fail"
, Max([MI_MRBIANAL].[MI_CRITANAL_PROB_OF_FAIL_UP_C])OVER
( PARTITION BY [MI_MRBIANAL].[MI_CRITANAL_RSK_CAT_C] )
"Max Prob of Fail"
, Avg([MI_MRBIANAL].[MI_CRITANAL_INSPE_PRIOR_UP_N])OVER
( PARTITION BY [MI_MRBIANAL].[MI_CRITANAL_RSK_CAT_C] )
"Avg Insp Priority"
, Count([MI_EQUIP000].ENTY_ID)OVER ( PARTITION BY
[MI_MRBIANAL].[MI_CRITANAL_RSK_CAT_C] ) "# Assets"
, Sum([MI_MRBIANAL].[MI_CRITANAL_LEAK_QUANTITY_N])OVER
( PARTITION BY [MI_MRBIANAL].[MI_CRITANAL_RSK_CAT_C] )
"Total Leak Quantity"
FROM {MIR_HSRBICMP}
 JOIN [MI_CCRBICOM] ON {MIR_HSRBICMP}.SUCC_ENTY_KEY =
[MI_CCRBICOM].ENTY_KEY
 JOIN {MIR_HSRBICMP} Has_RBI_Components1 ON
[MI_CCRBICOM].ENTY_KEY =
Has_RBI_Components1.SUCC_ENTY_KEY
 JOIN {MIR_RBICRAN} ON [MI_CCRBICOM].ENTY_KEY =
{MIR_RBICRAN}.PRED_ENTY_KEY
 JOIN [MI_EQUIP000] ON Has_RBI_Components1.PRED_ENTY_KEY
= [MI_EQUIP000].ENTY_KEY
 JOIN [MI_MRBIANAL] ON {MIR_RBICRAN}.SUCC_ENTY_KEY =
[MI_MRBIANAL].ENTY_KEY
ORDER BY [MI_MRBIANAL].[MI_CRITANAL_RSK_CAT_C] Asc
```

Results:

RiskCategory	MinProbofFail	MaxProbofFail	AvgInspPriority	#Assets	TotalLeakQuantity
	1	2	0	615	4
HIGH	1	2	2.4256756756756756	148	1,216,204.0043713066

RiskCategory	MinProbofFail	MaxProbofFail	AvgInspPPriority	#Assets	TotalLeakQuantity
LOW	3	5	22.439759036144458	166	590,251.125344949451

RiskCategory	MinProbofFail	MaxProbofFail	AvgInspPPriority	#Assets	TotalLeakQuantity
MEDIUM	2	5	16.33181818181818182	220	813,327.89638713

RiskCategory	MinProbofFail	MaxProbofFail	AvgInspPriority	#Assets	TotalLeakQuantity
MEDIUMHIGH	1	4	8.48892473118279956	186	1,305,554.66027321955

Additional Meta-SQL Functions

Meta-SQL constructs are used in functions that pass SQL strings.

The following table provides a list of Meta-SQL functions that you can use in query expressions in APM.

Description	Meta-SQL Function	Construction
Return the local time of the database server.	SysDate	**SysDate**
Evaluate one or more When expressions and return the appropriate Then expression. The Else condition is optional. The End statement is required. This function returns the datatype thenexpression. **Note:** This function is not a selectable option in the **Design** workspace, but you can enter the function directly in the **SQL** workspace.	Case	**Case evalexpression When whenexpression Then thenexpression [When whenexpression Then thenexpression] [Else thenexpression] End** **Evalexpression:** Any column or literal value. **Whenexpression:** Must be same datatype of evalexpression. **Thenexpression:** All must be the same datatype. Does not have to be the same datatype as evalexpression or whenexpression.

Modulus Function

```
SELECT  Modulus(ROUND([AQA REG All Fld Types].
[ARAQA_REG_ALL_FLD_TY_NUMER_NBR]), 2) "Modulus2"
, Modulus(ROUND([AQA REG All Fld Types].
[ARAQA_REG_ALL_FLD_TY_NUMER_NBR]), 3) "Modulus3"
FROM [AQA REG All Fld Types]
```

About Adding Hyperlinks to a Query

For each column in a query, you can configure one or more URLs to display as hyperlinks in the query results. When a user runs the query, the APM system will build the URLs as needed, passing data from the query results into the URL parameters if necessary, and will display in the query results hyperlinks that the user can select to access the associated feature or perform the associated function.

For example, you might define a URL on a field to an external website. When a user runs the query, the APM system will build a hyperlink from that URL and display the link to the user in the query results. The user will be able to select the hyperlink to open the website in a new browser tab.

If you configure only one URL for a field, when you run the query, that URL will be used to build a single hyperlink in that cell of the query results. If you have not modified the **Field** cell, whatever values are found in the database for that column will serve as the text for each hyperlink. For example, if an Asset ID column contains a URL to open a record in the Record Manager, when you run the query, the actual Asset IDs retrieved by the APM system will appear as hyperlinks in the results.

Reference Information: Query Expressions, Clauses, Prompts, and Operators

About the Expression Builder Window

The **Expression Builder** window contains various fields and controls to assist you in constructing an expression for your query criteria.

The **Expression Builder** window is divided into two sections, **Simple** and **Advanced**. You can toggle between the two sections by using the **Simple** and **Advanced** tabs at the top of the window.

If you access the **Expression Builder** window from the **Field** cell, the **Advanced** section is selected by default, and the **Simple** button is disabled. If you access the **Expression Builder** window from the **Criteria** cell or the **Or** cell, the **Simple** section is selected by default, and the **Advanced** button is enabled.

Simple Section

The **Simple** section of the **Expression Builder** window lets you define a simple expression using conditions on the field from which you accessed the **Expression Builder** window.

The **Simple** section of the **Expression Builder** window contains the following features:

- **And/Or**: A list that appears for all rows except the first. You can use the **And** and **Or** options to establish relationships between expressions.
- **Field** : A read-only text box that displays the value in the Field cell for the field you selected.
- **Operator**: A list that displays the valid operators to use in the expression, depending on the type of field on which the expression is built.

Field type	Options
Text -or- Character	equals
	not equals
	contains
	does not contain
	starts with
	ends with
	is null
	is not null
Numeric	equals
	is at most
	is at least
	not equals
	is less than
	is greater than
	is null
	is not null
Date	on
	at
	at or before
	at or after
	not at
	before
	after
	is null
	is not null
Logical	is true
	is false
	is null
	is not null

- **Value**: A text box where you can enter the value you want to include in the expression.
- **Add Condition** : A button that displays another row of options which you can use to create additional expressions on the field.

Advanced Section

The **Advanced** section of the **Expression Builder** window is divided into five main sections, as shown in the following image. These labels correspond to the numbered list following the image.
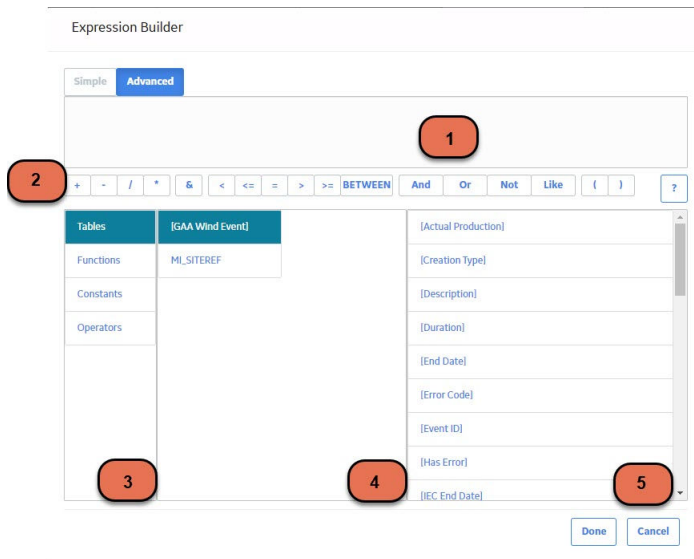
1. A text box that displays the expression itself. To build the expression manually, you can enter text directly into the text box. Otherwise, the expression will be built dynamically when you select tabs and values in the **Expression Builder** window.
2. Buttons that insert symbols to establish a relationship between parts of an expression, or that insert symbols to group those parts. Selecting a button will insert the corresponding symbol into the expression.

   Note:

   - If you are using date to build an expression, note that date can be entered in any of the acceptable formats. However, always be sure to select the date in the expression, and then select the **Date** button. Selecting the **Date** button converts the date in the expression to yyy-mm-dd format.

   - The [?] button opens the **Prompt Settings** section of the **Expression Builder** window, which has options that let you add a prompt to a query.
3. A list of tabs that you can use to toggle between categories for the valid components of the expression, which can include the following:

   - **System Codes**: This option appears only if a System Code Table Valid Values list has been defined for the selected field in the Rules Editor (i.e., if no Valid Values list has been defined, if the Valid Values list uses a static list of values, or if the Valid Values list is By Rule, this option does not appear). If this option appears and you select it, list **5** displays the following:

     ◦ All the codes and descriptions for the appropriate System Code Table if the Valid Values rule is defined as System Code Table Only.
     ◦ Only the codes and descriptions that meet the literal value criteria if the Valid Values rule is defined as System Code Table with Literal Reference. Additionally, if a literal reference has been defined, the reference value will appear in parentheses beside the table ID in list **4**.

- All the codes and descriptions for the appropriate System Code Table if the Valid Values rule is defined as **System Code with Field Reference** (i.e., in the absence of a field value to use as the reference, there is no way to limit the list).

  Note: When you select a System Code to include in your expression, the System Code ID (rather than the description) will be used. Even though System Code descriptions appear in the datasheet, the IDs are actually stored in the database. By using the ID instead of the description in the expression, your criteria will apply to all System Codes, including those with different, translated descriptions.

- **Tables**: The database tables from which you can select fields to include in the expression. When you select the **Tables** option in list **3**, list **4** displays the query source families, and list **5** displays all the fields in those families.

- **Functions**: The three types of functions that you can include in the expression. When you select a function category in list **3**, list **4** displays the types of functions in that category, and list **5** displays all the available functions for that category.

- **Constants**: The constants that you can include in the expression. A constant is a static value provided for comparative reasons. When you select the **Constants** option in list **3**, and list **5** displays the available constants.

- **Operators**: The symbols that you use to join parts of the expression. When you select the **Operators** option in list **3**, list **4** displays different classifications of operators, and list **5** displays all the valid operators.

- A list that displays a subset of options based on your selection in list **3**. Selecting an item in this list limits the options in list **5**.

- A list that displays a subset of values based on your selections in lists **3** and **4**. Selecting any item in this list will insert the value into the expression. The value will be inserted wherever the cursor is currently positioned.

What is an Expression?

An expression is a string of characters that, together, define a certain set of conditions to be applied to a query. In other words, an expression is the code that APM reads in order to determine what you want to retrieve from the database and how you want to display the query results.

Details

You can build simple expressions that limit the query results based on criteria that is applied to a single field, or you can build complex expressions that can be used to perform calculations, reformat stored values, concatenate stored values, and so on. As long as you understand the stored data and the way in which you want to present it to users, you can construct expressions to perform simple to very complex operations on the query data.

You can think of an expression as the combination of any of the following items that together define the conditions by which you want to limit the query results:

- Functions
- Clauses
- Operators
- Any other data required to yield results

In the grid in the **Conditions** section, you can construct an expression in the **Criteria** cell, the **Or** cell, and the **Field** cell. Expressions can be constructed manually, or using the **Expression Builder** window, which is accessible from the **FieldField**, **Criteria**, and **Or** cells of the grid in the **Conditions** section.

When building expressions, you must use the base (i.e., stored) values for Units of Measure, formatted values, and translated strings. For example, consider a query that contains an expression in a field that stores numeric values in inches. To filter your query results on that field, you must specify values as they are stored in that field (i.e., in inches) rather than as they are displayed when the query is run in formatted mode (e.g., in centimeters).

For instance, to return only the records in which 25 inches has been recorded in this field, your query expression must contain the value 25, not 63.5, which is the stored value converted to centimeters, and which might be displayed to some users. To determine how you should construct values in your expressions, you can run the query in unformatted mode.

Example: Expressions using Functions

The following text is an example of an expression:

```
DECODE([Air Cooled Heat Exchanger].[Asset Status],
'Active', 'A', 'Inactive', 'I', 'No Status')
```

This expression combines the DECODE function, the Asset Status field, and additional data that indicates that a stored value of Active should return the value A, a stored value of Inactive should return the value I, and any other stored value (including null values) should return the value No Status.

Example: Expressions Within Clauses

Expressions can exist within SELECT statements, WHERE clauses, or HAVING clauses, or they can exist outside of these SQL components. For example, consider the following WHERE clause:

```
                              WHERE [Asset].[ASSET MANUF CHR]
 = 'GOULDS'
```

The expression **[Asset].[ASSET MANUF CHR] = 'GOULDS'** is contained within the WHERE clause.

About Formatted Expressions on Character Fields

This topic contains syntax suggestions and requirements for expressions on character fields, as well as details about how APM reformats expressions on character fields.

When you create an expression on a character field, the actual field value(s) must be within single quotation marks.

Example: Expression Syntax on Character Fields

If you want to return records where the Asset Status field contains the value Active, the syntax would be:

```
WHERE [Asset].[ASSET STAT CHR] = 'Active'
```

...where **Active** is within single quotation marks because it is the stored value that you want to use for limiting the query results.

If you want to search on multiple values, such as Centrifugal Pump and Rotating Pump, you would again place the actual values within single quotation marks. The syntax would be:

```
WHERE ([Asset].[ASSET TYPE CHR] = 'Centrifugal Pump' OR
[Asset].[ASSET TYPE CHR] = 'Rotating Pump')
```

...where **Centrifugal Pump** and **Rotating Pump** are within single quotation marks because they are the actual values to be used in the query criteria.

Note: When you are using an Oracle schema, the value is case sensitive by default. This means, for example, that if you enter **WHERE_[Asset]. [ASSET_STAT_CHR]_=_'ACTIVE'** for a field where values are stored in the database as **Active**, the query will not return any results.

About Automatic Reformatting on Character Fields

If you enter something in the **Criteria** cell or the **Or** cell of the grid in the **Conditions** section of the **Design** workspace for a character field, and you do not enter single quotation marks yourself, APM will insert them automatically for you. For example, if you want to search for records where the Asset Status is Active, you can enter either `'Active'` or `Active`, APM will insert the single quotation marks automatically, reformatting the entered text as **'Active'**.

If the expression contains multiple words, or if you want to use operators other than is equal to, and you do not enter the single quotation marks yourself, APM will insert the single quotation marks around the entire phrase, including the operators. Depending on the values stored in the database, this may or may not return the appropriate results.

For example, suppose you create a query on the Shell and Tube Heat Exchanger family, and you add the Asset Manufacturer field to the grid in the **Conditions** section. If there are five different manufacturers of Shell and Tube Heat Exchangers, but you want to return Shell and Tube Heat Exchangers manufactured by only two of those manufacturers, Alco and Whitlock, the query expression on the Manufacturer field must be formatted as follows:

```
='Alco' or 'Whitlock'
```

If you enter `Alco` or `Whitlock` in the **Criteria** cell for the Asset Manufacturer field, APM will reformat the expression as 'Alco or Whitlock'. The query will not return any results because the syntax suggests that you want to find Shell and Tube Heat Exchangers whose manufacturer is Alco or Whitlock, which does not exist in the database.

Likewise, if you enter `=Alco` or `Whitlock`, APM will reformat the expression as **='Alco' or 'Whitlock'**, which, again, would not return any results because there is no manufacturer named =Alco or Whitlock in the database.

To return the appropriate results, you should enter `'Alco' or 'Whitlock'` in the **Criteria** cell, or enter `Alco` in the **Criteria** cell and `Whitlock` in the **Or** cell.

Note: If you enter an entirely numeric value (e.g., 123) or an expression that can be interpreted as a mathematical equation (e.g., 123-12), APM will not insert single quotation marks automatically. This will cause the expression to be invalid, and when you run the query, the system will display an error. To resolve this problem, add the single quotation marks manually.

About Formatted Expressions on Text Fields

This topic contains syntax suggestions and requirements for expressions on text fields, as well as details about how APM reformats expressions on text fields.

When you create an expression on a text field, the value(s) within the expression must be placed within single quotation marks.

> **Example: Expression Syntax on Text Fields**
>
> If you want to create an expression for the Asset Description field to return the records where the field contains the value This is a test, the expression would be:
>
> ```
> WHERE [Asset].[ASSET DESC CHR] = 'This is a test'
> ```
>
> ...where `This is a test` is within single quotation marks because it is the stored value that you want to use to limit the query results.
>
> Note that text fields are stored in the database differently than character fields and are, therefore, handled differently. One difference is that you cannot use the = operator with text fields. Instead, you must use the like operator. When creating text field expressions, be sure to specify the like operator with Oracle. The = operator works for other databases.
>
> **Note:** When you are using an Oracle schema, the value is case sensitive by default. This means, for example, that if you enter **WHERE [Asset].[ASSET DESC CHR] LIKE 'This is a test'** in a field where values are stored in the database as **This is a test**, the query will not return any results.

About Automatic Reformatting on Text Fields

If you enter something for a text field in the **Criteria** cell or the **Or** cell of the grid in the **Conditions** section, and you do not enter single quotation marks yourself, APM will insert them automatically for you. For example, if you want to search for records where the Asset Additional Information field contains the text This asset exists for testing purposes, you can enter the phrase with or without the single quotation marks. If you omit the single quotation marks, APM will insert them for you.

Note: While APM will insert the single quotation marks automatically, it will not insert the like operator. When creating text field expressions on an Oracle database, be sure to specify the like operator. Otherwise, APM will assume the is equal to (=) operator and the query will return an error.

About Formatted Expressions on Date Fields

This topic contains syntax suggestions and requirements for expressions on date fields, as well as details about how APM reformats expressions on date fields.

When you create an expression on a date field, you should use the following syntax:

```
(# :D 'yyyy-mm-dd')
```

Note: You will need to use the syntax **yyyy-mm-dd** on all workstations, regardless of your APM Culture setting or your Windows Regional and Language options. Even if we input the date in the mm-dd-yyyy format, the expression should still use the yyyy-mm-dd format. However, always be sure to select the date in the expression, and then select the **Date** button. Selecting the **Date** button converts the date to yyyy-mm-dd format.

Example: Expression Syntax on Date Fields

If you want to create an expression on the Asset Installation Date field to return the records where the field contains the value 05-04-2005 (where 05 represents May, and 04 represents the fourth day of the month), the expression would be:

```
                              WHERE [Asset].[ASSET INSL DT] =
 (# :D '2005-05-04')
```

About Formatted Expressions on Logical Fields

This topic contains syntax suggestions and requirements for expressions on logical fields, as well as details about how APM reformats expressions on logical fields.

A logical field represents a value of True or False. When you create an expression for a logical field, the exact syntax to use is either **'Y'** (True) or **'N'** (False).

Example: Expression Syntax on Logical Fields

If you want to create an expression on the Spared field to return records where the check box is selected (the value is True), the expression would be:

```
WHERE [Asset].[ASSET SPRD IND] = 'Y'
```

About Automatic Reformatting on Logical Fields

Some common syntax options that APM will reformat to match the syntax required for logical fields are:

- **True** (using any case combination, such as true or TRUE)
- **False** (using any case combination, such as false or FALSE)
- **Yes** (using any case combination, such as yes or YES)
- **No** (using any case combination, such as no or NO)

About Formatted Expressions on Numeric Fields

This topic contains syntax suggestions and requirements for expressions on numeric fields.

Details

When you create an expression on a numeric field, you must use the exact numeric value that are stored in the database. Some numeric values may be displayed differently than they are stored. For example, numbers may be formatted to display a certain number of decimal places, to include a currency symbol, or to be converted to a different unit of measure (UOM).

To determine how values are stored in the database, run the query in unformatted mode. Then, format numeric values in your query expressions exactly as they are displayed in the unformatted query results.

Unlike character and text field expression values, which require single quotation marks, the actual numeric values that you want to return must not have single quotation marks in your expression. Specifying the value as it is stored is sufficient to return the expected results. For example, an acceptable way to express that returned records should contain a value of 5 in the Number of Storage Tanks field in the Criticality Analysis family is:

```
WHERE [Criticality Analysis].[No of Storage Tanks] = 5
```

Expressions in the Field, Criteria, and Or Cells

You can build expressions in the **Field**, **Criteria**, and **Or** cells in the grid within the **Conditions** section in the **Design** workspace. This topic includes formatting tips for creating expressions in each of those cells.

Expressions in the Field Cell

When you access the **Expression Builder** window from the **Field** cell, the **Advanced** tab is selected by default, and the **Simple** tab is disabled. You can construct an expression on the **Field** cell to reformat the results that are returned by the query.

For example, suppose that you want to display an installation date in two formats: the stored format and a modified format. In this case, you would add the Installation Date field to the query twice. The first would return the stored value, and the second could contain a Date function in the **Field** cell to indicate how you want the date to appear in the results.

If you add an expression to the **Field** cell, you must enter the exact syntax that is required in order to run the query. APM will not reformat any text that you enter in the **Expression Builder** window for a **Field** cell.

Expressions in the Criteria Cell

When you access the **Expression Builder** window from the **Criteria** cell, the **Simple** tab is selected by default, but the **Advanced** tab is enabled. You can construct an expression in the **Criteria** cell to limit the results that are returned by the query.

Note: Anything you add to the **Criteria** cell will have the text WHERE or HAVING appended in front of the expression in the SQL code. WHERE and HAVING do not appear in the grid in the **Conditions** section.

For example, suppose that you want to return only those pieces of equipment manufactured by GOULDS. In this case, you would add the Manufacturer field to the query. You can then construct an expression in the **Criteria** cell using either the **Simple** or **Advanced** section.

If you access the **SQL** workspace, you will see the following code:

```
SELECT [MI_EQUIP000].[MI_EQUIP000_MFR_C] "Manufacturer"
FROM [MI_EQUIP000]
WHERE [MI_EQUIP000].[MI_EQUIP000_MFR_C] = 'GOULDS'
```

The expression [MI EQUIP000].[MI EQUIP000 MFR C] = 'GOULDS' is inserted automatically into the WHERE clause at the end of the SQL code.

Note: When you construct an expression in the **Criteria** cell, you can enter the exact syntax that is required to run the query, or you can enter something that is close to the required syntax, and then let the APM system reformat it automatically.

Constructing an expression in the Simple section

In the **Simple** section of the **Expression Builder** window, you could use the options to construct an expression that looks something like the following image:

Constructing an expression in the Advanced section

In the **Advanced** section of the **Expression Builder** window, you could construct an expression that looks something like the following code:

```
'GOULDS'
```

While this text alone does not constitute an entire expression, APM interprets the data in the remaining cells to construct a valid expression. The equal (=) operator is understood, and the **Field** and **Table** cells indicate the locations from which you want to retrieve data.

Expressions in the Or Cell

When you access the **Expression Builder** window from the **Or** cell, the **Simple** tab is selected by default, but the **Advanced** tab is enabled. You can construct an expression in the **Or** cell to limit the results that are returned by the query.

Note: Anything you add to the **Or** cell will have the text WHERE or HAVING appended in front of the expression in the SQL code. WHERE and HAVING do not appear in the grid in the **Conditions** section.

For example, suppose you want to return only those pieces of equipment manufactured by JENSEN or WESTERN SUPPLY. In this case, you would add the Manufacturer field to the query. You can then construct an expression in the **Or** cell using either the **Simple** or **Advanced** section.

If you access the **SQL** workspace, you will see the following code:

```
SELECT [MI_EQUIP000].[MI_EQUIP000_MFR_C] "Manufacturer"
 FROM [MI_EQUIP000]
 WHERE ([MI_EQUIP000].[MI_EQUIP000_MFR_C] = 'JENSEN'
 OR [MI_EQUIP000].[MI_EQUIP000_MFR_C] = 'WESTERN SUPPLY')
```

The expression [MI EQUIP000].[MI EQUIP000 MFR C] = 'JENSEN' OR [MI EQUIP000].[MI EQUIP000 MFR C] = 'WESTERN SUPPLY' is inserted automatically into the WHERE clause at the end of the SQL code.

Note: When you construct an expression in the **Criteria** cell, you can enter the exact syntax that is required to run the query, or you can enter something that is close to the required syntax, and then let the APM system reformat it automatically.

Constructing an expression in the Simple section

In the **Simple** section of the **Expression Builder** window, you could use the options to construct an expression that looks something like that shown in the following image:

Constructing an expression in the Advanced section

In the **Advanced** section of the **Expression Builder** window for the **Criteria** cell, you could construct an expression that resembles the following code:

```
'JENSEN'
```

Then, you could construct an expression in the **Advanced** section of the **Expression Builder** window for the **Or** cell that resembles the following code:

```
'WESTERN SUPPLY'
```

While this text alone does not constitute an entire expression, APM interprets the data in the remaining cells to construct a valid expression. The equal (=) operator is understood, and the **Field** and **Table** cells indicate the locations from which you want to retrieve data.

About the WHERE Clause

A WHERE clause defines conditions that you want to apply to a query.

Details

WHERE clauses are used to define conditions that you want to apply to the query to limit the results.

> **Example: WHERE Clause**
>
> You might want to calculate the total failure cost for all pieces of equipment, but you want to include in the calculation only failures with a total failure cost greater than $5,000.00. You could create a query like this:
>
> ```
> SELECT [Asset].[ASSET_ID_CHR] "Asset ID", Sum([Failure].
> [EFAIL_TOTCST_FRM]) "Total Failure Cost"
> FROM [Asset] JOIN SUCC [Failure] ON {Asset Has Failure}
> WHERE [Failure].[EFAIL TOTCST FRM] > 5000
> GROUP BY [Asset].[ASSET ID CHR]
> ```
>
> In this query, you can see that the WHERE clause is:
>
> ```
> WHERE [Failure].[EFAIL TOTCST FRM] > 5000
> ```

This WHERE clause returns the total failure cost for all pieces of equipment, but calculates only the failures with a total failure cost greater than $5,000.00.

A piece of equipment might have failures whose failure costs were $5,050.00, $1,000.00, and $500.00. Because the WHERE clause indicates that you want to return the total failure cost for all pieces of equipment but only calculate the failures with a total failure cost greater than $5,000.00, the total failure cost returned for this piece of equipment would be $5,050.00 (the failures with a cost of $1,000.00 and $500.00 are not included in the calculation because they are less than $5,000.00 each).

About the HAVING Clause

A HAVING clause defines conditions that you want to apply to an aggregate query.

Details

In an aggregate query, HAVING clauses are used to define conditions that you want to apply to each aggregate value after the calculation dictated by the aggregate function has been performed.

> **Example: HAVING Clause**
>
> You might want to see all pieces of equipment whose failures resulted in a total failure cost greater than $5,000.00. To do so, you could create a query like this:
>
> ```
> SELECT [Asset].[ASSET_ID_CHR] "Asset ID", Sum([Failure].
> [EFAIL_TOTCST_FRM]) "Total Failure Cost",
> Count([Failure].[MI_EVENT_ID]) "Failure Count"
> FROM [Asset] JOIN SUCC [Failure] ON {Asset Has Failure}
> GROUP BY [Asset].[ASSET ID CHR]
> HAVING Sum([Failure].[EFAIL TOTCST FRM]) > 5000
> ```
>
> In this query, you can see that the HAVING clause is:
>
> ```
> HAVING Sum([Failure].[EFAIL TOTCST FRM]) > 5000
> ```
>
> This HAVING clause returns pieces of equipment whose failures resulted in a total failure cost greater than $5,000.00.
>
> A piece of equipment might have failures whose failure costs were $3000.00, $1,000.00, and $500.00. If you add these values together, you can see that the total failure cost for failures associated with the piece of equipment is $4,500.00. Because the HAVING clause indicates that you want to return only pieces of equipment with a total failure cost greater than $5,000.00, this piece of equipment would not be returned.

About Prompts on Queries

You can construct a query that will, when run, prompt the user to enter or select values by which the results will be filtered.

Details

You can create a prompt on a character, numeric, date, or logical field in any type of query (Select, Crosstab, Append, Update, or Delete). Like all query options, you can construct prompts by modifying the SQL code directly, but APM provides the **Prompt Settings** section of the **Expression Builder** window, which guides you step-by-step through the process of creating prompts. This documentation focuses primarily on that feature.

Note: There are special considerations to take into account when you create a prompt on a date field.

When a user runs a query by opening it from the catalog, prompts appear in a window that disappears when the query results are returned.

Example: Limiting Results Based on the Manufacturer

If you create a prompt on the Asset Manufacturer field to limit the query results based on the manufacturer, when the query is run, a window appears, prompting the user to supply the desired value for the manufacturer.

About the Prompt Settings Section

Using the **Prompt Settings** section of the **Expression Builder** window, you can define the basic settings for the prompt that you want to create.

Prompt Settings

The following table includes details on the available settings on the prompt builder. After you have defined the prompt settings you want, you can select **Next** to further define the prompt. The content that appears on subsequent screens in the Prompt Builder depends on your selection in the **Valid Values** section. These screens are documented in more detail in the topics that explain how to create specific types of prompts.

Setting	Description	Notes
Prompt Caption	A label that will indicate to the user what type of value to enter or select for the prompt. The prompt caption will appear on the **Enter parameter values** window to identify the prompt.	Prompt captions are optional. • If you do not provide a prompt caption, the prompt ID will be displayed on the **Enter parameter values** window, where spaces are replaced with underscores. For example, a prompt ID of Task Type would be displayed as Task_Type. • If you do not specify a prompt caption or a prompt ID, the text Enter Parameter Value will be displayed on the **Enter parameter values** window.
Prompt Data Type	A property that identifies type of data that exists in the field at the time the query is run. Expressions may exist that convert the stored value to a runtime value (e.g., numeric values may be converted to character values or strings). You will want to choose a data type for the prompt that is appropriate for the runtime value. When you access the Prompt Builder, this setting will be set by default to the data type that corresponds to the stored value of the field from which you accessed the Expression Builder. You may need to change the default setting. The following options are available: • **Logical**: For logical values (i.e., True and False). • **Character**: For character values. • **Number**: For numeric values. • **Date/Time**: For date values. • **Key**: For key types (for example, ENTY_KEY, FMLY_KEY)	• When you create a prompt that allows multiple selections, you can select multiple default values for that prompt. Multiple default selections will be allowed only when the **Character** option is selected for this setting. • For prompts on numeric fields, the corresponding UOM will be displayed in the **Enter parameter values** window for that field if the **Number** option is selected for this setting. If you select any other option, the UOM will not be displayed. • The **Logical** option is enabled only if you accessed the Prompt Builder from a Logical field. • The **Ignore Time** check box is enabled only if you select the Date/Time data type. If you select the **Ignore Time** check box, the prompt will require only that you select a date. If you do not select this check box, the prompt will require that you select both a date and time.

Setting	Description	Notes
Prompt ID	A unique, alphanumeric ID for the prompt. This value is used internally by APM to identify the prompt.	The Prompt ID is required.
Valid Values	An option that defines the type of user-input value that will be required for the prompt. You can require users to specify a value manually, or you can present them with a list of valid options. Specifically, you have the following options for determining user-input values: • **No Valid Values:** The prompt will not offer a list of values. Rather, the prompt will display a text box into which users can type the value by which they want to limit the results. • **Static List of Values:** The prompt will display a static list of values that you define specifically for the prompt. • **Values from a System Code Table:** The prompt will display a list of values that you define specifically for the prompt. • **Values from a Query:** The prompt will display a list of values taken from the query results of a query that you select. • **Distinct List of Values From [X]:** The prompt will display a list of values pulled from field X of all records in a given family.	When the **Logical** option is selected for the **Prompt Data Type** selection, the Valid Values options will be disabled, and a list of valid values will be created automatically, providing the options True, False, and All (i.e., both True and False) when the query is run.

About Prompts on Date Fields

All date fields contain both a date and time value. When you apply specific criteria to a field, you can use the DAYOF operator and omit the time to return all the records that contain the specified date and any time.

Details

For example, specifying the criteria =DayOf(#:D '2000-03-10') would return records from any time in the same day, for example, 12:00:00 A.M., 2:00:00 P.M., 5:00:17 P.M., or any other time. You can use the DAYOF operator with prompts as well: DayOf(? :d :id=mydate).

Otherwise, you will need to know how to enter a date that will retrieve the desired results:

• If you enter a date with no time, the query will return records where the time is 12:00 A.M. Records that contain any other time will not be included in the results.

- If you enter a date and a specific time, the query will return records that contain that specific date and time.

  Note: You must specify the seconds value as 00. Entering any other value will cause the query to not return any results.
- If you use the Calendar to select a date, you will need to modify the time manually in order to include seconds, which are required to return results. Either enter a different time, or delete the existing time (to use a time of 12:00:00 A.M.).

All dates and times in APM are stored in UTC format, and, the time values will be displayed using the appropriate conversion from UTC to the time zone that is associated with the Security User who is logged in when the query is run. This allows any user who runs the query to provide a local date and time in the prompt.

About Prompts on Numeric Fields

Units of measure (UOMs) can be associated with numeric fields. UOM Conversion Sets can be defined on fields, and will convert stored numeric values to different values and UOMs.

Details

If a query has a prompt on a numeric field, when the query is accessed via the Catalog or a URL, the associated UOM will appear on the **Enter parameter values** window. For example, you would define a prompt on the Measurement Value field of the Thickness Measurement family to display returned values in inches.

Note: This behavior applies only to prompts on fields where the stored data type is numeric, has a UOM defined, and is not converted at runtime. If a numeric field is converted to a character field at runtime, the UOM will not appear. Similarly, if a character field is converted to a numeric field at runtime, no UOM will appear.

If you run a query in formatted mode, the **Enter parameter values** window will display the appropriate UOM, based on the UOM Conversion Set that is associated with your Security User account. In order for the query to return results, you will need to enter numeric values associated with the UOM that is displayed.

If you run a query in unformatted mode, the **Enter parameter values** window will display the stored values, and the base UOM will always be displayed.

About Configuring a Prompt to Accept a Percent Wildcard

Query prompts can help users limit the results that will be returned by the query. In some cases, however, users might not know what to select or type in a query prompt. This might be especially true if the prompt does not present a list of values and the user must enter the value as it is stored in the database in order to return any results.

To address this concern, you can configure a query prompt to accept the percent (%) wildcard. When a user enters only a percent symbol in the prompt text box, the query will return records where the field on which the prompt was built contains any value. When a user enters a combination of text and the percent symbol, the query will return records where the field on which the prompt was built contains a value that contains the specified text, preceded or followed by any combination of characters (depending on where the percent symbol is placed in the prompt text box).

To configure a prompt to accept a percent wildcard, you must use the Like operator instead of the equal (=) operator in the expression that defines the prompt.

For example, suppose you build a prompt on the Asset Manufacturer field with no list of valid values. The expression for the prompt would look like this:

```
                        [Asset].[ASSET MANUF CHR] =
(? :s :caption='Manufacturer' :id=Manufacturer)
```

If you want users to be able to enter a percent symbol in the prompt to return records where the Asset Manufacturer field contains any value, you can replace the equal operator (=) with the Like operator. The expression for the prompt would look like this:

```
                        [Asset].[ASSET MANUF CHR] Like
(? :s :caption='Manufacturer' :id=Manufacturer)
```

When you run the query, you could enter the percent symbol (%) in the **Manufacturer** box on the prompt window. In this case, the results would display records where the Asset Manufacturer field contains any value.

Additionally, you could enter SEI%, and the query would display records where the value in the Asset Manufacturer field begins with the letters SEI, followed by any combination of characters, such as SEIMENS-AL and SEIGER.

About Configuring a Prompt to Return Null Values

In some cases, records might contain fields that do not contain values. If you want to return a query with a prompt on these fields to see all records where the field is empty, you must construct the query in a certain way.

This query construction is best understood through an example. The following example assumes that you want to return records where the Asset Description field of Air Cooled Heat Exchanger records is empty. It also assumes that you want to see the Asset ID, Asset Description, and Asset Installation Date of the returned records.

To configure a prompt to return records where the Asset Description field is empty, you would need to add the following fields to the query:

- Asset ID
- Asset Description
- Asset Installation Date

You would need to add two expressions to the Asset Description field. In the **Criteria** cell, you would need to add the prompt **(? :s :caption='Description' :id=Description)**, and in the **Or** cell, you would need to add the expression **IS NULL**.

The design grid would look like the following image:

Field	Limit Results	Area Internal ID	▼ ..	Area Description	▼ ..	Asset Group
Alias		Area Internal ID	🌐	Area Description	🌐	Asset Group
Table		Functional Location	▼	Functional Location	▼	Equipment
Sort		None	▼	None	▼	None
Sort Index		0		0		0
Include		☑		☑		☑
Display		☑		☑		☑
Hyperlink			..		..	

When the query is run, the **Description** box appears in the prompt window. If a user selects **Done** without entering a value in the prompt, the results will display records where the Asset Description field is empty, like the ones in the following image:

ASSET ID	ASSET DESCRIPTION	ASSET INSTALLATION DATE
Exchanger-46-Tubeside	-	-
Exchanger-5A-Tubeside	-	-

If you enter a value in the **Description** box, the query will return the matching records and the records where the **Asset Description** box is blank.

About Operators to Use with Character Fields

This topic describes the operators that can be used to manipulate values in character fields.

Details

When you enter anything in the **Criteria** cell or the **Or** cell, the is equal to (=) operator is assumed at the beginning of the text unless a different operator is specified.

It is not necessary to enter the = operator at the beginning of the expression in a given cell. If you do not enter the operator, will insert it automatically, but it will not be displayed.

Available Operators for Character Fields

The following table provides examples of operators that you can use to manipulate values returned in character fields.

Purpose	Operator	Example	Outcome of Example
Return records that have an exact value stored in a given field.	=	='Rotating Pump'	Returns records where the field contains the exact value Rotating Pump. (See note)
Return records that have one value or another stored in a single field.	Or	='Rotating Pump' or 'Centrifugal Pump'	Returns records where the value stored in the field is Rotating Pump, and records where the value stored in the field is Centrifugal Pump.
Returns records that do not have a specified value in a given field.	Not != <>	Not 'Centrifugal Pump' != 'Centrifugal Pump' <> 'Centrifugal Pump'	Returns records where the field does not contain the value Centrifugal Pump.
Returns records where a given field contains the specified value.	Like	Like '%Pump'	Returns records where the specified field contains the term Pump, preceded by any number of characters, such as Centrifugal (Centrifugal Pump) or Rotating (Rotating Pump).

Purpose	Operator	Example	Outcome of Example
Returns records where a given field contains the specified value.	Like	Like 'Pump_'	Returns records where the specified field contains the term Pump, followed by any one character, such as 1 (Pump1).
Returns records where a given field does not contain the specified value.	Not Like	Not Like 'Centrifugal%'	Returns records where the field value does not contain Centrifugal followed by any number of characters. For example, this would eliminate Centrifugal Pump, Centrifugal Pump A, and Centrifugal Pump1.
Returns records where a given field does not contain the specified value.	Not Like	Not Like 'Pump_'	Returns records where the field value does not contain Pump followed by any one character. For example, this would eliminate PumpA and Pump1.
Return records that contain any values in a given field.	Is Not Null	Is Not Null	Returns records where the field contains any value.
Returns records that do not contain any values in a given field.	Is Null	Is Null	Returns records where the field is empty.

About Operators to Use with Text Fields

This topic describes the operators that can be used to manipulate values in text in an Oracle database. For SQL server and PostgreSQL, you can treat them the same as character fields.

Available Operators for Text Fields

The following table provides examples of operators that you can use to manipulate values returned in character fields.

Purpose	Operator	Example Expression	Outcome of Example
Return records where a given field contains the specified value.	Like	Like '%Pump'	Returns records where the field contains the term Pump, preceded by any number of characters, such as Centrifugal (Centrifugal Pump) or Rotating (Rotating Pump).
Return records where a given field contains the specified value.	Like	Like 'Pump_'	Returns records where the field contains the term Pump, followed by any one character, such as 1 (Pump1).
Return records where a given field does not contain the specified value.	Not Like	Not Like 'Centrifugal%'	Returns records where the field value does not contain Centrifugal followed by any number of characters. For example, this would eliminate Centrifugal Pump, Centrifugal Pump A, and Centrifugal Pump1.
Return records where a given field does not contain the specified value.	Not Like	Not Like 'Pump_'	Returns records where the field value does not contain Pump followed by any one character. For example, this would eliminate PumpA and Pump1.
Return records that contain any values in a given field.	Is Not Null	Is Not Null	Returns records where the field contains any value.
Return records that do not contain any values in a given field.	Is Null	Is Null	Returns records where the field is empty.

About Operators to Use with Date Fields

When you enter anything in the **Criteria** cell or the **Or** cell, the is equal to (=) operator is assumed at the beginning of the text unless a different operator is specified.

It is not necessary to enter the = operator at the beginning of the expression in a given cell. If you do not enter the operator, will insert it automatically, but it will not be displayed.

Available Operators for Date Fields

The following table provides examples of operators that you can use to manipulate values returned in date fields.

Purpose	Operator	Example	Outcome of Example
Return records that have an exact date stored in a field.	DAYOF=	=(#:D '2000-03-10')	Returns records where the date is March 10, 2000 and the time is any time.
Return records that have an exact date and time stored in a field.	=	=(# :dt '2006-01-01 17:00:00')	Returns records where the date is January 1, 2006 and the time is 5:00 P.M.
Return records that have a date greater than the specified date in a given field.	>	>(#:D '2000-03-10')	Returns records where the date is after March 10, 2000.
Return records that have a date less than the specified date in a given field.	<	<(#:D '2000-03-10')	Returns records where the date is before March 10, 2000.
Return records that have a date greater than or equal to the specified date in a given field.	>=	>=(#:D '2000-01-10')	Returns records where the date is on or after March 10, 2000.
Return records that have a date less than or equal to the specified date in a given field.	<=	<=(#:D '2000-03-10')	Returns records where the date is on or before March 10, 2000.
Return records that do not have the specified date in a given field.	Not !=	Not (#:D '2000-03-10') != (#:D '2000-03-10')	Returns records where the date is not March 10, 2000.
Return records that contain any values in a given field.	Is Not Null	Is Not Null	Returns records where the field contains any date.
Return records that do not contain any values in a given field.	Is Null	Is Null	Returns records where the field is empty.
Return records that have a date that falls within a certain range with respect to the current date.	Now()+ Now()-365	Now()-365	Returns records where the date is one year before the current date. The numeric value can be any number of days before or after the current date (see note).
Return records falling within a defined range of dates.	BETWEEN	BETWEEN (? :d :caption='Start Date' :id=startdate) AND (? :d :caption='EndDate' :id =enddate	Returns records whose dates fall within the defined range.

About Operators to Use with Logical Fields

This topic describes the operators that can be used to manipulate values in logical fields.

Details

When you enter anything in the **Criteria** cell or the **Or** cell, the is equal to (=) operator is assumed at the beginning of the text unless a different operator is specified.

It is not necessary to enter the = operator at the beginning of the expression in a given cell. If you do not enter the operator, will insert it automatically, but it will not be displayed.

Available Operators for Logical Fields

The following table provides examples of operators that you can use to manipulate values returned in logical fields.

Purpose	Operator	Example	Outcome of Example
Return records that have an exact value in a given field.	=	='Y''	Returns records where the field value is True.
Return records that do not contain any values in a given field.	Is Null	Is Null	Returns records where the field is empty.
Return records that do not have a specified value in a given field.	Not <> !=	Not 'Y' <> 'Y' != 'Y'	Returns records where the field value is not True (i.e., False or Null).

About Operators to Use with Numeric Fields

This topic describes the operators that can be used to manipulate values in numeric fields.

Details

When you enter anything in the **Criteria** cell or the **Or** cell, the is equal to (=) operator is assumed at the beginning of the text unless a different operator is specified.

It is not necessary to enter the = operator at the beginning of the expression in a given cell. If you do not enter the operator, will insert it automatically, but it will not be displayed.

Available Operators for Numeric Fields

The following table provides examples of operators that you can use to manipulate values returned in numeric fields.

Purpose	Operator	Example	Outcome of Example
Return records that have an exact value in a given field.	=	=10	Returns records where the specified field value is 10.
Return records where the value in a given field is the sum of the specified values.	+	=8+2	Returns records where the specified field value is the sum of 8 and 2, or 10.

Purpose	Operator	Example	Outcome of Example
Return records where the value in a given field is the difference between the specified values.	-	=11-1	Returns records where the specified field value is the difference between 11 and 1, or 10.
Return records where the value in a given field is the outcome of the specified values after they are divided.	/	=20/2	Returns records where the specified field value is the outcome of 20 divided by 2, or 10.
Return records where the value in a given field is the outcome of the specified values after they are multiplied.	*	=5*2	Returns records where the specified field value is the product of 5 multiplied by 2, or 10.
Return records that have a value greater than the specified value in a given field.	>	>9	Returns records where the specified field value is greater than 9.
Return records that have a value less than the specified value in a given field.	<	<11	Returns records where the specified field value is less than 11.
Return records that have a value greater than or equal to the specified value in a given field.	>=	>=10	Returns records where the specified field value is greater than or equal to 10.
Return records that have a value less than or equal to the specified value in a given field.	<=	<=10	Returns records where the specified field value is less than or equal to 10.
Return records that do not have the specified value in a given field.	Not != <>	Not 10 != 10 <> 10	Returns records where the specified field value is not 10.
Returns records that have multiple values in a given field.	And	<5 And >1	Returns records where the specified field value is less than 5 and greater than 1.
Return records that have one value or another in a given field.	Or	5 Or 10	Returns records where the specified field value is 5 or 10.
Return records where a given field contains a value that results from a specified grouping.	()	(1000+1)-1	Returns records where the specified field value is equal to 1000 plus 1, or 1001, minus 1, which equals 1000.

Purpose	Operator	Example	Outcome of Example
Return records that contain any value in a given field.	Is Not Null	Is Not Null	Returns records where the field is not empty.
Return records that do not contain any values in a given field.	Is Null	Is Null	Returns records where the field is empty.
Return records that contain the specified values in a given field.	In	In (5, 10, 20)	Returns records where the specified value is 5, 10, or 20.
Return records that do not contain the specified values in a given field.	Not In	Not In (5, 10, 20)	Returns records where the specified field value is not 5, 10, or 20.
Return records falling within a defined range of numbers.	BETWEEN	BETWEEN (? :n :caption='Starting':id= start) AND (? :n :caption='Ending':id=e nd	Returns records whose values fall within the defined range.