

MODULE 12: QUERIES

Module Overview

Queries specify datasets from the Microsoft Dynamics® NAV 2013 database for retrieval in a fast and efficient way. You can use queries to retrieve data from one or more tables as a single flat result set. You can specify how to join multiple tables in the result set, and how to order, group, aggregate, and filter the resulting data.

Queries retrieve data efficiently because they are always translated into a single SELECT statement, and are executed against the underlying Microsoft SQL Server database. Data is selected, joined, grouped, ordered, aggregated, and filtered at the SQL Server level. This makes sure that there is a minimum effect on performance.

You can do the following with queries:

- Use them as sources of charts in a RoleTailored client.
- Save them as XML or CSV files.
- Access them from C/AL code.
- Publish them as OData web services for later consumption from other clients, such as Microsoft PowerPivot data analysis add-in in Microsoft Office Excel 2010.

Objectives

- Present the Query Designer and its features.
- Explain the principles of the query design process.
- Show how to select, join, filter, aggregate, and order data.
- Show how to access queries from C/AL code.
- Explain how to export data from queries.

Query Design

A *query* describes a dataset of Microsoft Dynamics NAV data. Queries retrieve records from one or more tables and combine the records into rows and columns in a single dataset. You create queries in the Microsoft Dynamics NAV 2013 Development Environment by using Query Designer.

Query Designer lets you model the query definition that Microsoft Dynamics NAV 2013 translates to a Transact-SQL statement. Microsoft SQL Server uses Transact-SQL statements to retrieve data.

Queries have the following capabilities:

- Select subsets of fields from multiple tables.
- Join tables with different linking criteria.
- Filter tables by specifying filtering criteria.
- Group and aggregate data.
- Order data.
- Limit the number of rows to retrieve.
- Apply date methods.

In Microsoft Dynamics NAV 2013, you can use queries for the following purposes:

- Structuring multiple tables into simple datasets for fast and efficient data access from C/AL code. For example, page 9126, **Lot Numbers by Bin FactBox**, uses query 7300, **Lot Numbers by Bin Code**, to populate the fact box with temporary data from the dataset.
- Defining data sources for charts in a RoleTailored client. For example, query 760, **Trailing Sales Order Qry**, joins the information from tables 36, **Sales Header**, and 37, **Sales Line**, to be shown in the **Trailing Sales Orders** part of the **Sales Order Processor** role center.
- Publishing as OData web services to be used as data sources for Microsoft PowerPivot data analysis add-in for Microsoft Office Excel 2010.
- Exporting the datasets as CSV or XML.

Query Designer

You model queries in the **Query Designer**. To model a query, you specify the tables from which to collect data and how to link them together to create the resulting dataset. You also specify the fields that should be included. You can perform many other operations, such as filtering, aggregating, and grouping.

Module 12: Queries

The **Query Designer** is tabular. Each row represents an element of a query, such as a column of a resulting dataset. For each row in the **Query Designer** window, you can define the following important properties.

Property	Description
Type	Determines the type of the row. It can be one of the following: <ul style="list-style-type: none">• DataItem• Column• Filter
Data Source	Specifies the source table of a data item, or the field from the source table for columns and filters.
Name	Specifies the name of the data item, column or filter, because it will be referred to in the C/AL code or through the OData web services. The name must comply with the Common Language Specification. This means that the first character must be a letter. Other characters can be any combination of letters, whole numbers and underscores.

The following “Query Designer Window” figure shows the query 760, Trailing Sales Order Qry, in the **Query Designer** window:

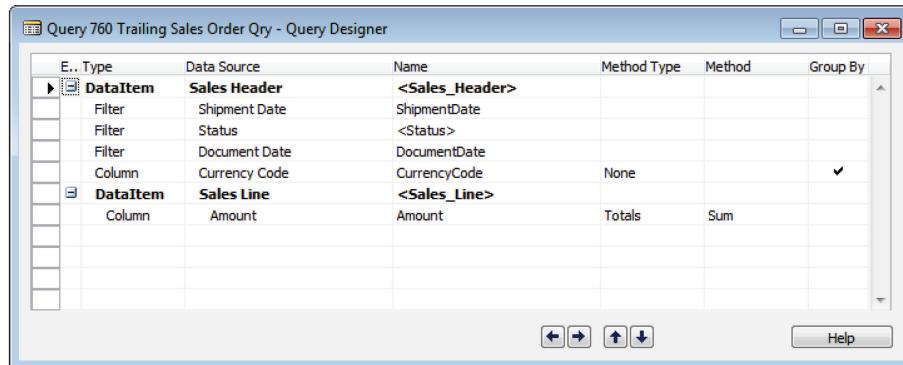


FIGURE 12.1: QUERY DESIGNER WINDOW

The definition of this query consists of the following:

- Data items – Tables **Sales Header** and **Sales Line**
- Columns – Fields **Currency Code** from the **Sales Header** table, and **Amount** from the **Sales Line** table
- Filters – Fields Shipment Date, Status, and Document Date from the Sales Header table.

When the query is executed, Microsoft Dynamics NAV 2013 translates its data model into a Transact-SQL SELECT statement and runs it against the underlying Microsoft SQL Server database.

Selecting Data

To select the data, you must define the tables from which you want to retrieve the data, and the fields that you want to include from those tables. When modeling a query in the **Query Designer**, a data item corresponds to a table; columns correspond to a field in the table.

Defining Data Items

To define a new data item, insert a new row, and set **Type** to **DataItem**. Then specify the source table in the **Data Source** column, by entering either the table ID or name, or by selecting the table from the list of available tables.

The Name of the data item is automatically set to the name of the source table. If the table name contains characters that are not supported by the Common Language Specification, those characters are trimmed or replaced with an underscore, depending on their location in the table name. You can replace the name with one of your choice, as long as it complies with the Common Language Specification.

 **Note:** When a row is of the **DataItem** type, the **Data Source** column directly corresponds to the **DataItemTable** property. When you enter the table name in **DataItemTable** property in the **Properties** window, it is the same as entering it in the **Data Source** column in the **Query Designer** window.

Defining Columns

For each field that you want to include in the resulting dataset, you must define a column in the query data model. To define a column, insert a new row under the data item and set Type to Column. To specify fields, do any the following:

- Type the field name or number directly in the **Data Source** column.
- Select the field from the list of available fields in the **Data Source** column.

- Use the **Field Menu** to insert multiple fields.

Demonstration: Creating and Running a Simple Query

The following demonstration shows how to create a simple query by specifying a data item and columns.

Demonstration Steps

1. Create a new query.
 - a. In Object Designer, click **Query**.
 - b. Click **New**.

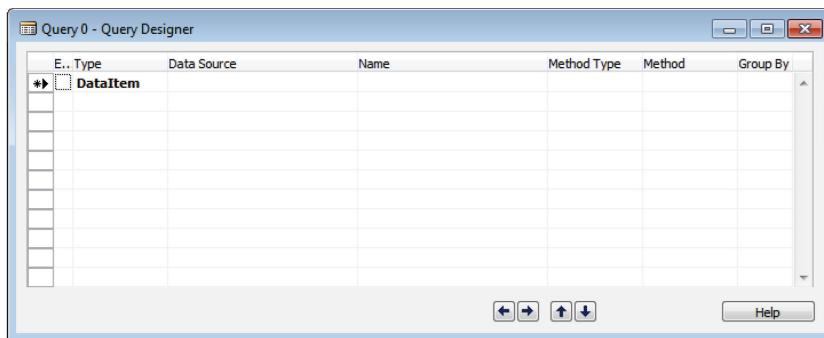


FIGURE 12.2: QUERY DESIGNER FOR A NEW QUERY

2. Add a data item for the **Item** table.
 - a. In the **Type** column, select **DataItem**.
 - b. In the **Data Source** column, enter "Item".

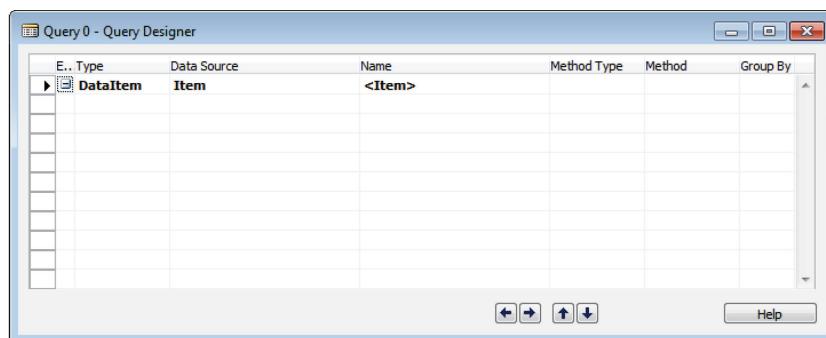


FIGURE 12.3: QUERY DESIGNER AFTER DEFINING A DATA ITEM

3. Add columns for the **No.**, **Description**, **Base Unit of Measure**, and **Unit Cost** fields.
 - a. In **Query Designer**, select the first empty row.
 - b. On the **View** menu, click **Field Menu**.
 - c. In the **Field Menu** window, select the **No.**, **Description**, **Base Unit of Measure**, and **Unit Cost** fields.

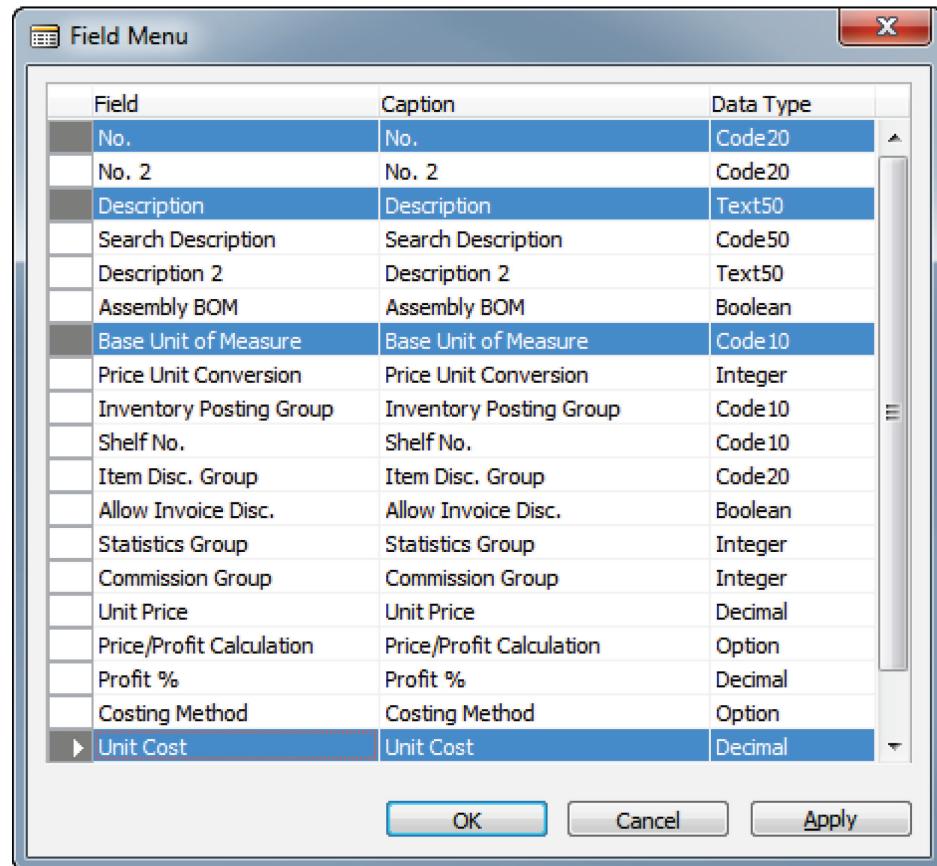


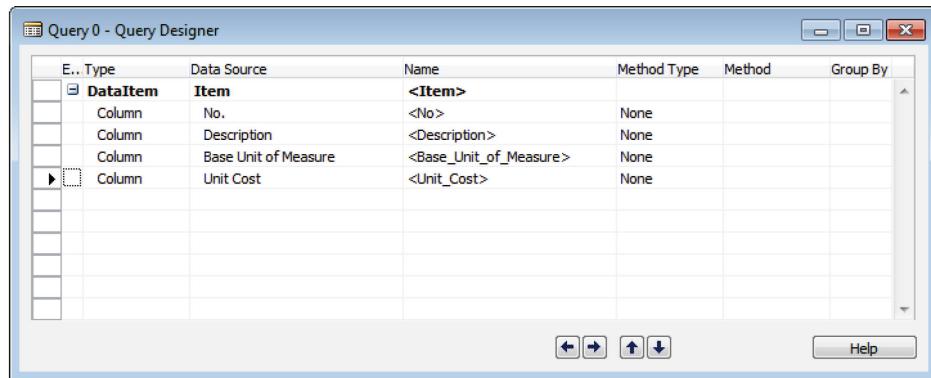
FIGURE 12.4: FIELD MENU WITH SELECTED FIELDS

- d. In the **Field Menu** window, click **OK**.
- e. In the confirmation dialog box, click **OK**.

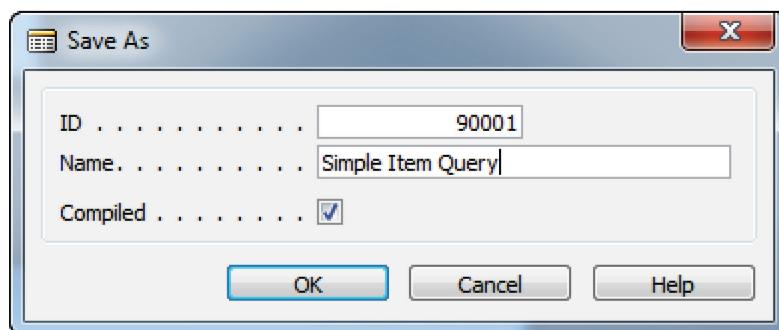


FIGURE 12.5: CONFIRMATION DIALOG

- f. In the **Query Designer** window, make sure that four columns were created for the fields that you selected in the **Field Menu**. Also make sure that they have unique names.

**FIGURE 12.6: QUERY DESIGNER WITH A DATA ITEM AND COLUMNS**

4. Save the **Query** object.
 - a. On the **File** menu, click **Save**.
 - b. In the **Save As** dialog box, in the **ID** field, enter "123456701".
 - c. In the **Name** field, enter "Simple Item Query".
 - d. Make sure that the **Compiled** check box is selected, and then click **OK**.

**FIGURE 12.7: SAVE AS DIALOG BOX**

5. Run the **Query**, and then close the **Query Designer**.
 - a. On the **File** menu, click **Run**.

The "Simple Item Query Results Preview" figure shows the results of the Simple Item Query in the preview mode.

The screenshot shows the 'About This Query' window titled 'About This Query: - Preview of first 1000 rows'. The window has tabs for 'Actions' (Recipient as Attachment, Export as XML, Print Page...) and 'General' (Send To). Below the tabs is a table titled 'About This Query: - Preview of first 1000 rows' with columns: No, Description, Base_Unit_of_..., and Unit_Cost. The table contains 18 rows of data, with the first row (No 1000, Bicycle) highlighted.

No	Description	Base_Unit_of_...	Unit_Cost
1000	Bicycle	PCS	350,594
1001	Touring Bicycle	PCS	350,594
1100	Front Wheel	PCS	129,671
1110	Rim	PCS	1,05
1120	Spokes	PCS	2
1150	Front Hub	PCS	12,441
1151	Axle Front Wheel	PCS	0,45
1155	Socket Front	PCS	0,77
1160	Tire	PCS	1,23
1170	Tube	PCS	1,75
1200	Back Wheel	PCS	129,6815
1250	Back Hub	PCS	12,4515
1251	Axle Back Wheel	PCS	0,33
1255	Socket Back	PCS	0,9
1300	Chain Assy	PCS	13,1565
1310	Chain	PCS	1,99

FIGURE 12.8: SIMPLE ITEM QUERY RESULTS PREVIEW

- b. Close the **About This Query** page.
- c. In the Microsoft Dynamics NAV 2013 Development Environment, close the **Query Designer** window.



Note: The preview mode shows up to 1000 rows. Use this mode as a quick validation, but not as a final output result.

Joining Data

The biggest advantage of queries comes from their ability to join data from multiple tables into a single resulting dataset. Therefore, you usually model queries to include multiple data items that are joined to combine the data from multiple tables into a single resulting dataset.

You join a data item to another data item by indenting it under another data item. The data item under which you have indented another data item becomes the parent data item. Each parent data item can have only one child data item.



Note: You almost never have to use the indentation buttons, because **Query Designer** indents new data items automatically. When you insert a new row in the query data model, it automatically indents under its parent. This behavior applies to data items, columns, and filters.

When you indent data items under one another, you only specify that data from one table will be joined to the data from another table. You have not defined how the data will be joined. To define the data joining rules, use the following data item properties.

Property	Description
DataItemLink	Sets a reference between one or more fields of the source table of the current data item and fields of one or more source tables of the data item or data items that are higher in the data model hierarchy.
DataItemLinkType	Defines how the resulting dataset should handle the rows from the current data item if there is no match in between values that were specified in the data item link, and data items that are higher in the data model hierarchy. <ul style="list-style-type: none">• You can select from the following values:• Use Default Values if No Match – If there is no match, the column values of the upper data item is set to default values for the column data type.• Exclude Row If No Match – If there is no match, the row is excluded from the resulting dataset.• SQL Advanced Options – Uses the value of the SQLJoinType property to determine how the data is handled if there is no match.



Additional Reading: To learn more about joining tables through SQL Advanced Options, refer to the *SQL Advanced Options for Data Item Link Types* topic in Microsoft Dynamics Developer and IT Pro Help.

Demonstration: Joining Data Items

This demonstration shows how to define multiple data items and join them to create a single resulting dataset.

To prepare for this demonstration, you must change several items, and define several purchase prices. To do this, follow these steps:

1. Open the Microsoft Dynamics NAV 2013 client for Windows.
2. In the **Search** box, enter "Items", and press ENTER two times.
3. In the **Items** list, right-click the item 70001, and then click **Edit**.
4. On the **Replenishment** FastTab, in the **Vendor No.** field, enter "30000".
5. Close the **Item Card** page.
6. Repeat steps 3 to 5 for items 70002, 70003, and 70010.
7. In the **Items** list, on the **Navigate** tab in the ribbon, in the **Purchases** group, click **Prices**.
8. In the **Purchase Prices** page, clear the **Item No. Filter** field.
9. Insert the following new purchase prices.

Vendor No.	Item No.	Direct Unit Cost	Starting Date
10000	70040	72,00	1/1/2013
10000	70041	14,60	1/1/2014
32456123	70060	8,00	1/1/2015

10. Close the **Purchase Prices** page.

Demonstration Steps

1. Add a data item for the **Vendor** table to the Simple Item Query query.
 - a. In **Object Designer**, select query 123456701, Simple Item Query, and then click **Design**.
 - b. In the first empty row, in the Type column, select DataItem.
 - c. In the Data Source column, enter "Vendor".
2. Add columns for the **Name** and **City** fields to the **Vendor** data item.
 - a. Select the first empty row.
 - b. On the **View** menu, click **Field Menu**.
 - c. Select the **Name** and **City** fields.
 - d. Click **OK**.

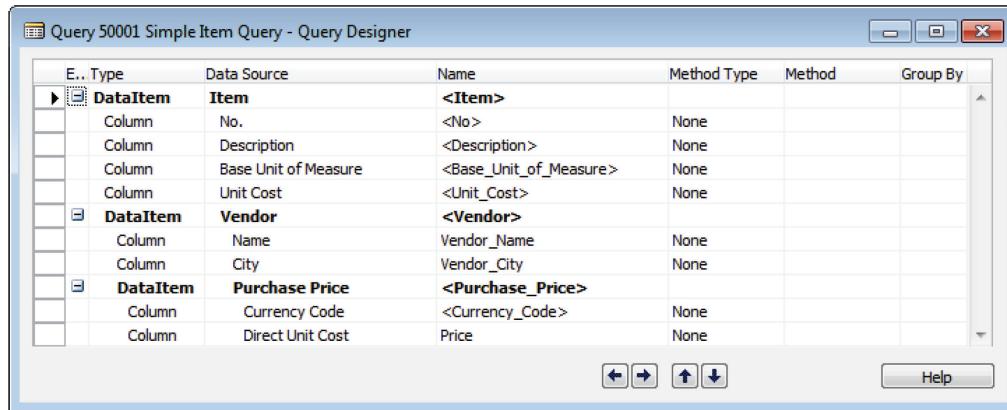
- e. In the confirmation dialog box, click **Yes**.
- f. In the Name column for the **Name** field, enter "Vendor_Name".
- g. In the Name column for the **City** field, enter "Vendor_City".

- 3. Add a data item for the **Purchase Price** table.

 - a. In the first empty row, in the Type column, select DataItem.
 - b. In the Data Source column, enter "Purchase Price".

- 4. Add columns for the **Currency Code** and **Direct Unit Cost** fields.

 - a. Select the first empty row.
 - b. On the **View** menu, click **Field Menu**.
 - c. Select the **Currency Code** and **Direct Unit Cost** fields.
 - d. Click **OK**.
 - e. In the confirmation dialog box, click **Yes**.
 - f. In the **Name** column for the **Direct Unit Cost** field, enter "Price".



The screenshot shows the 'Query 50001 Simple Item Query - Query Designer' window. It displays a grid of data items categorized by type (DataItem) and source (Item, Vendor, Purchase Price). The columns are E.. Type, Data Source, Name, Method Type, Method, and Group By. The data items listed are:

E.. Type	Data Source	Name	Method Type	Method	Group By
DataItem	Item	<Item>			
	Column	No.	<No>	None	
	Column	Description	<Description>	None	
	Column	Base Unit of Measure	<Base_Unit_of_Measure>	None	
DataItem	Vendor	<Vendor>			
	Column	Name	Vendor_Name	None	
	Column	City	Vendor_City	None	
DataItem	Purchase Price	<Purchase_Price>			
	Column	Currency Code	<Currency_Code>	None	
	Column	Direct Unit Cost	Price	None	

FIGURE 12.9: SIMPLE ITEM QUERY WITH MULTIPLE DATA ITEMS

- 5. Join the **Vendor** data item to the **Item** data item.

 - a. Select the **Vendor** data item row.
 - b. On the **View** menu, click **Properties**.
 - c. In the **DataItemLink** property, click the **AssistEdit** button to open the **DataItem Link** window.
 - d. In the **Field** column, select the **No.** field.
 - e. In the **Reference DataItem** column, select the **Item** data item.
 - f. In the **Reference Field** column, select the **Vendor No.** field.

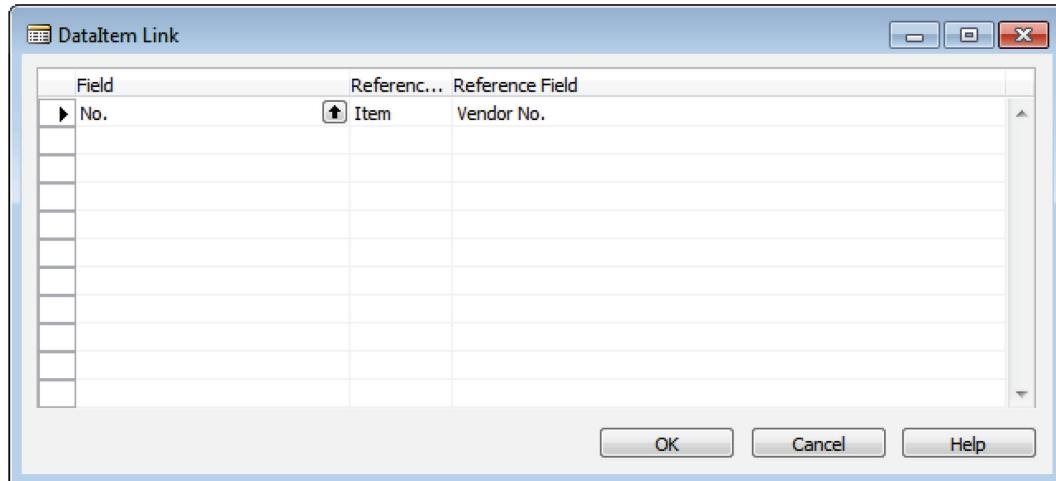
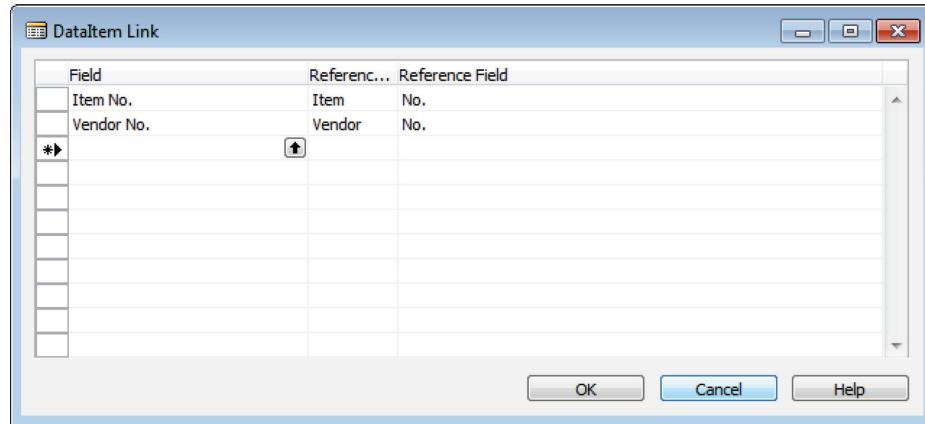


FIGURE 12.10: DATAITEM LINK FOR THE VENDOR DATA ITEM

Note: This joins the **Vendor** and **Item** data items so that each item displays vendors that have the same value in the **No.** field as the value of the **Vendor No.** field in the **Item** table. Because the **No.** field in the **Vendor** table is also its primary key, there can be only one vendor per item.

- g. Click **OK** to accept changes and close the **DataItem Link** window.
 - h. Close the **Properties** window.
6. Join the **Purchase Price** data item to the **Item** and **Vendor** data items.
- a. Select the **Purchase_Price** data item.
 - b. On the **View** menu, click **Properties**.
 - c. In the **DataItemLink** property, click the **AssistEdit** button to open the **DataItem Link** window.
 - d. In the **Field** column, select the **Item No.** field.
 - e. In the **Reference DataItem** column, select the **Item** data item.
 - f. In the **Reference Field** column, select the **No.** column.
 - g. In the next row, in the **Field** column, select the **Vendor No.** field.
 - h. In the **Reference DataItem** column, select the **Vendor** data item.
 - i. In the **Reference Field** column, select the **No.** column.

**FIGURE 12.11: DATAITEM LINK FOR THE PURCHASE PRICE DATA ITEM**

- j. Click **OK** to accept changes and close the **Dataitem Link** window.
- k. Close the **Properties** window.

7. Save and run the query, then view the results.
 - a. On the **File** menu, click **Save**.
 - b. In the **Save** dialog box, make sure that the **Compiled** check box is selected, and then click **OK**.
 - c. On the **File** menu, click **Run**.

No	Description	Base_Unit_of...	Unit_Cost	Vendor_Name	Vendor_City	Currency_Code	Price
1000	Bicycle	PCS	350,594			0	
1001	Touring Bicycle	PCS	350,594			0	
1100	Front Wheel	PCS	129,671	AR Day Propert...	Guildford	0	
1110	Rim	PCS	1,05	Custom Metals...	Birmingham	0	
1120	Spokes	PCS	2	Custom Metals...	Birmingham	0	
1150	Front Hub	PCS	12,441			0	
1151	Axle Front Wheel	PCS	0,45	Groene Kater B...	Humbeek	0	
1155	Socket Front	PCS	0,77	Groene Kater B...	Humbeek	0	
1160	Tire	PCS	1,23	Custom Metals...	Birmingham	0	
1170	Tube	PCS	1,75	Custom Metals...	Birmingham	0	
1200	Back Wheel	PCS	129,6815			0	
1250	Back Hub	PCS	12,4515			0	
1251	Axle Back Wheel	PCS	0,33	Custom Metals...	Birmingham	0	
1255	Socket Back	PCS	0,9	Custom Metals...	Birmingham	0	
1300	Chain Assy	PCS	13,1565			0	
1310	Chain	PCS	1,99	Groene Kater B...	Humbeek	0	
1320	Chain Wheel Fr...	PCS	4,66	Groene Kater B...	Humbeek	0	
1330	Chain Wheel B...	PCS	5,88	Groene Kater B...	Humbeek	0	
1400	Mudguard front	PCS	3,9	Groene Kater B...	Humbeek	0	

FIGURE 12.12: QUERY PREVIEW WITH JOINED DATA ITEMS

- d. Notice that there are several rows without a value in the **Vendor_Name** and **Vendor_City** columns.



Note: This is because of the **DataItemLinkType** property value on the **Vendor** data item. It is set to the default value of **Use Default Values if No Match**. There are rows in the **Item** table where the **Vendor No.** field is blank. Therefore, no matching vendor is found, and any fields from the **Vendor** table are blank.

8. Configure the join on the **Vendor** data item to only show those rows from the **Item** table that have a matching row in the **Vendor** table, and then run the query.
 - a. Select the **Vendor** data item.
 - b. On the **View** menu, click **Properties**.
 - c. In the **DataItemLinkType** property, select **Exclude Row If No Match**.
 - d. Close the **Properties** window.
9. Configure the join on the **Purchase Price** data item to only show those rows from the **Item** table that have a matching row in the **Purchase Price** table.
 - a. Select the **Purchase_Price** data item.
 - b. On the **View** menu, click **Properties**.
 - c. In the **DataItemLinkType** property, select **Exclude Row If No Match**.
 - d. Close the **Properties** window.
10. Save and run the query, then view the results.
 - a. On the **File** menu, click **Save**.
 - b. In the **Save** dialog box, make sure that the **Compiled** check box is selected, and then click **OK**.
 - c. On the **File** menu, click **Run**.

No	Description	Base_Unit_of...	Unit_Cost	Vendor_Name	Vendor_City	Currency_Code	Price
70001	Base	PCS	20,6	CoolWood Tec...	Portsmouth		22,7
70002	Top Panel	PCS	14,6	CoolWood Tec...	Portsmouth		16,1
70003	Rear Panel	PCS	15,1	CoolWood Tec...	Portsmouth		16,6
70010	Wooden Door	PCS	26,5	CoolWood Tec...	Portsmouth		29,2
70040	Drawer	PCS	55,1	London Postm...	London		72
70041	Shelf	PCS	11,9	London Postm...	London		14,6
70060	Mounting	PCS	6,7	Groene Kater B...	Humbeek	EUR	8

FIGURE 12.13: QUERY PREVIEW WITH DATA ITEMS TO EXCLUDE ROWS IF NO MATCH

- d. Notice that only those rows that have a matching vendor and are listed in the **Purchase Price** table are shown.



Note: This is because of the **DataItemLinkType** property value on the **Purchase Price** data item. This is set to the **Exclude Row if No Match**. There are only a few rows in the **Purchase Price** table, and only those items are shown that are represented in the **Purchase Price** table.

11. Reconfigure the join on the **Purchase Price** data item to show default values if there is no match. This displays the purchase price list that has all items that have a vendor, but without price information if a purchase price is not defined.
 - a. Select the **Purchase_Price** data item.
 - b. On the **View** menu, click **Properties**.
 - c. In the **DataItemLinkType** property, select **Use Default Values if No Match**.
 - d. Close the **Properties** window.
12. Save and run the query, and then view the results.
 - a. On the **File** menu, click **Save**.
 - b. In the **Save** dialog box, make sure that the **Compiled** check box is selected, and then click **OK**.
 - c. On the **File** menu, click **Run**.

The screenshot shows a Microsoft Dynamics NAV application window titled "About This Query: - Preview of first 1000 rows". The window has a toolbar with actions like "Recipient as Attachment", "Export as XML", "Print Page...", "Microsoft Word", and "Microsoft Excel". Below the toolbar is a "Send To" dropdown and a "General" tab. The main area displays a grid of data with the following columns: No, Description, Base_Unit_of_..., Unit_Cost, Vendor_Name, Vendor_City, Currency_Code, and Price. The data includes various items like TOKYO Guest..., MEXICO Swivel..., and MOSCOW Swivel..., along with their respective details such as unit cost, vendor information, and currency.

No	Description	Base_Unit_of_...	Unit_Cost	Vendor_Name	Vendor_City	Currency_Code	Price
1964-S	TOKYO Guest...	PCS	83,36214	AR Day Propert...	Guildford	0	
1968-S	MEXICO Swivel...	PCS	96,1	CoolWood Tec...	Portsmouth	0	
1972-S	MUNICH Swivel...	PCS	96,1	CoolWood Tec...	Portsmouth	0	
1980-S	MOSCOW Swivel...	PCS	96,1	CoolWood Tec...	Portsmouth	0	
1988-S	SEOUL Guest C...	PCS	97,5	AR Day Propert...	Guildford	0	
1996-S	ATLANTA White...	PCS	707,2	CoolWood Tec...	Portsmouth	0	
2000-S	SYDNEY Swivel...	PCS	96,1	CoolWood Tec...	Portsmouth	0	
70000	Side Panel	PCS	15,7	London Postm...	London	0	
70001	Base	PCS	20,6	CoolWood Tec...	Portsmouth	22,7	
70002	Top Panel	PCS	14,6	CoolWood Tec...	Portsmouth	16,1	
70003	Rear Panel	PCS	15,1	CoolWood Tec...	Portsmouth	16,6	
70010	Wooden Door	PCS	26,5	CoolWood Tec...	Portsmouth	29,2	
70011	Glass Door	PCS	36,9	London Postm...	London	0	
70040	Drawer	PCS	55,1	London Postm...	London	72	
70041	Shelf	PCS	11,9	London Postm...	London	14,6	
70060	Mounting	PCS	6,7	Groene Kater B...	Humbeek	EUR	8
70100	Paint, black	CAN	1,4	AR Day Propert...	Guildford	0	
70101	Paint, yellow	CAN	1,4	AR Day Propert...	Guildford	0	
70102	Paint, blue	CAN	1,4	AR Day Propert...	Guildford	0	
70103	Paint, red	CAN	1,4	AR Day Propert...	Guildford	0	
70104	Paint, green	CAN	1,4	AR Day Propert...	Guildford	0	
70200	Hinge	PCS	0,7	London Postm...	London	0	
70201	Doorknob	PCS	0,6	London Postm...	London	0	
80001	Computer III 53...	PCS	5,8	Service Electro...	Watford	0	
80002	Computer III 60...	PCS	7	Service Electro...	Watford	0	
80003	Computer III 73...	PCS	8,2	Service Electro...	Watford	0	

FIGURE 12.14: QUERY PREVIEW WITH DIFFERENT JOIN OPTIONS ON DIFFERENT DATA ITEMS

- d. Notice that all items that have a matching Vendor are shown, regardless of whether they have a Purchase Price.

Filtering Data

There are several ways to filter the data in resulting datasets of queries:

- By defining the **DataItemTableFilter** property on a data item
- By defining the **ColumnFilter** property on a column
- By creating rows of type Filter
- By writing C/AL code in the OnBeforeOpen trigger

DataItemTableFilter Property

You use the DataItemTableFilter property on data items to apply conditions on one or more fields of the table to limit the records in the resulting dataset of the query. You can filter on any field in the table, not just those fields that are included as columns in the resulting dataset.

To set up filters, you can enter either the filter syntax directly in the Value column for the DataItemTableField property in the **Properties** window or click the **AssistEdit** button in the Value column, and then use the **Table Filter** window to set up the filters.



Note: The **Table Filter** window works exactly the same way as with pages, reports, or XMLports.

The filters you define in the DataItemTableFilter property are static. Users or C/AL code cannot override them.

ColumnFilter Property

You use the ColumnFilter property on a column to apply a filter condition on a single field. The ColumnFilter property resembles the DataItemTableFilter property, but there are differences. The ColumnFilter property has the following behaviors:

- Unlike filters that are set by the DataItemTableFilter property, filters that are set by the ColumnFilter property can be overwritten at run time by calling the SETFILTER or SETRANGE functions from C/AL code.
- If the ColumnFilter property specifies a filter on the same field as the DataItemTableFilter property, then the filters of the two properties are combined. To be included in the query dataset, records must meet the condition of both the filters. For example, if the DataItemTableFilter property sets a filter on a field to include values less than fifty (<50) and the ColumnFilter property sets a filter on the same field to include values greater than twenty (>20), then the resultant filter on the field includes values that are greater than twenty and less than fifty.



Note: The **SETFILTER** and **SETRANGE** functions overwrite any filter on the same field that is set on a column or filter row by the ColumnFilter property in Query Designer. If a **SETFILTER** or **SETRANGE** function filters on the same field as a filter on a data item, as specified by the DataItemTableFilter property, then the function filter and DataItemTableFilter property filter are combined.

Filter Rows

You use rows for type Filter to enable dynamic filtering of the resulting dataset on fields that you do not want to include in the resulting dataset of a query. For example, you might want to let users or developers to filter on a date range. However, you do not want to include the date in the dataset.

To define a filter, insert a new row under the data item for which you want to add a filter. Select Type of Filter, and then select the field from the table on which you want to enable filtering.



Note: You cannot use **Field Menu** to create rows of type Filter. You can only create new filter rows by defining them individually.

You can set the following properties for filter rows:

Property	Description
DataSource	Specifies the field to be used as a filter. For rows of Type Filter, defining this property has the same effect as defining the Data Source property in the Query Designer window.
ColumnFilter	Specifies the set of predefined filters that are applied to the field that is specified in the DataSource property if the user or C/AL code do not override the value of the filter.

 **Note:** The ColumnFilter property of filter rows behaves the same way as with the columns.

OnBeforeOpen Trigger

The OnBeforeOpen trigger runs before its model is translated into a Transact-SQL statement and is executed against the underlying Microsoft SQL Server database.

You can use the OnBeforeOpen trigger to set filters on the query by using the SETFILTER or SETRANGE C/AL functions. When you use C/AL, you can only filter the query on columns or filters that are defined in the query. You cannot filter on any other fields that are present in any of the data items that you have not added to the query model as either a column or a filter.

The following example filters the query to include only those rows where Unit_Cost is not 0.

```
CurrQuery.SETFILTER(Unit_Cost,'<>%1',0);
```

Demonstration: Defining Filters in a Query

This demonstration shows how to set various types of filters in a query.

Demonstration Steps

1. Filter the Simple Item Query to only show items that use the purchase replenishment system by defining the DataItemTableFilter property on a data item.
 - a. In **Object Designer**, select query 123456701, **Simple Item Query**, and then click **Design**.
 - b. Select the **Item** data item. It is the first row in **Query Designer**.

- c. On the **View** menu, click **Properties**.
- d. In the **DataItemTableFilter** property, click the **AssistEdit** button.
- e. In the **Field** column, select the **Replenishment System** field.
- f. In the **Type** column, select CONST.
- g. In the **Value** column, enter "Purchase".

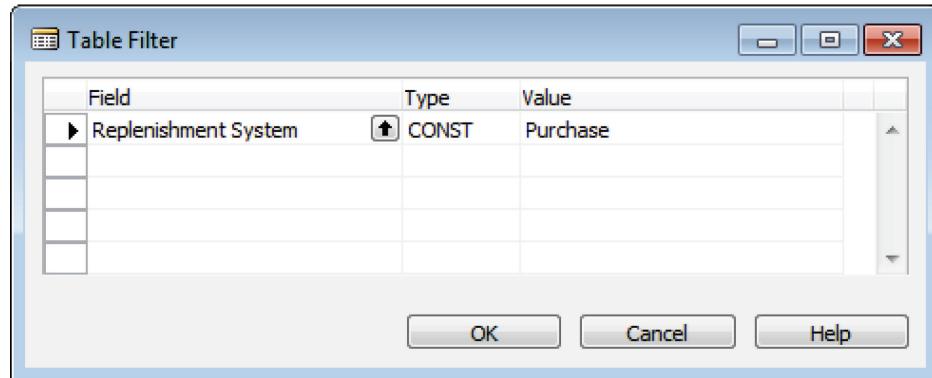


FIGURE 12.15: TABLE FILTER FOR THE ITEM DATA ITEM

- h. Click **OK** to accept the changes.
 - i. Close the **Properties** window.
-
2. Filter the query to only show those items that have unit cost defined by setting the **ColumnFilter** property on a column.
 - a. Select the **Unit_Cost** column.
 - b. On the **View** menu, click **Properties**.
 - c. In the **ColumnFilter** property, click the **AssistEdit** button to open the **Column Filter** window.
 - d. In the **Column** field, make sure that the **Unit_Cost** value is present.
 - e. In the **Type** field, select FILTER.
 - f. In the **Value** field, enter "<>0".

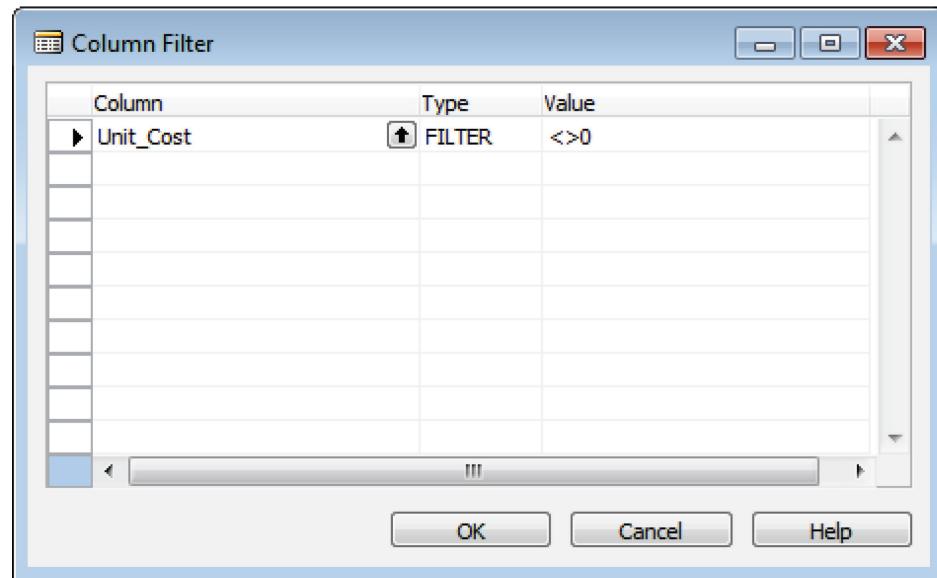
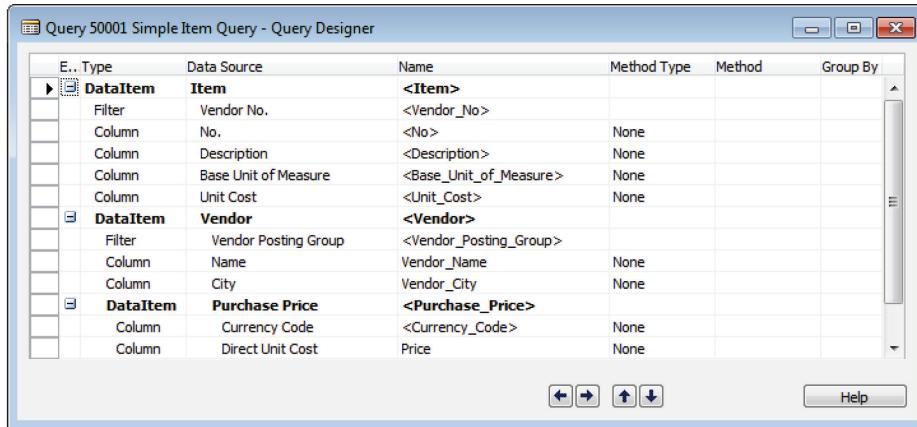


FIGURE 12.16: COLUMN FILTER FOR THE UNIT_COST COLUMN.

- g. Click **OK** to accept changes and close the **Column Filter** window.
 - h. Close the **Properties** window.
3. Enable users to dynamically filter the query on the **Vendor No.** and **Vendor Posting Group** fields without including the fields in the resulting dataset.
- a. Select the **No.** field under the **Item** data item.
 - b. On the **Edit** menu, click **New**.
 - c. In the **Type** column, select **Filter**.
 - d. In the **Data Source** column, select the **Vendor No.** field.
 - e. Select the **Name** field under the **Vendor** data item.
 - f. On the **Edit** menu, click **New**.
 - g. In the **Type** column, select **Filter**.
 - h. In the **Data Source** column, select the **Vendor Posting Group** field.

**FIGURE 12.17: SIMPLE ITEM QUERY WITH FILTERS**

4. Compile, save, close, and then run the query. Then, view the results.
 - a. On the **File** menu, click **Save**.
 - b. In the **Save** dialog box, make sure that the **Compiled** check box is selected, and then click **OK**.
 - c. In **Object Designer**, select query 123456701, **Simple Item Query**, and then click **Run**.



Note: The query now only shows those records where the **Replenishment System** field in the **Item** table is set to **Purchase**, and where **Unit Cost** field is different from 0. The query does not include **Vendor_No** and **Vendor_Posting_Group** information. However, you can dynamically filter on this information through C/AL code.

Aggregating Data

In a query, you use a totals method to perform a calculation on a column and return the calculated value in the resulting dataset. This calculation is frequently called aggregation.

You typically use aggregations with grouping to find totals for specified groups of columns. For example, you can sum the values in the **Amount** field per G/L account in the **G/L Entry** table or find the average **Quantity** per item in the **Item Ledger Entry** table. Totals methods in Microsoft Dynamics NAV 2013 queries correspond directly to Microsoft SQL Server aggregate functions.

Use the following totals methods.

Method Type	Description
Sum	Calculates the sum of the values of the specified column for all records that are grouped within the dataset.
Average	Calculates the average value in the specified column in a group.
Min	Retrieves the lowest value in the specified column in a group.
Max	Retrieves the highest value in the specified column in a group.
Count	Returns the number of records from data item tables that represent a group in the dataset.

To set the totaling methods in a Microsoft Dynamics NAV 2013 query, follow these steps:

1. For a column, set the **MethodType** property to **Totals**.
2. In the **Method** property, select the appropriate totals method, for example, Average.

 **Note:** When you apply a totals method to a column, and you have not changed the Name from the default, the Name of the column is automatically prefixed with the name of the totals method that is being applied. It reflects the fact that the column is aggregated. For example, when you apply the Average method to the Amount column, the name is automatically changed to Average_Amount. The column must be of type Decimal.

Ordering Data

You can order the data in a resulting dataset of a query by any number of columns that are contained in the query. To define the sorting order, set the OrderBy property on the query.

To access the OrderBy property, in **Query Designer**, select the first blank line, and then, on the **View** menu, click **Properties**.

To modify the OrderBy property, you can click the **AssistEdit** button next to the property. In the **Order By** window, you add a column and set its direction to ascending or descending. You can sort on multiple columns by adding additional columns to the **Order By** window. The query will sort the results by the first column in the **Order By** window, then by the second column, and so on.

You can also type values directly in the Value column of the OrderBy property in the **Properties** window. To sort on multiple columns, separate each column with a comma.



Note: You cannot sort by a column that is not present in the resulting dataset. If you want to order by such a column, you must first add it to the query model, and then include it in the OrderBy property.

Date Methods

When retrieving Date or DateTime fields from a Microsoft Dynamics NAV 2013 database, you may want to retrieve only the year, month or day, instead of the entire date. This is especially true when you group and total the data. For example, you may want to group revenue by customer and by year, or find average consumption of a production BOM component by month. You can achieve this by selecting a date method for a Date or DateTime column in the **Query Designer**.

Date methods are as follows:

- Day
- Month
- Year

To apply a date method for a column, follow these steps:

1. In the column definition for a Date or DateTime column, set the **MethodType** property to **Date**.
2. In the **Method** property, select the appropriate date method, depending on which date part you want to retrieve.



Note: When you apply a date method to a Date or DateTime column, and you have not changed the Name from the default, the Name of the column is automatically prefixed with the name of the date method that is being applied. For example, when you apply the Year date method to the Posting Date column, the name is automatically changed to Year_Posting_Date.

On the SQL server, date and time values are processed by using Coordinated Universal Time (UTC). If Microsoft Dynamics NAV solution uses a time zone other than UTC and the field on which you apply the date method has a data type of DateTime, then there might be a difference between the date value that is returned in the dataset for the field, and the actual day, month, or year for the field in the table. This occurs when the corresponding UTC date for a field falls on the next day or previous day because of the time of day and the time zone of Microsoft Dynamics NAV solution.

The differences in day, month, or year occur when date and time values are retrieved from the Microsoft Dynamics NAV database table. These values are converted from the regional settings of the Microsoft Dynamics NAV solution to the UTC date and time. The day, month, or year is calculated on the SQL server, and then returned to the query dataset as an integer. This integer does not consider the regional settings of the Microsoft Dynamics NAV solution.

To avoid this condition, you should use the date method on fields that have a Date data type instead of a DateTime data type when possible. You can also return the DateTime value and implement post processing for the day, month, and year as needed.

Demonstration: Aggregate and Order the Data

The following demonstration shows how to aggregate the data and order the resulting dataset of a query.

Demonstration Steps

1. Order the **Simple Item Query** by vendor name in alphabetical order, and by price from highest to lowest.
 - a. In **Object Designer**, select query 123456701, **Simple Item Query**, and then click **Design**.
 - b. Select the first blank row, and then on the **View** menu, click **Properties**.
 - c. In the **OrderBy** property, click the **AssistEdit** button to open the **Order By** window.
 - d. In the **Column** field on the first row, select the **Vendor_Name** column. Make sure that **Direction** is set to Ascending.
 - e. In the **Column** field on the second row, select the **Price** column. Make sure that **Direction** is set to Descending.

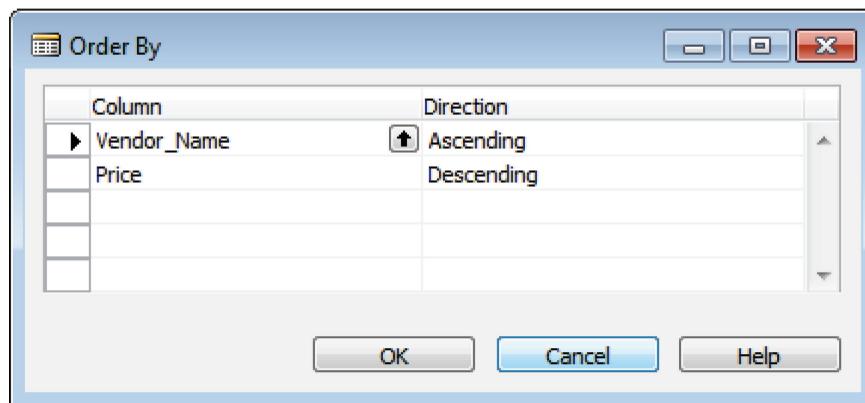


FIGURE 12.18: THE ORDER BY WINDOW FOR THE SIMPLE ITEM QUERY

- f. Click **OK** to accept changes and close the **Order By** window.
- g. Close the **Properties** window.

2. Configure the query to only show the average price for each item.
 - a. For the Price column, set the Method Type property to Totals, and the Method property to Average.
 - b. Notice that for each row of the Type column, the Group By property is selected automatically.

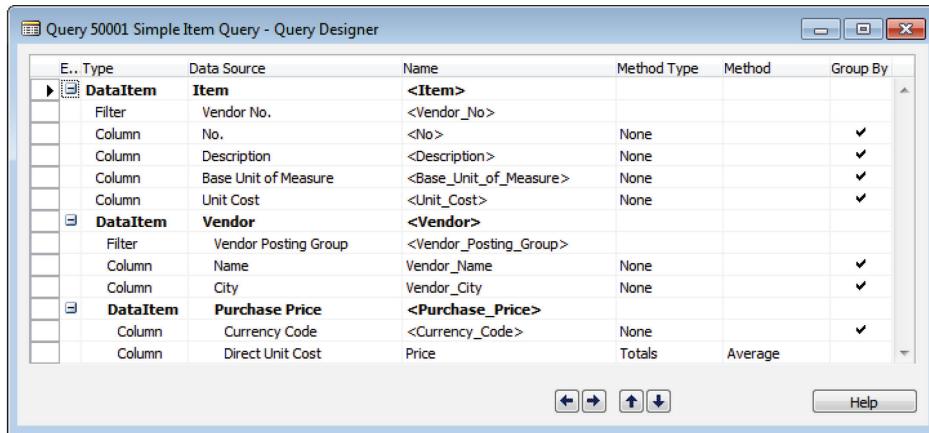
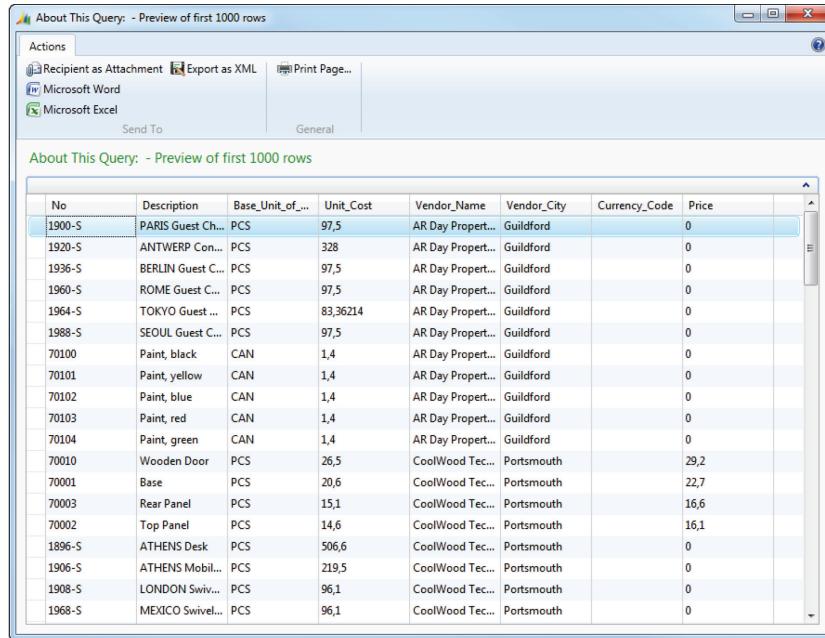


FIGURE 12.19: THE SIMPLE ITEM QUERY IN QUERY DESIGNER, AFTER APPLYING A TOTALS METHOD

3. Save, compile, close, and then run the query. Then, view the results.
 - a. On the **File** menu, click **Save**.
 - b. In the **Save** dialog box, make sure that the **Compiled** check box is selected, and then click **OK**.
 - c. Close the **Query Designer**.
 - d. In **Object Designer**, select query 123456701, **Simple Item Query**, and then click **Run**.



The screenshot shows a Microsoft Dynamics NAV application window titled 'About This Query: - Preview of first 1000 rows'. The window has a toolbar with actions like 'Recipient as Attachment', 'Export as XML', 'Print Page...', 'Microsoft Word', and 'Microsoft Excel'. Below the toolbar is a 'Send To' dropdown set to 'General'. The main area displays a table with 20 rows of data, sorted by Vendor_Name (ascending) and Price (descending). The columns are: No, Description, Base_Unit_of_..., Unit_Cost, Vendor_Name, Vendor_City, Currency_Code, and Price.

No	Description	Base_Unit_of_...	Unit_Cost	Vendor_Name	Vendor_City	Currency_Code	Price
1900-S	PARIS Guest Ch...	PCS	97,5	AR Day Propert...	Guildford	0	
1920-S	ANTWERP Con...	PCS	328	AR Day Propert...	Guildford	0	
1936-S	BERLIN Guest C...	PCS	97,5	AR Day Propert...	Guildford	0	
1960-S	ROME Guest C...	PCS	97,5	AR Day Propert...	Guildford	0	
1964-S	TOKYO Guest ...	PCS	83,36214	AR Day Propert...	Guildford	0	
1988-S	SEOUL Guest C...	PCS	97,5	AR Day Propert...	Guildford	0	
70100	Paint, black	CAN	1,4	AR Day Propert...	Guildford	0	
70101	Paint, yellow	CAN	1,4	AR Day Propert...	Guildford	0	
70102	Paint, blue	CAN	1,4	AR Day Propert...	Guildford	0	
70103	Paint, red	CAN	1,4	AR Day Propert...	Guildford	0	
70104	Paint, green	CAN	1,4	AR Day Propert...	Guildford	0	
70010	Wooden Door	PCS	26,5	CoolWood Tec...	Portsmouth	29,2	
70001	Base	PCS	20,6	CoolWood Tec...	Portsmouth	22,7	
70003	Rear Panel	PCS	15,1	CoolWood Tec...	Portsmouth	16,6	
70002	Top Panel	PCS	14,6	CoolWood Tec...	Portsmouth	16,1	
1896-S	ATHENS Desk	PCS	506,6	CoolWood Tec...	Portsmouth	0	
1906-S	ATHENS Mobil...	PCS	219,5	CoolWood Tec...	Portsmouth	0	
1908-S	LONDON Swivel...	PCS	96,1	CoolWood Tec...	Portsmouth	0	
1968-S	MEXICO Swivel...	PCS	96,1	CoolWood Tec...	Portsmouth	0	

FIGURE 12.20: ORDERED QUERY RESULTS AFTER APPLYING A TOTALS METHOD

- e. Notice that the results are sorted first by Vendor_Name in ascending (alphabetical) order, and then by Price in descending order. Also, the average price is shown.

Lab 12.1: Using a Query from a Chart

Scenario

Susan, the sales order processor at CRONUS International Ltd., needs a chart that shows the top ten customers by revenue. She wants to access this chart from her Role Center.

Isaac and Simon, the consultants at the ISV company that implements Microsoft Dynamics NAV 2013 for CRONUS International Ltd., will help Susan by creating and customizing the necessary objects, and then configure Microsoft Dynamics NAV 2013 according to Susan's requirements.

Objectives

The objectives of this lab are:

- Learn how to create a new query.
- Understand how to set query and data item properties.
- Understand how to use totaling.
- Understand how to use a query in a chart.

Exercise 1: Creating a query

Exercise Scenario

Isaac, the business software developer, creates a new query object that selects the data that Susan wants to see in a chart part on her role center. The query will join the data from the **Customer** and **Cust. Ledger Entries** tables, show only those customers who have ledger entries of type Invoice or Credit Memo, and display the Amount (LCY) column totaled by the Sum method.

Task 1: Create a new query object

High Level Steps

1. Create a new query object.
2. Select table 18, **Customer** as the first data item.

Detailed Steps

1. Create a new query object.
 - a. On the **Tools** menu, click **Object Designer**.
 - b. In the **Object Designer** window, click **Query**.
 - c. Click **New** to open the **Query Designer**.

2. Select table 18, **Customer** as the first data item.
 - a. In the **Query Designer** window, make sure that **Type** property is **DataItem**.
 - b. In the **DataSource** property, select table 18, **Customer**.

Task 2: Add another data item

High Level Steps

1. Select table 21, **Cust. Ledg. Entry** as the second data item, indented under Customer.
2. Set the properties for Cust_Ledger_Entry data item to link to the Customer data item, where the **Customer No.** field of Cust_Ledger_Entry is equal to the **No.** field of the **Customer** table, and to exclude the customers for which there are no ledger entries.
3. Filter the data item to show only entries of type Invoice and Credit Memo.

Detailed Steps

1. Select table 21, **Cust. Ledg. Entry** as the second data item, indented under Customer.
 - a. In the **Query Designer** window, move to the last row until the new row indicator is shown.
 - b. Set **Type** to **DataItem**, and in the **DataSource** property select table 21, **Cust. Ledg. Entry**.
2. Set the properties for Cust_Ledger_Entry data item to link to the Customer data item, where the **Customer No.** field of Cust_Ledger_Entry is equal to the **No.** field of the **Customer** table, and to exclude the customers for which there are no ledger entries.
 - a. Select the Cust_Ledger_Entry data item, on the **View** menu click **Properties**.
 - b. In the **Properties** window for the Cust_Ledger_Entry data item, in the **DataItemLink** property click the **AssistEdit** button.
 - c. In the **DataItem Link** window, select Customer No. as **Field**, Customer as **Reference DataItem**, and No. as **Reference Field**.
 - d. Click **OK**.
 - e. In the **Properties** window for the Cust_Ledger_Entry data item, select **Exclude Row If No Match** in the **DataItemLinkType** property.

3. Filter the data item to show only entries of type Invoice and Credit Memo.
 - a. In the **Properties** window for the Cust_LedgerEntry data item, in the **DataItemTableFilter** property, click the **AssistEdit** button.
 - b. In the **Table Filter** window, in the **Field** column select **Document Type**, in the **Type** column select **FILTER**, and then in the **Value** column enter "Invoice|Credit Memo".
 - c. Click **OK**.
 - d. Close the **Properties** window for the Cust_Ledger_Entry data item.

Task 3: Add columns to the query

High Level Steps

1. Add columns for the **No.**, **Name** and **Customer Posting Group** fields to the Customer data item.
2. Add a column for field **Amount (LCY)** to the Cust_Ledger_Entry data item.
3. Specify the Sum totaling for the Amount (LCY) column.

Detailed Steps

1. Add columns for the **No.**, **Name** and **Customer Posting Group** fields to the Customer data item.
 - a. In the **Query Designer** window, select the Customer data item.
 - b. On the **View** menu, click **Field Menu**.
 - c. In the **Field Menu** window, select the **No.**, **Name** and **Customer Posting Group** fields.
 - d. Click **OK**.
2. Add a column for field **Amount (LCY)** to the Cust_Ledger_Entry data item.
 - a. In the **Query Designer** window, select the Cust_Ledger_Entry data item.
 - b. On the **View** menu, click **Field Menu**.
 - c. In the **Field Menu** window, select the **Amount (LCY)** field.
 - d. Click **OK**.
3. Specify the Sum totaling for the Amount (LCY) column.
 - a. In the **Query Designer** window, select the Amount_LCY column.
 - b. Select **Totals** as the **Method Type** property.
 - c. Make sure that **Sum** is selected as the **Method** property.

Task 4: Set query properties

High Level Steps

1. Set the query properties to order the results by sum of **Amount (LCY)** in descending order.
2. Set the query properties to return only the top 10 rows.

Detailed Steps

1. Set the query properties to order the results by sum of **Amount (LCY)** in descending order.
 - a. In the **Query Designer** window, move to the last row until the new row indicator is shown.
 - b. On the **View** menu, click **Properties**.
 - c. In the **OrderBy** property, click the **AssistEdit** button.
 - d. Select **Sum_Amount_LCY** as Column, and **Descending** as Direction.
 - e. Click **OK**
2. Set the query properties to return only the top 10 rows.
 - a. In the **Properties** window for the query, specify "10" as **TopNumberOfRows**.
 - b. Close the **Properties** window for the query.

Task 5: Save the query

High Level Steps

1. Save the newly defined query object as 123456701, **Top 10 Cust. by Revenue**.
2. Close the **Query Designer**.

Detailed Steps

1. Save the newly defined query object as 123456701, **Top 10 Cust. by Revenue**.
 - a. In the **Query Designer** window, on the **File** menu, click **Save**.
 - b. In the **Save As** dialog box, in the **ID** text box, enter "123456702".
 - c. In the **Name** text box, enter "Top 10 Cust. By Revenue", and then click **OK**.
2. Close the **Query Designer**.
 - a. On the **File** menu, click **Close**.

Results

A new query object, 123456702, Top 10 Cust. by Revenue, that joins the data from the **Customer** and **Cust. Ledger Entry** tables

Exercise 2: Creating a chart

Exercise Scenario

Simon, the consultant, creates a new chart setup record in Microsoft Dynamics NAV 2013, to show the data from the query that was created by Isaac in the previous exercise. The chart will be a column chart, and will show Sum_Amount_LCY as a measure that is aggregated by the Sum method over the customer number as the X-axis dimension.

Task 1: Create a new chart setup

High Level Steps

1. Open the Microsoft Dynamics NAV 2013 client for Windows.
2. Create a new chart setup.

Detailed Steps

1. Open the Microsoft Dynamics NAV 2013 client for Windows.
 - a. On the **Start** menu, click Microsoft Dynamics NAV 2013.
2. Create a new chart setup.
 - a. In the Microsoft Dynamics NAV 2013 client for Windows, click **Departments > Application Setup > RoleTailored Client > Charts**.
 - b. On the **Home** tab, in the **New** group, click **New**.

Task 2: Configure the Chart Setup Card

High Level Steps

1. Set the new chart ID to 123456702-01 and name it as Top 10 Cust. by Revenue.
2. Set the data source of the chart to query 123456702, Top 10 Cust. by Revenue.
3. Define the Sum_Amount_LCY column as the measure. Set **Aggregation** to **Sum**, and **Graph Type** to **Column**.
4. Define the No column as the X-Axis dimension.

Detailed Steps

1. Set the new chart ID to 123456702-01 and name it as Top 10 Cust. by Revenue.
 - a. In the **Chart Setup Card** page, in the **ID** field, enter "123456702-01".
 - b. In the **Name** field, enter "Top 10 Cust. by Revenue".
2. Set the data source of the chart to query 123456702, Top 10 Cust. by Revenue.
 - a. In the **Chart Setup Card** window, in the **Source Type** field, select **Query**.
 - b. In the **Source ID** field, enter "123456702".
3. Define the Sum_Amount_LCY column as the measure. Set **Aggregation** to **Sum**, and **Graph Type** to **Column**.
 - a. On the **Measures (Y-Axis)** FastTab, in the **Data Column** field for Required Measure row, select Sum_Amount_LCY.
 - b. In the **Aggregation** field, select **Sum**
 - c. In the **Graph Type** field, select **Column**.
4. Define the No column as the X-Axis dimension.
 - a. On the **Dimensions (X- and Z-Axes)** FastTab, in the **X-Axis** field, enter "No".
 - b. In the **X-Axis Title** field, enter "No".
 - c. Click **OK** to close the page.

Results

A new chart setup record that shows data from query 123456702, Top 10 Cust. by Revenue.

Exercise 3: Adding the chart to the Role Center.

Exercise Scenario

Susan customizes her Role Center to include the chart part that Simon configured in the previous exercise.

Task 1: Customize the Role Center

High Level Steps

1. Customize the Role Center.

Detailed Steps

1. Customize the Role Center.
 - a. In the Microsoft Dynamics NAV 2013 client for Windows, click **Home** to navigate to the Role Center.
 - b. Click the Application menu, then click Customize > Customize This Page.
 - c. In the **Customize the Role Center** window, in Role Center Layout, click **Microsoft Outlook**.
 - d. In the **Available Parts** list, click **Chart Part**, and then click **Add>>**.
 - e. Click Customize Part.
 - f. In the **Customize Chart** window, select chart 123456702-01 Top 10 Cust. by Revenue, and then click **OK**.
 - g. Click **OK** to close the **Customize the Role Center** window.

Task 2: Preview the chart

High Level Steps

1. Restart the Microsoft Dynamics NAV 2013 client for Windows.
2. View the new chart part.

Detailed Steps

1. Restart the Microsoft Dynamics NAV 2013 client for Windows.
 - a. Close the Microsoft Dynamics NAV 2013 client for Windows.
 - b. Start the Microsoft Dynamics NAV 2013 client for Windows.
2. View the new chart part.
 - a. Make sure that the **Top 10 Cust. by Revenue** part is shown below the **Outlook** part.

Accessing Queries from C/AL

Queries can be an effective replacement for iterative access to data in Microsoft Dynamics NAV 2013 especially when you use multiple nested data iterations in the C/AL code. Query object includes a set of C/AL functions that you can use to access data, to filter the resulting dataset of a query, or to export the resulting dataset to the CSV or XML format.

Running Queries

You can run a query C/AL to iterate through the resulting dataset programmatically. Even though the principles of accessing a query resemble those of accessing tables, there is a different set of functions on a query object.

OPEN

The **OPEN** function runs a query object and generates a dataset that you can read. It also puts the query in the reading state.

The **OPEN** function returns a Boolean value. This indicates whether the query opened successfully. If you omit this optional return value and if the query does not open successfully, then a run-time error occurs. If you include a return value, then no run-time error occurs when the **OPEN** function is called and you will handle any errors.

If the **OPEN** function fails, you cannot call other functions or access the data in the query. If you try to do this, a run-time error occurs.

 **Note:** The **OPEN** function only runs the query object and generates a dataset. It does not return the first row of the result set. To access any row, you must call **READ**.

READ

The **READ** function reads a single row from the resulting dataset of a query. The function returns a Boolean value that indicates if a row was retrieved.

When you call the **READ** function, the next row in the dataset query is retrieved. When the query is in the reading state, you can access the values of columns in the row in the same manner that you access the fields in a record variable.

You can call the **READ** function multiple times after the **OPEN** function to read consecutive rows in the dataset. The first **READ** function call retrieves the first row from the resulting dataset. Each successive **READ** function call retrieves the next row from the dataset.

CLOSE

The **CLOSE** function closes a query dataset and returns the query to the initialized state. It lets the NAV Server know that you are finished using this object.

Calling **CLOSE** explicitly is optional. This function is called implicitly in any of the following situations:

- When the query variable goes out of scope
- If you call the **OPEN** function on a query variable that is currently open
- If you call the **SETFILTER** or **SETRANGE** functions on a query variable that is currently open

The **CLOSE** function does not clear any filters that you set on the query programmatically. If you want to clear such filters, then you must call the **CLEAR** function.

The following example shows how to open a query, iterate through it, and then close it from C/AL code.

Opening, iterating through, and closing a query

```
WITH SimpleItemQuery DO BEGIN  
    IF OPEN THEN BEGIN  
        WHILE READ DO BEGIN  
            // Do some logic  
        END;  
        CLOSE;  
    END;  
END;
```

Column Access

You can access the columns of a query on a query variable in a manner that is similar to accessing the fields of a table on a record variable. When you read values from columns C/AL, you reference columns exactly as you reference record fields.

The following example shows how to read a value from the Price column of the Simple Item Query.

Accessing a column of a query from C/AL

```
ItemPrice := SimpleItemQry.Price;
```

Programmatically, you can only access those columns that are been defined in the query, but not any other fields that exist in the tables from which the query is constructed.

You can see the list of all columns of a query in the **Symbol Menu**.

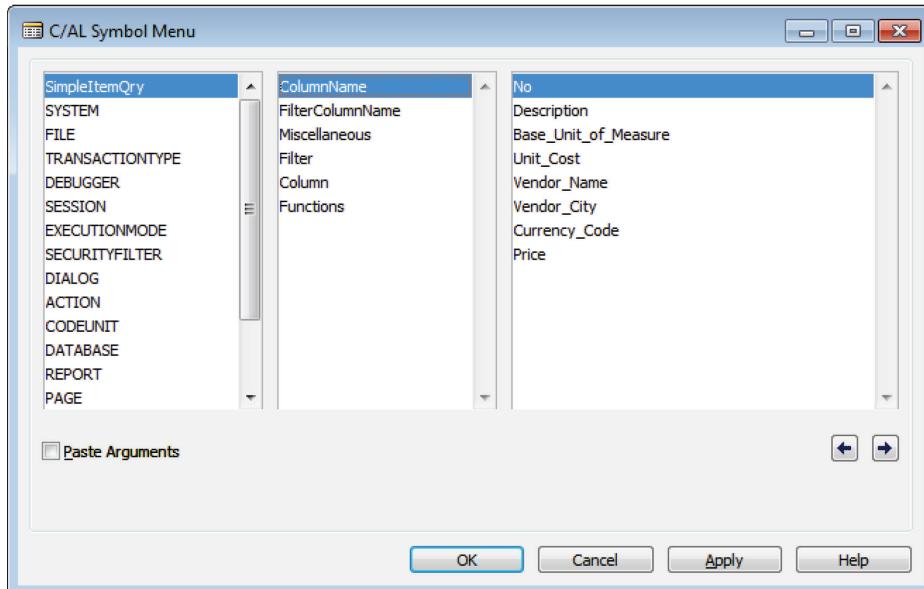


FIGURE 12.21: ACCESSING QUERY COLUMNS IN THE C/AL SYMBOL MENU

Filtering Queries

You can filter the data in queries to narrow the resulting datasets. You can only filter a query on a field which is included as a column or a filter in the query. You use the **SETRANGE** and **SETFILTER** functions to set filters on a query variable.

When you use the **SETFILTER** and **SETRANGE** functions to define a filter on the same field that is already filtered through the **ColumnFilter** property in **Query Designer**, then the filter that is defined in the **ColumnFilter** property is replaced by the filter that you set in the C/AL code.

If you use **SETFILTER** or **SETRANGE** functions on the same field that is included in the **DataItemTableFilter** property of a data item, then the function filter and **DataItemTableFilter** property filter are combined.

The following example shows how to use the **SETFILTER** and **SETRANGE** functions to filter the resulting dataset of a query.

Using SETFILTER and SETRANGE on a query

```
PendingProdOrders.SETRANGE(Status,1,3);

PendingProdOrders.SETFILTER(Due_Date,'>0D|<%1',WORKDATE);

IF PendingProdOrders.OPEN THEN

    WHILE PendingProdOrders.READ DO BEGIN

        Item.GET(PendingProdOrders.Item_No);

        Item.TESTFIELD(Blocked,FALSE);

    END;

PendingProdOrders.CLOSE;
```

 **Note:** Because no data is retrieved before the query is open, referencing columns in any way, including for specifying values of filtering functions, is not allowed. Therefore, in this example, the **SETRANGE** function is called on the Status column by providing the integer, instead of option values. If you specify the range by using **SETRANGE(Status,PendingProdOrder.Status::Planned,PendingProdOrder.Status::Released)** before the query is open, a run-time error occurs.

Calling **SETRANGE** or **SETFILTER** functions on a query that is already open automatically closes the query. To access data from such a query, you must make sure that you call **OPEN** before you call **READ**. The best practice is to set filters before the first call to **OPEN**, and to close the query by calling **CLOSE** immediately after all the rows are read.

You can have multiple calls to the **SETFILTER** function. If the **SETFILTER** function calls set filters on different columns, then the filters are combined and applied to the dataset. If consecutive **SETFILTER** function calls set filters on the same column, then the last **SETFILTER** function call is applied to the column.

TOPNUMBEROFRROWS

When you design a query, you can set the limit for the number of rows that the query returns by specifying its **TopNumberOfRows** property in the **Query Designer** window. At run time, you can check or change the value of this property by using the **TOPNUMBEROFRROWS** function. If you limit the number of rows by defining the **TopNumberOfRows** property, the **TOPNUMBEROFRROWS** function overwrites the **TopNumberOfRows** property.

If the value of the TopNumberOfRows property is undefined, the **TOPNUMBEROFRROWS** function returns 0. If you programmatically set **TOPNUMBEROFRROWS** to 0, all rows are returned.

The following example shows how to programmatically limit the number of rows that are returned from a query to 10 rows.

Calling **TOPNUMBEROFRROWS**

```
SimpleItemQuery.TOPNUMBEROFRROWS := 10;
```

 **Note:** You can call filtering and the **TOPNUMBEROFRROWS** functions on the *CurrQuery* variable from the *OnBeforeOpen* trigger in the query object. *CurrQuery* variable is implicit in the C/AL code of the query object. You do not have to reference it directly.

Saving Result Sets

You can save the resulting dataset of the query to an external file from C/AL. You can use the **SAVEASCsv** function to save the results to a comma-separated values (CSV) file, and use the **SAVEASXML** function to save the results to an XML file.

You can always call **SAVEASCsv** and **SAVEASXML** functions directly, without calling the **OPEN**, **READ** or **CLOSE** functions first. When **SAVEASCsv** or **SAVEASXML** functions are called, the query is implicitly opened, read and then closed. If you call **SAVEASCsv** or **SAVEASXML** functions on a query that is already opened, then the dataset is first retrieved again from the Microsoft Dynamics NAV 2013 database. After the dataset is saved to the file, the query is left in the closed state. This makes any later call to the **READ** function invalid. You must reopen the query to continue reading the data.

 **Best Practice:** Always call **SAVEASCsv** or **SAVEASXML** functions on separate variables. Never call them on variables that are used for iterating through the result set.

Both **SAVEASCsv** and **SAVEASXML** functions return a Boolean value that indicates whether the query was successfully saved. If you omit this optional return value and the query wasn't successfully saved, then a run-time error occurs.

The following example shows how to export a top number of rows in the resulting dataset of a query to an XML file.

Using **TOPNUMBEROFROWS** and **SAVEASXML** functions

```
// Top10ProdOrders is Query 5402  
  
// Text001 is 'The file was not saved, but execution continues. The problem was:  
%1'  
  
Top10ProdOrders.TOPNUMBEROFROWS(5);  
  
IF NOT Top10ProdOrders.SAVEASXML('C:\Temp\Top5Production.xml') THEN  
  
MESSAGE(Text001,GETLASTERRORTEXT);
```

Saving to CSV

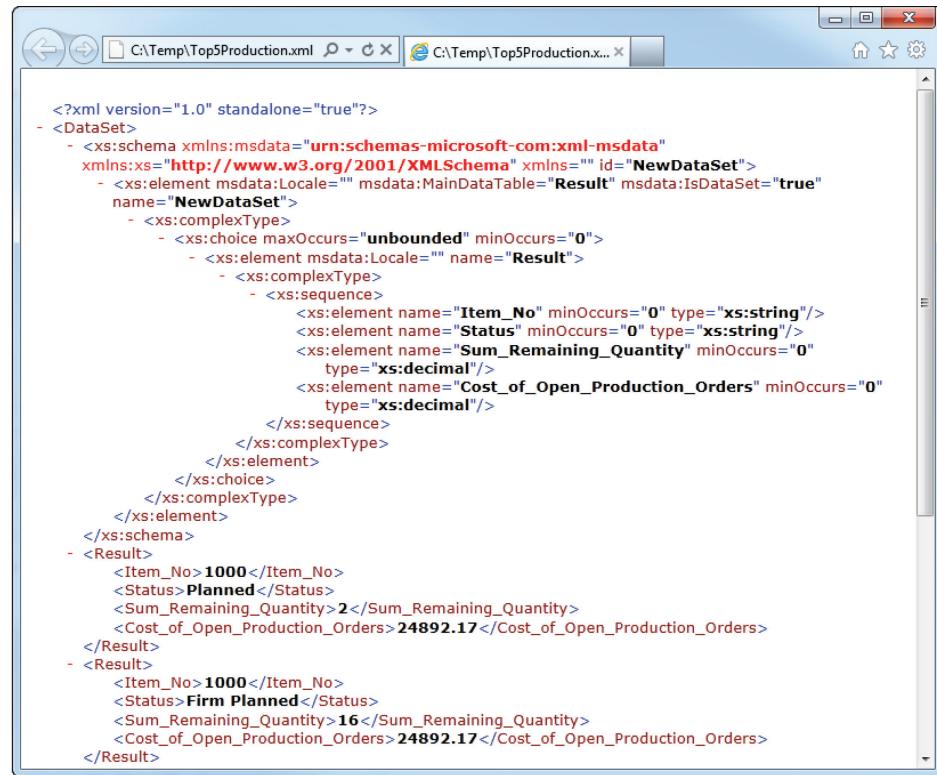
A CSV file stores the data in a plain text format. The files created through the **SAVEASCSV** function resemble the variable text format files that are exported from an XMLport. Each row of data in a CSV file resides in a separate line. The first line of a CSV file that is created through the **SAVEASCSV** function always contains the column names of the query. The column names are specified in the Name property of each column in the definition of the query.

 **Note:** CSV files follow a fixed set of rules that make it easy to import them in other applications, such as Microsoft Office Excel.

Saving to XML

Unlike the XML files that are exported from XMLports, the structure of the XML files that are created by **SAVEASXML** function always follows the same fixed structure. The data in the resulting XML document does not belong to a namespace, the root element is always `<DataSet>`, and each row is represented as a `<Result>` element. Columns are represented as child elements of the `<Result>` element. The name of each element that represents a column is equal to the name of the column as specified in its Name property. The XSD schema that describes the format of the resulting XML file is embedded in the file.

The "XML File Created by SAVEASXML" figure shows the XML file that is created by the **SAVEASXML** function. This includes the embedded schema.



```
<?xml version="1.0" standalone="true"?>
- <DataSet>
  - <xss:schema xmlns:msdata="urn:schemas-microsoft-com:xml-msdata"
    xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="" id="NewDataSet">
    - <xss:element msdata:Locale="" msdata:MainDataTable="Result" msdata:IsDataSet="true"
      name="NewDataSet">
      - <xss:complexType>
        - <xss:choice maxOccurs="unbounded" minOccurs="0">
          - <xss:element msdata:Locale="" name="Result">
            - <xss:complexType>
              - <xss:sequence>
                <xss:element name="Item_No" minOccurs="0" type="xs:string"/>
                <xss:element name="Status" minOccurs="0" type="xs:string"/>
                <xss:element name="Sum_Remaining_Quantity" minOccurs="0"
                  type="xs:decimal"/>
                <xss:element name="Cost_of_Open_Production_Orders" minOccurs="0"
                  type="xs:decimal"/>
              </xss:sequence>
            </xss:complexType>
          </xss:element>
        </xss:choice>
      </xss:complexType>
    </xss:element>
  </xss:schema>
- <Result>
  <Item_No>1000</Item_No>
  <Status>Planned</Status>
  <Sum_Remaining_Quantity>2</Sum_Remaining_Quantity>
  <Cost_of_Open_Production_Orders>24892.17</Cost_of_Open_Production_Orders>
</Result>
- <Result>
  <Item_No>1000</Item_No>
  <Status>Firm Planned</Status>
  <Sum_Remaining_Quantity>16</Sum_Remaining_Quantity>
  <Cost_of_Open_Production_Orders>24892.17</Cost_of_Open_Production_Orders>
</Result>
```

FIGURE 12.22: XML FILE CREATED BY SAVEASXML

Lab 12.2: Using Queries in C/AL

Scenario

Julia, the marketing executive at CRONUS International Ltd., wants to be able to automatically remove any credit limits from the most valuable customers as a part of the new sales and marketing strategy. She wants the function to automatically select the top ten customers and show them in a modal list page. If the modal list page is confirmed, Julia wants it to reset the **Credit Limit (LCY)** field to zero for all the customers in the modal list page. She wants this function to be available from her Role Center.

Isaac, the business software developer at the ISV company that implements Microsoft Dynamics NAV 2013 for CRONUS International Ltd., will create and customize the necessary objects to meet Julia's requirements.

Objectives

- Access queries through C/AL.
- Filter queries programmatically.
- Iterate through query resulting datasets.

Exercise 1: Create a codeunit which uses a query

Exercise Scenario

Isaac, the business developer, creates a codeunit that iterates through the Top 10 Cust. by Revenue query. This codeunit populates a temporary **Customer** table, and then shows the **Customer List** page over this temporary table. When the **Customer List** page is closed, the confirmation dialog box is displayed. The dialog box asks whether the credit limit should be reset to zero. If the user confirms by clicking **Yes**, the codeunit applies the new credit limit to the selected customers.

Task 1: Create a codeunit

High Level Steps

1. Create a new codeunit.
2. Save the codeunit as 50111, **Reset Top 10 Cust. Cred. Limit**.

Detailed Steps

1. Create a new codeunit.
 - a. On the **Tools** menu, click **Object Designer**.
 - b. In the **Object Designer** window, click **Codeunit**.
 - c. Click **New**.

2. Save the codeunit as 50111, **Reset Top 10 Cust. Cred. Limit**.
 - a. On the **File** menu, click **Save**.
 - b. In the **Save As** dialog box, in the **ID** field, enter "50111".
 - c. In the **Name** field, enter "Reset Top 10 Cust. Cred. Limit", and then click **OK**.

Task 2: Declare variables

High Level Steps

1. Declare a global variable for query 123456702, **Top 10 Cust. by Revenue**.
2. Declare a global variable for table 18, **Customer**.
3. Declare a global variable for table 18, Customer and make it temporary.
4. Declare a global text constant Text001, and set its value to "Do you want to reset the %1 to %2 for these customers?"
5. Declare a global text constant Text002, and set its value to "Action completed successfully."

Detailed Steps

1. Declare a global variable for query 123456702, **Top 10 Cust. by Revenue**.
 - a. In the **C/AL Editor** window for the Codeunit 50111, on the **View** menu, click **C/AL Globals**.
 - b. In the **C/AL Globals** window, declare a new variable and set its **Name** to Top10Cust.
 - c. Set its Type to Query, and then set its Subtype to Top 10 Cust. by Revenue.



Detailed Steps: You can also enter "123456702" in the Subtype property.

2. Declare a global variable for table 18, **Customer**.
 - a. In the **C/AL Globals** window, declare a new variable, and set its Name to Cust.
 - b. Set its Type to Record, and then set its Subtype to Customer.
3. Declare a global variable for table 18, Customer and make it temporary.
 - a. In the **C/AL Globals** window, declare a new variable, and set its Name to CustTmp.
 - b. Set its Type to Record, and then set its Subtype to Customer.
 - c. Select the CustTmp variable, and then on the **View** menu click **Properties**.

- d. In the **Properties** window for the CustTmp variable, in the **Temporary** property select **Yes**.
 - e. Close the **Properties** window.
4. Declare a global text constant Text001, and set its value to "Do you want to reset the %1 to %2 for these customers?"
 - a. In the **C/AL Globals** window, on the **Text Constants** tab, declare a new text constant.
 - b. Set the **Name** of the text constant to Text001.
 - c. Set the **ConstValue** of the text constant to "Do you want to reset the %1 to %2 for these customers?"
 5. Declare a global text constant Text002, and set its value to "Action completed successfully."
 - a. In the **C/AL Globals** window, on the **Text Constants** tab, declare a new text constant.
 - b. Set the **Name** of the text constant to Text002.
 - c. Set the **ConstValue** of the text constant to "Action completed successfully."
 - d. Close the **C/AL Globals** window.

Task 3: Iterate through the query

High Level Steps

1. Write code that filters the Top10Cust variable to only include those rows which have Sum_Amount_LCY higher than zero.
2. Write code that iterates through the **Top10Cust** variable.
3. Write code so that each iteration cycle copies the customer information into the **CustTmp** temporary table.

Detailed Steps

1. Write code that filters the Top10Cust variable to only include those rows which have Sum_Amount_LCY higher than zero.
 - a. In the **C/AL Editor** window, add the following code to the OnRun trigger:

```
Top10Cust.SETFILTER(Sum_Amount_LCY,'>%1',0);
```

2. Write code that iterates through the **Top10Cust** variable.
 - a. In the **C/AL Editor** window, add the following code to the OnRun trigger:

```
Top10Cust.OPEN;
```

```
WHILE Top10Cust.READ DO BEGIN  
      
    END;  
  
Top10Cust.CLOSE;
```

3. Write code so that each iteration cycle copies the customer information into the **CustTmp** temporary table.
 - a. In the **C/AL Editor** window, enter the following code to the BEGIN/END block of the iteration.

```
Cust.GET(Top10Cust.No);  
  
CustTmp := Cust;  
  
CustTmp.INSERT;
```

Task 4: Apply the business rule

High Level Steps

1. Add a function that resets the credit limit for all customers in the **CustTmp** temporary variable, and then displays the **Text002** as an information message. Set the function name to **ResetCreditLimit**.
2. Add a function that asks for a confirmation by showing the text constant **Text001**. If it is confirmed, it runs the **ResetCreditLimit** function. Set the function name to **ConfirmReset**.
3. Show the top ten customer list as a modal page. If the user clicks **OK** in the page, run the **ConfirmReset** function.
4. Compile and save the codeunit.

Detailed Steps

1. Add a function that resets the credit limit for all customers in the **CustTmp** temporary variable, and then displays the **Text002** as an information message. Set the function name to **ResetCreditLimit**.
 - a. In the **C/AL Editor** window, on the **View** menu, click **C/AL Globals**.
 - b. In the **Functions** tab, declare a new function named **ResetCreditLimit**.
 - c. Close the **C/AL Globals** window.
 - d. Enter the following code in the **ResetCreditLimit** function trigger.

```
IF CustTmp.FINDSET THEN  
  
REPEAT
```

```

Cust.GET(CustTmp."No.");
Cust."Credit Limit (LCY)" := 0;
Cust.MODIFY;
UNTIL CustTmp.NEXT = 0;
MESSAGE(Text002);

```

2. Add a function that asks for a confirmation by showing the text constant Text001. If it is confirmed, it runs the **ResetCreditLimit** function. Set the function name to ConfirmReset.
 - a. In the **C/AL Editor** window, on the **View** menu, click **C/AL Globals**.
 - b. In the **Functions** tab, declare a new function named ConfirmReset.
 - c. Close the **C/AL Globals** window.
 - d. Enter the following code in the **ConfirmReset** function trigger.

```

IF CONFIRM(Text001,
FALSE,
Cust.FIELDCAPTION("Credit Limit (LCY"),
0)
THEN
ResetCreditLimit;

```

3. Show the top ten customer list as a modal page. If the user clicks **OK** in the page, run the **ConfirmReset** function.
 - a. Append the following code to the end of the OnRun trigger.

```
IF PAGE.RUNMODAL(PAGE::"Customer List",CustTmp) = ACTION::LookupOK THEN
```

```
    ConfirmReset;
```

4. Compile and save the codeunit.
 - a. Click File > Save.
 - b. In the **Save** dialog box, make sure that the **Compiled** check box is selected, and then click **OK**.
 - c. Close the **C/AL Editor** window.

Task 5: Test the codeunit

High Level Steps

1. Run the codeunit to verify its functionality.

Detailed Steps

1. Run the codeunit to verify its functionality.
 - a. In Object Designer, select the codeunit 50111, Reset Top 10 Cust. Cred. Limit.
 - b. Click **Run**.
 - c. Verify that the confirmation dialog is displayed. Click **Yes**.
 - d. Verify that the information message is displayed. Click **OK**.

Advanced Query Concepts

When you use queries, it helps if you understand how accessing queries in the C/AL code relates to accessing tables. Also, it helps if you understand how queries map to Transact-SQL statements.

Queries and Records

When you are retrieving data, you can use records and queries and iterate through datasets in a similar manner. The C/AL functions that you use with records and queries are not the same. However, they achieve the same goal.

The following table shows a side-by-side comparison of the C/AL functions that you use when you access data with records and queries.

Goal	Record	Query
Initiate iteration through a dataset.	FIND('-') FINDSET	OPEN
Retrieve the next record from a dataset.	NEXT	READ
Close the dataset.	Not needed	CLOSE

Retrieving data with records and queries is very different. With records, the **FIND('')** and **FINDSET** functions also retrieve the first record, whereas with queries, the **OPEN** function merely translates the query model into a Transact-SQL statement and runs it against Microsoft SQL Server. With records, after the initial **FIND('')** or **FINDSET**, the record is already retrieved, and you can immediately access the fields. With queries, you must call **READ** one time before you can access the columns. Because of this, you write the iteration loops differently. With records, you use the **REPEAT..UNTIL** compound statement. With queries, you use the **WHILE..DO** compound statement.

C/SIDE Introduction in Microsoft Dynamics® NAV 2013

The following table shows the comparison between record and query iteration loops.

Record Iteration	Query Iteration
IF FIND('') THEN REPEAT // Do some processing UNTIL NEXT;	IF OPEN THEN WHILE READ DO BEGIN // Do some processing END;

Except for the syntactical differences in how you iterate through datasets with records and queries, there are also functional differences between retrieving records by using FIND('') or FINDSET, and using queries. Because of those differences, you cannot directly replace traditional record-based iteration with queries.

The following table summarizes different data access scenarios, and makes recommendations about the data access feature of Microsoft Dynamics NAV 2013 that you should use.

Scenario	Use	Remarks
Reading a partial set of data from a single table	FIND('')	FIND('') retrieves the first 50 rows, and requests the rest only if you must read beyond the initial 50 rows. If your iteration does not have to read beyond the initial 50 rows, use FIND('').
Reading a complete set of data from a single table	FINDSET	FINDSET retrieves all rows, regardless of the size of the table.
Modifying data	FIND('') or FINDSET	Queries cannot handle the writing of data. Therefore, when you must write data back, you must use either FIND('') or FINDSET.

Module 12: Queries

Scenario	Use	Remarks
Repetitive access to same datasets	FIND('') or FINDSET	Resulting datasets of queries are never cached. Therefore, when repetitively accessing the same datasets, use FIND('') or FINDSET to take advantage of Microsoft Dynamics NAV 2013 caching capabilities.
Reading from multiple tables	Query	A query translates into a single Transact-SQL statement that returns a single dataset. Reading from multiple tables with records requires multiple, usually nested, iterations. These iterations frequently decrease performance.
Reading a limited subset of fields from multiple tables, especially if Microsoft SQL Server can use a covering index strategy	Query	FIND('') and FINDSET retrieve and return all fields from the table. This makes them slower and requires more resources. Query only retrieves and returns those fields that are declared as columns in the query data model. A covering index includes all of the fields that a query requires, so Microsoft SQL Server can only access the index without even accessing the table itself.

Scenario	Use	Remarks
Reading summarized data (filtered and aggregated data)	Query	Queries can take advantage of Transact-SQL aggregation functions and other capabilities to return summarized data more efficiently than reading and iterating through the sets to perform aggregations.
Handling large quantities of data in read-only mode	Query	When you read large quantities of data, queries provide better performance than records.

Mapping Queries to Transact-SQL

When Microsoft Dynamics NAV 2013 runs a query, it first translates it into a Transact-SQL statement, and then runs that statement against the underlying Microsoft SQL Server database. One benefit of queries is that they enable you to model a dataset in a user-friendly way. Queries do not require knowledge of Transact-SQL querying language, or expertise in Microsoft SQL Server.

However, if you are familiar with Transact-SQL, you may want to understand exactly how the query features of Microsoft Dynamics NAV 2013 map to Transact-SQL.

The following table summarizes the elements and properties of query objects, and how they relate to Transact-SQL.

Transact-SQL	Query Feature
SELECT	Row of Type Column in the Query Designer window
FROM	Row of Type DataItem in the Query Designer window
JOIN type	DataItemLinkType and SQLJoinType query properties
ON	DataItemLink data item property
WHERE	DataItemTableFilter data item property ColumnFilter property of columns and filters Row of Type Filter

Transact-SQL	Query Feature
HAVING	ColumnFilter property of columns and filters, when aggregation is used
GROUP BY	Automatically switched on for each row of Type Column, when aggregation is used
ORDER BY	OrderBy query property
TOP	TopNumberOfRows query property

Analyzing Queries with SQL Server Profiler

You can use SQL Server Profiler that is included with Microsoft SQL Server to analyze queries and how they translate to Transact-SQL.

The "Query 760, Trailing Sales Order Qry." figure shows the data model of the query 760, Trailing Sales Order Qry. in the **Query Designer** window.

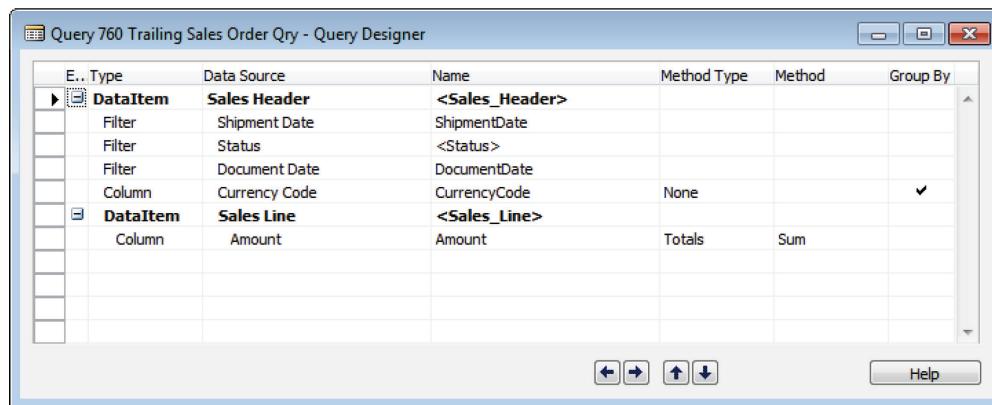


FIGURE 12.23: QUERY 760, TRAILING SALES ORDER QRY.

When you run this query, Microsoft Dynamics NAV 2013 translates the query data model into a Transact-SQL SELECT statement and executes it against the underlying Microsoft SQL Server database.

You can use SQL Server Profiler to capture the Transact-SQL translation of the query 760. Simplified, the Transact-SQL translation of this query looks as follows.

Simplified Transact-SQL Translation of a Query Data Model

```
SELECT
    "Sales_Header"."Currency Code" AS "CurrencyCode",
    SUM("Sales_Line"."Amount") AS "Amount"
FROM
    "CRONUS International Ltd_$Sales Header" AS "Sales_Header"
JOIN
    "CRONUS International Ltd_$Sales Line" AS "Sales_Line"
ON
    ("Sales_Line"."Document Type"="Sales_Header"."Document Type" AND
     "Sales_Line"."Document No_"="Sales_Header"."No_")
WHERE
    ("Sales_Header"."Document Type"= 1) AND
    ("Sales_Line"."Amount"<> 0)
GROUP BY
    "Sales_Header"."Currency Code"
ORDER BY
    "Sales_Header"."Currency Code" ASC
```

Module Review

Module Review and Takeaways

There are many situations in which iterative data access is incorrect. Microsoft Dynamics NAV 2013 can tap into the powerful relational database management engine of Microsoft SQL Server to select complex, multi-table sets of data in a fast and efficient way.

The Query object of Microsoft Dynamics NAV 2013 enables you to define relational data models that are translated into efficient SELECT statements. Microsoft SQL Server can execute these statements as single data retrieval operations. This guarantees optimal performance and minimal pressure on system resources.

In Microsoft Dynamics NAV 2013, you can use queries to do any of the following:

- Define data sources for charts.
- Export data as CSV or XML files.
- Publish them as OData web services.
- Iterate through the result sets from the C/AL code. In this case queries eliminate the need for multiple, nested data iteration loops.

Test Your Knowledge

Test your knowledge with the following questions.

1. Which of the following examples are valid use case scenarios for queries?
 - () Source for pages or data items on reports; data export through XMLports; iterating through rows of data in C/AL.
 - () Table relations in tables, source for pages, source for Charts in the RoleTailored client.
 - () Source for Charts in the RoleTailored client, source of OData web services, iterating through rows of data in C/AL, and exporting result sets as XML or text.
2. What property do you need to set and to what value, to specify that a row in the parent data item is skipped if it contains no child rows?

C/SIDE Introduction in Microsoft Dynamics® NAV 2013

3. How do you specify that the query should only return the year from a date column?

4. Which is not the valid Totals method for a Decimal column in a query?

- () Sum
- () Count
- () Average
- () Min
- () Exists

5. Which is a valid trigger in a query object?

- () OnBeforeOpen
- () OnBeforeOpenQuery
- () OnOpen
- () OnRun
- () OnBeforeRun

6. You can programmatically filter query results by using SETRANGE and SETFILTER functions.

- () True
- () False

7. Which function do you use to save query results into an XML file?

Module 12: Queries

8. Which function do you use to save query results into a delimited text file?

Test Your Knowledge Solutions

Module Review and Takeaways

1. Which of the following examples are valid use case scenarios for queries?
 - () Source for pages or data items on reports; data export through XMLports; iterating through rows of data in C/AL.
 - () Table relations in tables, source for pages, source for Charts in the RoleTailored client.
 - (√) Source for Charts in the RoleTailored client, source of OData web services, iterating through rows of data in C/AL, and exporting result sets as XML or text.
2. What property do you need to set and to what value, to specify that a row in the parent data item is skipped if it contains no child rows?

MODEL ANSWER:

Set DataItemLinkType to Exclude Row If No Match.

3. How do you specify that the query should only return the year from a date column?

MODEL ANSWER:

Set the Method Type to Date, and Method to Year.

4. Which is not the valid Totals method for a Decimal column in a query?

- () Sum
- () Count
- () Average
- () Min
- (√) Exists

5. Which is a valid trigger in a query object?

- () OnBeforeOpen
- () OnBeforeOpenQuery
- () OnOpen
- () OnRun
- () OnBeforeRun

6. You can programmatically filter query results by using SETRANGE and SETFILTER functions.

- () True
- () False

7. Which function do you use to save query results into an XML file?

MODEL ANSWER:

SAVEASXML

8. Which function do you use to save query results into a delimited text file?

MODEL ANSWER:

SAVEASCsv

