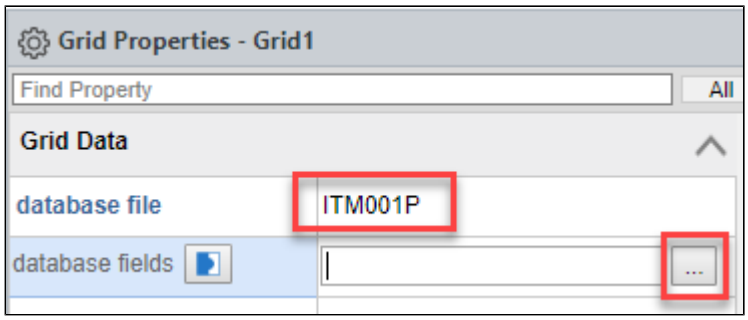
Database-driven Grids

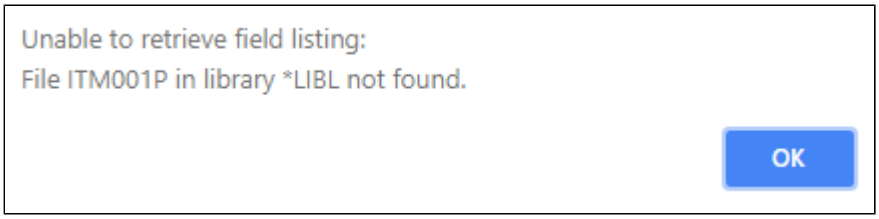
Data from database file

In this section, we will populate the Grid using a database file. In the illustration below, we are using an item master file named ITM001P for the database file property. The file name can be qualified with a library; however, if the library is omitted, the session's library list will be used to find the database file.

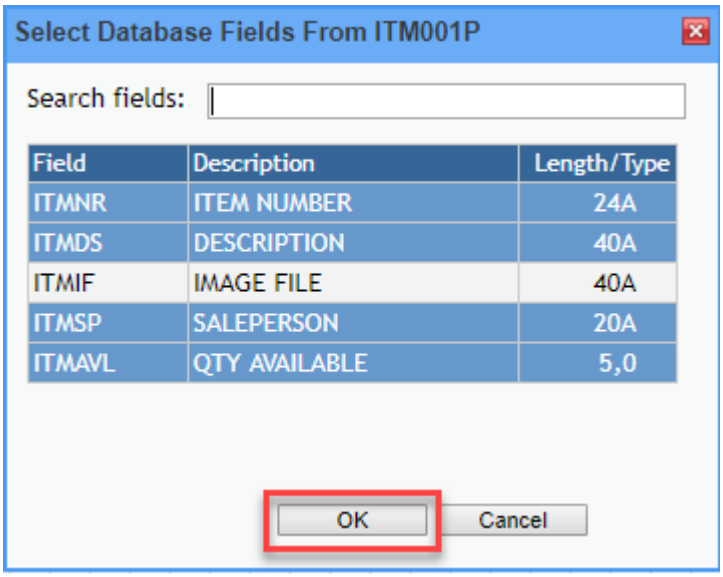
Next, you must specify the "database fields" property, which will be used to determine the database fields from which the data will be populated. You can type the field names separated by commas or choose them from a dialog. Click on the button circled in green to display the database fields dialog for the database file you specified.



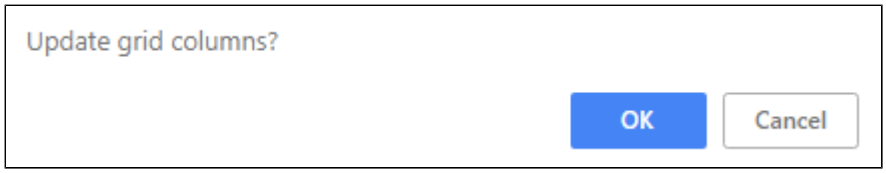
Note: Make sure that the library in which the database file exists is included within your session's library list. If the library does not exist, you will get a message that looks like this:



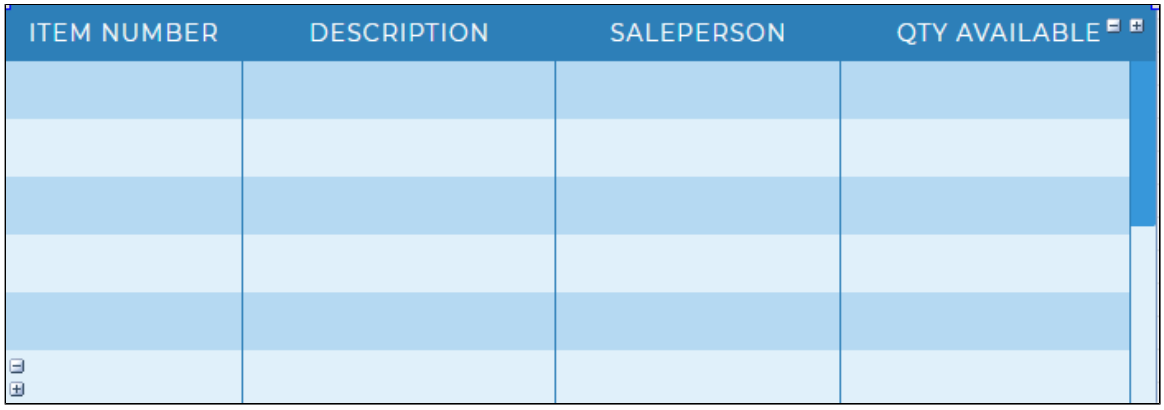
Otherwise, you will get a dialog with the available database field names to pick from. Select the appropriate fields from the list:



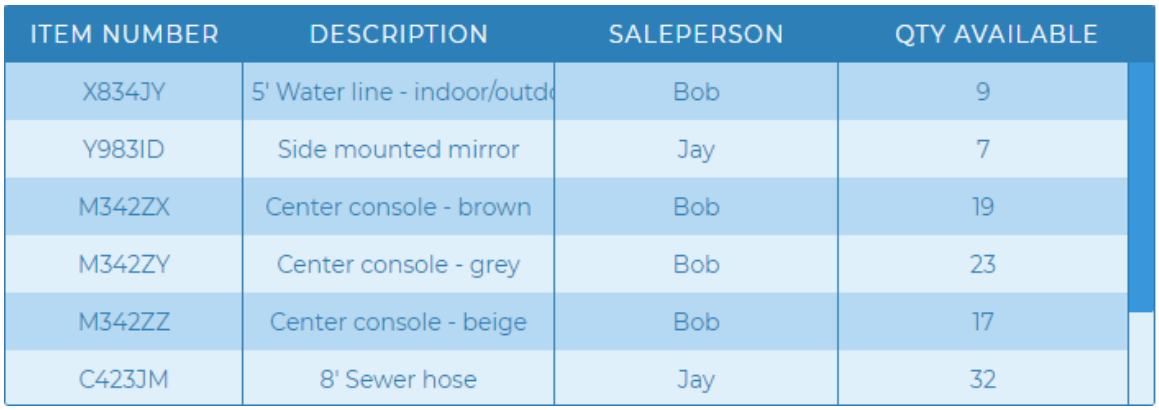
When you click OK, you will be presented with a dialog asking you if you want to update the grid columns based on the fields that you selected. If you click OK, the grid column headings will be updated.



The column headings should now appear on your Grid as in the screenshot below.

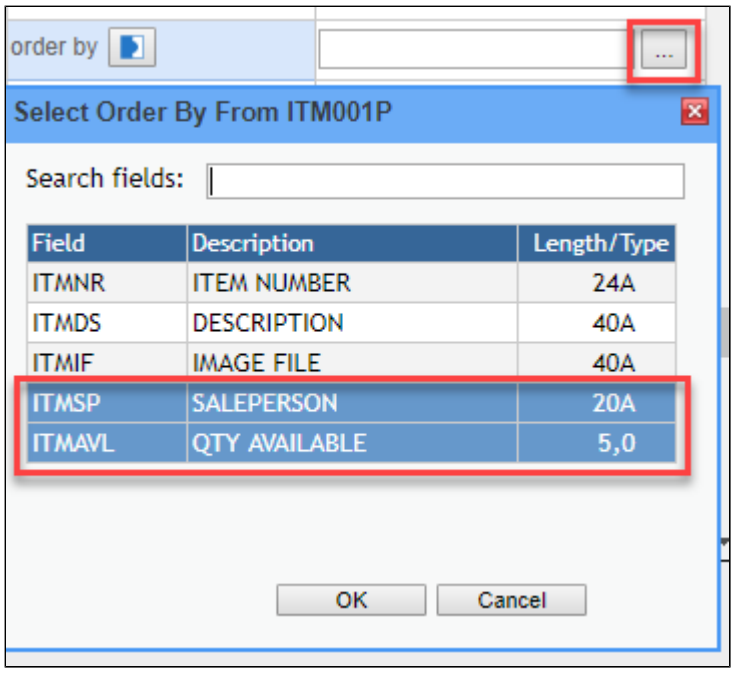


Save your work and exit Design mode to see the data populated in the Grid as in the screenshot below.



ITEM NUMBER	DESCRIPTION	SALEPERSON	QTY AVAILABLE
X834JY	5' Water line - indoor/outd	Bob	9
Y983ID	Side mounted mirror	Jay	7
M342ZX	Center console - brown	Bob	19
M342ZY	Center console - grey	Bob	23
M342ZZ	Center console - beige	Bob	17
C423JM	8' Sewer hose	Jay	32

Additionally, you can sort the data on certain columns by specifying the database fields that determine the order of the records in the “order by” property. See below.



This is the final result after the data was ordered by Sales Person and Quantity Available.

ITEM NUMBER	DESCRIPTION	SALEPERSON	QTY AVAILABLE
X834JY	5' Water line - indoor/outd	Bob	9
M342ZZ	Center console - beige	Bob	17
M342ZX	Center console - brown	Bob	19
M342ZY	Center console - grey	Bob	23
Y983ID	Side mounted mirror	Jay	7
C423JN	10' Sewer hose	Jay	10

Data from a custom SQL statement

In this section, we will populate the Grid using the same database file we used in the previous section (ITM001P). However, this time, we will use an alternative method to populate the data into our Grid: a custom SQL statement. In the SQL, the file name can be qualified with a library; however, if the library is omitted, the session's library list will be used to find the database file.

In the Grid's "custom sql" property, type in the SQL statement, such as the one shown below:

SELECT ITMNR, ITMDS, ITMSP, ITMAVL FROM ITM001P

custom sql	SELECT ITMNR, ITMDS, ITMSP, ITMAVL FROM ITM001P

Save your work and exit Design Mode. Now the Grid should be populated with the same data as shown in the previous example *(Data from database file).* Additionally, we can also add an ORDER BY command to our SQL statement to order the columns by Sales Person and Quantity Available as shown previously. Now, the SQL statement becomes:

SELECT ITMNR, ITMDS, ITMSP, ITMAVL FROM ITM001P ORDER BY ITMSP, ITMAVL

Save the work and exit Design Mode to see the results. You should get the same result as before. Furthermore, we can add a WHERE clause to the SQL statement to return the items for which the Quantity Available is 15 or more:

SELECT ITMNR, ITMDS, ITMSP, ITMAVL FROM ITM001P WHERE ITMAVL >= 15 ORDER BY ITMSP, ITMAVL

ITEM NUMBER	DESCRIPTION	SALEPERSON	QTY AVAILABLE
M342ZZ	Center console - beige	Bob	17
M342ZX	Center console - brown	Bob	19
M342ZY	Center console - grey	Bob	23
C423JM	8' Sewer hose	Jay	32

Note: Only SELECT statements are allowed in the "custom sql" property.

Filtering and Sorting Limitations with Custom SQL

Support for filtering and sorting were added to Custom SQL grids in Profound UI Version 5, Fix Pack 6.0. There are some limitations on what SQL query strings will work with grid filters and sorting:

- The query string cannot have an Order By clause at the end of the main query. Instead, the grid's "*order by*" property should be used.
- If the query string contains a sub-query with a WHERE clause, then the outer-most query must contain a WHERE clause.
- If the SELECT clause creates computed columns, then those columns must use aliases. (See section below, Filtering with Computed Columns.)

These limitations are due to our implementation in filtering and sorting. Rather than being able to parse and manipulate every possible query string, we filter

and sort by adding SQL code to the end of the custom SQL string:

- Filters are added to the end of the custom query as WHERE and AND clauses.
  - If the string ' WHERE ' (case-insensitive) exists in the custom SQL string, then we add an AND clause to the end of the SQL string for each filter.
  - Else, we add one WHERE clause for the first filter and AND clauses for additional filters.
- An ORDER BY clause is added to the end of the custom SQL to handle the column the user clicked to sort or to handle the "*order by*" property.

This simple design allows us to support many SELECT queries without the added overhead of a custom SQL parser.

Data from a program or web service

In Genie, Grids can also be populated using an external program. This program or web service can be written in any web capable language and hosted on either the IBM i or on any other platform.

Since the Grid expects JSON as its data format, the program or web service needs to send the data to the Grid in that format or data interchange. This can be accomplished by calling the program that generates the JSON data from within Genie through a URL.

From the properties dialog window, find the “data url” property and type in the URL of the program that you wish to call to send the JSON data as shown in the screenshot below:

data url	/genie/getData.pgm

The “data url” property sets the URL to a program or web service that returns the Grid’s columns definition and data in JSON format.

Here is a sample of the JSON data that is sent to the Grid through the program/web service to build and populate the Grid.

```
{"success":true, "response":{"colWidths":[24,40,20,5],"totalRecs":8,
  "results":
      [{"ITMNR":"H622AE","ITMDS":"Tent stakes","ITMSP":"Bob","ITMAVL":"25"},
       {"ITMNR":"I134NM","ITMDS":"Table clips","ITMSP":"Nancy","ITMAVL":"20"},
       {"ITMNR":"M342ZY","ITMDS":"Center console - grey","ITMSP":"Bob","ITMAVL":"30"},
       {"ITMNR":"M342ZZ","ITMDS":"Center console - beige","ITMSP":"Bob","ITMAVL":"40"},
       {"ITMNR":"KH19O3","ITMDS":"Tennis Shorts","ITMSP":"Jim","ITMAVL":"23"},
       {"ITMNR":"JD23ZZ","ITMDS":"Tennis Racquets","ITMSP":"Hany","ITMAVL":"9"},
       {"ITMNR":"UI439OS","ITMDS":"Babolat Shoes","ITMSP":"Hany","ITMAVL":"4"}
      ]}
}
```

Using Custom Web Services

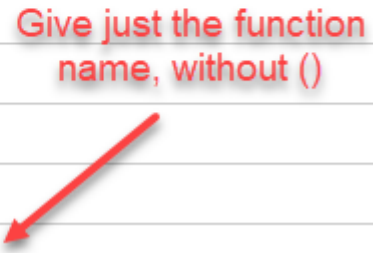
The grid's "data transform function" property allows use of a custom web service with the "data url" property. If you specify a JavaScript function name for this property, it will be called each time the grid receives data from the web service. The function will be passed a single argument that contains the HTTP response text from the web service. The function must return a JavaScript object conforming to the above pattern. The following example shows how the function can be used to transform XML data from a custom web service.

Example XML Data from Web Service

```
<root total="100">
  <row>
    <columnName1>Data</columnName1>
    <columnName2>Data</columnName2>
    <columnName3>Data</columnName3>
 </row>
 <row>
   <columnName1>Data</columnName1>
   <columnName2>Data</columnName2>
   <columnName3>Data</columnName3>
 </row>
</root>
```

The property can be set like this:

Grid Data	
database file	
database fields	
selection criteria	
order by	
custom sql	
allow any select statement	
parameter value	
data url	/grid.php
data transform function	transform



"transform()" Function Code

```
function transform(responseText) {

  var parser = new DOMParser();
  var xml = parser.parseFromString(responseText, "text/xml");
  var root = xml.documentElement;
  var transformed = {};
  transformed["totalRecs"] = root.getAttribute("total");
  var results = transformed["results"] = [];
  var rows = root.childNodes;
  for (var rowNum = 0; rowNum < rows.length; rowNum++) {

    var row = rows[rowNum];
    var cols = row.childNodes;
    var obj = {};
    for (var colNum = 0; colNum < cols.length; colNum++) {

      var col = cols[colNum];
      obj[col.nodeName] = (col.childNodes && col.childNodes.length == 1) ?
col.firstChild.nodeValue : "";

    }
    results.push(obj);

  }

  return transformed;

}
```

Filtering with Computed Columns

Both data-from-database-file and Custom-SQL type grids allow developers to specify SQL expressions as columns, resulting in computed columns. If the grid has the Filter feature enabled, then these computed columns must use an alias; otherwise, filtering will fail on the computed column.