

Introduction to SQL

Prepared by La Habra Tech

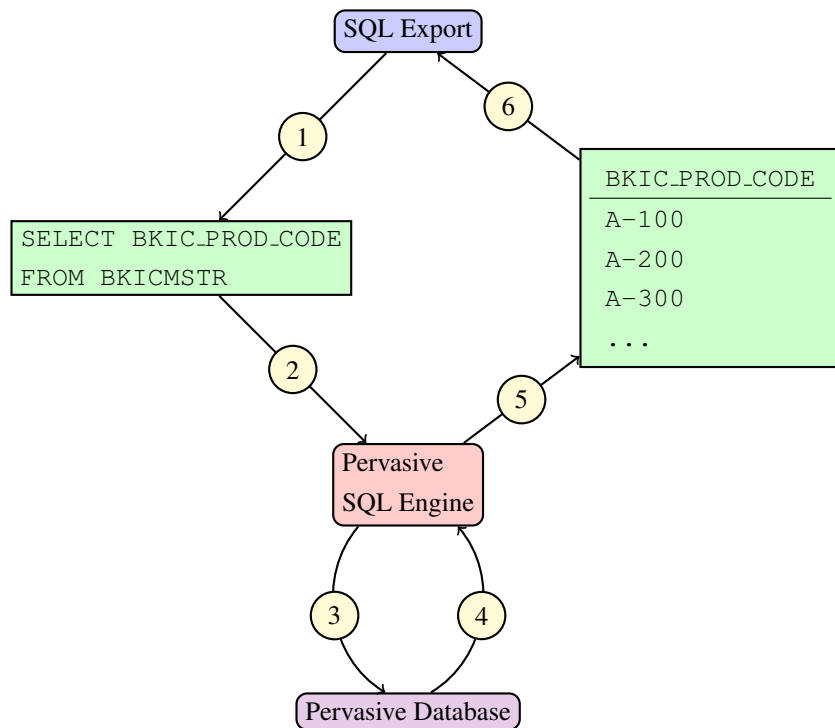
Contents

1	SQL Export	2
1.1	Program Overview	3
2	The Structure of an SQL Query	4
2.1	Primary Clauses	4
2.2	Example Queries	5
2.3	The ORDER BY Clause	6
2.4	Aliases	7
3	Operations and Functions	7
3.1	Aggregate Functions	8
4	Joins	8
4.1	Example Queries	9
4.2	Inner Joins and Outer Joins	10
5	The Query Wizard	10
5.1	Selecting Tables	11
5.2	Selecting Fields	11
5.3	Joining Tables	13
5.4	Adding Filters	14
6	Query Execution	15
7	Charts	16
8	Query Assistance and Custom Queries	18

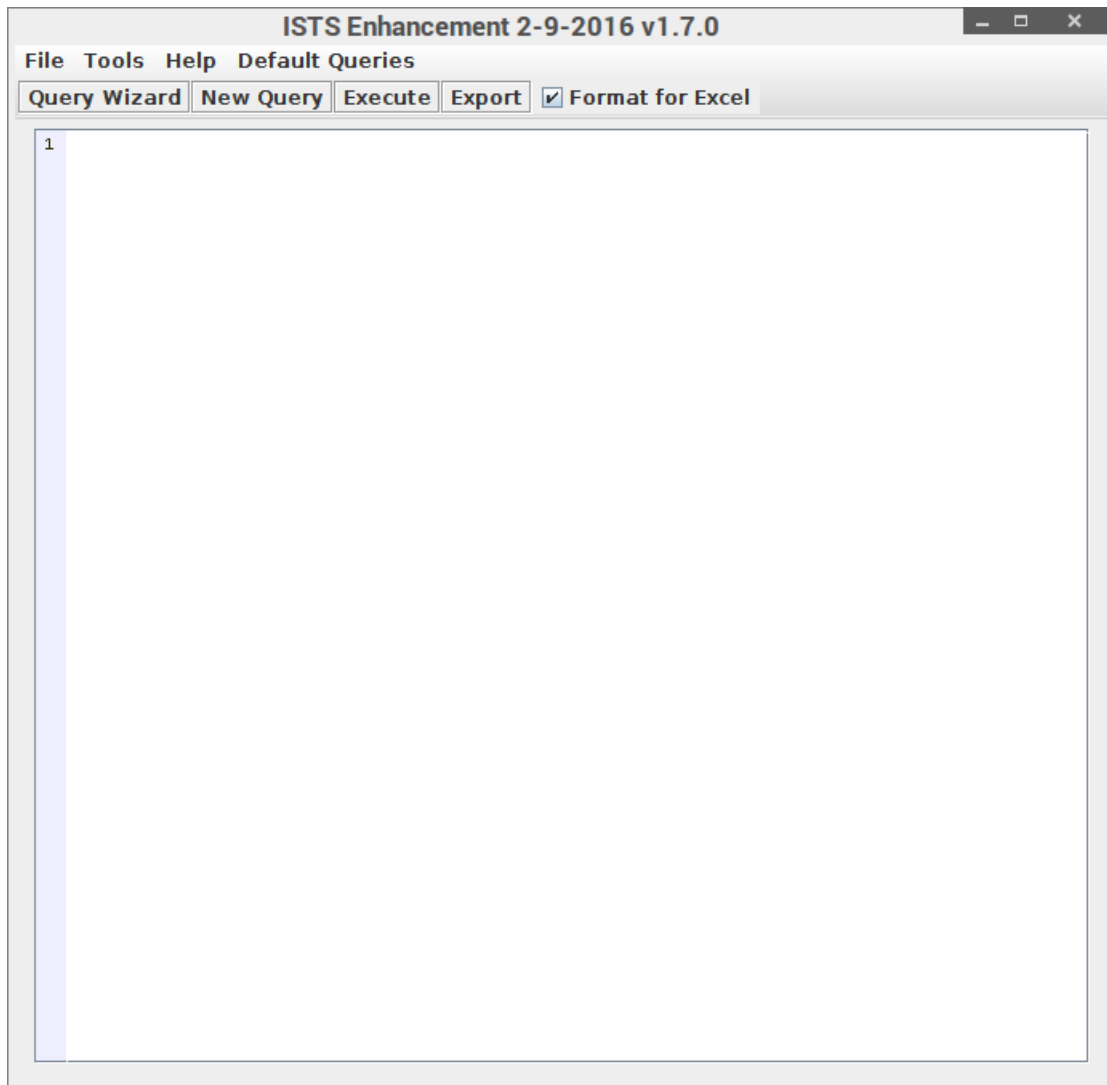
1 SQL Export

This program allows direct access to the underlying database used by Evo-ERP and DBA classic via the Structured Query Language (SQL). Conceptually, the program “speaks” SQL to Pervasive’s relational SQL engine, which is a service running on the server where Pervasive is installed. The engine can interpret the SQL statements and then perform the appropriate actions on the database, usually returning the result of a query back to the program. Because the program communicates directly with Pervasive, it has no understanding about concepts in Evo-ERP other than the structure of the tables in the database. Therefore, to use it properly it is necessary to have some understanding about the table structures employed by the Evo-ERP database.

The figure below shows the overall process of executing a query starting with the SQL export program sending the query text to the engine, the engine processing the query with the database, and finally the engine returning the query result back to the SQL export program.



1.1 Program Overview



The main screen in the program presents a text area for entering SQL statements and a tool bar to perform the following actions:

Query Wizard This opens the query wizard, which will assist in constructing a query if you are not familiar with the syntax of SQL. It does not include all features of SQL but has enough to construct many common queries. Once done with the query wizard, it will overwrite the content of the SQL text area with the query it generates. It does prompt to save before starting, so if you have a query already in the text area you should save it.

New Query This will clear the SQL text area for starting a new query.

Execute This attempts to execute the query present in the text area. If the SQL engine finds a problem with your query, you will get back an error that shows you approximately where it got stuck interpreting the query. If execution is successful, you will be presented with a new screen displaying the result of the query and allowing

more options.

Export This is a recently added feature that allows direct export of the result of a query without the intermediate result screen. We added this because the result screen requires storing the entire result of the query in memory, so it did not work properly for queries with very large result sets. This export option will go through the result in small pieces writing to the specified output file so there is no memory issue.

Format for Excel This setting should be used if you plan to open the file exported by the Export button in Excel. The Export format is a CSV file, however Excel sometimes interprets items in an unintended way, such as part numbers that happen to have two hyphens as a date (e.g., 10-10-1000). By selecting this option, we can wrap such items that we know are not dates to make Excel treat them as normal strings of characters, e.g., the previous part number would be exported as ="10-10-1000". In this way, Excel treats the value as a string of characters.

Along with the tool bar, there are several menus:

File Allows opening existing queries and saving queries from the text area. The queries are stored as simple text files with the .sql extension.

Tools Allows starting the Variable Query Wizard, which is for constructing queries in which you allow other users to enter specific values such as dates without being able to write complete queries themselves.

Default Queries This contains a list of existing queries provided by IS Tech Support that may be useful for many customers. They generally contain comments that describe their purpose and often require modifying a specific portion such as a date range in order to execute properly.

If you would like your own queries to appear in this list, place them in the `DefaultSQL` folder in the `EvolutionERP/DBAMFG` folder. When this program runs it scans that folder to find the queries to load. The name in the program is taken from the file name and the content of the file is copied in to the text area when the query is selected. You can also generate submenus here by placing your queries in subdirectories under `DefaultSQL`.

2 The Structure of an SQL Query

An SQL query is the text that gets sent to the Pervasive SQL engine to access the database. The queries must follow the syntax of SQL in order for the engine to properly interpret them. If you don't follow the language syntax, it would be like speaking Spanish to someone who can only speak Chinese; you won't get anywhere. The basics of SQL are presented below, but if you want an in-depth look at Pervasive's brand of SQL we recommend the book *The SQL Guide to Pervasive PSQL* by Rick van der Lans, which is available on Amazon at: <http://www.amazon.com/SQL-Guide-Pervasive-PSQL/dp/0557105439>.

2.1 Primary Clauses

In most queries you construct, there will be three primary clauses: the `SELECT` clause, the `FROM` clause, and the `WHERE` clause. Although they are written in that order, the SQL engine uses them in the following order:

1. The `FROM` clause tells the SQL engine which tables the query uses. This is the first thing the engine must figure out. For example, if your `FROM` clause says

```
FROM BKICMSTR
```

then the engine will only look at the `BKICMSTR` table.

2. The `WHERE` clause contains filters the engine uses to pick the proper results. For example, you may have a `WHERE` clause that says

```
WHERE BKAR_INV_ORDDTE >= '2015-01-01'
```

to limit your result to only include entries with order date in 2015 or later.

If you wanted to limit to only order dates within 2015, we would use the term `AND` and add a second filter:

```
WHERE BKAR_INV_ORDDTE >= '2015-01-01'
AND   BKAR_INV_ORDDTE <= '2015-12-31'
```

3. The `SELECT` clause is the last clause used by the engine. It strips away any columns in the table not explicitly mentioned. For example, if your `SELECT` clause says

```
SELECT BKIC_PROD_CODE, BKIC_PROD_DESC
```

then although the table has many other columns, only the values in those two columns are included in the result. You can also use the wildcard

```
SELECT *
```

to select all columns in a table.

2.2 Example Queries

Here is an example query that selects the part number, description, on hand quantity, and unit cost for all parts of type “F” (finished goods).

```
SELECT BKIC_PROD_CODE, BKIC_PROD_DESC,
       BKIC_PROD_UOH, BKIC_PROD_AVGC
FROM   BKICMSTR
WHERE  BKIC_PROD_TYPE = 'F'
```

Here is an example that gets more complicated: querying from multiple tables. In order to do this, you must match the keys of the table in the `WHERE` clause (more later). This query matches invoices to invoice lines and gets the customer name, item shipped, description, quantity, extension, and cost for anything shipped in 2015.

```
SELECT BKAR_INV_CUSNME, BKAR_INVL_PCODE,
       BKAR_INVL_PDESC, BKAR_INVL_PQTY,
       BKAR_INVL_PEXT, BKAR_INVL_PCOGS
```

```

FROM    BKARHINV, BKARHIVL
WHERE   BKAR_INV_NUM = BKAR_INVL_INVNM
AND     BKAR_INV_INVDTTE >= '2015-01-01'
AND     BKAR_INV_INVDTTE <= '2015-12-31'

```

Notice that when we wanted to insert a *literal value* (e.g., the type “F”) in the query, we had to place it between single-quotes. This is because otherwise, the engine would think you were trying to match on a field called F. The single-quotes inform the SQL engine that they contain a literal string of characters that should be compared. For dates, we must enter them as strings of characters in the format YYYY-MM-DD to be properly interpreted. For numbers, we enter them directly without single-quotes. This is because a field name can’t be a number, so the engine won’t get confused about whether you meant the number itself or a field with that name.

2.3 The ORDER BY Clause

There are some additional clauses that can be included in an SQL query. For now, we will look at one: the ORDER BY clause. Normally, when the engine gives back the result of a query, the order of the rows is effectively random – it’s however the database stores them in the files on the hard drive. Because entries are added and removed from the tables over time and new values are sometimes inserted to fill gaps for efficiency reasons, the engine won’t guarantee the order of the results.

To solve this, we can explicitly tell the engine to reorder the result before giving it back. We do this by inserting an ORDER BY clause after our WHERE clause. For example, if we want to query all of our part numbers and their descriptions that have type “O”, the query would be:

```

SELECT BKIC_PROD_CODE, BKIC_PROD_DESC
FROM    BKICMSTR
WHERE   BKIC_PROD_TYPE = 'O'

```

However, if we want the result to be in alphabetical order by part number, we must also include the appropriate ORDER BY clause:

```

SELECT    BKIC_PROD_CODE, BKIC_PROD_DESC
FROM      BKICMSTR
WHERE     BKIC_PROD_TYPE = 'O'
ORDER BY BKIC_PROD_CODE

```

If we want it sorted in descending order instead of ascending order, we can inform the engine by adding the term DESC after the field:

```

SELECT    BKIC_PROD_CODE, BKIC_PROD_DESC
FROM      BKICMSTR
WHERE     BKIC_PROD_TYPE = 'O'
ORDER BY BKIC_PROD_CODE DESC

```

If we wanted instead to select all parts but sort them by type, we would use the following. Note that we can simply remove the WHERE clause if we don’t want to the engine to filter the results.

```

SELECT  BKIC_PROD_CODE, BKIC_PROD_DESC, BKIC_PROD_TYPE
FROM    BKICMSTR
ORDER BY BKIC_PROD_TYPE

```

Because one type can be used by many parts, once we have sorted by type we may also want the result sorted by part number within each type. We can do this by including a second field in the ORDER BY clause:

```

SELECT  BKIC_PROD_CODE, BKIC_PROD_DESC, BKIC_PROD_TYPE
FROM    BKICMSTR
ORDER BY BKIC_PROD_TYPE, BKIC_PROD_CODE

```

You can add more fields to the clause to get more sub-sorting of the result.

2.4 Aliases

When you are selecting a field, you can also tell the engine that you would prefer the column label to be something other than the name of the column in the database. For example, instead of BKIC_PROD_CODE if you want it to say “Part” you can change the SELECT clause by adding the term `as` and the new name you want:

```

SELECT BKIC_PROD_CODE as Part, BKIC_PROD_DESC as Description
FROM   BKICMSTR
WHERE  BKIC_PROD_TYPE = 'O'

```

Because SQL uses “whitespace” (spaces, tabs, and new-lines) as delimiters between portions of a query, if the alias you want to use has a space in it you must put the whole alias in quotation marks:

```

SELECT BKIC_PROD_CODE as "Part Number",
       BKIC_PROD_DESC as "Part Description"
FROM   BKICMSTR
WHERE  BKIC_PROD_TYPE = 'O'

```

3 Operations and Functions

In addition to getting the value of a field, we can perform operations on fields either in the SELECT or WHERE clause. If our fields are numbers we can perform normal arithmetic operations using: `+`, `-`, `*`, or `/`.

Additionally, the `+` operator works for string fields by *concatenating* the strings together. In other words, if you have two strings `'abc'` and `'def'`, adding them together produces the string `'abcdef'`.

There are also many functions in SQL that can be used on fields we select or filter on. For example, the `UPPER` function takes a string and returns the contents all in upper-case. To use a function, you provide the name of the function and the thing(s) to apply it on in parentheses.

For example, the following query selects all part numbers and forces them to be upper-case:

```

SELECT UPPER(BKIC_PROD_CODE) as UPPER_PART

```

```
FROM BKICMSTR
```

Usually when we use a function it's best to provide an alias for the result because otherwise the engine automatically generates new names for each different expression in our `SELECT` clause.

3.1 Aggregate Functions

There are also *aggregate functions* that let us perform actions such as `SUM`, which would add up all of the values of a given field or `COUNT`, which tells you how many rows would be in the result. There are also functions like `MAX` or `MIN` that get the largest or smallest value from the result, respectively.

These are different from other functions because they don't just modify a single value but instead combine many values together to make one answer, hence the name "aggregate" function.

As a simple example, this query will tell you how many parts you have:

```
SELECT COUNT(*) as PART_COUNT
FROM BKICMSTR
```

Adding one more piece, the following will tell you how many parts have type "O":

```
SELECT COUNT(*) as PART_COUNT
FROM BKICMSTR
WHERE BKIC_PROD_TYPE = 'O'
```

What if you wanted to know how many of *each* type you had? In this case, we don't want to filter by the type, we just want to list each type along with the count for it. To do this, we have to introduce a new clause: the `GROUP BY` clause. Whenever you use an aggregate function and also want to select other fields that are not part of the aggregate function result, you must "group by" those other fields. Here is the query to show all part types with the total count for each:

```
SELECT BKIC_PROD_TYPE, COUNT(*) as PART_COUNT
FROM BKICMSTR
GROUP BY BKIC_PROD_TYPE
```

This query says to count the number of entries by grouping them by type first then counting how many are in each group.

4 Joins

Whenever we perform a query on two or more tables, we must describe to the engine how to properly *join* those tables. To do this, the tables should share a column (or columns) that uniquely identify the items from the other table. For example, each invoice line (table `BKARHIVL`) stores a column for the invoice it is connected to (table `BKARHINV`). In the invoice table, the field storing the invoice number (`BKAR_INV_NUM`) is unique to each invoice. We call such a field a *primary key*. In the invoice line table, each line also has a reference to the invoice number in field

BKAR_INVL_INVNM. In this case, the value is not unique because one invoice can have many lines. However, we can see exactly which invoice any given line is associated with by looking up the invoice with the matching number.

In a query, this is done by including a piece in the WHERE clause that equates the appropriate fields for each table. Therefore, for invoices and invoice lines, we would use:

```
WHERE BKAR_INV_NUM = BKAR_INVL_INVNM
```

to make sure our result matches the fields from each invoice line to the appropriate invoice. If you ever try to perform a query on multiple tables without properly *joining* them, your result will be a giant mess because the engine tries to attach every invoice line to every invoice. In math terms, the tables act as sets and the engine does a *cartesian product* on them when joining. The join portion of the WHERE clause strips away all the extra pieces of the cartesian product that don't match correctly.

4.1 Example Queries

The following example queries the customer information table and the invoice tables to get invoice details for customers in territory "USW" for invoices in 2015. In this case, we must perform a join between the customer and invoice tables by matching the customer code to the invoice. We don't match invoice number here because the customer table has no notion of an invoice number; every customer can have many invoices. Therefore, we store the customer code in the invoice table in order to match the invoices back to the customer. Additionally, we must join the invoice table to the invoice line table by invoice number as shown previously. Finally, although we are not selecting the BKAR_TERRITORY field, because it is part of the BKAR_CUST table in the FROM clause, we are able to use it in the WHERE clause. Remember, the SELECT clause is handled after the FROM and WHERE clauses.

```
SELECT  BKARHINV.BKAR_INV_CUSNME, BKARHINV.BKAR_INV_INVDT,
        BKARHIVL.BKAR_INVL_PCODE, BKARHIVL.BKAR_INVL_PDESC,
        BKARHIVL.BKAR_INVL_PQTY, BKARHIVL.BKAR_INVL_PEXT
FROM    BKAR_CUST, BKARHINV, BKARHIVL
WHERE   BKAR_CUST.BKAR_CUSTCODE = BKARHINV.BKAR_INV_CUSCOD
AND     BKARHIVL.BKAR_INVL_INVNM = BKARHINV.BKAR_INV_NUM
AND     BKAR_CUST.BKAR_TERRITORY = 'USW'
AND     BKARHINV.BKAR_INV_INVDT >= '2015-01-01'
AND     BKARHINV.BKAR_INV_INVDT <= '2015-12-31'
ORDER BY BKARHINV.BKAR_INV_CUSNME,
         BKARHINV.BKAR_INV_INVDT ASC
```

This next example retrieves the work order production details for work orders assigned to "USW" territory customers. We must match the work order table to the customer table by customer code (every work order has an associated customer). We must also match the work order table to the work order production receipts table. However, a work order is identified by two fields: the prefix (MTWO_WIP_WOPRE) and the suffix (MTWO_WIP_WOSUF). Therefore, when performing the join we must match both of these fields.

```
SELECT  WORKORD.MTWO_CUSTNAME, WORKORD.MTWO_WIP_CODE,
```

```

        WORKORD.MTWO_WIP_SFIN, WORKORD.MTWO_WIP_SQTY,
        WORECV.MTWOR_DATE, WORECV.MTWOR_QTY
FROM    WORKORD, WORECV, BKARCUST
WHERE   WORKORD.MTWO_WIP_WOPRE = WORECV.MTWOR_WOPRE
AND     WORKORD.MTWO_WIP_WOSUF = WORECV.MTWOR_WOSUF
AND     WORKORD.MTWO_CUSTCODE = BKARCUST.BKAR_CUSTCODE
AND     BKARCUST.BKAR_TERRITORY = 'USW'
ORDER BY WORKORD.MTWO_CUSTNAME ASC, WORKORD.MTWO_WIP_CODE ASC

```

4.2 Inner Joins and Outer Joins

The type of join discussed so far is an *inner join* which means that if one table does not have a matching entry in the other table, we get no result for that entry. There is an alternative called an *outer join* where it will give you “null” or blank values for anything that does not have a match in the other table. To do this, we can perform any of the following:

1. A *left outer join* makes null entries whenever something in the *left* table has no match in the *right* table.
2. A *right outer join* makes null entries whenever something in the *right* table has no match in the *left* table.
3. A *full outer join* makes null entries whenever something in either table has no match in the other table.

To do an outer join, we have to use some special syntax in the FROM clause. For example, to perform a left outer join on the invoice and invoice line tables we would use the following:

```

SELECT BKAR_INV_CUSNME, BKAR_INVL_PCODE,
       BKAR_INVL_PDESC, BKAR_INVL_PQTY,
       BKAR_INVL_PEXT, BKAR_INVL_PCOGS
FROM   BKARHINV LEFT OUTER JOIN BKARHIVL ON
       BKAR_INV_NUM = BKAR_INVL_INVNM
WHERE  BKAR_INV_INVNTE >= '2015-01-01'
AND    BKAR_INV_INVNTE <= '2015-12-31'

```

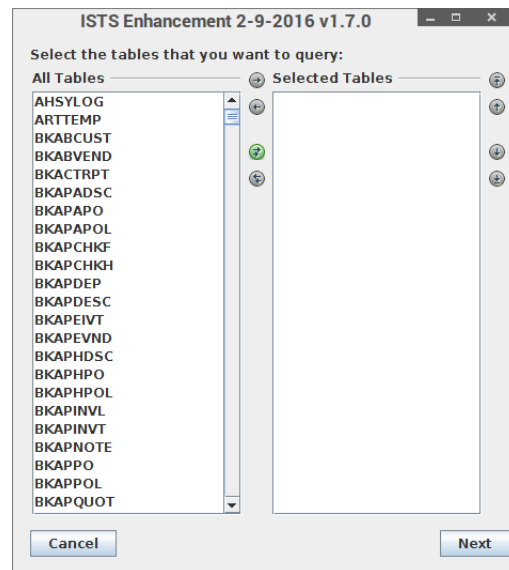
We will rarely use outer joins, but they can be useful in some scenarios such as troubleshooting when you think there should be a matching entry and want to find the cases where there isn't one. In most cases, use inner joins as described in the previous section.

5 The Query Wizard

The query wizard has been created to guide you through the process of constructing an SQL query. It can be useful if you are not familiar with SQL syntax or if you can't remember all of the different field and table names.

5.1 Selecting Tables

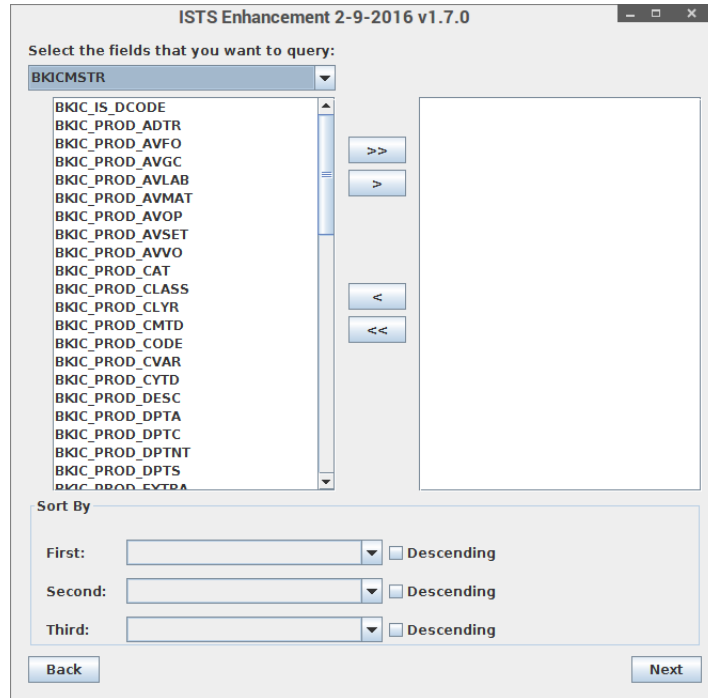
The first step in the query wizard is effectively the FROM clause of the query: selecting which tables to look at.



On this screen you are presented with two lists: on the left is the list of all tables in the database and on the right is the list of tables you have selected. You can select as many as you desire, but a query will usually have three or fewer tables. You can also start typing a table name to have it search in the list for tables that match that name.

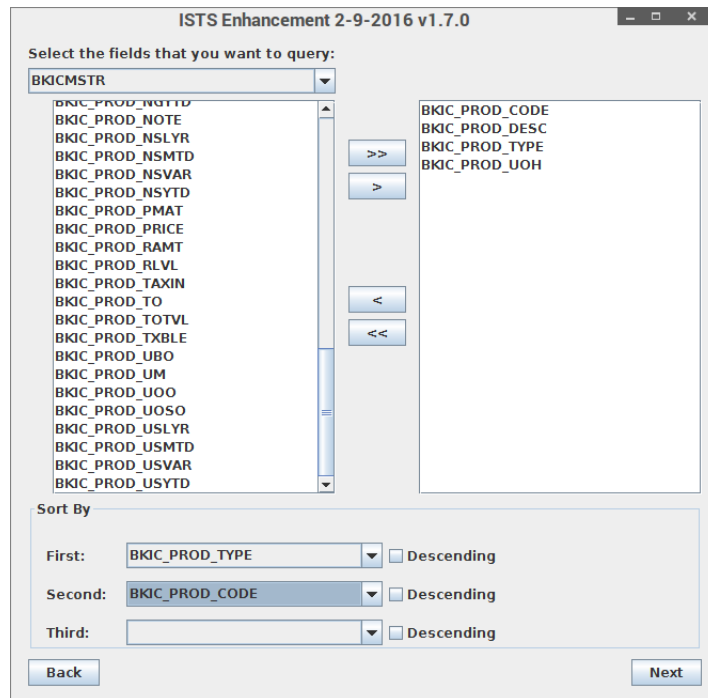
5.2 Selecting Fields

Once you have selected the tables to query, you move on to selecting the fields from those tables (the SELECT clause).

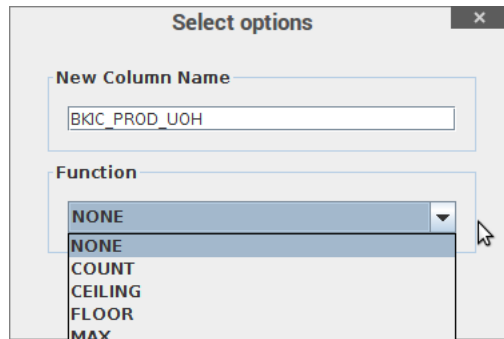


For field selection, you are presented at the top with the list of tables chosen in the previous step. When you choose a table its fields will populate the list below. You can select as many fields as you want.

Once you have chosen some fields, you can set the sort order below (the ORDER BY clause). The query wizard allows sorting by up to three fields. If you need to add more you can do so after the query is constructed at the end of the wizard. By default sorting is in ascending order but you can click the check box to choose descending order.

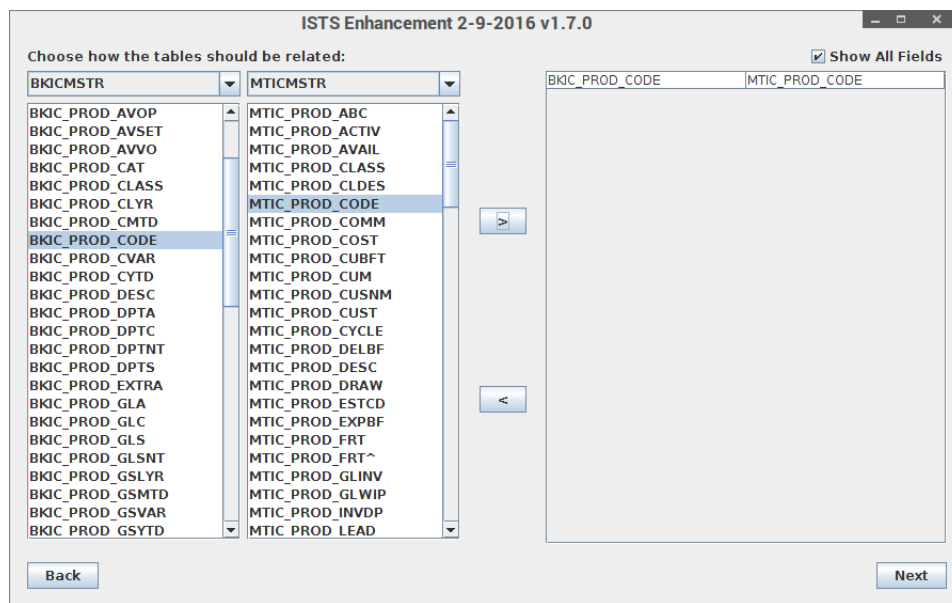


You can also double-click a selected field in the right-hand list to set an alias and/or a function to apply to the field. Depending on the type of data stored in the field the functions listed will be different. If you choose an aggregate function here the query wizard will automatically perform GROUP BY on all of the other fields.



5.3 Joining Tables

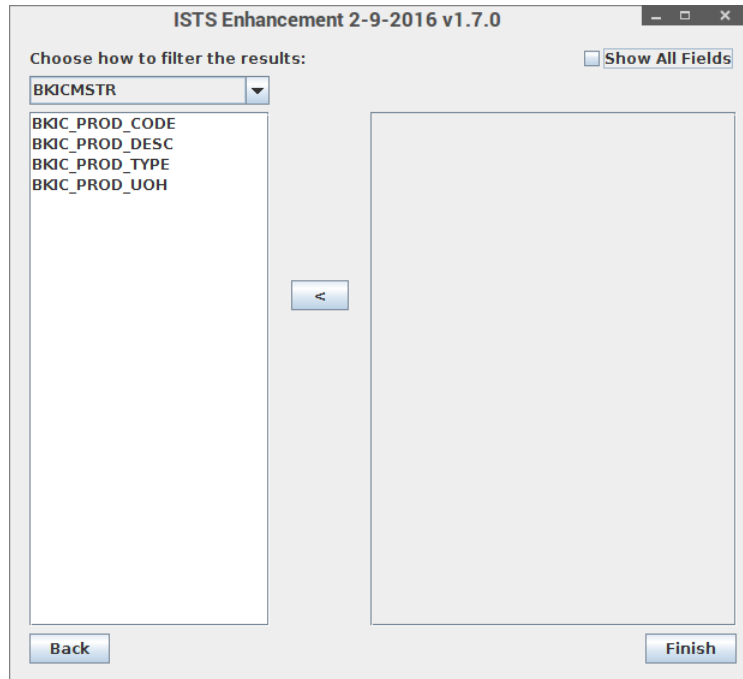
If you have selected more than one table, the next step is to choose the fields on which to join them. The query wizard only supports inner joins. By default, only fields that you have selected are shown, but you can join on any fields by selecting the checkbox in the upper-right corner.



To describe a join, you should choose one field from each list and then press the right arrow button to add those fields as a join. For example, BKICMSTR and MTICMSTR are related by the BKIC_PROD_CODE and MTIC_PROD_CODE fields respectively, so I choose those two fields as shown above. I then add that join to the right list by pressing the right arrow button. Now, the query will join those tables using those fields.

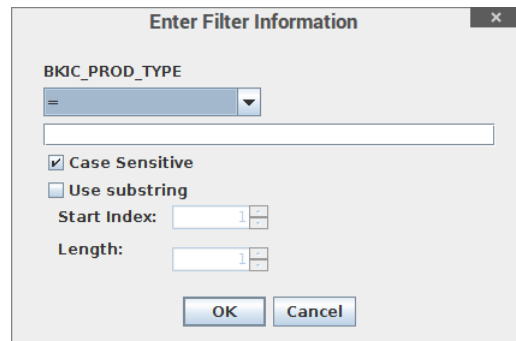
5.4 Adding Filters

Once joins have been established (or if you are only querying one table), you can specify any filters to be used in the query.



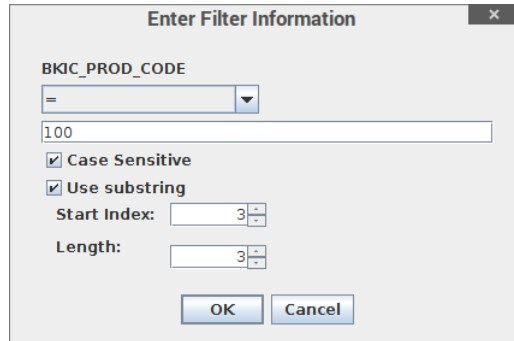
To specify a filter, first choose the table in the top-left box then the field to filter in the list below. As in the joins screen, by default only fields you are selecting are shown but you can show all fields by selecting the checkbox in the top-right.

Click any field in the list on the left to create a filter. This will open a new screen that allows you to enter the specifics.



You have different options depending on the type of the field you are filtering. For character string fields such as the type field pictured above, you can choose an operation such as =, <, >, <>, etc. and the value to compare with. You can also choose whether or not the comparison should be case-sensitive and whether or not to use a substring of the field's value instead of the entire value. A substring is a portion of a string starting from a certain position (start index) of a certain length.

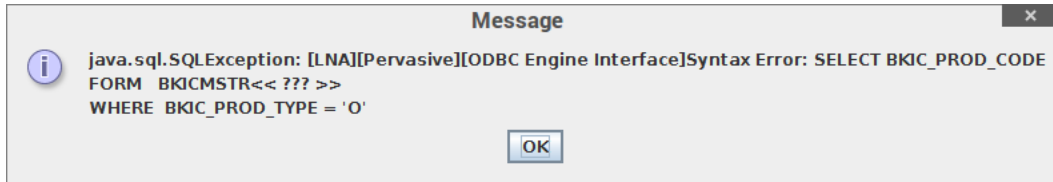
For example, suppose your part numbers follow the format A-NNN where A is a letter and NNN is a three-digit number. If you want all part numbers where the number portion is 100 but with any prefix, you can choose to compare the substring starting at position 3 of length 3 and compare for equality with the string 100 as shown below:



After specifying all of the desired filters, you can click finish and the query wizard will generate the appropriate SQL query then fill it in to the text area in the main screen.

6 Query Execution

Once you have a query constructed, the next step is to execute it. This sends it over the network to the Pervasive server to be processed. Once the response is back, you will either see an error message or the result screen. Below is an example of an error message due to invalid SQL syntax. In this case, I've accidentally entered the word FORM instead of FROM in the FROM clause.



When the engine can't figure out what your query says, it will send back a response trying to tell you where it got stuck. Unfortunately, this is not always easy to troubleshoot. For example, even in this simple case it is complaining about getting stuck *after* the FROM clause instead of at the word FORM. It shows where it got stuck by inserting << ??? >> in to the query text in the error message.

Assuming we reach the result screen, the result of a query is displayed as a table with all of the fields we selected. From here, you might just look through the results or decide to export them.

ISTS Enhancement 2-9-2016 v1.7.0

Export to CSV Format for Excel Create Chart

BKIC_PROD_CODE	BKIC_PROD_DESC	BKIC_PROD_TYPE	BKIC_PROD_UM	BKIC_PROD_CAT	BKIC_PROD_TXBLE	BKIC_PROD_CLASS	BKIC_PROD_RLVL	BKIC_PRC
*	NON-STOCK ITE...	N	EA	W	Y	WAG	0	
1000	DESK OFFICE ...	F	EA	F	Y	FURN	0	
1005	DESK FRAME AS...	B	EA	F	Y	FURN	10	
MAKE	Make From Par...	M	EA		Y	WAG	0	
THOUSAND	Buy by Thousa...	R	EA		Y	WAG	0	
DOZEN	Buy by Dozen ...	R	EA		Y	WAG	240	
KIT	Selling Kit ...	K	EA		Y	WAG	0	
COMPONENT	Component of ...	R	EA		Y	WAG	0	
PHANTOM	Phantom Suba...	B	EA		Y	WAG	0	
B-100	Wagon Body ...	A	EA		Y	WAG	0	
W-200	RED WAGON US...	F	EA	W	Y	WAG	0	
B-200	WHEEL ASSEMB...	B	EA	W	Y	WAG	400	
B-300	FORK ASSEMBL...	B	EA	W	Y	WAG	500	
FINITE	Finite Schedul...	F	EA		Y	WAG	0	
MM	Buy & Stock by ...	R	EA		Y	WAG	0	
1005-1	LEFT BRACE AS...	B	EA	F	Y	FURN	0	
1005-2	RIGHT BRACE A...	B	EA	F	Y	FURN	0	
1005-3	CENTER BRACE ...	B	EA	F	Y	FURN	0	
1006	DESK LEG ...	A	EA	F	Y	FURN	0	
1006-1	CASTER ...	R	EA	FDSC	Y	RP	0	
1100	DESKTOP ...	O	EA	F	Y	FURN	0	
1105	DESKTOP, 4 X 5...	A	EA	F	Y	FURN	0	
1110	DESKTOP, 5 X 6...	A	EA	F	Y	FURN	0	
1115	DESKTOP, 6 X 8...	A	EA	F	Y	FURN	0	
1150	DESKTOP EDGE...	O	EA	F	Y	FURN	0	
1155	ROUNDED EDGE...	M	EA	F	Y	FURN	0	
1166	SQUARE EDGE ...	M	EA	F	Y	FURN	0	
1200	COLOR ...	O	EA	F	Y	PAIN	0	
1205	GREY COLOR ...	R	GL	F	Y	PAIN	0	
1210	BLUE COLOR ...	R	GL	F	Y	PAIN	0	
1215	WHITE COLOR ...	R	GL	F	Y	PAIN	0	
1220	BLACK COLOR ...	R	GL	F	Y	PAIN	0	
1225	BROWN COLOR ...	R	GL	F	Y	PAIN	0	
1300	MODESTY PANE...	O	EA	F	Y	FURN	0	
1305	MODESTY PANE...	A	EA	F	Y	FURN	0	
1310	MODESTY PANE...	A	EA	F	Y	FURN	0	
1400	RETURN ...	O	EA	F	Y	FURN	0	
1405	RETURN, LEFT ...	A	EA	F	Y	FURN	0	
1410	RETURN, RIGHT ...	A	EA	F	Y	FURN	0	

The exporting here works as it does from the first screen: it will generate a CSV file with the query results and you can optionally include special formatting for Excel so that it properly interprets the types of the fields.

7 Charts

As an alternative to exporting, you can also directly construct various charts from a query result by pressing the “Create Chart” button from the result screen.

The program supports the following chart types:

Bar Chart If you would like to make a bar chart, you must enter a title and label for the Y-axis. The x-axis can be grouped by any field from the query result and you can display up to three different bar values. A bar chart is most useful when you have made a query that uses an aggregate function such as getting a sum grouped by another field. For the three bar values, you must specify a numeric field.

Pie Chart For a pie chart, you must specify a title, a field to act as the labels to use for each piece of the pie, and a numeric field for the sizes of each piece.

Line Charts There are three line charts: daily, monthly, and yearly. They all work the same way: you choose a title, a date field for the x-axis, and a numeric field for the y-axis. For daily and monthly you can also specify a year-by-year comparison where it makes a separate line for each year where data was found in the query result. The “enforce range limits” checkbox forces the chart to start at 0 and have a max value slightly larger than the highest data value in the result.

ISTS Enhancement 2-9-2016 v1.1

Title
Line Chart

Dates
BKIC_PROD_LSALE

Values
BKIC_PROD_RLVL

Year by Year Comparison
 Enforce Range Limits

Back Create

8 Query Assistance and Custom Queries

We also offer custom query construction and assistance in query construction for a fee. You can contact lynn@istechsupport.com or nick@istechsupport.com for assistance. We can prepare the query for you, help you set it up as a default query, and show you how you can modify specific pieces such as a date or part number range to make it more flexible for future use.