# 1 Query Builder

## *1.1 What is Query Builder*

## *1.2 Window Layout*

### 1.2.1 Query Structure Area

### 1.2.2 All Tables Area

### 1.2.3 Work Area

### 1.2.4 Tab Area

## *1.3 Buttons*

## *1.4 Query Concepts*

### 1.4.1 Tables and Buffers

### 1.4.2 Conditions

### 1.4.3 Expression Editor

#### 1.4.3.1 Field Tab

#### 1.4.3.2 Constant Tab

#### 1.4.3.3 Function Tab

#### 1.4.3.4 Button Panel

### 1.4.4 Keys and Joins

#### 1.4.4.1 Keys

#### 1.4.4.2 Joins

### 1.4.5 Table Parameters

### 1.4.6 Subqueries

### 1.4.7 Unions

### 1.4.8 Useful Information and Resources

# 1 Query Builder

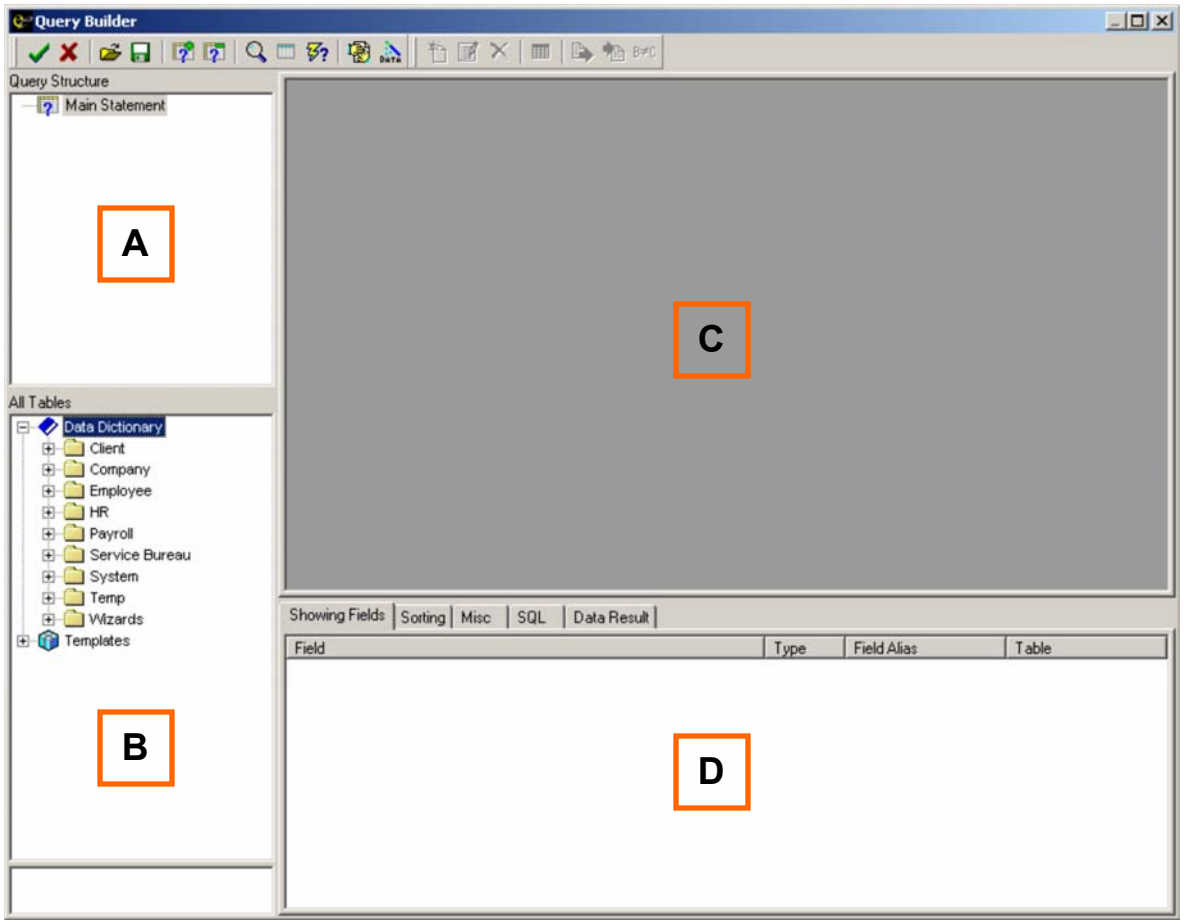
## *1.1 What is Query Builder*

Query Builder is the tool used to retrieve data from Evolution databases. It is a graphical tool that allows the user to drag and drop tables, buffers and fields to create SQL queries. Those queries can include inner, left and right outer joins and unions. Data returned by the query can be sorted and formatted inside Query Builder. New columns can be created that are calculated based on data returned by the query. This document is an introduction to Query Builder and its functionality.

## *1.2 Window Layout*

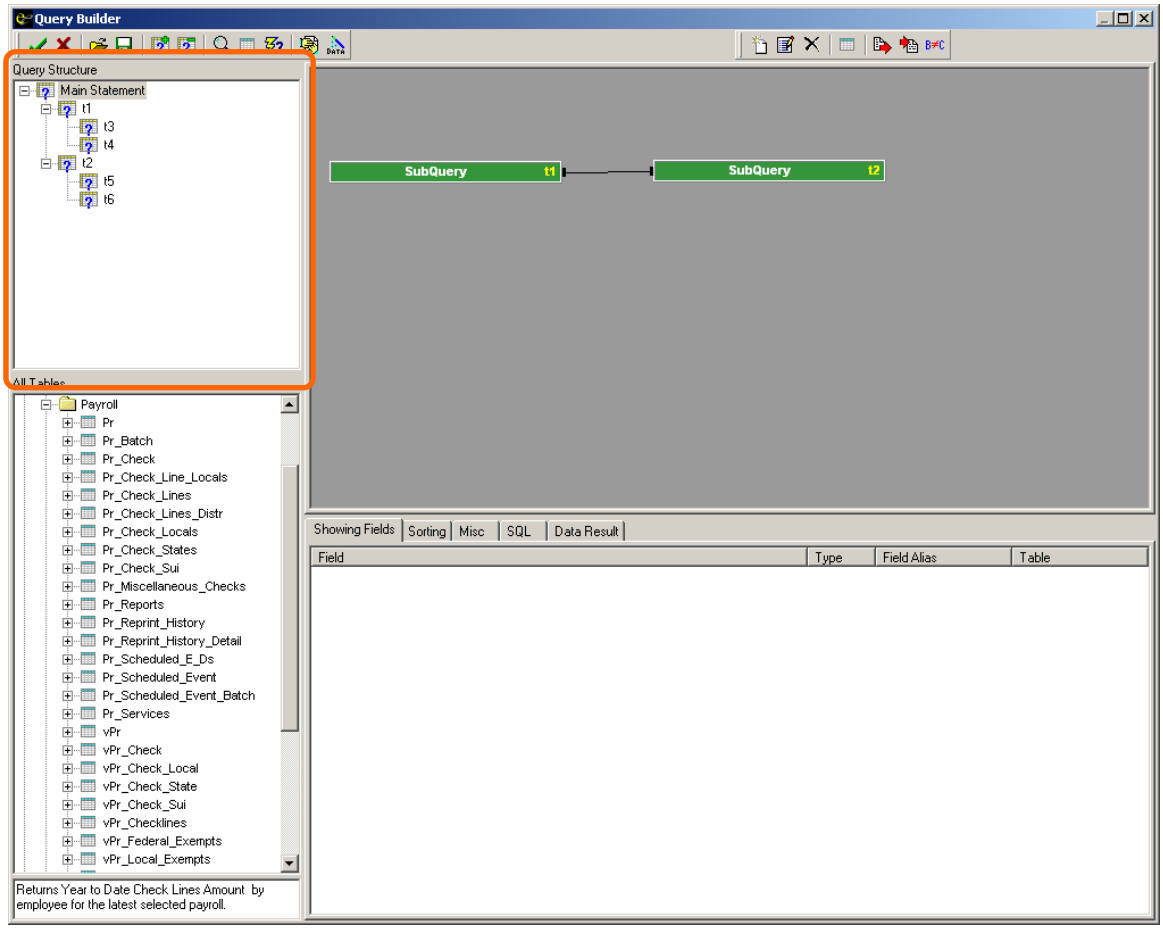
The Query Builder window has four main areas:

- **Query Structure Area (A)**
- **All Tables Area (B)**
- **Work Area (C)**
- **Tab Area (D)**



## 1.2.1 Query Structure Area

The ***Query Structure Area*** shows the layout of the query. There may be multiple levels of a report's query depending on the data that needs to be reported on. If there are multiple levels of the query (referred to as subqueries, to be explained in section 1.4.5), the Query Structure Area will show a tree structure of that query, as shown below:

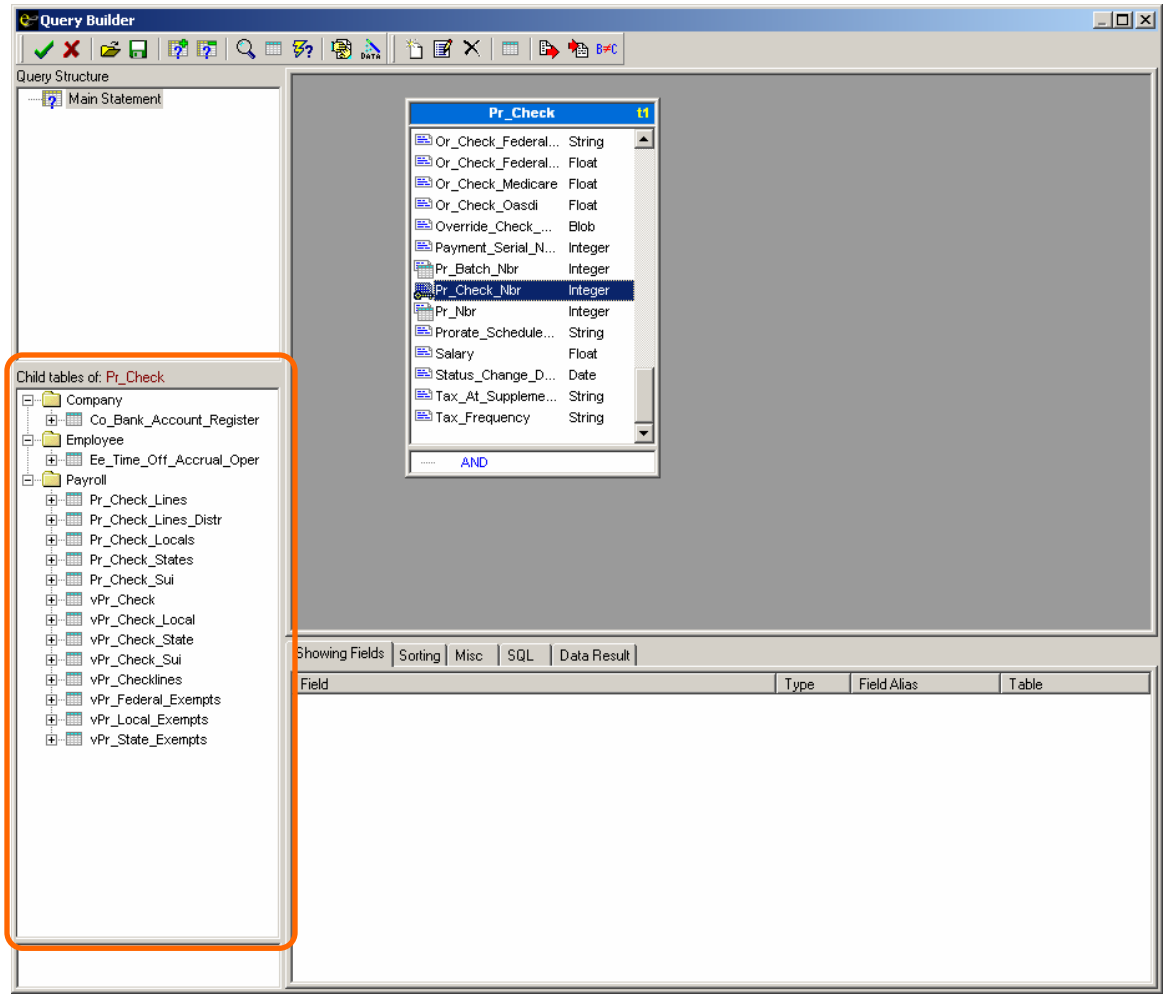



In cases where there are multiple levels in a query, the Work Area and Tab Area will show what is inside the selected subquery. In the example above, the top level of the query is Main Statement. Below that, are two subqueries, each having it's own pair of subqueries. With the **Main Statement** selected, based on what is shown in the Query Structure Area, the Work Area should show two subqueries, **t1** and **t2**, as it does. If the **t1** subquery in this example were selected, the Work Area would show its two subqueries, **t3** and **t4**.

## 1.2.2 All Tables Area

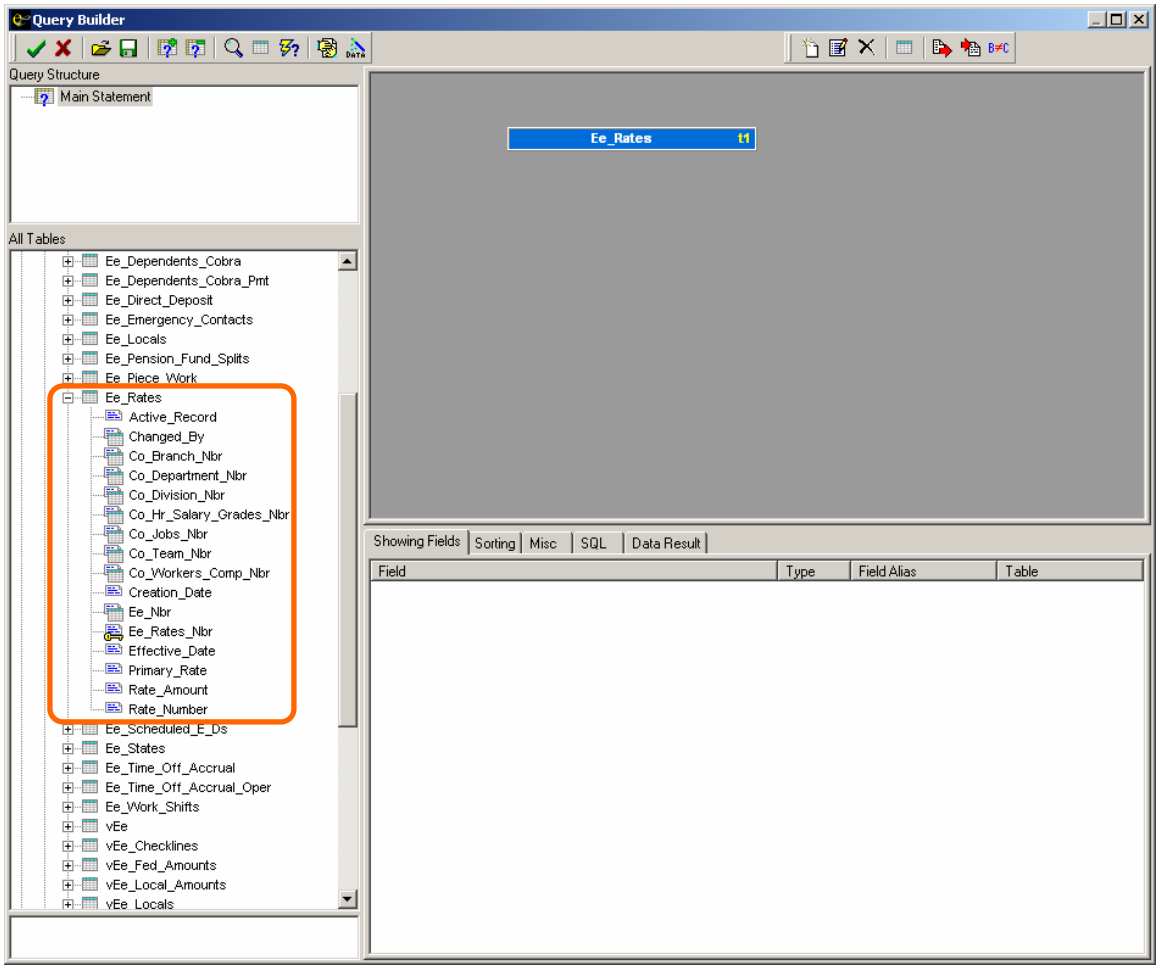
The ***All Tables Area*** shows all tables and buffers and their corresponding fields that are available for use in the query. The tables are split into different folders based on the type of data stored in each table. Tables that contain payroll data may be found in the Payroll folder. Tables that contain employee information are located in the Employee folder.

For a table or buffer to be used in a query, it must be dragged from the All Tables area and dropped into the Work area. If a table is open in the Work area, the All Tables area will become the **Child tables of: <selected table> Area**, as shown below. The Child tables of: <selected table> will show the tables that reference the key field of the selected table, in the case below, **pr_check_nbr**.



# Evolution Query Builder

Each table and buffer in the All Tables or Child tables of: <selected table> Areas can be further expanded to view the columns that exist in that table.




To the left of each column name, there is an icon. That icon helps describe the column. There are three different icons that can appear there, shown below. After each icon is a description of what that icon means to that field.

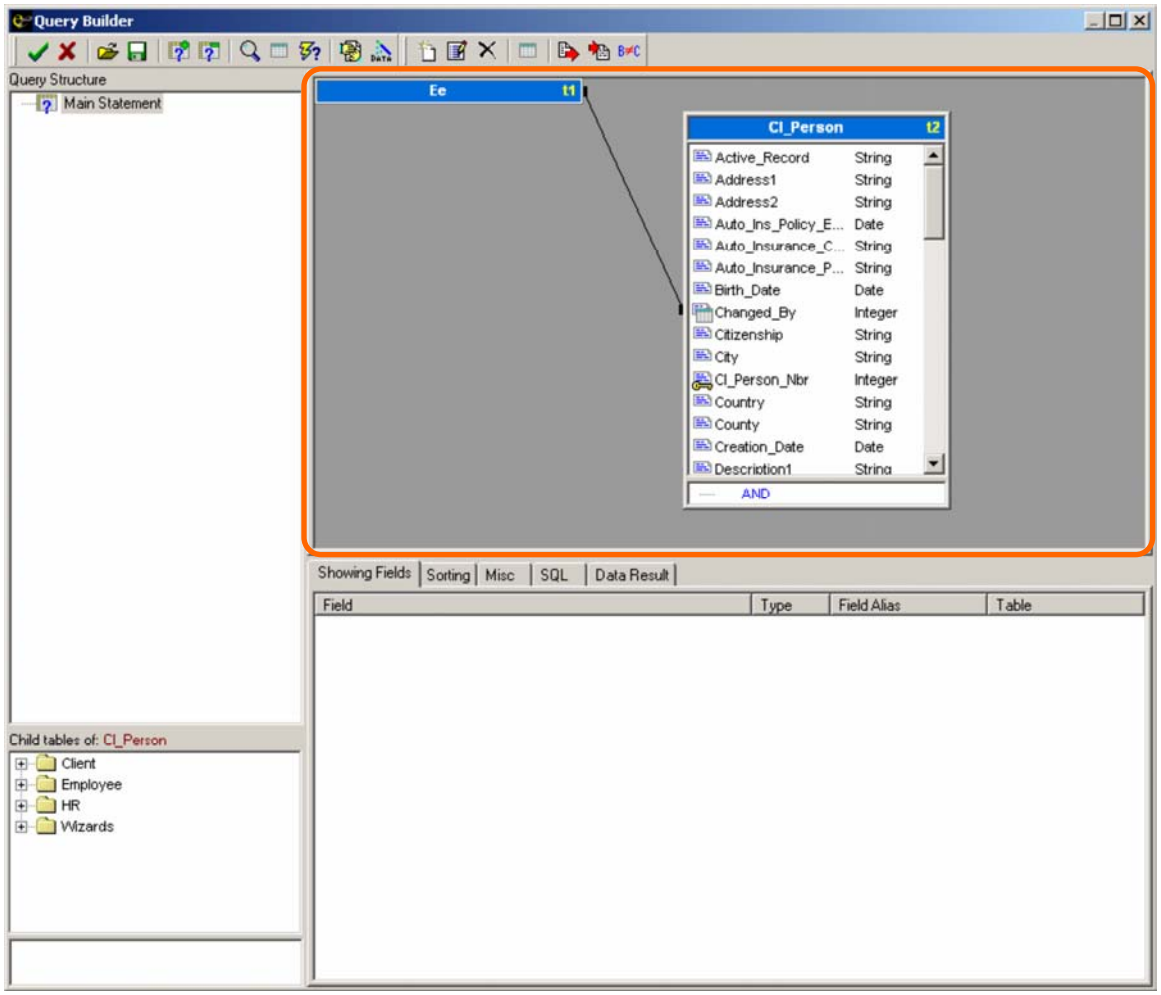
 This is a normal column used to store data relevant to this table.

 This is the key of the table. This column uniquely identifies each current record in this table.

 The data in this column references the key of a different table. This column may be dragged and dropped from the table in the Work Area to another part of the work area to add the table whose key this is, joining the two tables on that column.

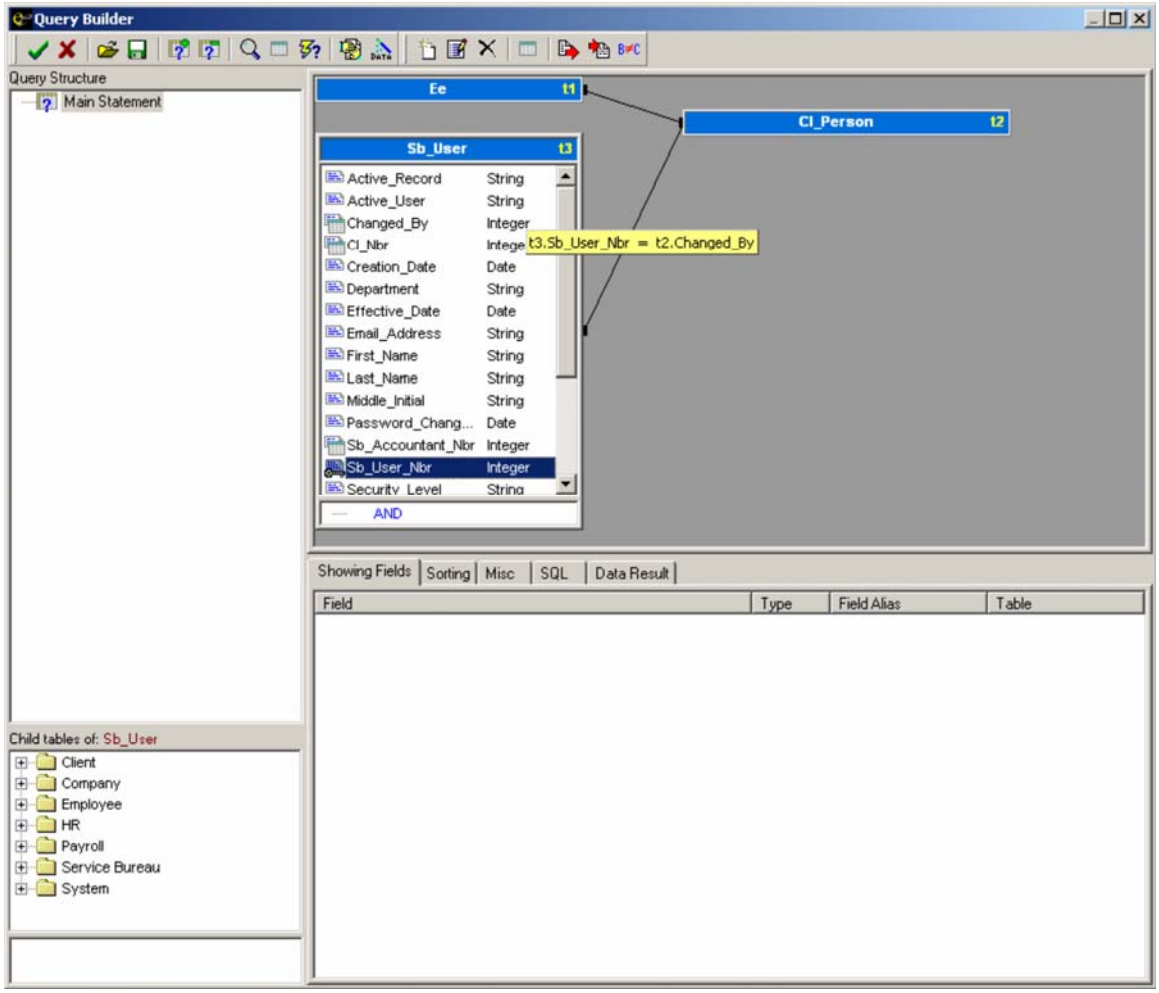
## 1.2.3 Work Area

The ***Work Area*** is where the query is built. Tables and buffers are dragged from the All Tables Area into the Work Area for use in the query. If there is a field that is needed in the query for any reason (joining, sorting, calculating), the table or buffer in which that field is found must be dropped into the Work Area.



# Evolution Query Builder

In the previous screen, the **Changed_By** field has the foreign key icon to the left of it. That means that the field can be dragged from the table in the Work Area into another part of the work area to add the table whose key value is stored in that field of the selected table. The result of this action is shown below.



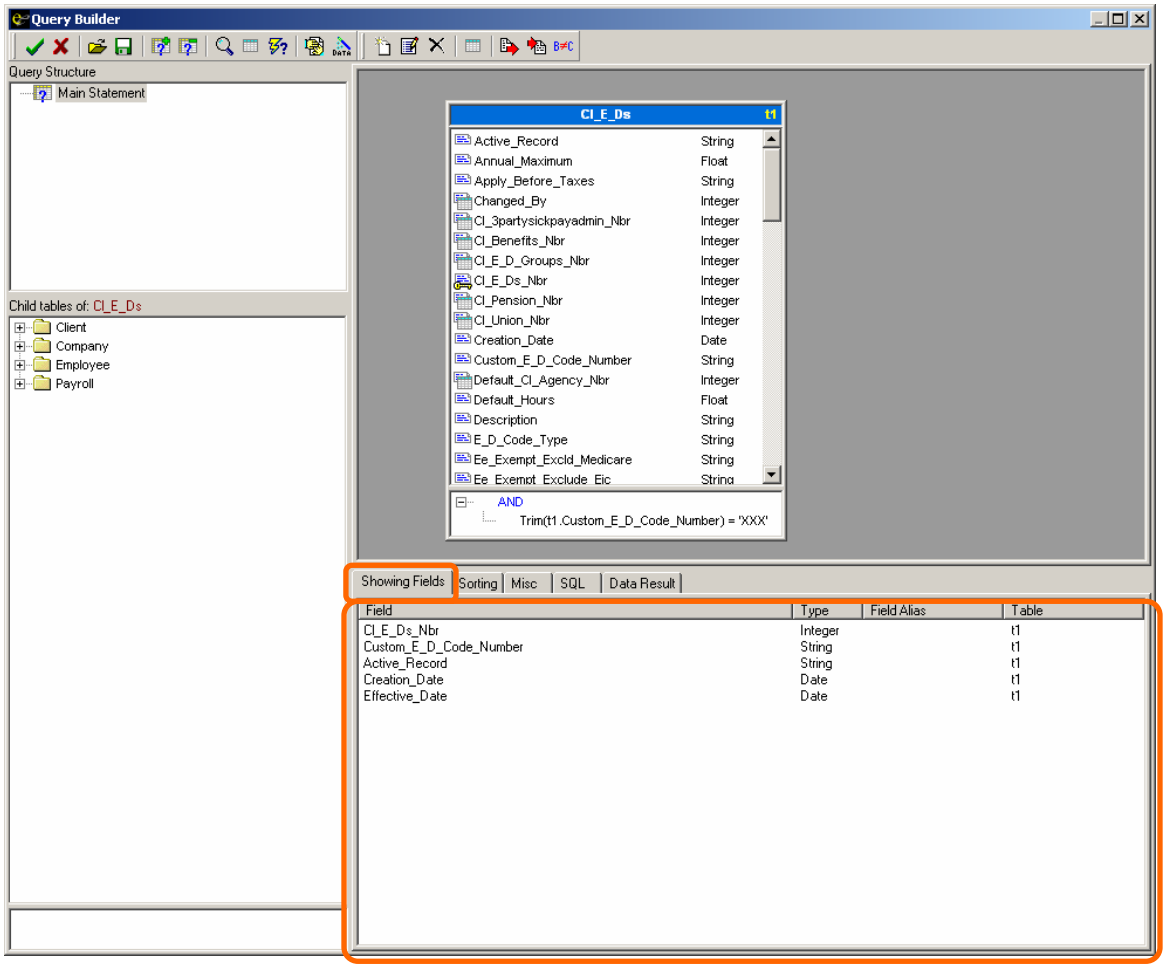

The **Sb_User** table was added to the query by dragging and dropping the **Changed_By** field from the **Cl_Person** table in the Work Area to an empty part of the Work Area. Both tables are now joined on the appropriate field – **Changed_By** in **Cl_Person** and **Sb_User_Nbr** in **Sb_User**. The **Changed_By** field in the **Cl_Person** table stores the internal user number (**Sb_User_Nbr**) of the person who last changed each record in the table. This is true for almost all tables in the Evolution database.

## 1.2.4 Tab Area

The ***Tab Area*** has five tabs:

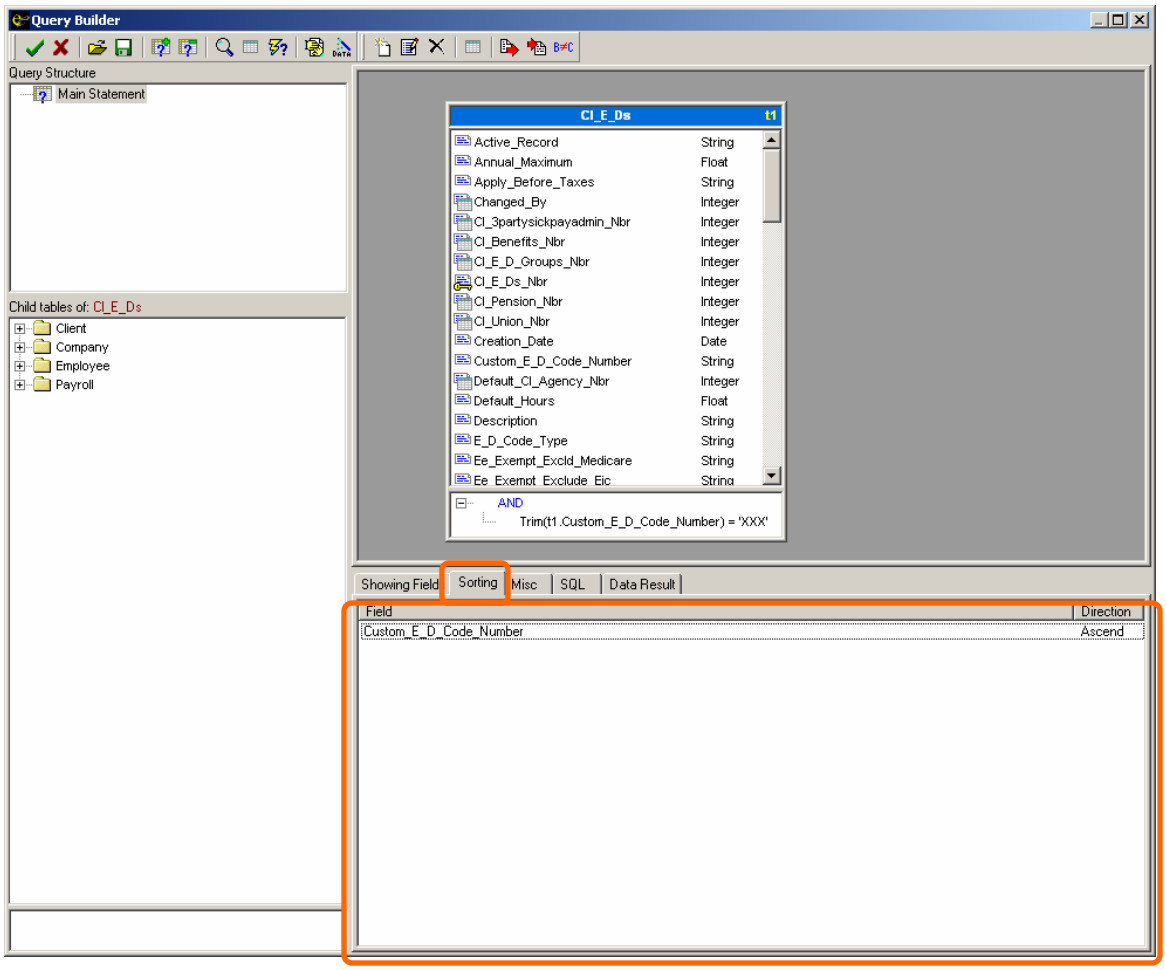
- **Showing Fields**
- **Sorting**
- **Misc**
- **SQL**
- **Data Result**

In subqueries (any query that has a parent query), the ***Showing Fields Tab*** must include any field that needs to be available to the parent query. In the top-level query, the Showing Fields tab must include any field that needs to be available to the Report in which the query exists. If this is a query outside of a report, the Showing Fields tab in the top-level query needs to include all fields to be shown in the result when the query is run from the Misc - Query Builder window, or from the top level of the query inside Query Builder.



# Evolution Query Builder

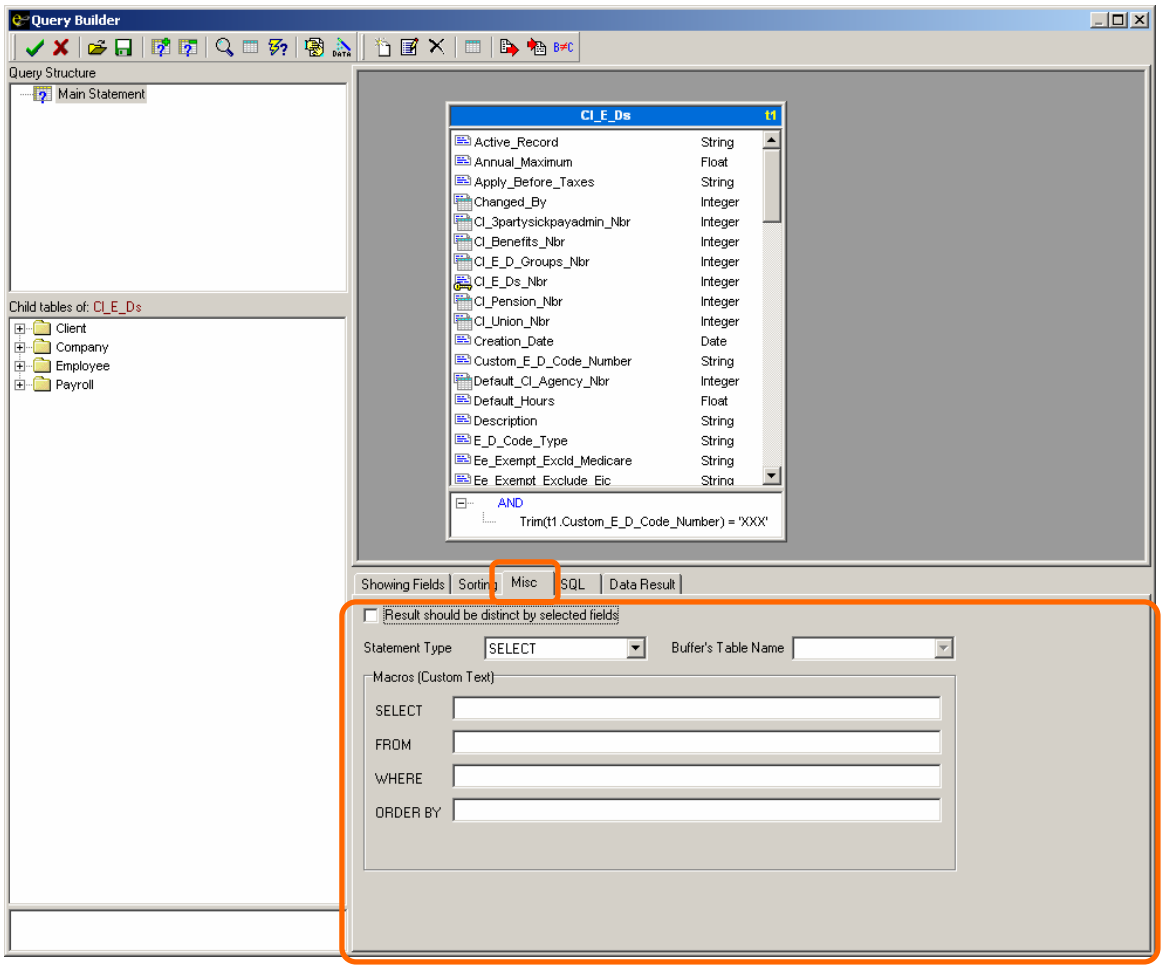
The ***Sorting Tab*** is only available at the top level of the query. Using this tab, the result returned by the query can be sorted on any fields on the Showing Fields tab. The results can be sorted on the selected fields in ascending or descending order.



The ***Misc Tab*** has a check box labeled "Result should be distinct by selected fields". If this check box is checked, the result returned by the query will not include duplicate rows of data. Each row will be unique. For example, a query selects check date and employee from the **Pr**, **Pr_Check** and **Ee** tables. If an employee has multiple checks for a single check date, the query would return different results with the check box unchecked versus checked.

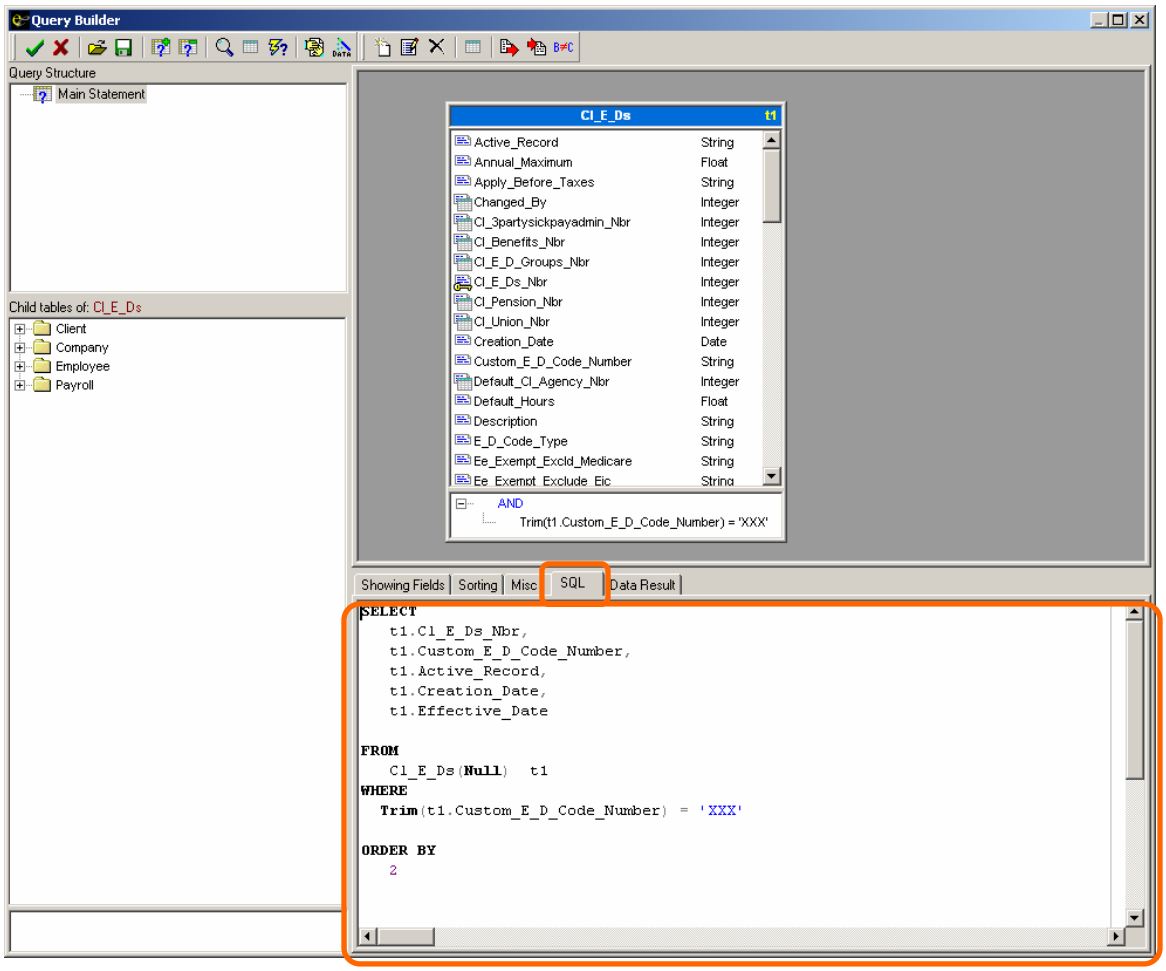
Employee 1 has three checks in payrolls check dated 2/16/2005. With the box unchecked, a row of data would be returned for each check. In this example, the result would be three rows of data for that employee and check date. With the box checked, all duplicates are removed from the result, leaving only unique rows of data. In this example, the result would include a single row of data for employee 1 for the 2/16/2005 check date, regardless of how many checks this employee had in payrolls check dated 2/16/2005.

All other settings on this tab should be ignored.



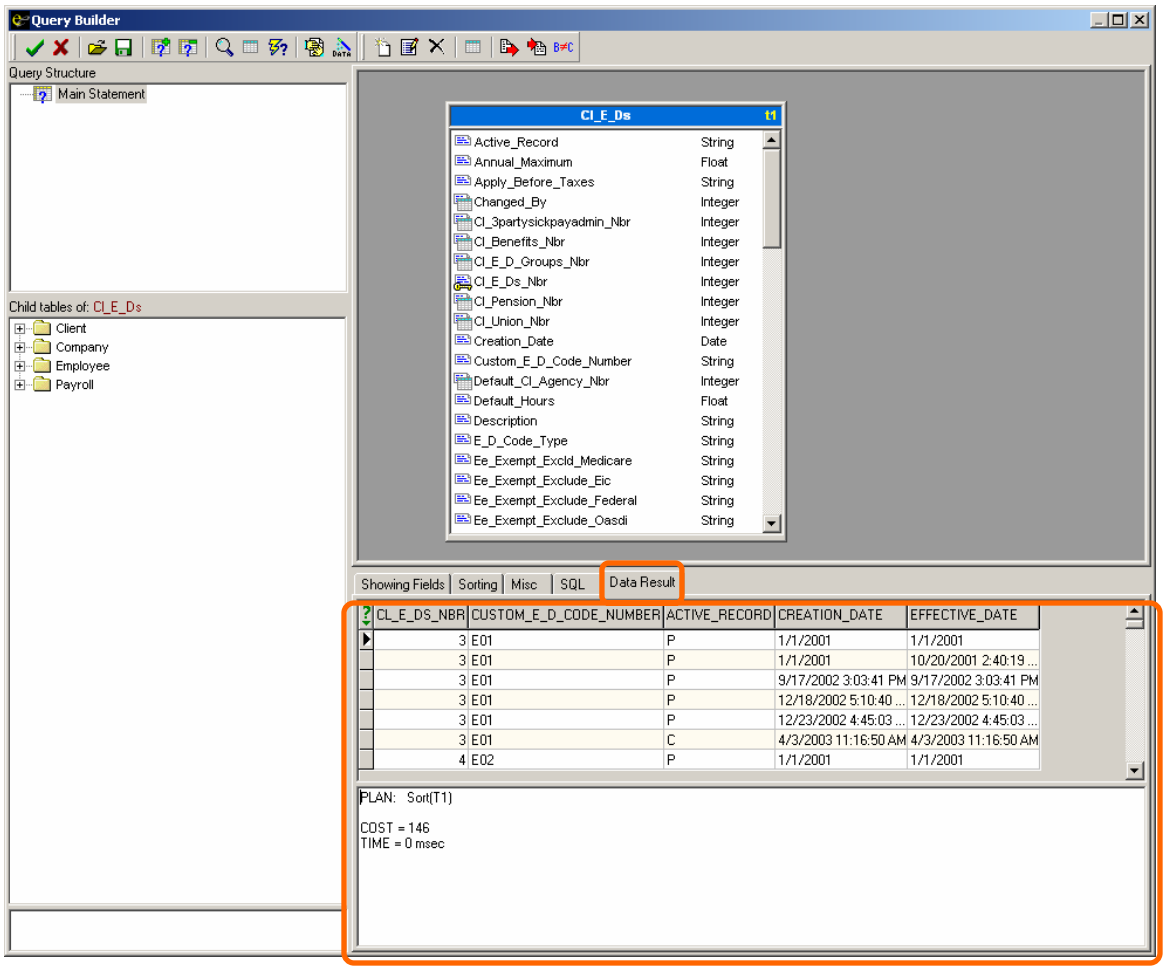
# Evolution Query Builder

The ***SQL Tab*** shows the SQL that is written by Query Builder as the query is being built. This tab is for viewing purposes only. The SQL on this tab cannot be changed.



# Evolution Query Builder

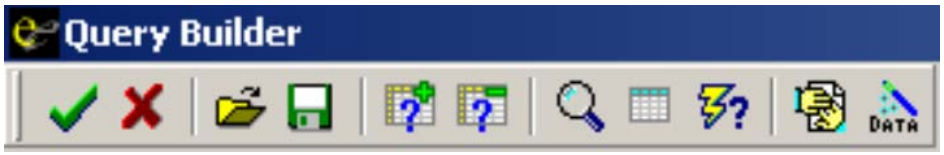
The ***Data Result Tab*** shows the data returned by the level of the query selected in the Query Structure Area. If the top level of the query is selected, it will show the data returned by the entire query. This tab will show a column of data for each field on the Showing Fields tab.



## 1.3 Buttons

Inside the Query Builder window is a panel of button located in the upper left corner by default. From left to right (as shown below), those buttons are:

- **Save & Close**
- **Cancel & Close**
- **Load Query from File**
- **Save Query to File**
- **Add Subquery**
- **Delete Subquery**
- **Search Field/Table (Ctrl+F)**
- **Content of Table**
- **SQL Data Result (F9)**
- **Hide/Show Panels (Ctrl+H)**
- **Wizard View**




***Save & Close*** – Saves any changes you made to the query and closes the Query Builder window.

***Cancel & Close*** – Closes the Query Builder window without saving any changes made to the query since the Query Builder window was opened.

***Load Query from File*** – Opens the Load Query from File dialog box to select a saved query file to open in Query Builder. Query files have the extension RWQ.

***Save Query to File*** – Opens the Save Query to File dialog box to save the current query open in Query builder to a file for later use. If a subquery is selected in the Query Structure Area, this function will save only what is inside of that subquery. To save the entire query, make sure the top level of the query is selected before clicking this button.

***Add Subquery*** – Adds either a subquery, parent query or parent query union to the part of the query currently selected in the Query Structure Area.

***Delete Subquery*** – Deletes the part of the query currently selected in the Query Structure Area.

***Search Field/Table (Ctrl+F)*** – Opens the Find dialog box. A buffer, table or field name may be entered here to be searched for in the All Tables Area by clicking the Find Next button in the Find dialog box.

***Content of Table*** – Shows all current records in the table or buffer selected in the All Tables Area.

***SQL Data Result (F9)*** – Runs the part of the query selected in the Query Structure Area, populating the Data Result tab in the Tabs Area.

***Hide/Show Panels (Ctrl+H)*** – Toggles visibility of the Query Structure, All Tables and Tabs Areas. This button is useful when working with several tables in the same level of a query, as it makes the entire Query Builder window the Work Area.

***Wizard View*** – Toggles between real and user-friendly alias names of database tables and fields in Query Builder. For example, in non-Wizard view, the company table is called **CO**. In Wizard view, it is call **Company**.

## *1.4 Query Concepts*

This section will explain in detail the following basic concepts and objects needed to build a query:

- **Table and buffers**
- **Conditions**
- **Joins**
- **Subqueries**

### 1.4.1 Tables and Buffers

Tables and buffers can be dragged and dropped from the All Tables area to the Work area for use in a query. The ways in which they can be used are explained later.

***Tables*** – These are permanent storage areas for data used in Evolution. Each table has columns and rows. Each column stores a different type of data, like employee name, company state or federal deposit frequency. Each row stores a single instance of each column. All data on that row is related. For example, there is a row in the **Employee** table (non-wizard name **Ee**) for each employee. Each employee has a custom employee code, pay frequency and hire date. There are columns in the Employee table for each of these three types of data – custom employee code, pay frequency and hire date – as well as several other columns directly related to the employee.

Each employee has a single row in the **Employee** table. Therefore, any other type of data that would exist only once for an employee may also be stored in the **Employee** table in the row in which that employee's data resides. Each employee has only one home division, branch, department or team, so these pieces of data are also stored in the **Employee** table in the relevant row. An employee may have multiple rates, states or scheduled E/Ds. Because of that, it doesn't make sense for this data to be stored in the Employee table. Each of these kinds of data has its own table, because each employee rate, state and scheduled E/D can be unique and customized for each employee.

***Buffers*** – These are temporary storage areas used in Query Builder and Report Writer to prepare sets of data for use in a query or report. Buffers have the same basic column/row structure that tables have. However, they may contain pieces of data from multiple tables, calculated columns or constants.

Buffers are defined outside of Query Builder. After the buffer is defined, a query runs. The buffer is then populated with selected data from that query. This process can happen multiple times to populate a single buffer. In the case of multiclient reports, a query is run on the first company, the data required for the report is written to the buffer, the query is run on the next company, that company's data is written to the buffer, and the process repeats until the query is run on the last company and that company's data is written to the buffer.

After the buffer is created, it can be made available for use in Query Builder. Its content may be viewed by selecting it in the All Tables area and clicking on the Content of Table button. It may be used in query builder in the same way that tables are used whether it contains data or not.

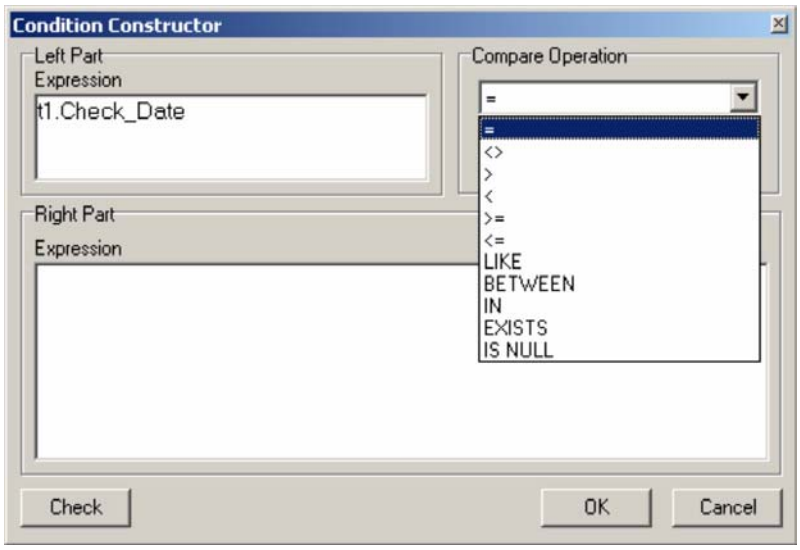
## 1.4.2 Conditions

After a table or buffer has been dropped into the Work area, conditions may be created using columns in that table or buffer. A condition is a statement that will be either true or false for each row of data in the table. Conditions are used to limit the data returned by a query. If the condition is true for a row, that row's data will be included in the query. If the condition is false for a row, that row will be excluded from the query.

Each condition has three pieces - a left part, a compare operation and a right part. They are defined as follows:

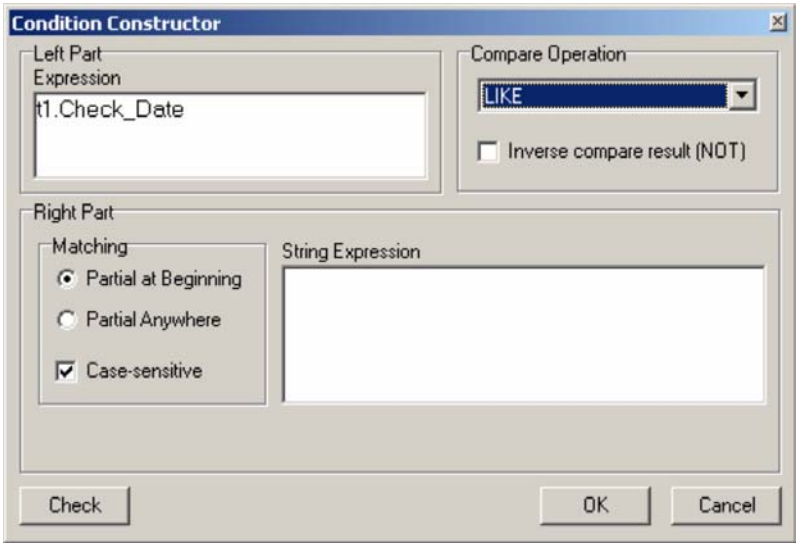
***Left and Right Parts*** – These are the values being compared. They can be calculated, constants or column values straight out of a table or buffer.

***Compare Operation*** – This defines how the left and right parts are being compared. There is a variety of compare operations, shown below:

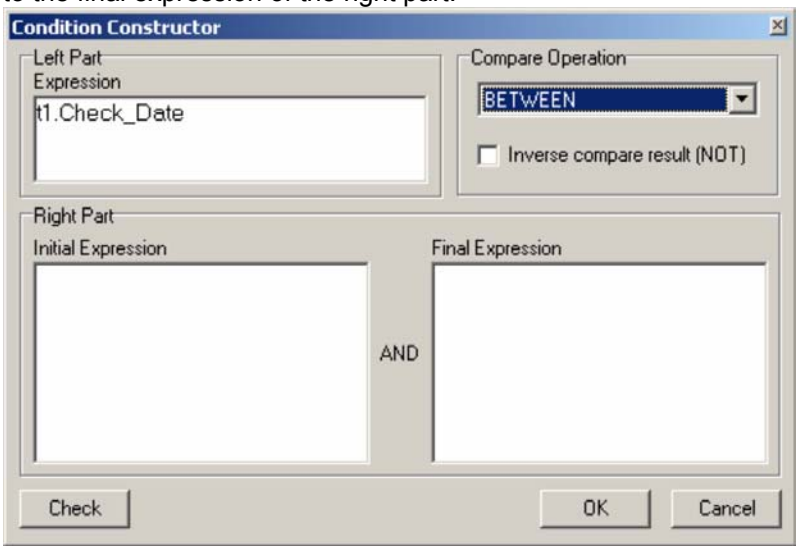



- **=** – Left and right parts must be equal.
- **<>** – Left and right parts must not be equal.
- **>** – Left part is greater than right part.
- **<** – Left part is less than right part.
- **>=** – Left part is greater than or equal to right part.
- **<=** – Left part is less than or equal to right part.

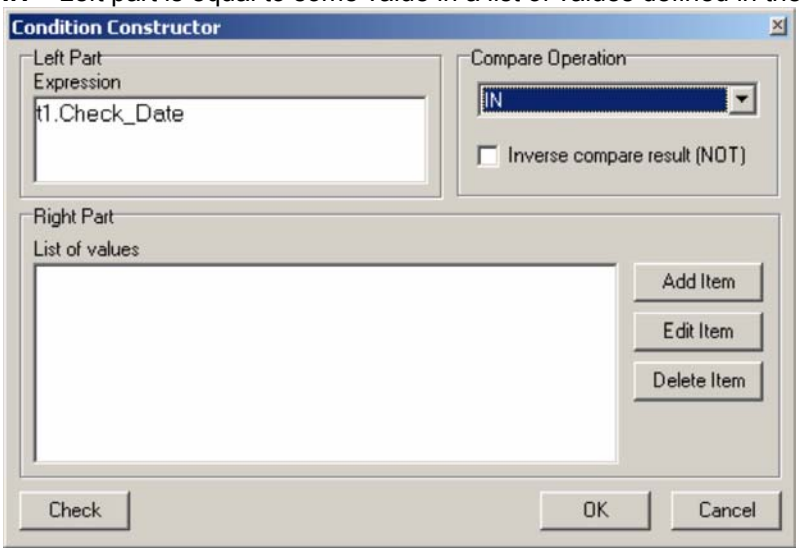
- ***LIKE*** – Left part includes the string expression in the right part. With the Partial at Beginning Matching option selected, the left part must start with what is in the right part. With the Partial Anywhere Matching option, the left part must include what is in the right part anywhere from beginning to end. With the Case-sensitive checkbox checked, the condition will be true only if the case of the left and right parts matches. So, if the left part was the string 'ABC123' and the right part was 'AB', this would be a match because the string 'AB' exists in both parts and is uppercase in both places. If the right part was 'Ab' however, this would not be a match because the left part includes the string 'AB' with both letters being uppercase, but the second letter of the right part is the lowercase letter 'b'. In this example, unchecking the Case-sensitive checkbox would make this a match.



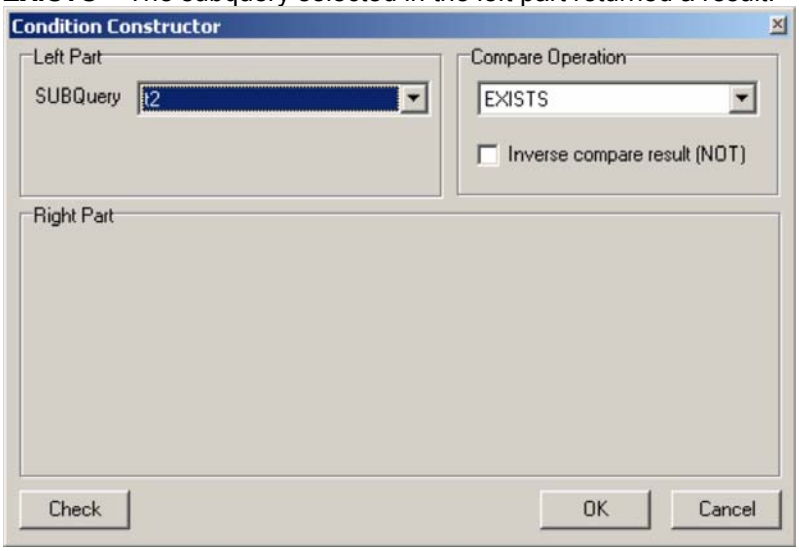

- ***BETWEEN*** – Left part is greater than or equal to the initial expression of the right part, and less than or equal to the final expression of the right part.



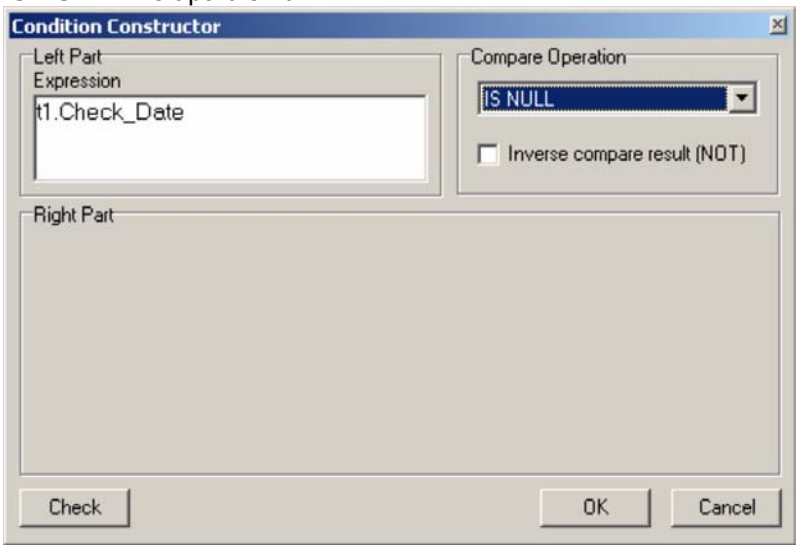
- ***IN*** – Left part is equal to some value in a list of values defined in the right part.




- ***EXISTS –*** The subquery selected in the left part returned a result.



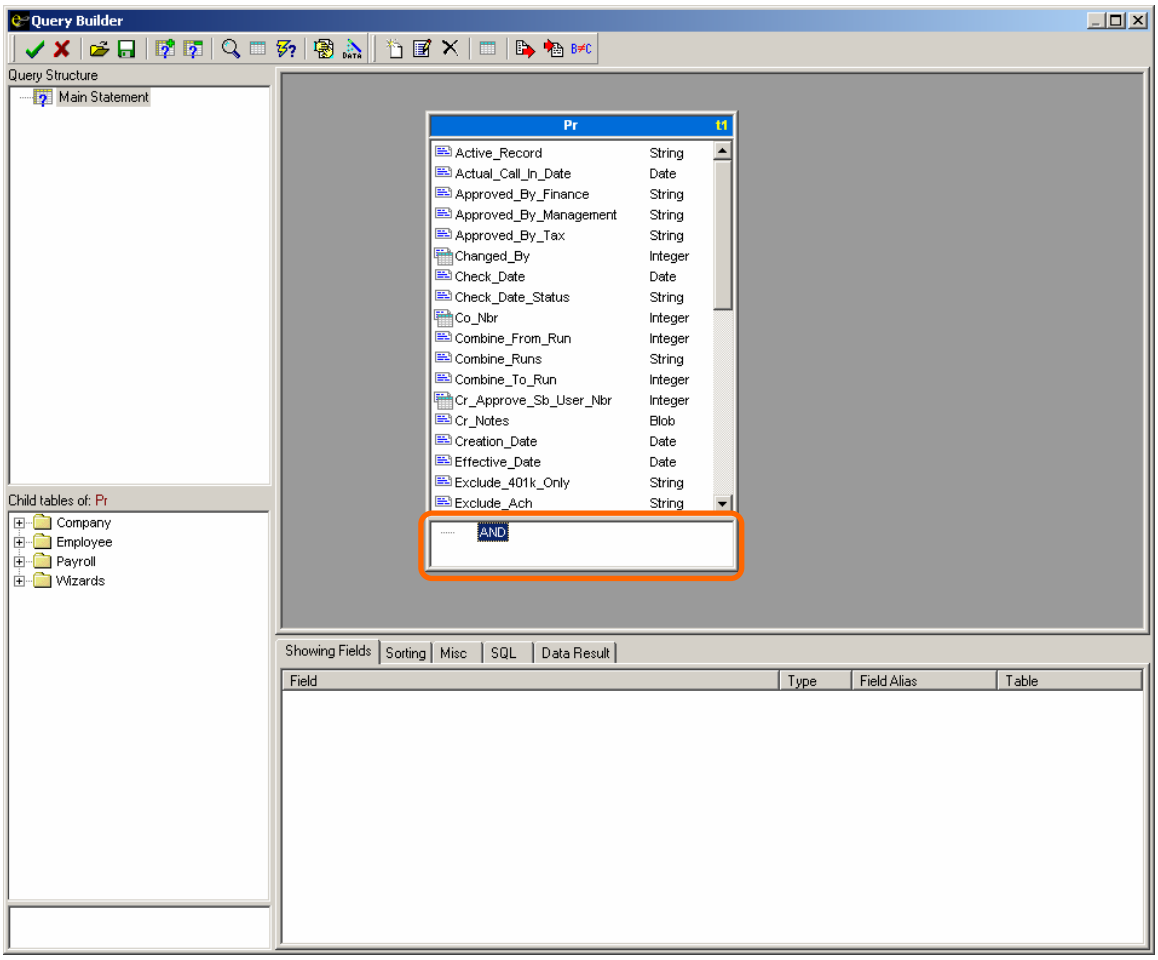

- ***IS NULL*** – Left part is null.



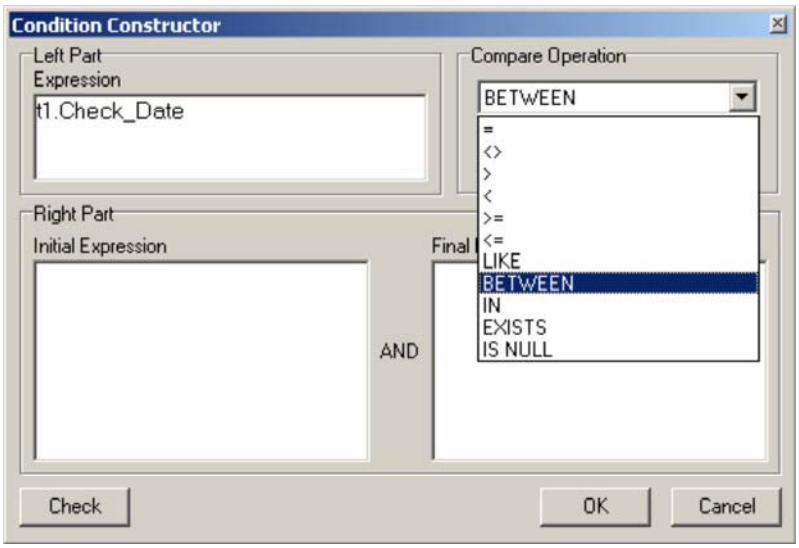
By dropping a specific column into the Condition area of a table, Query Builder will assume that this column is to be used as the left part of the condition. After the compare operation is selected in the dropdown, the next step is to define the right part. This is done by double-clicking in the white box located in the Right Part section of the Condition Constructor window. This will cause the Expression Editor window to appear. This window is used to define a value to be used for comparison in this case. It may also be used in the Showing Fields tab to define a field beyond just showing a value stored in a table or buffer column. The Expression Editor window will be further explained in the next section.

For example, the query is gathering payroll data, but only for a specific check date range. A condition may be created that finds only data related to payrolls with check dates within a specified date range. The steps to create that condition are as follows:

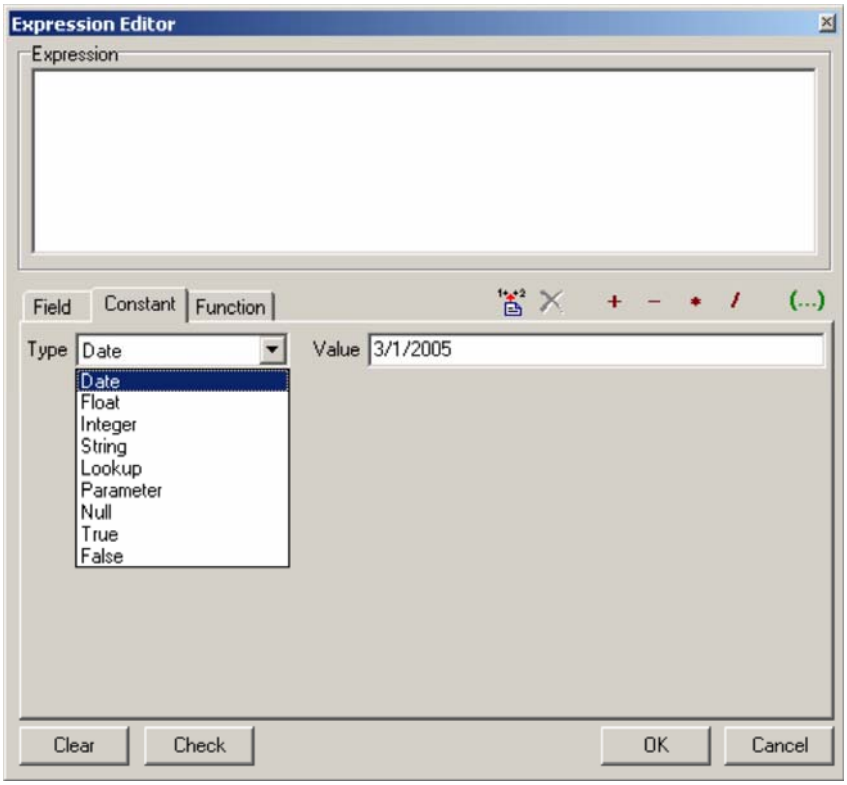
- Drag and drop the column from the table in the Work area down to the bottom of that table's Condition area. This is the area at the bottom of the table in the work space where the word "And" or "Or" is seen. This will only be visible while the table is expanded in the Work area. If the table is expanded and it is still not visible, move the mouse pointer over the bottom border of the table window, then find the two arrows pointing up and down with two horizontal lines between, click and drag up. The Condition area should now be visible.



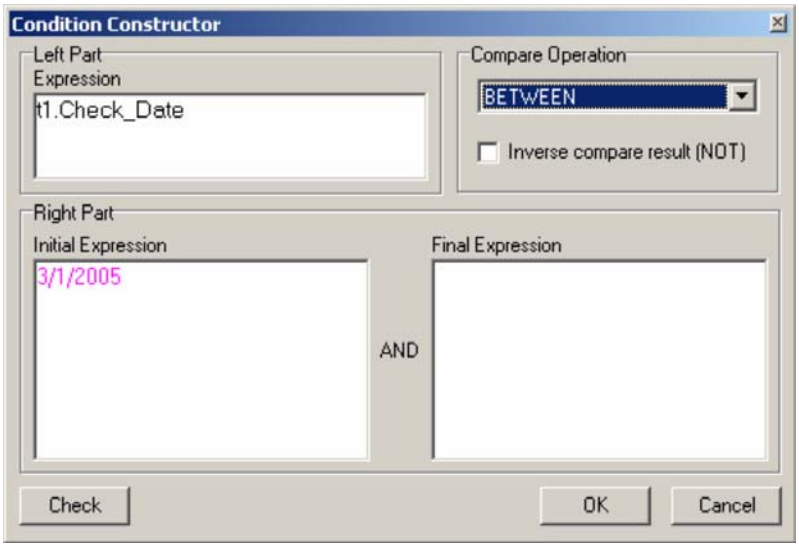
- After the column is dropped into the Condition area, the Condition Constructor window will appear, shown below. The compare operation needs to be changed to BETWEEN.



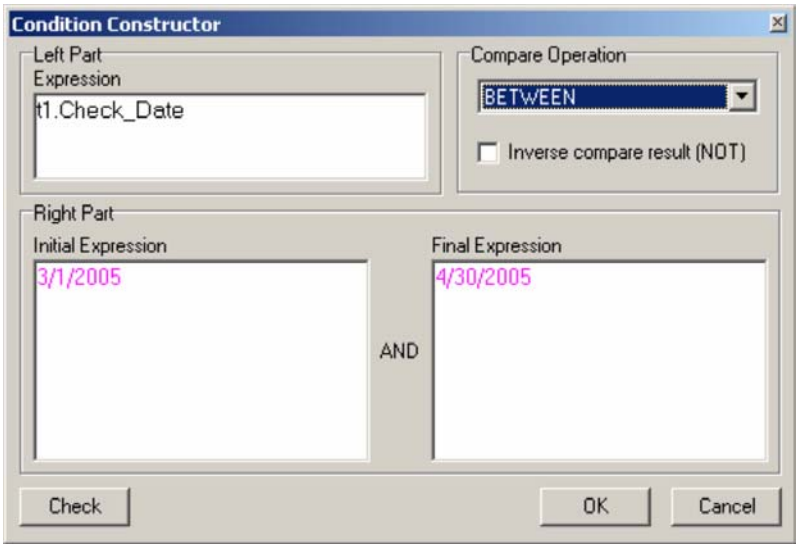

- Clicking in the Right Part Initial Expression box will open the Expression Editor window, shown below. This is where the beginning date of the date range is to be entered. To do that, select Date in the Type dropdown and enter a date in the format shown below in the Value box. Pressing Enter or Return on the keyboard with the cursor in the Value box will put the date into the Expression box above. Click OK.



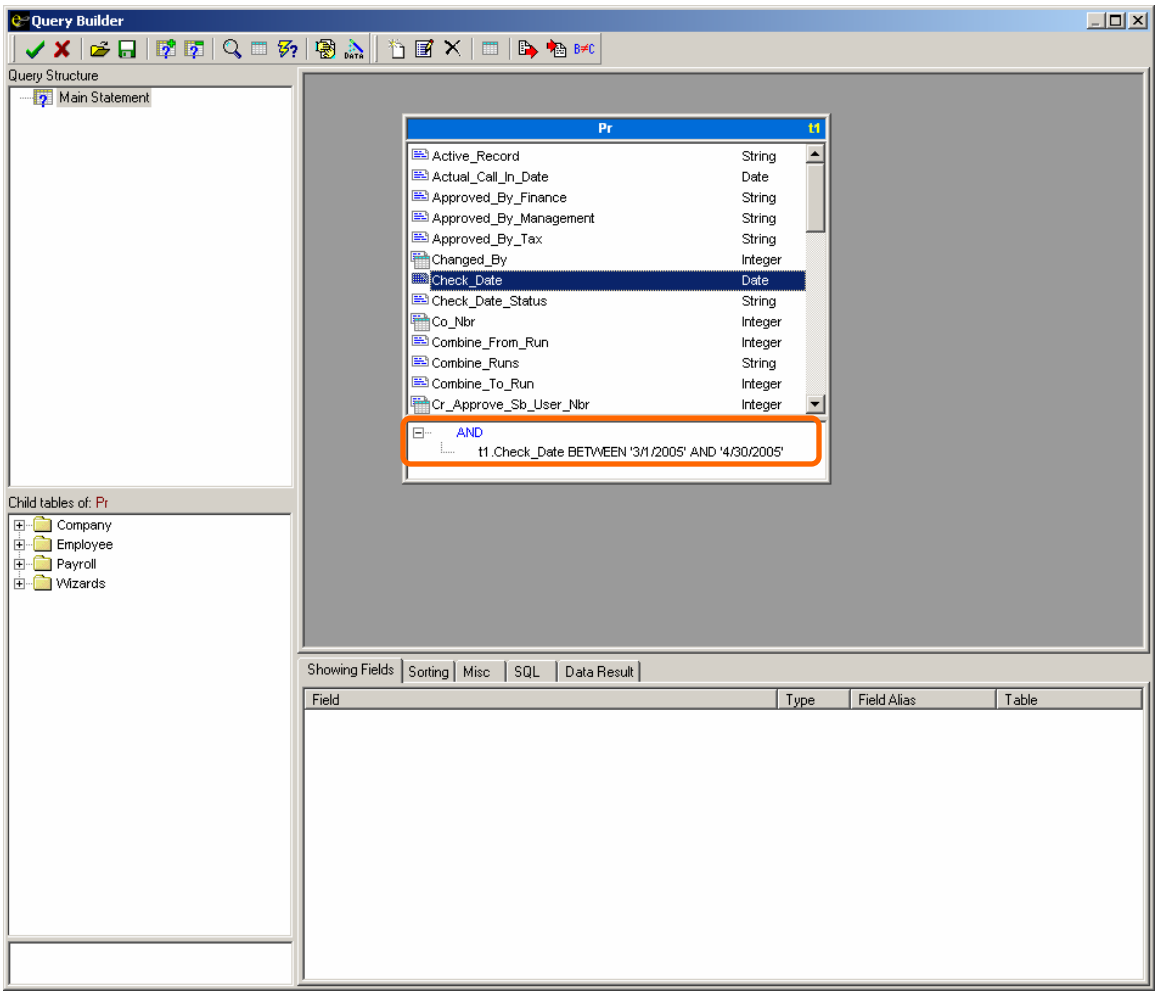
- At this point, the Condition Constructor window should be visible, the Right Part Initial Expression box populated with the date entered in the prior step. The same steps should be followed to set the ending date of the date range. The only difference is that this time, the Right Part Final Expression box should be clicked to bring up the Expression Editor window.




- After those steps are taken for the Final Expression, the window should look like this. When it does, click OK.



- At this point, the condition has been created. The condition will be visible at the bottom of the table in the Work area, shown below.

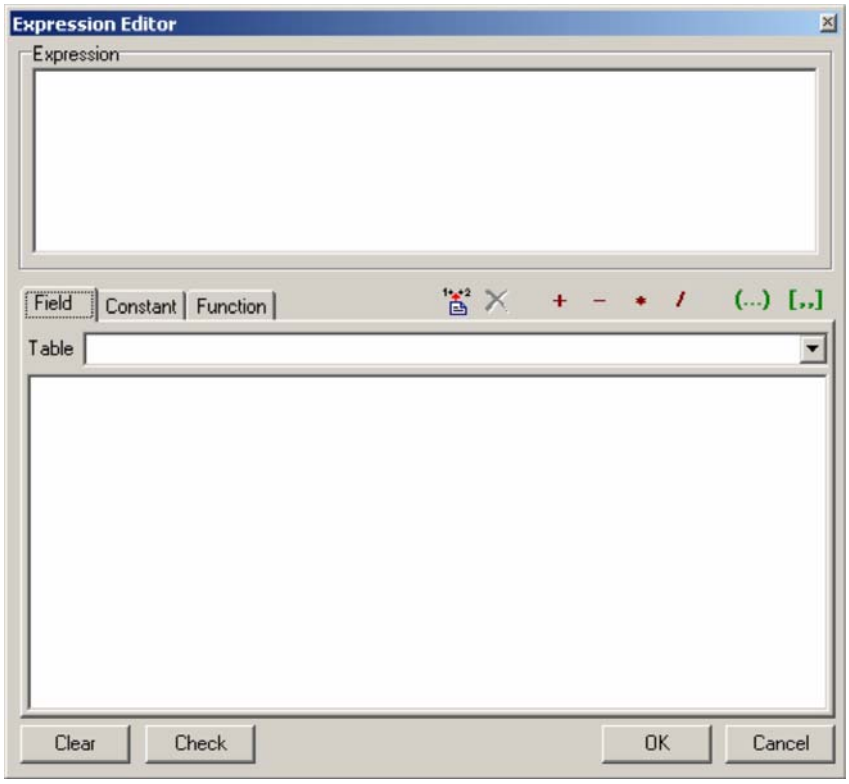


## 1.4.3 Expression Editor

The Expression Editor is used to by the Condition Constructor and in the Showing Fields tab to define conditions and to define what a field on the Showing Fields tab will show. Conditions were explained in the previous section. This section will show the Expression Editor in relation to the showing fields tab specifically. However, the ideas explained here may also be used in the Condition Constructor window.

The Condition Constructor window, shown below has three main areas:

- **Expression Box**
- **Tab Area**
- **Button Panel**




The ***Expression Box*** shows the current expression that has been built using the various features and functionality of the Tab Area.

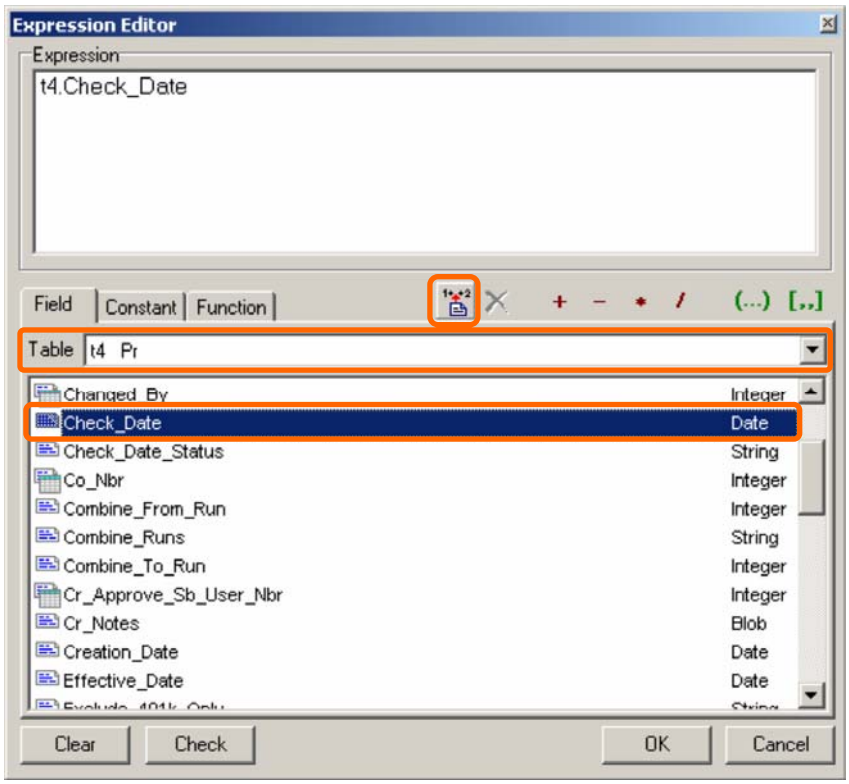
The ***Tab Area*** is where the pieces of the expression are selected. It is made up of 3 tabs:

- **Field**
- **Constant**
- **Function**

Regardless of tab, when the expression part is ready to be inserted into the Expression Box above, the Insert button is clicked to insert the part from the Tab Area into the Expression Box. The value from the Tab Area will always be inserted into the current cursor position of the Expression Box.

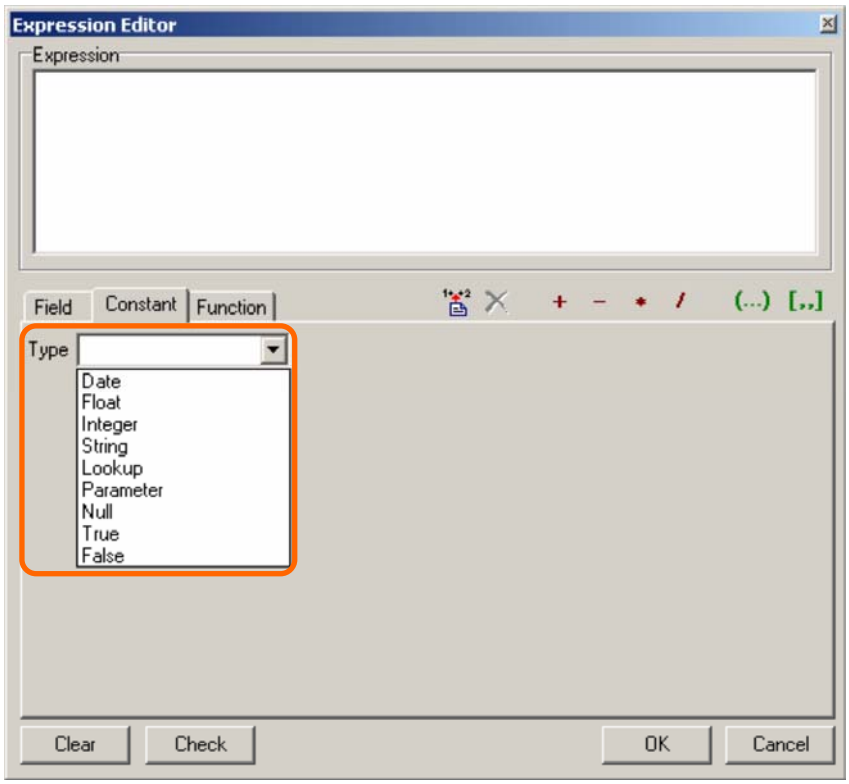
### 1.4.3.1 Field Tab

The ***Field Tab*** allows the user to select a table, and then a field within that table to insert into the expression. The table and field chosen must exist in the current level of the query in order to be available for selection here. With the table and field selected here, the expression part is ready to be inserted into the Expression Box.



## 1.4.3.2 Constant Tab

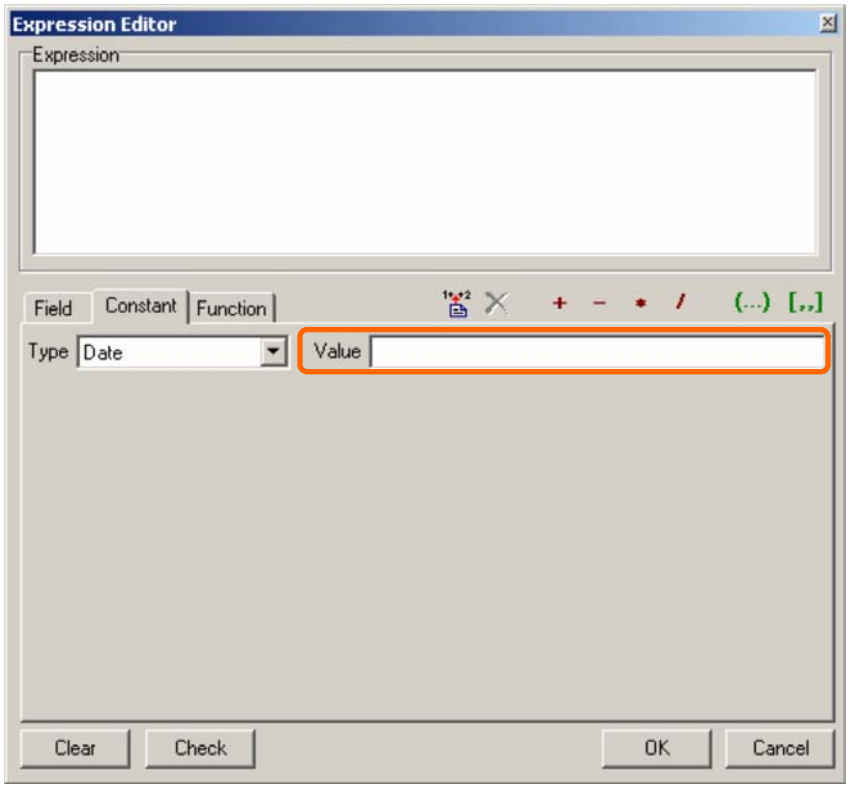
The ***Constant Tab*** allows the user insert a constant (unchanging value) into the Expression Box. For every constant, a type must be selected from the Type dropdown. The selected type will drive how the value of the constant is to be defined. For example, if Integer is selected as the type, the value of the constant must be a whole number, negative of those numbers or 0 (…-2, -1, 0, 1, 2…).



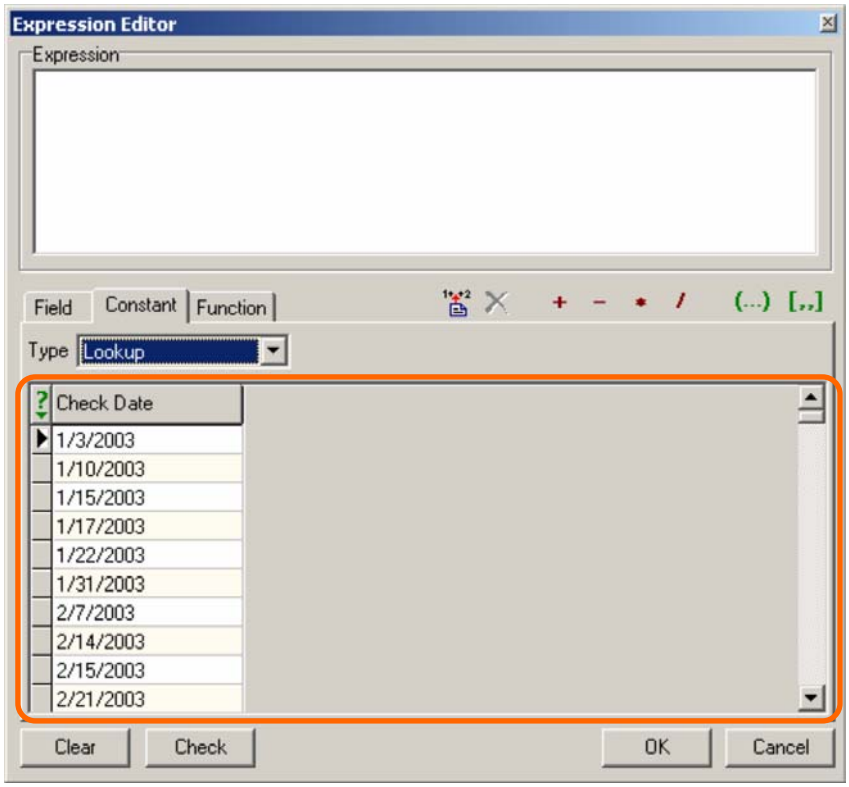

The following is a description of each type.

- ***Date*** – Date formatted as day/month/year. The year may be either a 2- or 4-digit year.
- ***Float*** – Number, possibly with decimal positions.
- ***Integer*** – Number with no decimal positions.
- ***String*** – Value made up of any combination of numeric, alphabetic or punctuation characters. Values of this type will always be enclosed in single quotes.
- ***Lookup*** – Special type that shows a list of valid values for a single table field located in the Left Part Expression box. For example, if the Left Part Expression box has the **Check_Date** field from the **Pr** table, selection the Lookup type in the Right Part Expression box's Expression Editor will show a list of check dates that exist in the current company.
- ***Parameter*** – Special type only for use in queries that are part of Report Writer or Report Master reports. Not for use inside Report Writer Wizard reports. Used to pass values from Report Writer or Report Master into Query Builder for use in conditions and the Showing Fields tab. For example, there is a parameter in Report Master called **Payrolls** that is part of one of the report templates. That parameter stores a list of payrolls selected on the input form of the report at runtime. That parameter may be used in queries inside this report to create a condition where the key value of the **Pr** table, **Pr_Nbr**, is equal to the **Payrolls** parameter, filtering out all payrolls other than those selected.
- ***Null*** – Empty value.
- ***True*** – Boolean value True. Can be used in function that require boolean parameters. Can also be used to as the left or right part of a condition to filter based on whether a condition in the other part is true.
- ***False*** – Boolean value False. Can be used in function that require boolean parameters. Can also be used to as the left or right part of a condition to filter based on whether a condition in the other part is false.

If the Date, Float, Integer or String types are selected, the Value box will appear. After the value is entered in the Value box, Enter or Return may be pressed on the keyboard to insert the constant into the Expression Box.

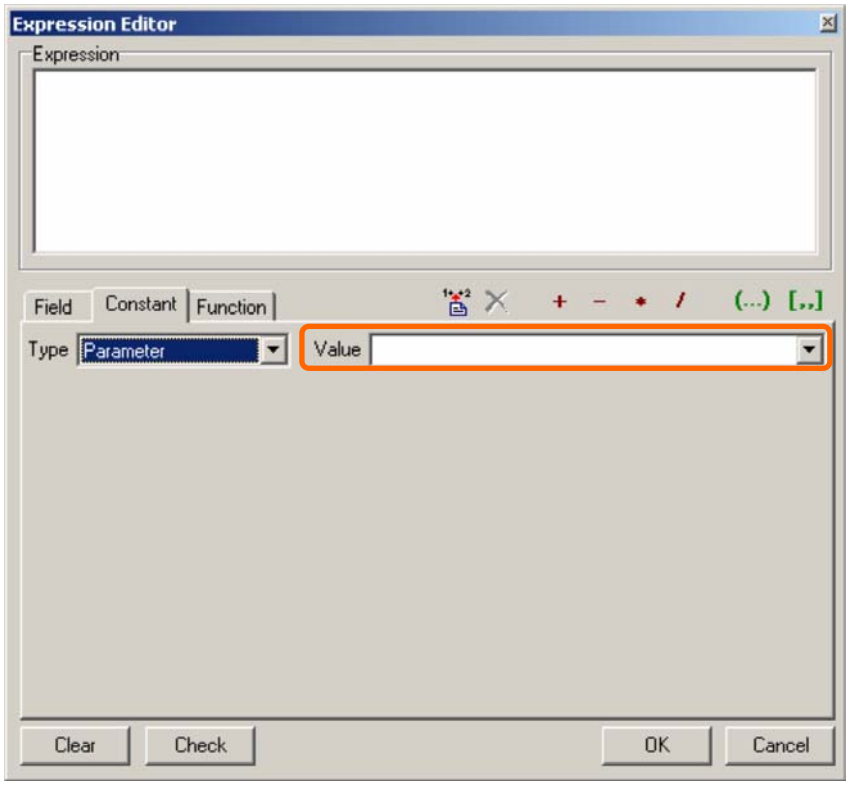



If the Lookup type is selected, a grid will appear below the Type dropdown with a list of valid values for the Left Part Expression. The item selected may be double-clicked on to insert it into the Expression Box.



# Evolution Query Builder

If the Parameter type is selected, the Value box will appear to the right of the type dropdown. If the query is inside of a Report Writer report, the parameter name must be typed into the Value box. If the query is inside of a Report Master report, the Value box will be a dropdown from which any parameter in the parent report may be selected.



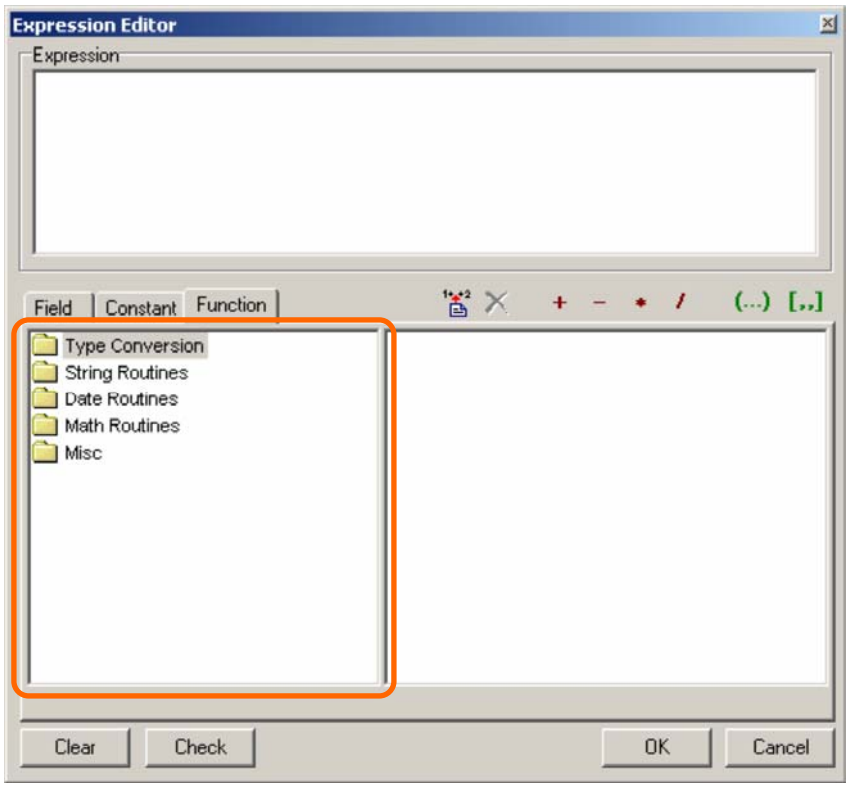
### 1.4.3.3 Function Tab

The ***Function Tab*** allows the user to apply a variety of predefined functions to field values, calculated values, variables or constants. Each function does something different and has a brief piece of documentation that describes what the function does, as well as its syntax for correct use.

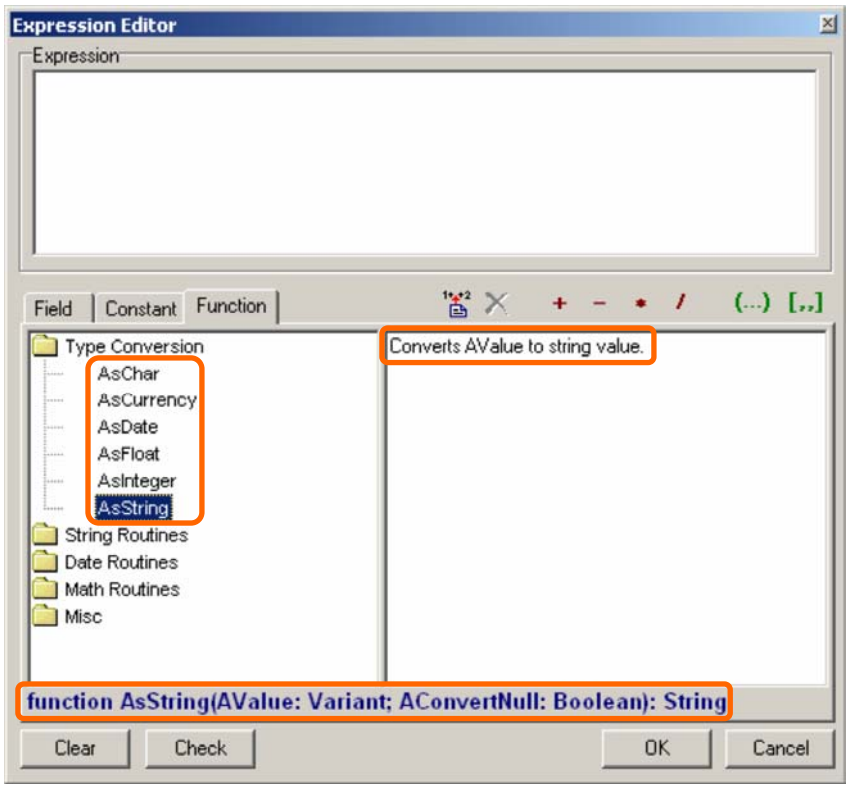
The available functions are divided into five folders by category:

- **Type Conversion**
- **String Routines**
- **Date Routines**
- **Math Routines**
- **Misc**

The category folders are located in the left pane of the Function Tab.



Double-clicking on a folder will show the functions inside that folder.




In the Expression Editor window above, the Type Conversion folder has been expanded and the AsString function selected in the left pane of the Function Tab. With this function selected, a description of what it does appears in the right pane of the tab:

**Converts AValue to string value.**

This description refers to a variable **AValue**. The description is saying that the function will convert the value found in the **AValue** variable to a string, which is a specific type of data.

With the AsString function selected, an emboldened line of text appears below the two panes in the Function Tab. This text shows the syntax of the function.

**Function AsString(AValue: Variant; AConvertNull: Boolean): String**

The first word, “Function”, says that this is a function.

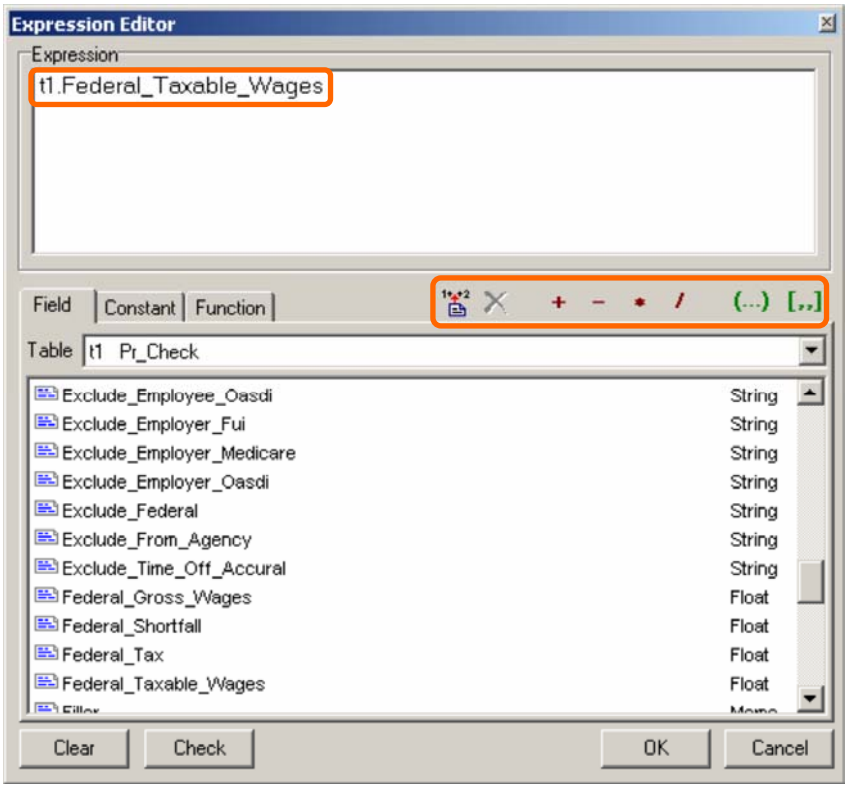
The name of the function immediately follows the word “Function”. This means that the name of the function is “AsString”, since that is what immediately follows “Function”.

Next is the opening parenthesis. This means that the following text describes the variables being passed into the function for the function to perform its job on. Each variable being passed to the function is separated from the next by a semicolon. Within each variable is the variable name and variable type, separated further by a colon.

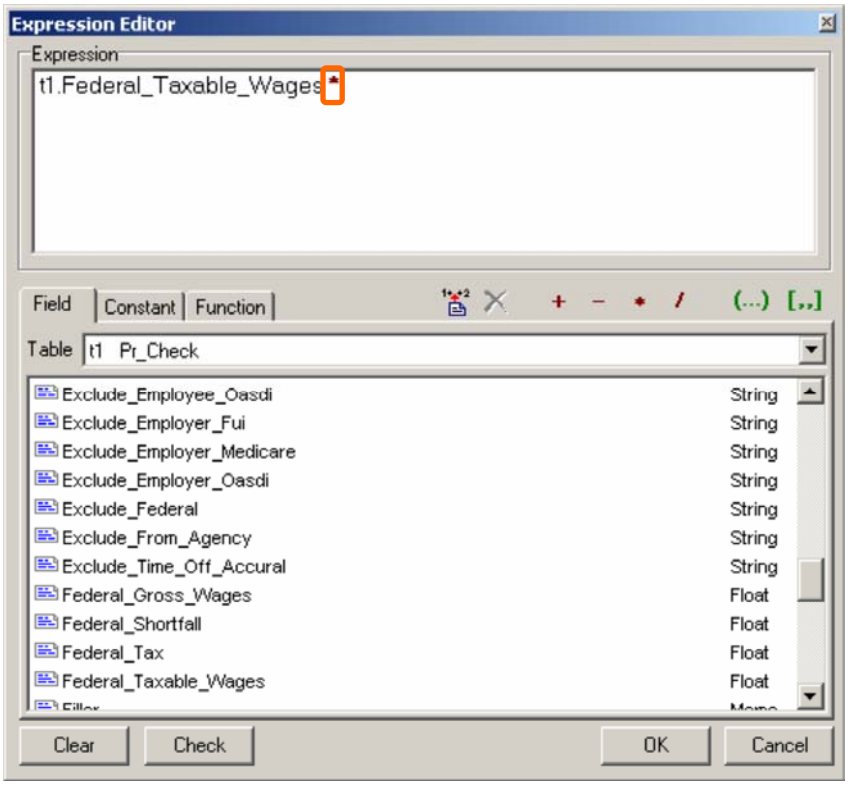
## 1.4.3.4 Button Panel

The Expression Editor's ***Button Panel*** allows a user to incorporate arithmetic into the Expression Box. For example, to show a dollar amount in a report with an implied decimal point (i.e. $125.50 would show as 12550), the simplest way is to multiply the dollar amount by 100. This is shown below using the example of multiplying the **Federal_Taxable_Wages** field from the **Pr_Check** table.

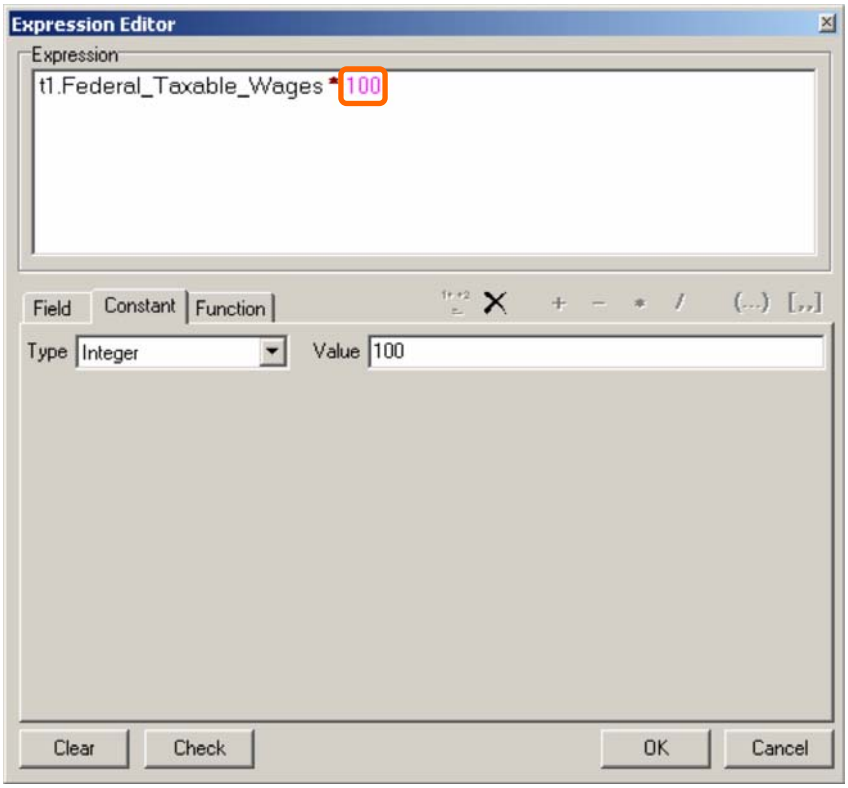
- First, the value to convert must be inserted into the Expression Box. In this case, that is the **Pr_Check** table and the **Federal_Taxable_Wages** field.



- The value in the Expression box needs to be multiplied by 100. The concatenation operator is the asterisk (*). To do this, while the cursor is to the right of the **Federal_Taxable_Wages** field, the Multiply button is clicked to insert the multiplication operator after the field that is to be multiplied by 100.



- With the value to multiply and the multiplication operator in the Expression box, the last piece of the expression is the value to multiply by. The number to multiply by is always going to be 100 in this example. Because the value will always be the same, it is a constant, to be defined on the Constant Tab. The constant is the number 100, which is an integer. This means that Integer is the appropriate selection in the Type dropdown. The constant 100 needs to be inserted immediately after the multiplication operator.

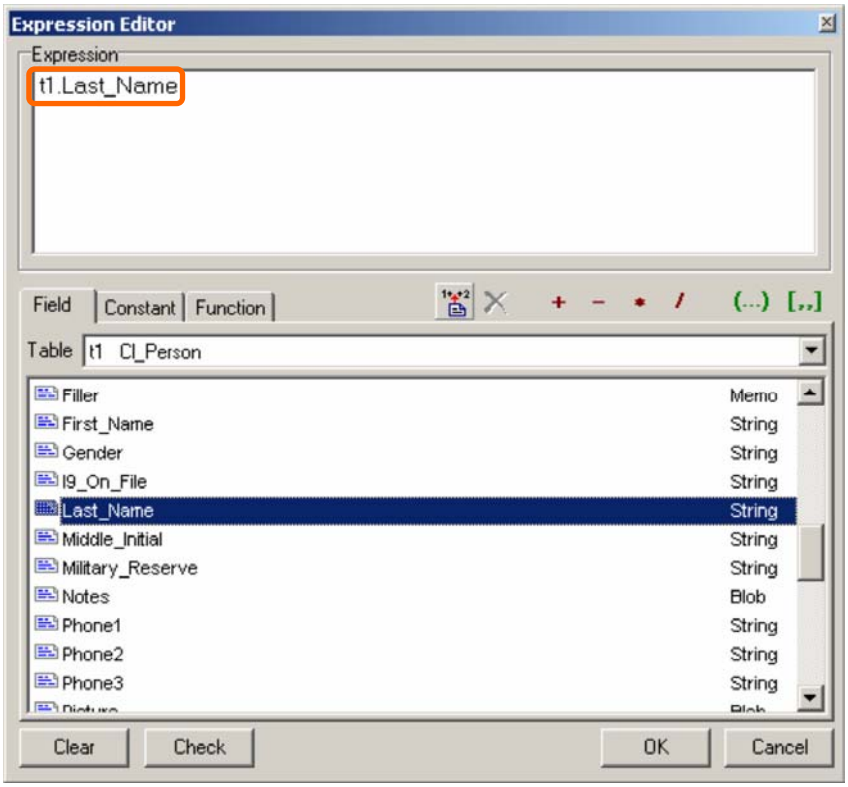



- Clicking the OK button in the lower right of the Expression Editor window saves the changes to the expression.

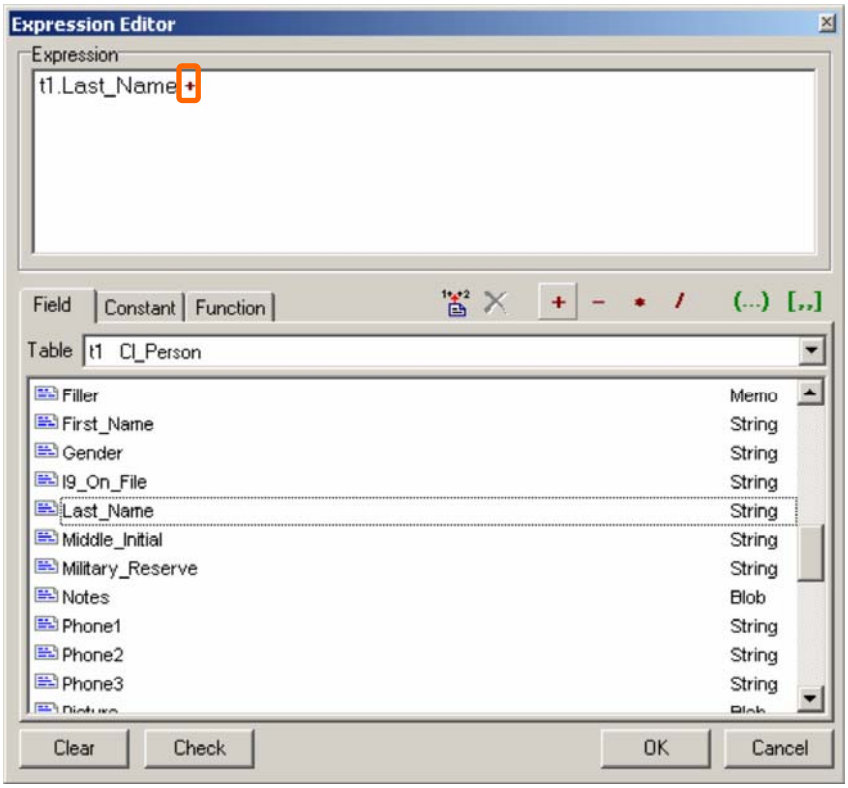
In the case of string expressions, the Button Panel allows for the concatenation of multiple strings into a single string.

For example, the user may wish to see the full name of the employee in a single field. In Evolution, the employee name is stored in three different string-type fields in the **CI_Person** table: **First_Name**, **Middle_Initial** and **Last_Name**. It is possible to show all three of these table fields in the same query field via concatenation. This example is shown below. The ending format will be LastName, FirstName MiddleInitial

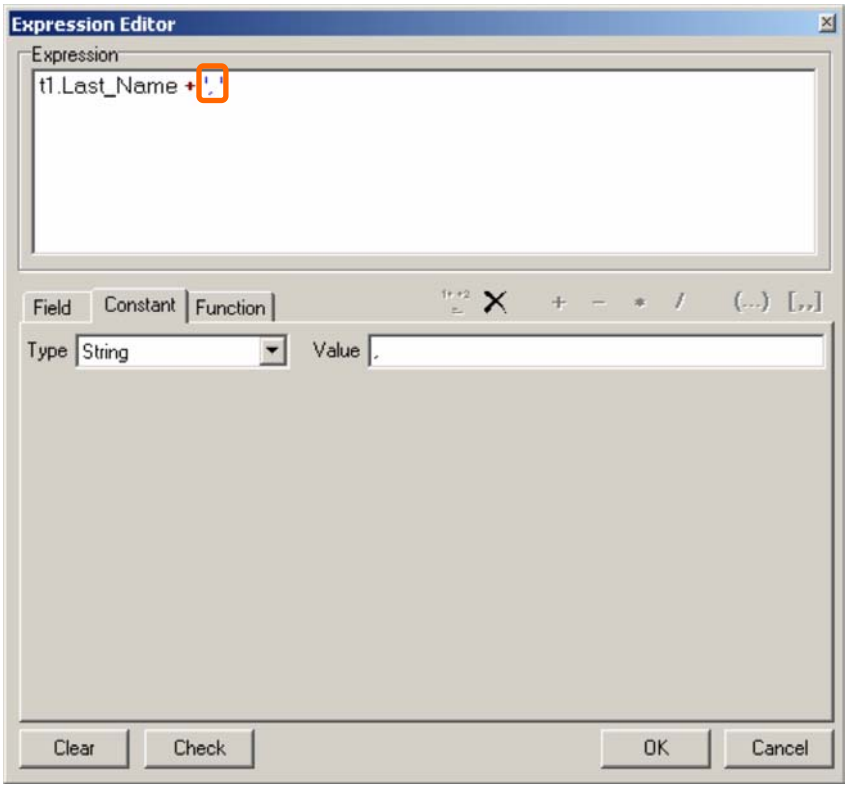
- The first value to be concatenated must be inserted into the Expression Box. In this case, that is the **CI_Person** table and the **Last_Name** field.



- The value in the Expression box needs to be concatenated with the next part of the expression. The concatenation operator is the plus sign (+). This must be inserted into the Expression Box after the **Last_Name** field. To do this, while the cursor is to the right of the **Last_Name** field, the Add button is clicked.



- The next string to be concatenated is the comma character. This is a constant, so it is inserted in the same way that any other string constant would be inserted. This needs to be done while the cursor is immediately following the previously added concatenation operator.



- The next piece to be inserted is another concatenation operator, done the same way as the first, with the cursor immediately following the comma portion of the expression.

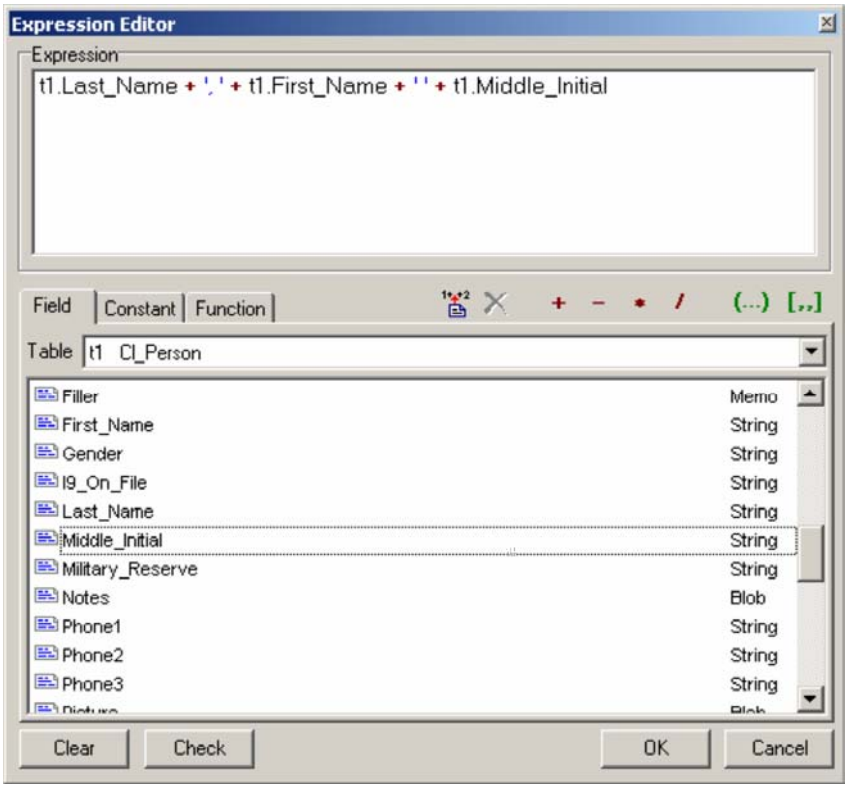
- The **First_Name** field comes next. This is inserted the same way the **Last_Name** field was inserted, immediately following the last concatenation operator.

- Another concatenation operator is inserted.

- The next string to be concatenated is the space between the **First_Name** and **Middle_Initial** fields. This is inserted the same way the comma was inserted, replacing the comma character with a space character in the Value box.

- One more concatenation operator is inserted here.

- The final piece is the **Middle_Initial** field. This field is inserted just like the other two fields inserted into the Expression Box in this example. The end result should look like this:



## 1.4.4 Keys and Joins

### 1.4.4.1 Keys

Sometimes data must be selected from two or more tables to get the desired result. Joins allow this to be accomplished.

Database tables are often referenced by other database tables via the key fields of those tables. A table's primary key is the column in that table with a unique value for each row of data. The purpose of referencing a table via its key is to associate data from one table with that of another.

For the examples in this section, the tables involved are **Pr_Check** and **Ee**. These two tables are associated with one another via the primary key of the **Ee** table – **Ee_Nbr**. The **Ee_Nbr** field in the **Pr_Check** table indicates the owner of that check.

In the **Ee** table below, the **Ee_Nbr** field is the primary key, meaning that no two rows can have the same value in the **Ee_Nbr** field. The **Ee_Nbr** field is unique and can be used to distinguish between two different people, regardless of other employee similarities.

In the tables below:

- The **Ee_Nbr** column stores the primary key of the **Ee** table .
- The **Pr_Check_Nbr** column stores the primary key of the **Pr_Check** table .
- The **Ee_Nbr** column in the **Pr_Check** table is used to reference a unique employee in the **Ee** table without using the employee's **Custom_Employee_Number**.

**Ee:**

| Ee_Nbr | Custom_Employee_Number |
|---|---|
| 1 | A100 |
| 2 | A200 |
| 3 | B100 |
| 4 | C100 |

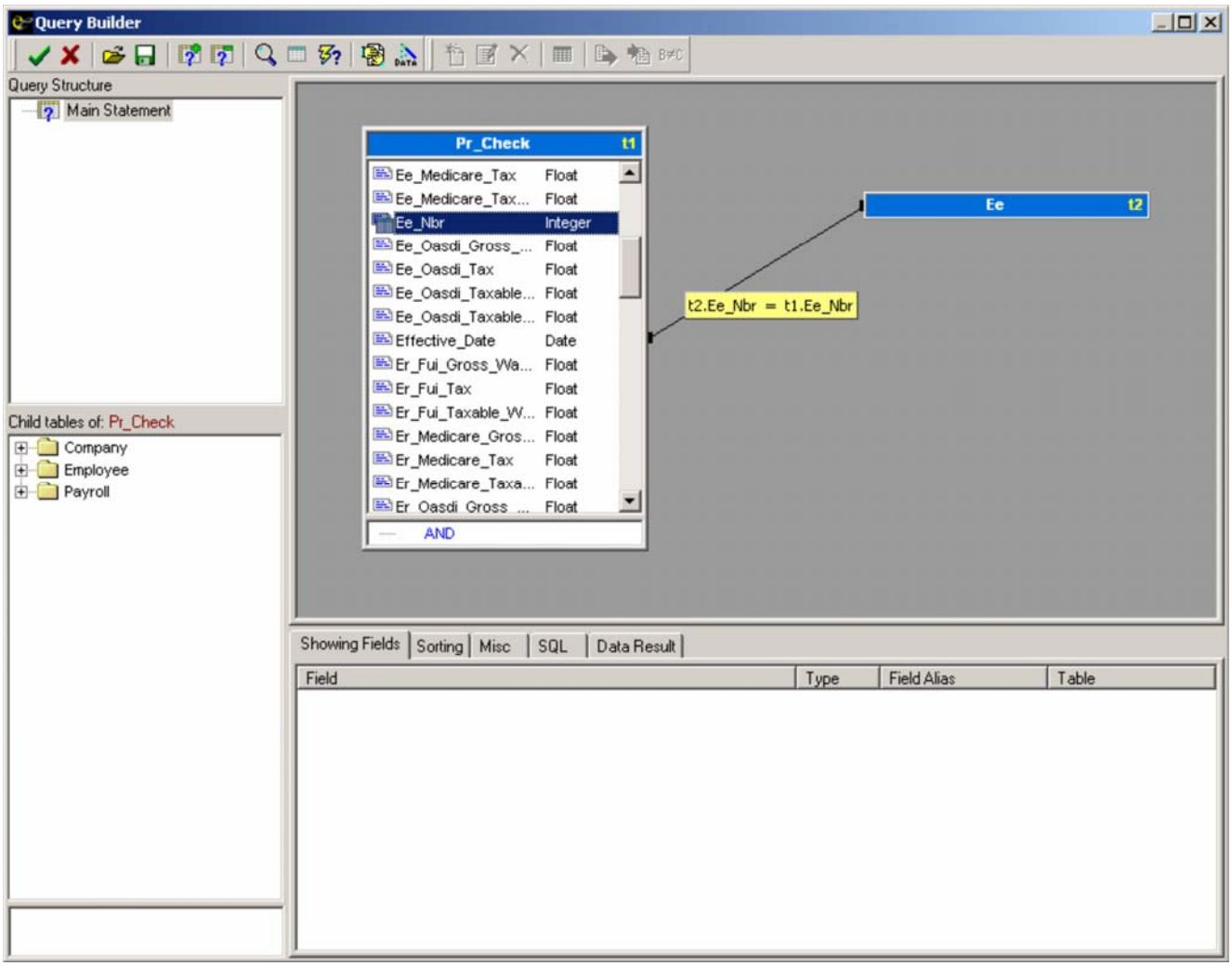
**Pr_Check:**

| Pr_Check_Nbr | Payment_Serial_Number | Check_Type | Ee_Nbr |
|---|---|---|---|
| 1 | 1024 | R | 1 |
| 2 | 1025 | R | 2 |
| 3 | 1026 | R | 4 |
| 4 | 1027 | R | 1 |
| 5 | 1028 | R | 2 |
| 6 | 1029 | R | 4 |
| 7 | 1030 | M | 1 |
| 8 | 1031 | M | 4 |

## 1.4.4.2 Joins

In Query Builder, there are two types of joins – **inner** and **outer**. Joins are the way data from one table may be matched up with data from a different table via that second table's primary key. They are created by dragging the field to be joined on from one table, and dropping that field either into an empty part of the Work Area, or on top of the other table to be involved in the join.

Dragging and dropping a foreign key field from a table in the Work Area into an empty part of the Work Area will add the table whose primary key field was dropped. In the example below, the **Pr_Check** table has been dropped into the work area already. The **Ee_Nbr** field was dragged and dropped into the Work Area from the **Pr_Check** table, adding the **Ee** table since the **Ee_Nbr** field is the primary key of the **Ee** table:




By default, the tables are joined via an inner join. That join can be changed to an outer join, as explained later.

In the case that both tables already exist in the query with no join, they can be joined in a similar way. The **Ee_Nbr** field can be dragged from the **Pr_Check** table and dropped onto the **Ee** table. The Add Join window will appear as shown below:


The Add Join window has the following dropdowns:

- **Table 1 –** The first table to be included in the join.
- **Field [Table 1] –** The field from Table 1 being joined and matched on.
- **Join –** The type of join. Join types include:
    - **INNER JOIN**
    - **OUTER [Table 1]**
    - **OUTER [Table 2]**
- **Table 2 –** The second table to be included in the join.
- **Field [Table 2] –** The field from Table 2 being joined and matched on.

The Table 1 and Table 2 dropdowns will include any table that exists in the currently selected subquery.

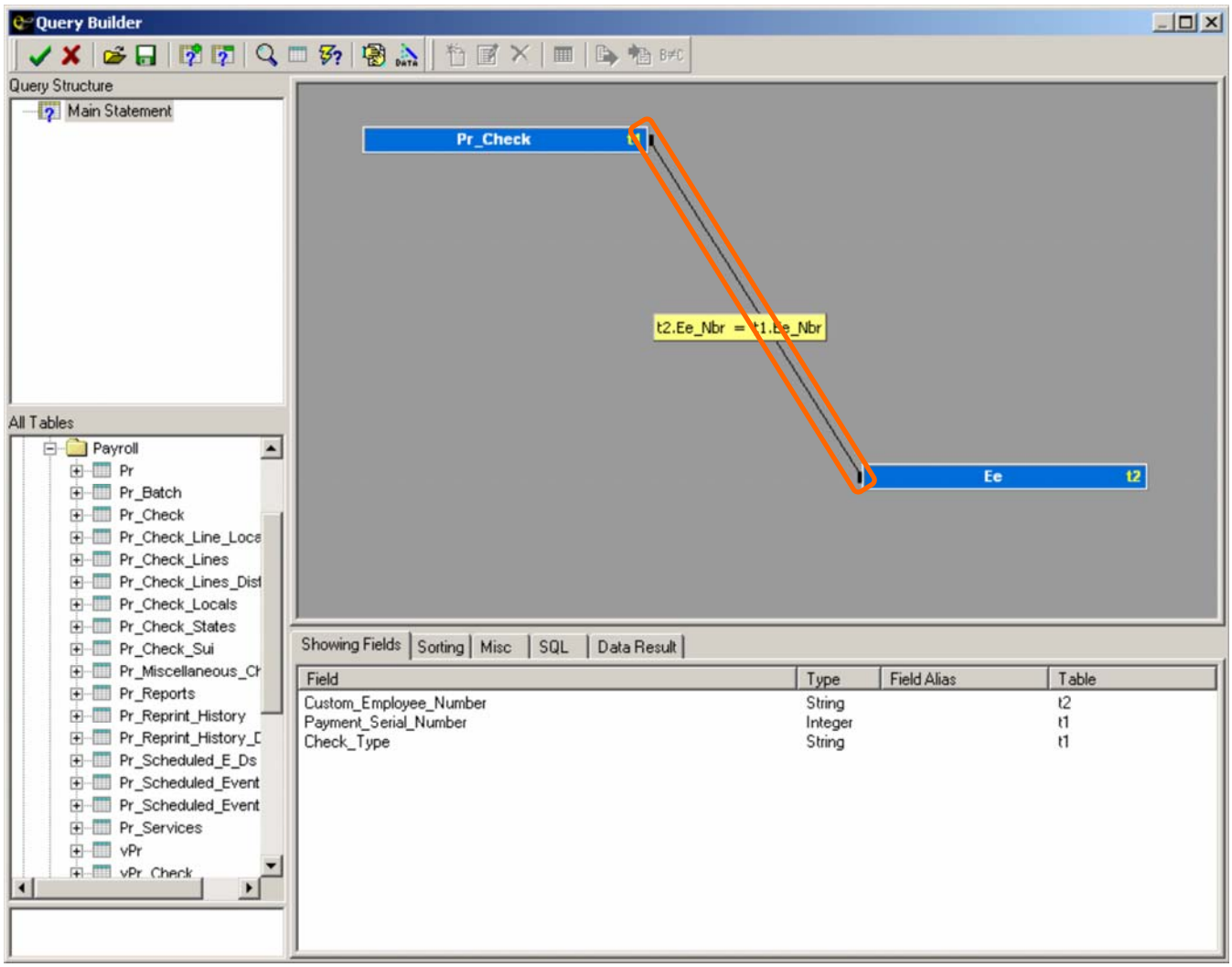
The Field dropdowns will show all fields included in the table selected in the corresponding Table dropdown

For the outer join type options, the table specified is referred to as Table 2 in the examples that follow.

**INNER JOIN –** Returns all rows from both tables where the value of the field being joined on in table 1 also exists in table 2.

In the example below, if there are rows in **Ee** that do not have matches in **Pr_Check** (in this example, **Ee_Nbr** = 3), those rows will not be included in the result.

The following example shows an inner join represented by a solid black line. This query will return **Custom_Employee_Number**, **Payment_Serial_Number** and **Check_Type** for all employees for which at least one matching check exists:



# Evolution Query Builder

Below is a diagram showing the **Ee** table on the left and the **Pr_Check** table on the right. Lines are drawn linking each row in the **Pr_Check** table with its corresponding row in the **Ee** table:

| Ee_Nbr | Custom_Employee_Number |
|---|---|
| 1 | A100 |
| 2 | A200 |
| 3 | B100 |
| 4 | C100 |

| Pr_Check_Nbr | Payment_Serial_Number | Check_Type | Ee_Nbr |
|---|---|---|---|
| 1 | 1024 | R | 1 |
| 2 | 1025 | R | 2 |
| 3 | 1026 | R | 4 |
| 4 | 1027 | R | 1 |
| 5 | 1028 | R | 2 |
| 6 | 1029 | R | 4 |
| 7 | 1030 | M | 1 |
| 8 | 1031 | M | 4 |

Based on the example tables above, this query will return the following result:

| Custom Employee Number | Payment_Serial_Number | Check_Type |
|---|---|---|
| A100 | 1024 | R |
| A100 | 1027 | R |
| A100 | 1030 | R |
| A200 | 1025 | R |
| A200 | 1028 | R |
| C100 | 1026 | R |
| C100 | 1029 | M |
| C100 | 1031 | M |

**Custom_Employee_Number** B100 does not exist in the **Pr_Check** table. As a result, the inner join excludes data where **Custom_Employee_Number** = B100.

**OUTER JOIN –** Returns all rows from table 1, even if there are no matches in table 2. If rows exist in table 1 that do not have matches in table 2, those rows from table 1 will still be listed.

For outer joins, Query Builder allows the user to specify which table is table 1 and which is table 2. Right-clicking on a pre-existing and selecting the Join type option enables the user to select one of two OUTER options. Each will specify a different table. The table specified in the selected option will be table 2 for that outer join.

In the example below, if there are rows in **Ee** that do not have matches in **Pr_Check** (in this example, **Ee_Nbr** = 3), those rows will still be included in the result.

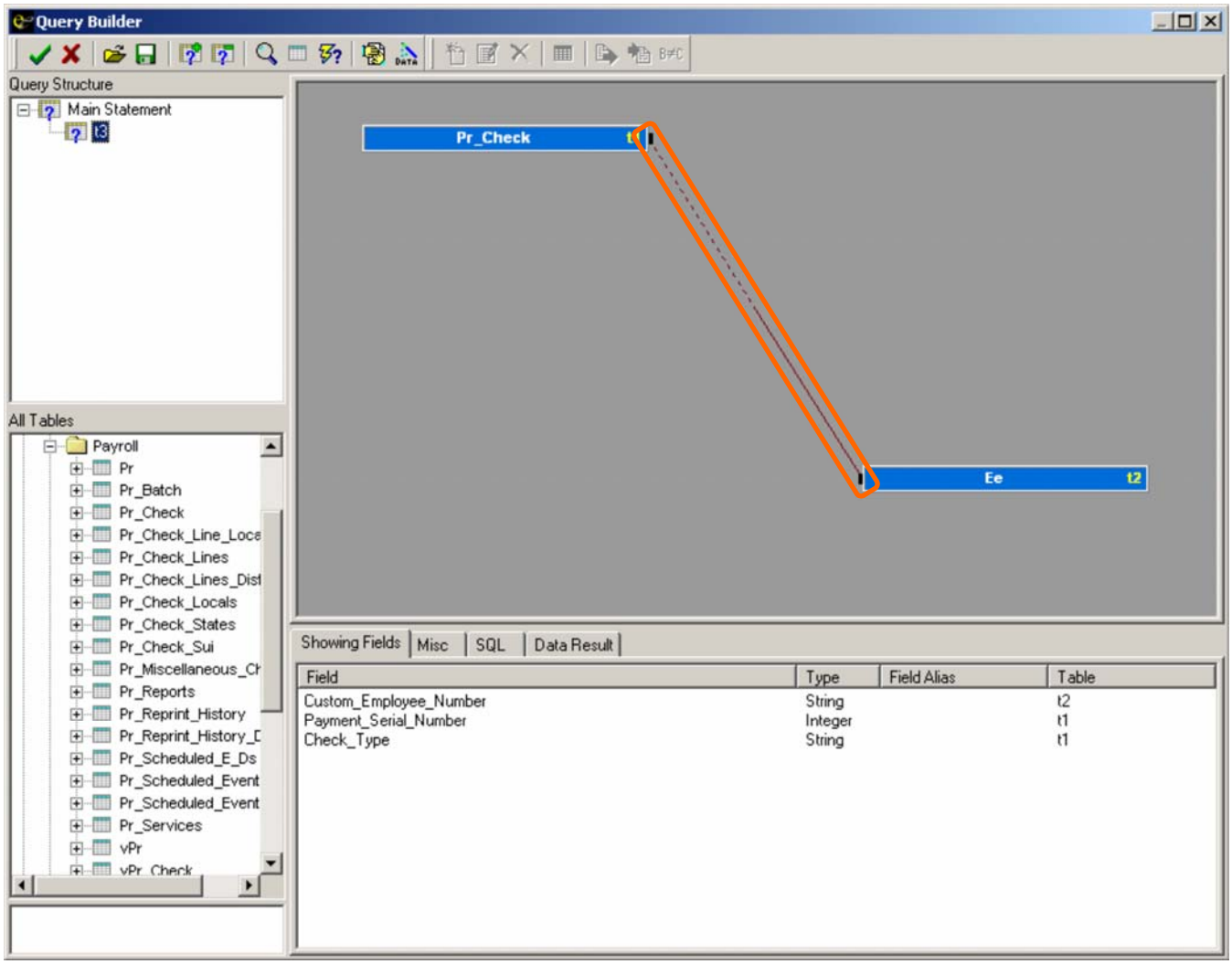
The following example shows an outer join represented by a dark red line, half solid and half dashed. The solid half is connected to the table in which data must exist in order for it to be included in the result (table 1). The dashed half is connected to the table in which data may or may not exist (table 2).

Data returned by an outer join includes:

- All rows from table 1, regardless of whether a matching row exists in table 2.
- Those rows in table 2 for which a match exists in table 1.

If table 2 includes rows that do not match those in table 1, those non-matching rows from table 2 will not be included in the result.

This query will return **Custom_Employee_Number**, **Payment_Serial_Number** and **Check_Type** for all employees, regardless of if a matching check exists:



Below are two tables being joined via an outer join.

| Ee_Nbr | Custom_Employee_Number |
|---|---|
| 1 | A100 |
| 2 | A200 |
| 3 | B100 |
| 4 | C100 |

| Pr_Check_Nbr | Payment_Serial_Number | Check_Type | Ee_Nbr |
|---|---|---|---|
| 1 | 1024 | R | 1 |
| 2 | 1025 | R | 2 |
| 3 | 1026 | R | 4 |
| 4 | 1027 | R | 1 |
| 5 | 1028 | R | 2 |
| 6 | 1029 | R | 4 |
| 7 | 1030 | M | 1 |
| 8 | 1031 | M | 4 |

If this outer join is performed on the **Ee** and **Pr_Check** tables above, the result will look like this:

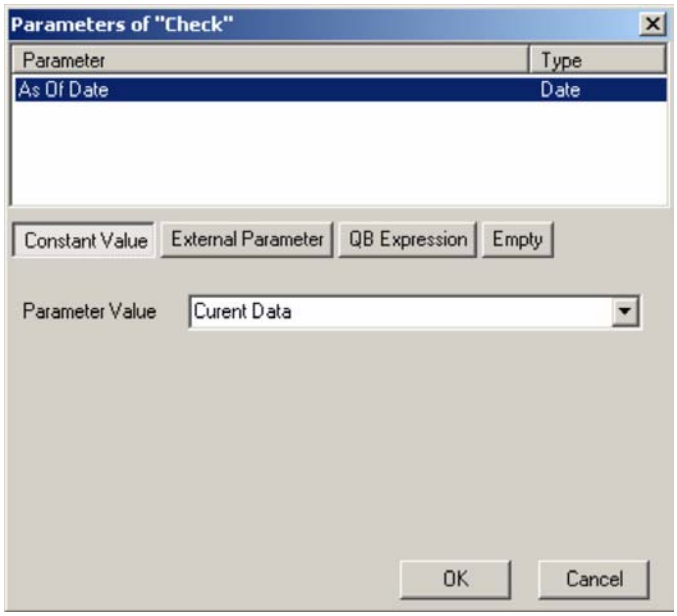
| Custom Employee Number | Payment_Serial_Number | Check_Type |
|---|---|---|
| A100 | 1024 | R |
| A100 | 1027 | R |
| A100 | 1030 | R |
| A200 | 1025 | R |
| A200 | 1028 | R |
| B100 | | |
| C100 | 1026 | R |
| C100 | 1029 | M |
| C100 | 1031 | M |

Note the row returned where **Custom_Employee_Number** = B100. There is no matching row in the **Pr_Check** table, so the result for that **Ee** row consists of the **Custom_Employee_Number** with no **Payment_Serial_Number** or **Check_Type**.

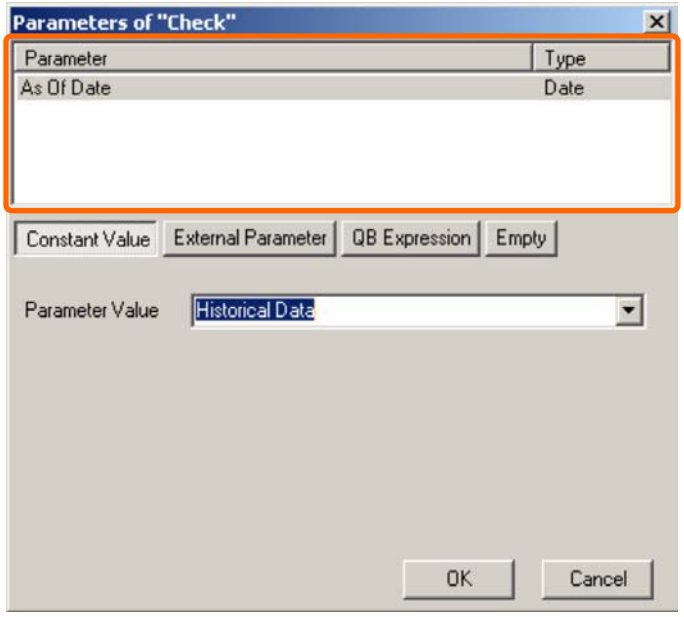
An extra option for outer joins is available by right-clicking on a particular table or subquery. The **Isolate Filtering from Outer Join** option tells Query Builder whether to apply the outer join before any conditions in that table, or vice versa. With this option checked, the outer join is performed first, filtering the result of the outer join. With the option unchecked, data is selected from the tables involved in the outer join, the individual tables are filtered as defined in the query and then the outer join is performed.

### 1.4.5 Table Parameters

Any table may have **table parameters**. Those parameters may be viewed in the Parameters of "<selected table>" dialog. A table parameter is a kind of condition applied to the table so that the set of data being worked with in that table is limited for efficiency. For example, almost every table in Query Builder has the As Of Date parameter by default. This parameter is used to determine as of what date data is to be viewed. By default, this parameter is set to Current Data.

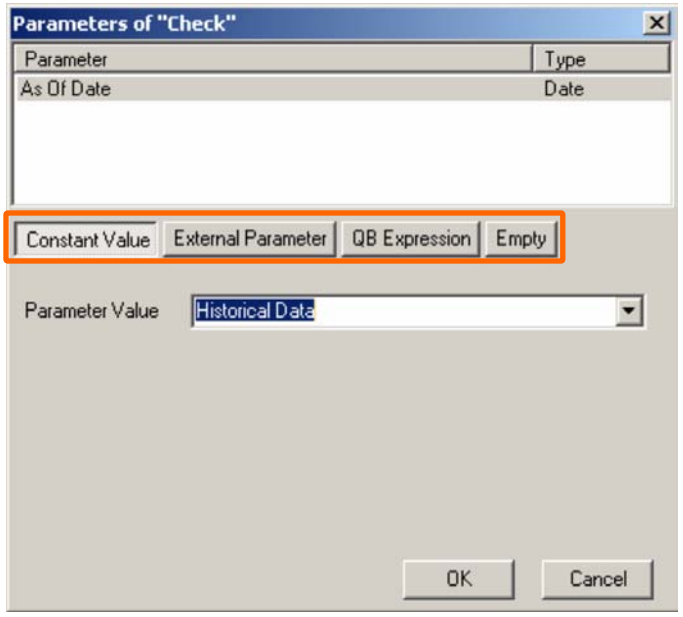


The parameters dialog shows the selected table's parameters in a list in the top of the dialog.




The bottom portion of the parameters dialog allows the user to define the parameter using one of the four available buttons:

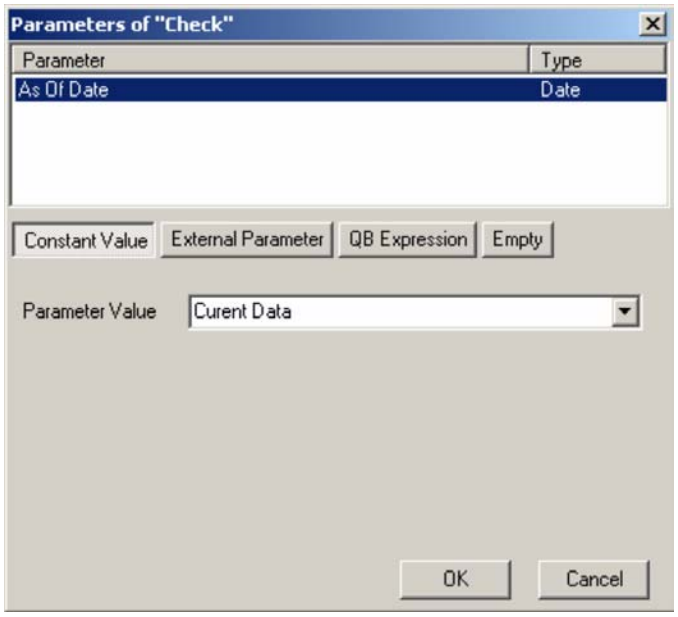
- **Constant Value**
- **External Parameter**
- **QB Expression**
- **Empty**



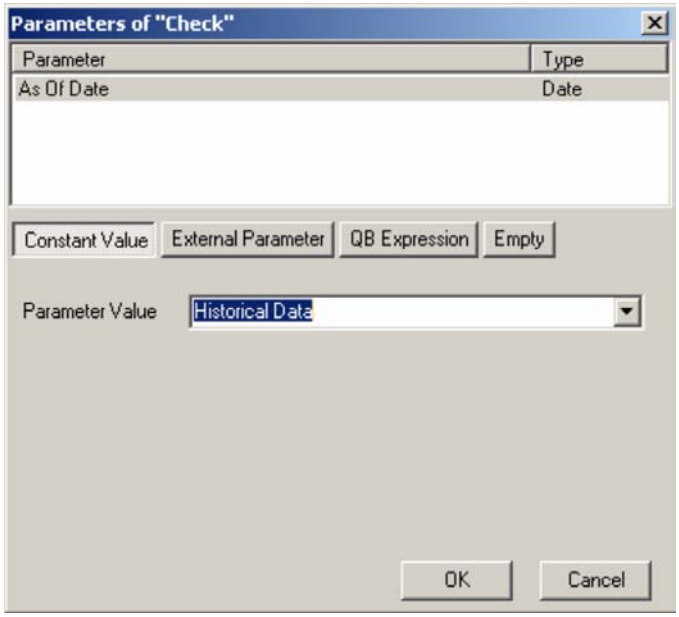

As of the date this document was written, the only useful button here is the Constant Value button. The other three will be useful in the future as virtual tables are added and completed.

# Evolution Query Builder

Because of the fact that data changes are tracked historically, that historic data may be fetched using Query Builder. By default, the As Of Date parameter is defined to fetch current data only. This is done by selecting Current data in the Parameter Value dropdown.

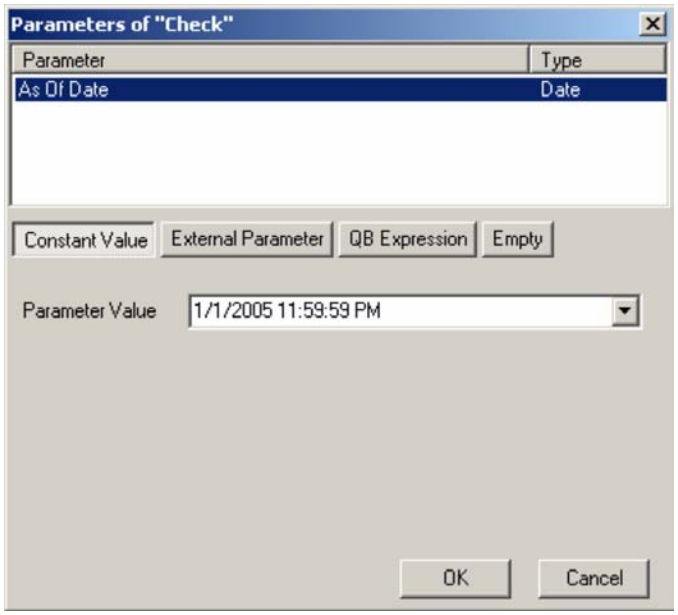



To fetch all historic data for a table, select Historical Data in the Parameter Value dropdown for the As Of Date parameter, shown below.



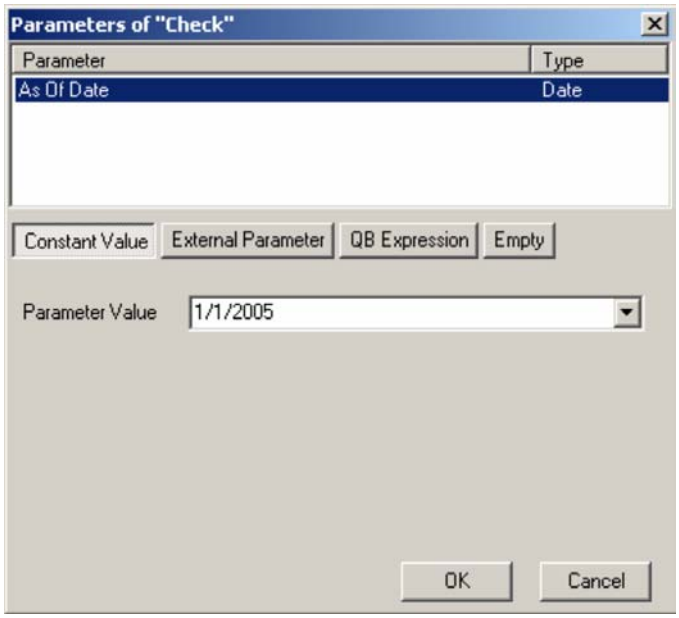

This can be a very useful troubleshooting tool when an issue may have possibly been caused by a field value being incorrect as of a specific date.

Data may also be fetched from a table as of a specific date and time. To do this, overwrite the Parameter Value dropdown with the date and time data is to be fetched as of. The format here is m/d/yyyy hh:mm:ss xm, where the time portion is optional. It is important to note that in Evolution terms, 1/1/2005 is before 1/1/2005 12:00:00 AM.



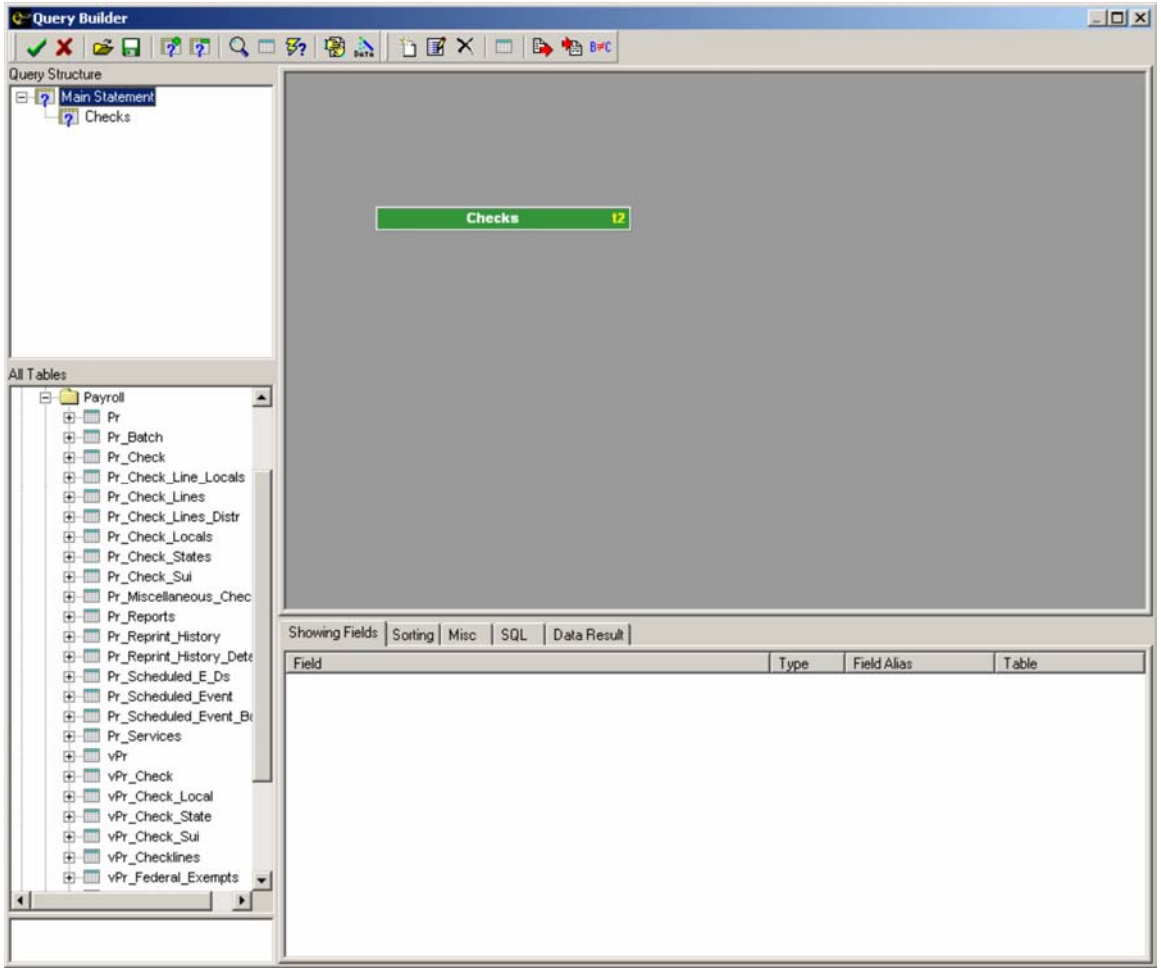

Defining the parameter above will return the row of data for that table that is effective as of 1/1/2005 11:59:59 PM, which happens to be the last second of the day.

The earliest effective date for 1/1/2005 is the date with no time. This is shown below.



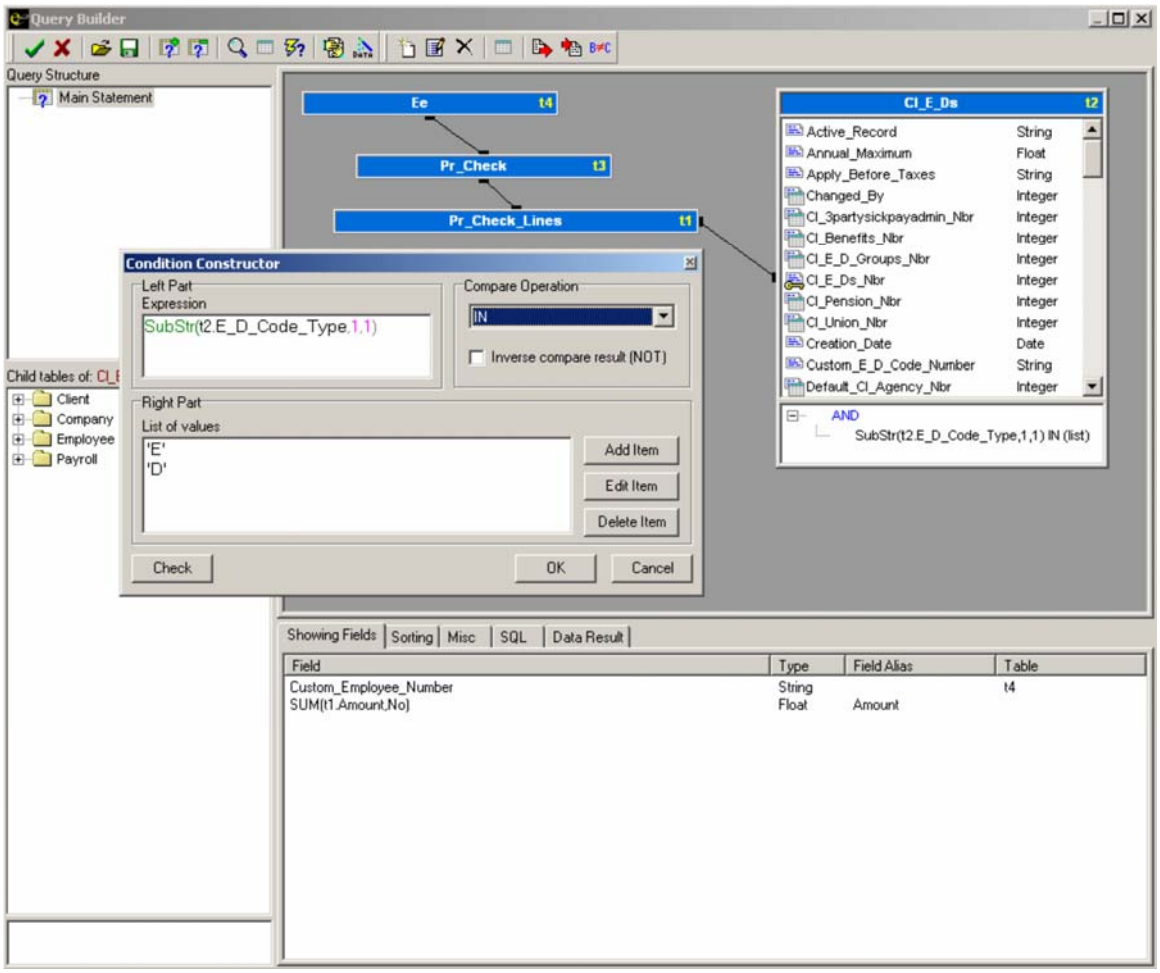
## 1.4.6 Subqueries

A subquery is a query that is part of another query. It is displayed in the Work Area in almost the same way as a table, except the title bar of a subquery is green instead of blue.



# Evolution Query Builder

The purpose of a subquery is to segregate different parts of a query from each other. This is useful when there are two pieces of information stored in the same column of the same table, but data from that column needs to be shown in different columns in the query result. An example of this would be a query that is to show a sum of all earnings in one column, and a sum of all deductions in another column. The amount of an earning or deduction is stored in the **Pr_Check_Lines** table in the **Amount** column. A query that included no subqueries and selected the sum of earnings and deductions for each employee would look like this:



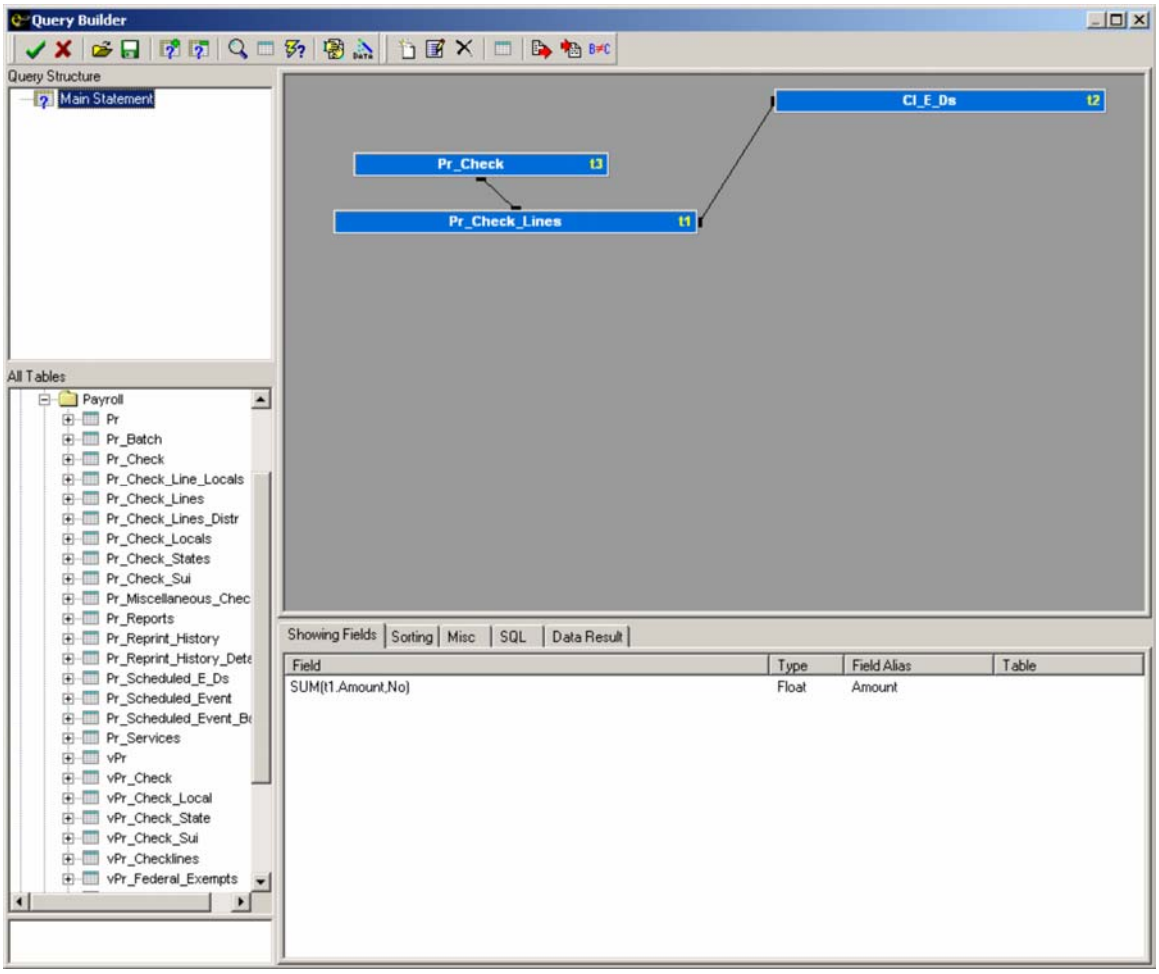

A condition exists on the **CI_E_Ds** table that allows the query to only return rows in the **Pr_Check_Lines** table where the matching row in the **CI_E_Ds** table has a value in the **E_D_Code_Type** column that starts with “E” or “D”. This query will return **Custom_Employee_Number** in the first column, and the sum of all earnings and deductions in the **Amount** column. This is because the condition on the **CI_E_Ds** table is applied to the whole query and cannot be applied to single fields in the Showing Fields Tab.

In order to get the desired result, this query must be split up into two subqueries. Each subquery will look much like the one just shown. There will be three differences:

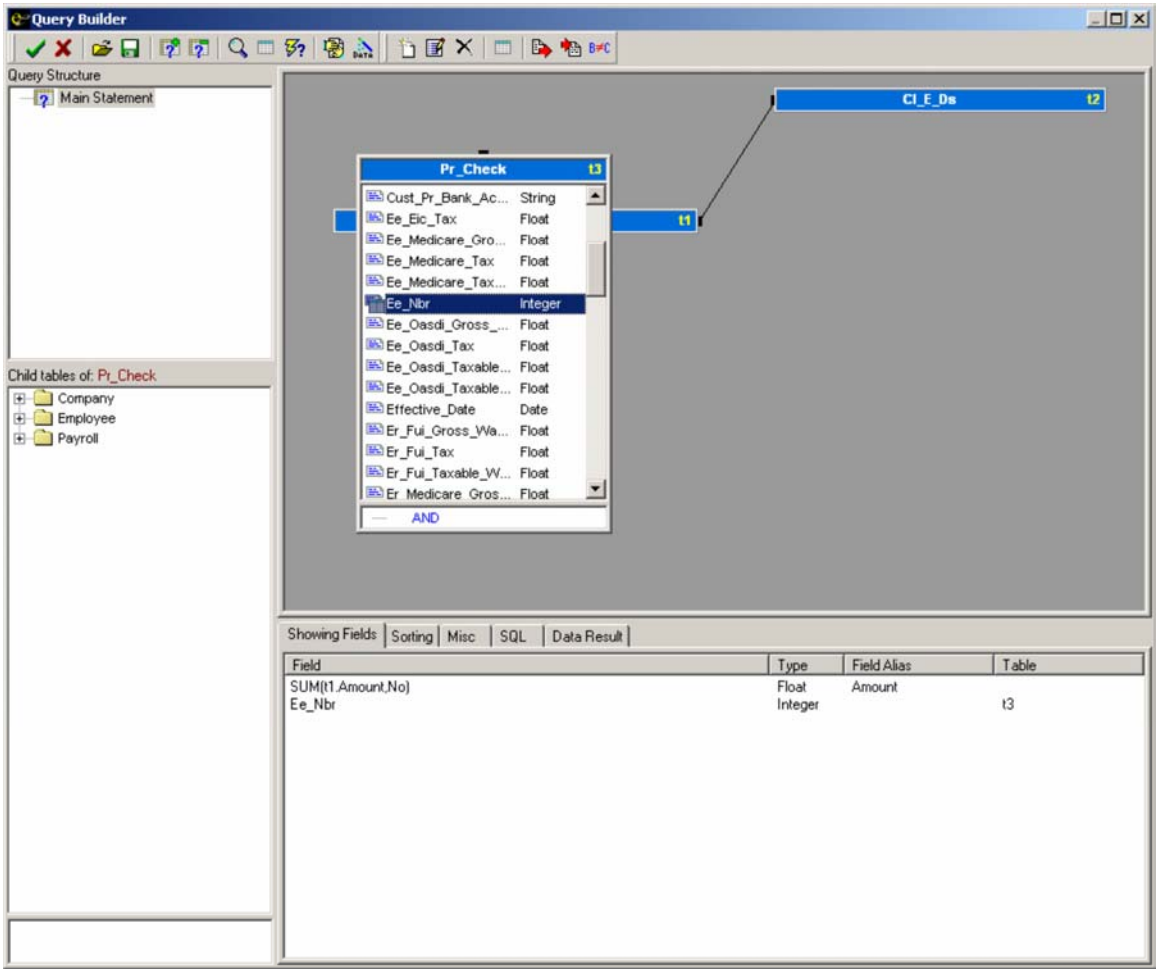
- The condition on the **Cl_E_Ds** table will be modified to only include "E" in the Right Part for one subquery, and "D" in the Right Part for the other subquery.
- The **Ee_Nbr** field will be included in each sub query's Showing Fields tab for joining purposes in the parent query.
- The **Ee** table will not be included in each subquery. It will be included in the parent query, and an outer join will link the Ee table to each subquery.

To create this query, the following steps are taken, starting with the query just shown:

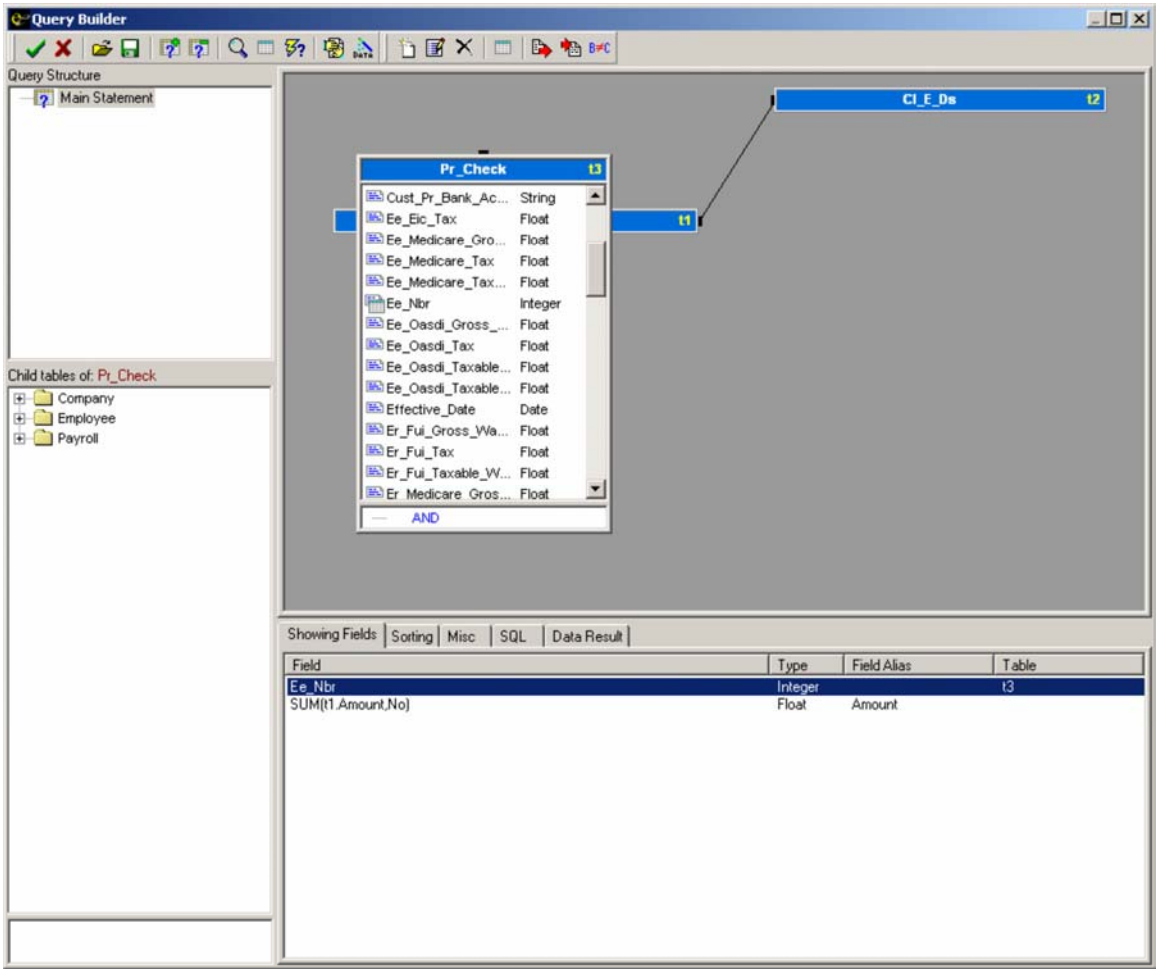
- Right-click on the **Ee** table in the Work Area and select Remove Table to remove the table from the query.



- Double-click on the **Ee_Nbr** field inside the **Pr_Check** table in the Work Area to add it to the Showing Fields Tab.

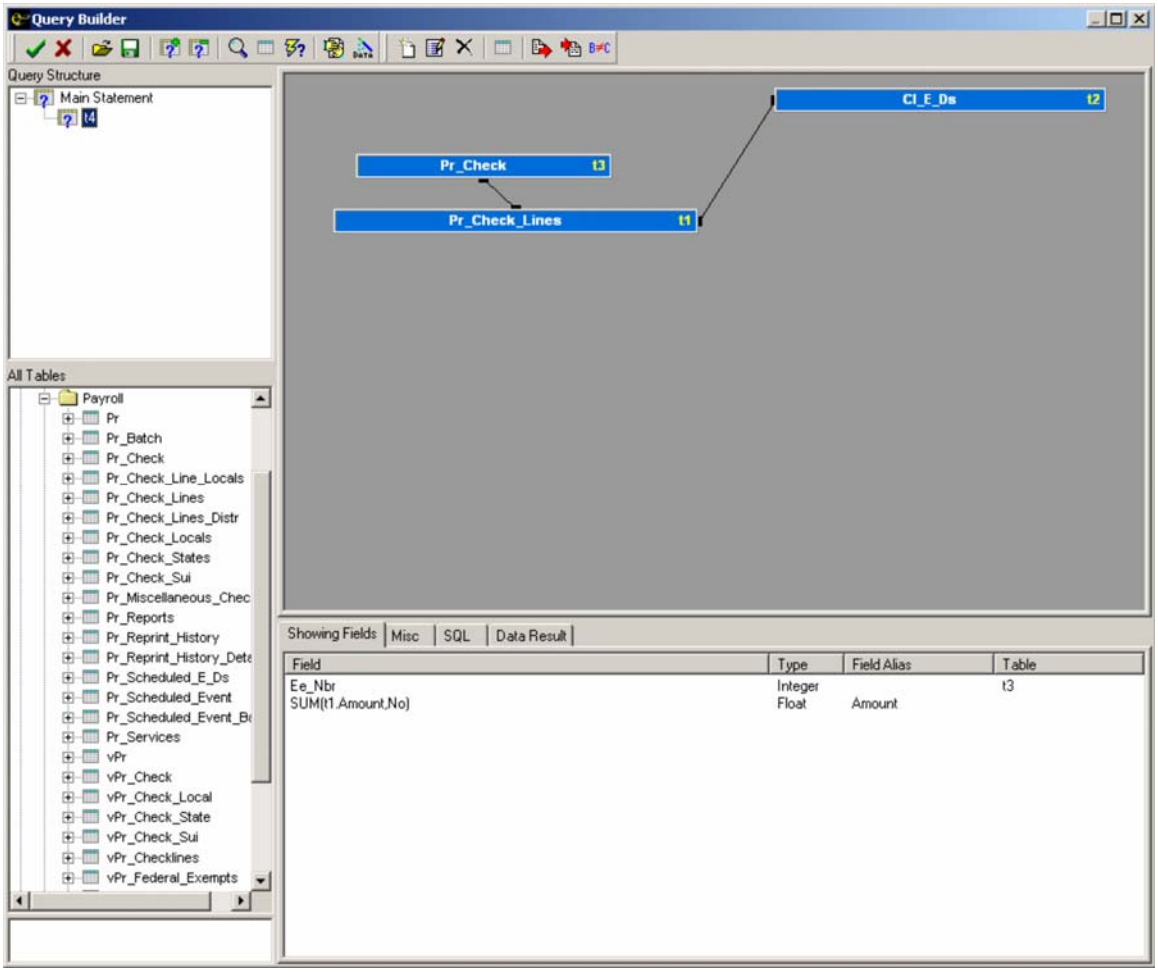


- Drag and drop the **Ee_Nbr** field in the Showing Fields Tab on top of the Amount field directly above it to rearrange the fields in the Showing Fields Tab so the **Ee_Nbr** is first.

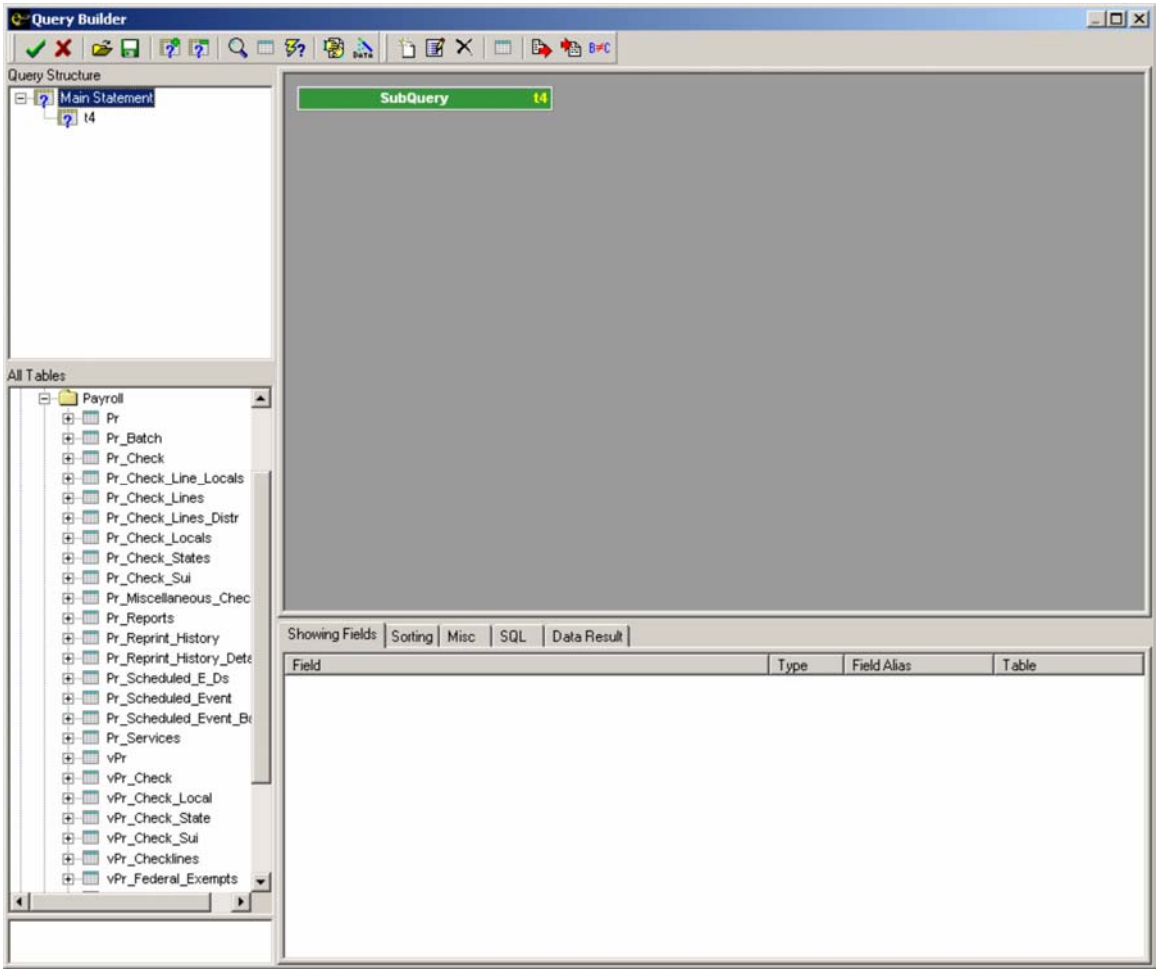


# Evolution Query Builder

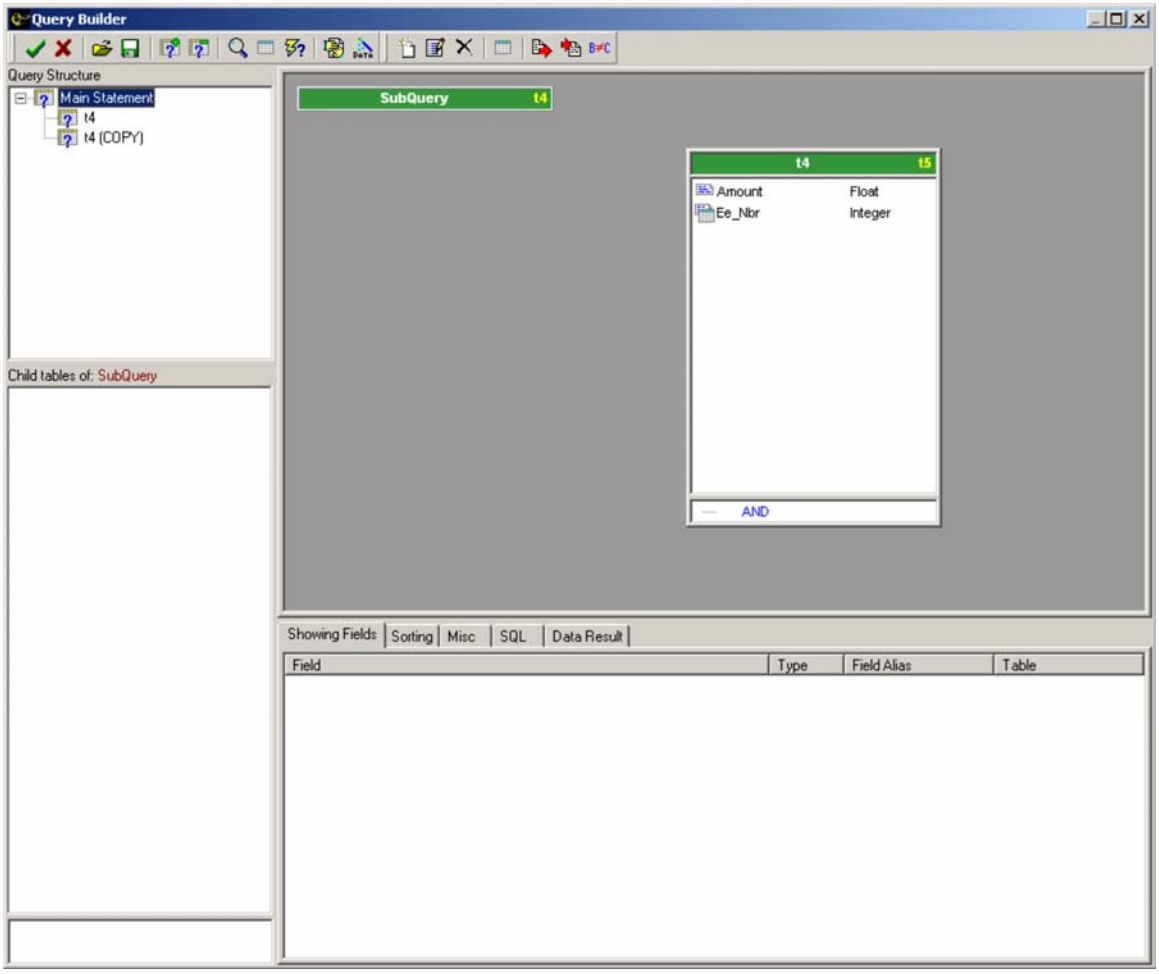
- Right-click on the Main Statement in the Query Structure Area and select Add → Parent Query.



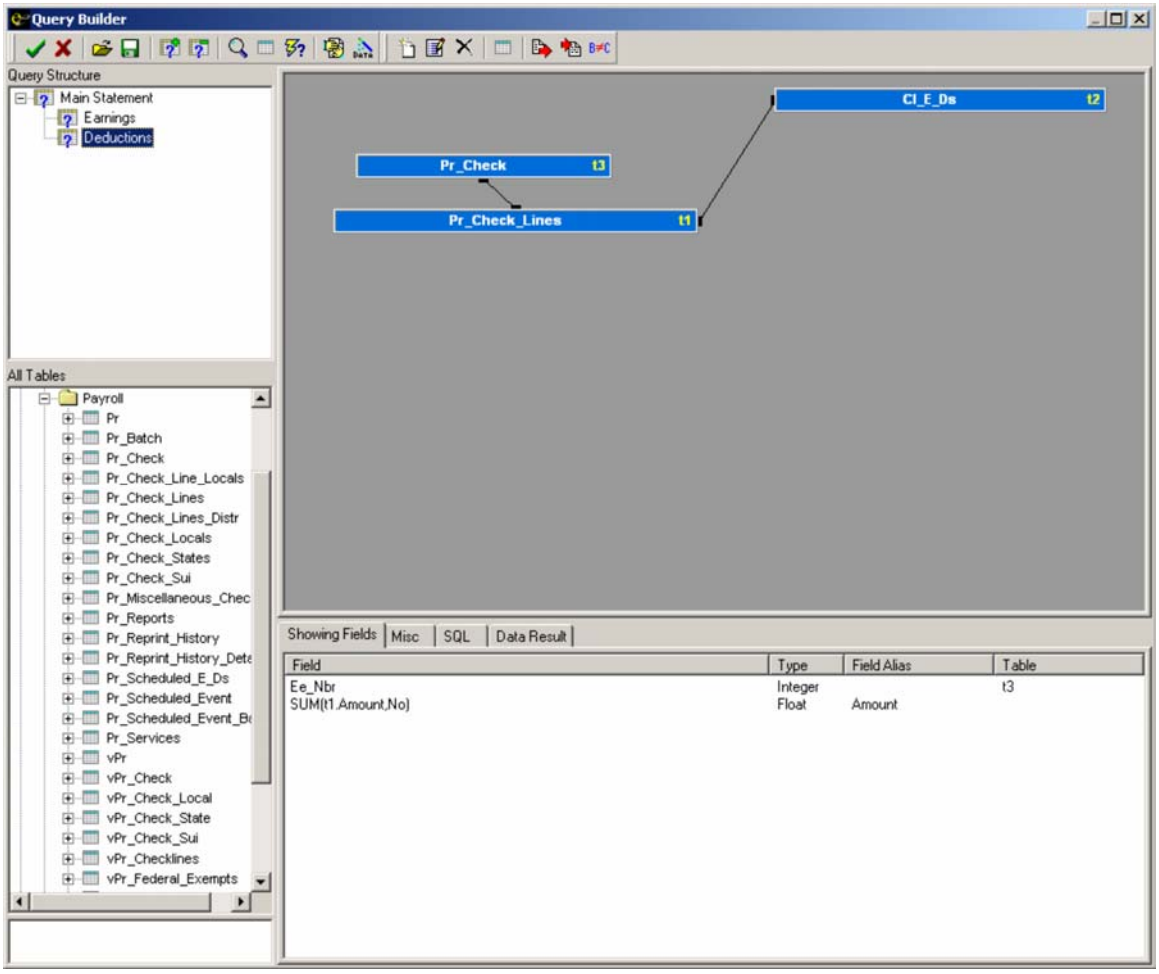
- Select the Main Statement in the Query Structure area. Right-click on the green subquery in the Work Area and select Copy.



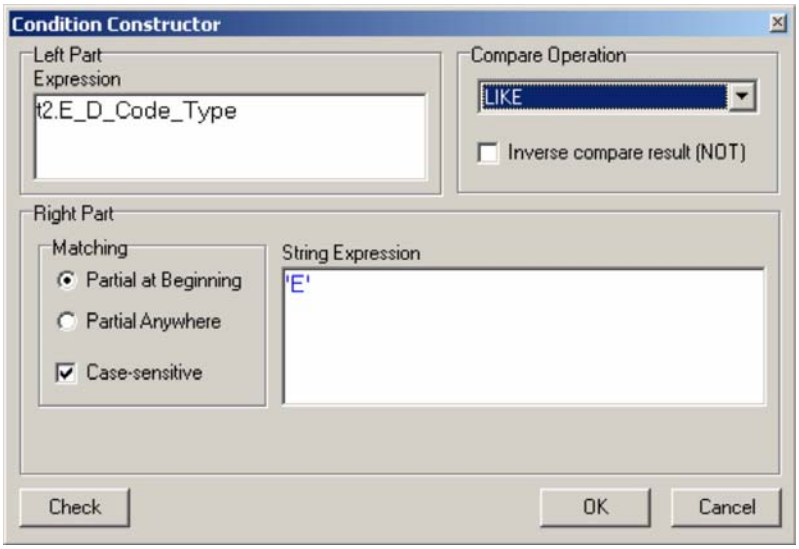
- Right-click in an empty part of the Work Area and select Paste Table to add another copy of the subquery to the Main Statement.



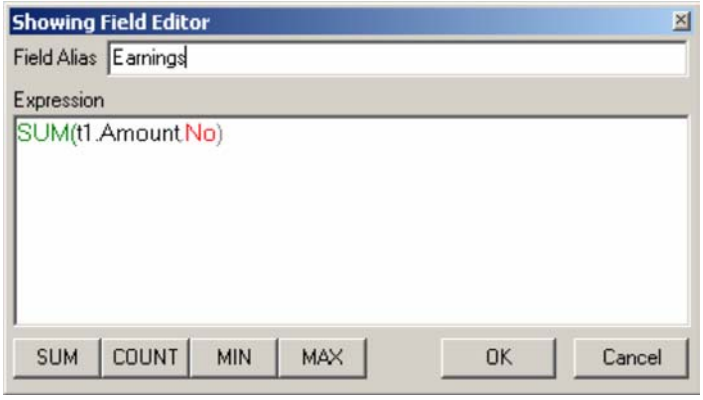
- Right-click on each subquery in the Query Structure Area inside the Main Statement and select Edit Query Description to rename each subquery. Name one “Earnings” and the other “Deductions”.



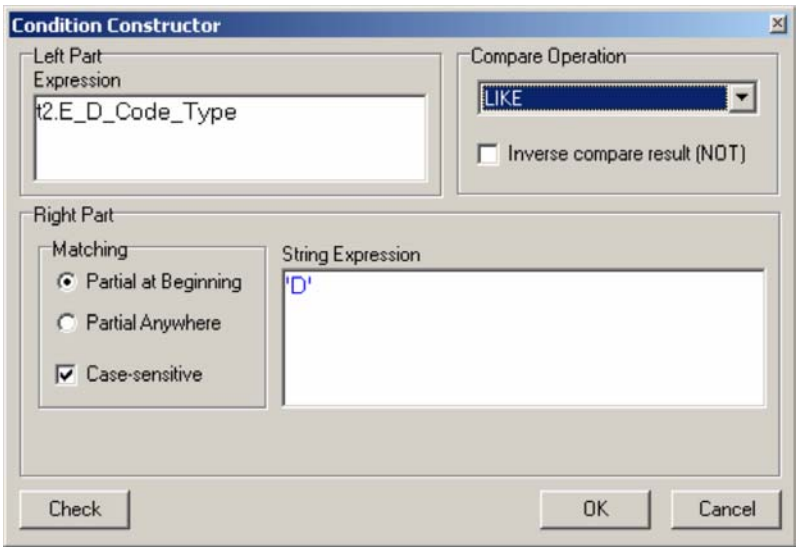

- Select the **Earnings** subquery in the Query Structure Area. Open the **CI_E_Ds** table in the Work Area, double-click on the condition and modify it to look like the one below and click OK:



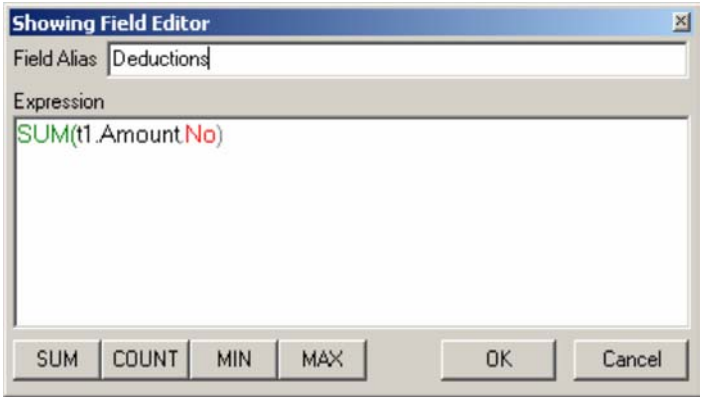
- Double-click on the **Amount** field in the Showing Fields Tab. Change the Alias of the field to “Earnings” and click OK:



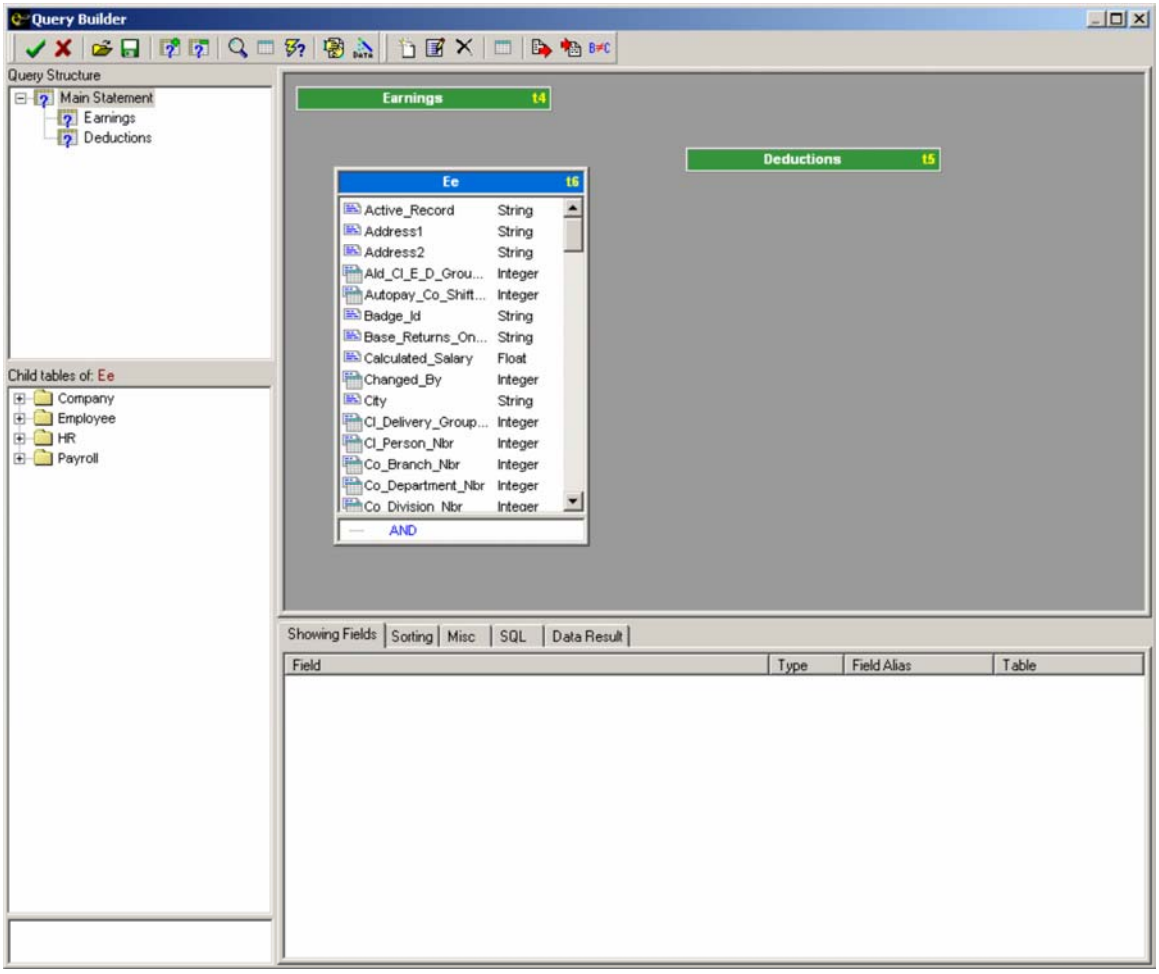

- Select the **Deductions** subquery in the Query structure Area. Open the **CI_E_Ds** table in the Work Area, double-click on the condition and modify it to look like the one below and click OK:



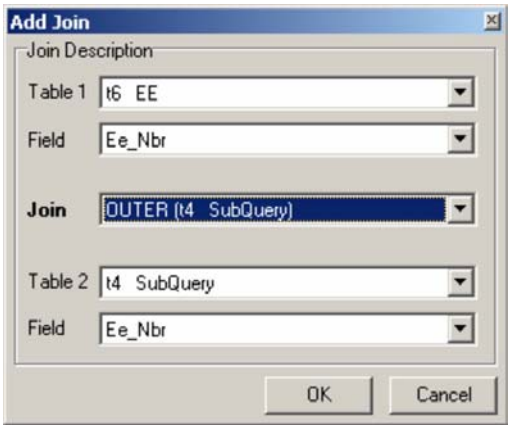

- Double-click on the **Amount** field in the Showing Fields Tab. Change the Alias of the field to “Deductions” and click OK:



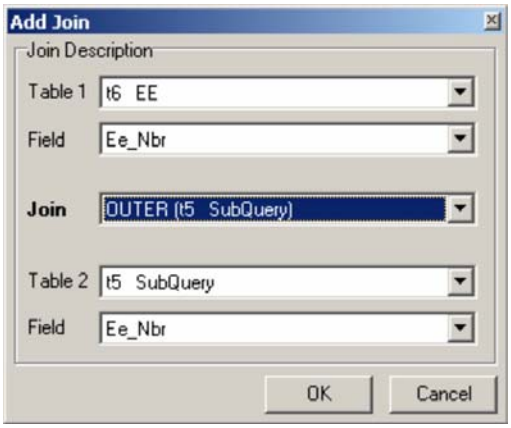
- Select the Main Statement in the Query Structure Area. Drag and drop the **Ee** table from the All Tables Area into the Work Area:



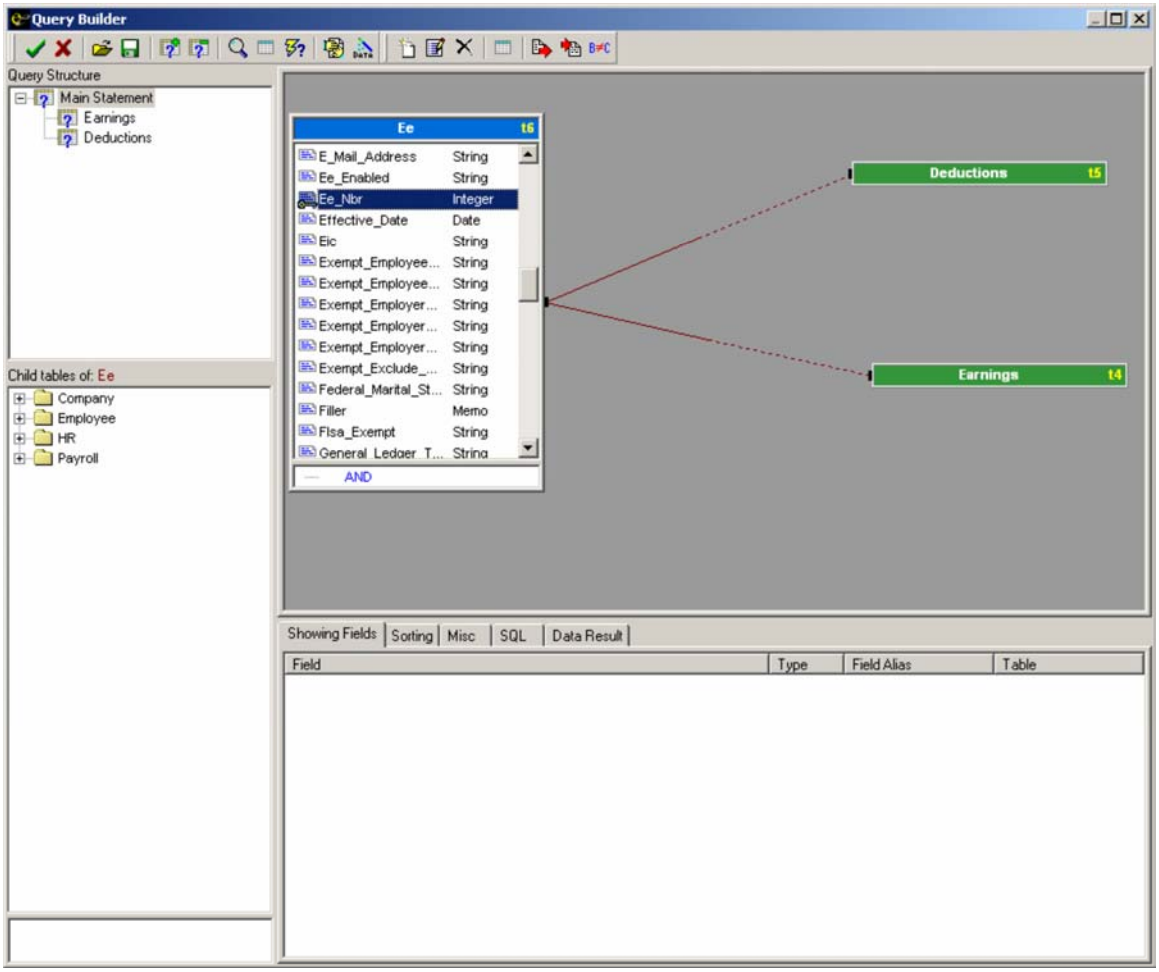

- Drag and drop the **Ee_Nbr** field from the **Ee** table in the Work Area onto the **Earnings** subquery. When the Add Join window appears, create an outer join where the **Earnings** subquery is selected, referenced by the subquery name in the upper right corner of the subquery – **t4** in this example – followed by the word "SubQuery", then click OK:



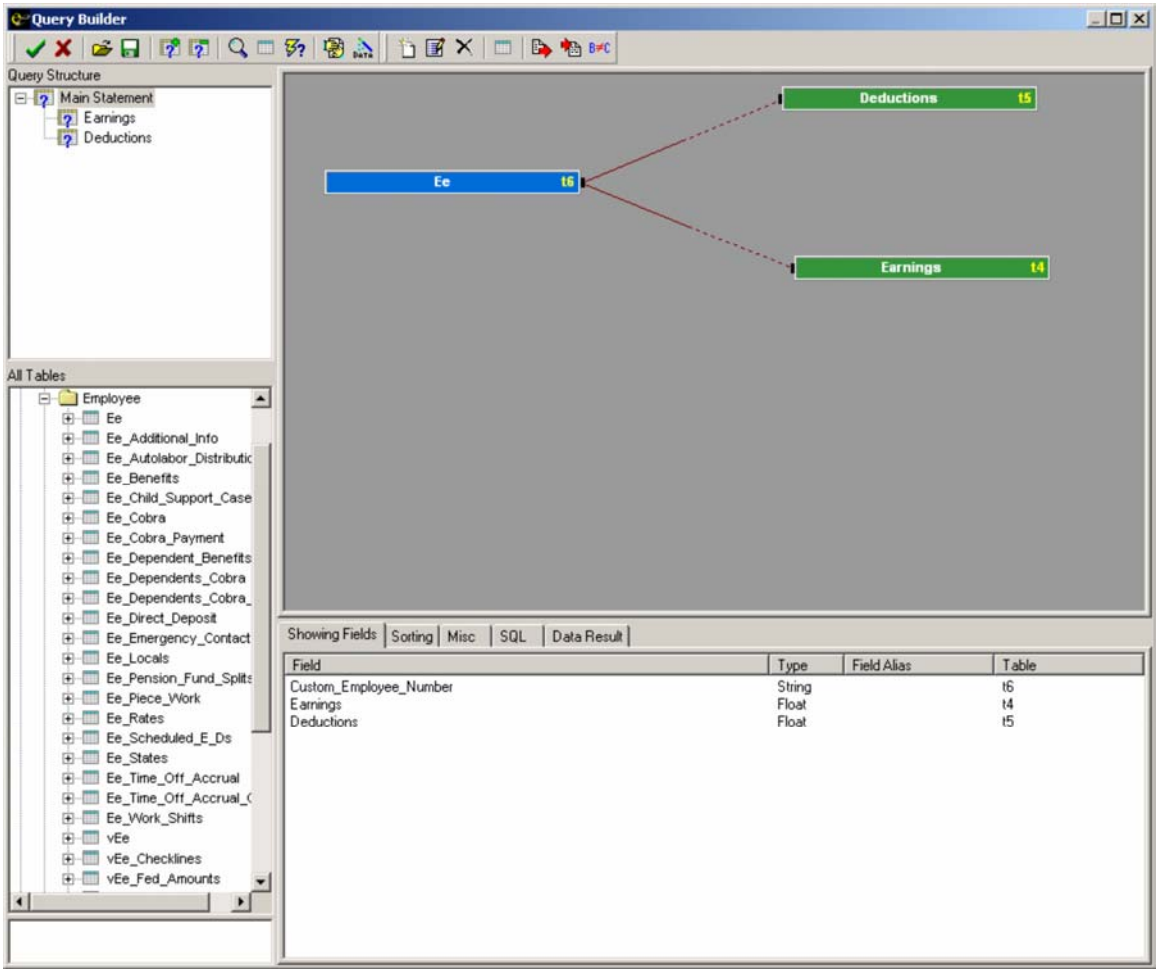
- Drag and drop the **Ee_Nbr** field from the **Ee** table in the Work Area onto the **Deductions** subquery. When the Add Join window appears, create an outer join where the **Deductions** subquery is selected, referenced by the subquery name in the upper right corner of the subquery – **t5** in this example – followed by the word "SubQuery", then click OK:




- The end result should look similar to the screen below. There should be two outer joins, each one linking a subquery to the **Ee** table, joining on the **Ee_Nbr** field in each case:



- Add the **Custom_Employee_Number** field from the **Ee** table, **Earnings** field from the **Earnings** subquery and **Deductions** field from the **Deductions** subquery to the Showing Fields Tab:



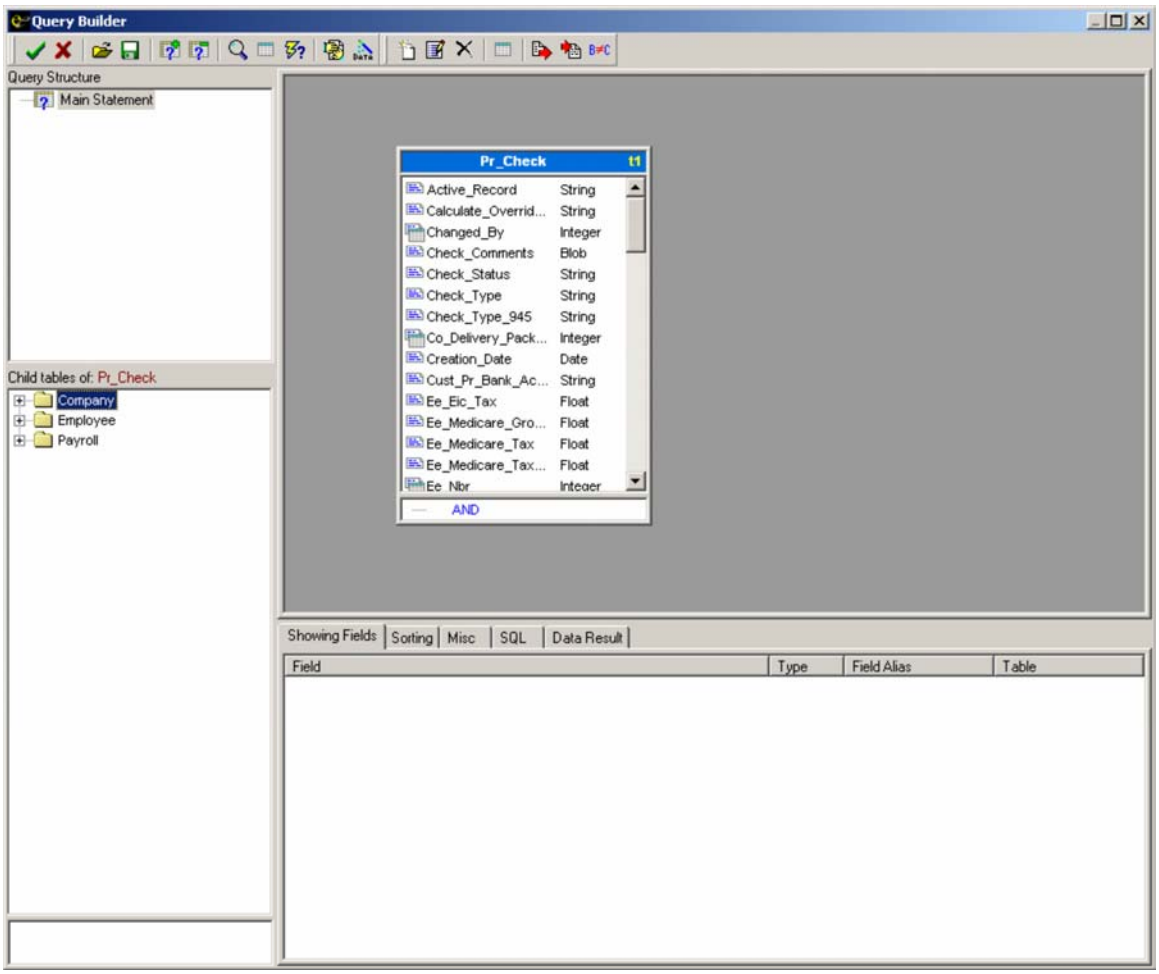
## 1.4.7 Unions

The purpose of a ***union*** is to combine the results of one subquery with the results of another. Each subquery in a union is referred to as a Union Item in Query Builder.

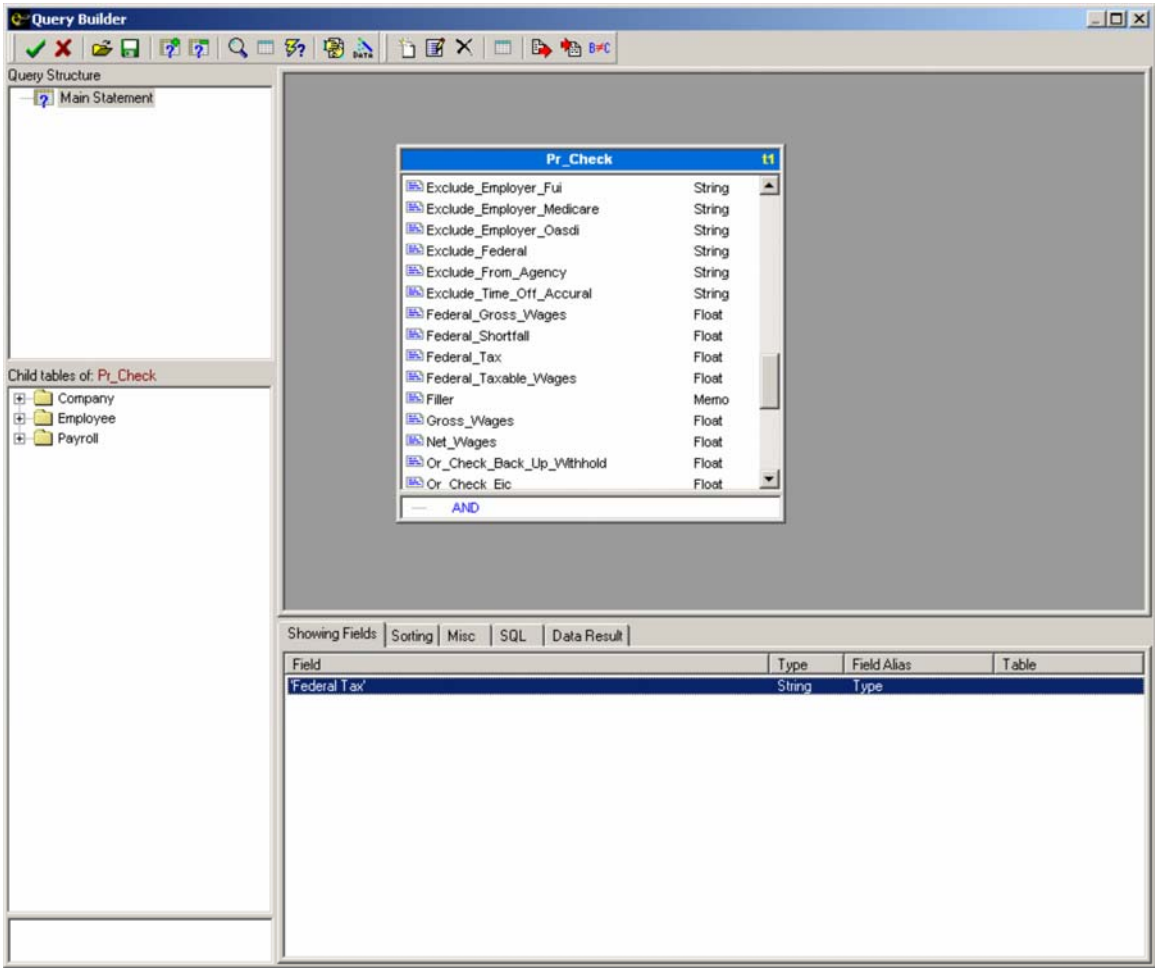
In a union, all corresponding columns in each union item need to have the same data type. So, if the union item #1's first column is of type Integer, then each of the remaining union items must also have a first column of type Integer. The second columns of all union items must be of matching type as well – String for example.

All rows returned by each union item will be shown as a separate row in the main union's results. For example, the Pr_Check table includes all federal tax information. Without a union, it would be difficult to return various federal taxes on the same check on different rows. The union makes this a much simpler task, as explained below.

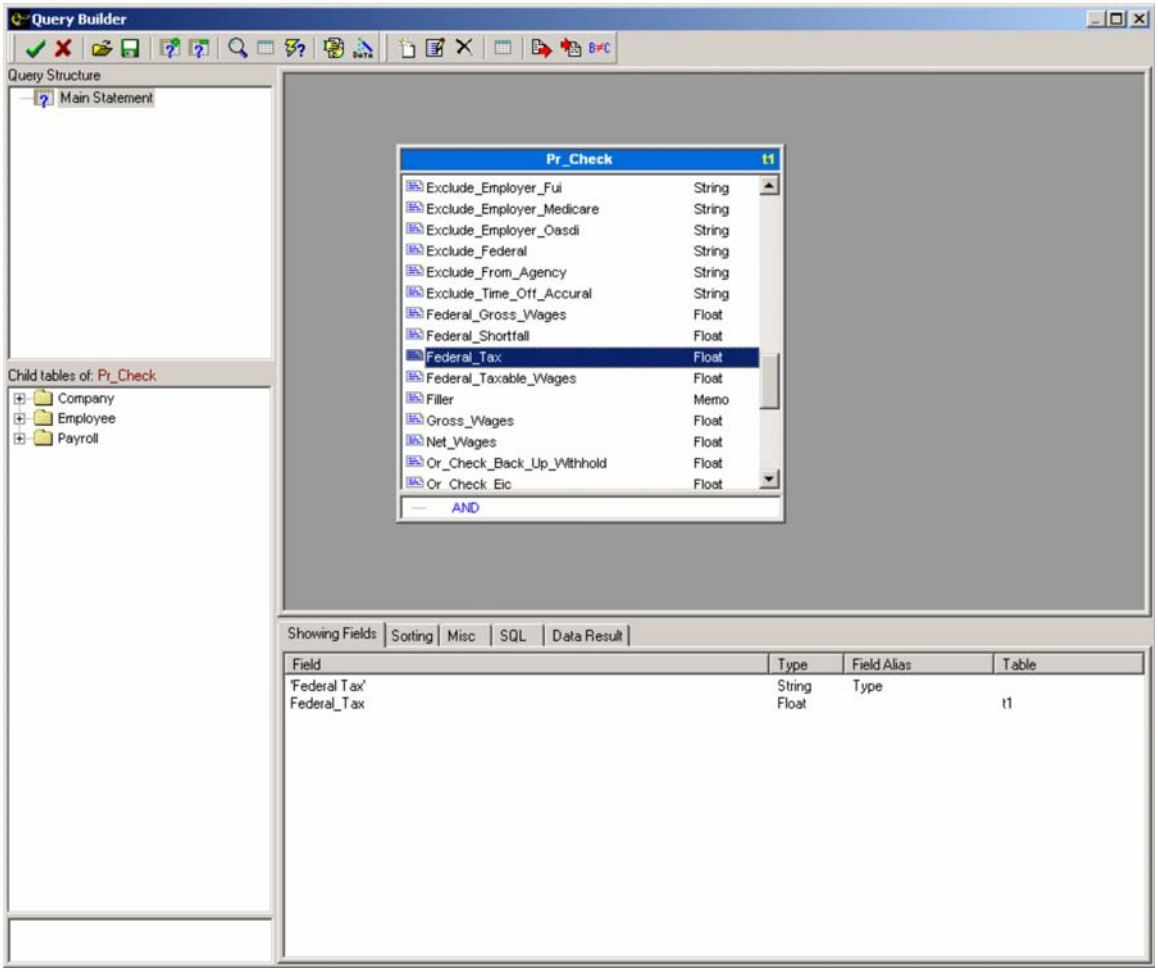
- Starting with an empty Main Statement, select the Main Statement in the Query Structure Area and add the **Pr_Check** table to the Work Area:



- Add a String constant to the Showing Fields Tab. Make the Field Alias “Type”, and the value “Federal Tax”:

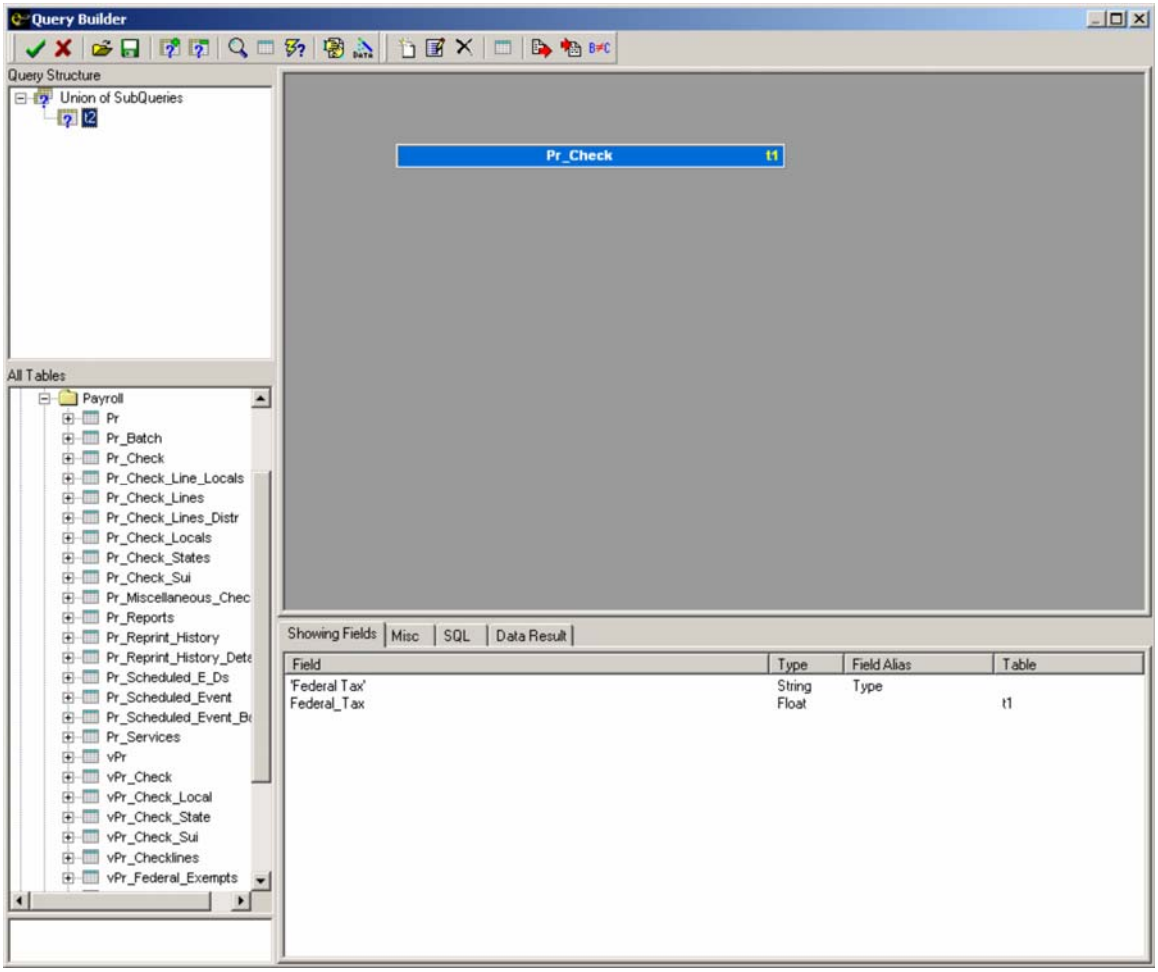


- Add the **Federal_Tax** field from **Pr_Check** to the Showing Fields Tab:

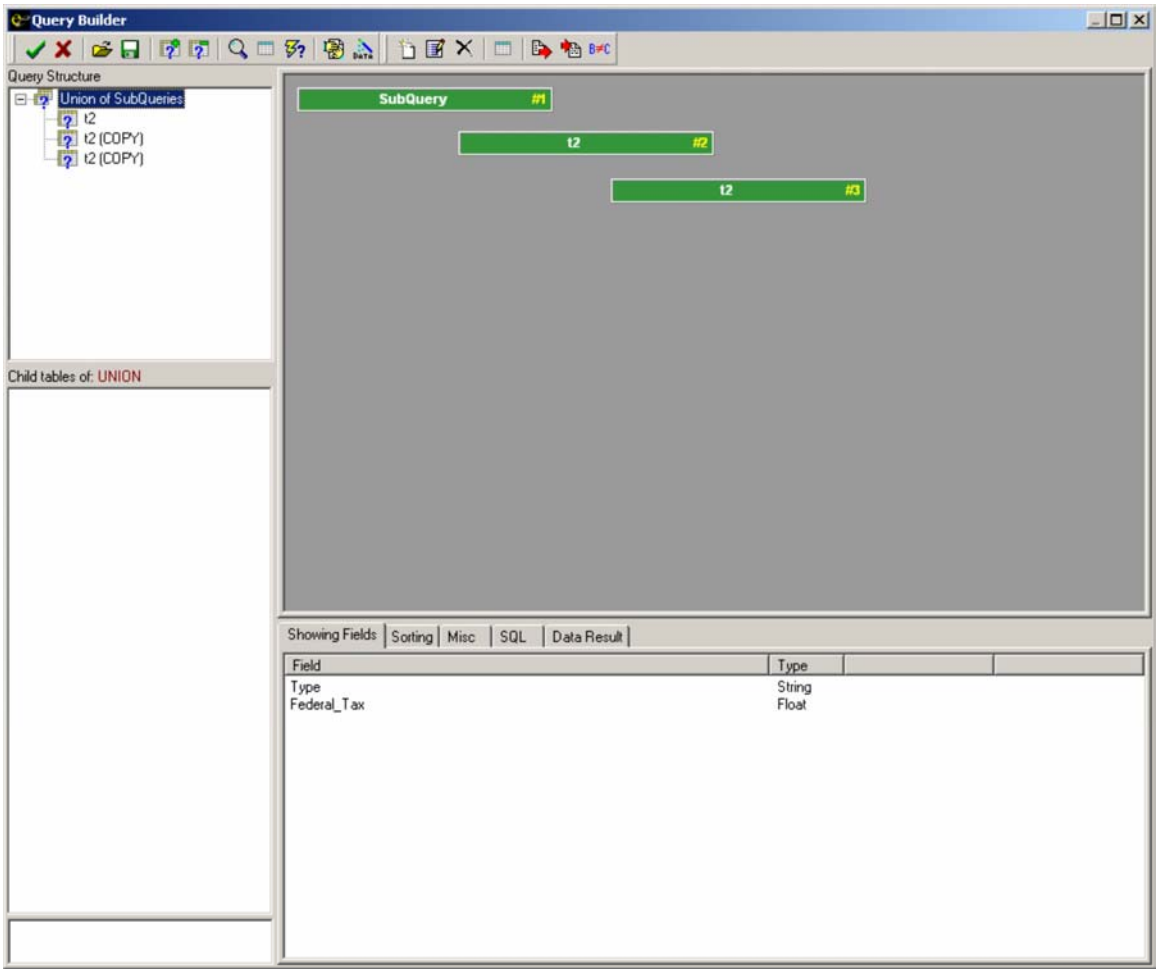


# Evolution Query Builder

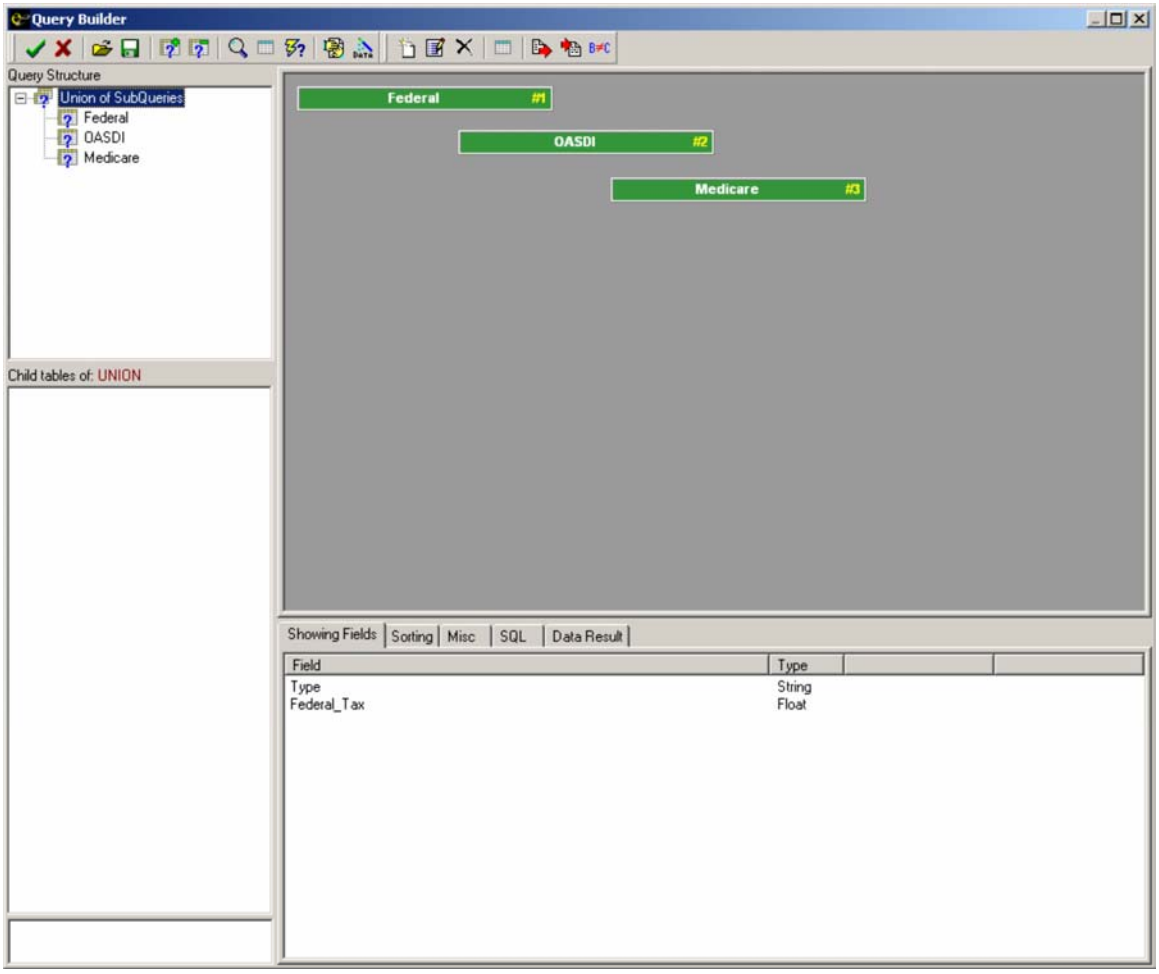
- Right-click on the Main Statement in the Query Structure Area and select Add → Parent Query (UNION):



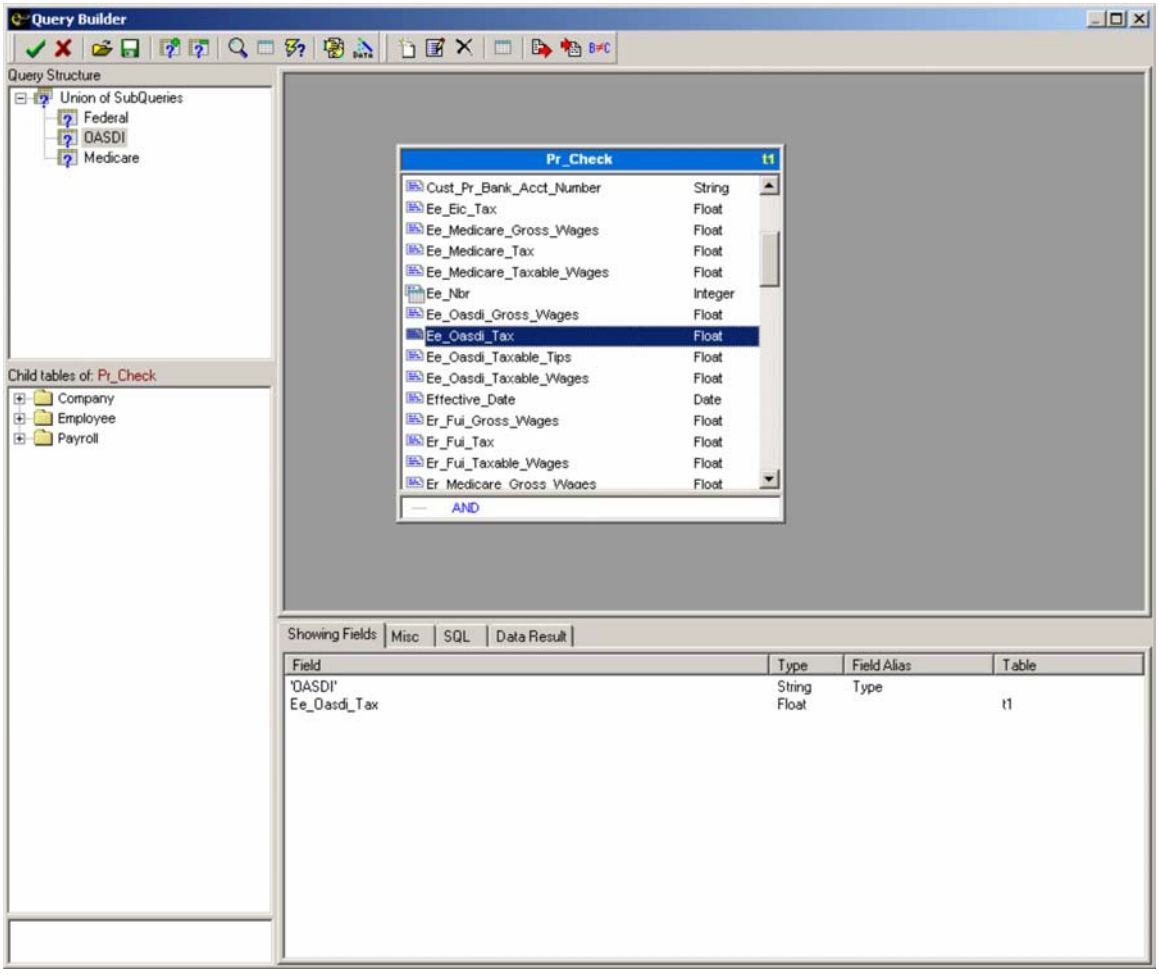
- Select the Union of SubQueries in the Query Structure Area. Right-click on the union item with "#1" in the upper-right corner and copy it as before, pasting a copy into the Work Area. Do this once more so that there are three union items in the Work Area with the Union of SubQueries selected in the Query Structure Area:



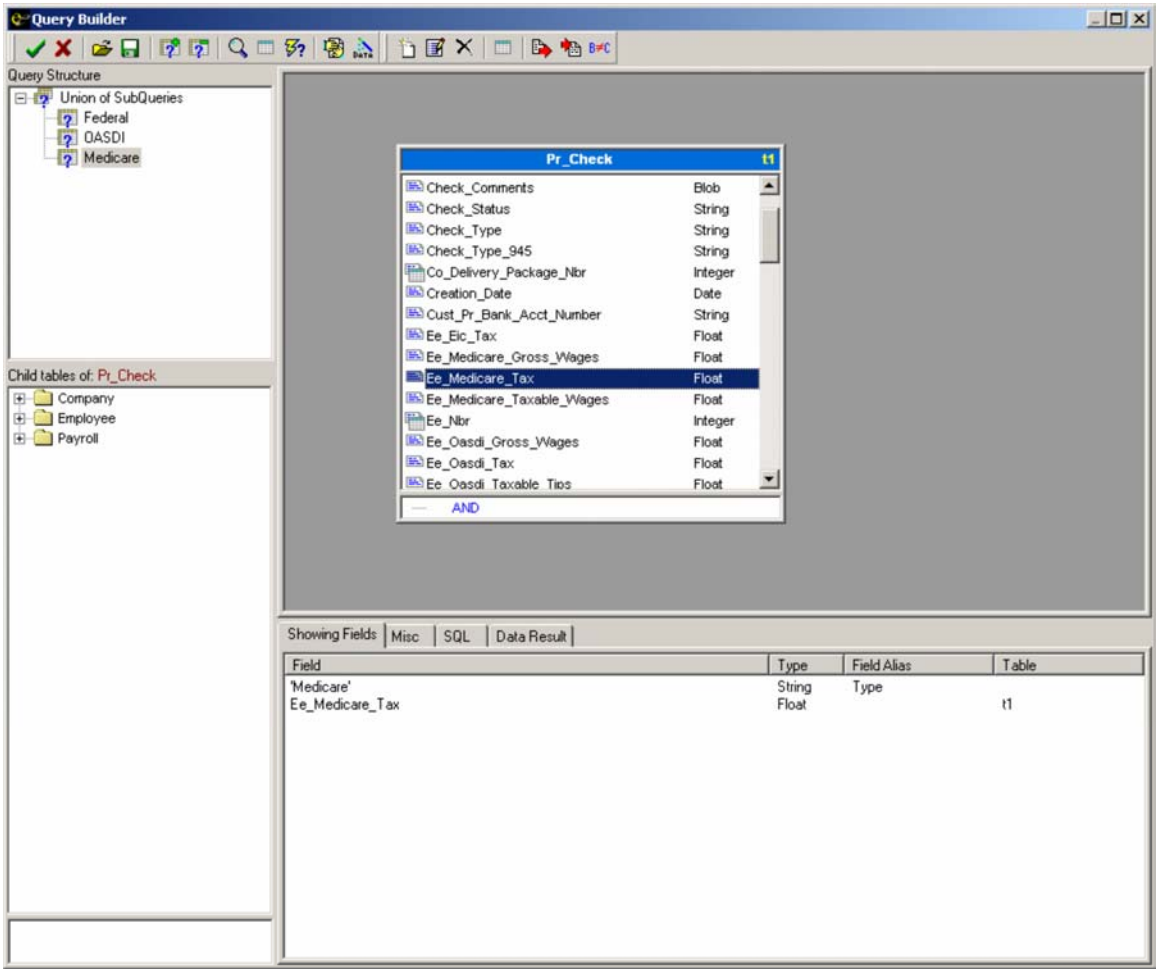
- Right-click on each union item in the Query Structure Area and rename them to “Federal”, “OASDI” and “Medicare”:



- Select **OASDI** in the Query Structure Area. Double-click the **Type** constant and change the value to “OASDI”. Remove **Federal_Tax** from the Showing Fields tab and add the **Ee_Oasdi_Tax** field:



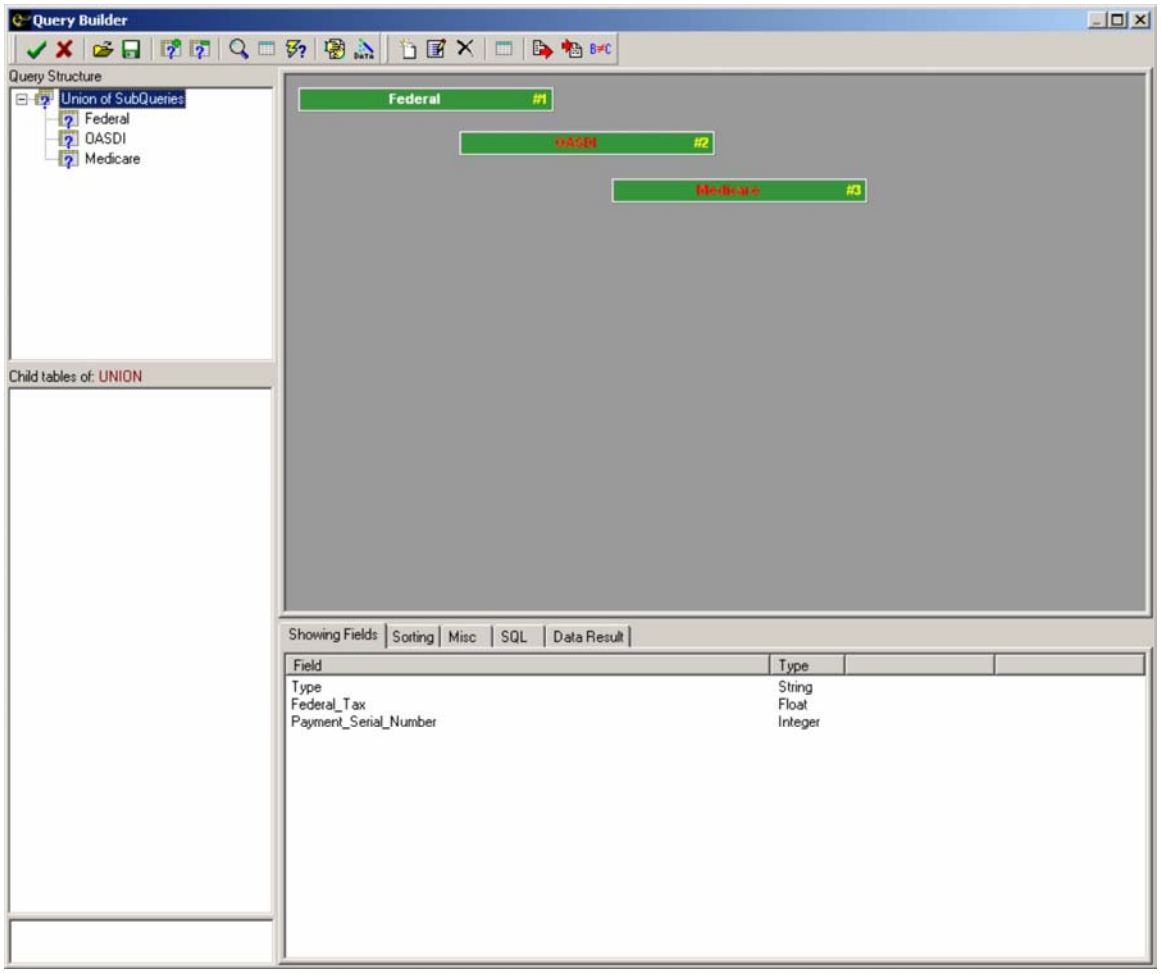
- Select **Medicare** in the Query Structure Area. Double-click the **Type** constant and change the value to "Medicare". Remove **Federal_Tax** from the Showing Fields tab and add the **Ee_Medicare_Tax** field:



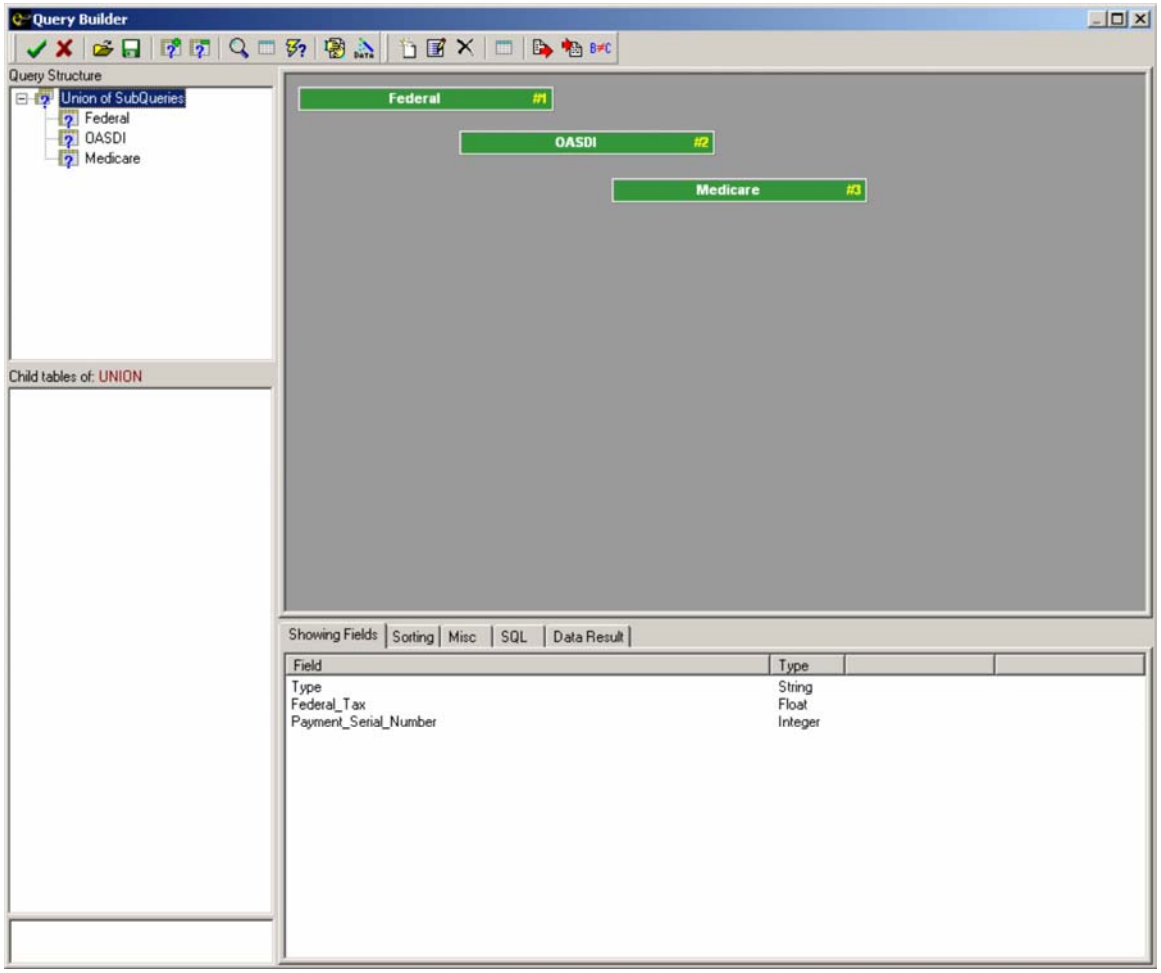

The union just created will show one row each for federal tax, OASDI and Medicare. In order to see what check each tax belongs to, the **Payment_Serial_Number** field needs to be included in the list of fields on the union's Showing Fields Tab. However, fields cannot be added to the Showing Fields Tab of a union by double-clicking or dragging and dropping fields from tables in the Work Area to the Showing Fields Tab. In a union, they are added by adding the field to the Showing Fields Tab of each individual union item.

As mentioned earlier, it is a requirement of the union that all union items in the same union are structurally identical, with the same number of columns, and with each column in the same position in each union item having the same data type. Query Builder will tell the user if there is a problem with a union item while that union is selected in the Query Structure Area. The name of the problematic union item will be red in the green bar at the top of that union item.

**Payment_Serial_Number** needs to be added to the union by adding this field to the Showing Fields Tab in each union item in the same place – in this case the third column. In the union shown below, the second two union items have red names. The problem is that **Payment_Serial_Number** has been added to the **Federal** union item, but not the **OASDI** or **Medicare** union items:



This problem is resolved by adding the **Payment_Serial_Number** to the Showing Fields Tab for the **OASDI** and **Medicare** union items. Once that is done, all union item names should turn white:



### 1.4.8 Other Useful Information and Resources

Sample Queries can be provided upon request. Email to info@paycoinc.com