What is Null and Why is it Important for Crystal Reports

The concept of "Null" is frequently misunderstood by people who are new to working with databases. Simply put, Null means "unassigned" or "no value". In most types of databases (dBase and Paradox are exceptions that I know of, there are probably others…), fields that have not had a value put in them return Null when included in a select query.

Null is not the same as an empty string. It is not the same as 0 in a numeric field. It is NOT the same as False.

Most types of databases provide a means for preventing Nulls – when defining a table the fields can be specified as "Not Null" which means that the field must *always* have a value. The application that adds or updates data in the database is then responsible for making sure that the data is provided for those fields. However, not all databases or applications are set up this way. Also, when you use "outer" joins to join the tables in your report, there will be Nulls when the data in one table doesn't have a match in the other table.

Crystal and Null Handling

In Crystal formulas, there are two options for handling Null Values. In the Formula Editor there is a drop-down where the user can select either "Exception for Nulls" or "Default Value for Null". The Default Value option will automatically substitute a default value – blank string for strings, 0 for numbers, etc. – when a value in the database is null. When Crystal is installed, the default is set to "Exception for Nulls". To change the default, go to the File menu, select "Options" and go to the "Formula Editor" tab. Change the value for "Null Treatment". My preference is to leave the default behavior and explicitly handle Nulls. All of the examples below assume that the Formula Editor is set to "Exception for Nulls".

Null Values and Comparisons

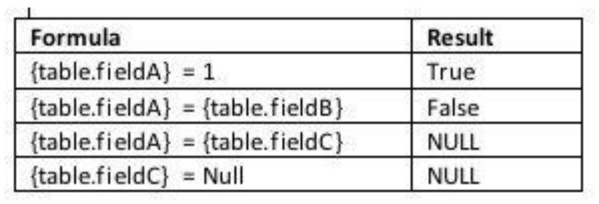
When a field with a Null value is used in a comparison (>, <, =, etc.) the result is *always* Null.

Assuming:

{table.fieldA} = 1

{table.fieldB} = 2

{table.fieldC} is Null



Formula	Result
{table.fieldA} = 1	True
{table.fieldA} = {table.fieldB}	False
{table.fieldA} = {table.fieldC}	NULL
{table.fieldC} = Null	NULL

ANY comparison where one or more values are Null will result in Null. Because Null means "unassigned", it never equals anything, even Null!

Because Null is not the same False, this can cause some unexpected results in If statements. For example:

If {table.fieldA} = {table.fieldC} then

'Yes'

else

'No'

Using the same assumptions as above, this formula will return *no value*. Here's why: the formula returns 'Yes' if {table.fieldA} = {table.fieldC} is true and 'No' if it is false. But in this case the result is Null, which is neither true nor false, so there is no return value. The formula can be re-written as follows to return 'No' when {table.fieldC} is Null:

If not IsNull({table.fieldC)} and {table.fieldA} = {table.fieldC} then

'Yes'

else

'No'

Note that we have to check for Null *before* checking for equality. Logical expressions are evaluated from left to right. When you have an "and" statement, if the first part is not True, evaluation stops. So, if we didn't check for Null first, the check for Null would never be processed.

<u>Null Values and Concatenation</u>

When joining fields together to create a longer string, if any of the fields is Null, there were will no result. For example, if you want to build an employee's name with the middle initial from the fields First_Name, Middle_Name, and Last_Name, it would seem logical to create a formula like this:

{employee.Last_Name} + ', ' + {employee.First_Name} + ' ' + left({employee.Middle_Name}, 1)

However, if the {employee.Middle_Name} field is Null, there will be no result from the formula and the name will be blank on the report. To get this to work correctly, this is the formula that is needed:

If IsNull({employee.Middle_Name}) then

{employee.Last_Name} + ', ' + {employee.First_Name}

else

{employee.Last_Name} + ',' + {employee.First_Name} + ' ' + left({employee.Middle_Name}, 1)

This can cause some very long, convoluted If statements when multiple fields might be Null. A possible solution is to create a separate formula for each field that might be Null that looks like this:

If IsNull({employee.Middle_Name} then '' else ' ' + {employee.Middle_Name}

When Nulls are handled this way, the final concatenation of the string becomes a simple matter of adding the formulas together like this:

{@LastName} + {@FirstName} + {@MiddleName}

Null Values and Math

With the exception of division, Nulls in mathematical formulas react the same way as in string concatenation – no result is returned. However, with division, if the Null value is in the denominator an error will occur and the report will not finish processing. Since a best practice with division formulas is to make sure the denominator value is not 0 before trying to divide, this is what division formula that handles Nulls should look like:

If not IsNull({table.denominator}) and {table.denominator} <> 0 then

{table.numerator}/{table.denominator}

else

0

Nulls, Outer Joins, and Selection Criteria

This concept is easiest to explain using a concrete example. Suppose you have this requirement: Show a list of clients and the dollar value of what they purchased last month. There are a couple of ways you can display this data.

If you just need to show the clients who actually purchased something last month, this is a pretty straight forward task:

1. Add the client table and the sales table to the report and join them together on client_id.
2. Group by client and sum the sales amounts.
3. Set the select expert to only select the data from the sales table where the sales date is last month.

However, if you need to show a list of ALL clients, whether or not they purchased anything last month and the amount they purchased, if any, things get a little more complex.

1. Add the client table and the sales table to the report and draw the join *from* client *to* sales on client_id.
2. Right-click on the join, and select “Link Options”. Change the join type from Inner Join to Left Outer join. This outer join means that all of the records in the client table will be included even if there is no corresponding record in the sales table.
3. Group by client and sum the sales amounts.
4. Here’s where things get a little tricky. We want to add up only those sales from last month. However, if a filter is set up to only include sales data from last month, the clients who had no sales last month won’t appear on the report because they have no sales dates in the date range. The trick for getting this to work is to edit the selection criteria so that clients with no sales during the month also show up. To do this:

    - Open the Select Expert.
    - If there are no selection criteria already set up, select any field to get to the Expert.
    - Click on the “Show Formula” button on the bottom right of the form and then on “Formula Editor”.
    - Enter something like the following:

```
IsNull({sales.client_id}) or {sales.sale_date} in LastFullMonth
```

By looking for Nulls in the sales table, the clients with no data in the date range will still be included in the report.

Summary

When you’re not used to including Null handling in your reports, it can be difficult to diagnose issues that may appear due to Null value. However, planning for the possibility of Null values in a report’s data is an important part of making sure that the report displays the correct results and working with Nulls is not difficult once you know to look for them.

Dell Stinnett-Christy, Senior Business Intelligence Consultant
Decision First Technologies
Dell.Stinnett@decisionfirst.com

Dell Stinnett-Christy has been working with Crystal Reports for 17 years and SAP BusinessObjects Enterprise for 7. As a Senior Consultant for DecisionFirst Technologies, she does BO system administration, upgrades, universe design and report design in both Web Intelligence and Crystal Reports. She is also an expert with the Java and .NET SDKs for both Crystal Reports and SAP BusinessObjects Enterprise.