Form Module Modification

MLR Database Project Prototype

Larry Bednar

June 14, 2004

Purpose

This document describes the modifications required for form modules to work correctly within the prototype mlr_db_client.mdb MS-Access database.

[LB – I expect I've gotten *some* details incorrect. However, I do believe it is correct for about 95% of what is provided.]

Copyright/License

Edit Forms

A mechanism used to enable/disable navigation buttons related to movement back to a related "record selection display" form has not yet been implemented in all edit forms. This disables the "selected records" button if the edit form has been opened directly, and enables that button if the edit form was opened from the "record selection display" form. Refined versions of this mechanism are yet to be copied into the MLR database from the MLR database used for development/refinement.

1. Alter module-level constants appropriately.
2. Alter "cboPrimaryYN_AfterUpdate procedure in forms that present "primary_yn" columns for user modification.

   The procedure must set the label used for the subject area in the message box displayed for the user.

   In these forms, it is typical that only one record of a set is allowed to have a 'yes' setting for the primary_yn column. The cboPrimaryYN_AfterUpdate procedure is used to trigger the procedure that performs this check and resets other rows to a 'no' value if the user has set the current row to a value of 'yes'.
3. [LB - ? Changes related to implementation of alterations to **cmdNew_Click** and **EditSelected** procedures? **CallingForm** Property Let procedure? **EnableNavControls** procedure?]

Donation Edit Form

Note that the mechanism used to enable/disable controls according to donation type is kind of rough – use of id values means that change in donation type codes won't necessarily result in behavior that is appropriate. Perhaps users should not be allowed to change "donation type" code definitions?

Record Selection Form

The most involving an difficult part of modifying a form module to use SelCtl class modules is the careful construction of the SQL fragments defining the use of control values within the constructed base SQL SELECT statement, and within the correlated subqueries used for related tables.

1. Decide which attributes in the database are to be used for record selection.
2. Place controls on the form that are appropriate for the intended record selection purposes.
3. Decide which SelCtl objects should be defined to make appropriate use of the controls now placed on the form.

   SelCtlCbo – Designed to make use of values entered in a combo box or

   SelCtlGrp – Designed for construction of filters against a "group membership" table, by comparing row attributes defining a group, membership start date, and membership stop date to entries made by the user in a "group" multi-select list box, and an "effective date" text box.

   SelCtlLst – Designed to use values entered in a multi-select list box.

   SelCtlTxt – Designed to use values in a text box control.

   SelCtlTxtSearch – Designed to use values from a text box control into which a text search pattern is entered.

Form Module Declaration Section

4. Define object variables in the declaration section of the form module for all SelCtl objects that will be needed.

   These should be defined as private, module-scope variables for the form module.

5. Decide which tables will be used as part of the base SELECT statement being constructed by the form, and which tables will be used in correlated subqueries.

   Tables used in subqueries typically will *not* be defined as part the base SELECT statement. (In some cases they *might* be defined both in the base SELECT statement and in a correlated subquery, but this would be rare.) The base SELECT typically will include only the main topic table and any associated tables that provide single related rows. "Associations" tables that are used to resolve many-to-many joins between tables typically will not be part of the base SELECT statement. The base SELECT statement needs to return one row for each row of the primary topic table that is matched by the selection conditions entered by the user.

InitSelCtls Procedure

This procedure creates new SelCtl objects corresponding to the module-level SelCtl object variables defined in the declaration section of the form module. Each SelCtl class has a defined **SetProperties** method that should be used. The names of the parameters the developer is prompted with are a substantial help in understanding what type of parameters must be provided.

A typical sequence of creation statements will be:

```
Rem Set SelCtl properties
Set msclContactMethod = New SelCtlList
msclContactMethod.SetProperties Me.lstContactMethodDefnId, 1, sclIn, sclNbr, _
                                "contact_method_defn_id", "c"
```

Creation of a SelCtlGrp object is an exception to this general pattern, since this object requires the specification of a subquery base, a multi-select list box and and a text box to be associated with it.

```
Rem Set SelCtl properties
Set mscgGroup = New SelCtlGrp
strSubqryBase = "SELECT gm.group_mbr_period_id " & _
                "FROM group_mbr_period AS gm, party AS py " & _
                "WHERE c.party_id=py.party_id " & _
                "AND py.party_id=gm.party_id"
mscgGroup.SetProperties Me.lstGroupId, Me.txtGroupMbrPeriodEffectiveDate,
                        scgExists, _
                         scgAnyGrp, scgMbr, strSubqryBase, "date_end",
                         "group_defn_id", _
                       "date_start", "gm"
```

Many of the parameters specified are enumerated constants that are defined to make it easier to remember the meaning/use of the constants. Typically, these indicate how the values entered in associated form controls are to be used in the constructed SQL expressions.

NOTE: In each case, the developer must enter the column name and the table alias of the database column that will be targeted by the SQL expression constructed by the select control object. The table aliases provided here must correlate correctly to either:

- the table aliases specified in the **strcTableList** constant in the **InitSelCtlObjList** procedure (if they are used as part of the base SELECT statement), or
- the table aliases specified to correspond to "related tables" added to the form's SelCtlObjList object in the **InitRelatedTables** procedure.

Naturally, the table aliases must correspond correctly for the overall SQL SELECT statement to work properly.

These are several of the table aliases used as "standards" so far:

NOTE: It might be good to spend some time strengthening this system of abbreviations. I'm not comfortable that the whole system is really quite polished and systematic enough yet. There are still some additional tables that need to be brought into the system, also.

ca	case
cac	case_counselor
cas	case_status
co	counselor
ct	contact
Cm	contact_method_defn
Cs	contact_subject
Csd	contact_subject_defn
Cy	county_defn
Del	deliv
Dn	donation
Dt	donation_type_defn
Fa	focus_area
Fd	fund

Gm	group_mbr_period
Gr	grant
Gt	group_type
Gtd	group_type_defn
L	loctn
Lt	loctn_type_defn
Nm	neap_milestone
Np	nepa_project
Npc	neap_project_contact
o	outreach
oe	outreach_exp
of	outreach_focus
op	outreach_party
opr	outreach_party_resp
orp	outreach_resp
osm	outreach_staff_mbr
pt	participant
Py	party
Pc	party_category
Pcr	party_contact_role
Pl	pledge
Pm	party_mbr
Pmr	party_mbr_role_defn
Pt	party_type_defn
Pn	person
Ph	phone
Pht	phone_type_defn
P	pub
Pi	pub_issue
R	region
Sm	staff_mbr
T	team
Tm	team_mbr
Ts	timber_sale

Tsc	timber_sale_contact
Tso	timber_sale_outreach
Va	vol_act
Vad	vol_act_defn
Vafa	vol_act_focus_area
Vap	vol_act_pref

6. Enter an appropriate set of statements for creation of each required SelCtl object.

InitSelCtlObjList Procedure

A single SelCtlObjList object is associated with each record selection form. The SelCtlObjList object manages the sometimes complex interactions and joint uses of the full set of SelCtl objects defined for use in the form.

7. Specify the table description part of the base SELECT statement as the value of the **strcTableList** constant in the **InitSelCtlObjList** procedure.

   It is probably wise to define table aliases that are standards used throughout the applications. At a minimum, this makes it easy to understand queries written throughout the application, since there is no need to reacquaint yourself with new abbreviations in each form module.

8. Specify the join description part of the base SELECT statement as the value of the **strcJoinDesc** constant in the **InitSelCtlObjList** procedure.

   Any column references should use the table alias defined for the table in the **strcTableList** constant. The "WHERE exp1 AND exp2…" style of equijoin should be used. Outer joins aren't supported by the SelCtl class modules.

9. Specify the row description part of the base SELECT statement as the value of the **strcRowDesc** constant in the **InitSelCtlObjList** procedure.

   This doesn't have to be any more than a single column, for instance the primary key of the "primary topic" table. Any column references should use the table alias defined for the table in the **strcTableList** constant.

10. Using the **AddSelCtl** method of the SelCtlObjList object, add each SelCtl object to the SelCtlObjList object defined for use by this form.

InitRelatedTables Procedure

This procedure adds information about "related tables" to the SelCtlObjList object created for use by the form. Each "related table" typically represents a table whose attributes will be employed in the overall SELECT statement through the use of a correlated subquery. "Association" tables required to resolve many-to-many joins will typically be used through a "related table" definition entered into the SelCtlObjList.

NOTE: The table alias entered for the related table must correspond exactly to the table alias specified for the SelCtl object that provides the values to be used in the subquery. The SelCtlObjList determines when to construct a subquery by looking for matches of table aliases specified to SelCtl objects and table aliases defined as part of a "related table" definition.

11. Using the SelCtlObjList object's AddRelatedTable method, add a definition of a related table that will be employed in a correlated subquery to the base SELECT statement for the form.

Control GotFocus Procedures

Any form control that is defined as a parameter to a SelCtl object (except those defined as a parameter to a SelCtlGrp object), must have an associated GotFocus event. The GotFocus event is used to trigger SelCtlObjList and SelCtl object methods that store the value of the form control at the time the user enters the control. This allows the user to revert to that original value later.

12. Add a GotFocus event procedure to each form control used in a SelCtl object that is not a SelCtlGrp object.

    The only action required of the GotFocus event is a call to the **SetCrrntCtlVal** procedure.

Control AfterUpdate Procedures (for SelCtl objects *other* than SelCtlGrp objects)

Any form control that is defined as a parameter to a SelCtl object (except those defined as a parameter to a SelCtlGrp object), must have an associated After update event. The AfterUpdate event is used to trigger SelCtlObjList and SelCtl object methods that query the data server for the count of rows matching the user's specifications after the current control update is applied. If no rows match, the user is offered the option of reverting the control to the value present when they first entered the control.

NOTE: Because SelCtlGrp objects have two form controls associated with them, the use of AfterUpdate events will not provide the correct functionality.

13. Add a AfterUpdate event procedure to each form control used in a SelCtl object that is not a SelCtGrp object.

    The only action required of the GotFocus event is a call to the **CheckSelCtlChange** procedure.

Command Button Procedures for SelCtlGrp objects

Any form control that is defined as a parameter to a SelCtl object (except those defined as a parameter to a SelCtlGrp object), must have an associated After update event. The AfterUpdate event is used to trigger SelCtlObjList and SelCtl object methods that query the data server for the count of rows matching the user's specifications after the current control update is applied. If no rows match, the user is offered the option of reverting the control to the value present when they first entered the control.

14. Add a command button an associated OnClick procedure for each pair of form controls used as part of a SelCtlGrp object.

    The only action required of the GotFocus event is a call to the **CheckSelCtlChange** procedure.

Assuming that all specifications have been made correctly, the form should now operate correctly.

Selected Records Display Form

[LB – I believe there will be some changes in this section associated with the newly implemented mechanism for passing record selections to coordinated reports. These will be manifest after revisions from the MLR db project are implemented in MLR files…]

1. Set value of module-level constants that define the base query whose return set is display in the **lstRecDisplay** list box:

    a. Set the value of the **mstrcDisplayBaseSelect** constant in form module declarations section – this is the query whose results are displayed in the **lstRecDisplay** list box. The SELECT keyword, row description and FROM clause of the SELECT statement should all be specified here. At present, a trailing space is required to ensure that the

concatenation of this portion of the SELECT statement with the join description is properly delimited.

b. Set the value of the **mstrcDisplayJoinDesc** constant in the form module declarations section – this is contains the SQL expressions that will define the WHERE clause that properly joins all tables listed in the FROM clause of the SELECT statement

2. Set properties of the **lstRecDisplay** list box:

c. revise the number of columns and column widths to properly display the columns listed in the value of the **mstrcDisplayBaseSelect** constant.

d. revise the RowSource property to include the same SELECT statement constructed by concatenation of the **mstrcDisplayBaseSelect** and **mstrcDisplayJoinDesc** constants. This ensures that the same columns are displayed in the lstRecDisplay list box at all times.

[LB – Might be desirable to have this set automatically as part of **FormLoad** procedure in the forms module.]

3. Set properties of **cboRptChoice** combo box:

e. Check SELECT statement specified in RowSource property

4. Check that reports to be provided as choices for the user are properly entered in records of the report_use_local table to be selected by the RowSource query of the **cboRptChoice** combo box.

Report Forms

Standard Report – 20Feb2004

The approach described here is intended for use in linking the user of "record selection" and "record selection display" forms in the application with associated reports. Reports that do not intend to make use of this type of integration need not follow the guidelines offered here.

This approach has now been implemented in another project that makes use of many of the same tools as the MLR project, and the results seem very favorable with regard to flexibility, generality and simplicity of use in composing additional reports that use the same central mechanism.

Construction of a report that makes use of the revised report row selection mechanism proceeds as follows:

1. Construct an MS-Access query that describes the data to be included in the report.

   Construct the query to provide only the data required by the report, and no additional data. This approach will serve to minimize the amount of data being passed across the network from server to client. Since the application is specifically destined for multiple-users in a networked setting, such considerations are important.

   This query will typically be used only during report construction. In it's final form, the constructed report will employ internal SQL statements rather than a predefined MS-Access query. The constructed query only serves to ease report construction at its initiation. It is recommended that the query be deleted after the construction of the associated report is completed. Accordingly, it is useful to name this query in a way that indicates both the data

subject of the query and the fact that this query is only "temporary". This will make it easier to clean up the "artifacts" of the report construction process after completion.

2. The basic report is composed using either MS-Access report construction wizards or manual design processes.

   The report should be based on the query constructed in step 1.

3. Verify that the report is displaying data summaries in the fashion desired for the report.

4. Place an unbound text box control named "**mtxtSelDesc**" in the report header just below the displayed form title.

   This text box will be used programmatically to display a description of the query filter used to define row selection when the report is executed. The size of the text box should be enlarged to provide a capacity for displaying about 200-300 characters.

   The **Visible** property of this control should be set to 'no'.

   It has been standard practice to center report titles and to specify the use of bold, 20 point text for the report title.

5. Open the VisualBasic code module associated with the report and copy in all VisualBasic code from a preexisting report code module using the revised filter mechanism (the "**Report_donation_rpt**" module is a good choice as a code source).

6. Revise the **RecordSource** property of the report to include a copy of the SQL SELECT statement defining the query from part one rather than the *name* of that query.

   This is useful for two reasons:

   > First, replacement of the query reference with the associated SELECT statement allows deletion of the query definition. This will help keep the number of objects included in the application small and will make it easier to manage the application and it's maintenance. In addition, it may be desirable to allow some expert users to construct custom queries for their own use. In such a case, it will be very desirable that no queries essential for basic application operation remain in the queries collection of the MS-Access database provided to the user.
   >
   > Second, in some cases the user or developer may have a need to open the report in a way that precludes the usual operation of the associated VB code module. In such a case, the mechanisms usually applied by that code module to assign a value to the **RecordSource** property may not be functioning. For this reason, it is useful to have the SELECT statement defining the **RecordSource** property represented in both the default **RecordSource** property of the report and in the associated VB code module.

7. In the report code module, alter the specification of the module-level constants **mstrcDisplayRowDesc**, **mstrcDisplayFromDesc**, and **mstrcDisplayWhereDesc** to accurately represent the SELECT statement of the SQL statement representing the query from step 1.

   This is usually a relatively easily task for an intermediate to advanced SQL programmer. The following sequence of steps facilitates the operation:

   a. Open the query constructed in step 1, move to the "SQL view" of the query, and cut and paste the full SQL statement to a text editor like Notepad, etc.

   b. In the text editor, edit the SQL statement to visually separate the "row description", "from clause" and "where clause" components of the statement.

It is useful to add string delimiters (") and VB continuation characters (' & _') while formatting the statement to neatly display the query. These additions are sometimes easier within a text editor than within the VB editor. It is an advantage to make use of VB continuations to present the statement in a very readable fashion within VB. It really helps with troubleshooting, which is *often* needed.

c. Cut and paste each component into the declaration statement for the corresponding constant.

If tables are named in a fashion that does not strictly require the use of square brackets around table names and column names, readability of the statement will be improved by removing them. In many cases, MS-Access automatically places these brackets within the SQL SELECT statement representing a query, whether they are needed or not. Only in cases where column or table names include spaces or other special characters are these absolutely needed. In other situations, they create visual clutter for the programmer to wade through without really providing compensating value.

Here is an example of the representation style that has been found most useful in development work:

```
Rem Define base SELECT statement used to define rows used in report
Rem Row description should NOT include SELECT keyword
Private Const mstrcDisplayRowDesc As String = _
         "lq.loctn_id, " & _
         "lq.party_id, " & _
         "py.name AS party_name, " & _
         "ltq.abbr AS loctn_type, " & _
         "IIf(lq.primary_yn=Yes,""Yes"",""No"") AS primary, " & _
         "lq.organization_name AS loctn_name, " & _
         "lq.address_1, " & _
         "lq.address_2, " & _
         "lq.city, " & _
         "lq.state, " & _
         "lq.postal_code "

Rem FROM description should NOT include FROM keyword
Private Const mstrcDisplayFromDesc As String = _
         "loctn AS lq, " & _
         "loctn_type_defn AS ltq, " & _
         "party AS py "
Rem If ANSI SQL89 query style is used, the join description should _
    contain both join criteria AND row subselection criteria.  If _
    ANSI SQL92 query stile is used, the join description will contain _
    only row subselection criteria, since join criteria will be _
    specified in the FROM clause using INNER JOIN, RIGHT JOIN, etc. _
    clauses
Private Const mstrcDisplayWhereDesc As String = _
         "py.party_id=lq.party_id " & _
         "AND lq.loctn_type_defn_id=ltq.loctn_type_defn_id "
```

8. Alter the value assigned to the module-level **mlngcDataSubject** constant to accurately represent the data subject of the form.

The assigned value is provided as an argument to the **basRptShared.RptFilterClear** method used in the **Report_Close** procedure. A public enumerated list defined in the basRptShared module is used – the values presently defined for use are:

rfsContact

rfsCounty
rfsDonation
rfsEasement
rfsParty
rfsPerson

9. Alter the values assigned to the module-level constants **mstrcIdCol** and **mstrcIdColAlias** to accurately specify the name of the primary key column that uniquely identifies each row to be included in the report, and to accurately reflect the table alias used in the report's base SELECT statement to identify the table in which that column is located.

   NOTE: The "alias" may in fact be the complete table name (i.e. "person", etc.).

10. Modify the **Report_Activate** procedure.

    The statement that assigns a value to the **strSelDec** variable must be altered to indicate the retrieval of the correct property from the basShared module. The revised statement should look like

    ```
    strSelDesc = basShared.datasubjectSelDesc,
    ```

    where *datasubject* is changed to reflect the basShared property that is set by the "selected records display" form used to execute the report.

11. Check the "event" properties of the report to ensure that the following events trigger associated "event procedures":
    a. On Open
    b. On Close
    c. On Activate
    d. On No Data

12. A row should be added to the **report_use_local** table specifying the report_name and the name of the "selected record display" form that is used to start the report.

    Because the report includes VB code that specifically indicates a data subject, and a subject-specific property retrieval from the basShared module, a report should probably be used in coordination with only a single "record selection display" form. If a similar report is to be used with a different "record selection display" form, a copy of the current report can be used to streamline the set up of a report specific to that use.

Data Export Report – 20Feb2004

Automated exports of addresses, etc for selected rows is handled through very nearly the same mechanism used for standard reports:

A. The user is expected to specify a record selection mechanism using the provided "record selection" forms
B. The user moves to the associated "record selection display" form to review the results of their specification.
C. The user selects a report designed for data export and triggers that report.

D. The report that is displayed also opens a pop-up window that allows the user to either cancel the process or send the displayed report to a system file. *The pop-up window and related functions are the only difference between a data export report and a "standard" report.*

Appendix A – Report Row Selection Mechanism History

Purpose

This appendix provides a description of the earlier mechanism proposed for use in linking reports with "record selection display" forms, the liabilities of that approach, and the reasons for the proposed design now implemented in the application

History

Initial Report Mechanism – 10Feb2004

NOTE: The mechanism currently employed in coordinating reports and associated "record selection" forms still needs work. This section describes the *current* mechanism rather than that *final* mechanism.

As of 12Feb2004, reports are expected to retrieve the query filter constructed by a record selection form. The retrieved query filter is then added to the report's already-defined base SELECT query. When properly constructed, the report then details only the rows previewed in the associated "selected records display" form.

The most complex issues with construction of report forms have to do with specification of SQL SELECT statements defining the records to be represented on the report. These SELECT statements must be constructed in a way that allows them to interact correctly with query filters constructed by a "record selection" form and then eventually passed to the report. A coordinated approach to use of table aliases in both "record selection" forms and reports is therefore required. A report will not run correctly if the same table alias is used in different meanings in the report and in the query filter constructed by an associated "record selection" form. Neither will a report run correctly if the retrieved query filter employs table aliases that are not defined in the combined SELECT statement.

Typically, both reports and record selection forms can be considered to use a single table as the "primary" table. The name of a the record selection form is usually an obvious indicator – for example, the "party selection" form constructs query filters that use the "party" table as the "primary table". Each report should similarly focus on a single "primary table", and should only accept query filters from "record selection" forms that employ the same "primary table".

1. Specify the base SELECT statement for the report in the **mstrcDisplayBaseSelect** constant defined in the report's module-level declaration section.

   This constant should contain the "SELECT" keyword, the row description, the "FROM" keyword and the table description portions of the SELECT statement. At present, a trailing space is required to ensure proper concatenation of a WHERE clause to this value.

2. Specify the join description to be applied in the base SELECT statement for the report in the **mstrcDisplayJoinDesc** constant defined in the report's module-level declaration section.

3. The form's **RecordSource** property may be set to any query specification that assists in initial development and troubleshooting of the report. This may help the clarity of the overall report for a developer, but the report form typically overwrites the RecordSource property with a constructed SELECT statement at the time the report opens. That constructed statement combines the values of the mstrcDisplayBaseSelect and mstrcDisplayJoinDesc variables and the query filter passed to the report.

4. Check that the RecDisplayRowSource procedure correctly extracts the proper query filter.

   This filter is usually extracted from a property defined in the basShared module, using a statement like

   ```
   strFilter = basShared.PartySelFilter
   ```

Proposal for Revised Report Row Selection Mechanism

NOTE: This proposal outlines essentially the same mechanism implemented and for which needed modifications are outlined above.

After reviewing a few reports in some depth, I believe I now recall that this is an area where there is still some "design" work to be done.

At present, a report accepts a query filter from an associated "record selection" form by extracting a property value from the basShared module. Each "record selection" form stores the value of it's constructed query filter there when a command to move to a "selected record display" form is issued. So the filter is also then available for use by the report if the user requests the report.

There are two problems with this approach:

1. The set of tables that can be involved in any "record selection" query filter is quite variable. The tables actually included in the filter expression depend on what controls the user chooses to employ in their record selection specifications. It is probably desirable that a report NOT include EVERY table that MIGHT be used in the SELECT statement defining it's row source.
2. The “record selection form” construction presently used in the application applies the older style “WHERE exp1 AND exp2 AND…” style of specifying table joins, which will make it difficult to use RIGHT JOIN and similar expressions in the base SELECT statement of a report.

   There will probably be some reports that will require outer joins. I believe this will be difficult with the current system of “passing a query filter”, since the clearest expression of an outer join employs a fundamentally different approach to specifying table joins. Because the “query filter” use under the proposed approach is very direct and simply, I believe it will be easier to accommodate than the more complex and variable filter query typically obtained from a “record selection form”.

My high-level concept for doing this more flexibly:

A. When a command to generate a report is issued, a procedure is run that fills writes the primary key values of the selected records from the "primary table" in the report to a table in residing in the server.

   I think server location will enhance subsequent query performance, since all other data is also resident there... It is desirable to avoid running queries that require major network traffic... The server location will be particularly desirable if the final implementation includes a “server” DBMS that supports stored procedures. In this case, it will be possible to simply call a “server” procedure that performs this operation entirely within the server itself…I think this would result in nearly the absolute minimum of network traffic for an MS-Access client/”server” DBMS implementation approach.

B. The primary key values written to the "rpt_ids" table must be stamped with some “report key”code that allows the "proper" client to then specify their use in the report's base query.

The value of this stamp could include username/timestamp, OR it could simply be a randomly generated key, like a concatenation of 6 randomly selected characters each from the set {01..9AB.Z}, etc. I believe the approach employing a random sequence of characters is advantageous, requiring a value that is both simpler to generate, very easy to increase in size and more compact for the same information content (Each character can hold something like 36 different values). This again would minimize network traffic and the tendency of the resulting values to identify unique rows with reduced numbers of characters will convert into performance advantages in most types of DBMS indexing.

C. The “report key” value is passed to the report.

D. The report module constructs a filter that includes an "IN (SELECT ri.id_val FROM rpt_row_id AS ri WHERE rri.rpt_key='blahzzz')" subquery and modifies the report's base SELECT query by including that condition when the report is run.

E. When the user closes the report, rows with the matching "use stamp" in the rpt_ids table are deleted.

I THINK this would completely get around the issue of varying table involvements in reports... issues of how to employ OUTER JOIN clauses in the base SELECT statement of a report, etc.

I believe the initial implementation would still place the rpt_row_id table in the server, even when that server is an MS-Access Jet database. I do not believe that this approach would substantially increase the amount of network traffic caused within an all-Jet implementation, and I feel it would lay the groundwork for a later transition to a server DBMS supporting stored procedures.

NOTE: I’ve prototyped this approach and gotten it pretty close to the form that would be required for final implementation. (This required about 10 hours of programming, and perhaps 4-5 hours of “thinking/planning” time.) The prototype is in the test.mdb database. Queries in this database illustrate that this approach allows the use of both ANSI 89 SQL and ANSI 92 SQL to define the base SELECT statement for the report. This flexibility allows the developer to employ classic uses of FROM and WHERE clauses to define equi-joins, or the more modern INNER JOIN, LEFT JOIN, RIGHT JOIN approaches of the more recent ANSI SQL standards.