Blaise Audit Trail Data in a Relational Database

Joel Devonshire, Youhong Liu, and Gina-Qian Cheung
Survey Research Center, University of Michigan

1 Introduction

One principle that has become increasingly clear to many technical team members at the University of Michigan's Survey Research Center, Survey Research Operations (SRO) is that paradata are only as useful as the ease with which they can be accessed, manipulated, and analyzed by end users. Even for a relatively small survey research study, the sheer volume of paradata that can potentially be collected can be overwhelming. A conundrum often exists between needing to have high-level questions guide decisions about what data to collect versus needing to see all of the data first in order to know what questions are possible to answer. The end result is often the collection of large quantities of data that are never examined in any meaningful way. To the extent that this happens, it may represent an inefficient use of technical resources and project funds, as well as potentially less robust survey methods. In other words, such an approach may not be optimally reliant on principles of responsive survey design. The concept of responsive design implies that all available paradata inputs be evaluated during the survey design phase as potential indicators of cost and error, and that decisions be made during data collection according to the insights gleaned from such inputs (Groves & Heeringa, 2006). This, in turn, implies that the paradata inputs are readily available for real-time analysis during data collection.

This paper focuses on one specific and very important element of paradata: the Blaise Audit Trail, or ADT. In particular, we describe the development of a new way to store ADT data that we hope will enable end users to more easily access the raw data as it is collected and to write their own custom queries using tools of their choice. By storing most of the raw elements of the ADT file in a relational database format, and doing so on a nightly basis as data collection is happening, a wide range of options become available for data analysis that are more flexible and efficient.

1.1 Background

In 2004, SRO created an application called "ATReport." The main purpose of the ATReport application was to consolidate and process individual ADT files for analysis. It functioned by creating a local Microsoft Access database on the user's desktop, and storing aggregate data in a handful of predefined tables. The application was also able to generate standard summary reports (e.g., timings by item, section, or interviewer), and could identify specific keystrokes during the interview (e.g., backing up and changing answers, invoking help, switching languages, etc.). While this application significantly advanced the ability to use ADT data, it had some distinct disadvantages, ones we surmise are related to why the application is used only rarely at SRO.

For example, when a user wishes to analyze ADT data with the existing system, she must first gain access to the individual ADT files, which she may or may not already have access to. She then needs to install the ATReport application on her local machine, and process the files of interest. The processing itself can take many hours, especially for a large number of lengthy interviews. When the processing is complete, the user is left with a local database that only contains predefined aggregated data. While the user can then go on and write her/his own queries, the results are limited to what can be gleaned from the ATReport tables. If a user wants to isolate one specific case and examine it in more detail, then she needs to use a different SRO application called Playback, which uses the ADT and BDB files to recreate the individual interview session.

To illustrate why these aspects of usability are important, consider a hypothetical scenario in which a survey director is concerned about reports in the field that interviewers are getting stuck on one particular field in the data model. She finds out about this problem over the weekend and she would ideally like to have a follow-up plan to recommend at a team meeting on Monday. Specifically, she'd like to know the overall amount of time all interviewers spend in this field, and obtain a report by interviewer so she can identify specific staff members of concern. She also wants to assess what behavior interviewers engage in immediate before, during, and after entering this field. Are they backing up, for example, or entering an F2 comment? Finally, she wants to know how interviewers resolve the issue; do they tend to suspend out of the instrument at that point, or fill in random data, or something else?

While it is true that we currently have the tools to answer all of the questions raised in this scenario, it would be a somewhat complicated process. For example, it is the currently the role of one of the technical team members at SRO, the Data Manager, to have access to the data and to create data sets that can be analyzed by others. In this scenario, the Data Manager would need to be contacted, and arrangements would have needed to be made to allow access to a range of ADT files. The processing and analyzing of them would have needed to wait until Monday morning and would have been time-consuming and inefficient.

Compare this with a vision in which the survey director is able to open up a web browser at home, log in to our project management site, WebTrak, and run a query on the ADT data by selecting a few menu options. The results are automatically up-to-date, and the query can easily be saved to run again during the next week. After running the query, she sees that the "problem" is really only happening with 5 out of 60 interviewers, and they are all new staff. It seems that they are repeatedly encountering a hard check in Blaise due to incorrect data entry a few questions earlier. They try repeatedly to enter the same data, and then give up and suspend out of the instrument. Going into Monday's meeting, she therefore knows that this is a staff training issue rather than a programming issue.

1.2 A New Approach: The Centralized Relational Database

To better utilize the audit trail data and to provide more security and efficiency, users have requested that we consolidate all the individual audit trail files into a centralized relational database. The advantages of this approach include:

- Users will not need direct access to the ADT files themselves, which are often located in protected server locations. This helps to ensure that ADT files, and the potentially identifying respondent information included in them, are not unnecessarily copied to multiple – and undocumented – locations.

- Instead of requiring users to create their own ADT database in MS Access, having a central relational database located on a secure SRO server is more efficient, and specific user permissions could be granted as needed. Users link to the tables instead of creating new ones. Unlike point #1, this speaks to the efficient use of the end-data, not the ADT files themselves.

- By processing the ADT files through our automated nightly interview data merge process, we greatly reduce the time it takes to analyze data, since the ADT file consolidation would already be complete. In addition, the nightly processing time itself would be reduced since only a small number of ADTs would be processed each night.

- Nightly processing the ADT data into other nightly batch reporting systems, such as WebTrak, to provide automatic detail on problem cases or interviewer behavior (e.g., aggregate timings by interviewer).

- Users could connect to the ADT database using familiar tools such as SAS or MS Access. They can create their own queries and more easily merge to other databases (e.g., SurveyTrak paradata). This is possible with the current implementation, but would be made easier with a central database.

2 Database Structure

In its current state of development, the database is minimal, comprising only five operational tables. For the purposes of this paper, only two will be described, tADTField, which contains the raw data from the ADT files, and tSuspendVariables, which captures the specific Blaise fields in which instrument suspensions occurred.

The table tADTField contains columns for most of the individual pieces of data for the ADT, and these are listed in Table 1. Notice that the table is able to capture information about the interviewer, the data model, the number of visits to the data model and each field, the time spent within each field, and a number of other Boolean values related to hot keys and events within each Blaise field.

Table 1. Structure of table tADTField

Field Name	Data Type	Size	Description
ProjectID	Text	30	Unique project identifier
CaseID	Text	30	Sample ID within a project
FldSeq	Long Integer	4	Sequential counter; increments by one for each ADTrow
MetaName	Memo	-	Name of Blaise data model
MetaTime	Text	200	Date/Time Blaise data model was created
UserID	Text	30	Interviewer ID for this particular instrument entry
FldName	Memo	-	Blaise long field name
BlockName	Text	100	Blaise block name
FormVstNum	Long Integer	4	Instrument visit number; increments by one for each entry
FieldVstNum	Long Integer	4	Field visit number; increments by one for each entry
PrevLeaveLineNo	Long Integer	4	Previous field Blaise line number
EnterLineNo	Long Integer	4	Current field entered Blaise line number
LeaveLineNo	Long Integer	4	Current field exited Blaise line number
ISLEAVEFORM	Long Integer	4	Interviewer exited Blaise at this field (Yes/No)
EnterDate	Text	30	Date interviewer entered the Blaise field
EnterTime	Text	30	Time interviewer entered the Blaise field
LeaveDate	Text	30	Date interviewer left the Blaise field
LeaveTime	Text	30	Time interviewer left the Blaise field
Fld_Time	Text	30	Time (sec) spent within the Blaise field
Fld_SS	Text	30	Time (sec) spent within the Blaise field (different format)
Fld_Time_Mss	Long Integer	4	Time (milisec) spent within the Blaise field
Btw_Time	Text	30	Time (sec) spent between last Blaise field and this one
Btw_SS	Text	30	Time (sec) spent between last Blaise field and this one
Btw_Time_Mss	Long Integer	4	Time (milisec) spent between last Blaise field and this one
Adj_Time	Text	30	Fld_Time + Btw_Time
Adj_SS	Text	30	Fld_Time + Btw_Time
Adj_Time_Mss	Long Integer	4	Fld_Time + Btw_Time
RspLat_Time	Text	30	Time (sec) between entering field and first keystroke
RspLat_SS	Text	30	Time (sec) between entering field and first keystroke
RspLat_Time_Mss	Long Integer	4	Time (milisec) between entering field and first keystroke
Key_Count	Long Integer	4	Number of keystrokes while in Blaise field
Enter_Value	Memo	-	The value of the Blaise field upon entry
Leave_Value	Memo	-	The value of the Blaise field upon exit
Leave_Cause	Text	50	Action that initiated interviewer to leave Blaise field
Leave_Status	Text	50	Field leave value is normal or DK/RF
Prev_Lang	Long Integer	4	Language was switched to previous while in field (Yes/No)
Next_Lang	Long Integer	4	Language was switched to next while in field (Yes/No)

Set_Lang	Long Integer	4	Language was set while in field (Yes/No)
CtrlL_SetLang	Long Integer	4	Language was changed with hot key while in field (Yes/No)
ALTXExit	Long Integer	4	Alt-X interview suspension was initiated at this field (Yes/No)
RemClk	Long Integer	4	Interviewer remark was initiated at this field (Yes/No)
RemChng	Long Integer	4	Interviewer remark was changed at this field (Yes/No)
QHelp	Long Integer	4	QXQ Help was initatiated at this field (Yes/No)
BlaiseHelp	Long Integer	4	Blaise Help was iniated at this field (Yes/No)
Error_Esc	Long Integer	4	Blaise check was closed at this field (Yes/No)
Error_Esc_Text	Memo	-	Text of the Blaise check encountered before closing
Error_Supp	Long Integer	4	Blaise check was suppressed (Escape) at this field (Yes/No)
Error_Supp_Text	Memo	-	Text of the Blaise check encountered before suppressing
Error_Jmp	Long Integer	4	Blaise field that interviewer jumps to after Blaise check
Error_Jmp_Text	Memo	-	Text of the Blaise check encountered before jumping
Media_Start	Long Integer	4	Blaise launched media file while in this field (Yes/No)
Mouse_Click	Long Integer	4	Any mouse click detected while in this field (Yes/No)
F1	Long Integer	4	F1 hot key was pressed while in this field (Yes/No)
F2	Long Integer	4	F2 hot key was pressed while in this field (Yes/No)
F3	Long Integer	4	F3 hot key was pressed while in this field (Yes/No)
F4	Long Integer	4	F4 hot key was pressed while in this field (Yes/No)
F5	Long Integer	4	F5 hot key was pressed while in this field (Yes/No)
F6	Long Integer	4	F6 hot key was pressed while in this field (Yes/No)
F7	Long Integer	4	F7 hot key was pressed while in this field (Yes/No)
F8	Long Integer	4	F8 hot key was pressed while in this field (Yes/No)
F9	Long Integer	4	F9 hot key was pressed while in this field (Yes/No)
F10	Long Integer	4	F10 hot key was pressed while in this field (Yes/No)
F11	Long Integer	4	F11 hot key was pressed while in this field (Yes/No)
F12	Long Integer	4	F12 hot key was pressed while in this field (Yes/No)
CtrlD	Long Integer	4	Ctrl-D hot key was pressed while in this field (Yes/No)
CtrlR	Long Integer	4	Ctrl-R hot key was pressed while in this field (Yes/No)

The table tSuspendVariables contains columns to track up to 10 instrument suspensions, and also easily identifies where the last suspension occurred. Table 2 lists these columns.

Table 2. Structure of table tSuspendVariables

Name	Type	Size	Description
ProjectId	Text	30	Unique project identifier
CaseID	Text	30	Sample ID within a project
vTotalSuspends	Long Integer	4	Total number of suspends for this sample line
vCaseComplete	Yes/No	1	Is the case completed?
vSuspendedVariable1	Memo	-	Blaise field at which first suspension occurred
vSuspendedVariable2	Memo	-	Blaise field at which second suspension occurred
vSuspendedVariable3	Memo	-	Blaise field at which third suspension occurred
vSuspendedVariable4	Memo	-	Blaise field at which fourth suspension occurred
vSuspendedVariable5	Memo	-	Blaise field at which fifth suspension occurred
vSuspendedVariable6	Memo	-	Blaise field lhich sixth suspension occurred
vSuspendedVariable7	Memo	-	Blaise field at which seventh suspension occurred
vSuspendedVariable8	Memo	-	Blaise field at which eighth suspension occurred
vSuspendedVariable9	Memo	-	Blaise field at which ninth suspension occurred
vSuspendedVariable10	Memo	-	Blaise field at which tenth suspension occurred
vDate1	Text	30	Date on which first suspension occurred
vDate2	Text	30	Date on which second suspension occurred
vDate3	Text	30	Date on which third suspension occurred
vDate4	Text	30	Date on which fourth suspension occurred
vDate5	Text	30	Date on which fifth suspension occurred
vDate6	Text	30	Date on which sixth suspension occurred
vDate7	Text	30	Date on which seventh suspension occurred
vDate8	Text	30	Date on which eighth suspension occurred
vDate9	Text	30	Date on which ninth suspension occurred
vDate10	Text	30	Date on which tenth suspension occurred
vTime1	Text	30	Time at which first suspension occurred
vTime2	Text	30	Time at which second suspension occurred
vTime3	Text	30	Time at which third suspension occurred
vTime4	Text	30	Time at which fourth suspension occurred
vTime5	Text	30	Time at which fifth suspension occurred
vTime6	Text	30	Time at which sixth suspension occurred
vTime7	Text	30	Time at which seventh suspension occurred
vTime8	Text	30	Time at which eighth suspension occurred
vTime9	Text	30	Time at which ninth suspension occurred
vTime10	Text	30	Time at which tenth suspension occurred
vLastSuspendedVariable	Memo	-	Last Blaise variable at which a suspension occurred
vLastSuspendedDate	Text	30	Date on which the last suspension occurred
vLastSuspendedTime	Text	30	Time at which the last suspension occurred

3 Integration with Other SRO Systems

As mentioned above, ADT files will be processed on a nightly basis when all active production projects at SRO undergo the Blaise data merge. Prior to this implementation, nightly ADT processing at SRO was limited to a calculation of overall interview length, which was then written to a column in our production sample management system, SurveyTrak. This new process will allow a tight integration of detailed ADT information into all other aspects of daily project management.

For example, SurveyTrak currently has the capability to capture a wide array of paradata, including things such as contact attempts, appointment times, incentives and letters sent to respondents, characteristics of the sample (e.g., mode and language of collection and interviewer assignment), and so forth. One direct advantage of processing ADT files nightly is that keystroke data can then be joined with any of the other "real-time" SurveyTrak paradata that we collect. One example of how this might be useful is in the early stages of data collection. If reports were coming in from the field that interviews were taking much longer than usual, one could easily query interview lengths by mode, for example, to see if face-to-face interviews were taking longer than telephone, or whether the increase in length was due to some other factor.

This integration also allows these kinds of questions to be included in the reporting specifications during the initial project design. Our daily programmed Field Progress Reports (FPR) could include information on keystroke data, and key queries could be built into WebTrak, our web-based project management application. Such integration encourages project staff to consider questions related to keystroke data, and could potentially become a key component in the work we do at SRO.

4 Practical Applications

As is often the case with paradata, the practical applications of ADT data in a relational database depend significantly on what questions are of interest to survey directors and project managers. We have already mentioned above a few relevant questions or scenarios in which this data could be applied. In this section, however, we present a more detailed workflow diagram of how a few key questions might potentially be approached with the new system.

4.1 Estimating Length of Recorded Interviews

A real-life scenario for one of the authors of this paper involved the following scenario. We were asked by project management to take a proposed digital recorded interview (DRI) capture list specification[1] and estimate how long the actual DRI recordings might be if the capture list were used (and before the recordings were viewed by our quality control team). The project managers wanted capture list timings summaries by sample ID and instrument section, using existing data that had recently been collected.

Because the capture list is a document that can be easily imported into a database system such as MS Access, the Blaise fields specified in the capture list can be joined with the ADT database data to pull out only the fields included in the capture list. In SAS, this was handled by a simple PROC SQL statement:

```
PROC SQL;
    SELECT DISTINCT a.*
    FROM tADTField a JOIN CaptureList b
    ON a.fldname  =  b.BlaiseName_Full
    ORDER BY a.caseid, a.fld_name;
QUIT;
```

From there, instrument sections can be defined via queries that examine the Blaise field name prefixes, and section and interview subtotals can be calculated with simple SQL queries:

```
PROC SQL;
```

[1] A DRI capture list is a specification document (text) that defines which Blaise fields should be digitally recorded during the interview. When a sample line is flagged to record, Blaise uses the capture list to control when the recording software pauses and resumes the recording of sound and screen activity.

```
    SELECT caseid,
           sum(Adj_Time) as CaptureList_Length
    FROM capture_list
    GROUP BY caseid;
QUIT;
```

The resulting data might look something like this:

Table 3. Interview Length Calculated Using Capture List Specification

CaseID	Length of Interview
6000040020	0:28:26.274
6000260010	0:37:58.675
6000470010	1:11:30.885
6000510010	0:47:37.999
6000540020	0:53:29.781
6000540021	0:56:10.560
6000600010	0:34:11.024

4.2 Examine Specific Keystroke Behavior

In a relational database format, certain kinds of questions become incredibly easy to answer. For example, finding out what fields were associated with F2 comments by the interviewer is as simple as writing the following query:

```
PROC SQL;
    SELECT DISTINCT fldname
    FROM tADTField
    WHERE F2 = 1
           ORDER BY fldname;
QUIT;
```

To determine what specific fields had values changed by the interviewer (as opposed to retaining their preloaded values), one could use the following query:

```
PROC SQL;
    SELECT DISTINCT fldname
    FROM tADTField
    WHERE Leave_Value <> Enter_Value
           ORDER BY fldname;
QUIT;
```

To determine what fields were associated with backing up to the previous field, and to count how many times this occurred for each field, one could use this query:

```
PROC SQL;
           SELECT DISTINCT fldname,
                  Count(fldname) as Count
```

```
    FROM tADTField
    WHERE Leave_Cause IN ("Move Left","Move Up")
    GROUP BY fldname
    ORDER by fldname;
QUIT;
```

If one wished to focus in on a particular field and develop a summary keystroke report, examining how these changed between versions of the data model, one could use the following query:

```
PROC SQL;
    SELECT ProjectID,
            MetaTime AS DataModel_Version,
            Count(MetaTime) AS N,
            Avg(Fld_Time_Mss) AS Avg_FldTime,
            Avg(Btw_Time_Mss) AS Avg_BtwTime,
            Max(FieldVstNum) AS Num_Visits,
            Avg(Key_Count) AS AvgKey_Count,
            Max(Key_Count) AS MaxKey_Count
    FROM tAdtField
    WHERE fldName = "SecB.Marriage2.B055_"
    GROUP BY ProjectID, MetaTime;
QUIT;
```

The resulting output might look something like this:

Table 4. Keystroke Summary for One Blaise Field By Data Model Version

DataModel_Version	N	Avg_FldTime	Avg_BtwTime	Num_Visits	AvgKey_Count	MaxKey_Count
Friday, January 06, 2012 1:08:04 PM	5	36781	48	3	1	3
Tuesday, January 24, 2012 1:48:20 PM	2	8944	48	2	1	2
Wednesday, February 01, 2012 9:50:26 AM	1	592	43	1	2	--

4.3 Examining Interview Suspensions and Checks

A final example that will be considered in this paper relates to the analysis of particular events during the interview session, namely suspensions and hard or soft checks. While we currently have tools at SRO to examine individual cases to see where or how a check or interview suspension occurred, we have had no easy way to analyze the entire data set. By having the ADT data stored in a relational database, we can now easily find patterns of interviewer behavior, such as common suspension points, where errors or checks tend to occur, and how interviewers tend to respond to these checks.

Using the table tSuspendVariables, for example, one could determine how often the interview is suspended before a critical junction, or within a complex set of questions. One could see whether interview suspensions are, on average, associated with longer total interview times, and one could determine how long, on average, the duration is between entries into the instrument.

For soft checks, one could see how many times interviewers suppress the check, and whether this changes as a function of interviewer experience, data model versions, or other factors. Since the database stores the text of the check, one could also calculate summaries of the checks themselves, seeing if certain ones are triggered more than others. One could see whether checks and suspensions as a whole were more common toward the beginning of data collection, or occurred more frequently among certain interviewers.

One interesting example involves the detection of possible interviewer falsification, in which the interviewer attempts to enter erroneous data. For example, if a query were written to isolate all of the hard checks encountered during the interview, it might look something like this:

```
PROC SQL;
    SELECT ProjectID,
        CaseID,
        UserID,
        FldName,
        Enter_Value,
        Leave_Value,
        Error_Jmp,
        Error_Jmp_Text
    FROM tAdtField
    WHERE tAdtField.Error_Jmp =1
    ORDER BY tAdtField.CaseID;
QUIT;
```

Let’s say that this query finds that the following check occurred several times: “This is not a valid zip code. Please check.” The analyst then looks at the value stored in the field Enter_Value and discovers many instances of the value “99999.” In this case, the interviewer is attempting to bypass the zip code question by entering in fake data.

5 Further Development and Conclusion

The above examples are just a few of the kinds of things that potentially could be examined using the ADT data in this format. However, this initiative is still in its early stages, and one could imagine the development of additional tables to hold aggregate data based on the raw data in the ADT. For example, if project managers knew in advance that they wanted to focus on a particular group of interviewers or sample from a particular geographic area, summary tables could be automatically updated daily with this aggregate data. In another example, if regular timings reports for each section of the interview were required, a crosswalk table could be created to link each field name within the data model to a section name (if the Blaise instrument was not already programmed in straightforward section blocks). Stored procedures could then be run daily to update a table of aggregate section timings. These types of enhancements would make the raw ADT data easier and faster to analyze.

Ultimately, the development of such a tool will be driven by the questions that need to be answered to assist in collecting high quality survey data. However, paradata often involves a chicken-and-egg scenario in which some questions are not asked because people do not know the data exist or do not readily have access to them. We hope that by making audit trail data more easily accessible, the data will used more often and become a standard part of responsive survey design decisions.

6 References

Groves, R. M. and Heeringa, S. G. (2006). Responsive design for household surveys: tools for actively controlling survey errors and costs. Journal of the Royal Statistical Society, Series A, Statistics in Society. Volume: 169, Issue: 3, Pages: 439-457.