Title Page	
Project	Portrait Tree
Chapter	03
Content	Stored Procs and Early UX

Stored Procs and (beginning) the User Experience

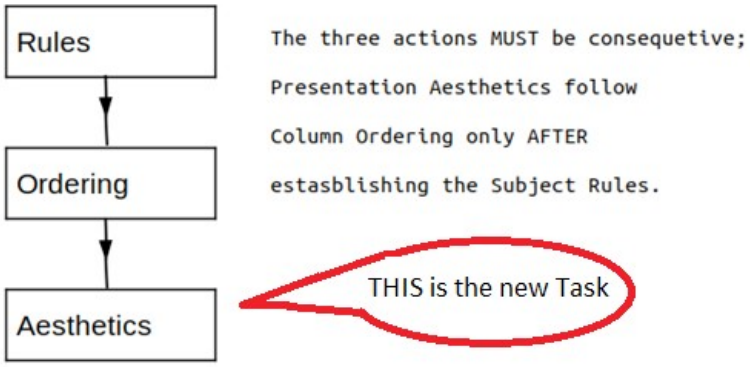
The code we have so far has contributed to the first two tasks needed every time we make the a new Portrait Tree. The end of the last chapter clearly hinted at using a database. Here the database is brought in. The expression “Stored Procs” refers to major functions that the database performs. The future users of this product will have screens that enable them to guide the making of “Portrait Trees”. The product for them is about “Building Trees” and NOT database/progamming technology.

This new viewpoint fits well with the new task that we are adding here. In other words, constructing the new task will give a reasonable base for how the future users will experience the product itself.

Background Context

The work in chapter-02 implemented basic rule-processing and the document hinted at how columns can be re-ordered. Rule Processing covers the first two tasks, Rules and Ordering. The obvious way for the table to support this is to add to it, a column we can call “Sequence Push”. This overrides the existing “Rule Number” unless two rules have the same push. In other words, there is a way to stay within the existing rules and still influence the order items appear under the same root.

Having assumed that the Rule Processor works, we know where to start our new task.



Getting to Stored Procs

Ever since the second stage of this series, the emerging solution had assumed it had a database of SQL tables. From that, we know that we can write Stored Procedures. They can handle the data-functions performed on our data tables. This functionality is native to the SQL-Engine itself and includes many aspects of the data manipulation language necessary to do this.

For readers who lack familiarity with SQL itself, I have a Technical Outline which gives a starting point for understanding the command lines that make up these Stored Procs.

Technical Outline within SQL

Most of the work is handled by “Data Manipulation Language”. In this sequence I mention some of the straightforward techniques that have gone into the material.

1. SELECT … ORDER BY Field-1, Field-2- You would remember from Chapter-02 that the order by which rules enter the system affects the Left to Right results for items under the same root.

2. DELETE FROM … Table Name (WHERE) - The ordinary use of this is to empty part of a table depending on the “WHERE” clause; however when the clause is missing, it completely empties the table.

3. INSERT INTO … Table Name (f1,f2,f3) VALUES(~, ~, ~) - Appends a line onto an existing table understanding the values in the same order as the explicit field names.

4. INSERT INTO … Table Name (f1,f2,f3) VALUES(~, ~, ~) - Changes values within all existing rows which match the WHERE clause at the end.

There are times when simple Update/Insert/Delete commands CAN’T finish the job because updating multiple tables or affecting relationships which are tricky to implement. To cope with this, I have also used two “Data Definition Language” constructions; these allow temporary tables which accept column headers and content WITHOUT the complex relationships that the original contained. the material.

1. DROP … Table Name - You CANT create a table that already exists. By getting rid of it you also lose whatever it contained.

2. CREATE … Table Name as <full select statement> ... - Normally select statements are read in by Software or Reported to the screen. This form of the command simply places it into a new table which can then be used to interact with existing tables.

Using Stored Procs

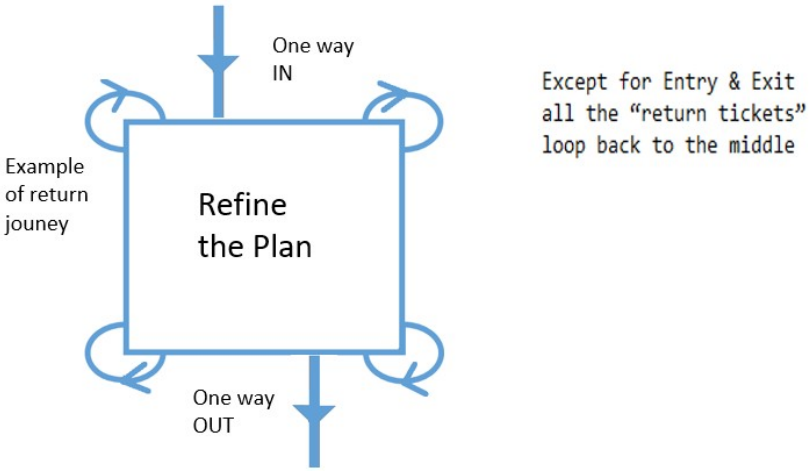
Normally select statements are read in by Software or Reported to the screen. This form of the command simply places it into a new table which can then be used to interact with existing tables.

Infrastructure for the Programmer

Since we have ASSUMED that the Rule Processor works. That really leaves this phase with a handful of obvious "actions within the task". For easy reference later on, they are placed together at the end of this section.

Taking the broadest possible view, BEFORE the task (not completed) there is one way in; similarly, AFTER the task (not even started) there is some graphics and/or publishing – which means one way out.

For technology reasons, one of these "return tickets" needs a middle stop. Ignoring that one detail gives a schematic resembling this:



Focus on the User-Concept

In our previous development stage, there was an algorithm which automatically presented a credible version of the rules which was inadequate for human readers. The future-user of this product is going to edit this into a more acceptable form. Thus, our user becomes the **editor** in charge of making the product output more acceptable.

When you are NOT sure it is handy to have a "best so far" mark so that you can go back to it when you need to. Generally, people are unhappy with too little feedback to assure them that the system did what it was asked to do.

Thus, extra work, which refers to the database itself gives supporting-evidence that the action succeeded. Although this can be quite sophisticated, I just wanted to have users feeling that the action gave some achievement.

All the above, centers on a single main page where each "button" brings the editor back home after seeing what it did. The **editor** we started with has one action for each of these buttons.

Portrait Tree

Button Action Technology

The specific jargon for the coming list is:

Basic is a starting point from the automatic algorithm. **Carrier** holds Simple instructions for building the result as a spreadsheet. Adjuster is a parameter table defines the changes from Basic to the Carrier. On one side of our workflow, the term **Main** means the "current pair of Adjuster and Carrier". On the other, the term **Latest** means the "Best so Far (saved version) pair of Adjuster and Carrier".

Button List

- Reset All
- View Main Adjuster
- View Main Carrier
- Go Back to LATEST Adjuster and Carrier
- Save Main Adjuster and Carrier as LATEST (which is) Best So Far.

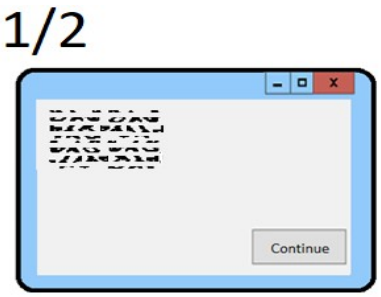
Button Action Technology

The specific jargon for the coming list is:

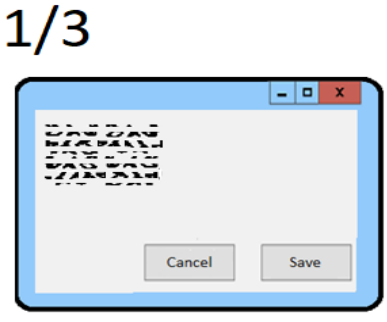
Basic is a starting point from the automatic algorithm. **Carrier** holds Simple instructions for building the result as a spreadsheet. Adjuster is a parameter table defines the changes from B

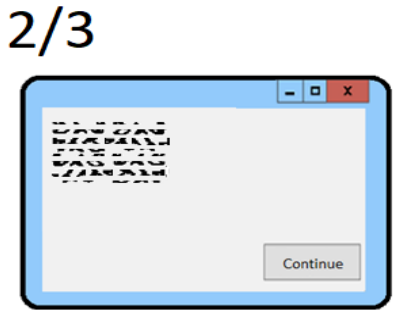
Generic Types of Loop

For non-editing cases, your "return ticket" brings you back in one click (at the halfway point):



For editing cases, your "return ticket" brings you back in two clicks (one third and two thirds):



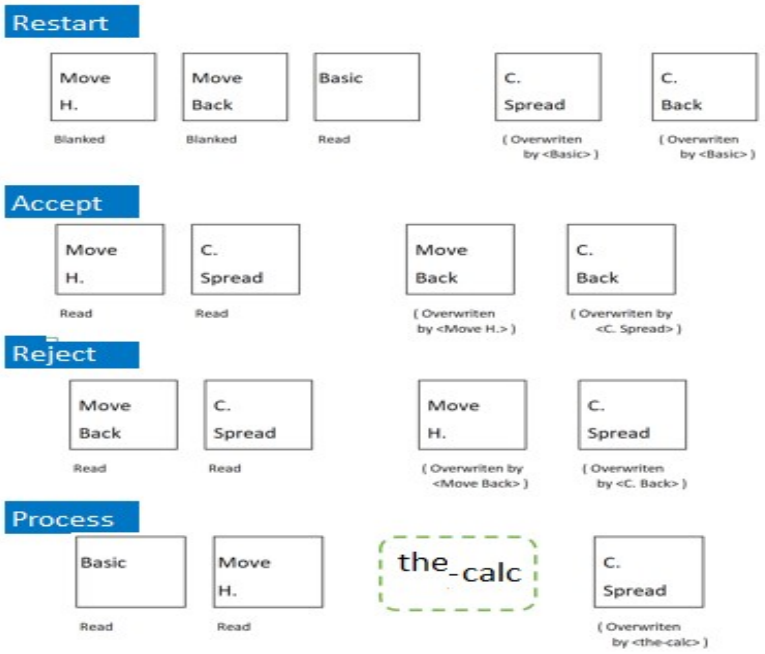


Formalizing the Database

Since the bulk of the forms are directly based on real data tables and we have already outlined the generic groupings of how these forms work. That puts us in a position to view the database-application in a more conventional way. Firstly, we have the Data-Tables:

Symbols representing the REAL Tables in this Database	
Basic	Full Name is : Basic It holds the auto-generated default.
C. Spread	Full Name is : Carry to SpreadSheet It holds the result of applying Move H to Basic
C. Back	Full Name is : Carry Back It holds the previous "Best so Far" of C. Spread
Move H.	Full Name is : Move Horizontal It holds the instructions for changing Basic into C. Spread
Move Back	Full Name is : Move Back It holds the previous "Best so Far" of Move H.

Overview of our Stored Procs



Further Explaining our Stored Procs

- Restart - The Adjustments from the Move H. Table are neutralized: E.g. multiply by ONE and then add ZERO gives the number you started with.

- Process - As explained earlier, the Carrier is formed from Basic using the Adjuster. Pushing the "Process" button makes it happen.

Bringing it all Together

Schedule for Chapter 04

1. The two classes remaining from Chapter-02 should now be finalized.
2. There is only one data-editing User Form. It should be properly designed and documented.
3. Simple read-only versions of <2> can now bring all four Data Forms into the HTML world.
4. We need New Graphics Capabilities to accept the output we have so far.

Technical Points

Those wishing to assemble the code, need to know “what goes where”. The forms-workaround came from LibreOffice and does NOT make up a permanent part of the development path. However, some brief notes give an idea of what was needed to get this working.

These form the Technical sub-Sections that follow. –

For the first time there is an SQL file in the GitHub, for consistency with the file-naming context used so far, it is called C3_Contents of STORED_PROCS.SQL

The aaa_copyForChaper03_index.php gives the starting file pointing at the “cent-AND-LoopBacks” folder which contains theICON-SUPPLY folder .

Other than this there are a few types of file:

1. AAA_Cent_00.PHP
2. the Loop...PHP series
3. A special text file: C3_Text-Report.txt
4. ICO----png series

The first two of these are PHP files and belong in the “centr-AND-LoopBacks” folder. The final one is for pictures and they belong in the ICON-SUPPLY folder within. For consistency, the Text-Report file keeps the status of all unfinished Loop… files.

Chapter Summary

The first two of these are PHP files and belong in the AND-LoopBacks" folder. The final one is for pictures and they belong in the ICON-SUPPLY folder within. For consistency, the Text-Report file keeps the status of all unfinished Loop… files.

Conclusion

We introduced both Stored Procs and Guided Form Navigation. More resources are needed to complete this project including data-forms which require CSS, HTML and JavaScript. Beyond this, we need a context for client-side graphics, such as Canvas or even SVG.