CHAPTER

Nine

A Reporting Tool Using Programming by Example for Format Designation

TETSUYA MASUISHI
Hitachi, Ltd.

NOBUO TAKAHASHI
Hitachi, Ltd.

S
R
L

ABSTRACT

This chapter describes a report tool in which report formats are designated by PBE-like (programming by example) operations. Users specify a sample layout of an example row of relational table data on a sheet and select an iteration pattern of the sample layout. The tool extracts a set of general formatting rules from the sample layout. The rules consist of absolute positions of noniterative data, relative positions of iterative data, the iteration pattern, and the increment of the iteration. The tool interprets the rules and generates new reports of the format for different table data.

9.1 Introduction

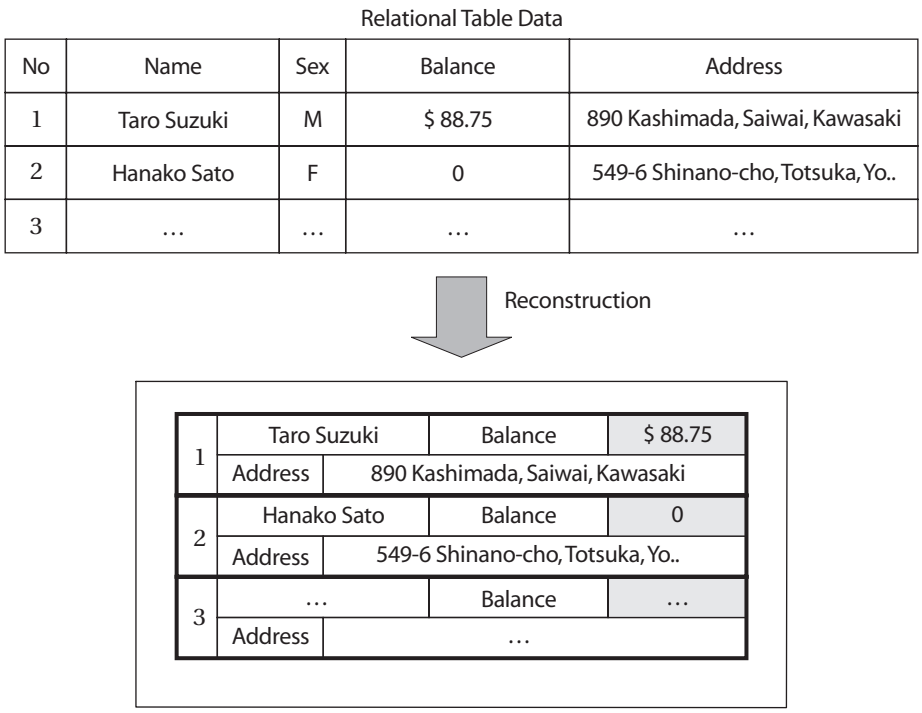
Data warehouse technologies have enabled centralized data management for decision support and planning applications. The decision makers print various formats of reports based on the data of the centralized database. Many end users, sometimes decision makers themselves, have to design the report formats, since the formats represent the view of the data and cause considerable effects on the decision. Such reports for decision making in Japan are reconstructed table-style reports that include instance text strings and numerical data, while most of such reports in the states include graphs that already imply analysis of the results.

The reconstruction is needed because relational tables in practical use usually have many columns. (We have assumed the data are stored in a relational database.) The reconstruction is applied so that several columns of data can fit within the width of the paper. It is difficult to grasp reports that exceed both of the width and the height of the paper. Since the number of rows of tables is usually big enough to exceed the height of the paper, we need to reconstruct the relational table to make reports of many pages flow in one direction that we can handle easily. Figure 9.1 shows an example of a Japanese-style reconstructed table report. Japanese reports usually include many ruled lines to separate adjacent data, rather than tabulation spaces, perhaps because Japanese words are not separated by spaces.

Many software packages called *reporting tools* support such reconstruction. Reporting tools usually have a scripting language to make programs. A program generates a report of a fixed format by embedding various data of a relational database.

S
R
L

FIGURE **9.1**



Typical Japanese reconstructed report.

This chapter describes the user interface of a report tool that designates formats of reports. The user interface is designed in a programming-by-example (PBE) manner (Cypher 1993; Myers 1992). A user creates a sample report for example table data. The tool extracts the implied formatting rules. The tool interprets the rules to generate reports reading relational table data as input. The extraction process is deterministic and does not include any statistical recognition or learning process.

Sugiura and Koseki (1996) show a data entry system from email text into a database. The system generates a macro program that reads email text, extracts data from the text, and inserts the data into database. A generalization process is employed to generate a general–purpose macro program from a history of sample operations. Our problem is easier because the input is a structured table in a database, not email text in natural language.

Instead, our system employs a simple generalization process for just one example.

Myers (1991) uses just one example for text formatting. The generalization process includes parsing the example text special words (e.g., *Chapter, Section,* and *Appendix*), numbers, separators, and decorations (e.g., lines and boxes). The parsing process encounters ambiguity. Our problem is well structured enough to make the system deterministic and easy to use.

9.2 System Overview

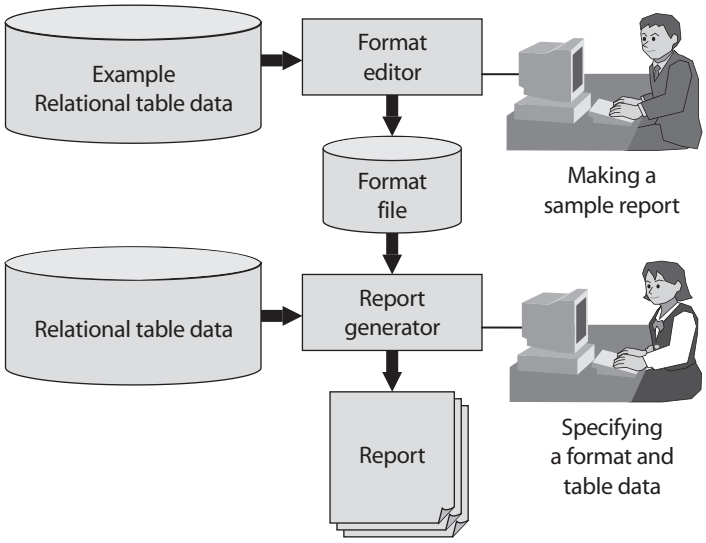
The reporting tool is used in two phases, the (1) format-making phase and the (2) report-generating phase. The two phases can be done by different users, and some users may do just the latter phase.

- *Making formats:* In this phase, a user makes a format, not a report. Conventional reporting tools do not require table data for making formats, but this tool requires table data to make a sample report. (Users always need table data even for the usual tools used when they "debug" their formats.) The tool extracts a set of formatting rules and stores them in a persistent file called a *format file.*
- *Generating reports:* Other users generate reports by specifying a format file and table data. Table data can be specified by a file name if the data are stored in a file such as in CSV (Comma Separated Value) file format or a set of retrieval statements such as select statements in SQL (Structured Query Language) for a connected relational database.

9.2.1 System Configuration

The tool consists of the Format Editor, which enable users to edit interactively to make format files, and the Report Generator, which allows users to generate reports by specifying a format file and relational table data (Fig. 9.2). The Format Editor is an interactive editor for a user to make a sample report using a set of relational table data, extract a set of formatting rules, and save them into a format file. (A format file is a conventional file that includes a set of formatting rules in our proprietary file format.) The editor reads a format file and example table data.

FIGURE **9.2**



System configuration.

9.3 User Interface of Format Editor

We have employed a PBE-like user interface for the Format Editor. A user makes a sample report to make formatting rules, without needing to specify general formatting rules directly. The set of rules works as a program that generates reports for other sets of relational table data. So, the process of the format editor is a kind of programming by example. Since some of the rules are specified directly by the user for practical use, the process is a simple and practical version of PBE.

9.3.1 Window Configuration

The Format Editor consists of two main windows: the *Sheet* window and the *Table* Data window. The *Sheet* window shows the sample report as a sheet image. The *Table* Data window shows the example table data as a relational table image. The user can copy and paste table data (both numerical and string) to the *Sheet* window.

The editing scheme of the *Sheet* window is as follows:

1. Put objects on the *Sheet* window.
2. Specify iterative objects, called a *block*.
3. Specify iteration pattern for the block.
4. Specify increment of the iteration.

The objects on the *Sheet* window consist of a fixed text string, including column names; functions; ruled lines; and example data. Any text strings can be put on the *Sheet* window, and usual text string attributes such as fonts can be specified. The user can also copy and paste useful column names from the *Table* Data window to the *Sheet* window. Some built-in functions are provided (e.g., date, time, etc.); they work as variables. Ruled lines can be drawn on the *Sheet* window, and usual line attributes such as width and pattern can be specified.

When starting up the Format Editor, a default example row—the first row—is highlighted on the Table Data window. The user can change the example row by clicking the mouse button. Multiple example rows can be selected for special cases. For example, seven example rows are necessary to make a report showing weekly trends generated from daily table data.

9.3.2 Specifying Iteration

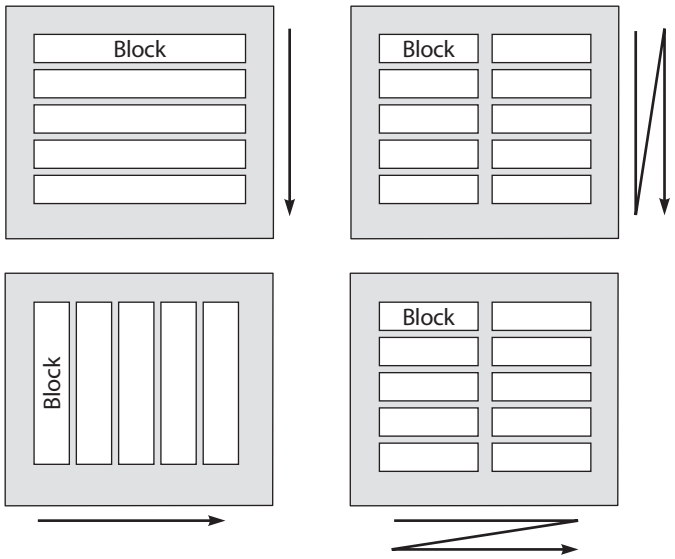
The system makes a report by iterating a print of a specified set of objects with a fixed increment of position. The user specifies which object is iterative or not. A rectangular area that surrounds the iterative objects is called a *block*. The user can specify iterative objects by specifying a block by a rectangular rubber band of the mouse.

The system also provides iteration patterns, some of which are shown in Figure 9.3. The user chooses a pattern from the menu of the provided patterns. The increment of position for the iteration can be specified in two ways: the user can specify the increment directly, or the system can calculate the increment when the user put in data other than those from the example row(s) (Fig. 9.4).

The *Sheet* window displays the final image of the sample report, which includes iterated images of the iterative objects. The iterated images are produced by the example table data of the corresponding rows, which are calculated by incrementing the row numbers of the specified example row(s) by the number of the example row(s), typically one.

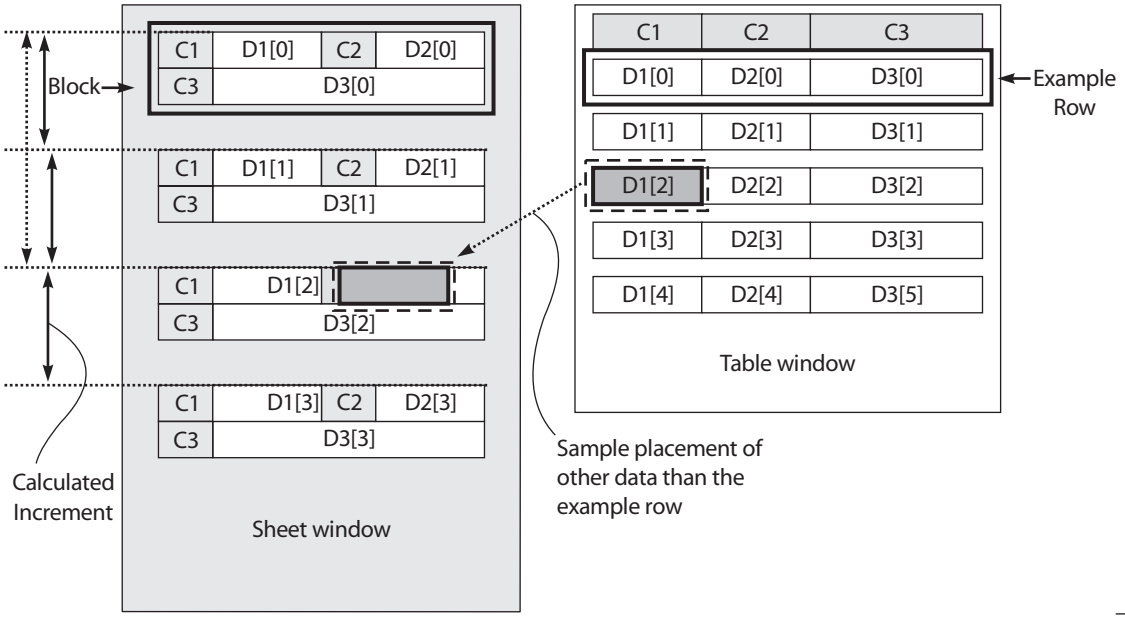
S
R
L

FIGURE **9.3**



Examples of iteration patterns.

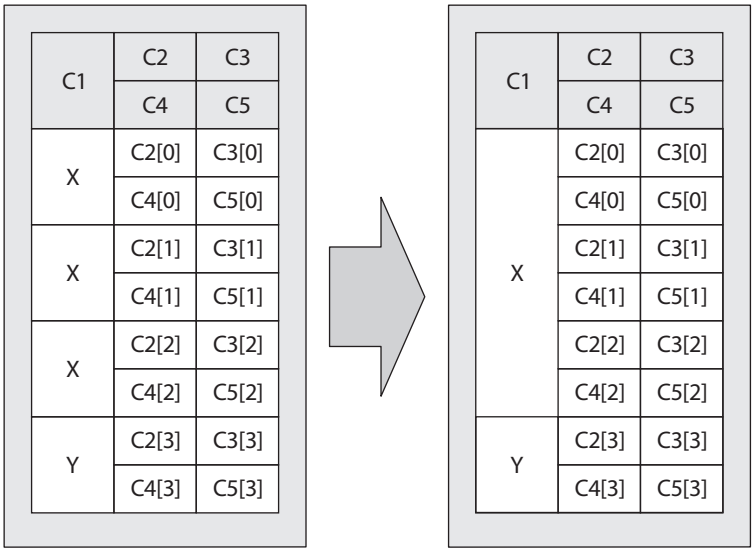
FIGURE **9.4**



Calculating the increment and the image of the sample report.

S
R
L

FIGURE 9.5



Unification of repeatedly occurring data.

9.3.4 Adjustment

After the image of the sample report is displayed, some adjustments can be made to create sophisticated reports. For example, conditions for page and column breaks can be specified. The typical condition is "When the data of the specified column change, create a new page." Also, when a column has repeatedly occurring data, they may be unified (Fig. 9.5).

9.4 Extracting Formatting Rules

The Format Editor stores a set of formatting rules for a sample report. Formatting rules consist of the following information:

- the list of objects on the sheet,
- information about the block, including its absolute position and size;

- the attributes of noniterative objects and their absolute position on the sheet;
- the attributes of iterative objects, including, for example data, (1) column name, (2) relative row displacement in the example table from the base of the example row, and (3) relative position in the sheet from the base position of the block, and, for other objects, (1) their attributes and (2) relative position in the sheet from the base position of the block (delta X and delta Y);
- the specified iteration pattern and the increment; and
- specified adjustments.

9.5 Generating Reports

When a format file and a set of table data are given to the Report Generator, it creates a report according to the algorithm, briefly described as follows:

1. Put the entire noniterative object on the sheet.
2. Iterate the following substep for all rows of the table data: Put all of the iterative objects at the current position, substituting the example data to the given table data at the current row of the iteration.
3. Apply any adjustments.

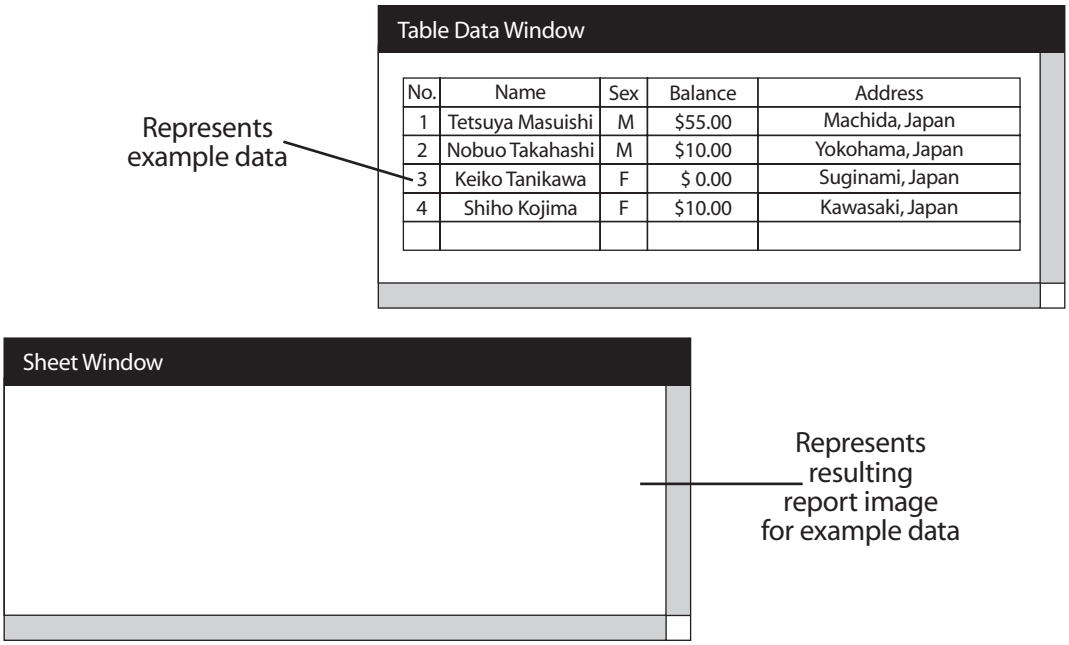
9.6 Example of the Process

We describe here an example of how the system works, starting with making a format. Figure 9.6 shows the window status of the Format Editor just after the user has specified an example data file. The *Table Data* window displays the example data as a table. The *Sheet* window displays an empty sheet on which the user is drawing an example report.

The user draws an example report on the sheet window by copying and pasting objects from the *Table Data* window to the *Sheet* window (Figs. 9.7

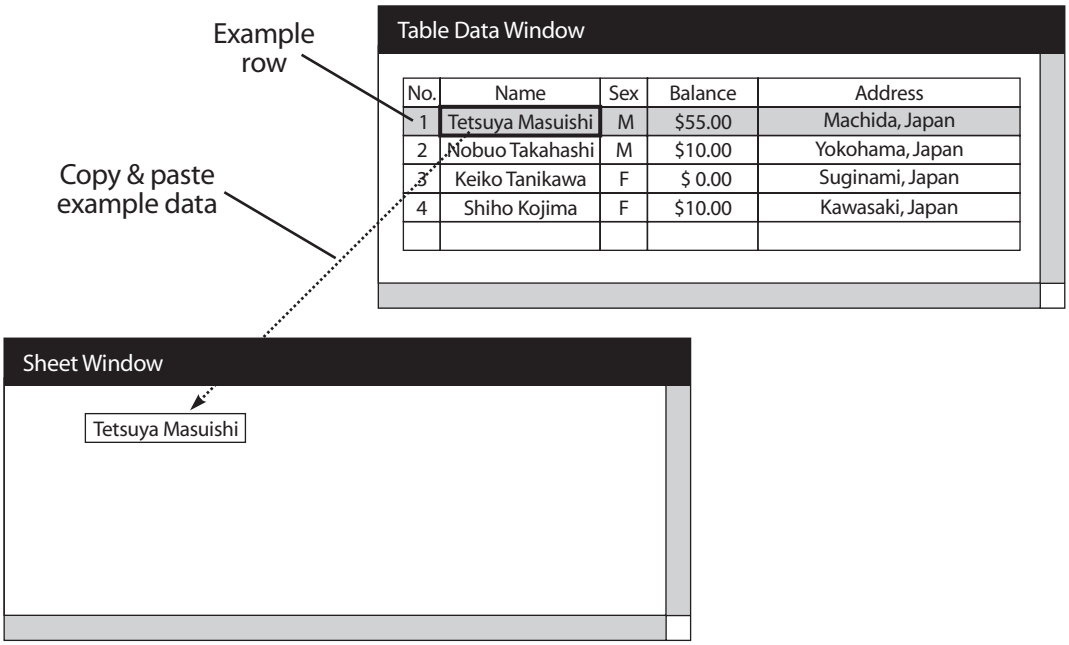
S
R
L

FIGURE 9.6



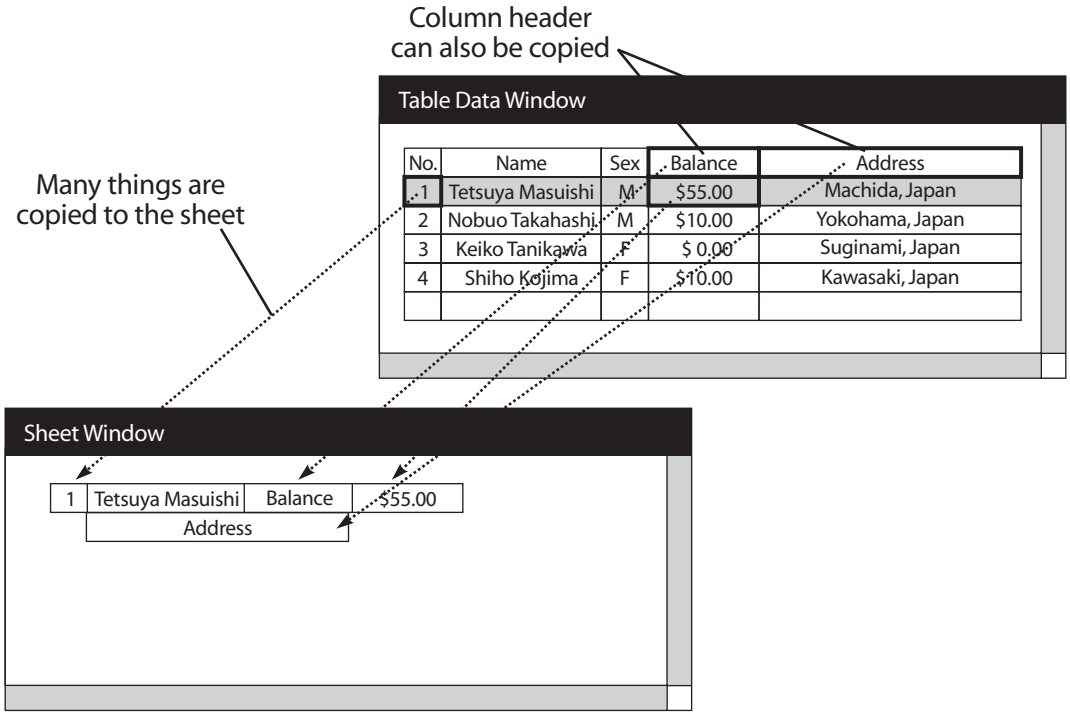
Format editor.

FIGURE 9.7



Copy and paste.

FIGURE **9.8**



Copy and paste (continued).

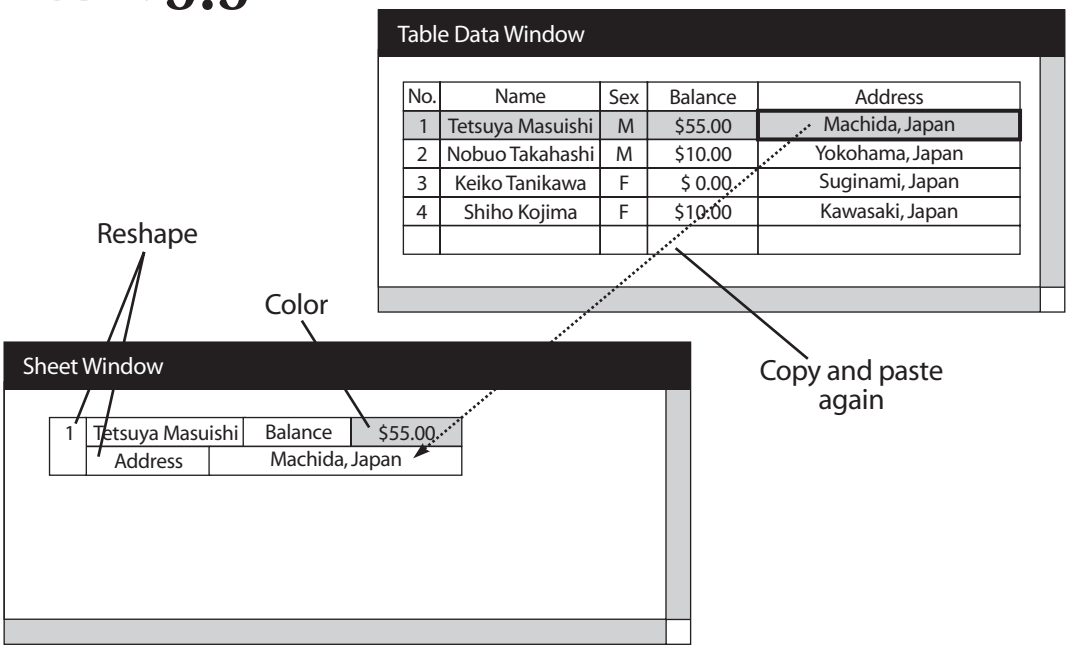
and 9.8). The objects include the table data of the example row and column header. The *Sheet* window works as a usual drawing program (Fig. 9.9). The user can put text strings directly from the keyboard and draw lines using the mouse.

When the user has finished editing the block, the next thing to do is specifying the iteration. The user changes the Format Editor to the iteration mode and specifies the block and the iteration displacement with the mouse (Fig. 9.10). He or she can imply the iteration by copy-and-pasting the other data from the example row (Fig. 9.11). This implication does not require that the user change the editor mode.

When the user sees the example report image on the *Sheet* window, the user saves the format to a file, as one usually does in drawing programs. The

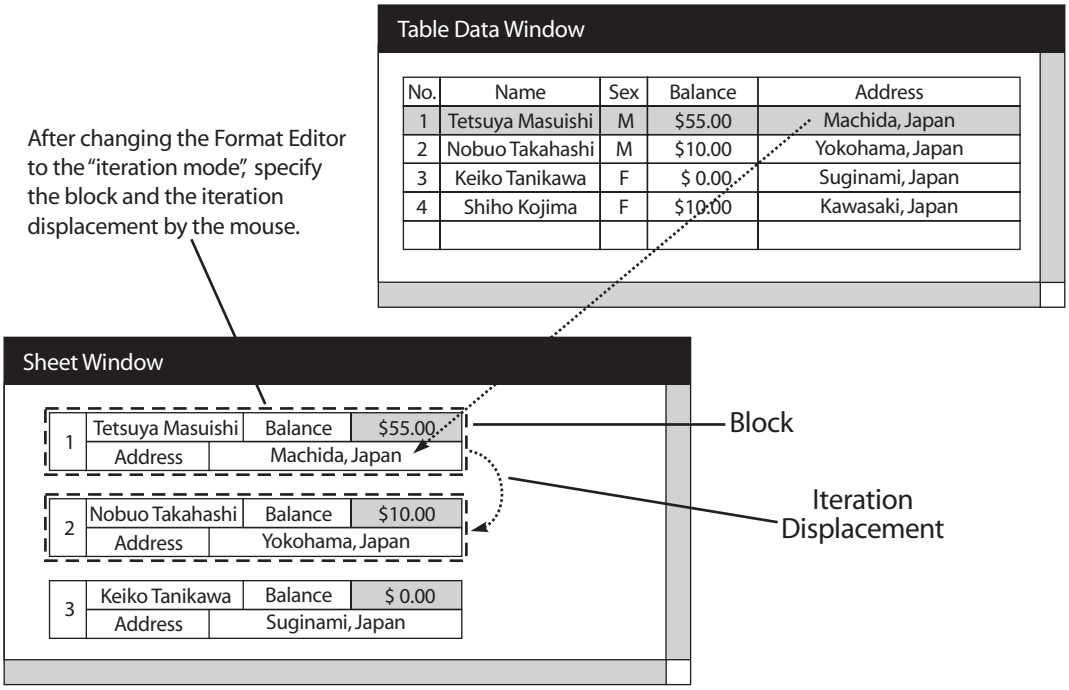
S
R
L

FIGURE **9.9**



Edit on the Sheet window.

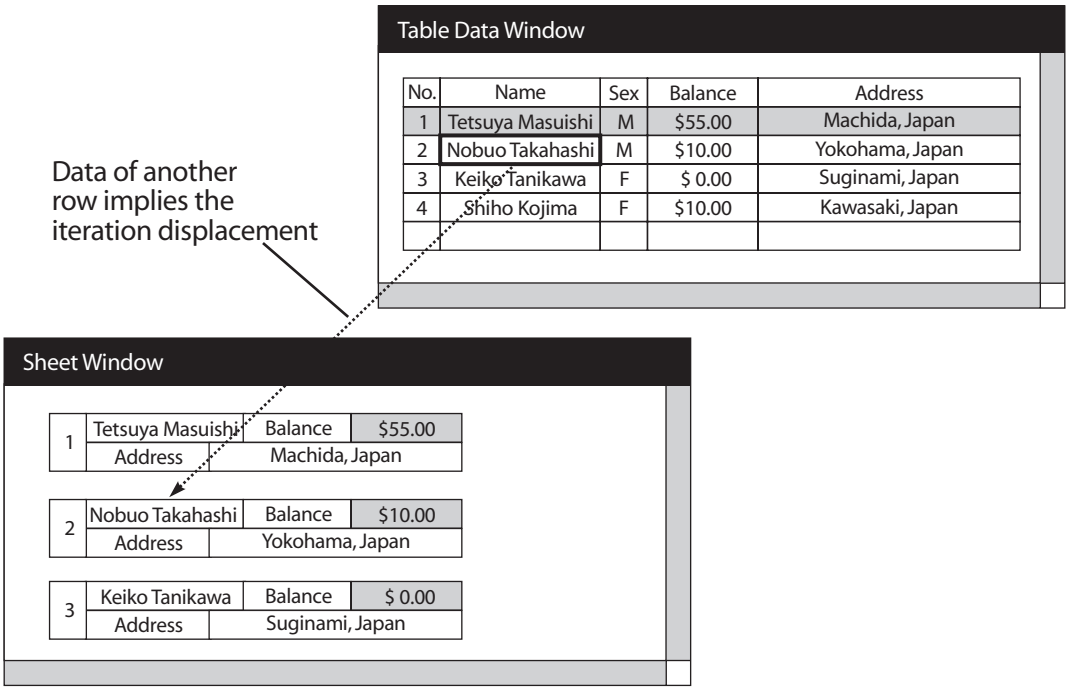
FIGURE **9.10**



S
R
L

Specifying iteration.

FIGURE **9.11**



Implication of iteration.

Format Editor extracts the format rules from the example report (Fig. 9.12) and save the rules to the file.

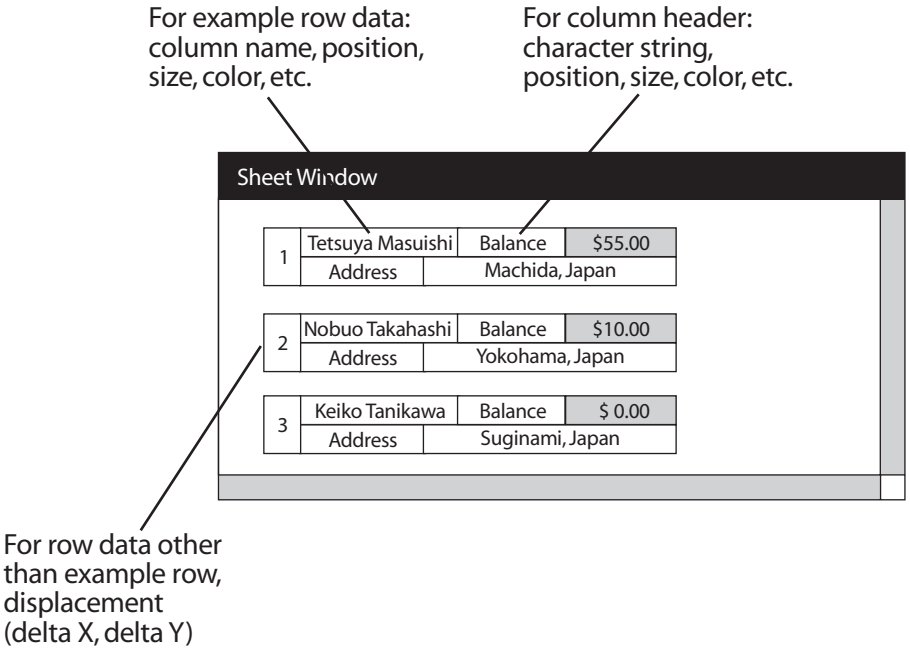
The extracted formatting rule works as a report generation program. Figure 9.13 shows a pseudo-expression of the formatting rules in C-like language. Another user can apply the formatting rules in the file to other data and generate another report (Fig. 9.14).

9.7 Evaluation

We have applied the system described here to a real project at a Japanese company. End users and information technology (IT) staff there have developed more than five hundred formats. The applied formats include the following:

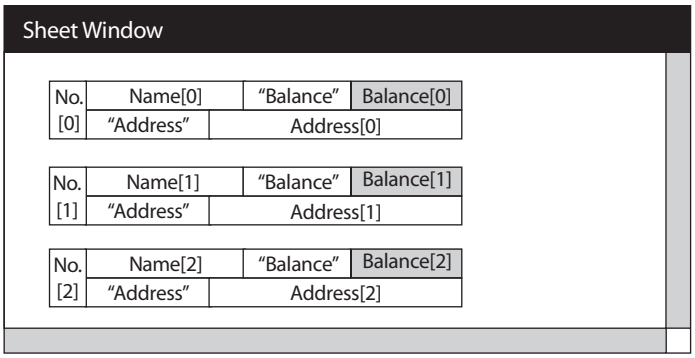
S
R
L

FIGURE 9.12



Extracted information.

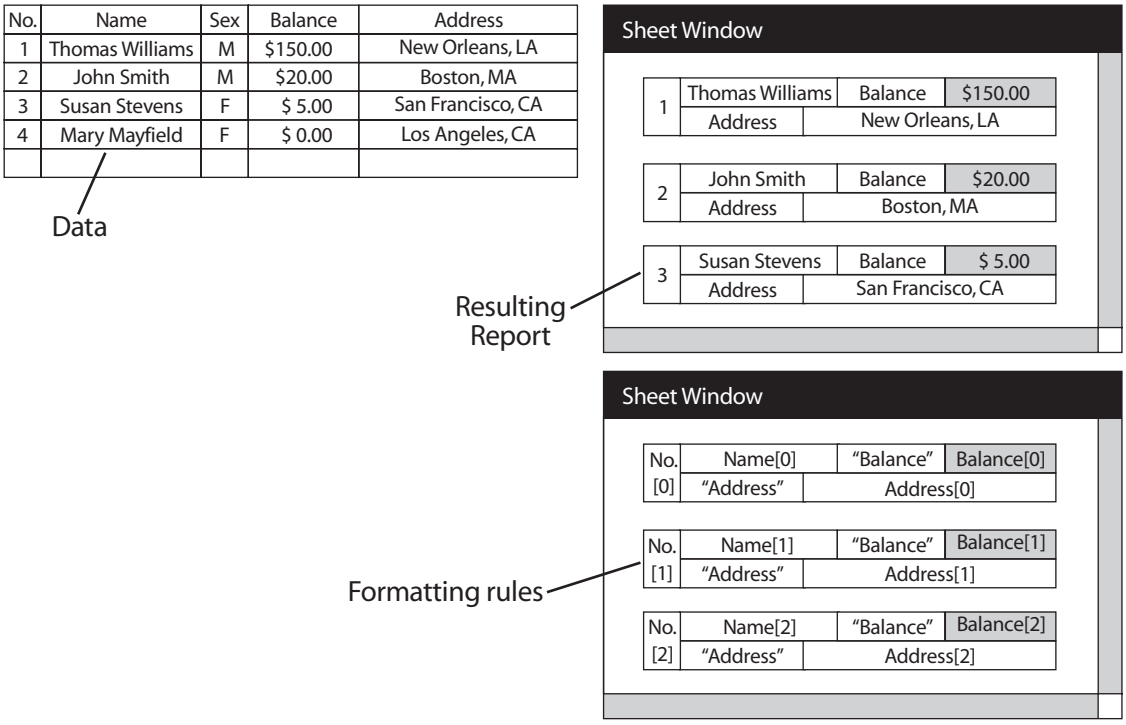
FIGURE 9.13



Pseudo-expression of formatting rules.

S
R
L

FIGURE **9.14**



Report generation for other data.

- *Reports for planning and decision making*—sales reports for sales persons and divisions, sales reports for products and areas, and cost reports for products
- *Mission-critical forms*—monthly reports to customers, bills, and in-house order forms
- *Images*—employee files including facial photo images and price lists with product photo images

We have been informed that staff could make all the formats using the Format Editor. They required using some functions for more detailed presentation, such as round corners for crossing ruled lines.

S
R
L

End users could make report formats as well as IT people–there was no difference between the two groups' products. End users, who usually does not make general programs, could also make report formats that generate many reports according to data. This result indicates that the PBE paradigm worked well at this evaluation site.

9.8 Conclusion

We have designed the extraction process to be deterministic for practical use. The system extracts formatting rules deterministically, not statistically. Just one sample report is needed for extracting general formatting rules. To make this possible, some generating information, such as iteration, can be specified directly. This design was possible because the problem of generating reports from relational table data is well structured.

References

Cypher, Allen, ed. 1993. *Watch what I do: Programming by demonstration.* Cambridge, Mass.: MIT Press.

Myers, Brad A. 1991. Text formatting by demonstration. In *Proceedings CHI.*

———. 1992. Demonstrational interfaces: A step beyond direct manipulation. *IEEE Computer* (August): 61–73.

Sugiura, A., and Y. Koseki. 1996. Simplifying macro definition in programming by demonstration. In *Proceedings UIST.*

S
R
L