Workshop on Databases and Software Engineering

A Table Presentation Language for Database Applications

Woei-Kae Chen[*,1], Ping-Hung Chen[**], and Wen-Tseng Huang[***]

[*]Department of Computer Science and Information Engineering,
National Taipei University of Technology, 1, Sec 3, Chung-Hsiao E. Rd., Taipei 106, Taiwan, R.O.C.;
[**]Computer Center, National Taipei University of Technology,
1, Sec 3, Chung-Hsiao E. Rd., Taipei 106, Taiwan, R.O.C.; and
[***]Department of Electronic Engineering, National Taipei University of Technology,
1, Sec 3, Chung-Hsiao E. Rd., Taipei 106, Taiwan, R.O.C.

Abstract

Database applications often utilize a large number of reports. There are tools, report generators, as well as programming-level supports that facilitate the generation of reports. However, these supports are restricted to simple predefined presentation formats with limited flexibility. In general, a report consists of one or more data sources, which are not necessarily stored in formats that are ready for presentation. Therefore programmers must write application programs to convert query results into suitable presentation formats. In particular, the aim is to place the right data into the right position so that the report is easy to understand. Since the results of a query are inherently tabular, by restricting the reports into tabular formats, the problem reduces to the transformation of data between tables. This paper studies how the position of one data is transferred to another. We characterize and formulate fundamental transformations into high-level transformation operations, including *normal*, *transpose*, *fill*, *combine*, and *matching*. We then propose a high-level Table Presentation Language (TPL) to realize these operations. TPL can be used to create sophisticated reports without the overhead of writing tedious low-level programs. Therefore the complexity of generating reports is greatly reduced. We have used TPL to construct several web-based database applications. In our experience, it is convincing that the language is very useful and practical for creating database query and report applications.

Keywords: Database, Report Generation, Table Presentation, Language, Web, CGI Generator

---

[1] To whom correspondence should be addressed. E-mail: wkc@csie.ntut.edu.tw Tel: 886-2-2771-2171 x 4230

1. Introduction

It can be argued that report generation is one of the most frequently performed tasks in database applications. A report is the results of queries, which are organized and formatted to satisfy a particular presentation requirement. To facilitate the generation of reports, there are tools (e.g., [2], [7], and [9]), report generators (e.g., [1] and [8]), and programming-level supports (e.g., function libraries) that can be used. However, these supports are restricted to simple predefined presentation formats with limited flexibility. For example report tools that offer printing of tables (the results of SQL queries) can easily be found, with the limitation that only fixed and regular formats can be used.

In general, a report usually consists of several data sources, which are not necessarily stored in formats that are ready for presentation. Therefore, before printing, programmers must write application programs to convert different data sources into suitable presentation formats for each report. In particular, the aim is to place the right data into the right position so that the report is easy to understand. Although these conversion programs are not difficult to write, they are very tedious and require the intervention of programmers.

As an example, Figure 1 shows the transformation from the information of reservations into a seat arrangement of a theater. In this example, there are two input tables (data sources): RESERVATIONS and PRICES. The RESERVATIONS table contains information for each reservation, e.g., the seat for Mr. A is reserved at row 1 and column 1. The PRICES table contains information for the price of the seats, e.g., a seat in row 1 costs 100 dollars. The report of this example is the table SEATS, which is a presentation that shows the actual layout of the theater, the price of each row, and the location that everyone sits.

For the generation of the SEATS table, there are two transformation operations to be done. The first is to transfer the information of each row of the RESERVATIONS table into the correct position of the SEATS table, which shows the layout with a two-dimensional presentation. The next is to transfer the price of each row from the PRICES table into corresponding position of the SEATS table. It is easy to see that writing programs to perform these transformations is not difficult. In fact, simple loops suffice to accomplish these operations. It is also possible to simplify coding efforts by using

libraries that supports these transformation operations. However, writing programs such as embedded C++ or 4GL (4th generation language) to perform these conversions is considered a low-level solution, which is very tedious and inflexible.

RESERVATIONS

Row	Column	Name
1	1	A
1	2	B
2	3	C
3	2	D
3	4	E
4	3	F
:	:	:

PRICES

Row	Price
1	100
2	200
3	300
4	250

SEATS

	1	2	3	4	Price
1	A	B			100
2			C		200
3		D		E	300
4			F		250

Figure 1. An example of transferring reservations into seats

This paper studies a restricted form of report, i.e., reports in tabular format. Using a tabular format to present a report is frequently used, because the data shown in a table is easy to understand, and can easily be used, analyzed, and compared. In fact, many non-tabular presentations can also be considered as tables without part or all of the borders. Since the results of an SQL query is inherently tabular, by restricting the target reports also into tabular formats, the problem of writing programs is reduced to the transformation of data between tables. In this paper, we study the problem of transferring data from one table to another with emphasis on how the positions of the data are rearranged. We characterize and formulate fundamental transformations into high-level transformation operations. These operations include *normal*, *transpose*, *fill*, *combine*, and *matching*. We show that by using combinations of these operations, reports of sophisticated formats can be created.

We propose a Table Presentation Language (called TPL) to support these operations. TPL is designed as a scripting language and is both high-level and concise. The language also supports several output formats for different applications. Users of TPL can generate sophisticated reports

without writing tedious low-level programs. Therefore the complexity of generating reports is greatly reduced.

TPL supports both client/server and web-based applications. For web-based applications, our TPL system can be considered as a CGI (Common Gateway Interface) generator. TPL has a strong instruction sets that can perform transformation operations, which makes TPL different form the other CGI generators (e.g., [2], [7], and [9]). That is TPL is capable of presenting reports of arbitrary format; the other tools cannot.

The rest of this paper is organized as follows. Section 2 describes researches and tools that are related to this paper. Section 3 discusses the transformation operations. Section 4 defines the TPL language. The implementation and applications of TPL is given in Section 5, and a conclusion is given in Section 6.

2. Related Works

There are researches studying the generation of reports. In particular, Tarassenko [8] proposed a "reporter" system that can produce sophisticated read-to-use reports. By using the reporter system, the development of a report is focusing at describing tables with column/rows, rules of data selection and summarization, and formatting the report in destination document. Chan [1] proposed a document-driven approach to report generation, which is more flexible than the traditional schema-driven approach. Contents of a report can be specified using a transformation language together with queries that retrieve data from different databases.

A natural application of the TPL language is a dynamic web-page generator for database applications. There are several possibilities in writing dynamically generated web pages. The most fundamental solution is the writing of CGI scripts. These are programs, when invoked, running on HTTP servers that dynamically construct web pages in HTML format. Scripting languages (e.g., Perl) are usually used to code CGI programs. However, for web-based database applications, CGI programs tend to be too low-level to code. Therefore, CGI generators, which automate or simplify the process of writing web-based database applications, become important.

There are researches and tools (e.g., [2][7] and [9]) address CGI generators for database applications. The fundamental insight of these researches is based on the fact that the content of a dynamic web page is the result of SQL queries with additional HTML formats. Therefore by substituting the SQL statement and HTML templates, different web pages can be created. For example, the Microsoft IDC (Internet Database Connectivity) [7] accompanied with IIS (Internet Information Server) use an "idc" file for SQL source and an "htx" file for HTML templates. The major disadvantage of these tools is that they are all limited to simple formats without flexibility. Although, they are good for certain fixed-format database applications, extensions are usually difficult or impossible. Therefore, as a replacement, using HTML-embedded language becomes the current trend.

With the support of HTTP servers, using HTML-embedded languages is usually more efficient than CGI scripts both in terms of execution and development time. As an example, using PHP (Hypertext Preprocessor) [3], an HTML-embedded scripting language, web developers can write dynamically generated web pages quickly. Since PHP is strong with database support, generating a database report by PHP scripts is also easy. However, using PHP exhibits the same problem as using C++ or 4GL discussed in the previous section. Other tools such as ASP (Active Server Page of Microsoft) [6] and JSP (Java Server Page) [5] also have the same problem.

3. Transformation Operations

This section discusses the transformation operations that can be performed between two tables. A transformation operation is composed of three major factors: (1) the source, (2) the target, and (3) the type of transformation to be performed. We will discuss these factors for each of these transformations.

An entry in a table is called a *cell*. The *position* of a cell is denoted as $C_x R_y$, where C_x is the column index and R_y is the row index of the cell. We use $C_0 R_0$ to indicate the cell that is located at the top-left corner of a table. A rectangular *block* of cells is denoted as $C_{x1} R_{y1} .. C_{x2} R_{y2}$. We use "$*$" to denote the last row or the last column of a table; an expression of $*-x$ is also allowed.

Figure 2 shows cells of six different shapes, including *single cell*, *single column*, *multiple columns*, *single row*, *multiple row*s, and *block*. Figure 3 gives the block expressions for each shape of the example shown in Figure 2. For simplicity, we also use simplified expressions, which are abbreviations of block expressions.

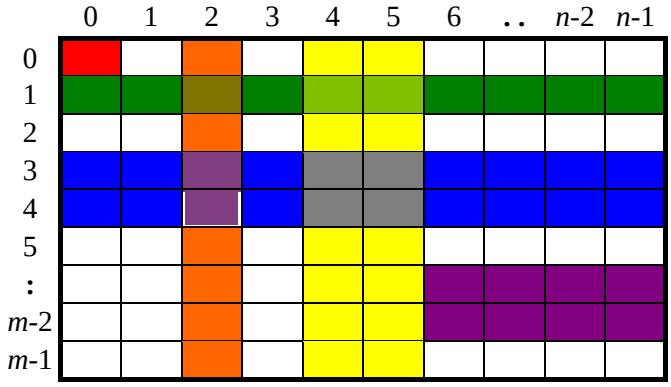


Figure 2. Cells with different shapes

Type	Block Expression	Simplified Expression
single cell	$C_0R_0..C_0R_0$	C_0R_0
single column	$C_2R_0..C_2R_*$	C_2
multiple column	$C_4R_0..C_5R_*$	$C_4..C_5$
single row	$C_0R_1..C_*R_1$	R_1
multiple row	$C_0R_3..C_*R_4$	$R_3..R_4$
block	$C_6R_6..C_*R_{*-1}$	$C_6R_6..C_*R_{*-1}$
entire table	$C_0R_0..C_*R_*$	$C_0..C_*$ or $R_0..R_*$

Figure 3. Expressions for cells with different shapes shown in Figure 2

We now turn our attention to how cells are transferred from one position to another. In the following subsections, we discuss each of the five different transformations, namely *normal*, *transpose*, *fill*, *combine*, and *matching*.

3.1 Normal Transformation

The most fundamental transformation is to translate the position of a cell (or a block of cells) to another position. Figure 4 gives an example that translates $C_1R_1..C_4R_2$ to $C_1R_0..C_4R_1$. We use the notation $Source \xrightarrow{operation_{option}} Target$ to express a transformation, where *Source* indicates the cells to be transformed, *Target* indicates the target position, and $operation_{option}$ specifies the transformation operation to be performed with an optional *option*. For the operation of Figure 4, it can be expressed as $C_1R_1..C_4R_2 \xrightarrow{normal} C_1R_0$. Note that a normal transformation does not

change shapes. Therefore, we do not need to specify the entire range of the target cells. Only the target position, which indicates the top-left corner of the target block, must be specified.

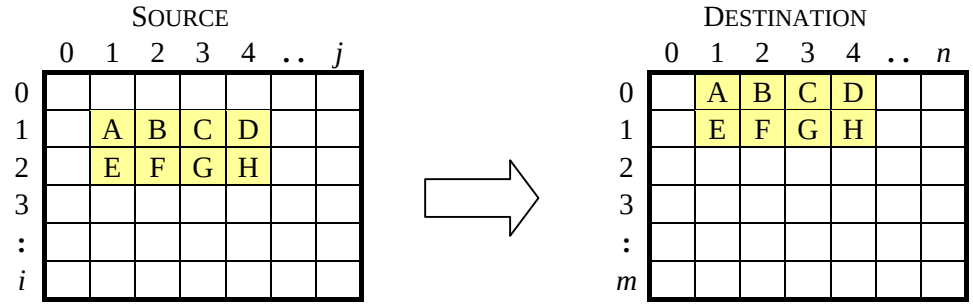


Figure 4. An example of normal transformation ($C_1R_1..C_4R_2 \xrightarrow{normal} C_1R_0$)

3.2 Transpose Transformation

A transpose transformation alters the position of a cell by interchanging the row and column indices of the cell. A block of cells can be transformed simultaneously. During a transpose transformation, a target position can be assigned, resulting a normal transformation at the same time. For example, Figure 5 performs a transpose transformation for $C_1R_1..C_4R_2$ with a target position C_2R_0. The result of the transformation is the block $C_2R_0..C_3R_3$. By using transpose transformations, vertical presentations can be switched to horizontal ones, and vice versa.

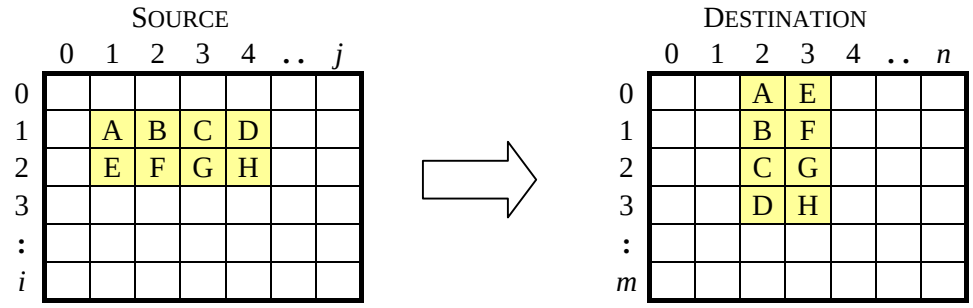


Figure 5. An example of a transpose transformation ($C_1R_1..C_4R_2 \xrightarrow{transpose} C_2R_0$)

3.3 Fill Transformation

The fill transformation is used to alter the presentation layout in order to fit certain height and/or width constraints. A fill transformation with row major ordering reads cells sequential form the source block and places the cells in a row-major ordering (zigzag) into the specified target position. For example, Figure 6 shows a fill transformation with row-major ordering and number of columns set as

3. Figure 7 is another example with column-major ordering. Note that the source cells are always processed in row major ordering. In case that a column major ordering for the source is required, a transpose transformation should be performed first.

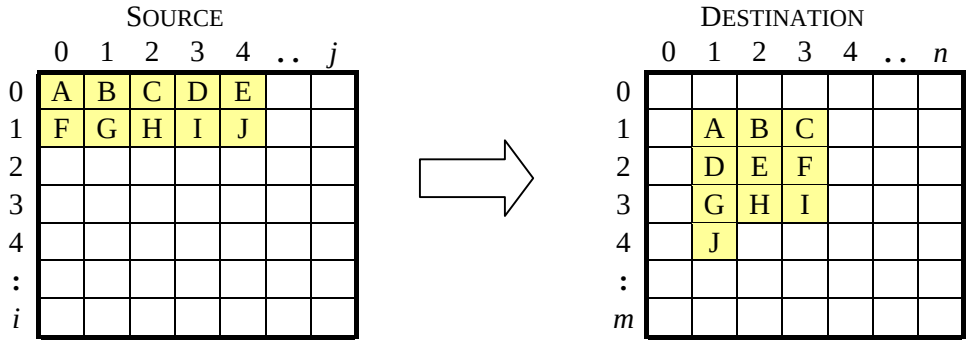


Figure 6. A fill transformation with row major ordering ($C_0R_0..C_4R_1 \xrightarrow{fill_{column=3}} C_1R_1$)

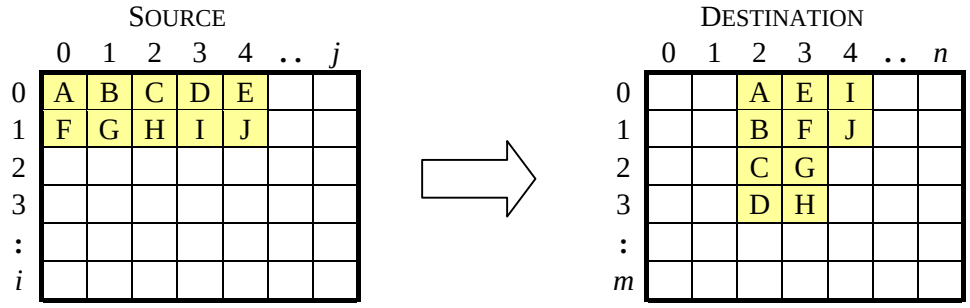


Figure 7. A fill transformation with column major ordering ($C_0R_0..C_4R_1 \xrightarrow{fill_{row=4}} C_2R_0$)

3.4 Combine Transformation

A combine transformation performs merging of cells. It can be used to merge two or more cells into a single cell, two or more columns into a single column, two or more rows into a single row, or a block into a cell. There are three kinds of merge options, namely *column combine*, *row combine*, and *column and row combine*. Figure 8, 9, and 10 show the result of combine transformation with different options. Note that for the column and row combine option, the cells of the first row in the source are combined first. This is because the source cells are considered as the result of database queries. Therefore the data cells of the same row contain information that is closely related. After a column and row combine transformation, the result becomes a single cell.

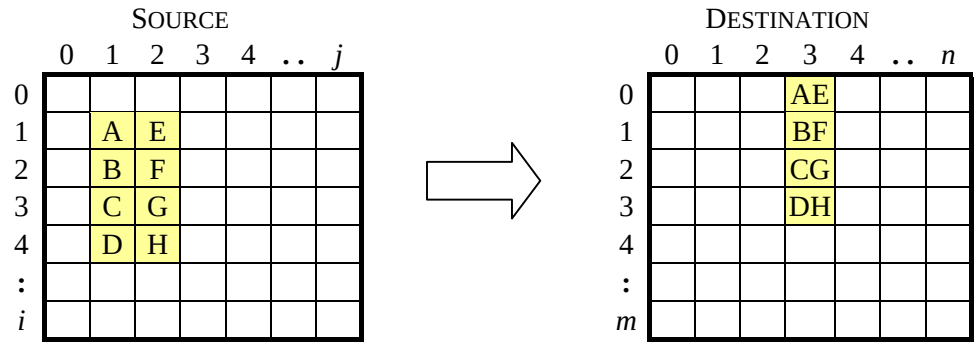


Figure 8. An example of a column-combine transformation ($C_1R_1..C_2R_4 \xrightarrow{combine_{column}} C_3R_0$)

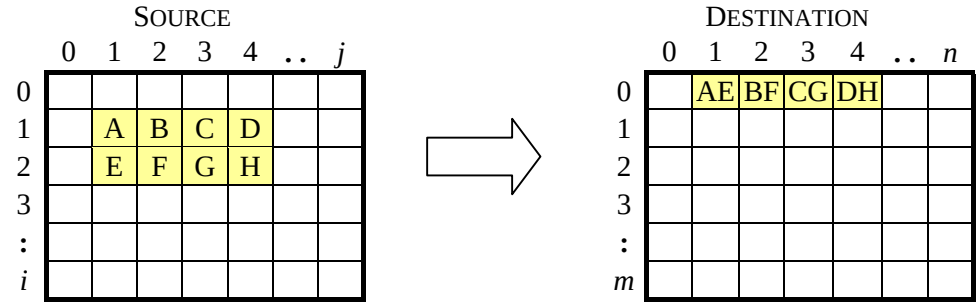


Figure 9. An example of a row-combine transformation ($C_1R_1..C_4R_2 \xrightarrow{combine_{row}} C_1R_0$)

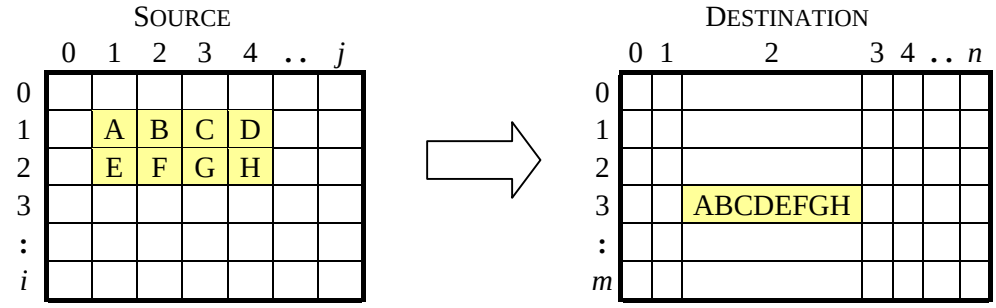


Figure 10. An example of a column and row combine transformation

($C_1R_1..C_2R_4 \xrightarrow{combine_{column,row}} C_2R_3$)

3.5 Matching Transformation

The matching transformation is a lot different from the other transformations described in the previous sections. For the normal, transpose, fill, and combine transformations, the final position of a cell depends only on the type of transformation to be performed and the target position that is assigned. For matching transformations, the final position of a cell, however, depends also on the content of the

source cells. For example, the seat of a person, shown in Figure 1, is determined by its row and column index of the RESERVATIONS table.

To perform a matching transformation, keys must be assigned for both the source and the destination tables. A matching operation transfers the cells of the source table with a certain key into the position of the destination table that matches the key. Figure 11 is an example of a row matching. In this example, the block $C_3R_0..C_4R_4$ is to be transferred from the SOURCE table to the DESTINATION table with C_1 of the SOURCE table and C_0 of the DESTINATION table assigned as keys (denoted as $C_1:C_0$). For row 0 of the SOURCE table, the key of the cells labeled *A* and *F* is 1. Therefore, the cells *A* and *F* are transferred to row 4 of the DESTINATION table where the keys match. Where does the key value of the destination tables come from? They can either be assigned manually or be transformed from the source tables.

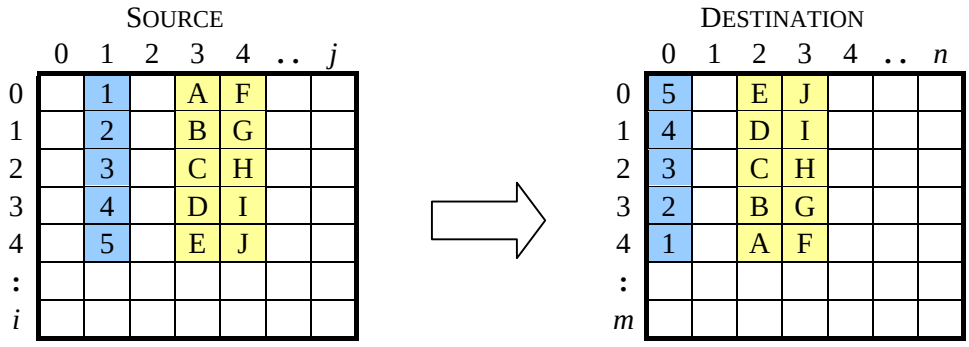


Figure 11. An example of using row-matching transformation ($C_3R_0..C_4R_4 \xrightarrow{matching_{c1:c0}} C_2$)

Note that the key of the source table must be expressed in C_x, where x is the column to be matched. The key of the destination table, on the other hand, can be either C_x, R_y, or both. The reason is that, for the source tables (data obtained from databases), the data cells of the same row constitute a record of information. Therefore, for the cells within the same row, it is natural to set one of the cells as the key and to transfer the other cells according to the key. Hence the key of the source table must always be a column. In case that the data cells of the source table must be processed in a column major format, a transpose translation should be performed first.

For the destination table, the key may either be a column or a row, which results in different matching strategies. Therefore, the keys of a matching may either be $C_x:C_y$ or $C_x:R_y$, which

are called *row matching* and *column matching* respectively. It is possible to perform both column matching and row matching simultaneously. Figure 12 gives an example. Note that, since both the column and row indices of a target cell are determined by matching, no target position should be specified.

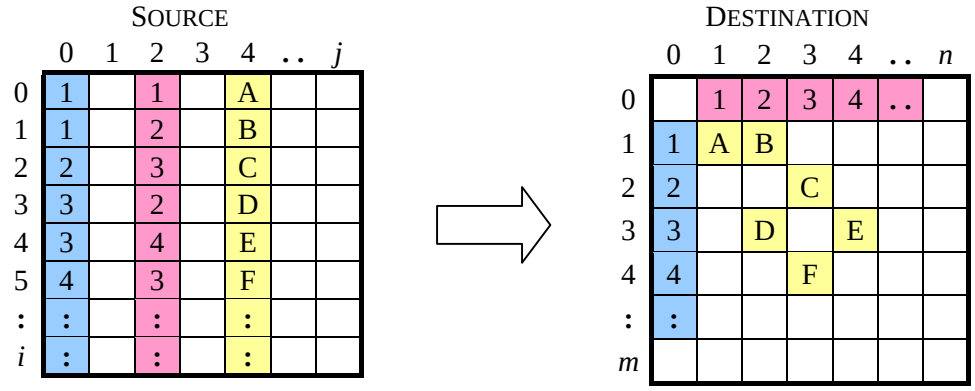


Figure 12. An example of using column and row matching ($C_4R_0..C_4R_* \xrightarrow{matching_{C_0:C_0,C_2:R_0}}$)

After matching transformation, it is possible that two or more cells end up with the same target position. These cells are combined as the previous section described.

3.6 Replace, Append, and Separator

In general, to generate a report, many transfer operations must be performed. Therefore, it is possible that the destination table is non-empty before a transfer operation is performed. If a source cell *s* is transferred to a target cell *t* that is non-empty, there are two possibilities to handle the situation: *replace* and *append*. The default behavior is the *replace* option, which simply replace *t* with *s*. An *append* option, on the other hand, replaces *t* with the concatenation of *s* and *t*.

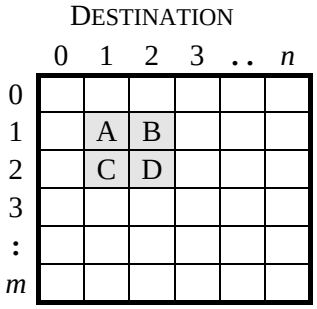


Figure 13. A non-empty destination table

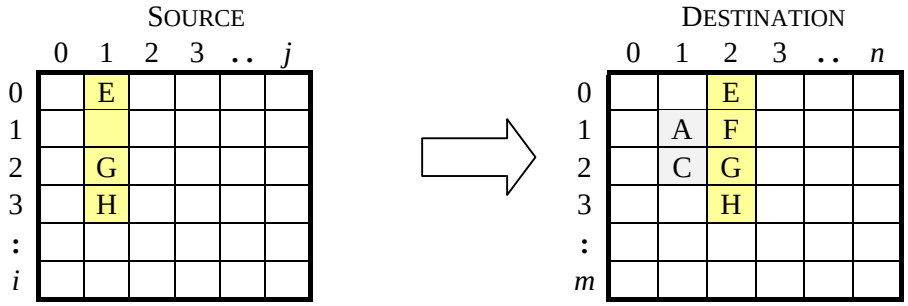


Figure 14. An example of using a replace option ($C_1R_0..C_1R_3 \xrightarrow{normal} C_2R_0$)

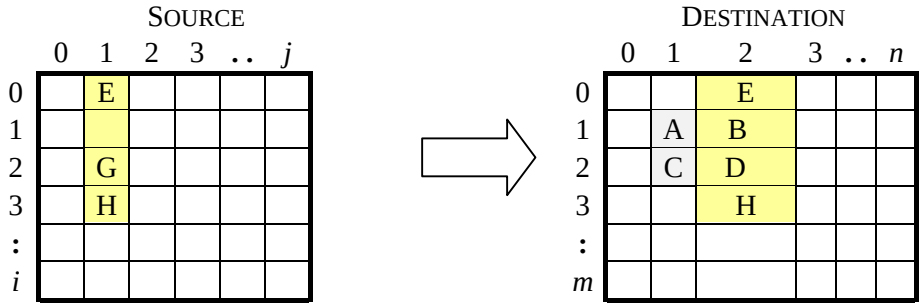


Figure 15. An example of using an append option ($C_1R_0..C_1R_3 \xrightarrow{normal_{append}} C_2R_0$)

Figure 13 shows a non-empty DESTINATION table. In Figure 14, a block of cells is transferred from the SOURCE table into the DESTINATION table shown in 13. It can be seen that the cells labeled *F* and *G* replaces the original cells. In Figure 15, the same operation is performed with an append option, which results concatenation of *B* and *F*, and *D* and *G*.

Sometimes, it is useful to concatenate related data in a single cell, but leave them distinguishable for later use. For example, a report may print out a cell that contain two values with a line break to distinguish the two values. In this paper, we propose the concept of using *separators* to resolve this problem. A *separator* is a special tag that is inserted between two values that are concatenated. If the separator option is enabled, a separator is inserted automatically when a concatenate operation is performed.

Figure 16 is an example that specifies the separator option. In this figure, we use the "•" notation to denote a separator, which is supposedly invisible. TPL is designed to recognize the existence of separators. In practice, users of TPL may replace the separators into meaningful notations or control characters before the resulting table is outputted. It should be noted that the

append and separator options also apply to other transfer operations, such as combine and matching.

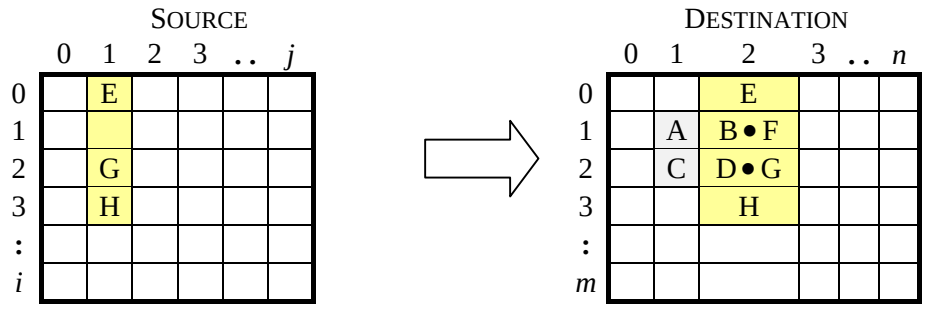


Figure 16. An example of using both append and separator options

$$(C_1R_0..C_1R_3 \xrightarrow{normal_{append,separator}} C_2R_0)$$

3.7 Cooperation of Transformation Operations

It is possible to perform two transformation operations simultaneously. However, not every pair of operations can be used together. To avoid ambiguity, we have to define whether a certain combination is allowed so that the semantics of the operations makes sense. Figure 17 gives whether one operation is allowed to cooperate with another. In the Figure, a "Yes" denotes that the cooperation of two operations is allowed. It is easy to see that only combine and matching can be executed simultaneously. The other pair of operations does not produce meaningful results.

	normal	transpose	fill	combine	matching
normal		No	No	No	No
transpose	No		No	No	No
fill	No	No		No	No
combine	No	No	No		Yes
matching	No	No	No	Yes	

Figure 17. The cooperation of transformation operations

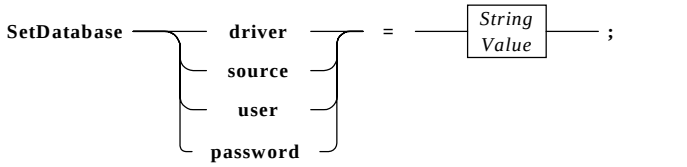
4. The Table Presentation Language

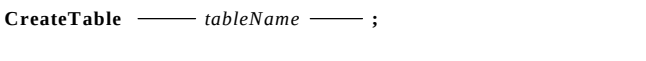
In this paper, we propose a table presentation language (called TPL) to realize the transformation operations described in section 3, and to support various input and output formats. TPL is deliberately designed to be as simple as possible. As shown in Figure 18, there are only 10 instructions, which are classified as the *input*, *transfer*, and *output* operations. A typical TPL script applies the input instructions to obtain information from databases, processes inputs with the transfer operations, and then generates reports (outputs) with the output instructions.

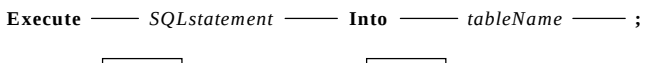
Group	Instruction	Usage
Input	SetDatabase	Assign database connectivity
	CreateTable	Create a table
	Execute	Execute an SQL
	Set	Assign the content of cells
Transfer	ClearTable	Clean up the content of a table
	Copy	Perform transfer operations
	ChangeSeparator	Replace separator with strings
Output	SetHTML	Assign output table in HTML formats
	Output	Output a table
	Print	Print a string

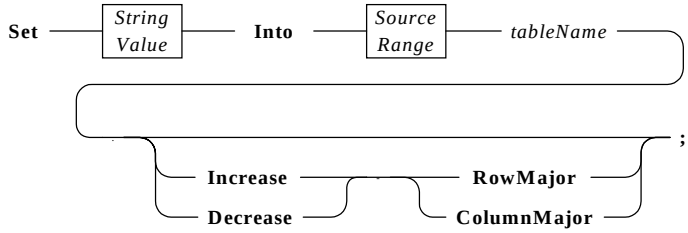
Figure 18. TPL instruction set

TPL is designed as a scripting language. Except for tables, no declaration of identifiers is required. The language itself is case insensitive. The following grammar diagram defines TPL. The *copy* instruction, which performs transfer operations as described in Section 3, is probably the most complicated instruction. As shown in the *output* instruction, currently TPL supports three different output formats: text, CSV, and HTML. The other instructions are self-explanatory; we will skip the explanations for each instruction.

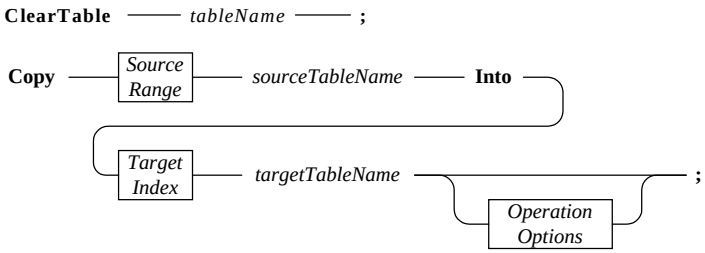


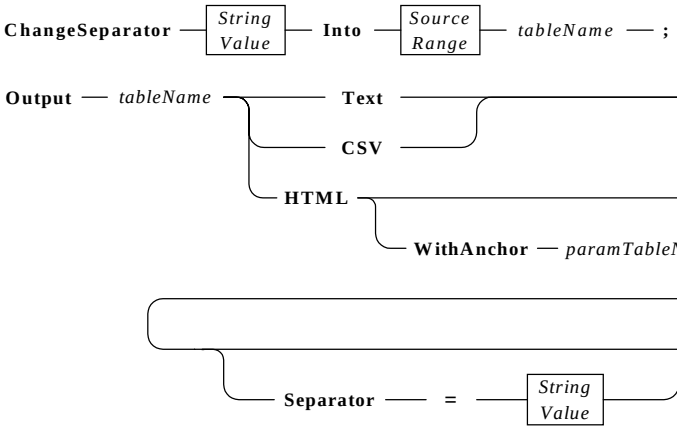


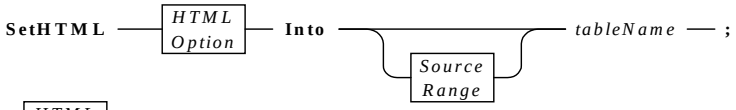


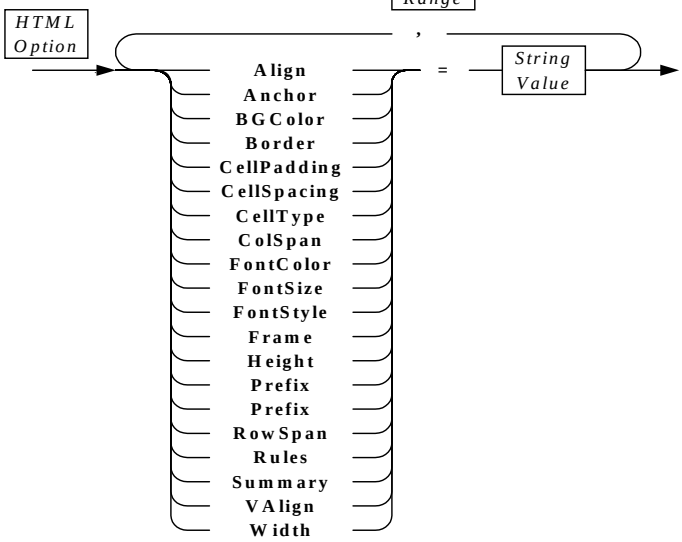


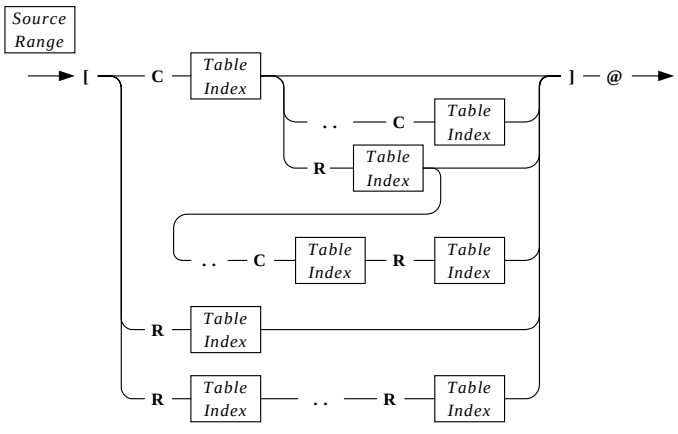
Note : The *String Value* must can be converted to integer while using **Increase** or **Decrease** option.

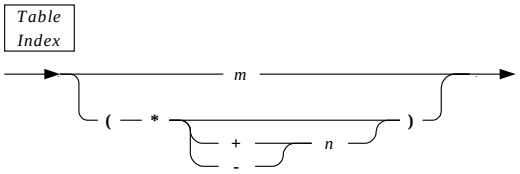




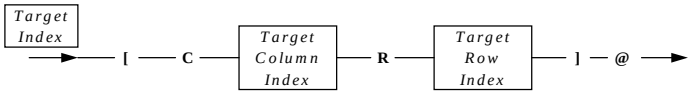


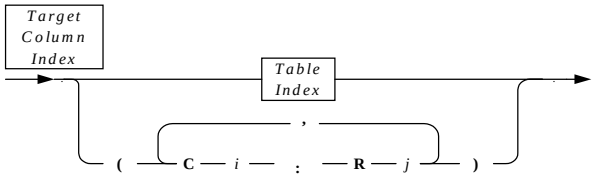




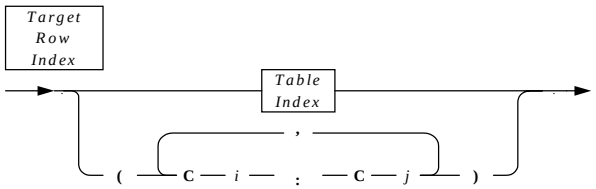


m, *n* : integer, *m* >= 0, *n* >= 0.

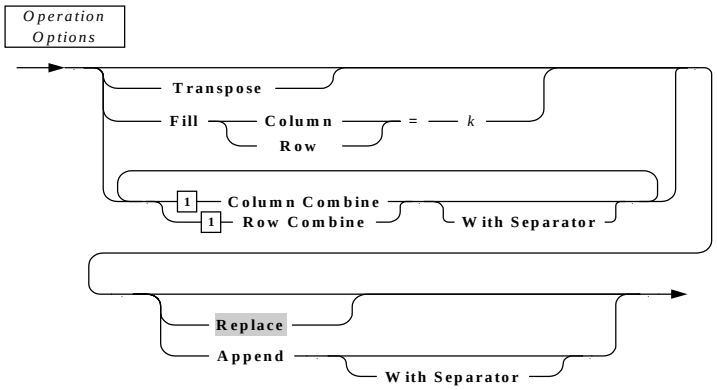




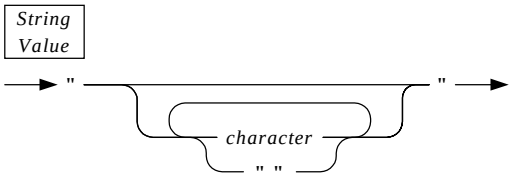
i : Column index of Source table, integer, *i* >= 0
j : Row index of Target table, integer, *j* >= 0



i : Column index of Source table, integer, *i* >= 0
j : Column index of Target table, integer, *j* >= 0



k : integer, *k* > 0.



Note : *character* is any ASCII character.

In addition to the ability of acquiring data from databases, TPL is capable of using parameters from its execution environments, e.g., command line arguments and environment variables. This is especially important for web-based applications, where a POST or a GET request stores parameters in standard input and environment variables. Typically, these parameters are used to customize part of the SQL queries in a TPL script so that different results can be generated with the same script. Our TPL system offers a mechanism called *parameter substitution*. A “foo” within a string indicates a request of using parameter substitution for the variable foo. That is all occurrences of “foo” is replaced with the actual value given by the execution environment.

5. The Implementation and Applications of TPL

The TPL system architecture is shown in Figure 19. We implemented a TPL interpreter (called TPLI) to process TPL instructions. The main input of TPLI is a TPL source file, which contains TPL instructions to obtain data sources from database servers and to perform transformation operations. The operation of the TPLI can be described as follows.

1. TPLI reads instructions from TPL source file.
2. TPLI creates tables $S_1, S_2, \ldots, S_x$ according to the number of **Execute** instructions. TPLI then requests database servers to execute SQL and stores the results of these queries into corresponding tables.
3. TPLI sequentially executes the next TPL instruction of the input file to perform transformations, creating intermediate tables ($T_1, T_2, \ldots, T_y$) and/or output tables ($O_1, O_2, \ldots, O_z$), if necessary.
4. Repeat step 3 until all instructions are executed.

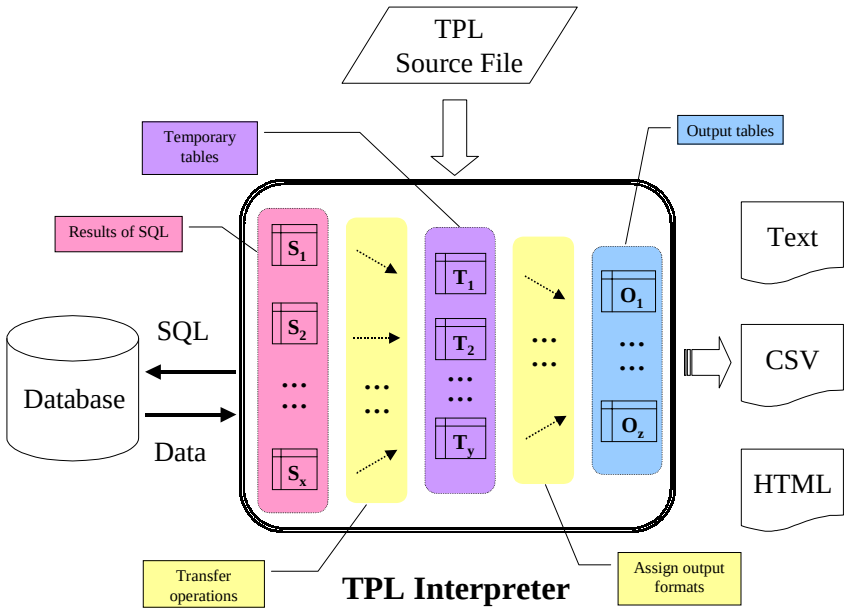


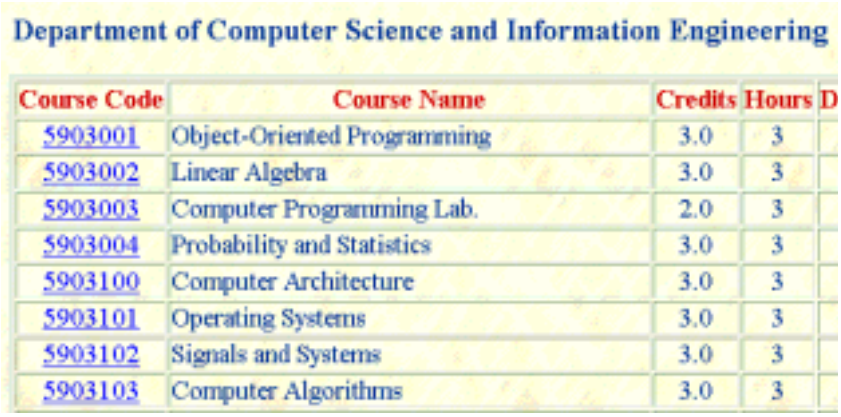
Figure 19. TPL system architecture

Several implementation issues were considered. We decide to implement an interpreter instead of a compiler. Without the overhead of compilation, a large amount of time developing applications

can be saved. It is also easier to use and to debug. The data of each data cell is considered type less; strings are used as internal representations for all data cells. We use Java to implement TPLI, because Java is platform independent and, with the aid of JDBC [4], it is easy to implement database operations.

One of the applications of TPL is the web-based query application. By using TPL scripts, dynamic web pages can be generated on the fly by TPLI. For example, TPLI can be used as a CGI program. An URL of the form "http://machine.domain/cgi-bin/tpl?filename=tpl_script" indicates a request to run the "tpl_script" script by TPLI (located at /cgi-bin/tpl).

In order to demonstrate the use of TPL for web-based applications, we have written a few applications at "http://dbs.cc.ntut.edu.tw/~cbh/TPL.html" (note: to fit into our system, our URL format for TPLI is slightly different from the one described above). This is a replacement to "http://dbs.cc.ntut.edu.tw/~dbquery/netoffice/course.html," which was written in Informix 4GL. A screenshot of an example web page is shown in Figure 20. It can be seen that TPL can produce results that are almost the same as those of 4GL, with much less coding efforts.



Department of Computer Science and Information Engineering

Course Code	Course Name	Credits	Hours	D
5903001	Object-Oriented Programming	3.0	3	
5903002	Linear Algebra	3.0	3	
5903003	Computer Programming Lab.	2.0	3	
5903004	Probability and Statistics	3.0	3	
5903100	Computer Architecture	3.0	3	
5903101	Operating Systems	3.0	3	
5903102	Signals and Systems	3.0	3	
5903103	Computer Algorithms	3.0	3	

Figure 20. A web page produced by TPL scripts

6. Conclusions

We proposed the concept of using transformation operations to generate reports. We showed that, by using combinations of a small set of transfer operations, sophisticated reports could be created. Previous works such as [1] and [8] focus on the modeling of report generation. Our results contribute to the data selection process of [8] and transformation language of [1].

We defined a simple and high-level scripting language TPL to support transfer operations. We

have also implemented a TPL interpreter in Java. Our implementation supports both client/server and web-based applications. The web-based support can be used as dynamic web-page generators for database applications. In comparison with popular web-based solutions for databases such as [3], [5], and [6], using TPL has the advantage of being simple and high-level. Although, TPL is not as flexible, the reduction of developing time is significant. In comparison with CGI generators such as [2], [7], and [9], TPL is not only more high-level, but also much more powerful in presentation capability.

We have applied TPL scripts for a few online applications. Examples can be found in the course description of "http://www.ntut.edu.tw/english/newpage6.htm." According to our experience, writing TPL scripts is indeed much easier and more efficient than writing dedicated programs. There are limitations to TPL. For example, the output of TPL in text or CSV format is not ready for printing. It must be post-processed by spreadsheet applications for printouts. We propose the following improvements for future studies:

- A visual environment for the development of TPL scripts.
- Integrate TPL with JSP for performance acceleration.
- Make an HTML-embedded version of TPL.
- Enhance TPL to support other output formats such as XML and WML.
- Enhance TPL to support high-quality printouts directly.

7. References

1. Chan D.K.C., Database and Expert Systems Applications, Proceedings. Ninth International Workshop, pp.925-930, 1998.
2. Hunter A., Ferguson R.I., Hedges S., SWOOP: An Application Generator for ORACLE/WWW Systems, The fourth International World Wide Web Conference, pp.234-242, December 11-14, 1995.
3. Hypertext Preprocessor, http://www.php.net/.
4. Java JDBC, http://java.sun.com/products/jdbc.
5. Java Server Pages, http://java.sun.com/products/jsp.
6. Microsoft Active Server Pages, http://msdn.microsoft.com/asp/.
7. Microsoft Internet Database Connector, http://msdn.microsoft.com/library/default.asp?url=/library/en-us/iisref/html/psdk/asp/idcg0tgy.asp
8. Tarassenko P.F., Bukharova M.S., System for database reports generating, Science and Technology, KORUS '01. Proceedings. The Fifth Russian-Korean International Symposium, pp.84-88 vol.1, 2001.
9. WebDBC, http://www.ndev.com/index.php.