Constructing Data Warehouses Based on Operational Metadata-driven Builder Pattern

Huanhuan Wang
College of Computer and Information Technology
China Three Gorges University
Yichang, China
Corresponding author: tinawang_053101@126.com

Jingcan Zhang
College of Computer and Information Technology
China Three Gorges University
Yichang, China
372924558@qq.com

Jingyi Guo
College of Computer and Information Technology
China Three Gorges University
Yichang, China
gjyxy168@gmail.com

***Abstract*—At present methods for constructing data warehouses are almost "case-to-case" pattern. This study illustrates the "case-to-case" pattern for ETL process in the course of the data warehouse construction with an example and shows that its biggest shortcoming is lower efficiency. The "builder pattern" is one of design patterns in the area of software engineering. A method, called "operational metadata-driven builder pattern" for constructing data warehouses is put forward by this study, which takes generic knowledge and object-specific knowledge apart from each other. To design and develop ETL process of data warehouse based on this method can eliminate the unnecessary repetitive operations under the "case-to-case" pattern.**

Keywords—data warehouse; ETL; design pattern; operational metadata

I. INTRODUCTION

In the era of big data, data warehouses and data mining are necessary for making decision, which has increasingly been becoming a focus of business community. Data warehouse is a "subject-oriented, integrated, time-variant nonvolatile collection of data" in enterprise management and decision-making [1]. Extraction-Transformation-Loading (ETL) is the first and important processes during constructing data warehouse, which are responsible for the data extraction from heterogeneous operational data sources, data transformation (from low-level, raw data to somewhat structured information, called conversion, cleaning, etc.) and getting data loaded into data warehouses [2-4]. The workload of ETL is so heavy that it is a big concern about how to improve its efficiency.

The existing researches on the ETL mainly concentrated on the following two aspects: modeling for the ETL [5-6], research on the ETL process [7-12]. There are some commercial ETL tools, such as Datastage from IBM, Powercenter from Informatica, Data Transformation Services (DTS) from Microsoft, ETL Automation of NCR Teradata, and some open-source tools, such as Kettle、 Talend、 CloverETL and so on [13]. But these ETL tools are almost good at traditional relational data warehouse. And because these tools are based on the high level operating system, some development tool and database management system, the efficiency is lower for constructing data warehouse with them. For constructing data warehouse based on the command line tool, for example SQL Plus of Oracle, and the SQL statements is better and efficient because they are close to the hardware of computer. But a ETL system like this is needed to be developed by ourselves instead of using commercial or open-source ETL tools.

It has been found that the ETL process based on the command line and SQL statements for constructing data warehouse is nearly case-to-case pattern. We can take an example to explain what the case-to-case pattern is. Assume that there are 100 tables needed to be deleted. We can use a SQL statement "DELETE FROM <database name>.<table name>" to delete one table. If we delete the other 99 tables, the above SQL statement needed to be copied 99 times and be modified 99 times. This is the case-to-case pattern. It is low efficient because there are lots of repetitions. Therefore, it is essential that the ETL pattern used for data warehouse is improved.

"Builder pattern" is a design idea in the area of software engineering. The construction of a complex object can be separated from its representation using it, said that the same construction process can create different representations [14]. Inspired by "builder pattern", recognition of both generic knowledge and object-specific knowledge and how to separate them are studied in this paper. And a new ETL pattern of data warehouse is presented in this paper. In the following content of this study, an example is introduced for demonstrating the advantage of constructing data warehouse with the "builder pattern" contracted to the case-to-case pattern.

This work is supported by 2014 Hubei Provincial Department of Education Fund (No. Q20141212).

978-1-4799-1891-1/15/$31.00 ©2015 IEEE

II. CONSTRUCTING DATA WAREHOUSES BASED ON CASE-TO-CASE PATTERN

The case-to-case pattern for constructing data warehouse is illustrated with an example. In this example, some column values in a table or a flat file are needed to be cleared during they are exacted from business application to data warehouse. Assume there is a table named "*Education program*" storing the education plan of all majors. In this table, there is a column named "*type*" storing the course period for every course. The data type of "*type*" is character string, and specific data type is related to the DBMS used. The course period of all theory courses is indicated as "units" (a unit is 45 minutes), for example "56", while the course period of all practical courses is indicated as "weeks", for example "1.5 W". Before the records in the table are exacted into the data warehouse, we want to unify the course period of theory and practical course, which means the "weeks" of practical courses are needed to be transformed to "units". The following SQL statement can be used to complete this task.

```
UPDATE Education program SET
type=STR(16*CAST(LEFT(type, CHAR INDEX('w', type)-1)
AS DECIMAL(4,1)))  WHERE RIGHT(RTRIM(type),1)='w'
```

But in the table, the course period of some records of practical courses is upper-case of "W". While the above SQL statement can only process the low-case of "w". This problem can be solved by getting low-case of "w" turned into upper-case of "W" in the condition part of the above SQL statement.

An education program file is likely to be edited by several managers, and both half-width and full-width character for the decimal point in "1.5w" are likely to be used in a table. Similarly such situation can be solved by modifying the condition part of the above statement again as followed.

```
update Education program set type=replace(type,'. ','.')
where char index('. ',type)>0.
```

But there are many majors in a university. If every major has its own education program flat file, similar operations could be repeated many times. And the other tables in the data warehouse also have the same problems with the "education program" table. Anyway there are too many similar situations and more complicated situations during the ETL process for constructing data warehouse. If we always modify the original SQL statement for each situation, we must repeat the following operation procedure many times: copy-paste-find-replace-verify-execute. Such a cleaning process for these column values is low efficient because the repeatable works take us too many unnecessary energies.

Through analyzing the above cleaning process for the column value carefully, it has been found that there are two types of repeat, content repeat and operation repetition.

Content repeat: key words in the SQL grammar.

Operation repeat: copy-paste-find-replace-verify-execute.

It is the above two types of repetition that result in the low efficiency of cleaning process.

III. OPERATIONAL METADATA-DRIVEN BUILDER PATTERN

A. Introduction of Builder Pattern

In the area of software engineering, the idea of "builder pattern" is to separate the construction of complex object from its representation so that the same construction process can create different representation. Fig.1 is a general UML diagram for "builder pattern".

B. Generic knowledge and object-specific knowledge

With "case-to-case" pattern for cleaning column values, the first thing standing out should be the repetitions. There are two kinds of repetitions here:

- Contents: The phrases *"UPDATE ***SET***WHERE***"* of the complete statements.
- Operations: The editing operations required to produce the subsequent lots of "UPDATE ***SET***WHERE***" phrases, i.e., "copy (the first statement)-paste (it lots of times)-search (the table names and file names lots of times)-replace (the old table name and file name with the new one lots of times)-adjust (if necessary lots of times)-verify (whether the resulting statement is correct lots of times)."

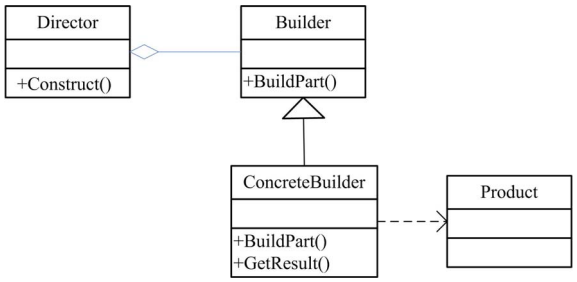


Fig.1 UML diagram for builder pattern

It is the generality of the *"UPDATE ***SET***WHERE***"* phrase for all similar situations due to the SQL-syntax, and perhaps due to the programming style convention that leads to the content repetition. Such repeated contents is considered as the carriers of the "generic knowledge." By contrast, in our example, it is the table names and flat file names that do not repeat at all. Each statement contains its unique specific table name or flat file name. Such things is called the carriers of the "object-specific knowledge." In our case, a condition phrase is an object. With the case-to-case pattern, both types of knowledge are mixed and cannot be easily separated. The result is generic knowledge is distributed to large numbers of design specifications, programs and test specifications repetitively, which leads to intensive repeated works of development and test departments. This is the main reason of inefficiency of the case-to-case pattern.

C. Constructing data warehouse based on operational Metadata-driven Builder Pattern

Based on the above analysis, it is two kinds of repetition that leads to inefficiency of data warehousing with case-to-

case pattern. And it has been found that the carriers of the generic knowledge repeat, while those of the object-specific knowledge do not. So "operational metadata-driven builder pattern" for data warehousing is put forward to avoid repeating generic knowledge by separating the generic knowledge and object- specific knowledge.

In the Fig.2, *Builder* is a table called operational metadata. The operational metadata is similar with the metadata atomically generated by the database management while operational metadata is created by users in which cleaning rules are stored for all the tables needed to be cleaned and then put into data warehouse. And these cleaning rules in the operational metadata table are just the object-specific knowledge. *Director* is a store procedure and one of its parameters is *tablename*. This storage procedure aims to generate a SQL statement which can clean a given table. The parameters of this storage procedure are parts of "object-specific knowledge", while the SQL grammar and keys for that SQL statement is "generic knowledge". And this store procedure generates the SQL statement for the cleaning of a concrete table needed to be cleaned according its parameter automatically. And a record in the operational metadata table is just *Concretebuilder*. Thus when the storage procedure is executed, the generated SQL statement integrate "generic knowledge" and "object-specific knowledge" to realize the cleaning for a given table determined by the parameter of storage procedure.

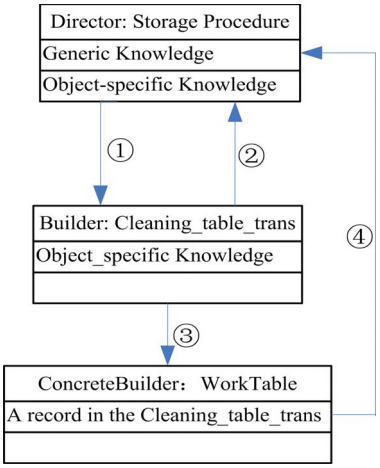


Figure 2. Process of Operational metadata-driven builder pattern

Here what needs to illustrate again is operational metadata is a table storing column cleaning rules or a mapping from source table to destination table. The operational metadata is created and maintained by users or system automatically. Generally speaking, operational metadata is just the object-specific knowledge which do not repeat.

IV. CONSTRUCTING DATA WAREHOUSES BASED ON OPERATIONAL METADATA-DRIVEN BUILDER PATTERN

In this section, the example used in the case-to-case pattern is also adopted to illustrate how to construct data warehouse with the metadata-driven builder pattern.

Firstly a metadata table is created and named "*Cleaning_table_trans*", and its structure and the records are shown in Tab.1.

TABLE I. CLEANING_TABLE_TRANS

Table-name	Colounm-name	value	replace_value
Education program	course period	charindex(' . ',type)>0	replace(type,'. ','.')
Education program	course period	right(rtrim(type),1)='W'	replace(type,'W','w')
Education program	course period	NULL	replace(type,' 1 ','1')
Education program	course period	right(rtrim(type),1)='w'	str(16*cast(left(type,charindex('w',type)-1) as decimal(4,1)))
Stude-nts	spec		ltrim(spec)='Computer science and technology (software engineering)' 'software engineering'

Then a storage procedure is designed as followed.

```
ALTER PROCEDURE [dbo].[CleaningColunms_t]
    @tablename varchar(50)   --Table name needed to be cleaned
    AS
    BEGIN
        DECLARE CL  cursor
        FOR SELECT colounmname, value, replace_value
                FROM MetaData.Cleaning_table_trans
                WHERE tablename=@tablename
        OPEN CL
        DECLARE
         @cn varchar(50),        --Column name
         @ov varchar(500),       --Original value, logical expression
         @rv varchar(500),       --Objective value, expression
         @str varchar(2000)      --SQL statement
      FETCH NEXT FROM Cl INTO @cn,@ov,@rv
      WHILE (@@fetch_status=0)
        BEGIN
          SET @str='update '+@tablename+' set
'+@cn+'='+@rv
          IF (ltrim(@ov)!='')
            BEGIN
              SET @str=@str+' where '+ @ov+' ; '
            END
          ELSE
            BEGIN
              SET @str=@str+' ; '
            END
          INSERT INTO          dbo.sql_statements
SELECT NULL,@str        --Output SQL statement generated
          FETCH NEXT FROM Cl INTO @cn,@ov,@rv
          END
      CLOSE CL
```

The above storage procedure is the Director in "operational metadata-driven builder pattern". Then execute the above storage procedure with the following SQL code.

```
SELECT sql_x FROM dbo.sql_statements
```

Then SQL statements is generated by executing the storage procedure and can be displayed with the following SQL code.

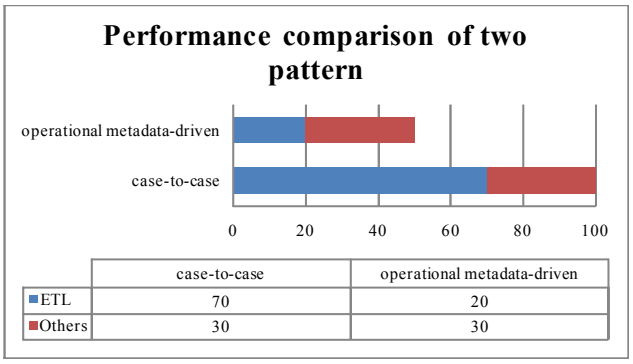
```
EXEC    dbo. CleaningColunms_t
        @tablename = ' Education program '
```

And the output of the above SQL code is displayed as followed. With the storage procedure the SQL statement is created for column cleaning of the given table in the data warehouse.

```
update Education program set type=replace(type,'. ','.')
where char index('. ',type)>0;

update Education program set type=replace(type, 'W', 'w')
where RIGHT(RTRIM(type),1)='w';

update Education program set type= replace(type,' 1 ','1');

UPDATE Education program SET type=STR(16*CAST(LEFT(type, CHARINDEX('w', type)-1) AS DECIMAL(4,1))) WHERE RIGHT(RTRIM(type),1)='w';
```

The SQL statement is just *ConcreteBuilder*. Just through executing the above storage procedure with the given table name as its parameter, the SQL statement for column cleaning of given table is generated automatically without amount of repeatable programming.

The "operational metadata-driven pattern" aims to make people avoid lots of repeatable, fallible and senseless operations and works. "Operational metadata-driven pattern" has already been applied in reality by us and actually got a better performance than "case-to-case pattern". As shown in Fig.3, with "Operational metadata-driven pattern" , performance of ETL can be improved significantly compared with "case-to-case pattern".



	case-to-case	operational metadata-driven
ETL	70	20
Others	30	30

Figure 3. Performance comparison of two pattern

V. CONCLUSION

This study puts forward an approach called "operational metadata-driven builder pattern" for constructing data warehouse. Because this pattern is based on the classification of "generic knowledge" and "object-specific knowledge", the ETL of data warehouse is more flexible and extensible with this pattern. The practical application of this study will have a positive effect on the ETL method in the field of research and practice.

REFERENCE

[1] W. H. Inmon. Building the Data Warehouse. NJ: John Wiley, 1992.

[2] S. Luján-Mora, P. Vassiliadis, J. Trujillo. "Data Mapping Diagrams for Data Warehouse Design with UML". Springer, pp. 191-204, Proc. of the 23rd International Conference on Conceptual Modeling, Shanghai, China, November 8-12, 2004.

[3] J. Trujillo and S. Lujan-Mora. "A UML Based Approach for Modeling ETL Processes in Data Warehouses". New York: ACM, pp. 7-320, Proc. of the 22nd International Conference on Conceptual Modeling, Chicago, IL, USA, October 13-16, 2003.

[4] P. Vassiliadis, A. Simitsis, P. Georgantas, and M. Terrovitis. "A Generic and Customizable Framework for the Design of ETL Scenarios". Information Systems, 2005, 30(7), pp. 492- 525.

[5] H. Zhong, W. Feng, H. Tan and T. Huang. "Design and Implementation of ETL System for Data Integration". Journal of computer science, 2004, 31 (9), pp. 87-89. (In Chinese)

[6] T. Sellis. "Formal specification and optimization of ETL scenarios". Proceedings of the 9th ACM international workshop on data warehousing and OLAP, New York: ACM, 2006, pp. 1-2.

[7] Dimitrios Skoutas and Alkis Simitsis. "Ontology-based Conceptual Design of ETL Processes for both Structured and Semi-structured Data". Semantic web and information systems, 2007, 3 (4), pp. 1-24.

[8] R. Kimball and J. Caserta. The Data Warehouse ETL Toolkit. NJ: John Wiley & Sons, 2004.

[9] Oracle. Oracle Warehouse Builder Product Page. URL http://otn.oracle.com/products/warehouse/content.html.

[10] A. Halevy, A. Rajaraman and J. Ordille. "Data integration: the teenage years". New York: ACM, pp. 9-16, Proceedings of the 32nd international conference on very large data bases, Seoul, Korea, September 12-15, 2006.

[11] P. Vassiliadis, Z. Vagena, S. Skiadopoulos, N. Karayannidis and T. K. Sellis. "Arktos: towards the modeling, design, control and execution of ETL processes". Information Systems, 2001, 26 (1), pp. 537-561.

[12] J. G. Xu and Y. Pei. "Overview of data extraction, transformation and loading". Journal of Computer Science, 2011, 38 (4), pp. 15-20. (In Chinese)

[13] IBM. IBM Data Warehouse Manager. URL http://www-3.ibm.com/software/data/db2/datawarehouse.

[14] Erich Gamma, Richard Helm, Ralph Johnson and J. Vlissides. "Design Patterns: Abstraction and Reuse of Object-Oriented Design". Springer, 707, pp 406-431, 7th European Conference Kaiserslautern, Germany, July 26–30, 1993.