A Real-Time Computing Approach for Derived Attribute Values in Object-Oriented Application Design

Takuya Kawakami† Shigeo Kaneda†
† Graduate School of Engineering, Doshisha University
E-mail: † tkawakami@ishss10.doshisha.ac.jp,

Abstract Under the object-oriented design scheme of information systems, the time-consuming calculation of the derived attribute values is often pre-scheduled due to the application domain requirement. However, in this approach, the function of the persistent objects, in the model layer of the MVC model, deeply depends upon the requirement of the upper domain application view. This degrades the independence of the persistent objects, and the maintainability of the application system. To resolve this problem, this paper proposes a design methodology to calculate the derived attribute values in real-time. Concretely, the proposed method adopts the autonomous objects and difference calculation of time-consuming derived attribute values. The proposed approach simplifies the design process of the higher layer of information systems.

Keyword Conceptual Data Modeling, Object Oriented Approach, Database Design

1. Introduction

Maintainability and extensibility are important factors in information system development. Conceptual Data Modeling (CDM),[1][2] advocated by the MASP association, is a powerful tool for creating an object-oriented conceptual model of the target business. The CDM clarifies the essential “information flow” in the business and improves the maintainability and extensibility of the application systems.

However, the CDM business model cannot be directly applicable to the object-oriented detailed design. Usually, the conventional design approaches, such as Rational Unified Process, are employed in the detailed design phase. The object-oriented CDM business model is only used in forming consensus among stakeholders. One of the reasons is that the CDM object often includes the derived attributes value, for example the aggregation of the other attribute, calculated by the “count” aggregation function of the relational database management system.

In the conventional object-oriented approach, the derived attribute value calculation often has been processed as batch processing, a “midnight job,” because the aggregate values should maintain the value in real-time. This midnight calculation blocks the computational load in the daytime.

In this paper, we employ an event-driven processing framework to resolve the above problem. Domain objects of the CDM model are directly converted into objects of the implementation. The “midnight” batch processing is eliminated and the incremental calculation of the derived attribute value is employed. In this scheme, the domain objects of the system are equal to the persistent objects of the CDM model. This is an ideal status of the three-tier MVC object-oriented application architecture.

In the field of relational database design, the normalization process is essential. This process eliminates redundant data and puts one fact in one place. As a result, even if one datum in the database is updated, overall data consistency can be ensured and a conflicting process is avoided by normalization. However, in fact, it is difficult to normalize a database to remove redundancy perfectly for speeding up the process. Therefore, we sometimes break normalization to design the database. It seems like a contradiction that we treat only normalization on a theoretical basis even though we sometimes break normalization for convenience and high-speed performance. Considering the memory size and processing power of current computers, there should be some redundancy left in database tables. In other words, the tables should be a useful interface from an application program rather than designed by a minimum description of normalization.

The objects derived from CDM are drawn to appropriate scale for recognition by people. Therefore, a system designed with CDM is easy to understand, maintain and extend. In other words, the set of objects, which is considered to be a set of persistent objects, is optimized from the application point of view. We should implement these objects as one layer of the application system. This implementation criterion makes it easy to understand the system and make future enhancements.
In this paper, we consider a method for keeping the data of derived values, such as the aggregated value included in the object derived from the CDM, up-to-date.

2. Database design using CDM

2.1 CDM

Conceptual Data Modeling (CDM), proposed by the MASP association, is a method for building a model of the information flow of a target business and identifying the desirable information flows. CDM includes the Static Model diagram, Dynamic Model diagram, and Organization Model diagram. We can assume that the objects extracted from CDM are agreed on by stakeholders participating in the system development project.

In CDM’s Static Model diagram, we draw a core object called “Mono” in the target business, and the relationship between one object and another is shown. The relationship in the Static Model diagram is not one such as “is-a” or “part-of” in the Entity-Relationship diagram but is an essential relation that is called “Koto.” Information flow that changes the state of “Mono” is represented as “Koto.” We must remain that the object appears on Static Model diagram is “domain object” at the same time is “persistent object”. Rather, because the object is persistent, as it appears on the Static Model diagram, the model behaves as a persistent object in the Model-View-Controller pattern.

2.2 Making a layer of the CDM’s object

Considering the set of objects from CDM as a layer, it is expected that the system will become easier to design and maintain. The table is not the smallest element and includes the derived attributes. The database sometimes expresses the character of “View;” for example, a result of total amount is required for business. It takes a lot of time to calculate the total amount, so the user usually deals with batch processing at a certain interval. However, this approach makes for poor maintenance performance because the object of the domain side should always consider the use of the application side. If attribute values are updated in real time, even for an object like View, it will be expected to become easier to design and maintain the system on the application side.
3. Application for Product Management System

In this chapter, we discuss database design referring to the above CDM Static Model diagram (Figure 1). There are abstract objects like “Product” and concrete objects such as “Incoming Product” and “Outgoing Product” in Fig. 1. The reason these objects appear in the model is that the model expresses recognition of people such as “Incoming Product = incoming + Product” or “Outgoing Product = outgoing + Product.” Also, “Product” is an appropriate grain size for recognizing business. There is an identifier, “product name,” in the Product object and others. This is because people often distinguish the Product by its name, not its ID. If we write this object in Entity-Relationship Diagram, we rewrite “product name” as “product ID.”

This Product object has attributes such as “total quantity,” “back order quantity,” and “stock quantity on order.” These values are derived attribute values that can be determined from the values of other attributes. There are two ways to calculate the derived attribute value “total quantity,” as written in the following.

【Batch Processing】
When one day’s work such as storing and taking out products has been completed, the user (or the specially designed “midnight” process) calculates the total number of incoming product and outgoing product for every product, and the user stores them into the attribute “total” in the Product table. Then, the attribute “total” in every Product is calculated. This processing usually runs at midnight.

【Real-time Processing】
For every incoming product and outgoing product, the total number should be updated in the database. In this case, the programmer of the newly developed application can access the object freely because the up-to-date data is always available.

4. Evaluation experiment

4.1 Experimental Methodology
In this experiment, we made the following Relationship-Entity Diagram (Figure 3), except for the objects such as Customer, Delivery-Route on CDM (Figure 1).

We process an aggregation of derived attribute values, and we compare the two execution times: real-time processing and batch processing. The following is the execution environment.
• Programming language: Java
• Database: MySQL
• API for connection Java and a database: JDBC

In advance, we have stored each value into the Product table, Stored-Product table, and Storage-rack table. We made a simulation, such as putting products in storage after a receiving order, getting products from storage, and after giving an order. A main program statement has cycles of ordering products. When the order happened, a main program calls the method to stock or deliver the product. These are common conditions for two kinds of processes.

In fact, getting and receiving orders are performed at 10, 100, and 1,000 times. In other words, each 10, 100, and 1,000 records are made at the Incoming Product table and Outgoing Product table. The total quantity of product is calculated from these.

4.2 Algorithms

Batch processing
After a receiving order is conducted, the main program will call the stock product method. The method gives number of orders as an argument to the Incoming Product class. In this class, the given number is stored in the database. Thus, we simulate the condition in which each product is stored. After all the storing and taking products from storage have been completed, the program takes the total number of each product from the Product table, and then these numbers are displayed. Batch processing to
obtain the total quantity runs all at once.

Real-time processing
In addition to the above factors, the Incoming-Product class takes the number of orders as an argument from the stock product method. At the same time, the program takes the total number of Products that have the same Product ID as the Incoming Product and adds the order number to the total number. About the Out-Product class, in the same way, it subtracts the receipt-order number from the total number. Thus, the total number is updated as every order happens. In addition to the Batch processing, after all the processing has been completed, the total number of each Product is displayed. Therefore, real-time processing runs at the same frequency as ordering.

4.3 Evaluation result
We will demonstrate the result as stated above. It takes 162.7 seconds to process 1,000 records in Batch processing. In real-time processing, it takes 216.8 seconds to process 1,000 records.

Table 1. Execution times of each number of orders

<table>
<thead>
<tr>
<th>Number of orders</th>
<th>Batch</th>
<th>Real-time</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>1.6sec.</td>
<td>2.1sec.</td>
</tr>
<tr>
<td>100</td>
<td>15.8sec.</td>
<td>21.4sec.</td>
</tr>
<tr>
<td>1,000</td>
<td>162.7sec.</td>
<td>216.8sec.</td>
</tr>
</tbody>
</table>

5. Discussion
As shown in the preceding chapter, the execution times increase linearly according to the number of records. In the batch processing, for the “Order (of purchased goods)” n times, the process of taking “Order” instructions, writing to the database, and displaying the conclusive total number of each product, requires order n, O(n) algorithm. The number of records and the execution time are in a linear relation. On the other hand, in real-time processing, the execution time is similar to the batch processing time. In this case, the system has to find and count the total number of each product that has the same Product-ID for every selection or delivery of the product. Then, the total selection or delivery count is updated. The execution time of real-time updating slightly increased compared with the batch processing case. The evaluation shows a 30% increase for calculating the derived attribute values in real time.

In this CDM’s Static Model diagram (Figure 1), there is an abstract object named “Product.” This object has a feature similar to a kind of “View.” This means that users have interest in the statistical quantity that attributes have, for example, total quantity of product. On the other hand, products in each shipment are also expressed as other objects in this model. In the approach of this proposal, a system can be designed with the assumption that each object always has updated attributes. For this, the CPU’s processing load increases by 30%. We can easily suppose that improvement of recent computer performance will cover this load increase.

In this paper, the evaluation example was a simple process “aggregation.” We have compared real-time processing with batch processing based on the “Toy-sized” application. We would like to evaluate the performance of difference calculation function, with a heavier load in the near future.

6. Conclusion
The persistent objects designed by the CDM approach are a good platform for Web 2.0 or SOA services. This is because entities of the CDM static model are “Persistent Objects,” whose grain size has overall consensus among many applications. The service development is easy and effective in this case.

In this case, however, each newly developed application demands good access performance. The applications are required to access the up-to-date attribute’s value any time, even though the value is calculated based on attributes of the other object.

In the case of real-time calculation of the derived attribute values, a table with a large number of values needs to be referred to sequentially to calculate the difference between the current data and the previous one. This calculation of derived attribute values seems to be a crosscutting concern. We would like to think that this process can be expressed by aspect-oriented programming.

References
[1] MASP Association
   http://masp-assoc.org/