DATA MANAGEMENT IN GROUP PURCHASING SYSTEM

Zongfeng Zou, Tao Yu
CIMS&Robot Center, Shanghai UniversityShanghai,China, Email:zfzou,tyu@mail.shu.edu.cn

Abstract: Introduces a method to solve group purchasing and Price Comparison System framework. It stresses on data loading and data management including data model, database, algorithm and buffer Data Structure. The framework offers a efficient method for loading large and complex scientific databases.

Key words: Group purchasing, Price Comparison System, Data management

Modern group-companies are often made of many subsidiary companies. Stocking centralization can economize much fund for its large-scale purchasing action. Subordinate companies hope to choose the best provider according to reliable data in most possible area. Advanced data in variety units is massive amounts of data that must be post-processed and organized to support provider evaluation actions. Increasing data volumes from all of the companies challenge to state-of-the-art database system and data-loading techniques. Price Comparison System consists of an algorithm for data loading, data structure to support data integrity, and counterpoising system. The system will do great benefit to group economy.

1. INTRODUCTION

With the development of technologies, purchase action research has taken place great changes. Advanced data collection technologies can collect many data. They have been built to contain the data and to be used as valuable resources for economy analyses and other purpose. The characteristics are reflected in several demands that must be addressed when loading such data. Firstly, data loading speed must keep up with data

Please use the following format when citing this chapter:
Zou, Zongfeng, Yu, Tao, 2006, in International Federation for Information Processing (IFIP), Volume 207, Knowledge Enterprise: Intelligent Strategies In Product Design, Manufacturing, and Management, eds. K. Wang, Kovacs G., Wozny M., Fang M., (Boston: Springer), pp. 379-385.

acquirement speed. Secondly, it must be possible to put to different database tables from one source file. Thirdly, it often have to perform complex data transformations and computations during the loading process. At last, automatic error recovery is required during the lengthy data-loading process.

The Price Comparison System is a collaborative system among the Shanghai Electric Group Co. Price Comparison is a multi-time, multi-item category purchase recorder at the Shanghai Electric Group Co. located in all area of Shanghai city. The purchase recorder consists of 18 kinds of information and can give evaluation of providers. In contrast to traditional purchases, all of the indent must be recorded in the system and examined and approved by experts of group purchase department. System allows users to statistically analyze the price and trend contributes to the data of the whole group.

2. DATA LOADING REQUIREMENT AND METHOD

The large data-collection rates and volumes noted in the previous part give the necessity for a fast data repository loading process that is capable of keeping up over time with the speed of data get. Much need contribute to the difficulty of achieving this goal. Collected raw data and computed catalog data are usually archived in a mass storage system that is separated from the database server. The catalog data that should be transferred from the mass storage system to load the database repository typically saturates the available network bandwidth, introducing the network as the first bottleneck to fast data loading.

Purchase action data encompasses information of many different types, from different information level specifications to the purchase details of goods. This variety of information is interleaved in the catalog data set that is generated when the raw data is processed. During the data-loading process the complex catalog data must be listed, the correct destination tables must be identified, and the data must be loaded into multiple target tables in the repository. Loading data into multiple tables is further complicated by the presence of multiple relationships among tables. Relationships that must be maintained by obey the primary and outer key constraints during the loading process.

Additional operations are also performed during the data loading process. These operations include transformations to convert data types and change precision, validation to filter out errors, and calculation of values such as the Hierarchical Triangular Mesh ID (HTMID). All such intensive operations place an additional burden on the loading process. Finally, because data

loading is typically a lengthy process, a mechanism of automatic recovery from errors is a basic requirement.

To solve the problems in building the repository, we designed framework, called Price Comparison System, which is made up of (1) a logical network to join the different data position of all of the companies. (2)an efficient algorithm to perform bulk data loading, (3)an effective data structure to maintain table relationships and allow proper error handling, (4)active database and system tuning to achieve the best data-loading performance. Using this framework we can bulk load data, insert data into multiple database tables at the same time without locking and constraints, and recover the loading process from errors. The Price Comparison System framework has improved the ability of data loading.

3. THE PRICE COMPARISON SYSTEM FRAMEWORK

3.1 Data Model and Price Comparison System Mission

The original data created by the company in the past are archived in mass storage system. A program is arranged on the data to extract catalog data, which includes a wide range of information. Typically, the catalog data includes information on the user's position, the amount they wanted, the parameters described, the providers presented, the frames generated, and the feedback written. The catalog information is first written to an ASCII file, which is saved in the mass storage system and then uploaded to a repository database. The format of the catalog file varies depending on the extraction program used. Commonly, different aspects of the catalog information are interleaved in the file. For example, a row of frame information is followed by four rows of frame information, and a row of object information is followed by four rows of finger information. Each row in the catalog data file usually has a tag or a keyword that can be used to determine the destination table in the database.

Oracle 10g is the best choice to host the data repository. The repository database has been designed to store the catalog data and support data analysis. Each table stores a unique aspect of the purchase information. For example, metadata related to a purchase action such as company position, state in use, and created time. Metadata related to the company such as company id and attribute goes into the table company_columns. Detailed information related to purchase goods goes into the merchandise table.

3.2 Data Loading Algorithm of Price Comparison System

Data loading allows multiple operations to be inserted into a single area and waiting for one database call, minimizing network communication and disk I/O. It is straightforward to perform bulk loading to a single table. If the data belonging to a child table is loaded before the corresponding parent keys, a foreign key constraint is disarranged. To avoid this problem is to first buffer the data into separate arrays designated for different tables, and then to follow the parent-child relationship order when performing the bulk inserts. The table-loading order is illustrated in Figure 1.

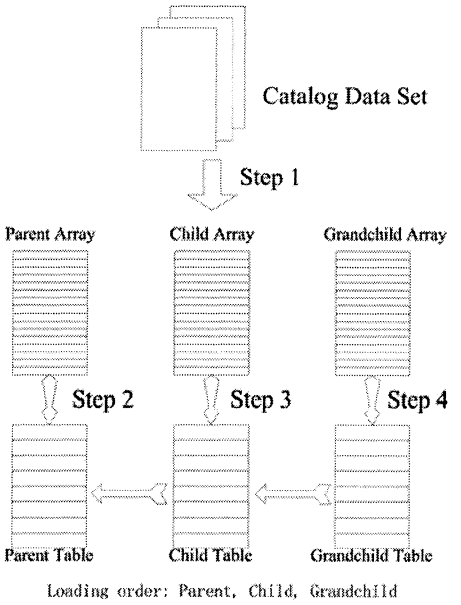

Figure 1. Data Loading Order with Multiple Tables

Another difficulty in data loading is recoverability in a lengthy data-loading process. The catalog data set to be loaded sometimes contains errors such as missing and/or invalid values. The algorithm not only speeds up data loading by a factor of 7 to 9, but also maintains the relationships of multiple tables and enables the system to recover from errors during data loading. The algorithm is introduced in Figure 2.

When any data array reaches array-size (Line 5), the batch_row procedure is called (Line 10) for each array based on the parent-child relationship. The array for the parent table is processed first, followed by the child tables. This processing sequence depends entirely on the data model.

Our approach does not cover circular parent-child relationships, as a good database design does not have circular dependencies between tables.

```
Input: a series of input data files
Output: populated database tables
int array-size /* the size of an array */
int batch-size /* the size of a batch; typically <= array-size */
Procedure data_loading
(1) for each data file {
(2) open the file
(3) for each row {
(4) parse the row, do validation, transformation and
computation, and buffer it in a designated array based on the
destination table;
(5) if (any array.size >= array-size) {
(6) for each array ordered by parent-child relationship {
(7) first_idx = 0;
(8) last_idx = array.size;
(9) while (first_idx <= last_idx)
(10) first_idx = batch_row(array, destination_table,
first_idx, last_idx)
(11) } /* for each array */
(12) } /* if reach array-size */
(13) } /* for each row */
(14) } /* for each data file */
Function batch_row (array, destination_table, first_idx,
last_idx)
(15) while (first_idx <= last_idx) {
(16) prepare SQL statement;
(17) add to batch;
(18) if (batch-size reached) { /* time to insert */
(19) insert batch into the destination table;
(20) if (successful insert) {
(21) first_idx += batch-size;
(22) } else { /* if an error occurred skip that row */
(23) skip_one_row;
(24) return (the_next_index);
(25) }
(26) } else if ( first_idx == last_idx) { /* array done */
(27) insert batch into the destination table;
(28) if (successful insert) {
(29) return (last_idx + 1 );
(30) } else { /* if an error occurred skip that row */
(31) skip_one_row;
(32) return (the_next_index);
(33) }
(34) }
(35) } /* while there are more rows to process */
```

Figure 2. data loading Algorithm

The data loading algorithm has been implemented using the JDBC core API. In the best case, that is when the data set is error-free, the algorithm will generate database calls and result in database I/Os. In the worst case, for example primary key violations on every row caused by repeatedly loading duplicate rows, data loading will deteriorate to a series of singleton insert operations which make database calls and perform database I/Os. This behavior results from the algorithm breaking up the problematic batch, skipping the error row, and repacking the batch to continue each time that an error is encountered.

3.3 Buffer Data Structure

The interleaving of data for multiple target tables, combined with the presence of multiple relationships among tables, relationships that must be maintained by obeying with the primary and foreign key constraints during

the loading process—makes bulk loading especially challenging. To manage the crossing data and complex table relationships, and to improve quick recovery when an error is detected during the data-loading process, an effective data structure, array-set, is necessary.

The number of arrays in the array-set at a given time during data loading depends on the degree to which the data in the catalog data set is interleaved. As the input catalog data set is processed, the framework creates a new array in array-set whenever it reads an input row targeted for a database table for which no array is currently maintained. When any of the arrays in array-set are fully moved, data loading finished. At the end of the data loading cycle, the arrays in array-set are destroyed and their memory released. The framework resumes reading the input catalog data and creates new arrays as required to buffer the incoming table rows. In order to load the catalog data items into different destination tables and keep the proper relationships, array-set can buffer the data and execute the data loading in the order of parent-child sequence.

3.4 Active Database and System Adjusting

In Price Comparison System framework, active database and system adjusting are set to achieve the best configuration. The performance adjusting is very important to achieve the fast loading of massive data volumes required for repository. Since the Price Comparison System is a multi-year continuous effort, the price repository must serve two purposes at the same time. First, it must be a warehouse to store increasingly loaded data. Second, it must act as a query engine to support economic analyzing research. For example, query performance is necessary to create indices on database tables. However, indices usually make data loading slower because every insert requires an update of all index entries.

Allocating a smaller database data cache actually improves the data-loading performance. Since a database writer needs to scan the entire data cache when writing new data from data cache to disk, the reduced data cache size minimizes the work that the database writer has to do each time . This reduced cache configuration should be adjusted after the intensive data-loading phase is entirely because a larger data cache often performs better for user queries.

4. CONCLUSIONS

The Price Comparison System is a project to collect, archive, process, and distribute trade data for economics collaborations. The repository being

built at Shanghai Electric Group Co. is to hold catalog data for the complex provider analyzing. The first significant was to load the catalog data into the repository database in a in time fashion. Data loading with array buffering is a good method to accomplish the task. System framework consists of an efficient algorithm for data loading, an effective data structure to support data integrity and proper error handling during the loading process and guidelines for database and system tuning. This framework can help to decrease much of the loading time.

5. REFERENCES

1. Du Zhihui, Chenyu, Liupeng. *Grid Computing.* Tsinghua University Publishing Press. Apr 2002.
2. Gabrielle Allen, Thomas Dramlitsch, Ian Foster, Nick Karonis, Matei Ripeanu, Ed Seidel, Brian Toonen, Supporting Efficient Execution in Heterogeneous Distributed Computing Environments with Cactus and Globus, Supercomputing 2001. [Winning Paper for Gordon Bell Prize (Special Category)]
3. Donald McMullen, Randall Bramley, et al. The Xport Collaboratory for High-Brilliance X-ray Crystallography, http://www.cs.indiana.edu/ngi/sc2000/index.html
4. http://www.iumsc.indiana.edu/
5. Y. Dora Cai, Ruth Aydt, Robert J. Brunner. " Optimized Data Loading for aMulti-Terabyte Sky Survey Repository". National Center for Supercomputing Applications (NCSA),Nov, 2005
6. S. Amer-yahia and S. Cluet. "A Declarative Approach to Optimize Bulk Loading into Databases". ACM Transactions on Database Systems, Vol. 29, Issue 2, June 2004.
7. G. Reese. Database Programming with JDBC and Java. O'Reilly. 2nd Edition, Aug. 2000.