Index Structure for the Fact Table of a Star-Join Schema and Template Query Processing

Anna Rozeva

Abstract: *Star-join is a database scheme designed for the purpose of data warehousing. A typical query on a star-join scheme as well as an algorithm for its processing has been examined. An index structure for the star-join fact table has been proposed with the aim of facilitating template query processing. A multilevel index for the fact table's composite key has been build having time_key as a top level. Stream processing technique has been implemented for resolving query's application and join constraints as well as the aggregation step of the query processing algorithm.*

Key words: *Star-Join Schema, Fact Table, Multilevel Index, Query processing, Stream Processing*

INTRODUCTION

Star-join (4) is the database schema designed for implementing the dimensional data model. This model has turned out to be the most natural and comprehensive one for describing business activity. A typical business is dealing with selling products in a number of markets and measures its performance over time. The star-join schema of a business is shown in fig.1.

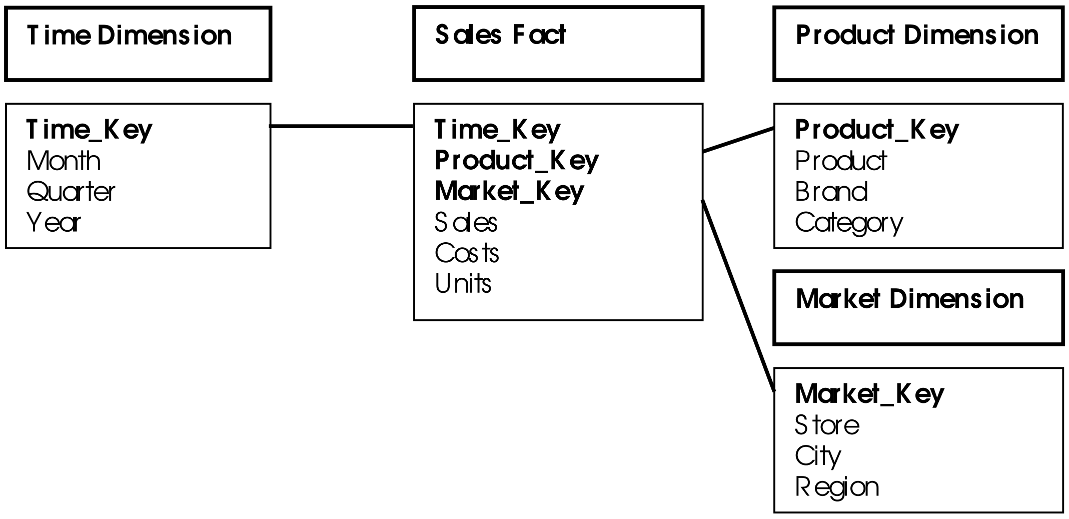

Fig.1. Star-join schema

Unlike the entity relationship schema star-join consists of very few tables. The table which houses numerical measurements of the business, i.e. sales, costs, units, etc. is the fact table. It is the centre table of the star schema. The other tables are dimension tables. They are highly denormalized and store the textual descriptions of the numerical facts. They represent focuses or perspectives of the business. The fact table is the only one with multiple join connectivity to the dimension tables. Dimension tables are linked only through the fact table. Each of the measurements in the fact table is taken at the intersection of all the dimensions. Numerical facts appear in the course of time. Any combination of time, product and market generates a different record in the fact table. The periodicity with which records for the fact table are generated is called a grain. Dimension tables have simple primary keys. The primary key of each dimension table is present in the fact table as a foreign key. As a result of the referential integrity requirement a dimension key in the fact table must have its counterpart in a dimension table. The key of the fact table is a composite or concatenated one from the dimension keys. An index structure for the fact table's composite key is to be designed. The structure is to serve the processing of a typical star-join query. Algorithms concerning query processing implementing the index will be presented.

TEMPLATE STAR-JOIN QUERY AND PROCESSING ALGORITHM

A typical query on a star-join schema (4) states:

Find all product categories sold in the fourth quarter of 1998 and present the total sales and units sold. A category refers to a collection of products. The answer set returned from such a query will look as follows:

Category	Sales	Units
Hard disks	15,000	2,000
Processor Chips	23,000	4,800
Colour Monitors	18,000	3,200

The query tool (client application) that produces the above answer set generates the following SQL statement:

```
Select p.category, sum (f.sales), sum (f.units)
From fact f, product p, time t
Where f.product_key = p.product_key
          And f.time_key = t.time_key
          And t.quarter = '4 Q 1998'
Qroup By p.category
Order By p.category
```

The statements in the where clause represent two types of constraints – join and application constraints. Join constraints are the referential integrity constraints on the fact and dimension tables. The application constraint is restricting a dimension (here the time dimension) to some subset of records. An algorithm for the star-join query processing drawn from this SQL template is shown in fig.2.

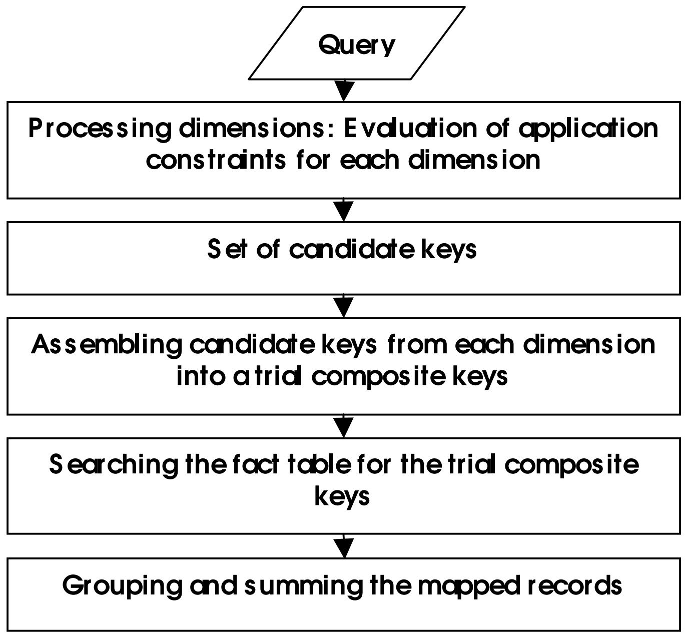
Query
Processing dimensions: Evaluation of application constraints for each dimension
Set of candidate keys
Assembling candidate keys from each dimension into a trial composite keys
Searching the fact table for the trial composite keys
Grouping and summing the mapped records

Fig.2. Algorithm for a typical star-join query processing

All queries on a star-join schema are "select" statements, eighty percent of them being browses (single table), and twenty percent – multitable joins (4). Multitable joins are to be considered further on.

INDEX STRUCTURE FOR THE FACT TABLE

Index structures for time-stamped records in temporal databases have been presented in (1, 5, 6). These temporal indexes are used to cluster data on their temporal relationships. They have been designed to support efficient query retrieval based on time. When examining the fact table of a star-join schema its key is a combination of the foreign keys of the dimensions:

$K_F = \{K_{D1}, K_{D2}, \ldots, K_{Dn}\}$

The fact table always has the time as a foreign key. Depending on the granularity that has been chosen records can be viewed as a history of time slices, each at a time = Time_key. Each time slice will contain a set of records having time foreign key = Time_key. Most often queries concerning the star schema as shown in the template query as well are looking for records that have been generated at a certain time. In order to optimize processing of such queries the index of the fact table is designed as a multilevel one . The first index level indexes records on time foreign key. The next index level will index on product and market. Each one of them will form a sublevel. The join constraint in the query will determine the sublevel to be used. In case that there are other dimensions in the star schema additional levels can be build, depending on the frequency with which constraints on them appear in the queries. The multilevel index structure is shown in fig.3.

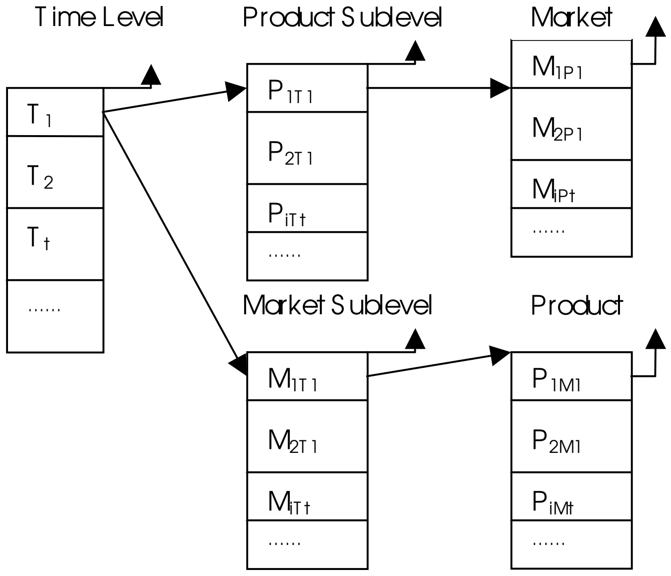

Fig.3. Fact table index structure

Each time_key value in the time level has lists of associated product and market keys. The sublevels serve join constraints on the corresponding dimensions. An element of a sublevel has an associated list of keys from the other sublevel. Thus the product sublist has a list of market keys, that are associated to each element. In case that query application constraints concern time dimension - output sales numbers for a stated time period, only the index's time level is processed. If application or join constraints refer to other dimensions besides time – output sales for a product category for a time period, the corresponding sublevel is processed as well. If query constraints involve both dimensions, output sales for a product category in a particular region, both sublevels are to be accessed consecutively. The record structure for the index's levels is shown in fig.4.

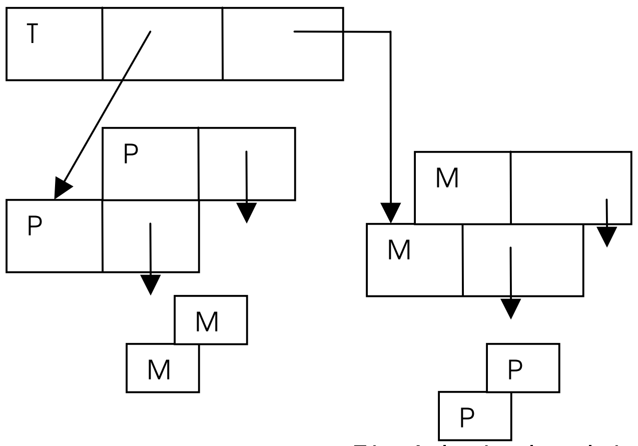

Fig.4. Index levels' record structure

ALGORITHMS FOR PROCESSING A TEMPLATE STAR-JOIN QUERY

From the algorithm for query processing shown in fig.2. the last two steps will be implemented, i.e. searching the fact table for the trial composite keys and aggregating the records found. Browsing the dimension tables for evaluating application constraints won't be discussed. It is assumed that the assembled candidate keys from each dimension is passed as a stream to the fact table index. Stream processing techniques used in languages as C++ (2, 3) deal with an ordered sequence of data objects. Application of this approach to the fact table's index structure proposed in the previous section, turns out to be straightforward. Algorithms concerning evaluation of time periods, joins and aggregations and their implementation through stream processing will be presented further on.

- **Time period algorithm.**

The problem to be solved consists in selecting the records for every time_key value in the time level of the fact table index that belongs to the interval of the time application constraint. By applying stream processing approach to the index file it is treated as a stream of input or output. Given a data object, its size can be decided by using the C++ function *sizeof()*. When the exact position (the exact address) from which a data object is stored is decided by using C++ function *seekg()*, the data object can be retrieved and the file pointer moves to the next data object. Given the index file with the top time level and the product/market sublevels time period operation retrieves the records corresponding to a time period (Ti, Tj). This can be performed by using either sequential or a binary search of the time level to decide the addresses A_{Ti}, A_{Tj} of records, corresponding to the time period (Ti, Tj). The algorithm is illustrated by using C++ pseudo-code.

```
Search($A_{Ti}$, $A_{Tj}$, Ti, Tj)
For (int i=0; i<$n_T$; i++)
fileIndex.seekg(i*sizeof(T)+( $A_{Ti}$-1)*sizeof(Ts));
  /*seek the address of the first Ts with time_key in (Ti, Tj) */
For (int j=0; j< $A_{Tj}$,- $A_{Ti}$;  j++)
{fileIndex,read((char*) & Struc_Buf(j), sizeof(Ts));
Buf<<Struc_Buf(j)};
  /*Tj-Ti+1 records Ts of the object Ti+1 are sequentially read from the index file and
kept in Struc_Buf(j)*/
```

Records in the Struc_Buf are used to access fact table records, that satisfy the stated application constraints.

- **Join algorithm.**

Join of the fact table and the dimension tables has to be performed in order to satisfy join constraints in the template query. The tables that have been referred to in the query are fact, product and time. Objects T of time dimension table and P of the product dimension tables are retrieved and the values of their keys are obtained. Given these keys the address of the fact F in the fact table is determined. The fact F is retrieved and the predicate is evaluated. If true T, P and F are joined. The process is repeated till all objects in the Time and Product tables are visited. The algorithm can be expressed using the following C++ pseudo code:

```
For (int i=0; i<n_T; i++)
For (int j=0; j< n_P;  j++)
{FileTime.read((char*) & Buf_T, sizeof(T));
FileProduct.read((char*) & Buf_P, sizeof(P));
FileIndex.seekg(Time.time_key, Product.product_key)
  /*according to the values of time_key and product_key locate the address of F in
fact table*/
FileFact.read((char*) & Buf_F, sizeof(F));
  If  (predicate) Buf<<(join T and P and F);}
```

- **Aggregation algorithm.**

In order to perform aggregation such as Sum, Avg, Max, etc. an operator agg_func (7) will be used in the C++ pseudo code that illustrates the algorithm:

```
Search(A_Ti, A_Tj, Ti, Tj)
For (int i=0; i<n_T; i++)
fileIndex.seekg(i*sizeof(T)+( A_Ti -1)*sizeof(Ts));
For (int j=0; j< A_Tj,- A_Ti;  j++)
{fileIndex,read((char*) & Struc_Buf(j), sizeof(Ts));
agg_func(Struc_Buf.itemi, func, value);
  /*perform an aggregation function func for a specified attribute, i.e. itemi*/
Buf << value;
            Void agg_func(item, char*func, float value);
            Int m=Tj-Ti+1;
            Switch(func)
            {
                      case sum:
                      for (int i=0; i<m; i++)
                      value+=item(i);
                      break;
                      case avg:
                      { ...
                      case max:
                      {...
                      }
```

CONCLUSIONS AND FUTURE WORK

A typical query on a star-join schema has been discussed and a processing algorithm presented. An index structure for the fact table of the schema has been proposed that will facilitate the template query processing. It is a multilevel one with time key chosen for the top level and the next level consisting of two sublevels. Algorithms for the steps of the star-join processing dealing with searching the fact table index for the trial composite keys have been designed. Implementation of time period, join and aggregation operations using C++ stream processing technique has been presented. Future work will include index maintenance when new records are loaded in the fact table of the schema as well as browse queries support and implementation for the dimension tables.

REFERENCES

(1) Ang,C.H., K.P.Tan, The Interval B-tree, Information Processing Letters 53 (2) (1995) 85-89.

(2) Carrano,F.M., Data Abstraction and Problem Solving with C++, Benjamin Reading, MA, USA, 1995.

(3) Cliff Leung,T.Y., R.R.Muntz, Stream Processing: Temporal Query Processing and Optimization, in: A.U.Tansel (Ed.), Temporal Databases: Theory, Design and Implementation, Benjamin Reading, MA, USA, 1993.

(4) Kimball,R., The Data Warehouse Toolkit. Practical Techniques for Building Dimensional Data Warehouses, John Wiley & Sons, inc., 1995.

(5) Kumar,A., V.J.Tsotras, C.Faloustos, Access Methods for Bi-temporal Databases, in: Proceedings of the International Workshop on Temporal Databases, 1995, pp.235-254.

(6) Salzberg,B., V.J.Tsotras, A Comparison of Access Methods for Time-evolving Data, ACM Computing Surveys, 1994

(7) Wang,L., M.Wing, C.Davis, N.Revell, An Algebra for a Temporal Object Data Model, LNCS 1134, Database and Expert Systems Applications, (DEKA96), Proceedings, Springer-Verlag, Zurich, Switzerland, September 9-13, 1996, pp.667-677

ABOUT THE AUTHOR

Assoc.Prof. Anna Rozeva, PhD, Department of Computer Systems and Informatics, University of Forestry Sofia, Phone: +359 2 91 907 340, Å-mail: arozeva@ltu.acad.bg