Journal of Engineering and Applied Sciences 13 (3): 746-751, 2018
ISSN: 1816-949X

Column-Based Storage Structure for Bigdata Processing

Jeong-Joon Kim
Department of Computer Engineering, Korea Polytechnic University, Gyeonggi-do,
237 Sengideahak-ro, 15073 Siheung-si, South Korea

Abstract: The user's query that is requested by the DBMS often accesses fewer columns than accessing all row values. However, the existing NSM (Narray Storage Model) storage model that saves in row units can not handle this properly. Also, in the OLAP environment, it is a feature to frequently use analysis tasks and aggregate functions to process with value of a specific column. It is a well-known fact that column-based storage model is necessary in OLAP environment in other studys, etc., already. Therefore, a column-based storage model is required. Therefore, the model proposed in this study presents a model that is advantageous for access by record and has high space efficiency.

Key words: OLAP, column-based storage, DSM, DBMS, aggregate, NSM

INTRODUCTION

User queries requested to the DBMS are more likely to access fewer columns than accessing the values of all rows (Jagadish *et al*., 2005; Ramakrishnan and Gehrke, 2000). However, existing NSM (Narray Storage Model) storage models that store in row units do not handle them appropriately. In addition, the OLAP environment is characterized by the use of a large number of analytic functions and aggregate functions that process values of a particular column (Copeland and Khoshafian, 1985; Ailamaki *et al*., 2002). It is a well-known fact that column based storage model is needed in OLAP environment in other papers. Therefore, a column-based storage model is required. The proposed storage model provides the following advantages and advantages (Francois and Rainer, 1986; Halevy *et al*., 2006).

Instead of using the RID method at all, we use a join index to create a single record but since the proposed model uses RID, we need to add a join index it is expected to be more advantageous in terms of access to records and space efficiency because it is not necessary to create (Ross, 2008; Amir *et al*., 2001).

Literature review: In the NSM Model, the records are stored sequentially as shown in Fig. 1 and the start offset of each record is stored at the end of the page. NSM is not suitable for OLAP environment and storage models such as DSM and PAX are available. Decomposition Storage Model (DSM) is a method of storing a table in a column-by-column table without modifying the storage method inside the DBMS. It has the following structure (Fig. 2).

If you want NSM to query "print a record with id = 12305".

```
Select*from employee where id = 12305
```

The same query is made in DSM as follows.

```
Select id, name, age, gender from a1-a3
Where a1.id = 12305, a2.id = a1.id and a3.id = a1.id
```

The advantages of DSM are:

- If frequently used queries are often processed using a specific column, disk I/O
- Because the store is made up of two related keys and a specific column there are few cache misses

The disadvantages of DSM are:

- Multiple tables must be modified when a record is updated
- There is a lot of duplication of related key and the storage space is wasted

SYSTEM DESIGN

Suggested model background: By implementing the column based storage model in the general purpose DBMS through the proposed technology, users can provide table with the storage model that can be used together in the OLAP (column based) and OLTP (row based) environments. In particular, the proposed architecture is designed to be implemented within the

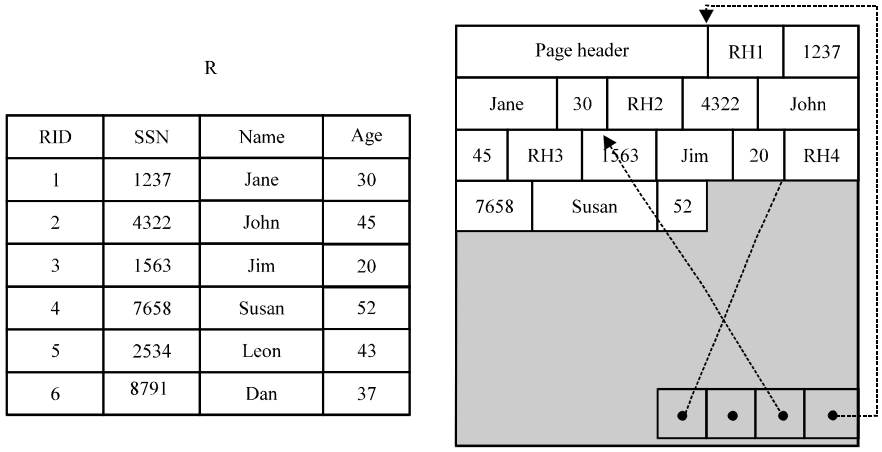

Fig. 1: NSM Model

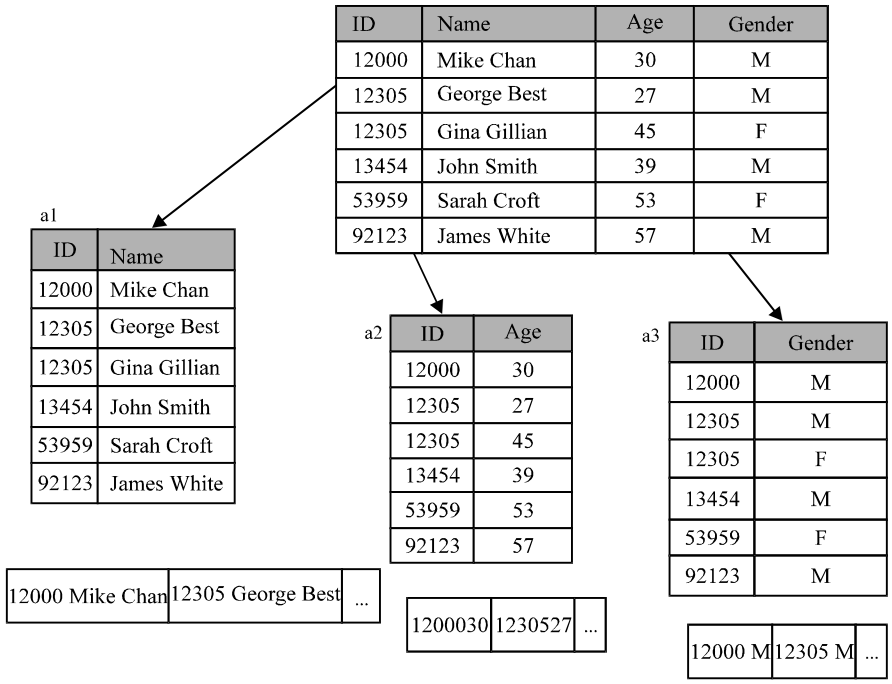

Fig. 2: DSM Model

range of existing Row based DBMS architecture, so it is very efficient in implementation in general purpose DBMS.

First, we examine the storage configuration of existing DBMS and propose a new structure. Managing disk tables in a typical DBMS is done in the following hierarchical structure.

Table space->Segment->Extent->Page

Table space, extent, page are physical structures and segment is a table created by the user. When inserting/deleting/updating data in a specific table, the segment descriptor is searched for and the data is accessed by searching for a page existing in the extents pointed by the segment descriptor (Fig. 2).

In a typical DBMS, the page internal structure is a slightly modified form based on NSM. However, depending on the type of record there may be records that span multiple pages. Such records can be stored on multiple pages in various forms such as.

From Fig. 3 and 4, Fig. 3 shows a case where a certain column with a large size of one row exists in another page and the right Fig. 3

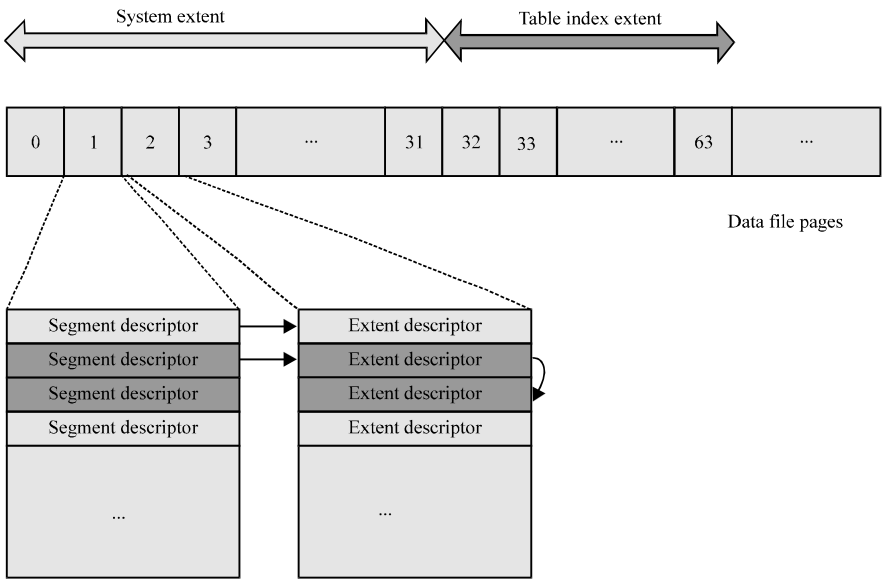

Fig. 3: Disk table management of common DBMS

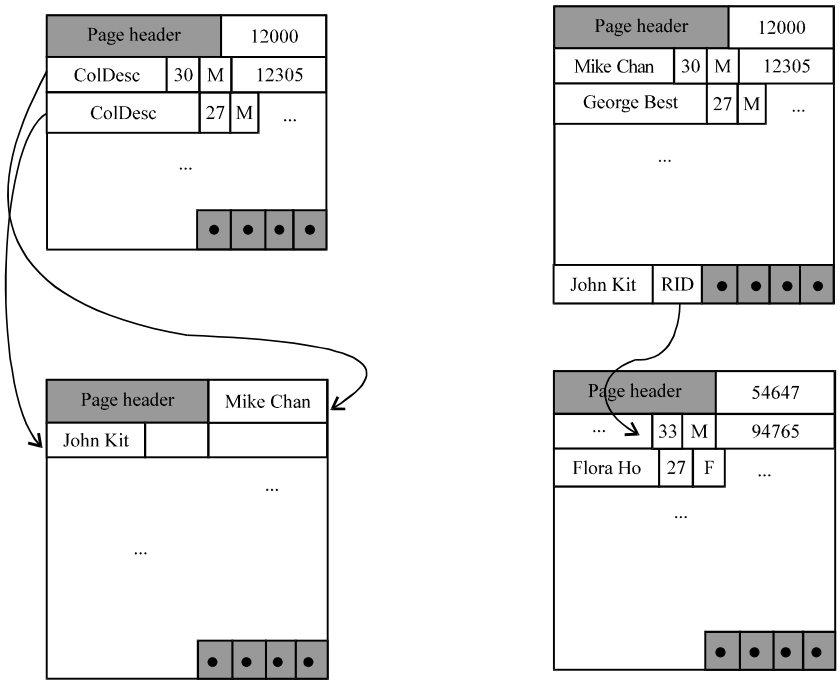

Fig. 4: Example of storing records in a normal DBMS

shows an example of a case where one row is not recorded on one page and the rest is stored on another page.

Proposed model (hybrid storage table architecture using group segment): The proposed technology is a new type of architecture that can be stored on a column basis while maintaining the existing row-based data structure described before. Use the segment structure as described above. In addition, column value in one row is designed to be able to implement table with column based storage architecture using existing structure already implemented, so that, it exists in other page. Hybrid storage table can be divided into user interfaces and storage models.

User interface: The user is able to specify which columns should be stored in consecutive pages through the following syntax when creating a specific table. Column based table generation syntax is as follows.

```
Create Table T1 ( (C1 integer, C2 char (5)) G1, C3 integer G2, C4
varchar (20) G3) column based table
```

When you create a table with the above syntax, a table is created in which C1 and C2 have G1 column group, C3 has G2 column group and C4 has G4 column group. Here, one group is stored as row based and one group is stored as column based.

Storage model: As described, in a typical DBMS, one table has one segment. On the other hand, the column based table consists of a base segment which identifies the information of records in one table and group segments which means column group. Segments are created for a single table in the form shown below and each of these segments knows an extent that contains information about the space in which their data will be inserted. The insert/update/delete/select operation using the above structure is performed as follows (Fig. 4).

Insert: For example when a query called insert into T1 values (1, 'AAA', 2, 'BB') arrives, it processes in the following order. A group segment is used to allocate a page to insert the corresponding column. Then, each group column is inserted into the allocated page and the RID is memorized. Finally, the base segment is allocated space for inserting records and records the stored RIDs. If you store the records according to the above algorithm, the records and column values are stored as Fig. 5 and 6.

Update: The update operation can provide various methods depending on the condition and the type of update. The following examples are used to illustrate this.

```
Ex1> update operations that are processed within a group
a update T1 set C2 = BBB where C1 = 1
```

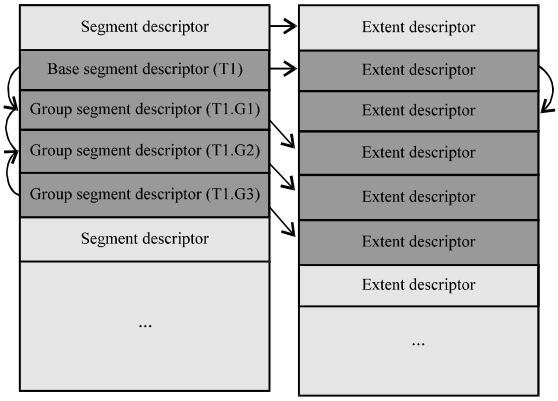

Fig. 5: Proposal storage model

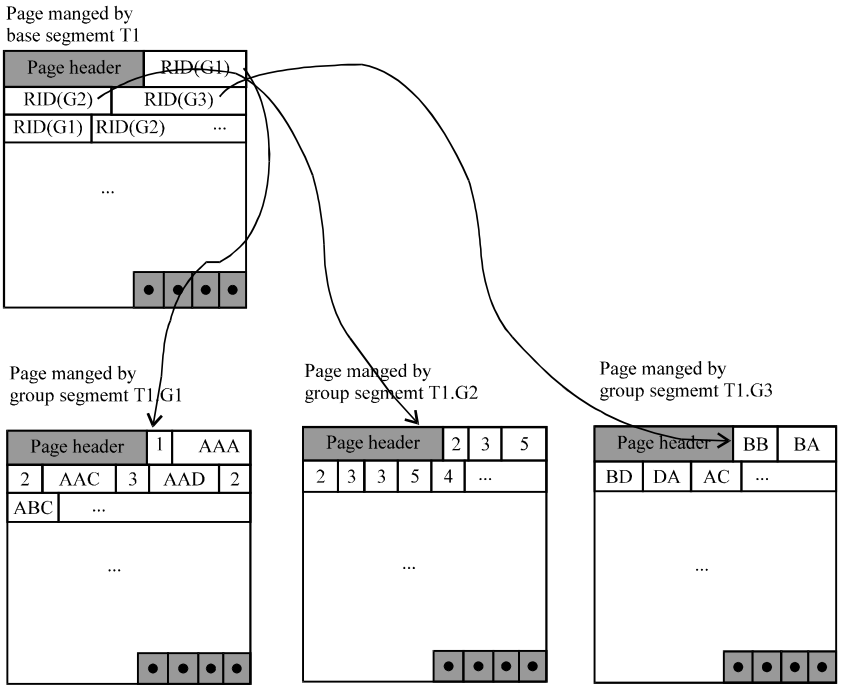

Fig. 6: Insert procedure example

```
Ex2> update operations that are processed in different groups
a update T1 set C2 = BBB where C4 = BB
```

In this case, C1 and C2 are a group, so they are stored together in one group segment page. Therefore, only the group segment T1. G1 can be accessed and the condition can be checked and updated without accessing the base segment. Variable columns are not considered.

In this case, first access the base segment, read the records one by one and use the RID (G3) to access the page where the C4 is stored and compare the predicate. When a record satisfying the condition is retrieved, it goes to RID (G1) of the corresponding record and updates C2-BBB.

Delete: The delete operation accesses the base segment, follows all the RIDs stored in the record, deletes the mark and deletes the record.

Select: The search is described as an example having a type similar to an update operation.

```
Ex1>querying a column within a group
a select AVG (C1) from T1 where C2 like 'AA%' or select
count (C3) from T1 where C3 = 5
```

This query can be processed simply by accessing the group segment T1. G1 without having to access the base segment. That is, it is possible to access T1. G1 and directly access the C1 column of the record matching the condition.

```
Ex2>query accessing columns in multiple groups
a select*from T1
```

A query that accesses the entire records used mainly in the OLTP environment accesses the base segment, accesses the pages pointed to by the base segment and constructs a row to return. In this case, depending on the number of column groups, it is expected that many buffer misses will not occur because the number of designated pages is accessed at one time. In this example, four pages must be accessed to retrieve all the columns of a record and the next record access is accessible from the page already in the buffer, so, no buffer miss occurs.

CONCLUSION

It is a well-known fact that a column-based storage model is required in the OLAP environment. Therefore, a column-based storage model is required. Therefore, the model proposed in this study proposes a model that is more advantageous in accessing records and more space efficient.

Since, the proposed model has all the row/column based storage architecture there are many advantages over the existing 0 technology and the OLAP and OLTP property queries can be tuned to the user's choice in one table. Also, there is a difference that it is possible to access by record basis without join index or join operation which is a problem of existing column based storage model.

IMPLEMENTATION

Finally, it is easy to implement because it has a structure that can be well integrated with the storage architecture of the general purpose DBMS.

ACKNOWLEDGEMENT

This researcher was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (No. 2017R1A2B4011 243).

REFERENCES

Ailamaki, A., D.J. DeWitt and M.D. Hill, 2000. Data page layouts for relational databases on deep memory hierarchies. VLDB. J. Intl. Very Large Data Bases, 11: 198-215.

Amir, A., R. Kashi and N.S. Netanyahu, 2001. Analyzing quantitative databases: Image is everything. Proceedings of 27th International Conference on Very Large Data Bases (VLDB'01), September 11-14, 2001, Morgan Kaufmann Publishers Inc, San Francisco, California, USA., pp: 89-98.

Copeland, G.P. and S. Khoshafian, 1985. Adecomposition storage model. Proceedings of the ACM SIGMOD International Conference on Management of Data, May 28-31, 1985, ACM, Austin, Texas, pp: 268-279.

Francois, B. and M. Rainer, 1986. Checking consistency of database constraints: A logical basis. Proceedings of the 12th International Conference on Very Large Data Bases (VLDB'86), August 25-28, 1986, Morgan Kaufmann Publisher, Kyoto, Japan, pp: 13-20.

Halevy, A., A. Rajaraman and J. Ordille, 2006. Data integration: The teenage years. Proceedings of the 32nd International Conference on Very Large Data Bases (VLDB'06), September 12-15, 2006, VLDB Endowment, Seoul, Korea, pp: 9-16.

Jagadish, H.V., B.C. Ooi and Q.H. Vu, 2005. Baton: A balanced tree structure for peer-to-peer networks. Proceedings of the 31st International Conference on Very Large Data Bases (VLDB'05), August 30-September 02, 2005, ACM, Trondheim, Norway, ISBN:1-59593-154-6, pp: 661-672.

Ramakrishnan, R. and J. Gehrke, 2000. Database Management Systems. McGraw-Hill Higher Education, Boston, Massachusetts, ISBN: 9780072465358, Pages: 740.

Ross, K.A., 2008. Modeling the performance of algorithms on flash memory devices. Proceedings of the 4th International Workshop on Data Management on New Hardware, June 13, 2008, ACM, Vancouver, Canada, ISBN:978-1-60558-184-2, pp: 11-16.