

The Bitmap Join Index OLAP Tool

Anderson Chaves Carniel¹, Thiago Luís Lopes Siqueira^{2,3}

São Paulo Federal Institute of Education Science and Technology (IFSP)

¹Salto Campus – 13.320-271 – Salto – SP – Brazil

²São Carlos Campus – 13.565-905 – São Carlos – SP – Brazil

³Computer Science Department

Federal University of São Carlos (UFSCar) – 13.565-905 – São Carlos – SP – Brazil

accarniel@gmail.com, prof.thiago@ifsp.edu.br

Abstract. *This paper introduces the BJIn OLAP Tool that manipulates only bitmap join indices to enable efficient roll-up, drill-down, slice-and-dice and pivoting multidimensional operations over data warehouses. We validated our tool through a performance evaluation to assess its efficiency and to corroborate the feasibility of adopting the bitmap join index in OLAP. The performance results reported expressive performance gains if compared to existing solutions.*

1. Introduction

Online Analytical Processing (OLAP) Tools have been developed for processing, analysis, interpretation and visualization of multidimensional data stored in data warehouses (DW), aiming at improving strategy, tactics, and operations in enterprises [1]. *Drill-down, roll-up, slice-and-dice* and *pivoting* are the most relevant operations supported by OLAP tools such as Oracle BI [2] and Mondrian [3]. These operations require costly joins among huge fact and dimension tables, grouping and filtering, then impairing the query processing performance [1]. View materialization and indices improve this performance [4][5]. However, to the best of our knowledge, none of existing OLAP tools adopts exclusively the bitmap join index [6] to efficiently process OLAP operations. Therefore, in this paper we introduce the Bitmap Join Index OLAP Tool (BJIn OLAP Tool) whose major contributions are:

- It is a novel OLAP Tool that enables *drill-down, roll-up, slice-and-dice* and *pivoting* OLAP operations over DW, based intrinsically on the bitmap join index;
- It is a scalable software for voluminous DW (differently from Mondrian OLAP Server) and for high cardinality attributes (differently from bitmap join indices of Oracle BI [2]);
- It is interoperable with existing open-source technologies.

This paper is organized as follows. Section 2 summarizes the technical background necessary to comprehend this paper. Section 3 introduces the BJIn OLAP Tool. Section 4 discusses the validation of our tool throughout an experimental evaluation. Section 5 concludes the paper and addresses future work.

2. Technical Background

The DW is an integrated, time-variant, non-volatile and multidimensional database often implemented as a star schema with one fact table and several dimension tables [4]. The fact table maintains business measurements and foreign keys to dimension tables that contextualize facts with descriptive attributes. Hierarchies of attributes enable data aggregation on different granularity levels. Fig. 1 describes a star schema for a retail

application, derived from the Star Schema Benchmark (SSB) [7]. A *drill-down* operation decomposes fact data to lower levels of a defined hierarchy, then increasing data details. Inversely, a *roll-up* operation aggregates fact data to upper levels of a defined hierarchy, then summarizing data. The *slice-and-dice* operation applies filters over dimensions, filtering results. Finally, the *pivoting* operation enables reordering results by switching columns and rows axis of the *cross table* where results are organized [1].

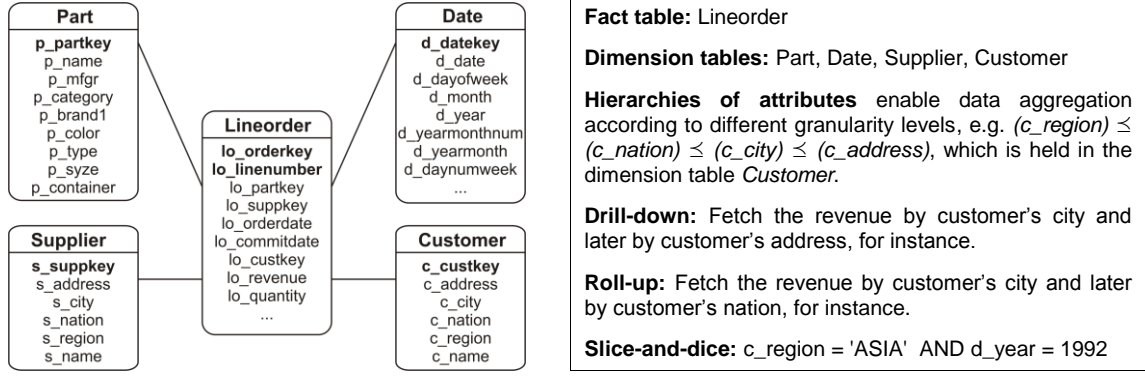


Fig. 1 A star schema of a retail DW application

The costly way to process a query over a DW is to join tables of the star schema and then perform filters, groupings and sorting (i.e. to process the *star-join*) [1]. Materialized views and indices are used to enhance the query processing performance in DW [4][5]. A *materialized view* stores pre-computed data from a fact table that was joined to dimension tables, and whose measures were aggregated. As a result, joins and groupings are avoided and the number of rows is reduced, benefiting filters. For instance, a materialized view can be created as $G_{SUM}(lo_revenue)(\Pi_{c_region, s_nation, d_year, lo_revenue}(Customer \bowtie Lineorder \bowtie Supplier \bowtie Date))$. On the other hand, the *bitmap index* [5] builds one bit-vector for each distinct value v of the indexed attribute A . The attribute cardinality, $|A|$, is the number of distinct values of A and determines the quantity of bit-vectors. All bit-vectors have as many bits as the number of rows found in the indexed table. If for the i -th record of the table we have that $A = v$, then the i -th bit of the bit-vector built for value v is set to 1. Otherwise, the bit is set to 0. Queries are efficiently processed by performing bit-wise operations among bit-vectors, and binning, encoding and compression techniques minimize the drawbacks of indexing high cardinality attributes [5]. Currently, FastBit [8] is an efficient implementation of the bitmap index. Furthermore, a *bitmap join index* [6] on the attribute B of a dimension table indicates the set of rows of the fact table to be joined with a certain value of B . Then, each bit determines the rows of the fact table where a given value of B exists. For instance, the attribute c_region is indexed by applying $\Pi_{c_region}(Customer \bowtie Lineorder)$ and then building the bit-vectors. Once the index is built, queries are processed by accessing the index, avoiding joins. In this context, our BJIn OLAP Tool was validated through performance tests that compared its *bitmap join indices* implemented using FastBit to both the *materialized views* and the *star-join* used by other OLAP tools or DBMSs.

3. The Bitmap Join Index OLAP Tool

The BJIn OLAP Tool is a novel open source OLAP server written in Java that accesses bitmap join indices to speed up *drill-down*, *roll-up*, *slice-and-dice* and *pivoting* OLAP operations. On the server side, it operates both the DBMS and the FastBit to build indices and to issue queries over them. On the client side, the user browses Java Server Pages to

submit queries to the server, and to analyze multidimensional data that are rendered on cross tables and charts produced by the Open Ajax Toolkit. Sections 3.1 and 3.2 describe building and query processing operations of the BJIn OLAP Tool. We encourage the reader to visit the BJIn OLAP Tool Portal by referring to <http://gbd.dc.ufscar.br/bjinolap/>.

3.1. Building Bitmap Join Indices

Fig. 2a illustrates the data flows to build the bitmap join indices using the BJIn OLAP Tool. Firstly, the user installs and executes the Mondrian Schema Workbench and specifies the attributes to be indexed as well as dimension and fact tables, measures, and hierarchies of attributes that exist in the DW schema (flow 1). Star schemas and snowflake schemas are currently supported. These inputs are validated by the Workbench checking the DW schema (flow 2). If the validation is successful, the Workbench generates a XML file that specifies the DW schema and the attributes to be indexed (flow 3). This file is uploaded by the user (flow 4). Once uploaded, the XML file is read by the BJIn OLAP Tool, which issues SQL and dump commands on the DBMS (flow 5) in order to compute necessary joins to build the index, as described in Section 2. The produced temporary table is dumped to a set of CSV (comma-separated values) files that are stored into the BJIn OLAP Directory (flow 6). Then, our tool issues *ardea* and *ibis* commands to FastBit (flow 7). The former reads CSV files to transform data into binary format (flow 8), while the latter effectively builds the bitmap index and stores it into the Directory (flow 9). Finally, the BJIn OLAP Tool writes a log file into the Directory with a complete description of the building operation (flow 10). Note that the reuse of Workbench promotes the interoperability between Mondrian OLAP Server and BJIn OLAP Tool, since the produced XML document can be used by both.

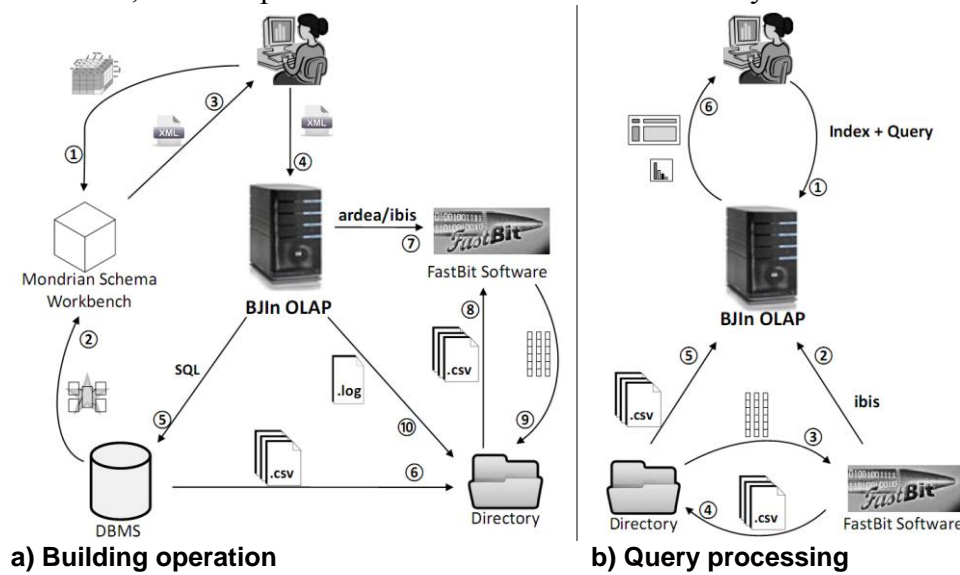


Fig. 2 The BJIn OLAP Tool and the manipulation of indices and software

3.2. Query Processing

The query language is composed of only SELECT-FROM-WHERE clauses, and does not require joins or group-by clauses. The columns listed in the SELECT clause are used to aggregate results, and the *slice-and-dice* operation is described as restrictions in the WHERE clause. Once the index is built, it is available for query processing according to the data flows shown in Fig. 2b. Firstly, the user chooses the index to be queried, and types the desired query (flow 1). Then, the BJIn OLAP Tool submits the proper *ibis* command containing the

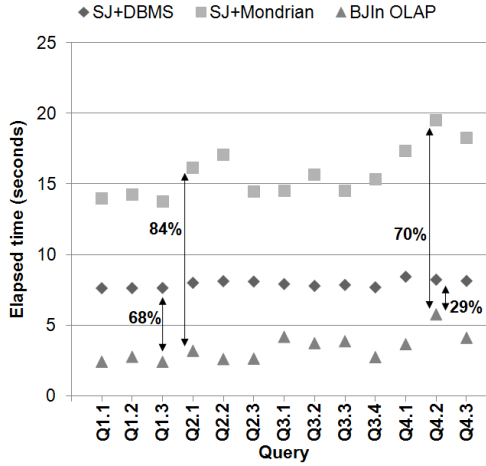
query and the chosen index to FastBit (flow 2). Thereafter, FastBit accesses the index to process the query (flow 3). After processing the query, a CSV file containing the query results is written in the Directory (flow 4). This file is read by our tool to build the cross table and render it on Java Server Pages (flow 5), which are displayed to the user (flow 6). Charts are also available for visualization. After a query execution, besides typing another query the user is able to perform OLAP operations. Once results are displayed, the user can drag and drop columns or rows to switch the axis for *pivoting* operations processed on client side. *Drill-down* and *roll-up* operations are also allowed if there is at least one hierarchy of attributes involved in the previous query. For instance, if the previous query involved the *s_nation* attribute, a combo-box enables the attribute *s_region* for *roll-up* operation, and the attributes *s_city* and *s_address* for *drill-down* operation. Furthermore, *roll-up* and *drill-down* operations are executed on the server side, and correspond to issuing a new query (flows 1 to 6). However, since results of the previous query were cached by the server and contain partial results of the new query, the performance is benefited.

4. Experimental Evaluation

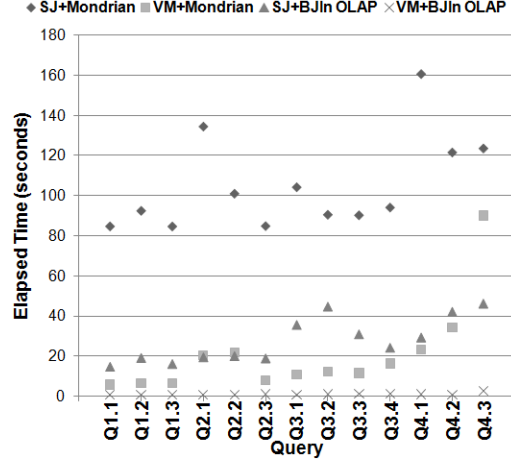
To conduct the validation of our novel BJIn OLAP Tool, we built two synthetic datasets as shown in Fig. 1 using the Star Schema Benchmark (SSB) [7]. The DW1 dataset had scale factor 1 and produced 6 million tuples in the fact table, while the DW10 dataset had scale factor 10 and then was 10 times more voluminous than DW1. We used also a real dataset, namely *webpide*, which is a star schema for educational data with 9 dimensions and 830,000 facts. All tests were performed locally to avoid network latency. We gathered the average elapsed time of 5 queries. We utilized a computer with an Athlon™ II X2 2.8 GHz processor, 320 GB SATA 7200 RPM hard drive, and 2 GB of primary memory. The operating system was Fedora 14 with Kernel 2.6.35.6-45.fc14.x86_64, with FastBit1.2.3, PostgreSQL 9.0.4, JDK 1.6.0_25, Apache Tomcat 7.0.12, Open Ajax Toolkit 2.8, Mondrian Schema Workbench 3.2.0 and Mondrian OLAP Server 3.2.1.13885.

4.1. Querying the synthetic datasets DW1 and DW10

Firstly, we assessed the SSB's four groups of queries Q1, Q2, Q3 and Q4 [7]. Each group determines an intrinsic number of joins and filters, as well as groupings and sorting. The filters in the WHERE clause enable *slice-and-dice* operations. The tests considered the following configurations to execute the queries over the DW1 dataset: *SJ+DBMS* was the DBMS computing the star-join; *SJ+Mondrian* was the Mondrian OLAP Server computing the star-join; and *BJIn OLAP* avoided joins by accessing bitmap join indices. All SSB queries were issued, and the system cache was flushed after the execution of each query. The elapsed times to process each query were reported in Fig. 3a, and the minimum and maximum time reductions that the BJIn OLAP Tool produced over Mondrian and over the DBMS were highlighted. The time reduction determines how much faster our tool was than other configurations. Clearly, the BJIn OLAP Tool remarkably outperformed the other configurations, corroborating the use of the bitmap join index even when indexing high cardinality attributes, such as $|lo_revenue| = 3,345,588$ in query groups Q2, Q3 and Q4 that were issued over the DW1 dataset. Also, the results confirmed the feasibility of using our tool to improve the query processing performance of the *slice-and-dice* operation. On the other hand, Mondrian drastically impaired the query processing performance as it mapped MDX (Multidimensional Expressions) to SQL, accessed the DBMS to perform queries and had also to prepare Java Server Pages and render cross tables.



a) Results on DW1 dataset



b) Results on DW10 dataset

Fig. 3 Results on SSB's queries for DW1 and DW10 datasets.

Secondly, we created one specific materialized view for each SSB query over DW10, drastically reducing the quantity of tuples and attributes' cardinalities. We assessed our BJIn OLAP Tool for its scalability, performing new experiments with a greater data volume and the following configurations: *SJ+Mondrian* computed the star-join; *MV+Mondrian* avoided joins by accessing specific materialized views (i.e. Aggregate Tables); *SJ+BJIn OLAP* avoided joins by accessing bitmap join indices built on the DW10 dataset; and *MV+BJIn OLAP* avoided joins by accessing the indices built on the materialized views. The results were reported in Fig. 3b. Notably, Mondrian did not scale well to perform the star-join over the voluminous DW10 dataset. Therefore, materialized views were essential for Mondrian when the data volume increased. In most cases the response time was longer for bitmap join indices built over the DW10 dataset (*SJ+BJIn OLAP*) than for Mondrian accessing materialized views (*MV+Mondrian*). However, the indices built over the materialized views (*MV+BJIn OLAP*) provided a query response time shorter than 3 seconds for every query, outperforming all the other configurations. These results revealed that our tool scaled better than Mondrian when processing queries over voluminous DW. Regarding storage, the DW10 dataset occupied 8.47 GB, all materialized views occupied a sum of 3.84 GB, and the bitmap join indices on materialized views occupied a sum of 3.81 GB. As a result, the user could simply drop materialized views and keep only indices, obtaining a similar storage requirement and a much shorter query response time for queries.

Thirdly, *drill-down* and *roll-up* operations were evaluated according to SSB's query groups Q3 and Q4. In this test we used DW1 and DW10 datasets, the Mondrian and the BJIn OLAP Tool configurations for the star-join (prefix SJ) and the materialized views (prefix MV). We executed the queries of each group consecutively, without flushing the system cache between each query, to allow the cache use and therefore to rapidly fetch partial results of the query. The results reported in Table 1 and Table 2 revealed that bitmap join indices on the DW1 dataset were more efficient than both Mondrian computing the star-join and Mondrian accessing materialized views. These indices were not more efficient than Mondrian accessing materialized views built over the DW10 dataset. However, the indices built over these materialized views provided much shorter query response times than Mondrian accessing the same materialized views. The Time Reduction columns show how much faster BJIn OLAP Tool was than the best result of Mondrian. Our tool overcame the

latter from 83% to 93% (on Q3 and Q4 roll-up operations respectively). These experiments corroborated the use of the BJIn OLAP Tool to perform *drill-down* and *roll-up* operations.

Table 1 Roll-up and drill-down operations over the DW1 dataset.

		SJ+Mondrian (s)	MV+Mondrian (s)	SJ+BJIn OLAP (s)	Time Reduction
Q3	Roll-up	63.1602	15.9788	10.6118	33.59 %
	Drill-down	55.758	14.1906	9.2894	34.54 %
Q4	Roll-up	36.33	28.6996	12.765	55.52 %
	Drill-down	39.0544	28.3402	11.2476	60.31 %

Table 2 Roll-up and drill-down operations over the DW10 dataset.

		MV+Mondrian (s)	SJ+BJIn OLAP (s)	MV+BJIn OLAP (s)	Time Reduction
Q3	Roll-up	43.2386	139.361	6.9762	83.87 %
	Drill-down	43.2702	158.882	5.936	86.28 %
Q4	Roll-up	223.8478	132.1088	15.21	93.21 %
	Drill-down	224.6458	138.7696	16.8696	92.49 %

4.2. Querying the real dataset webpide

We issued two queries over the *webpide* dataset: “Q1: Show the average score on Math for 4th grade students enrolled in State Schools located in the Brazilian southwest region in 2001 and 2003, by ethnic group, by sex, whose houses (do not) have water supply”; and “Q2: Show the average score on Math for 3rd high school students enrolled in State, Federal and Private schools in 1997, 2001 and 2003”. Results are given in Tables 3 and 4. The BJIn OLAP Tool was 73% to 79% faster than Mondrian. These results corroborate the feasibility of adopting our tool on real world applications, even for less voluminous DW.

Table 3 Elapsed time for webpide Q1

	Mondrian	BJIn OLAP	Time Reduction
Q1	6.6422	1.7900	73.05%

Table 4 Elapsed time for webpide Q2

	Mondrian	BJIn OLAP	Time Reduction
Q2	6.6376	1.3802	79.21%

5. Conclusions and Future Work

In this paper, we described the novel BJIn OLAP Tool to efficiently perform *drill-down*, *roll-up*, *slice-and-dice* and *pivoting* OLAP operations over DW, by accessing bitmap join indices. The BJIn OLAP Tool was validated through performance tests that corroborated its efficiency and feasibility. It is compatible to PostgreSQL, MySQL and IBM DB2®. As future work, we will investigate fact constellation schemas and *drill-across* operations.

References

- [1] Chaudhuri, S. and Dayal, U. (1997) “An overview of data warehousing and OLAP technology”, *SIGMOD Rec.*, vol. 26, pp. 65-74.
- [2] Fogel, S., et al. (2010) “Oracle 11g Database Administrator’s Guide”.
- [3] Casters, M., Bouman, R. and Dongen, J. (2010) “Pentaho® Kettle Solutions”, Sybex.
- [4] Harinarayan, V., Rajaraman, A., and Ullman, J. (1996) “Implementing data cubes efficiently”, *SIGMOD Rec.*, v. 25, pp. 205-216.
- [5] Stockinger, K. and Wu, K. (2006) “Bitmap Indices for Data Warehouses”, *Data Warehouses and OLAP*, R. Wrembel and C. Koncilia, eds., IRM Press, pp. 157-178.
- [6] O’Neil, P. and Graefe, G. (1995) “Multi-table joins through bitmapped join indices”, *SIGMOD Rec.*, vol. 24, 1995, pp. 8-11.
- [7] O’Neil, P. et al. (2009) “The Star Schema Benchmark and Augmented Fact Table Indexing”, *TPC Technical Conference*, Springer-Verlag, 2009, pp. 237-252.
- [8] Rübel, O., et al. (2009) “FastBit: interactively searching massive data”, *Journal of Physics: Conference Series*, vol. 180, pp. 12053.