A Data Mining Approach for Selecting Bitmap Join Indices

Ladjel Bellatreche
LISI/ENSMA - Poitiers University - France
bellatreche@ensma.fr

Rokia Missaoui
University of Quebec in Outaouais - Canada
Rokia.Missaoui@uqo.ca

Hamid Necir, Habiba Drias
Institut National d'Informatique - Algeria
{h_necir, h_drias}@ini.dz

Index selection is one of the most important decisions to take in the physical design of relational data warehouses. Indices reduce significantly the cost of processing complex OLAP queries, but require storage cost and induce maintenance overhead. Two main types of indices are available: mono-attribute indices (e.g., B-tree, bitmap, hash, etc.) and multi-attribute indices (join indices, bitmap join indices). To optimize star join queries characterized by joins between a large fact table and multiple dimension tables and selections on dimension tables, bitmap join indices are well adapted. They require less storage cost due to their binary representation. However, selecting these indices is a difficult task due to the exponential number of candidate attributes to be indexed. Most of approaches for index selection follow two main steps: (1) pruning the search space (*i.e.*, reducing the number of candidate attributes) and (2) selecting indices using the pruned search space. In this paper, we first propose a data mining driven approach to prune the search space of bitmap join index selection problem. As opposed to an existing our technique that only uses frequency of attributes in queries as a pruning metric, our technique uses not only frequencies, but also other parameters such as the size of dimension tables involved in the indexing process, size of each dimension tuple, and page size on disk. We then define a greedy algorithm to select bitmap join indices that minimize processing cost and verify storage constraint. Finally, in order to evaluate the efficiency of our approach, we compare it with some existing techniques.

Categories and Subject Descriptors: Abstract Interpretation [**Programming Language**]:

General Terms:
Additional Key Words and Phrases: Bitmap Join Indices, Frequent Itemset, Data Mining, Star Join Queries, Relational Data Warehouses

---

1. INTRODUCTION

A data warehouse stores large volumes of aggregated and non volatile data across entire organization. It is usually accessed through complex queries for key business operations. Data warehouses are frequently modelled using relational schemas like star and snowflake schemas. A star schema consists of a single large fact table that is related to multiple dimension tables via foreign keys. Dimension tables are relatively small compared to the fact table and rarely updated. They are typically non normalized so that the number of needed join operations are reduced. Operations in data warehouse applications are mostly read ones and are dominated by large and complex queries. The typical queries on the star schema are called *star join queries*. They are characterized by: (i) a multi-table join among a large fact table and dimension tables, (ii) each one of the dimension tables involved in the join operation has multiple selection predicates[1] on its descriptive attributes, and (iii) there is no join operation between dimension tables. Due to the interactive nature of decision support applications, having a fast query response time is a critical performance goal.

Without efficient optimization techniques, queries addressed to data warehouses may take hours or days, which is unacceptable in most cases [Chatziantoniou and Ross 2007]. In order to cope with complex and time-consuming decision support queries, there is an urgent need for efficient and sophisticated physical design techniques [Chaudhuri and Narasayya 2007]. The major bottleneck in evaluating such queries is the *join* between a large fact table and the surrounding dimension tables [Stöhr et al. 2000]. To optimize join operations in both OLTP and OLAP environments, many techniques have been proposed that we can classify into two main categories: (1) *non redundant structures* and (2) *redundant structures*. The first category concerns different implementations of the join operation: nested loop, sort merge join and hash join. These structures are efficient when (a) the size of joined tables is reasonable, which is not the case of tables in relational data warehouses and (b) join concerns two tables [Golfarelli et al. 2002]. Redundant structures, like materialized views [Gupta 1999; Rizzi and Saltarelli 2003] and join indices [Valduriez 1987], are more efficient to speed up join operation involving many tables [Oneil 1995]. Their main drawbacks are the extra storage requirement and the maintenance overhead. However, such optimization techniques are inevitable in data warehouse environments. The main peculiarity of indices is the fact that they may be combined with other optimization techniques such as materialized views, horizontal and vertical partitioning. In other words, any access path with a table structure can be indexed.

This paper focuses on the selection of join bitmap indices (BJIs) and is organized as follows. Section 2 provides a background on indices and data mining. Section 3 presents our formalization of the problem of selecting BJIs as an optimization problem, and provides a description of the unique and most related work by showing its limitations using a running example. Section 4 presents our two-step approach for first generating BJI candidates using data mining based pruning algorithms, and

[1] A selection predicate has the following form: $D_i.A_j\ \theta\ value$, where A_j is an attribute of dimension table D_i and θ is one of the six comparison operators $\{=, <, >, \leq, \geq\}$, and *value* is a constant belonging to the domain of attribute A_j).

then selecting the final BJIs using a greedy algorithm. Section 5 validates our work through experimentations. Finally, Section 6 concludes the paper by summarizing the main results and suggesting new directions.

2. BACKGROUND

In the following, we review three key topics related to our proposal: (i) indexing techniques, (ii) index selection problem, and (iii) frequent closed itemset computation.

2.1 Indexing Techniques

A number of indexing strategies have been suggested for data warehouses. They can be classified into two categories: single-table indices and (b) multiple table indices. In the first category, indices are defined either on one or several attributes of a single table. An index may be either clustered or non-clustered. In the second one, indices are defined on two or more tables. A number of indexing strategies belonging to these two categories have been suggested for data warehouses: value-list index, projection index [O'Neil and Quass 1997], bitmap index [Chan and Ioannidis 1998; O'Neil and Quass 1997], bit-sliced index [Chan and Ioannidis 1998], data index [Datta et al. 1999], join index [Valduriez 1987], star join Index [Systems 1997] and bitmap join indices [O'Neil and Graefe 1995].

—*Value-list index and Bitmap index.* A value-list index consists of two parts. The first part is a balanced tree structure and the second part is a mapping scheme. The mapping scheme is attached to the leaf nodes of the tree structure and points to the tuples in the table being indexed. The tree is generally a B-tree with varying percentages of utilization. Oracle provides a B*-tree whose utilization may go to 100%. Two different types of mapping schemes are in use. First, one consists of a RowID (row identifier) list, which is associated with each unique search-key value. This list is partitioned into a number of disk blocks chained together. The second scheme uses bitmaps. A bitmap is a vector of bits whose value depend on predicate values. A bitmap B lists all rows with a given predicate P such that for each row r with ordinal number j that satisfies the predicate P, the j^{th} bit in B is set to 1. Bitmaps represent efficiently low-cardinality data. However to make this indexing scheme practical for high-cardinality data, compression techniques must be used. Value-list indices have been shown in [O'Neil and Quass 1997] to outperform other access methods in queries involving MIN or MAX aggregate functions, as well as queries that compute percentile values of a given column. Bitmap indices can substantially improve performance of queries with the following characteristics [Chee-Yong 1999]:

  —The WHERE clause contains multiple predicates on low-or-medium-cardinality columns (e.g., a predicate on Gender that has two possible values: female or male or a predicate on city with three possible values as shown in Figure 1).

  —Bitmap indices have been created on some or all of these low-or-medium-cardinality columns.

  Besides disk saving (due to the binary representation and possible compression [Johnson 1999]), bitmap indices speed up queries having Boolean operations (such

Customer

RID^C	CID	Name	Age	City
1	616	Gilles	20	LA
2	515	Yves	42	Paris
3	414	Patrick	21	Tokyo
4	313	Didier	52	Tokyo
5	212	Eric	18	LA
6	111	Pascal	17	LA

Bitmap index on City

RID	LA	Paris	Tokyo
1	1	0	0
2	0	1	0
3	0	0	1
4	0	0	1
5	1	0	0
6	1	0	0

Figure 1. An Example of a bitmap index defined on City.

Customer

RID^C	CID	Name	Age	City
1	616	Gilles	20	LA
2	515	Yves	42	Paris
3	414	Patrick	21	Tokyo
4	313	Didier	52	Tokyo
5	212	Eric	18	LA
6	111	Pascal	17	LA

Projected Index on Age

Age
20
42
21
52
18
17

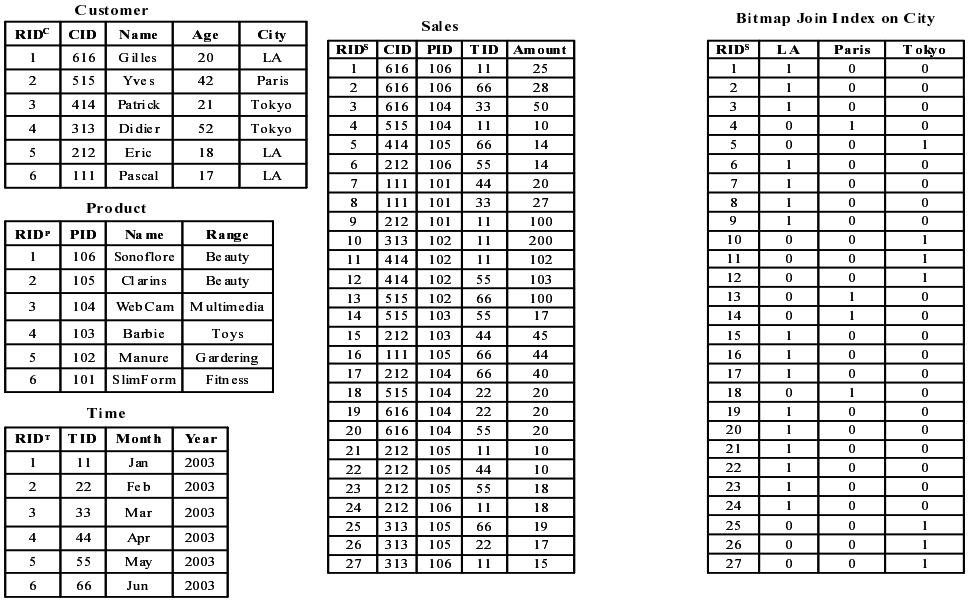
Figure 2. An Example of a projection index.

as AND, OR and NOT) and COUNT operations. They are supported by most of commercial DBMSs (Oracle, SQL Server, etc.).

—*Projection Index.* A projection index is equivalent to the column being indexed. If C is the column being indexed, then the projection index on C consists of a stored sequence of column values from C in the same order as the ordinal row number in the table from where the values are extracted (see Figure 2). It has been shown in [O'Neil and Quass 1997] that projection indices outperform other indexing schemes for the execution of queries that involve computation on two or more column values, and seem to perform acceptably well in GROUP-BY queries.

—*Bit-sliced Index* A bit sliced index represents the key values of the column to be indexed as binary numbers and projects a set of bitmap slices, which are orthogonal to the data, held in the projection index. This index has been shown in [Chee-Yong 1999] to particularly perform well for computing sums and averages. Also, it outperforms other indexing approaches for percentile queries when the underlying data is clustered, and for range queries whose range is large.

—*Join Index.* A join index pre-computes a join operation between two relations [Valduriez 1987]. It is the result of joining two tables on join attributes and projecting the keys (or tuple IDs) of the two tables. A join index is used to find the matching tuples from the tables to be joined. This index has been proposed in the context of traditional databases in medium of 80's, and was later extended



Customer

RID^C	CID	Name	Age	City
1	616	Gilles	20	LA
2	515	Yves	42	Paris
3	414	Patrick	21	Tokyo
4	313	Didier	52	Tokyo
5	212	Eric	18	LA
6	111	Pascal	17	LA

Product

RID^P	PID	Name	Range
1	106	Sonoflore	Beauty
2	105	Clarins	Beauty
3	104	WebCam	Multimedia
4	103	Barbie	Toys
5	102	Manure	Gardering
6	101	SlimForm	Fitness

Time

RID^T	TID	Month	Year
1	11	Jan	2003
2	22	Feb	2003
3	33	Mar	2003
4	44	Apr	2003
5	55	May	2003
6	66	Jun	2003

Sales

RID^S	CID	PID	TID	Amount
1	616	106	11	25
2	616	106	66	28
3	616	104	33	50
4	515	104	11	10
5	414	105	66	14
6	212	106	55	14
7	111	101	44	20
8	111	101	33	27
9	212	101	11	100
10	313	102	11	200
11	414	102	11	102
12	414	102	55	103
13	515	102	66	100
14	515	103	55	17
15	212	103	44	45
16	111	105	66	44
17	212	104	66	40
18	515	104	22	20
19	616	104	22	20
20	616	104	55	20
21	212	105	11	10
22	212	105	44	10
23	212	105	55	18
24	212	106	11	18
25	313	105	66	19
26	313	105	22	17
27	313	106	11	15

Bitmap Join Index on City

RID^S	LA	Paris	Tokyo
1	1	0	0
2	1	0	0
3	1	0	0
4	0	1	0
5	0	0	1
6	1	0	0
7	1	0	0
8	1	0	0
9	1	0	0
10	0	0	1
11	0	0	1
12	0	0	1
13	0	1	0
14	0	1	0
15	1	0	0
16	1	0	0
17	1	0	0
18	0	1	0
19	1	0	0
20	1	0	0
21	1	0	0
22	1	0	0
23	1	0	0
24	1	0	0
25	0	0	1
26	0	0	1
27	0	0	1

Figure 3. An Example of a bitmap join index.

by Red Brick Systems [Systems 1997] to a multi-table join index in relational data warehouses by concatenating columns from different dimension tables and listing RowIDs in the fact table from each concatenated value.

—*Data Index.* A data index, like the projection index, exploits the positional indexing strategy [Datta et al. 1999]. The Data index avoids duplication of data by storing only the index and not the column being indexed. The data index can be of two specific types: basic data index and join data index (for more information, see [Datta et al. 1999]).

—*Bitmap Join Index.* It is a multi-table join index and a variant of join index. It is defined as a bitmap index on a table R based on a single column of a table S, where S commonly joins with R in a specific way. Bitmap join indices are supported by most of commercial database/warehouse systems (e.g., Oracle, SQL Server, etc.).

Example 1. *To illustrate the use and interest of bitmap join indices, let us consider the following data (see Figure 3) that will serve as a running example throughout the paper. Suppose we have a relational data warehouse represented by three dimension tables (TIME, CUSTOMERS and PRODUCTS) and one fact table (SALES). Let Q be the query that the administrator would like to optimize:*

```
SELECT Count(*)
FROM CUSTOMERS C, SALES S
WHEERE C.City='Poitiers'
AND C.CID=S.CID
```

To do so, he/she creates a bitmap join index SC_IDX between the fact table SALES and the dimension table CUSTOMERS based on City attribute as follows:

```
CREATE BITMAP INDEX SC_IDX
ON SALES(CUSTOMERS.City)
FROM SALES S, CUSTOMERS C
WHERE S.CID= C.CID
```

To execute the above query, the query optimizer just accesses the bitmap corresponding to the column representing *Poitiers*, without joining SALES and CUSTOMERS tables.

This example shows the efficiency of bitmap join indices for executing this kind of queries.

2.2 Index Selection

Index selection problem (ISP) has been studied since the early 70's and its importance is well recognized. It is one of the most important issues in physical design of advanced database applications. The task of index selection consists in automatically selecting an appropriate set of indices for a data warehouse and a workload under resource constraints (storage, maintenance, etc.). It is a challenging problem for the following reasons [Chaudhuri 2004]: The size of a relational data warehouse schema may be large (many tables with several columns) for real applications, and indices can be defined on a set of columns from different tables (multiple table indices). A large spectrum of research studies has been proposed to deal with this problem. By exploring the state of the art, we believe that most studies concentrate on single table index selection. To the best of our knowledge, only two studies have been proposed for dealing with multiple table index selection problem [Aouiche et al. 2005; Bellatreche et al. 2007]. On the industry side, several index selection tools were developed (e.g., *AutoAdmin* [Chaudhuri and Narasayya 1998] for self-tuning and self-administering databases and data warehouses).

In single table index selection problem, most of existing approaches use two main phases: (1) *generation of attribute candidates* and (2) *selection of a final configuration*. The first phase prunes the search space of index selection problem by eliminating some *non relevant* attributes. In the second phase, the final indices are selected using heuristics like greedy algorithms [Chaudhuri and Narasayya 1997], linear programming algorithms [Chaudhuri 2004], etc. The *quality of the final set of indices depends essentially on the pruning phase*. To prune the search space of index candidates, many approaches were proposed [Chaudhuri and Narasayya 1997; Valentin et al. 2000; Chaudhuri 2004; Labio et al. 1997; Aouiche et al. 2005] using enumeration-driven heuristics. For instance, [Chaudhuri and Narasayya 1997] propose a greedy algorithm which uses the optimizer cost estimate in SQL Server to decide the goodness of a given configuration of indices. The weakness of this work is that it *imposes the number of generated candidates*. DB2 Advisor is another example belonging to this category [Valentin et al. 2000], where the query parser is used to pick up selection attributes used in workload queries. The generated candidates are obtained by a few *simple combinations* of selection attributes [Valentin et al. 2000].

2.3 Frequent Closed Itemset Computation

Let $\mathcal{I} = \{i_1, i_2, \cdots, i_m\}$ be a set of m distinct items (table attributes). A transaction (tuple) T contains a set of items in $\mathcal{I}$, and has an associated unique identifier called *TID*. A subset X of $\mathcal{I}$ where $k = |X|$ is referred to as a $k-$itemset (or simply an itemset), and k is called the length of X. A transaction database (TDB), say $\mathcal{D}$, is a set of transactions. The number of transactions in $\mathcal{D}$ that contain an itemset X is called the *absolute support* of X whereas the fraction of transactions is called its *relative support* (both denoted by $sup(X)$). Thus, an itemset is frequent (or large) when $sup(X)$ reaches at least a user-specified minimum threshold called *minsup*.

One of the most frequently used technique in data mining is rule mining which is conducted in Apriori-like algorithms [Agrawal and Srikant 1994] in two steps: detection of all frequent itemsets, and utilization of frequent itemsets to generate association rules (e.g., $X \Rightarrow Y$) that have a confidence $\geq$ *minconf*. While the second step is relatively easy and cost-effective, the first one presents a great challenge because the set of frequent itemsets (FIs) may grow exponentially with the whole set of items. To reduce the size of the FI set, some studies were conducted on frequent closed itemsets FCIs and maximal frequent itemsets (i.e., itemsets for which every superset is infrequent [Burdick et al. 2001]).

The Close algorithm [Pasquier et al. 1999] is one of the first procedures for FCI generation. Like Apriori, it performs a level-wise computation within the powerset lattice. However, it exploits the notion of generators of FCIs [2] to compute closed itemsets. ChARM [Zaki and Hsiao 2002] is another procedure which generates FCIs in a tree organized according to inclusion. The computation of the closure and the support is based on a efficient storage and manipulation of *TID-sets* (i.e., the set of transactions per item). Closure computation is accelerated using *diffsets*, the set difference on the TID-sets of a given node and its unique parent node in the tree.

Closet [Han et al. 2000] and its variant Closet+ [Wang et al. 2003] both generate FCIs as maximal branches of a *FP-tree*, a structure that is basically a prefix tree (or *trie*) augmented with transversal lists of pointers.

In the next section, we follow the above mentioned steps to select bitmap join indices: generation of attribute candidates and selection of a final configuration. The first step will be based on the computation of frequent closed itemsets using either Close or ChARM.

3. BITMAP JOIN INDEX SELECTION PROBLEM

To ease the understanding of our proposed approach, we provide a motivation and a formulation of the BJI selection problem. Then, we present and discuss the most related work.

As indicated earlier, a BJI is defined on one or several non key dimension attributes with low cardinality [3] (called indexable columns) for joining dimension tables with the fact table. An indexable attribute A_j of a given dimension table D_i for a BJI is a column $D_i.A_j$ such that there is a condition of the form $D_i.A_j \ \theta \ Expression$

[2] A generator of an FCI Y is a subset of that itemset such that its closure is equal to Y.

[3] The domain of this attribute should be an enumerated domain like *gender*, *color*, etc.

in the WHERE clause. The operator θ must be among $\{=, <, >, \leq, \geq\}$.

Let $A = \{A_1, A_2, ¨A_K\}$ be the set of indexed candidate attributes for BJIs. Then, the possible number of BJIs with *only one* group of attributes, grows exponentially with K, and is given by:

$$\binom{K}{1} + \binom{K}{2} + ... + \binom{K}{K} = 2^K - 1 \quad (1)$$

For $K = 4$, this number is 15. The possible number of BJIs with any combination of attribute groups is given by:

$$\binom{2^K-1}{1} + \binom{2^K-1}{2} + ... + \binom{2^K-1}{2^K-1} = 2^{2^K-1} \quad (2)$$

For $K = 4$, the number of possible cases is (2^{15}). Therefore, the problem of efficiently finding the set of BJIs that minimizes the total query processing cost while satisfying a storage constraint cannot be handled by first enumerating all possible BJIs and then computing the query cost for each candidate BJI. As a consequence, BJI selection problem can be formulated as follows:
Given a data warehouse with a set of dimension tables $D = \{D_1, D_2, ..., D_d\}$ and a fact table F, a workload Q of queries $Q = \{Q_1, Q_2, ..., Q_m\}$, where each query Q_i $(1 \leq i \leq m)$ has an access frequency, and a storage constraint S, the aim of BJI selection problem is to find a set of BJIs among a pre-computed subset of all possible candidates, which minimizes the query processing cost and satisfies the storage requirements S.

The single work dealing with this problem using a data mining approach is the one proposed by Aouiche et al. [Aouiche et al. 2005]. Intuitively, selecting bitmap join indices means partitioning the set of indexable attribute A into disjoint groups. This motivates the authors in [Aouiche et al. 2005] to propose an approach based on data mining (and Close algorithm [Pasquier et al. 1999]) to prune the search space by computing the set of frequent closed attribute groups rather than all the possible combinations indicated above. The groups generated in the pruning step are then used in the second step to select the final configuration of indices.

Aouiche et al.'s work has two main limitations: (1) no formalization is proposed, (2) the proposed approach only uses *frequencies of attributes* to generate frequent closed itemsets, where each one represents an attribute set to be indexed. The frequency parameter is *not sufficient to be a pruning metric*. Since in the vertical partitioning of databases [4] the frequency parameter is not sufficient in getting a good fragmentation schema [Fung et al. 2003], we believe that a similar observation holds for the bitmap join index selection problem. Indeed, Fun et al. [Fung et al. 2003] showed the weakness of affinity-based algorithms (that use only query frequencies) in reducing the query processing cost. To get a better result, they recommend the use of other parameters, like the size of tables, the length of each tuple, the size of disk page, etc.

To overcome these limitations, we propose new pruning algorithms, called DynaClose and DynaCharm that take into account the above parameters as part of the pruning metric, since the cost of join operations depends heavily on the size of joined tables [Getoor et al. 2001]. Once the pruning phase is processed, a greedy

[4] Each fragment contains a subset of attributes that are frequently used together.

Table I. Query description.

(1) select S.channel_id, sum(S.quantity_sold) from S, C where S.channel_id=C.channel_id and **C.channel_desc='Internet'** group by S.channel_id
(2) select S.channel_id, sum(S.quantity_sold), sum(S.amount_sold) from S, C where S.channel_id=C.channel_id and **C.channel_desc ='Catalog'** group by S.channel_id
(3) select S.channel_id, sum(S.quantity_sold),sum(S.amount_sold) from S, C where S.channel_id=C.channel_id and **C.channel_desc ='Partners'** group by S.channel_id
(4) select S.cust_id, avg(quantity_sold) from S, C where S.cust_id=C.cust_id and C.cust_gender='M' group by S.cust_id
(5) select S.cust_id, avg(quantity_sold) from S, C where S.cust_id=C.cust_id and *C.cust_gender='F'* group by S.cust_id

algorithm is executed to select a set of BJSs that reduces the query processing cost and satisfies the storage constraint.

3.1 Illustration of Weakness of the Existing Pruning Approach

To show the limitations of Aouiche's approach in pruning search space of bitmap join indices, we consider the following example with a part of a star schema (used in our experimental study) containing two dimension tables *CHANNELS* (denoted by Ch) and *CUSTOMERS* (denoted by C) and a fact table *SALES* (denoted by S). The cardinalities of these tables (number of instances) are: $||SALES|| = 16260336, ||CHANNELS|| = 5$ and $||CUSTOMERS|| = 50000$. Let us assume that five queries are most frequently executed on the corresponding data cube (see Table I). In the above queries, two main join operations are used: one between SALES and CUSTOMERS ($J_1 : SALES \bowtie CUSTOMERS$), and another one between SALES and CHANNELS ($J_2 : SALES \bowtie CHANNELS$). Basically, the cost of J_1 is higher than J_2 since the size of CUSTOMERS (50 000 instances) is larger than the size of CHANNELS (5 instances).

With $minsup = 3$ (in absolute value), the solution returned by [Aouiche et al. 2005] is a bitmap join index (that we call *sales_desc_bjix*) defined on CHANNELS and SALES using *channel_desc* attribute. This is due to the fact that there are three occurrences of the same selection predicate defined on that attribute in the five queries. However, no bitmap join index is proposed between SALES and CUSTOMERS since the *cust_gender* attribute is not so frequent as *minsup*. As a consequence, *only* the join J_2 will be optimized, but not the global query set.

To overcome this limitation, we enrich the pruning function by considering other parameters like the size of tables, the length of an instance of tables, and page size.

4. THE PROPOSED APPROACH

Given a star schema with a set of dimension tables $\{D_1, D_2, ..., D_l\}$ and a fact table F. Let $||T_i||$ and LC_i be the cardinality of a (dimension or fact) table T_i and the length of an instance of D_i, respectively. The number of disk pages occupied by a table T, denoted by $|T|$ is calculated as follows:

$$|T| = \left\lceil \frac{||T|| \times LT}{PS} \right\rceil \tag{3}$$

where PS represents the page size (in bytes) on disk.

In order to select a set of bitmap join indices minimizing the overall query processing cost and satisfying the storage constraint S, we consider an approach with two steps commonly found in the classical index selection approaches: (1) generation of candidate attributes, and (2) selection of the final configuration and BJIs. These steps will be described in the following subsections.

4.1 Generation of Candidate Attributes

The input of this step is the context matrix. It is constructed using the set of queries $Q = \{Q_1, Q_2, ..., Q_m\}$ and a set of indexable attributes $A = \{A_1, A_2, ..., A_l\}$. The matrix has rows and columns that represent queries and indexable attributes, respectively. A value of this matrix is given by:

$$Uses(Q_i, A_j) = \begin{cases} 1 \text{ if query } Q_i \text{ uses a selection predicate defined on } A_j \\ 0 \text{ otherwise} \end{cases}$$

Example 2. *Recall that the size of CUSTOMERS, CHANNELS and SALES is: 50 000, 5 and 16 260 33 instances, respectively. Instance length of CUSTOMERS, CHANNELS and SALES is: 24, 24 and 36, respectively, and page size PS = 65536. We consider the same five queries as defined in Table 1. To facilitate the construction of the context matrix, we rename the indexable attributes as follows: Sales.cust_id = A_1, Customers.cust_id = A_2; Customers.cust_gender = A_3, Channels.channel_id = A_4, Sales.channel_id = A_5; Channels.channel_desc = A_6. The matrix is given below.*

	A_1	A_2	A_3	A_4	A_5	A_6
Q_1	0	0	0	1	1	1
Q_2	0	0	0	1	1	1
Q_3	0	0	0	1	1	1
Q_4	1	1	1	0	0	0
Q_5	1	1	1	0	0	0
Support	$\frac{2}{5}$	$\frac{2}{5}$	$\frac{2}{5}$	$\frac{3}{5}$	$\frac{3}{5}$	$\frac{3}{5}$

To prune the search space of the BJI selection problem, we propose two algorithms: *DynaClose* and *DynaCharm* which are an adaptation of Close [Pasquier et al. 1999] and ChARM [Zaki and Hsiao 2002], respectively. We used Close because it has been exploited by Aouiche et al. [Bellatreche and Boukhalfa 2005] for the same purpose (selecting BJIs), and hence will allow us to better compare the performance of our approach against Aouiche's approach (see Section 5 for more details). We also used ChARM because it is a well-known and efficient algorithm for frequent closed itemset generation. Our adaptation concerns especially the pruning metric which is different from the one used in Close and ChARM, where only frequencies of attributes are used. In order to prune the search space, we propose a new metric $Fitness(X)$, called fitness metric on a FCI X, that penalizes the FCIs defined on small dimension tables.

$$Fitness(X) = \frac{1}{n} \times (\sum_{i=1}^{n} sup_i \times \alpha_i) \quad (4)$$

where n represents the number of non-key attributes A_i in X, α_i is equal to $\frac{|D_j|}{|F|}$, where $|D_j|$ and $|F|$ represent the number of pages needed by the dimension table

D_j that includes A_i and the fact table F, respectively. The elements sup_i is the support of A_i.

Given a *minsup*, $minfit$ (minimal fitness value) can be computed as follows:

$$minfit = \frac{minsup}{|F|} \times \left\lceil (\sum_{j=1}^{d} \frac{|D_j|}{d}) \right\rceil \tag{5}$$

EXAMPLE 3. *From our trivial example, two closed itemsets may be generated:* $CFI_1 = \{A_1, A_2, A_3\}$ *with a support equal to 0.4 and* $CFI_2 = \{A_3, A_5, A_6\}$ *with 0.6 support. With* $minsup = 0.6$*, Aouiche's approach will retain* $CFI_2 = \{A_3, A_5, A_6\}$ *and hence suggest a BJI on A*6 *to match the facts in Sales with records in Channels according to the values of* channel_desc*. To illustrate our pruning technique, we first compute the fitness function for each one of the generated CFIs.*

$$\begin{aligned} Fitness(CFI_1) &= sup(A_3) \times \alpha_3 \\ &= \frac{2}{5} \times \frac{\left\lceil \frac{50000 \times 24}{65536} \right\rceil}{\left\lceil \frac{||1626033|| \times 36}{65536} \right\rceil} \\ &= \frac{2}{5} \times \frac{19}{894} \\ &= 0.0085 \end{aligned}$$

One may notice that CFI_1 *contains only one non-key attribute which is* A_3*, and hence* $Fitness(CFI_1)$ *will take into account* A_3 *only.* $Fitness(CFI_2)$ *is computed based on* A_6 *only and is equal to 0.00067. The value of* $minfit$ *is 0.0067 (=* $\frac{0.6}{894} \times \frac{19+1}{2}$*). Our pruning metric will then select* CFI_1 *rather than* CFI_2*.*

4.2 Elimination of Dirty FCIs

The set of the FCIs generated by DynaClose and DynaCharm must be purified to avoid the erroneous bitmap join indices. Let us recall that a bitmap join index is built between a fact table and dimension table(s) based on non-key attributes. Therefore, in its definition, we should find key-attributes and non-key attributes. There are two main requirements of bitmap join indices: (1) if a bitmap join index is defined on k ($k > 1$) attributes of the same dimension table, it shall have only one join operation between this table and the fact table. (2) if a bitmap join index is defined on k ($k > 1$) attributes of distinct dimension tables, it shall have k join operations between this table and the fact table. Based on these requirements, we distinguish three scenarios that lead to a purification (i.e., the elimination of the concerned FCI):

(1) The FCI contains only key attributes (of dimension tables) or only foreign keys of the fact table.

(2) The FCI has a number of key attributes significantly higher than the number of non-key attributes. For instance, the case of a FCI (customers.cust_gender, sales.cust_id, customers.cust_id, sales.prod_id, products.prod _id) having three keys and one non-key attribute. For one non-key attribute, we only need two key attributes for building BJIs. Generally, for N selection attributes, $2 \times N$ key attributes are required.

(3) The FCI is composed of only non-key attributes.

After this elimination, we get a set of candidate dimension attributes for selecting bitmap join indices, denoted by CA ($CA \subseteq A$), where A is the initial set of indexable attributes. CA is the union of all attributes belonging to the purified FCIs. To select the final configuration of bitmap join indices, we consider only CA as described in the following section.

4.3 Selection of the Final Configuration

Now, we have the set of all candidate attributes needed to build the final configuration of BJIs using a greedy algorithm. The input of this algorithm covers: (a) a star schema with a fact table and a set of dimension tables, (b) a set of queries: $Q = \{Q_1, Q_2, \cdots, Q_p\}$, (c) CA and (d) a storage constraint S. Our algorithm starts with a configuration having a bitmap index defined on an attribute of CA. Since the cardinality of CA may be large, we choose the first attribute as the one with the lowest cardinality, let say, I_{min}. This choice assumes a sort of elements in CA according to their cardinality. Then, the procedure iteratively improves the initial configuration by considering other attributes of CA while the constraint on S is maintained and a further reduction in the total query processing cost is possible. To measure efficiency of a configuration, we use a mathematical cost model, an adaptation of an existing one [Aouiche et al. 2005] that computes the number of disk page accesses (IO costs) when executing the set of queries. In order to estimate storage cost required for a bitmap join index, we adapt our cost model developed in [Bellatreche et al. 2000]. The main steps of our approach are given in Algorithm 1.

Algorithm 1 Greedy Algorithm for BJIs Selection

Input :
CA, Q, S(storage bound)
Output:
$Config$: set of selected BJIs.
Notations:
BJI_j: a bitmap join index defined on attribute A_j.
$Size(BJI_j)$: storage cost required for BJI_j
begin
$SCA = SORT(CA)$; /* a sequence of attributes */
$Config = BJI_{min}$;
$S := S - Size(BJI_{min})$;
$SCA := SCA - A_{min}$; A_{min} *is the attribute used to defined* BJI_{min}
WHILE ($Size(Config) \leq S$) DO
FOR each next element A_j in SCA DO
IF ($COST[Q, (Config \cup BJI_j))] < COST[Q, Config]$)
AND (($Size(Config \cup BJI_j) \leq S$)) THEN
$Config := Config \cup BJI_j$;
$Size(Config) := Size(Config) + Size(BJI_j)$;
$SCA := SCA - A_j$; /* remove A_j from SCA */
end

5. EXPERIMENTAL STUDY

To evaluate our approach, we first implemented all algorithms (Close, DynaClose, Charm, DynaCharm and the greedy algorithm) using Java language and a Pentium IV with 512 MB of memory. We conducted several experimentations using the same dataset and the forty OLAP queries as in [Aouiche et al. 2005]. The star schema of the data warehouse has one fact table SALES and five dimension tables: TIME, CUSTOMERS, PRODUCTS, PROMOTIONS and CHANNELS (see Tables II).

Table II. Table cardinalities used in the experiments.

Tables	**Number of rows**
SALES	16 260 336
CUSTOMERS	50 000
PRODUCTS	10 000
TIME	1 461
PROMOTIONS	501
CHANNELS	5

The experiments were conducted according to four scenarios: (1) identification of the value of minsup that gives an important number of FCIs, (2) evaluation of different approaches (Close, DynaClose, Charm and DynaCharm) by executing the forty queries on non indexed tables without considering storage constraint, (3) evaluation of different approaches by considering the storage constraint, and (4) computation of CPU bound of different approaches.

First, we carried out experiments to set the appropriate value of minsup that allows the generation of a large set of FCIs. The results show that the appropriate minsup value should be set to 0.05.

5.1 Evaluation without Storage Constraint

Figure 4 shows how different indexing approaches reduce the cost of executing the forty queries with an increasing number of minimum support. The main result is that DynaClose outperforms approaches for almost all values of minsup. However, its performance deteriorates (in the sense that no candidate indices can be generated) when the minsup value becomes high. We notice that for minsup values exceeding 0.475, Close, Charm and DynaCharm stop generating new FCIs, and hence the query processing cost remains stable.

A comparison between DynaCharm and DynaClose shows that they have a similar performance for small minsup values (ranging between 0.05 and 0.0175). These results coincide with the experimental study of Zaki et al. [Zaki and Hsiao 2002]. However, as we increase minsup, the performance gap between DynaClose and DynaCharm becomes larger. This is due to the fact that DynaCharm processes branches in a depth-first fashion, and FCIs are formed only at the end of an iteration.

5.2 Evaluation of DynaClose et Dynacharm with Storage Constraint

The set of BJIs generated by DynaClose and Dynacharm for a minsup=0.05 requires storage of 146,88 MB (see Figure 6). This value is very high if we compare it with

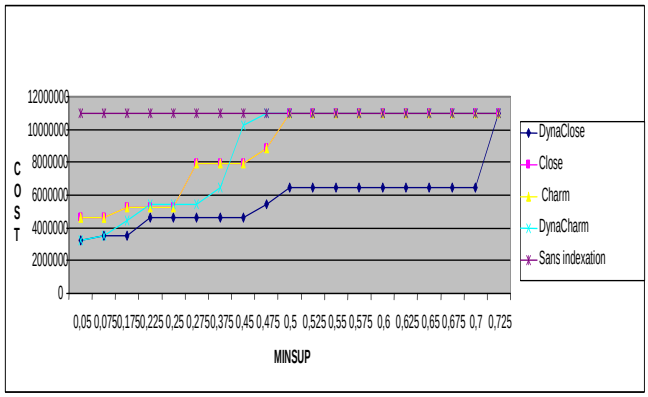


Figure 4. Quality of our approaches.

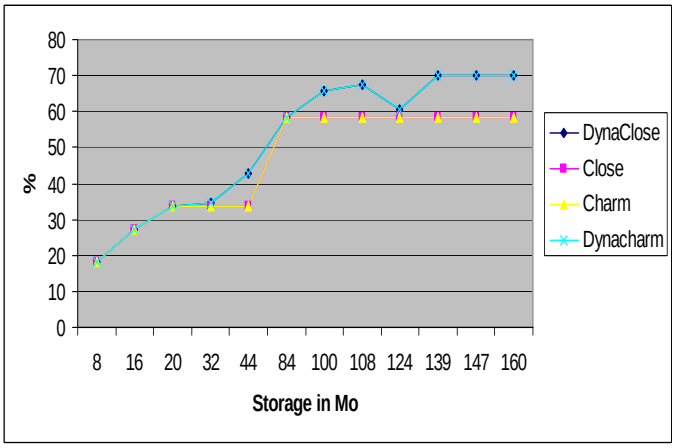


Figure 5. Behavior of our approaches according to storage constraint.

the size of the fact table which is 372,17 MB. Consequently, we execute our greedy algorithm for selecting BJIs by considering various storage values with a fixed value of minsup equal to 0.05. This value allows the generation of a large number of index candidates. Figure 5 shows that Dynaclose and Dynacharm improve the performance with a gain of 43% (compared to the solution without indexing) for 44 MB of storage (almost 3 times smaller than the initial space (146,88 MB)). With the same storage, Charm and Close give a 33,56% gain. Therefore, our proposed variants Dynaclose and Dynacharm provide a better performance than the traditional approach for all the values of the considered storage, except for the storage space 84 MB (where all approaches provide the same gain of 58,19%).

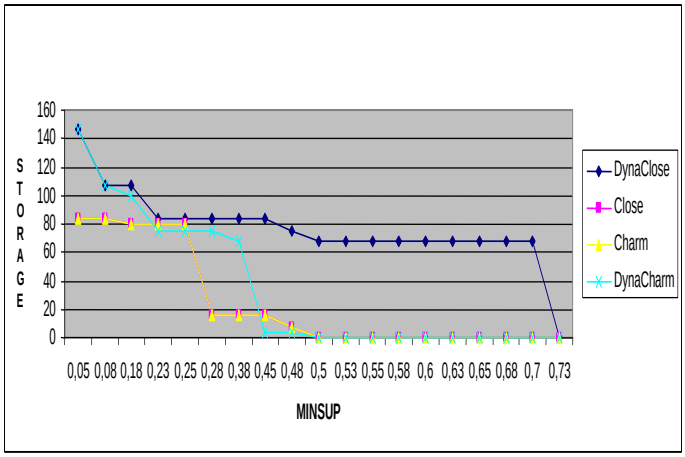


Figure 6. Storage vs. Minsup.

5.3 Evaluation of Different Approaches based on CPU Bound

We have conducted experiments about the execution time of each algorithm (in microseconds) according to a varying value of minsup. Table 3 shows that DynaCharm and DynaClose need more execution time to prune the search space compared to Charm and Close. This result was foreseeable since contrary to traditional approaches (Closed and Charm) which prune according to the minsup, our two approaches take more time since the pruning phase involves the computation of the fitness function for each generated FCI. The results show that DynaCharm is the approach which requires the highest time for almost all minsup values that we consider, but it remains stable for high minsup. This is due to the pruning phase of DynaCharm since it is carried out only when we get maximized closed itemsets.

Table III. Execution Time (in milliseconds) of Different Approaches.

Minsup	*DynaClose*	*Close*	*Charm*	*DynaCharm*
0, 04	2794	3079	3424	2984
0, 075	2473	2518	3121	2961
0, 175	2403	2491	2708	3109
0, 225	2158	1574	1540	3865
0, 25	2210	1680	1504	3457
0, 275	2220	2063	1043	3830
0, 375	2113	1737	1046	4181
0, 45	2195	1730	683	4168
0, 475	1985	1319	288	4000
0, 5	1910	1294	303	3758
0, 7	1945	1294	303	3758

6. CONCLUSION

In this paper, we have explored the problem of indexing in relational data warehousing, where we distinguish between two main categories: single table indices and multiple table indices. Single table index selection problem has been largely studied in traditional database compared to multiple table indices. We motivate the use of multiple table indices in relational data warehouses to speed up complex queries requiring join operations. We concentrate on bitmap join indices which are a case of multiple join indices. First, we formalize the problem of selecting a set of bitmap join indices subject to storage constraint. To deal with this problem, we propose an approach based on rule mining, and more specifically on frequent closed itemset generation. Our approach proceeds in two main steps: (1) it uses data mining procedures, called DynaClose and DynaCharm (adaptations of Close and Charm algorithms) to prune the search space of the bitmap join index selection problem by identifying frequent closed itemsets that represent candidate indexable attributes, and (2) it uses a greedy algorithm to select the final configuration of indices. The main peculiarity of our pruning approach, compared to the existing approaches where only frequency of dimension attributes (in queries) is used, lies in the utilization of other parameters like table size, page size, and record size. Once the pruning phase is executed, we exploit our greedy algorithm that uses a cost model to generate the appropriate set of indices. Our approach was validated through experimentation. Additional and large scale experiments are underway to check our findings.

We plan to extend this work into two directions: (i) study the dynamic computation of BJIs when significant changes occur in the input, and (ii) handle the problem of view materialization in data warehouses using a similar approach based on data mining and cost model estimation.

ACKNOWLEDGMENTS

The authors are grateful to the NSERC for financial support and to Kamel Boukhalfa, a Ph.D. student at Poitiers University for his help in finalizing this work.

REFERENCES

Agrawal, R. and Srikant, R. 1994. Fast algorithms for mining association rules. *20th International Conference on Very Large Data Bases (VLDB94)*, 487–499.

Aouiche, K., Boussaid, O., and Bentayeb, F. 2005. Automatic Selection of Bitmap Join Indexes in Data Warehouses. *7th International Conference on Data Warehousing and Knowledge Discovery (DAWAK05)*, 64–73.

Bellatreche, L. and Boukhalfa, K. 2005. An evolutionary approach to schema partitioning selection in a data warehouse environment. *Proceeding of the International Conference on Data Warehousing and Knowledge Discovery (DAWAK'2005)*, 115–125.

Bellatreche, L., Karlapalem, K., and Li, Q. 2000. Evaluation of indexing materialized views in data warehousing environments. *Proceeding of the International Conference on Data Warehousing and Knowledge Discovery (DAWAK'2000)*, 57–66.

Bellatreche, L., Missaoui, R., Necir, H., and Drias, H. 2007. Selection and pruning algorithms for bitmap index selection problem using data mining. In *International Conference on Data Warehousing and Knowledge Discovery (DaWaK2007)*. 221–230.

Burdick, D., Calimlim, M., and Gehrke, J. 2001. Mafia: a maximal frequent itemset algorithm for transactional databases. *ICDE01*, 443–452.

Chan, C. Y. and Ioannidis, Y. E. 1998. Bitmap index design and evaluation. *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 355–366.

Chatziantoniou, D. and Ross, K. A. 2007. Partitioned optimization of complex queries. *Information Systems 32,* 2 (April), 248–282.

Chaudhuri, S. 2004. Index selection for databases: A hardness study and a principled heuristic solution. *IEEE Transactions on Knowledge and Data Engineering 16,* 11 (November), 1313–1323.

Chaudhuri, S. and Narasayya, V. 1997. An efficient cost-driven index selection tool for microsoft sql server. *Proceedings of the International Conference on Very Large Databases*, 146–155.

Chaudhuri, S. and Narasayya, V. 1998. Autoadmin 'what-if' index analysis utility. *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 367–378.

Chaudhuri, S. and Narasayya, V. 2007. Self-tuning database systems: A decade of progress. *Proceedings of the International Conference on Very Large Databases*, 3–14.

Chee-Yong, C. 1999. Indexing techniques in decision support systems. Phd. thesis, University of Wisconsin - Madison.

Datta, A., Ramamritham, K., and Thomas, H. 1999. Curio: A novel solution for efficient storage and indexing in data warehouses. *Proceedings of the International Conference on Very Large Databases*, 730–733.

Fung, C.-H., Karlapalem, K., and Li, Q. 2003. Cost-driven vertical class partitioning for methods in object oriented databases. *VLDB Journal 12,* 3 (December), 187–210.

Getoor, L., Taskar, B., and Koller, D. 2001. Selectivity estimation using probabilistic models. *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 461–472.

Golfarelli, M., , and Rizzi, S. Saltarelli, E. 2002. Index selection for data warehousing. *Proceedings 4th International Workshop on Design and Management of Data Warehouses (DMDW'2002), Toronto, Canada*, 33–42.

Gupta, H. 1999. Selection and maintenance of views in a data warehouse. Ph.d. thesis, Stanford University. September.

Han, J., Pei, J., and Yin, Y. 2000. Mining Frequent Patterns without Candidate Generation. In *Proceedings of the ACM-SIGMOD 2000 Conference, Dallas, Texas, USA*. 1–12.

Johnson, T. 1999. Performance measurements of compressed bitmap indices. *Proceedings of the International Conference on Very Large Databases*, 278–289.

Labio, W., Quass, D., and Adelberg, B. 1997. Physical database design for data warehouses. *Proceedings of the International Conference on Data Engineering (ICDE)*.

Oneil, P. 1995. Multi-table joins through bitmapped join indioces. *SIGMOD 24,* 03.

O'Neil, P. and Graefe, G. 1995. Multi-table joins through bitmapped join indices. *SIGMOD Record 24,* 3 (September), 8–11.

O'Neil, P. and Quass, D. 1997. Improved query performance with variant indexes. *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 38–49.

Pasquier, N., Bastide, Y., Taouil, R., and Lakhal, L. 1999. Discovering frequent closed itemsets. *ICDT*, 398–416.

Rizzi, S. and Saltarelli, E. 2003. View materialization vs. indexing : Balancing space constraints in data warehouse design. *in 15th International Conference on Advanced Information Systems Engineering (CAiSE2003)*, 502–519.

Stöhr, T., Märtens, H., and Rahm, E. 2000. Multi-dimensional database allocation for parallel data warehouses. *Proceedings of the International Conference on Very Large Databases*, 273–284.

Systems, R. B. 1997. Star schema processing for complex queries. *White Paper*.

Valduriez, P. 1987. Join indices. *ACM Transactions on Database Systems 12,* 2 (June), 218–246.

Valentin, G., Zuliani, M., Zilio, D. C., Lohman, G. M., and Skelley, A. 2000. Db2 advisor: An optimizer smart enough to recommend its own indexes. *ICDE00*, 101–110.

Wang, J., Han, J., and Pei, J. 2003. Closet+: searching for the best strategies for mining frequent closed itemsets. *in Proceedings of international conference on Knowledge discovery and data mining (ACM SIGKDD03)*, 236–245.

Zaki, M. J. and Hsiao, C. 2002. Charm: An efficient algorithm for closed itemset mining. *In Proceeding of the 2nd SIAM International Conference on Data Mining (ICDM02).*

Ladjel Bellatreche received the engineer diploma in Computer Science (CS) from Sidi Bel Abbes University Algeria in 1992, the Master in CS from Paul-Sabatier University Toulouse France in 1994 and Ph.D. in CS from Blaise Pascal University Clermont Ferrand France. He is an Assistant Professor at Poitiers University - France since 2002. He has been a visiting researcher at Hong Kong University of Science and Technology from 1996 to 1999 and at Purdue University - USA during summer 2001. He has worked extensively in the areas: distributed databases, data warehousing, ontology-based data integration and ontology-based databases. He has published more than 70 research papers in these areas in leading international journals and conferences. Ladjel has been associated with many conferences as program committee member.

Rokia Missaoui received her Ph.D. degree in Computer Science from Université de Montréal in 1988. She has been a full Professor in the Department of Computer Science and Engineering at UQO (Université du Québec en Outaouais) since August 2002. Before joining UQO, she was a professor at UQAM (Université du Québec à Montréal) between 1987 and 2002. While at the beginning of her career she was concerned with performance and design issues in databases, her current research interests include data mining, formal concept analysis (mainly concept lattice construction and manipulation, and association rule mining), data warehousing, as well as content-based image retrieval and mining.

Hamid Necir received his master degree in Computer Science from Bab Ezzouar University - Algeria in December 2001. In 2002, he joined Scientific Research Center for the development of the Arabic language in Algiers. He worked on arabic language processing. He is currently doing his Ph.D in Bab Ezzouar University. His main research interests are data mining and data warehousing.

Habiba Drias received the master degree in Computer Science from Case Western Reserve University, Cleveland OHIO USA (1984) and the Ph.D. from Paris 6 University and Algiers USTHB University (1993). She has directed the Computer Science Institute of USTHB and then the Laboratory of Research in Artificial Intelligence for many years. She has published more than a hundred of papers in the areas: artificial intelligence, e-commerce, computational complexity and the satisifiability problem. She is currently the president of the Algerian National Institute of Informatics INI-.