Automatic Selection of Bitmap Join Indexes in Data Warehouses Using CFPGrowth++ Algorithm

Mohammed Yahyaoui[*], Noura Aknin, Souad Amjad, Lamia Benameur
Information Technology and Modeling, Systems Research Unit, FS, Abdelmalek Essaadi University, Tetouan, Morocco
E-mail: myahyaoui@uae.ac.ma, noura.aknin@uae.ac.ma, s.amjad@uae.ac.ma, l.benameur@uae.ac.ma
[*]Corresponding author

Keywords: data warehouse, OLAP, indexes, bitmap join indexes, frequent itemsets, data mining, CFPGrowth++ algorithm

Received: December 12, 2024

In the context of complex data warehousing, Typically, the analysis and decision-making process for Data Warehouses schematized in a relational star model is conducted through OLAP (On-Line Analytical Processing) queries. These queries are generally complex, characterized by several operations of selections, joins, grouping and aggregations on voluminous tables. Which requires a lot of computing time and therefore a very high response time. The cost of running OLAP decision queries on large tables is very high. The reduction of this cost becomes essential to allow decision-makers to interact within a reasonable time frame. The objective of this study is to enhance system performance by minimizing the response time of OLAP decision-making queries. The approach proposed in this article aims to search for frequent patterns for the automatic selection of binary join indexes used for reducing the execution costs of OLAP decision-making queries. To automatically generate the configuration of binary join indexes minimizing response time, an implementation of the CFPGrowth++ frequent pattern matching algorithm was well carried out and then applied to a load of queries on a test Data Warehouse created using the Analytical Processing Benchmark 1 (ABP-1) test bench, in order to validate our approach. The results of the experiment indicate that the index configuration produced by the proposed approach leads to a significant improvement in performance improvement of approximately 75%. We note that for a large portion of the load, execution time is significantly improved after applying our approach. The overall query execution time decreased compared to the general context. The overall execution time for queries decreased from 20,032.57 seconds before the application of our approach to 5,388.49 seconds after applying our approach. The experiments carried out show that the index configuration generated by the proposed approach allows a very performance gain.

Povzetek: Predlagana metoda samodejne izbire binarnih indeksov za OLAP z algoritmom CFPGrowth++ zmanjša čas izvajanja poizvedb za približno 75 %.

1 Introduction

In decision-making computing, Data Warehouses have still experienced a very significant boom today. They are fed by analytical data supporting a set of analysis processes coming from different distributed and heterogeneous sources [1][2][3][4] and their volumes are destined to increase.

In 1990, Bill Inmon conceptualized the Data Warehouse as a comprehensive and dynamic data repository designed to be subject-oriented, integrated, time-variant, and non-volatile, serving as a valuable resource for management decision-making [5].

In the data warehouse, information is structured in the configuration of a multidimensional cube [4]. Here, each dimension acts as an analytical axis, and each cell encapsulates the analyzed fact [6]. This organization is specifically designed to proficiently facilitate the operations of online analysis [7], commonly known as OLAP (online analytical processing) [4].

The ROLAP (Relational On-Line Analytical Processing) relational model is introduced for physical storage.

Data warehouses are usually modeled by a star schema [8]. This schema is characterized by a fact table very voluminous (Giga or Terra Bytes) linked by foreign keys to a set of smaller size dimension tables thus forming a star schema (Figure 1).

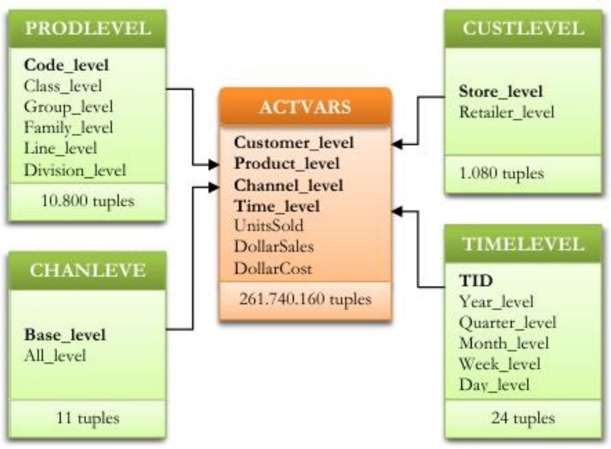

Figure 1 : The relational model for storing a data warehouse.

Queries formulated within a star schema exhibit significant complexity, involving the integration of joins, selections, and aggregations. Specifically, these joins, denoted as star joins, traverse the entirety of the fact table. Selection operations in this context, referred to as selection predicates, are executed on dimension attributes identified as selection attributes. Achieving satisfactory performance in the context of intricate decisional queries and their extended response times poses a significant challenge for Data Warehouse administrators [9]. Effectively addressing this challenge requires administrators to possess a robust understanding of optimization structures [10] and to apply logical and physical design methods for selecting the most optimal design policy.

Within the fact table, attributes comprise both activity measures and foreign keys leading to dimension tables [11]. This model's advantage lies in its utilization of pre-existing database management systems, resulting in a reduction of implementation costs. However, the challenge persists, as optimizing the performance of Data Warehouses remains an increasingly critical concern.

In this study, the concept of optimal performance specifically refers to the reduction of query execution costs in OLAP (Online Analytical Processing) environments, with a particular focus on minimizing query response time. Since decision-support queries in data warehouses often involve complex join operations over large volumes of data, improving response time is crucial for maintaining system efficiency and user satisfaction [12].

While other performance aspects such as storage space, index maintenance overhead, and system scalability are important considerations in data warehouse optimization, this paper primarily targets response time reduction as the key performance metric. The proposed approach achieves this by selecting effective binary join indexes based on frequent pattern discovery, using the CFPGrowth++ algorithm with multiple support thresholds to enhance query processing efficiency.

Performance optimization in data warehouses is primarily concerned with reducing the query response time for decision-support queries. This allows users to quickly obtain the information they need, improving the overall efficiency of analysis and decision-making. In this work this is achieved by minimizing the query response time for decision-support (OLAP) through efficient Selection of binary join indexes.

Indexing is one of the important techniques used in the physical design phase to optimize OLAP queries [13] in relational Data Warehouses. In this research paper, we're using an indexing approach.

The proposed approach is designed to assist Data Warehouse administrators and system designers in making informed decisions regarding index selection to optimize query performance.

Although the solution is partially automated - as it automatically extracts frequent query patterns and suggests candidate binary join indexes - human expertise is still valuable in the final decision-making process. Administrators need to have a basic understanding of the data warehouse schema, query workload characteristics, and index management principles to evaluate the relevance of the recommended indexes before implementation.

In practice, this approach serves as a decision-support tool rather than a fully autonomous system. It aims to reduce the complexity of index selection by narrowing down the most relevant candidates based on query patterns, leaving administrators with clearer, data-driven recommendations.

Some of these techniques inherit from those proposed in the context of traditional databases such as B trees [14][15]. Others are proposed to optimize selections defined on low cardinality attributes such as binary indexes.

Binary join indexes have demonstrated their utility in mitigating the execution costs of decisional OLAP queries formulated on a relational star schema. [13].

The task of index selection is categorized as an NP-Complete problem [16] due to its inherent complexity, stemming from the fact that the number of potential indexes grows exponentially with the total number of attributes present in the Data Warehouse [17][18][19].

Given the aforementioned challenge, our focus lies in the automatic selection of a set of indexes (configuration) minimizing the cost of executing OLAP queries.

By mining frequent itemsets from OLAP query workloads, we can identify the most commonly accessed attribute combinations. Creating indexes based on these patterns ensures that the most relevant data retrieval paths are optimized, leading to faster query execution, reduced response times, and more efficient resource utilization.

In the context of OLAP (Online Analytical Processing), queries often involve analyzing large datasets with multiple dimensions and measures. To make these queries faster, databases use indexes special data structures that help retrieve data quickly.

Now, frequent itemsets come from data mining techniques, especially association rule mining. They identify combinations of items or attribute values that appear together often in the data.

Here's how these frequent itemsets can help optimize OLAP query execution:

Identifying Common Patterns:

By discovering which attribute combinations occur frequently, the system can prioritize creating indexes on those combinations. This means that when a query involves these attributes, the data warehouses can quickly locate relevant data without scanning the entire dataset.

Selective Index Creation:

Instead of indexing every possible combination (which can be costly), the system uses frequent itemsets to select only the most relevant attribute combinations. This targeted approach ensures efficient use of storage and maintenance resources.

Improved Query Performance:

When an OLAP query involves dimensions that match these frequent itemsets, the pre-existing indexes allow for rapid data retrieval, significantly reducing query response times.

Dynamic Optimization:

As data evolves, the system can periodically re-analyze data to find new frequent itemsets, updating indexes accordingly. This dynamic approach ensures that the indexing strategy remains aligned with actual usage patterns.

Leveraging frequent itemsets helps in smart index selection by focusing on the most commonly co-occurring attribute combinations. This targeted indexing accelerates OLAP queries, making data analysis more efficient and responsive.

The decision to adopt CFPGrowth++ over alternative frequent itemset mining algorithms is based on several key advantages that align with the specific requirements of OLAP index selection:

Multiple Minimum Support Thresholds:

Unlike classical algorithms such as Apriori, CLOS, or the original FP-Growth, which operate with a single global support threshold, CFPGrowth++ introduces the capability to assign different minimum support thresholds to individual items. This flexibility is crucial in OLAP environments, where data distributions are often skewed and certain attributes or combinations appear frequently within specific query contexts but remain infrequent globally. By supporting multiple thresholds, CFPGrowth++ can identify both globally frequent and contextually important patterns, leading to a more relevant and effective set of candidate indexes.

Improved Efficiency and Scalability:

CFPGrowth++ builds upon the FP-Tree structure but optimizes it by reducing redundant node traversals and minimizing memory overhead through enhanced pruning strategies. This makes it better suited for handling the large, high-dimensional datasets commonly found in data warehouses, where conventional algorithms like Apriori suffer from high computational costs due to repeated data warehouses scans and exponential candidate generation.

Suitability for Index Selection in Decision-Support Systems:

Index selection in OLAP systems demands mining not just frequent patterns, but patterns that contribute meaningfully to query optimization. CFPGrowth++'s capacity to uncover rare yet strategically valuable itemsets thanks to its variable support thresholds enables the discovery of indexes that specifically target performance bottlenecks in diverse and heterogeneous workloads.

Reduced Computational Overhead During Mining:

Compared to algorithms CLOS, CFPGrowth++ demonstrates lower memory consumption and faster runtime for datasets with varying item frequencies. This balance between computational efficiency and mining depth makes it a practical choice for real-time or near-real-time decision-support applications.

CFPGrowth++ was chosen because it combines high efficiency, flexibility in support thresholds, scalability, and the ability to handle large, complex datasets features that are vital for effective and practical index selection in OLAP systems. Its advantages over other algorithms make it particularly well-suited for our goal of reducing query execution costs through intelligent binary join index recommendations.

The main contribution of this study is an implementation of the CFPGrowth++ frequent pattern matching algorithm To automatically generate the configuration of binary join indexes minimizing response time and select the frequent itemsets. Then for the approach validation to a load of queries applied on a test Data Warehouse created using ABP-1 test bench [20].

The article is orgnized into five sections. Section 2 poutlines the overall workflow adopted in the study, emphasizing the integration of frequent itemset mining using CFPGrowth++ with automatic binary join index (BJI) selection for OLAP query optimization. Section 3 provides an overview of the primary studies that have been advanced to tackle the intricacies of binary join index selection, with a specific emphasis on the data mining application techniques for resolution. Then, Section 4 delves into our unique approach to the selection of frequent itemsets, achieved through the adaptation of the CFPGrowth++ algorithm. Section 5 details the experimental phase, offering validation for our proposed binary join index selection methodology. Section 6 compares the method against traditional indexing approaches, Furthermore, it acknowledges practical challenges such as index maintenance, scalability for high-dimensional schemas, and the absence of a fully integrated cost model. Lastly, in Section 7, the article concludes by encapsulating the main findings and proposing avenues for future research.

2 Proposed workflow, novelty, and main findings

2.1 Proposed workflow

The proposed approach aims to enhance the performance of OLAP query execution in data warehouses by automatically selecting binary join indexes (BJIs) based on frequent pattern mining techniques. The workflow of the study is structured as follows :

1. OLAP Query Workload Collection :

Extraction of historical OLAP queries containing join and selection predicates from a simulated decision-support workload based on the ABP-1 benchmark.

2. Transaction Dataset Generation:

Transformation of each query into a transaction containing attributes involved in selection and join predicates.

3. Frequent Itemset Mining with CFPGrowth++:

Application of an improved CFPGrowth++ algorithm with multiple minimum support thresholds to extract frequent attribute combinations from the transaction dataset.

4. Binary Join Index Candidate Generation:

Mapping of frequent itemsets into candidate BJI configurations.

2.2 Novelty of the study

This work introduces several novel contributions to the domain of OLAP optimization:

- Integration of a Multi-support CFPGrowth++ Algorithm:

Unlike conventional approaches using single-support thresholds, our method applies multiple minimum supports to better capture frequent attribute combinations of varying significance.

- Automatic Binary Join Index Selection Framework:

The proposed system automates the identification and selection of BJIs from frequent query patterns, reducing the manual intervention traditionally required in index configuration.

- Statistically Validated Performance Improvement:

The study not only demonstrates query performance gains but also rigorously validates these results through statistical analysis, including confidence intervals and significance tests.

- Practical Benchmarking on a Customized ABP-1 Environment:

The experimental setup leverages a workload derived from the ABP-1 benchmark, enhanced with real-world query structures, ensuring realistic and relevant performance validation.

2.3 Main findings

Through extensive experimental evaluation:

- The proposed approach achieved a performance improvement between 74.20% and 75.96% on OLAP query workloads.
- Statistical analysis confirmed these improvements to be significant ($p < 0.001$) at a 95% confidence level.
- The approach proved effective across various query types, particularly for complex multi-join and range aggregation queries.

The comparison of Recent Methods Related to OLAP Optimization and Indexing is represented in Table 1 :

Table 1 : Summary of recent methods related to OLAP optimization and indexing

Approach	Features	Limitations
CFPGrowth++ (Proposed)	Utilizes an enhanced frequent pattern mining algorithm to identify frequent motifs, supporting multiple support thresholds for flexible exploration. Aims to reduce query execution costs in OLAP environments.	Computational complexity may increase with very large datasets; effectiveness depends on the quality of identified motifs.
Traditional Apriori-based methods	Use classic frequent pattern mining with straightforward implementation. Suitable for smaller datasets but can be computationally intensive.	Scalability issues; limited support for multiple support thresholds; less efficient for large-scale data.
FP-Growth	Efficient pattern mining without candidate generation, faster than Apriori.	Less flexible in handling multiple support thresholds; may still face challenges with very large datasets.
Existing index selection approaches	Often rely on heuristic or rule-based methods, focusing on specific query workloads.	May not adapt well to dynamic data or varying query patterns; limited in discovering complex motif structures.

This comparison illustrates that our CFPGrowth++ approach offers a flexible and efficient method for discovering frequent motifs, which can significantly improve index selection for OLAP query optimization. While it introduces some computational overhead, its ability to handle multiple support thresholds and explore complex patterns provides a notable advantage over traditional methods.

2.4 Benchmark dataset comparison

The ABP-1 test bench was selected as it offers a balanced schema structure and a set of decision-support queries typical in OLAP workloads. It also allows for controlled workload scaling and realistic index selection scenarios, making it better suited for validating index optimization approaches compared to standard synthetic benchmarks. The Benchmark Dataset Comparison is represented in Table 2 :

Table 2 : Benchmark dataset comparison

Benchmark / Dataset	Schema Complexity	Query Diversity	Realism for Decision-Support Workloads	Used in This Study
TPC-DS	High	Very High	Moderate	No
Star Schema Benchmark (SSB)	Moderate	Mediu m	Low	No
ABP-1 (Customized Version)	Moderate	High	High	Yes

3 Related works

3.1 Binary join index selection problem

The selection of indexes in Data Warehouses is a difficult problem [21] seen the large number of candidate attributes of dimension tables participant in the construction of indexes [22].

The problem consists on building an index configuration that minimizes the cost of executing a set of frequent OLAP queries.

To reduce the number of potential attributes in the construction of indexes, Numerous previous works proposed a data mining technique [23][24][17][25] in order to generate frequent itemsets (patterns) which will constitute the candidate attributes in the indexing process.

Several research works have shown the usefulness of automatic index selection [26][22] from a set of candidate indexes extracted from a query by appealing to using administrator expertise [10][21][14].In the context of Data Warehouses, binary join indexes are well suited to speed up OLAP queries known for their large number of join operations [13].

Recent approaches [10][6] propose the use of data mining techniques to generate the set of candidate indexes to reduce the significant number of potential candidate attributes for the construction of indexes. They have exploited in particular the technique of searching for frequent itemsets for the generation of candidate indexes [27][28]. The basic idea is inspired by the frequent itemsets search technique, widely used in data mining. The more frequently an attribute or group of attributes is present in the query load, the more valuable it is considered in the index selection process.

Prior work in the field of index selection, such as the studies by Aouiche et al. [27] and Bellatreche et al. [28][29], has laid important groundwork by exploring the use of frequent pattern mining techniquesspecifically, the Close algorithm to identify candidate attributes for binary join index configuration. However, these approaches have notable limitations. For instance, Aouiche et al. focus primarily on the frequency of attribute usage within query workloads, which can lead to the elimination of potentially beneficial indexes on attributes from large dimension tables that are infrequently accessed but still critical for join operations. Bellatreche et al. address this issue by incorporating additional parameters, such as table size and system page size, to refine index selection, but their method still relies on the traditional Close algorithm, which can generate an overwhelming number of frequent patterns, impacting scalability and efficiency.

Our approach explicitly addresses these limitations by introducing the CFPGrowth++ algorithm, an enhanced and scalable method for mining frequent itemsets. Unlike previous algorithms like Apriori or Close, CFPGrowth++ is designed to be faster, more efficient, and capable of handling large volumes of data with multiple support thresholds. This allows for a more nuanced and precise selection of index candidates, thereby improving the effectiveness of index configuration for OLAP query optimization.

The novelty of our approach lies in leveraging CFPGrowth++ to overcome the scalability and efficiency challenges faced by prior methods, enabling more accurate and comprehensive index selection that better supports complex, large-scale data environments. This makes our contribution both necessary and timely in advancing the state of the art in index optimization techniques.

Here is a synthetic comparison of frequent pattern discovery algorithms used in previous studies on index selection techniques.

Table 3 : Synthetic comparison of frequent pattern discovery algorithms

Benchmarks used	Apriori	FP-Growth	CLOSE	CFPGrowth ++
Complexity	High (multiple passes)	Modera te (fewer passes)	Modera te (but more filterin g)	Low (optimized)
Memory	High (candidat e generatio	Modera te (FP-tree)	Low (closed pattern s)	Low (partitioni ng)

	n)			
Methodologies	Frequent patterns	Frequent patterns	Closed patterns	Frequent patterns
Ease of implementation	Simple	Moderate	Complex	Complex
Performance outcomes	Poor	Good	Good	Very good

3.2 Approaches based on frequent itemsets

The discovery of frequent itemsets consists in finding groupings of items appearing together with a significant frequency. The discovery of these frequent itemsets is the main step in solving a number of useful knowledge extraction problems.

Formally, the binary join index selection issue is formulated as an optimization problem in the following form: given:

(1) A Data Warehouse modeled by a star schema formed by a fact table ***F*** and ***D*** = {D_1, ...,D_d} dimension tables,

(2) A set of most frequent queries ***Q*** = {q_1, ..., q_m} with their access frequencies ***f*** = {f_1, ..., f_m},

The objective is to select a set of indexes that reduces the query execution cost.

4 Proposed approach

4.1 Search for frequent itemsets for index selection

We believe that the relevance of an index is strongly correlated with the frequency of its use in the set of queries in a load.Data mining is a growing field of research aimed at extracting knowledge from enormous amounts of data.In this article, we are interested in the extraction of frequent itemsets.

The objective is to extract knowledge useful for the choice of indexes. The search for frequent itemsets is an appropriate way to account for this correlation and thus facilitate the choice of indexes to be built.

4.2 Frequent itemsets

The problem of research frequent itemsets can be formulated as follows :

Let ***I*** = {i_1, i_2, ... i_m} a set of ***m*** distinct symbols called *itemset* and ***B*** = t_1, . . , t_n a database of ***n*** transactions. Each transaction is composed of a subset of items ***I'*** ⊆ ***I***. A subset ***I'*** of size ***k*** is called a ***k****-itemset*. A transaction t_i contains a pattern ***I'*** if and only if ***I'*** ⊆ ***ti***. The support of a pattern ***I'*** is the proportion of transactions in ***B*** that contain ***I'***. The support is given by the following

$$support(I') = \frac{|\{t \in B, I' \subseteq t\}|}{|\{t \in B\}|}$$

Figure 2 : The support of a pattern.

Support defined as the absolute frequency of an itemset (or index) in the query workload.

Minimum support will refer to the threshold value that is applied during the frequent itemset mining phase to determine the minimum frequency required for an itemset to be considered frequent.

A pattern whose support is greater than or equal to the minimum threshold of minsup support, defined by the user, is called a frequent pattern.

The objective is to select a set of indexes reducing the cost of query execution.

4.3 CFPGrowth++ algorithm

The CFPGrowth++ algorithm, an extension of CFPGrowth, is devised for the extraction of frequent itemsets through the incorporation of multiple minimum support thresholds [30][31]. Its application involves inputting a transaction database alongside a list denoting minimum support threshold, wherein each threshold corresponds to the requisite minimum support for a specific item.

A transaction database in this context is characterized as a compilation of transactions, each constituting a unique list of items or symbols. For illustration, contemplate a transaction database encompassing 5 transactions (T1, T2, ..., T6) and 8 items (A, B, C, D, E, F, G, H). As an instance, transaction T1 encapsulates the set of items {A, C, D, F}. Notably, the stipulation that an item cannot recur within the same transaction is upheld, and items are presumed to be arranged in lexicographical order within a given transaction.

The list of minimum support threshold is provided as input for the algorithm :

Table 4 : Example of transactions

Transaction ID	**Items**
T1	{A, C, D,F}
T2	{A, C, E, F, G}
T3	{A, B, C, F, H}
T4	{B, F, G}
T5	{B, C}

Table 5 : The list of minimum support thresholds supported to be used for the items

Items	**Minimum support threshold**
A	1
B	2
C	3
D	3
E	2
F	3
G	2
H	1

The support of an itemset is quantified by the count of transactions that encompass said itemset. An itemset attains the status of a frequent itemset when its support equals or exceeds the most conservative minimum support threshold among the individual thresholds assigned to its constituent items. For instance, consider the itemset {A, B, H}, which attains the status of a frequent itemset as it appears in one transaction (T3). Crucially, its support surpasses the minimum support thresholds associated with its individual items: item 1, item 2, and item 8, with respective minimum support thresholds of 1, 2, and 1.

In this study, we determine multiple minimum support thresholds based on the distinct characteristics and importance of each item within the dataset. Unlike traditional frequent itemset mining approaches that rely on a single global minimum support value, our method assigns a specific minimum support threshold to each item according to its frequency and relevance in the context of OLAP query optimization. This approach ensures that both frequent and less frequent, but potentially valuable, itemsets can be identified without being excluded by a uniform threshold.

The multiple thresholds are defined prior to the frequent pattern mining process by analyzing the distribution of item frequencies within the transaction database. Items with higher occurrence rates are assigned higher minimum support values, while those with lower frequencies, yet significant in terms of query performance optimization, are given lower thresholds. This strategy allows a more flexible and adaptive mining process.

The CFPGrowth++ algorithm, which we implemented and enhanced for this work, efficiently handles these multiple support values by extending the classical FP-Growth framework. It integrates the assigned thresholds during the construction of the conditional FP-trees and throughout the recursive pattern generation process. This enables the discovery of frequent itemsets that respect their corresponding minimum support constraints, ultimately supporting the identification of relevant binary join indexes for reducing the execution costs of OLAP decision-support queries.

4.4 Approach for automatic index selection

The proposed approach, whose general principle is represented in Figure 3, consists of the following steps :

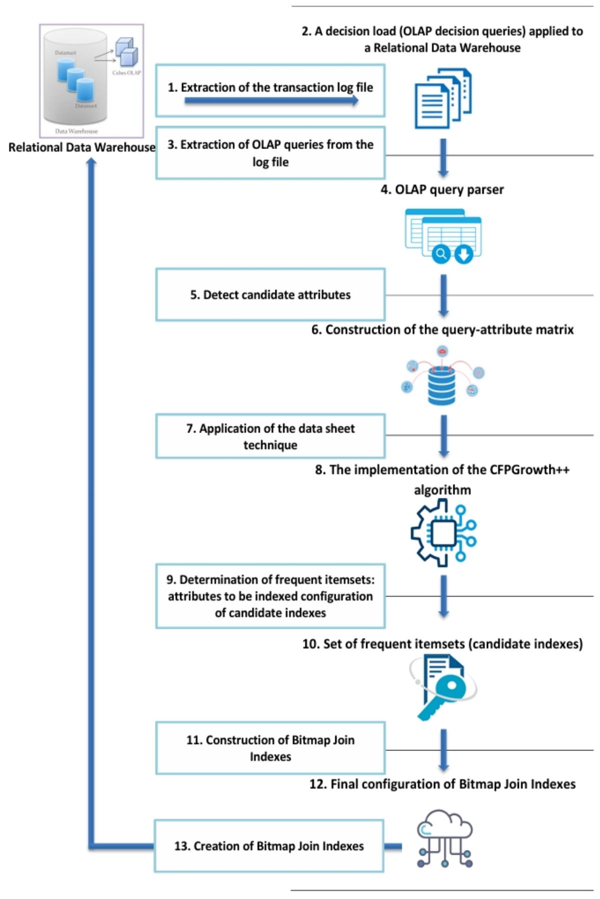
Relational Data Warehouse
2. A decision load (OLAP decision queries) applied to a Relational Data Warehouse
1. Extraction of the transaction log file
3. Extraction of OLAP queries from the log file
4. OLAP query parser
5. Detect candidate attributes
6. Construction of the query-attribute matrix
7. Application of the data sheet technique
8. The implementation of the CFPGrowth++ algorithm
9. Determination of frequent itemsets: attributes to be indexed configuration of candidate indexes
10. Set of frequent itemsets (candidate indexes)
11. Construction of Bitmap Join Indexes
12. Final configuration of Bitmap Join Indexes
13. Creation of Bitmap Join Indexes

Figure 3 : Proposed approach for constructing Bitmap Join Indexes.

1) Selection of a load of OLAP decisional queries, assumed to be representative, of the system activity.

2) Structuring of the indexable attributes contained in the load in the form of a transactional database where the queries represent the transactions and the attributes represent the patterns. This represents our context for extracting frequent itemsets.

3) Generation of candidate indexes by the CFPGrowth++ algorithm implemented in Java.

4) Comparison of the response times of the execution of the OLAP decisional queries without indexes and then with the indexes generated by the implementation of CFPGrowth++.

Technical implementation details :

- **Detection of candidate attributes :**

Candidate attributes are detected by analyzing a representative workload of OLAP decisional queries. Specifically, we parse the SQL query statements to identify attributes involved in WHERE clauses (selection predicates), GROUP BY clauses, and JOIN conditions. These attributes are considered indexable since they directly impact query performance in a decision-support context.

- **Criteria for attribute selection :**

The primary selection criterion is the frequency of an attribute’s appearance within the query workload. Attributes that frequently appear together in queries are treated as potential candidates for binary join indexes.

- **Application of the data sheet technique:**

The data sheet technique is applied by transforming the query workload into a transactional format where each query represents a transaction and the indexable attributes it contains form the transaction items. This transactional dataset is then used as input for the CFPGrowth++ algorithm to extract frequent itemsets.

- **Technical details of selecting and Creating bitmap join indexes :**

The frequent itemsets generated by CFPGrowth++ serve as candidate index configurations. Each frequent itemset represents a combination of attributes that often co-occur in queries, making them suitable candidates for indexing. The actual creation of Bitmap Join Indexes is simulated in our test environment by generating index creation scripts based on these frequent attribute combinations. Subsequently, query execution times are measured before and after applying these indexes to evaluate their impact on performance.

Detailed Pseudocode for the Adapted CFPGrowth++ Algorithm for Automatic Bitmap Join Index Selection in OLAP

Context :

This adapted CFPGrowth++ algorithm for automatic selection of Bitmap Join Indexes aimed at improving OLAP query performance. The general approach involves analyzing a representative workload of OLAP queries to identify frequently co-accessed attributes (frequent itemsets) using multiple support thresholds. These frequent attribute combinations are then used to configure candidate bitmap join indexes.

Pseudocode: adapted CFPGrowth++ for index selection :

Table 6 : Pseudocode : automatic binary join index selection with CFPGrowth++

```
Input:
  - Q: a representative workload of OLAP queries
  - MIS: a list of minimum support threshold per attribute
  - D: relational data warehouse schema
Output:
  - F: set of frequent itemsets (candidate bitmap join indexes)
Steps:
1. ExtractQueriesFromLogFile(log_file):
    Parse the transaction log to retrieve OLAP queries.
    Return set of queries Q.
2. IdentifyIndexableAttributes(Q, D):
    For each query in Q:
        Extract attributes involved in selection, grouping, and join predicates.
    Return the set of indexable attributes A.
3. BuildTransactionDatabase(Q, A):
    Create a transaction for each query:
        Items in a transaction = attributes accessed by the query.
    Return transaction database T.
4. ApplyCFPGrowthPlusPlus(T, MIS):
    a. Scan T to compute item supports.
    b. Sort items in each transaction in descending order of support.
    c. Build an initial CFP-tree:
        For each transaction:
            Insert items into the tree following the sorted order.
            Update item counts along the path.
    d. Recursively mine the CFP-tree:
        For each frequent item i:
            Generate conditional pattern base for i.
            Construct conditional CFP-tree.
            If conditional tree is not empty:
                Recursively mine conditional tree.
            Collect frequent itemsets meeting MIS thresholds.
    e. Return set of frequent itemsets F.
5. ConfigureCandidateBitmapJoinIndexes(F):
    For each frequent itemset f in F:
        Map items in f to attributes in D.
        Propose a bitmap join index configuration on these attributes.
6. EvaluateIndexImpact(F, Q, D):
    For each index configuration:
        Measure query response time on Q with and without the index.
    Retain index configurations improving query performance.
7. Return final selected frequent itemsets F as candidate bitmap join indexes.
```

Explanation of Adaptation for Index Selection Context:

- Transaction Construction:

Each OLAP query from the workload is treated as a transaction, and the attributes involved in selection, grouping, and joins are considered as items. This enables mapping query attribute usage into a transaction database suitable for frequent pattern mining.

- Multiple Minimum Support Thresholds:

The MIS list allows setting lower thresholds for attributes representing important query predicates. when indexed, could significantly optimize query performance.

• Frequent Itemsets as Candidate Indexes:

Frequent itemsets identified by the algorithm represent groups of attributes often queried together. These are directly translated into candidate bitmap join index configurations.

• Index Evaluation:

Before finalizing the index selection, each candidate's impact on query performance is empirically evaluated. Only those improving execution times are retained.

Insights into Minimum Support Threshold Selection and Its Impact.

Selecting appropriate minimum support thresholds is crucial for balancing mining efficiency and index relevance. In our approach:

Higher thresholds prioritize frequent patterns shared across numerous queries, potentially limiting index diversity but ensuring high-impact optimizations.

Lower thresholds allow the inclusion of less frequent but strategically valuable patterns, at the cost of increased computational overhead.

To address this trade-off, we employ differentiated support thresholds based on workload analysis:

Attributes heavily involved in query predicates receive lower thresholds.

Less critical attributes maintain higher thresholds to limit unnecessary pattern mining.

This strategy enhances the algorithm's behavior by focusing computational resources on workload-relevant patterns, leading to a more effective and workload-tailored index configuration.

5 Experiments and results

To confirm the efficiency of our strategy for selecting Bitmap join indexes, we employed it in a Data Warehouse configured with a star relational schema running on Oracle 11g. The experimentation was conducted on an Intel Core2Duo machine with a 2GB main memory.Our experimental study is conducted in the following steps :

1) Implementation of the CFPGrowth++ algorithm in Java. Besides its portability, java is chosen for its automatic memory management. This feature is crucial because the manipulated data structures are mainly linked lists and trees. Our implementation is applied to the selection context.

2) Creating a Data Warehouse using the Analytical Processing Benchmark 1 ABP-1 business intelligence workbench [3]. This warehouse is composed of one fact table Actvars and four dimension tables ProdLevel, TimeLevel, CustLevel and ChanLevel.

The schema follows the classical star schema model and is composed of one fact table and several dimension tables as detailed below:

Fact table:

Actvars

Attributes: Customer_level, Product_level, Channel_level, Time_level, UnitsSold, DollarSales, DollarCost

This table records the sales transactions and is linked to the dimension tables through foreign keys.

Dimension tables:

ProdLevel

Attributes: Code_level, Class_level, Group_level, Family_level, Line_level, Division_level

TimeLevel

Attributes: Tid, year_level, quarter_level, month_level, week_level, day_level

ChanLevel

Attributes: Base_level, all_level

CustLevel

Attributes: Store_level, Retailer_level

This structure provides a multidimensional framework suitable for typical OLAP operations, including aggregation, drill-down, roll-up, and slicing/dicing queries.

Query workload specification

The experimental query workload was designed based on the ABP-1 benchmark's [20] guidelines and consisted of a mix of decision-support queries representing realistic OLAP operations. The workload includes:

- **Aggregation queries:**

Queries calculating total sales, costs, or quantities based on one or more dimensions, such as total DollarSales per Product_level or per Time_level.

- **Drill-down and roll-up queries:**

Queries navigating through different granularity levels within dimensions, for instance, moving from year_level to month_level in TimeLevel or from Division_level to Code_level in ProdLevel.

- **Slice and dice queries:**

Queries selecting specific data subsets based on certain conditions, like sales for a particular Retailer_level during a specific quarter_level.

- **Multi-dimensional analysis queries:**

Complex queries involving multiple dimensions and aggregate measures, for example, computing average UnitsSold and total DollarCost for various Channel_level and Product_level combinations over time.

Table 7 summarizes the characteristics of the tables forming the warehouse. We considered a load of 60 OLAP decisional queries defined on this Data Warehouse.

Table 7 : Characteristics of the tables in the data warehouse used.

Table	**Number of n-tuples**	**Size (Octet)**
ACTVARS	261 740 160	2 142 250 000
PRODLEVEL	10 800	1 048 576
TIMELEVEL	24	65 536
CHANLEVEL	11	65 536
CUSTLEVEL	1 080	65 536

A set of queries using several selection predicates defined on one or more attributes has been considered to cover all the attributes of the warehouse. These queries belong to several categories: queries using aggregation functions such as Sum, Min, Max, queries with dimension attributes in the SELECT clause, count(*) type queries with and without aggregations. (Table 8) shows an extract of the load composed of five queries.

Table 8 : Example of queries extracted from a load

```
(Q1)
SELECT
    A.Time_level,
    AVG(A.UnitsSold) AS AverageUnitsSold
FROM
    ACTVARS A
JOIN
    TIMELEVEL T ON A.Time_level = T.Tid
WHERE
    T.Quarter_level IN ('Q1', 'Q2')
GROUP BY
    A.Time_level;
```

```
(Q2)
SELECT
    P.Division_level,
    COUNT(*) AS RecordCount
FROM
    ACTVARS A
JOIN
    PRODLEVEL P ON A.Product_level =
P.Code_level
WHERE
    P.Group_level = 'RQ'
GROUP BY
    P.Division_level;
```

```
(Q3)
SELECT
    A.Retailer_level,
    AVG(A.UnitsSold) AS AverageUnitsSold
FROM
    ACTVARS A
JOIN
    PRODLEVEL P ON A.Product_level =
P.Code_level
JOIN
    Custlevel C ON A.Customer_level =
C.Store_level
WHERE
    P.Division_level = 'UV'
GROUP BY
    A.Retailer_level;
```

```
(Q4)
SELECT
    A.Product_level,
    AVG(A.UnitsSold) AS AverageUnitsSold
FROM
    ACTVARS A
JOIN
    TIMELEVEL T ON A.Time_level = T.Tid
WHERE
    T.Year_level = '2025'
GROUP BY
    A.Product_level;
```

```
(Q5)
SELECT
    P.Division_level,
    AVG(A.UnitsSold) AS AverageUnitsSold
FROM
    ACTVARS A
JOIN
    TIMELEVEL T ON A.Time_level = T.Tid
JOIN
    PRODLEVEL P ON A.Product_level =
P.Code_level
WHERE
    T.Month_level = '7'
GROUP BY
    P.Division_level;
```

3) We have created the extraction context after generating the Data Warehouse. It is a "query-attribute" matrix where each row designates a query of the load. The columns define the candidate attributes for the indexing procedure. The existence of an indexable attribute in a query is symbolized by 1 and its absence by 0 [To each query Qi and each attribute Aj, we associate a usage value of the attribute which is equal to 1 if the query uses the attribute Aj, 0 otherwise]. We illustrate the construction of this matrix through the following example. The "query-attribute" matrix obtained after the syntactic analysis of the load is composed of eleven columns and five rows (Table 9).

Table 9 : Query-attribute matrix

Tables	**PRODLEVEL**			**TIMELEVEL**			
Queries/Attributes	***Code_level***	***Group_level***	***Division_level***	***Tid***	***Year_level***	***Quarter_level***	***Month_level***
Q1	0	0	0	1	0	1	0
Q2	1	1	0	0	0	0	0
Q3	1	0	1	0	0	0	0
Q4	0	0	0	1	1	0	0
Q5	1	0	0	1	0	0	1

Tables	CUSTLEVEL	ACTVARS		
Queries/Attributes	*Store_level*	*Customer_level*	*Product_level*	*Time_level*
Q1	0	0	0	1
Q2	0	0	1	0
Q3	1	1	1	0
Q4	0	0	0	1
Q5	0	0	1	1

The "query-attribute" matrix obtained after parsing the payload is composed of 11 columns and 5 rows (Figure 5). It is subdivided according to the tables used in the payload for reasons of clarity and readability. This matrix is used by the CFPGrowth++ algorithm.

We applied our implementation of the CFPGrowth++ algorithm to the extraction context in order to select the most used candidate attributes in the system history that represent interesting candidates for the indexing operation.

4) We proceeded in the last step to the execution of the load of the queries on the generated Data Warehouse according to two scenarios (1) without creation of the indexes, (2) after creation of the indexes generated by the CFPGrowth++ algorithm.

During the execution of the load of queries on the Data Warehouse generated without creating indexes we calculated the execution time of each query of the load, After that we proceeded to calculate the execution time of each query of the load after creation of the indexes generated by the CFPGrowth++ algorithm. Figure 4 shows the times taken for the execution of the quests after the creation of the generated binary join indexes.

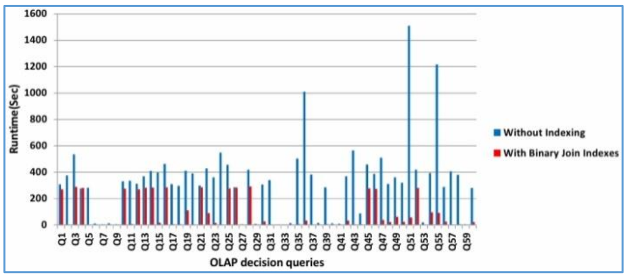

Figure 4 : Query execution time without creating indexes and after creating generated binary join indexes.

Our approach with generating candidate indexes by the CFPGrowth++ algorithm provides excellent performance improvements compared to running the query load without indexing.

Through our experimental study, we notice that the execution time is significantly improved after creating the binary join indexes. The Figure 5 illustrates the overall execution time saving for the entire load of OLAP decisional queries considered.

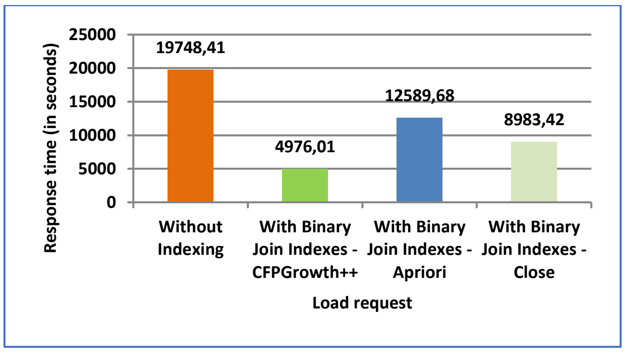

Figure 5 : Overall execution time of the load without creating indexes and after creating the generated binary join indexes.

As shown in Figure 5, the response time for decision-support query workloads without indexing was 19748.41 seconds, while the use of Binary Join Indexes generated with CFPGrowth++ reduced this to 4976.01 seconds, achieving a performance gain of approximately 75%. In contrast:

Apriori-based indexing reduced response time to 12589.68 seconds

Close-based indexing reduced it to 8983.42 seconds

This confirms that CFPGrowth++ outperforms these alternative frequent itemset mining algorithms in index selection efficiency for OLAP workloads.

The results obtained confirm the great usefulness of binary join indexes for the optimization of OLAP decisional queries. The execution time of these queries is significantly lower with the use of binary join indexes.

The experiments carried out show that the index configuration generated by the proposed approach allows a significant performance gain of around 75%.

To evaluate the effectiveness and robustness of the proposed approach for frequent pattern-based binary join index selection in OLAP query optimization, a series of 11 independent performance tests were conducted on varying OLAP query workloads. The experimental results demonstrated consistent performance improvements ranging from **74.20% to 75.96%**.

The statistical analysis confirmed the reliability of these performance gains. The mean improvement across all tests was **74.58%**, with a standard deviation of **0.50**, indicating low variability in the observed results. A one-sample t-test was performed to determine whether the observed improvements were statistically significant compared to a baseline of no performance gain. The test yielded a **t-statistic of 493.72** and a corresponding **p-value of 2.86×10^{-23}**, which is well below the conventional significance threshold of 0.001. This confirms that the observed improvements are statistically significant at a **95% confidence level**.

Moreover, a 95% confidence interval for the mean performance improvement was calculated as **[74.24%, 74.92%]**, further supporting the consistency and robustness of the proposed method’s effectiveness under different query scenarios. These results clearly validate the practical benefits and generalizability of the

CFPGrowth++-based approach in optimizing complex OLAP workloads.

6 Discussion

This study is situated within the context of selecting optimal indexes to enhance the performance of decision-making queries, particularly in OLAP environments. Several approaches have been proposed previously, notably those by Aouiche et al. [27], who employed closed frequent pattern mining using the Close algorithm to prune the search space, primarily based on attribute usage frequency. However, this method can lead to the elimination of potentially relevant indexes, especially for attributes belonging to large dimension tables, where usage frequency alone does not guarantee effective optimization.

On the other hand, Bellatreche et al. [28][29][32] emphasized that access frequency alone is not a sufficient criterion for effective index selection. Their approach incorporates additional parameters, such as table sizes and page characteristics, to better balance the relevance of indexes. Nevertheless, this increased complexity can result in higher computational costs during pattern generation.

Our approach, utilizing the CFPGrowth++ algorithm, offers a significant advancement over these methods. Indeed, CFPGrowth++ enables faster and more scalable extraction of frequent patterns. Its ability to handle multiple support thresholds allows for a more refined exploration of itemsets, which contributes to better adaptation to large data volumes and performance requirements.

Regarding performance, the improvements offered by CFPGrowth++ can be attributed to its optimized structure, which reduces processing time and memory consumption compared to traditional algorithms like Apriori or Close. It also better manages the trade-offs between computational cost and effectiveness. However, this increased efficiency may come with some storage overhead, especially if a large number of frequent patterns are generated, necessitating careful management to avoid memory overload.

In our proposed approach, CFPGrowth++ was chosen for its efficiency in mining frequent patterns through a compact tree-based structure and its flexibility in handling multiple support thresholds. However, we recognize that as the number of attributes and queries increases in real-world data warehouses, the size of the pattern base and the computational overhead can grow significantly.

To manage this, several strategies can be applied, such as:

Adjusting support thresholds dynamically to limit the number of frequent itemsets generated in dense workloads.

The evaluation with applying a constraint on the total storage space consumed by the generated indexes.

Partitioning the workload or focusing on query subsets that target the most resource-intensive operations.

Parallelizing the mining process across distributed computing environments to improve processing times and scalability.

CFPGrowth++ stands out for its speed, scalability, and accuracy, making it a promising solution for index selection in contexts where efficient management of large data volumes is critical. It represents a notable improvement over previous techniques, while also requiring attention to potential storage and computational costs.

7 Conclusion

Within the realm of Data Warehouses schematized in a star relational model carried out via OLAP decision-making queries, very high response time remain more than ever a crucial issue.The goal of this work is to improve the Data Warehouse performance.The proposed approach for optimizing system performance by minimizing response time is based on finding frequent patterns for automatic selection of binary join indexes in relational Data Warehouses modeled by a star schema through generation of a configuration of binary join indexes based on the implementation of the CFPGrowth++ algorithm.The results show the particular performance of the binary join indexes recommended by the CFPGrowth++ algorithm implemented for relational Data Warehouses. Our study demonstrates that applying data mining techniques for the automatic selection of binary join indexes in relational Data Warehouses is a promising strategy, offering encouraging results and opening up several opportunities for future research and optimization. Other possible improvement is to consider multiple parameters to generate the final index configuration. In future work it important to consider in particular of the selectivity factors and the cardinalities of the attributes or the sizes of the dimension tables.

References

[1] A. Vaisman, E. Zimányi, ‘Data Warehouse Systems - Design and Implementation’. Data-Centric Systems and Applications. Springer, 2014. https://doi.org/10.1007/978-3-642-54655-6

[2] I. Kovacic, G. Christoph Schuetz, B. Neumayr, M. Schrefl, ‘OLAP Patterns: A pattern-based approach to multidimensional data analysis’, Data & Knowledge Engineering, Volume 138, 2022. https://doi.org/10.1016/j.datak.2021.101948

[3] S. Chaudhuri, U. Dayal, Narasayya, V., ‘An overview of business intelligence technology’. Commun. ACM 54(8), 88–98, 2011. https://doi.org/10.1145/1978542.1978562

[4] A. Cuzzocrea, ’Evolving OLAP and BI towards Complex, High-Performance BigOLAP-Data-Cube-Processing Analytics Frameworks: How to Speed-Up Large-Scale, High-Dimensional Queries over Clouds’, Procedia Computer Science 246 4169–4175, 2024. https://doi.org/10.1016/j.procs.2024.09.256

[5] H. Inmon, ‘Building the data warehouse’. John Wiley & sons, 2005. https://books.google.co.ma/books?id=QFKTmh5IFS4C&printsec=frontcover&hl=fr&source=gbs_ge_summary_r&cad=0#v=onepage&q&f=false

[6] R. Kimball, M. Ross, ‘The Kimball Group Reader: Relentlessly Practical Tools for Data Warehousing and Business Intelligence’, John Wiley & Sons, 2010. https://doi.org/10.1002/9781119228912

[7] D. M. Mosquera, R. Navarrete, S. L. Mora, L. Recalde, A. A. Cabrera, ’Integrating OLAP with NoSQL Databases in Big Data Environments: Systematic Mapping’, Big Data and Cognitive Computing, 8, 64, 2024. https://doi.org/10.3390/bdcc8060064

[8] N. Dedic, C. Stanier, ‘An evaluation of the challenges of multilingualismin data warehouse development’. In ICEIS 2016, Proceedings of the 18th International Conference on Enterprise Information Systems, Vol. 1, Rome, Italy, 196–206, 2016. https://doi.org/10.5220/0005858401960206

[9] S. Roy, S. Raj, T. Chakraborty, A. Chakrabarty, A. Cortesi, S. Sen, ’Efficient OLAP query processing across cuboids in distributed data warehousing environment’, Expert Systems with Applications Volume 239, 2024. https://doi.org/10.1016/j.eswa.2023.122481

[10] S. Chaudhuri, V. Narasayya, ‘Self-tuning database systems: A decade of progress’. In Proceedings of the International Conference on Very Large Databases, 3–14, 2007. https://dl.acm.org/doi/10.5555/1325851.1325856

[11] R. Kimball, M. Ross, ‘The Data Warehouse Toolkit: The Definitive Guide to Dimensional Modeling’, John Wiley & Sons , 2013. https://dl.acm.org/doi/10.5555/2543973

[12] H. Necir, H. Drias. 2015. A distributed maximal frequent itemset mining with multi agents system on bitmap join indexes selection. Int. J. Inf. Technol. Manage. 14, 2/3, April 2015. https://doi.org/10.1504/IJITM.2015.068470

[13] M. Yahyaoui, S. Amjad, L. Benameur. I. Jellouli, ‘Efficient of bitmap join indexes for optimising star join queries in relational data warehouses’, Int. J.Computational Intelligence Studies, Vol. 9, No. 3, pp.220–233, 2020. https://doi.org/10.1504/ijcistudies.2020.109604

[14] R. Strohm, ‘Oracle Database Concepts, 11g Release 1 (11.1)’ B28318-03, Octobre 2007. https://www.appservgrid.com/documentation111/docs/rdbms11gr1/server.111/b28318/memory.htm

[15] D. Zhang, ‘B Trees’, Chapter 15 of Handbook of Data Structures and Applications, D. P. Mehta, S. Sahni (editors), Chapman & Hall/CRC, 2004. https://doi.org/10.1201/9781420035179

[16] S. Chaudhuri, M. Datar, V. Narasayya. Index Selection for Databases: A Hardness Study and a Principled Heuristic Solution. IEEE Trans. Knowl. Data Eng. 26, 1313–1323, 2004. https://doi.org/10.1109/TKDE.2004.75

[17] R. Kain, D. Manerba, R. Tadei ‘The index selection problem with configurations and memory limitation: A scatter search approach’, Computers & Operations Research, Volume 133, 2021. https://doi.org/10.1016/j.cor.2021.105385

[18] D. Comer, ‘The difficulty of optimum index selection’. ACM Transactions on Database Systems, 3 (4), 440–445, 1978. https://doi.org/10.1145/320289.320296

[19] K. Stockinger, K. Wu, ‘Bitmap Indices for Data Warehouses, Data Warehouses and OLAP’, R. Wrembel and C. Koncilia, eds., IRM Press, 157-178, 2006. https://escholarship.org/uc/item/8zv9t143

[20] Olap Council.: ABP-1 Benchmark, http://www.olapcouncil.org/

[21] S. Chaudhuri, M. Datar, V. Narasayya. (2004). ‘Index selection for databases: a hardness study and a principled heuristic solution’. IEEE Transactions Knowledge on Data Engineering, Volume 16, Issue 11, Novombre 2004. https://doi.org/10.1109/TKDE.2004.75

[22] B. Ziani, A. Benmlouka, Y. Ouinten. Improving Index Selection Accuracy for Star Join Queries Processing: An Association Rules Based Approach. Management Intelligent Systems. Advances in Intelligent Systems and Computing, vol 220. Springer, Heidelberg, 2013. https://doi.org/10.1007/978-3-319-00569-0_9

[23] A. Rakesh, S. Ramakrishnan, ‘Fast Algorithms for Mining Association Rules’, International Conference on Very Large Databases, pp. 487-499, September 1994. https://dl.acm.org/doi/10.5555/645920.672836

[24] A. Netz, S. Chaudhuri, J. Bernhardt, U. Fayyad, ‘Integration of Data Mining and Relational Databases’, International Conference on Very Large Data Bases, pp. 719-722, September 2000. https://www.microsoft.com/en-us/research/wp-content/uploads/2016/02/integration-of-data-mining.pdf

[25] N. Bruno, S. Chaudhuri, Automatic physical database tuning: a relaxation-based approach. Proceedings of the SIGMOD Conference, 2005. https://doi.org/10.1145/1066157.1066184

[26] M. Golfarelli, S. Rizzi, E. Saltarelli, ‘Index selection for data warehousing. Proceeding’s 4th International Workshop on Design and Management of Data Warehouses (DMDW'2002), Toronto, Canada, pp. 33-42, 2002. https://ceur-ws.org/Vol-58/RIZZI.pdf

[27] K. Aouiche, J. Darmont. Data Mining-based Materialized View and Index Selection in Data Warehouses. Journal of Intelligent Information Systems 33(1), 65–93, 2009. https://doi.org/10.1007/s10844-009-0080-0

[28] L. Bellatreche, R. Missaoui, H. Necir, H. Drias, Selection and pruning algorithms for bitmap index selection problem using data mining. LNCS, vol. 4654, pp. 221–230. Springer, Heidelberg, 2007. https://doi.org/10.1007/978-3-540-74553-2_20

[29] L. Bellatreche, Techniques d'optimisation des requêtes dans les data warehouses. In Sixth International Symposium on Programming and Systems, 2003. https://hal.science/hal-03759388v1
[30] R.U. Kiran, P.K. Reddy, 'Novel Techniques to Reduce Search Space in Multiple Minimum Supports-Based Frequent Pattern Mining Algorithms', EDBT/ICDT '11, 21 March 2011. https://doi.org/10.1145/1951365.1951370
[31] H. Ya-Han, C. Yen-Liang, 'Mining association rules with multiple minimum supports: a new mining algorithm and a support tuning mechanism', Decision Support Systems, Volume 42, Issue 1, 2006. https://doi.org/10.1016/j.dss.2004.09.007
[32] L. Bellatreche, R. Missaoui, H. Necir, H. Drias. 'A Data Mining Approach for Selecting Bitmap Join Indices'. JCSE.2007.1.2.177, December 2007. https://doi.org/10.5626/JCSE.2007.1.2.177