ASADD: Attribute Similarity-Aware Database Designer for Partitioning and Indexes

Rym Bouchakri[1] and Ladjel Bellatreche[2]

[1] National High School of Computer Science Algiers, Algeria
r_bouchakri@esi.fr
[2] LISI/ENSMA Poitiers University, Futuroscope, France
bellatreche@ensma.fr

Abstract. We present an automatic database design tool that exploits similarities between attributes when recommending horizontal data partitioning (HDP) and bitmap join indexes (BJI). Although there is a couple of related work done on how to select an appropriate set of BJI and HDP schemes for a given workload, none of this work has explored the effect of similarity of attributes candidate for HDP and BJI. Commercial DBMS offer advisors recommending indexes and partitioning using a workload or generate a hypothetical workload for a specified schema. Most of these tools consider each physical design technique in isolation and ignore their interdependencies (an attribute can be either used for HDP or BJI). Our tool identifies a set of BJI and partitioning schemes by taking into account their interaction materialized by attribute interchangeability, which can dramatically improve query performance. The originality of our design is that it assigns to each optimization technique its own attributes before launching its selection algorithm. This assignment is done using K-means method. We compare our designer with a state-of-the-art work on one workload APB-1. Our results show that a similarity-aware database designer can improve significantly query performance within the less space budget.

1 Introduction

In traditional databases, the main tasks of administrators (DBA) are mainly concentrated on the management of users, the selection of a restricted number of optimization techniques such as mono table indexes and the choice of relevant implementation of join operations (e.g., nested loop, sort merge, etc.). Recently, medium and large companies increase their demand on developing applications requiring extremely large databases, such as *data warehouses* and *scientific databases*. Administering such databases becomes crucial performance and economic issues. DBA have to perform several complicated tasks during the physical design phase: (1) the choice of optimization techniques (OT), (2) the choice of their selection mode (isolation or combined), (3) the development of selection algorithms, (4) the generation of scripts corresponding to each selected OT and (5) the validation and deployment of the obtained solutions. These tasks

increase the workload of DBA, consequently, their costs become more significant compared to the total cost of ownership [21].

Face to this situation, one of the economic strategies of companies is to simplify the role of DBA by making their tasks less demanding in terms of time and expertise in order to reduce their costs [9]. To satisfy company's requirements, commercial DBMS editors have recently proposed tools to make *self-managing databases* [9]. One of their objectives is to find solutions toward alleviating the burden on the DBA or, more ambitiously, totally replacing the DBA [9]. The most important commercial tools are: *SQL Server Database Tuning Advisor (DTA)*, *Oracle SQL Access Advisor* and *DB2 index advisor*. They allow *what-if* design exploration and propose DBA useful user interfaces. These tools are based on *greedy heuristics* to select relevant OT for a given workload. Although these heuristics make the existing design tools practical, they prune away large fractions of the search space and often suggest locally optimal solutions instead of the globally optimal one. To overcome this limitation, academic tools have been proposed (such as *SimulPh.D.* [3] and *PARINDA* [15]). They use of some advanced algorithms such as genetic, simulated annealing, etc.

Most of these tools are suitable when a single OT is asked. When a DBA wants to select more then one OT, a trivial strategy would consist of multiple runs of the tool equal to the number of suitable OT. The main drawback of this strategy is its ignorance of interaction between OT. Ignoring this interaction can significantly compromise the quality of recommendations proposed by tools [17]. Two types of interaction have been considered by a couple of tools. In *SQL Database Tuning Advisor*, interaction between materialized views and indexes is based essentially on their *structure* – both are redundant (they duplicate data), compete for the same resource representing storage space and cause update overhead [17]. In *DB2 Design Advisor* tool, where four optimization techniques are supported: indexes (defined on a single table), materialized views, partitioning and clustering), interaction is based on the order of selection process of OT [22]. To capture this interaction, two relations between pairs of OT have been proposed: textitstrongly and *weakly dependency*. An OT ot_1 "strongly" depends on optimization technique ot_2, if a change in selection of ot_2 often results in a change in that of ot_1. Otherwise, we say ot_1 "weakly" depends on ot_2. Based on the type of relation between pairs of OT, a selection order is established to select them. The main limitation of the studied interactions is their ignorance of the analysis of the bodies of OT[3]. Note that extremely large databases use many attributes to encode related information (e.g., Sloan Digital Sky Survey's data management project [10]). These attributes are usually candidate to define the same OT. Recently, some research efforts focused on *body analysis* of OT and its impact on their selection. Project *Coradd* is an example of these efforts [12]. The authors show the impact of correlation between attributes in defining two redundant OT materialized views and indexes. In our work, we focus on the phenomenon of attribute *interchangeability*, where an attribute may be candidate to define different OT. This phenomenon is studied on two OT: HDP and

[3] The body describes the used tables, attributes, etc.

BJI. HDP is a *non redundant* that consists in segmenting a table into multiple fragments each containing a subset of rows based on restriction attribute(s). BJI is *redundant* structure (needs storage and maintenance costs). It is bitmap for the table to be indexed is built for values coming from the joined tables. A BJI optimizes selection and joins operations. It has been effectively utilized in many major commercial DBMS (Oracle, IBM, Sybase, etc.).

Based on the identified similarities between HDP and BJI, we issue the following research reflexion that needs to be debated: *if an attribute is used to partition/index and this choice optimizes well queries then why it continues to be candidate for indexing/partitioning?*. Suppose it is discarded from indexing, a gain of storage and maintenance overhead may be guaranteed. If it is not considered for partitioning, it may reduce the number of partitions of the database and consequently facilitates its manageability. Note that the interaction between HDP and BJI is not well established by commercial and academic tools, since to best of our knowledge, only Oracle11G supports a large variety of partitioning modes (e.g., referential partitioning [8]). Few academic works attempt to deal with the combined selection. In [5], the authors propose the use of HDP is used to prune the search space of BJI selection problem. The basic idea of their proposal is to start by partitioning a data warehouse, and then selecting BJI for only queries that do not get benefit from HDP (called *non profitable queries*). The authors do not explicit the identification procedure of profitable queries and ignores the attribute interchangeability phenomena. This work is used in *SimulPh.D.*.

In this paper, we present a tool, named, *ASADD*, that recommends HDP (with two modes) and BJI that takes into account attribute interchangeability. Our proposal is based on the following reflexion: *instead of selecting HDP and BJI schemes for a given database based on all attributes candidate*, our tool first assigns to each OT its relevant attribute(s) and then each one is selected based on its own attributes. This approach allows a reduction of complexity of each selection problem. We propose to use K-means method to perform this assignment [13].

This paper is organized into four sections. Section 2 presents background related to HDP and BJI. A genetic algorithm for selecting BJI is also given. Section 3 describes our clustering approach based on K-means method. Section 4 experimentally compares selection algorithms used by *ASADD* with existing studies. Section 5 concludes the paper.

2 Background

In this section, some concepts related to partitioning and BJI are given.

2.1 Horizontal Data Partitioning

HDP is supported by most commercial DBMS (Oracle, SQL Server, DB2, etc.) and non commercial (Postgress, MySQL, etc.), where a native data definition

language is proposed. Two main types of HDP exist and supported by commercial DBMS: *mono table partitioning* and *table-dependent partitioning*. In the mono table partitioning, a table is partitioned using its own attributes. Several modes are proposed to implement this partitioning: *Range*, *List*, *Hash*, *Round Robin* (supported by Sybase), *Composite* (*List-List, Range-List, Range-Range*, ...), etc. Mono table partitioning may be used to optimize selections, especially when partitioning key matches with their attributes (partition pruning). In table-dependent partitioning, a table inherits the partitioning characteristics from other table. For example a fact table of a star schema of a given data warehouse may partitioned based on the fragmentation schemes of dimension tables[4]. This partitioning is feasible if a *parent-child relationship* among these tables exists [8]. It optimizes selections and joins simultaneously. This partitioning is recently supported by Oracle11G under the name *referential partitioning*.

Example 1. The following SQL statement partitions a dimension table *CUSTOMER* into four horizontal fragments based on two attributes (that we call *fragmentation attributes*): *Gender* and *Age*.

```
CREATE TABLE CUSTOMER(CID NUMBER, Name Varchar2(20), Gender CHAR, Age Number)
PARTITION BY RANGE (Age)
SUBPARTITION BY LIST (Gender)
SUBPARTITION TEMPLATE (SUBPARTITION Female VALUES ('F'), SUBPARTITION Male VALUES ('M'))
(PARTITION Cust_0_60 VALUES LESS THAN (60),
PARTITION Cust_60_120 VALUES LESS THAN (MAXVALUE));
```

The problem of HDP is formalized in the context of relational data warehouses as follows [4, 14, 16]:
Given (i) a representative workload $\mathcal{Q} = \{Q_1, ..., Q_n\}$, where each query Q_i $(1 \leq i \leq n)$ has an access frequency f_i, defined on a relational data warehouse schema with d dimension tables $\{D_1, ..., D_d\}$ and a fact table F and (ii) a constraint (called maintenance bound B given by DBA) representing the maximum number of fact fragments that he/she wants. The problem of HDP consists in identifying dimension table(s) that could be used to referential partition the fact table F into N fragments, such that $(\sum_{Q_i \in \mathcal{Q}} f_i \times Cost(\mathcal{Q}, FS))$ is minimized and maintenance constraint is satisfied $(N \leq B)$, where FS represents the generated fragmentation schema. This problem is an NP-hard [4].

Several types of algorithms to find a near-optimal solution are proposed: genetic, simulated annealing, greedy, data mining driven algorithms [4, 14].

2.2 Bitmap Join Indexes

a BJI computes the joins between the fact table and one or more dimension tables using one or more attributes. This join is materialized through a set of bit vectors built on the fact table based on one or more dimension attributes of *low cardinality*. The BJI are effective for COUNT, AND, OR, NOT queries. The size of the binary index is proportional to the cardinality of the indexed attributes. Note that the problem of selecting an indexing scheme is NP-hard [6].

[4] A fragmentation schema is the result of partitioning process.

Example 2. The following statement defines a BJI on a fact table *SALES* using the attribute *City* of dimension table *CUSTOMER*.

```
CREATE BITMAP INDEX sales_cust_gender
ON SALES(CUSTOMER.Gender)
FROM SALES S, CUSTOMER
WHERE S.CID= C.CID
```

Month	City	Year	Country	TypeProduct
0	1	1	0	1

Fig. 1. An example of chromosome

Examples 1 and 2 show the attribute interchangeability phenomenon, where *Gender* is used to partition and index the data warehouse. The formalization of the problem of selecting BJI is quite similar to HDP problem, except it uses a storage cost (S) as a constraint [1, 2]. The selected BJI shall minimize the query processing cost and satisfy the storage constraint S. To the best of our knowledge, only two classes of algorithms were proposed to deal with BJI selection problem: one uses greedy heuristics [2] and others based on data mining techniques [1]. To overcome the limitations of greedy heuristics; we propose a genetic algorithm (GA). GA have been used in the database physical design [11, 20]. Given a well-defined search space they apply three different genetic search operations, namely, *selection*, *crossover* and *mutation*, to transform an initial population of chromosomes, with the objective to improve their quality.

For BJI selection, each chromosome is represented by an array of bits, where each cell corresponds to an indexable attribute. A cell value is set 1, if its corresponding attribute is used by a BJI, 0 otherwise. Figure 1 shows an example of chromosome involving five indexable attributes. This coding generates three different GJI defined on *City*, *Year* and *Typeproduct*.

Note that each chromosome c_i of our GA represents a configuration of BJI. Let $Config_{c_i}$ and N_{c_i} be the set of selected indexes and its cardinal. To evaluate the quality of this configuration, two cost models are needed: (a) one for estimating the storage cost of $Config_{c_i}$ another to calculate the global query processing cost (in terms of inputs outputs)in the presented of $Config_{c_i}$. The storage cost required for a BJI bji_j of $Config_{c_i}$ defined on attribute A_k is given by [1, 19]: $storage(bji_j) = (\frac{|A_k|}{8} + 16) \times ||F||$, where $|A|$ and $||F||$ represent respectively, the cardinality of the attribute A_k and the number of instances of the fact table F. The cost of executing a query Q_i $(1 \leq i \leq n)$ in presence of a BJI bji_j is given by: $Cost(Q_i, bji_j) = log_m|A| - 1 + \frac{|A|}{m-1} + d \times \frac{||F||}{8 \times PS} + |F| \times (1 - e^{-\frac{N_r}{||F||}})$, where $|F|$, N_r, PS and V represent the number of pages occupied by fact table F, the number of tuples accessed by BJI bji_j, the size of disk page and the number of bitmaps used to evaluate the query Q_i. The global cost of executing all n queries in the presence of the configuration $Config_{c_i}$ is given by: $Cost(Q, Config) =$

$\sum_{k=1}^{n} \sum_{j=1}^{N_{c_i}} Cost(Q_k, idx_j)$. To penalize a chromosome generating a configuration violating the storage constraint, a penalty value is introduced as a part of the fitness function. It is defined as follows: $Pen(Config_{c_i}) = \frac{storage(Config_{c_i})}{S}$, where $storage(Config_{c_i}) = \sum_{j=1}^{N_{c_i}} storage(bji_j)$. Our fitness function is defined as follows: $F(Config_{c_i}) = \{^{Cost(Q,Config_{c_i}) \times Pen(Config_{c_i}), if Pen(Config_{c_i}) > 1}_{Cost(Q,Config_{c_i}), if Pen(Config_{c_i}) \leq 1}$

3 Problem of Attribute Interchangeability

In this section, we position the problem of interchangeability of attributes and we formalize it. This problem is an integral part of the whole problem that selects conjointly HDP and BJI schemes that is formalized as follows: given a (i) workload $\mathcal{Q} = \{Q_1, Q_2, \cdots, Q_m\}$, where each query Q_j has an access frequency f_j, (ii) a set of restriction attributes $\mathcal{R}$ extracted from $\mathcal{Q}$, (iii) storage capacity S for BJI and (iv) a threshold B representing the number of fact fragments. This problem consists in selecting HDP and BJI schemes that reduce the query processing cost and satisfy the defined constraints. The resolution of this problem requires an exploration of the combined search space of HDP and BJI problems. Let Ins_{HDP} and Ins_{BJI} be the number of instances of both problems. The global research space that needs to be explored is given by: $2^{Ins_{HDP}+Ins_{BJI}}$, since their instances may interact [22]. To reduce this complexity, we propose to DBA to first share attributes between these HDP and BJI and to select each technique using her/his favourite algorithm.

Let $\mathcal{R}$ be a set of attributes candidate for HDP and BJI. To split $\mathcal{R}$ between these two OT, DBA may perform it manually using its experience. This is feasible when the cardinal of $\mathcal{R}$ is small. For extremely large databases with a large number of attributes, this solution cannot work. As consequence, an automatic and an efficient clustering method is required. To select the best clustering, exhaustive enumeration is required to search for all potential clustering (given by $2^{||\mathcal{R}||}$, where $||\mathcal{R}||$ represents the cardinal of $\mathcal{R}$) and each one is used to select HDP and BJI and finally the optimal one is the clustering offering less cost of executing a set of queries respecting the constraints is chosen. It is impracticable to explore all clustering exhaustively, therefore the development of a non costly solution is mandatory.

3.1 Clustering-based Approach for Attribute Interchangeability

In this section, we propose an approach to cluster attributes between HDP and BJI. Before presenting it, an identification of how good an attribute is for each OT is required.

Identification of Clustering Criteria This identification is done using experiments on Oracle11G and data set of APB1 benchmark [7]. These experiments allow identifying three clustering criteria:

1. *Access frequency of restriction attribute*: this criterion represents the number of appearance of each attribute in queries. Our experiments show that usually HDP gives better performance when it is defined on most frequently attributes. Also BJI defined on these attributes are efficient, especially for some classes of queries like COUNT queries. When an attribute gives a similar performance when it is used in both HDP and BJI, we privilege its use for HDP, due to the non redundant nature of partitioning.
2. *Attribute cardinality*: when the cardinalities of attributes are higher, the size of each fragment may increase, since HDP selection is based on the decomposition of attribute's domains into sub domains [4]. As result, a small piece of fact table is required for some kind of queries. Unfortunately, this scenario is not always possible, because HDP is constraint by a maintenance bound B (Section 2). When BJI defined on higher cardinality, the storage cost increases dramatically. Based on this observation, the following clustering rule is established[5]: a *high cardinality attribute is recommended for partitioning whereas a lower cardinality attribute for BJI.*
3. *Selectivity factor defined on restriction predicates*: Let A_i an attribute used by $k_i (k \geq 0)$ restriction predicates $\{P_1, ..., P_{k_i}\}$. When a BJI defined on A_i with lower selectivity factors is recommended since, only a small piece of fact table will be loaded to execute a given query (especially for sum, avg, min, max,... queries).
   We define the *selectivity factor* (SF) of a restriction attribute A_i as the average of all its selectivity factors: $SF(A_i) = \frac{\sum_{i=1}^{k_i} Sel(P_i)}{k_i}$.

K-means Method In order to split $\mathcal{R}$, several clustering techniques may be used such as K-means, decision trees, etc. In this work, we choose the K-means method for the following reasons: (1) it is well adapted to our sharing problem and (ii) it has been used to partition XML data warehouses [14]. It classifies a given data set T through k clusters a priori fixed [13]. The main idea is to define k centroids, one for each cluster, and then assign each point to one of the k clusters so as to minimize a measure of dispersion within the clusters. The algorithm is composed of the following steps:

1. Place k initial points into the space represented by the data set T;
2. Assign each object x_i to the group that has the closest centroid c_j (the proximity is often evaluated with the euclidean metric);
3. Recompute the positions of the k centroids when all objects have been assigned;
4. Repeat Steps 2 and 3 until the centroids no longer move.

The best grouping is the partition of the data set T that minimizes the sum of squares of distances between data and the corresponding cluster centroid.

The following correspondences between the general form of K-means method and our clustering problem are done as follows:

[5] This rule follows partially the proposal in [18]

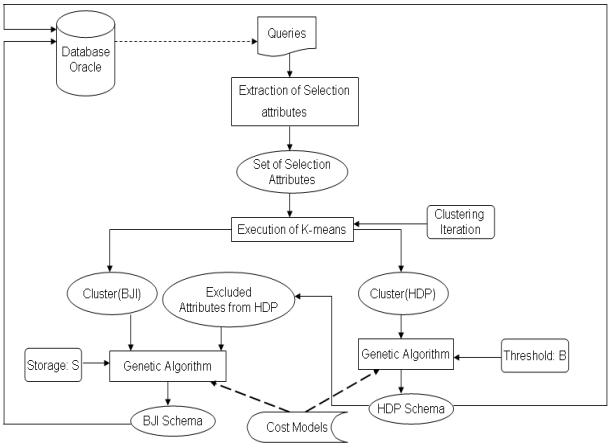

Fig. 2. Architecture of ASADD

- the data set of K-means represents our set of restriction attributes $\mathcal{R}$;
- K is equal to 2, since our combined selection problem concerns two OT: HDP and BJI.
- The attributes are represented in $\Re^2$ space with coordinates (x, y) computed as follows: We define a classification weight for each restriction attribute A_i based on the three above criteria: $Weight(A_i) = Frc(A_i) + SF(A_i) + Card(A_i)$, where $Frc(A_i)$, $SF(A_i)$ and $Card(A_i)$ represent respectively, the frequency, selectivity factor and cardinality of A_i. During the development of the weights of attributes, we have noticed that the three criteria have different scales. To make the weight consistent, normalization is necessary. Once the weight is calculated, the coordinates in $\Re^2$ of each attribute A_i are specified as follows:
  $(x, y) = (position\ of\ attribute\ A_i, weight(A_i)$.

Example 3. Let us consider a set of queries involving five attributes: $\mathcal{R} = \{Month, Year, City, Country, Class\}$. The weight of each attribute is given in Table 1. The coordinates of each attribute are given in Table 2.

Table 1. Weight computation

Attribute	Frc	SF	Card	NFrc	NSF	NCard	Weight
Year	11	0.5	23	1.14	0.53	0.01	1.70
Month	5	0.33	12	0.26	1.41	-0.3	1.37
City	6	0.1	55	0.41	-0.13	0.94	1.22
Country	9	0.09	20	0.85	-0.2	-0.07	0.57
Class	3	0.02	62	0.02	-0.67	1.14	0.44

$NFrc$, NSF and $NCard$ represent respectively the normalized factor of frequency and selectivity factor and cardinality criteria.

Table 2. Coordinates of restriction attributes

Attribute	Year	Month	City	Country	Class
Coordinates	[1, 1.70]	[2, 1.37]	[3, 1.22]	[4, 0.57]	[5, 0.44]

Figure 3 shows a classification of restriction attributes into two subsets $Cluster_{BJI} = \{Country, Class\}$ and $Cluster_{HDP} = \{Year, Month, City\}$.

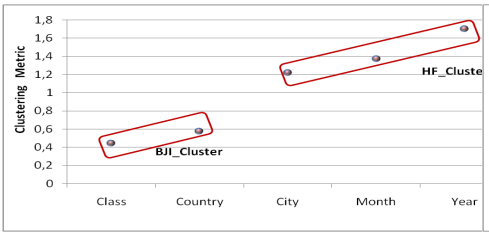

Fig. 3. Result of our classification

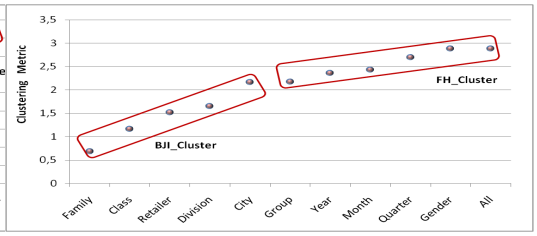

Fig. 4. Result of K-means

4 Performance Study

To validate algorithms used by ASADD, we conduct intensive experiments using data set of ABP1 benchmark [7] and 47 queries[6] involving 11 restriction attributes. The schema of the used warehouse contains one fact table *Actvars* (24 786 000 tuples) and 4 dimension tables *Prodlevel* (9000 tuples), *Custlevel* (900 tuples), *Timelevel* (24 tuples) and *Chanlevel* (9 tuples). A Core 2 Duo machine with 2 GB of memory is used. The section of HDP schema is ensured by an adaptation of the genetic algorithm developed in [4]. The selection of BJI is done by the genetic algorithm developed in Section 2. All our algorithms are implemented using JAVA Eclipse with two API. The first one dedicated for the K-means clustering method and the second, named, *JGAP* (Java Genetic Algorithms Package[7]) used to implement our two genetic algorithms. JGAP requires essentially coding of chromosome and fitness function. To compute the real execution cost for each query, we developed a JAVA class named *ORACLECOST* that calls *EXPLAIN PLAN* Oracle Optimizer tool that displays execution plans and then accesses *PLAN_TABLE* (a system table) to get the query's cost.

[6] 47 queries available at: http://www.lisi.ensma.fr/ftp/pub/documents/reports/2010/2010-LISI-.pdf

[7] A framework implementing genetic algorithms: http://jgap.sourceforge.net

4.1 Steps of our Selection Methodology

To conduct our experiments, we establish the following steps:

1. Classification of the 11 restriction attributes: for 50 iterations, *k-means* generates two clusters: $Cluster_{HDP} = \{Gender, Month, Year, All, Quarter, Group\}$ and $Cluster_{BJI} = \{Family, Division, Class, City, Retailer\}$ (Figure 4).
2. Selection of HDP and BJI schemes: HDP is executed with a maintenance constraint $B = 70$. The execution of our algorithm gives a HDP schema with 64 fragments of the fact table. BJI genetic algorithm generates 4 BJI defined on attributes belonging to $Cluster\_BJI$. These indexes occupy 500 Mo.
3. The obtained optimization schemes are directly applied on the real data warehouse using appropriate scripts.

4.2 Tests and Results

In this section, we present different experiments that we consider relevant for our studies. Unfortunately we could not compare our tool methodology to solve the combined selection problem of HDP and BJI with commercial and academic ones because they do not support well referential partitioning and its interaction with BJI. Therefore, we compare the ASADD methodology, baptised OWC with two other methodologies dealing with the combined selection: (i) Boukhalfa et al.'s methodology [5] based on the principle of profitable queries (Section 1), named, OPQ and a (ii) methodology that ignores the *interchangeability attribute problem*, named, OWS (described by only the two last steps described in Section 4.1). For each methodology, the cost of executing of 47 queries is computed on Oracle11G. Each methodology is compared against the non optimization mode (none OT is selected) and isolated mode characterized by two sub modes: (i) only HDP is used and (ii) only BJI are used. Our methodology is implemented as follows: we first partition the warehouse and then index it ($HP\&BJI$). The storage space reserved for BJI is 500 MB. Figure 5 and 6 present the costs of executing the 47 queries and the rate of optimized queries offered by each mode. These results show that $HP\&BJI$ outperforms the other modes, especially when attribute clustering is used. Indeed, the cost raised from 28.4 to 12.5 million of inputs outputs (I/O), which represents a reduction of 56 % of the total cost and 91 % of the optimized queries. Several lessons can be learned from these results: (a) the combined selection outperforms largely the isolated one, (b) the HDP defined on attributes identified relevant by k-means gives a better results than HDP done on all restriction attributes. Indeed, for OWC approach, the most appropriate attributes are selected for HDP, (c) the OPQ outperforms slightly OWS only for $HP\&BJI$ mode. Indeed, OPQ selects BJI only for non profitable queries. As results, it participates in improving their performance. Generally, OWC outperforms other methodologies whatever the used mode.

To study the impact of the maintenance bound B (representing the maximum number of fact fragments) on query performance, we varied B while fixing the storage constraint $S = 500\ Mo$. For each value of B, we consider the three

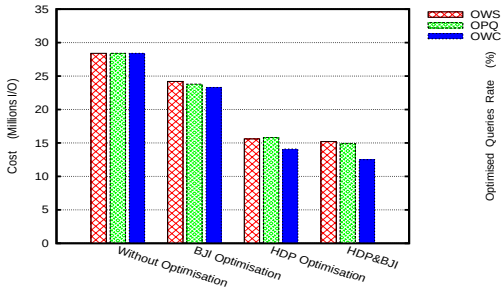

Fig. 5. Query performance with different optimization modes

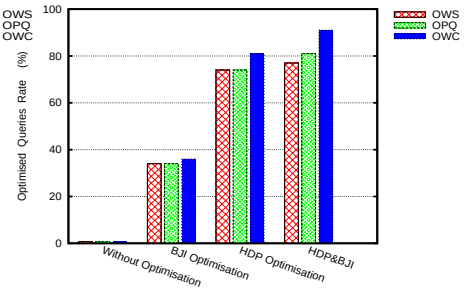

Fig. 6. Optimised queries Rate with different optimization modes

methodologies for the combined problem. The results illustrated in Figure 7 show that the best optimization is achieved when OWC is used, especially for $W > 100$. Indeed, the cost is reduced from 58 % to 61 %, where 91% of 47 of queries are optimized. When the maintenance bound B becomes larger, the probability that all attributes will be used in the partitioning process becomes higher.

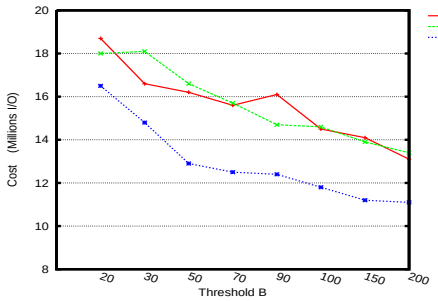

Fig. 7. Query Performance vs. Maintenance Bound B

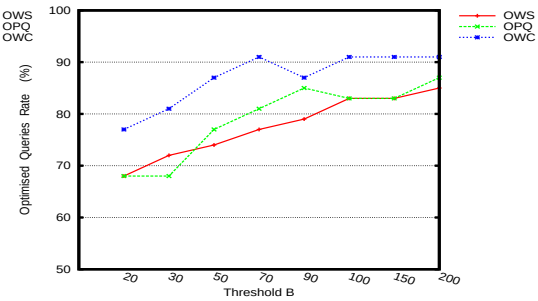

Fig. 8. Optimised Queries Rate vs. Maintenance Bound B

We conduct other experiments to evaluate the impact of the storage constraint S on query performance. To do so, we vary S while fixing the value of maintenance bound $B = 20$ (this value leaves for indexing a large number of attributes). Figure 9 shows the obtained results. We note that, for $S < 900\ Mo$, OPQ gives better results than OWS. Indeed, for these values of S, choosing restriction attributes candidate for indexing from a subset of queries (not profitable) reduces the complexity of the problem of selection indexes. But, when storage space increases, the costs corresponding to OPQ and OWS become linear. Whatever the value of S, our approach outperforms OPQ and OWS. This is

because the selected attributes for indexing are those that give the most benefit for BJIs. Indeed, the query processing cost, for OWC is reduced by 19.1 million I/O when $S = 50MO$ to 15.8 million I/O when $S > 600MBO$.

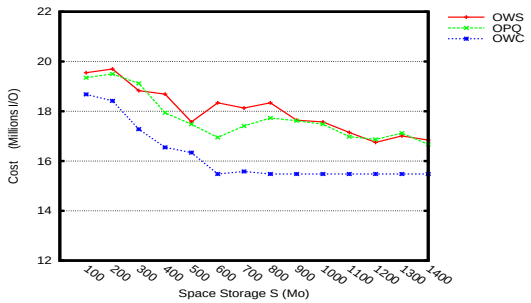

Fig. 9. Query Performance vs. Space Storage S

Tests conducted so far selects HDP schema and then BJI (HDP then BJI). We conduct experiments by considering two other selection orders to show their impact on query performance: BJI then HP and $HP//BJI$ (meaning that each selection is made on its corresponding classes of attributes without taking into account the attributes not selected by the other selection). B and S are set to 500 Mo and 70. Figure 10 summarizes the obtained results. The best performance of combined selection is obtained when OWC selects HDP schema and then BJI which requires 12.5 million of I/O to execute the 47 queries. The second best approach is OWC with $HP//BJI$ order (13.7 millions of I/O). The worst strategy is OPQ with HP followed by BJI order (15.1 millions of IO). Several lessons can be learned from these results: (a) BJI followed by HP gives worst performance. Indeed, the non selected attributes by BJI will be added to the cluster of attributes dedicated for HDP and may distort the choice of optimization scheme. Contrary to the selection of BJI, where an attribute is either chosen or rejected, the selection of a scheme of HDP can choose an attribute with a reduced number of fragments, (b) the results of OWS and OPQ in the mode HP followed by BJI (15.6 and 15.1 million I/O) are better than OWC in the mode BJI followed by HP (17.2 million I/O). This is mainly due to the selection of HDP schema. Indeed, HDP in OWS or OPQ (HP followed by BJI) is done on all restriction attributes. On the other hand, HDP in OWC (BJI then HP) relies on only restriction attributes identified by clustering process, (c) the order $HP//BJI$ is less beneficial than the HDP followed by BJI, since the separation of the two selections do not allow BJI to take into account attributes not selected by HDP.

Recall that the weight classification is based on three factors: Frc, SF and $Card$. To investigate their relevance, we conduct an experiment, where only OWC methodology is used, by changing the weight formula. This gives seven

possible combinations. The results are shown in Figure 11. We note that the best optimization is obtained when all factors are used. These experiments confirm our choice of these criteria.

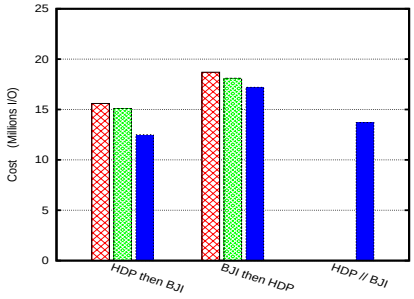

Fig. 10. Variation of selection order between Partitioning and BJI

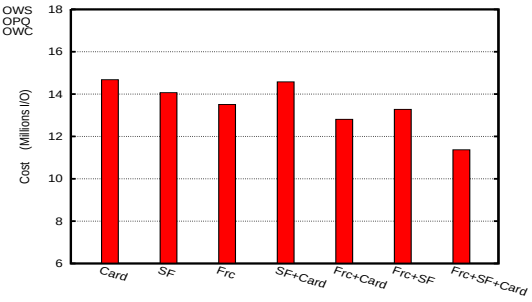

Fig. 11. Factors relevancy for the weight classification

4.3 Description of ASADD

To ease the use of our proposal, we develop a tool, called, ASADD that assists DBA to perform an isolated and a combined selection of HDP and BJI. This tool is developed in Eclipse IDE and integrates all proposed algorithms. ASADD is connected to Oracle11G and offers DBA nice user interfaces to realize the following main tasks: (1) visualization of the data warehouse schema and the workload; (2) choosing of type of selection: isolated and combined with various modes; (3) parameterization of different used algorithms; (4) visualization of the different recommendations proposed by each OT (Figure 13): partitioning and indexable attributes, number of partitions, query execution cost before and after optimization (Figure 12) and (5) generation of final scripts implementing the obtained optimizations.

5 Conclusion

To meet the complex queries requirement, a combined selection of OT has become a key solution. It is more complex than single selection due the large search space that should be explored. In addition to this complexity, interaction between certain OT shall be considered during the selection process. In this paper, we identify a new phenomenon called *attribute interchangeability* that affects dramatically the combined selection problem of two OT: HDP (considered as a non redundant structure) and BJI (considered as a redundant structure). Similarities are identified – both are defined on attributes of dimension tables and optimize restrictions and joins. These similarities are exploited to propose a solution of

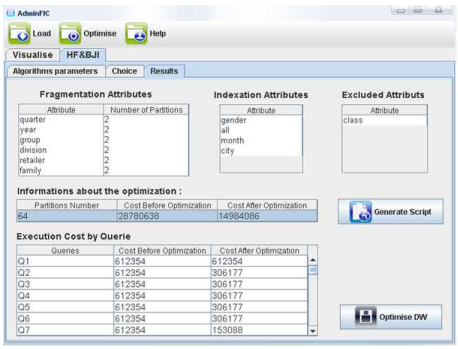

Fig. 12. General view

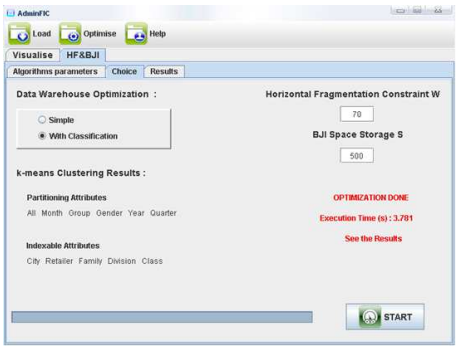

Fig. 13. Recommendations

the combined selection problem that reduces the complexities of two problems. Instead of dealing with the combined problem exploring large search space (for both HDP and BJI selection problems), we propose to first partition restriction attributes between HDP and BJI and then each OT is selected using its own attributes. The clustering is ensured using a K-means method with three criteria identified by empirical experimentations: *attribute frequency*, *selectivity factor* and *cardinality of attributes*. To validate our proposal, we conduct intensive experiments, where a comparison with the existing approaches was proposed. The obtained results show the efficiency of recommendations. To facilitate the use of our methodology and algorithms, we developed a tool (ASADD) offering graphical interfaces to DBA to perform her/his tasks during the physical design phase.

It is interesting to consider other criteria when sharing the restriction attributes, such as storage and profiles of used queries. Other directions that should be addressed are: (i) incorporation in ASADD other OT such as parallel processing and (ii) make it able to handle dynamic changes.

References

1. K. Aouiche, O. Boussaid, and F. Bentayeb. Automatic Selection of Bitmap Join Indexes in Data Warehouses. pages 64–73, August 2005.
2. L. Bellatreche and K. Boukhalfa. Yet another algorithms for selecting bitmap join indexes. In *International Conference on Data Warehousing and Knowledge Discovery (DaWaK'2010)*, pages 105–116, September 2010.
3. L. Bellatreche, K. Boukhalfa, and Zaia Alimazighi. Simulph.d.: A physical design simulator tool. In *20th International Conference on Database and Expert Systems Applications (DEXA'09)*, pages 263–270, 2009.
4. L. Bellatreche, K. Boukhalfa, and P. Richard. Referential horizontal partitioning selection problem in data warehouses: Hardness study and selection algorithms. *International Journal of Data Warehousing and Mining*, 5(4):1–23, March 2009.
5. K. Boukhalfa, L. Bellatreche, and Z. Alimazighi. Hp&bji: A combined selection of data partitioning and join indexes for improving olap performance. *Annals of*

Information Systems, Special Issue on new trends in data warehousing and data analysis, Springer, 3:179–2001, November 2008.
6. S. Chaudhuri. Index selection for databases: A hardness study and a principled heuristic solution. *IEEE Transactions on Knowledge and Data Engineering*, 16(11):1313–1323, November 2004.
7. OLAP Council. Apb-1 olap benchmark, release ii. *http://www.olapcouncil.org/research/bmarkly.htm*, 1998.
8. G. Eadon, E. I. Chong, S. Shankar, A. Raghavan, J. Srinivasan, and S. Das. Supporting table partitioning by reference in oracle. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 1111–1122, 2008.
9. K. EL Gebaly and A. Aboulnaga. Robustness in automatic physical database design. In *in 11th International Conference on Extending Database Technology (EDBT'08)*, pages 145–156, 2008.
10. J. Gray and D. Slutz. Data mining the sdss skyserver database. Techreport Technical Report MSR-TR-2002-01, Microsoft Research, 2002.
11. Y. Ioannidis and Y. Kang. Randomized algorithms algorithms for optimizing large join queries. *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 9–22, 1990.
12. H. Kimura, G. Huo, A. Rasin, S. Madden, and S. Zdonik. Coradd: Correlation aware database designer for materialized views and indexes. *PVLDB*, 3(1):1103–1113, 2010.
13. J. B. MacQueen. Some methods for classification and analysis of multivariate observations. In *Proceedings of 5th Berkeley Symposium on Mathematical Statistics and Probability", Berkeley, University of California Press*, pages 281–297, 1967.
14. H. Mahboubi and J. Darmont. Data mining-based fragmentation of xml data warehouses. In *ACM 11th International Workshop on Data Warehousing and OLAP (DOLAP'08)*, pages 9–16, 2008.
15. C. Maier, D. Dash, I. Alagiannis, A.Ailamaki, and T. Heinis. Parinda: an interactive physical designer for postgresql. In *13th International Conference on Extending Database Technology (EDBT'10)*, pages 701–704, 2010.
16. S. Papadomanolakis and A. Ailamaki. Autopart: Automating schema design for large scientific databases using data partitioning. pages 383–392, June 2004.
17. A. Sanjay, C. Surajit, and V. R. Narasayya. Automated selection of materialized views and indexes in microsoft sql server. In *Proceedings of the International Conference on Very Large Databases*, pages 496–505, September 2000.
18. T. Stöhr, H. Märtens, and E. Rahm. Multi-dimensional database allocation for parallel data warehouses. In *Proceedings of the International Conference on Very Large Databases*, pages 273–284, 2000.
19. M.-C. Wu. Query optimization for selections using bitmaps. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 227–238, 1999.
20. J. X. Yu, C-H. Choi, and G. Gou. Materialized view selection as constrained evolution optimization. *IEEE Transactions On Systems, Man, and Cybernetics, Part 3*, 33(4):458–467, November 2004.
21. D. C. Zilio, S. Lightstone, K. A. Lyons, and G. M. Lohman. Self-managing technology in ibm db2 universal database. In *ACM International Conference on Information and Knowledge Management (CIKM)*, pages 541–543, 2001.
22. D. C. Zilio, J. Rao, S. Lightstone, G. M Lohman, A. Storm, C. Garcia-Arellano, and S. Fadden. Db2 design advisor: Integrated automatic physical database design. In *Proceedings of the International Conference on Very Large Databases*, pages 1087–1097, August 2004.

6 Appendix

Name	Queries
Q1	select Time_level,count(*) from ACTVARS A,PRODLEVEL P where A.PRODUCT_LEVEL=P.CODE_LEVEL and P.Class_LEVEL='P70J55L4HYBV' group by Time_level
Q2	select max(Time_level) from ACTVARS A,PRODLEVEL P where A.PRODUCT_LEVEL=P.CODE_LEVEL and P.GROUP_LEVEL='P66L03PSAFVQ'
Q3	select Time_level,Avg(UNITSSOLD) from ACTVARS A,Timelevel T where A.TIME_LEVEL=T.TID and (t.quarter_level='Q1' or t.quarter_level='Q2') Group by Time_level
Q4	select division_level,count(*) from ACTVARS A,PRODLEVEL P where A.PRODUCT_LEVEL=P.CODE_LEVEL and P.group_LEVEL='S7JWEUJRYIWN' group by division_level
Q5	select Time_level,count(*) from ACTVARS A,PRODLEVEL P where A.PRODUCT_LEVEL=P.CODE_LEVEL and P.family_LEVEL='URT1B3VOSSHH' group by Time_level
Q6	select year_level,sum(Dollarcost) from ACTVARS A,PRODLEVEL P, Timelevel T where A.PRODUCT_LEVEL=P.CODE_LEVEL and A.time_level=T.TID and P.GROUP_LEVEL='P66L03PSAFVQ' group by year_level
Q7	select Avg(Unitssold) from ACTVARS A,PRODLEVEL P where A.PRODUCT_LEVEL=P.CODE_LEVEL and P.family_LEVEL='URT1B3VOSSHH' and P.GROUP_LEVEL='P66L03PSAFVQ'
Q8	select Customer_level,Avg(Unitssold) from ACTVARS A,PRODLEVEL P where A.PRODUCT_LEVEL=P.CODE_LEVEL and P.family_LEVEL='IXG0C4HQG2DW' and P.GROUP_LEVEL='P66L03PSAFVQ' group by Customer_level
Q9	select Product_level,count(*) from ACTVARS A, Timelevel T where A.TIME_LEVEL=T.TID and T.month_level='1' and (t.quarter_level='Q3' or t.quarter_level='Q4') group by Product_level
Q10	select retailer_level,Avg(unitssold) from ACTVARS A,PRODLEVEL P ,Custlevel C where A.PRODUCT_LEVEL=P.CODE_LEVEL AND A.customer_level=C.Store_level and P.DIVISION_LEVEL = 'E6QF1IHDEV0E' group by retailer_level
Q11	select count(*) from ACTVARS A,PRODLEVEL P where A.PRODUCT_LEVEL=P.CODE_LEVEL and P.DIVISION_LEVEL = 'E6QF1IHDEV0E'
Q12	select Avg(Unitssold) from ACTVARS A,TIMELEVEL T where A.TIME_LEVEL=T.TID and t.year_level='1996'
Q13	select Product_level,count(*) from ACTVARS A,TIMELEVEL T where A.TIME_LEVEL=T.TID and T.year_LEVEL = '1995' group by Product_level
Q14	select division_level,Avg(Unitssold) from ACTVARS A,TIMELEVEL T, Prodlevel P where A.TIME_LEVEL=T.TID AND A.product_level=P.Code_level and t.month_level='7' group by division_level
Q15	select count(*) from ACTVARS A,TIMELEVEL T where A.TIME_LEVEL=T.TID and t.year_level='1995'
Q16	select division_level,Sum(Dollarcost) from ACTVARS A,TIMELEVEL T ,Prodlevel P where A.TIME_LEVEL=T.TID AND A.product_level=P.Code_level and (t.month_level='1' or t.month_level='2') and p.Class_level='P70J55L4HYBV' group by division_level
Q17	select month_LEVEL, Avg(Unitssold) from ACTVARS A,TIMELEVEL T, custlevel c where A.TIME_LEVEL=T.TID and A.CUSTOMER_LEVEL=C.STORE_LEVEL and c.gender_level='M' group by month_LEVEL
Q18	select time_level, Avg(unitssold) from ACTVARS A,prodlevel p, custlevel c where A.CUSTOMER_LEVEL=C.STORE_LEVEL and A.PRODUCT_LEVEL=P.CODE_LEVEL and P.group_LEVEL='S7JWEUJRYIWN' and C.RETAILER_level ='ZSTV6MYCBS7U' group by time_level
Q19	select product_level, sum(dollarcost) from actvars a,timelevel t, prodlevel p where a.time_level=t.tid and a.product_level=p.code_level and (t.month_level='1' or t.month_level='2') and (t.quarter_level='Q1' or t.quarter_level='Q2') and P.DIVISION_LEVEL = 'E6QF1IHDEV0E' group by product_level

Name	Queries
Q20	select year_level, Max(unitssold) from ACTVARS A, Prodlevel p, custlevel c, timelevel t where a.time_level=t.tid and A.CUSTOMER_LEVEL= C.STORE_LEVEL and A.PRODUCT_LEVEL=P.CODE_LEVEL and P.DIVISION_LEVEL = 'E6QF1IHDEV0E' and C.RETAILER_level ='ZSTV6MYCBS7U' group by year_level
Q21	select Product_level,Time_level,Avg(unitssold) from ACTVARS A,TIMELEVEL T, custlevel c where A.TIME_LEVEL=T.TID and A.CUSTOMER_LEVEL=C.STORE_LEVEL and (t.month_level='1' or t.month_level='2') and c.gender_level='F' group by Product_level,Time_level
Q22	select year_level,month_level, Max(unitssold) from ACTVARS A,prodlevel p ,Timelevel T where A.PRODUCT_LEVEL=P.CODE_LEVEL and A.time_level=T.TID and (t.month_level='1' or t.month_level='2') and P.DIVISION_LEVEL = 'E6QF1IHDEV0E' group by year_level,month_level
Q23	select count(*) from ACTVARS A,CUSTLEVEL C where A.CUSTOMER_LEVEL=C.STORE_LEVEL and C.City_LEVEL='Dijon'
Q24	select class_level, month_level, Min(unitssold) from ACTVARS A,CUSTLEVEL C, prodlevel p, timelevel t where A.PRODUCT_LEVEL=P.CODE_LEVEL AND A.CUSTOMER_LEVEL=C.STORE_LEVEL and A.TIME_LEVEL=T.TID and C.City_LEVEL='Dijon' and C.RETAILER_level ='ZSTV6MYCBS7U' group by class_level, month_level
Q25	select max(Unitssold) from ACTVARS A,CHANLEVEL CH where A.CHANNEL_LEVEL=CH.BASE_LEVEL AND CH.ALL_LEVEL ='EFGHIJKLMNOP'
Q26	select count(*) from ACTVARS A,CHANLEVEL CH,TIMELEVEL T where A.CHANNEL_LEVEL=CH.BASE_LEVEL and A.TIME_LEVEL=T.TID and (t.quarter_level='Q1' or t.quarter_level='Q2') AND t.year_level='1995' and CH.ALL_LEVEL ='BCDEFGHIJKLM'
Q27	select Time_level,count(*) from ACTVARS A,CHANLEVEL CH,CUSTLEVEL C where A.CHANNEL_LEVEL=CH.BASE_LEVEL and A.CUSTOMER_LEVEL=C.STORE_LEVEL and C.City_LEVEL='Dijon' and CH.ALL_LEVEL ='EFGHIJKLMNOP' group by Time_level
Q28	select channel_Level, sum(dollarcost) from ACTVARS A,CUSTLEVEL C,PRODLEVEL P where A.CUSTOMER_LEVEL=C.STORE_LEVEL and A.PRODUCT_LEVEL=P.CODE_LEVEL and P.CLASS_LEVEL='P70J55L4HYBV' AND C.RETAILER_LEVEL='M5TAHN7GUMLT' and c.gender_level='F' group by channel_Level
Q29	select sum(dollarcost) from ACTVARS A, PRODLEVEL P,TIMELEVEL T where A.PRODUCT_LEVEL=P.CODE_LEVEL and A.TIME_LEVEL=T.TID and (t.quarter_level='Q1' or t.quarter_level='Q2') AND P.DIVISION_LEVEL = 'PHPET5VW6SLG'
Q30	select sum(dollarcost), Avg(Unitssold) from ACTVARS A, CUSTLEVEL C,TIMELEVEL T where A.CUSTOMER_LEVEL=C.STORE_LEVEL and A.TIME_LEVEL=T.TID and (T.month_LEVEL = '1' or T.month_LEVEL = '2') and C.RETAILER_level ='ZSTV6MYCBS7U' and C.CITY_LEVEL='Dijon'
Q31	select Customer_Level, Time_level,Min(unitssold) from ACTVARS A,TIMELEVEL T, CUSTLEVEL C where A.CUSTOMER_LEVEL=C.STORE_LEVEL AND A.TIME_LEVEL=T.TID and (t.quarter_level='Q3' or t.quarter_level='Q4') AND C.GENDER_LEVEL='M' group by Customer_level, Time_level
Q32	select month_Level, all_level, Time_level, Sum(Dollarcost) from ACTVARS A, CUSTLEVEL C,TIMELEVEL T ,Chanlevel H where A.CUSTOMER_LEVEL=C.STORE_LEVEL and A.channel_level=H.Base_level and A.TIME_LEVEL=T.TID and T. month_level='2' and T.YEAR_LEVEL='1995' and C.RETAILER_LEVEL ='COGHFPROJP9Z' AND (c.city_level='Paris'or c.city_level='Poitiers') group by month_level,all_level, Time_level
Q33	select count(*) from ACTVARS A,CHANLEVEL CH,TIMELEVEL T where A.CHANNEL_LEVEL=CH.BASE_LEVEL and A.TIME_LEVEL=T.TID and (t.quarter_level='Q1' or t.quarter_level='Q2') AND T.month_LEVEL = '7' and CH.ALL_LEVEL ='BCDEFGHIJKLM'

Name	Queries
Q34	select time_level, Max(Unitssold) from ACTVARS A, CHANLEVEL H,CUSTLEVEL C,TIMELEVEL T where A.CUSTOMER_LEVEL=C.STORE_LEVEL and A.CHANNEL_LEVEL=H.BASE_LEVEL and A.TIME_LEVEL=T.TID and (T.month_LEVEL = '1' or T.month_LEVEL = '2') and t.year_level = '1996' and c.city_level='Paris' and C.RETAILER_level ='ZSTV6MYCBS7U' and H.ALL_LEVEL='BCDEFGHIJKLM' group by time_level
Q35	select Min(Unitssold) from ACTVARS A, CHANLEVEL H,CUSTLEVEL C,PRODLEVEL P,TIMELEVEL T where A.CUSTOMER_LEVEL=C.STORE_LEVEL and A.PRODUCT_LEVEL=P.CODE_LEVEL and A.CHANNEL_LEVEL=H.BASE_LEVEL and A.TIME_LEVEL=T.TID and (T.month_LEVEL = '1' or T.month_LEVEL = '2') and t.year_level='1996' and P.family_LEVEL='IXG0C4HQG2DW' and H.ALL_LEVEL='BCDEFGHIJKLM' and t.quarter_level='
Q36	select count(*) from ACTVARS A,CHANLEVEL CH,TIMELEVEL T, custlevel c where A.CHANNEL_LEVEL=CH.BASE_LEVEL and A.TIME_LEVEL=T.TID and A.CUSTOMER_LEVEL=C.STORE_LEVEL and c.gender_level='M' and T.month_LEVEL = '7' and CH.ALL_LEVEL ='BCDEFGHIJKLM'
Q37	select count(*) from ACTVARS A, Prodlevel p,TIMELEVEL T, CUSTLEVEL C where A.PRODUCT_LEVEL=P.CODE_LEVEL and A.CUSTOMER_LEVEL=C.STORE_LEVEL AND A.TIME_LEVEL=T.TID and P.group_LEVEL='S7JWEUJRYIWN' AND (t.quarter_level='Q3' or t.quarter_level='Q4') AND C.GENDER_LEVEL='M'
Q38	select product_level, count(*) from ACTVARS A,CHANLEVEL CH, CUSTLEVEL C where A.CHANNEL_LEVEL=CH.BASE_LEVEL and A.CUSTOMER_LEVEL=C.STORE_LEVEL and CH.ALL_LEVEL ='BCDEFGHIJKLM' and C.RETAILER_level ='ZSTV6MYCBS7U' group by product_level
Q39	select max(Dollarcost) from ACTVARS A, prodlevel p where A.PRODUCT_LEVEL=P.CODE_LEVEL and P.family_LEVEL='HNMC1GO57W3Y' and P.DIVISION_LEVEL = 'E6QF1IHDEV0E'
Q40	select count(*) from ACTVARS A,Custlevel C where A.CUSTOMER_LEVEL=C.STORE_LEVEL and C.GENDER_LEVEL='F'
Q41	select month_level, Sum(Dollarcost) from ACTVARS A,Custlevel C, timelevel t, prodlevel p where A.CUSTOMER_LEVEL=C.STORE_LEVEL and A.PRODUCT_LEVEL=P.CODE_LEVEL and A.TIME_LEVEL=T.TID and C.GENDER_LEVEL='M' and C.RETAILER_LEVEL='ZSTV6MYCBS7U' and P.family_LEVEL='URT1B3VOSSHH' group by month_level
Q42	select count(*) from ACTVARS A, CUSTLEVEL C, prodlevel p where A.PRODUCT_LEVEL=P.CODE_LEVEL and A.CUSTOMER_LEVEL=C.STORE_LEVEL and P.family_LEVEL='HNMC1GO57W3Y' and P.DIVISION_LEVEL = 'E6QF1IHDEV0E' and P.GROUP_LEVEL='P66L03PSAFVQ' and C.City_LEVEL='Dijon'
Q43	select avg(Dollarcost) from ACTVARS A,CUSTLEVEL C where A.CUSTOMER_LEVEL=C.STORE_LEVEL and c.city_level='Dijon' and c.gender_level='M'
Q44	select gender_level,max(Dollarcost) from ACTVARS A,CUSTLEVEL C, prodlevel p where A.CUSTOMER_LEVEL=C.STORE_LEVEL and A.PRODUCT_LEVEL=P.CODE_LEVEL and C.RETAILER_LEVEL='ZSTV6MYCBS7U' and P.DIVISION_LEVEL = 'E6QF1IHDEV0E' group by gender_level
Q45	select product_level, sum(dollarcost) from actvars a,timelevel t, prodlevel p, chanlevel h where a.time_level=t.tid and a.product_level=p.code_level and a.channel_level=h.base_level and t.year_level = '1996' and p.class_level= 'P70J55L4HYBV' and h.all_level='EFGHIJKLMNOP' and P.family_LEVEL='HNMC1GO57W3Y' group by product_level
Q46	select product_level, sum(dollarcost) from actvars a,timelevel t, prodlevel p where a.time_level=t.tid and a.product_level=p.code_level and (t.quarter_level='Q1' or t.quarter_level='Q2') and P.group_LEVEL='S7JWEUJRYIWN' group by product_level
Q47	select Sum(Dollarcost) from ACTVARS A, timelevel t, Custlevel C where A.TIME_LEVEL=t.TID and A.CUSTOMER_LEVEL=C.STORE_LEVEL and C.City_LEVEL='Dijon' and (t.quarter_level='Q1' or t.quarter_level='Q2')