J. King Saud Univ., Vol. 10, *Comp. & Info. Sci.*, pp. 81-106 (A.H. 1418/1998)

An Intelligent Tool for Boosting Database Performance

Hassan Mathkour
Computer Science Department, College of Computer & Information Sciences, King Saud University, P.O. Box 51178, Riyadh 11543, Saudi Arabia

(Received 08 June 1996, accepted for publication 01 October 1996)

ABSTRACT. An intelligent database monitoring and tuning tool is designed and developed. The tool is an integration of a learning system, an expert system shell, and a tuning system that is linked to an RDBMS. The learning system makes use of two learning algorithms to extract commonalties exhibited by the database queries. The concepts learnt by the learning system are verified by a knowledge based system (KBS) maintaining knowledge about the database application, heuristics, and expertise about database systems. The KBS is constructed using the expert system shell. The tool is implemented using the programming language C and the shell Exsys Professional Version 4.0 and linked to the RDBMS Oracle. It is argued that the learning algorithms alone are not sufficient to arrive at the most suitable tuning decisions as they only recognize the syntactical forms of the queries. The presence of knowledge based systems makes it possible to see through the semantic aspects of the attributes involved in the queries.

Keyword(s): Database Systems: Physical Design, Database Administration, Performance, Monitoring, Tuning; Artificial Intelligence: Concept Learning, Expert Systems.

1. Introduction

A relational database system should be able to serve the users in a timely, responsive, and cost-effective manner. To achieve these objectives, monitoring the performance of the database and taking corrective actions to enhance service to the users are important functions which greatly contribute to the acceptance and success of the database. Monitoring the performance is necessary in evolving and large database applications where maintaining an optimal performance of the database is of utmost importance. Monitoring the performance and anticipating potential problems allow for tuning the system appropriately to maintain a proper level of performance. Tuning the system may require reorganization of the physical database which, in turn, may affect the logical

database. This indicates that the process of monitoring and tuning a database can be of a complex nature. Thus, a careful planing for such a process is necessary.

In relational database systems, a query optimizer determines the optimal execution plan for a given query using the available structural information, but it does not add any new performance tuning mechanism to the database [9]. Consequently, monitoring and fine tuning the system's performance becomes the responsibility of the Database Administrator (DBA). The growing demands of modern and complex database applications have made this task more difficult to be handled by a human-being (i.e., the DBA). Although some monitoring tools have become an integral part of the database packages, they lack the ability of fine tuning the system. Such tools provide some sort of bench-marking facilities which assist the DBA to analyze the information collected by the tools and take proper action manually to correct the problems [3, 18, 19].

The information that are collected while monitoring a database includes transaction execution time, number of database accesses per transaction, number of database accesses per table, number of database accesses per disk pack, transaction response time, number of *ad-hoc* queries processed and their characteristics with respect to database access, record type access, etc. Such information is then used to tune the performance of a specific statement of the query language such as SQL, the performance of a specific database application, and/or the overall performance of all concurrent users and applications [16, 18].

The tuning of a database starts at the physical level, and this is usually accomplished by adjusting new indices [5]. Adjustment of the new indices without any proper planning may result in a worse performance. The process of adjusting indices should take into consideration the proper situation, location, and structure [12]. Moreover, it should be clear where and when those indices should be added or removed. This variable nature of indices is due to the changes in the real-world needs that are represented by the structure of the database at hand. It is observed that an effective tuning decision depends on accurate comprehensive and useful information collected through monitoring.

The impact of tuning the database system based on the information that are collected during monitoring has been observed in actual application environments. For example, while working on a large database for a telecommunication organization running DB2 on an IBM machine, the response time for some application program was reduced from ten hours to two hours after a proper monitoring and tuning. Although, it still is a long execution time for an application program, it demonstrates the role tuning plays in improving the performance of a database system. The availability of some additional information might have made it possible to further reduce the response time [22, 23, 24].

Several work to enhance the database performance is available in the literature [2, 3, 4, 10, 11, 12, 19, 21, 26]. Related to our work is that in [12] where several learning algorithms [6, 7, 13, 14, 17] are discussed and studied. They conclude that the learning algorithms UNIMEM [13] and COBWEB [6] are suitable to extract useful information from database queries. They propose some modifications to these algorithms to take into account pertinent database operations such as *Join*. We follow their findings and adopt their modified algorithms in the implementation of our tool. However, we note here that the learning algorithms seem to address the syntactical aspects of the attributes appearing in the queries, but fail to realize the semantic aspects of the attributes. Further, the work in [12] is not integrated into any DBMS due to their LISP implementations [12, 13].

In this paper, we report on the design and development of a tool which is intelligent and integrated with the Oracle DBMS. The tool partly uses two learning algorithms UNIMEM and COBWEB to collect information about the queries based on their syntactical forms. It uses knowledge maintained by the expert system shell about the database to unveil the semantic properties of the database attributes. These syntactical and semantic information are consolidated to suggest appropriate and valuable actions to fine tune the database. The tuning is further automated by incorporating knowledge about the database, heuristics, and expertise collected from a variety of DBAs and maintained by the expert system shell Exsys Professional.

The remaining of this paper is organized as follows: In Section 2, we outline and compare the main features of the learning algorithms, UNIMEM [13], COBWEB[6], and CLASSIT [7]. In Section 3, we give the architecture of our proposed tool and describe its components. In Section 4, we describe the knowledge acquisition phase of our tool. In Section 5, we discuss the behavior of the system on a real-world database application and point out pertinent remarks. Finally, in Section 6 we give our conclusion and future work.

2. The Learning Algorithms

The classification properties of the learning algorithms [1, 6, 7, 8, 13, 14, 17] such as UNIMEM [13], COBWEB [6], and CLASSIT [7] appear to be useful in extracting commonalties from the database queries. The extracted information may be used for improving database performance. The learning algorithms build hierarchies of concepts. Each concept in the hierarchy represents a classification of examples in the selected domain. The input to the algorithms is a series of examples (or instances) that are fed one at a time. Each instance is represented as a set of attribute-value pairs. Although the incoming instances are not pre-classified, the algorithms can detect similarities and organize the instances into a *concept hierarchy*. These algorithms are

unsupervised learning algorithms which are capable of learning the concepts through observation of their positive instances. They are able to learn incrementally, updating the concept hierarchy as a new instance arrives. All these algorithms perform the task of conceptual clustering by generating a concept hierarchy, but differ in the methods by which the concept hierarchy is maintained. They also use different evaluation functions to add an instance to the concept hierarchy. In the concept hierarchy generated by the algorithms, the nodes at higher levels represent general concepts, and those at the lower levels in the hierarchy represent sub-generalizations of the higher level nodes, i.e., the children are more specific concepts of the parent node. In CLASSIT, concepts lower in the hierarchy have attributes with lower *standard deviations*. In COBWEB, the instances input to the algorithm are stored only at the terminal nodes of the concept hierarchy. CLASSIT and UNIMEM store an instance in a concept which need not to be a terminal node. A complete example of the construction of concept hierarchies using UNIMEM and COBWEB, which are implemented in our tool, is given in section 5.1.

A brief comparison among the three learning algorithms is given in Table 1. The algorithms are analyzed and compared with respect to knowledge representation, the search mechanism, the information that can be derived from the knowledge stored in the hierarchy, overlapping concepts (i.e., an instance is classified in more than one concept), and performance of the hierarchies.

3. The Proposed Tool

The proposed tool makes use of the learning algorithms and the database knowledge to analyze user's queries on a database and attempt to arrive at possible tuning suggestions. This is illustrated with an example in Section 5.1 and Section 5.2. The suggestions are passed to the tuning module of the tool which performs the restructuring of the database. The proposed tool also makes use of an expert system shell, which maintains a knowledge base to facilitate the tuning function of the database. The tool also performs possible reorganization of the database under the supervision of the database administrator (DBA). The overall structure of the proposed tool is illustrated in Fig. 1. In the next sections we describe the main functions of each of its components.

3.1. The expert system shell

The shell is a back-end component of the tool which is used to assist the learning system to learn about the database attributes, and the tuning system to make appropriate tuning actions. The learning system uses the knowledge stored in the shell about the queries and the database at hand. This knowledge is based on the heuristics and characteristics of the actual attributes of the input queries as they are stored in the data dictionary. The tuning system uses the knowledge constructed from the information obtained from the learning system, database expertise, previous experience (from the

Table 1. Comparative summary of the algorithms

Characteristic	Unimem	Cobweb	Classit
1- Attribute value	• Nominal, single and multivalued (sets) • numeric	• Nominal, single valued	• Real-valued
2- Concept descriptions	• Terminal and non-terminal nodes	• Terminal nodes	• Terminal and non-terminal nodes
3- Handling missing values	• Yes	• Yes	• Yes
4- Overlapping concepts	• Yes	• No	• No
5- Hill climbing search	• Yes	• Yes	• Yes
6- Concept deletion	• Yes (deletes an overly specific concept)	• No	• No.
7- Sensitivity to the order of inputs	• Yes	• Node merging and node splitting operators are provided to recover from the sensitivity of the hierarchy to the order of inputs	
8- Supports multiple inheritance	• -	• -	• -
9- Handles exceptions	• Yes	• -	• -
10- Predictive ability	• No	• Yes	• Yes
11-Representation of ignorance	• No	• No	• No
12-Overlapping domains	• Yes	• No	• Yes

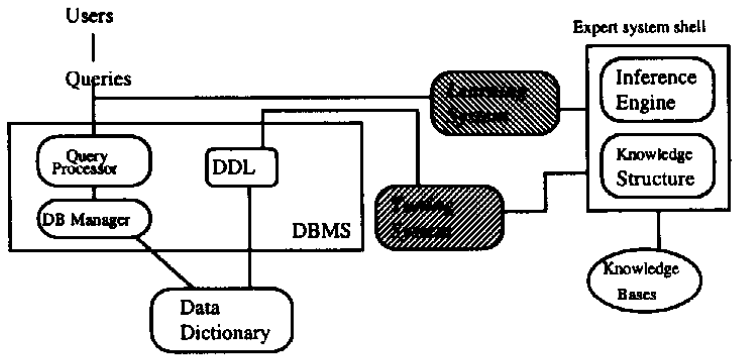


Fig. 1. System architecture of the proposed tool.

DBA), and heuristics as well as the characteristics of the DBMS at use. The knowledge is represented in the form of rules. A sample of the rules is given in Table 2 and explained in Section 4.2. The shell also uses the statistics that can be obtained from statistical tool which is an integral part of the database management system. The shell checks for the consistency of the statistical data, that are collected when the queries are executed, with the information obtained from the learning system. A sample of the statistical data is given in the Appendix. The shell adopted for our tool is a readily available rule-based system. Simplicity, availability, and connectivity were the main criteria to select the shell [15, 20]. We integrated Exsys Professional Version 4.0 for Windows as part of our system. This shell supports forward and backward chaining inferences for all or specific parts of the knowledge base. Its connectivity with other programs and applications is done via calls, datafiles, direct commands, etc. This makes it flexible for automatic and real time data acquisition. It also makes it possible to perform certain calculations using routines that are written in a programming language such as C which is being used in the development of our tool. These routines are activated by the rules to perform necessary calculations to assign values for the appropriate parameters that appear in the rules (see Table 2).

Table 2. A sample of the rules

If No_of_Concept_Levels = 5 **and** Level_of_Concept = 2 **then** Concept_Level = HIGH

If Table_Size = LARGE **and** DASD_Availability = YES **and**
CPU_Power_Availability = LOW and DBMS_Buffer_Size = MEDIUM **and**
Type_of_Table_Applications = CRITICAL **then** Index_Construction_Overhead = HIGH

If Concept_level = HIGH **and** Concept_Example_Number = HIGH **and**
Concept_No_of_Features = HIGH **and** Concept_No_of_Descendants = NONE **then**
Concept_Worthiness = HIGH

If Feature_Contribution = MEDIUM **and** Table_ Contribution = HIGH
and Query_Performance = LOW **then** Initial_Index_Indicator = HIGH

If Index_Used_On_Feature = YES **and** Index_Used_On_Other_Features = HIGH **and**
Sort_Involved = YES **and** Join_Performed = YES **then**
Query_Plan_Overhead_Characteristics = EXCESSIVE

If Query_Execution = LONG **and** Query_Plan_Overhead_Characteristics = EXCESSIVE **and**
Query_Expected_Results = SHORT **then** Query_Performance = MEDIUM

If Object_Number=MANY **and** Join_Method_Used_In_Plan = MERGE **then**
Index_Join_Applicability = YES

3.2. The tuning system

The tuning system uses the expert system shell to suggest appropriate tuning actions for the database. It is semi-automatic and it may require the supervision of the DBA to perform the correct decision during the reorganization process of the database. It consists of two basic subsystems (see Fig. 2): The *Main Process Subsystem(MPS)* and the *Input Process Subsystem(IPS).*

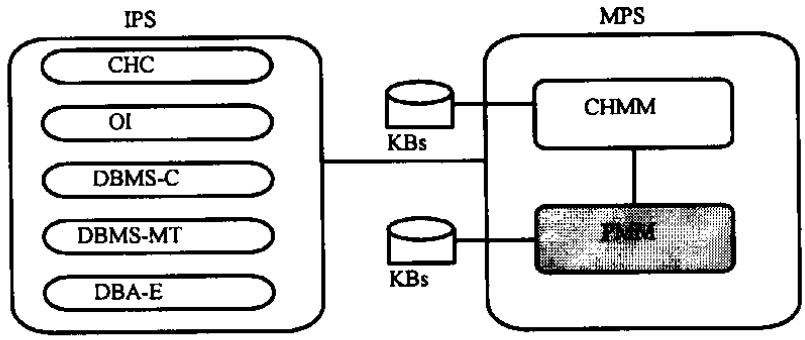


Fig. 2. The tuning system.

The *Input Process Subsystem (IPS)* is responsible for managing all requests for the MPS. It is the interface of the tuning system with the DBMS, learning system, and the DBA. It consists of five modules (see Fig. 2) which are outlined as follows:

The Conceptual Hierarchy Conversion module (CHC): This module provides the MPS with the conceptual hierarchy's related knowledge which includes parameters such as the number of concept levels, classified examples, and features.

The Object Information module (OI): This module provides the MPS with the information related to the DBMS objects. Such information is mainly obtained from the data dictionary (DD). It includes parameters such as object type, table size, number of table attributes, and average number of attributes.

The DBMS Characteristics module *(DBMS-C)*: This module monitors resources of the operating system s pertaining to the DBMS. It also provides information about the various options available by the DBMS. The information provides values for the DBMS parameters such as buffer size, number of pages in the buffer, and the DBMS indexing support.

The DBMS Monitoring Tools module (DBMS-MT): This module analyzes the query plans. It provides information for parameters such as the query execution time, join Method, and type of table applications.

The DBA Expectation module (DBA-E): This module provides information on parameters such expected use of indices, expected CPU time, and expected use of resources.

The *Main Process Subsystem (MPS)* is responsible for analyzing the inputs from the IPS and the monitoring tool with the assistance of the expert system shell. It then generates possible tuning suggestions. It follows the steps given in Fig. 3, and consists of two modules: the Conceptual Hierarchy Manager Module (CHMM) and the Plan Manager Module (PMM) which are outlined below. A more detailed description of the tuning system can be seen in [22, 23].

a. The Conceptual Hierarchy Manager Module (CHMM): This module analyzes the information provided by the learning system together with the collected knowledge, the same way a DBA would, to produce possible tuning actions (an *initial tuning plan*). It works in three phases (see steps 1 to 4 in Fig. 3). In the first phase, the position, contents, and number of related queries of a concept within the conceptual hierarchy are examined to determine how valuable the concept is. In the second phase, CHMM examines the characteristics of each feature, the database objects it references, and the performance of the queries which contain it to decide whether an index should be created or not. In the third phase, the structural features of the suggested indices are recommended.

b. The Plan Manager Module (PMM) receives the *initial tuning plan* from the CHMM and applies several tests on the plan to arrive at a final tuning plan to be applied directly to the target database. Some of the tests performed by the PMM require the expert system shell to make use of the DBA's knowledge that has previously been acquired and stored.

3.3. The learning system

This system generates part of the information based on the database queries. It executes the two learning algorithms that provide for the construction of hierarchies of features in the queries to identify repeating queries. A discussion about the behavior of the algorithms using an actual database is given in Section 4.2. The algorithms were implemented using the programming language Con a SUN SPARC 690MP with UNIX platform. The learning system is linked to the Oracle database management system running on the same server. Ioannnidis et al [12] discuss an implementation of the algorithms using LISP. Their work is not a part of an integrated monitoring and tuning tool, and it is not directly used by a specific DBMS. The choice of the programming language LISP makes it difficult to use their work with a commercially available DBMS for automatic monitoring and tuning.

1. For each node (concept) in the concept hierarchy:
   Examine whether the concept is valuable (i.e., whether the concept is worth considering) or not.

2. For each valuable concept:
   Assign a judgment value ("LOW", "MEDIUM", or "HIGH") to the feature indicating its contribution, call it feature contribution.

   Examine the information of the table pertaining to the attributes in the feature and assign a judgment value indicating the effect of such information, call it table contribution.

   Examine the information in the query execution plan and assign a judgment value identifying the contribution of the parameters in the execution plan, call it plan contribution.

3. Use the feature, table, and execution plan contribution to assign a judgment value on how applicable the feature's attributes are for indexing.

4. Use the type of the feature operation, the number of tables, the DBMS's parameters to suggest a possible indexing structure.
   The suggested indices from step 3 together with the suggested structures constitute a preliminary tuning plan, referred to as the *initial tuning plan*.

5. Use the table and environment information to assign a judgment value for possible overhead arising from the index construction arrived at in steps 3 and 4, call it overhead value.

6. Use the overhead value to determine the practicality of the suggested indices from step 3 and 4. All indices that are considered practical constitute the *final tuning plan*.

Fig. 3. Steps of the tuning system.

The algorithms build concept hierarchies based on the input that arrives. The resultant concept hierarchy may be non-linear. Therefore, the data structure is defined around a non-linear hierarchy tree. Each node of the hierarchy representing a concept is a combination of two lists: A list that maintains the names of instances which are added to the concept and another list which maintains the list of features. The data structures are expressed using self-referential pointers in the programming language C as follows:
The list of instances stored at the node is represented as:

```
typedef struct list_instance
        {char item[25];
        struct list_instance  *next;
        } inode;
```

Each feature is represented by the name of the attribute, the value, and the confidence value. The list of features is represented as follows:

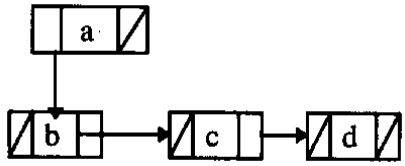
```
typedef struct feature_list
        {char attr[20];
        char value[20];
```

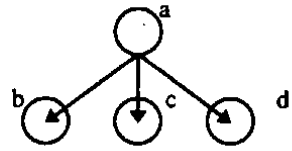
```
int feature_conf; /* UNIMEM stores feature confidence values */
struct feature_list *next;
} fnode;
```

The hierarchy is represented by the tree structure as follows:

```
typedef struct hierarchy_node /* identifies a concept node in the hierarchy */
        {inode *in;
        fnode *fm;
        struct hierarchy-node *left, *right;
        } hnode;
```

The *hierarchy_node* type definition identifies the concept node of the clustering hierarchy. The left pointer points to the child node and the right pointer points to its sibling (see the following figures):





The inputs to the algorithm are attribute-value pair represented as follows:

```
typedef struct /* an attribute-value pair defines an instance feature */
        {
        char attr [20];
        char value [20];
        } atval
```

An instance contains a set of features, and each feature is a representation of an attribute and its value.

The input to the learning system is a set of queries. The module parses the queries and extracts features from them. Features are represented as attribute-value pairs. The extraction of the features is done by the first module of the learning system which parses the incoming queries. At the beginning, the input data to the learning system are unclassified. The goal is to achieve a classification of these instances and to build conceptual hierarchies indicating the frequencies of the database attributes appearing in the queries. The important information of these hierarchies is passed on to the tuning system after validating it with the knowledge maintained by the expert system shell. The tuning system uses such information together with the information obtained from the

built-in database tools and pertinent knowledge from the expert system shell to suggest certain tuning actions to boost the performance of the database system. Such tuning actions may result in the reorganization of the database.

4. The Knowledge Acquisition

The knowledge collected for our tool is divided into two categories: application specific and general database performance knowledge. The application specific knowledge is dependents on the application at hand and is acquired during the design phase of the database application. Such knowledge helps in understanding the semantics of the attributes. The general database performance knowledge requires a special research and acquisition procedure. With the exception of the specific DBMS information, such knowledge will be permanently available with the system and it is applied to identify possible tuning actions as explained in the university database example in Section 5.2.

The general database performance knowledge was acquired from two primary sources: Domain experts and textual documents. The domain experts considered are experienced database administrators (DBAs) and application programmers. A number of DBAs and application programmers were selected for interviews and questionnaire surveys. The documents that were taken into accounts included available literature as well as performance and training manuals from various vendors such as available documents on IBM DB2 V-2.3 RDBMs and Oracle V-7.1 (more details can be seen in [25]).

In acquiring textual knowledge from the available documents, related books and research papers were reviewed first. Such documents were helpful in establishing the foundation necessary for understanding the theoretical aspects of the monitoring and tuning process. The manuals were found helpful in unveiling related monitoring and tuning parameters, and in explaining aspects such as tuning applications, performance problems, frequencies of tuning, etc. In other words, pertinent monitoring and tuning parameters were identified through the textual knowledge acquisitions. These parameters were explored in the domain-expert knowledge through questionnaires and interviews with a selected group of DBAs and application programmers.

The domain-expert knowledge acquisition addressed mainly the technical know-how aspects of the monitoring and tuning process. Before acquiring the desired knowledge from the domain experts, presentations of our views of a monitoring and tuning tool were given to the participants. A set of questionnaires was then developed and delivered to the participants. Following that series of interviews were scheduled with the participants. The interviews appear to be the more productive acquisition

methods for both sides despite being time consuming. The interviews addressed aspects such as the tuning techniques and their effectiveness, the situations to be considered, the monitoring tools, extracting knowledge from the monitoring tools, and the indexing techniques.

As a result, a set of about 700 rules were developed. These rules constitute the knowledge base of the tool. The knowledge base is validated by the expert system shell used, namely, Exsys Professional Version 4.0 for Window. Exsys is equipped with a validation function which when invoked, it performs a completeness check of the rules as well as any possible contradictions that may arise from different rules in the knowledge base.

The rules address the various parameters collected in the acquisition phase and the interrelationships among them. Each parameter belongs to one of five interrelated categories which constitute the domain of the knowledge base. These categories are: the hierarchies constructed by the learning algorithms, the DBMS object information from the data dictionary, the DBMS characteristics, the DBMS monitoring statistics, and the DBA heuristics. The hierarchies provide information that are related to the concepts (their levels and frequencies), classified queries, syntactic aspects of the attributes, etc. The DBMS object information includes: referenced object type, table size, average number of attributes in a database, etc. The DBMS characteristics include: the buffer size, buffer number of pages, CPU power availability, etc. Information from the DBMS monitoring statistics include: execution time, join method used in plan, table position in plan, etc. The DBA heuristics include: expected use of indices, suitable index structures, expected CPU time, expected use of resources, etc.

The parameters in the rules are assigned numeric values as well as judgment values such as "LOW", "MEDIUM", and "HIGH". For example, the parameters "No-of-Concept-Levels" and "Level-of-Concept" in the first rule in Table 2 are assigned the values *5* and *2* respectively indicating the number of levels in the hierarchy and the level at which the concept appears. The contribution of both parameters suggests the assignment of the judgment value "HIGH" to the parameter "Concept-Level" in the same rule indicating that the concept appears at a high level of the hierarchy which means that features in this concept are of special importance. This is based on experiments with the hierarchies generated by the learning algorithms as also observed in [12]. The judgment value may also be obtained through the invocation of attached procedures to approximate certain numeric values for the parameters depending on their natures. These are the routines (that are written in the programming language C) alluded to earlier which are invoked by Exsys automatically upon the activities of a pertinent rule. In some cases, numerical scales have been developed based on heuristics and DBA's expertise. A parameter may be assigned a value, that fall within a corresponding scale, which will be in the range of the value "HIGH", "MEDIUM", or "LOW". A parameter may also be

assigned a value as a result of assigning various values to different but interrelated parameters that affect it. For example in the second rule in Table 2 the value "HIGH" is assigned to the parameter "Index-Construction-Overhead" to indicate the cooperative effect of the parameters Table-Size, DASD-Availability, CPU-Power-Availability, DBMS-Buffer-Size, Type-of-Table-Application when they are assigned the judgment values, "LARGE", "YES" "LOW", "MEDIUM", and "CRITICAL", respectively.

The rules together with a set of attached procedures (C routines) are used to suggest the possible benefit from the application of indices to the tables in question and the suitability of a given attribute to be taken as an index. In addition, an appropriate index structure will be suggested. An index may be structured as a B-tree, B^+-tree, or a hash structure. For example, "index_Structure_Information=BPLUS" indicates that a B^+-tree structure is suitable for the situation at hand. The choice of an appropriate index structure is based on the information available about the queries, the tables, the attributes, and the heuristics and essential knowledge about the database system which is incorporated in the knowledge base.

5. A Case Study

A database example is employed to experiment the proposed system and to gather statistics on a set of queries. The database example is of a reasonable complexity and it is representative of the real-world databases. The database is for an educational institute with several colleges each of which has several departments. Each department offers several courses for students. A department has several instructors teaching different classes in different rooms. A portion of the schemes of this database is given in Table 3. We refer to this database as the university database.

The database was created and loaded with real data on Oracle database management system (DBMS), Version 7.12 running on the SUN machine. Several queries were executed and statistical data was collected through the utilities provided with the Oracle DBMS. The queries were also fed to the learning system to collect information to be used for tuning purposes.

Table 3. An example database - university database

COLLEGE	=	[CG-CODE, CG-NAME, ADDRESS]
DEPARTMENT	=	[D-CODE, D-NAME, CG-CODE]
INSTRUCTOR	=	[I-ID, I-NAME, RANK, PHONE, SEX, D-CODE, LOAD]
STUDENT	=	[ST-ID, ST-NAME, ST-LEVEL, ST-CR-HRS, MAJOR, AGE, D-CODE]

5.1. Query analysis

In this section, we discuss the behavior of the learning system implementing the learning algorithms UNIMEM and COBWEB. A sequence from the following input data extracted from queries on the university database were fed to the learning system:

Q1 : (DEPARTMENT.D-NAME = "CS") (DEPARTMENT.D-CODE = INSTRUCTOR.D-CODE) (INSTRUCTOR.LOAD = 10)
Q2 : (DEPARTMENT.D-NAME = "CS") (DEPARTMENT.D-CODE = INSTRUCTOR.D-CODE)
Q3 : (DEPARTMENT.D-NAME = "CS") (DEPARTMENT.D-CODE = INSTRUCTOR.D-CODE) (INSTRUCTOR.RANK = "PROFESSOR")
Q4 : (DEPARTMENT.D-NAME = "CS")(DEPARTMENT.D-CODE = STUDENT.D-CODE)(STUDENT.ST-LEVE = 4)
Q5 : (DEPARTMENT.D-NAME = "CS") (DEPARTMENT.D-CODE = STUDENT.D-CODE)(STUDENT.ST-LEVEL = 4)(STUDENT.ST-CR-HRS> 20)
Q6 : (INSTRUCTOR.SEX = "FEMALE) (INSTRUCTOR.LOAD = 10) (INSTRUCTOR.RANK = PROFESSOR)
Q7 : (STUDENT.ST-LEVEL = 4) (STUDENT.ST-CR-HRS > 20)
Q8 : (DEPARTMENT.D-NAME = "CS") (DEPARTMENT.D-CODE = STUDENT.D-CODE) (STUDENT.ST-LEVEL = 4) (STUDENT.ST-CR-HRS > 20) (STUDENT. AGE<20)

Given the sequence Q1, Q2, Q3, Q2, Q2, Q1, Q1, Q2, Q4, Q5, Q4, Q5, Q2, Q5, Q6, Q7, Q8, Q2, Q8 formed from the above queries as input to the learning system's module implementing the UNIMEM algorithm, the module constructs the concept hierarchies as depicted in Fig. 4 to Fig. 10 and explained in Step 1 through Step 6.

The UNIMEM module uses depth-first search strategy to update the hierarchy. As unclassified instances arrive, the module searches down the hierarchy to match the features of the unclassified instance with the features stored at a concept to form new concepts. Feature confidence values are stored with every feature in the concept and are modified with every new query entering the system. The result of the module is a dynamically changing hierarchy with incoming queries. The result of clustering is a concept hierarchy. Nodes at the higher levels may contain useful monitoring information. Features with high confidence values at the higher levels of the hierarchy are analyzed. The attributes appearing in such features may be considered for indexing that corresponding tables provided that the semantic of the database captured by the knowledge (maintained by the expert system shell) supports such suggestions. The result of the UNIMEM module on the set of queries for our example database is given in the hierarchy structure of Fig. 10. The details of the construction of the concept hierarchies are given in Step 1 through Step 6 and in Figure 4 through Fig.10.

Step 1: The system initializes its hierarchy to a single root-node. A concept is created at the root. Since this is the first instance, no classification takes place. Fig. 4 demonstrates the result of Step 1.

Step 2: When the second query enters the system, the UNIMEM module creates a child node and computes the frequency counts (shown in brackets) of the extracted features based on the similarity among them. The first two features in Q1 match with that in Q2. Fig. 5 demonstrates the result of Step 2.

Step 3: Now Q3 is observed. Q3 is first placed at the root and a depth-first search is applied to the concept hierarchy. It finds Concept-2 and tries to match its features with that of Q3. The match is successful with respect to the first two features of Q3. Q3 is placed in Concept-2 and the corresponding feature counts are updated. The resulting hierarchy is shown in Fig. 6.

Step 4: After the submission of Q2 twice, the hierarchy is updated in the same way as pointed out in Steep 3. The structure of the hierarchy does not change but the features counts in the node of Concept-2 are incremented. When Q1 is resubmitted, the first two features are matched and the third one creates a new child node since Q1 is detected as previously stored instance. The third submission of Q1 does not change the structure but updates the frequency counts of the corresponding features. Similarly, the re-submission of Q2 and Q4 updates the counts but causes no change in the structure. Fig. 7 demonstrates the hierarchy after the sequence: Q1, Q2, Q3, Q2, Q2, Q1, Q1, Q2, Q4. When Q5 is submitted, its first three features match with the features of Q4 which are found at the root. As a result, a new node is created for the matching features of Q4 and Q5, namely, DEPARTMENT.D-NAME="CS", Department.D-Code=Student.D-Code, and Student.St-Level=4. The frequencies counts are updated across the hierarchy. As a result, the frequency count of the feature in the node for Concept-3 decreases which leads to the deletion of that node. Fig. 8 shows the resulting hierarchy. This module of the learning system has lost some of the information that was previously learnt. This is a characteristic of UNIMEM which is implemented by this module.

Step 5: Submission of Q4 updates the frequencies but causes no change in the hierarchy. Submission of Q5 creates a new node in a similar fashion as pointed out earlier. Fig. 9 depicts the new hierarchy.

Step 6: Submission of the queries Q2, Q5, and Q6 updates the frequency counts but leaves the hierarchy unchanged. Finally after the submission of the remaining queries Q7, Q8, Q2, and Q8, the hierarchy in Fig. 10 is constructed in the same way as explained earlier. This is the final hierarchy constructed by the learning system module implementing UNIMEM. It should be noted that if the sequence of queries changes then a different hierarchy my be generated.

Figure 10 shows a three-level concept hierarchy which is the output of the learning system's module implementing UNIMEM when it is fed with the input queries. The root at the zero level represents unclassified concepts, namely, Q6 and Q7. This

means that Q6 and Q7 were not repeated. This does not mean that the features appearing in such queries should be discarded. This is due to the nature of the application at hand. Information from nodes other than the root of the conceptual hierarchy indicates that the features in Q7 may be useful. However, no observation is made for the features of Q6. Consequently, it is reasonable therefore to discard Q6 but not Q7. Therefore, it is concluded that in database applications the information provided by the root of the hierarchy may not be sufficient to discard or qualify certain concepts. Knowledge about the semantic of the database is necessary to support or refute the suggestions from the learning system.

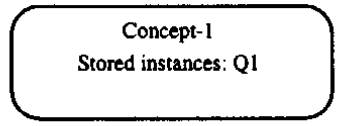


Fig. 4. Result after Q1 (Root is initially empty).

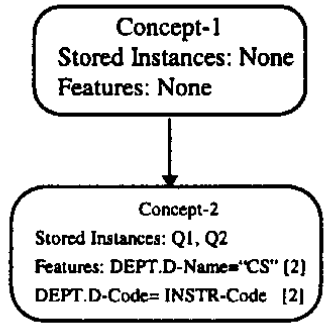


Fig. 5. Result after Q1, Q2.

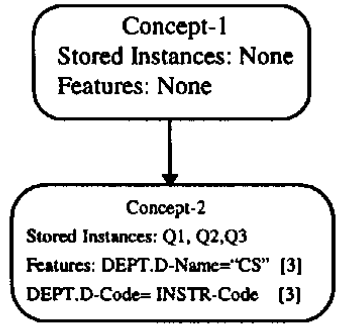


Fig. 6. Result after sequence Q1, Q2, Q3.

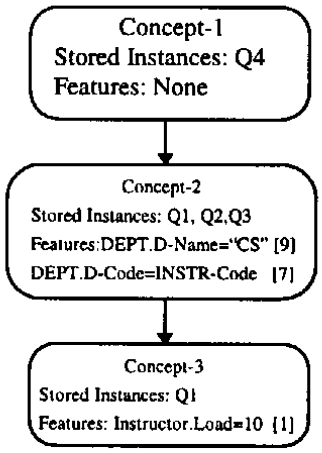


Fig. 7. Result after the sequence Q3, Q1, Q2, Q3, Q2, Q2, Q1, Q1,Q2,Q4.

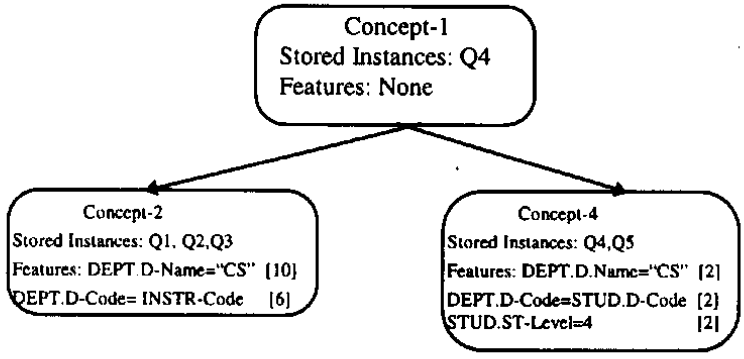


Fig. 8. Result after the sequence Q1,Q2, Q2, Q2, Q1,Q1, Q2, Q4, Q5.

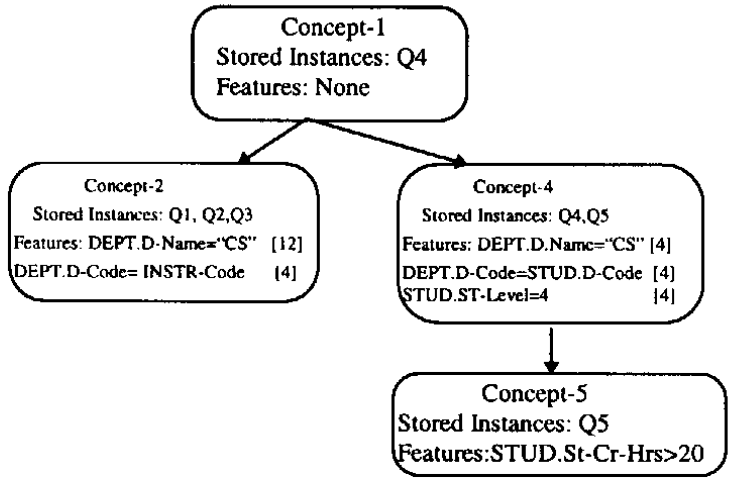


Fig. 9. Result after the sequence Q1, Q2, Q3, Q2, Q2,Q1, Q1, Q2, Q4, Q5, Q4, Q5.

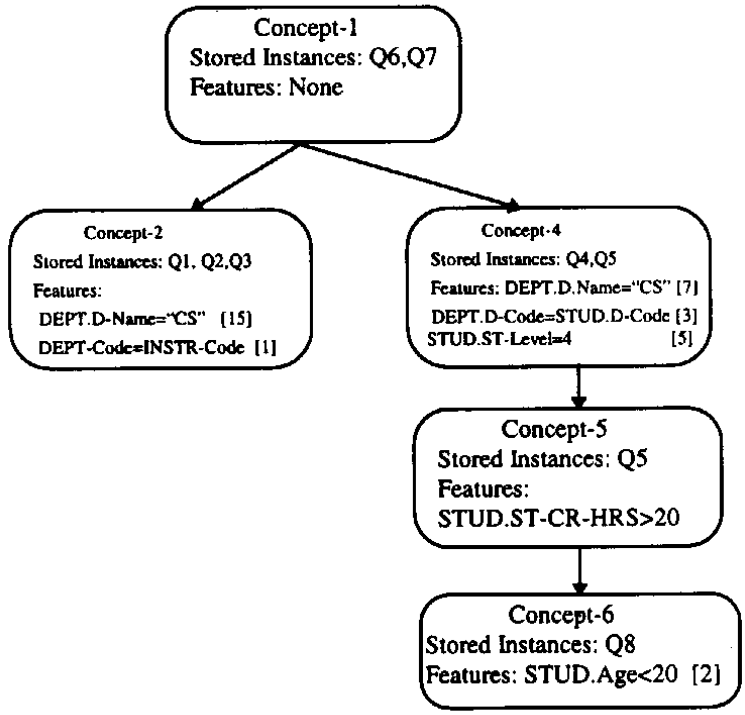


Fig. 10. Result after the complete sequence of the input queries to UNIMEM.

Now, we look at the information provided by the rest of the hierarchy nodes depending on the level and the frequency counts of the features. The higher the level the more pertinent the count is. The nodes at level one (a high level) indicates high count for the feature DEPARTMENT.D-NAME= "CS" (F1). This suggests that special attention should be given to the attribute "D-NAME" of the table "DEPARTMENT". Further analysis concerning the type and applicability of "D-NAME" as an index, and the associated overhead leads to a proper decision of whether a tuning action with respect to "D-NAME" should take place or not. The knowledge stored in the shell is used to support or refute the learning system findings so that a proper tuning decision is reached. Obviously, the characteristics of this attribute in this database confirms the suggestion of the learning system when UNIMEM is applied.

The nodes in the same level (i.e., level 1) show a medium count (considering the input queries) for the feature DEPARTMENT.D-CODE=STUDENT.D-CODE (F2). Considering the conclusion reached from feature F1, this provides a confirmation to consider indexing on "D-CODE" for the table "DEPARTMENT". However, consultation of the expert system shell is necessary to raise the confidence to index "STUDENT" on the foreign key "D-CODE". The frequency count of the feature DEPARTMENT.D-CODE = INSTRUCTOR.D-CODE does not provide sufficient confidence to consider

"D-CODE" as a foreign key for "INSTRUCTOR". The count for the feature STUDENT.ST-LEVEL=4 (on level 1) suggests consideration of indexing "STUDENT" on "ST-LEVEL". Such information will not be supported by the knowledge acquired about the database which should prevail in this case. The frequency counts for the features appearing in the nodes in the second and third levels are not high enough to pursue the attributes appearing in these features. Such information agrees with the knowledge acquired about the database.

The analysis of the concept hierarchy constructed by the learning system when applying the UNIMEM module suggests that the information learnt by such an algorithm should always be verified by the heuristics and knowledge about the database at hand and which is provided by the experts system shell. Further, the learning module may not provide important information about certain attributes. Such information can be obtained from the knowledge base maintained by the tool. The simple computation involved in constructing the hierarchies using UNIMEM makes this algorithm attractive. The integration of a learning system based on UNIMEM and the expert system shell should prove useful in providing sound suggestions to arrive at appropriate and correct tuning actions.

Figure 11 depicts the resulting concept hierarchy when the learning system applies the COBWEB algorithm. Each node indicates the probability of the concept *P(C)*, and the probability and frequency count of the features. The frequency-count represents the number of times a concept has seen that feature. The frequency-count is displayed along the features only in the internal nodes of the hierarchy. However, in the actual implementation they appear along each features stored in every node. The frequency-counts are translated into conditional probabilities of each feature stored at the concept.

The root node of the concept hierarchy in COBWEB is independent of the order of the incoming queries. The root node may provide important information concerning attributes that appear in it. The features at the root node that have high conditional probabilities are considered for further analysis to obtain information of their constituting attributes. The subtrees of the hierarchy are traversed starting from the root node in the direction where the conditional probabilities are high, and thus identifying those features that are more frequently used together in queries. Features with high probabilities at the root node and which appear also in the internal nodes indicate useful information for possible changes in the database. For example, the features (DEPARTMENT.D-NAME = "CS") and (DEPARTMENT.D-CODE = STUDENT.D-CODE) have high conditional probabilities at the root and appear also in an internal node as indicated by the hierarchy in Fig. 11.

The root node in the output of COBWEB indicates a high probability for the feature DEPARTMENT.D-NAME = "CS", thereby, realizing that the attribute

"D-NAME" is of a special interest. This agrees with the UNIMEM module and the information obtained from the expert system shell. It also highlights STUDENT.D-CODE as an important attribute as it gives it a high probability. This information is not captured by the UNIMEM module. The knowledge of the expert system working with the learning system supports such a finding. The COBWEB module favors the feature DEPARTMENT.D-CODE = INSTRUCTOR.D-CODE which is not indicted by the UNIMEM module as favorable. The UNIMEM module misses this feature as it undoes some of its learning over time. In this case, the semantics of the database captured in the expert system shell, laid out as knowledge that was acquired when the database was designed, indicates an agreement with the UNIMEM module rather than with the COBWEB module.

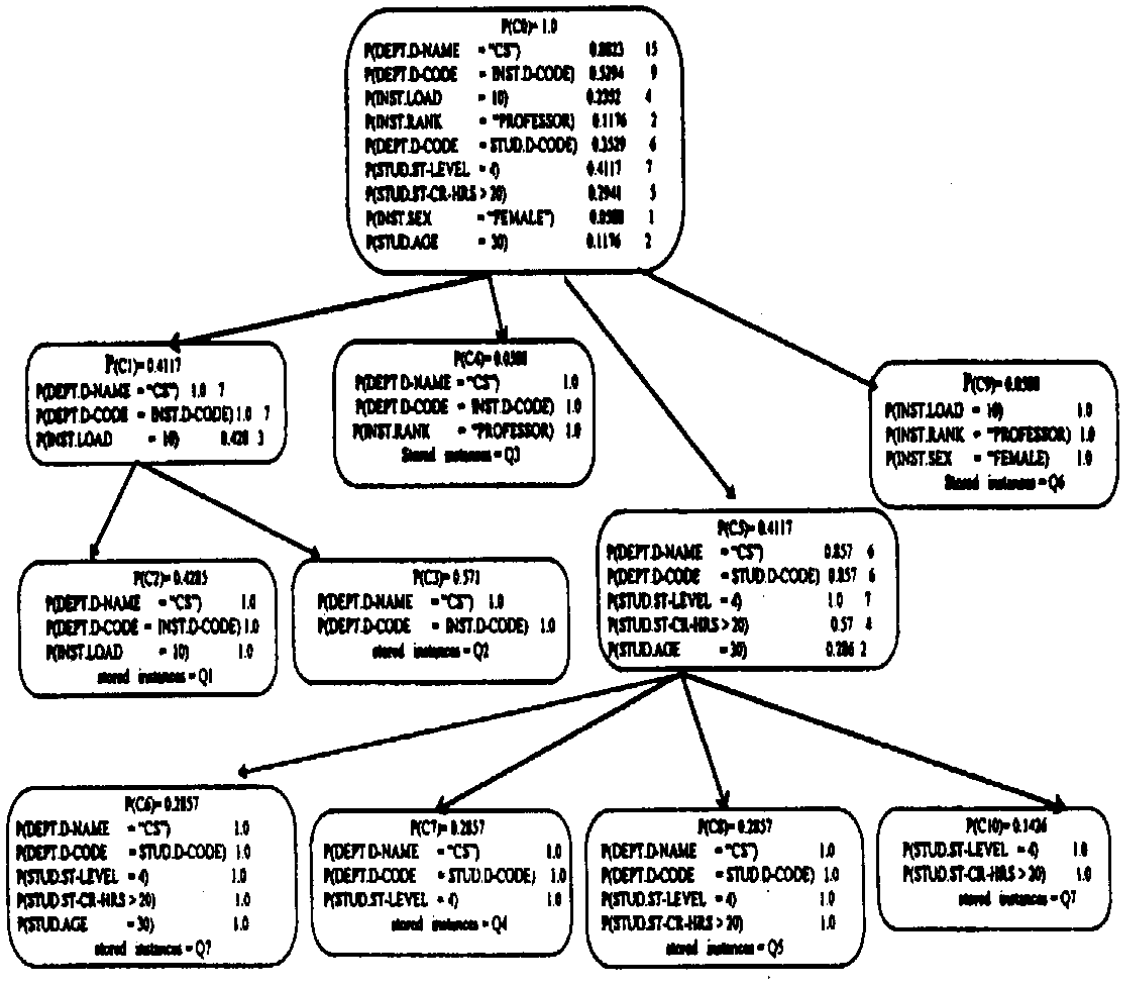


Fig. 11. Result after the complete sequence of the input queries to COBWEB.

Thus, while the COBWEB module is able to capture information that may be missed by the UNIMEM module, such information could be extraneous and misleading. The knowledge stored in the shell seems to be able to correct the findings of the algorithm by supporting or refuting the suggestion of the learning modules. This is reasonable since the knowledge addresses the semantics of the database while the learning algorithms rely on the syntactical form of the queries. The knowledge also gives consideration to other attributes that may be lost as a result of the dynamic changes of the concept hierarchy maintained by the learning system. Such attributes may influence the tuning decisions. Collection of such knowledge for the shell does not introduce extra overhead as it is automatically provided at the design phase of the database. It should also be pointed out that the COBWEB module also favors the attributes ST-CR-HRS and ST-LEVEL which is not favored by the UNIMEM module and not supported by the knowledge of the expert system shell as prominent attributes. In this case, the findings of the UNIMEM module and the expert system shell is more reasonable than that of COBWEB. Considering the computation simplicity of the UNIMEM module, we feel that the UNIMEM module integrated with the expert system shell should suffice. The presence of the COBWEB module may be useful for cross checking the findings.

5.2. Working with the tuning system

The tuning system follows the steps outlined in Fig. 3 to arrive at possible tuning actions. The hierarchies generated by the learning system are passed to the Input Process Subsystem (IPS) of the tuning system. Using these hierarchies together with other data, the IPS prepares the data and knowledge needed by the Main Process Subsystem (MPS). Each module of the IPS is responsible for a specific chunk of such data and knowledge as pointed out earlier. Considering the university database example, the IPS, through its first module, passes the information in the hierarchy which is constructed by the learning system to the MPS. For illustration purpose, we consider here the hierarchy in Figure 10. The first module of the MPS, namely, CHMM, analyzes the information in the hierarchy to determine whether a given attribute that appears in the nodes (i.e., concepts) of the hierarchy is a reasonable candidate for an index or not. In doing so, it first examines whether the concept itself is worth considering. Due to the characteristics of UNIMEM, Concept-1, the root of the hierarchy (see Fig. 10) is not considered except for observing the unclassified examples, namely, Q6 and Q7. Concept-2 in Fig. 10 is tested to determine further possible investigation of its contents. The test considers the concept's level relative to the highest level of the hierarchy, its decedents, the examples (quarries), and the features it addresses. The concept is then assigned one of three values: "HIGH", "MEDIUM" or "LOW" to reflect its worthiness of further consideration. The value "LOW" is a basis to reject the concept. In this example, concepts-2 is assigned the value "HIGH" which means it is considered for further analysis. Once the concept is accepted for further consideration, its features are analyzed and assigned a "HIGH", "MEDIUM" or "LOW" value to reflect its contribution to the attributes that appear in it. The number of objects (and their types) addressed by the feature, the frequency of the feature in other

concepts with the same operation, and the frequency of the feature in other concepts with a different operation are used to arrive at a value for the feature contribution. Consequently the feature DEPT.D-NAME= "CS" in Concept-2, is assigned the value "HIGH". To arrive at such a value, CHMM activates several rules that deal with the feature being analyzed, and the parameters influencing the feature together with the interrelationships among them. Activation of the rules may invoke certain attached procedures that are written in the programming language C (i.e., the C routines mentioned earlier) to calculate necessary values of the appropriate parameters.

After determining the contribution of the concept's features, the CHMM conducts an analysis of the table involving the attributes of the feature to determine the suitability of indexing it. The information needed for such an analysis is provided by the remaining modules of the IPS. The information includes: the table size, the attributes, the key, the existing indices, the buffer size and its number of pages, etc.

Based on the analysis of the feature DEPT.D_NAME = "CS" in Concepts-2, the table is found suitable for possible indexing on the attribute DEPT.D_NAME. Next, the CHMM determines an appropriate structure type for the index (e.g., B^+-tree) and whether a clustering scheme is suitable or not. The information needed for determining the index structure is supplied by all input modules, such information includes: data about the table and its constituents residing in the data dictionary (DD), data about the DBMS, data from the hierarchy related to the concept, the features operation, and encoded knowledge reflecting the DBAs expertise and heuristics.

At this point, the CHMM produces an initial tuning plan which is a suggestion of indexing structures (such as B^+-tree, clustered B^+-tree, etc.) for certain table attributes. In our example, a hash structure is suggested for the attribute DEPT.D_NAME and a clustered B^+-tree is suggested for DEPT.D_CODE. Because of the information obtained from the learning system, the suggestions depend on the queries that are being analyzed.

The initial tuning plan is now analyzed by the PMM to confirm that such a plan is feasible keeping the environment parameters in mind. Such an analysis takes into consideration the table size, the availability of direct access storage device (DASD), the CPU power available, the buffer, and the application type (i.e., critical or otherwise). These parameters were determined by the DBAs who were consulted while developing the proposed tool. Based on the knowledge pertaining to these parameters and which reflect the DBAs expertise, the PMM determines possible overhead that may arise during making such suggestions establishing, therefore, the practicality of the initial tuning plan. A final tuning plan is then generated. In our example, no excessive overhead is determined by the system and therefore the final tuning plan is the same as the initial tuning plan. This is due to the environment under which the experiment was conducted.

6. Conclusions and Future Remarks

A complete, integrated tool to automate the monitoring and tuning of a database system has been proposed. The tool consists of three main components: the learning system, expert system shell, and the tuning system. The learning system and the expert system shell work together to arrive at useful suggestions to be considered for tuning the database. The tuning tool further screens these suggestions and works with expert system shell to refine and implement appropriate and correct tuning actions to boost the performance of the database. The learning system parses the queries to the database and extracts features representing certain concepts. Hierarchies for these concepts are constructed using the learning modules implementing the algorithms; COBWEB and UNIMEM.

It is observed that the learning algorithms do not necessarily give sound suggestions about the attributes of the database based on their learning from the queries The knowledge acquired about each database (the university database in our case) is of utmost importance to support or discard the findings of the learning algorithms. This is due to the fact that the learning modules recognize the syntactical aspects of the attributes appearing in the queries whereas the knowledge about the database captures the semantic aspects of the attributes.

The learning and tuning systems have been implemented using the programming language C and linked to the Oracle database system installed on a SUN machine running UNIX. Integrated with the learning system is the expert systems shell Exsys Professional to make an integrated and intelligent tool that can be used with RDBMSs. As indicated by the rules, certain terms appear to be of a fuzzy nature, e.g., "HIGH", "MEDIUM". We have developed an expert system shell that is capable of dealing with fuzzy terms. This shell will be integrated with our system in the near future to further enhance its performance.

Acknowledgment. I will like to thank Dr. Abad Ali Shah for his valuable comments and discussions, and Mr. Tarik S. Zakzouk for his valuable discussions.

References

[1] Biswas, G. J., Weinberg, Q. Y. and Koller, G. "Conceptual Clustering and Exploratory Data Analysis." *Proceedings of the 8th International Workshop on Machine Learning (Evanston, II, June 1991)*, USA, Calif., (1991), 591-595.

[2] Borgida, A. and Williamson, K. E. "Accommodating Exceptions in Databases, and Refining the Schema by Learning from Them." *Proceedings of Very Large databases*, (1985, Stockholm),72-81.

[3] Brockmann, W. "On-line Machine Learning For Adaptive Control." *IEEE Intl. Workshop on Emerging Technologies and Factory Automation - Technology for Intelligent Factory*, Australia, (1992),190-195.

[4] Dietrich, S. W., M. Brown, E. Rello and Wunderlin, S. "A Practitioner's Introduction to Database Performance Benchmarks and Measurements." *The Computer Journal, 35, No. 4* (1992), 322-331.

[5] Elmasri, R. and Navathe, S.B. "Fundamentals of Database Systems." The Benjamin/Cummings Publishing Company, Inc., (1992).

[6] Fisher, D. H. "Knowledge Acquisition via Incremental Conceptual Clustering." *Machine Learning* 2, No. 2 (1987), 139-172.

[7] Gennari, J. H., Langley, P. and Fisher, D. "Models of Incremental Concept Formation." *Artificial Intelligence*, 40 (1989),61.

[8] Gluck, M. and Corter, J. "Information, Uncertainty, and the Utility Of Categories." *Proceedings of the Seventh Annual Conference of the Cognitive Science Society*, USA, (1985), 283-287.

[9] Graefe, G. "Options in Physical Database Design." *SIGMOD Record*, 22, No. 3 (1993),76-83.

[10] Guynes, C.S. and Pelley, L. "Monitoring Database Performance in an User Environment." *Journal of System Management*, 44, No. 8, 27-30.

[11] Hua, K.A. and Lee, C. "Handling Data Skew in Multiprocessor Database Computers Using Partition Tuning." *Proc. of the Seventeenth Intl. Conf. on Very Large Databases*, USA, (1991),525-535.

[12] Ioannidis, Y. E., T. Saulys, and A. J. Whitsit, "Conceptual Learning in Database Design." *ACM Transactions on Information Systems, 10, No. 3* (July 1992), 265-293.

[13] Lebowitz, M. "Experiments with Incremental Concepts Formation: UNIMEM." *Machine Learning*, 2, No. 2 (1987), 103-138.

[14] Lebowitz, M. "Concept Learning in a Rich Input Domain: Generalization-Based Memory." In: *Machine Learning: An Artificial Intelligence Approach*, Vol. II, J. G. Carbonell, R.S. Michalski and T.M. Mitchell, (Eds.), Morgan Kaufmann, Los Altos, Calif., (1987), 193-214.

[15] Mathkour H. "Expert System Development Tools: A Case Study." *First International Conference on AI Applications* , Cairo, Egypt, 1992.

[16] McFadden, F. and Hoffer, J. "Database Management." 2nd ed., Benjamin/ Cumming Publishing Company, 1992.

[17] Quinlan, J. R. "Induction of Decision Trees." *Machine Learning*, 1, No. 1 (1986), 81-106.

[18] Ricardo, C. *Database System Principles, Design and Implementation*. Maxwell Macmillan International Editions, 1990.

[19] Roddick, J.F. "Dynamically Changing Schemas within Database Models." *The Australian Computer Journal*, 23, No. 3 (August 1991), 105-109.

[20] Stylianou, A. *et al*, "Selection Criteria for Expert System Shells: A Socio Techincal Framework.", *Communication of ACM*, 35, No. 10 (1992).

[21] Weikum, G. *et al*, "The COMFORT Project." *Proc. of the Second Intl Conf. on Parallel and Distributed Information Systems*, USA, (1993), 158-161.

[22] Zakzouk,T.S., Mathkour, H. and Shah, A. "Design of the Plan Manager For the STDT." *Proceedings of the IASTED International Conference on Modeling and Simulation*, Sri Lanka (July 26-28, 1995), 241-244.

[23] Zakzouk,T.S., Mathkour, H. and Shah, A. "STDT: A Self Tuning Tool For Database Systems." *Proceedings of the Third International Conference on Artificial Intelligent Applications*, Cairo, Egypt, January 4 -7(1995).

[24] Zakzouk,T.S., Mathkour, H. and Shah, A. "A Self Tuning Database Tool." *Proceedings of the VIII International Symposium in Informatics Application (INFONOR'94)*, Antofagasta, Chile, November 21-25 (1994), 79-86.

[25] Zakzouk,T. S., Mathkour, H. and Shah, A. "Knowledge Engineering Issues in the Design of the Conceptual Hierarchy Manager of STDT." *the Journal of Mathematical Modeling and Scientific Computing (special issue)*, 6 (1996).

[26] Zhou, G. and Birdwell, J.D. "PID Autotuner Design Using Machine Learning." *IEEE Symposium on Computer-Aided Control System Design (CACSD)*, USA, (1992), 173-179.

Appendix

The information collected through the execution of the queries include data about:

count: number of times OCI procedure was executed
cpu: cpu time in seconds executing
elapsed: elapsed time in seconds executing
disk: number of physical reads of buffers from disk
query: number of buffers gotten for consistent read
current: number of buffers gotten in current mode (usually for update)
rows: number of rows processed by the fetch or execute call
parse: this step will translates the SQL statement into an execution plan. It includes checks for proper security authorization and checks for the existence of tables, columns, and other referenced objects.
execute: this step is the actual execution of the statement by ORACLE. For INSERT, UPDATE, and DELETE statement, this modifies the data. For SELECT statement, the step identifies the selected rows.
fetch: this step retrieves the rows the satisfy a query. ORACLE only performs this step for SELECT statement.

Actual statistics obtained from the query
SELECT *
FROM STUDENT, DEPARTMENT
WHERE STUDENT.D-CODE = DEPARTMENT.D-CODE
is given below.

Call	Count	Cpu	Elapsed	Disk	Query	Current	Rows
Parse	1	0.43	0.58	6	35	4	0
Execute	1	0.00	0.00	0	0	0	0
Fetch	1	0.03	0.08	2	2	4	14

Missing in library cache during parse: 1
Parsing user id : 16

Explain plan

Rows	Execution plan
14	Merge join
4	Sort join
4	Table access (full) of 'department'
14	Sort join
14	Table access (full) of 'student'

نظام ذكي لتحسين أداء نظم قواعد المعلومات

حسن مذكور

قسم علوم الحاسب، كلية علوم الحاسب والمعلومات، جامعة الملك سعود،

ص ب ٥١١٧٨ ، الرياض ١١٥٤٣، المملكة العربية السعودية

(قدّم للنشر في ١٩٩٦/٦/٨م؛ وقبل للنشر في ١٩٩٦/١٠/١م)

ملخص البحث. يناقش هذا البحث تصميم وتطوير نظام ذكي متكامل لمراقبة قواعد المعلومات وضبطها ويضم النظام أداة تعلّم وبرنامج خبرة متّصلين بنظام إدارة قواعد المعلومات الترابطية. ويستخدم النظام قواعد معرفة من واقع مجالات تطبيق قواعد المعلومات ومن المعرفة والخبرة التي تمّ جمعهما عن نظم قواعد المعلومات. وقد تم تطوير النظام بلغة البرمجة C ونظام الخبرة إكسس. وقد توصلنا في هذا البحث إلى أن تراكيب عناصر قواعد المعلومات يجب أن تدرس من الناحية الجوهرية حتى يمكن التوصّل إلى استنتاجات عملية. ولقد وجدنا أن قواعد الخبرة لها فائدة كبيرة في هذا المجال.