US 20050119999A1

(19) **United States**

(12) **Patent Application Publication** (10) **Pub. No.: US 2005/0119999 A1**

Zait et al. (43) **Pub. Date: Jun. 2, 2005**

(54) **AUTOMATIC LEARNING OPTIMIZER**

(75) Inventors: **Mohamed Zait**, San Jose, CA (US); **Benoit Dageville**, Foster City, CA (US); **Dinesh Das**, Redwood City, CA (US); **Khaled Yagoub**, San Mateo, CA (US); **Mohamed Ziauddin**, Pleasanton, CA (US)

Correspondence Address:
BINGHAM, MCCUTCHEN LLP
THREE EMBARCADERO CENTER
18 FLOOR
SAN FRANCISCO, CA 94111-4067 (US)

(73) Assignee: **ORACLE INTERNATIONAL CORPORATION**, REDWOOD SHORES, CA

(21) Appl. No.: **10/935,906**

(22) Filed: **Sep. 7, 2004**

Related U.S. Application Data

(60) Provisional application No. 60/500,490, filed on Sep. 6, 2003.

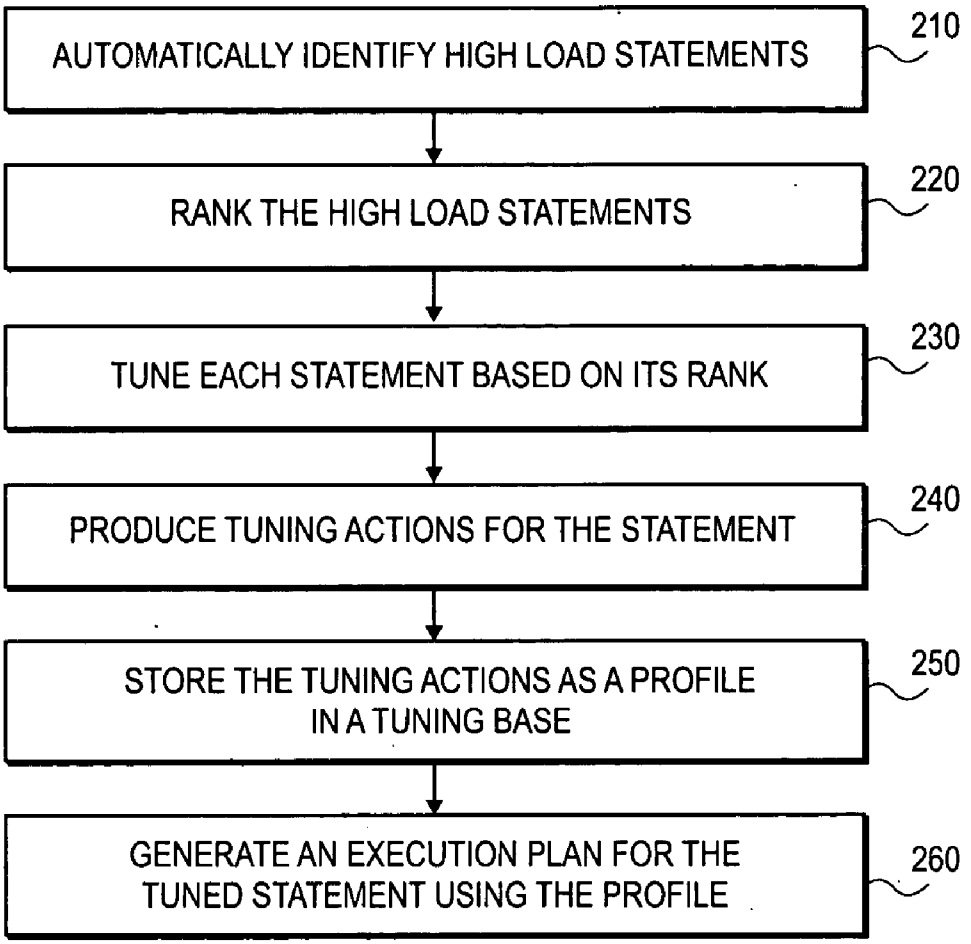
Publication Classification

(51) **Int. Cl.**[7] **G06F 17/00**

(52) **U.S. Cl.** **707/3**

(57) **ABSTRACT**

A method of gathering performance information about a workload, and automatically identifying a set of high-load database query language statements from the workload based on the performance information, is disclosed.



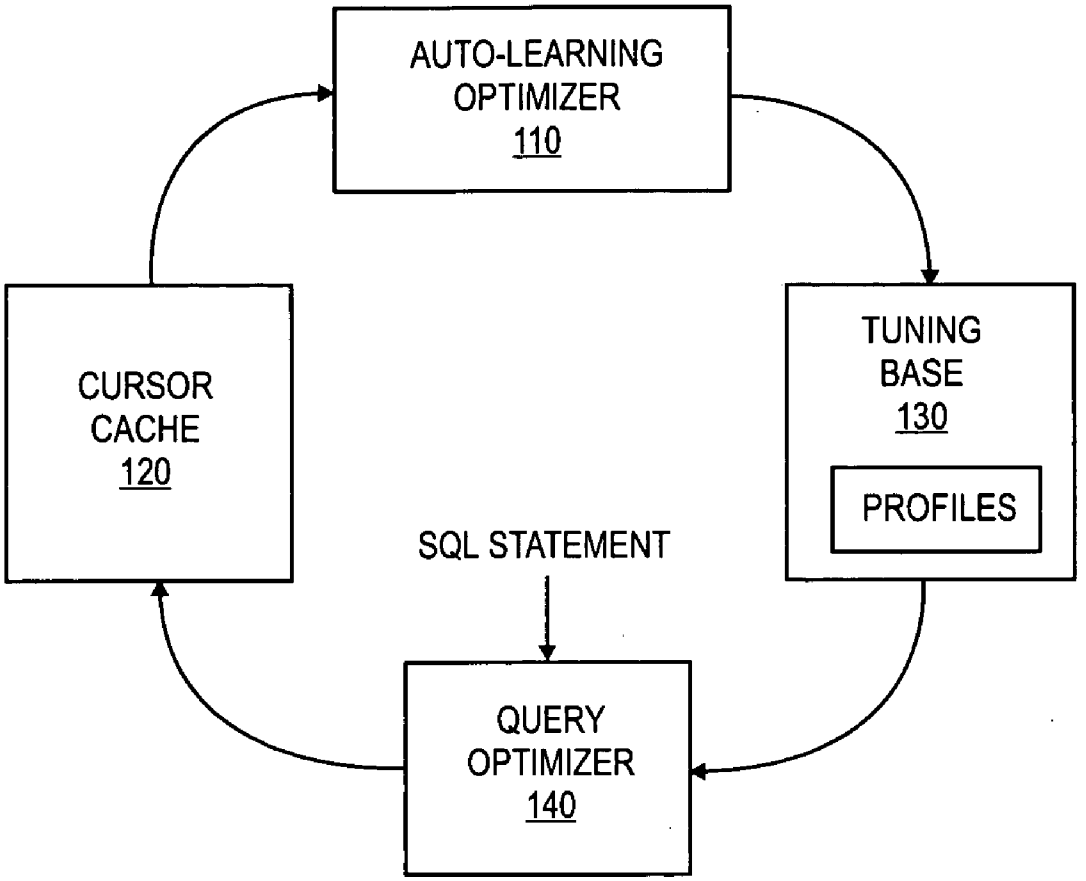

AUTO-LEARNING
OPTIMIZER
110
CURSOR
CACHE
120
TUNING
BASE
130
PROFILES
SQL STATEMENT
QUERY
OPTIMIZER
140

FIG. 1

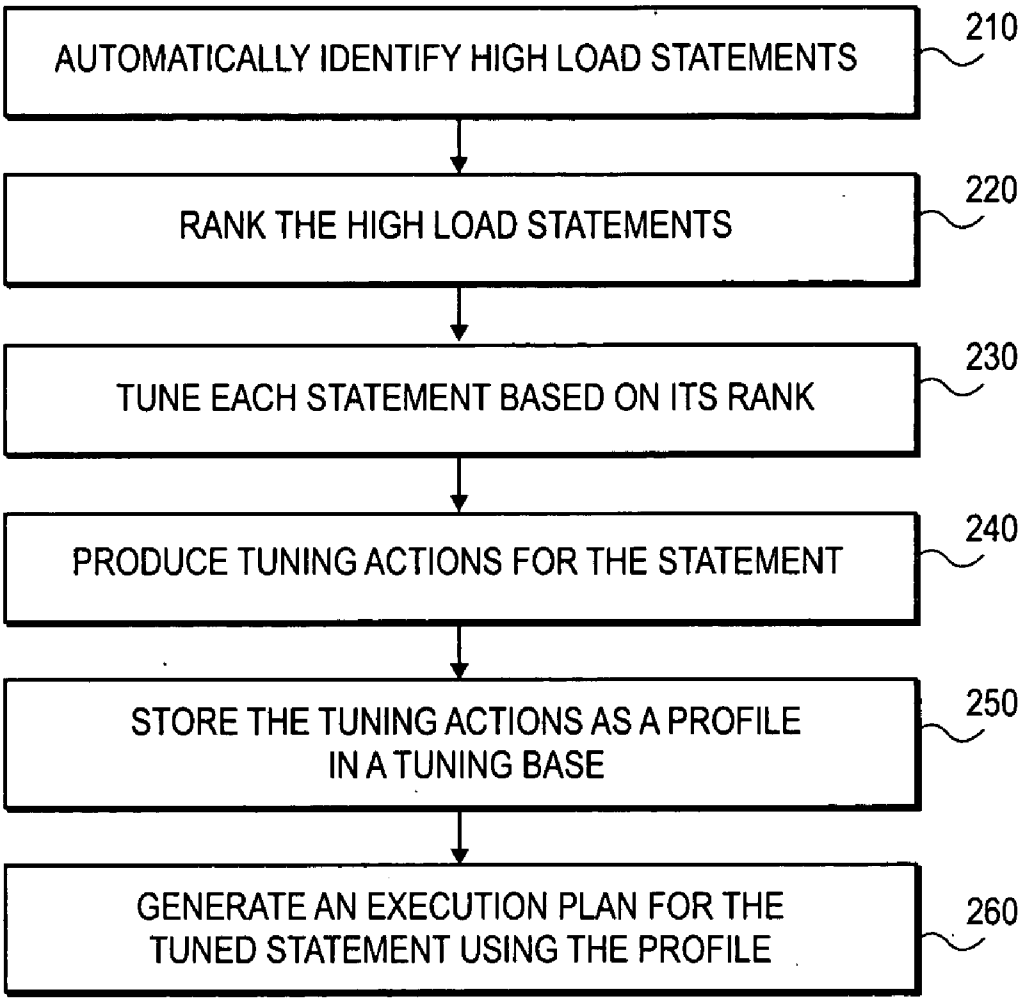

AUTOMATICALLY IDENTIFY HIGH LOAD STATEMENTS
210
RANK THE HIGH LOAD STATEMENTS
220
TUNE EACH STATEMENT BASED ON ITS RANK
230
PRODUCE TUNING ACTIONS FOR THE STATEMENT
240
STORE THE TUNING ACTIONS AS A PROFILE IN A TUNING BASE
250
GENERATE AN EXECUTION PLAN FOR THE TUNED STATEMENT USING THE PROFILE
260

FIG. 2

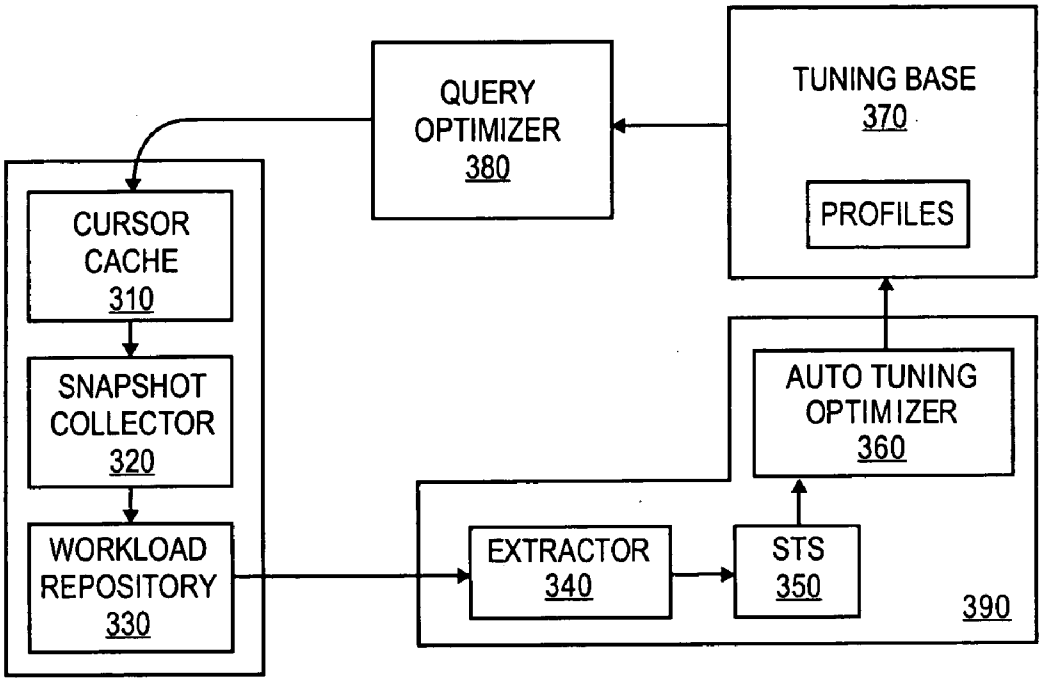

QUERY
OPTIMIZER
380
TUNING BASE
370
PROFILES
CURSOR
CACHE
310
SNAPSHOT
COLLECTOR
320
WORKLOAD
REPOSITORY
330
AUTO TUNING
OPTIMIZER
360
EXTRACTOR
340
STS
350
390

FIG. 3

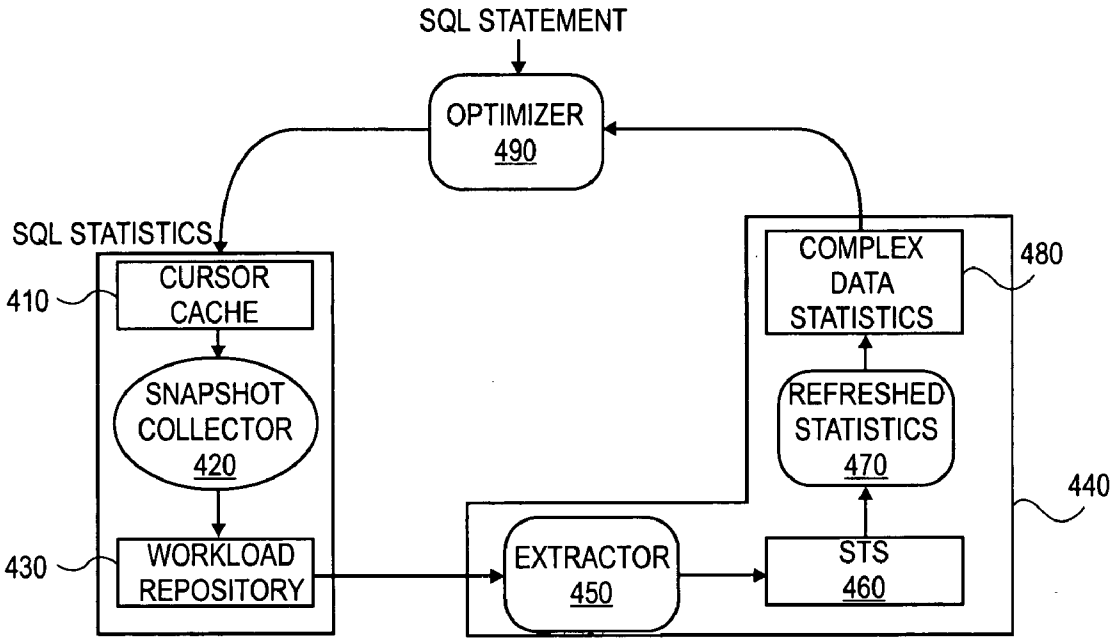

SQL STATEMENT
OPTIMIZER
490
SQL STATISTICS
410
CURSOR
CACHE
SNAPSHOT
COLLECTOR
420
430
WORKLOAD
REPOSITORY
EXTRACTOR
450
COMPLEX
DATA
STATISTICS
480
REFRESHED
STATISTICS
470
STS
460
440

FIG. 4

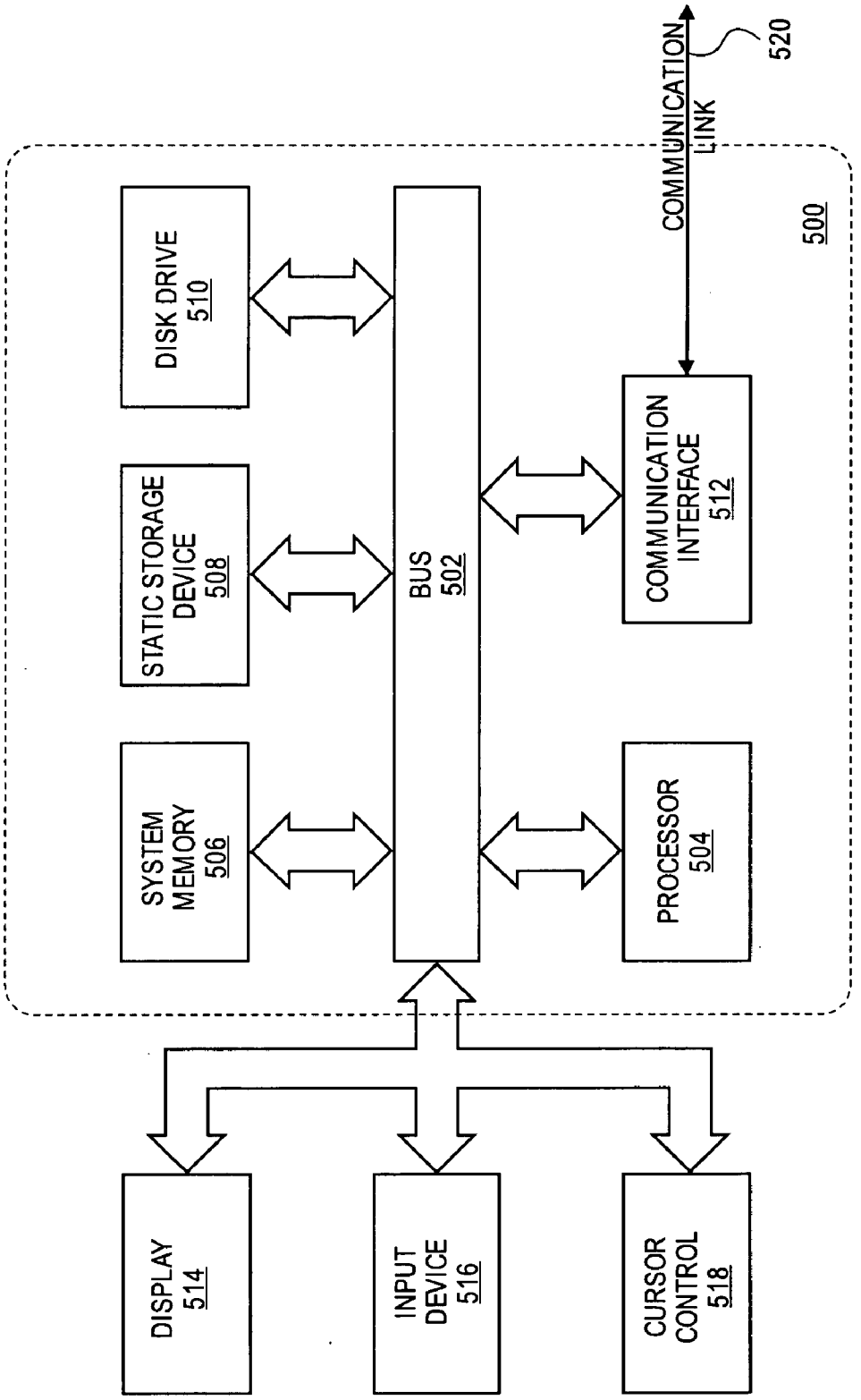

COMMUNICATION LINK
520
500
DISK DRIVE 510
STATIC STORAGE DEVICE 508
SYSTEM MEMORY 506
BUS 502
COMMUNICATION INTERFACE 512
PROCESSOR 504
DISPLAY 514
INPUT DEVICE 516
CURSOR CONTROL 518

FIG. 5

AUTOMATIC LEARNING OPTIMIZER

CROSS-REFERENCE TO RELATED APPLICATIONS

[0001] This application claims the benefit of U.S. Provisional Application No. 60/500,490, filed Sep. 6, 2003, which is incorporated herein by reference in its entirety. This application is related to co-pending applications "SQL TUNING SETS," Ser. No. _____ Attorney Docket No. OI7036272001; "AUTO-TUNING SQL STATEMENTS," Ser. No. _____ Attorney Docket No. OI7037042001; "SQL PROFILE," Ser. No. _____ Attorney Docket No. OI7037052001; "GLOBAL HINTS," Ser. No. _____ Attorney Docket No. OI7037062001; "SQL TUNING BASE," Ser. No. _____ Attorney Docket No. OI7037072001; "AUTOMATIC PREVENTION OF RUN-AWAY QUERY EXECUTION," Ser. No. _____ Attorney Docket No. OI7037092001; "METHOD FOR INDEX TUNING OF A SQL STATEMENT, AND INDEX MERGING FOR A MULTI-STATEMENT SQL WORKLOAD, USING A COST-BASED RELATIONAL QUERY OPTIMIZER," Ser. No. _____ Attorney Docket No. OI7037102001; "SQL STRUCTURE ANALYZER," Ser. No. _____ Attorney Docket No. OI7037112001; "HIGH-LOAD SQL DRIVEN STATISTICS COLLECTION," Ser. No. _____ Attorney Docket No. OI7037122001; "AUTOMATIC SQL TUNING ADVISOR," Ser. No. _____ Attorney Docket No. OI7037132001, all of which are filed Sep. 7, 2004 and are incorporated herein by reference in their entirety.

FIELD OF THE INVENTION

[0002] This invention is related to the field of electronic database management systems.

BACKGROUND

[0003] SQL tuning is a critical aspect of database performance tuning. It is an inherently complex activity requiring a high level of expertise in several domains: query optimization, to improve the execution plan selected by the query optimizer; access design, to identify missing access structures; and SQL design, to restructure and simplify the text of a badly written SQL statement. Furthermore, SQL tuning is a time consuming task due to the large volume and evolving nature of the SQL workload and its underlying data.

[0004] Over the past decade two clear trends have occurred: (a) the database systems have been deployed in new areas, such as electronic commerce, bringing a new set of database requirements, and, (b) the database applications have become increasingly complex with support for very large numbers of concurrent users. As a result, the performance of database systems has become highly visible and thus critical to the success of the businesses running these applications. For example, database systems continue to be deployed in new areas, such as electronic commerce, and the database applications have increasingly become sophisticated to support more users and provide more functionalities, making the query optimization task more complex.

[0005] The database system vendors deal with this increased complexity by enhancing the optimizer capabilities to deal with new SQL constructs, add better searching techniques, or a richer cost model. While this conventional approach can solve some problems, it is not capable to deal with the dynamic nature of the database application, e.g., dynamic changes in the application workload. Indeed the conventional optimizer will always face situations where mistakes are unavoidable. For example, the optimizer can lack of information about the objects accessed by a SQL statement. The optimizer logic may also not be prepared to deal with certain kinds of problems.

SUMMARY

[0006] An automatic learning optimizer is able to automatically tune a database query language statement by automatically identifying high load or top database query language statements that are responsible for a large share of the application workload and system resources based on the past database query language statement execution history available in the system, automatically generating ways to improve execution plans produced by a compiler for these statements, and automatically performing corrective actions to generate better execution plans for poorly performing statements.

BRIEF DESCRIPTION OF THE DRAWINGS

[0007] **FIG. 1** shows an example of the Automatic Learning Optimizer architecture.

[0008] **FIG. 2** shows an example of the process flow of the auto-learning process.

[0009] **FIG. 3** represents another exemplary illustration of the automatic learning optimizer device.

[0010] **FIG. 4** shows another example of a system to perform the auto-learning process for database system management.

[0011] **FIG. 5** is a block diagram of a computer system suitable for implementing an embodiment of coverage computation for verification.

DETAILED DESCRIPTION

[0012] Overview

[0013] The embodiments of the invention are described using the term "SQL", however, the invention is not limited to just this exact database query language, and indeed may be used in conjunction with other database query languages and constructs.

[0014] An automatic learning optimizer has learning capabilities that make it able to learn from past execution history of SQL statements which can be repeated in the future. For example, a database application is often repetitive, i.e., the same SQL statements are submitted over and over, some more frequently than others. Information about these statements can be collected from various sources, e.g., execution statistics for some or all operations of the query execution plan (number of output rows, amount of memory, number of disk reads), or the caching ratio of the database objects.

[0015] An auto-learning capability for the auto-tuning optimizer provides a component of a fully self-managed database system. The auto-learning optimizer can execute a background task which identifies potential optimizer mistakes made for a target SQL statement, and automatically produce optimizer corrective actions. Hence, the auto-learn-

ing optimizer can gradually repair the suboptimal executions plans run by the database system.

[0016] The auto-learning process starts by identifying a small subset of SQL statements which are potential candidates for auto-learning. For example, this subset may correspond to all SQL statements which are known to have a suboptimal execution plan and a high impact on the overall performance of the system. Alternatively, the auto-learning detection mechanism may focus on the subset of high-load SQL statements. Once the set of high-load SQL statements has been identified, the auto-learning optimizer can uncover its mistakes by analyzing each SQL statement in the set. Based on this analysis, if optimizer related problems are then found, corrective actions are produced and stored in a computer readable medium, such as a disk. Because the corrective actions are permanently stored, the auto-learning optimizer can perform an iterative learning process which accumulates, over time, more and more knowledge on problematic queries, and also prevents the corrected problems from recurring.

[0017] In one embodiment, the auto-learning optimizer performs the auto-tuning function to generate the corrective actions and store them in SQL Profiles. Hence, corrective actions are automatically placed in one or more profiles, which are stored in the tuning base. The auto-learning process may be on-line, with the auto-learning process running almost continuously as a background task. In this mode, high-load SQL statements stored in the cursor cache are targeted. Hence, the on-line mode can address critical SQL tuning issues while having a very low overhead on the system performance.

[0018] In another embodiment, the auto-learning optimizer performs learning off-line. In this mode, the auto-learning process is executed during the maintenance window as an automated manageability task. This off-line mode can have more time and system resources to perform the auto-tuning functions. Hence, the auto-learning optimizer can have time to refresh corrective actions produced in the past, and auto tune less critical SQL statements.

[0019] **FIG. 1** shows an example of the auto-learning optimizer feedback loop process for the on-line auto-learning optimizer. The on-line auto-learning background process is performed by the auto-learning optimizer **110**, which identifies the set of high-load SQL statements from the cursor cache **120**. This can be achieved by keeping, for each cursor in the cache, a record of the total elapsed time accumulated since the last auto-learning cycle. When an auto-learning cycle starts, cursors in the cache can be ranked based on this cumulative elapsed time metric. The auto tune process is then applied to each cursor in their ranking order. For example, in one embodiment, cursors which account for more than 1% of the total cumulated elapsed time are considered by the auto-learning optimizer **110**.

[0020] The auto-learning optimizer **110** can also auto tune recurring cursors with a long average elapsed time, even when their cumulated elapsed time is not necessarily high, because these cursors could negatively affect the response time of individual end users even if their overall impact on the system performance is limited. A ranking procedure can be used for these cursors by ranking them based on the average elapsed time (instead of the cumulative elapsed time).

[0021] The auto-tuning process may skip cursors which have not been executed more than once since they have been first loaded in the cursor cache, in order to prevent the auto-learning process from spending time on non recurring cursors (e.g. ad-hoc queries). Also, cursors with hintsets or outlines already defined on them may be skipped, since these cursors have been tuned already.

[0022] Once the set of cursors to auto tune is identified, the auto-learning optimizer starts the learning process. This process can run continuously in the background, up to the beginning of the next auto-learning cycle, which reduces the overall impact on the system. This background task may be accomplished by pacing each auto tune execution based on the total load of the system. Profiles produced for each auto-tuned statement may be stored in the Tuning base, even if no corrective actions have been produced. In that case, a VOID profile may be generated to keep track of statements which have been auto tuned.

[0023] Once a SQL statement is auto tuned, the learned corrective actions (i.e. a profile) are permanently stored in the Tuning base **130**. The next execution of that statement will automatically trigger a hard parse, except when the associated hintset is 'VOID'. When the statement is recompiled, the query optimizer **140** takes into account the extra knowledge carried by the profile to generate an improved execution plan.

[0024] The execution plan produced by optimizer **130** is sent to the cursor cache **120**, which closes the auto-learning feedback loop. The statements that have been improved by the auto-learning optimizer then have execution plans that are no longer high-load. They drop out of the set of high-load SQL identified by the auto-learning optimizer **110**, because they are no longer high-load, or because a tuning profile now exists for them. As the automatic learning process continues, the formerly lesser high-load statements will grow in rank to the point that they now become targets for auto-learning optimization process. With a stable SQL workload, the system can rapidly converge with all high-load statements auto-tuned.

[0025] An example of a method of performing the on-line auto-learning process is shown in **FIG. 2**. High load SQL statements are identified, **210**. The high load statements are ranked based on a performance metric, such as execution time, **220**. Each high load statement is tuned by the automatic tuning optimizer based on its rank, **230**. A profile of tuning actions is created for the high load statement, **240**, which is stored in a tuning base, **250**. A query optimizer generates an execution plan for the statement using the profile from the tuning base, **260**.

[0026] The off-line auto-learning process can be performed by the auto-learning optimizer system as shown in **FIG. 3**. Workload information is automatically captured at regular interval, by default once every 30 minutes, from the cursor cache **310** by snapshot collector **320**, and placed in the workload repository **330**. The off-line functions of the auto-learning process can be executed as an automated manageability task, scheduled to be performed within the maintenance window. For example, a database administrator (DBA) can define that automatic manageability actions (e.g. index rebuild, space coalesce, auto analyze, auto learn) are to be executed at night from 10 pm to 6 am.

[0027] The snapshot collection of the cursor cache can be performed by collector **310** at regular intervals, such as

every 30 minutes, to snapshot performance statistics, and save them in the workload repository **330**, to capture information on high-load SQL. For example, the information on SQL that can be saved in the repository can include data to auto-tune the statement at a later time.

[**0028**] The high load SQL statements can be tuned within the maintenance window by the auto-learning optimizer **390**. The high load extractor **340** identifies high load statements and retrieves the information on the high-load SQL statements captured in the workload repository **330** since the last off-line auto-learning session. A SQL Tuning Set (STS) **350** is created to store the set of high-load SQL statements. These statements can be ranked using their cumulative elapsed time, and/or high average elapse time. Each statement in the STS is auto-tuned in ranked order by the auto-tuning optimizer **360**. If a statement has been recently auto-tuned, it can be skipped. Once a SQL statement is auto tuned, the learned corrective actions (i.e. a profile) are permanently stored in the Tuning base **370**, and retrieved by the query optimizer **380** to generate a well tuned execution plan for the statement.

[**0029**] The auto-learning optimizer can optimize and execute an ad-hoc SQL statement the first time it comes into the system. The auto-learning optimizer can do this by learning about important properties of SQL statements such that the information learned from one statement can be used for another statement. For example, by analyzing the set of high-load SQL statements, the auto-learning optimizer can detect data skew between two columns and use this information to generate appropriate statistics such as either a multi-column histogram or a multi-dimensional histogram. Because the information learned from the ad-hoc SQL can become excessively large, making it impractical to collect the corresponding statistics, the past SQL execution history is used to identify and weed out many of the SQL properties that offer either one-time or insignificant performance gains.

[**0030**] **FIG. 4** shows an example of the auto-learning optimizer system with a feedback loop for automatically learning tuning information on an ad hoc SQL statement basis. The cursor cache **410** is examined by the snapshot collector **420** to gather information that is stored in the workload repository **430**. Using SQL information collected in the workload repository, the extractor **450** identifies high load statements and places them in a STS **460**. The auto-learning optimizer **440** determines the appropriate set of complex statistics (like multi-column and multi-table histograms, predicate statistics, . . .) to collect and refresh. The refreshed statistics **470** are added to the complex data statistics **480**, which are then used by the query optimizer **490** to generate an execution plan for a SQL statement. The ad-hoc auto-learning process is therefore able to increase the number of statements in the SQL workload whose execution plans would benefit from these extra statistics while limiting the cost (time to collect and time to refresh) for these extra statistics.

[**0031**] According to one embodiment of the invention, computer system **500** performs specific operations by processor **504** executing one or more sequences of one or more instructions contained in system memory **506**. Such instructions may be read into system memory **506** from another computer readable medium, such as static storage device **508** or disk drive **510**. In alternative embodiments, hard-wired circuitry may be used in place of or in combination with software instructions to implement the invention.

[**0032**] The term "computer readable medium" as used herein refers to any medium that participates in providing instructions to processor **504** for execution. Such a medium may take many forms, including but not limited to, non-volatile media, volatile media, and transmission media. Non-volatile media includes, for example, optical or magnetic disks, such as disk drive **510**. Volatile media includes dynamic memory, such as system memory **506**. Transmission media includes coaxial cables, copper wire, and fiber optics, including wires that comprise bus **502**. Transmission media can also take the form of acoustic or light waves, such as those generated during radio wave and infrared data communications.

[**0033**] Common forms of computer readable media includes, for example, floppy disk, flexible disk, hard disk, magnetic tape, any other magnetic medium, CD-ROM, any other optical medium, punch cards, paper tape, any other physical medium with patterns of holes, RAM, PROM, EPROM, FLASH-EPROM, any other memory chip or cartridge, carrier wave, or any other medium from which a computer can read.

[**0034**] In an embodiment of the invention, execution of the sequences of instructions to practice the invention is performed by a single computer system **500**. According to other embodiments of the invention, two or more computer systems **500** coupled by communication link **520** (e.g., LAN, PTSN, or wireless network) may perform the sequence of instructions to practice the invention in coordination with one another. Computer system **500** may transmit and receive messages, data, and instructions, including program, i.e., application code, through communication link **520** and communication interface **512**. Received program code may be executed by processor **504** as it is received, and/or stored in disk drive **510**, or other non-volatile storage for later execution.

[**0035**] In the foregoing specification, the invention has been described with reference to specific embodiments thereof. It will, however, be evident that various modifications and changes may be made thereto without departing from the broader spirit and scope of the invention. The specification and drawings are, accordingly, to be regarded in an illustrative rather than restrictive sense.

We claim:

1. A method comprising:

gathering performance information about a workload; and

automatically identifying a set of high-load database query language statements from the workload based on the performance information.

2. The method of claim 1, further comprising:

automatically generating tuning actions for each high-load statement.

3. The method of claim 2, further comprising:

automatically storing the tuning actions for each high-load statement in a profile.

4. The method of claim 3, further comprising:

persistently storing each profile in a tuning base.

5. The method of claim 4, further comprising:

receiving one of the high-load statements at a compiler;

retrieving the profile for the received statement from the tuning base; and

generating an execution plan for the statement with the profile.

6. The method of claim 1, wherein the database query language statement is a SQL statement.

7. A method comprising:

gathering performance information about a workload; and

automatically identifying a set of high-load database query language statements from the workload based on the performance information.

8. The method of claim 7, further comprising:

automatically generating tuning actions for each high-load SQL statement.

9. The method of claim 8, further comprising:

automatically storing the tuning actions for each high-load statement in a profile.

10. The method of claim 9, further comprising:

persistently storing each profile in a tuning base.

11. The method of claim 10, further comprising:

receiving one of the high-load statements at a compiler;

retrieving the profile for the received statement from the tuning base; and

generating an execution plan for the statement with the profile.

12. The method of claim 7, wherein the database query language statement is a SQL statement.

13. A method comprising:

gathering performance information about a workload; and

automatically identifying a set of high-load database query language statements from the workload based on the performance information.

14. The method of claim 13, further comprising:

automatically generating tuning actions for each high-load statement.

15. The method of claim 14, further comprising:

automatically storing the tuning actions for each high-load statement in a profile.

16. The method of claim 15, further comprising:

persistently storing each profile in a tuning base.

17. The method of claim 16, further comprising:

receiving one of the high-load statements at a compiler;

retrieving the profile for the received statement from the tuning base; and

generating an execution plan for the statement with the profile.

18. The method of claim 13, wherein the database query language statement is a SQL statement.

* * * * *