Context-sensitive Indexes in RDBMS for Performance Optimization of SQL Queries in Multi-tenant/Multi-application Environments

Arjun K. Sirohi[1] and Vidushi Sharma[2]
[1]*PSR (Performance, Scalability, Reliability) Engineering, Oracle U.S.A. Inc., Bellevue, WA, U.S.A.*
[2]*School of Information and Communication Technology, Gautam Buddha University, Greater Noida, India*

Keywords: Indexes, Mixed-load Queries, Multi-tenant Queries, Multi-application Queries, SQL Performance Optimization.

Abstract: With the recent shift towards cloud-based applications and Software as a Service (SaaS) environments, relational databases support multi-tenant and multi-application workloads that query the same set of data stored in common tables, using SQL queries. These SQL queries have very different query constructs and data-access requirements leading to different optimization needs. However, the business-users' expect sub-second response times in getting the data that they requested. The current RDBMS architectures where indexes "belong" to a table without any object privileges of their own, and, therefore, must be considered and used by the optimizer for all SQLs referencing the table(s), pose multiple challenges for the optimizer as well as application architects and performance tuning experts, especially as the number of such indexes grows. In this paper, we make the case for "Context-Sensitive Indexes", whereby applications and tenants could define their own indexes on the shared, transactional database tables to optimize the execution of their SQL queries, while at the same time having the optimizer keep such indexes isolated from other applications and tenants/users for the purposes of query optimization.

1 INTRODUCTION

Today, there is an increasing need for relational database management systems (RDBMS) to support mixed-load and multi-tenant queries. Additionally, the changing landscape of enterprise business applications, coupled with the easy availability of cloud services and software-as-a-service paradigm, many small and medium sized businesses who could not afford to implement applications like Customer Relationship management (CRM) in-house, have now started adopting these applications through cloud providers as hosted application services. All this is leading to a situation where the back-end databases need to support multiple concurrent applications that send SQL queries against the same schema and data, albeit with very different query constructs, data access requirements and optimization goals. The providers of SaaS and Cloud environments also face the challenge to support multi-tenancy, wherein the applications' schema in the database may be shared and yet the data stored in those schema tables is isolated and protected from each other. What is not changing, though, is the business-users' expectations of sub-second response times in getting the data that they requested of the database. Cloud and SaaS providers are finding themselves implementing various solutions that can act as a differentiator.

Most commercial enterprise business applications use a relational database like Oracle at the back end and all processing of data is managed through SQL queries. At the lowest grain, the performance of such SQL queries often dictates the performance and scalability of the application, among other factors. In this paper we examine the existing application and database approach towards use of indexes and their impacts on the performance of SQL queries. A large number of performance problems arise from the fact that most SQL queries are dynamically generated at run-time by middleware SQL-generation engines based on an abstract layer of the logical data model, which results in complex queries. It has been lamented that the use of indexes in the database to help performance of SQL queries is as old as relational databases and yet we find inadequacy and poor quality of indexes (Leach and Lahdenmaki, 2005). So we feel that it is time to take a fresh look at the humble indexes in the context of multi-tenant and

K. Sirohi A. and Sharma V..
Context-sensitive Indexes in RDBMS for Performance Optimization of SQL Queries in Multi-tenant/Multi-application Environments.
DOI: 10.5220/0005350802590270
In *Proceedings of the 17th International Conference on Enterprise Information Systems* (ICEIS-2015), pages 259-270
ISBN: 978-989-758-096-3
Copyright © 2015 SCITEPRESS (Science and Technology Publications, Lda.)

multi-application environments to see how indexes can be better leveraged to improve query performance.

This paper begins by providing the motivation and background work done on this topic. Next, we discuss the contributions of our paper followed by a discussion of existing architectures and their limitations as related to the topic of our research. This includes highlighting the two most important limitations in existing architectures. Next, we discuss our proposal of context-sensitive indexes and their implementation aspects, including maintenance. The paper then presents results from experiments conducted to support the proposal. The paper concludes with ideas for taking action and possible directions for implementation.

2 MOTIVATION AND BACKGROUND WORK

Modern business applications have to process increasing volumes of data stored in transactional databases. This means that SQL queries have to process more data but still achieve acceptable query processing times. This has added increased pressure on RDBMS providers to come up with new techniques to handle increased volumes of data on one hand and to keep query processing times within acceptable limits, if not improve further, on the other. Adding to this complexity are the Cloud and SaaS offerings from independent service and application providers. To support such application models, database vendors have tried to make changes and adjustments to the database architectures. For example, the offerings and multi-tenant scenarios from IBM, like separate databases for tenants, shared database but different schemas or shared database and shared schema for multiple tenants have been discussed (Chong, 2012). There are similar offerings made by Microsoft Corporation (Chong, Carraro and Wolter, 2006). Oracle's latest database release 12c also provides many enhancements to support multi-tenancy (Oracle, 2014). All such offerings and architectures are focused around the use and sharing of database instances, databases, schemas and applications. Most research has been focused on optimizing such architectures in order to achieve user satisfaction in terms of query processing times.

Improving query processing time is an ongoing, complex research subject and advancements have been made in recent years by RDBMS vendors in many different areas of query optimization. One such area of advancement is the use of various types of indexes used in query optimization. We find that most of the existing work done by researchers on indexes has been mostly on the aspects of internal implementation of various types of indexing mechanisms. For example, the issues related to optimizing of multidimensional index trees for main memory access were researched and addressed by using two-dimensional CR-tree index structure that performs searches much faster than the ordinary R-tree and at the same time consuming less memory space (Kihong and Sang, 2001). These researchers have focused on the physical design structures of indexes and better algorithms to optimize storage and access. The impacts of multiple and different types of applications like OLTP applications and transactional business intelligence applications sending a mixed and varied query load to the database, on the performance and scalability of queries has not been the focus of much research. The origins of mixed query workloads like OLTP and analytics workloads and their characteristics have been researched and discussed earlier (Powley, Martin and Bird, 2008). However, much of the research has been directed at finding ways and means to somehow shield the transactional business applications' queries from the effects of analytical and business intelligence type of queries. For example, in one solution, it was proposed to adjust the memory allocation in order to meet response time goals for mixed workloads (Brown, Mehta, Carey, and Livny, 1994). Another solution proposed was an algorithm for memory allocation and prioritization based on resource usage, workload characteristics and performance statistics (Pang, Carey and Livny, 1995). Other researchers put their focus on creating benchmark standards that could effectively bench such mixed-load applications (Krueger, Tinnefeld, Grund, Zeier, and Plattner, 2010). To the best of our knowledge, there is no current published research on query optimization through creation and use of context-sensitive indexes, whereby tenants/users/schemas and applications could decide which indexes should be used as input by the optimizer to arrive at the best access paths for their SQL queries instead of the optimizer being burdened with this task of evaluating all available indexes on a given table.

3 CONTRIBUTIONS OF THIS PAPER

In this paper, we present a novel way to optimize query processing for the shared environments where a single shared-database, shared-schema approach provides multi-tenancy or multi-applications environments or both. In these approaches, multiple instances of an application and/or multiple different applications point to the shared database and shared schema tables that hold the applications' data. The tenants are identified by a unique column in every table that stores the application data, most commonly named "Tenant_Id". While this shared approach provides considerable cost savings in terms of infrastructure and operational costs, there are performance and scalability considerations and concerns arising out of this architecture. Similarly, even in single-tenant database environments, there are often multiple applications and modules within applications with different optimization goals and SQL query constructs that are a cause for performance concerns.

In this paper, we propose the creation and handling of indexes in a manner that can lead to better optimized query processing, thereby resulting in better performance and scalability of such multi-tenant and/or multi-applications environments. We focus on improving the performance of SQL queries in a mixed-load and/or multi-tenant enterprise business application through the creation and use of context-sensitive indexes, owned by an application and/or tenant/user, shared with other applications/tenants/users if needed, and used by the RDBMS optimizer solely for SQL queries originating from these owner applications' modules and actions and/or tenants/users.

For the purposes of this paper, we refer to the Oracle database's implementation of schema and indexes in order to compare with our proposed solution since Oracle is the most widely used commercial database. Also, in this paper, we use the term "context-sensitive indexes" to describe indexes that are only visible to and/or owned by specific applications, schemas and/or tenants and used by the RDBMS optimizer ONLY to optimize access paths for queries originated by the specific tenant/user/schema and/or applications' modules and sub-modules termed as Actions. These have the potential to significantly improve the performance of the SQL queries in a shared, multi-tenant and/or multi-application architecture and as a result improve the performance and scalability of such applications.

A word on the limits of the scope of our research and this paper's proposal is in order here. We want to clarify that we do not delve into the physical implementation characteristics and optimizations of table and index structures. We also do not propose any new type of index structures. What we have researched and what we propose is a methodology for the segregation of ownership of the tables that contain application data from the creation and use of context-sensitive indexes that support the search and retrieval of data from such tables by multiple tenants/users/schemas and/or applications' Modules and Actions.

4 EXISTING ARCHITECTURES AND LIMITATIONS

In this section, we provide a snapshot of the existing architectures and their limitations as related to the topic of our research. For example, in Oracle database, tables and indexes are schema objects that exist in different namespaces. (Oracle Database SQL Language Reference 12c, 2014). A query is defined as an operation that retrieves data from one or more tables or views. In this reference, a top-level SELECT statement is called a query. Further, Oracle documentation defines indexes as structures that allow fast access path to data and are meant to reduce disk I/O, thereby improving query performance. Indexes are independent of the data in the tables and can be created or dropped without affecting the data. Similarly, SQL statements are written independent of the indexes and are not affected by any changes in indexes. (Oracle® Database Concepts, 2014).

Indexes can be created in one's own schema well as in another schema, with appropriate privileges. One key property of indexes in Oracle database that affects query optimization is whether the index is visible or invisible to the optimizer. An invisible index is maintained by Data Manipulation Language (DML) operations, but it is not used by the optimizer during query hard parsing and optimization unless explicitly asked for.

Similarly, in IBM's DB2 UDB database, an index is described as "a set of pointers that are logically ordered by the values of one or more keys. Indexes are used to improve performance and ensure uniqueness" (IBM DB2, 2014). In the Hierarchy of DB2 UDB authorities and privileges on database objects like tables and indexes, the only privilege that can be granted on indexes is "CONTROL", which grants the privilege to drop the index (Wasserman, 2012). DB2 UDB documentation further states that "Although the query optimizer decides whether to use a relational index to access relational table data, it is up to you to decide which indexes might improve performance and to create those indexes" (IBM DB2, 2014).

The optimizer is described as built-in database software that determines the most efficient way to execute a SQL statement by considering factors related to the objects referenced and the conditions specified in the statement. The Oracle database optimizer receives the parsed query and generates a set of potential plans for the SQL statement based on available access paths and hints. It estimates the cost of each plan based on statistics in the data dictionary. Optimizer statistics are created for the purposes of query optimization and are stored in the data dictionary. The cost of plans is an estimated value proportional to the expected resource use needed to execute the statement with a particular plan. It compares the costs of plans and chooses the lowest-cost plan, known as the query plan. As documented, "To choose an access path, the optimizer first determines which access paths are available by examining the conditions in the statement's WHERE clause and it's FROM clause. The optimizer then generates a set of possible execution plans using available access paths and estimates the cost of each plan, using the statistics for the index, columns, and tables accessible to the statement. Finally, the optimizer chooses the execution plan with the lowest estimated cost" (Oracle® Database Performance Tuning Guide, 2014).

In the existing architecture, we note that the RDBMS optimizer is burdened with all the analysis and decision-making that goes into finding the most optimal index access paths for the data being sought by an SQL query. The popular RDBMS implementations like Oracle, DB2 UDB and SQL Server provide limited control to the applications and developers, by way of SQL constructs, using index hints or altering session properties that can influence the optimizer's access path selection in using specific indexes built on tables. During its query plan generation, the optimizer takes into consideration all indexes created on tables with few exceptions like cases where in Oracle an index has been made unusable or invisible. We find that one limitation in the existing architecture is that during this process, the optimizer does not consider the already available contextual metadata in terms of index ownership and the application's module/action from which the SQL originated. The documented reason for this is that "some schema objects, such as clusters, indexes, triggers, and database links, do not have associated object privileges" (Oracle® Database Security Guide, 2014)

This is best illustrated with examples. Let's say in Oracle database, a user/schema 'B' representing a specialized application, has read access and index creation privileges on table 'T1' which is owned by another user/schema 'A'. Now indexes can be created on table 'T1' and owned by user/schema 'B' such as 'B.T1-IDX1'. Suppose that user/schema 'A' has two other existing indexes on table 'T1', namely 'A.T1-IDX2' and 'A.T1-IDX3'. Now, when a SQL from user/schema 'A' comes to the optimizer that involves data access from table 'T1', the optimizer takes into consideration all three indexes - 'A.T1-IDX2', 'A.T1-IDX3' as well as 'B.T1-IDX1'. It does not matter that index 'B.T-IDX1' was created by application designers to only support specific SQLs coming from user/schema 'B' for a specific application's module and action. Under this current architecture where indexes "belong" to a table without any object privileges of their own, and, therefore, must be considered and used by the optimizer for all SQLs referencing the table, we find two performance related problems as described in the following sections.

4.1 Higher Hard Parse times

The first problem in the existing architecture where indexes "belong" to tables and thus all indexes on all tables referenced in the query must be evaluated by the optimizer during hard parse, is that this can often lead to higher hard parse times and locking contentions, especially when multiple hard parse requests come in to the database concurrently accessing the same objects. In most N-Tier modern business applications, the SQLs are dynamically generated by middleware SQL-generation engines based on an abstracted logical data model layer. As a result of and due to various limitations in the process of run-time dynamic SQL generation, the number of tables referenced in SQLs has been getting larger. In addition, the number of indexes created on each table has also increased significantly due to the number and type of applications that need to be supported. For example, in the latest Oracle Fusion Applications, we frequently find SQLs referencing 20 to 40 tables, each with a large number of indexes. The final result is that the number of indexes for all tables in a SQL that need to be considered and evaluated by the optimizer has increased manifold. In addition, the optimizer goes through more permutations at hard parse time to get the most optimal execution plan due to many more query transformations in newer releases of RDBMSs like Oracle, DB2 UDB and SQL Server. In cases where there is a cost tie between different plans, the optimizer needs to decide which of the competing indexes to use. For example, Oracle database optimizer sorts index names alphabetically as the method of choosing one index over another, when both have the same cost. This results in increased query hard parse time.

4.2 Potential for Sub-optimal Plans

The second problem relates to sub-optimal execution plans chosen by the optimizer for some SQL queries, in part due to the large number of, and sometimes, competing indexes. The problem itself emanates from the combined effects of the complexity and size of the dynamically generated SQLs, the increased number of tables referenced in these SQLs, the large number of indexes created on these tables, the increased complexity and newer features of the optimizer code itself. The net result is that the optimizer has to now evaluate many more potential access paths and combinations of optimizations to satisfy the data requested by SQL queries. This results in a higher probability of the optimizer making some sub-optimal choices, which can be very detrimental to an application's performance and scalability. The contribution of the large number of indexes to this problem of sub-optimal plan choices is quite significant and thus, cannot be ignored. Oracle documents this problem in Oracle® Database Performance Tuning Guide under Section 14.1.1 titled 'Tuning the Logical Structure' as follows: "Note that creating an index to tune one statement can affect the optimizer's choice of execution plans for other statements. For example, if you create an index to be used by one statement, then the optimizer can choose to use that index for other statements in the application as well. For this reason, re-examine the application's performance and execution plans, and rerun the SQL trace facility after you have tuned those statements that you initially identified for tuning" (Oracle® Database Performance Tuning Guide, 2014). We note that this is a very limiting factor that creates additional testing and analysis workload for application architects and performance tuning experts when they want to create a new index for a specific application's module and/or action as noted in the documentation above.

We also note that in databases like Oracle, a lot of useful information about a SQL query is captured for reporting and performance analysis purposes but is currently not utilized by the optimizer as additional inputs into the metadata and statistics on which it relies to generate efficient query plans. For example, applications typically pass certain parameters along with the SQL request to the database including MODULE and ACTION using the package DBMS_APPLICATION_INFO.SET_MODULE.

However, this information is not currently utilized by the optimizer for the purposes of optimizing query plans, though it is used for certain other functions including Enterprise Manager's performance graphs, Active Session History (ASH) and Automatic Workload Repository (AWR) reports.

In brief, when run-time, dynamically generated SQLs from multiple applications' modules and actions execute SQL queries against the same tables, with very different SQL constructs and optimization goals, we find that this current architecture puts undue burden on the optimizer to make the access plan choices that can sometimes result in sub-optimal performance of such queries and resultant poor application performance.

We would like to note one additional limitation with the current architecture. Databases like Oracle have traditionally provided 'HINTS' to be used in SQLs as a way to influence the optimizer's decision-making and choices in the selection of access paths, including indexes. The hints include ways to specify that the optimizer should use named indexes as provided in the hint section of a SQL query. As documented, "The INDEX hint instructs the optimizer to use an index scan for the specified table. You can use the INDEX hint for function-based, domain, B-tree, bitmap, and bitmap join indexes." (Oracle Database Online Documentation, 2014). While this approach worked for older applications that used hand-crafted SQLs, the approach has many serious pitfalls for the run-time, dynamically generated SQLs that are produced by SQL-generation engines based on an abstracted logical data model, that are more the norm in N-tier application architectures that we see in the industry today. Adding hints in such dynamically generated SQLs has been found to be very risky in terms of unintended consequences for optimizer plan selection and performance of SQL queries. For example, in one implementation of Oracle Fusion Applications, addition of hints in ADF View Objects (VOs) by some developers and performance tuning experts, to tune one set of queries, resulted in many other queries originating from other modules/actions to suddenly perform extremely poorly. Subsequent performance analysis of such SQLs showed that the added hints were the root cause of performance degradation.

To the best of our knowledge and in our research, we do not find any current implementation of our proposed concept of “Context-Sensitive Indexes”. We describe "Context-Sensitive Indexes" as database index structures built on tables that are used in a manner which provides methods for the RDBMS optimizer to establish the context in terms of Tenant/User/Schema as well as Applications' MODULE and ACTION for the optimizer to use such established context for minimizing the number of indexes evaluated during hard parse with the aim of achieving the most optimal access paths and execution plan possible for the SQL query with reduced hard parse.

5 CONTEXT-SENSITIVE INDEXES FOR SQL QUERY PERFORMANCE

There are many factors that affect the performance of SQLs. Our focus in this paper is on making the use of indexes context-sensitive by the optimizer to arrive at the most optimal execution plan for a given tenant/user/schema or application’s module/action. The idea behind our proposal is two-fold. One, reducing the hard parse time by limiting the number of indexes that the optimizer needs to evaluate while arriving at query plans. Two, minimizing the possibility of the optimizer not choosing the most appropriate index and thus using an inefficient execution plan, by providing the optimizer a restricted list of indexes based on the application owner/architect’s knowledge of the application and data model including data shape in tables.

Context for indexes can be defined in terms of any one or more of the following:

1. Tenant context in case of multi-tenant applications or Schema/User context in case of multiple schemas/users accessing data from the same tables in the database.
2. Application Module context in both cases of single and multi-application environments.
3. Application Module’s Action context in case of both single and multi-application module environments.

For example, the new Fusion Applications from Oracle comprise of a number of different applications like CRM (Customer Relationship Management), HCM (Human Capital Management), CDM (Customer Data Management), OTBI (Oracle Transactional Business Applications), and Business Intelligence Applications (BI Apps) among others. Within each of the applications, there are various Modules, each with one or more Actions. As mentioned earlier, the SQLs are dynamically generated at run time, by ADF and BI Server. Within these applications, a number of schema tables are shared not only within applications’ modules but also across the applications. Even though the tables are shared, each application has some unique requirements that necessitate use of specific indexes for SQLs originating from those applications’ modules and actions. For example, SQLs for the Fusion CRM Applications generated for the transactional application and User Interface (UI) can be quite different from the SQLs generated by the Business Intelligence Server (BI Server) for OTBI and BI Apps Extract, Transform, Load (ETL) queries, which have entirely different SQL constructs and optimization goals due to their very nature. Even within the Fusion CRM applications, modules that serve the UI use cases are very different from uses cases like Microsoft Outlook integration using web services. In one internal implementation of Fusion Applications in Oracle, there are over four hundred module-action combinations. In a large number of such cases, “context-sensitive indexes” will provide application architects a very useful way of defining and creating indexes to serve specific use cases without the fear of interfering with the performance stability of queries from other applications and modules. All this becomes possible with our proposal to make indexes context-sensitive. In the next few paragraphs, we discuss how context for indexes is important in different scenarios.

5.1 The Multi-tenant/User/Schema Context-sensitive Indexes

Oracle 12c has introduced the concept of pluggable databases (PDB), whereby one PDB could be used for each tenant in the multi-tenant architecture (Oracle Database Online Documentation, 2014). However, for the use cases where multiple tenants share the same database and schema, whether in Oracle or other RDBMS systems, it has been a challenge for application architects and developers to achieve high performance. Our proposed concept of context-sensitive indexes provides for allowing the tenant/user/schema context in the creation and use of indexes. In our proposed architecture, it should be possible for RDBMSs to provide a methodology to not only segregate the ownership of specific indexes amongst tenants/users/schemas but also partition/filter the common indexes based on tenant id. This is possible if we move away from the historical architecture of indexes belonging to a table, rather than belonging to a tenant/user/schema and then providing visibility of those partitions to specific tenants/users/schemas whose data is indexed in those partitions.

In the context of tenants/users/schemas, we propose that the indexes (or index partitions) on common tables should be created and owned by tenants/users/schemas and the indexed data in such indexes should only contain filtered data based on tenant id.

Our proposed architecture is slightly different from the existing concept of partitioned tables and indexes. The existing global partitioned indexes in databases provide the means to create indexes on different partitioning keys and it is currently possible to partition the tables and/or indexes by tenant key. By creating such partitioned indexes, we can take advantage of the optimizer's ability to use partition-pruning and technique known as partition-wise joins to improve performance. However, this approach requires the creation and maintenance of such partitioned indexes for all tenants. This current methodology works for use cases where the indexes on the same columns are required for use by all tenants. However, it does not work in use cases where different tenants require indexes on different sets of columns. For example, let us say there are 100 tenants and an index named Index1 on Colum1, Column2 and Column3, partitioned by Tenant-ID. In the current architecture, this index would result in creation and maintenance of 100 partitions of Index1. If this index were needed only by 10 tenants, then the remaining 90 partitions will not only add to the maintenance overhead but we also put these 90 tenants confronting the problems of higher hard parse time and instable execution plans that we have described earlier in the paper due to high number of indexes. In our proposal, tenants/users/schemas would have the flexibility to create indexes that they need and have the index filtered down to only their own data. Other tenants/users/schemas would therefore not be burdened with such indexes that they do not have a need for. Overall, the database will also have lower cost of index maintenance by restricting the number, size and partitions of indexes.

It may be argued that our proposed methodology may lead to more number of indexes overall and arguably slightly higher cost of maintenance. However, this need not be the case if due diligence is done by application architects in creating a scheme of visible/invisible indexes from amongst the existing indexes itself. Also, the performance gains for queries from tenants/users will more than offset the slightly increased maintenance costs, if any. Such tenant specific filtered indexes (or index partitions) will be smaller in size and possibly have lower heights, with fewer levels and fewer leaf blocks because these only contain index entries for a specific tenant. Specifically, index fast full scan operations will benefit because their performance is directly proportional to the index size. There will be lesser number of index splits due to inserts, updates and deletes because now such operations will be divided and spread across the many different tenant/user indexes. DML operations like insert, update and delete done by a specific tenant/user will not require the updating of indexes for another tenant/user, thereby isolating index maintenance costs to the specific tenant.

Another key advantage in our proposed architecture is that tenants/users will get the freedom to create their own indexes that help with their specific types of queries and data shapes without burdening the optimizer from considering their indexes during creation of query plans for other tenants/users. This is very significant in terms of index creation and maintenance and will provide further isolation between tenants/users. The performance gains for SQL queries will therefore come from savings in hard parse time due to the fewer number of indexes to be evaluated by the optimizer, from potentially more stable, optimal execution plans as well as from better execution times due to the smaller traversal paths through such smaller and more compact indexes. We establish the performance advantages of our proposed architecture in the experimental results presented in Section 6 of this paper.

5.2 Applications' Module and Action Context-sensitive Indexes

To recall, in the current RDBMS architectures, indexes do not have object privileges and for the purposes of the optimizer's choice of SQL execution plan, it must consider all available indexes for tables referenced in the SQL query. Often, a large number of indexes get created on a table to support many different SQL queries to support multiple applications, modules and sub-modules (Actions). When this happens, the optimizer, in some cases, is unable to choose the best possible query plan due to the large number of competing indexes. We propose that indexes should be treated as database objects on which privileges can be granted and revoked. The reason for this proposal is to give application architects the ability to make the fine-grained choices about which indexes they wish to have access to for the purposes of optimization of their applications' SQL queries and which indexes to ignore as irrelevant for the optimization of their SQL queries. A common use case scenario these days is the different types of applications accessing data in the same tables, but with different SQL constructs and optimization goals. For example, in Oracle's Fusion Applications, the needs of the transactional Fusion Application's queries are very different from the requirements of OTBI queries. There are also many additional types of applications like mobile applications, bulk data loading, and BI Apps ETL for pulling data to the data warehouse. By modifying the RDBMS architecture to allow database objects like indexes to have object privileges, we can facilitate the creation of an application's Module and Action context which can then be used by the optimizer to limit the indexes it considers for optimization of SQL query plans.

The same concept can be further expanded to different Modules and Actions within modules of an application, whereby, it should be possible for application architects to specify to the RDBMS optimizer, which indexes should be considered while parsing and optimizing SQL queries originating from specific modules/actions of an application. Our proposed solution improves upon and provides a more generic and broader way to guide the optimizer in choosing efficient query plans over the existing methodology of specifying particular index usage hints in the text of specific SQLs, which is becoming very hard to achieve for dynamically generated SQLs.

It can be argued that one down-side of this feature could be generation of multiple child cursors for the same SQL because of the context and possibly resulting in increased number of hard parses. However, the effects of this feature would be similar to the effects of existing database features like adaptive cursor sharing, which create new child cursors based on factors like bind-peeking etc. Thus, the benefits of reduced hard parse times and more optimal execution plans should outweigh the effect of some additional hard parsing and creation of child cursors. Similar to any other database optimizer feature like adaptive cursor sharing, this new feature could also have a parameter to switch it on or off.

5.3 Implementation Aspects of Context-sensitive Indexes

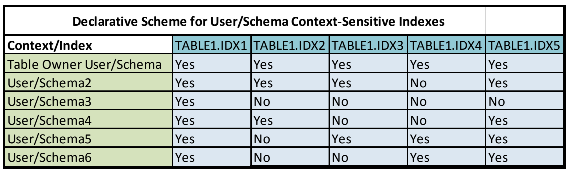
In the current RDBMS architectures, indexes do not have object privileges. For the purposes of the optimizer's choice of SQL execution plan, it must consider all available indexes for a given table referenced in the SQL query. Our proposed architecture can be achieved by RDBMS vendors in multiple ways. The first and foremost requirement is to upgrade the status of indexes in databases by declaring indexes as database objects on which VISIBLE privilege can be granted and revoked. "Context-Sensitive Indexes" implementation will need to be done at two levels of context as discussed in subsequent sections.

5.3.1 Tenant/User/Schema Context

The first enhancement to implement "context-sensitive indexes" would be to declare indexes as database objects on which an object privilege, VISIBLE, can be granted or revoked. This can be implemented similar to currently used privileges on objects like TABLE (SELECT, INSERT etc). By granting VISIBLE privilege on specific indexes to tenants/users/schemas, the optimizer can eliminate from evaluation certain indexes during hard parse on which the tenant/user/schema does not have VISIBLE privilege. This will give tenants/users/schemas the ability to make the fine-grained choices about which indexes they wish to have access to for the purposes of optimization of their SQL queries and which indexes to ignore as irrelevant. In addition, a Global Visible scheme for indexes can help application architects to grant VISIBLE to all Tenants/Users/Schemas for some or all indexes as needed. Alternately, the default privilege could be VISIBLE for all tenants/users/schemas on all indexes, as in current RDBMS implementations. Database administrators/application architects could then selectively revoke VISIBLE privilege on specific indexes that they do not wish to be used by the optimizer for their SQLs. This will achieve tenant/user/schema context-sensitive indexes. For example, say there are two users 'A' and 'B'. The following statements will grant and revoke VISIBLE privilege:

```
    GRANT VISIBLE ON <index_name> TO
<User B>;
    REVOKE VISBILE ON <index_name> FROM
<User A>;
```

The VISBILE privilege's grant and revoke architecture can address the implementation of context-sensitive indexes for multi-tenant and multi-user/schema environments as depicted in Figure 1.



Declarative Scheme for User/Schema Context-Sensitive Indexes

Context/Index	TABLE1.IDX1	TABLE1.IDX2	TABLE1.IDX3	TABLE1.IDX4	TABLE1.IDX5
Table Owner User/Schema	Yes	Yes	Yes	Yes	Yes
User/Schema2	Yes	Yes	Yes	No	Yes
User/Schema3	Yes	No	No	No	No
User/Schema4	Yes	Yes	No	No	Yes
User/Schema5	Yes	No	Yes	Yes	Yes
User/Schema6	Yes	No	No	Yes	Yes

Figure 1: Declarative Scheme for Tenant/User/Schema Context.

5.3.2 Applications' Module and Action Context

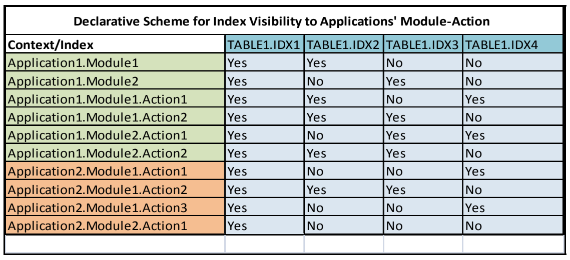
The aim is to make it possible for creating and declaring indexes that should only be used by specified applications' modules and actions. Hence, the visibility of such context-sensitive indexes will need to be controlled by an additional, optional scheme of granting or revoking visibility on indexes to/from specific combinations of Applications, Modules and Actions. This would require creation and maintenance of metadata related to application Modules and Actions by the RDBMS. For example, new object types – MODULE and ACTION – can be created that could be owned by tenants/users/schemas and stored in dictionary tables, for example, DBA_MODULES, DBA_MODULE_ACTIONS. These modules and actions could be granted VISIBLE privilege on indexes. Further, metadata for index visibility privileges could be stored in a new dictionary table, for example, DBA_IDX_PRIVS, similar to existing table privileges in DBA_TAB_PRIVS.

Application architects and owners can then register/create application modules and actions with/in the database in the same manner as other objects are currently created. Once created/registered, the application modules and actions could be used for index visibility grants. VISIBLE grant could be granted to an application's specific module and action once registered with the database using CREATE MODULE and CREATE ACTION statements. For example, user/schema 'A' could create/register MODULE1 and ACTION1, ACTION2 under that module and then control the visibility on indexes as under:

```
CREATE MODULE <Module1> FOR USER <User
A>;

CREATE ACTION <Action1> FOR MODULE
<User A>.<Module1>;

CREATE ACTION <Action2> FOR MODULE <User
A>.<Module1>;

GRANT VISIBLE ON <User A>.<Idx3> TO
<Module1.Action1>;

REVOKE VISIBLE ON <User A>.<Idx3> TO
<Module1.Action2>;
```

The above example would allow the index IDX3 to be used by the optimizer for SQLs originating from MODULE1.ACTION1 for user 'A' but not used for SQLs originating from MODULE1.ACTION2 for the same user. Providing such a granular, declarative way for the implementation of context-sensitive indexes to application architects and performance tuning professionals, the RDBMS optimizer's burden of making the right index choices will be reduced. As a result, the optimizer will do less work, save on time and database resources and arguably produce efficient, stable execution plans for SQL queries. Based on testing and performance review, applications' architects, database administrators and SQL tuning professionals will be able to make adjustments to the context-sensitive indexes declarative scheme to further fine-tune the matrix for optimal performance

A simplified chart example of such a declarative scheme of context-sensitive indexes is shown in Figure 2.



Declarative Scheme for Index Visibility to Applications' Module-Action

Context/Index	TABLE1.IDX1	TABLE1.IDX2	TABLE1.IDX3	TABLE1.IDX4
Application1.Module1	Yes	Yes	No	No
Application1.Module2	Yes	No	Yes	No
Application1.Module1.Action1	Yes	Yes	No	Yes
Application1.Module1.Action2	Yes	Yes	Yes	No
Application1.Module2.Action1	Yes	No	Yes	Yes
Application1.Module2.Action2	Yes	Yes	Yes	No
Application2.Module1.Action1	Yes	No	No	Yes
Application2.Module1.Action2	Yes	Yes	Yes	No
Application2.Module1.Action3	Yes	No	No	Yes
Application2.Module2.Action1	Yes	No	No	No

Figure 2: Declarative Scheme for Application Module/ Action Context-Sensitive Indexes.

Our other proposed methodology described above - of making indexes as objects on which grants can be given or revoked using tenants/user/schemas for the purposes of granting privileges on indexes - would co-exist with the applications' module/action context declarative scheme under our proposed granular architecture. Such context-sensitive indexes will go a long way in addressing the problems of performance and execution plan stability for SQL queries in a multi-tenant and/or multi-application mixed-load environment.

5.3.3 Maintenance of Context-sensitive Indexes

It could be argued that the proposed context-sensitive indexes may increase the maintenance overhead. However, we feel that the overhead in managing context-sensitive indexes is expected to be quite insignificant, especially in the context of packaged enterprise business applications which have many applications/modules/actions and users/schemas/tenant. The database optimizer, though very sophisticated, does not have a better understanding of the applications, their modules and actions than the application and SQL-tuning architects, especially when there are separate architects dedicated to different applications, modules and actions. Even under the present arrangement, the database optimizer does not play any significant role in the index design process. It is the application and SQL-tuning architects who design indexes and go through the SQL-tuning process iteratively, to add, remove and modify indexes. They do so, trying to find the elusive fine balance of just the right indexes that can serve all applications, modules and actions as well as tenants/users/schemas. This is simply based on their deep knowledge of the data model, application flows and SQL queries for those applications. What we have observed in the case of enterprise business applications like Oracle Fusion Applications, is that the design and development teams for different applications and modules/actions often have to add or modify indexes to tune the queries for their respective applications. Based on the review and analysis of hundreds of poorly performing SQL queries in Oracle Fusion Cloud Applications, we observe that very often, such actions to add or modify indexes, inadvertently cause adverse effects on the performance of many other applications' queries, which does not come to be realized until late in the release cycle, very often after customer complaints of sudden performance degradation after an upgrade or patch cycle. In addition, with the current architecture, it is not only very expensive to carry out meaningful regression testing for all applications/modules/actions when one or more indexes are added by developers for tuning one particular application/module/action but is nearly impractical to do so. The proposed context-sensitive indexes will provide a simple methodology for application and SQL-tuning architects to add, modify or remove indexes for the purposes of the optimizer's consideration while optimizing the execution plans by simply making the indexes visible or invisible. There is no additional overhead than what application architects currently do except deciding which indexes to use out of all the available ones.

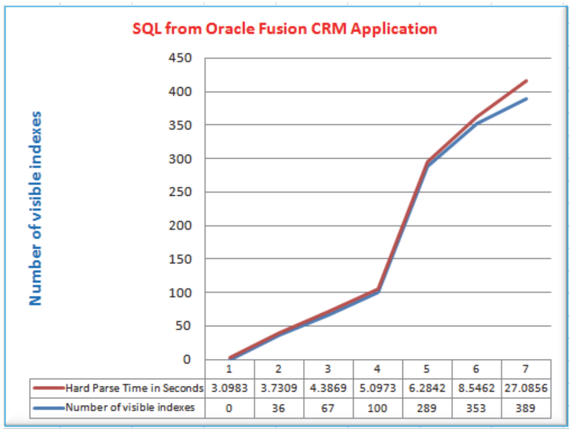
6 EXPERIMENTAL RESULTS

We sampled some SQLs from Oracle’s currently deployed 0 indexes visible or invisible. The following commands were used:

```
ALTER INDEX <Index Name> INVISIBLE;
ALTER INDEX <Index1> VISIBLE;
```

6.1 Benchmark Results: Higher Hard Parse times Use Case

We benchmarked representative SQLs from Oracle Fusion CRM Application against Oracle 11.2.0.3 database to find the co-relation between the number of visible indexes and the SQL hard parse time. One such SQL had 34 tables referenced, which had a combined total of 389 indexes. As the graph in Figure 3 demonstrates, the hard parse time closely followed the graph line for the number of visible indexes. Similar results were recorded for many SQLs benchmarked in a similar manner. What we found is that the higher the number of visible indexes, the higher the hard parse time.



	1	2	3	4	5	6	7
Hard Parse Time in Seconds	3.0983	3.7309	4.3869	5.0973	6.2842	8.5462	27.0856
Number of visible indexes	0	36	67	100	289	353	389

Figure 3: Co-relation between number of visible indexes and hard parse time.

6.2 Benchmark Results: Sub-optimal Plan Use Case

In the current Oracle Fusion Cloud Applications, high hard parse time and sub-optimal execution plans are very significant problems that cause poor user experience. The optimizer not picking the right indexes is a frequent finding in the slow SQL tuning analysis. We benchmarked such a representative SQL from Oracle Fusion CRM Application against Oracle 11.2.0.3 database to highlight use cases where the optimizer may choose the wrong indexes resulting in sub-optimal execution plan and extremely poor performance. The benchmarked SQL references 27 tables which have a total of 272 indexes. As Figure 4 and 5 demonstrate, performance in terms of execution time, CPU time and logical I/Os (buffer gets) is dramatically better in the use case where the number of visible indexes was restricted.

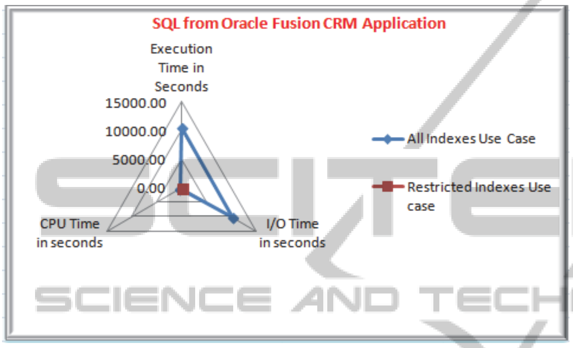


Figure 4: Co-relation between number of visible indexes and execution, CPU and I/O time.

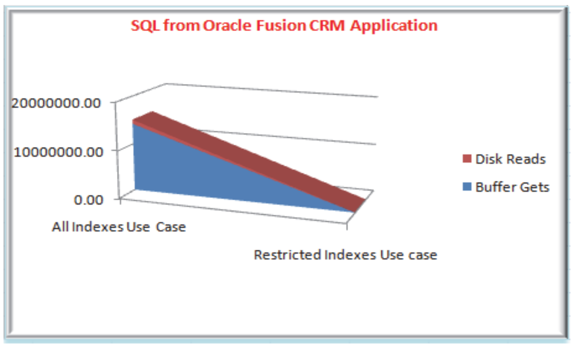


Figure 5: Co-relation between number of visible indexes and Buffer Gets and Disk Reads.

7 CONCLUSIONS

The use of indexes in various forms and physical implementations has long been established as an industry standard practice for RDBMSs and is used in many types of applications including enterprise business applications. However, indexes have largely been treated as being tightly coupled with tables for the purposes of privileges and visibility to the optimizer for use in generation of query execution plans. The effects of having a large number of indexes on the optimizer's performance have also been much researched. However, treating indexes as schema objects on which privileges and grants can be given has not been the focus of much research. In this paper we have researched, documented and presented a novel architecture whereby indexes can be made context-sensitive to accommodate a declarative scheme of telling the optimizer which indexes to consider for specific tenants/users/schema and/or applications' modules and actions. The current methodology of the optimizer taking into consideration all indexes created on a table without regard to the context in which a SQL query has been sent to the database, poses performance problems and can be a risk to the success of enterprise business applications. We have proposed a solution that takes into account context attributes for indexes under which the optimizer should process SQL queries to improve the performance of such SQLs, for which a US Patent application has been filed (ORA150617-US-PSP). Adoption of the proposed solution by RDBMS vendors can thus provide significant performance improvements in RDBMS optimizer query plan generation, thereby improving the performance and scalability of enterprise applications on one hand and consuming fewer database resources on the other.

REFERENCES

M. Leach, T. Lahdenmaki, 2005. *Relational database index design and the optimizers: DB2, Oracle, SQL Server et al*, John Wiley & Sons, Inc, Page: 4, 2005, ISBN-13 978-0-471-71999-1.

Chong, R. F., 2012. *Designing a database for multi-tenancy on the cloud - Considerations for SaaS vendors*. https://www.ibm.com/developerworks/data/library/techarticle/dm-1201dbdesigncloud.

Chong F., Carraro G., and Wolter R., 2006. *Multi-Tenant Data Architecture* http://msdn.microsoft.com/en-us/library/aa479086.aspx#mlttntda_topic2.

Oracle White Paper - *Application Development with Oracle Database 12c*, http://www.oracle.com/technetwork/database/multitenant/overview/index.html.

K.S.K. Kihong, K. C. Sang, 2001. Optimizing Multidimensional Index Trees for Main Memory. In proceedings of the 2001 ACM SIGMOD international conference on Management of data.

W. Powley, P. Martin, and P. Bird, 2008. Dbms workload control using throttling: experimental insights. *In Proceedings of the conference of the center for advanced studies on collaborative research, pages 1{13, New York, NY, USA, 2008. ACM.*

Brown, K. P., Mehta, M., Carey, M. J., and Livny, M., 1994. Towards Automated Performance Tuning for Complex Workloads. *In VLDB.*

Pang, H., Carey, M. J., and Livny, M., 1995. Multiclass Query Scheduling in Real-Time Database Systems. *In IEEE Trans. on Knowl. And Data Eng.*

Krueger, J., Tinnefeld, C., Grund, M., Zeier, A., & Plattner, H., 2010. A case for online mixed workload processing. *In DBTest.*

Oracle® Database *SQL Language Reference 12c Release 1(12.1)* http://docs.oracle.com/cd/E16655_01/server.121/e17209/sql_elements008.htm#SQLRF51129.

Oracle® *Database Concepts 12c Release 1 (12.1).* http://docs.oracle.com/cd/E16655_01/server.121/e17633/indexiot.htm#CNCPT721.

IBM *DB2 10.1 for Linux, UNIX, and Windows documentation*. http://pic.dhe.ibm.com/infocenter/db2luw/v10r1/index.jsp?topic=/com.

Wasserman, Ted J., 2012. *DB2 UDB security, Part 4: Understand how authorities and privileges are implemented in DB2 UDB*. http://www.ibm.com/developerworks/data/library/techarticle/dm-0601wasserman/

Oracle® Database Performance Tuning Guide 11g Release 2 (11.2) Chapter 11 *The Query Optimizer*. https://docs.oracle.com/cd/E29597_01/server.1111/e16638/optimops.htm.

Oracle® Database Security Guide 12c Release 1 (12.1). http://docs.oracle.com/cd/E16655_01/network.121/e17607/authorization.htm#DBSEG99910.

Oracle Database Online Documentation 11g Release 2 (11.2) / *Database Administration, Database SQL Language Reference*. https://docs.oracle.com/cd/E11882_01/server.112/e41084/sql_elements006.htm#SQLRF51098.

Oracle® Database Performance Tuning Guide 11g Release 2 (11.2) Chapter 19 *Using Optimizer Hints*. https://docs.oracle.com/cd/E11882_01/server.112/e41573/hintsref.htm#PFGRF005.

Oracle Database Online Documentation 12c Release 1 (12.1) / Database Administration Chapter 17 *Introduction to the Multitenant Architecture.* https://docs.oracle.com/database/121/CNCPT/cdbovrvw.htm#CNCPT89234.