Unlock the power of using Oracle Optimizer Hints to tune SQL statements

Common SQL tuning methods

SQL tuning is the process to improve a SQL statement's performance up to the user's expectation. There are at least three methods which are commonly used by developers or DBAs, These methods are creating new indexes, rewriting the SQL syntax and applying Oracle Optimizer Hints to the SQL statements. Each method has its pros and cons and suitable for different stages of application cycle. Let's discuss these three methods in the following.

Create new indexes for SQL statement

Creating new indexes for SQL statements are a very common method to improve SQL performance, it is especially important during database development. As new indexes to a SQL statement are not only affecting current SQL, it is also affecting other SQL statements running on the same database. So, it should be used very carefully in in production database. Normally, users are required to make impact analysis to other relevant SQL statements for the newly-created indexes.

Rewrite SQL statement

There are a lot of people teaching SQL rewrite skills over the internet. Most of those rewrite rules are inherited from Oracle rule-based SQL optimizer or older version of cost-based SQL optimizer. For example, some people may think that the following SQL may have different performances:

```
Select * from A where A.unique_key in (select B.unique_key from B);
Select * from A where exists (select ‘x’ from B where A.unique_key=B.unique_key);
```

Actually, if you put these two SQLs into Oracle database, you may probably get the same query plan from Oracle; it is because Oracle SQL optimizer will rewrite these two SQLs into one standard form internally to enable better query plans generation. A stronger internal rewrite ability was developed by Oracle SQL optimizer in last two decades. So, some obvious problems such as "KEY IS NOT NULL" or "NVL(KEY,'ABC') ='ABC' " were not able to use indexes are solved by Oracle already. Of course, there are still some limitations in Oracle SQL optimizer for complex SQL transformation, so experience users may still be able to tune a SQL statement through SQL rewrite to influence Oracle SQL optimizer to generate a better query plan. But this approach is getting more difficult to apply by DBAs since the relationship between SQL syntax and final query plan generation is getting weaker, this is because Oracle cost-based optimizer (CBO) is getting smarter and the syntax of the SQL text is no longer a dominating factor to the cost calculation and plans generation.

SQL rewrite is still useful both in development and production database, since it is an isolated change to a database and no other SQLs' performance will be affected, and it is safer than building new indexes. But it requires SQL code changes in program sources, so unit test or integration test may still be necessary. In contrast, using hints to tune a SQL is relatively safer, since it is not easy to change the semantics of the SQL accidentally.

Apply hints to SQL statement

Most databases in the market provide some sorts of query plan control language to help its optimizer to better optimize a specific SQL statement. For example; Optimization Guidelines of IBM LUW and Plan Guides of SQL Server are provided for many years already. Oracle provides Optimizer Hints that embedded with SQL text as a remark to tell Oracle SQL optimizer how to optimize the SQL statement. As hints will not affect the semantic of the SQL statement, so it is relatively safer than SQL rewrite and building new indexes. There are more than a hundred of documented Optimize Hints provided by Oracle. Unless you are an expert in using Optimizer Hints; using the right hints to tune a SQL is not easy to be mastered even by a human expert, since it is close to a million of permutations if we just pick 3 hints out from a hundred of hints.

Let's use a simple SQL example with the follow Optimizer Hints to show how Optimizer Hints works, the Hints is used to tell Oracle SQL optimizer to use index of DEPARTMENT table during query plan selection if possible. Oracle will try to find the lowest cost plan among all plans with index retrieval of DEPARTMENT table.

```
Select /*+ INDEX(@QB2 DEPARTMENT) */ *
From employee
Where emp_dept in (Select /*+ QB_NAME(QB2) */ dpt_id
                          From department
                          Where dpt_name LIKE 'D%');
```

Oracle SQL Hints is not a programing language

Oracle Optimizer Hints is not a programing language that come with proper syntax check or error return. It means an invalid Hints instruction to a SQL statement that Oracle SQL optimizer will not return with error message. Furthermore, even if it is a valid Hints instruction that Oracle SQL optimizer actually cannot comply with, there will be no error message returned too. So, users have to do a lot of trial and error before it can influence SQL optimizer to generate a specific better query plan.

Knowing the solution

It is a very common tuning practice that people are trying to find the best query plan for a bad performance SQL statement, it works like a human mimic Oracle SQL optimizer's job to optimize a SQL statement with human's knowledge. If the SQL statement is simple and Oracle SQL optimizer had made an obvious mistake, probably human intervention may work fine for this kind of simple SQL.

Here is an example:

```
Select * from A where A.KEY<'ABC';
```

If Oracle SQL optimizer fails to use index to retrieve records from table A and using index of KEY1 is actually a better solution. You can use Optimizer Hints to instruct Oracle to use index instead of full table scan for this SQL statement.

Select /+ INDEX(KEY1 KEY1_INX) */ from A where A.KEY1<'ABC' and A.KEY2<123 and A.KEY3<'C';*

Knowing the best solution is easy for simple SQL statements, but it is difficult for complex SQL statements. Since there are a lot of execution steps in the query plan for complex SQL, human beings are not able to estimate each step and work out a series of query steps to compose the best performance query plan. So, using Oracle Optimizer Hints to instruct SQL optimizer to generate specific query plan for complex SQL may not be easy even for an experienced SQL tuning expert.

Knowing the problem

Instead of knowing the solution of a SQL statement, it is relatively easier for a human expert to find where the problem is in a complex query plan. The way to tell Oracle to bypass the problem is applying hints with prefix "NO_" such as NO_INDEX or NO_USE_HASH. It tells Oracle not to use the specified operation in the query plan and select another operation instead with the lowest cost. This approach is not commonly adopted in the market due to people are normally bound by solution oriented thinking.

For example:

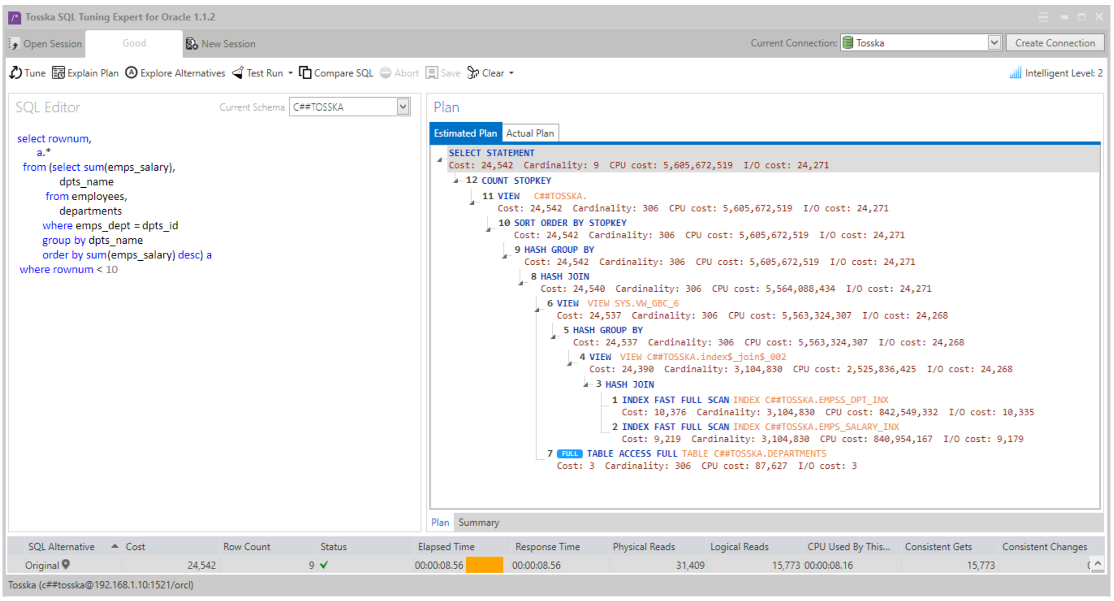
Select /+ NO_INDEX(KEY2 KEY2_INX) */ * from A where A.KEY1<'ABC' and A.KEY2<123 and A.KEY3<'C';*

If you think the KEY2_INX is not a good index to retrieve records from table A, you can disable the KEY2 index by applying /*+ NO_INDEX(KEY2 KEY2_INX) */ and let Oracle to select other index to retrieve the records.

Unlock the potential power of Oracle optimizer hints

Whether you are using solution oriented approach or problem bypassing approach, you need to know the details of the SQL's query plan, data distribution, cost of each step, and try to predict what will be the final actual aggregated cost of the plan. Sometimes you have to use both techniques for complex SQL statements; you have to provide the best query steps for parts of the query plan and use "NO_*" to bypass those known issues in the query plan. It is a very complicated process and it is not easy to carry out by common SQL developers without in-depth knowledge of SQL tuning.

The following is an example shows a SQL that took 8.56 seconds to finish and the query plan looks normal. The SQL syntax is intact, so it is not much room for SQL rewrite to take place. May be parallel execution can help to improve the SQL statement, but which table or index should be used for parallel execution and how the new parallel execution steps are affecting the entire query plan. It is not an easy job even for an experienced SQL tuning expert. That is why the potential of Oracle optimizer hints is not be fully explored from the beginning.



```
select rownum,
     a.*
 from (select sum(emps_salary),
           dpts_name
       from employees,
           departments
      where emps_dept = dpts_id
      group by dpts_name
      order by sum(emps_salary) desc) a
 where rownum < 10
```

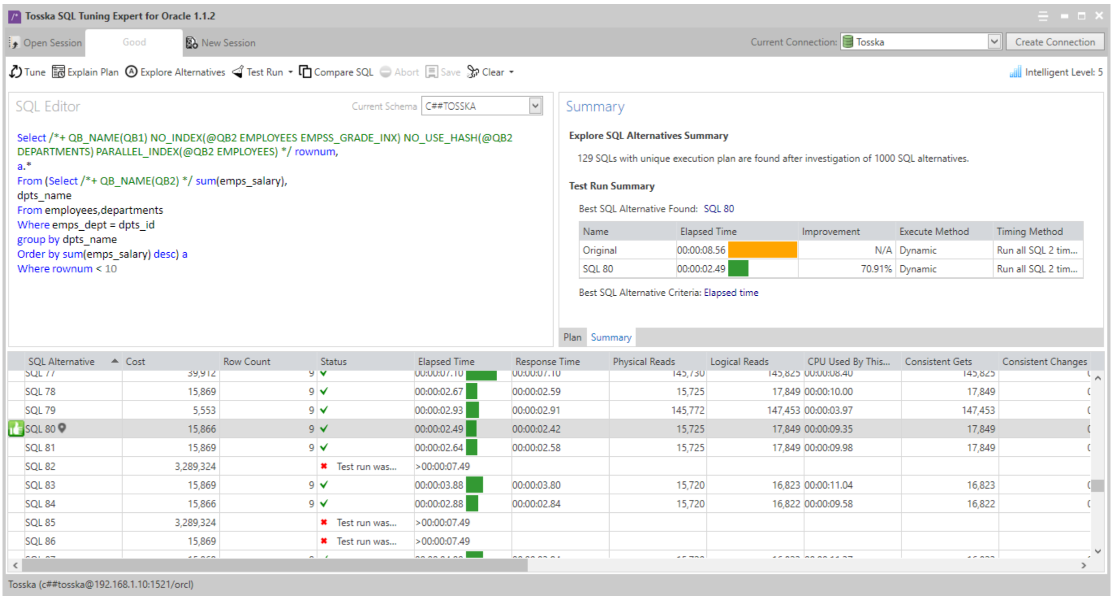
```
SELECT STATEMENT
Cost: 24,542  Cardinality: 9  CPU cost: 5,605,672,519  I/O cost: 24,271
  12 COUNT STOPKEY
    11 VIEW  C##TOSSKA.
       Cost: 24,542  Cardinality: 306  CPU cost: 5,605,672,519  I/O cost: 24,271
      10 SORT ORDER BY STOPKEY
         Cost: 24,542  Cardinality: 306  CPU cost: 5,605,672,519  I/O cost: 24,271
        9 HASH GROUP BY
          Cost: 24,542  Cardinality: 306  CPU cost: 5,605,672,519  I/O cost: 24,271
          8 HASH JOIN
            Cost: 24,540  Cardinality: 306  CPU cost: 5,564,088,434  I/O cost: 24,271
            6 VIEW  VIEW SYS.VW_GBC_6
              Cost: 24,537  Cardinality: 306  CPU cost: 5,563,324,307  I/O cost: 24,268
              5 HASH GROUP BY
                Cost: 24,537  Cardinality: 306  CPU cost: 5,563,324,307  I/O cost: 24,268
                4 VIEW  VIEW C##TOSSKA.index$_join$_002
                  Cost: 24,390  Cardinality: 3,104,830  CPU cost: 2,525,836,425  I/O cost: 24,268
                  3 HASH JOIN
                    1 INDEX FAST FULL SCAN INDEX C##TOSSKA.EMPSS_DPT_INX
                      Cost: 10,376  Cardinality: 3,104,830  CPU cost: 842,549,332  I/O cost: 10,335
                    2 INDEX FAST FULL SCAN INDEX C##TOSSKA.EMPS_SALARY_INX
                      Cost: 9,219  Cardinality: 3,104,830  CPU cost: 840,954,167  I/O cost: 9,179
            7 FULL TABLE ACCESS FULL TABLE C##TOSSKA.DEPARTMENTS
              Cost: 3  Cardinality: 306  CPU cost: 87,627  I/O cost: 3
```

SQL Alternative	Cost	Row Count	Status	Elapsed Time	Response Time	Physical Reads	Logical Reads	CPU Used By This...	Consistent Gets	Consistent Changes
Original	24,542	9	✓	00:00:08.56	00:00:08.56	31,409	15,773	00:00:08.16	15,773	

Tosska (c##tosska@192.168.1.10:1521/orcl)

With the help of a latest AI algorithm, a computer-searching engine can dramatically release user effort to discover the combination of Hints without going through a huge Hints permutation space. It makes Hints SQL tuning become easier and can solve more problems than you expected.

The following solution shows a series of hints combination that tell Oracle not to use EMPLOYEES.EMPSS_GRADE_INX index and exclude Hash Join to join table DEPARTMENTS and then use parallel index scan of table EMPLOYEES. It makes a new query plan that runs 70% faster than original plan. The whole tuning process is finished without human intervention and the result is safe since it does not involve any syntax rewrite.



```
Select /*+ QB_NAME(QB1) NO_INDEX(@QB2 EMPLOYEES EMPSS_GRADE_INX) NO_USE_HASH(@QB2
DEPARTMENTS) PARALLEL_INDEX(@QB2 EMPLOYEES) */ rownum,
a.*
From (Select /*+ QB_NAME(QB2) */ sum(emps_salary),
dpts_name
From employees,departments
Where emps_dept = dpts_id
group by dpts_name
Order by sum(emps_salary) desc) a
Where rownum < 10
```

Explore SQL Alternatives Summary

129 SQLs with unique execution plan are found after investigation of 1000 SQL alternatives.

Test Run Summary

Best SQL Alternative Found: SQL 80

Name	Elapsed Time	Improvement	Execute Method	Timing Method
Original	00:00:08.56	N/A	Dynamic	Run all SQL 2 tim...
SQL 80	00:00:02.49	70.91%	Dynamic	Run all SQL 2 tim...

Best SQL Alternative Criteria: Elapsed time

SQL Alternative	Cost	Row Count	Status	Elapsed Time	Response Time	Physical Reads	Logical Reads	CPU Used By This...	Consistent Gets	Consistent Changes
SQL 77	39,912	9	✓	00:00:07.10	00:00:07.10	145,730	145,825	00:00:08.40	145,825	
SQL 78	15,869	9	✓	00:00:02.67	00:00:02.59	15,725	17,849	00:00:10.00	17,849	
SQL 79	5,553	9	✓	00:00:02.93	00:00:02.91	145,772	147,453	00:00:03.97	147,453	
SQL 80	15,866	9	✓	00:00:02.49	00:00:02.42	15,725	17,849	00:00:09.35	17,849	
SQL 81	15,869	9	✓	00:00:02.64	00:00:02.58	15,725	17,849	00:00:09.98	17,849	
SQL 82	3,289,324		✖ Test run was...	>00:00:07.49						
SQL 83	15,869	9	✓	00:00:03.88	00:00:03.80	15,720	16,823	00:00:11.04	16,823	
SQL 84	15,866	9	✓	00:00:02.88	00:00:02.84	15,720	16,822	00:00:09.58	16,822	
SQL 85	3,289,324		✖ Test run was...	>00:00:07.49						
SQL 86	15,869		✖ Test run was...	>00:00:07.49						

Tosska (c##tosska@192.168.1.10:1521/orcl)

https://tosska.com/tosska-sql-tuning-expert-tse-oracle/

Tuning SQL without touching the its SQL text

If you are a packaged application user, you might be wondering how you can tune your SQL if you can't edit a query directly. Or, if you are an application developer, you want to have a quick fix on SQL performance without the time to go thought source code change and unit test. There are multiple features provided by Oracle such SQL Profiles, SQL Plan Baselines and SQL patch that you can use to tell Oracle to fix a SQL's query plan without the need to change the SQL text. But the using of these features are limited by Hints injection only, you cannot rewrite a SQL with different syntax and ask the original SQL to accept a rewritten SQL's query plan. So, hints-based SQL tuning is becoming more important in today's rapidly changing database applications.

Actually, SQL performance should be detached from application source code, the SQL performance should be manageable to deploy and rollback anytime, anywhere. It should also be tailorable for one source SQL code to fit different sizes of databases. Hints-based SQL tuning will be the key to unlock the potential power of SQL performance management for Oracle databases and it is becoming more important in modern database SQL tuning process.