Query Transformations

Randolf Geist
Freelance Consultant
Mannheim, Germany

Schlüsselworte:

CBO, Cost Based Optimizer, Troubleshooting, Query Optimizations

Einleitung:

Most people are unaware of the fact that the Oracle Optimizer applies various attempts to rewrite the provided SQL into more efficient forms as part of the optimization phase. These rewrites are known as "Query Transformations".

It is also not commonly known that many of these Query Transformations depend on Constraints declared which means that certain rewrites are only legal under certain assumptions. Providing this additional information in form of Constraints to the optimizer therefore potentially can speed up query execution dramatically.

Furthermore the most common query transformations will be explained along with how to control those using hints and manual query rewrites.

Finally some useful, but not yet implemented query transformations will be demonstrated and how to perform them using a manual rewrite.

Query Transformations

No matter what kind of query is submitted to the database, it will always undergo a transformation phase. The ultimate goal of these potential transformations is increased performance by opening up new access and joins paths that are otherwise not available. Up to Oracle 9i these transformations where based on heuristics and there was a clear distinction between the transformation and the optimization phase. First transformations were applied based on heuristic rules and afterwards the transformed query was optimized by finding the execution plan with the lowest cost. Starting with Oracle 10g these two phases are now interleaved: The new Cost Based Query Transformation feature determines the cost of different transformed and untransformed variations of the query and chooses the one with the lowest cost. This also means that the same query with exactly the same inputs to the optimizer in form of statistics can end up in 10g with a completely different execution plan than in 9i. It also means that this new feature not always results in superior execution plans because the chosen execution plan is based on estimates and as we all know estimates can be wrong. Since with every release Oracle improves the transformation capabilities it also means that execution plan regressions potentially happen with every upgrade. Note that there are various reasons for SQL regressions – this is just a common one.

Constraints

A lot of query transformations are only legal under certain assumptions. These assumptions are mostly based on constraints that need to be declared in the database, otherwise the optimizer and the transformation engine are not aware of them. Basically all kinds of constraints apply: NOT NULL, unique, primary key, foreign key and check constraints. Hence a very important aspect of constraints is often overlooked: By declaring the corresponding constraints one potentially can speed up query execution, sometimes dramatically. As usual a good balance needs to be found between the overhead of the enforcing the constraints at data manipulation time versus the potential gain at query execution time. In particular it needs to be considered that in many applications the data is only written once but queried a multitude of times, therefore the gain at read time might easily outweigh the potential loss at write time. It is a typical trade-off that needs to be evaluated for every individual application and workload.

The important message however is this: Constraints do potentially slow down at data manipulation time (this is well known), but they potentially also speed up query execution (this is not very popular)

Basic Transformations

View Merging

This is probably the most important transformation. Oracle usually attempts to merge all views so that a single query remains that then will be optimized. The reason for this is that an unmerged view will be optimized separately from other parts of the query, which potentially denies a lot of otherwise possible access and join paths. Of course starting from Oracle 10g on the merged and unmerged variants will be subject to the costing of the optimizer due to the Cost Based Query Transformation feature, but there is still some view merging that is performed based on heuristics. With each release Oracle improves the view merging capabilities; in particular this applies to the so called complex view merging where views contain DISTINCT, UNIQUE, subqueries or aggregates using GROUP BY. So with each upgrade it is possible that some views will be eligible for view merging that weren't before the upgrade – this is a common reason for performance problems after upgrades, because as outlined above the cost and or cardinality estimates of the optimizer might be wrong for various reasons and therefore the new transformations options could lead to worse performing execution plans.

Subquery Unnesting

If you make use of correlated or uncorrelated subqueries like ANY, ALL, [NOT] EXISTS, [NOT] IN, then in many cases these (and other kind of) subqueries can be unnested which means that they can be transformed into a join. Oracle supports special join methods like SEMI or ANTI joins that can be used to implement an [NOT] EXISTS operator using a join. If the subquery is not unnested then it will run as a so called FILTER subquery which potentially needs to be executed for every row generated by the main query. Although Oracle has a feature called FILTER subquery caching that attempts to reduce the number of executions of the filter subquery the unnested variant has some advantages that make it perform better in many cases. In particular the cardinality estimates of a filter subquery are often worse than the cardinality estimates based on a join. Also Oracle by default runs filter subqueries late which means that potentially more data has to be processed than necessary. This behaviour can also be controlled via two less known hints (PUSH_SUBQ, NO_PUSH_SUBQ)

Other transformations covered

The presentation will also cover other important transformations:
- Join Predicate Pushdown: This transformation can be helpful with views that at present cannot be merged by the transformation engine
- Or Expansion: This transformation can be useful if there are multiple OR expressions and each of them would benefit from different access paths
- Filter Pushdown: This is not exactly a transformation but explains what a filter pushdown is and how it can be prevented in the rare case it is desired

Manual rewrites – transformations not supported yet

Finally the presentation will cover some examples of manual SQL rewrites that can be helpful under certain conditions. It shows for example that there are cases where adding a join surprisingly can speed up query execution. Other helpful rewrites show how to optimize a query that contains a subquery combined with a Boolean OR condition – something that the transformation engine at present cannot transform into a join. An important point to consider when applying a manual rewrite is to ensure that the rewritten query is semantically equivalent and produces the same results. The LNNVL function that existed for a long time but was officially documented as of release 10.2 can be helpful to ensure that NULL values are treated correctly and therefore the rewritten query produces the same output.

Other topics covered

Advanced options of the DBMS_XPLAN package that can be helpful to get a better understanding of execution plans in general and query rewrites in particular

ANSI joins and their potential impact on hint usage

Advanced hint syntax introduced in Oracle 10g

Explanations regarding the special case of NULL values and why they are important for the query transformation options

Various examples of execution plans along with some explanations of unusual execution plan shapes, what these mean and what they actually should look like

Kontaktadresse:

Randolf Geist
Freelance Consultant
Ersteiner Straße 15
D-68229 Mannheim

Telefon:	+49 (0) 170-758 1171
E-Mail	info@sqltools-plusplus.org
Internet:	http://oracle-randolf.blogspot.com