Chase the Optimizer Every Step of the Way

Mauro Pagano
Accenture Enkitec Group
Miami, FL, USA

Keywords:
Oracle Database, Performance, SQL Tuning, Cost-Based Optimizer, CBO, Query Transformation

Introduction
The goal of this paper is to present a systematic way to review event 10053 trace files. In order to do so few concepts, fundamental to the understanding of the internal working mechanisms of the CBO, such as the Query Transformation framework, will be presented.

The Cost-Based Optimizer (CBO) is responsible of finding an optimal execution plan for any SQL statement executed by the database. Such task is especially challenging since the CBO has only partial representation of the underlying data (through statistics, collected based on sample and sometimes even misleading) and having to complete its job within milliseconds while using as little memory and CPU as possible.
The strict requirements the CBO operates under, conflict with the nature of the job itself, i.e. execute many calculations in order to find an optimal execution plan.
The optimizer tackles this challenge using two types of optimization: physical and logical.
Both optimization don't operate directly on the original SQL statement but on smaller subset(s) of it, called *query block(s)*.

Query Block
SQL allows an individual statement to be composed of many smaller unit, each somehow connected with the other. Some of those smaller units could even be executed as individual SQL, while some are so interrelated with other query blocks that it would not be possible to execute successfully standalone.
This "smaller" SQL pieces are referred to as query blocks. Basically each `SELECT` keyword in the original SQL statement denotes the beginning of a query block. Take for example, the following SQL statement:

```
select owner, object_name
  from t1
 where last_ddl_time > (select median(last_ddl_time)
                          from t2
                         where t1.owner = t2.owner);
```

Which has two query blocks, a top level one (table T1) and one for the subquery (table `T2`). From corresponding 10053 trace we see:

```
Registered qb: SEL$1 0x71b6df90 (PARSER)
---------------------
QUERY BLOCK SIGNATURE
---------------------
  signature (): qb_name=SEL$1 nbfros=1 flg=0
```

```
    fro(0): flg=4 objn=24765 hint_alias="T1"@"SEL$1"

Registered qb: SEL$2 0x71b5e260 (PARSER)
---------------------
QUERY BLOCK SIGNATURE
---------------------
  signature (): qb_name=SEL$2 nbfros=1 flg=0
    fro(0): flg=4 objn=24766 hint_alias="T2"@"SEL$2"
```

The query block approach of "count the `SELECT` keyword" is slightly incorrect though, since the SQL parser identifies as query blocks even other less obvious construct that are needed by the CBO for some internal reason. For example, the following SQL statement:

```
select *
  from t1
 union all
select *
  from t2;
```

Which has a total of three query blocks, one per each branch of the `UNION ALL` plus one representing the whole set, as we can see from corresponding 10053:

```
Registered qb: SEL$1 0x719d65b8 (PARSER)
---------------------
QUERY BLOCK SIGNATURE
---------------------
  signature (): qb_name=SEL$1 nbfros=1 flg=0
    fro(0): flg=4 objn=24765 hint_alias="T1"@"SEL$1"

Registered qb: SET$1 0x719c6610 (PARSER)
---------------------
QUERY BLOCK SIGNATURE
---------------------
  signature (): qb_name=SET$1 nbfros=1 flg=0
    fro(0): flg=0 objn=0 hint_alias="NULL_HALIAS"@"SET$1"

Registered qb: SEL$2 0x719c6c68 (PARSER)
---------------------
QUERY BLOCK SIGNATURE
---------------------
  signature (): qb_name=SEL$2 nbfros=1 flg=0
    fro(0): flg=4 objn=24766 hint_alias="T2"@"SEL$2"
```

In case of complex SQL statements with many views involved, it's not always a trivial task to determine how many query blocks are involved. Thus the best way to obtain this information is from the top section of a 10053 trace file.

Optimization

As previously mentioned, physical and logical optimizers operate on query blocks, using an iterative approach where the logical optimizer "output" is used as physical optimizer "input". This is because each optimization has different responsibility.

The Physical optimizer was the first component of the CBO, and it's basically the calculator part of the CBO since it is responsible for computing the cost of specific operations like:

- Access tables using different methods
- Determine an optimal join order between tables
- Determine an optimal join method between tables

This is the part of the optimizer that is always the focus of some many investigations, trying to determine why access method A was selected instead of B.
Plenty of literature already exists on it.

The Logical optimizer acts as the "brain" the CBO, and it is responsible for applying *transformations* to both individual query blocks as well as manipulating multiple query blocks at the same time in case of more complex transformations. It's paramount for the logical optimizer to keep the semantic of the original SQL.
Two types of transformation can be applied by the logical optimizer:

- Heuristic-based, which are always beneficial to apply
- Cost-based, for which the potential benefit needs to be evaluated

The evaluation in this case happens on a cost basis. The logical optimizer applies transformation and invokes the physical optimizer for the costing of each query block. The cost of each transformation "combination" is recorded and the logical optimizer determines which transformations are to be used in the end.

Most common transformations
There are several dozen transformations the CBO can apply. Which are used in any specific environment depends on many factors, including the "coding style" of the developers who wrote the code. For example, sites where extensive use of subquery is made, more likely encourage the popular Subquery Unnesting transformation.

The three most common transformations are:

- Subquery Unnesting – heuristic and cost-based
- View Merging – heuristic and cost-based
- Join Predicate Push Down – mostly cost-based nowadays

Examples and further details will be provided during the session.

Query Block Registry
Oracle provides a tracing event to dump all the calculations performed by the CBO through database event 10053. Unfortunately, the information generated by this event is not structured to be easily parsed by any tool (contrary to 10046 tracing info, designed to be parsed by tkprof), thus the 10053 trace file needs to be read by whomever is interested in understanding which decision lead to a specific plan.
Such task is particularly tedious because of the size for the raw file (usually in the order of hundreds of thousands of lines) and the highly repetitive nature of the information present makes it easier to get lost inside the file itself.

One of the last section in the 10053 trace file is the Query Block Registry that acts as a map of all the transformations that have been applied on each query block. It also documents how query blocks interacts with each other when complex transformations are applied.
For example, from the 10053 trace

```
Query Block Registry:
SEL$3 0x4c50a700 (PARSER) [FINAL]
  SEL$8771BF6C 0x0 (SUBQUERY UNNEST SEL$1; SEL$3;)
    SEL$7C12A527 0x0 (COMPLEX SUBQUERY UNNEST SEL$8771BF6C)
    SEL$5DB0472E 0x0 (SPLIT/MERGE QUERY BLOCKS SEL$8771BF6C)
    SEL$2AD7F9D9 0x0 (PUSHED PREDICATE SEL$291F8F59; SEL$8771BF6C; "VW_NSO_11"@"SEL$8771BF6C" 4)
  SEL$2E20A9F9 0x0 (SUBQUERY UNNEST SEL$7511BFD2; SEL$3; SEL$2;)
    SEL$CC348667 0x0 (COMPLEX SUBQUERY UNNEST SEL$2E20A9F9)
  SEL$291F8F59 0x0 (SUBQ INTO VIEW FOR COMPLEX UNNEST SEL$3)
    SEL$555A942D 0x0 (VIEW MERGE SEL$C149BB3C; SEL$291F8F59)
    SEL$C149BB3C 0x0 (PROJECTION VIEW FOR CVM SEL$291F8F59)
      SEL$555A942D 0x0 (VIEW MERGE SEL$C149BB3C; SEL$291F8F59)
    SEL$2AD7F9D9 0x0 (PUSHED PREDICATE SEL$291F8F59; SEL$8771BF6C; "VW_NSO_11"@"SEL$8771BF6C" 4)
SEL$2 0x4c50b6a0 (PARSER)
  SEL$C772B8D1 0x4c50ff78 (SUBQUERY UNNEST SEL$7511BFD2; SEL$2) [FINAL]
    SEL$841DDE77 0x0 (VIEW MERGE SEL$C772B8D1; SEL$683B0107)
    SEL$C6423BE4 0x0 (PUSHED PREDICATE SEL$683B0107; SEL$C772B8D1; "VW_SQ_1"@"SEL$7511BFD2" 4)
  SEL$2E20A9F9 0x0 (SUBQUERY UNNEST SEL$7511BFD2; SEL$3; SEL$2;)
    ...
  SEL$683B0107 0x4c50b6a0 (SUBQ INTO VIEW FOR COMPLEX UNNEST SEL$2) [FINAL]
    SEL$841DDE77 0x0 (VIEW MERGE SEL$C772B8D1; SEL$683B0107)
    SEL$C6423BE4 0x0 (PUSHED PREDICATE SEL$683B0107; SEL$C772B8D1; "VW_SQ_1"@"SEL$7511BFD2" 4)
SEL$1 0x4c50ff78 (PARSER)
  SEL$8771BF6C 0x0 (SUBQUERY UNNEST SEL$1; SEL$3;)
    ...
  SEL$7511BFD2 0x4c50ff78 (VIEW ADDED SEL$1)
    SEL$C772B8D1 0x4c50ff78 (SUBQUERY UNNEST SEL$7511BFD2; SEL$2) [FINAL]
      ...
    SEL$2E20A9F9 0x0 (SUBQUERY UNNEST SEL$7511BFD2; SEL$3; SEL$2;)
```

Each query block is represented individually as a starting point for transformations. The name for the new query blocks are also reported.
The indentation represents the nesting of transformations (interleaved transformations) while entries at the same level represents crossroads (juxtaposed transformations).
Query blocks identified by the CBO as part of the final plan are marked with the tag `[FINAL]`

Using the query block registry as a map to identify transformations / query blocks of interest allow to quickly move within large 10053 with accuracy, stepping into the details oft he physical optimizer in a matter of seconds and with a precise understanding of the step of the optimization the CBO was in.

A detailed example to show this method applied to a real case will be shown during the session.

Contact address:

Mauro Pagano
Accenture Enkitec Group
5605 North MacArthur Blvd., Suite 600
Irving, TX 75038

Phone: + 1 407 403 0685
Email mauro.pagano@gmail.com
Internet: http://mauro-pagano.com