ORACLE COST-BASED OPTIMIZER BASICS

Randolf Geist

http://oracle-randolf.blogspot.com
info@sqltools-plusplus.org
ABOUT ME

- Independent consultant
  - Available for consulting
  - In-house workshops
    - Cost-Based Optimizer
    - Performance By Design
  - Performance Troubleshooting
- Oracle ACE Director
- Member of OakTable Network
OVERVIEW

- Optimizer Basics – Key Concepts
- Proactive: Performance by design
- Reactive: Troubleshooting
Three main questions you should ask when looking for an efficient execution plan:

- How much data? How many rows / volume?
- How scattered / clustered is the data?
- Caching?

=> Know your data!
Why are these questions so important?

Two main strategies:

- One “Big Job”
  => How much data, volume?

- Few/many “Small Jobs”
  => How many times / rows?
  => Effort per iteration? Clustering / Caching
Optimizer’s cost estimate is based on:

- How much data? How many rows / volume?
- How scattered / clustered? (partially)
- (Caching?) Not at all
HOW MANY ROWS?

- Single table cardinality
- Join cardinality
- Filter subquery / Aggregation cardinality
Selectivity of predicates applying to a single table

```sql
select *
from t1, t2
where -- Filter predicates
t1.attr1 = 1
and t2.attr1 = 1
-- Filter predicates
-- can also be applied to
-- join predicates
and t1.id > 0
and t2.id > 0
-- Join predicates
and t1.id = t2.id
```
Selectivity of predicates applying to a single table
Selectivity of predicates applying to a single table

```sql
select *
from t1, t2
where
  -- Filter predicates
  t1.attr1 = 1
and  t2.attr1 = 1
  -- Filter predicates
  -- can also be applied to
  -- join predicates
  and  t1.id > 0
and  t2.id > 0
  -- Join predicates
  and  t1.id = t2.id
```
Selectivity of predicates applying to a single table
SINGLE TABLE CARDINALITY

- Optimizer challenges
  - Skewed column value distribution
  - Gaps / clustered values
  - Correlated column values
  - Complex predicates and expressions
  - Bind variables
SINGLE TABLE CARDINALITY

Demo!

optimizer_basics_single_table_cardinality_testcase.sql
SINGLE TABLE CARDINALITY

- Impact **NOT** limited to a "single table"

- Influences the favored Single Table Access Path (Full Table Scan, Index Access etc.)

- Influences the Join Order and Join Methods (NESTED LOOP, HASH, MERGE)

=> An incorrect single table cardinality potentially screws up whole execution plan!
JOIN CARDINALITY

- Oracle joins exactly two row sources at a time.
- If more than two row sources need to be joined, multiple join operations are required.
- Many different join orders possible (factorial!)
JOIN CARDINALITY

- Tree shape of execution plan
Challenges

- Getting the join selectivity right!

- A join can mean anything between no rows and a Cartesian product
Getting the join selectivity right
Getting the join cardinality right

Join cardinality = \text{Cardinality T1} \times \text{Cardinality T2} \times \text{Join selectivity}
JOIN CARDINALITY

- Challenges
  - Semi Joins (EXISTS (), = ANY())
  - Anti Joins (NOT EXISTS (), <> ALL())
  - Non-Equi Joins (Range, Unequal etc.)
Even for the most common form of a join - the Equi-Join – there are several challenges:

- Non-uniform join column value distribution
- Partially overlapping join columns
- Correlated column values
- Expressions
- Complex join expressions (multiple AND, OR)
JOIN CARDINALITY

Demo!

optimizer_basics_join_cardinality_testcase.sql
**JOIN CARDINALITY**

- Influences the Join Order and Join Methods (NESTED LOOP, HASH, MERGE)

=> An incorrect join cardinality/selectivity potentially screws up whole execution plan!
Data is organized in blocks

Many rows can fit into a single block

According to a specific access pattern data can be either scattered across many different blocks or clustered in the same or few blocks

Does make a tremendous difference in terms of efficiency of a “Small Job”
1,000 rows => visit 1,000 table blocks: 1,000 * 5ms = 5 s
1,000 rows => visit 10 table blocks: 10 * 5ms = 50 ms
Scattered data means potentially many more blocks to compete for the Buffer Cache for the same number of rows

=> Caching!

Scattered data can result in increased
- physical disk I/O
- logical I/O
- write disk I/O (Log Writer, DB Writer)
- free buffer waits
Most OLTP data has a natural clustering. Data arriving around the same time is usually clustered together in a heap organized table. Depends on the physical organization. Partitioning for example can influence this clustering even for heap organized tables.
Clustering of data can be influenced by physical implementation

Physical design matters
- Segment space management (MSSM / ASSM)
- Partitioning
- Index/Hash Cluster
- Index Organized Tables (IOT)
- Index design / multi-column composite indexes

There is a reason why the Oracle internal data dictionary uses clusters all over the place
No table access => only index blocks are visited!
There is only a single measure of clustering in Oracle: The index clustering factor

The index clustering factor is represented by a single value

The logic measuring the clustering factor by default does not cater for data clustered across few blocks (ASSM!)
Challenges

- Getting the index clustering factor right

- There are various reasons why the index clustering factor measured by Oracle might not be representative
  - Multiple freelists / freelist groups (MSSM)
  - ASSM
  - Partitioning
  - SHRINK SPACE effects
Re-visiting the same recent table blocks
HOW SCATTERED / CLUSTERED?

- Challenges
  - There is no inter-table clustering measurement
  - The optimizer therefore doesn’t really have a clue about the clustering of joins
  - You may need to influence the optimizer’s decisions if you know about this clustering
HOW SCATTERED / CLUSTERED?
HOW SCATTERED / CLUSTERED?
HOW SCATTERED / CLUSTERED?
HOW SCATTERED / CLUSTERED?

Demo!

optimizer_basics_inter_table_clustering_testcase.sql
The optimizer’s model by default doesn’t consider caching of data.

Every I/O is assumed to be physical I/O.

But there is a huge difference between logical I/O (measured in microseconds) and physical I/O (measured in milliseconds).
You might have knowledge of particular application data that is “hot” and usually stays in the Buffer Cache.

Therefore certain queries against this “hot” data can be designed based on that knowledge.

The optimizer doesn’t know about this. You may need to influence the optimizer’s decisions.
Oracle obviously played with the idea of introducing an **per object** caching component into the cost calculation in 9i and 10g.

You can see this from the undocumented parameters **_optimizer_cache_stats** and **_cache_stats_monitor** as well as the columns **AVG_CACHED_BLOCKS** and **AVG_CACHE_HIT_RATIO** in the data dictionary.
It is important to point out that even logical I/O is not “free”

So even by putting all objects entirely in the Buffer Cache inefficient execution plans may still lead to poor performance

Excessive logical I/O, in particular on “hot blocks”, can lead to latch contention and CPU starvation
SUMMARY

- Cardinality and Clustering determine whether the “Big Job” or “Small Job” strategy should be preferred

- If the optimizer gets these estimates right, the resulting execution plan will be efficient within the boundaries of the given access paths

- Know your data and business questions
How to apply these concepts, where to go from here?

- Read Jonathan Lewis’ article “Designing Efficient SQL” at Red Gate’s “Simple Talk”

Probably the best coverage of the concepts outlined here including clustering and caching

http://www.simple-talk.com/sql/performance/designing-efficient-sql-a-visual-approach/
PERFORMANCE BY DESIGN

How to apply these concepts, where to go from here?

Read Jonathan Lewis' article "Designing Efficient SQL" at Red Gate's "Simple Talk".

http://www.simple-talk.com/sql/performance/designing-efficient-sql-a-visual-approach/
How to apply these concepts, where to go from here?

- Read one of Tom Kyte’s books to learn more about the pro’s and con’s of clusters and index organized tables.
PERFORMANCE BY DESIGN

- How to apply these concepts, where to go from here?

Read one of Tom Kyte's books to learn more about the pro's and con's of clusters and index organized tables.
How to apply these concepts, where to go from here?

- Learn how to read, interpret and understand Oracle execution plans
  => Chapter 6 of “Troubleshooting Oracle Performance” by Christian Antognini

- This knowledge is required in order to compare your understanding of the query to the optimizer’s understanding
How to apply these concepts, where to go from here?

- Learn how to read, interpret and understand Oracle execution plans. => Chapter 6 of "Troubleshooting Oracle Performance" by Christian Antognini
- This knowledge is required in order to compare your understanding of the query to the optimizer's understanding.

Figure 6-2. Parent-child relationships between execution plan operations
How to apply these concepts, where to go from here?

- Be aware of Query Transformations: The optimizer might rewrite your query to something that is semantically equivalent but potentially more efficient.

- This might take you by surprise when trying to understand the execution plan favored by the optimizer.

Query transformation examples by courtesy of Joze Senegacnik (OOW 2010)
How to apply these concepts, where to go from here?

Be aware of Query Transformations:
The optimizer might rewrite your query to something that is semantically equivalent but potentially more efficient.

This might take you by surprise when trying to understand the execution plan favored by the optimizer.

Query transformation examples by courtesy of Joze Senegacnik (OOW 2010)
How to apply these concepts, where to go from here?

- Be aware of Query Transformations: The optimizer might rewrite your query to something that is semantically equivalent but potentially more efficient. This might take you by surprise when trying to understand the execution plan favored by the optimizer.

Query transformation examples by courtesy of Joze Senegacnik (OOW 2010)
If you want a more formal approach

- Read “SQL Tuning” by Dan Tow
  - Teaches a formal approach how to design and visualize an execution plan
  - Focuses on “robust” execution plans in an OLTP environment
  - The formal approach doesn’t take into account clustering and caching, however it is mentioned in the book at some places
If you want a more formal approach, read “SQL Tuning” by Dan Tow. The book teaches a formal approach for designing and visualizing an execution plan, focusing on robust execution plans in an OLTP environment. While the formal approach does not take into account clustering and caching, it is mentioned in the book at some places.

Check for any unique filter conditions that you would annotate with an asterisk (Step 6). In the case of this example, there are no such conditions.

Then, place all of these numbers onto the query diagram, as shown in Figure 5-5.

![Figure 5-5. The completed query diagram for the second example](image)

**Shortcuts**

Although the full process of completing a detailed, complete query diagram for a many-way join looks and is time-consuming, you can employ many shortcuts that usually reduce the process to a few minutes or even less:
If you want a more formal approach

Read “Relational Database Index Design and the Optimizers” by Tapio Lahdenmäki and Michael Leach

- Focuses on index design
- Provides simple and more advanced formulas allowing to predict the efficiency of queries and indexes
- Covers clustering and caching
If you want a more formal approach, read "Relational Database Index Design and the Optimizers" by Tapio Lahdenmäki and Michael Leach. Focuses on index design, provides simple and more advanced formulas allowing to predict the efficiency of queries and indexes, covers clustering and caching.
PERFORMANCE BY DESIGN

- For application developers
  - Read “Use the Index, Luke” by Markus Winand
    - Focuses on index design
    - Provides a lot of examples how to design efficient database access using different front-end languages (Java, Perl, PHP, etc.)
    - Also available as free eBook
    - Cross database (Oracle DB2, MySQL…)

http://use-the-index-luke.com/
For application developers, "Use the Index, Luke" by Markus Winand focuses on index design and provides a lot of examples for designing efficient database access using different front-end languages (Java, Perl, PHP, etc.). The book is also available as a free eBook.

The following figures show what it means to use an index on an index. For comparison, we will first look at an index access on a heap-table. The search shown in Figure 5.2 uses an index on the sale date to find all sales for 23rd May 2011. The execution involves two steps: (1) the INDEX RANGE SCAN; (2) the TABLE ACCESS BY INDEX ROWID.

Figure 5.2. Index-Based Access on a Heap-Table

Although the table access might become the bottleneck, it is still limited to one access per row. That is because the index has the ROWID, which is a straight pointer to the row. Knowing the exact position allows the database to directly...
If you want to dive into the details of the Cost-Based Optimizer:

- Read "Cost-Based Oracle: Fundamentals" by Jonathan Lewis
  - Almost six years old
  - Still the best book about the Oracle optimizer
  - Covers the key concepts mentioned here in great detail
In Figure 4-1, we have a table with four blocks and 20 rows, and an index on the column V1, whose values are shown. If you start to walk across the bottom of the index, the first rowid points to the third row in the first block. We haven't visited any blocks yet, so this is a new block, so we count 1. Take one step along the index, and the rowid points to the fourth row of the second block—we've changed block, so increment the count. Take one step along the index, and the rowid points to the second row of the first block—we've changed block again, so increment the count again. Take one step along the index, and the rowid points to the fifth row of the first block—we haven't changed blocks, so don't increment the count.

In the diagram, I have put a number against each row of the table—this is to show the value of the counter as the walk gets to that row. By the time we get to the end of the index, we have changed table blocks ten times, so the clustering factor is 10.