Optimizer Terms (002)

Oracle Categories:

A - Architecture
B - Backup and recovery
C - Cloud
D - DataGuard
E - Exadata
F - FRA
G - Goldengate
H - High Availability - RAC
--
P - Patching/Performance tuning
U - Upgrade
T - Troubleshooting Skills
--

Skew :
---

https://stackoverflow.com/questions/35509868/what-is-skewed-column-in-oracle

Skewed columns are columns in which the data is not evenly distributed among the rows.

For example, suppose:

You have a table order_lines with 100,000,000 rows
The table has a column named customer_id
You have 1,000,000 distinct customers
Some (very large) customers can have hundreds of thousands or millions of order lines.
In the above example, the data in order_lines.customer_id is skewed. On average, you'd expect each distinct customer_id to have 100 order lines (100 million rows divided by 1 million distinct customers). But some large customers have many, many more than 100 order lines.

This hurts performance because Oracle bases its execution plan on statistics. So, statistically speaking, Oracle thinks it can access order_lines based on a non-unique index on customer_id and get only 100 records back, which it might then join to another table or whatever using a NESTED LOOP operation.

But, then when it actually gets 1,000,000 order lines for a particular customer, the index access and nested loop join are hideously slow. It would have been far better for Oracle to do a full table scan and hash join to the other table

---

Cardinality (NDV) :

---

http://dbaparadise.com/2017/08/5-things-to-better-understand-selectivity-and-cardinality/

https://en.wikipedia.org/wiki/Cardinality_(SQL_statements)

Distinct values in a particular column (attribute) of a database table.SQL databases use cardinality to help determine the optimal query plan for a given query.

Example:

Cardinality: select count(distinct (deptno)) from employee;

Values of cardinality

When dealing with columnar value sets, there are three types of cardinality: high-cardinality, normal-cardinality, and low-cardinality.

High-cardinality refers to columns with values that are very uncommon or unique. High-cardinality column values are typically identification numbers, email addresses, or user names. An example of a data table column with high-cardinality would be a USERS table with a column named USER_ID. This column would contain unique values of 1-n. Each time a new user is created in the USERS table, a new number would be created in the USER_ID column to identify them uniquely. Since the values held in the USER_ID column are unique, this column's cardinality type would be referred to as high-cardinality.

Normal-cardinality refers to columns with values that are somewhat uncommon. Normal-cardinality column values are typically names, street addresses, or vehicle types. An example of a data table column with normal-cardinality would be a CUSTOMER table with a column named LAST_NAME, containing the last names of customers. While some people have common last names, such as Smith, others have uncommon last names. Therefore, an examination of all of the values held in the LAST_NAME column would show "clumps" of names in some places (e.g. a lot of Smiths) surrounded on both sides by a long series of unique values. Since there is a variety of possible values held in this column, its cardinality type would be referred to as normal-cardinality.

Low-cardinality refers to columns with few unique values. Low-cardinality column values are typically status flags, Boolean values, or major classifications such as gender. An example of a data table column with low-cardinality would be a CUSTOMER table with a column named NEW_CUSTOMER. This column would contain only two distinct values: Y or N, denoting whether the customer was new or not. Since there are only two possible values held in this column, its cardinality type would be referred to as low-cardinality. [2]

Selectivity :

Always between 0 (or) 1

http://dbaparadise.com/2017/08/5-things-to-better-understand-selectivity-and-cardinality/
http://www.dbarepublic.com/2016/02/selectivity-vs-cardinilaty.html

Selectivity is the ratio of cardinality to the number of records of an Indexed column.
Selectivity = (Distinct Values/Total number of records)

Syntax:-

```
SELECT (distinct_keys / num_rows) AS Selectivity
FROM dba_indexes
WHERE index_name like 'Index_Name' and Owner ='Table_Owner';
```

Selectivity:

```
SELECT (distinct_keys / num_rows) AS Selectivity, distinct_keys, num_rows
FROM dba_indexes
WHERE index_name like 'COMPOSITE_INDEX' and Owner ='BANIYA';
```

Predicates :

---

https://use-the-index-luke.com/sql/explain-plan/oracle/filter-predicates

The Oracle database uses three different methods to apply where clauses (predicates):

Access predicate (“access”)
The access predicates express the start and stop conditions of the leaf node traversal.

Index filter predicate (“filter” for index operations)
Index filter predicates are applied during the leaf node traversal only. They do not contribute to the start and stop conditions and do not narrow the scanned range.

Table level filter predicate (“filter” for table operations)
Predicates on columns that are not part of the index are evaluated on table level. For that to happen, the database must load the row from the table first.

Extended Statistics :

Multi-Column (Column Group) Statistics

Individual column statistics are fine for working out the selectivity of a specific column in a where clause, but when the where clause includes multiple columns from the same table, the individual column statistics provide no indication of the relationship between the columns. This makes working out the selectivity of the column group very difficult.

Oracle uses workload analysis to generate column groups, but they can also be manipulated manually using the DBMS_STATS package. The CREATE_EXTENDED_STATS procedure is used to explicitly create multi-column statistics.

Histograms

Column level statistics

11g
Frequency
Height Balanced
12c
Top Frequency Histograms
Hybrid Histograms

Bind Variable Peeking(11.1) :

---

https://hourim.wordpress.com/2013/05/18/literal-bind-variable-and-adaptive-cursor-sharing-simplify-them-please/
https://igorusoltsev.wordpress.com/2011/11/13/the-elements-of-bind-aware-cursor-sharing-and-cardinality-feedback-in-oracle-11g/

It is an Oracle method in which before generating an execution plan. the CBO peeks at the
values of bind variables and uses them as Literals for generating better execution plan. The loop hole with this
approach was that generated execution plan depends on the values provided by the first execution. Note that
depending on value provided in where condition at time Full Table scan is required while at other time Index
usage is required. Since bind variable peeking was based on the values provided by the first execution so this
approach also produced bad execution plan at times. To counter this. Oracle added

little more intelligence into
the software with "Adaptive Cursor Sharing".

Adaptive cursor sharing(11.1 & 11.2) :

However, bind variable peeking occurs only at hard parse time which means as far as the query is not hard parsed it will share the same execution plan that corresponds to the last hard parsed bind variable. In order to avoid such situation Oracle introduces in its 11gR2 release, Adaptive Cursor Sharing allowing Oracle to adapt itself to the bind variable when necessary without having to wait for a hard parse of the query.

In this approach Oracle CBO do not just
blindly use cursor even if it has sub-optimal plan (like it used to happen when using bind variable peeking)
instead it recognizes when the re-use of an already available cursor will lead to inefficient SQL execution. If CBO
using Adaptive Cursor Sharing finds that existing plan will not prove efficient then it will generate yet another child
cursor with different execution plan.

Statistics Feedback (Cardinality Feedback) (11.2)

Cardinality feedback was introduced in Oracle Database 11g Release 2. When the optimizer generates an execution plan the presence of missing statistics, stale statistics, complex predicates or complex operators may trigger the optimizer to monitor the cardinality of operations in the plan. Once the execution is complete, if there is a significant difference between the estimated and actual cardinalities, the actual cardinalities are stored in the SGA for later use and the statement is marked as reoptimizable. On next execution the statement is reoptimized using the stored cardinalities, allowing a better plan to be determined. Cardinality feedback is statement specific and is lost if the instance is restarted or the statement is aged out of the shared pool. In Oracle Database 12c, cardinality feedback has been renamed to statistics feedback.

Force Matching Signature :

Internally, the force_matching_signature column derives from the kglobt49 column of the x$kglcursor_child table.
These signature parameters can be used to indicate the status of the cursor_sharing

parameter:

When Cursor_sharing=force ==> force_matching_signature applies
When Cursor_sharing=exact ==> exact_matching_signature applies

By setting force_match to true, the SQL profile additionally targets all SQL statements that have the same text after the literal values in the WHERE clause have been replaced by bind variables. This setting may be useful for applications that use only literal values because it enables SQL with text differing only in its literal values to share a SQL profile. If both literal values and bind variables are in the SQL text, or if force_match is set to false (default), then the literal values in the WHERE clause are not replaced by bind variables.

Dynamic Sampling/Dynamic Statistics(12.1) :

This feature was enhanced and renamed Dynamic Statistics in Oracle Database 12c.
it is used when regular statistics are not sufficient to get good quality cardinality estimates.
If the available statistics are not enough, DS will be used. It is typically used to compensate for missing or insufficient statistics that would otherwise lead to a very bad plan.

Adaptive Plans (12.1)

Adaptive Plans in Oracle Database 12c allow runtime changes to execution plans. Rather than selecting a single "best" plan, the optimizer will determine the default plan, and can include alternative subplans for each major join operation in the plan. At runtime the cardinality of operations is checked using statistics collectors and compared to the cardinality estimates used to generate the execution plan. If the cardinality of the operation is not as expected, an alternative subplan can be used

---

Adaptive Query Optimization (12.1 and 12.2) :

Adaptive Query Optimization is a term used in Oracle Database 12c to describe a collection of features that work together to allow the cost based optimizer (CBO) to improve the accuracy of execution plans. Some of the features are renamed versions of functionality from previous releases, while others are new to Oracle Database 12c.

In Oracle 12.2 the OPTIMIZER_ADAPTIVE_FEATURES parameter has been removed and replace by two new parameters.

OPTIMIZER_ADAPTIVE_PLANS : Default (TRUE). Enables/disables adaptive plans, star transformation bitmap pruning and the adaptive parallel distribution method.
OPTIMIZER_ADAPTIVE_STATISTICS : Default (FALSE). Enables/disables SQL plan directives, statistics feedback for joins, performance feedback and adaptive dynamic sampling for parallel execution.

Adaptive SQL Plan Management (SPM)(12.1 and 12.2)

SQL Plan Management was introduced in Oracle 11g to provide a "conservative plan selection strategy" for the optimizer. The basic concepts have not changed in Oracle 12c, but there have been some changes to the process of evolving SQL plan baselines. As with previous releases, auto-capture of SQL plan baselines is disabled by default, but evolution of existing baselines is now automated. In addition, manual evolution of sql plan baselines has been altered to a task-based approach.

Automatic Reoptimization (12.1)

Automatic Reoptimization is based on learning lessons from a previous execution and feeding that information back to the optimizer,
so it can make a better decision the next time round.

B-Tree & Bitmap Indexes

The bitmap index : columns with lots of duplicate values (low cardinality),
b-tree index : best for high cardinality columns.

Loops

Nested Loops = When subquery gives Less number of records
Hash Join = When Subquery Gives More number of records

Access Method/Access Path

An access method or access path shows the way the data will be accesses from each table/index. This can be seen in the operation column tab in EXPLAIN PLAN.

Oracle Supports the below access methods.

Full Table SCAN (FTS)
Table Access by ROW-ID
Index Unique Scan
Index Range Scan
Index Skip Scan
Full Index Scan
Fast Full Index Scans
Index Joins
Hash Access
Cluster Access
Bit Map Index

Full Table SCAN (FTS) :

Every Row in the table is accessed and those rows are filtered which do not happen to meet selection criteria.Those blocks are read which are below HWM (A block has to be read to know whether it contains data.irrespective of the data residing on the block or not),Hence the number of logical reads depends on the number of blocks, not on the number of rows.

When a FTS is performed, all the blocks are read sequentially as blocks are adjacent to one another.Based on the init parameter DB_FILE_MULTIBLOCK_READ_COUNT,larger I/O calls can be made to read more blocks in one shot.
Large amount of data whenever accessed FTS are cheaper than index range scans which is because full table scans can use larger I/O calls, and making fewer large I/O calls is cheaper than making many smaller calls.

It's a very Costly operation based on the # of rows present in the table.In Many Cases this is a performance killer and mainly happens if indexes are not properly created or are unusable or large amount of data or parallelism or hints or when optimizer decides to go for FTS for whatever reason.

Table Access by ROWID:

Rowid of a row specifies datafile,data block within the file and location of row within that block.Oracle initially obtains the rowid's either from a where clause predicate or through an index scan of one or more of table's indexes.Once these indexes are obtained,the required rows are selected based on the rowid'sand does a row by row access.

When the Optimizer Uses Rowids:
This is generally the second step after retrieving the rowid from an index. The table access might be required for any columns in the statement not present in the index.Access by rowid does not need to follow every index scan. If the index

contains all the columns needed for the statement, then table access by rowid might not occur.

Index Unique Scan:

As the name says, it returns at most, a single rowid only.Oracle performs a unique scan if a statement contains a UNIQUE or a PRIMARY KEY constraint that guarantees that only a single row is accessed.

Index Range Scan:

It's a pretty common operation for accessing the data which is Selective. This can be an in-bout or out-bound on both the ends. By algorithm the data is sorted in the ascending order of the indexed columns.

An index range scan is a common operation for accessing selective data. It can be bounded (bounded on both sides) or unbounded (on one or both sides). Data is returned in the ascending order of index columns. Multiple rows with identical values are sorted in ascending order by rowid.

Index Range scan descending: It's conceptually the same as an index range scan but used when it's needed a situation alike of .. "order by descending".

When the Optimizer Uses Index range scan :

The optimizer uses a range scan when it finds one or more leading columns of an index specified in conditions, such as the following:

```
col1 = :b1
col1 < :b1
col1 > :b1
```

AND combination of the preceding conditions for leading columns in the index col1 like 'ASD%' wild-card searches should not be in a leading position otherwise the condition col1 like '%ASD' does not result in a range scan.
Range scans can use unique or nonunique indexes. Range scans avoid sorting when index columns constitute the ORDER BY/GROUP BY clause.

Index Skip Scan:

As we know ..Often, scanning index blocks is faster than scanning table data blocks.

Normally,in order for an index to be used,the prefix of the index key(leading rdge of the index) would be referenced in the query.However, if all the other columns in an index are referenced in the query except the first (leading one)…. Oracle prefers index skip scan,i.e.. to skip the first column of the index and use the rest of it.

Simply put, Skip scanning lets a composite index be split logically into smaller subindexes. In skip scanning, the initial column of the composite index is not

specified in the query. In other words, it is skipped and the number of logical subindexes is determined by the number of distinct values in the initial column.

This can be advantageous if there are few distinct valuesin the leading column of the concatenated index and many distinct values in the non-leading key of the index.

Full Scans:

A full scan is available if a predicate references one of the columns in the index. The predicate does not need to be an index driver. A full scan is also available when there is no predicate, if both the following conditions are met:

All of the columns in the table referenced in the query are included in the index.
At least one of the index columns is not null.
A full scan can be used to eliminate a sort operation, because the data is ordered by the index key. It reads the blocks singly.

Full Index Scan:

A full index scan does not read every block in the index structure,contrary to the name suggests. An index full scan processes all of the leaf blocks of an index,but only enough of the branch blocks to find the first leaf block.It is used when all of the columns necessary to satisfy the statement are inindex and it is cheaper than scanning than table.

It uses single block i/0 and may be used in any of the following situations.

An order by clause has all of the index columns in it and the order is the same as in the index.(This can also contain a subset of the columns in the index.
The query requires a sort-merge join and all the columns referenced in the query are in the index.
order of the columns referenced in the query matches the order of the leading index columns.
A group by clause is present in the query and the columns in the group by clause are present in the index.

Fast Full Index Scans

Fast full index scans are an alternative to a full table scan when the index contains all the columns that are needed for the query, and at least one column in the index key has the NOT NULL constraint. A fast full scan accesses the data in the index itself, without accessing the table. It cannot be used to eliminate a sort operation, because the data is not ordered by the index key. It reads the entire index using multiblock reads, unlike a full index scan, and can be parallelized.

Fast full index scans cannot be performed against bitmap indexes.

A fast full scan is faster than a normal full index scan in that it can use multiblock I/O and can be parallelized just like a table scan.

Index Joins:

It's a join of several indexes on the same tablethat collectively contain all of the columns that are referenced in the query from that of the table. Whenevr Index Join is used no table access is required as all the relevant columns are retrieved from the joined indexes.

This can never be used to eliminate a sort operation.

Hash Access:

A hash scan is used to locate rows in a hash cluster, based on a hash value. In a hash cluster, all rows with the same hash value are stored in the same data block. To perform a hash scan, Oracle first obtains the hash value by applying a hash function to a cluster key value specified by the statement. Oracle then scans the data blocks containing rows with that hash value.

Cluster Access:

A cluster scan is used to retrieve, from a table stored in an indexed cluster, all rows that have the same cluster key value. In an indexed cluster, all rows with the same cluster key value are stored in the same data block. To perform a cluster scan, Oracle first obtains the rowid of one of the selected rows by scanning the cluster index. Oracle then locates the rows based on this rowid.