Tuning Oracle Query Execution Performance

The performance of SQL queries in Oracle can be modified in a number of ways:

- By selecting a specific query optimization approach and goal.
- By creating and maintaining indexes on database tables.
- By collecting and maintaining database statistics.
- By providing expilict query execution hints to the query compiler.

Selection of Optimier approach and goal

Oracle has two built in optimizers:

- Cost-Based Optimizer (CBO);
- Rule-Based Optimizer (RBO).

Oracle can optimize queries for two different *goals*:

- throughput: the total amount of resources (I/O, memory, processor time) to process **all rows** accessed by the SQL statement;
- response time: the amount of resources (I/O, memory, processor time) to process the first row accessed by the SQL statement.

Optmizing for throughput spends the least amount of resources on computing the answer to the entire query. Convenient when queries are processed off-line.

Optimizing for response time spend the least amount of resources before commencing the output (but may take longer to produce the entire answer). Convenient when queries are processed on-line.

You can select the appropriate optimizer and optimizer behavior (goal) using the `ALTER SESSION` command. Oracle stores current choice of optimizer and optimizer goal in a parameter `OPTIMIZER_MODE`. To set new optimizer behavior, issue the following SQL command:

```
ALTER SESSION SET OPTIMIZER_MODE = <Value>;
```

The following values for the `OPTIMIZER_MODE` parameter are accepted.

`Value`	Optmizer	Goal	Comment
`CHOOSE`	*selected by Oracle*	throughput	*default value*. Lets Oracle select which optimizer to choose
`ALL_ROWS`	CBO	throughput	forces the use of CBO, optimizes for throughput
`FIRST_ROWS_n`	CBO	response time	forces the use of CBO, optimizes for response time for first n rows
`FIRST_ROWS`	CBO and RBO	response time	directs Oracle to minimize response time by any means necessary
`RULE`	RBO	N/A	forces the use of RBO

Behavior of `CHOOSE`:

- If statistics for at least one access table exist, uses `CBO`.
- If no statistics exist, uses `RBO`.

Choosing optimizer mode for a single query. Using Oracle hints it is possible to override Oracle's optimizer mode for a single query in a session. The following list of hints is available:

`OPTIMIZER_MODE` value	hint
`CHOOSE`	`CHOOSE`
`ALL_ROWS`	`ALL_ROWS`
`FIRST_ROWS_n`	`FIRST_ROWS(n)`
`FIRST_ROWS`	`FIRST_ROWS`
`RULE`	`RULE`

Note: See more on hints in sections below.

Indexes in Oracle

Oracle **automatically creates** indexes on all

- Primary keys of relations;
- keys identified by `UNIQUE` constraints in `CREATE TABLE` statements.

Database developers can create additional indexes manually using `CREATE INDEX` command.

CREATE INDEX command. The syntax of `CREATE INDEX` command is as follows [1]:

```
CREATE [UNIQUE] INDEX  <IndexName>
ON <TableName> [<Alias>] ( <Columns> )
```

[1] This is somewhat simplified syntax

Here:

- `UNIQUE` specifies that index keys must be unique.
- `<IndexName>` is the name of the index.
- `<TableName>` is the name of the table which is indexed (you can alias it if needed).
- `<Columns>` is the list of columns which are being indexed.

Example. The following SQL commands create a table, and create two indexes on it.

```
CREATE TABLE Items (
  Receipt INT,
  Item INT,
  Quantity INT,
  PRIMARY KEY(Receipt, Item)
);

CREATE INDEX IDX_ITEM
ON Items(Item);

CREATE INDEX IDX_RECEIPT
ON Items(Receipt);
```

Deleting indexes. To delete indexes use `DROP INDEX` command:

```
DROP INDEX <Name>;
```

Checking existing indexes on your tables. Oracle uses `USER_INDEXES` table to store information about indexes on the tables in user's database. You can view all attributes of `USER_INDEXES` using SQL*Plus command

```
describe USER_INDEXES
```

To view indexes with the tables they refer to issue the following query:

```
SELECT index_name, table_name FROM USER_INDEXES;
```

Optimizer Statistics and their collection and maintenance

Oracle optimizer collects the following information (statistics) about the relational tables in its tablespace.

- Table Statistics (for each table)
  - Number of rows ($T(R)$)

  - Number of blocks ($B(R)$)
  - Average row length[2]

- Column statistics (for individual columns of the tables)
  - Number of distinct values (NDV) ($V(R, A)$)
  - Number of nulls in column
  - Histogram (of data distribution by value)

- Index statistics (for each index)
  - Number of leaf blocks (Oracle uses B+trees)
  - Number of levels
  - Clustering factor [3]

- System statistics
  - I/O performance and utilization
  - CPU performance and utilization

Gathering Statistics. Oracle 10 (and up) automatically gathers statistics on all database objects using a background job (`GATHER_STATS_JOB`). Typically, this job is run every day at night (at the beginning of the maintenance window for the system).

Manual Statistics Gathering. It is possible to "force" Oracle to collect statistics upon user request. Oracle provides a PL/SQL package `DBMS_STATS` which contains a number of stored procedures that collect statistics. The two procedures of interest are `GATHER_INDEX_STATS`, which gathers index statistics for a given database index, and `GATHER_TABLE_STATS`, which gathers table and column statistics for individual tables[4].

GATHER_INDEX_STATS. This procedure gathers statistics for a specified index file. It has two required parameters: `ownname` (index owner/schema) and `indname` (index name), and a number of optional parameters that guide the behavior of the procedure and allow to retrieve gathered statistics. The full list is below:

[2]For fixed-length records, this will be simply row length, or record size. For variable-length records, the average will be applied.

[3]A relation is *clustered* if all records with the same value of an index key are located in as few blocks as possible. It us *not clustered* if records with the same value of an index key are located on as many blocks as possible. Clustering factor essentially quantifies the degree to which the relational table is clustered w.r.t., specific index.

[4]Other `GATHER_...._STATS` procedures exist, but they gather statistics at a much higher granularity level.

Parameter name	meaning
`ownname`	schema of index
`indname`	name of index
`partname`	name of partition (we do not use partitions)
`estimate_percent`	percentage of rows to estimate. Default: `NULL` means compute. range of values: [0.000001, 100].
`stattab`	name of table in which to store the statistics
`statown`	schema of `stattab` (default: `ownname`
`degree`	degree of paralellism, i.e., how many parallel processes to use. default: `NULL` (i.e., serial execution).
`granularity`	granularity of statistics to be gathered. See table below.
`no_invalidate`	do not invalidate dependent cursors if set to `TRUE` (the default behavior is to invalidate them)
`force`	gather statistics even the object (index) is locked

Values of the `granularity` parameter (we should not be worried too much about these values, default, 'ALL' or 'AUTO' will work):

value	meaning
`'ALL'`	gather all statistics
`'AUTO'`	default value: Oracle determines the granularity
`'GLOBAL'`	gathers global statistics
`'GLOBAL AND PARTITION'`	gathers global and partition, but not sub-partition statistics
`'PARTITION'`	gathers partition-level statistics
`'SUBPARTITION'`	gathers sub-partition-level statistics

The most simple way of invoking the index statistics gathering is (assuming the owner of the schema is `'alex'`:

```
DBMS_STATS.GATHER_INDEX_STATS(ownname=>'alex',
                              indname=>'IDX_ITEM');
```

You can create a simple script:

```
BEGIN
DBMS_STATS.GATHER_INDEX_STATS(ownname=>'alex',
                              indname=>'IDX\_ITEM');
END
/
```

and run it from sql*plus command line. (note the syntax for parameter passing).

GATHER_TABLE_STATS. This stored procedure has two mandatory parameters: `ownname` and `tabname`, name of the table, and a number of optional paramters that guide its behavior. Statistics are gathered for the specified table. The full paramter list is:

Parameter name	meaning
ownname	schema of table
tabname	name of table
partname	name of partition (we do not use partitions)
estimate_percent	percentage of rows to estimate. Default: NULL means compute. range of values: [0.000001, 100].
block_sample	If set to TRUE, use random block sampling if set to FALSE (default), use random row sampling
method_opt	specifies "accents" (see table below)
degree	degree of paralellism, i.e., how many parallel processes to use. default: NULL (i.e., serial execution).
granularity	granularity of statistics to be gathered. See table for GATHER_INDEX_STATS.
cascade	if set to 'TRUE' triggers gathering index statistics on the table
stattab	name of table in which to store the statistics
statid	identifier (optional) to associate with statistics in the stats table
statown	schema of stattab (default: ownname
no_invalidate	do not invalidate dependent cursors if set to TRUE (the default behavior is to invalidate them)
force	gather statistics even the object (index) is locked

method_opt parameters accepts the following values:

- FOR ALL [INDEXED| HIDDEN] COLUMN [<sizeClause>]
- FOR COLUMNS [<sizeClause>] <columnName>|<attribute> [<sizeClause>], (<columnName>|<attribute> [<sizeClause>])*

<sizeClause> controls how big a histogram will be produced. Its format is SIZE (<number>|REPEAT|AUTO|SKEWONLY). Here:

value	meaning
<number>	number of histogram buckets (1...254)
REPEAT	collect histograms only if a column already has a histogram
AUTO	Let Oracle determine which histograms to collect based on data distribution and worlkload
'SKEWONLY'	Oracle determines which histograms to collect based the data distribution

Here is an example of a statistics gathering command:

```
DBMS_STATS.GATHER_TABLE_STATS(ownname => 'alex',
                              tabname => 'ITEMS',
                              method_opt => 'FOR ALL COLUMNS SIZE 20',
                              cascade => 'TRUE',
                              stattab => 'alexStats',
                              statid  => 'item01' );
```

This call asks Oracle to gather statistics for all columns of alex.ITEMS table, collect column statistics in the form of the 20-bucket histograms for all columns of the table, collect index statistics for all indexes of the table and put all collected statistics into the alexStats table under item01 tag.

Index statistics table. You can create your own table for storing index statistics using the CREATE_STAT_TABLE stored procedure from the DBMS_STATS package. The procedure takes two mandatory parameters, ownname and statttab, the name of schema and the name of the table respectively.

For example, to create stats table use the following script:

```
BEGIN
DBMS_STATS.CREATE_STAT_TABLE('alex', 'alexStats');

END
/
```

or, directly from sql*plus type:

```
EXEC DBMS_STATS.CREATE_STAT_TABLE('alex', 'alexStats');
```

You can use `DBMS_STATS.DROP_STAT_TABLE(ownname, stattab)` command to drop the stats table.

Optimizer Hints

To improve/alter behavior of the Oracle query processor on specific queries we use optimizer hints.

Optimizer hints syntax and use. Optimizer hints are applicable to the following four SQL commands:

- `SELECT`
- `DELETE`
- `UPDATE`
- `INSERT`

Optimizer hints are inserted directly into the query in the form of a special comment that immediately follows the `INSERT|DELETE|UPDATE|SELECT` keyword. The syntax of the hints is:

```
(INSERT|DELETE|UPDATE|SELECT) /*+ hint [text] [hint [text]]* */  ...
```

or

```
(INSERT|DELETE|UPDATE|SELECT) --+ hint [text] [hint [text]]*
 ...
```

Here,

- `"+"` is a signal to Oracle to look for optimizer hints in the comment. There should be no whitespace between the comment start (`"/*"` or `"--"` and `"+"`.
- `hint` entries represent the actual optimizer hints.
- `text` entries represent any other texts that is treated as regular comments.

Example. Here is an example of an optimizer hint, which invokes a rule-based optimizer for the query.

```
SELECT /*+ RULE */ *
FROM Goods g, Items i
WHERE g.id = i.item and g.price < 7;
```

Types of optimizer hints. Oracle supports the following types of optimizer hints:

- Hints for optimization approach and goal. (See "Selection of Optimizer approach and goal" section). These hints override the session optimizer behavior to process current query using a different appraoch/goal.
- Hints for access paths. These hints specify how Oracle will access data from database tables.
- Hints for query transformations. These hints guide the work of the query rewriter by allowing/disallowing certain query transformations.
- Hints for join orders. These hints determine in which order multiple tables are joined.
- Hints for join operations. These hints specify which join algorithm to use for specific joins in the query.
- Additional hints. These mostly deal with operation-specific issues (e.g., hints, specific to `INSERT` operation).

Hints for optimization approach and goal. These hints, `ALL_ROWS, FIRST_ROWS(n), CHOOSE, RULE` are described above.

Hints for access paths. An access path is the method of access to/retrieval of the data from database tables. Oracle supports multiple access paths to relational tables, and provides a hint for each of them:

Hint syntax	Access path	Comment
`FULL(<TableName>)`	Full scan	selects full table scan to access `<TableName>`
`ROWID(<TableName>)`	Rowid scan	typically used for small outputs
`INDEX(<TableName> <Index> <Index>*)`	Index scan	use index(es) to access the table
`INDEX_ASC(<TableName> <Index> <Index>*)`	Index scan	explicitly tells to scan index(es) in ascending order
`INDEX_DESC(<TableName> <Index> <Index>*)`	Index scan	explicitly tells to scan index(es) in descending order
`INDEX_JOIN(<TableName> <Index> <Index>*)`	Index scan	use join of indexes to access table
`INDEX_FFS(<TableName> <Index> <Index>*)`	Fast index scan	use fast full index scan
`NO_INDEX(<TableName> <Index>*)`		prohibits the use of specified indexes.

Notes: The default behavior of `INDEX` and `INDEX_ASC` are the same. If no indexes are specified in `NO_INDEX` hint, Oracle essentially is forced to do a full table scan. Fast Full Index Scan

Hints for query transformations. The following hints for query transformations are used in Oracle:

Hint syntax	Explanation
USE_CONCAT	forces OR condidtions in WHERE clauses to be transformed into UNION ALL operations
NO_EXPAND	prevents CBO from considering OR-expansion of IN-lists
EXPAND_GSET_TO_UNION	transform grouping queries to unions using UNION ALL
MERGE(<Name>)	allows optimizer to merge the view's query (nested query) into accessing statement
NO_MERGE[(<Name>)]	disallows merging view's query (nested query) into accessing statement
STAR_TRANSFORMATION	select the best plan that uses the star transformation
FACT(<TableName>	specifies the "fact" table for the star transformation
NO_FACT(<TableName>	specifies that the table is **not** the "fact table" in the star transformation

Note: OR-expansion of IN-lists replaces the <Attribute> IN <Query> expression with an OR of <Attribute> = <Value> for each <Value> returned by <Query>.

Note: The star transformation applies to a situation where multiple (typically, small) tables are joined with a single (very) large table, i.e., when one, large, table is found in all join conditions in a query. The transformation adds subqueries to the join query, which may potentially allow for index-based access path to be more efficient than a full table scan for the large table. It is called star transformation because a typcial example of its use comes from data warehouses employing the so-called star schema, i.e., a database schema with one very large "fact table" and many small "dimension tables" connected to the fact table via foreign keys.

Hints for join orders. Oracle has two hints to control join order:

Hint syntax	Explanation
ORDERED	join tables in the order in which they appear in the FROM clause
STAR	join the largest table last (using nested loops join)

Hints for join operations. Oracle has three main methods for joining tables: nested loops join, sort-based join and hash-based join.

Hint syntax	Explanation
USE_NL(<Table> <Table>*)	use nested loops join to join all mentioned tables
USE_MERGE(<Table> <Table>*)	use sort-based join to join all mentioned tables
USE_HASH(<Table> <Table>*)	use hash-based join to join all mentioned tables
LEADING(<Table>)	specifies the first table in join order
HASH_AJ, MERGE_AJ, NL_AJ	specifies the type of join operation in the NOT IN subqueries
HASH_SJ, MERGE_SJ, NL_SJ	specifies the type of join operation in the EXISTS subqueries

Note: Only one LEADING table may be specified. ORDERED hint takes precedence over LEADING hint.

Plan Explanation

Oracle allows users to review query plans for any SELECT, DELETE, UPDATE or INSERT statement. The SQL command for producing a plan is EXPLAIN PLAN, and is syntax is as follows:

```
EXPLAIN PLAN [INTO <TableName>] FOR <SQLStatement>;
```

Here, <TableName> is the name of the table into which the plan details are stored. If table name is not included, Oracle stores the plan in the table called plan_table.

The schema of plan_table can be discovered via DESCRIBE command:

```
SQL> describe plan_table
Name                                     Null?    Type
-----------------------
----------------- -------- ----------------------------
STATEMENT_ID                                      VARCHAR2(30)
PLAN_ID                                           NUMBER
TIMESTAMP                                         DATE
REMARKS                                           VARCHAR2(4000)
OPERATION                                         VARCHAR2(30)
OPTIONS                                           VARCHAR2(255)
OBJECT_NODE                                       VARCHAR2(128)
OBJECT_OWNER                                      VARCHAR2(30)
OBJECT_NAME                                       VARCHAR2(30)
OBJECT_ALIAS                                      VARCHAR2(65)
OBJECT_INSTANCE                                   NUMBER(38)
OBJECT_TYPE                                       VARCHAR2(30)
OPTIMIZER                                         VARCHAR2(255)
SEARCH_COLUMNS                                    NUMBER
ID                                                NUMBER(38)
PARENT_ID                                         NUMBER(38)
DEPTH                                             NUMBER(38)
POSITION                                          NUMBER(38)
COST                                              NUMBER(38)
CARDINALITY                                       NUMBER(38)
BYTES                                             NUMBER(38)
OTHER_TAG                                         VARCHAR2(255)
PARTITION_START                                   VARCHAR2(255)
PARTITION_STOP                                    VARCHAR2(255)
PARTITION_ID                                      NUMBER(38)
OTHER                                             LONG
OTHER_XML                                         CLOB
DISTRIBUTION                                      VARCHAR2(30)
CPU_COST                                          NUMBER(38)
IO_COST                                           NUMBER(38)
TEMP_SPACE                                        NUMBER(38)
ACCESS_PREDICATES                                 VARCHAR2(4000)
FILTER_PREDICATES                                 VARCHAR2(4000)
PROJECTION                                        VARCHAR2(4000)
TIME                                              NUMBER(38)
QBLOCK_NAME                                       VARCHAR2(30)
```

Of these, the following columns are of specific interest:

Column	Explanation
OPERATION	Operation being performed (see list below)
OPTIONS	modifications/options of the operation
OBJECT_NAME	table (or other object) on which the operation is performed
ID	id of the oepration in the plan
PARENT_ID	pointer to the parent (in the query plan) of the operation

Possible values of the OPERATION attribute are:

```
DELETE STATEMENT      AND-EQUAL       DOMAIN INDEX      HASH JOIN
INSERT STATEMENT      CONNECT BY      FILTER            MERGE JOIN
```

```
SELECT STATEMENT     CONCATENATION   FIRST ROW      NESTED LOOPS
UPDATE STATEMENT     COUNT           FOR UPDATE     UNION
INLIST ITERATOR      INDEX           PARTITION      INTERSECTION
TABLE ACCESS         REMOTE          SEQUENCE       MINUS
                     SORT            VIEW
```

To view the plan from the `plan_table`, run the following command:

```
select
  substr (lpad(' ', level-1) || operation || ' (' || options || ')',1,30 ) "Operation",
  object_name                                                          "Object"
from
  plan_table
start with id = 0
connect by prior id=parent_id;
```

For example,

```
SQL>  explain plan for
  2  select receipt, food, flavor, price
  3  from goods g, items i
  4   where i.item = g.gid and price < 5;

Explained.

SQL> select
  substr (lpad(' ', level-1) || operation || ' (' || options || ')',1,30 ) "Operation",
  object_name                                                          "Object"  2
from
  plan_table
start with id = 0
connect by prior id=parent_id;

Operation                      Object
------------------------------ ------------------------------
SELECT STATEMENT ()
 MERGE JOIN ()
  TABLE ACCESS (BY INDEX ROWID GOODS
   INDEX (FULL SCAN)           SYS_C0027499
  SORT (JOIN)
   TABLE ACCESS (FULL)         ITEMS

6 rows selected.

SQL> explain plan for
  2  select /*+  rule  */  receipt, food, flavor, price
  3  from goods g, items i
  4  where i.item = g.gid and price < 5;

Explained.

SQL> select
  substr (lpad(' ', level-1) || operation || ' (' || options || ')',1,30 ) "Operation",
  object_name                                                          "  2  Object"
from
  plan_table
start with id = 0
connect by prior id=  3  parent_id;
  4    5    6    7
Operation                      Object
```

```
------------------------------ ------------------------------
SELECT STATEMENT ()
 NESTED LOOPS ()
  TABLE ACCESS (FULL)           ITEMS
  TABLE ACCESS (BY INDEX ROWID GOODS
   INDEX (UNIQUE SCAN)          SYS_C0027499
```

Timing

sql*plus has a `set timing on` and `set timing off` pair of commands that allow users to collect information about the running time of the queries.

```
SQL> select count(*)
from goods g, items i, receipts r, customers c
    where i.item = g.gid and
          g.price < 5 and
          c.cid = r.customer and
          i.r  2  eceipt = r.rnumber;
  3    4    5    6
  COUNT(*)
----------
       464

Elapsed: 00:00:00.03
```