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 explicit query execution hints to the query compiler.

Selection of Optimizer 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.

Optimizing 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.

<table>
<thead>
<tr>
<th>Value</th>
<th>Optimizer</th>
<th>Goal</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHOOSE</td>
<td>selected by Oracle</td>
<td>throughput</td>
<td>default value. Lets Oracle select which optimizer to choose</td>
</tr>
<tr>
<td>ALL_ROWS</td>
<td>CBO</td>
<td>throughput</td>
<td>forces the use of CBO, optimizes for throughput</td>
</tr>
<tr>
<td>FIRST_ROWS_n</td>
<td>CBO</td>
<td>response time</td>
<td>forces the use of CBO, optimizes for response time for first n rows</td>
</tr>
<tr>
<td>FIRST_ROWS</td>
<td>CBO and RBO</td>
<td>response time</td>
<td>directs Oracle to minimize response time by any means necessary</td>
</tr>
<tr>
<td>RULE</td>
<td>RBO</td>
<td>N/A</td>
<td>forces the use of RBO</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>OPTIMIZER_MODE value</th>
<th>hint</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHOOSE</td>
<td>CHOOSE</td>
</tr>
<tr>
<td>ALL_ROWS</td>
<td>ALL_ROWS</td>
</tr>
<tr>
<td>FIRST_ROWS_n</td>
<td>FIRST_ROWS (n)</td>
</tr>
<tr>
<td>FIRST_ROWS</td>
<td>FIRST_ROWS</td>
</tr>
<tr>
<td>RULE</td>
<td>RULE</td>
</tr>
</tbody>
</table>

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> )
```

¹This 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.

```sql
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: \texttt{ownname} (index owner/schema) and \texttt{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 is 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 parallelism, i.e., how many parallel processes to use. default: NULL (i.e., serial execution).
granularity | granularity of statistics to be gathered. See table below.
nocall | 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):

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
</table>
| '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'):

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

You can create a simple script:

```sql
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 parameters that guide its behavior. Statistics are gathered for the specified table. The full parameter list is:
Parameter name | meaning
--- | ---
ownname | schema of table
tablename | 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 parallelism, 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:

<table>
<thead>
<tr>
<th>value</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;number&gt;</td>
<td>number of histogram buckets (1...254)</td>
</tr>
<tr>
<td>REPEAT</td>
<td>collect histograms only if a column already has a histogram</td>
</tr>
<tr>
<td>AUTO</td>
<td>Let Oracle determine which histograms to collect based on data distribution and workload</td>
</tr>
<tr>
<td>‘SKEWONLY’</td>
<td>Oracle determines which histograms to collect based on the data distribution</td>
</tr>
</tbody>
</table>

Here is an example of a statistics gathering command:

```sql
DBMS_STATS.GATHER_TABLE_STATS(ownname => 'alex',
    tablename => '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 statsttab, 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 approach/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:

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

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

<table>
<thead>
<tr>
<th>Hint syntax</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORDERED</td>
<td>join tables in the order in which they appear in the FROM clause</td>
</tr>
<tr>
<td>STAR</td>
<td>join the largest table last (using nested loops join)</td>
</tr>
</tbody>
</table>

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

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

**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 its 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:

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

Possible values of the OPERATION attribute are:

- DELETE STATEMENT
- AND-EQUAL
- DOMAIN
- INDEX
- HASH
- JOIN
- INSERT STATEMENT
- CONNECT BY
- FILTER
- MERGE
- JOIN
To view the plan from the \textit{plan_table}, run the following command:

\begin{verbatim}
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;
\end{verbatim}

For example,

\begin{verbatim}
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 
  2  substr (lpad(' ', level-1) || operation || ' (' || options || ')',1,30 ) "Operation",
  3    object_name "Object"
from 
  4   plan_table
start with id = 0
connect by prior id=parent_id;
\end{verbatim}

\begin{verbatim}
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.
\end{verbatim}

\begin{verbatim}
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 
  2  substr (lpad(' ', level-1) || operation || ' (' || options || ')',1,30 ) "Operation",
  3    object_name "Object"
from 
  4   plan_table
start with id = 0
connect by prior id=parent_id;
\end{verbatim}

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

Elapsed: 00:00:00.03
```