12

Configuring Options for Optimizer Statistics Gathering

In Oracle Database, **optimizer statistics collection** is the gathering of optimizer statistics for database objects, including fixed objects.

The database can collect optimizer statistics automatically. You can also collect them manually using the `DBMS_STATS` package.

Purpose of Optimizer Statistics Collection

The contents of tables and associated indexes change frequently, which can lead the optimizer to choose suboptimal execution plan for queries. To avoid potential performance issues, statistics must be kept current.

To minimize DBA involvement, Oracle Database automatically gathers optimizer statistics at various times. Some automatic options are configurable, such enabling AutoTask to run `DBMS_STATS`.

User Interfaces for Optimizer Statistics Management

You can manage optimizer statistics either through Oracle Enterprise Manager Cloud Control (Cloud Control) or using PL/SQL on the command line.

Graphical Interface for Optimizer Statistics Management

The Manage Optimizer Statistics page in Cloud Control is a GUI that enables you to manage optimizer statistics.

Accessing the Database Home Page in Cloud Control

Oracle Enterprise Manager Cloud Control enables you to manage multiple databases within a single GUI-based framework.

To access a database home page using Cloud Control:

1. Log in to Cloud Control with the appropriate credentials.
2. Under the **Targets** menu, select **Databases**.
3. In the list of database targets, select the target for the Oracle Database instance that you want to administer.
4. If prompted for database credentials, then enter the minimum credentials necessary for the tasks you intend to perform.

> **See Also:**
>
> Cloud Control online help

Accessing the Optimizer Statistics Console

You can perform most necessary tasks relating to optimizer statistics through pages linked to by the Optimizer Statistics Console page.

To manage optimizer statistics using Cloud Control:

1. In Cloud Control, access the Database Home page.
2. From the **Performance** menu, select **SQL**, then **Optimizer Statistics**.

   The Optimizer Statistics Console appears.

> **See Also:**
>
> Online Help for Oracle Enterprise Manager Cloud Control

Command-Line Interface for Optimizer Statistics Management

The `DBMS_STATS` package performs most optimizer statistics tasks.

To enable and disable automatic statistics gathering, use the `DBMS_AUTO_TASK_ADMIN` PL/SQL package.

> **See Also:**
>
> *Oracle Database PL/SQL Packages and Types Reference* to learn how to use `DBMS_STATS` and `DBMS_AUTO_TASK_ADMIN`

Setting Optimizer Statistics Preferences

This topic explains how to set optimizer statistics defaults using `DBMS_STATS.SET_*_PREFS` procedures.

About Optimizer Statistics Preferences

The **optimizer statistics preferences** set the default values of the parameters used by automatic statistics collection and the `DBMS_STATS` statistics gathering procedures.

Purpose of Optimizer Statistics Preferences

Preferences enable you to maintain optimizer statistics automatically when some objects require settings that differ from the default.

Preferences give you more granular control over how Oracle Database gathers statistics. You can set optimizer statistics preferences at the following levels:

- Table
- Schema
- Database (all tables)
- Global (tables with no preferences and any tables created in the future)

The `DBMS_STATS` procedures for setting preferences have names of the form `SET_*_PREFS`.

Examples of Statistics Preferences

Set preferences using the `pname` parameter of the `SET_*_PREFS` procedures.

Preferences that you can set include, but are not limited to, the following:

- `ESTIMATE_PERCENT`

  This preference determines the percentage of rows to estimate.

- `CONCURRENT`

  This preference determines whether the database gathers statistics concurrently on multiple objects, or serially, one object at a time.

- `STALE_PERCENT`

  This preference determines the percentage of rows in a table that must change before the database deems the statistics stale and in need of regathering.

- `AUTO_STAT_EXTENSIONS`

  When set to the non-default value of `ON`, this preference enables a SQL plan directive to trigger the creation of column group statistics based on usage of columns in the predicates in the workload.

- `INCREMENTAL`

  This preference determines whether the database maintains the global statistics of a partitioned table without performing a full table scan. Possible values are `TRUE` and `FALSE`.

  For example, by the default setting for `INCREMENTAL` is `FALSE`. You can set `INCREMENTAL` to `TRUE` for a range-partitioned table when the last few partitions are updated. Also, when performing a partition exchange operation on a nonpartitioned table, Oracle recommends that you set `INCREMENTAL` to `TRUE` and `INCREMENTAL_LEVEL` to `TABLE`. With these settings, `DBMS_STATS` gathers table-level synopses on this table.

- `INCREMENTAL_LEVEL`

  This preference controls what synopses to collect when `INCREMENTAL` preference is set to `TRUE`. It takes two values: `TABLE` or `PARTITION`.

- `APPROXIMATE_NDV_ALGORITHM`

  This preference controls which algorithm to use when calculating the number of distinct values for partitioned tables using incremental statistics.

- `ROOT_TRIGGER_PDB`

  This preference controls whether to accept or reject the statistics gathering triggered from an application root in a CDB.

By default, when gathering statistics for a metadata-linked table in the application root, if the statistics the application PDB are stale, the database does *not* trigger statistics gathering on the application PDB. When set to `TRUE`, `ROOT_TRIGGER_PDB` triggers statistics gathering on the application PDB, and then derives the global statistics in the application root.

> **See Also:**
>
> *Oracle Database PL/SQL Packages and Types Reference* to learn about the `DBMS_STATS` procedures for setting optimizer statistics preferences

DBMS_STATS Procedures for Setting Statistics Preferences

The `DBMS_STATS.SET_*_PREFS` procedures change the defaults of parameters used by the `DBMS_STATS.GATHER_*_STATS` procedures. To query the current preferences, use the `DBMS_STATS.GET_PREFS` function.

When setting statistics preferences, the order of precedence is:

1. Table preference (set for a specific table, all tables in a schema, or all tables in the database)
2. Global preference
3. Default preference

The following table summarizes the relevant `DBMS_STATS` procedures.

Table 12-1 DBMS_STATS Procedures for Setting Optimizer Statistics Preferences

Procedure	Scope
`SET_TABLE_PREFS`	Specified table only.
`SET_SCHEMA_PREFS`	All existing tables in the specified schema. This procedure calls `SET_TABLE_PREFS` for each table in the specified schema. Calling `SET_SCHEMA_PREFS` does not affect any new tables created after it has been run. New tables use the `GLOBAL_PREF` values for all parameters.
`SET_DATABASE_PREFS`	All user-defined schemas in the database. You can include system-owned schemas such as `SYS` and `SYSTEM` by setting the `ADD_SYS` parameter to `true`. This procedure calls `SET_TABLE_PREFS` for each table in the specified schema. Calling `SET_DATABASE_PREFS` does not affect any new objects created after it has been run. New objects use the `GLOBAL_PREF` values for all parameters.

Table 12-1 (Cont.) DBMS_STATS Procedures for Setting Optimizer Statistics Preferences

Procedure	Scope
SET_GLOBAL_PREFS	Any table that does not have an existing table preference. All parameters default to the global setting unless a table preference is set or the parameter is explicitly set in the DBMS_STATS.GATHER_*_STATS statement. Changes made by SET_GLOBAL_PREFS affect any new objects created after it runs. New objects use the SET_GLOBAL_PREFS values for all parameters. With SET_GLOBAL_PREFS, you can set a default value for the parameter AUTOSTATS_TARGET. This additional parameter controls which objects the automatic statistic gathering job running in the nightly maintenance window affects. Possible values for AUTOSTATS_TARGET are ALL, ORACLE, and AUTO (default). You can only set the CONCURRENT preference at the global level. You cannot set the preference INCREMENTAL_LEVEL using SET_GLOBAL_PREFS.

See Also:

- "About Concurrent Statistics Gathering"
- *Oracle Database PL/SQL Packages and Types Reference* to learn about the DBMS_STATS procedures for setting optimizer statistics preferences

Statistics Preference Overrides

The preference_overrides_parameter statistics preference determines whether, when gathering optimizer statistics, to override the input value of a parameter with the statistics preference. In this way, you control when the database honors a parameter value passed to the statistics gathering procedures.

When preference_overrides_parameter is set to FALSE (default), the input values for statistics gathering procedures are honored. When set to TRUE, the input values are ignored.

Set the preference_overrides_parameter preference using the SET_TABLE_PREFS, SET_SCHEMA_PREFS, or SET_GLOBAL_PREFS procedures in DBMS_STATS. Regardless of whether preference_overrides_parameter is set, the database uses the same order of precedence for setting statistics:

1. Table preference (set for a specific table, all tables in a schema, or all tables in the database)
2. Global preference
3. Default preference

Example 12-1 Overriding Statistics Preferences at the Table Level

In this example, legacy scripts set estimate_percent explicitly rather than using the recommended AUTO_SAMPLE_SIZE. Your goal is to prevent users from using these scripts to set preferences on the sh.costs table.

Table 12-2 Overriding Statistics Preferences at the Table Level

Action	Description

`SQL> SELECT DBMS_STATS.GET_PREFS ('estimate_percent', 'sh','costs') AS "STAT_PREFS" FROM DUAL;`
 	
`STAT_PREFS`
`----------`
`DBMS_STATS.AUTO_SAMPLE_SIZE`
	No preference for `estimate_percent` is set for `sh.costs` or at the global level, so the preference defaults to `AUTO_SAMPLE_SIZE`.

`SQL> EXEC DBMS_STATS.SET_TABLE_PREFS ('sh', 'costs', 'preference_overrides_parameter', 'true');`
 	
`PL/SQL procedure successfully completed.`
	By default, Oracle Database accepts preferences that are passed to the `GATHER_*_STATS` procedures. To override these parameters, you use `SET_TABLE_PREFS` to set the `preference_overrides_parameter` preference to `true` for the `costs` table only.

`SQL> EXEC DBMS_STATS.GATHER_TABLE_STATS ('sh', 'costs', estimate_percent=>100);`
 	
`PL/SQL procedure successfully completed.`
	You attempt to set `estimate_percent` to `100` when gathering statistics for `sh.costs`. However, because `preference_overrides_parameter` is `true` for this table, Oracle Database does not honor the `estimate_percent=>100`setting. Instead, the database gathers statistics using `AUTO_SAMPLE_SIZE`, which is the default.

Example 12-2 Overriding Statistics Preferences at the Global Level

In this example, you set `estimate_percent` to `5` at the global level, which means that this preference applies to every table in the database that does *not* have a table preference set. You then set an override on the `sh.sales` table, which does not have a table-level preference set, to prevent users from overriding the global setting in their scripts.

Table 12-3 Overriding Statistics Preferences at the Global Level

Action	Description

`SQL> SELECT DBMS_STATS.GET_PREFS ('estimate_percent', 'sh','sales') AS "STAT_PREFS" FROM DUAL;`
 	
`STAT_PREFS`
`----------`
`DBMS_STATS.AUTO_SAMPLE_SIZE`
	No preference for `estimate_percent` is set for `sh.sales` or at the global level, so the preference defaults to `AUTO_SAMPLE_SIZE`.

Table 12-3 (Cont.) Overriding Statistics Preferences at the Global Level

Action	Description

`SQL> EXEC DBMS_STATS.SET_GLOBAL_PREFS ('estimate_percent', '5');`
 	
`PL/SQL procedure successfully completed.`
	You use the `SET_GLOBAL_PREFS` procedure to set the `estimate_percent` preference to `5` for every table in the database that does not have a table preference set.

`SQL> SELECT DBMS_STATS.GET_PREFS ('estimate_percent', 'sh','sales') AS "STAT_PREFS" FROM DUAL;`
 	
`STAT_PREFS`
`----------`
`5`
	Because `sh.sales` does not have a preference set, the global setting applies to this table. A query of the preferences for `sh.sales` now shows that the `estimate_percent` setting is `5`, which is the global setting.

`SQL> EXEC DBMS_STATS.SET_TABLE_PREFS ('sh', 'sales', 'preference_overrides_parameter', 'true');`
 	
`PL/SQL procedure successfully completed.`
	You use `SET_TABLE_PREFS` to set the `preference_overrides_parameter` preference to `true` for the `sh.sales` table only.

`SQL> EXEC DBMS_STATS.GATHER_TABLE_STATS ('sh', 'sales', estimate_percent=>10);`
 	
`PL/SQL procedure successfully completed.`
	You attempt to set `estimate_percent` to `10` when gathering statistics for `sh.sales`. However, because `preference_overrides_parameter` is `true` for the `sales` table, and because a global preference is defined, Oracle Database actually gathers statistics using the global setting of `5`.

> **See Also:**
>
> *Oracle Database PL/SQL Packages and Types Reference* to learn about the `DBMS_STATS` procedures for setting optimizer statistics

Setting Statistics Preferences: Example

This example illustrates the relationship between `SET_TABLE_PREFS`, `SET_SCHEMA_STATS`, and `SET_DATABASE_PREFS`.

Table 12-4 Changing Preferences for Statistics Gathering Procedures

Action	Description

`SQL> SELECT DBMS_STATS.GET_PREFS ('incremental', 'sh','costs') AS "STAT_PREFS" FROM DUAL;`
 	
`STAT_PREFS`
`----------`
`TRUE`
	You query the `INCREMENTAL` preference for `costs` and determine that it is set to `true`.

`SQL> EXEC DBMS_STATS.SET_TABLE_PREFS ('sh', 'costs', 'incremental', 'false');`
 	
`PL/SQL procedure successfully completed.`
	You use `SET_TABLE_PREFS` to set the `INCREMENTAL` preference to `false` for the `costs` table only.

`SQL> SELECT DBMS_STATS.GET_PREFS ('incremental', 'sh', 'costs') AS "STAT_PREFS" FROM DUAL;`
 	
`STAT_PREFS`
`----------`
`FALSE`
	You query the `INCREMENTAL` preference for `costs` and confirm that it is set to `false`.

`SQL> EXEC DBMS_STATS.SET_SCHEMA_PREFS ('sh', 'incremental', 'true');`
 	
`PL/SQL procedure successfully completed.`
	You use `SET_SCHEMA_PREFS` to set the `INCREMENTAL` preference to `true` for every table in the `sh` schema, including `costs`.

`SQL> SELECT DBMS_STATS.GET_PREFS ('incremental', 'sh', 'costs') AS "STAT_PREFS" FROM DUAL;`
 	
`STAT_PREFS`
`----------`
`TRUE`
	You query the `INCREMENTAL` preference for `costs` and confirm that it is set to `true`.

`SQL> EXEC DBMS_STATS.SET_DATABASE_PREFS ('incremental', 'false');`
 	
`PL/SQL procedure successfully completed.`
	You use `SET_DATABASE_PREFS` to set the `INCREMENTAL` preference for all tables in all user-defined schemas to `false`.

`SQL> SELECT DBMS_STATS.GET_PREFS ('incremental', 'sh', 'costs') AS "STAT_PREFS" FROM DUAL;`
 	
`STAT_PREFS`
`----------`
`FALSE`
	You query the `INCREMENTAL` preference for `costs` and confirm that it is set to `false`.

Setting Global Optimizer Statistics Preferences Using Cloud Control

A global preference applies to any object in the database that does *not* have an existing table preference. You can set optimizer statistics preferences at the global level using Cloud Control.

To set global optimizer statistics preferences using Cloud Control:

1. In Cloud Control, access the Database Home page.
2. From the **Performance** menu, select **SQL**, then **Optimizer Statistics**.

   The Optimizer Statistics Console appears.
3. Click **Global Statistics Gathering Options**.

   The Global Statistics Gathering Options page appears.
4. Make your desired changes, and click **Apply**.

> **See Also:**
>
> Online Help for Oracle Enterprise Manager Cloud Control

Setting Object-Level Optimizer Statistics Preferences Using Cloud Control

You can set optimizer statistics preferences at the database, schema, and table level using Cloud Control.

To set object-level optimizer statistics preferences using Cloud Control:

1. In Cloud Control, access the Database Home page.
2. From the **Performance** menu, select **SQL**, then **Optimizer Statistics**.

   The Optimizer Statistics Console appears.
3. Click **Object Level Statistics Gathering Preferences**.

   The Object Level Statistics Gathering Preferences page appears.
4. To modify table preferences for a table that has preferences set at the table level, do the following (otherwise, skip to the next step):
   a. Enter values in **Schema** and **Table Name**. Leave **Table Name** blank to see all tables in the schema.

      The page refreshes with the table names.
   b. Select the desired tables and click **Edit Preferences**.

      The General subpage of the Edit Preferences page appears.
   c. Change preferences as needed and click **Apply**.
5. To set preferences for a table that does *not* have preferences set at the table level, do the following (otherwise, skip to the next step):
   a. Click **Add Table Preferences**.

      The General subpage of the Add Table Preferences page appears.
   b. In **Table Name**, enter the schema and table name.

c. Change preferences as needed and click **OK**.

6. To set preferences for a schema, do the following:
   a. Click **Set Schema Tables Preferences**.

      The General subpage of the Edit Schema Preferences page appears.
   b. In **Schema**, enter the schema name.
   c. Change preferences as needed and click **OK**.

> **See Also:**
>
> Online Help for Oracle Enterprise Manager Cloud Control

Setting Optimizer Statistics Preferences from the Command Line

If you do not use Cloud Control to set optimizer statistics preferences, then you can invoke the `DBMS_STATS` procedures from the command line.

Prerequisites

This task has the following prerequisites:

- To set the global or database preferences, you must have `SYSDBA` privileges, or both `ANALYZE ANY DICTIONARY` and `ANALYZE ANY` system privileges.
- To set schema preferences, you must connect as owner, or have `SYSDBA` privileges, or have the `ANALYZE ANY` system privilege.
- To set table preferences, you must connect as owner of the table or have the `ANALYZE ANY` system privilege.

To set optimizer statistics preferences from the command line:

1. In SQL*Plus or SQL Developer, log in to the database as a user with the necessary privileges.
2. Optionally, call the `DBMS_STATS.GET_PREFS` procedure to see preferences set at the object level, or at the global level if a specific table is not set.

   For example, obtain the `STALE_PERCENT` parameter setting for the `sh.sales` table as follows:

   ```
   SELECT DBMS_STATS.GET_PREFS('STALE_PERCENT', 'SH', 'SALES')
   FROM   DUAL;
   ```

   Note that `FROM DUAL` is no longer necessary for queries requiring no table access.
3. Execute the appropriate procedure from Table 12-1, specifying the following parameters:
   - `ownname` - Set schema name (`SET_TAB_PREFS` and `SET_SCHEMA_PREFS` only)
   - `tabname` - Set table name (`SET_TAB_PREFS` only)
   - `pname` - Set parameter name
   - `pvalue` - Set parameter value

- `add_sys` - Include system tables (optional, `SET_DATABASE_PREFS` only)

The following example specifies that 13% of rows in `sh.sales` must change before the statistics on that table are considered stale:

```
EXEC DBMS_STATS.SET_TABLE_PREFS('SH', 'SALES', 'STALE_PERCENT', '13');
```

4. Optionally, query the `*_TAB_STAT_PREFS` view to confirm the change.

   For example, query `DBA_TAB_STAT_PREFS` as follows:

```
COL OWNER FORMAT a5
COL TABLE_NAME FORMAT a15
COL PREFERENCE_NAME FORMAT a20
COL PREFERENCE_VALUE FORMAT a30
SELECT * FROM DBA_TAB_STAT_PREFS;
```

   Sample output appears as follows:

```
OWNER TABLE_NAME      PREFERENCE_NAME      PREFERENCE_VALUE
----- --------------- -------------------- ------------------------------
OE    CUSTOMERS       NO_INVALIDATE        DBMS_STATS.AUTO_INVALIDATE
SH    SALES           STALE_PERCENT        13
```

> **See Also:**
>
> *Oracle Database PL/SQL Packages and Types Reference* for descriptions of the parameter names and values for program units

Configuring Options for Dynamic Statistics

Dynamic statistics are an optimization technique in which the database uses recursive SQL to scan a small random sample of the blocks in a table.

The sample scan estimate predicate selectivities. Using these estimates, the database determines better default statistics for unanalyzed segments, and verifies its estimates. By default, when optimizer statistics are missing, stale, or insufficient, dynamic statistics automatically run recursive SQL during parsing to scan a small random sample of table blocks.

About Dynamic Statistics Levels

The **dynamic statistics level** controls both when the database gathers dynamic statistics, and the size of the sample that the optimizer uses to gather the statistics.

Set the dynamic statistics level using either the `OPTIMIZER_DYNAMIC_SAMPLING` initialization parameter or a statement hint.

> **Note:**
>
> Dynamic statistics were called *dynamic sampling* in releases earlier than Oracle Database 12c Release 1 (12.1).

The following table describes the levels for dynamic statistics. Note the following:

- If dynamic statistics are enabled, then the database may choose to use dynamic statistics when a SQL statement uses parallel execution.
- If `OPTIMIZER_ADAPTIVE_STATISTICS` is `TRUE`, then the optimizer uses dynamic statistics when relevant SQL plan directives exist. The database maintains the resulting statistics in the SQL plan directives store, making them available to other queries.

The number of blocks sampled depends on the level.

Table 12-5 Dynamic Statistics Levels

Level	When the Optimizer Uses Dynamic Statistics	Sample Size (Blocks)
0	Do not use dynamic statistics.	n/a
1	Use dynamic statistics for all tables that do not have statistics, but only if the following criteria are met: • At least one nonpartitioned table in the query does not have statistics. • This table has no indexes. • This table has more blocks than the number of blocks that would be used for dynamic statistics of this table.	32
2	Use dynamic statistics if at least one table in the statement has no statistics. This is the default value.	32
3	Use dynamic statistics if any of the following conditions is true: • At least one table in the statement has no statistics. • The statement has one or more expressions used in the `WHERE` clause predicates, for example, `WHERE SUBSTR(CUSTLASTNAME,1,3).`	32
4	Use dynamic statistics if any of the following conditions is true: • At least one table in the statement has no statistics. • The statement has one or more expressions used in the `WHERE` clause predicates, for example, `WHERE SUBSTR(CUSTLASTNAME,1,3).` • The statement uses complex predicates (an `OR` or `AND` operator between multiple predicates on the same table).	32
5	The criteria are identical to level 4, but the database uses a different sample size.	64
6	The criteria are identical to level 4, but the database uses a different sample size.	128
7	The criteria are identical to level 4, but the database uses a different sample size.	256
8	The criteria are identical to level 4, but the database uses a different sample size.	1024
9	The criteria are identical to level 4, but the database uses a different sample size.	4096

Table 12-5 (Cont.) Dynamic Statistics Levels

Level	When the Optimizer Uses Dynamic Statistics	Sample Size (Blocks)
10	The criteria are identical to level 4, but the database uses a different sample size.	All blocks
11	The database uses adaptive dynamic sampling automatically when the optimizer deems it necessary.	Automatically determined

> **See Also:**
>
> - "When the Database Samples Data"
> - *Oracle Database Reference* to learn about the `OPTIMIZER_DYNAMIC_SAMPLING` initialization parameter

Setting Dynamic Statistics Levels Manually

Determining a database-level setting that would be beneficial to all SQL statements can be difficult.

When setting the level for dynamic statistics, Oracle recommends setting the `OPTIMIZER_DYNAMIC_SAMPLING` initialization parameter at the session level.

Assumptions

This tutorial assumes the following:

- You want correct selectivity estimates for the following query, which has `WHERE` clause predicates on two correlated columns:

```
SELECT *
FROM   sh.customers
WHERE  cust_city='Los Angeles'
AND    cust_state_province='CA';
```

- The preceding query uses serial processing.
- The `sh.customers` table contains 932 rows that meet the conditions in the query.
- You have gathered statistics on the `sh.customers` table.
- You created an index on the `cust_city` and `cust_state_province` columns.
- The `OPTIMIZER_DYNAMIC_SAMPLING` initialization parameter is set to the default level of `2`.

To set the dynamic statistics level manually:

1. In SQL*Plus or SQL Developer, log in to the database as a user with the necessary privileges.
2. Explain the execution plan as follows:

```
EXPLAIN PLAN FOR
  SELECT *
```

```
  FROM   sh.customers
  WHERE  cust_city='Los Angeles'
  AND    cust_state_province='CA';
```

3. Query the plan as follows:

```
SET LINESIZE 130
SET PAGESIZE 0
SELECT *
FROM   TABLE(DBMS_XPLAN.DISPLAY);
```

The output appears below (the example has been reformatted to fit on the page):

```
------------------------------------------------------------------------------
|Id| Operation                    | Name             |Rows|Bytes|Cost | Time   |
------------------------------------------------------------------------------
| 0| SELECT STATEMENT             |                   | 53| 9593|53(0)|00:00:01|
| 1|  TABLE ACCESS BY INDEX ROWID|CUSTOMERS          | 53| 9593|53(0)|00:00:01|
|*2|   INDEX RANGE SCAN           |CUST_CITY_STATE_IND| 53| 9593| 3(0)|00:00:01|
------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   2 - access("CUST_CITY"='Los Angeles' AND "CUST_STATE_PROVINCE"='CA')
```

The columns in the `WHERE` clause have a real-world correlation, but the optimizer is not aware that Los Angeles is in California and assumes both predicates reduce the number of rows returned. Thus, the table contains 932 rows that meet the conditions, but the optimizer estimates 53, as shown in bold.

If the database had used dynamic statistics for this plan, then the `Note` section of the plan output would have indicated this fact. The optimizer did not use dynamic statistics because the statement executed serially, standard statistics exist, and the parameter `OPTIMIZER_DYNAMIC_SAMPLING` is set to the default of `2`.

4. Set the dynamic statistics level to `4` in the session using the following statement:

```
ALTER SESSION SET OPTIMIZER_DYNAMIC_SAMPLING=4;
```

5. Explain the plan again:

```
EXPLAIN PLAN FOR
  SELECT *
  FROM   sh.customers
  WHERE  cust_city='Los Angeles'
  AND    cust_state_province='CA';
```

The new plan shows a more accurate estimate of the number of rows, as shown by the value 932 in bold:

```
PLAN_TABLE_OUTPUT
------------------------------------------------------------------------------
Plan hash value: 2008213504
```

```
-------------------------------------------------------------------------
| Id  | Operation         | Name      |Rows |Bytes |Cost (%CPU)|Time    |
-------------------------------------------------------------------------
|   0 | SELECT STATEMENT  |           | 932 | 271K|  406 (1)| 00:00:05 |
|*  1 |  TABLE ACCESS FULL| CUSTOMERS | 932 | 271K|  406 (1)| 00:00:05 |
-------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   1 - filter("CUST_CITY"='Los Angeles' AND "CUST_STATE_PROVINCE"='CA')

Note
-----
   - dynamic statistics used for this statement (level=4)
```

The note at the bottom of the plan indicates that the sampling level is `4`. The additional dynamic statistics made the optimizer aware of the real-world relationship between the `cust_city` and `cust_state_province` columns, thereby enabling it to produce a more accurate estimate for the number of rows: 932 rather than 53.

> **See Also:**
>
> - *Oracle Database SQL Language Reference* to learn about setting sampling levels with the `DYNAMIC_SAMPLING` hint
> - *Oracle Database Reference* to learn about the `OPTIMIZER_DYNAMIC_SAMPLING` initialization parameter

Disabling Dynamic Statistics

In general, the best practice is not to incur the cost of dynamic statistics for queries whose compile times must be as fast as possible, for example, unrepeated OLTP queries.

To disable dynamic statistics at the session level:

1. Connect SQL*Plus to the database with the appropriate privileges.
2. Set the dynamic statistics level to `0`.

   For example, run the following statement:

   ```
   ALTER SESSION SET OPTIMIZER_DYNAMIC_SAMPLING=0;
   ```

> **See Also:**
>
> *Oracle Database Reference* to learn about the `OPTIMIZER_DYNAMIC_SAMPLING` initialization parameter

Dynamic Statistics for PL/SQL Functions

Generation of dynamic statistics is supported for PL/SQL functions.

As of Oracle Database 23.8, the database lets you enable or disable dynamic sampling globally for PL/SQL functions or for selected functions. Dynamic sampling is the process that gathers data for the generation of dynamic statistics.

PL/SQL functions are sometimes used in `WHERE` clause predicates and table functions in `FROM` clauses. In such cases, cardinality estimates are sensitive to the values a PL/SQL function returns during SQL execution. By default, these values are unavailable when a SQL statement is hard parsed, which can lead to poor execution plans. Dynamic sampling at parse time provides sufficient data to enable the optimizer to find better execution plans. This in turn contributes to improved database performance.

However, dynamic sampling to acquire statistics is not always efficient. For example, in the case of a long-running PL/SQL function, sampling at parse time may impact performance. You have several methods to control whether or not dynamic statistics are generated for a specific function as well as with all of the functions in a PL/SQL package. This also applies to standalone PL/SQL functions.

The Use Case for PL/SQL Dynamic Statistics

PL/SQL dynamic statistics can be generated in cases where a PL/SQL function is referenced in a `WHERE` clause, or where a query selects from a PL/SQL function returning a table.

Consider a PL/SQL package:

```
create or replace package ds_test as
  function fnum (p_n in number) return number deterministic;
  function fnum2 (p_n in number) return number deterministic;
  function ftab (p_rows in number) return num_tab deterministic;
  function fpipe (p_rows in number) return num_tab pipelined deterministic;
end ds_test;
/
```

Without dynamic statistics, cardinality estimates for the following queries will be inaccurate:

```
SELECT *  -- non-table functions
FROM   t1
WHERE  (a = ds_test.fnum(20) and b = ds_test.fnum2(20))
OR     a = 10;

SELECT * -- table function
FROM table(ds_test.ftab(1000));
```

For simple queries such as these, cardinality misestimates are not a serious issue, but in complex queries they can lead to poor performance. Better estimates can be achieved by enabling dynamic sampling for PL/SQL, which generates dynamic statistics:

By default, when dynamic sampling is used, it is automatically used with non-table PL/SQL functions, but not with table functions. To globally enable dynamic sampling for both types of functions:

```
exec dbms_stats.set_global_plsql_prefs('dynamic_stats','ON')
```

Controls for Dynamic Sampling With PL/SQL Functions

You can direct how the optimizer uses dynamic sampling with PL/SQL functions.

The following controls are available to manage when and how dynamic sampling is used with PL/SQL functions.

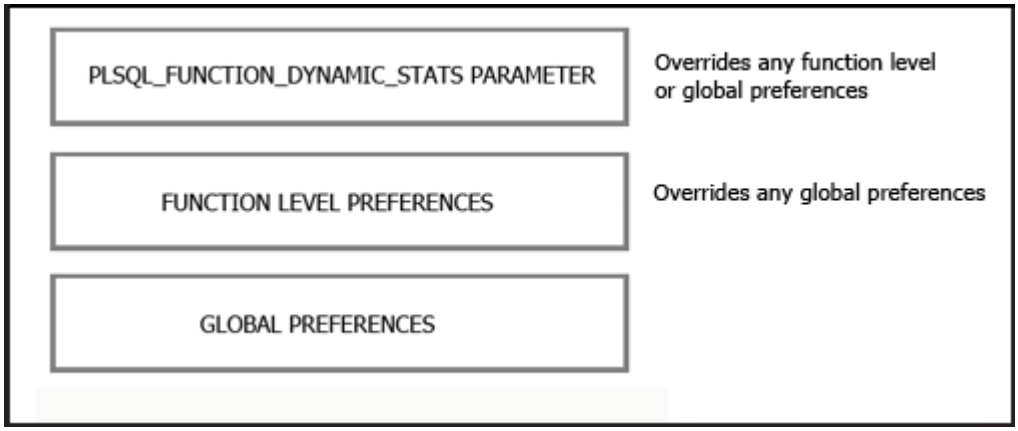
- Set the session-level `PLSQL_FUNCTION_DYNAMIC_STATS` parameter. This parameter overrides function-level and global-level preferences.
- Use procedures in the `DBMS.STATS->DBMS_STATS` PL/SQL package to set and unset PL/SQL preferences at the function-specific or global level.

Order of Precedence for Dynamic Sampling Controls of PL/SQL Functions

For the current session only, the `PLSQL_DYNAMIC_STATS` session-level parameter has precedence over all other preference settings.

Function-level preferences have precedence over global preferences.

Figure 12-1 PL/SQL Dynamic Sampling Controls Order of Precedence



> **See Also:**
>
> Summary of DBMS_STATS Subprograms in the *Database PL/SQL Packages and Types Reference*. This package provides the following APIs for controlling dynamic sampling with PL/SQL functions: `SET_PLSQL_PREFS`, `SET_GLOBAL_PLSQL_PREFS`, `RESET_GLOBAL_PLSQL_PREFS_DEF`, `GET_PLSQL_PREFS`, `GET_PLSQL_PREFS`, `DELETE_PLSQL_PREFS`.

Setting Global Level PL/SQL Preferences

Global preferences are the primary control mechanism for PL/SQL dynamic sampling.

Use the `DBMS_STATS.SET_GLOBAL_PLSQL_PREFS` procedure to set global preferences. The syntax is as follows:

```
DBMS_STATS.SET_GLOBAL_PLSQL_PREFS(
pname IN VARCHAR2,
pvalue IN VARCHAR2);
```

Parameter	Description
`pname`	The preference to set. Currently, the available preference is `'dynamic_stats'`.
`pvalue`	The value of the preference. The value can either be `NULL` or one of these three options: • `'ON'` - Allow dynamic sampling for PL/SQL functions. • `'OFF'` - Disallow dynamic sampling for PL/SQL functions. • `'CHOOSE'` - Leave enabling or disabling dynamic statistics to the discretion of the optimizer. If `pvalue` is NULL, the global preference is set to the default – `CHOOSE`.

Global Level Preference Examples

Turn on dynamic sampling for all PL/SQL functions:

```
EXEC DBMS_STATS.SET_GLOBAL_PLSQL_PREFS('dynamic_stats', 'ON')
```

Turn off PL/SQL dynamic sampling for all PL/SQL functions:

```
EXEC DBMS_STATS.SET_GLOBAL_PLSQL_PREFS('dynamic_stats', 'OFF')
```

Set the global preference to the default value `CHOOSE`:

```
EXEC DBMS_STATS.SET_GLOBAL_PLSQL_PREFS('dynamic_stats', NULL)
```

View the current value of the global preference:

```
SELECT DBMS_STATS.GET_PLSQL_PREFS('dynamic_stats')
       '<PREF_VALUE>' FROM DUAL;
```

Setting Function-Level Preferences

Dynamic statistics preferences can be set for individual PL/SQL functions.

Use the `DBMS_STATS.SET_PLSQL_PREFS` procedure to set function-level preferences. The syntax is as follows:

```
DBMS_STATS.SET_PLSQL_PREFS(
    ownname         IN VARCHAR2,
    package_name    IN VARCHAR2,
    function_name   IN VARCHAR2,
    pname           IN VARCHAR2,
    pvalue          IN VARCHAR2
);
```

Parameter	Description
`ownname`	Owner of the function. If `NULL`, the current user is used.
`package_name`	The PL/SQL package name. If `NULL`, this specifies a stand-alone, top-level function.
`function_name`	Name of the PL/SQL function. To apply the property to all functions within the package, specify `NULL`.
`pname`	The preference that you want to set. Currently only `'dynamic_stats'` is accepted.
`pvalue`	The value of the preference. These are the options for the `'dynamic_stats'` preference: • `'ON'` - Allow dynamic statistics for this PL/SQL function • `'OFF'` - Disallow dynamic statistics for this PL/SQL function • `'CHOOSE'` - Leave enabling or disabling dynamic statistics to the discretion of the Optimizer. • If `pvalue` is `NULL`, the dynamic statistics preference set for this function is deleted and the global preference is used.

Function-Level Preference Examples

Turn on dynamic statistics for a specific function. The default owner is the current schema and so the `OWNNAME` parameter can be set to `NULL`.

```
EXEC SET_PLSQL_PREFS(NULL, '<PACKAGE_NAME>', '<FUNCTION_NAME>',
'dynamic_stats', 'ON')
```

Turn off dynamic statistics for a specific function. Again in this case, by default the owner is the current schema so the first parameter can be NULL.

```
EXEC SET_PLSQL_PREFS(NULL, '<PACKAGE_NAME>', '<FUNCTION_NAME>',
'dynamic_stats', 'OFF')
```

Turn on dynamic statistics for all functions in the named package. In this example, the `FUNCTION_NAME` parameter is set to NULL. Therefore the dynamic statistics preference is set on all functions within the named PL/SQL package.

```
EXEC SET_PLSQL_PREFS('OWNNAME', '<PACKAGE_NAME>', NULL, 'dynamic_stats', 'ON')
```

> **Note:**
>
> You can use `DBMS_STATS.GET_PLSQL_PREFS` to retrieve the current preference setting.
>
> ```
> SELECT DBMS_STATS.GET_PLSQL_PREFS('dynamic_stats',
>       '<OWNNAME>,'<PACKAGE_NAME>','<FUNCTION NAME>')
>       PREF_VALUE FROM DUAL;
> ```

Setting the PLSQL_FUNCTION_DYNAMIC_STATS Session Level Parameter

You can set PL/SQL dynamic statistics behavior for the current session.

> **Note:**
>
> This parameter is a temporary override of the global and function preference. Use global settings to control a preference at the system level.

Set `PLSQL_FUNCTION_DYNAMIC_STATS` as follows:

```
ALTER SESSION SET PLSQL_FUNCTION_DYNAMIC_STATS = '<VALUE>';
```

Value	Description
`'ON'`	Allow dynamic statistics for all PL/SQL functions, both table and non-table functions.
`'OFF'`	Disallow dynamic statistics for all PL/SQL functions.
`'PREFERENCE'`	Check function and/or global level preferences for dynamic statistics behavior. This is the default.
`'CHOOSE'`	Let the optimizer choose whether or not to perform dynamic statistics on PL/SQL functions. The optimizer currently chooses to do dynamic statistics for non-table functions, but not for table functions.

Managing SQL Plan Directives

A **SQL plan directive** is additional information and instructions that the optimizer can use to generate a more optimal plan.

A directive informs the database that the optimizer is misestimate cardinalities of certain types of predicates, and alerts `DBMS_STATS` to gather additional statistics in the future. Thus, directives have an effect on statistics gathering.

The database automatically creates and manages SQL plan directives in the SGA, and then periodically writes them to the data dictionary. If the directives are not used within 53 weeks, then the database automatically purges them.

You can use `DBMS_SPD` procedures and functions to alter, save, drop, and transport directives manually. The following table lists some of the more commonly used procedures and functions.

Table 12-6 DBMS_SPD Procedures

Procedure	Description
`FLUSH_SQL_PLAN_DIRECTIVE`	Forces the database to write directives from memory to persistent storage in the `SYSAUX` tablespace.
`DROP_SQL_PLAN_DIRECTIVE`	Drops a SQL plan directive. If a directive that triggers dynamic sampling is creating unacceptable performance overhead, then you may want to remove it manually. If a SQL plan directive is dropped manually or automatically, then the database can re-create it. To prevent its re-creation, you can use `DBMS_SPM.ALTER_SQL_PLAN_DIRECTIVE` to do the following: • Disable the directive by setting `ENABLED` to `NO` • Prevent the directive from being dropped by setting `AUTO_DROP` to `NO` To disable SQL plan directives, set `OPTIMIZER_ADAPTIVE_STATISTICS` to `FALSE`.

Prerequisites

You must have the Administer SQL Management Object privilege to execute the `DBMS_SPD` APIs.

Assumptions

This tutorial assumes that you want to do the following:

- Write all directives for the `sh` schema to persistent storage.
- Delete all directives for the `sh` schema.

To write and then delete all sh schema plan directives:

1. In SQL*Plus or SQL Developer, log in to the database as a user with the necessary privileges.
2. Force the database to write the SQL plan directives to disk.

   For example, execute the following `DBMS_SPD` program:

   ```
   BEGIN
     DBMS_SPD.FLUSH_SQL_PLAN_DIRECTIVE;
   END;
   /
   ```

3. Query the data dictionary for information about existing directives in the `sh` schema.

   Example 12-3 queries the data dictionary for information about the directive.
4. Delete the existing SQL plan directive for the `sh` schema.

The following PL/SQL program unit deletes the SQL plan directive with the ID 1484026771529551585:

```
BEGIN
  DBMS_SPD.DROP_SQL_PLAN_DIRECTIVE ( directive_id => 1484026771529551585 );
END;
/
```

Example 12-3 Display Directives for sh Schema

This example shows SQL plan directives, and the results of SQL plan directive dynamic sampling queries.

```
SELECT TO_CHAR(d.DIRECTIVE_ID) dir_id, o.OWNER, o.OBJECT_NAME,
       o.SUBOBJECT_NAME col_name, o.OBJECT_TYPE object, d.TYPE,
       d.STATE, d.REASON
FROM   DBA_SQL_PLAN_DIRECTIVES d, DBA_SQL_PLAN_DIR_OBJECTS o
WHERE  d.DIRECTIVE_ID=o.DIRECTIVE_ID
AND    o.OWNER IN ('SH')
ORDER BY 1,2,3,4,5;

DIR_ID               OWN OBJECT_NA COL_NAME   OBJECT  TYPE     STATE      REASON
-------------------- --- --------- ---------- ------- -------- ---------- ------------
1484026771529551585  SH CUSTOMERS COUNTRY_ID  COLUMN DYNAMIC_ SUPERSEDED SINGLE TABLE
                                                       SAMPLING           CARDINALITY
                                                                          MISESTIMATE
1484026771529551585  SH CUSTOMERS CUST_STATE_ COLUMN DYNAMIC_ SUPERSEDED SINGLE TABLE
                                  PROVINCE            SAMPLING           CARDINALITY
                                                                          MISESTIMATE
1484026771529551585  SH CUSTOMERS               TABLE DYNAMIC_ SUPERSEDED SINGLE TABLE
                                                       SAMPLING           CARDINALITY
                                                                          MISESTIMATE
9781501826140511330  SH dyg4msnst5           SQL STA DYNAMIC_     USABLE VERIFY
                                             TEMENT  SAMPLING           CARDINALITY
                                                     _RESULT            ESTIMATE
9872337207064898539  SH TIMES                  TABLE DYNAMIC_     USABLE VERIFY
                                                       SAMPLING           CARDINALITY
                                                       _RESULT            ESTIMATE
9781501826140511330  SH 2nk1v0fdx0           SQL STA DYNAMIC_     USABLE VERIFY
                                             TEMENT  SAMPLING           CARDINALITY
                                                     _RESULT            ESTIMATE
```

See Also:

- "SQL Plan Directives"
- *Oracle Database PL/SQL Packages and Types Reference* for complete syntax and semantics for the `DBMS_SPD` package.
- *Oracle Database Reference* to learn about `DBA_SQL_PLAN_DIRECTIVES`