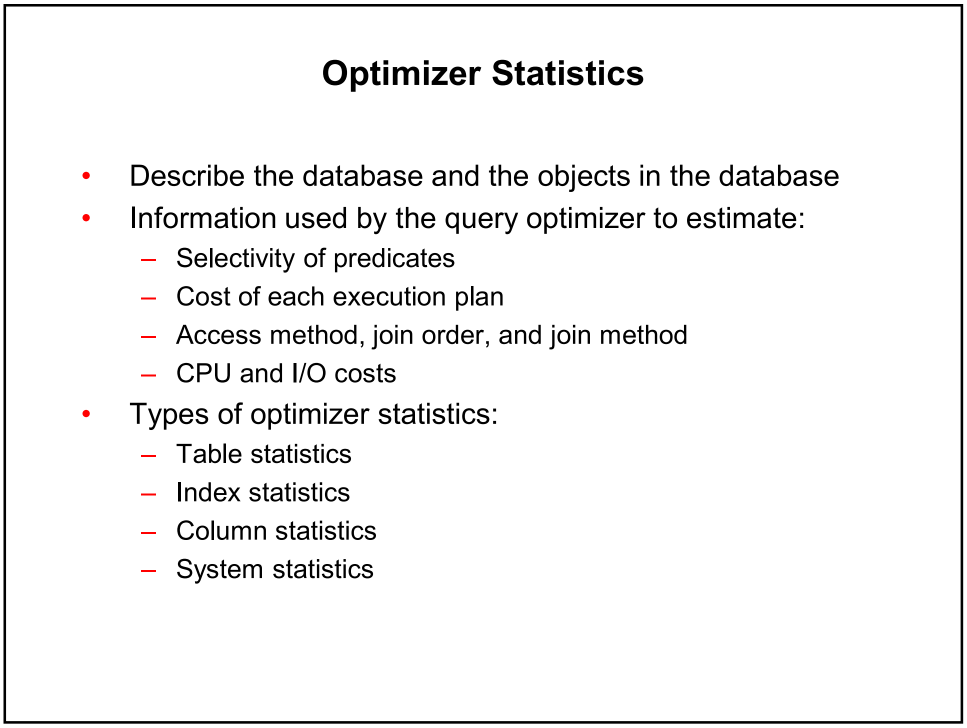
10

# Optimizer Statistics

## Objectives

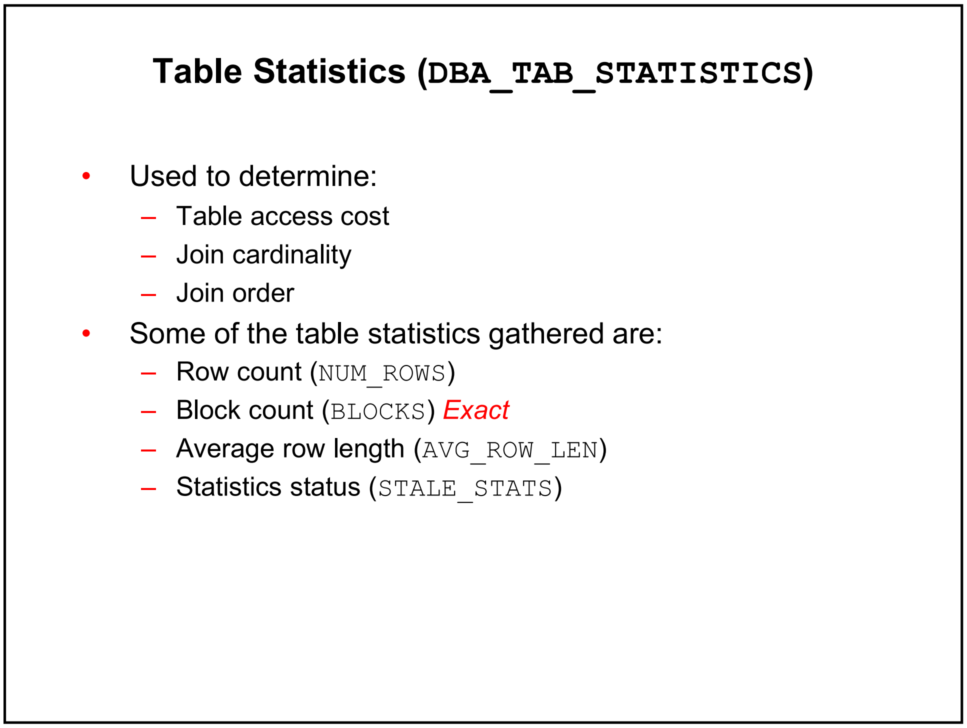
After completing this lesson, you should be able to:

- Describe optimizer statistics
  - Table statistics
  - Index statistics
  - Column statistics (histogram)
  - Column statistics (extended statistics)
  - System statistics
- Gather optimizer statistics
- Set statistic preferences
- Use dynamic sampling
- Manage optimizer statistics
- Discuss optimizer statistics best practices



## Optimizer Statistics

- Describe the database and the objects in the database
- Information used by the query optimizer to estimate:
  - Selectivity of predicates
  - Cost of each execution plan
  - Access method, join order, and join method
  - CPU and I/O costs
- Types of optimizer statistics:
  - Table statistics
  - Index statistics
  - Column statistics
  - System statistics



## Table Statistics (`DBA_TAB_STATISTICS`)

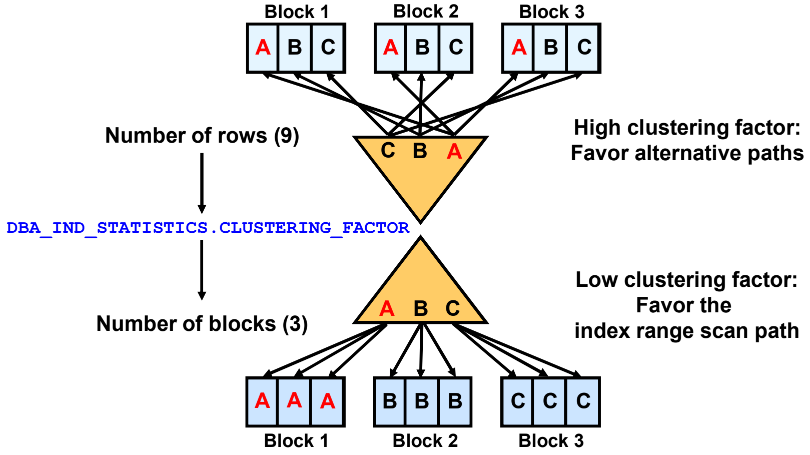
- Used to determine:
  - Table access cost
  - Join cardinality
  - Join order
- Some of the table statistics gathered are:
  - Row count (`NUM_ROWS`)
  - Block count (`BLOCKS`) *Exact*
  - Average row length (`AVG_ROW_LEN`)
  - Statistics status (`STALE_STATS`)

## Index Statistics (DBA_IND_STATISTICS)

- Used to decide:
  - Full table scan versus index scan
- Statistics gathered are:
  - B*-tree level (BLEVEL) *Exact*
  - Leaf block count (LEAF_BLOCKS)
  - Clustering factor (CLUSTERING_FACTOR)
  - Distinct keys (DISTINCT_KEYS)
  - Average number of leaf blocks in which each distinct value in the index appears (AVG_LEAF_BLOCKS_PER_KEY)
  - Average number of data blocks in the table pointed to by a distinct value in the index (AVG_DATA_BLOCKS_PER_KEY)
  - Number of rows in the index (NUM_ROWS)

## Index Clustering Factor

Must read all blocks to retrieve all As



Only need to read one block to retrieve all As

## Column Statistics (`DBA_TAB_COL_STATISTICS`)

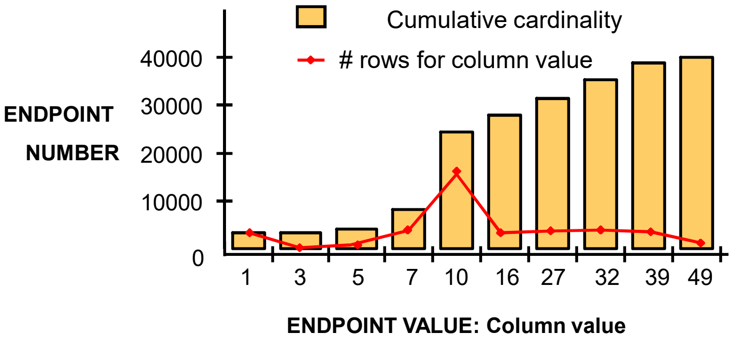
- Count of distinct values of the column (`NUM_DISTINCT`)
- Low value (`LOW_VALUE`) *Exact*
- High value (`HIGH_VALUE`) *Exact*
- Number of nulls (`NUM_NULLS`)
- Selectivity estimate for nonpopular values (`DENSITY`)
- Number of histogram buckets (`NUM_BUCKETS`)
- Type of histogram (`HISTOGRAM`)

## Column Statistics: Histograms

- The optimizer assumes uniform distributions; this may lead to suboptimal access plans in the case of data skew.
- Histograms:
  - Store additional column distribution information
  - Give better selectivity estimates in the case of nonuniform distributions
- With unlimited resources, you could store each different value and the number of rows for that value.
- This becomes unmanageable for a large number of distinct values, and a different approach is used:
  - Frequency histogram (#distinct values ≤ #buckets)
  - Height-balanced histogram (#buckets < #distinct values)
- They are stored in `DBA_TAB_HISTOGRAMS`.

# Frequency Histograms

### 10 buckets, 10 distinct values



Distinct values: 1, 3, 5, 7, 10, 16, 27, 32, 39, 49

Number of rows: 40001

# Viewing Frequency Histograms

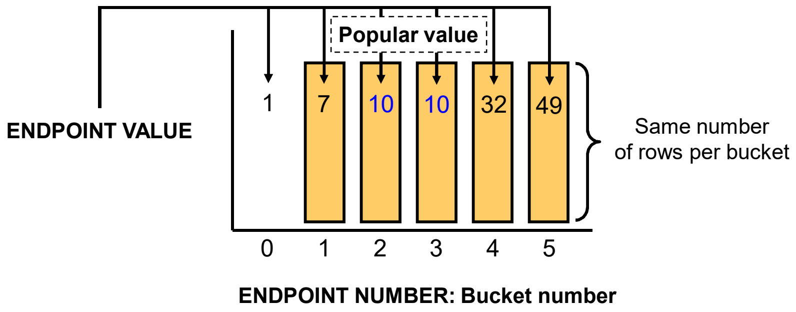
```
BEGIN
 DBMS_STATS.gather_table_STATS (OWNNAME=>'OE', TABNAME=>'INVENTORIES',
       METHOD_OPT => 'FOR COLUMNS SIZE 20 warehouse_id');
END;
```

```
SELECT column_name, num_distinct, num_buckets, histogram
FROM   USER_TAB_COL_STATISTICS
WHERE  table_name = 'INVENTORIES' AND
       column_name = 'WAREHOUSE_ID';

COLUMN_NAME  NUM_DISTINCT NUM_BUCKETS HISTOGRAM
------------ ------------ ----------- ---------
WAREHOUSE_ID            9           9 FREQUENCY
```

```
SELECT endpoint_number, endpoint_value
FROM   USER_HISTOGRAMS
WHERE  table_name = 'INVENTORIES' and column_name = 'WAREHOUSE_ID'
ORDER BY endpoint_number;

ENDPOINT_NUMBER ENDPOINT_VALUE
--------------- --------------
36                           1
213                          2
261                          3
…
```

## Height-Balanced Histograms

**5 buckets, 10 distinct values**
**(8000 rows per bucket)**



Distinct values: 1, 3, 5, 7, 10, 16, 27, 32, 39, 49

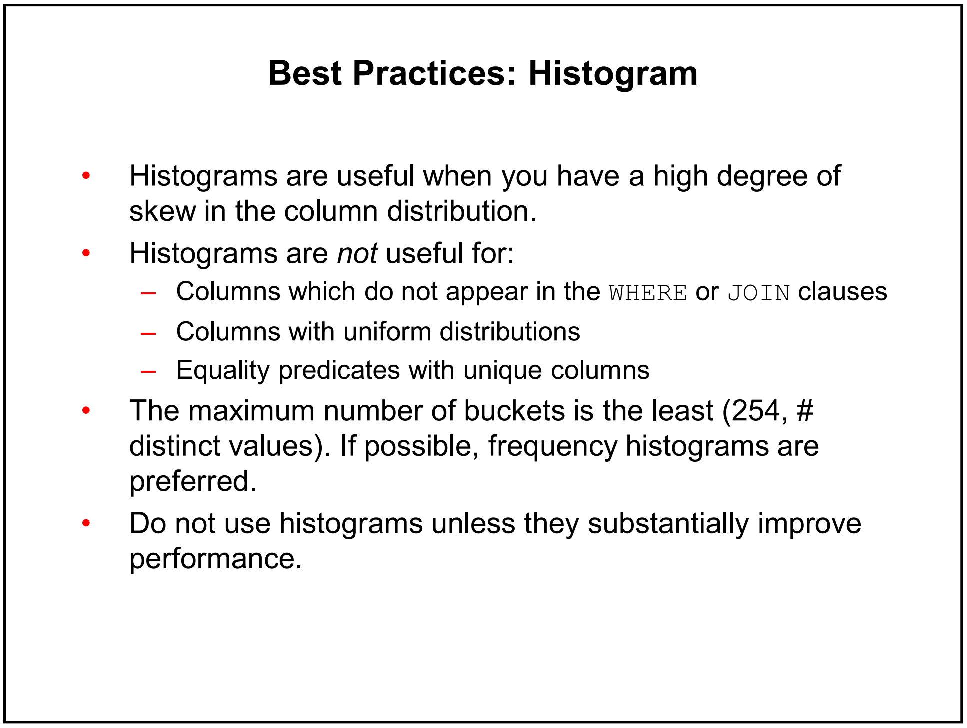
Number of rows: 40001

## Viewing Height-Balanced Histograms

```
BEGIN
  DBMS_STATS.gather_table_STATS(OWNNAME =>'OE', TABNAME=>'INVENTORIES',
  METHOD_OPT => 'FOR COLUMNS SIZE 10 quantity_on_hand');
END;
```

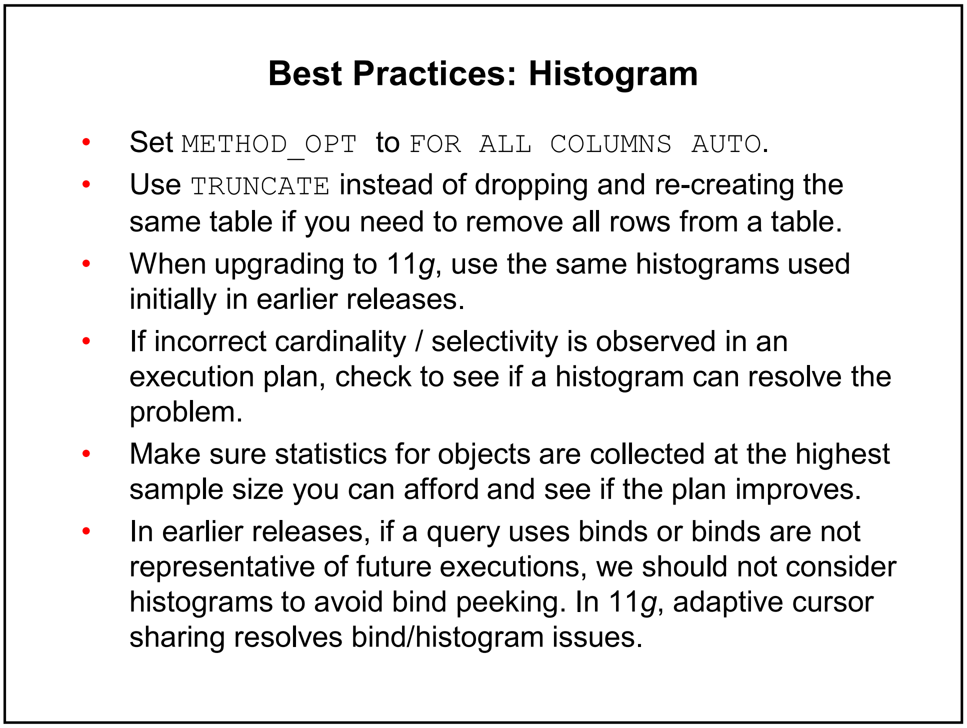
```
SELECT column_name, num_distinct, num_buckets, histogram
  FROM USER_TAB_COL_STATISTICS
 WHERE table_name = 'INVENTORIES' AND column_name = 'QUANTITY_ON_HAND';

COLUMN_NAME                    NUM_DISTINCT NUM_BUCKETS HISTOGRAM
------------------------------ ------------ ----------- ---------------
QUANTITY_ON_HAND                        237          10 HEIGHT BALANCED
```

```
SELECT endpoint_number, endpoint_value
FROM USER_HISTOGRAMS
WHERE table_name = 'INVENTORIES' and column_name = 'QUANTITY_ON_HAND'
ORDER BY endpoint_number;

ENDPOINT_NUMBER ENDPOINT_VALUE
--------------- --------------
              0              0
              1             27
              2             42
              3             57
…
```



## Best Practices: Histogram

- Histograms are useful when you have a high degree of skew in the column distribution.
- Histograms are *not* useful for:
  - Columns which do not appear in the `WHERE` or `JOIN` clauses
  - Columns with uniform distributions
  - Equality predicates with unique columns
- The maximum number of buckets is the least (254, # distinct values). If possible, frequency histograms are preferred.
- Do not use histograms unless they substantially improve performance.



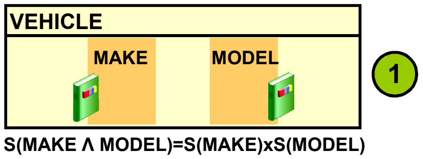
## Best Practices: Histogram

- Set `METHOD_OPT` to `FOR ALL COLUMNS AUTO`.
- Use `TRUNCATE` instead of dropping and re-creating the same table if you need to remove all rows from a table.
- When upgrading to 11*g*, use the same histograms used initially in earlier releases.
- If incorrect cardinality / selectivity is observed in an execution plan, check to see if a histogram can resolve the problem.
- Make sure statistics for objects are collected at the highest sample size you can afford and see if the plan improves.
- In earlier releases, if a query uses binds or binds are not representative of future executions, we should not consider histograms to avoid bind peeking. In 11*g*, adaptive cursor sharing resolves bind/histogram issues.

## Column Statistics: Extended Statistics

- The optimizer poorly estimates selectivity *on Highly Correlated Column Predicates:*
  - Columns have values that are highly correlated.
  - Actual selectivity is often much lower or higher than the optimizer estimates. For example,
    `WHERE cust_state_province = 'CA'`
    `AND country_id=52775;`
- The optimizer poorly estimates *Expression on Columns:*
  - `WHERE upper(model)='MODEL'`
  - When a function is applied to a column in the `WHERE` clause, the optimizer has no way of knowing how that function affects the selectivity of the column.
- In these cases, a group of columns within a table or an expression on a column can be gathered to obtain a more accurate selectivity value.

## Multicolumn Statistics

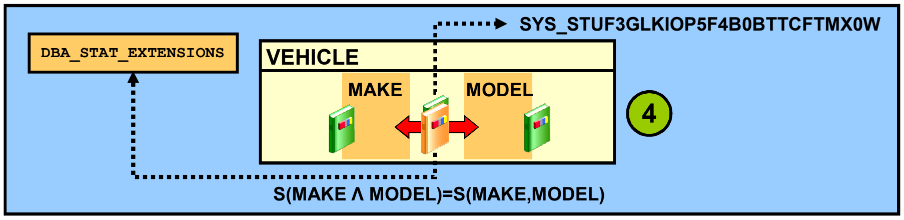


```
select
dbms_stats.create_extended_stats('jfv','vehicle','(make,model)')
from dual;
```

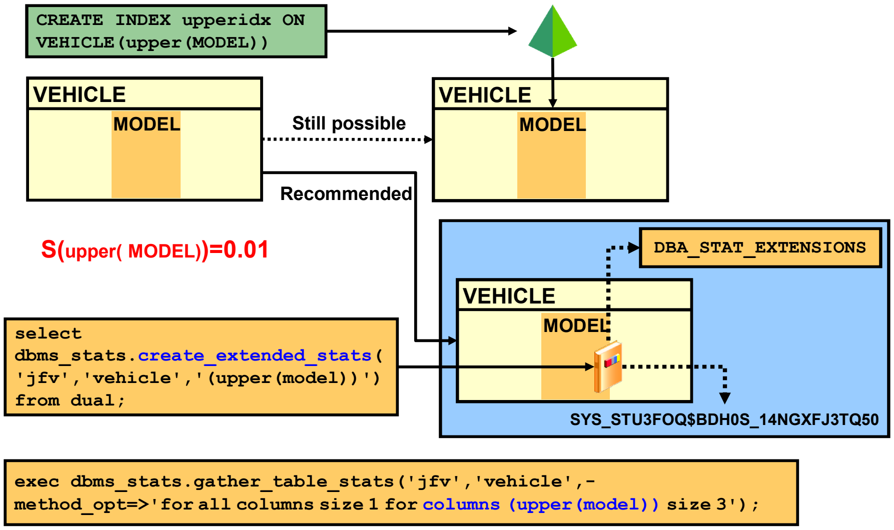
2

```
exec dbms_stats.gather_table_stats('jfv','vehicle',-
method_opt=>'for all columns size 1 for columns (make,model) size 3');
```

3



# Expression Statistics



```
CREATE INDEX upperidx ON
VEHICLE(upper(MODEL))
```

```
select
dbms_stats.create_extended_stats(
'jfv','vehicle','(upper(model))')
from dual;
```

```
exec dbms_stats.gather_table_stats('jfv','vehicle',-
method_opt=>'for all columns size 1 for columns (upper(model)) size 3');
```

# System Statistics

- System statistics are used to estimate:
  - I/O performance and utilization
  - CPU performance and utilization
- System statistics enable the query optimizer to estimate I/O and CPU costs more accurately, enabling the query optimizer to choose a better execution plan.
- Procedures:
  - `DBMS_STATS.GATHER_SYSTEM_STATS`
  - `DBMS_STATS.SET_SYSTEM_STATS`
  - `DBMS_STATS.GET_SYSTEM_STATS`

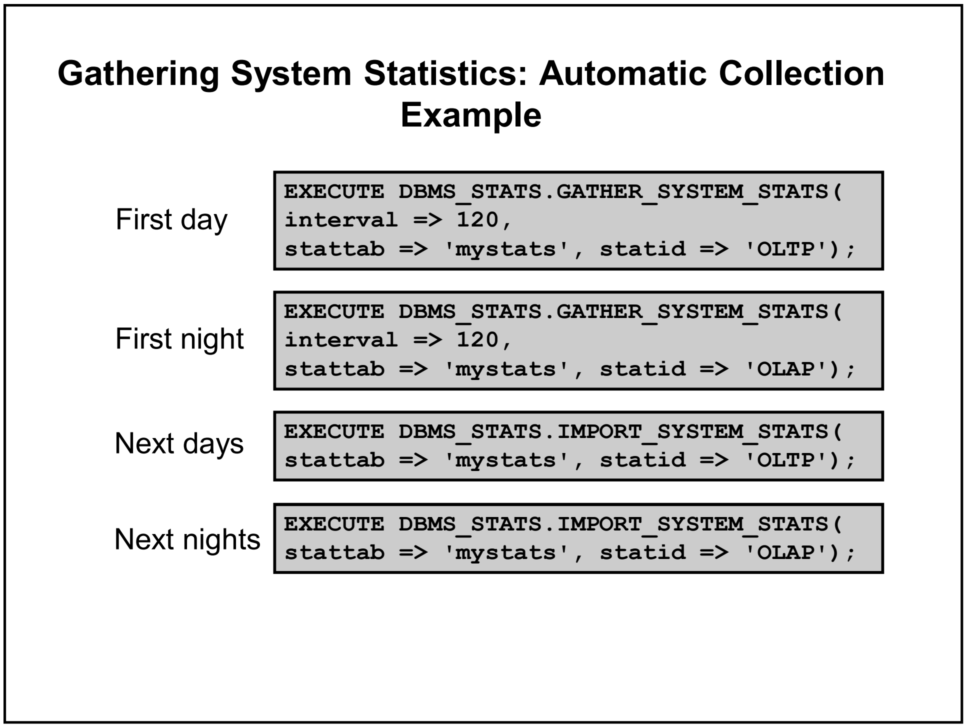
## System Statistics: Example

Viewing System Statistics:

```
SELECT * FROM sys.aux_stats$;

SNAME                 PNAME              PVAL1      PVAL2
-------------------- ------------------ ---------- -------------------
SYSSTATS_INFO       STATUS                         COMPLETED
SYSSTATS_INFO       DSTART                         08-09-2001 16:40
SYSSTATS_INFO       DSTOP                          08-09-2001 16:42
SYSSTATS_INFO       FLAGS                       0
SYSSTATS_MAIN       SREADTIM                7.581
SYSSTATS_MAIN       MREADTIM               56.842
SYSSTATS_MAIN       CPUSPEED                  117
SYSSTATS_MAIN       MBRC                        9
```

## Best Practices: System Statistics

- System statistics must be gathered on a regular basis; this does not invalidate cached plans.
- Gathering system statistics equals analyzing system activity for a specified period of time.
- When gathering the optimizer system statistics:
  - It is highly recommended that you gather system statistics during normal workload for several hours.
  - If no real workload is available, you can also gather `NORWORKLOAD` statistics.



## Gathering System Statistics: Automatic Collection Example

First day

```
EXECUTE DBMS_STATS.GATHER_SYSTEM_STATS(
interval => 120,
stattab => 'mystats', statid => 'OLTP');
```

First night

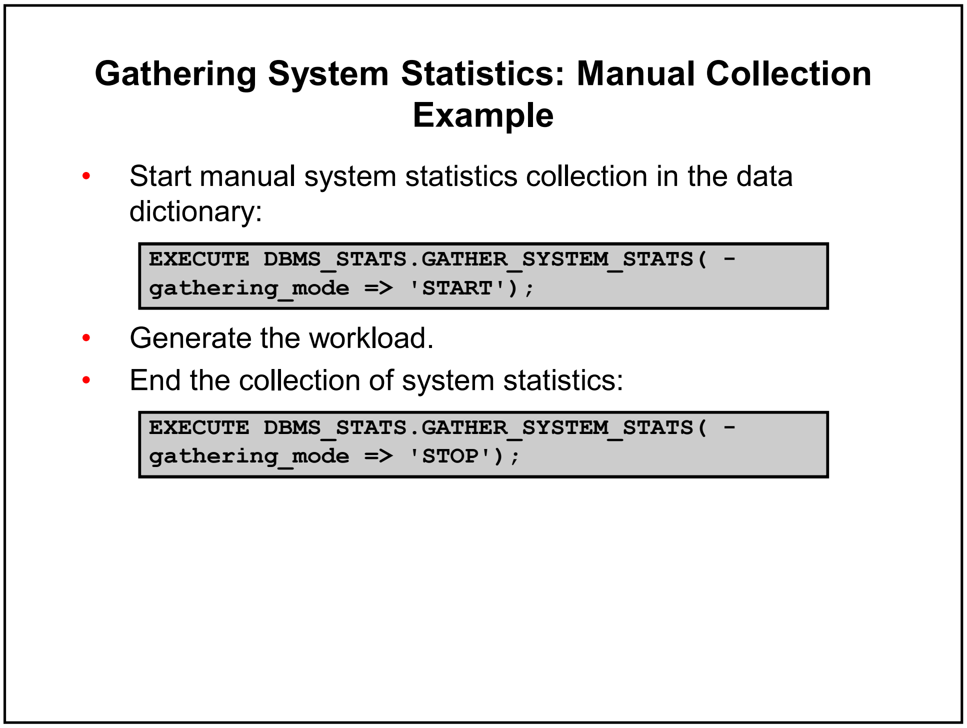
```
EXECUTE DBMS_STATS.GATHER_SYSTEM_STATS(
interval => 120,
stattab => 'mystats', statid => 'OLAP');
```

Next days

```
EXECUTE DBMS_STATS.IMPORT_SYSTEM_STATS(
stattab => 'mystats', statid => 'OLTP');
```

Next nights

```
EXECUTE DBMS_STATS.IMPORT_SYSTEM_STATS(
stattab => 'mystats', statid => 'OLAP');
```



## Gathering System Statistics: Manual Collection Example

- Start manual system statistics collection in the data dictionary:

```
EXECUTE DBMS_STATS.GATHER_SYSTEM_STATS( -
gathering_mode => 'START');
```

- Generate the workload.
- End the collection of system statistics:

```
EXECUTE DBMS_STATS.GATHER_SYSTEM_STATS( -
gathering_mode => 'STOP');
```

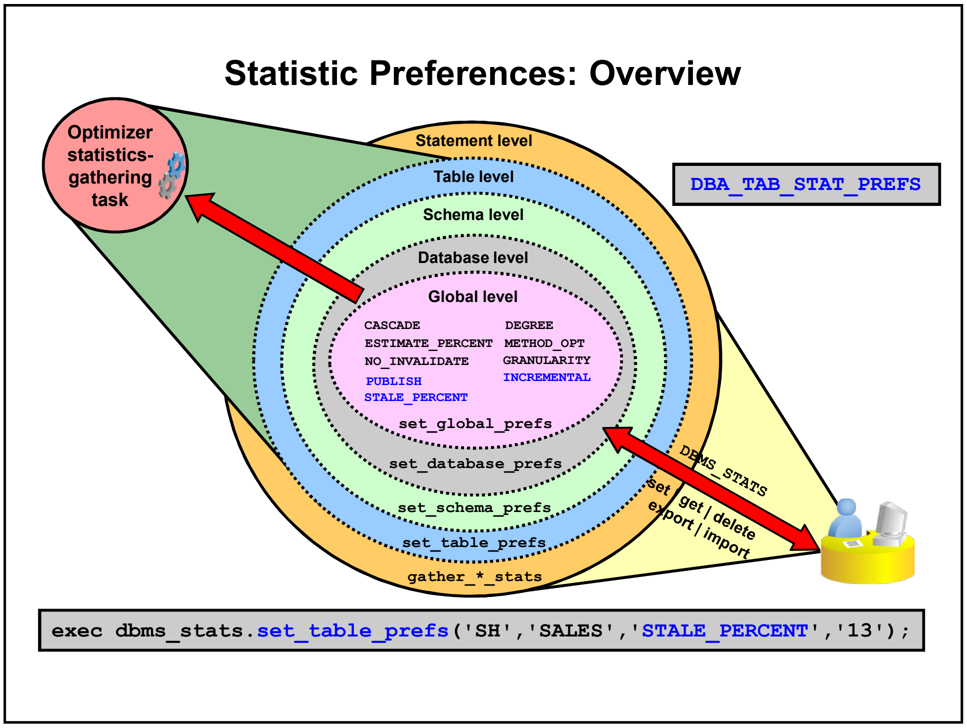
## Gathering Statistics: Overview

- Automatic statistics gathering
  - `gather_stats_prog` automated task
- Manual statistics gathering
  - `DBMS_STATS` package
- Dynamic sampling
- When statistics are missing
- Vendor-recommended gathering

| | |
|---|---|
| **Selectivity:** | |
| **Equality** | **1%** |
| **Inequality** | **5%** |
| **Other predicates** | **5%** |
| **Table row length** | **20** |
| **# of index leaf blocks** | **25** |
| **# of distinct values** | **100** |
| **Table cardinality** | **100** |
| **Remote table cardinality** | **2000** |

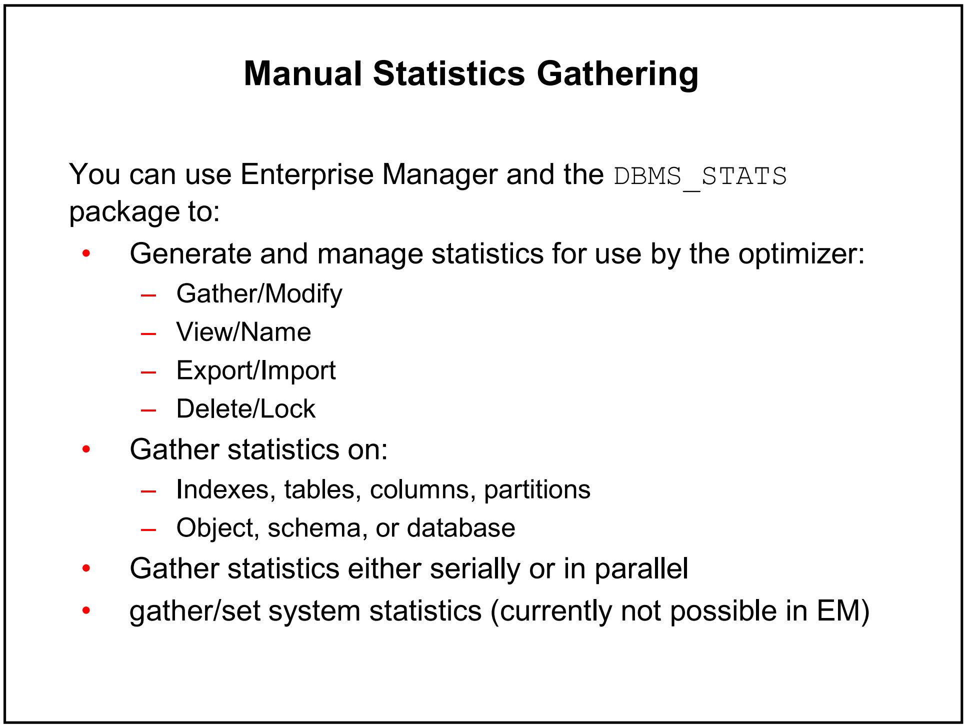
## Automatic Statistics Gathering

- Oracle's recommended method for collecting statistics
- Oracle Database 11*g* automates optimizer statistics collection:
  - Statistics are gathered automatically only on all database objects that have no statistics or have stale statistics (> 10% of rows modified)
  - The `gather_stats_prog` automated task is used for statistics collection and maintenance.
- Automated statistics collection:
  - Eliminates need for manual statistics collection
  - Significantly reduces the chances of poor execution plans
- The Statistic Preferences feature is available in Oracle 11*g* for some objects that require statistics collection settings that are different from the database default.



## Statistic Preferences: Overview

```
exec dbms_stats.set_table_prefs('SH','SALES','STALE_PERCENT','13');
```



## Manual Statistics Gathering

You can use Enterprise Manager and the DBMS_STATS package to:

- Generate and manage statistics for use by the optimizer:
  - Gather/Modify
  - View/Name
  - Export/Import
  - Delete/Lock
- Gather statistics on:
  - Indexes, tables, columns, partitions
  - Object, schema, or database
- Gather statistics either serially or in parallel
- gather/set system statistics (currently not possible in EM)

## When to Gather Statistics Manually

- Rely mostly on automatic statistics collection:
  - Change the frequency of automatic statistics collection to meet your needs.
  - Remember that `STATISTICS_LEVEL` should be set to `TYPICAL` or `ALL` for automatic statistics collection to work properly.
- Gather statistics manually for:
  - Objects that are volatile
  - Objects modified in batch operations (Gather statistics as part of the batch operation.)
  - External tables, system statistics, fixed objects
  - New objects (Gather statistics right after object creation.)

## Manual Statistics Collection: Factors

- Monitor objects for DMLs.
- Determine the correct sample sizes.
- Determine the degree of parallelism.
- Determine if histograms should be used.
- Determine the cascading effects on indexes.
- Procedures to use in `DBMS_STATS`:
  - `GATHER_INDEX_STATS`
  - `GATHER_TABLE_STATS`
  - `GATHER_SCHEMA_STATS`
  - `GATHER_DICTIONARY_STATS`
  - `GATHER_DATABASE_STATS`
  - `GATHER_SYSTEM_STATS`

## Gathering Object Statistics: Example

```
dbms_stats.gather_table_stats
('sh'                  -- schema
,'customers'           -- table
, null                 -- partition
, 20                   -- sample size(%)
, false                -- block sample?
,'for all columns'     -- column spec
, 4                    -- degree of parallelism
,'default'             -- granularity
, true );              -- cascade to indexes
```

```
dbms_stats.set_param('CASCADE',
                     'DBMS_STATS.AUTO_CASCADE');
dbms_stats.set_param('ESTIMATE_PERCENT','5');
dbms_stats.set_param('DEGREE','NULL');
```

## Best Practices: Object Statistics

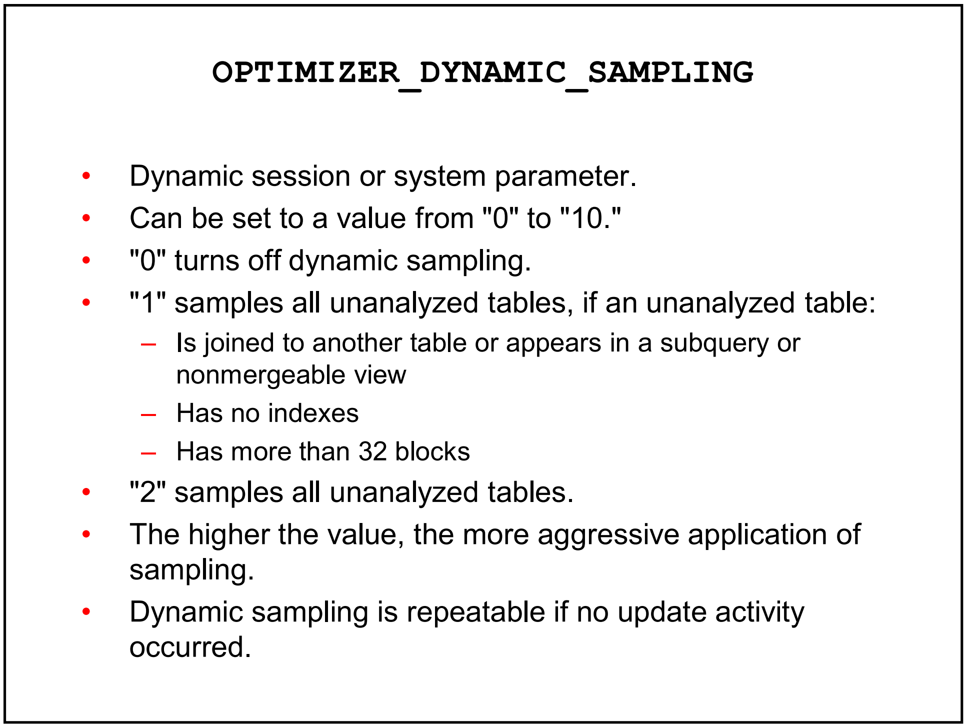
- Ensure that all objects (tables and indexes) have statistics gathered.
- Use a sample size that is large enough if feasible.
- Gather optimizer statistics during periods of low activity.
- If partitions are in use, gather global statistics if possible.
- Use Oracle Database11*g* pending statistics to verify effect of new statistics when tuning to minimize risk.
- Gather statistics after data has been loaded (>10% added), but before indexes are created.

## Optimizer Dynamic Sampling: Overview

- Dynamic sampling can be done for tables and indexes:
  - Without statistics
  - Whose statistics cannot be trusted, starting with 11*g*R2 if object statistics are stale and sampling level => 4
- Used to determine more accurate statistics when estimating:
  - Table cardinality
  - Predicate selectivity
- Feature controlled by:
  - `OPTIMIZER_DYNAMIC_SAMPLING` parameter
  - `OPTIMIZER_FEATURES_ENABLE` parameter
  - `DYNAMIC_SAMPLING` hint
  - `DYNAMIC_SAMPLING_EST_CDN` hint

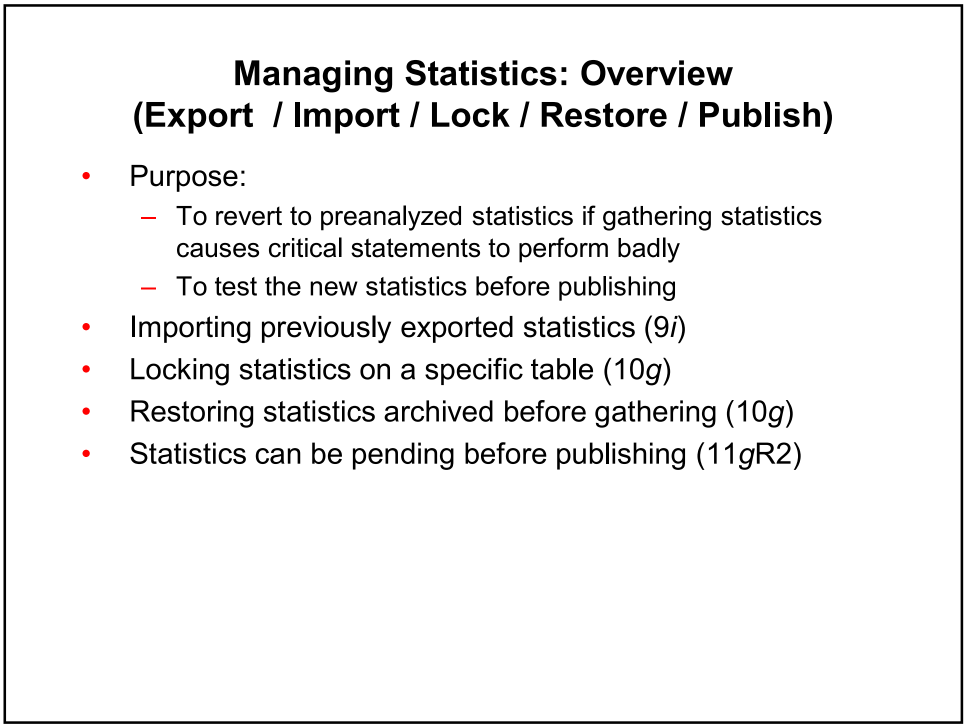
## Optimizer Dynamic Sampling at Work

- Sampling is done at compile time.
- If a query benefits from dynamic sampling:
  - A recursive SQL statement is executed to sample data.
  - The number of blocks sampled depends on the `OPTIMIZER_DYNAMIC_SAMPLING` initialization parameter.
- During dynamic sampling, predicates are applied to the sample to determine selectivity.
- Use dynamic sampling when:
  - Sampling time is a small fraction of the execution time (like Data Warehouse, not OLTP).
  - Volatile data is used with `DELETE_*_STATS` and `LOCK_*_STATS`.
  - Correlated columns are used in the `WHERE` clause.
  - Global temporary tables are used.
  - The query is executed many times.
  - You believe a better plan can be found (during testing).



## `OPTIMIZER_DYNAMIC_SAMPLING`

- Dynamic session or system parameter.
- Can be set to a value from "0" to "10."
- "0" turns off dynamic sampling.
- "1" samples all unanalyzed tables, if an unanalyzed table:
  - Is joined to another table or appears in a subquery or nonmergeable view
  - Has no indexes
  - Has more than 32 blocks
- "2" samples all unanalyzed tables.
- The higher the value, the more aggressive application of sampling.
- Dynamic sampling is repeatable if no update activity occurred.



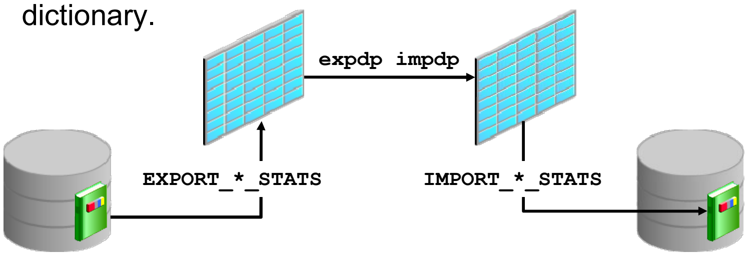
## Managing Statistics: Overview (Export / Import / Lock / Restore / Publish)

- Purpose:
  - To revert to preanalyzed statistics if gathering statistics causes critical statements to perform badly
  - To test the new statistics before publishing
- Importing previously exported statistics (9*i*)
- Locking statistics on a specific table (10*g*)
- Restoring statistics archived before gathering (10*g*)
- Statistics can be pending before publishing (11*g*R2)

## Export and Import Statistics

Use DBMS_STATS procedures:

- CREATE_STAT_TABLE creates the statistics table.
- EXPORT_*_STATS moves the statistics to the statistics table.
- Use Data Pump to move the statistics table.
- IMPORT_*_STATS moves the statistics to the data dictionary.



## Locking Statistics

- Prevents automatic gathering
- Is mainly used for volatile tables:
  - Lock without statistics implies dynamic sampling.

```
BEGIN
  DBMS_STATS.DELETE_TABLE_STATS('OE','ORDERS');
  DBMS_STATS.LOCK_TABLE_STATS('OE','ORDERS');
END;
```

  - Lock with statistics for representative values.

```
BEGIN
  DBMS_STATS.GATHER_TABLE_STATS('OE','ORDERS');
  DBMS_STATS.LOCK_TABLE_STATS('OE','ORDERS');
END;
```

- The FORCE argument overrides statistics locking.

```
SELECT stattype_locked FROM dba_tab_statistics;
```

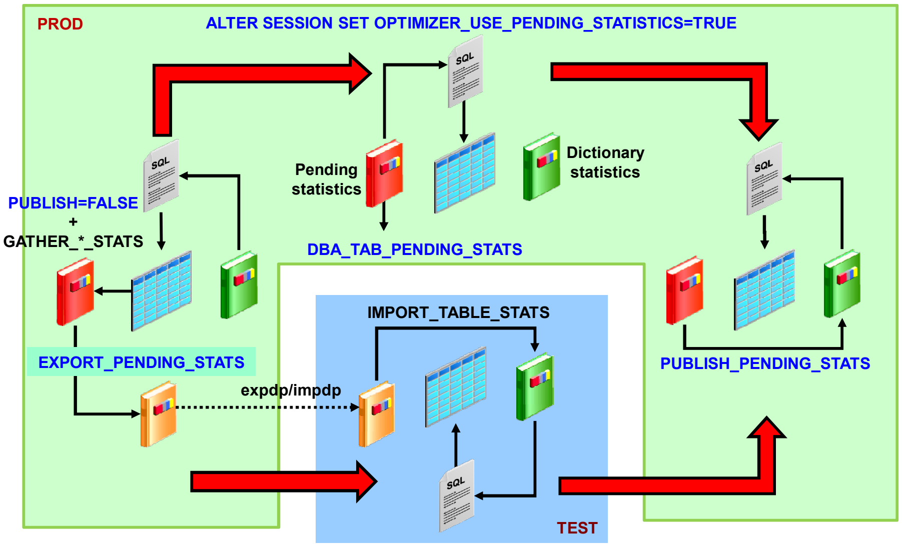
## Restoring Statistics

- Past statistics may be restored with the DBMS_STATS.RESTORE_*_STATS procedures.

```
BEGIN
 DBMS_STATS.RESTORE_TABLE_STATS(
   OWNNAME=>'OE', TABNAME=>'INVENTORIES',
   AS_OF_TIMESTAMP=>'15-JUL-10 09.28.01.597526000 AM -05:00');
END;
```

- Statistics are automatically stored:
  - With the timestamp in DBA_TAB_STATS_HISTORY
  - When collected with DBMS_STATS procedures
- Statistics are purged:
  - When the STATISTICS_LEVEL is set to TYPICAL or ALL automatically
  - After 31 days or time defined by DBMS_STATS.ALTER_STATS_HISTORY_RETENTION

## Deferred Statistics Publishing: Overview



## Deferred Statistics Publishing: Example

```
exec dbms_stats.set_table_prefs('SH','CUSTOMERS','PUBLISH','false');
```

①

```
exec dbms_stats.gather_table_stats('SH','CUSTOMERS');
```

②

```
alter session set optimizer_use_pending_statistics = true;
```

③

Execute your workload from the same session. ④

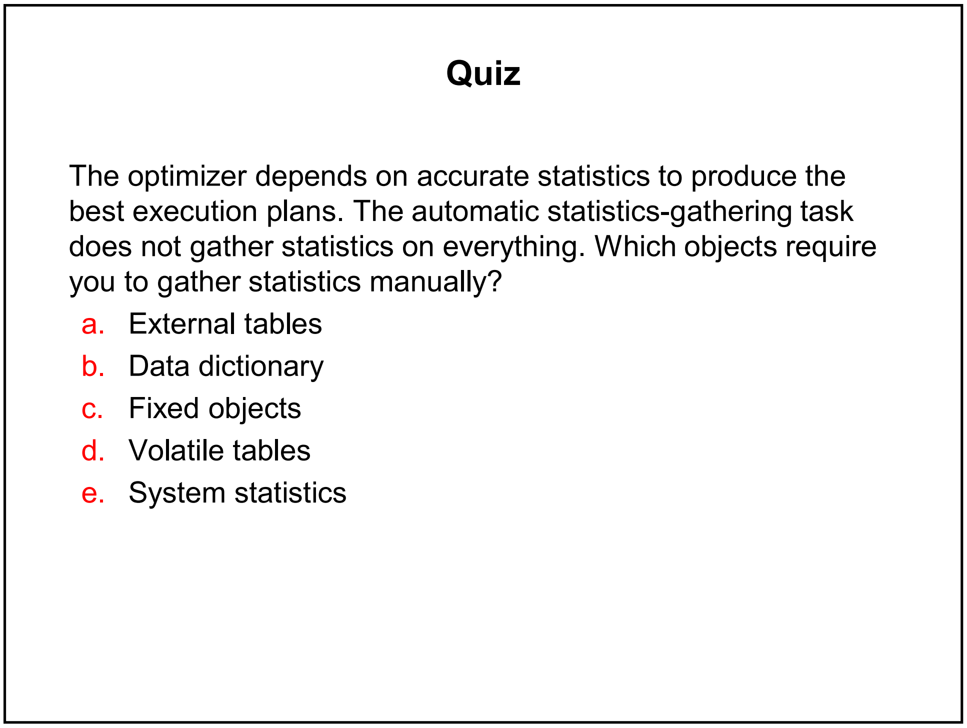
```
exec dbms_stats.publish_pending_stats('SH','CUSTOMERS');
```

⑤

## Quiz

When there are no statistics for an object being used in a SQL statement, the optimizer uses:

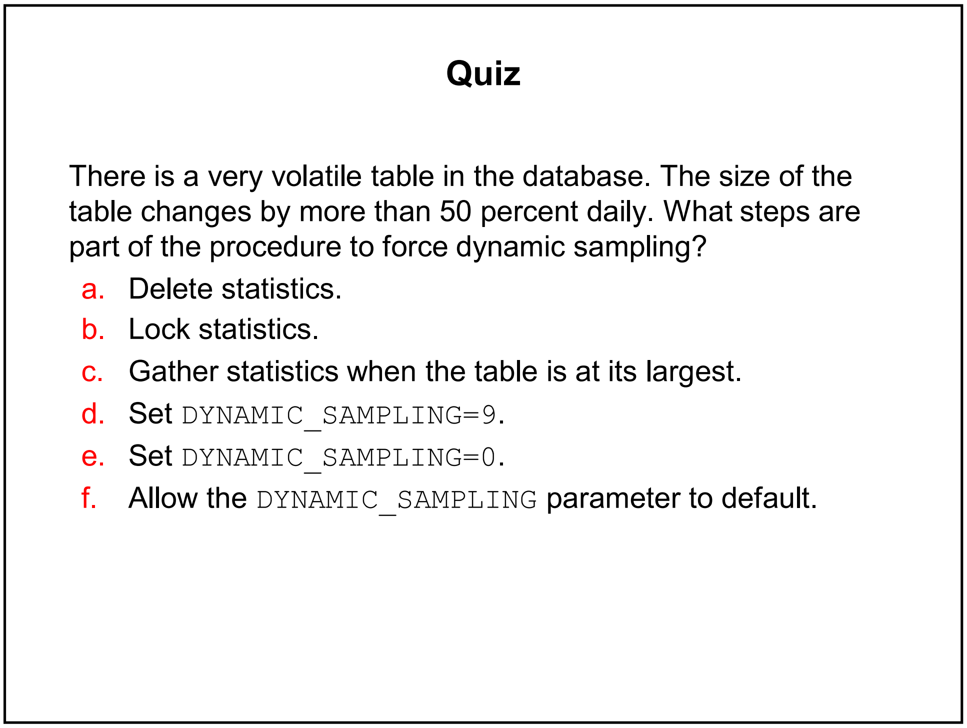
a. Rule-based optimization
b. Dynamic sampling
c. Fixed values
d. Statistics gathered during the parse phase
e. Random values



## Quiz

The optimizer depends on accurate statistics to produce the best execution plans. The automatic statistics-gathering task does not gather statistics on everything. Which objects require you to gather statistics manually?

a. External tables
b. Data dictionary
c. Fixed objects
d. Volatile tables
e. System statistics



## Quiz

There is a very volatile table in the database. The size of the table changes by more than 50 percent daily. What steps are part of the procedure to force dynamic sampling?

a. Delete statistics.
b. Lock statistics.
c. Gather statistics when the table is at its largest.
d. Set `DYNAMIC_SAMPLING=9`.
e. Set `DYNAMIC_SAMPLING=0`.
f. Allow the `DYNAMIC_SAMPLING` parameter to default.

## Summary

In this lesson, you should have learned how to:

- Describe optimizer statistics
  - Table statistics
  - Index statistics
  - Column statistics (histogram)
  - Column statistics (extended statistics)
  - System statistics
- Gather optimizer statistics
- Set statistic preferences
- Use dynamic sampling
- Manage optimizer statistics
- Discuss optimizer statistics best practices

## Practice 10: Overview

This practice covers the following topics:

- Using system statistics
- Using automatic statistics gathering