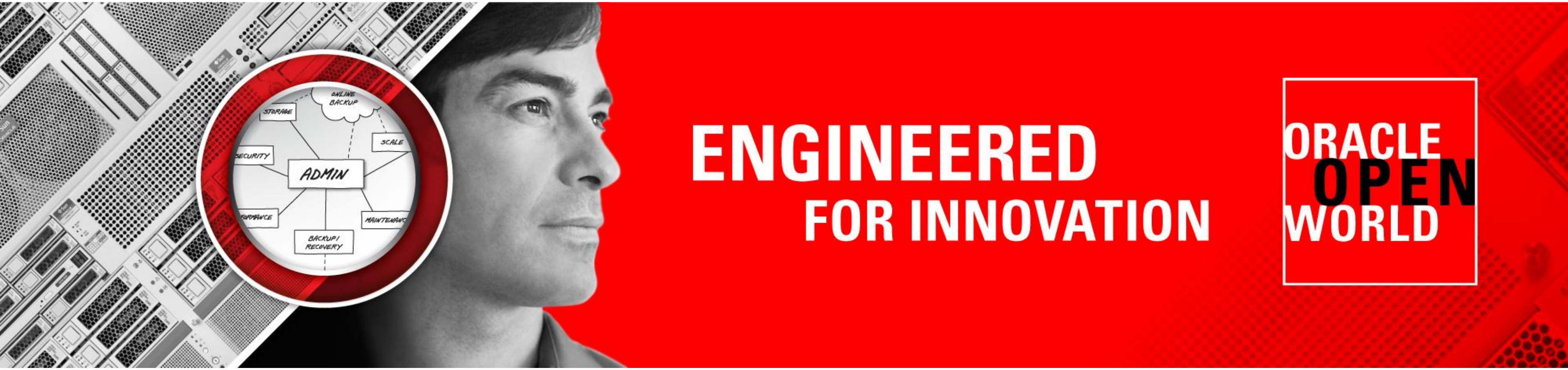
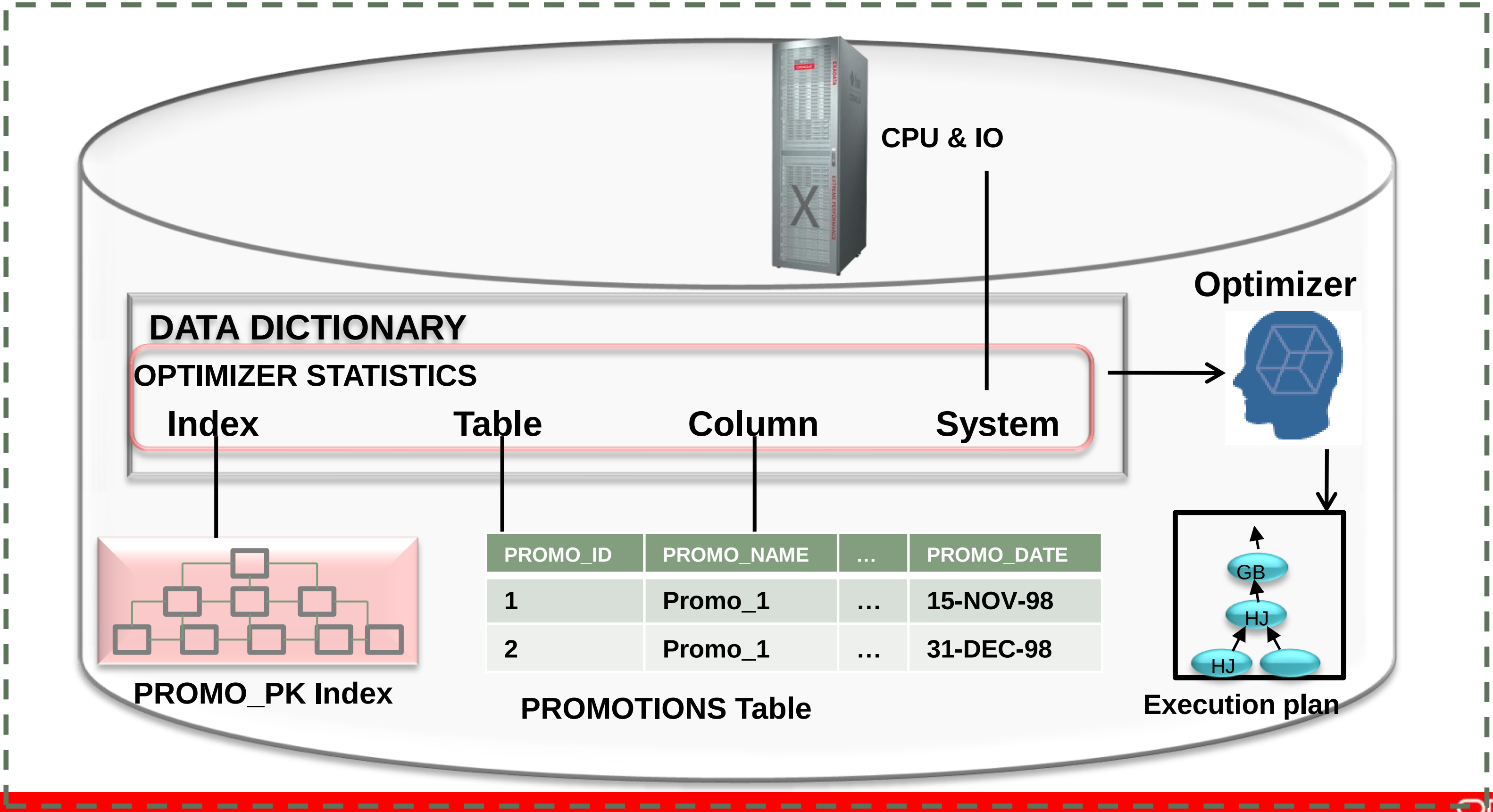
ORACLE®


ONLINE BACKUP
STORAGE
SCALE
SECURITY
ADMIN
FORMANCE
MAINTENANC
BACKUP/ RECOVERY
ENGINEERED
FOR INNOVATION
ORACLE
OPEN
WORLD

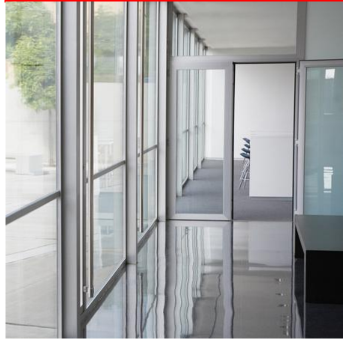
Optimizer Statistics



PROMO_ID	PROMO_NAME	…	PROMO_DATE
1	Promo_1	…	15-NOV-98
2	Promo_1	…	31-DEC-98

PROMOTIONS Table

Agenda



- How to gather statistics
- What basic statistics to collect
- Additional types of statistics
- When to gather statistics
- Statistics gathering performance
- When not to gather statistics

How to Gather Statistics

Use DBMS_STATS Package

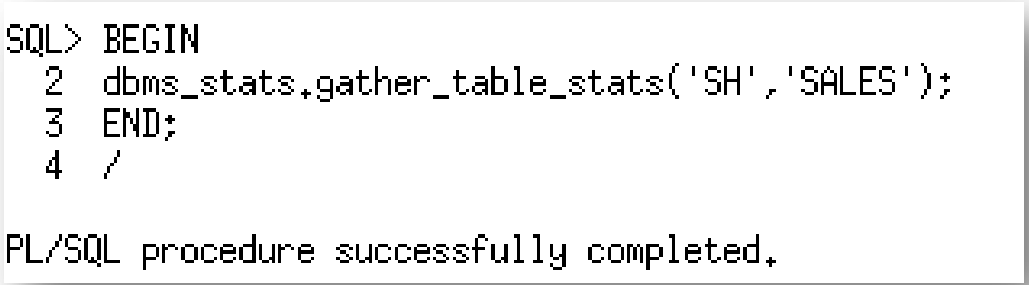
- Analyze command is deprecated
  - Only good for row chaining

- The GATHER_*_STATS procedures take 13 parameters
  - Ideally you should only set the first 2-3 parameters
    - SCHEMA NAME
    - TABLE NAME
    - PARTITION NAME

How to Gather Statistics

Use DBMS_STATS Package

- Your gather statistics commands should be this simple



```
SQL> BEGIN
  2  dbms_stats.gather_table_stats('SH','SALES');
  3  END;
  4  /

PL/SQL procedure successfully completed.
```

How to Gather Statistics

Changing Default Parameter Values for Gathering Statistics

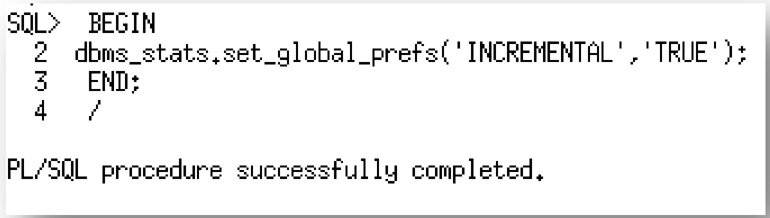
- Occasionally default parameter values may need to change
- For example - features not automatically on by default
  - Incremental Statistics
    - Ability to accurate generate global statistics from partition level statistics
    - Controlled by the parameter INCREMENTAL (default is FALSE)
  - Concurrent Statistics Gathering
    - Ability to gather statistics on multiple objects concurrently under a GATHER_SCHEMA_STATS command
    - Controlled by the parameter CONCURRENT (default is FALSE)

ORACLE

How to Gather Statistics

Changing Default Parameter Values for Gathering Statistics

- Can change the default value at the global level
  - DBMS_STATS.SET_GLOBAL_PREF
  - This changes the value for all existing objects and any new objects



```
SQL>  BEGIN
  2  dbms_stats.set_global_prefs('INCREMENTAL','TRUE');
  3   END;
  4   /

PL/SQL procedure successfully completed.
```

- Can change the default value at the table level
  - DBMS_STATS.SET_TABLE_PREF

How to Gather Statistics

Changing Default Parameter Values for Gathering Statistics

- Can change the default value at the schema level
  - DBMS_STATS.SET_SCHEMA_PREF
  - Current objects in the schema only
  - New objects pick up global preferences
- Can change the default value at the database level
  - DBMS_STATS.SET_DATABASE_PREF
  - Current objects in the Database only
  - New objects pick up global preferences

How to Gather Statistics

Changing Default Parameter Values for Gathering Statistics

- The following parameter defaults can be changed:

- CASCADE
- CONCURRENT
- DEGREE
- ESTIMATE_PERCENT
- METHOD_OPT
- NO_INVALIDATE
- GRANULARITY
- PUBLISH
- INCREMENTAL
- STALE_PERCENT
- AUTOSTATS_TARGET (SET_GLOBAL_PREFS only)

How to Gather Statistics

Sample Size

- # 1 most commonly asked question
  - “*What sample size should I use?”*
- Controlled by ESTIMATE_PRECENT parameter
- From 11g onwards use default value AUTO_SAMPLE_SIZE
  - New hash based algorithm
  - Speed of a 10% sample
  - Accuracy of 100% sample

How to Gather Statistics

Sample Size

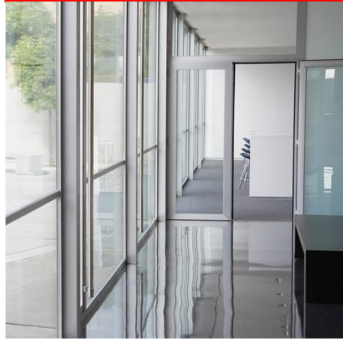
- Speed of a 10% sample

Run Num	AUTO_SAMPLE_SIZE	10% SAMPLE	100% SAMPLE
1	00:02:21.86	00:02:31.56	00:08:24.10
2	00:02:38.11	00:02:49.49	00:07:38.25
3	00:02:39.31	00:02:38.55	00:07:37.83

- Accuracy of 100% sample

Column Name	NDV with AUTO_SAMPLE_SIZE	NDV with 10% SAMPLE	NDV with 100% SAMPLE
C1	59852	31464	60351
C2	1270912	608544	1289760
C3	768384	359424	777942

Agenda



- How to gather statistics
- What basic statistics to collect
- Additional types of statistics
- When to gather statistics
- Statistics gathering performance
- When not to gather statistics

What basic statistics to collect

- By default the following basic table & column statistic are collected
  - Number of Rows
  - Number of blocks
  - Average row length
  - Number of distinct values
  - Number of nulls in column
- Index statistics are automatically gathered during creation and maintained by GATHER_TABLE_STATS and include
  - Number of leaf blocks
  - Branch Levels
  - Clustering factor

What basic statistics to collect

Histograms

- Histograms tell Optimizer about the data distribution in a Column
- Creation controlled by METHOD_OPT parameter
- Default create histogram on any column that has been used in the WHERE clause or GROUP BY of a statement AND has a data skew
- Relies on column usage information gathered at compilation time and stored in SYS.COL_USAGE$
- Two types of histograms
  - Frequency
  - Height-balanced

Creating a Height-Balance Histogram

Step 1: SELECT cust_city_id FROM customers ORDER BY cust_city_id;

Row count	CUST_CITY_ID
1	**51040**
2	**51040**
:	**:**
219	**51043**
:	**:**
5256	**51166**
:	**:**
5475	**51166**
:	**:**
55500	**52531**

Creating a Height-Balance Histogram

Step 2: Assign an equal number of rows per bucket

Row count	CUST_CITY_ID	
1	**51040**	
2	**51040**	
:	**:**	
219	**51043**	Bucket 1 has end point 51043
:	**:**	
5256	**51166**	Bucket 24 has end point 51166
:	**:**	
5475	**51166**	Bucket 25 has end point 51166
:	**:**	
55500	**52531**	Bucket 254 has end point 51531

Creating a Height-Balance Histogram

Step 3: If endpoint of 1st bucket is not min value add 0 bucket

Row count	CUST_CITY_ID	
1	**51040**	Bucket 0 has end point 51040
2	**51040**	
:	**:**	
219	**51043**	Bucket 1 has end point 51043
:	**:**	
5256	**51166**	Bucket 24 has end point 51166
:	**:**	
5475	**51166**	Bucket 25 has end point 51166
:	**:**	
55500	**52531**	Bucket 254 has end point 51531

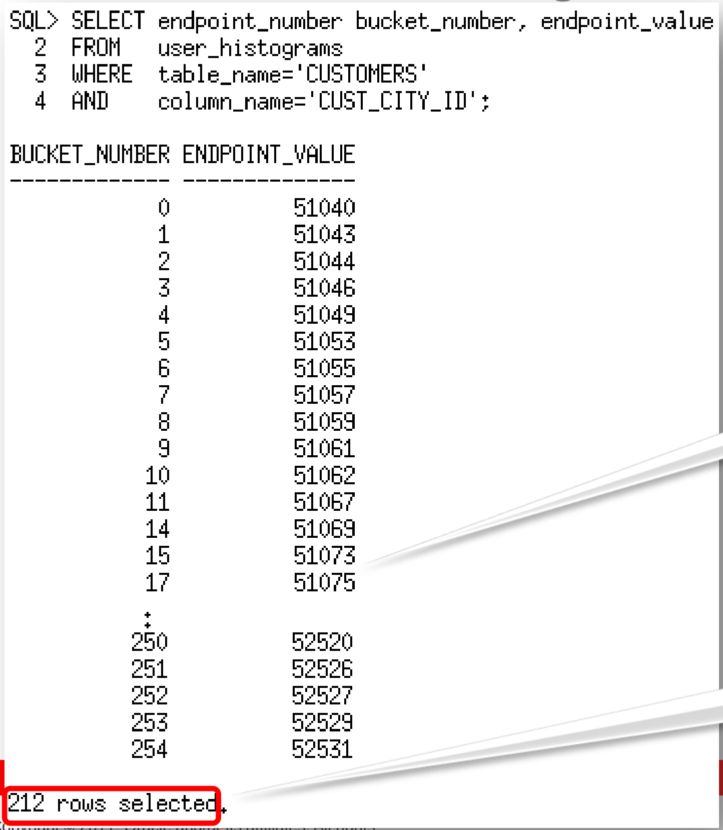
Creating a Height-Balance Histogram

Step 4: Compress duplicate buckets

Row count	CUST_CITY_ID		
1	**51040**		Bucket 0 has end point 51040
2	**51040**		
:	**:**		
219	**51043**		Bucket 1 has end point 51043
:	**:**		
5475	**51166**		Bucket 25 has end point 51166
:	**:**		
55500	**52531**		Bucket 254 has end point 51531

Monitoring Histograms

Information on Histograms found in USER_HISTOGRAMS



```
SQL> SELECT endpoint_number bucket_number, endpoint_value
  2  FROM   user_histograms
  3  WHERE  table_name='CUSTOMERS'
  4  AND    column_name='CUST_CITY_ID';

BUCKET_NUMBER ENDPOINT_VALUE
------------- --------------
            0          51040
            1          51043
            2          51044
            3          51046
            4          51049
            5          51053
            6          51055
            7          51057
            8          51059
            9          51061
           10          51062
           11          51067
           14          51069
           15          51073
           17          51075
          :
          250          52520
          251          52526
          252          52527
          253          52529
          254          52531

212 rows selected.
```

Bucket 16 is missing because buckets 15 & 16 had the same endpoint value

Not all 254 buckets used due to compression step

How the Optimizer uses Histograms

Optimizer used two different formulas depend on the popularity of the value

- Popular value means values that are the endpoint for two or more buckets
- Formula used is:

$$\frac{\text{Number of endpoint buckets}}{\text{total number of buckets}} \times \text{number of rows in the table}$$

How the Optimizer uses Histograms

Optimizer used two different formulas depend on the popularity of the value

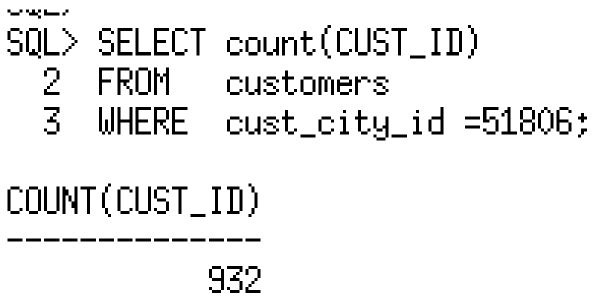
- Non-popular value means values that are the endpoint for only one bucket or are not an endpoint at all
- Formula used is:

  DENSITY X number of rows in the table

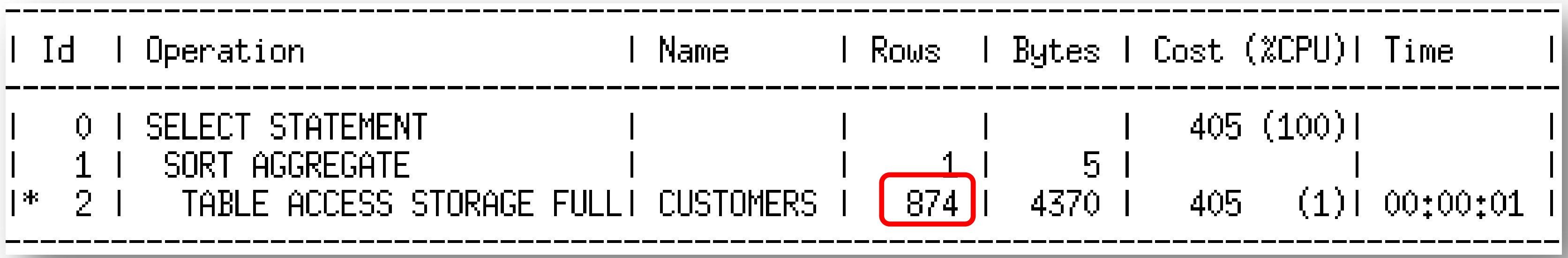
NOTE: Density from 10.2.0.4 is calculated on the fly based on histogram information and is not the value show in USER_HISTOGRAMS

How the Optimizer uses Histograms

Popular values use histogram information



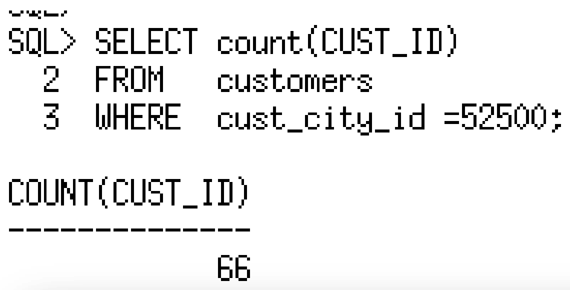
```
SQL> SELECT count(CUST_ID)
  2  FROM   customers
  3  WHERE  cust_city_id =51806;

COUNT(CUST_ID)
--------------
           932
```



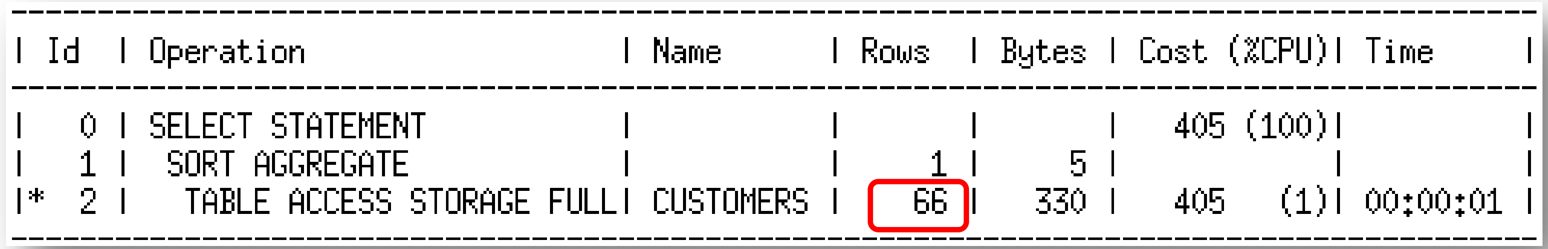
```
-------------------------------------------------------------------------------------
| Id  | Operation                  | Name      | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT           |           |       |       |   405 (100)|          |
|   1 |  SORT AGGREGATE            |           |     1 |     5 |            |          |
|*  2 |   TABLE ACCESS STORAGE FULL| CUSTOMERS |   874 |  4370 |   405   (1)| 00:00:01 |
-------------------------------------------------------------------------------------
```

How the Optimizer uses Histograms

Non-popular values use Density



```
SQL>
SQL> SELECT count(CUST_ID)
  2  FROM   customers
  3  WHERE  cust_city_id =52500;

COUNT(CUST_ID)
--------------
            66
```



```
------------------------------------------------------------------------------------------
| Id  | Operation                  | Name       | Rows  | Bytes | Cost (%CPU)| Time     |
------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT           |            |       |       |   405 (100)|          |
|   1 |  SORT AGGREGATE            |            |     1 |     5 |            |          |
|*  2 |   TABLE ACCESS STORAGE FULL| CUSTOMERS  |    66 |   330 |   405   (1)| 00:00:01 |
------------------------------------------------------------------------------------------
```

How to Gather Statistics

Why people hate histograms

- Two main hurt points with Histograms

1. Bind peeking interacts with histograms
2. Nearly popular values

Bind Peeking and Histograms Prior to 11g

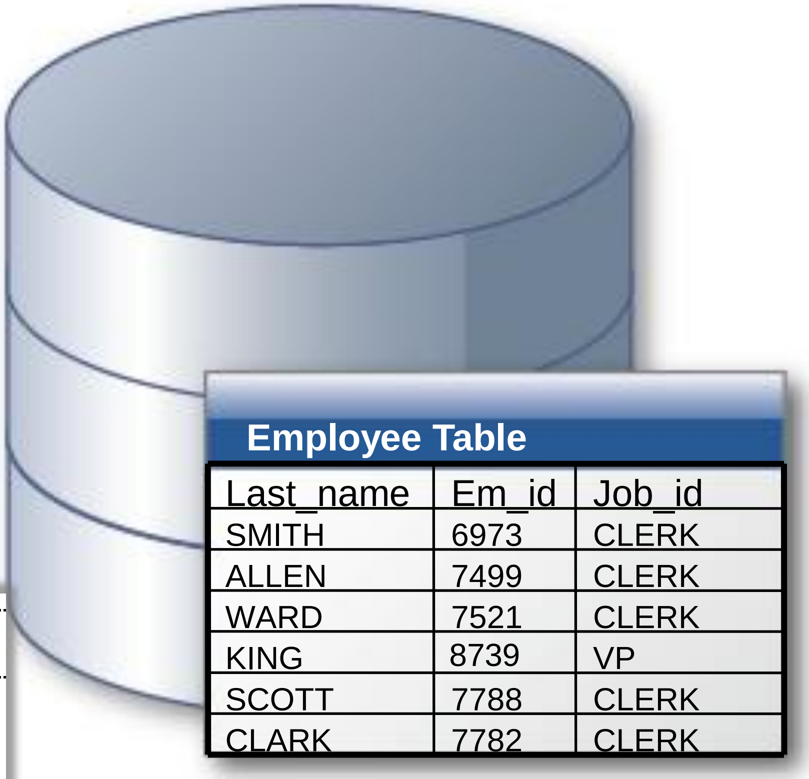
- The optimizer peeks bind values during plan selection
- Initial value of the binds determines the plan
- Same execution plan shared regardless of future bind values
- Potential for plan changes when the first value peeked is popular or unpopular

Bind Peeking and Histograms Prior to 11g

SELECT * FROM Employee

WHERE Job_id = :B1;

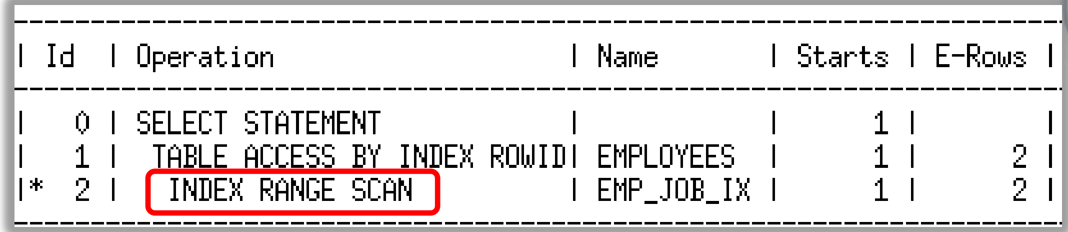
NAME	ENUM	JOB
KOCHHAR	101	AD_VP
DE HAAN	102	AD_VP



Employee Table

Last_name	Em_id	Job_id
SMITH	6973	CLERK
ALLEN	7499	CLERK
WARD	7521	CLERK
KING	8739	VP
SCOTT	7788	CLERK
CLARK	7782	CLERK

If the value of bind B1 is AD_VP at hard parse then an index range scan will be selected because only 2 rows returned



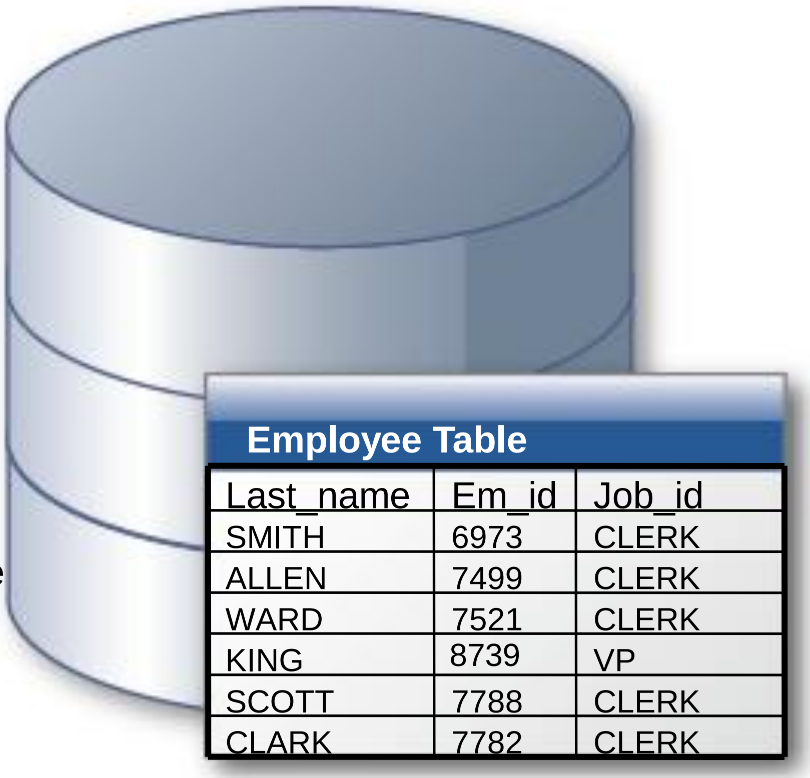
```
-----------------------------------------------------------------------
| Id  | Operation                   | Name       | Starts | E-Rows |
-----------------------------------------------------------------------
|   0 | SELECT STATEMENT            |            |      1 |        |
|   1 |  TABLE ACCESS BY INDEX ROWID| EMPLOYEES  |      1 |      2 |
|*  2 |   INDEX RANGE SCAN          | EMP_JOB_IX |      1 |      2 |
-----------------------------------------------------------------------
```

Bind Peeking and Histograms Prior to 11g

SELECT * FROM Employee

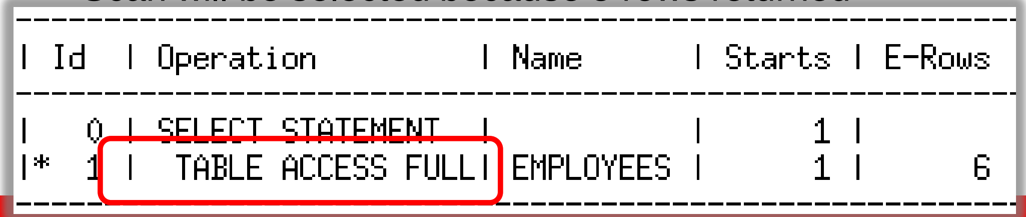
WHERE Job_id = :B1;

NAME	ENUM	JOB
SMITH	6973	CLERK
ALLEN	7499	CLERK
WARD	2021	CLERK
CLARK	7782	CLERK
BROWN	4040	CLERK



Last_name	Em_id	Job_id
SMITH	6973	CLERK
ALLEN	7499	CLERK
WARD	7521	CLERK
KING	8739	VP
SCOTT	7788	CLERK
CLARK	7782	CLERK

If the value of bind B1 is CLERK at hard parse then an Full Table Scan will be selected because 5 rows returned



```
-------------------------------------------------------------
| Id  | Operation           | Name      | Starts | E-Rows
-------------------------------------------------------------
|   0 | SELECT STATEMENT    |           |      1 |
|*  1 |  TABLE ACCESS FULL| EMPLOYEES |      1 |       6
-------------------------------------------------------------
```

Solutions for Bind Peeking and Histograms Prior to 11g

- Applications that only have statements with binds
  - Drop histogram using `DBMS_STATS.DELETE_COL_STATS`
  - Use `DBMS_STATS.SET_PARM` to change default setting for method_opt parameter to prevent histogram from being created
  - Re-gather statistics on the table without histogram

- Applications that have statements with bind and literals
  - Switch off bind peeking *_optim_peek_user_binds* = *false*

With Adaptive Cursor Sharing

You Can Have BOTH Plans For Our Statement

```
SELECT * FROM Employee
WHERE Job_id = :B1;
```

B1 = CLERK

NAME	ENUM	JOB
SMITH	6973	CLERK
ALLEN	7499	CLERK
WARD	2021	CLERK
CLARK	7782	CLERK
BROWN	4040	CLERK

Full Table Scan is optimal

B1 = AD_VP

NAME	ENUM	JOB
KOCHHAR	101	AD_VP
DE HAAN	102	AD_VP

Index Access is optimal

Peek all binds & take the plan that is optimal for each bind set

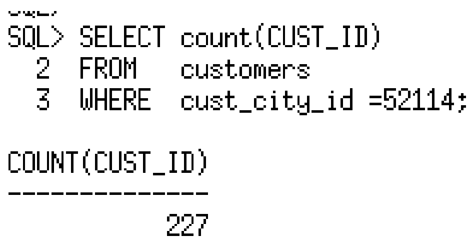
ORACLE

Adaptive Cursor Sharing

- Share the plan when binds values are “equivalent”
  - Plans are marked with selectivity range
  - If current bind values fall within range they use the same plan
- Create a new plan if binds are not equivalent
  - Generating a new plan with a different selectivity range
- Controlled by init.ora parameter *_optim_peek_user_binds*
- Monitoring - V$SQL has 2 new columns
  - IS_BIND_SENSITIVE - Optimizer believes the plan may depend on the value of bind
  - IS_BIND_AWARE - Multiple execution plans exist for this statement

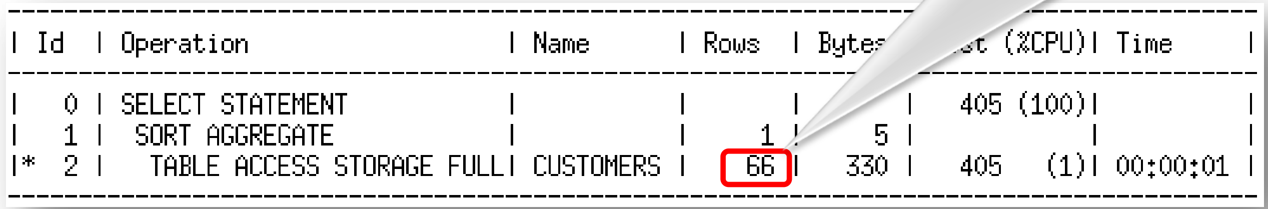
Nearly Popular Values

- Nearly popular value means the value is classified as non-popular but the density calculation is not accurate for them



```
SQL> SELECT count(CUST_ID)
  2  FROM   customers
  3  WHERE  cust_city_id =52114;

COUNT(CUST_ID)
--------------
           227
```

Same estimate used as for non-popular. Here density is not good enough to get accurate cardinality estimate



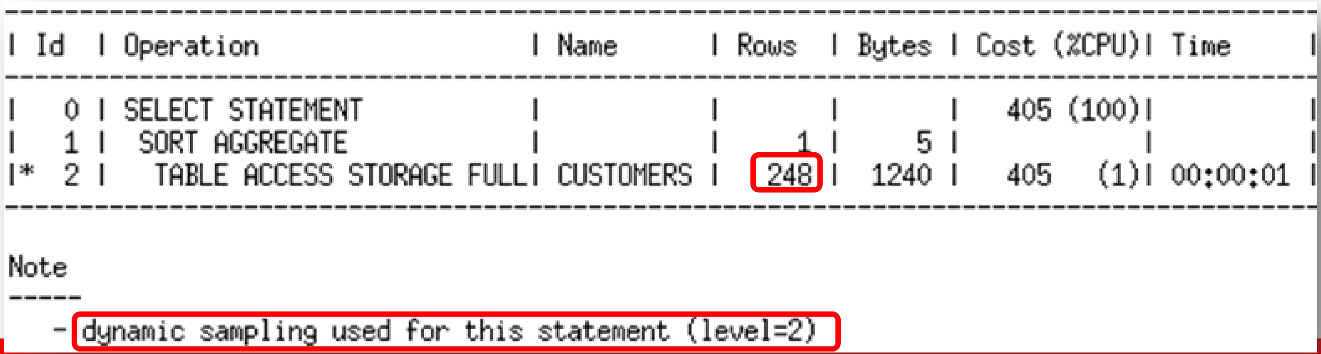
```
------------------------------------------------------------------------------------------
| Id  | Operation                 | Name      | Rows  | Byte       st (%CPU)| Time     |
------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT          |           |       |       |   405 (100)|          |
|   1 |  SORT AGGREGATE           |           |     1 |     5 |            |          |
|*  2 |   TABLE ACCESS STORAGE FULL| CUSTOMERS |    66 |   330 |   405   (1)| 00:00:01 |
------------------------------------------------------------------------------------------
```

Nearly Popular Values

Solution Dynamic Sampling

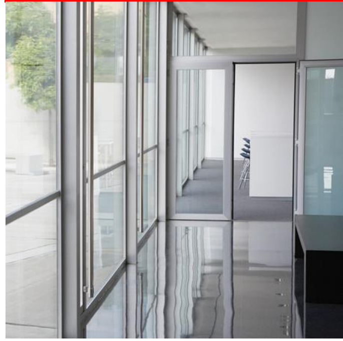
- To get an accurate cardinality estimate for nearly popular values use dynamic sampling

```
SQL> SELECT /*+ dynamic_sampling(customers 2) */ count(CUST_ID)
  2  FROM   customers
  3  WHERE  cust_city_id =52114;

COUNT(CUST_ID)
--------------
           227
```



```
------------------------------------------------------------------------------------
| Id  | Operation                  | Name      | Rows  | Bytes | Cost (%CPU)| Time     |
------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT           |           |       |       |   405 (100)|          |
|   1 |  SORT AGGREGATE            |           |     1 |     5 |            |          |
|*  2 |   TABLE ACCESS STORAGE FULL| CUSTOMERS |   248 |  1240 |   405   (1)| 00:00:01 |
------------------------------------------------------------------------------------

Note
-----
   - dynamic sampling used for this statement (level=2)
```

Agenda



- How to gather statistics
- What basic statistics to collect
- Additional types of statistics
- When to gather statistics
- Statistics gathering performance
- When not to gather statistics

Additional Types of Statistics

When Table and Column Statistics are not enough

- Two types of Extended Statistics
  - Column groups statistics
    - Column group statistics useful when multiple column from the same table are used in where clause predicates
  - Expression statistics
    - Expression statistics useful when a column is used as part of a complex expression in where clause predicate
- Can be manually or automatically created
- Automatically maintained when statistics are gathered on the table

Extended Statistics – Column Group Statistics

SELECT * FROM vehicles

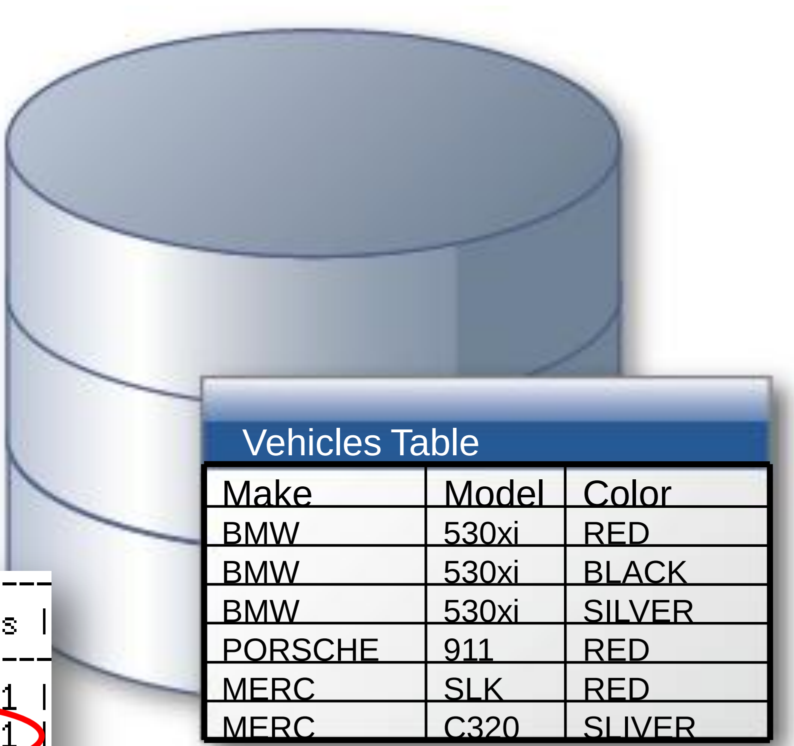
WHERE model = ‘530xi’

AND color = 'RED’;

MAKE	MODEL	COLOR
BMW	530xi	RED

$$\text{Cardinality} = \#\text{ROWS} * \frac{1}{\text{NDV c1}} * \frac{1}{\text{NDV c2}} \Rightarrow 12 * \frac{1}{4} * \frac{1}{3} = 1$$

```
-----------------------------------------------------------------------
| Id  | Operation          | Name     | Starts  | E-Rows  | A-Rows |
-----------------------------------------------------------------------
|   0 | SELECT STATEMENT   |          |       1 |         |      1 |
|*  1 |  TABLE ACCESS FULL | VEHICLES |       1 |       1 |      1 |
-----------------------------------------------------------------------
```



Vehicles Table

Make	Model	Color
BMW	530xi	RED
BMW	530xi	BLACK
BMW	530xi	SILVER
PORSCHE	911	RED
MERC	SLK	RED
MERC	C320	SLIVER

Extended Statistics – Column Group Statistics

SELECT * FROM vehicles

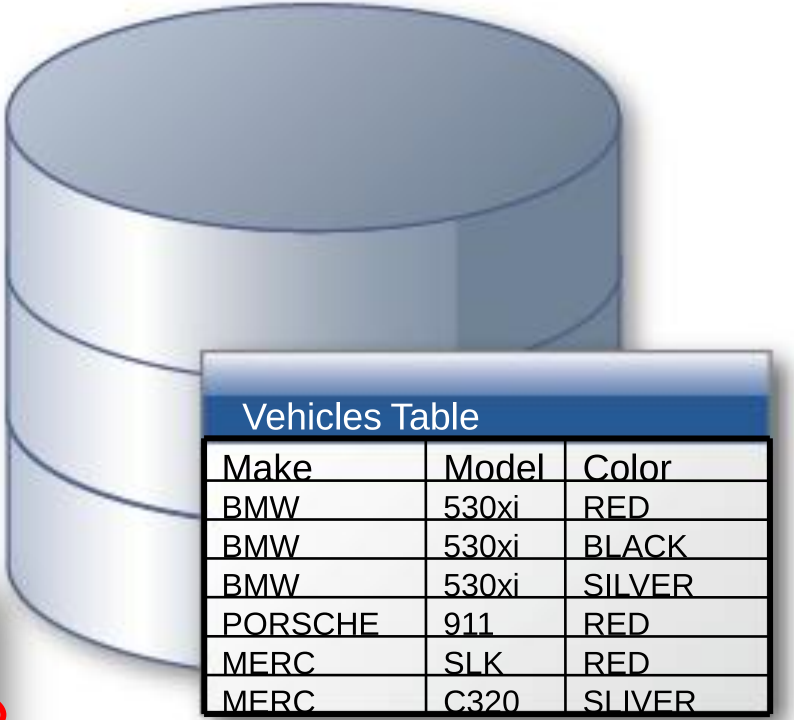
WHERE model = ‘530xi’

AND make = ‘BMW’;

MAKE	MODEL	COLOR
BMW	530xi	RED
BMW	530xi	BLACK
BMW	530xi	SLIVER

$$\text{Cardinality} = \#\text{ROWS} * \frac{1}{\text{NDV c1}} * \frac{1}{\text{NDV c2}} \Rightarrow 12 * \frac{1}{4} * \frac{1}{3} = 1$$

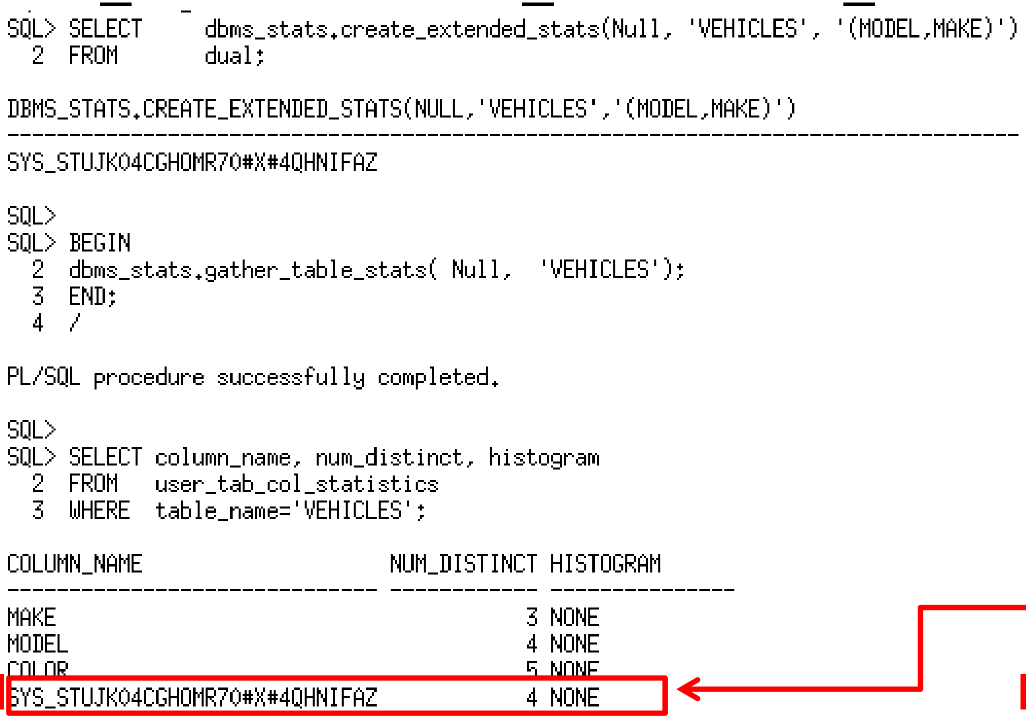
```
-----------------------------------------------------------------------
| Id  | Operation         | Name     | Starts | E-Rows | A-Rows |
-----------------------------------------------------------------------
|   0 | SELECT STATEMENT  |          |      1 |        |      3 |
|*  1 |  TABLE ACCESS FULL| VEHICLES |      1 |      1 |      3 |
-----------------------------------------------------------------------
```


Vehicles Table

Make	Model	Color
BMW	530xi	RED
BMW	530xi	BLACK
BMW	530xi	SILVER
PORSCHE	911	RED
MERC	SLK	RED
MERC	C320	SLIVER

Extended Statistics – Column Group Statistics

- Create extended statistics on the Model & Make columns using DBMS_STATS.CREATE_EXTENDED_STATS



```
SQL> SELECT   dbms_stats.create_extended_stats(Null, 'VEHICLES', '(MODEL,MAKE)')
  2  FROM     dual;

DBMS_STATS.CREATE_EXTENDED_STATS(NULL,'VEHICLES','(MODEL,MAKE)')
----------------------------------------------------------------------------------
SYS_STUJKO4CGHOMR7O#X#4QHNIFAZ

SQL>
SQL> BEGIN
  2  dbms_stats.gather_table_stats( Null,  'VEHICLES');
  3  END;
  4  /

PL/SQL procedure successfully completed.

SQL>
SQL> SELECT column_name, num_distinct, histogram
  2  FROM   user_tab_col_statistics
  3  WHERE  table_name='VEHICLES';

COLUMN_NAME                    NUM_DISTINCT HISTOGRAM
------------------------------ ------------ ---------------
MAKE                                      3 NONE
MODEL                                     4 NONE
COLOR                                     5 NONE
SYS_STUJKO4CGHOMR7O#X#4QHNIFAZ            4 NONE
```

New Column with system generated name

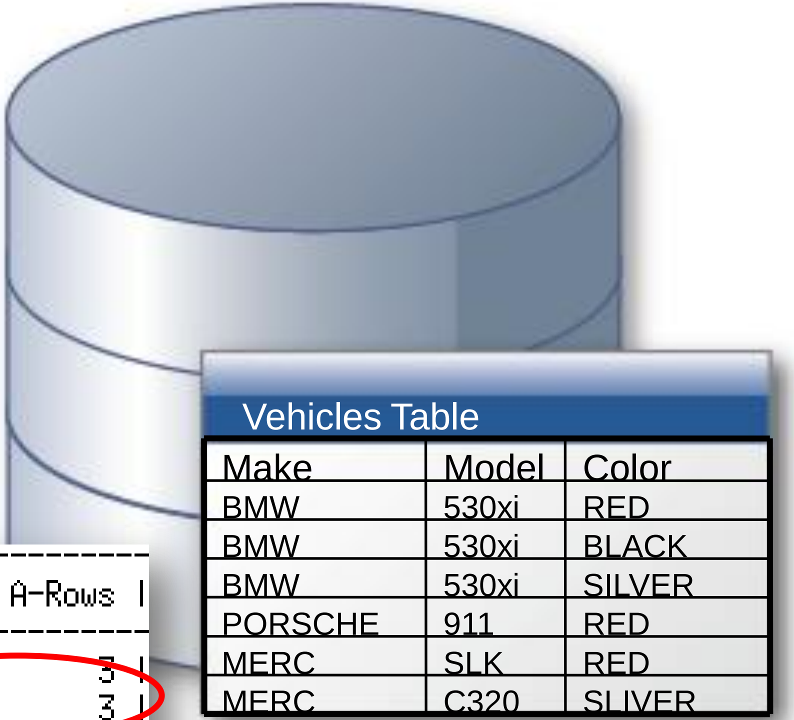
Extended Statistics – Column Group Statistics

SELECT * FROM vehicles

WHERE model = ‘530xi’

AND make = ‘BMW’;

MAKE	MODEL	COLOR
BMW	530xi	RED
BMW	530xi	BLACK
BMW	530xi	SLIVER



Vehicles Table

Make	Model	Color
BMW	530xi	RED
BMW	530xi	BLACK
BMW	530xi	SILVER
PORSCHE	911	RED
MERC	SLK	RED
MERC	C320	SLIVER

Cardinality calculated using column group statistics

```
--------------------------------------------------------------------
| Id  | Operation          | Name      | Starts | E-Rows | A-Rows |
--------------------------------------------------------------------
|   0 | SELECT STATEMENT   |           |      1 |        |      3 |
|*  1 |  TABLE ACCESS FULL| VEHICLES |      1 |      3 |      3 |
--------------------------------------------------------------------
```

Extended Statistics – Expression Statistics example

SELECT *

FROM Customers

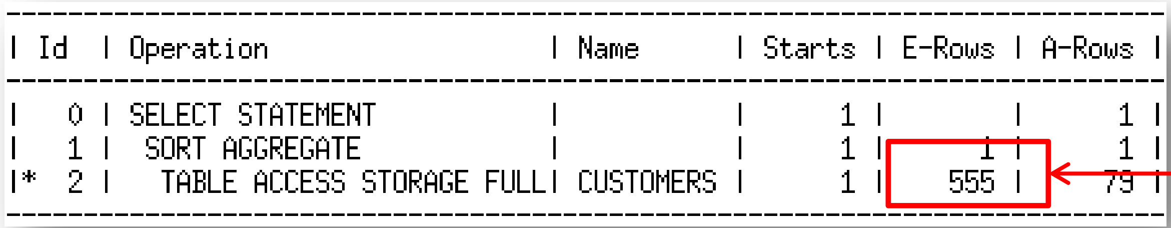
WHERE UPPER(CUST_LAST_NAME) = ‘SMITH’;

- Optimizer doesn’t know how function affects values in the column
- Optimizer guesses the cardinality to be 1% of rows

SELECT count(*) FROM customers;

<u>COUNT(*)</u>

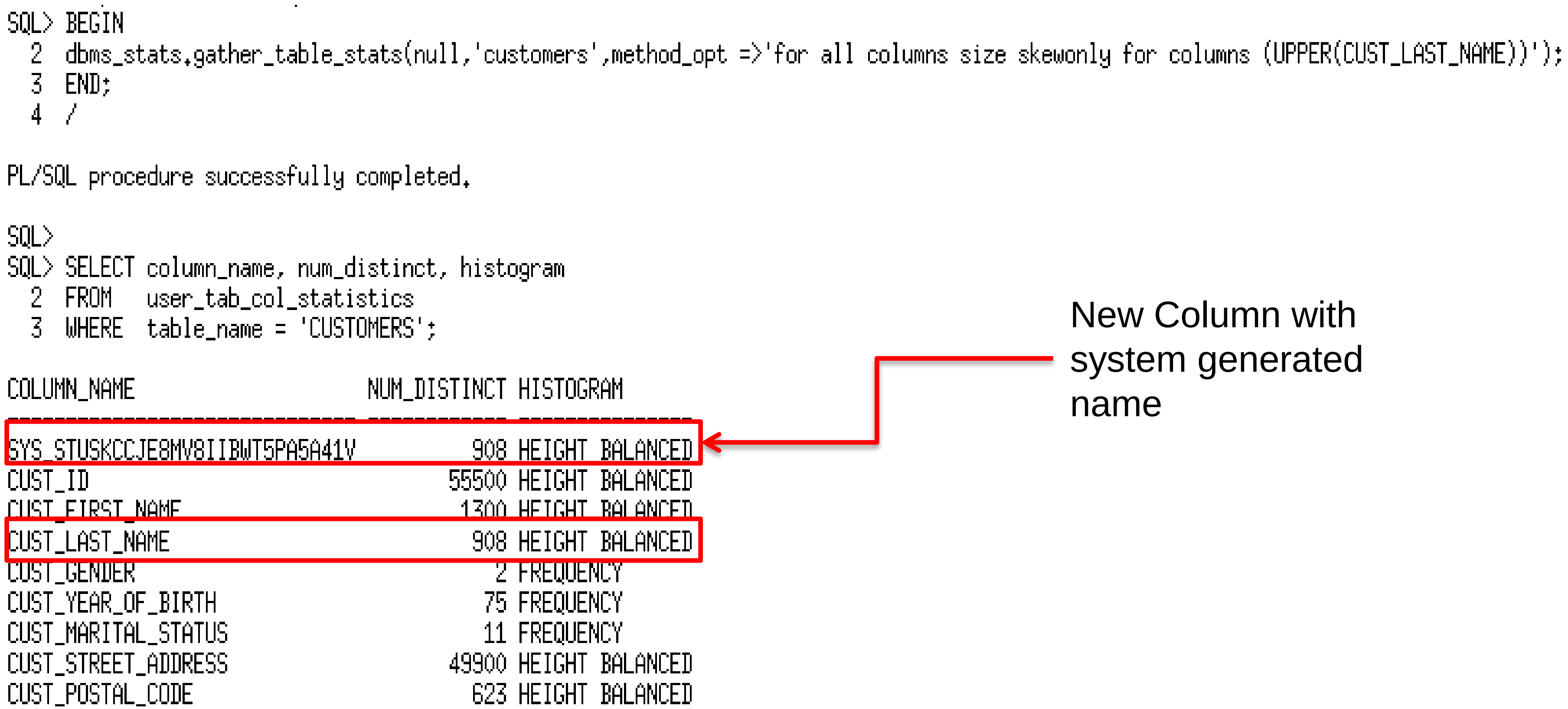
55500



```
----------------------------------------------------------------------------
| Id  | Operation                  | Name      | Starts | E-Rows | A-Rows |
----------------------------------------------------------------------------
|   0 | SELECT STATEMENT           |           |      1 |        |      1 |
|   1 |  SORT AGGREGATE            |           |      1 |      1 |      1 |
|*  2 |   TABLE ACCESS STORAGE FULL| CUSTOMERS |      1 |    555 |     79 |
----------------------------------------------------------------------------
```

Cardinality estimate is 1% of the rows

Extended Statistics – Expression Statistics Solution



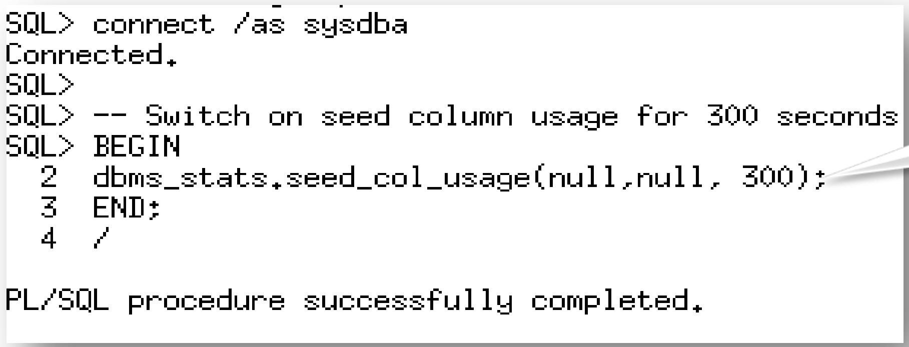
```
SQL> BEGIN
  2  dbms_stats.gather_table_stats(null,'customers',method_opt =>'for all columns size skewonly for columns (UPPER(CUST_LAST_NAME))');
  3  END;
  4  /

PL/SQL procedure successfully completed.

SQL>
SQL> SELECT column_name, num_distinct, histogram
  2  FROM   user_tab_col_statistics
  3  WHERE  table_name = 'CUSTOMERS';

COLUMN_NAME                    NUM_DISTINCT HISTOGRAM
------------------------------ ------------ ---------------
SYS_STUSKCCJE8MV8IIBWT5PA5A41V          908 HEIGHT BALANCED
CUST_ID                               55500 HEIGHT BALANCED
CUST_FIRST_NAME                        1300 HEIGHT BALANCED
CUST_LAST_NAME                          908 HEIGHT BALANCED
CUST_GENDER                               2 FREQUENCY
CUST_YEAR_OF_BIRTH                       75 FREQUENCY
CUST_MARITAL_STATUS                      11 FREQUENCY
CUST_STREET_ADDRESS                   49900 HEIGHT BALANCED
CUST_POSTAL_CODE                        623 HEIGHT BALANCED
```

New Column with system generated name

Extended Statistics

Automatic Column Group Creation

1. Start column group usage capture



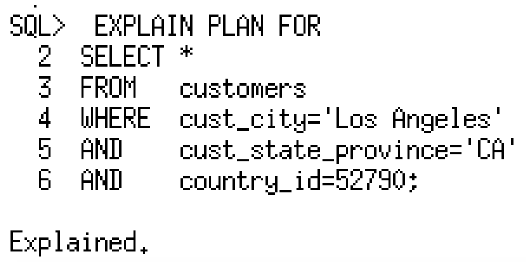
```
SQL> connect /as sysdba
Connected.
SQL>
SQL> -- Switch on seed column usage for 300 seconds
SQL> BEGIN
  2  dbms_stats.seed_col_usage(null,null, 300);
  3  END;
  4  /

PL/SQL procedure successfully completed.
```

Switches on monitoring for 300 seconds or the next 5 minutes. An statement executed will be monitored for columns used in the where and group by clauses

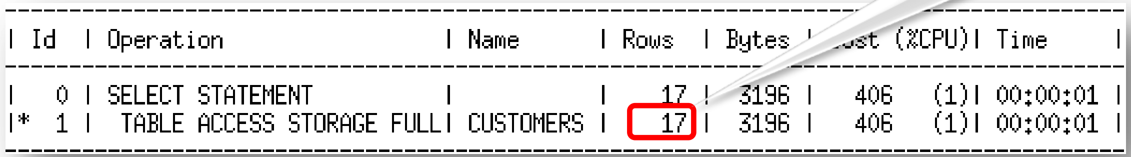
Extended Statistics

Automatic Column Group Creation

2. Run your workload



```
SQL>  EXPLAIN PLAN FOR
  2  SELECT *
  3  FROM   customers
  4  WHERE  cust_city='Los Angeles'
  5  AND    cust_state_province='CA'
  6  AND    country_id=52790;

Explained.
```



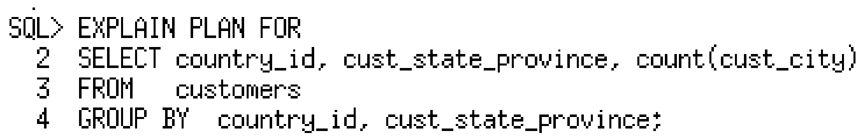
```
---------------------------------------------------------------------------------------
| Id  | Operation                  | Name      | Rows  | Bytes |  ost (%CPU)| Time     |
---------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT           |           |    17 |  3196 |   406   (1)| 00:00:01 |
|*  1 |  TABLE ACCESS STORAGE FULL| CUSTOMERS |    17 |  3196 |   406   (1)| 00:00:01 |
---------------------------------------------------------------------------------------
```

Actual number of rows returned by this query 932. Optimizer underestimates because it assumes each predicate will reduce num rows

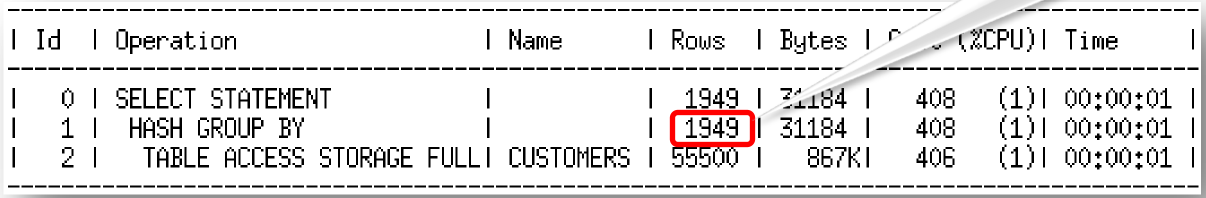
Extended Statistics

Automatic Column Group Creation

2. Run your workload



```
SQL> EXPLAIN PLAN FOR
  2  SELECT country_id, cust_state_province, count(cust_city)
  3  FROM   customers
  4  GROUP BY  country_id, cust_state_province;
```



```
--------------------------------------------------------------------------------
| Id  | Operation                  | Name      | Rows  | Bytes | Cost (%CPU)| Time     |
--------------------------------------------------------------------------------
|   0 | SELECT STATEMENT           |           |  1949 | 31184 |   408   (1)| 00:00:01 |
|   1 |  HASH GROUP BY             |           |  1949 | 31184 |   408   (1)| 00:00:01 |
|   2 |   TABLE ACCESS STORAGE FULL| CUSTOMERS | 55500 |  867K|   406   (1)| 00:00:01 |
--------------------------------------------------------------------------------
```

Actual number of rows returned by this query 145. Optimizer overestimates because it assumes no relationship between country and state

Extended Statistics

Automatic Column Group Creation

3. Check we have column usage information for our table

SQL> **SELECT dbms_stats.report_col_usage(user, 'customers') FROM dual;**

EQ means column was used in equality predicate in query 1

```
COLUMN USAGE REPORT FOR SH.CUSTOMERS
1. COUNTRY_ID                          : EQ
2. CUST_CITY                           : EQ
3. CUST_STATE_PROVINCE                 : EQ
4. (CUST_CITY, CUST_STATE_PROVINCE,   COUNTRY_ID)      : FILTER
5. (CUST_STATE_PROVINCE, COUNTRY_ID)   : GROUP_BY
```

FILTER means columns used together as filter predicates rather than join etc. Comes from query 1

GROUP_BY columns used in group by expression in query 2

Extended Statistics

Automatic Column Group Creation

4. Create extended stats for customers based on usage

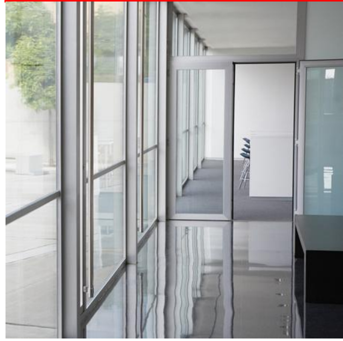
```
SQL> SELECT dbms_stats.create_extended_stats(user, 'customers')
              FROM dual;
```

```
EXTENSIONS FOR SH.CUSTOMERS
1.  (CUST_CITY, CUST_STATE_PROVINCE,   COUNTRY_ID):
   SYS_STUMZ$C3AIHLPBROI#SKA58H_N         created
2.  (CUST_STATE_PROVINCE, COUNTRY_ID)   :
    SYS_STU#S#WF25Z#QAHIHE#MOFFMM_      created
```

Column group statistics will now be automatically maintained every time you gather statistics on this table

Agenda

- How to gather statistics
- What basic statistics to collect
- Additional types of statistics
- When to gather statistics
- Statistics gathering performance
- When not to gather statistics



When to Gather Statistics

Automatic Statistics Gathering

- Oracle automatically collect statistics for all database objects, which are missing statistics or have stale statistics
- AutoTask run during a predefined maintenance window
- Internally prioritizes the database objects
  - Both user schema and dictionary tables
  - Objects that need updated statistics most are processed first
- Controlled by DBMS_AUTO_TASK_ADMIN package or via Enterprise Manager

Automatic Statistics Gathering

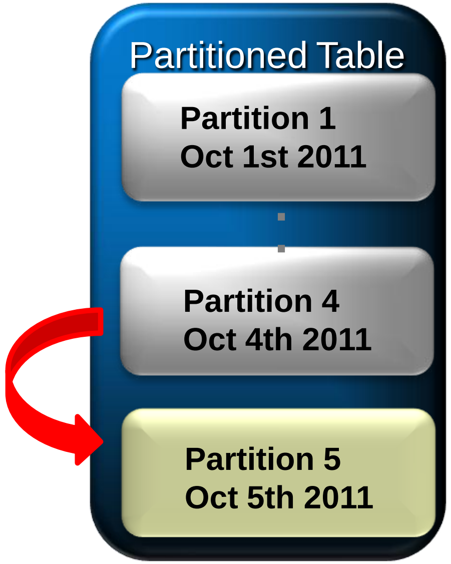
- If you want to disable auto job for application schema leaving it on for Oracle dictionary tables
- The scope of the auto job is controlled by the global preference AUTOSTATS_TARGET
- Possible values are

– AUTO	Oracle decides what tables need statistics (Default)
– All	Statistics gathered for all tables in the system
– ORACLE	Statistics gathered for only the dictionary tables

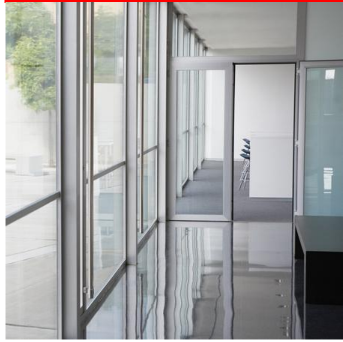
When to Gather Statistics

If the Auto Statistics Gather Job is not suitable

- After a large data load
  - As part of the ETL or ELT process gather statistics
- If trickle loading into a partition table
  - Used dbms.stats.copy_table_stats()
    - Copies stats from source partition
    - Adjust min & max values for partition column
      - Both partition & global statistics
    - Copies statistics of the dependent objects
      - Columns, local (partitioned) indexes* etc.
      - Does not update global indexes



Agenda



- How to gather statistics
- What basic statistics to collect
- Additional types of statistics
- When to gather statistics
- Statistics gathering performance
- When not to gather statistics

Statistics Gathering Performance

How to speed up statistics gathering

- Three parallel options to speed up statistics gathering
  - Inter object using parallel execution
  - Intra object using concurrency
  - The combination of Inter and Intra object
- Incremental statistics gathering for partitioned tables

Statistics Gathering Performance

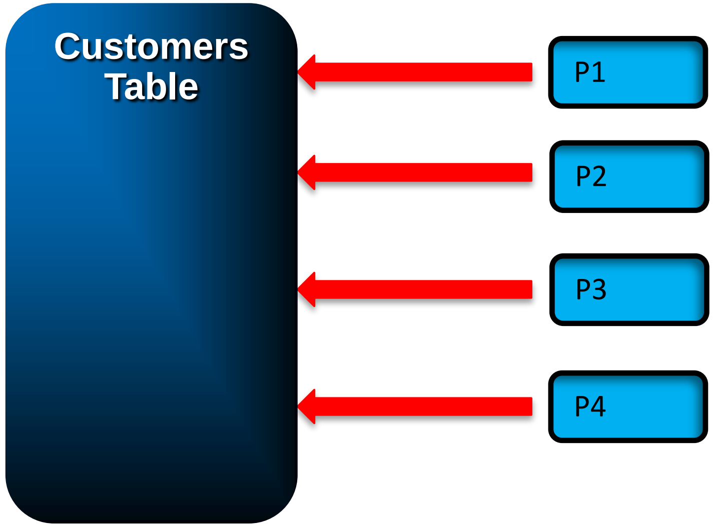
Inter Object using parallel execution

- Controlled by GATHER_*_STATS parameter DEGREE
- Default is to use parallel degree specified on object
- If set to AUTO Oracle decide parallel degree used
- Works on one object at a time

Statistics Gathering Performance

Inter Object using parallel execution

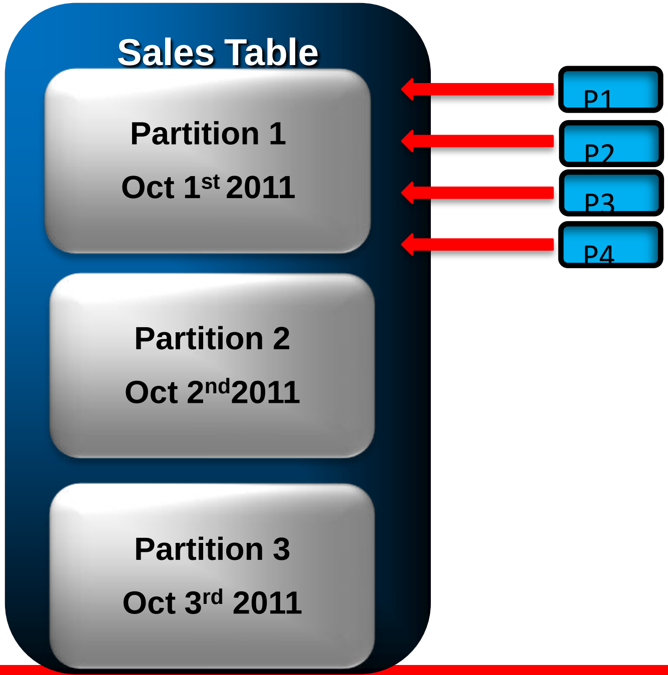
- Customers table has a degree of parallelism of 4
- 4 parallel server processes will be used to gather stats



Statistics Gathering Performance

Inter Object using parallel execution

- Exec DBMS_STATS.GATHER_TABLE_STATS(null, ‘SALES’);



Each individual partition will have statistics gathered one after the other

The statistics gather procedure on each individual partition operates in parallel BUT the statistics gathering procedures won’t happen concurrently

Statistics Gathering Performance

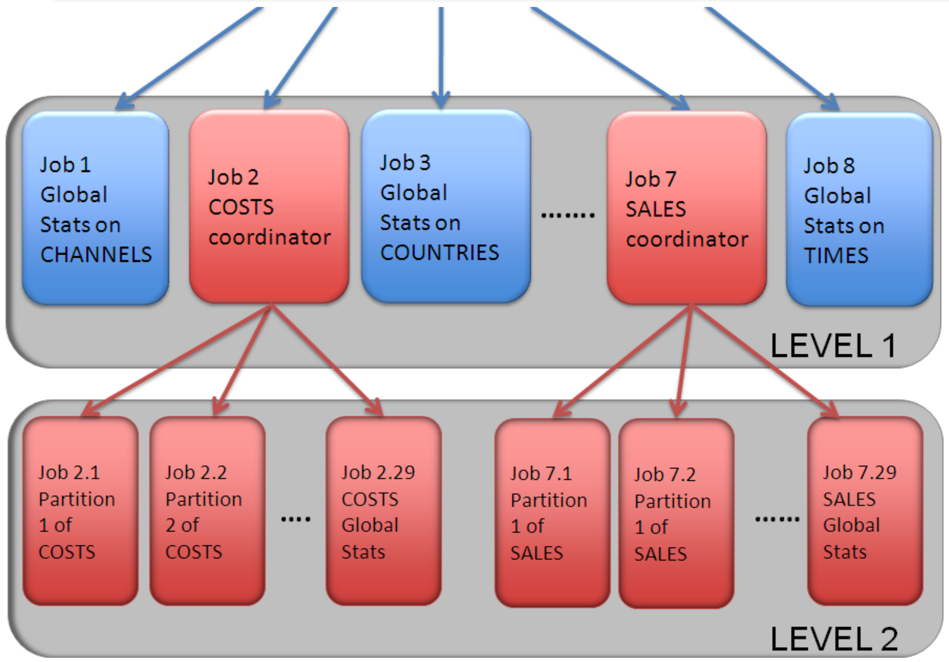
Intra Object

- Gather statistics on multiple objects at the same time
- Controlled by DBMS_STATS preference, CONCURRENT
- Uses Database Scheduler and Advanced Queuing
- Number of concurrent gather operations controlled by job_queue_processes parameter
- Each gather operation can still operate in parallel

Statistics Gathering Performance

Intra Object Statistics Gathering for SH Schema

```
Exec DBMS_STATS.GATHER_SCHEMA_STATS('SH');
```


Job 1 Global Stats on CHANNELS
Job 2 COSTS coordinator
Job 3 Global Stats on COUNTRIES
....
Job 7 SALES coordinator
Job 8 Global Stats on TIMES
LEVEL 1
Job 2.1 Partition 1 of COSTS
Job 2.2 Partition 2 of COSTS
....
Job 2.29 COSTS Global Stats
Job 7.1 Partition 1 of SALES
Job 7.2 Partition 1 of SALES
......
Job 7.29 SALES Global Stats
LEVEL 2

A separate statistics gathering job is created for each table and each partition in the schema

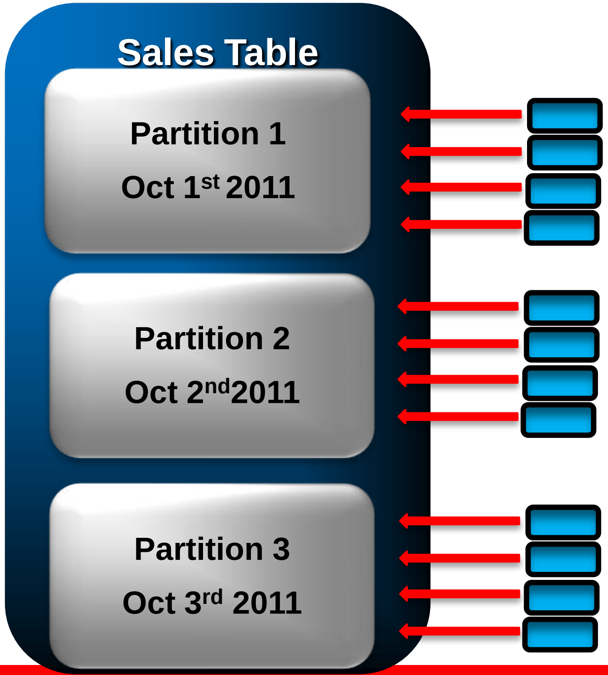
Level 1 contain statistics gathering jobs for all non-partitioned tables and a coordinating job for each partitioned table

Level 2 contain statistics gathering jobs for each partition in the partitioned tables

Statistics Gathering Performance

Intra and Inter working together for Partitioned Objects

```
Exec DBMS_STATS.GATHER_TABLE_STATS('SH','SALES');
```



The number of concurrent gathers is controlled by the parameter job_queue_processes

In this example it is set to 3

Remember each concurrent gather operates in parallel

In this example the parallel degree is 4

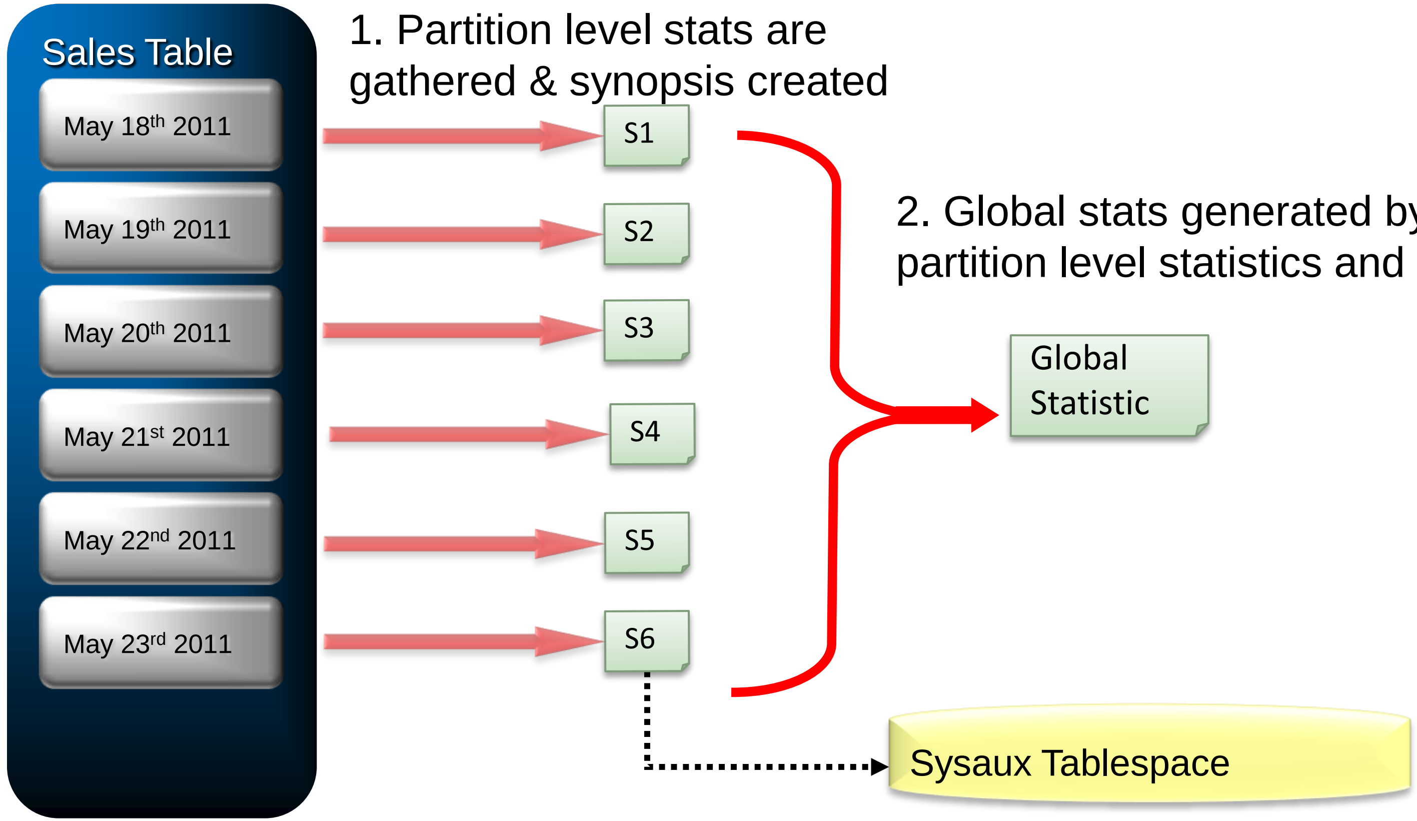
Statistics Gathering Performance

Incremental Statistics Gathering for Partitioned tables

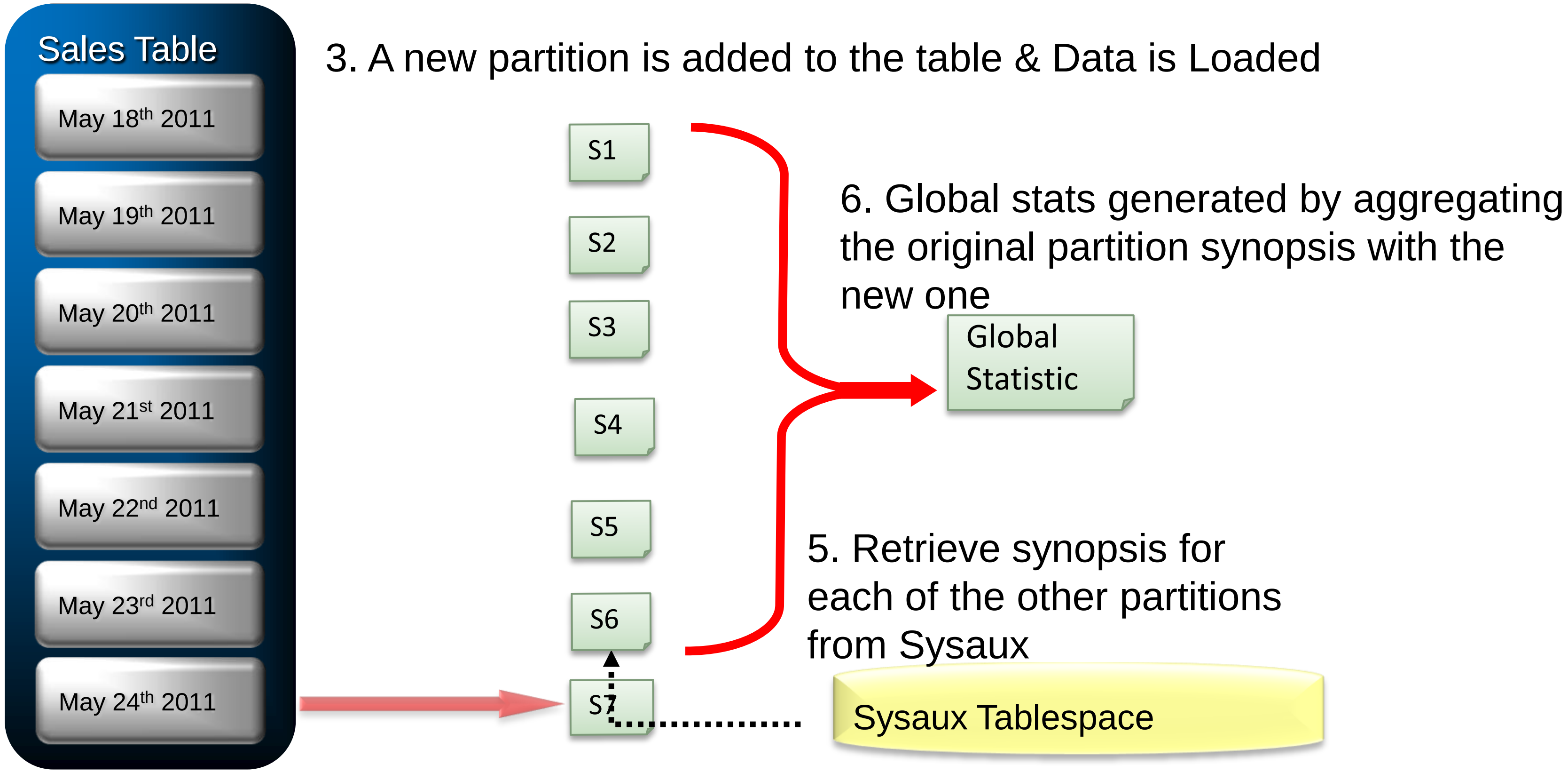
- Typically gathering statistics after a bulk loading data into one partition would causes a full scan of all partitions to gather global table statistics
  - Extremely time consuming

- With Incremental Statistic gather statistics for touched partition(s) ONLY
  - Table (global) statistics are accurately built from partition statistics
  - Reduce statistics gathering time considerably
  - Controlled by INCREMENTAL preference

Incremental Statistics Gathering

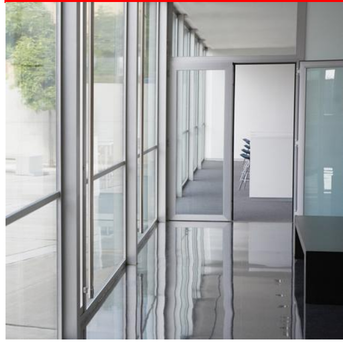

Sales Table
May 18th 2011
May 19th 2011
May 20th 2011
May 21st 2011
May 22nd 2011
May 23rd 2011
1. Partition level stats are gathered & synopsis created
S1
S2
S3
S4
S5
S6
2. Global stats generated by aggregating partition level statistics and synopsis
Global Statistic
Sysaux Tablespace

Incremental Statistics Gathering



Sales Table
May 18th 2011
May 19th 2011
May 20th 2011
May 21st 2011
May 22nd 2011
May 23rd 2011
May 24th 2011
3. A new partition is added to the table & Data is Loaded
S1
S2
S3
S4
S5
S6
S7
6. Global stats generated by aggregating the original partition synopsis with the new one
Global Statistic
5. Retrieve synopsis for each of the other partitions from Sysaux
Sysaux Tablespace

Agenda



- How to gather statistics
- What basic statistics to collect
- Additional types of statistics
- When to gather statistics
- Statistics gathering performance
- When not to gather statistics

When Not to Gather Statistics

- Volatile Tables
  - Volume of data changes dramatically over a period of time
  - For example orders queue table
    - Starts empty, orders come in, order get processed, ends day empty
- Global Temp Tables
  - Application code stores intermediate result
  - Some session have a lot of data, some have very little
- Intermediate work tables
  - Written once, read once and then truncated or deleted
  - For example part of an ETL process

When Not to Gather Statistics

Volatile Tables

- Data volume changes dramatically over time
- When is a good time to gather statistics?

- Gather statistics when the table has a representative data volume
- Lock statistics to ensure statistics gathering job does not over write representative statistics

When Not to Gather Statistics

Intermediate Work Tables

- Often seen as part of an ETL process
- Written once, read once, and then truncated or deleted
- When do you gather statistics?

- Don’t gather statistics it will only increase ETL time
- Use Dynamic sampling
  - Add dynamic sampling hint to SQL statements querying the intermediate table

When Not to Gather Statistics

Intermediate Work Tables

- Add dynamic sampling hint or et it at the session or system level

```
SELECT /*+ dynamic_sampling(cst 2) */ *
FROM   customers_staging_tab cst
WHERE  cust_address_change ='Y';
```

Oracle Optimizer Schedule for Oracle Open World

Date	Title	Location	Speaker
Monday Oct 3rd 12:30 PM	Oracle Optimizer: Prevent Suboptimal Execution Plans **Hands-on-Lab**	Marriott Marquis - Salon 12/13	Maria Colgan Senior Principal Member of Technical Staff Oracle
Wednesday Oct 5th 10:15 AM	Oracle Optimizer: Best Practices for Managing Optimizer Statistics	Moscone South - 103	Maria Colgan Senior Principal Member of Technical Staff Oracle
Thursday Oct 6th 12:00 PM	Oracle Database Optimizer: Tips for Preventing Suboptimal Execution Plans	Moscone South - 104	Maria Colgan Senior Principal Member of Technical Staff Oracle Mohamed Zait Architect Oracle

Q&A

ORACLE®