



ORACLE®

**Oracle Database 10g and 11g:
What to Expect From the Optimizer**

Maria Colgan Principal Product Manager



Agenda

- Changes in behaviour
 - Init.ora Parameters
 - DBMS_STATS subprograms
 - New auto stats job
- New features
 - Adaptive Cursor Sharing
 - Optimizer statistics enhancements
 - SQL Plan Management
- Pre-upgrade checklist
- Post-upgrade checklist
- Correcting regressed SQL Statements
 - SQL Testcase Builder
 - SQL Repair Advisor

Change in Behavior



Init.ora Parameters

Parameter	9i Value	10g Value	11g Value
Optimizer_mode	Choose	All_rows	All_rows
Dynamic_Sampling	1	2	2
Secure_view_merging	N/A	True	True
Optimizer_use_invisible_indexes	N/A	N/A	False
Optimizer_use_pending_statistics	N/A	N/A	False
Optimizer_capture_SQL_plan_baselines	N/A	N/A	False
Optimizer_use_SQL_plan_baselines	N/A	N/A	True

New DBMS_STATS Subprograms

Subprogram	Function	In 10gR2	In 11g
Gather_System_Stats	Gathers stats on CPU and IO speed of H/W	Yes	Yes
Gather_Dictionary_Stats	Gathers stats on dictionary objects	Yes	Yes
Gather_Fixed_Object_Stats	Gather stats on V\$views	Yes	Yes
Publish_Pending_stats	Pending stats allows stats to be gather but not published immediate	N/A	Yes
Restore_Table_Stats	Revert stats back to what they were before most recent gather	10.2.0.4	Yes
Compare_Table_Stats	Compare stats for a table from two different sources	10.2.0.4	Yes
Create_Extended_stats	Gathers stats for a user specified column group or an expression	N/A	Yes
Set_*_Prefs	Sets stats preferences at a table, schema, database or global level	N/A	Yes

DBMS_STAT.SET*_PREFS

Offers a finer granularity of control with 4 procedures

- DBMS_STAT.SET_TABLE_PREFS
 - Changes parameter value for the specified table
- DBMS_STAT.SET_SCHEMA_PREFS
 - Changes parameter value for all tables in the specified schema
 - Calls DBMS_STAT.SET_TABLE_PREFS for each table
- DBMS_STAT.SET_DATABASE_PREFS
 - Changes parameter value for all tables in user-defined schemas
 - Calls DBMS_STAT.SET_TABLE_PREFS for each table
- DBMS_STAT.SET_GLOBAL_PREFS
 - Changes parameter value for all tables without a table preference and all future tables

Hierarchy: Parameter value in gather stats stmt if specified

↑ table preference if specified

↑ global preference



Automatic statistics gathering job

- Introduced in 10g
- Gathers statistics on objects where
 - Statistics are missing
 - Statistics are Stale (10% of rows have changed `USER_TAB_MODIFICATIONS`)
- In 10g its an Oracle Scheduler job
 - Runs during maintenance window (default 10pm – 6 am)
- In 11g its an Autotask
 - Runs during maintenance window (default 10pm – 6 am)
- `DBMS_STATS.GATHER_DATABASE_STATS_JOB_PROC`
 - Parameter values can be change using `SET_GLOBAL_PERFS`

New Features in 11g





New 11g Optimizer features

- Bind peeking doesn't work when there is a data skew
- Gathering Optimizer Statistics takes too long
- Cardinality estimate is wrong so plan goes wrong
- Plans change unexpectedly especially during upgrades



New 11g Optimizer features

- Bind peeking doesn't work when there is a data skew
 - Enhanced plan sharing with binds
- Gathering Optimizer Statistics takes too long
 - Faster statistics gathering
 - Improved statistics quality
- Cardinality estimate is wrong so plan goes wrong
 - Collect appropriate statistics
 - Eliminate wrong cardinality estimates
- Plans change unexpectedly especially during upgrades
 - Guaranteed plan stability and controlled plan evolution
 - Controlled statistics publication

Adaptive Cursor Sharing

Enhanced Bind Peeking





Adaptive Cursor Sharing

Business Requirement

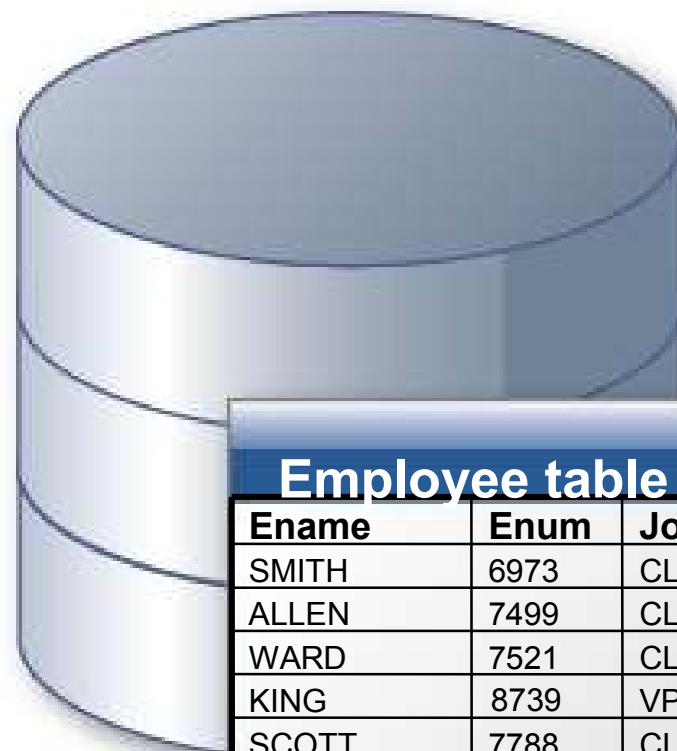
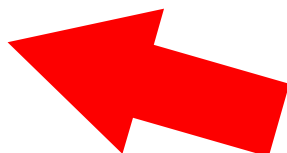
- The optimizer peeks bind values during plan selection
- Initial value of the binds determines the plan
- Same execution plan shared regardless of future bind values
- During the business day a cursor gets aged out at the next hard parse a different bind value is used and a different plan gets generated

One plan not always appropriate for all bind values

Example with 10g

SELECTFROM..WHERE Job = :B1
Value of B1 = CLERK

Ename	Enum	Job
SMITH	6973	CLERK
ALLEN	7449	CLERK
WARD	7521	CLERK
SCOTT	7788	CLERK
CLARK	7782	CLERK



Ename	Enum	Job
SMITH	6973	CLERK
ALLEN	7499	CLERK
WARD	7521	CLERK
KING	8739	VP
SCOTT	7788	CLERK
CLARK	7782	CLERK

- If clerk is the bind value at hard parse five out six records will be selected

Id	Operation	Name	Starts	E-Rows	A-Rows
* 1	TABLE ACCESS FULL	EMP	1	5	5

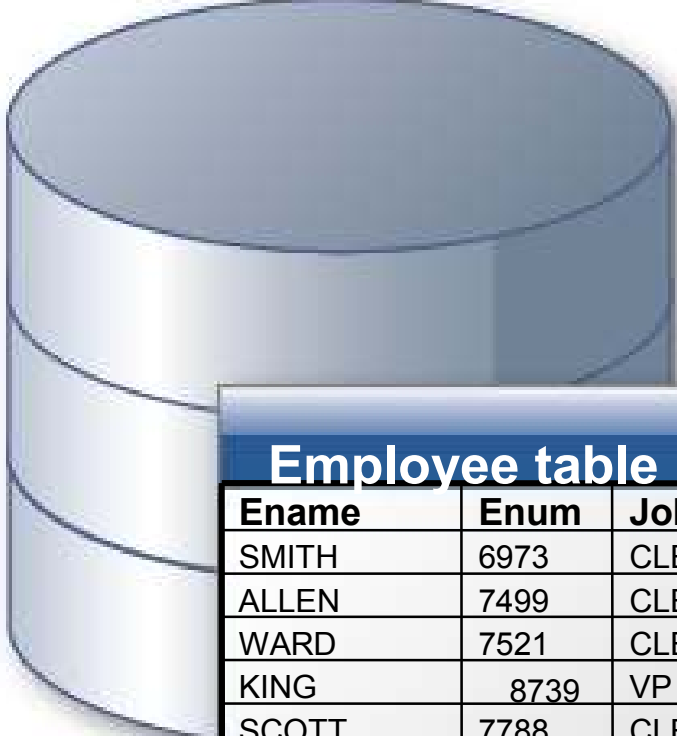
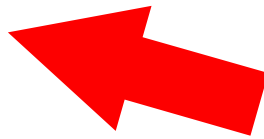
Note Plan Output from dbms_xplan.display_cursor

Example with 10g cont.

```
SELECT .....FROM..WHERE Job = :B1
```

Value of B1 = VP

Ename	Enum	Job
KING	8739	VP



Ename	Enum	Job
SMITH	6973	CLERK
ALLEN	7499	CLERK
WARD	7521	CLERK
KING	8739	VP
SCOTT	7788	CLERK
CLARK	7782	CLERK

- If VP is the bind value at hard parse one out six records will be selected

```
-----  
| Id  | Operation          | Name           | Starts | E-Rows | A-Rows |  
-----  
|* 1  | INDEX RANGE SCAN | IND_EMP_JOB    |        | 1      | 1      |  
-----
```



Solutions for bind peeking and histograms in 10g

- Drop histogram using `DBMS_STATS.DELETE_COL_STATS` for just the effected table
- Regather statistics on this table without histogram
- Use `DBMS_STATS.SET_PARM` to change default setting for `method_opt` parameter
 - New default in 10g `FOR_ALL_ROWS_SIZE_AUTO`
 - Oracle automatically gathers histograms based on column usage
- Switch off bind peeking `set _optim_peek_user_binds = false`

With 11g

```
SELECT .....FROM. .WHERE Job = :B1
```

YOU CAN HAVE BOTH PLANS

B1 = CLERK

Ename	Enum	Job
SMITH	6973	CLERK
ALLEN	7449	CLERK
WARD	7521	CLERK
SCOTT	7788	CLERK
CLARK	7782	CLERK

Full Table Scan is optimal

B1 = VP

Ename	Enum	Job
KING	6973	VP

Index Access is optimal

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



Adaptive Cursor Sharing

Solution

- 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



Adaptive Cursor Sharing – in detail

- Controlled by init.ora parameter
 - *_optim_peek_user_binds*
 - Determines if the optimizer will peek at bind values
 - Set to TRUE by default in 11gR1
- Monitor
 - V\$SQL has 2 new columns
 - IS_BIND_SENSITIVE – A histogram is present on column used with Bind
 - IS_BIND_AWARE – An alternative plan has been found for SQL STMT



Optimizer Statistics

Improved Efficiency and
Quality

Q

Improved Efficiency and Quality

New statistics gathering algorithm

Business problem

- “ .. Compute statistics gives accurate results but takes too long ..”
- “ .. Sampling is fast but not always accurate ..”
- “ .. AUTO SAMPLE SIZE does not always work with data skew ..”

Solution

- New groundbreaking implementation
 - Faster than sampling
 - Accuracy comparable to compute statistics
- Used by default with AUTO_SAMPLE_SIZE value
- No need to use manual sampling anymore

FASTER AND BETTER

Speed of sampling with the accuracy of compute



Improved Efficiency and Quality

Incremental Statistics Maintenance

Business Requirement

- Gathering statistics on one partition (e.g. after a bulk load) causes a full scan of all partitions to gather global table statistics
- Extremely time consuming

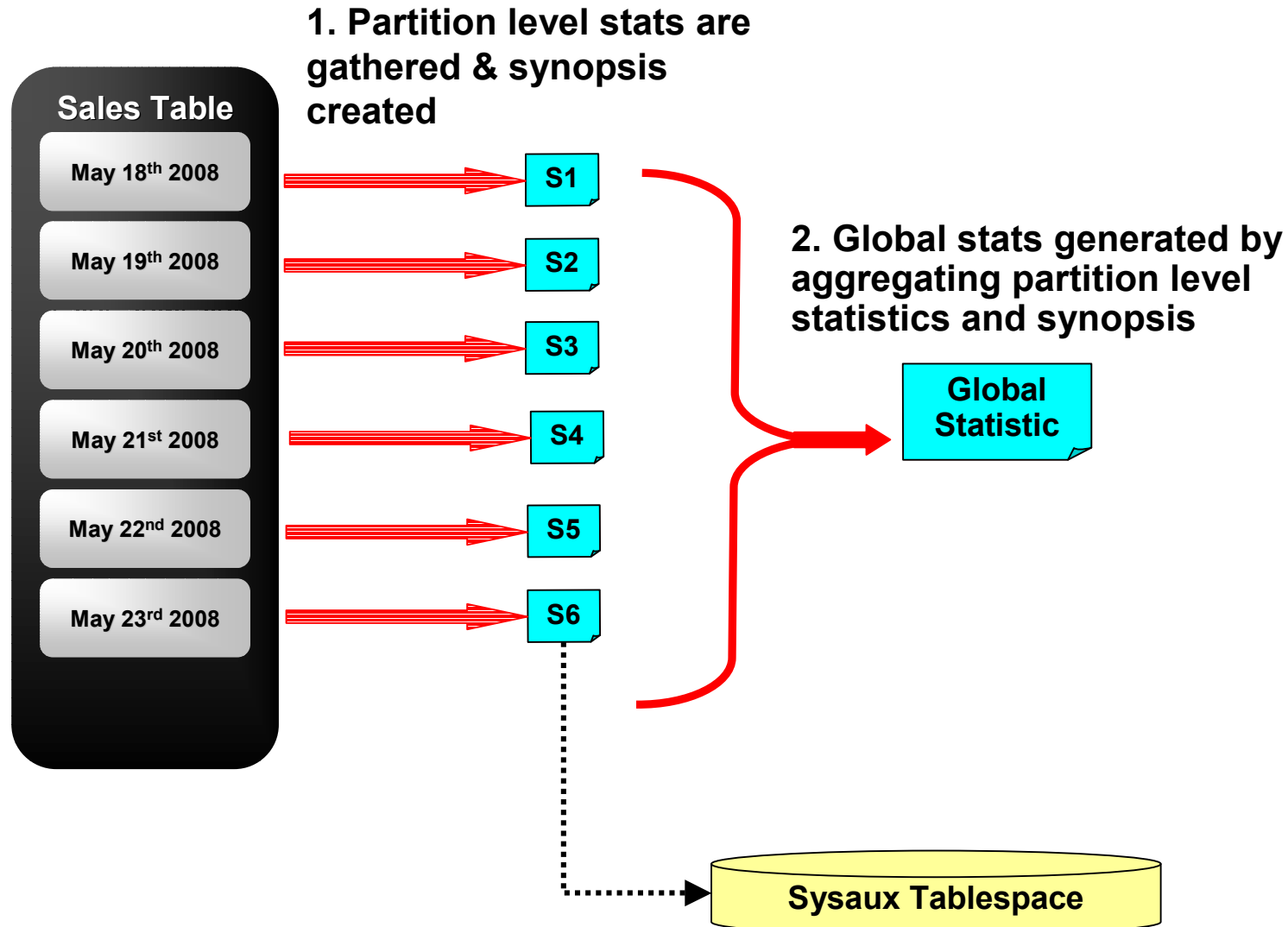
Solution

- Gather statistics for touched partition(s) ONLY
- Table (global) statistics are built from partition statistics

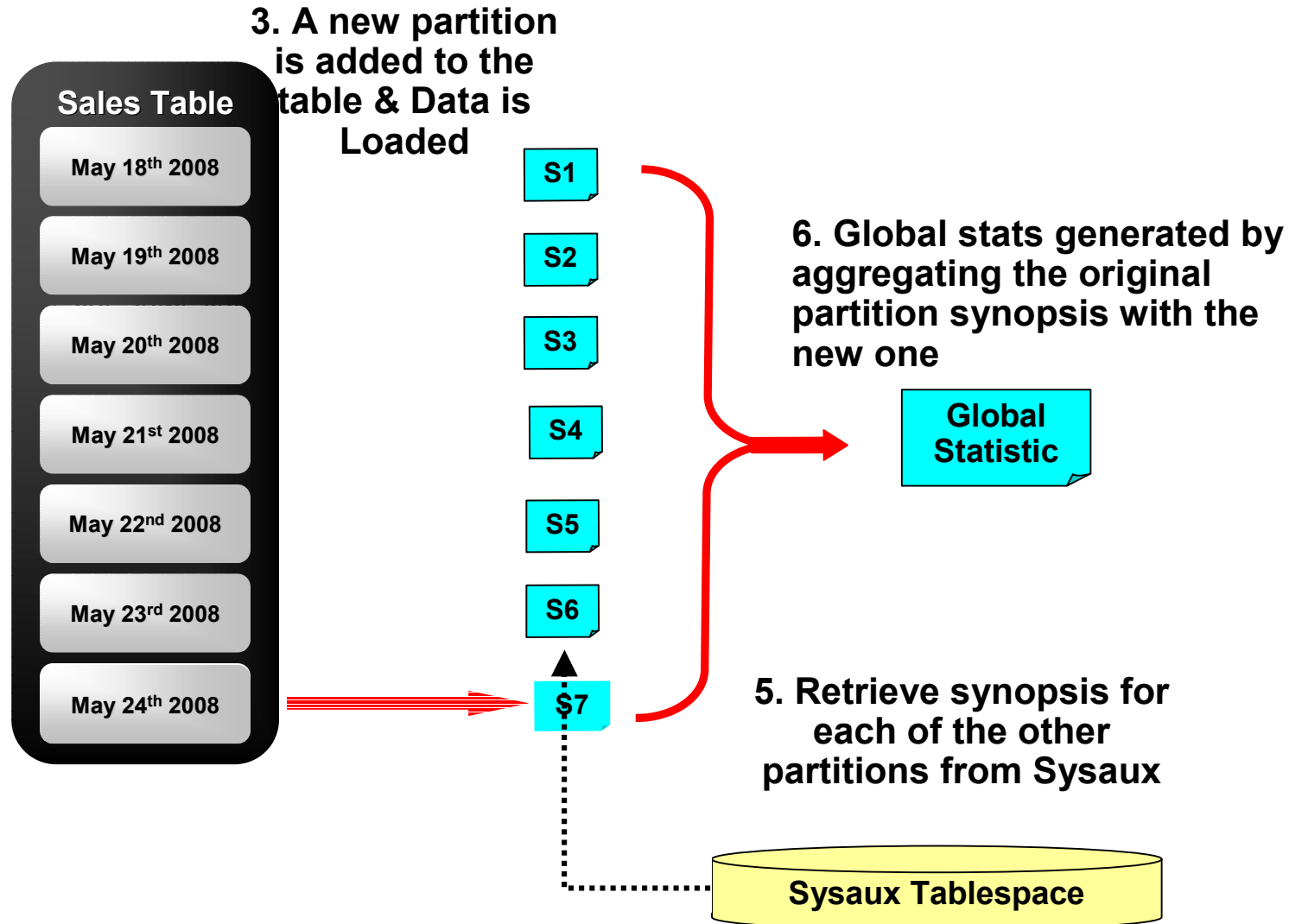
Refreshed WITHOUT scanning the NON touched partitions



Incremental Global Statistics



Incremental Global Statistics Cont'd





Copy Statistics

Business Requirement

- New partition is added to a table and data is loaded into it - Statistics for this partition do not reflect actual data volume or values
- Optimizer prorates cardinality based on distance between predicate value and current max value for column RESULT very low cardinality

Solution

- Use `dbms_stats.copy_table_stats()`
- Derives statistics for new partition:
 - Column statistics (min,max, NDV, histogram, etc)
 - It adjusts min/max for partitioning column but not histogram
 - Partition statistics (number of rows, blocks, etc)
 - Local index statistics (NOT global)
- Requires patch on top of 10.2.0.4 - bug 7687788



Setting Optimizer Statistics

Business Requirement

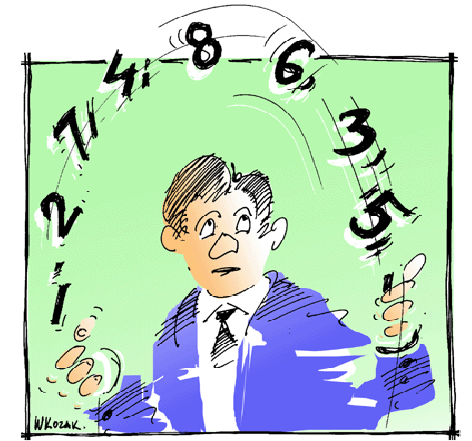
- Temporary table used in transaction logic – Original empty but gets a lot of data added during the course of the transaction

Solution

- Gather statistics when table is full (before end of trans) & lock them
- OR
- Use `dbms_stats.set_table_stats`
 - Requires you to know what best possible stats are

Extended Optimizer Statistics

Eliminate wrong cardinality
estimates





Extended Optimizer Statistics

Business problem - Correlated Columns

- Real data often shows correlations between various attributes
 - e.g. job title influences salary, car model influences make, seasons affect the amount of sold goods (e.g. snow shoes in winter)
- Optimizer has to estimate the correct cardinality
 - *“Does an additional filter reduce the result set or not?”*

Solution

- Extended Optimizer Statistics provides a mechanism to collect statistics on a group of columns
- Full integration into existing statistics framework
 - Automatically maintained with column statistics
 - Instantaneous and transparent benefit for any migrated application

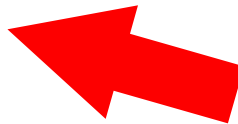
Improved Cardinality leads to Improved Plans

Extended Statistic Example

single column

```
SELECT .....FROM..  
WHERE model = '530xi'
```

Make	Model	Color
BMW	530xi	RED
BMW	530xi	BLACK
BMW	530xi	SILVER



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

- Three records selected
 - Single column statistics are accurate

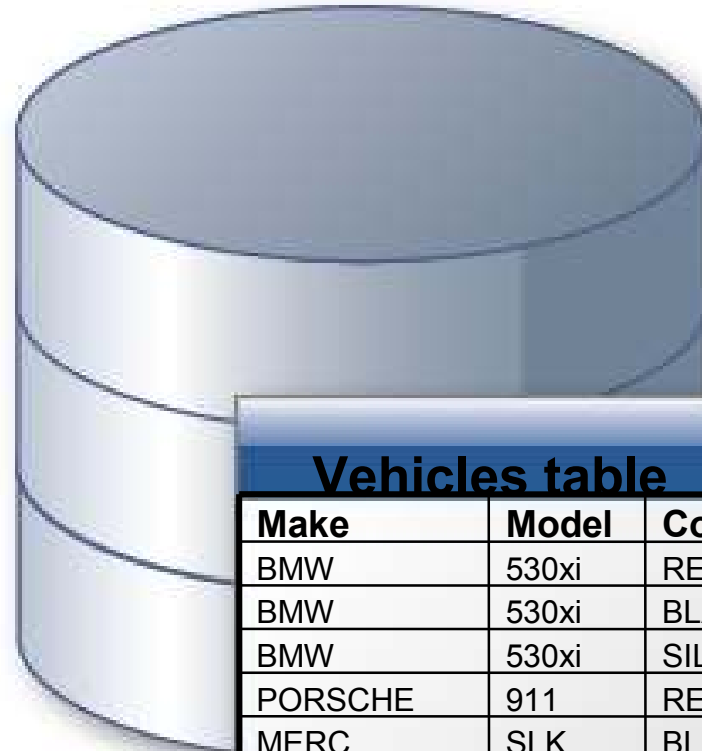
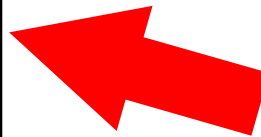
```
-----  
| Id | Operation | Name | Starts | E-Rows | A-Rows  
-----  
|* 1 | TABLE ACCESS FULL | CARS | 1 | 3 | 3 |  
-----
```

Example

non-correlated columns

```
SELECT .....FROM..
WHERE model = '530xi'
AND color = 'RED'
```

Make	Model	Color
BMW	530xi	RED



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

- One record selected
 - No correlated columns
 - Additional predicate reduces result set
 - Single column statistics are sufficient

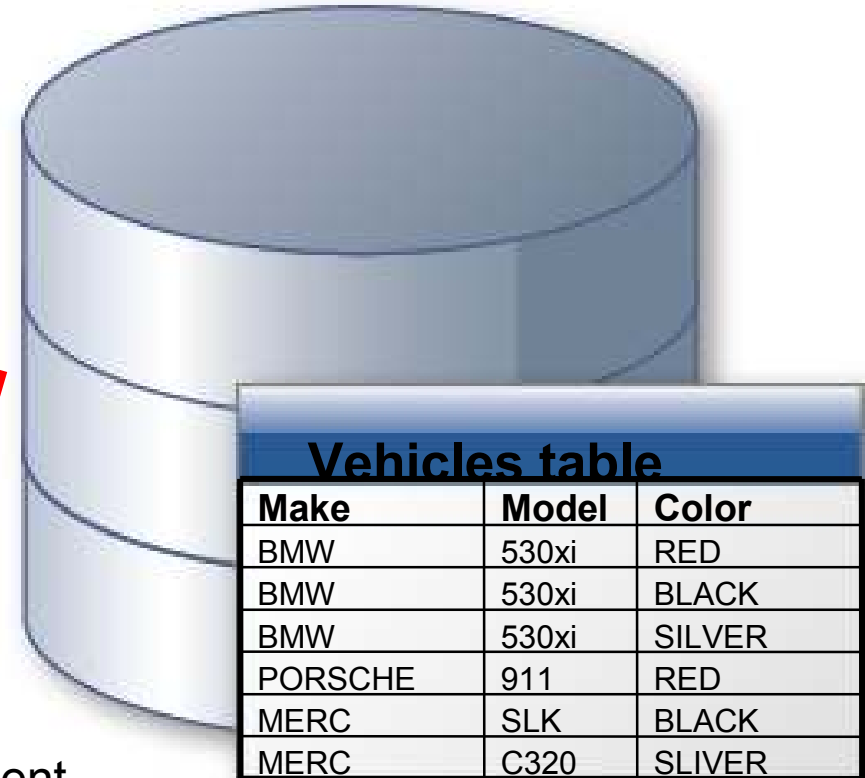
```
-----
| Id | Operation          | Name      | Starts | E-Rows | A-Rows
-----
|*  1 | INDEX RANGE SCAN | CAR_MC   |        1 |        1 |        1
-----
```

Example

correlated columns, no extended statistics

```
SELECT .....FROM..  
WHERE model = '530xi'  
AND make = 'BMW';
```

Make	Model	Color
BMW	530xi	RED
BMW	530xi	BLACK
BMW	530xi	SILVER



- Three records selected
 - Correlated columns
 - Additional predicate has no effect
 - Single column statistics are **NOT** sufficient

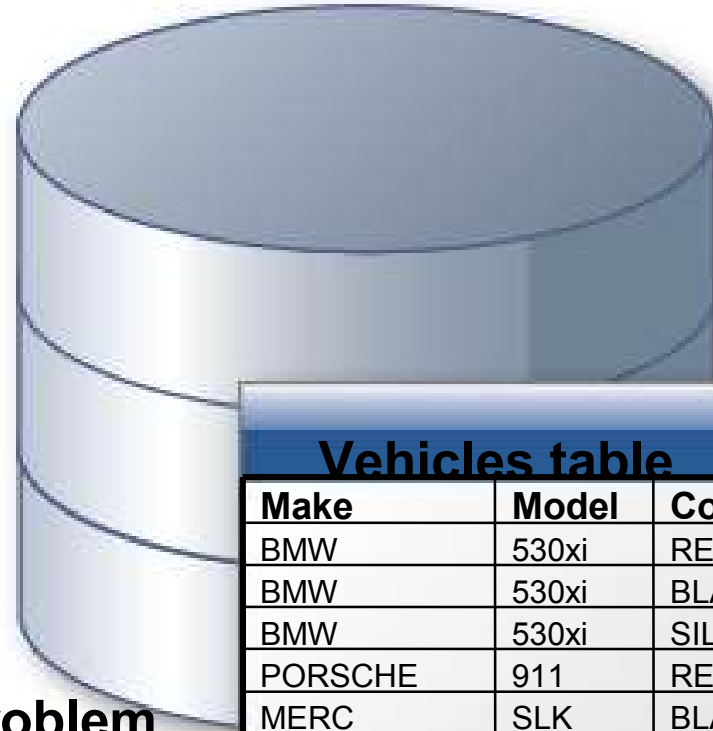
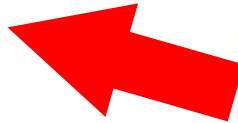
Id	Operation	Name	Starts	E-Rows	A-Rows
* 1	INDEX RANGE SCAN	CAR_MC	1	1	3

Example

correlated columns, extended statistics

```
SELECT .....FROM..
WHERE model = '530xi'
AND make = 'BMW' ;
```

Make	Model	Color
BMW	530xi	RED
BMW	530xi	BLACK
BMW	530xi	SILVER



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

- Three records selected.
 - Multi-column statistics **solve the problem**

Id	Operation	Name	Starts	E-Rows	A-Rows
* 1	TABLE ACCESS FULL	CARS	1	3	3



Extended Statistics – in detail

- Use `dbms_stats` package
 - `Create_extended_stats`
 - Manually specify the group of columns
 - `Show_extended_stats_name`
 - Displays the system generated name for the column group
 - `Drop_extended_stats`
 - Drop a column group and all the statistics associated with it
- Monitor
 - New dictionary table `user_stat_extensions`
 - Shows sys generated name & actual column group desc
 - Look at dictionary table `user_tab_col_statistics`
 - New row with sys generated name will be add for each column group



Pending Statistics

Controlled statistics
publication





Pending Statistics

Business Requirement

- Statistics are published as soon as we complete gathering
=> Possibly unpredictable changes of execution plans
- Today you have 'freeze' critical plans or statistics

Solution

- Gather statistics and save as pending
- Verify the new statistics don't change plans adversely
 - Either on the same or a different system
- Publish verified statistics

Controlled and DBA-verified statistics management



Pending Statistics – in detail

- Controlled by init.ora parameter
 - ***optimizer_use_pending_statistics***
 - Determines if the optimizer will use pending statistics
 - Set to false by default in 11gR1
- Use dbms_stats package
 - set_table_prefs
 - All tables preferences have “publish” set to true by default
 - publish_private_stats
 - Once stats have been tested publish them for general use
- Monitor
 - Look at dictionary table user_*_pending_stats (* = tab, col, ind)

SQL Plan Management

Guaranteed plan stability and
controlled plan evolution





SQL Plan Management

Business Requirement

- Unpredictable changes in execution plans can happen
 - New Statistics
 - Changes in the Environment
 - Software upgrades
- Today you have to 'freeze' critical plans or statistics

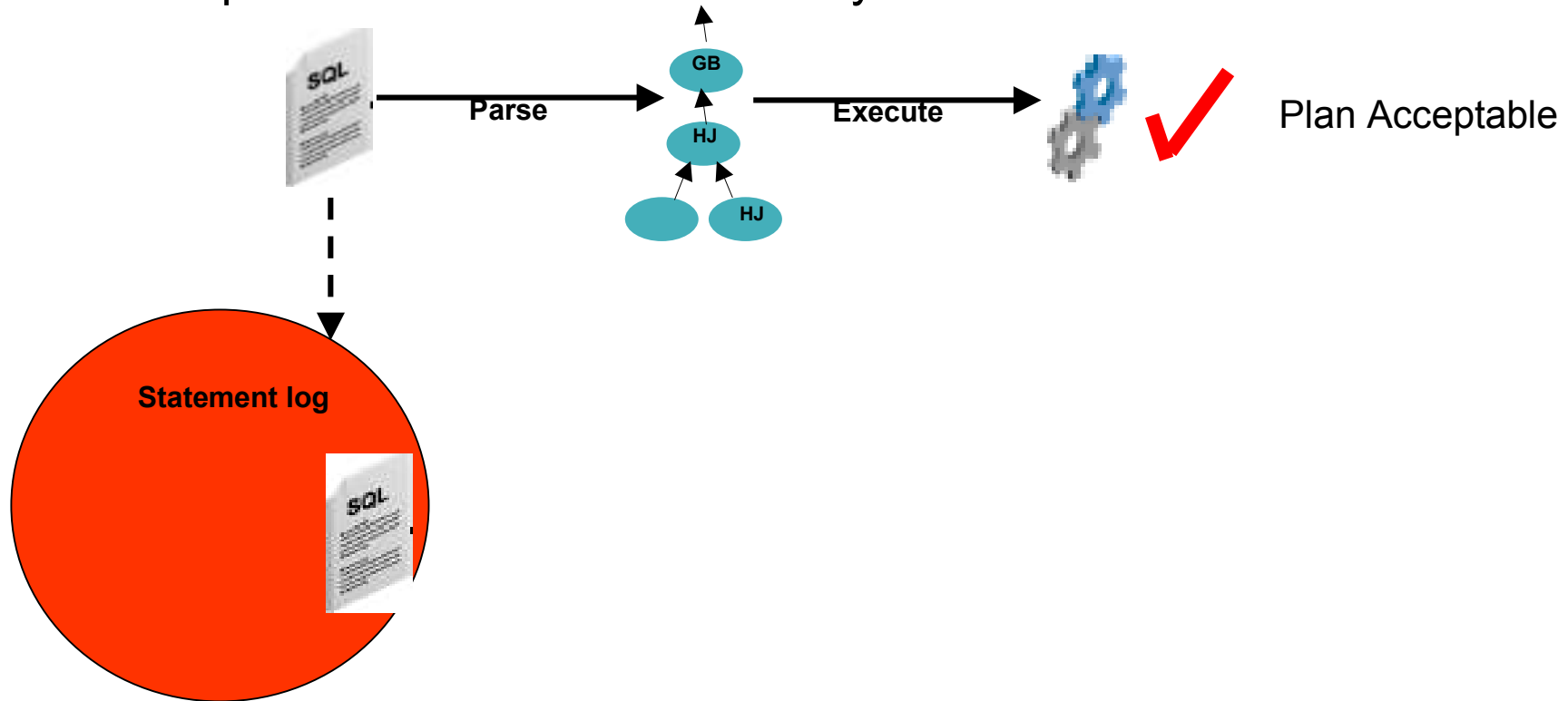
Solution

- Optimizer automatically manages 'execution plans'
 - Only known and **verified** plans are used
- Plan changes are automatically verified
 - Only comparable or better plans are used going forward

SQL Plan Management is controlled plan evolution

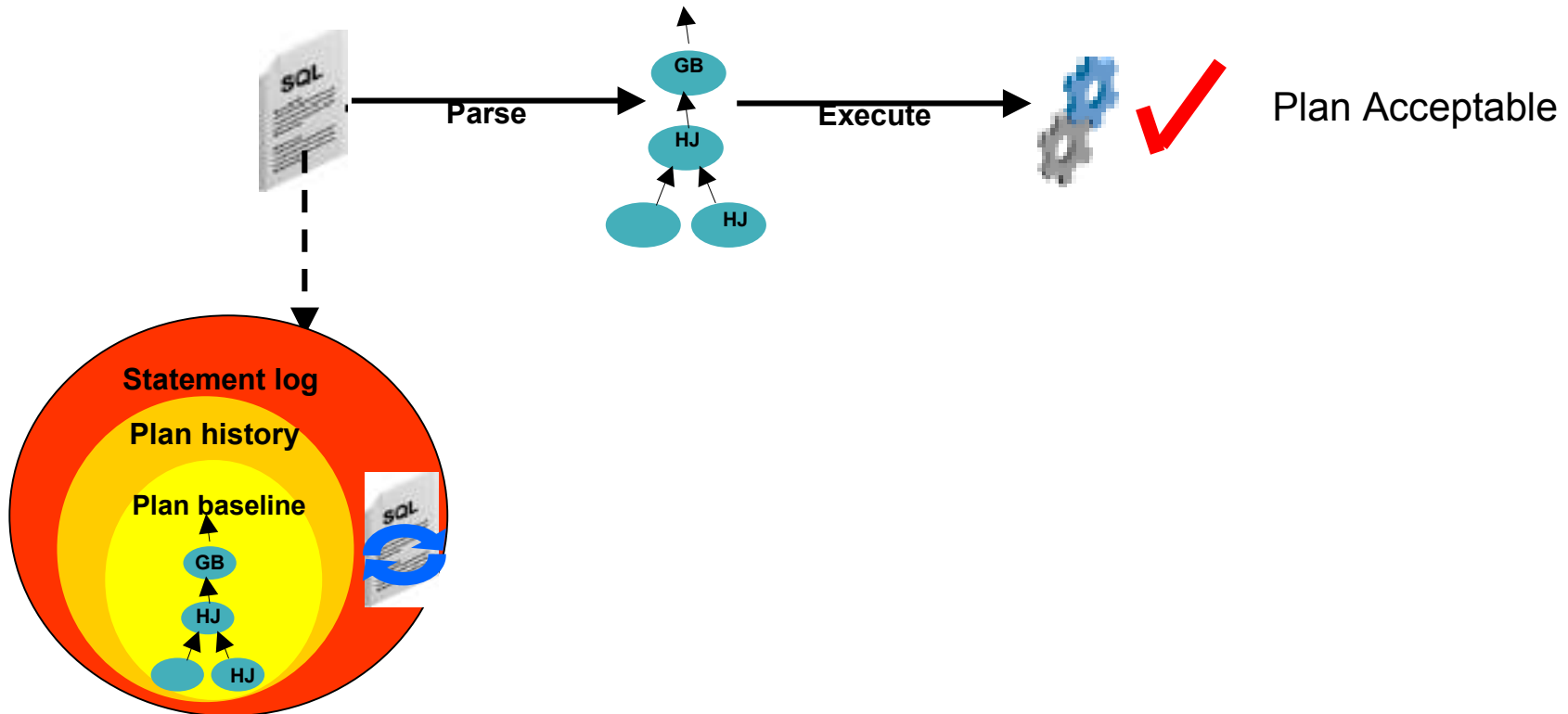
With SQL Plan Management

- SQL statement is parsed for the first time and a plan is generated
- Check the log to see if this is a repeatable SQL statement
- Add SQL statement signature to the log and execute it
- Plan performance is still “verified by execution”



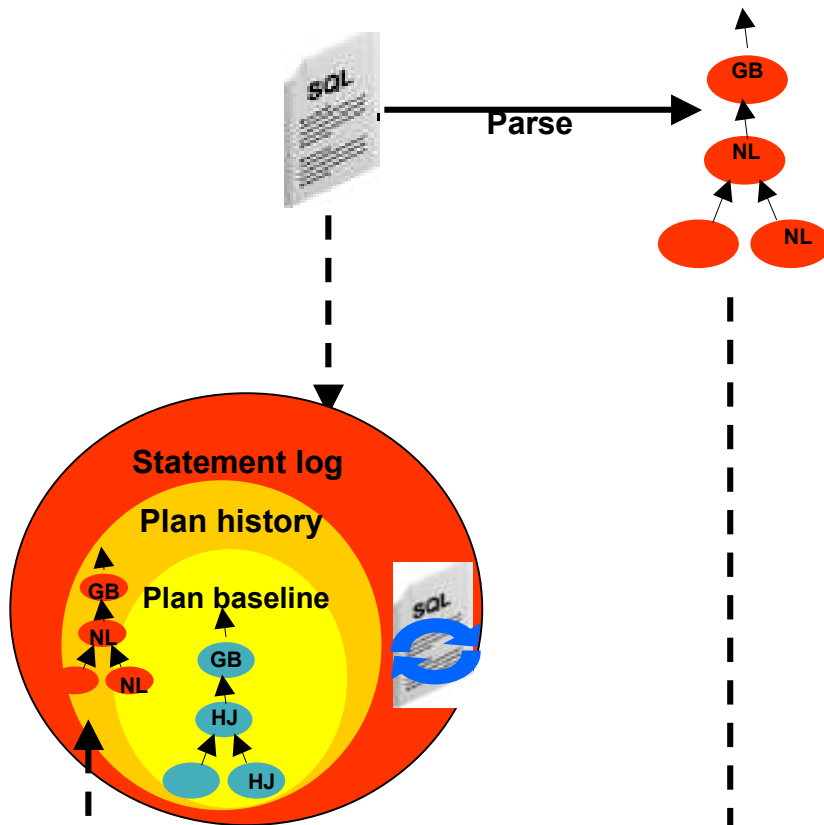
With SQL Plan Management

- SQL statement is parsed again and a plan is generated
- Check log to see if this is a repeatable SQL statement
- Create a Plan history and use current plan as SQL plan baseline
- Plan performance is “verified by execution”



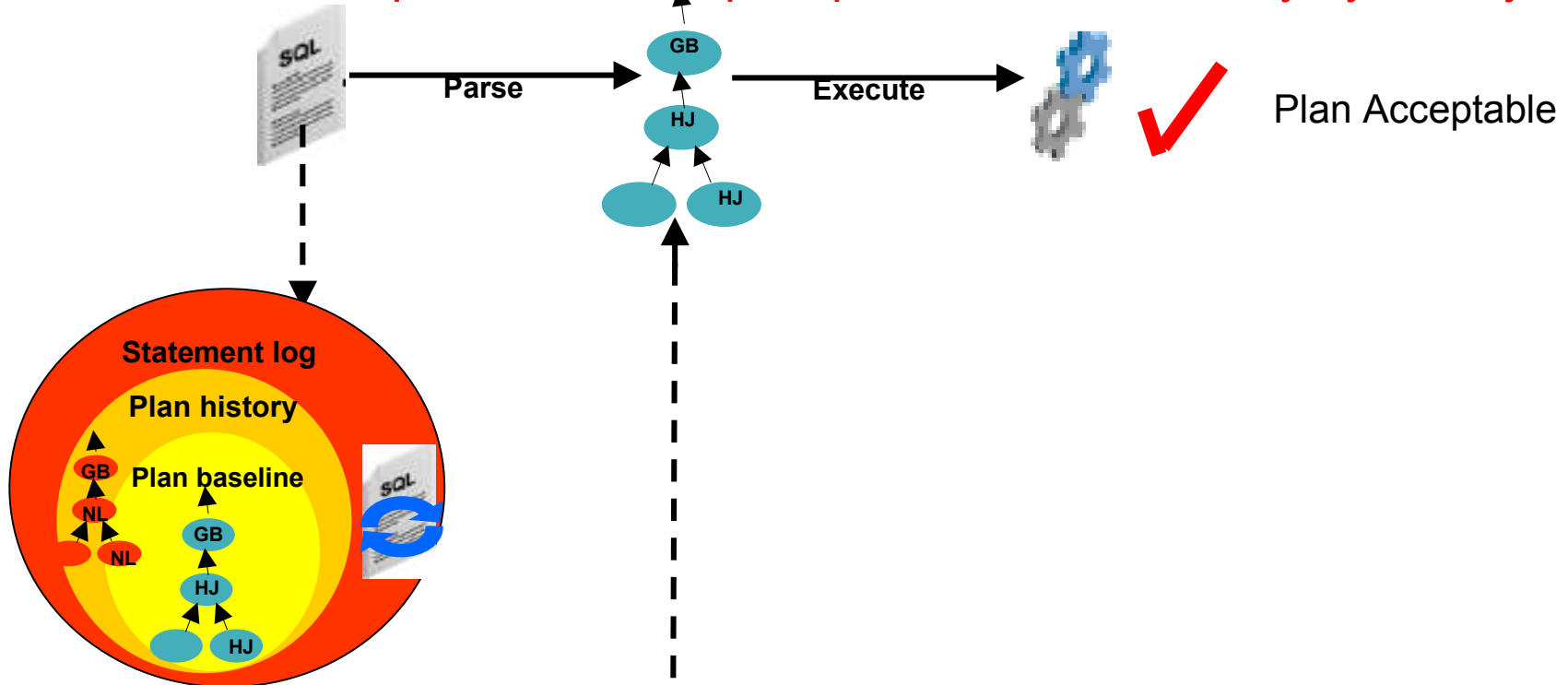
With SQL Plan Management

- Something changes in the environment
- SQL statement is parsed again and a **new plan is generated**
- **New plan is not the same as the baseline – new plan is not executed but marked for verification**



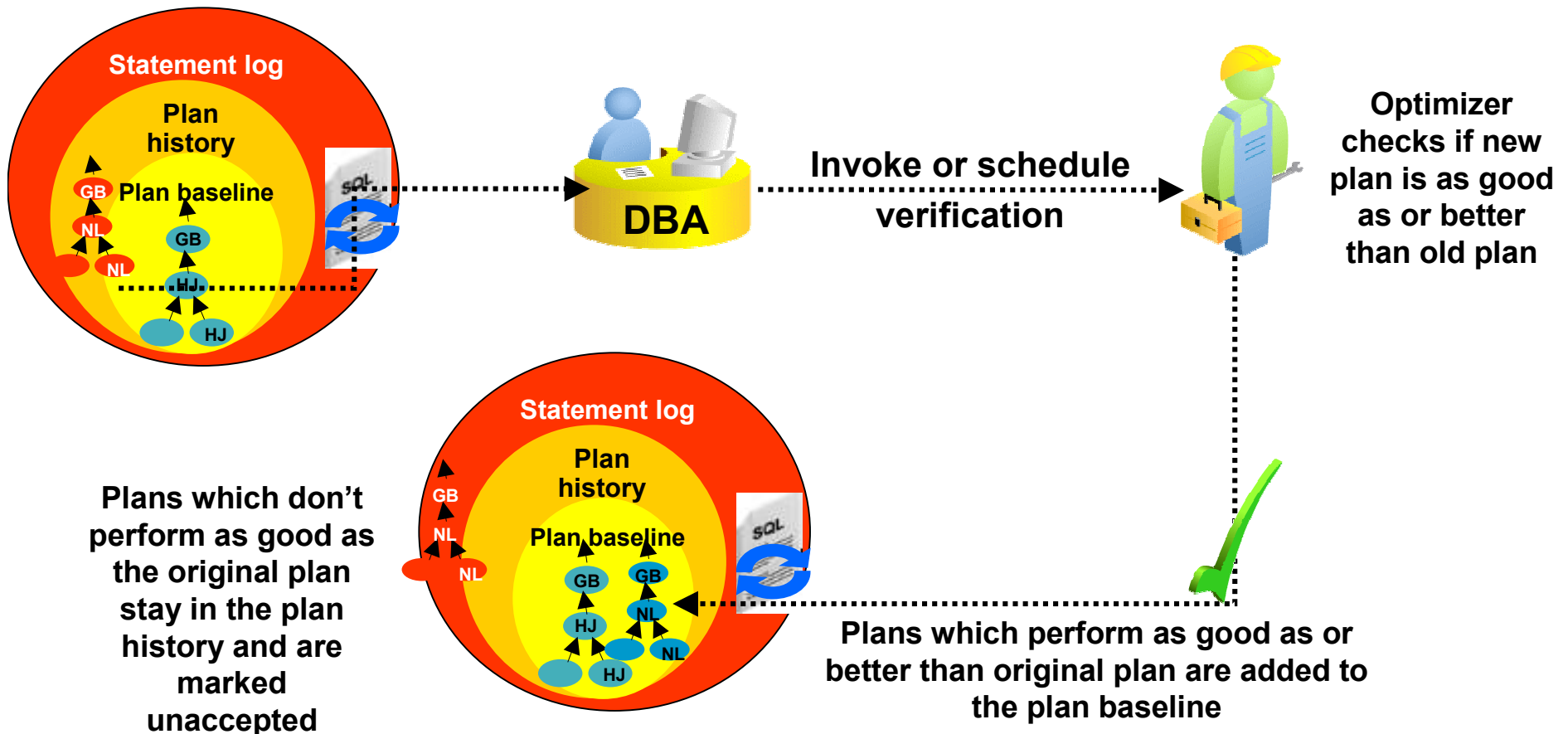
With SQL Plan Management

- Something changes in the environment
- SQL statement is parsed again and a **new plan is generated**
- New plan is not the same as the baseline – **new plan is not executed** but marked for verification
- **Execute known plan baseline - plan performance is “verify by history”**



Verifying the new plan

- Non-baseline plans will not be used until verified
- DBA can verify plan at any time





SQL Plan Management – the details

- Controlled by two init.ora parameter
 - ***optimizer_capture_sql_plan_baselines***
 - Controls auto-capture of SQL plan baselines for repeatable stmts
 - Set to false by default in 11gR1
 - ***optimizer_use_sql_plan_baselines***
 - Controls the use of existing SQL plan baselines by the optimizer
 - Set to true by default in 11gR1
- Monitoring SPM
 - Dictionary view DBA_SQL_PLAN_BASELINE
 - Via the SQL Plan Control in EM DBControl
- Managing SPM
 - PL/SQL package DBMS_SPM or via SQL Plan Control in EM DBControl
 - Requires the administer sql management object privilege



SPM Plan Capture – Bulk

- From SQL Tuning Set (STS)
 - Captures plan details for a (critical) set of SQL Statement
 - Load these plans into SPM as baseline plans
 - Next time statements are executed baseline plans will be used
- From Cursor Cache
 - Load plans from the cursor cache into SPM as baseline plans
 - Filters can be specified (SQL_ID, Module name, schema)
 - Next time statements are executed baseline plans will be used
- From staging table
 - SQL plan baselines can be captured on another system
 - Exported via a table (similar to statistics) and imported locally
 - Plan are “unpacked” from the table and loaded into SPM

Pre-Upgrade Checklist

What to do before the upgrade





Testing on the new database release

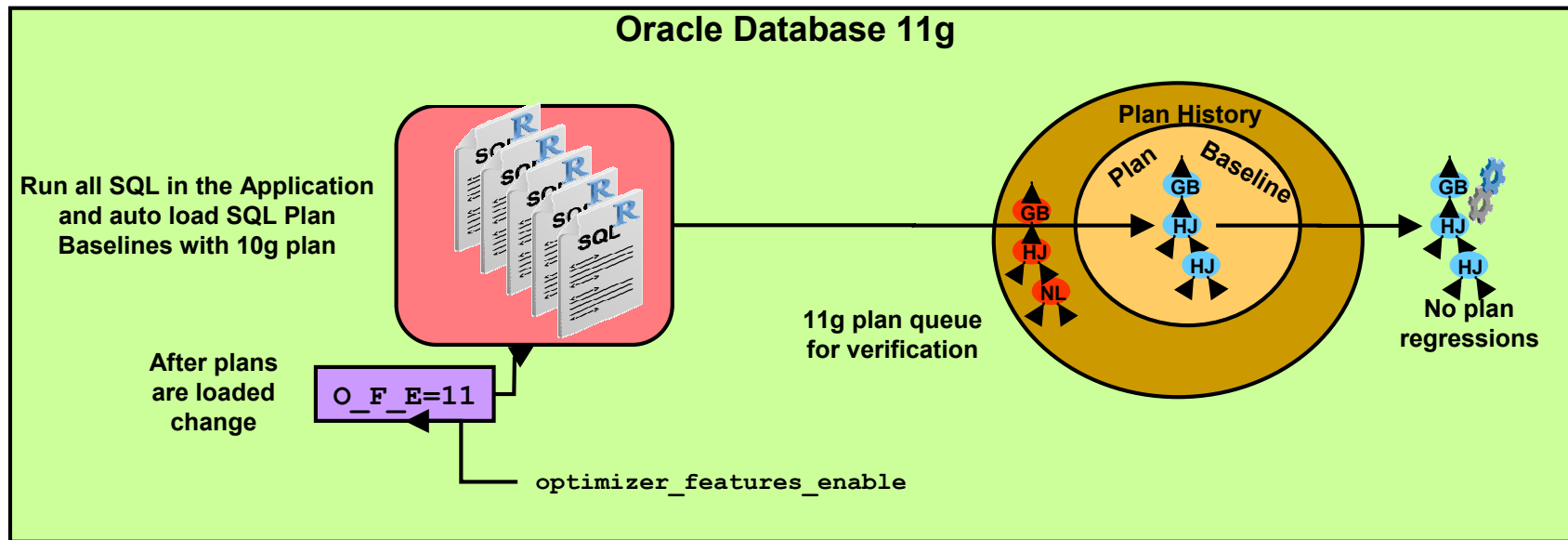
- Conduct tests on hardware identical to product
 - Same CPU brand and speed
 - Same Memory size and architecture
 - Same OS release
 - Same disk array & # of physical disk
- Use a copy of the 'live' data from product
 - 'Hand-crafted' data sets lead to unrealistic test results
- Ensure all important queries and reports are tested
 - Current high-load SQL
 - End of month / year batch jobs
- Capture all necessary performance information during tests
 - Elapse times
 - Execution plans
 - Statspack reports
 - System statistics / characteristics (IOSTAT, VMSTAT etc)
- Ensure comparable test results are available for your current Oracle release



Removing old Optimizer hints

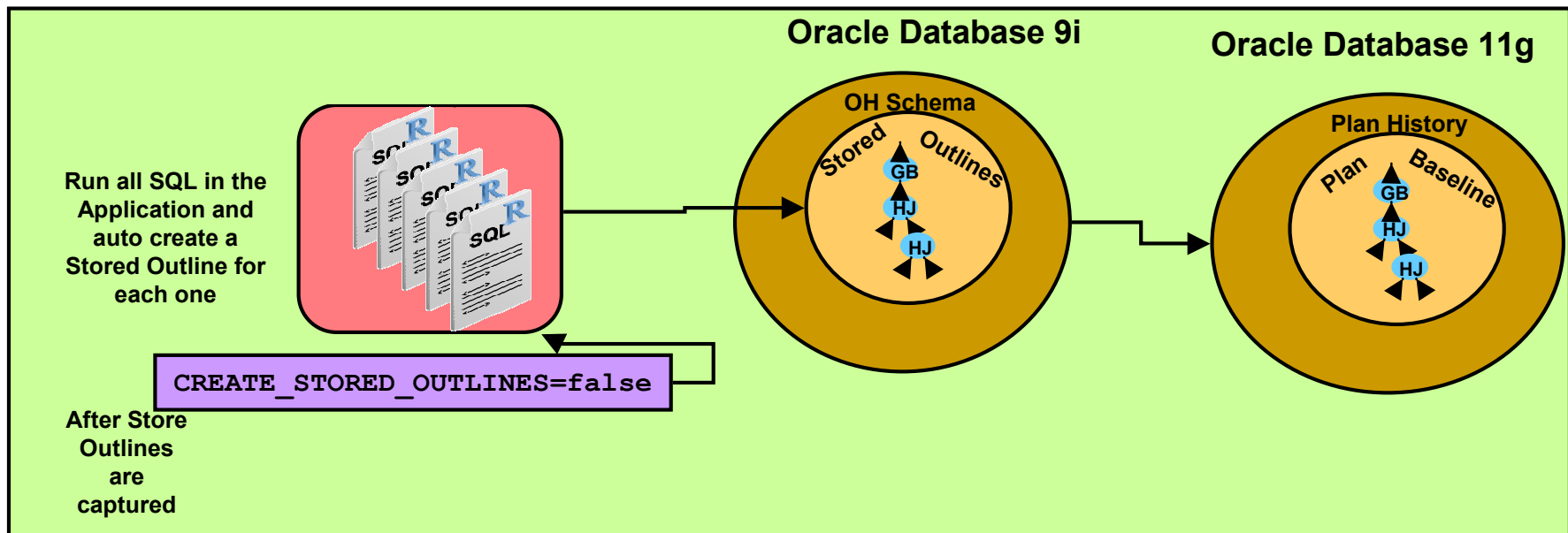
- If there are hints for every aspect of the execution plan the plan won't change between releases (Stored Outline)
- Partial hints that worked in one release may not work in another
- Test all SQL stmts with hints on the new release using the parameter `_optimizer_ignore_hints=TRUE`
 - Chance are the SQL stmts will perform better without any hints

SQL Plan Management - general upgrade strategy



- Seeding the SQL Plan Baselines with 10g plans No plan change on upgrade
- After all SQL Plan Baselines are populated switch Optimizer_Features_Enable to 11g
 - new 11g plans will only be used after they have been verified

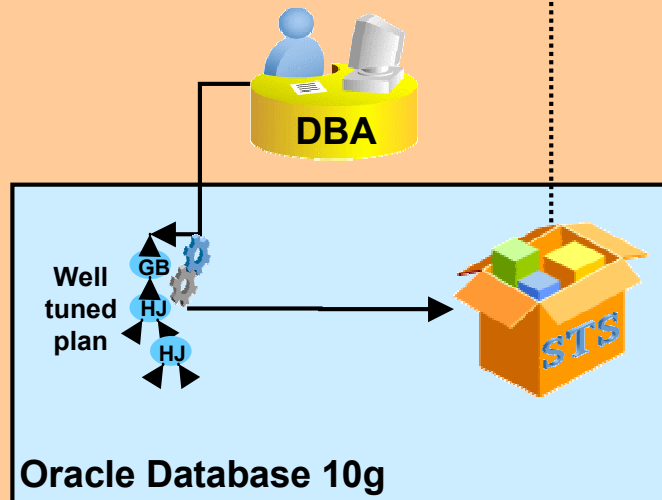
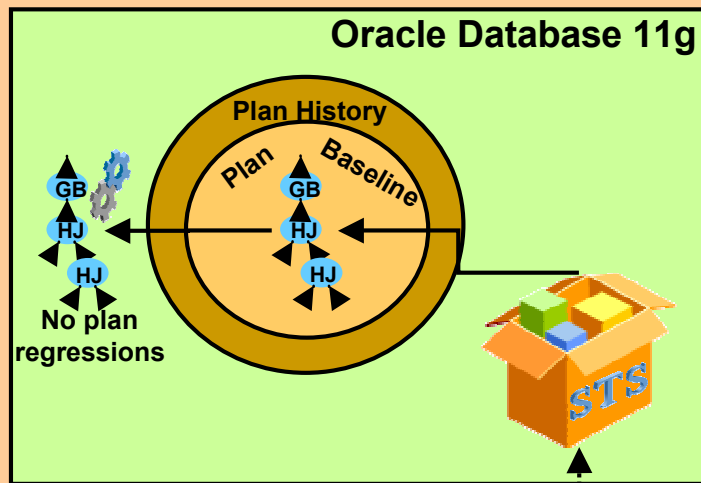
Upgrade Strategy from 9i or 10g using Stored Outlines



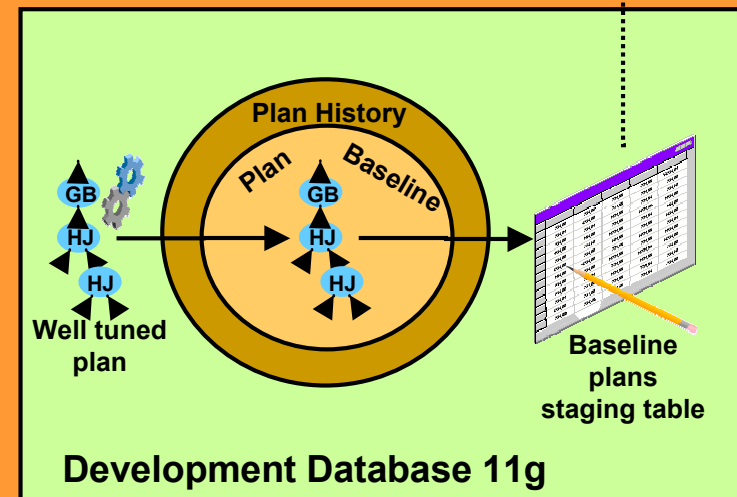
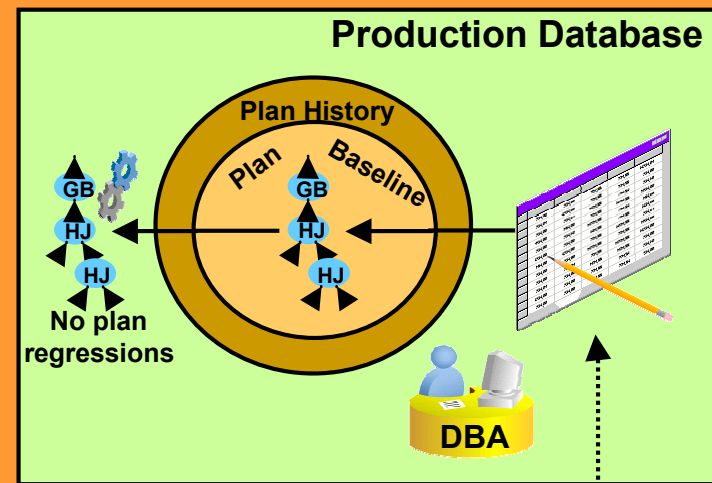
- Auto capture Stored Outlines for top SQL Statement
- Stored Outlines will provide a safety net should any SQL regress after the upgrade
- After upgrade activate Stored Outline for an regressed SQL stmts and capture the plan

Other Upgrade SQL Plan Management Scenarios

Database Upgrade using SQL Tuning Sets



Database Upgrade after 11g testing on another environment



Pre-Upgrade Checklist

- Gather Instance-wide performance statistics from the Production database (during peak load times) as a baseline
 - Hourly level 7 Statspack reports or AWR reports
 - OS stats including CPU, memory and IO (such as sar, vmstat, iostat)
- Export the Statspack or AWR schema owner
- Export Optimizer statistics into a stats table & export the table
- Make a backup of your init.ora file
- Create a SQL Tuning Set including plans for critical SQL
- Or create Stored Outlines for all key SQL statements as a backup mechanism to ensure you have a way to revert back to the 10g

Post-Upgrade Checklist

What to monitor after the upgrade





What to do with statistics after upgrade

- Use last known good set of 10g stats until system is stable
- Switch on incremental statistics for partitioned tables
 - `DBMS_STATS.SET_GLOBAL_PREFS('INCREMENTAL','TRUE');`
- Temporarily switch on pending statistics
 - `DBMS_STATS.SET_GLOBAL_PREFS('PENDING','TRUE');`
- Gather 11g statistics
 - `DBMS_STATS.GATHER_TABLE_STATS('sh','SALES');`
- Test your critical SQL statement with the pending stats
 - `Alter session set optimizer_use_pending_statistics=TRUE;`
- When proven publish the 11g statistics
 - `DBMS_STATS.PUBLISH_PENDING_STATS();`



Post-Upgrade Checklist

- Install or upgrade Statspack & set the level to 7
- Schedule Statspack snapshots every hour
- If licensed for Diagnostic Pack use AWR
- Capture OS statistics, which coincide with your statspack or AWR reports
- Identify the expensive SQL (top SQL by time, buffer gets)
- Compare these SQL statements to the top SQL statements you had prior to the upgrade
- If they are not the same, you will need to investigate why

SQL Test Case Builder





SQL Test Case Builder

Business Requirement

- Bug resolution
 - Test case required for fast bug resolution
- Not always easy to provide a test case
 - What information should be provided?
 - How much data is need?
- Getting the test case to Oracle can be tricky

Solution

- Oracle automatically creates a test case
- Collects necessary information relating to a SQL incident
- Collected data is packaged to be sent to Oracle
 - Collected data allows a developer to reproduce the problem



SQL Testcase Builder