VISCOSITY
PROFESSIONAL SERVICES

ORAPUB
POWERED BY VISCOSITY

Oracle AI Database 26ai New SQL Performance Features

NLOUG Database Cloud & Developer Day
Nov 13, 2025

NL.OUG
NL ORACLE USER GROUP

Gary Gordhamer

Oracle ACE Director
Managing Principal Consultant

@ggordham.bsky.social
linkedin.com/in/ggordhamer/
gary.gordhamer@viscosityna.com

Gary Gordhamer

Oracle ACE Director
Managing Principal Consultant

@ggordham.bsky.social
linkedin.com/in/ggordhamer/
gary.gordhamer@viscosityna.com

30+ years with Oracle technology

- Database 6.x – 23c
- Oracle Financials 7, E-Business Suite R11, R12.1, R12.2
- IDM integrations and Security hardening

Worked in many industries including

- manufacturing
- financial
- marketing
- healthcare
- utilities
- Government

Wrote the book:
“Oracle Database Performance Tuning: Pocket Solution Guide Series For Upgrading Oracle Databases”

Active member of the Oracle user community presenting frequently!

Oracle ACE Director

Viscosity's Oracle ACEs

The Oracle ACE Program

The Oracle ACE Program recognizes and rewards individuals for their contributions to the Oracle community.

Charles Kim
CEO | Co-Founder

@racdba

ACE Director

Rich Niemiec
Chief Innovation Officer

@richniemiec

ACE Director

Craig Shallahamer
Applied AI Scientist

@orapub

ACE Director

Sean Scott
Principal Consultant

oraclesean.com

ACE Director

Gary Gordhamer
Principal Consultant

ggordham.bsky.social

ACE Director

Marco Pereira
Software Architect

@markuspg

ACE Associate

viscosityna.com

@ViscosityNa

Viscosity Pillars and Delivery Models

DATA

Oracle, SQL Server, Postgres
Performance Tuning
Data Replication
Data Warehousing Analytics
Data Integration
ERP Blueprints
Database Upgrades

APPS

Oracle APEX
Web/Mobile Apps
.Net and C#
E-Business Suite
SAAS/PAAS
Custom AI Products

CLOUD

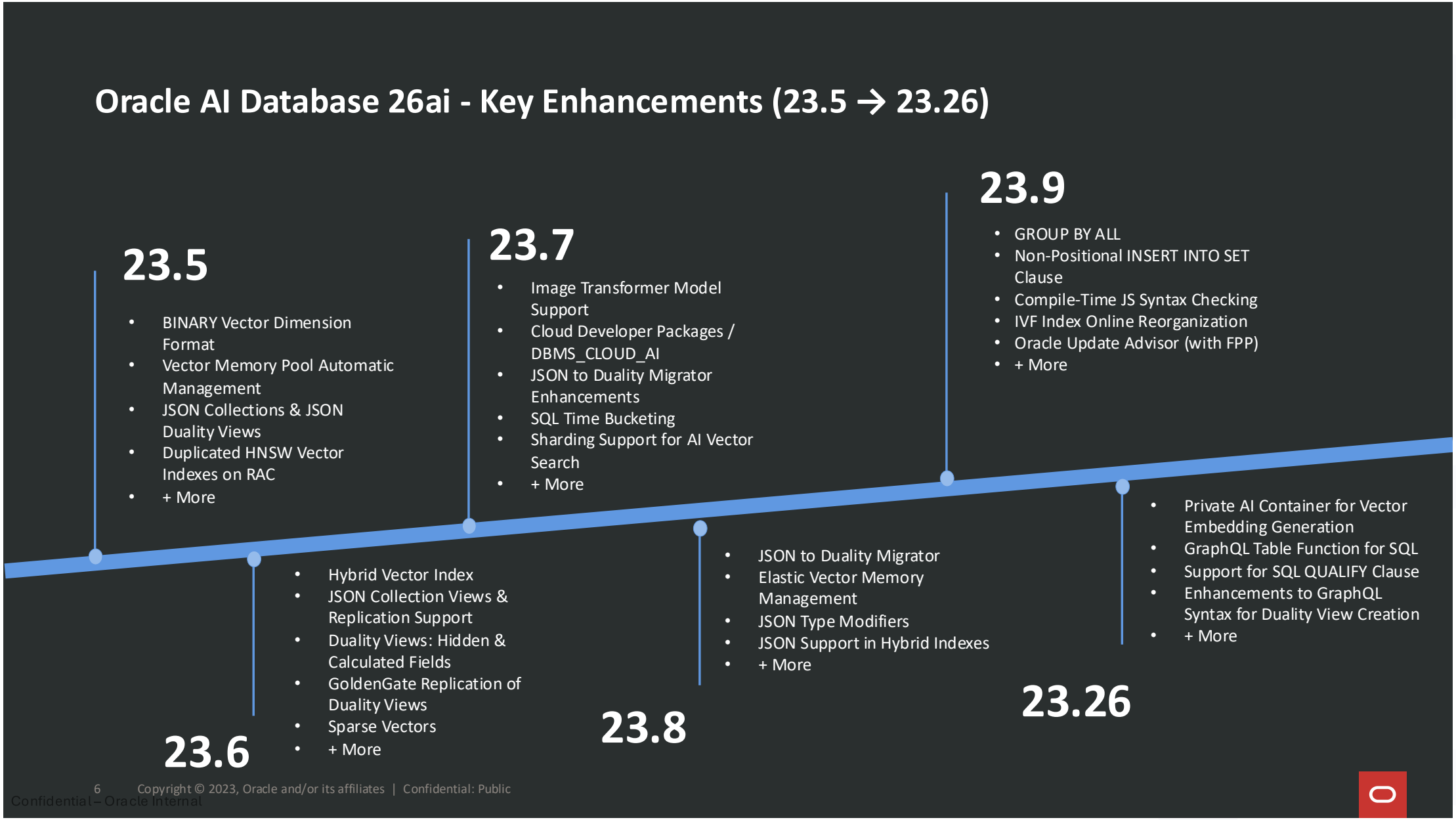
Azure Gold Partner
Cloud Migrations
Engineered Systems
Oracle Cloud Partner
Google Partner
AWS Partner Hybrid Cloud

Workshops | Assessments | Proof of Concepts | Training | Turnkey Projects | Managed Services

Agenda

- Automatic PL/SQL to SQL Transpiler
- Staging Tables
- Real-Time SQL Monitoring Enhancements
- Automatic SQL Tuning Sets (ASTS)
- Automatic SQL Plan Management (ASPM)
- SPM add verified SQL Plan Baseline
- SQL – Subsumption of views & group by placement
- Time Bucket function (23.7)
- Bonus – 19c tip – Hint report

Note: any feature in 19.x has always been in Autonomous Database

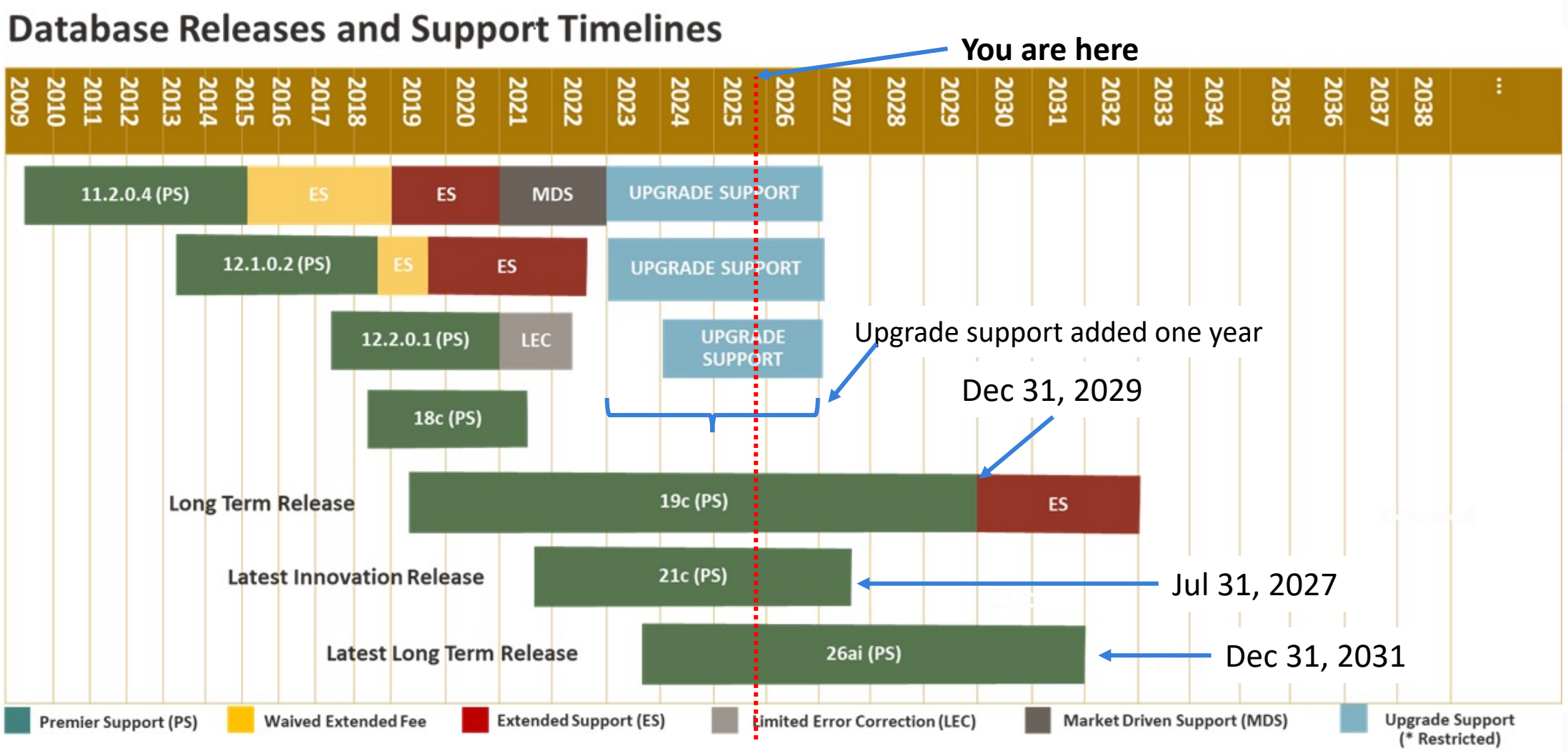


Available in the cloud (OCI, AZURE, AWS, Google)

Available on-premise for engineered systems (Exadata / ODA)

Version what?

```
SQL> select version, version_legacy, version_full from v$instance;

VERSION        VERSION_LEGACY    VERSION_FULL
-------------- ----------------- -----------------
19.0.0.0.0     19.0.0.0.0        19.27.0.0.0   <- Apr 2025 RU

23.0.0.0.0     23.0.0.0.0        23.9.0.25.07  <- Jul 2025 RU

23.0.0.0.0     23.0.0.0.0        23.26.0.0.0   <- Oct 2025 RU

23.0.0.0.0     23.0.0.0.0        23.26.1.0.0   <- Jan 2026 RU
```



Release Schedule of Current Database Releases (Doc ID 742060.1)

Want to try this out / follow along?

You can download all the scripts from GitHub

https://github.com/ggordham/ora-presentations/tree/main/26ai-sql-perf-NLOUG

Or: https://bit.ly/4oplTYu `bit.ly / 4opITYu`

All information are ”as is” and meant for teaching purposes only

DO NOT run these in any production system or system relied upon by your development teams.

PL/SQL to SQL Transpiler

When possible, move PL/SQL function in-line to SQL to reduce overhead of context switch.

Set parameter at database or session level (off by default):

```
alter session set sql_transpiler = ON;
```

```
SELECT hr.mnth_abv(hire_date) AS hire_month ...
```

Internally the Optimizer converts this into:

```
SELECT to_char(hire_date, 'MON', 'NLS_DATE_LANGUAGE=English' )
   AS hire_month ...
```

Note: lots of "rules" for eligibility, see the documentation

PL/SQL to SQL Transpiler - Example

DEMO

```
CREATE OR REPLACE FUNCTION mnth_abv (date_value DATE) RETURN VARCHAR2 IS
begin
  return to_char ( date_value, 'MON', 'NLS_DATE_LANGUAGE=English' );
end;
```

```
SELECT employee_id, first_name, last_name
  FROM employees
  WHERE mnth_abv ( hire_date ) = 'MAY';
```

```
Predicate Information (identified by operation id):
---------------------------------------------------
1 - filter("MNTH_ABV"("HIRE_DATE")='MAY')
```

```
ALTER SESSION SET sql_transpiler = ON;
```

```
Predicate Information (identified by operation id):
---------------------------------------------------
1 - filter(TO_CHAR(INTERNAL_FUNCTION("HIRE_DATE"),'MON','nls_date_language=''ENGLISH''')='MAY')
```

Function code moved in-line to make it SQL code vs PL/SQL

Real Time SQL Analysis Report

New addition to Live Monitor report (in OEM / SQL Developer)

Provides advice about the SQL statement from the Optimizer point of view

- Points out code that disables index usage
- Points out expensive operations
- Points out implicit data conversions that might ignore indexes
- Identifies where there might be missing joins

Also available in `dbms_xplan`

SQL Analysis Report - Example

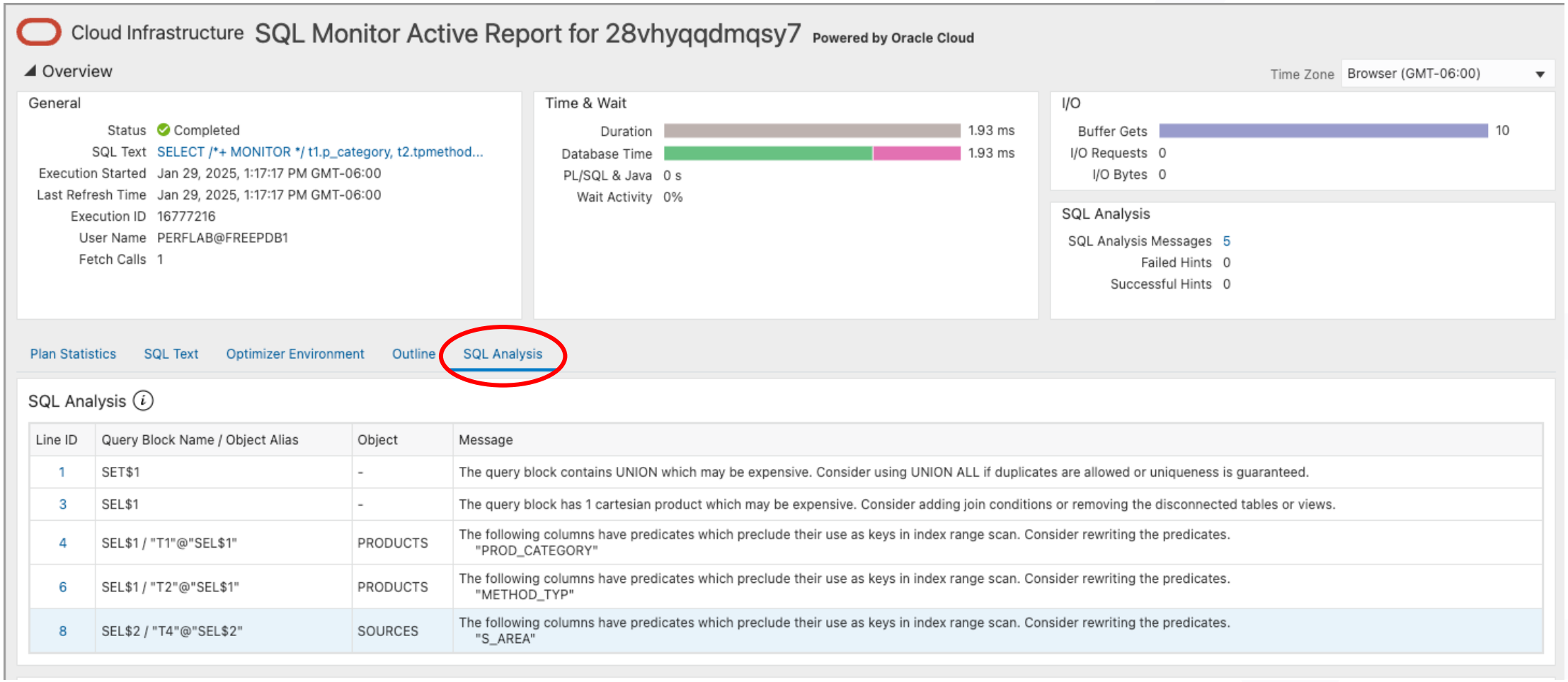
DEMO

```
SELECT t1.p_category, t2.tpmethod
  FROM products t1, products t2
  WHERE t1.prod_category || '_1' = 'SFD_1'
    AND t2.method_typ != 'SEA'
UNION
SELECT t3.p_category, t4.tpmethod
  FROM products t3, sources t4
  WHERE t3.scid = t4.scid
    AND t4.carrier   = 'AAC'
    AND t4.s_area    = :bind1;

set linesize 200
set tab off
set trims on
set pagesize 1000
column plan_table_output format a180

SELECT *
  FROM table(DBMS_XPLAN.DISPLAY_CURSOR());
```

SQL Analysis Report - Example

```
SQL Analysis Report (identified by operation id/Query Block Name/Object Alias):
--------------------------------------------------------------------------------
1 -  SET$1
           -  The query block contains UNION which may be expensive.
              Consider using UNION ALL if duplicates are allowed or
              uniqueness is guaranteed.
3 -  SEL$1
           -  The query block has 1 cartesian product which may be
              expensive. Consider adding join conditions or removing the
              disconnected tables or views.
4 -  SEL$1 / "T1"@"SEL$1"
           -  The following columns have predicates which preclude their
              use as keys in index range scan. Consider rewriting the
              predicates.
                "PROD_CATEGORY"
6 -  SEL$1 / "T2"@"SEL$1"
           -  The following columns have predicates which preclude their
              use as keys in index range scan. Consider rewriting the
              predicates.
                "METHOD_TYP"
8 -  SEL$2 / "T4"@"SEL$2"
           -  The following columns have predicates which preclude their
              use as keys in index range scan. Consider rewriting the
              predicates.
                "S_AREA"
```

SQL Analysis Report - HTML



Available in OEM + manual creation using DBMS_SQL_MONITOR.REPORT_SQL_MONITOR

viscosityna.com @ViscosityNa

SQL Monitoring – SQL Diagnostic Report

Built in tool to help with troubleshooting performance issues with a SQL statement.

Generates an HTML based report

- Deep diagnostic information on the SQL, including:
  - Execution plan history
  - Non-default database parameters
  - Statistics history
  - Index information

```
dbms_sqldiag.report_sql(sql_id => 'fk70ks7ztdspf', level => 'TYPICAL');
```

Like SQL Health Check script
SQL Tuning Health-Check Script (SQLHC) (Doc ID 1366133.1)

SQL Diagnostic Report - Example

SQL Report SQLID: f5nkjzxn2ra8r Report Date: 30-JAN-25 07:52

PLANS

Last Execution Plan
Plan Summary
All Execution Plans
Historical Execution Plans
Plan Control Objects

GLOBAL

SQL Text
SQL History
Parameters(Non-Default Values)
Observations
DBMS_STATS Table Preferences
Non Standard Optimizer Parameters
SQL Sharing Info

CURSOR SHARING AND BINDS

TABLES

Table Summary
Table Columns
Table Partitions
Table Constraints
Table Modifications

INDEXES

Index Summary
Index Columns

ACTIVE SESSION HISTORY(ASH)

MISCELLANEOUS

▼ SQL Text

The SQL text of the target query

Text
select /* SPM_TEST_QUERY_Q1 */ sum(t2.amount) from sales_area1 t1, sales_area2 t2 where t1.sale_code = t2.sale_code and t1.sale_type = 1

Staging Tables

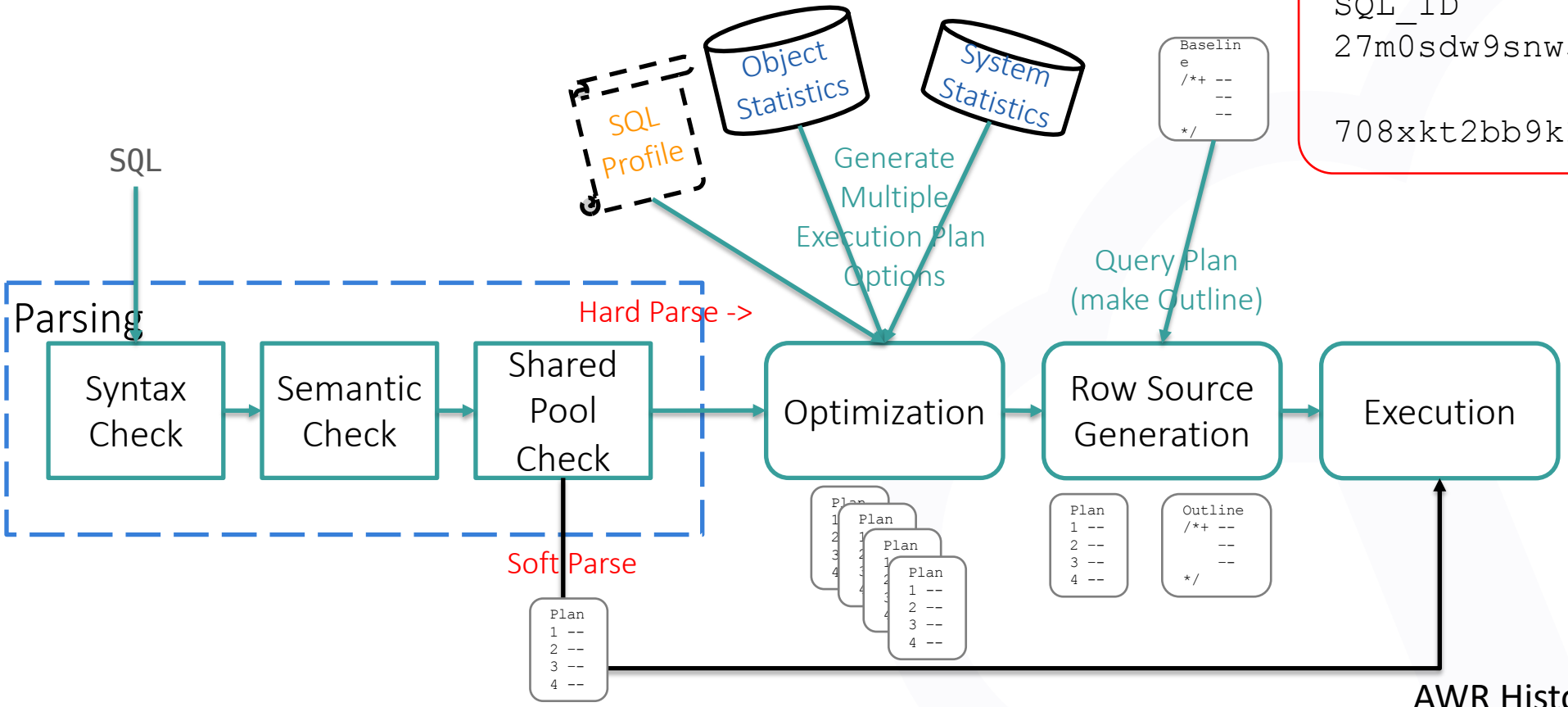
Pre optimized for data loads

- Heap organized – no free space is saved in blocks for updates, and the fewest number of blocks are used
- No statistics – dynamic sampling is used
- Bypasses the recycle bin
- Compression is not allowed, partitioning is allowed

```
CREATE TABLE oe.sales_hist_stag
  (order_date date,
   order_id   number,
   total_purchase number)
  FOR STAGING;
```

SQL Plan Stability

Optimizer SQL Steps



Cursor Cache (few hours?)

SQL_ID	Text	Plan
27m0sdw9snw59	SELECT…	Plan 1 -- 2 --
708xkt2bb9k1f	SELECT…	Plan 1 -- 2 --

AWR History (Top 30?)

SQL_ID	Text	Plan
5nc2ws664t1gx	SELECT…	Plan 1 -- 2 --
708xkt2bb9k1f	SELECT…	Plan 1 -- 2 --

Executions plans exist in cursor cache until aged out and in AWR history if they are in top 30 resource consumers.

Note: AWR could be more than 30 if you have modified the setting.

Worst SQL performance issue?

This was running fine yesterday, what happened?

What was the execution plan yesterday?

- Is it in cursor cache? – probably not
- Is it in the top 30 AWR? - well if it ran well, probably not

If you knew about this coming you probably would have saved the SQL to a SQL Tuning Set (STS)

- Like AWR history, except you pick what SQL to save
- Except, you must save the SQL

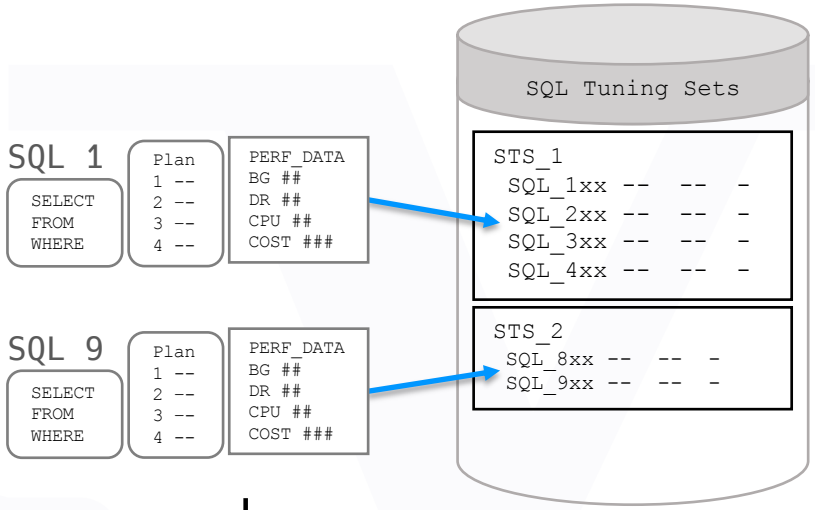
Enter 23ai – Automatic SQL Tuning Set!

SQL Tuning Set (STS)

Way to capture information on a specific or set of SQL statements

Store in SYSAUX tablespace

Traditionally you would create an STS and manually add statements from cursor cache



View SQL in: `dba_sqlset_statements`

Used by tuning tasks!

Automatic STS: `'SYS_AUTO_STS'`
Enabled by default in 23ai, runs every 900 seconds.
Was introduced in 19.7 but needs to be enabled (see MOS note).

Automatic SQL Tuning Sets

Stored in STS called “SYS_AUTO_STS”

Background job captures statements from cursor cache

- Capturing all your SQL statements (well not all – rules not fully documented)

```
BEGIN
  DBMS_Auto_Task_Admin.Enable(
     Client_Name => 'Auto STS Capture Task’,
     Operation => NULL, Window_name => NULL);
END;
```

Excludes:
Internal SQL
SQL without plans
Dynamic SQL

```
SELECT enabled
  FROM dba_autotask_schedule_control
  WHERE task_name = 'Auto STS Capture Task';
```

- Note: it will exclude large quantities of dynamic SQL

To see what statements have been captured query:

```
SELECT *
  FROM dba_sqlset_statements
  WHERE sqlset_name = 'SYS_AUTO_STS';
```

Provides SQL plan history for automated tuning tasks!

Automatic SQL Tuning Sets

Used by following tuning tasks / tools:

- SPM Evolve task
- Automatic SPM task
- SPM “One-shot”
- Real-time SQL Plan Management – 23ai
- Automatic: indexing, partitioning, clustering, materialized view

You can use information from ASTS for your own tuning!

Unused statements are purged after 53 weeks

Enhanced SQL history – for DEVOPS

NOT STS

Previously only queries over 5 seconds long or parallel were tracked

Once enabled best effort all SQL history will be stored in a new view

- V$SQL_HISTORY

Set the following parameter

- SQL_HISTORY_ENABLED = TRUE

Number of SQL depends on memory, tries for 50 per session

Parameter documentation:
https://docs.oracle.com/en/database/oracle/oracle-database/23/refrn/SQL_HISTORY_ENABLED.html#REFRN-GUID-2DA2C15D-9C95-4D3E-A740-D1959F0F1E79

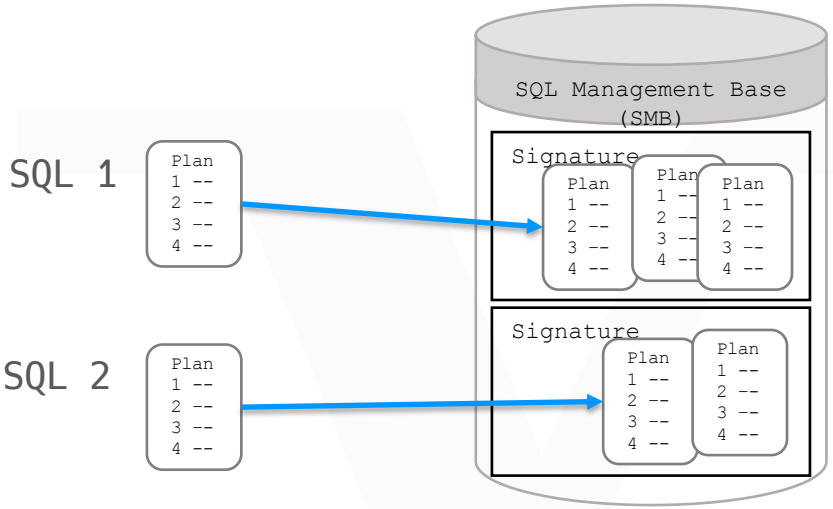
V$SQL_HISTORY documentation:
https://docs.oracle.com/en/database/oracle/oracle-database/23/refrn/V-SQL_HISTORY.html#GUID-D8DD4D0C-034C-40E1-8D7F-2CF732092294

SQL Plan Management (SPM)

First appeared in Oracle 11.1

- Replaced OUTLINES
- Allows for more control on how the optimizer chooses execution plans

SQL Execution plans are captured and stored in SQL Management Base (SMB) part of the SYSAUX tablespace



Plans are ”Accepted” to prefer their usage

Creates “PLAN STABILITY”

SPM (Baselines) vs. Profiles

SQL Profile

- Licensed, part of tuning pack
- Stored HINT for SQL
- Statistical modification of cardinality calculation
  OR just a stored hint
- Only one profile per SQL
- Can be ignored (HINT)
- Can be FORCE matching

Influences plan creation

SQL Baseline

- Included in all editions
- Stored Execution Plan
- Multiple baselines (Plans) allowed in Enterprise Edition
- Can adjust plan based on bind variables
- Evolve over time
- Automated testing

Influences plan choice

Both are useful tools and can be used together. Neither is a replacement for fixing the source problem of table structure design, data layout, or SQL code.

SPM – over the years

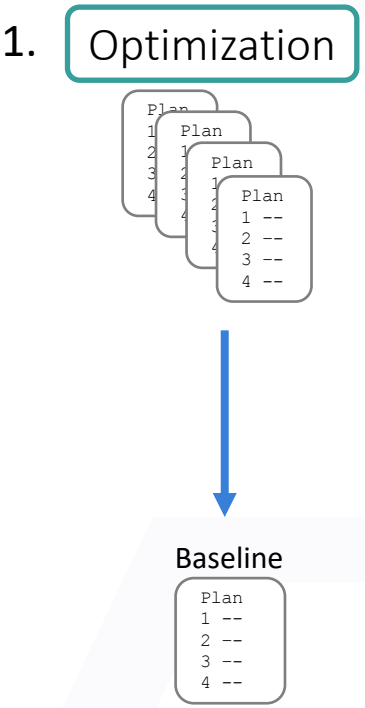
SPM Steps

1. Capture baselines (SQL Execution Plans) – manual or automatic (all?)
2. Mark as usable (verified or manually)
3. Evolve – look for new or better SQL Execution Plans

Features by version

- 11g – auto or manual capture; use; evolve baselines manually
- 12c – Daily `SYS_AUTO_SPM_EVOLVE_TASK` in the maintenance window
- 19c – Automatic SPM – high frequence background task (*19.22 and above*)
  - Runs tasks every 15 – 60 minutes (configurable) – in the background (disabled by default)
  - Looks for high consumption SQL statements in AWR + ASTS
  - Looks for alternate plans, runs evolve task and generates new baselines if applicable
- 23ai – Real-time SPM
  - Optimizer identifies SQL Execution Plan change during hard parse
  - After execution, makes a baseline for best plan (new or old) based on performance

SPM Auto Capture (11g +)



Set database parameter

```
OPTIMIZER_CAPTURE_SQL_PLAN_BASELINES = TRUE
```

Can be set at database level, or session level

Configure capture using DBMS_SPM package:

```
EXEC DBMS_SPM.CONFIGURE(
            parameter_name => 'AUTO_CAPTURE_SQL_TEXT',
              parameter_value => '%TEST_ONLY%',
            allow => false);
```

Excludes all SQL that contains text “TEST_ONLY” in upper case, can filter on ACTION, MODULE, PARSING SCHEMA, or SQL TEXT.

Note: FALSE in this syntax is to exclude, to include only SQL containing the text “TEST_ONLY” then set TRUE.

Once a SQL statement has a baseline, new baselines will be collected automatically unless there is a FIXED baseline for that statement.

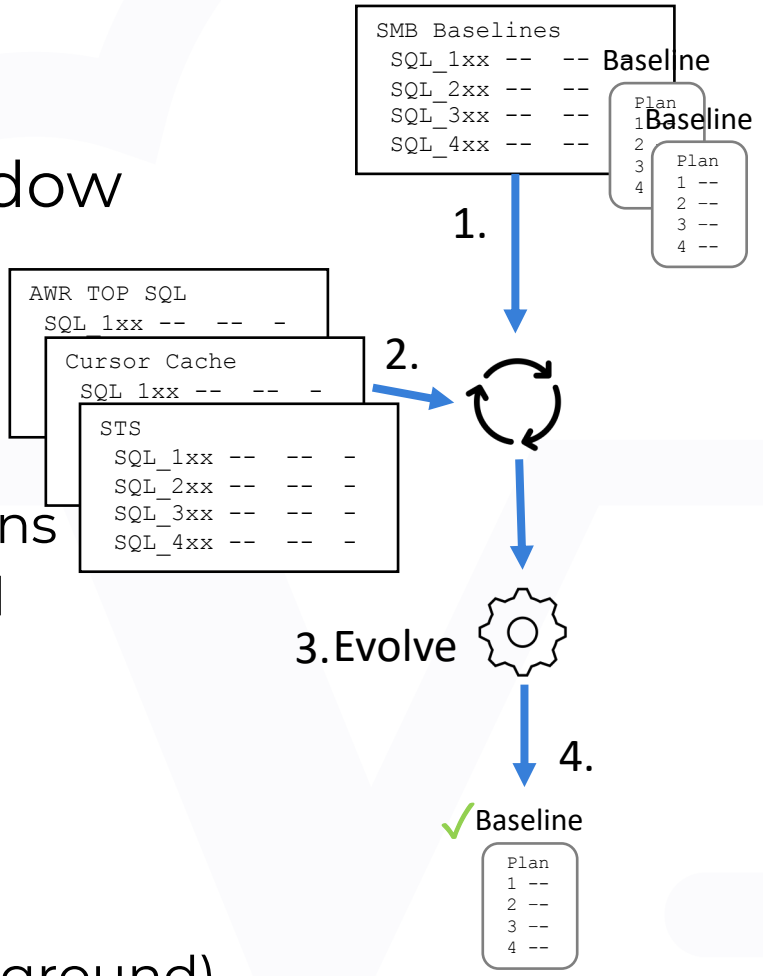
Auto Evolve SPM (12c +)

Enabled by default, part of the Automatic SQL Tuning task that runs in the maintenance window

```
SELECT CLIENT_NAME, STATUS FROM   DBA_AUTOTASK_CLIENT
WHERE  CLIENT_NAME = 'sql tuning advisor';
```

Steps:

1. Look at existing captured baselines
2. Search AWR, Cursor Cache, STS for alternate plans
3. Test new baselines that have not been accepted
4. Accept baselines that perform better



Issues:

- Only looks at existing baselines
- Won't look at SQL with FIXED baselines
- Testing uses additional resources (running in background)

Automatic SPM (19.22 +)

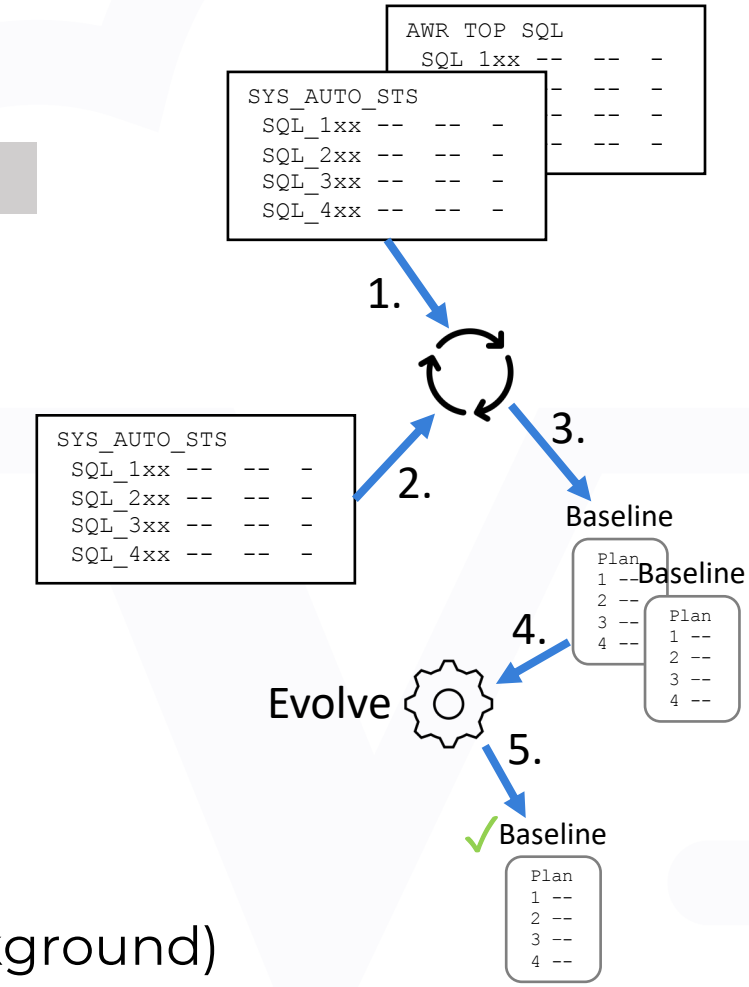
aka: High Frequency SPM Evolve Advisor Task

Disabled by default, to enable:

```
exec dbms_spm.configure('AUTO_SPM_EVOLVE_TASK', 'ON')
```

Steps:

1. Look for resource intensive SQL in AWR / ASTS
2. Find alternative execution plans in ASTS
3. Capture execution plans as baselines (SPM)
4. Evolve (test) the execution plans (baselines)
5. Accept the best execution plans (baseline)



Issues:

- Delay between execution and finding better plans
- Same plan may not be reproduced
- Testing uses additional resources (running in background)

Real-Time SPM (23ai +)

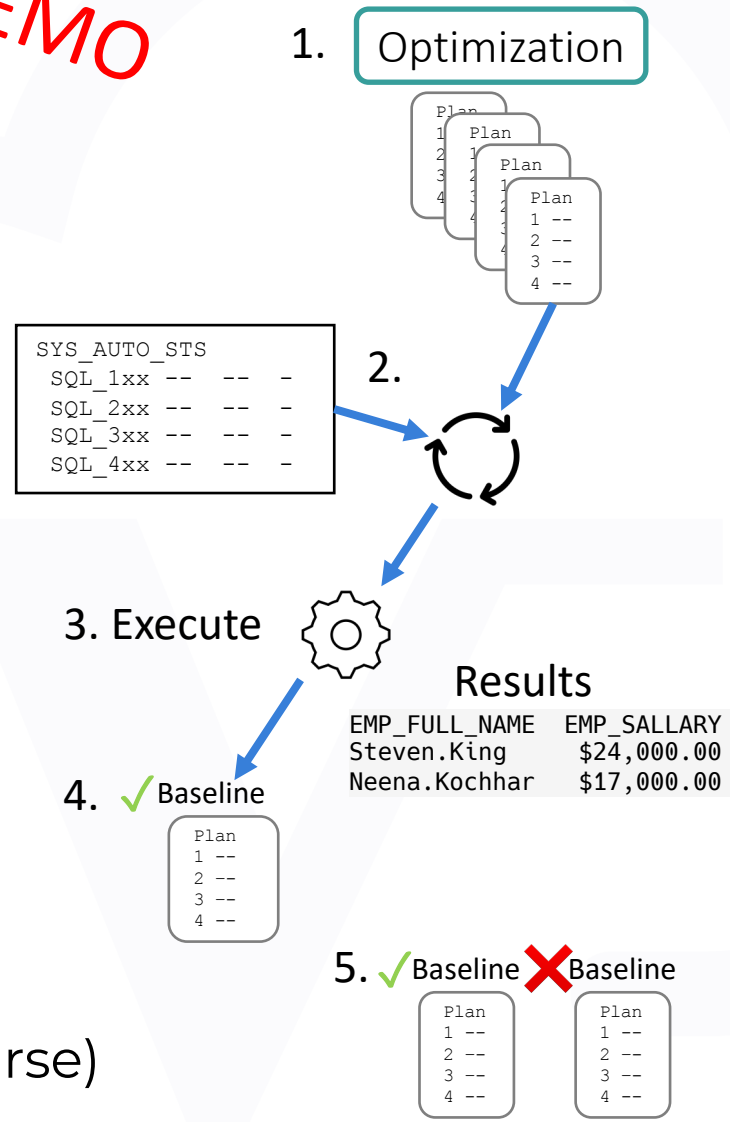
DEMO

Disabled by default, to enable:

```
exec dbms_spm.configure('AUTO_SPM_EVOLVE_TASK', 'AUTO')
```

Steps:

1. Detects plan change (during hard parse)
2. Find previous execution plans in ASTS
3. Verify execution cost, run the lowest cost plan
4. If new plan is better, make baseline, if not make baseline for previous plan
5. If during future execution, performance regresses – re-evaluate the baselines and disables as needed



Benefits:

- Tested in foreground
- Resolves issues immediately! (worst case next hard parse)

SPM “One-Shot” – quick fix

One call to fix performance issue on a single statement:

- Looks for SQL Plans (ASTS, AWR, SPM, cursor cache)
- Creates baselines and tests them.
- Automatically accepts good performing baselines.

Included in 23ai

Back ported to 19c – patch 34534504 (19.11 – 19.21)

Included in RU 19.22 (Jan 2024)

```
exec :report := dbms_spm.add_verified_sql_plan_baseline('9y5vtn2knayhs');

select :report from dual;
```

A note on FIXED baselines

When you mark a baseline as FIXED, Oracle will try to use this execution plan over others.

- You can have multiple FIXED baselines (EE / ADB)
- New baselines will not be captured automatically
- Auto and Real-Time SPM will not attempt to evolve plans

Three more items

Subsumption of views

The Optimizer will replace multiple subqueries or views with one

```
SELECT * FROM
  (SELECT avg(p.unit_price) AS avg_canc
     FROM co.orders o, co.products p,
          co.order_items oi
     WHERE o.order_id = oi.order_id
       AND oi.product_id = p.product_id
       AND ORDER_STATUS = 'CANCELLED') v1,
   (SELECT avg(p.unit_price) AS avg_ref
      FROM co.orders o, co.products p,
           co.order_items oi
      WHERE o.order_id = oi.order_id
        AND oi.product_id = p.product_id
        AND ORDER_STATUS = 'REFUNDED') v2;
```

```
SELECT AVG(CASE WHEN  ORDER_STATUS = 'CANCELLED'
           THEN p.unit_price ELSE NULL END)
         AS avg_canc,
       AVG(CASE WHEN  ORDER_STATUS = 'REFUNDED'
           THEN p.unit_price ELSE NULL END)
         AS avg_ref
  FROM co.orders o, co.products p,
       co.order_items oi
  WHERE o.order_id = oi.order_id
    AND oi.product_id = p.product_id
    AND ORDER_STATUS IN ('CANCELLED', 'REFUNDED');
```

Automatic feature, if OPTIMIZER_FEATURES_ENABLED = 23.1.0.0

Subsumption of views - Example

DEMO

The Optimizer will replace multiple subqueries or views with one

```
Plan hash value: 948351665

-----------------------------------------------
| Id  | Operation              | Name         |
-----------------------------------------------
|   0 | SELECT STATEMENT       |              |
|   1 |  NESTED LOOPS          |              |
|   2 |   VIEW                 |              |
|   3 |    SORT AGGREGATE      |              |
|*  4 |     HASH JOIN          |              |
|*  5 |      HASH JOIN         |              |
|*  6 |       TABLE ACCESS FULL| ORDERS       |
|   7 |       TABLE ACCESS FULL| ORDER_ITEMS  |
|   8 |      TABLE ACCESS FULL | PRODUCTS     |
|   9 |   VIEW                 |              |
|  10 |    SORT AGGREGATE      |              |
|* 11 |     HASH JOIN          |              |
|* 12 |      HASH JOIN         |              |
|* 13 |       TABLE ACCESS FULL| ORDERS       |
|  14 |       TABLE ACCESS FULL| ORDER_ITEMS  |
|  15 |      TABLE ACCESS FULL | PRODUCTS     |
-----------------------------------------------
```

```
Plan hash value: 2683287004

-----------------------------------------------
| Id  | Operation              | Name         |
-----------------------------------------------
|   0 | SELECT STATEMENT       |              |
|   1 |  VIEW                  | VW_SVS_FB088594 |
|   2 |   SORT AGGREGATE       |              |
|*  3 |    HASH JOIN           |              |
|   4 |     TABLE ACCESS FULL  | PRODUCTS     |
|*  5 |     HASH JOIN          |              |
|*  6 |      TABLE ACCESS FULL | ORDERS       |
|   7 |      TABLE ACCESS FULL | ORDER_ITEMS  |
-----------------------------------------------
```

@ViscosityNa

Group By Pushdown (GBP)

Multiple features GROUP BY features exist, these are new ones moving GROUP by to earlier in the process.

- Push GROUP BY from main query to sub-views of query to reduce row sets

```
SELECT
  FROM view_1,
       view_2
   WHERE
   GROUP BY col_1
```

```
SELECT
  FROM (SELECT FROM view_1
        GROUP BY col_1),
        (SELECT FROM view_2
         GROUP BY col_1)
   WHERE
   GROUP BY col_1
```

- Decompose expressions, to perform group by earlier

```
SELECT t1.region_id,SUM(t1.amount*t2.icount)
  FROM sales t1, sales_emea t2
  WHERE t1.sale_code = t2.sale_code
  GROUP BY t1.region_id;
```

```
SELECT t1.region_id, SUM(t1.amount*v_gbc_3.sb)
  FROM sales t1,
    (SELECT t2.sale_code, SUM(t2.icount) as sb
       FROM sales_emea t2
       GROUP BY t2.sale_code) v_gbc_3
  WHERE t1.sale_code = v_gbc_3.sale_code
  GROUP BY t1.region_id;
```

Automatic feature, if OPTIMIZER_FEATURES_ENABLED = 23.1.0.0

SQL Time Bucketing – 23ai RU7 (23.7)

Simplifies bucketing of time windows / groups

Example: splitting data into 5 minute intervals:

```
WITH rws AS (
   SELECT date'2025-02-01' origin, datetime, 1440 units_in_day, 5 stride_interval
     FROM time_data),
  buckets AS (
    SELECT FLOOR(ROUND((datetime - origin) * units_in_day, 9) / stride_interval) bucket#, r.*
      FROM rws r),
  intervals AS (
    SELECT origin + ( bucket# * stride_interval / units_in_day ) start_datetime,
           origin + ( ( bucket# + 1 ) * stride_interval / units_in_day ) end_datetime
      FROM buckets)
SELECT /*+ MONITOR */ COUNT(*), start_datetime, end_datetime
  FROM intervals
  GROUP BY start_datetime, end_datetime
  ORDER BY start_datetime;
```

Add interval to source data

Generate buckets

Put data into buckets

SQL Time Bucketing – 23ai RU7 (23.7)

Simplifies bucketing of time windows / groups

Example: splitting data into 5 minute intervals:

```
SELECT /*+ MONITOR */ COUNT(*),
       TIME_BUCKET (datetime, interval '5' minute, date'2025-02-01') start_date,
       TIME_BUCKET (datetime, interval '5' minute, date'2025-02-01', end) end_date
  FROM time_data
  GROUP BY start_date, end_date
  ORDER BY start_date;
```

19c tip – Hint report

Hinting a SQL without changing code

https://github.com/ggordham/ora-presentations/tree/main/NoCOUG-sql-hint

```
SELECT c.cust_state_province, sum(s.amount_sold)
  FROM sh.sales s, sh.customers c,
    (SELECT ch.channel_id FROM sh.channels ch
       WHERE ch.channel_class_id = 13) chan13
  WHERE c.cust_id = s.cust_id
    AND s.channel_id = chan13.channel_id
    AND c.cust_city_id IN (52114,52292,51738,51508)
  GROUP BY c.cust_state_province;
```

```
-----------------------------------------------------------------------------------------------------------------------------------
| Id  | Operation                | Name      | E-Rows |E-Bytes| Cost (%CPU)| E-Time   | Pstart| Pstop |  OMem |  1Mem | Used-Mem |
-----------------------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT         |           |        |       |   945 (100)|          |       |       |       |       |          |
|   1 |  HASH GROUP BY           |           |      2 |    80 |   945   (2)| 00:00:01 |       |       |  1034K|  1034K|  622K (0)|
|*  2 |   HASH JOIN              |           |  19420 |   758K|   944   (2)| 00:00:01 |       |       |  2546K|  2546K|  750K (0)|
|*  3 |    TABLE ACCESS FULL     | CHANNELS  |      2 |    12 |     3   (0)| 00:00:01 |       |       |       |       |          |
|*  4 |    HASH JOIN             |           |  46608 |  1547K|   941   (2)| 00:00:01 |       |       |  1399K|  1399K| 2691K (0)|
|*  5 |     TABLE ACCESS FULL    | CUSTOMERS |    358 |  7518 |   423   (1)| 00:00:01 |       |       |       |       |          |
|   6 |     PARTITION RANGE ALL  |           |    918K|    11M|   515   (2)| 00:00:01 |     1 |    28 |       |       |          |
|   7 |      TABLE ACCESS FULL   | SALES     |    918K|    11M|   515   (2)| 00:00:01 |     1 |    28 |       |       |          |
-----------------------------------------------------------------------------------------------------------------------------------
```

I'd like to hint the query to use indexes, and don't have access to the source code.

Problem 1 – Hints not working, why?

```
SELECT /*+ monitor no_adaptive_plan full( s ) gather_plan_stats
           index(sales SALES_CHANNEL_BIX ) index( c customers_pk ) full( c ) */
       c.cust_state_province, sum(s.amount_sold)
  FROM sh.sales s, sh.customers c,
    (SELECT ch.channel_id FROM sh.channels ch
       WHERE ch.channel_class_id = 13) chan13
  WHERE c.cust_id = s.cust_id
    AND s.channel_id = chan13.channel_id
    AND c.cust_city_id IN (52114,52292,51738,51508)
  GROUP BY c.cust_state_province;
```

```
-----------------------------------------------------------------------------------------------------------------------------------
| Id  | Operation              | Name      | E-Rows |E-Bytes| Cost (%CPU)| E-Time   | Pstart| Pstop |  OMem |  1Mem | Used-Mem |
-----------------------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT       |           |        |       |   945 (100)|          |       |       |       |       |          |
|   1 |  HASH GROUP BY         |           |      2 |    80 |   945   (2)| 00:00:01 |       |       |  1034K|  1034K|  622K (0)|
|*  2 |   HASH JOIN            |           |  19420 |   758K|   944   (2)| 00:00:01 |       |       |  2546K|  2546K|  750K (0)|
|*  3 |    TABLE ACCESS FULL   | CHANNELS  |      2 |    12 |     3   (0)| 00:00:01 |       |       |       |       |          |
|*  4 |    HASH JOIN           |           |  46608 |  1547K|   941   (2)| 00:00:01 |       |       |  1399K|  1399K| 2691K (0)|
|*  5 |     TABLE ACCESS FULL  | CUSTOMERS |    358 |  7518 |   423   (1)| 00:00:01 |       |       |       |       |          |
|   6 |     PARTITION RANGE ALL|           |    918K|    11M|   515   (2)| 00:00:01 |     1 |    28 |       |       |          |
|   7 |      TABLE ACCESS FULL | SALES     |    918K|    11M|   515   (2)| 00:00:01 |     1 |    28 |       |       |          |
-----------------------------------------------------------------------------------------------------------------------------------
```

19c new feature – Hint report

```
Hint Report (identified by operation id / Query Block Name / Object Alias):
Total hints for statement: 6 (U - Unused (2), N - Unresolved (1), E - Syntax error (1))
------------------------------------------------------------------------------

   0 -  STATEMENT
          -  no_adaptive_plan

   0 -  SEL$1
         E -  gather_plan_stats

   1 -  SEL$F5BB74E1
         N -  index( sales SALES_CHANNEL_BIX )

   5 -  SEL$F5BB74E1 / C@SEL$1
         U -  full( c ) / hint conflicts with another in sibling query block
         U -  index( c customers_pk ) / hint conflicts with another in sibling query block

   7 -  SEL$F5BB74E1 / S@SEL$1
          -  full( s )
```

Ah! - Lots of issues ☺

I should have done less cut ‘n’ paste

SQL Monitor

Cloud Infrastructure **SQL Monitor Active Report for 5raftduabwup9** Powered by Oracle Cloud

Overview

Time Zone: Browser (GMT-05:00)

General

Status	Completed
SQL Text	SELECT /*+ monitor no_adaptive_plan ful...
Execution Started	Mar 11, 2025, 8:35:42 AM GMT-05:00
Last Refresh Time	Mar 11, 2025, 8:35:42 AM GMT-05:00
Execution ID	16777218
User Name	PERFLAB
Fetch Calls	2

Time & Wait

Duration	76.51 ms
Database Time	76.51 ms
PL/SQL & Java	0 s
Wait Activity	0%

I/O

Buffer Gets	3,129
I/O Requests	0
I/O Bytes	0

SQL Analysis

SQL Analysis Messages	0
Failed Hints	2 1 1
Successful Hints	0

Plan Statistics | SQL Text | Optimizer Environment | Outline | SQL Analysis

SQL Analysis

Line ID	Query Block Name / Object Alias	Object	Message
No data to display.			

Failed Hints

Line ID	Query Block Name / Object Alias	Object	Hint Text	Reason	Status
0	SEL$1	-	gather_plan_stats	-	Syntax Error
1	SEL$F5BB74E1	-	index(sales SALES_CHANNEL_BIX)	-	Unresolved
5	SEL$F5BB74E1 / C@SEL$1	CUSTOMERS	full(c)	hint conflicts with another in sibling query block	Unused
	SEL$F5BB74E1 / C@SEL$1	CUSTOMERS	index(c customers_pk)	hint conflicts with another in sibling query block	Unused

Old school method

```
SELECT *
  FROM table(
    DBMS_XPLAN.DISPLAY_CURSOR(FORMAT=>'ALL ALLSTATS LAST +cost +bytes +OUTLINE'));
```

If you are not seeing the hint report, add to the FORMAT:

`+HINT_REPORT`

More information at:

Hint Usage Reporting - 19c New Feature (Doc ID 2735444.1)

Hinted statement

```
SELECT /*+ monitor no_adaptive_plan gather_plan_statistics index( s SALES_CHANNEL_BIX )
         index( c customers_pk ) */
       c.cust_state_province, sum(s.amount_sold)
  FROM sh.sales s, sh.customers c,
    (SELECT ch.channel_id
       FROM sh.channels ch
       WHERE ch.channel_class_id = 13) chan13
  WHERE c.cust_id = s.cust_id
    AND s.channel_id = chan13.channel_id
    AND c.cust_city_id IN ( 52114,52292,51738,51508)
  GROUP BY c.cust_state_province;
```

```
SQL_ID  009ym2v4nkdjp, child number 0
-------------------------------------------------------------------------------------------------------------------------------
| Id  | Operation                                | Name             | Starts | E-Rows |E-Bytes| Cost (%CPU)| E-Time   | A-Rows |
-------------------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                         |                  |      1 |        |       |  9205 (100)|          |      2 |
|   1 |  HASH GROUP BY                           |                  |      1 |    133 |  5187 |  9205   (1)| 00:00:01 |      2 |
|   2 |   NESTED LOOPS                           |                  |      1 |    358 | 13962 |  9204   (1)| 00:00:01 |     26 |
|   3 |    NESTED LOOPS                          |                  |      1 |   7059 | 13962 |  9204   (1)| 00:00:01 |   3614 |
|   4 |     VIEW                                 | VW_GBC_10        |      1 |   7059 |   124K|  2143   (1)| 00:00:01 |   3614 |
|   5 |      HASH GROUP BY                       |                  |      1 |   7059 |   130K|  2143   (1)| 00:00:01 |   3614 |
|   6 |       NESTED LOOPS                       |                  |      1 |    382K|  7103K|  2133   (1)| 00:00:01 |    118K|
|   7 |        NESTED LOOPS                      |                  |      1 |    382K|  7103K|  2133   (1)| 00:00:01 |    118K|
|*  8 |         TABLE ACCESS FULL                | CHANNELS         |      1 |      2 |    12 |     3   (0)| 00:00:01 |      2 |
|   9 |         PARTITION RANGE ALL              |                  |      2 |        |       |            |          |    118K|
|  10 |          BITMAP CONVERSION TO ROWIDS     |                  |     56 |        |       |            |          |    118K|
|* 11 |           BITMAP INDEX SINGLE VALUE      | SALES_CHANNEL_BIX|     56 |        |       |            |          |     20 |
|  12 |        TABLE ACCESS BY LOCAL INDEX ROWID | SALES            |    118K|    229K|  2916K|  2133   (1)| 00:00:01 |    118K|
|* 13 |     INDEX UNIQUE SCAN                    | CUSTOMERS_PK     |   3614 |      1 |       |     0   (0)|          |   3614 |
|* 14 |    TABLE ACCESS BY INDEX ROWID           | CUSTOMERS        |   3614 |      1 |    21 |     1   (0)| 00:00:01 |     26 |
```

Problem 2 – Adding the hint? - Patch

From the plan OUTLINE of the hinted SQL pull the relevant parts (abbreviated outline)

```
Outline Data
-------------
 /*+
    BEGIN_OUTLINE_DATA
    INDEX(@"SEL$09FF82E3" "C"@"SEL$1" ("CUSTOMERS"."CUST_ID"))
    BITMAP_TREE(@"SEL$C1C92257" "S"@"SEL$1" AND(("SALES"."CHANNEL_ID")))
    END_OUTLINE_DATA
 */
```

```
set echo on
DECLARE
  pname VARCHAR2(4000);
BEGIN
  pname := sys.dbms_sqldiag.create_sql_patch(
                sql_id    => 'fc25kvvvb1aar',
                hint_text => 'INDEX(@"SEL$F5BB74E1" "C"@"SEL$1" ("CUSTOMERS"."CUST_ID")) '||
                             'BITMAP_TREE(@"SEL$F5BB74E1" "S"@"SEL$1" AND(("SALES"."CHANNEL_ID")))',
name     => 'sql_fc25kvvvb1aar_index');
END;
/
```

Original SQL_ID

Update ”view” name use name from original query

Problem 2 – Option B – copy whole plan

Script at: https://github.com/ggordham/ora-presentations/tree/main/UTOUG-sql-hint

```
@mk-patch.sql fc25kvvvb1aar 009ym2v4nkdjp
-- Note: <original_sql> <hinted_sql> - both must be in cursor cache
```

```
-- run the following block to create the patch
DECLARE
  pname VARCHAR2(4000);
BEGIN
  pname := sys.dbms_sqldiag.create_sql_patch(
                   sql_id    => 'fc25kvvvb1aar',
                   hint_text =>'IGNORE_OPTIM_EMBEDDED_HINTS '||
                               'OPTIMIZER_FEATURES_ENABLE(''19.1.0'') '||
                               'DB_VERSION(''19.1.0'') '||
                               'ALL_ROWS '||
                               'OUTLINE_LEAF(@"SEL$C1C92257") '||
                               'OUTLINE_LEAF(@"SEL$09FF82E3") '||
                               'PLACE_GROUP_BY(@"SEL$F5BB74E1" ( "S"@"SEL$1"   "CH"@"SEL$2" ) 10) '||
                               'OUTLINE(@"SEL$67910E1A") '||
                               'OUTLINE(@"SEL$F5BB74E1") '||
                               'MERGE(@"SEL$2" >"SEL$1") '||
                               'OUTLINE(@"SEL$1") '||
                               'OUTLINE(@"SEL$2") '||
                               'NO_ACCESS(@"SEL$09FF82E3" "VW_GBC_10"@"SEL$67910E1A") '||
                               'INDEX(@"SEL$09FF82E3" "C"@"SEL$1" ("CUSTOMERS"."CUST_ID")) '||
                               'LEADING(@"SEL$09FF82E3" "VW_GBC_10"@"SEL$67910E1A" "C"@"SEL$1") '||
                               'USE_NL(@"SEL$09FF82E3" "C"@"SEL$1") '||
                               'NLJ_BATCHING(@"SEL$09FF82E3" "C"@"SEL$1") '||
                               'USE_HASH_AGGREGATION(@"SEL$09FF82E3") '||
                               'FULL(@"SEL$C1C92257" "CH"@"SEL$2") '||
                               'BITMAP_TREE(@"SEL$C1C92257" "S"@"SEL$1" AND(("SALES"."CHANNEL_ID"))) '||
                               'LEADING(@"SEL$C1C92257" "CH"@"SEL$2" "S"@"SEL$1") '||
                               'USE_NL(@"SEL$C1C92257" "S"@"SEL$1") '||
                               'NLJ_BATCHING(@"SEL$C1C92257" "S"@"SEL$1") '||
                               'USE_HASH_AGGREGATION(@"SEL$C1C92257") ',
                   name      =>'sql_fc25kvvvb1aar_patch');
END;
/
-- end of patch block
```

Post patch – Explain plan

```
SQL_ID  fc25kvvvb1aar, child number 1
-------------------------------------
SELECT c.cust_state_province, sum(s.amount_sold)   FROM sh.sales s,
sh.customers c,     (SELECT ch.channel_id FROM sh.channels ch
WHERE ch.channel_class_id = 13) chan13   WHERE c.cust_id = s.cust_id
 AND s.channel_id = chan13.channel_id     AND c.cust_city_id IN
(52114,52292,51738,51508)   GROUP BY c.cust_state_province

Hint Report (identified by operation id / Query Block Name / Object Alias):
Total hints for statement: 2
---------------------------------------------------------------------------

  12 -  SEL$C1C92257 / S@SEL$1
          -  BITMAP_TREE(@"SEL$F5BB74E1" "S"@"SEL$1" AND(("SALES"."CHANNEL_ID")))

  13 -  SEL$09FF82E3 / C@SEL$1
          -  INDEX(@"SEL$F5BB74E1" "C"@"SEL$1" ("CUSTOMERS"."CUST_ID"))

Note
-----
   - SQL patch "sql_fc25kvvvb1aar_patch" used for this statement
   - this is an adaptive plan
```

Q&A

References

- SQL Transpiler – SQL Tuning Guide: https://docs.oracle.com/en/database/oracle/oracle-database/23/tgsql/introduction-to-sql-tuning.html#GUID-C25CC846-7515-4527-8345-DAE2896EDAC8
- SQL Plan Management – SQL Tuning Guide: https://docs.oracle.com/en/database/oracle/oracle-database/23/tgsql/overview-of-sql-plan-management.html#GUID-BC65F4D4-1CEE-4014-8188-C849A0D9251A
- Monitoring SQL Statements with Real-Time SQL Monitoring (Doc ID 1380492.1)
- Enhanced Query monitoring: https://docs.oracle.com/en/database/oracle/oracle-database/23/nfcoa/oltp_manageability_perf.html#GUID-91196-1
- Automatic SQL Tuning Sets (ASTS) 19c RU 19.7 Onwards (Doc ID 2686869.1)
- The Automatic SQL Tuning Set https://docs.oracle.com/en/database/oracle/oracle-database/19/tgsql/sql-tuning-advisor.html#GUID-8694BFDF-E2E6-43EF-8291-B03000BFA250
- Bug 34534504 - PERF_DIAG: Need a dbms_spm API to Create Plan Baseline to Get Good Execution Plan for a SQL_ID (Doc ID 34534504.8)
- What is automatic SQL plan management and why should you care? – Nigel Bayliss https://blogs.oracle.com/optimizer/post/what-is-automatic-sql-plan-management-and-why-should-you-care

References – Cont.

- How to Use SQL Plan Management (SPM) - Plan Stability Worked Example (Doc ID 456518.1)
- SQL Plan Baseline Not always created (Doc ID 788853.1)
- How To Use DBMS_XPLAN.COMPARE_PLANS & Hint Usage Report To Troubleshoot Plan Reproducibility Issues (SPM Baselines & SQL Profiles) From 19c (Doc ID 2736319.1)
- FAQ: SQL Plan Management (SPM) Frequently Asked Questions (Doc ID 1524658.1)
- Transporting SQL PLAN Baselines from One Database to Another. (Doc ID 880485.1)
- Loading Hinted Execution Plans into SQL Plan Baseline. (Doc ID 787692.1)
- Master Note: Plan Stability Features (Including SQL Plan Management (SPM)) (Doc ID 1359841.1)
- How to Drop Plans from the SQL Plan Management (SPM) Repository (Doc ID 790039.1)
- How to Load SQL Plans into SQL Plan Management (SPM) from the Automatic Workload Repository (AWR) (Doc ID 789888.1)
- SRDC - How to Collect Standard Information for an Issue Where a SQL Plan Management (SPM) Baseline is Not Used (Doc ID 1644732.1)
- Using the Automatic SQL Tuning Set in Oracle Database 19c – Nigel Bayliss https://blogs.oracle.com/optimizer/post/using-asts-in-19c

References – Cont.

- Real-Time SQL Monitoring: a MUST for SQL Tuning – Ulrike Schwinn https://blogs.oracle.com/coretec/post/oracle-database-real-time-sql-monitoring-one-of-the-most-important-tools
- License Change for Automatic SQL Plan Management (yes, you can do more) – Nigel Bayliss https://blogs.oracle.com/optimizer/post/license-change-for-auto-spm
- What is automatic SQL plan management and why should you care? – Nigel Bayliss https://blogs.oracle.com/optimizer/post/what-is-automatic-sql-plan-management-and-why-should-you-care
- SQL Tuning Health-Check Script (SQLHC) (Doc ID 1366133.1)
- Release Schedule of Current Database Releases (Doc ID 742060.1)
- Hint Usage Reporting - 19c New Feature (Doc ID 2735444.1)

Follow Us Online!


Facebook.com/ViscosityNa


LinkedIn.com/company/Viscosity-North-America


@ViscosityNA


Viscosity North America


Viscosity_NA