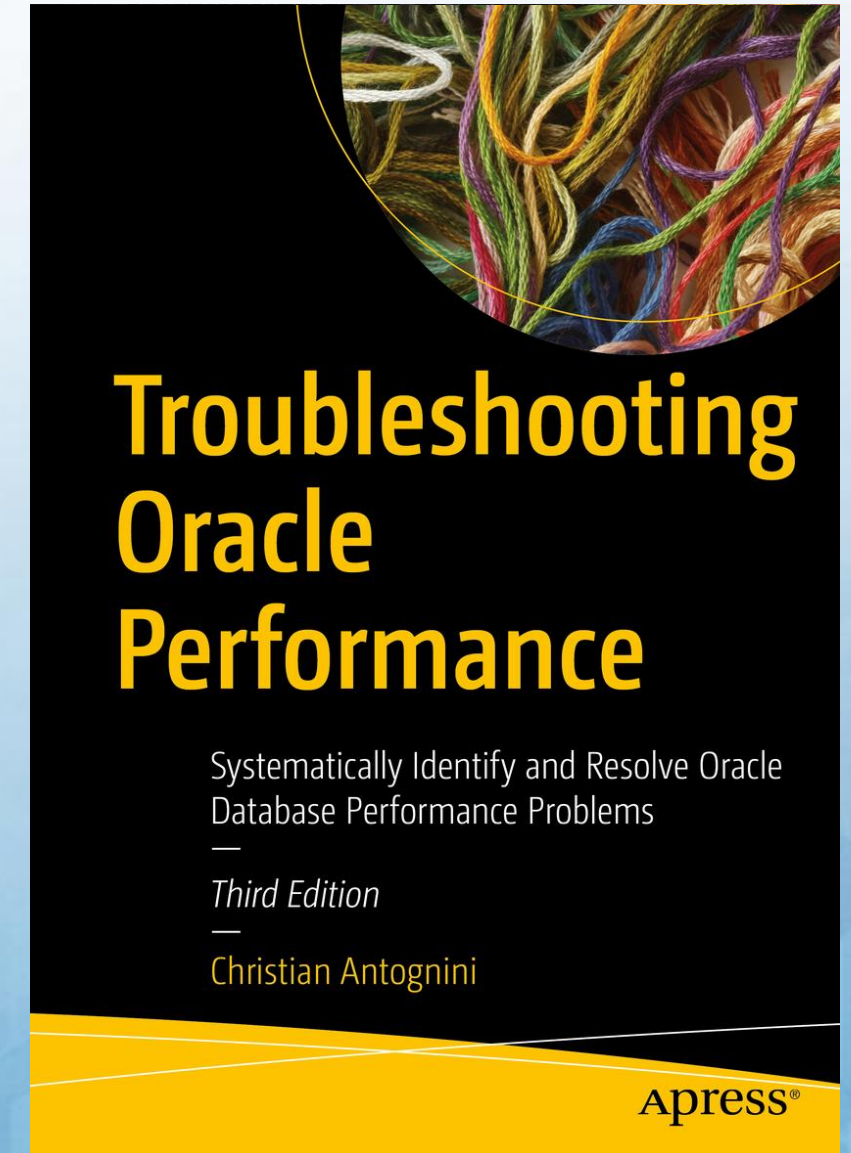


What's New in the Third Edition of *Troubleshooting Oracle Performance?*

Chris Antognini
SOUG Day 2026
Zürich, 2 October 2026

The Book

- Publisher: Apress
- Publication date: 21 December 2026
- Manuscript length: 890 pages



Differences between the 2nd and the 3rd Edition

- Add content about all the versions that have been released since the publication of the second edition (12.1.0.2 – 23.26.3).
- Remove content specific to versions up to and including 12.1.0.1.
- Add information about Python in the parts that cover specific programming languages.
- Reorganize part of the material for better readability.
- Fix errata and generally enhance the text.

Signatures

- They are used to recognize SQL statements that are equivalent from a functional point-of-view.
- They are generated by applying a hash function to a normalized version of their text.
 - DBMS_SQLTUNE.SQLTEXT_TO_SIGNATURE

```
SQL> SELECT sql_text, exact_matching_signature, force_matching_signature
       FROM v$sqlarea
       WHERE regexp_like(sql_text, 'SELECT.+FROM\s+t\s+', 'i');
```

SQL_TEXT	EXACT_MATCHING_SIGNATURE	FORCE_MATCHING_SIGNATURE
SELECT * FROM t WHERE n = 42	3093989503526381832	12455727565333211617
select * from t where n = 42	3093989503526381832	12455727565333211617
SELECT * FROM t WHERE n=42	3093989503526381832	12455727565333211617
SELECT * FROM t WHERE n = 042	2269657263435952095	12455727565333211617

Signatures and SQL Baselines/Profiles/Patches

- SYS.SQLLOG\$ contains the signature of SQL statements that were executed in the past.
- The name (SQL_HANDLE) of a SQL plan baseline is the concatenation of a prefix (SQL_) and the hexadecimal value of the exact-matching signature.
- The query optimizer uses the SQL statement's signatures to match SQL statements to SQL profiles.
- The query optimizer uses the SQL statement's exact-matching signature to match SQL statements to SQL patches.

Signatures and Bind Variables

- Different bind variable names lead to different signatures.

SQL_TEXT	EXACT_MATCHING_SIGNATURE	FORCE_MATCHING_SIGNATURE
SELECT * FROM t WHERE n = :b1	9259967161407683323	9259967161407683323
SELECT * FROM t WHERE n = :b2	7994124921606720908	7994124921606720908

- Up to and including version 19.11, because of bug 32238935, a mix of both literals and bind variables leads to a wrong computation of the force-matching signature.

SQL_TEXT	EXACT_MATCHING_SIGNATURE	FORCE_MATCHING_SIGNATURE
SELECT * FROM t WHERE n = :b1 OR n = 42	6145600014625508115	6145600014625508115
SELECT * FROM t WHERE n = :b1 OR n = 43	3829258159527087686	3829258159527087686

The correct value as of 19.12 is
8892087917950646497

Group-by Pushdown to Union All (23+)

- The purpose of this cost-based query transformation is to push GROUP BY clauses inside views or inline views that contain UNION ALL operators.

```
SELECT t1.n2, v.n2, sum(t1.n3)
FROM t1, (SELECT n1, n2
          FROM t2
          UNION ALL
          SELECT n1, n2
          FROM t3) v
WHERE t1.n1 = v.n1
GROUP BY t1.n2, v.n2
```

```
SELECT t1.n2, v.n2, sum(t1.n3 * v.cnt)
FROM t1, (SELECT n1, n2, count(*) AS cnt
          FROM t2
          GROUP BY n1, n2
          UNION ALL
          SELECT n1, n2, count(*) AS cnt
          FROM t3
          GROUP BY n1, n2) v
WHERE t1.n1 = v.n1
GROUP BY t1.n2, v.n2
```

Subsumption of Views and Subqueries (23+)

- The purpose of this heuristic-based query transformation is to replace multiple query blocks that are created due to views, inline views, or subqueries accessing the same tables with one query block.

```
SELECT
  (SELECT sum(t2.n1)
   FROM t1 JOIN t2 ON t1.id = t2.t1_id
   WHERE t1.n1 = 6) AS sum,
  (SELECT count(*)
   FROM t1 JOIN t2 ON t1.id = t2.t1_id
   WHERE t2.n1 = 19) AS count
FROM dual
```

```
SELECT vw_svs.sum, vw_svs.count
FROM (
  SELECT sum(CASE WHEN t1.n1 = 6
                  THEN t2.n1
                  ELSE NULL END) sum,
         count(CASE WHEN t2.n1 = 19
                     THEN 1
                     ELSE NULL END) count
        FROM t1, t2
        WHERE t1.id = t2.t1_id
      ) vw_svs, dual
```

DBMS_STATS – Gather Stale

- `OPTIONS = GATHER STALE` can be specified when gathering object statistics at the database or schema level.
 - Only objects with stale statistics are processed.
 - As of version 21, objects for which the column usage history has changed can also be processed.

DBMS_STATS – Reason to Gather (23.7+)

- To find out more about the most recent processing of a specific object, the DBA_AUTO_STAT_OBJ_GATHER_DETAILS view is the source of information you need.
- You can find out when it took place and why, whether it failed and the reason why.
- The reason why the statistics were gathered, provided by the REASON_TO_GATHER column, isn't available elsewhere.
 - E.g.: GLOBAL STATS NEEDED, MISSING HISTOGRAM,

Dynamic Statistics and PL/SQL Functions (23.8+)

- By default, sampling queries only call PL/SQL functions that are referenced in predicates.
 - Sampling only takes place for the left SQL statement:

```
SELECT * FROM t WHERE id > one()
```

```
SELECT * FROM table(range(10))
```

- DYNAMIC_STATS preference
 - CHOOSE: this is the default behavior.
 - ON: sampling queries can call any PL/SQL function.
 - OFF: dynamic statistics cannot call PL/SQL functions.
- The parameter PLSQL_FUNCTION_DYNAMIC_STATS also exists.
 - Accepted values: PREFERENCE, CHOOSE, ON and OFF
 - It can only be set at the session level.

Add Verified SQL Plan Baseline (23.3+)

- You can let the database engine search for the best-known execution plan for a given SQL statement.
 - It searches through the cursor cache, AWR and ASTS.
 - The goal is to collect all known execution plans, verify their performance, and create a SQL plan baseline containing accepted and nonaccepted execution plans.
- To carry out such an operation, the DBMS_SPM package provides the `ADD_VERIFIED_SQL_PLAN_BASELINE` function.
 - It takes one parameter, `SQL_ID`, which identifies the SQL statement.
- The function has been backported to 19c. It's included in 19.22+.

Real-time SQL Plan Management (23+)

- When enabled, the query optimizer can verify, during a hard parse, whether the execution plan is the same as one contained in ASTS.
 - `AUTO_SPM_EVOLVE_TASK = AUTO`
- If the execution plans differ, the query optimizer can automatically create a SQL plan baseline that contains both the current and former execution plans.
- This technique eliminates the need to wait for the next execution of an automatic evolve task to address performance regressions.
- Nevertheless, note that the foreground process carrying out the hard parse must perform much more processing.

IN Conditions with Many Expressions (23.4+)

- Since long lists of expressions frequently lead to performance problems, IN conditions with 64 or more expressions are implemented differently when a full table scan is executed.
- Instead of using disjunctive predicates, a key vector containing the expressions is created.

```
SELECT /*+ full(t) */ * FROM t WHERE id IN (1, 2, 3, ... 62, 63, 64);
```

Id	Operation	Name	Starts	A-Rows
*	SELECT STATEMENT TABLE ACCESS FULL	T	1 1	64 64

```
1 - filter(SYS_OP_KEY_VECTOR_FILTER("ID", :KV0000))
```

Parallel Executions – Downgrades

- Downgrades are explicitly outlined in real-time monitoring reports.
- But the reason for the downgrade isn't decoded.
- To find out what the code means, you can run a query like the following:

```
SQL> SELECT qksxareasons  
FROM x$qksxa_reason  
WHERE qksxareasons LIKE '%DOP%' AND indx = 354;
```

QKSXAREASONS

DOP downgrade due to insufficient number of processes

General

Status Completed

SQL Text `SELECT count(pad) FROM t`

Execution Plan 40

Execution Started 26 Jun 20

Last Refresh Time 26 Jun 20

Execution ID 1677721

User Name CHRIS@

Fetch Calls 1

Degree of Parallelism: 40

Degree of Parallelism Downgrade: 20%

Degree of Parallelism Downgrade Reason: 354

Parallel Execution Servers Requested: 50

Parallel Execution Servers Allocated: 40

What does 354 mean?

Parallel Creation and Validation of Constraints

- Up to and including version 18, the DOP defined at the table level is used for the recursive query.
- The session-level settings don't influence the recursive query.
 - It's irrelevant whether at the session level the parallel queries and DDL statements are enabled, forced, or disabled.
 - The value of `PARALLEL_DEGREE_POLICY` is also irrelevant.
- From version 19 onward, parallel queries must be enabled and the `PARALLEL_DEGREE_POLICY` is honored.

Logical Partition Change Tracking (23+)

- For refreshes and query rewrites, the database engine can track the staleness at the table and partition level.
 - For *partition change tracking* (PCT), a partitioned table is required.
- With logical PCT, the database engine is able to use logical range partitions.

```
CREATE LOGICAL PARTITION TRACKING ON sales
PARTITION BY RANGE (time_id) (
  PARTITION sales_h1_2019 VALUES LESS THAN (to_date('2019-07-01', 'YYYY-MM-DD')),
  ...
  PARTITION sales_maxvalue VALUES LESS THAN (MAXVALUE)
)
```

- Logical PCT enables *fast refreshes with LPCT* and *PCT rewrite*.
 - The MV must contain the partition key.

Q&A