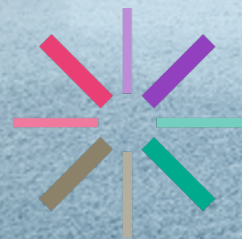


Dani Schnider

# How Intelligent is the SQL Analysis Report in Oracle 23c?



**Callista**

# About me



## Dani Schneider

- Working for Callista
- Oracle ACE Director
- Member of Symposium 42
- Hobby: Craft Beer Brewing



# Live Demo



intro\_example.sql

# Motivation

```
SQL> EXPLAIN PLAN FOR SELECT * FROM addresses WHERE UPPER(ctr_code) = 'GB';
```

Explained.

```
SQL> SELECT * FROM TABLE(dbms_xplan.display);
```

```
-----  
| Id | Operation          | Name          | Rows | Bytes | Cost (%CPU)| Time     |  
-----  
|  0 | SELECT STATEMENT   |               |    239 | 13384 |    61  (0)| 00:00:01 |  
|*  1 |  TABLE ACCESS FULL| ADDRESSES     |    239 | 13384 |    61  (0)| 00:00:01 |  
-----
```

Predicate Information (identified by operation id):

```
-----  
1 - filter(UPPER("CTR_CODE")='GB')
```

SQL Analysis Report (identified by operation id/Query Block Name/Object Alias):

```
-----  
1 - SEL$1 / "ADDRESSES"@SEL$1  
   - The following columns have predicates which preclude their  
     use as keys in index range scan. Consider rewriting the  
     predicates.  
     "CTR_CODE"
```



# Oracle Optimizer

Follow:

## New SQL Analysis Report in Oracle Database 23c Free!

April 6, 2023 | 5 minute read



[Nigel Bayliss](#)  
Product Manager

Now that [Oracle Database 23c Free is available](#), it's time to take a look at a new feature we've built into the optimizer to help you avoid some mistakes when building queries.

It's called the SQL Analysis Report and appears in a new section at the end of a SQL execution plan.

Here are a few simple examples to get you going:

[Copy code snippet](#)

```
select sum(t1.c1),sum(t2.c1)
from   t1,t2;

SELECT * FROM table(DBMS_XPLAN.DISPLAY_CURSOR(FORMAT=>'typical'));
```

<https://blogs.oracle.com/optimizer/post/sql-analysis-report-in-23c-free>

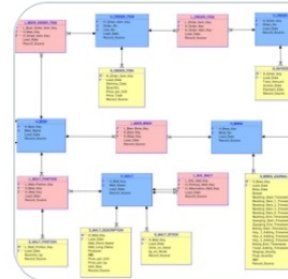


**Dani Schnider** @dani\_schnider · Apr 24



Another nice surprise in [@OracleDatabase #23c](#):  
SQL Performance Tips in Execution Plans

I wasn't aware of this enhancement, but I think it can be useful to avoid typical SQL mistakes.



[danischnider.wordpress.com](https://danischnider.wordpress.com)

SQL Performance Tips in Execution Plans of Oracl...

Oracle 23c is like Christmas for me. There are many surprises to be found. This weekend I discovered ...



2



14



31



2,762



**Jonathan Lewis**



@JLOracle

It would be interesting to see if the optimizer made the same comment when both

- a) there was a suitable function-based index and
- b) for other reasons (e.g. clustering\_factor / selectivity) the FBI was not used

In other words, is it "intelligent" or an auto response to func(col)

9:23 PM · Apr 25, 2023 · **624** Views

7 Likes

# Live Demo



predicates.sql



# Detail Analysis: Predicates

- The following columns have predicates which preclude their use as keys in index range scan. Consider rewriting the predicates.



Analysis report is displayed...

- When index on column is available
- When predicate on column is used in WHERE condition
- When neither function-based index, virtual column nor extended statistics are available for the predicate

Note:

- Optimizer costs (based on cardinality, clustering factor, etc.) are ignored

# Supported Use Cases

- Recognition of Cartesian Products

- The query block has 1 cartesian product which may be expensive. Consider adding join conditions or removing the disconnected tables or views.

- Replace UNION by UNION ALL

- The query block contains UNION which may be expensive. Consider using UNION ALL if duplicates are allowed or uniqueness is guaranteed.

- Predicates on Indexed Columns

- The following columns have predicates which preclude their use as keys in index range scan. Consider rewriting the predicates.

Worksheet Query Builder

```

1 EXPLAIN PLAN FOR
2 SELECT * FROM emp
3 WHERE deptno = 10
4 UNION
5 SELECT * FROM emp
6 WHERE deptno = 20
7 UNION
8 SELECT * FROM emp
9 WHERE deptno = 30;
10
11

```

Query Result x

SQL | All Rows Fetched: 30 in 0,045 seconds

PLAN\_TABLE\_OUTPUT

```

1 Plan hash value: 3091781740
2
3 -----
4 | Id | Operation | Name | Rows | Bytes |
5 -----
6 | 0 | SELECT STATEMENT | | 14 | 546 |
7 | 1 | HASH UNIQUE | | 14 | 546 |
8 | 2 | UNION-ALL | | 14 | 546 |
9 | 3 | TABLE ACCESS BY INDEX ROWID BATCHED | EMP | 3 | 117 |
10 |* 4 | INDEX RANGE SCAN | EMP_DEPTNO_IDX | 3 | |
11 | 5 | TABLE ACCESS BY INDEX ROWID BATCHED | EMP | 5 | 195 |
12 |* 6 | INDEX RANGE SCAN | EMP_DEPTNO_IDX | 5 | |
13 | 7 | TABLE ACCESS BY INDEX ROWID BATCHED | EMP | 6 | 234 |
14 |* 8 | INDEX RANGE SCAN | EMP_DEPTNO_IDX | 6 | |
15 -----
16
17 Predicate Information (identified by operation id):
18 -----
19
20 4 - access("DEPTNO">=10)
21 6 - access("DEPTNO">=20)
22 8 - access("DEPTNO">=30)
23
24 SQL Analysis Report (identified by operation id/Query Block Name/Object Alias):
25 -----
26
27 1 - SET$1
28 - The query block contains UNION which may be expensive.
29 Consider using UNION ALL if duplicates are allowed or
30 uniqueness is guaranteed.

```

Worksheet Query Builder

```

1 EXPLAIN PLAN FOR
2 SELECT * FROM emp
3 WHERE deptno = 10
4 UNION ALL
5 SELECT * FROM emp
6 WHERE deptno = 20
7 UNION ALL
8 SELECT * FROM emp
9 WHERE deptno = 30;
10
11 SELECT * FROM TABLE(dbms_xplan.display);

```

Query Result x

SQL | All Rows Fetched: 21 in 0,036 seconds

PLAN\_TABLE\_OUTPUT

```

1 Plan hash value: 3221407431
2
3 -----
4 | Id | Operation | Name | Rows | Bytes |
5 -----
6 | 0 | SELECT STATEMENT | | 14 | 546 |
7 | 1 | UNION-ALL | | 14 | 546 |
8 | 2 | TABLE ACCESS BY INDEX ROWID BATCHED | EMP | 3 | 117 |
9 |* 3 | INDEX RANGE SCAN | EMP_DEPTNO_IDX | 3 | |
10 | 4 | TABLE ACCESS BY INDEX ROWID BATCHED | EMP | 5 | 195 |
11 |* 5 | INDEX RANGE SCAN | EMP_DEPTNO_IDX | 5 | |
12 | 6 | TABLE ACCESS BY INDEX ROWID BATCHED | EMP | 6 | 234 |
13 |* 7 | INDEX RANGE SCAN | EMP_DEPTNO_IDX | 6 | |
14 -----
15
16 Predicate Information (identified by operation id):
17 -----
18
19 3 - access("DEPTNO">=10)
20 5 - access("DEPTNO">=20)
21 7 - access("DEPTNO">=30)

```

Worksheet Query Builder

```

1 EXPLAIN PLAN FOR
2 SELECT * FROM emp
3 WHERE deptno = 10
4 UNION
5 SELECT * FROM emp
6 WHERE deptno = 20
7 UNION
8 SELECT * FROM emp
9 WHERE deptno = 30;
10
11 SELECT * FROM TABLE (dbms_xplan.display);

```

Query Result x

SQL | All Rows Fetched: 30 in 0,045 seconds

PLAN\_TABLE\_OUTPUT

```

1 Plan hash value: 3091781740
2
3 -----
4 | Id | Operation | Name | Rows | Bytes |
5 -----
6 | 0 | SELECT STATEMENT | | 14 | 546 |
7 | 1 | HASH UNIQUE | | 14 | 546 |
8 | 2 | UNION-ALL | | 14 | 546 |
9 | 3 | TABLE ACCESS BY INDEX ROWID BATCHED | EMP | 3 | 117 |
10 |* 4 | INDEX RANGE SCAN | EMP_DEPTNO_IDX | 3 | |
11 | 5 | TABLE ACCESS BY INDEX ROWID BATCHED | EMP | 5 | 195 |
12 |* 6 | INDEX RANGE SCAN | EMP_DEPTNO_IDX | 5 | |
13 | 7 | TABLE ACCESS BY INDEX ROWID BATCHED | EMP | 6 | 234 |
14 |* 8 | INDEX RANGE SCAN | EMP_DEPTNO_IDX | 6 | |
15 -----
16
17 Predicate Information (identified by operation id):
18 -----
19
20 4 - access("DEPTNO"=10)
21 6 - access("DEPTNO"=20)
22 8 - access("DEPTNO"=30)
23
24 SQL Analysis Report (identified by operation id/Query Block Name/Object Alias):
25 -----
26
27 1 - SET$1
28 - The query block contains UNION which may be expensive.
29 Consider using UNION ALL if duplicates are allowed or
30 uniqueness is guaranteed.

```

DA Dear ChatGPT, can you improve a SQL query for me?

Of course, I'd be happy to help you improve your SQL query. Please provide me with the details of your query and let me know what specific improvements or optimizations you're looking for. The more information you can provide about your database schema and the purpose of the query, the better I can assist you.

DA

```

SELECT * FROM emp
WHERE deptno = 10
UNION
SELECT * FROM emp
WHERE deptno = 20
UNION
SELECT * FROM emp
WHERE deptno = 30

```

Your SQL query retrieves all rows from the `emp` table for three different department numbers (10, 20, and 30) and combines them using UNION. This can be optimized to a single query using the IN operator or a WHERE clause with OR conditions to reduce the number of queries against the `emp` table. Here's an improved version of your query using the WHERE clause with OR conditions:

```

sql
SELECT *
FROM emp
WHERE deptno IN (10, 20, 30);

```

Copy code

This single query achieves the same result as your original query but is more concise and efficient because it doesn't require multiple separate SELECT statements.

# Detail Analysis: UNION

- The query block contains UNION which may be expensive. Consider using UNION ALL if duplicates are allowed or uniqueness is guaranteed.



Analysis report is displayed...

- When at least one UNION is available in query block

Note:

- Semantics of SQL statement are not considered
- No similar message for MINUS/MINUS ALL and INTERSECT/INTERSECT ALL

# What is wrong with this SQL statement?

```
EXPLAIN PLAN FOR
SELECT e.empno, e.ename, d.dname, d.loc
FROM emp e JOIN dept d
ON e.deptno = e.deptno;
```

```
SELECT * FROM TABLE(dbms_xplan.display);
```

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT		56	1736	10 (0)	00:00:01
1	MERGE JOIN CARTESIAN		56	1736	10 (0)	00:00:01
2	TABLE ACCESS FULL	DEPT	4	72	3 (0)	00:00:01
3	BUFFER SORT		14	182	7 (0)	00:00:01
4	TABLE ACCESS FULL	EMP	14	182	2 (0)	00:00:01

SQL Analysis Report (identified by operation id/Query Block Name/Object Alias):

1 - SEL\$58A6D7F6

- The query block has 1 cartesian product which may be expensive. Consider adding join conditions or removing the disconnected tables or views.

# Live Demo



cartesian.sql

format.sql

# Detail Analysis: Cartesian Products

- The query block has 1 cartesian product which may be expensive. Consider adding join conditions or removing the disconnected tables or views.



Analysis report is displayed...

- When join condition between two tables is missing
- When CROSS JOIN is used

Note:

- If one of the tables returns only one row (guaranteed with unique index), no message is displayed



# DBMS\_XPLAN Format Parameter

## dbms\_xplan.display:

```
SELECT * FROM TABLE(dbms_xplan.display(format => '+sql_analysis_report'));
```

⇒ Syntactically correct, but useless (except for format => 'basic')

```
SELECT * FROM TABLE(dbms_xplan.display(format => '-sql_analysis_report'));
```

⇒ Syntactically wrong, does not work

 Fixed in version 23.3



## dbms\_xplan.display\_cursor:

```
SELECT * FROM TABLE(dbms_xplan.display_cursor(format => 'iostats last +sql_analysis_report'));
```

⇒ Parameter is ignored, but no error message

```
SELECT * FROM TABLE(dbms_xplan.display_cursor(format => 'iostats last typical -bytes -cost'));
```

⇒ Workaround to show SQL Analysis Report



# Some more details... about licensing



**Mohamed Hour** · 1st

Oracle Performance and Tuning Consultant chez Generali Assurance  
1w · 🌐

Oracle evolves from version to version. Take a look at how, in 23c, the execution plan Note has been enriched with an optimization tip

SQL\_ID g2a4cbtdfc4fu, child number 0

```
select t1.n1, t1.n3 from t1, lateral (select t2.n1 from t2,
t3 where t3.n1 = t1.n1 and t2.n1 = t1.n1 ) (+) vw_1
```

Plan hash value: 1261264704

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT				61 (100)	
* 1	HASH JOIN OUTER		100	1300	61 (2)	00:00:01
2	TABLE ACCESS FULL	T1	100	600	3 (0)	00:00:01
3	VIEW	VW_DCL_2E85B742	100K	683K	57 (0)	00:00:01
4	MERGE JOIN CARTESIAN		100K	683K	57 (0)	00:00:01
5	INDEX FULL SCAN	T3_PK	100	300	1 (0)	00:00:01
6	BUFFER SORT		1000	4000	56 (0)	00:00:01
7	INDEX FAST FULL SCAN	T2_PK	1000	4000	1 (0)	00:00:01

Predicate Information (identified by operation id):

```
1 - access("ITEM_0"="T1"."N1" AND "N1"="T1"."N1")
```

SQL Analysis Report (identified by operation id/Query Block Name/Object Alias):

```
4 - SEL6B1519A3D
- The query block has 1 cartesian product which may be
expensive. Consider adding join conditions or removing the
disconnected tables or views.
```

Prashant Dixit and 352 others

17 comments · 18 reposts



**Porus Homi H.** · 2nd

Cloud Architect / Database Engineer / Oracle Certified Master

1w ...

Does this require the Tuning pack? The tip sounds familiar to what the Tuning advisor used to produce.

Like · 🌐 1 | Reply · 3 Replies



**Mohamed Hour** **Author**

Oracle Performance and Tuning Consultant chez Generali Ass...

1w ...

I don't think this requires any tuning license. In fact, this execution plan comes from a base where the tuning pack is not active. I might be wrong of course.

```
SQL> show parameter control_management_pack_access
NAME                                 TYPE        VALUE
-----                                 -
control_management_pack_access       string      NONE
SQL> select count(*) from gv$active_session_history;
COUNT(*)
-----
0
SQL> select count(*) from dba_hist_active_sess_history;
COUNT(*)
-----
0
```

Like | Reply



**Porus Homi H.** · 2nd

Cloud Architect / Database Engineer / Oracle Certified Master

1w ...

Ok. But always better to check with your oracle contact in sales.

Like · 🌐 1 | Reply



# Conclusion

- 👍 Useful tips to avoid typical SQL mistakes
- 👉 More or less “hard-coded” rules
- 👉 Only three rules (so far)
- 👉 Confusing logic of format parameter

# Wish List

🙏 More rules (e.g. for DISTINCT)

🙏 More semantic checks (based on cardinality)

(✓) 🙏 Usage of format parameter

(✓) 🙏 Explained in Oracle documentation



▼ Part VI Optimizer Controls

▼ 19 Influencing the Optimizer

19.1 Techniques for Influencing the Optimizer

▶ 19.2 Influencing the Optimizer with Initialization Parameters

▼ 19.3 Influencing the Optimizer with Hints

▶ 19.3.1 About Optimizer Hints

19.3.2 Guidelines for Join Order Hints

▼ 19.3.3 Reporting on Hints

19.3.3.1 Purpose of Hint Usage Reports

19.3.3.2 User Interface for Hint Usage Reports

19.3.3.3 Reporting on Hint Usage: Tutorial

19.3.3.4 Hint Usage Reports: Examples

19.3.4 SQL Analysis Report

▶ 20 Improving Real-World Performance Through Cursor Sharing

▶ Part VII Monitoring and Tracing SQL

▶ Part VIII Automatic SQL Tuning

## 19.3.4 SQL Analysis Report

SQL analysis checks your SQL statement and reports on changes that will improve SQL execution performance and also constructs that hinder the ability of the optimizer to generate optimal SQL execution plans.

### User Interface for SQL Analysis Report

SQL analysis is enabled by default.

The examples below describe the `DBMS_XPLAN` interface that provides the SQL Analysis Report. The report includes the analysis of the SQL statement. Note that there is also an SQL Analysis section in the SQL Monitor Report, which you can view with `DBMS_SQLTUNE.REPORT_SQL_MONITOR`.

### Report Access

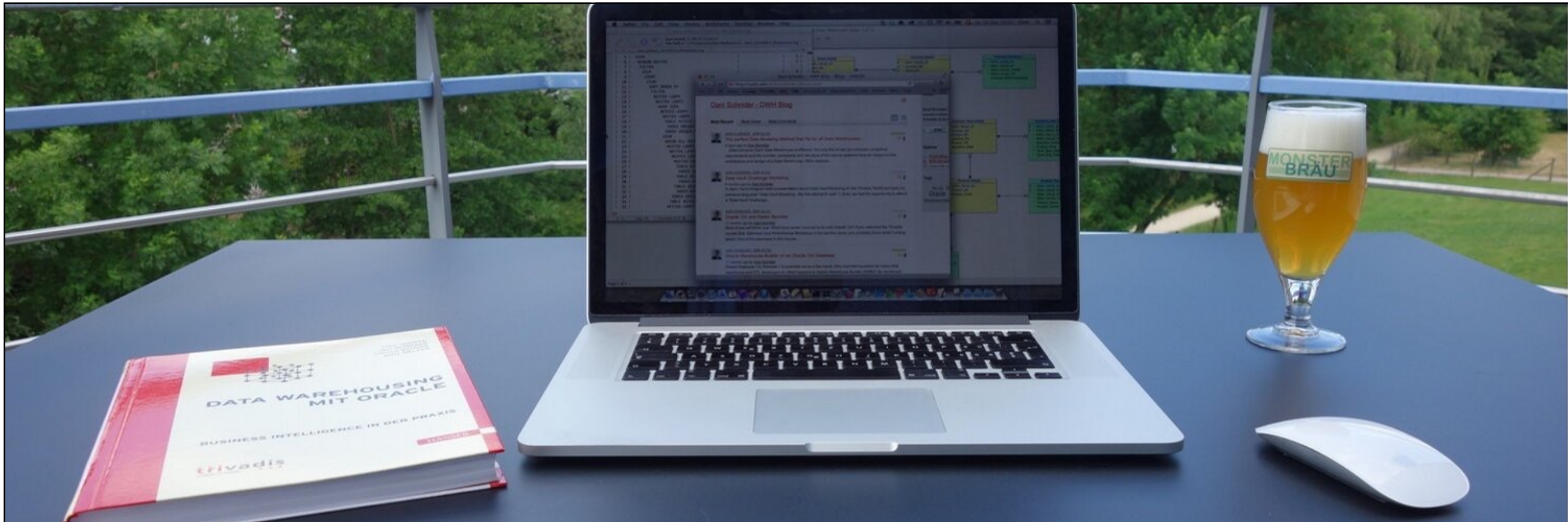
You can generate variants of the report by using the following `DBMS_XPLAN` functions:

- `DISPLAY`
- `DISPLAY_CURSOR`
- `DISPLAY_WORKLOAD_REPOSITORY`
- `DISPLAY_SQL_PLAN_BASELINE`
- `DISPLAY_SQLSET`

In the examples below, the `DBMS_XPLAN.DISPLAY_CURSOR` function is used to show the execution plan of the loaded cursor. The plan includes the SQL analysis report.

### Formatting the SQL Analysis Report

The SQL analysis report has two format options: `TYPICAL` and `BASIC`. The default value is `'TYPICAL'`:



 <https://danischnider.wordpress.com>

 [https://twitter.com/dani\\_schnider](https://twitter.com/dani_schnider)

 <https://www.linkedin.com/in/danischnider/>

# Q&A

A wide-angle landscape photograph of a mountain range. The foreground features a large, calm lake with a small island in the middle. The middle ground shows rolling hills with sparse vegetation. The background is dominated by towering, rugged mountains with significant snow cover and patches of glaciers. The sky is filled with soft, grey clouds.

**DRIVEN BY EXCELLENCE**