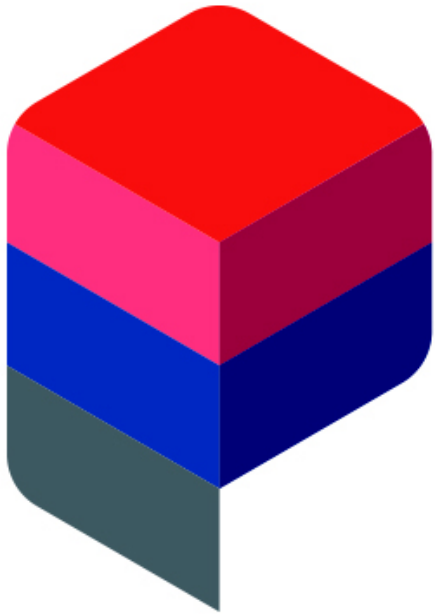




**performing  
databases**

# (Nicht) Hints und Kunz - Oracle Performance Classic (and with AI)

Martin Klier   
Performing Databases GmbH  
Mitterteich / Germany

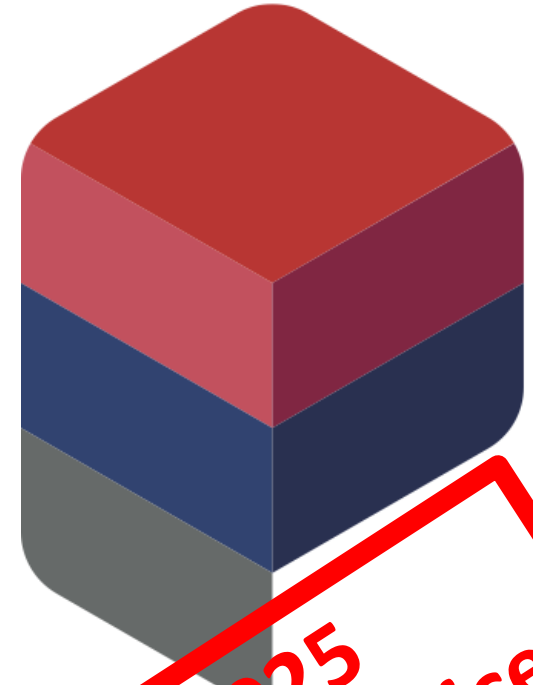
# Speaker

- Martin Klier
- Solution Architect and Database Expert
- My focus:
  - Performance + Tuning
  - Highly available systems
  - Cluster and Replication
- Linux since 1997
- Oracle Database since 2003

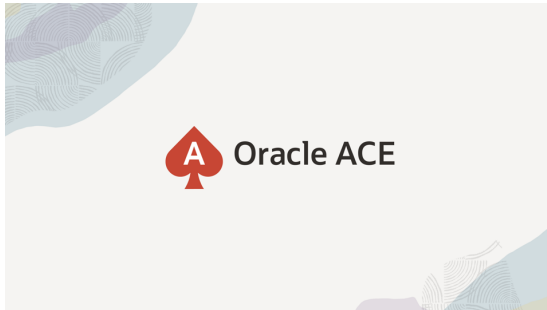


# Performing Databases

- Three Experts for Database technology
  - Concepts and Project Competence
  - Architecture- and System planning
  - Licensing
  - Implementation and Troubleshooting
- Get in touch
  - Performing Databases GmbH  
Wiesauer Strasse 27  
95666 Mitterteich // Germany
  - <http://www.performing-databases.com>
  - Social: @PerformingDB



# ... it's all about Community!



**SYMPOSIUM<sup>L2</sup>**



## ora2know

The German Oracle Database Community

<https://www.ora2know.de>

Artificial Intelligence?

Machine Learning?

Oracle?

# Parsing

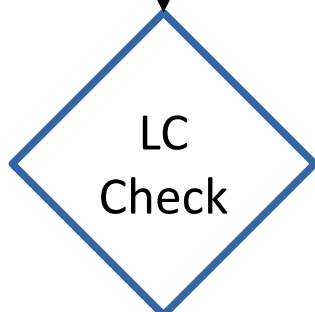
SQL



# = 'select \* from customer  
where id= :a'

Syntax

Semantic



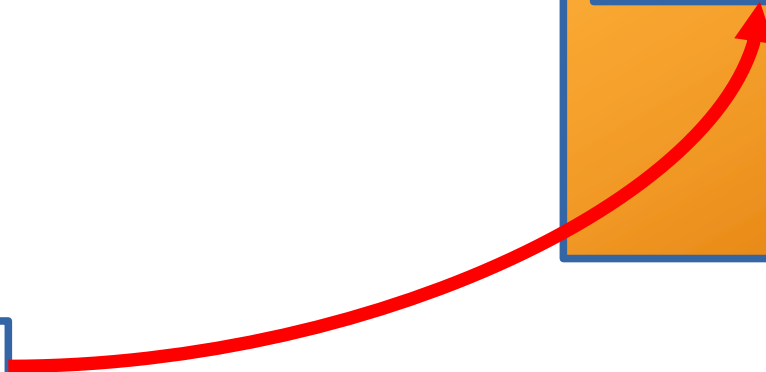
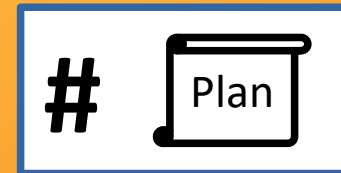
Optimizer



Execution

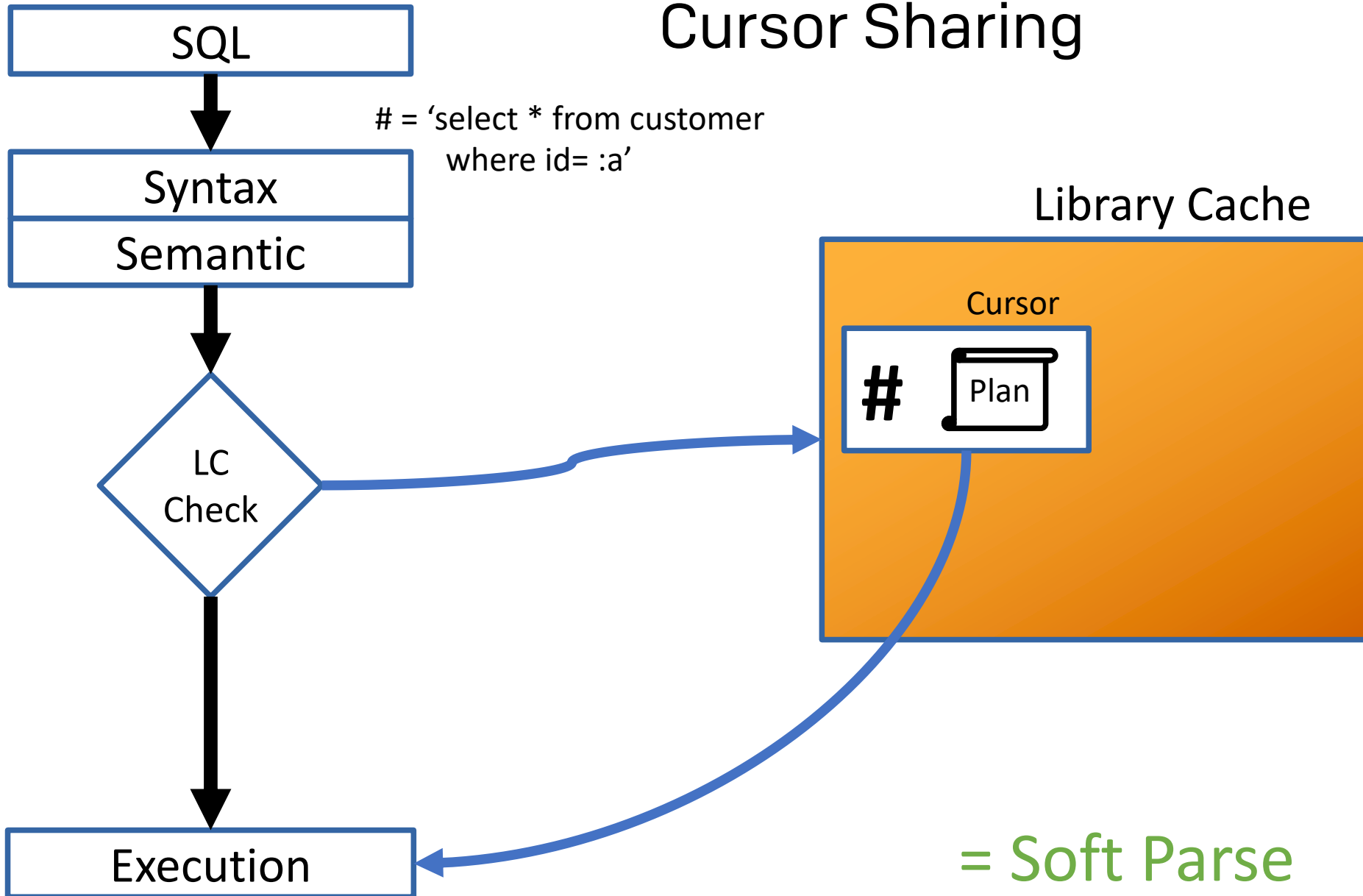
Library Cache

Cursor



= Hard Parse

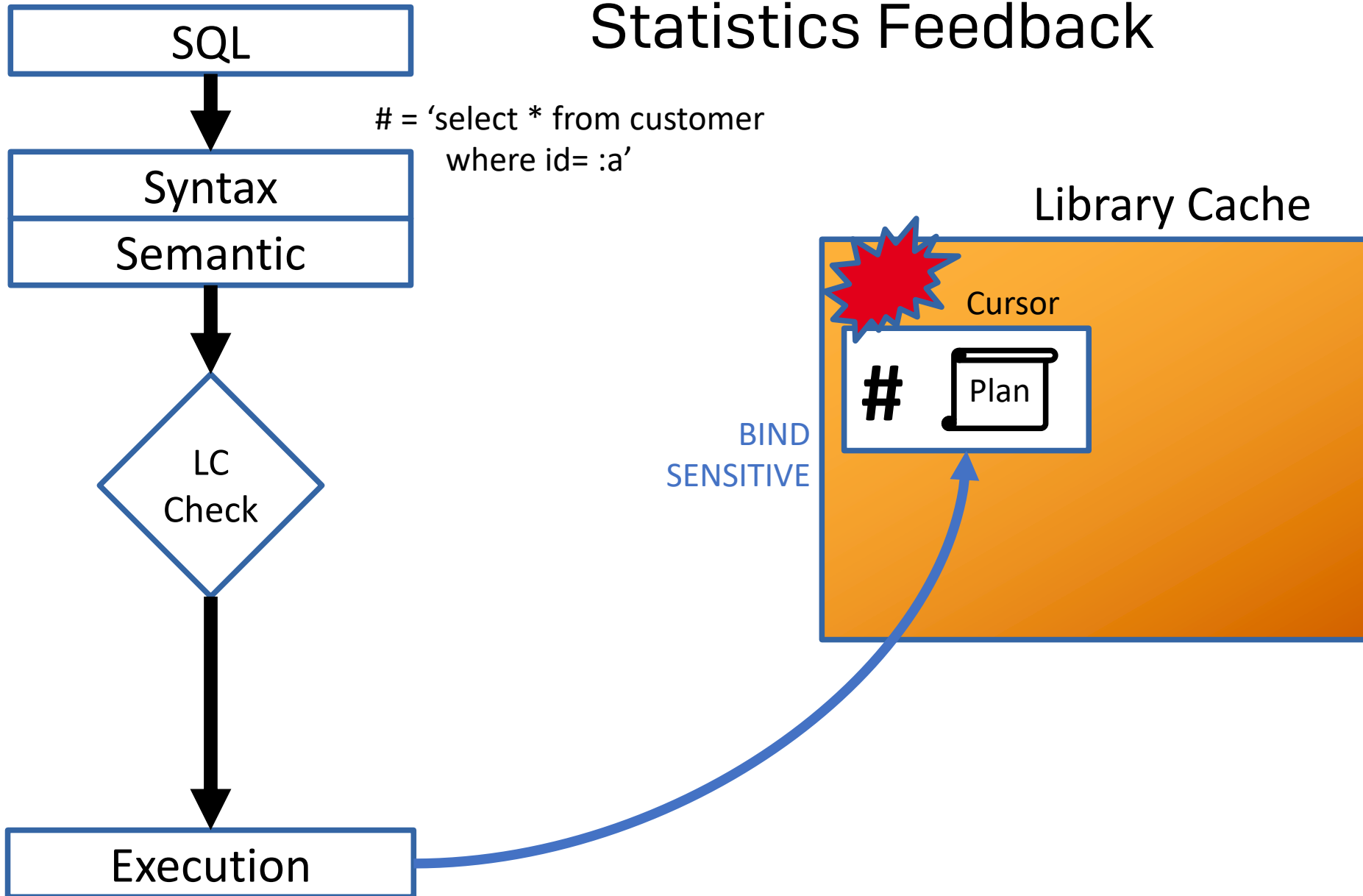
# Cursor Sharing



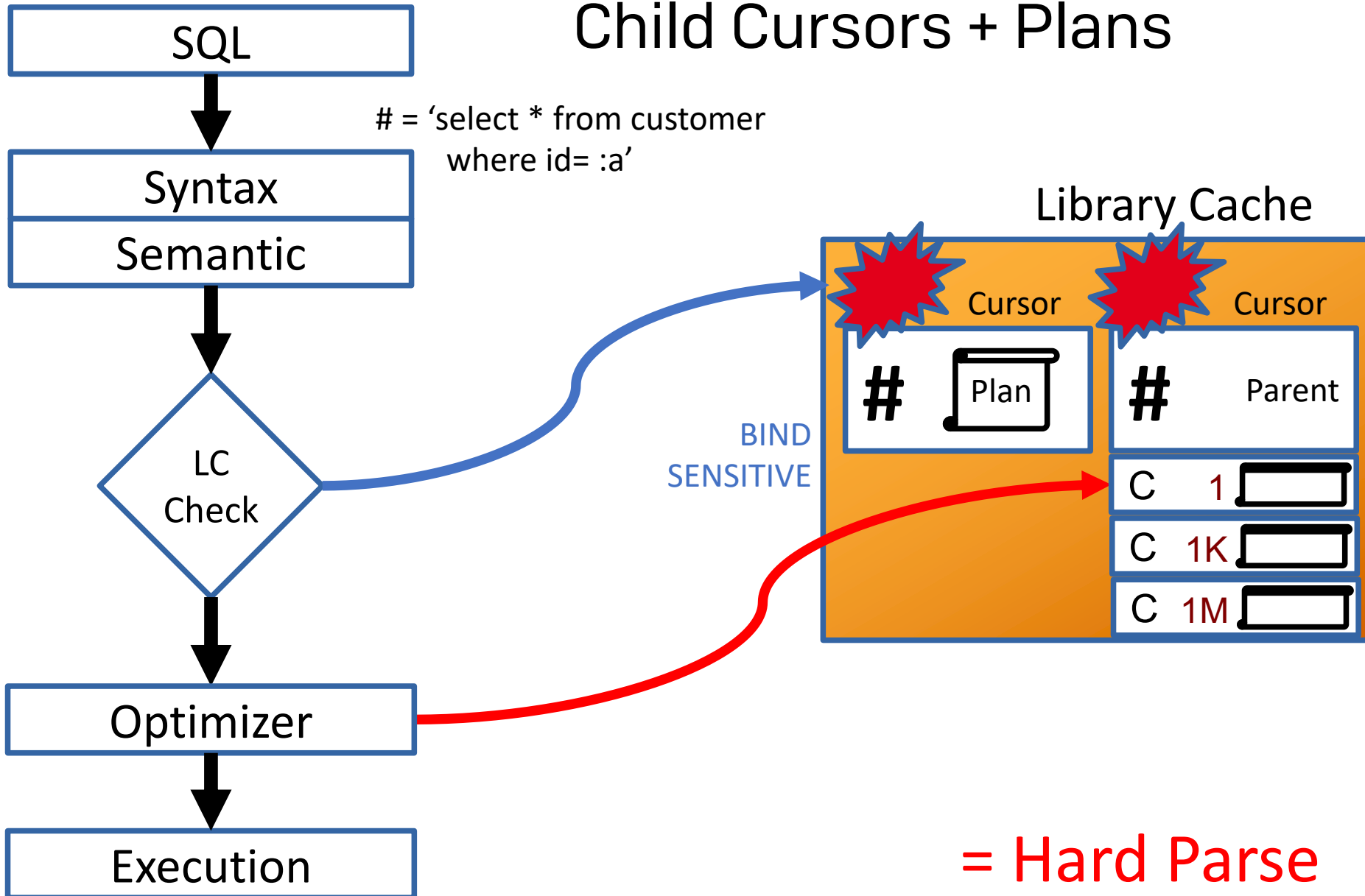
= Soft Parse



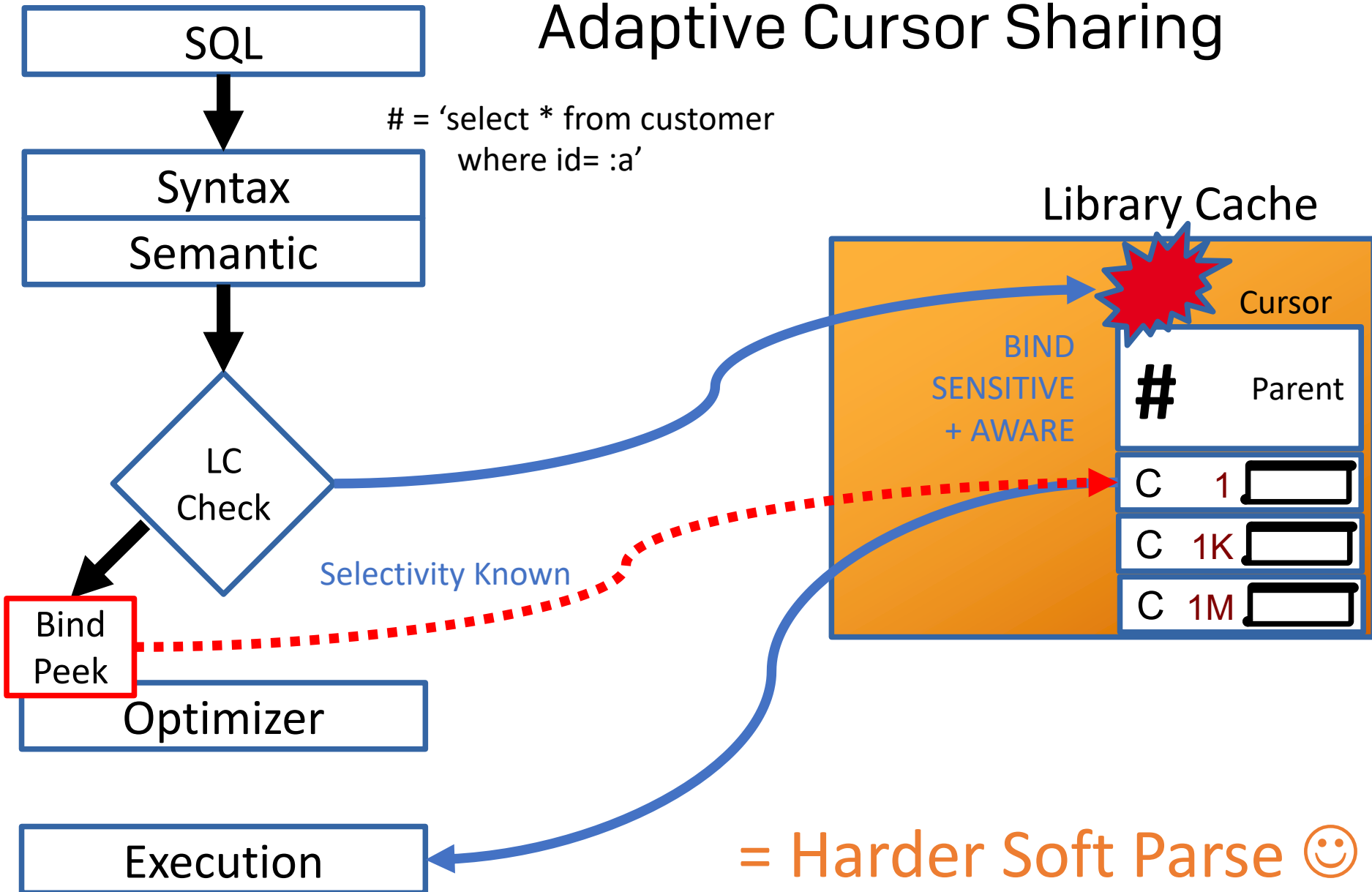
# Statistics Feedback



# Child Cursors + Plans

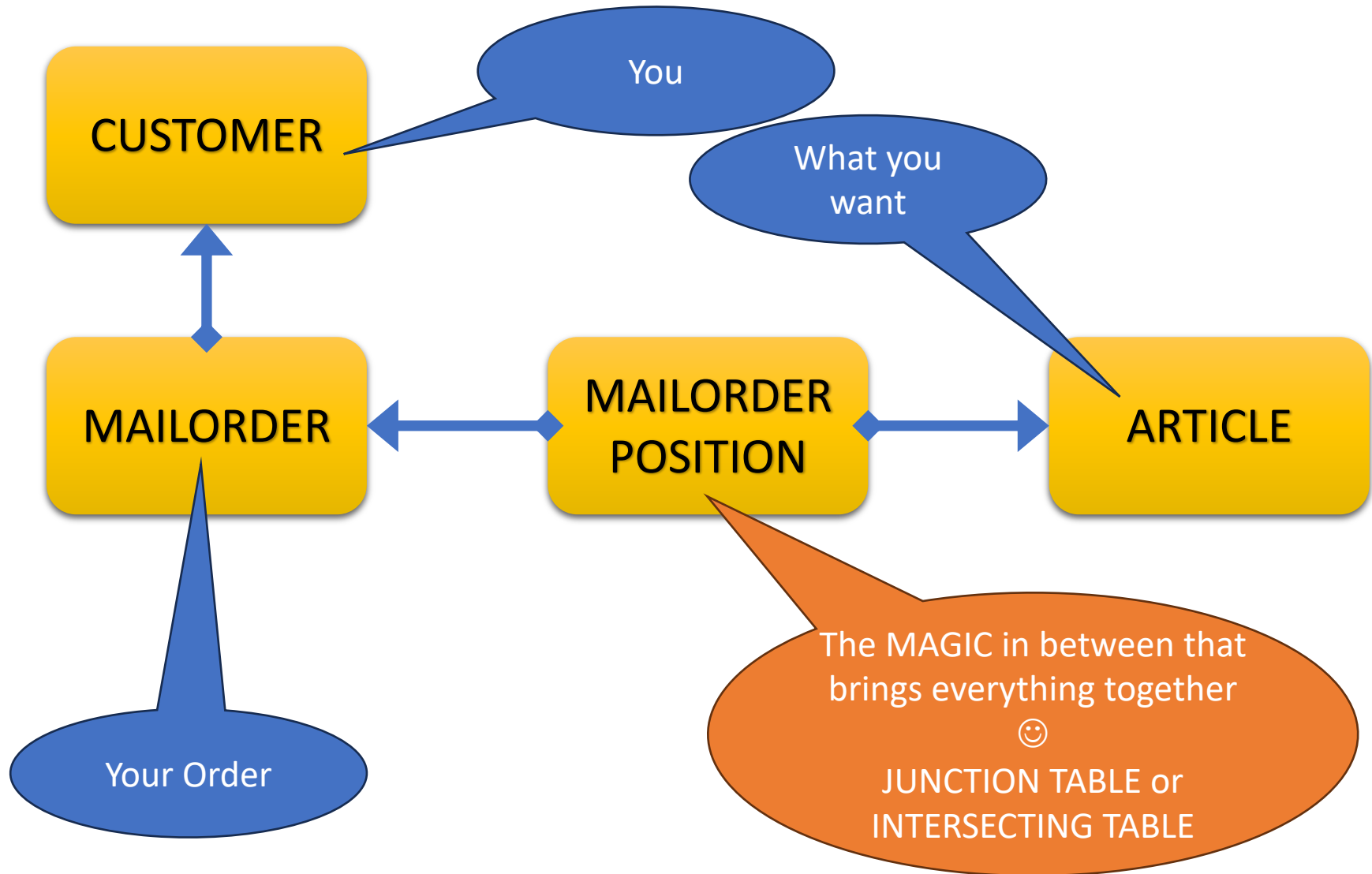


# Adaptive Cursor Sharing



# Lab Data Model

# Lab Data Model



# Lab Data Model

| KLM.CUSTOMER     |              |                    |
|------------------|--------------|--------------------|
| P *              | ID           | NUMBER             |
| *                | CU_NO        | NUMBER             |
| *                | CU_NAME      | VARCHAR2 (20 CHAR) |
| *                | CU_FIRSTNAME | VARCHAR2 (20 CHAR) |
| *                | CU_STREET    | VARCHAR2 (20 CHAR) |
|                  | CU_CITY      | VARCHAR2 (20 CHAR) |
|                  | CU_POSTCODE  | VARCHAR2 (20 CHAR) |
|                  | CU_COUNTRY   | VARCHAR2 (20 CHAR) |
| CUSTOMER_PK (ID) |              |                    |

54 customers

| KLM.MAILORDER                          |                |                     |
|----------------------------------------|----------------|---------------------|
| P *                                    | ID             | NUMBER              |
| F *                                    | MO_CUSTOMER_ID | NUMBER              |
| *                                      | MO_NO          | NUMBER              |
| *                                      | MO_NAME        | VARCHAR2 (200 CHAR) |
|                                        | MO_DATE        | DATE                |
|                                        | MO_STATUS      | NUMBER              |
| MAILORDER_PK (ID)                      |                |                     |
| MAILORDER_CUSTOMER_FK (MO_CUSTOMER_ID) |                |                     |
| IDX_FK_MO_CUSTOMER_ID (MO_CUSTOMER_ID) |                |                     |

1.000 mailorders

| KLM.MAILORDERPOS                             |                  |        |
|----------------------------------------------|------------------|--------|
| P *                                          | ID               | NUMBER |
| F *                                          | MOP_MAILORDER_ID | NUMBER |
| *                                            | MOP_NO           | NUMBER |
|                                              | MOP_QUANTITY     | NUMBER |
| F *                                          | MOP_ARTICLE_ID   | NUMBER |
|                                              | MOP_STATUS       | NUMBER |
| MAILORDERPOS_PK (ID)                         |                  |        |
| MAILORDERPOS_MAILORDER_FK (MOP_MAILORDER_ID) |                  |        |
| MAILORDERPOS_ARTICLE_FK (MOP_ARTICLE_ID)     |                  |        |
| IDX_FK_MOP_MAILORDER_ID (MOP_MAILORDER_ID)   |                  |        |
| IDX_FK_MOP_ARTICLE_ID (MOP_ARTICLE_ID)       |                  |        |

1.100.000 mailorderpos's

| KLM.ARTICLE     |                    |                     |
|-----------------|--------------------|---------------------|
| P *             | ID                 | NUMBER              |
| *               | ART_NO             | NUMBER              |
| *               | ART_NAME           | VARCHAR2 (200 CHAR) |
|                 | ART_SALES_PRICE    | NUMBER (10,2)       |
|                 | ART_PURCHASE_PRICE | NUMBER (10,2)       |
| ARTICLE_PK (ID) |                    |                     |

101 articles

# Simple Cases Advanced Problems

# TABLE ACCESS FULL vs. INDEX+TABLE

```
6 select /*+noparallel*/ *
7 from mailorderpos
8 where MOP_NO=8306
9 ;
```

One out of 10k+

Autotrace x

SQL HotSpot 2,426 Sekunden

| OPERATION        | OBJECT_NAME  | OPTIONS | CARDINALITY | COST | LAST_CR_BUFFER_GETS | LAST_ELAPSED_TIME |
|------------------|--------------|---------|-------------|------|---------------------|-------------------|
| SELECT STATEMENT |              |         |             | 375  | 1382                | 3030              |
| TABLE ACCESS     | MAILORDERPOS | FULL    | 1           | 375  | 1382                | 3030              |

Filter Predicates  
MOP\_NO=8306

No Index on MOP\_NO

```
1 select /*+noparallel*/ *
2 from mailorderpos
3 where ID=368602
4 ;
```

Autotrace x

SQL HotSpot 2,423 Sekunden

| OPERATION        | OBJECT_NAME     | OPTIONS        | CARDINALITY | COST | LAST_CR_BUFFER_GETS | LAST_ELAPSED_TIME |
|------------------|-----------------|----------------|-------------|------|---------------------|-------------------|
| SELECT STATEMENT |                 |                |             | 2    | 3                   | 26                |
| TABLE ACCESS     | MAILORDERPOS    | BY INDEX ROWID | 1           | 2    | 3                   | 26                |
| INDEX            | MAILORDERPOS_PK | UNIQUE SCAN    | 1           | 1    | 2                   | 16                |

Access Predicates  
ID=368602

PK Index on (MOP) ID



# AI to the Rescue?

```
select *  
from MAILORDERPOS  
where MOP_QUANTITY <30;
```

=> SLOW!

DBMS\_XPLAN.DISPLAY\_CURSOR() => Text => Copy => Paste

ChatGPT 4o ▾

How can I improve this queryplan in an oracle database 19c?

# AI to the Rescue?

```
select /*KLMTEST2*/ * from mailorderpos where MOP_QUANTITY < 30
```

Plan hash value: 2142752586

```
-----  
-----  
| Id | Operation      | Name      | Starts | E-Rows | E-Bytes | Cost  
(%CPU)| E-Time   | A-Rows | A-Time   | Buffers |  
-----  
-----  
| 0 | SELECT STATEMENT |          |      1 |        |        | 1306 (100)|  
| 50 | 00:00:00.01 |      10 |  
|* 1 | TABLE ACCESS FULL| MAILORDERPOS |      1 | 31932 | 841K|  
1306 (1)| 00:00:01 |      50 | 00:00:00.01 |      10 |  
-----  
-----
```



# AI to the Rescue?



Improving the query plan for the provided query in Oracle Database 19c involves **reducing** the **TABLE ACCESS FULL** operation, which can be expensive, especially for large tables. Here are steps and considerations to improve performance:

## True!

### 1. Check Index Availability

- If there's an index on the `MOP_QUANTITY` column, Oracle might be able to use it for faster access.
- Create an index if it doesn't exist:

sql

Code kopieren

```
CREATE INDEX idx_mop_quantity ON mailorderpos (MOP_QUANTITY);
```

- After creating the index, run the query again to see if Oracle chooses it ( `INDEX RANGE SCAN` instead of `TABLE ACCESS FULL` ).

# AI to the Rescue?

## 2. Gather Statistics

- Ensure that Oracle has up-to-date statistics on the `mailorderpos` table and its indexes:

sql

 Code kopieren

```
BEGIN
```

```
  DBMS_STATS.GATHER_TABLE_STATS('YOUR_SCHEMA', 'MAILORDERPOS'); <...>
```

```
END;
```

```
/
```

- Updated statistics help Oracle's optimizer make better decisions.

True!  
But not helpful.  
LLM misses information.

# AI to the Rescue?

## 3. Filter Selectivity

- Check the **selectivity** of the `MOP\_QUANTITY`

Here we go!

# Why NOT to force the optimizer

[Not even the AI suggested it... just sayin']

# Data Distribution in MAILORDERPOS

| KLM.MAILORDERPOS     |                  |        |
|----------------------|------------------|--------|
| P *                  | ID               | NUMBER |
| F *                  | MOP_MAILORDER_ID | NUMBER |
|                      | MOP_NO           | NUMBER |
|                      | MOP_QUANTITY     | NUMBER |
| F *                  | MOP_ARTICLE_ID   | NUMBER |
|                      | MOP_STATUS       | NUMBER |
| MAILORDERPOS_PK (ID) |                  |        |

Index + Histogram on MOP\_STATUS

Nearly All  
"DONE"

Some  
"OPEN"

Very Few  
"IN WORK"

| MOP_STATUS | X       |
|------------|---------|
| 99         | 1093272 |
| 1          | 6727    |
| 50         | 1       |

# Data Distribution in MAILORDERPOS

```
select * from dba_tab_col_statistics
where owner='KLM'
and table_name='MAILORDER'
and column_name='MO_STATUS'
;
```

## DBA\_TAB\_COL\_STATISTICS

|       |            |             |              |           |            |         |           |             |                     |             |              |            |             |             |             |
|-------|------------|-------------|--------------|-----------|------------|---------|-----------|-------------|---------------------|-------------|--------------|------------|-------------|-------------|-------------|
| OWNER | TABLE_NAME | COLUMN_NAME | NUM_DISTINCT | LOW_VALUE | HIGH_VALUE | DENSITY | NUM_NULLS | NUM_BUCKETS | LAST_ANALYZED       | SAMPLE_SIZE | GLOBAL_STATS | USER_STATS | NOTES       | AVG_COL_LEN | HISTOGRAM   |
| KLM   | MAILORDER  | MO_STATUS   | 3            | C102      | C164       | 0,0005  | 0         | 3           | 15.03.2024 17:15:26 | 1000        | YES          | NO         | HYPERLOGLOG |             | 3 FREQUENCY |

FREQUENCY based  
histogram on  
MOP\_STATUS

## DBA\_TAB\_HISTOGRAMS

```
select * from dba_tab_histograms
where owner='KLM'
and table_name='MAILORDER'
and column_name='MO_STATUS'
order by column_name
;
```

Three Buckets  
show SKEWED value  
distribution

|       |            |             |                 |                |                       |                 |
|-------|------------|-------------|-----------------|----------------|-----------------------|-----------------|
| OWNER | TABLE_NAME | COLUMN_NAME | ENDPOINT_NUMBER | ENDPOINT_VALUE | ENDPOINT_ACTUAL_VALUE | ENDPOINT_ACTUAL |
| KLM   | MAILORDER  | MO_STATUS   | 6               | 11             |                       | C102            |
| KLM   | MAILORDER  | MO_STATUS   | 7               | 50             | 50                    | C133            |
| KLM   | MAILORDER  | MO_STATUS   | 1000            | 99 99          |                       | C164            |



# Conflict by Cursor Sharing

Query for the Mailorder Position

**IN WORK**

EXECUTE :a := **50**

select \*

from MAILORDERPOS

where MOP\_STATUS=:a;

**1 row matches**

INDEX RANGE SCAN

IDX\_MOP\_STATUS

(non-unique column)

+

TABLE ACCESS BY INDEX ROWID BATCHED

MAILORDERPOS

(we want to see all columns)



performing  
databases

Query for the Mailorder Position

**DONE**

EXECUTE :a := **99**

select \*

from MAILORDERPOS

where MOP\_STATUS=:a;

**1.000.000 rows match**

TABLE ACCESS FULL  
MAILORDERPOS  
(we want to see all columns)

**VS.**

# Adaptive Cursor Sharing

After a few executions of both, cursor becomes a PARENT CURSOR

IS\_BIND\_AWARE = Y  
IS\_BIND\_SENSITIVE=Y

GV\$SQL

| SQL_ID        | CHILD_NUMBER | PLAN_HASH_VALUE | OPTIMIZER_COST | EXECUTIONS | IS_BIND_SENSITIVE | IS_BIND_AWARE | INST_ID | SQL_TEXT                |
|---------------|--------------|-----------------|----------------|------------|-------------------|---------------|---------|-------------------------|
| d95h4wsmrywsb | 1            | 2142752586      | 1307           | 6          | Y                 | Y             | 2       | select * from mailorder |
| d95h4wsmrywsb | 2            | 4102084357      | 4              | 5          | Y                 | Y             | 2       | select * from mailorder |
| d95h4wsmrywsb | 3            | 4102084357      | 4              | 4          | Y                 | N             | 2       | select * from mailorder |

CHILD 1 = TABLE ACCESS FULL  
CHILD 2 = INDEX + TABLE

# Adaptive Cursor Sharing

DBMS\_XPLAN.DISPLAY\_CURSOR('<SQL\_ID>',null,'COST,IOSTATS,LAST,ADVANCED,ADAPTIVE')

SQL\_ID d95h4wsmrywsb, child number 1

select \* from mailorderpos where mop\_status=:a

Plan hash value: 2142752586

CHILD 1

| Id  | Operation         | Name         | Starts | E-Rows | E-Bytes | Cost (%CPU) | E-Time   | A-Rows | A-Time      | Buffers |
|-----|-------------------|--------------|--------|--------|---------|-------------|----------|--------|-------------|---------|
| 0   | SELECT STATEMENT  |              | 1      |        |         | 1307 (100)  |          | 1093K  | 00:00:00.12 | 6963    |
| * 1 | TABLE ACCESS FULL | MAILORDERPOS | 1      | 1093K  | 27M     | 1307 (1)    | 00:00:01 | 1093K  | 00:00:00.12 | 6963    |

TABLE ACCESS FULL

6963  
Buffer Gets

SQL\_ID d95h4wsmrywsb, child number 2

select \* from mailorderpos where mop\_status=:a

Plan hash value: 4102084357

CHILD 2

| Id  | Operation                           | Name           | Starts | E-Rows | E-Bytes | Cost (%CPU) | E-Time   | A-Rows | A-Time      | Buffers |
|-----|-------------------------------------|----------------|--------|--------|---------|-------------|----------|--------|-------------|---------|
| 0   | SELECT STATEMENT                    |                | 1      |        |         | 4 (100)     |          | 1      | 00:00:00.01 | 4       |
| 1   | TABLE ACCESS BY INDEX ROWID BATCHED | MAILORDERPOS   | 1      | 1      | 26      | 4 (0)       | 00:00:01 | 1      | 00:00:00.01 | 4       |
| * 2 | INDEX RANGE SCAN                    | IDX_MOP_STATUS | 1      | 1      |         | 3 (0)       | 00:00:01 | 1      | 00:00:00.01 | 3       |

INDEX RANGE SCAN  
+ TABLE ACC.

4  
Buffer Gets

# My Bug or Your Feature?



# Somebody meant to help ...

(Got rid of some TABLE ACCESS FULL by forcing the CBO into our STATUS index)

```
select /*+ INDEX(MAILORDERPOS IDX_MOP_STATUS) */ *  
  from MAILORDERPOS  
 where MOP_STATUS=:a;
```

But if :a = 99  
-> 1.000.000 rows match

Hint **forces** the CBO into:

INDEX RANGE SCAN  
 IDX\_MOP\_STATUS  
 +  
TABLE ACCESS BY INDEX ROWID BATCHED  
 MAILORDERPOS

=> ???

Without tampering:

TABLE ACCESS FULL  
MAILORDERPOS

=> 6963 Buffer Gets

# Somebody meant to help ...

(Got rid of some TABLE ACCESS FULL by forcing the CBO into our STATUS index)

SQL\_ID 06yfbwymbd86p, child number 0

```
select /*+ INDEX(MAILORDERPOS IDX_MOP_STATUS) */ * from mailorderpos
where mop_status=:a
```

DBMS\_XPLAN.DISPLAY\_CURSOR()

Plan hash value: 4102084357

| Id  | Operation                           | Name           | Starts | E-Rows | E-Bytes | Cost (%CPU) | E-Time   | A-Rows | A-Time      | Buffers |
|-----|-------------------------------------|----------------|--------|--------|---------|-------------|----------|--------|-------------|---------|
| 0   | SELECT STATEMENT                    |                | 1      |        |         | 10635 (100) |          | 1093K  | 00:00:00.35 | 11380   |
| 1   | TABLE ACCESS BY INDEX ROWID BATCHED | MAILORDERPOS   | 1      | 1093K  | 27M     | 10635 (1)   | 00:00:01 | 1093K  | 00:00:00.35 | 11380   |
| * 2 | INDEX RANGE SCAN                    | IDX_MOP_STATUS | 1      | 1093K  |         | 2279 (1)    | 00:00:01 | 1093K  | 00:00:00.13 | 4450    |

INDEX RANGE SCAN  
+ TABLE ACC.

11.380  
Buffer Gets

## Increases Buffer Gets +60%

## 6.963 vs. 11.380

Detour via Index is inefficient here.

# More unlikely, but worse

(Got rid of some inefficient INDEX ACCESS by forcing the CBO into TABLE ACC. FULL)

**/\*+ NO\_INDEX(MAILORDERPOS) \*/**

DBMS\_XPLAN.DISPLAY\_CURSOR()

| Id  | Operation         | Name         | Starts | E-Rows | E-Bytes | Cost (%CPU) | E-Time   | A-Rows | A-Time      | Buffers |
|-----|-------------------|--------------|--------|--------|---------|-------------|----------|--------|-------------|---------|
| 0   | SELECT STATEMENT  |              | 1      |        |         | 1307 (100)  |          | 1      | 00:00:00.03 | 4787    |
| * 1 | TABLE ACCESS FULL | MAILORDERPOS | 1      | 1      | 26      | 1307 (1)    | 00:00:01 | 1      | 00:00:00.03 | 4787    |

TABLE ACCESS FULL

4787  
Buffer Gets

**Increases Buffer Gets Factor 1.000**  
**4 vs. 4787**

TABLE ACCESS FULL is inefficient here.

# The CBO is smart!

It's way more flexible than

~~Hints~~

~~Stored Outlines~~

~~SQL Profiles~~



# The Emergency Brake of the CBO:

## Adaptive Plans

# Count MAILORDERs of One Customer

```
select /* SHOWCASE2 */ cu.cu_no, count(*) x
  from    klm.customer cu,
          klm.mailorder mo
 where    mo.mo_customer_id=cu.id
        and cu.id=73
 group by cu.cu_no
 order by x desc;
```

Bad (no) Statistics

lead to severe misestimates

Adaptive Plan corrects

=> 4 Buffer Gets

| Id   | Operation                   |                       |
|------|-----------------------------|-----------------------|
| 0    | SELECT STATEMENT            |                       |
| 1    | SORT ORDER BY               |                       |
| 2    | HASH GROUP BY               |                       |
| - *  | HASH JOIN                   |                       |
| 4    | NESTED LOOPS                |                       |
| -    | STATISTICS COLLECTOR        |                       |
| 6    | VIEW                        | VW_GBF_7              |
| 7    | TABLE ACCESS BY INDEX ROWID | CUSTOMER              |
| * 8  | INDEX UNIQUE SCAN           | CUSTOMER_PK           |
| * 9  | INDEX RANGE SCAN            | IDX_FK_MO_CUSTOMER_ID |
| - 10 | TABLE ACCESS FULL           | MAILORDER             |

Changes  
HASH JOIN for NESTED LOOP

Buffer Gets 4 instead of ~400

Collects Cardinality Info

Out: TABLE ACC FULL on MO  
In: RANGE SCAN on FK INDEX



# Cheap, but Effective

- Indexes (also Hidden)
  - Multi-Column
  - Function-Based
  - Careful w/ removing Indexes!
- Statistics
  - Extended (think Zodiac vs. Birthday)
  - Hidden Statistics
- Constraints

# Tools (not) (only) (for) Fools

# AHF Future Diagnostic Assistant

## Diagnostic Assistant's on-demand responses

Diagnostics Specific  
Product Specific

What are the CRS issues from alert log around 3AM ?



| Timestamp           | Type  | Event    | Hostname         | Description                                                                                                                                                                             |
|---------------------|-------|----------|------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 2024-05-31T02:49:48 | ERROR | CRS_1607 | ahfexacs3-voef52 | [OCSSD(54628)]CRS-1607: Node ahfexacs3-voef51 is being evicted in cluster incarnation 60677358; details at (CSSNM00007) in /u01/app/grid/diag/crs/ahfexacs3-voef52/crs/trace/ocssd.trc. |

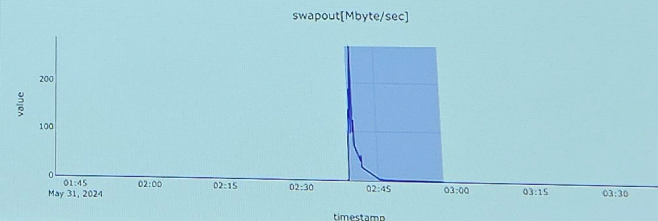


What is the purpose of log writer process ?

 The Log Writer Process (LGWR) is a background process in Oracle Database responsible for writing redo logs, which are records of changes made to the database. When a transaction is committed, LGWR writes the corresponding redo logs to ensure data consistency and durability. It also performs checkpointing, which periodically writes the current state of the database to the redo logs to protect against instance failures. The log file sync wait event relates to the time LGWR spends waiting for I/O operations to write redo logs to disk.

Any OS Issues identified 15 minute before 3AM ?

 Swapping was detected on node **ahfexacs3-voef51** from 02:39:30 on 31 May, 2024 to 02:57:40 on 31 May, 2024



What is a node eviction issue ?

Support / DevOps

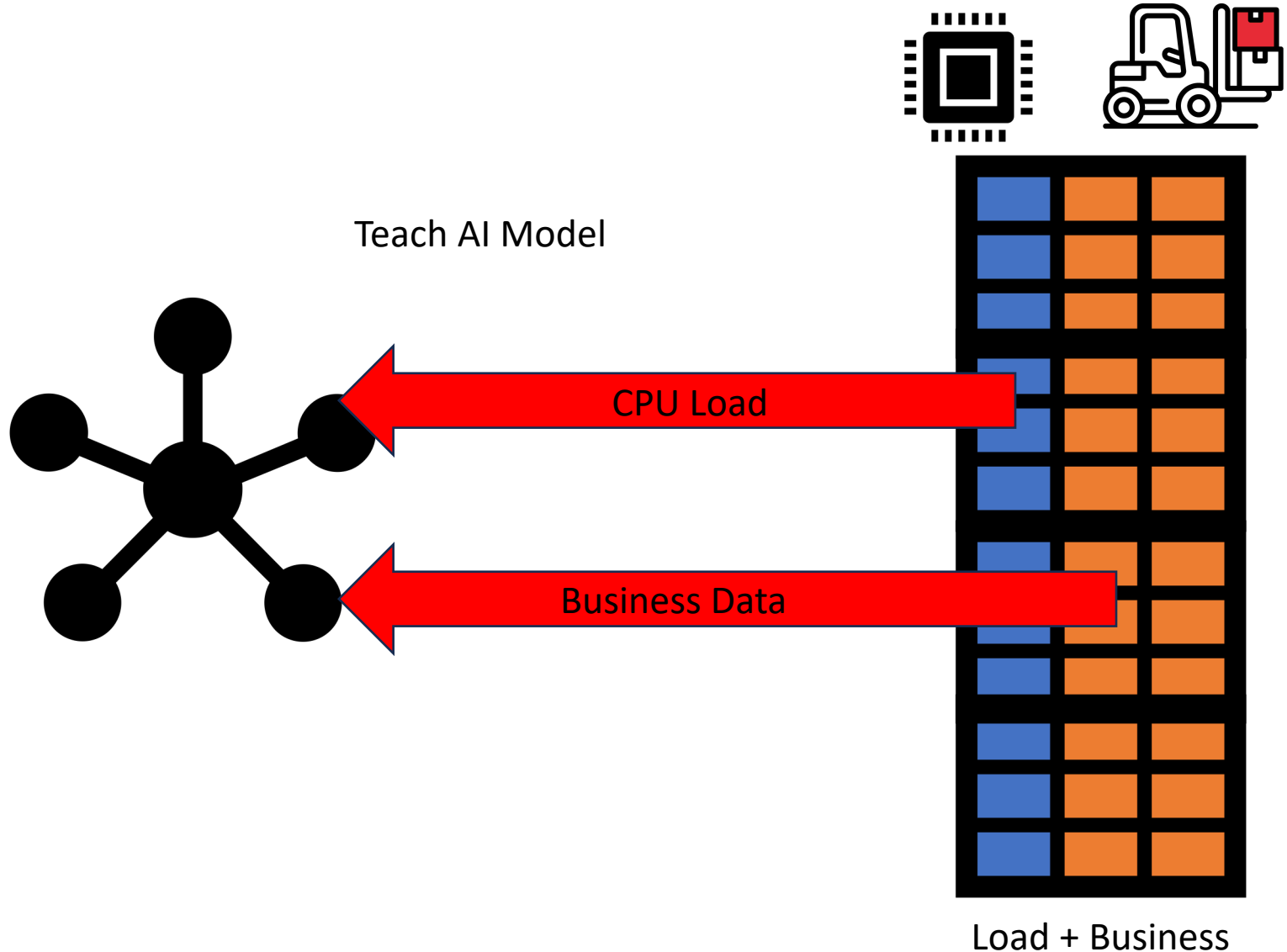
 In the context of a cluster environment, a node eviction refers to the removal of a node from the cluster due to certain conditions or issues. There are different reasons why a node might be evicted, such as network failure, software failure in the local CSS daemon, or even intervention by a vendor node monitor.

When a node is evicted, it can no longer participate in the cluster and may need to be addressed to rejoin. The specific cause of the eviction determines the steps required to resolve the issue. For example, if the eviction was due to network failure, then verifying the stability of the private network between the affected nodes would be necessary. If the eviction was due to software failure in the local CSS daemon, then configuring a voting device and restarting the cluster might be required.

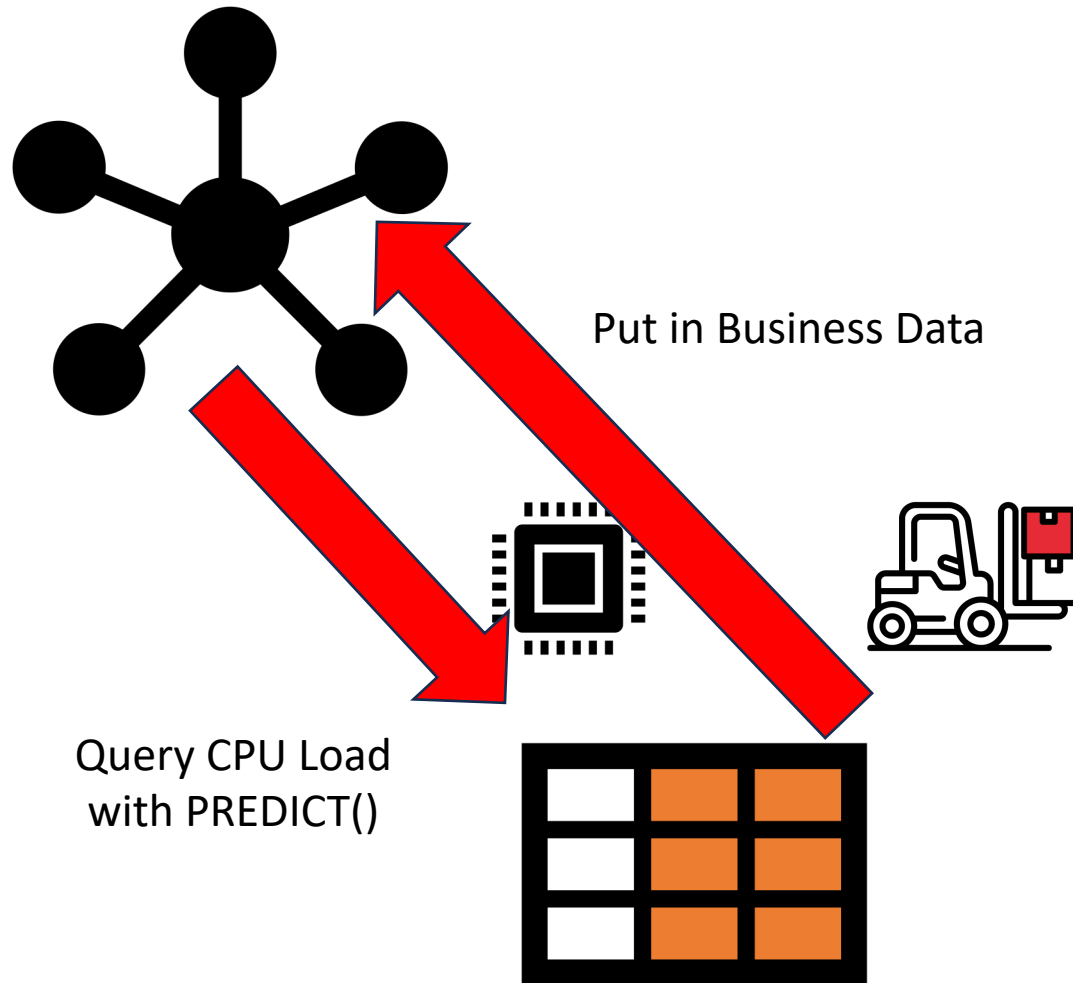
It's important to note that node evictions can impact the availability and performance of the cluster, so addressing the underlying cause promptly is crucial to ensure the continuity of the system.

# My Last Research

# Predicting Workload



# Predicting Workload





# My Prediction Model – Free!

BEGIN

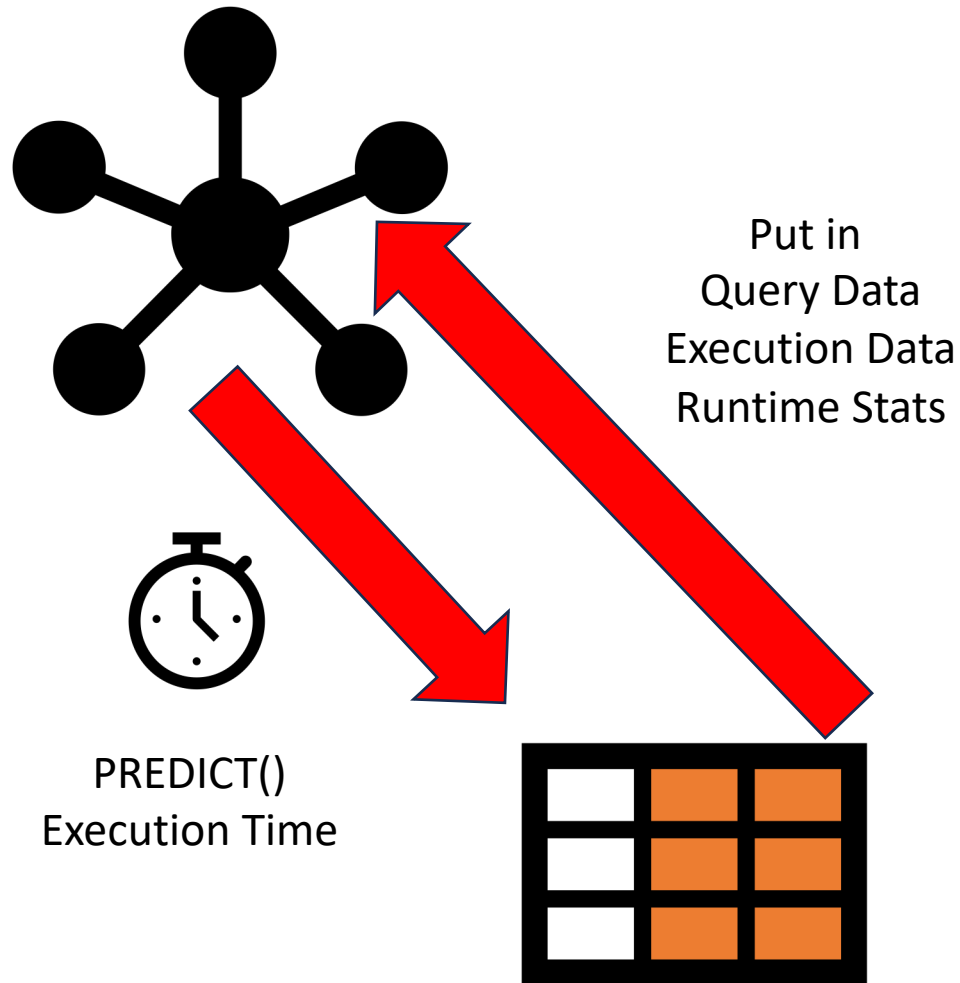
```
DBMS_DATA_MINING.CREATE_MODEL(  
    model_name           => 'klm_cpu_usage_model',  
    mining_function       => DBMS_DATA_MINING.REGRESSION,  
    data_table_name       => 'TRAINING_DATA', -- here is our input  
    case_id_column_name   => 'ROW_IDF', -- no PK available  
    target_column_name    => 'SYSMETRIC_CPU_CENTISEC_PER_SEC',  
    settings_table_name   => 'model_settings');
```

END;

```
select PREDICTION(klm_cpu_usage_model USING *)  
from SITUATION_DATA;
```

# My Next Research

# Predicting Execution Times



# Keep the AI busy!

# Generative AI - SQL

select AI showsql

what is our topselling article in 2023,  
and how much was sold, and how much money made

```
1  "SELECT
2      "A"."ART_NAME" AS "ArticleName",
3      SUM("MOP"."MOP_QUANTITY") AS "TotalQuantitySold",
4      SUM("MOP"."MOP_QUANTITY" * "A"."ART_SALES_PRICE") AS "TotalRevenue"
5  FROM
6      "KLM"."MAILORDERPOS" "MOP"
7  JOIN
8      "KLM"."ARTICLE" "A" ON "MOP"."MOP_ARTICLE_ID" = "A"."ID"
9  JOIN
10     "KLM"."MAILORDER" "MO" ON "MOP"."MOP_MAILORDER_ID" = "MO"."ID"
11 WHERE
12     EXTRACT(YEAR FROM "MO"."MO_DATE") = 2023
13 GROUP BY
14     "A"."ART_NAME"
15 ORDER BY
16     "TotalQuantitySold" DESC
17 FETCH FIRST 1 ROWS ONLY"
```

# Keep the AI busy!

select ai chat 'can you improve this query?

"SELECT "A"."ART\_NAME" AS "ArticleName", <...>

1. **\*\*Use of Aliases\*\*:** 

The use of table aliases is good, but you can make them more intuitive.

2. **\*\*Use of CTE\*\*:** 

If the query becomes more complex, using a Common Table Expression (CTE) can improve readability.

3. **\*\*Indexing\*\*:** 

Ensure that the columns used in JOINS and WHERE clauses are indexed for better performance.

4. **\*\*Use of LIMIT\*\*:** 

Depending on the **SQL dialect**, `LIMIT` can be used instead of `FETCH FIRST` for better readability.

Natural ~~Stupidity~~ Creativity  
beats  
Artificial Intelligence  
any time.

# Meet & Greet

[martin.klier@performing-db.com](mailto:martin.klier@performing-db.com)

[www.performing-databases.com](http://www.performing-databases.com)

Many national // international events

Next:

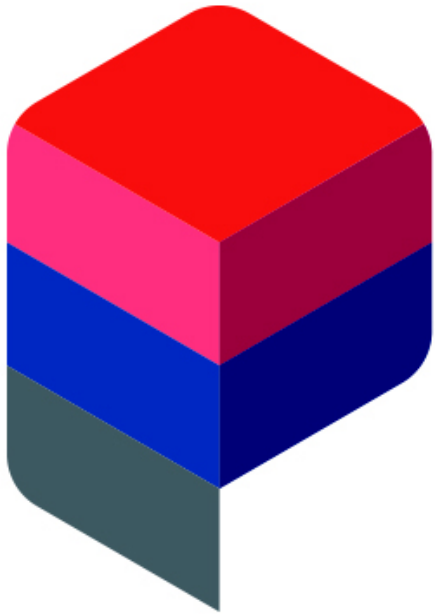


This year:



@MartinKlierDBA  
[www.performing-databases.com](http://www.performing-databases.com)





**performing  
databases**