

robotron[®]

DOAG Konferenz 2012

SQL ausführen – der Optimizer hat einen Plan

Marco Mischke – 20.11.2012



Zahlen und Fakten

Robotron Datenbank-Software GmbH

robotron[®]

Gründungsjahr **1990**

Mitarbeiterzahl **283** (Stand 07/2012)

Stammkapital **2,4 Mio. EUR**

Umsatz 2012 **31,1 Mio. EUR**

Geschäftssitz **Stuttgarter Str. 29
01189 Dresden**



ORACLE Platinum
Partner

Oracle-Partner

ORACLE APPROVED
EDUCATION CENTER



ISO 9001 zertifiziert

ORACLE Platinum
Partner



Optimizer

- Aufgaben
- Möglichkeiten
- Randbedingungen

Zugriffspfade

- Welche gibt es
- Wie funktionieren diese

Ausführungspläne

- Aufbau
- Interpretation
- Besondere Möglichkeiten

Gut zu wissen

- Indexe
- Statistiken
- Cursor Sharing



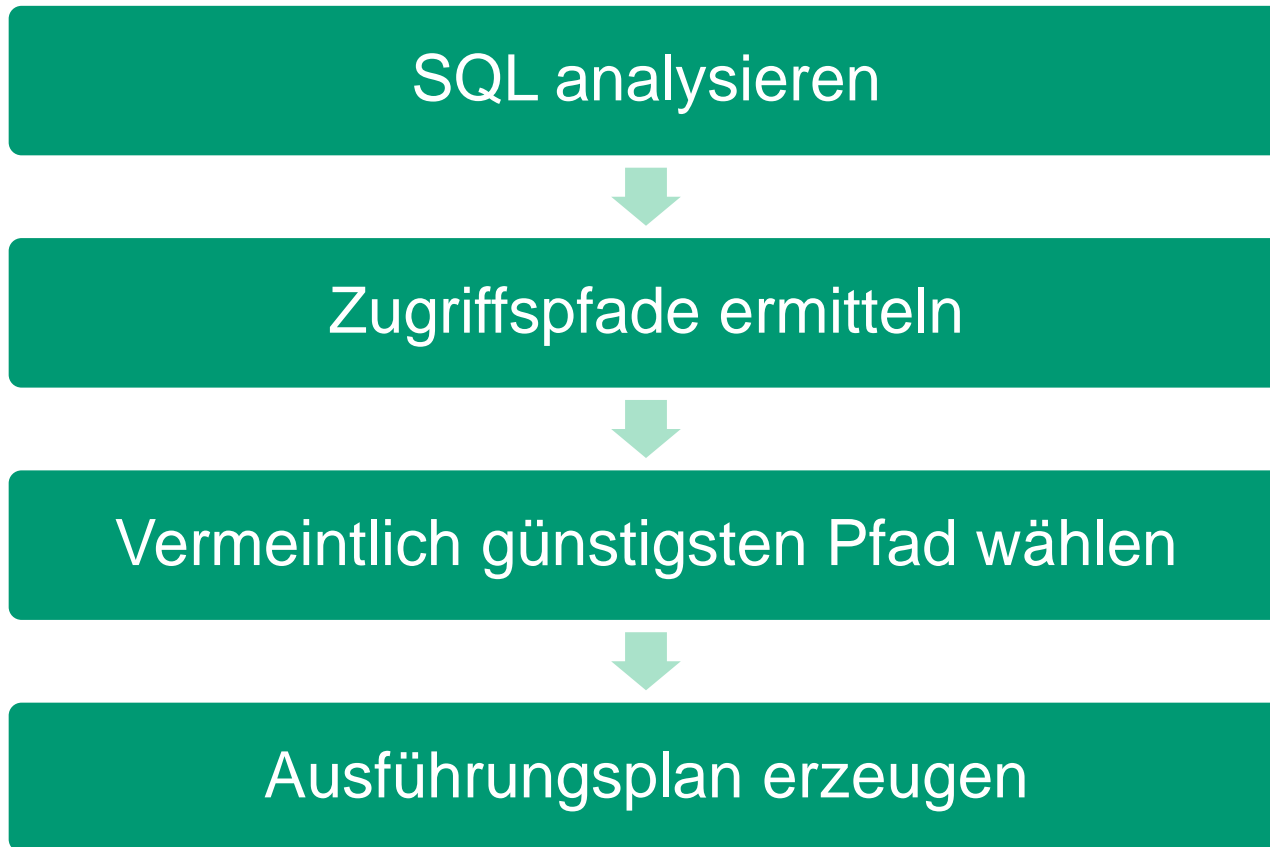
```
Select
  a.a, a.b, b.a
From
  mytable_a a,
  mytable_b b
Where
  a.id = b.id
```

Hier geschieht ein Wunder

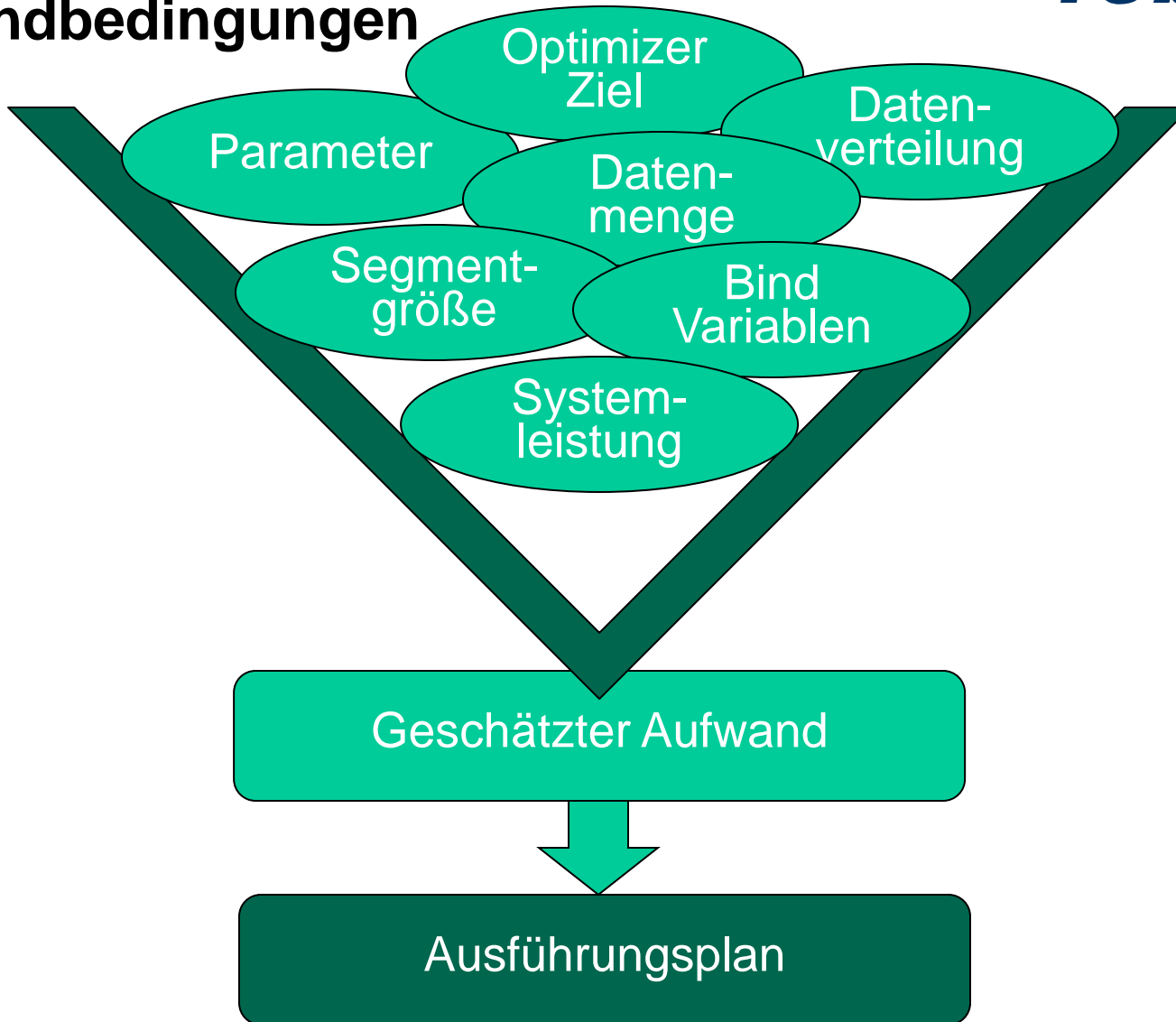
a.a	a.b	b.a
A	B	C
AA	CC	BB
BC	BA	CB
CBA	CBO	ABA



Optimizer – Was macht der eigentlich?



Optimizer – Randbedingungen



Optimizer – die heutige Welt

robotron[®]

- ▶ Post Oracle 8 - Cost Based Optimizer
 - Unterstützt neue Features
 - Bitmap Indexe
 - Hash Joins
 - ...
 - schätzt den wahrscheinlichen Arbeitsaufwand
 - Objektstatistiken (Anzahl Zeilen, ...)
 - Systemstatistiken (I/O Leistung, ...)
 - Der günstigste Plan wird genommen
 - Kosten enthalten I/O, CPU, Netzwerkressourcen
 - Kosten dienen nur zum internen Vergleich der Pläne!



Optimizer

- Aufgaben
- Möglichkeiten
- Randbedingungen

Zugriffspfade

- Welche gibt es
- Wie funktionieren diese

Ausführungspläne

- Aufbau
- Interpretation
- Besondere Möglichkeiten

Gut zu wissen

- Indexe
- Statistiken
- Cursor Sharing

singleblock I/O

- ▶ ROWID über physikalische Adresse
- ▶ Index Unique Scan über eindeutigen Wert
- ▶ Index Range Scan über Wertebereich oder nicht eindeutigen Wert
- ▶ Index Skip Scan über zusammengesetzten Index
- ▶ Index Full Scan um Daten zu sortieren

multiblock I/O

- ▶ Index Fast Full Scan alle selektierten Daten im Index
- ▶ Full Table Scan einmal alles bitte

- ▶ Nested Loop geschachtelte Schleife (For Loop)
kann unvollständig ausgeführt werden
→ effizient für die ersten Datensätze
- ▶ Hash Join „Übereinanderlegen“ der Daten
Join wird in einem Durchgang ausgeführt
→ effizient für gesamte Ergebnismenge
- ▶ Sort Merge Join Sortieren und Nebeneinanderlegen
der Spaltenspalten
→ Optimal für „ungleich“ Operatoren



Optimizer

- Aufgaben
- Möglichkeiten
- Randbedingungen

Zugriffspfade

- Welche gibt es
- Wie funktionieren diese

Ausführungspläne

- Aufbau
- Interpretation
- Besondere Möglichkeiten

Gut zu wissen

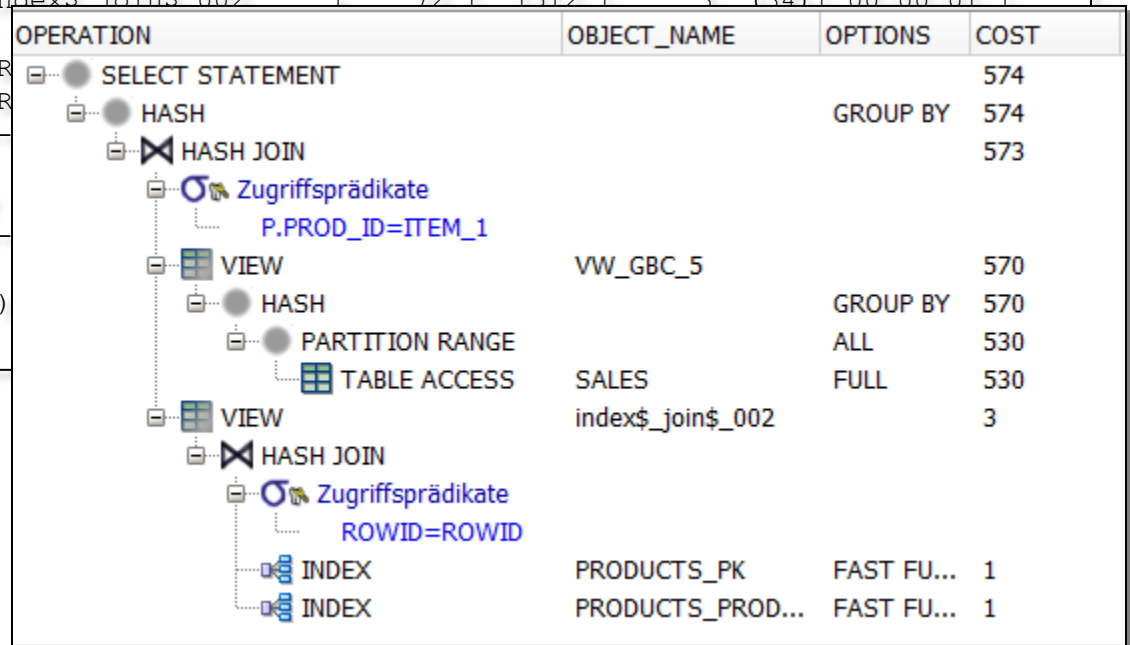
- Indexe
- Statistiken
- Cursor Sharing

Ausführungspläne – was sind das?

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT				574 (100)	
1	HASH GROUP BY		5	255	574 (10)	00:00:07
* 2	HASH JOIN		72	3672	573 (10)	00:00:07
3	VIEW	VW_GBC_5	72	2160	570 (10)	00:00:07
4	HASH GROUP BY		72	648	570 (10)	00:00:07
5	PARTITION RANGE ALL		918K	8075K	530 (3)	00:00:07
6	TABLE ACCESS FULL	SALES	918K	8075K	530 (3)	00:00:07
7	VIEW	index\$_join\$_002	72	1512	3 (34)	00:00:01
* 8	HASH JOIN					
9	INDEX FAST FULL SCAN	PR				
10	INDEX FAST FULL SCAN	PR				

Predicate Information (identified by

- 2 - access("P"."PROD_ID"="ITEM_1")
- 8 - access(ROWID=ROWID)



Ausführungspläne – Wo bekomme ich die her?

▶ EXPLAIN PLAN FOR

- Keine Binds, evtl. andere Umgebung → geratener Plan

▶ DBMS_XPLAN

- Plan aus dem Cache
- `.DISPLAY_CURSOR` → Plan des letzten Statements
- `.DISPLAY_CURSOR(sqlid, child#)` → ein beliebiger Plan

▶ SQL*Plus

- `set autotrace traceonly` → benutzt im Hintergrund `EXPLAIN PLAN FOR`

▶ SQL Trace / tkprof

- Tracefile enthält den ermittelten Plan sofern Hard Parse oder `dbms_monitor.[...]_trace_enable(... plan_stat=>'ALL_EXECUTIONS')`
- **Achtung!** „explain=user/pwd“ macht nur `EXPLAIN PLAN FOR`

➤ Ausführungspläne – Wahrheit und Dichtung

- ▶ Angaben in Ausführungsplan zu
 - Anzahl Zeilen
 - Anzahl Bytes
 - Ausführungszeit

Entsprechen den Schätzungen

- ▶ Tatsächliche Werte können stark abweichen
- ▶ Werden ermittelt durch erweiterte Statistiken
 - `alter session set statistics_level=all;`
 - `select /*+ GATHER_PLAN_STATISTICS */ ...`
- ▶ Werden ausgewertet mit
 - `tkprof`
 - `dbms_xplan.display_cursor`

Ausführungspläne – Wahrheit und Dichtung - tkprof

```
▶ SQL> exec dbms_monitor.session_trace_enable(waits=>true, binds=>true,  
                                              plan_stat=>'ALL_EXECUTIONS')
```

PL/SQL-Prozedur erfolgreich abgeschlossen.

```
▶ SQL> SELECT prod_category, AVG(amount_sold)  
 2 FROM sales s, products p  
 3 WHERE p.prod_id = s.prod_id  
 4 GROUP BY prod_category;
```

Abgelaufen: 00:00:01.95

```
▶ SQL> exec dbms_monitor.session_trace_disable;
```

PL/SQL-Prozedur erfolgreich abgeschlossen.

```
▶ tkprof marco_ora_12330.trc marco_ora_12330.trc.tkp sys=no
```

Ausführungspläne – Wahrheit und Dichtung - tkprof

Rows (1st) Row Source Operation

```
-----  
      5 HASH GROUP BY (cr=1641 pr=1095 pw=1095 time=1830994 us cost=8  
        size=150 card=5)  
918843 HASH JOIN      (cr=1641 pr=1095 pw=1095 time=3068320 us cost=7  
        size=300 card=10)  
918843 PARTITION RANGE ALL PARTITION: 1 28  
        (cr=1635 pr=0 pw=0 time=5077500 us cost=4  
        size=90 card=10)  
918843 TABLE ACCESS FULL SALES PARTITION: 1 28 Tatsächliche Menge  
        (cr=1635 pr=0 pw=0 time=1980264 us cost=4  
        size=90 card=10) Schätzung  
72 VIEW index$_join$_002  
        (cr=6 pr=0 pw=0 time=4386 us cost=3 size=1512  
        card=72)  
72 HASH JOIN (cr=6 pr=0 pw=0 time=4094 us)  
72 INDEX FAST FULL SCAN PRODUCTS_PK  
        (cr=3 pr=0 pw=0 time=187 us cost=1 size=1512  
        card=72)  
72 INDEX FAST FULL SCAN PRODUCTS_PROD_CAT_IX  
        (cr=3 pr=0 pw=0 time=454 us cost=1 size=1512  
        card=72)
```


Ausführungspläne – Wahrheit und Dichtung - DBMS_XPLAN



▶ SQL> alter session set statistics_level=all;

Session wurde geändert.

▶ SQL> SELECT prod_category, AVG(amount_sold)
2 FROM sales s, products p
3 WHERE p.prod_id = s.prod_id
4 GROUP BY prod_category;

Abgelaufen: 00:00:15.22

▶ SQL> select * from table(dbms_xplan.display_cursor(format=>'ALLSTATS LAST'));

Ausführungspläne – Wahrheit und Dichtung - DBMS_XPLAN

Id	Operation	Name	Starts	E-Rows	A-Rows	A-Time
0	SELECT STATEMENT		1		5	00:00:15.11
1	HASH GROUP BY		1	5	5	00:00:15.11
* 2	HASH JOIN		1	10	918K	00:00:13.21
3	PARTITION RANGE ALL		1	10	918K	00:00:05.21
4	TABLE ACCESS FULL	SALES	28	10	918K	00:00:02.18
5	VIEW	index\$_join\$_002	1	72	72	00:00:00.01
* 6	HASH JOIN		1	72	72	00:00:00.01
7	INDEX FAST FULL SCAN	PRODUCTS_PK	1	72	72	00:00:00.01
8	INDEX FAST FULL SCAN	PRODUCTS_PROD_CAT_IX	1	72	72	00:00:00.01

Predicate Information (identified by operation id):

- 2 - access ("P"."PROD_ID"="S"."PROD_ID")
- 6 - access (ROWID=ROWID)

Geschätzte Menge
(estimated rows)

Tatsächliche Menge
(actual rows)



Optimizer

- Aufgaben
- Möglichkeiten
- Randbedingungen

Zugriffspfade

- Welche gibt es
- Wie funktionieren diese

Ausführungspläne

- Aufbau
- Interpretation
- Besondere Möglichkeiten

Gut zu wissen

- Indexe
- Statistiken
- Cursor Sharing

Ausführungspläne – was geht besser? Indexe und NULL

```
-----  
| Id | Operation          | Name | Rows | Bytes | Cost (%CPU) |  
-----  
|  0 | SELECT STATEMENT   |      |  1   |  40   |  3 (0)      |  
|*  1 | TABLE ACCESS FULL| EMP  |  1   |  40   |  3 (0)      |  
-----
```

Predicate Information (identified by operation id):

```
1 - filter("EMP"."MGR") IS NULL)
```

```
create index IX_MGR  
on EMP (MGR, 0);
```

Ausführungspläne – was geht besser? Indexe und NULL

```
-----  
| Id | Operation                | Name      | Rows | Bytes | Cost |  
-----  
| 0  | SELECT STATEMENT         |           | 1    | 40    | 2 (0)|  
| 1  | TABLE ACCESS BY INDEX ROWID | EMP      | 1    | 40    | 2 (0)|  
|* 2  | INDEX RANGE SCAN         | IX_MGR   | 1    |       | 1 (0)|  
-----
```

Predicate Information (identified by operation id):

```
-----  
2 - access("MGR" IS NULL)
```

Ausführungspläne – was geht besser? GROSS/klein Schreibung

Id	Operation	Name
0	SELECT STATEMENT	
1	NESTED LOOPS	
2	NESTED LOOPS	
3	NESTED LOOPS	
* 4	TABLE ACCESS FULL	DEPARTMENTS
5	TABLE ACCESS BY INDEX ROWID	LOCATIONS
* 6	INDEX UNIQUE SCAN	LOC_ID PK
* 7	INDEX RANGE SCAN	EMP_DE
8	TABLE ACCESS BY INDEX ROWID	EMPLOY

Predicate Information (identified by operation)

- 4 - filter(UPPER("D"."DEPARTMENT_NAME")='SALES')
- 6 - access("D"."LOCATION_ID"="L"."LOCATION_ID")
- 7 - access("E"."DEPARTMENT_ID"="D"."DEPARTMENT_ID")

```
create index  
mm_dep_name_upper  
on departments (  
upper(department_name)  
);
```

Ausführungspläne – was geht besser? GROSS/klein Schreibung

Id	Operation	Name
0	SELECT STATEMENT	
1	NESTED LOOPS	
2	NESTED LOOPS	
3	NESTED LOOPS	
4	TABLE ACCESS BY INDEX ROWID	DEPARTMENTS
* 5	INDEX RANGE SCAN	MM_DEP_NAME_UPPER
6	TABLE ACCESS BY INDEX ROWID	LOCATIONS
* 7	INDEX UNIQUE SCAN	LOC_ID_PK
* 8	INDEX RANGE SCAN	EMP_DEPARTMENT_IX
9	TABLE ACCESS BY INDEX ROWID	EMPLOYEES

Predicate Information (identified by operation id):

- 5 - access("D"."SYS_NC00005\$"='SALES')
- 7 - access("D"."LOCATION_ID"="L"."LOCATION_ID")
- 8 - access("E"."DEPARTMENT_ID"="D"."DEPARTMENT_ID")

Ausführungspläne – was geht besser? Spaltenzusammenhänge

```
SQL> select * from emp where job='SALESMAN' and deptno=30;
```

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT		1	40	3 (0)	00:00:01
* 1	TABLE ACCESS FULL	EMP	1	40	3 (0)	00:00:01

Predicate Information (identified by operation id):

```
1 - filter("JOB"='SALESMAN' AND "DEPTNO"=30)
```

```
SQL> begin
2   dbms_stats.gather_table_stats(
3     user,
4     'EMP',
5     method_opt=>'for columns (job,deptno)');
6   end;
7   /
```


Ausführungspläne – was geht besser? Spaltenzusammenhänge

```
SQL> select * from emp where job='SALESMAN' and deptno=30;
```

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT		4	160	3 (0)	00:00:01
* 1	TABLE ACCESS FULL	EMP	4	160	3 (0)	00:00:01

Predicate Information (identified by operation id):

```
1 - filter("JOB"='SALESMAN' AND "DEPTNO"=30)
```

Ausführungspläne

Adaptive Cursor Sharing – Lauf 1

```
SQL> exec :id := 1;
SQL> select /*TEST*/ * from viel where nummer = :id;
```

19 Zeilen ausgewählt.

```
SQL> select sql_id ,child_number,IS_BIND_AWARE, IS_BIND_SENSITIVE,
2> executions, buffer_gets from v$sql where sql_id= :sql_id;
```

SQL_ID	CHILD_NUMBER	I	I	EXECUTIONS	BUFFER_GETS
gnacnx2xb9pxj	0	N	Y	1	13

Id	Operation	Name	Rows	Bytes	Cost
0	SELECT STATEMENT				8
1	TABLE ACCESS BY INDEX ROWID	VIEL	19	38076	8
* 2	INDEX RANGE SCAN	IX_VIEL	19		1

Ausführungspläne

Adaptive Cursor Sharing – Lauf 2

```
SQL> exec :id := 10;  
SQL> select /*TEST*/ * from viel where nummer = :id;
```

980 Zeilen ausgewählt.

```
SQL> select sql_id ,child_number,IS_BIND_AWARE, IS_BIND_SENSITIVE,  
2> executions, buffer_gets from v$sql where sql_id= :sql_id;
```

SQL_ID	CHILD_NUMBER	I	I	EXECUTIONS	BUFFER_GETS
gnacnx2xb9pxj	0	N	Y	2	475

Cursor bleibt unverändert → Ausreißer und Sondereffekte ausschließen

Ausführungspläne

Adaptive Cursor Sharing – Lauf 3

```
SQL> exec :id := 10;
SQL> select /*TEST*/ * from viel where nummer = :id;
```

980 Zeilen ausgewählt.

```
SQL> select sql_id ,child_number,IS_BIND_AWARE, IS_BIND_SENSITIVE,
2> executions, buffer_gets from v$sql where sql_id= :sql_id;
```

SQL_ID	CHILD_NUMBER	I	I	EXECUTIONS	BUFFER_GETS
gnacnx2xb9pxj	0	N	Y	2	475
gnacnx2xb9pxj	1	Y	Y	1	403

Neuer Child Cursor als „is_bind_aware“

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT				96 (100)	
* 1	TABLE ACCESS FULL	VIEL	980	1917K	96 (0)	00:00:02

