Gute Datenbankperformance ist kein Zufall!

Johannes Ahrends

… über mich

- Oracle Spezialist seit 1992
  - 1992: Presales bei Oracle in Düsseldorf
  - 1999: Projektleiter bei Herrmann & Lenz Services GmbH
  - 2005: Technischer Direktor ADM Presales bei Quest Software GmbH
  - 2011: Geschäftsführer CarajanDB GmbH
- 2011 → Ernennung zum Oracle ACE
- Autor der Bücher:
  - Oracle9i für den DBA, Oracle10g für den DBA, Oracle 11g Release 2 für den DBA

- DOAG Themenverantwortlicher Datenbankadministration, Standard Edition
- Hobbies:
  - Drachen steigen lassen (Kiting) draußen wie drinnen (Indoorkiting)
  - Motorradfahren (nur draußen)

Oracle Software Preiskalkulation

Enterprise Edition

- Robuste Datenbank
- Hochverfügbarkeit durch Real Application Clusters
- Desaster Recovery mit Data Guard
- Extreme DWH-Performance durch inMemory
- Sicherheit durch Daten Verschlüsselung und Maskierung (Redaction)
- Umfangreiche Diagnose- / Tuningmöglichkeiten durch Oracle Enterprise Manager
- Optimale Ressourcennutzung durch Multitenant Database
- …

… aber?

- HP DL 380 G10 2 x Intel Xeon Gold 6130 16 Kerne

Funktion	Prozessorpreis	Serverpreis
Enterprise Edition	47.500,00	380.000,00
inMemory Option	23.000,00	184.000,00
Advanced Security Option	15.000,00	120.000,00
Diagnostic und Tuning Pack	12.500,00	100.000,00
Multitenant Database Option	17.500,00	140.000,00
Gesamt	**115.500,00**	**924.000,00**

Fehlt da nicht etwas?

- Hochverfügbarkeit + Desaster Recovery = Server * 4
  - 4 x HP DL 380 G10 2 x Intel Xeon Gold 6130 16 Kerne

Funktion	Prozessorpreis	Serverpreis	4 Server
Enterprise Edition	47.500,00	380.000,00	1.520.000,00
inMemory Option	23.000,00	184.000,00	736.000,00
Advanced Security Option	15.000,00	120.000,00	480.000,00
Diagnostic und Tuning Pack	12.500,00	100.000,00	400.000,00
Multitenant Database Option	17.500,00	140.000,00	560.000,00
Gesamt	**115.500,00**	**924.000,00**	**3.696.000,00**

Da fehlt immer noch etwas!

- Hochverfügbarkeit + Desaster Recovery = Server * 4
  - 4 x HP DL 380 G10 2 x Intel Xeon Gold 6130 16 Kerne

Funktion	Prozessorpreis	Serverpreis	4 Server
Enterprise Edition	47.500,00	380.000,00	1.520.000,00
inMemory Option	23.000,00	184.000,00	736.000,00
Advanced Security Option	15.000,00	120.000,00	480.000,00
Diagnostic und Tuning Pack	12.500,00	100.000,00	400.000,00
Multitenant Database Option	17.500,00	140.000,00	560.000,00
Real Application Clusters	23.000,00	184.000,00	736.000,00
Active Data Guard	10.000,00	80.000,00	320.000,00
Gesamtsumme	**148.500,00**	**1.188.000,00**	**4.752.000,00**

und dann?

- Software Update License & Support = 22%
- ➔ 4.752.000,00 * 22% = 406.560,00 (pro Jahr!)

Rundherum glücklich?

- Partitioning Option $ 17.500,00
- Real Application Testing $ 11.500,00
- Advanced Compression $ 11.500,00
- Database Vault $ 11.500,00
- Spatial and Graph $ 17.500,00
- Data Masking and Subsetting Pack $ 11.500,00
- Database Lifecycle Management Pack $ 12.000,00

- Zusätzlich $ 2.976.000,00 Lizenzpreis + $ 595.000,00 Support pro Jahr

es fehlt immer noch etwas!

- Mehrere Stages
  - Entwicklung
  - Test
  - Vorproduktion
- Okay, kleinere Server (außer Vorproduktion!!!)
- Okay kein RAC für Test
  - → Wer testet den Failover?
  - → Was ist mit Sequences bei RAC?

Gesamtrechnung

- 4 * 8 Prozessoren Produktion
- 4 * 8 Prozessoren Vorproduktion
- 4 * 4 Prozessoren Test
- 2 * 4 Prozessoren Entwicklung

  ➔ 88 Prozessoren
- Lizenzpreis ohne besondere Optionen = 88 * $ 148.500,00 = $ 13.068.000,00
- Preisnachlass 75% ➔ $ 3.267.000,00
- Supportkosten ➔ $ 718.740,00
- Lizenzpreis mit besonderen Optionen = $ 21.252.000,00

Performanceoptimierung

CARAJAN DB

Warum ist eine Anwendung langsam?

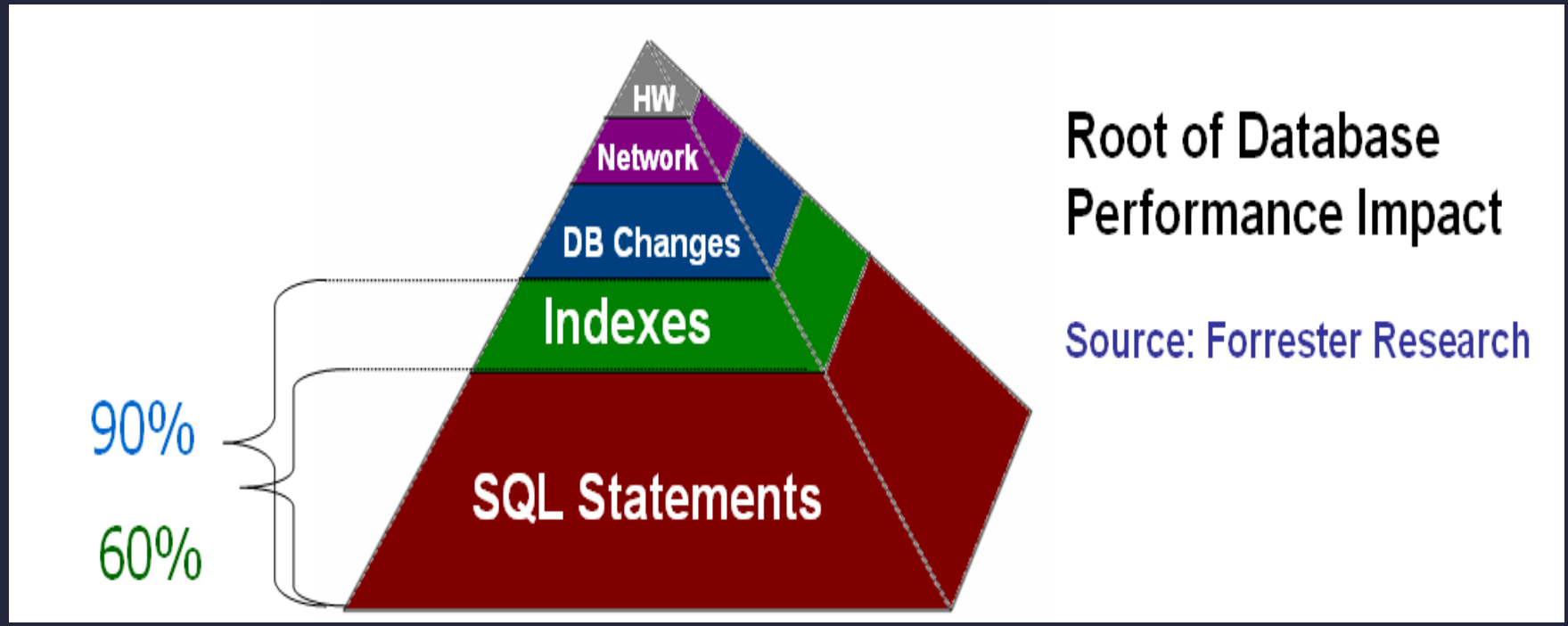
- Ungünstig programmiert
  - Verarbeitung zu aufwändig
  - Es werden zu viele Daten gelesen
- Datenverarbeitung zu langsam
  - Datenselektion zu aufwändig
  - Änderung der Daten zu kompliziert
- Datenbank zu langsam
  - Unterdimensionierte Hardware
  - Zu viel Last
  - Falsch konfigurierte Datenbank

Häufigster Grund für Klagen?

- Subjektive Ursachen
  - Schlechtes Wetter
  - Persönliche Probleme
  - Neue Softwareversion „Früher war alles besser“
- Objektive Ursachen
  - Datenmenge hat zugenommen
  - Fehler durch Batchverarbeitung
  - Falscher Ausführungsplan
  - Falsche Statistiken

Ursachen von Performanceproblemen

- Forrester Research



Aktuelles Beispiel

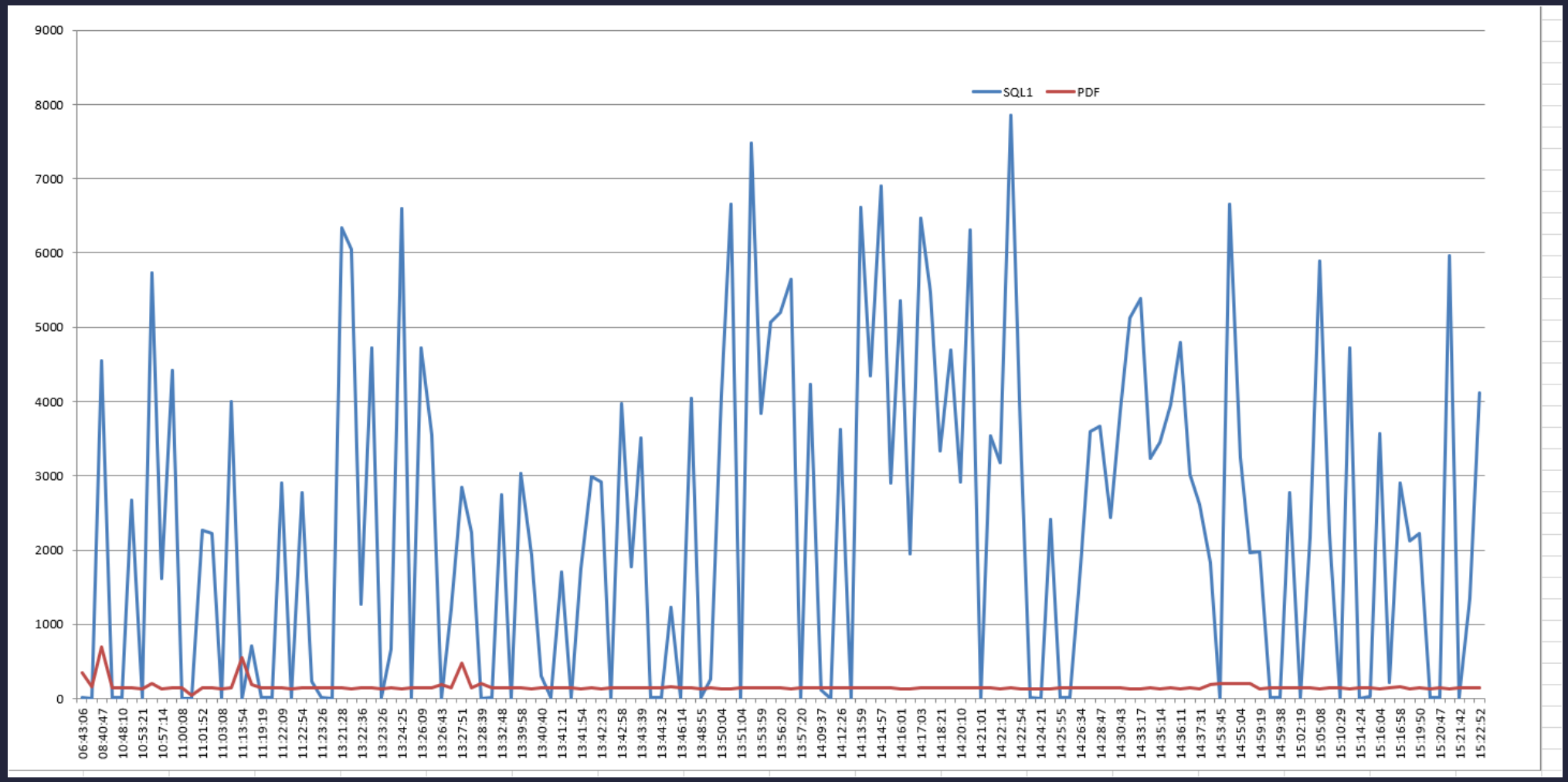
Finanzdienstleister

- Oracle 12.1.0.1 Standard Edition
- Oracle Real Application Clusters für Hochverfügbarkeit und Desaster Recovery
  - ASM Mirroring
  - 2 Rechenzentren
  - Je ein Knoten pro RZ
- kritsche Anwendung hat Antwortzeiten bis 35 Sekunden

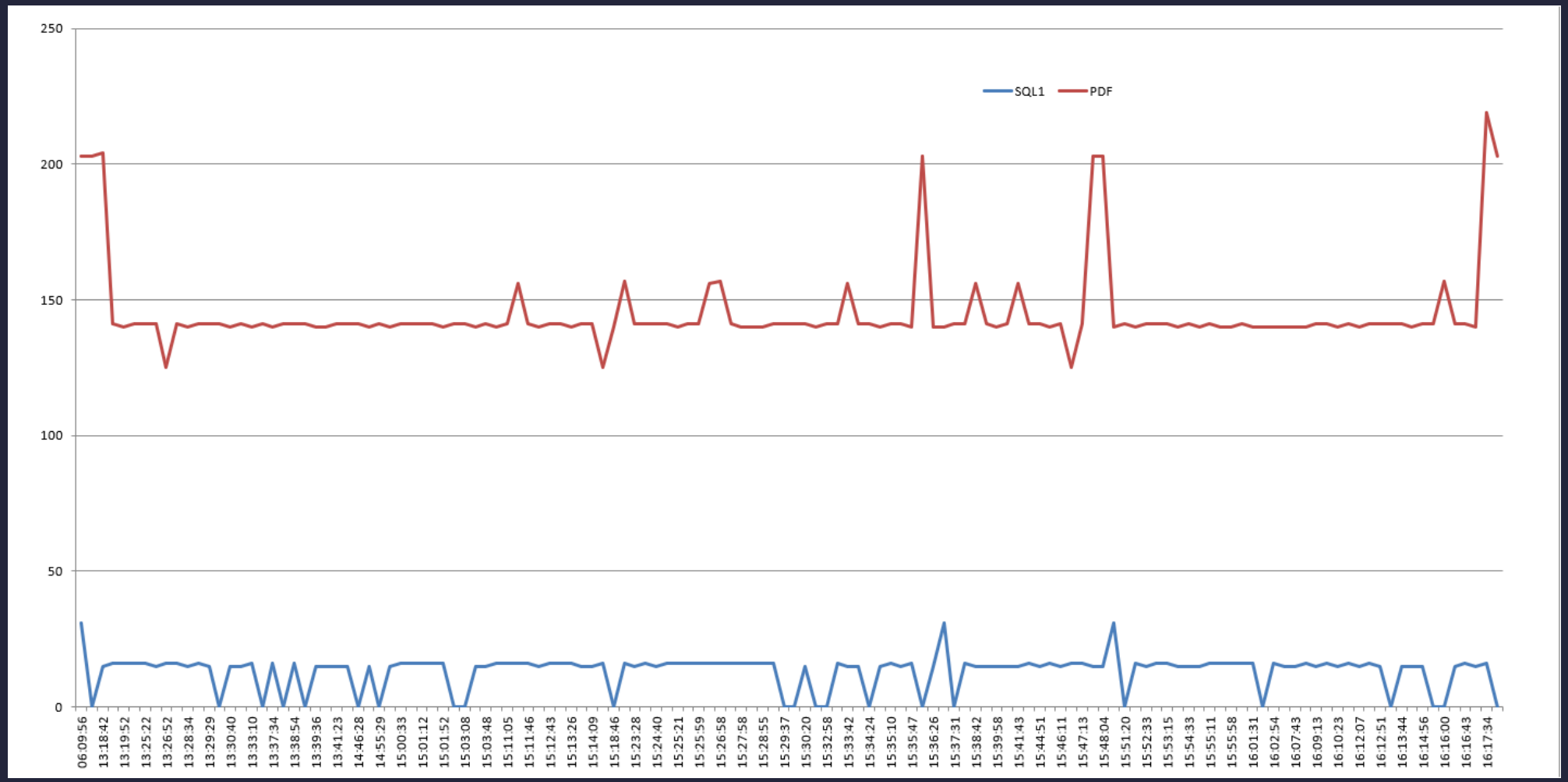
Status

- Oracle 12.1.0.1 End of Support 31. August 2016
- Anwendung läuft nur auf einem Knoten
- Latenzen bei Zugriff von Storage über zwei RZ zu erwarten

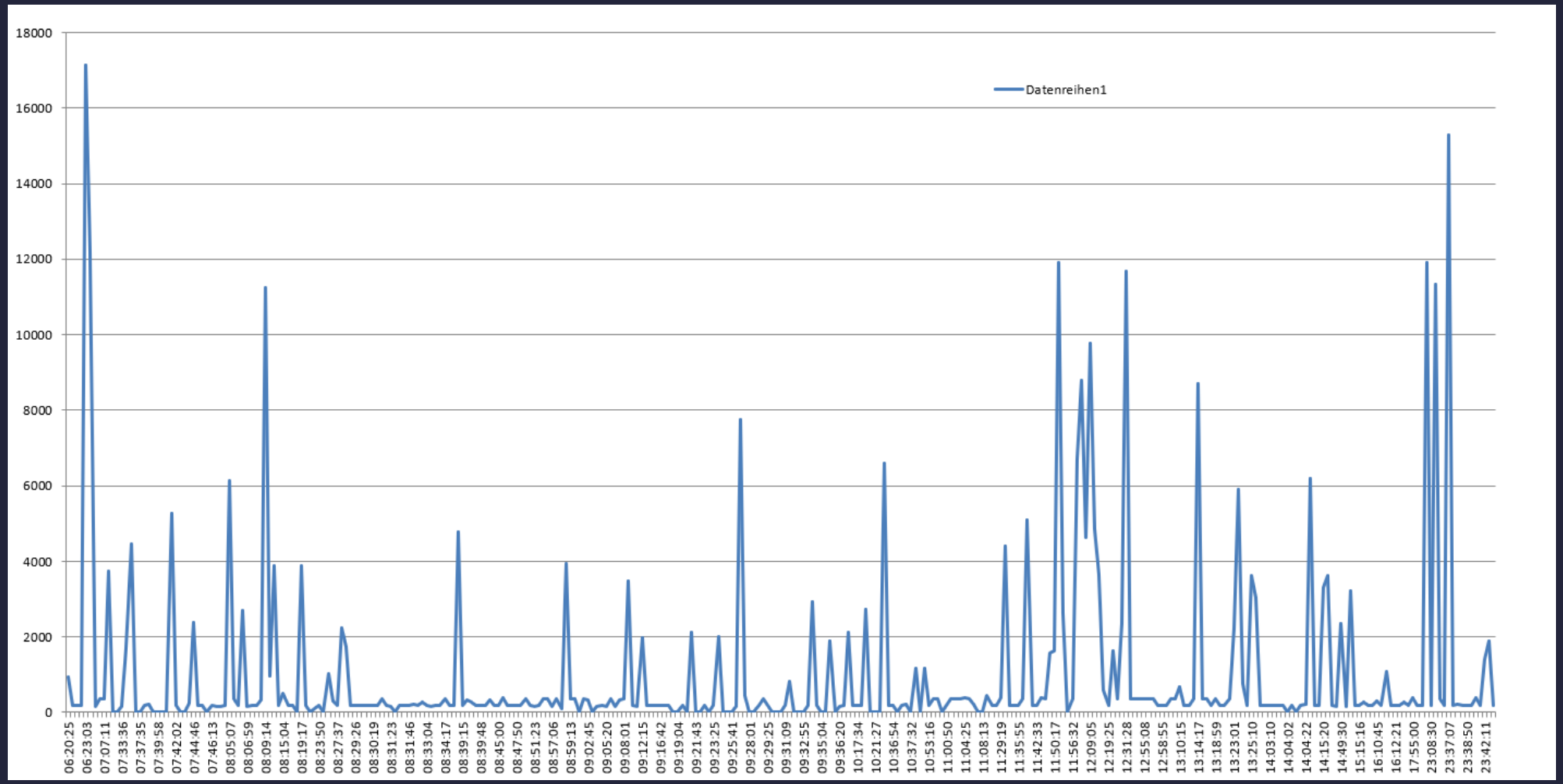
Anwendung 1 - 04.10.2018



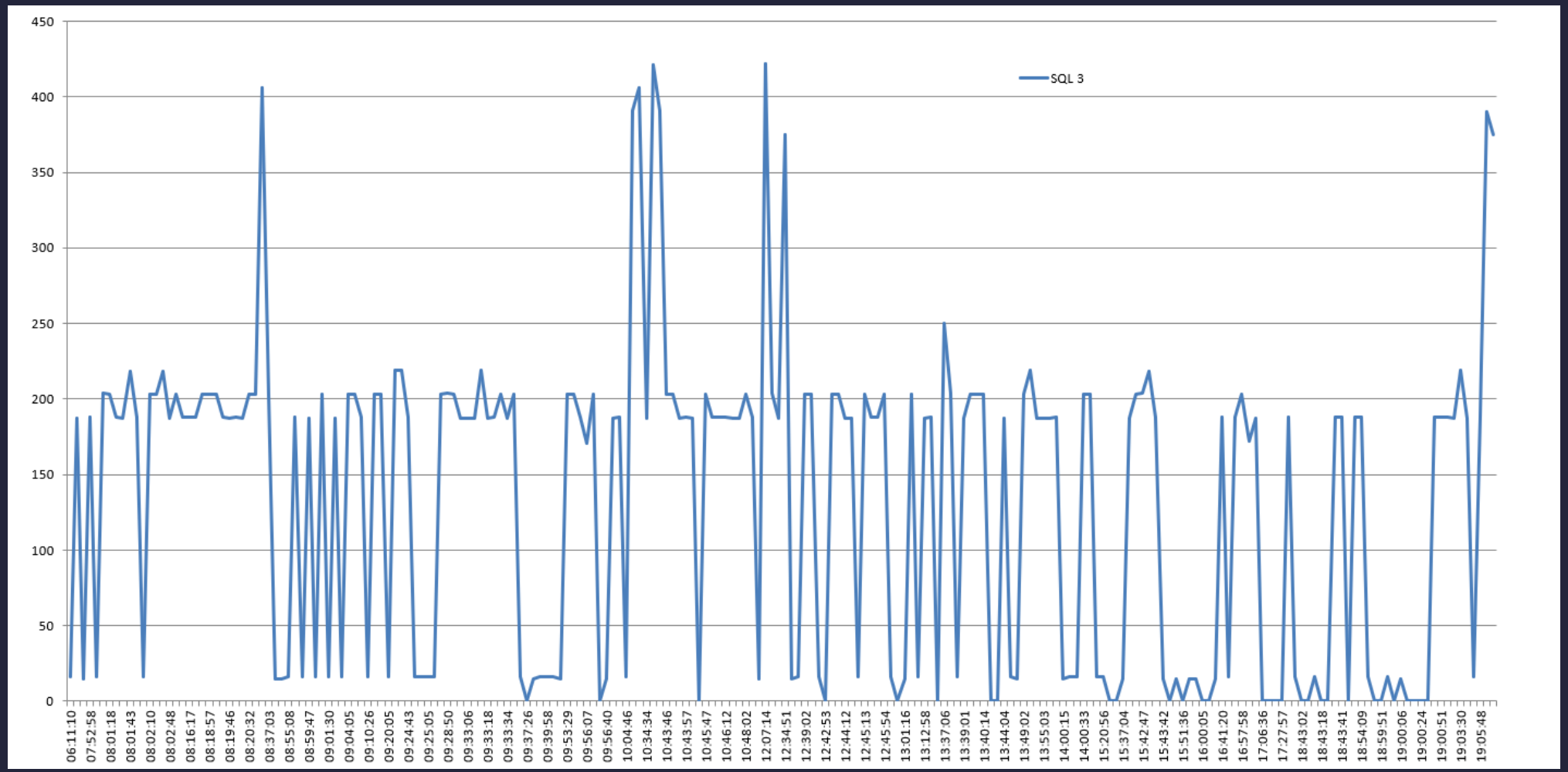
Anwendung 1 - 17.12.2018



Anwendung 2 – 17.10.2018



Anwendung 2 – 19.12.2018



Lösung

- Grund: Netzwerklatenzen
  - Tool: Oracle Statspack
- Lösung: Datenbankinstanz auf Knoten 2 gestoppt!

Indizierung

Einfache Abfrage

- Index nur auf den Primary Key (PK)

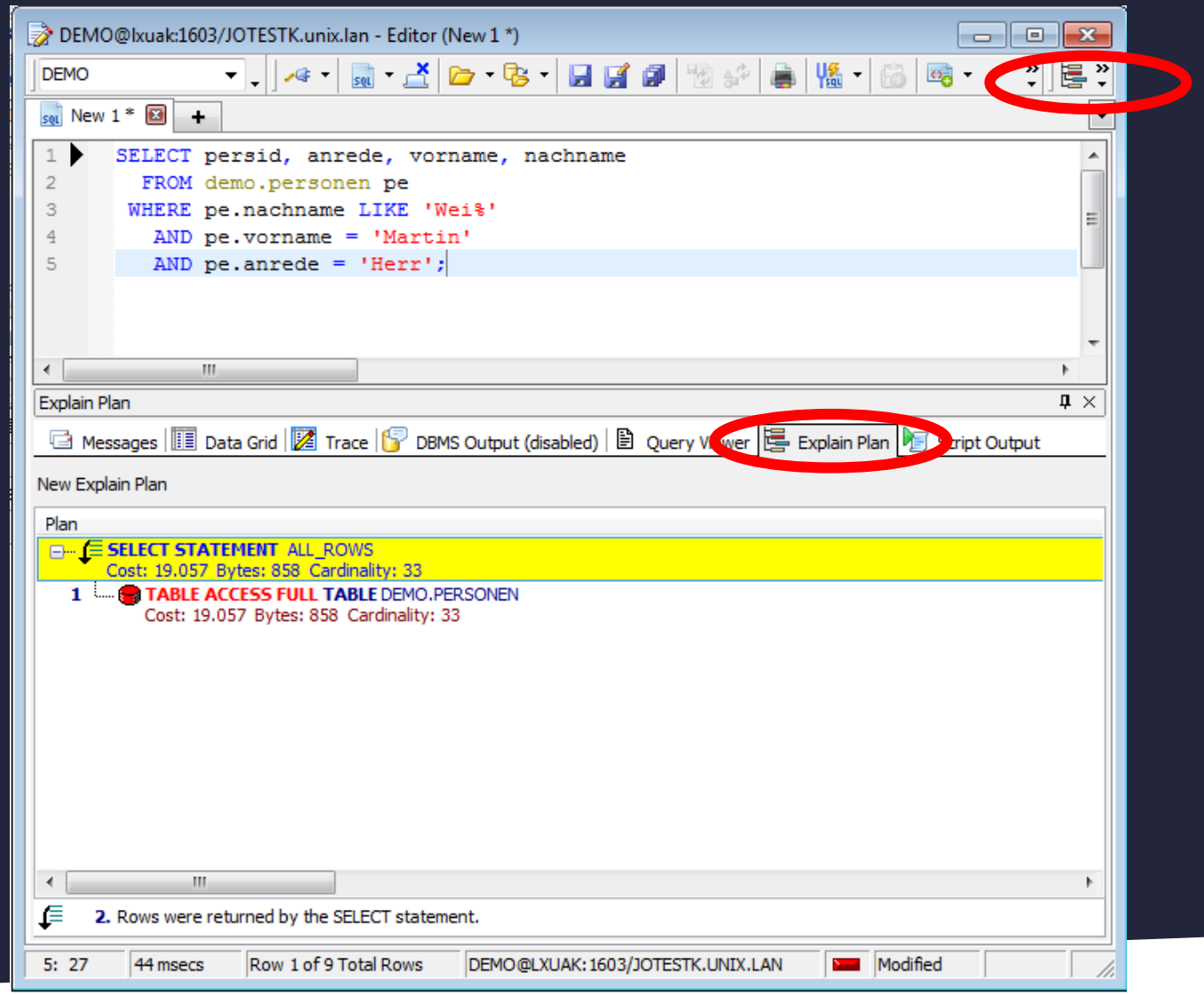
```
SELECT persid, anrede, vorname, nachname
  FROM demo.personen pe
 WHERE pe.nachname = 'Weiß'
   AND pe.vorname = 'Martin'
   AND pe.anrede = 'Herr';
```

CARAJAN DB

Einfache Abfrage

```
-------------------------------------------------------------------------------
| Id  | Operation          | Name     | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |          |    33 |   858 | 19057   (2)| 00:00:01 |
|*  1 |  TABLE ACCESS FULL| PERSONEN |    33 |   858 | 19057   (2)| 00:00:01 |
-------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   1 - filter("PE"."VORNAME"='Martin' AND "PE"."ANREDE"='Herr' AND
              "PE"."NACHNAME" LIKE 'Wei%')

Note
-----
   - dynamic statistics used: dynamic sampling (level=2)
   - 1 Sql Plan Directive used for this statement


Statistics
----------------------------------------------------------
        157  recursive calls
          0  db block gets
      26379  consistent gets
          0  physical reads
          0  redo size
        963  bytes sent via SQL*Net to client
        552  bytes received via SQL*Net from client
          2  SQL*Net roundtrips to/from client
         11  sorts (memory)
          0  sorts (disk)
          9  rows processed
```

Explain Plan im Toad



Mögliche Indizierungen

```
SELECT persid, anrede, vorname, nachname
  FROM demo.personen pe
 WHERE pe.nachname = 'Weiß'
   AND pe.vorname = 'Martin'
   AND pe.anrede = 'Herr';
```

- VORNAME
- NACHNAME
- ANREDE
- VORNAME, NACHNAME
- NACHNAME, VORNAME
- ANREDE, VORNAME, NACHNAME
- NACHNAME, VORNAME, ANREDE

Indizierung 1

- Indizes auf einzelnen Spalten

```
CREATE INDEX idx_nachname
   ON personen (nachname);
CREATE INDEX idx_vorname
   ON personen (vorname);
```

```
SELECT persid, anrede, vorname, nachname
  FROM demo.personen pe
 WHERE pe.nachname = 'Weiß'
   AND pe.vorname = 'Martin'
   AND pe.anrede = 'Herr';
```

Indizierung 1

```
-----------------------------------------------------------------------------------------------
| Id  | Operation                            | Name         | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                     |              |     1 |    26 |    57   (4)| 00:00:01 |
|*  1 |  TABLE ACCESS BY INDEX ROWID BATCHED| PERSONEN     |     1 |    26 |    57   (4)| 00:00:01 |
|   2 |   BITMAP CONVERSION TO ROWIDS        |              |       |       |            |          |
|   3 |    BITMAP AND                        |              |       |       |            |          |
|   4 |     BITMAP CONVERSION FROM ROWIDS    |              |       |       |            |          |
|*  5 |      INDEX RANGE SCAN                | IDX_VORNAME  |   593 |       |     4   (0)| 00:00:01 |
|   6 |     BITMAP CONVERSION FROM ROWIDS    |              |       |       |            |          |
|*  7 |      INDEX RANGE SCAN                | IDX_NACHNAME |   593 |       |    52   (2)| 00:00:01 |
-----------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   1 - filter("PE"."ANREDE"='Herr')
   5 - access("PE"."VORNAME"='Martin')
   7 - access("PE"."NACHNAME"='Weiß')

Statistics
----------------------------------------------------------
          7  recursive calls
          0  db block gets
         28  consistent gets
          7  physical reads
          0  redo size
        841  bytes sent via SQL*Net to client
        552  bytes received via SQL*Net from client
          2  SQL*Net roundtrips to/from client
          1  sorts (memory)
          0  sorts (disk)
          2  rows processed
```

Indizierung 1 – aber ….

- Indizes auf einzelnen Spalten

```
CREATE INDEX idx_nachname
   ON personen (nachname);
CREATE INDEX idx_vorname
   ON personen (vorname);
```

```
SELECT persid, anrede, vorname, nachname
  FROM demo.personen pe
 WHERE pe.nachname like 'Wei_'
   AND pe.vorname = 'Martin'
   AND pe.anrede = 'Herr';
```

Wahrscheinlicher: einer der beiden Indizes

```
-------------------------------------------------------------------------------------------------
| Id  | Operation                             | Name         | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                      |              |    33 |   858 |   591   (1)| 00:00:01 |
|*  1 |  TABLE ACCESS BY INDEX ROWID BATCHED| PERSONEN     |    33 |   858 |   591   (1)| 00:00:01 |
|*  2 |   INDEX RANGE SCAN                    | IDX_VORNAME |   593 |       |     4   (0)| 00:00:01 |
-------------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   1 - filter("PE"."ANREDE"='Herr' AND "PE"."NACHNAME" LIKE 'Wei_')
   2 - access("PE"."VORNAME"='Martin')

Note
-----
   - dynamic statistics used: dynamic sampling (level=2)
   - 1 Sql Plan Directive used for this statement


Statistics
----------------------------------------------------------
         12  recursive calls
          0  db block gets
        591  consistent gets
        550  physical reads
          0  redo size
        841  bytes sent via SQL*Net to client
        552  bytes received via SQL*Net from client
          2  SQL*Net roundtrips to/from client
          1  sorts (memory)
          0  sorts (disk)
          2  rows processed
```

Indizierung 2:

- Zusammengesetzter Index

```
CREATE INDEX idx_name
   ON personen (anrede, vorname, nachname);
```

```
SELECT persid, anrede, vorname, nachname
  FROM demo.personen pe
 WHERE pe.nachname = 'Weiß'
   AND pe.vorname = 'Martin'
   AND pe.anrede = 'Herr';
```

Indizierung 2

```
-------------------------------------------------------------------------------------------------
| Id  | Operation                             | Name     | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                      |          |     1 |    26 |     4   (0)| 00:00:01 |
|   1 |  TABLE ACCESS BY INDEX ROWID BATCHED| PERSONEN |     1 |    26 |     4   (0)| 00:00:01 |
|*  2 |   INDEX RANGE SCAN                    | IDX_NAME |     1 |       |     3   (0)| 00:00:01 |
-------------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   2 - access("PE"."ANREDE"='Herr' AND "PE"."VORNAME"='Martin' AND
              "PE"."NACHNAME"='Weiß')


Statistics
----------------------------------------------------------
          7  recursive calls
          0  db block gets
         25  consistent gets
          2  physical reads
          0  redo size
        841  bytes sent via SQL*Net to client
        552  bytes received via SQL*Net from client
          2  SQL*Net roundtrips to/from client
          1  sorts (memory)
          0  sorts (disk)
          2  rows processed
```

CARAJAN DB

„Anrede“ fehlt

```
CREATE INDEX idx_name
   ON personen (anrede, vorname, nachname);
```

```
SELECT persid, anrede, vorname, nachname
  FROM demo.personen pe
 WHERE pe.nachname ='Weiß'
   AND pe.vorname = 'Martin';
```

Index Skip Scan

```
-------------------------------------------------------------------------------------------------
| Id  | Operation                             | Name     | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                      |          |     2 |    52 |     7   (0)| 00:00:01 |
|   1 |  TABLE ACCESS BY INDEX ROWID BATCHED| PERSONEN |     2 |    52 |     7   (0)| 00:00:01 |
|*  2 |   INDEX SKIP SCAN                     | IDX_NAME |     2 |       |     4   (0)| 00:00:01 |
-------------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   2 - access("PE"."VORNAME"='Martin' AND "PE"."NACHNAME"='Weiß')
       filter("PE"."VORNAME"='Martin' AND "PE"."NACHNAME"='Weiß')


Statistics
----------------------------------------------------------
          7  recursive calls
          0  db block gets
         35  consistent gets
          8  physical reads
          0  redo size
        841  bytes sent via SQL*Net to client
        552  bytes received via SQL*Net from client
          2  SQL*Net roundtrips to/from client
          1  sorts (memory)
          0  sorts (disk)
          2  rows processed
```

„Anrede“ und „Vorname“ fehlen

```
CREATE INDEX idx_name
   ON personen (anrede, vorname, nachname);
```

```
SELECT persid, anrede, vorname, nachname
  FROM demo.personen pe
 WHERE pe.nachname ='Weiß';
```

Full Table Scan!

```
-------------------------------------------------------------------------------
| Id  | Operation         | Name     | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT  |          | 19042 |   483K| 19057   (2)| 00:00:01 |
|*  1 |  TABLE ACCESS FULL| PERSONEN | 19042 |   483K| 19057   (2)| 00:00:01 |
-------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   1 - filter("PE"."NACHNAME"='Weiß')


Statistics
----------------------------------------------------------
          7  recursive calls
          0  db block gets
      26319  consistent gets
      26288  physical reads
          0  redo size
        841  bytes sent via SQL*Net to client
        552  bytes received via SQL*Net from client
          2  SQL*Net roundtrips to/from client
          1  sorts (memory)
          0  sorts (disk)
          2  rows processed
```

Indizierung 3:

- Zusammengesetzter Index

```
CREATE INDEX idx_name
   ON personen (nachname,vorname,anrede);
```

Indizierung 3

```
-------------------------------------------------------------------------------------------------
| Id  | Operation                            | Name     | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                     |          |     1 |    26 |     4   (0)| 00:00:01 |
|   1 |  TABLE ACCESS BY INDEX ROWID BATCHED| PERSONEN |     1 |    26 |     4   (0)| 00:00:01 |
|*  2 |   INDEX RANGE SCAN                   | IDX_NAME |     1 |       |     3   (0)| 00:00:01 |
-------------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   2 - access("PE"."NACHNAME"='Weiß' AND "PE"."VORNAME"='Martin' AND
              "PE"."ANREDE"='Herr')


Statistics
----------------------------------------------------------
          7  recursive calls
          0  db block gets
         25  consistent gets
          2  physical reads
          0  redo size
        841  bytes sent via SQL*Net to client
        552  bytes received via SQL*Net from client
          2  SQL*Net roundtrips to/from client
          1  sorts (memory)
          0  sorts (disk)
          2  rows processed
```

„Anrede“ fehlt

```
CREATE INDEX idx_name
   ON personen (nachname,vorname,anrede);
```

```
SELECT persid, anrede, vorname, nachname
  FROM demo.personen pe
 WHERE pe.nachname ='Weiß'
   AND pe.vorname = 'Martin';
```

Index Range Scan

```
-------------------------------------------------------------------------------------------------
| Id  | Operation                                | Name     | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                         |          |     2 |    52 |     6   (0)| 00:00:01 |
|   1 |  TABLE ACCESS BY INDEX ROWID BATCHED| PERSONEN |     2 |    52 |     6   (0)| 00:00:01 |
|*  2 |   INDEX RANGE SCAN                       | IDX_NAME |     2 |       |     3   (0)| 00:00:01 |
-------------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   2 - access("PE"."NACHNAME"='Weiß' AND "PE"."VORNAME"='Martin')


Statistics
----------------------------------------------------------
          7  recursive calls
          0  db block gets
         25  consistent gets
          0  physical reads
          0  redo size
        841  bytes sent via SQL*Net to client
        552  bytes received via SQL*Net from client
          2  SQL*Net roundtrips to/from client
          1  sorts (memory)
          0  sorts (disk)
          2  rows processed
```

„Anrede“ und „Vorname“ fehlen

```
CREATE INDEX idx_name
   ON personen (nachname,vorname,anrede);
```

```
SELECT persid, anrede, vorname, nachname
  FROM demo.personen pe
 WHERE pe.nachname ='Weiß';
```

Index Join!

```
-------------------------------------------------------------------------------------------
| Id  | Operation              | Name             | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT       |                  | 19042 |   483K| 10484   (3)| 00:00:01 |
|*  1 |  VIEW                  | index$_join$_001 | 19042 |   483K| 10484   (3)| 00:00:01 |
|*  2 |   HASH JOIN            |                  |       |       |            |          |
|*  3 |    INDEX RANGE SCAN    | IDX_NAME         | 19042 |   483K|    84   (2)| 00:00:01 |
|   4 |    INDEX FAST FULL SCAN| PK_PERSONEN      | 19042 |   483K| 12893   (2)| 00:00:01 |
-------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   1 - filter("PE"."NACHNAME"='Weiß')
   2 - access(ROWID=ROWID)
   3 - access("PE"."NACHNAME"='Weiß')


Statistics
----------------------------------------------------------
          7  recursive calls
          0  db block gets
      10151  consistent gets
      10098  physical reads
          0  redo size
        841  bytes sent via SQL*Net to client
        552  bytes received via SQL*Net from client
          2  SQL*Net roundtrips to/from client
          1  sorts (memory)
          0  sorts (disk)
          2  rows processed
```

Indizierung 4 –unscharfe Suche

- Zusammengesetzter Index

```
CREATE INDEX idx_name
   ON personen (nachname,vorname,anrede);
```

```
SELECT persid, anrede, vorname, nachname
  FROM demo.personen pe
 WHERE pe.nachname like 'Wei_'
   AND pe.vorname = 'Martin'
   AND pe.anrede = 'Herr';
```

Unscharfe Suche

```
-------------------------------------------------------------------------------------------
| Id  | Operation                            | Name     | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                     |          |    33 |   858 |     5   (0)| 00:00:01 |
|   1 |  TABLE ACCESS BY INDEX ROWID BATCHED| PERSONEN |    33 |   858 |     5   (0)| 00:00:01 |
|*  2 |   INDEX RANGE SCAN                   | IDX_NAME |     2 |       |     3   (0)| 00:00:01 |
-------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   2 - access("PE"."NACHNAME" LIKE 'Wei_' AND "PE"."VORNAME"='Martin' AND "PE"."ANREDE"='Herr')
       filter("PE"."VORNAME"='Martin' AND "PE"."ANREDE"='Herr' AND "PE"."NACHNAME" LIKE 'Wei_')

Note
-----
   - dynamic statistics used: dynamic sampling (level=2)
   - 1 Sql Plan Directive used for this statement


Statistics
----------------------------------------------------------
          0  recursive calls
          0  db block gets
        217  consistent gets
          0  physical reads
          0  redo size
        841  bytes sent via SQL*Net to client
        552  bytes received via SQL*Net from client
          2  SQL*Net roundtrips to/from client
          0  sorts (memory)
          0  sorts (disk)
          2  rows processed
```

„Anrede“ fehlt

```
CREATE INDEX idx_name
   ON personen (nachname,vorname,anrede);
```

```
SELECT persid, anrede, vorname, nachname
  FROM demo.personen pe
 WHERE pe.nachname like 'Wei_'
   AND pe.vorname = 'Martin';
```

```
-------------------------------------------------------------------------------------------
| Id  | Operation                            | Name     | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                     |          |    33 |   858 |     5   (0)| 00:00:01 |
|   1 |  TABLE ACCESS BY INDEX ROWID BATCHED| PERSONEN |    33 |   858 |     5   (0)| 00:00:01 |
|*  2 |   INDEX RANGE SCAN                   | IDX_NAME |     2 |       |     3   (0)| 00:00:01 |
-------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   2 - access("PE"."NACHNAME" LIKE 'Wei_' AND "PE"."VORNAME"='Martin')
       filter("PE"."VORNAME"='Martin' AND "PE"."NACHNAME" LIKE 'Wei_')

Note
-----
   - dynamic statistics used: dynamic sampling (level=2)
   - 1 Sql Plan Directive used for this statement


Statistics
----------------------------------------------------------
         19  recursive calls
          0  db block gets
        664  consistent gets
          0  physical reads
          0  redo size
        841  bytes sent via SQL*Net to client
        552  bytes received via SQL*Net from client
          2  SQL*Net roundtrips to/from client
          1  sorts (memory)
          0  sorts (disk)
          2  rows processed
```

„Anrede“ und „Vorname“ fehlen

```
CREATE INDEX idx_name
   ON personen (nachname,vorname,anrede);
```

```
SELECT persid, anrede, vorname, nachname
  FROM demo.personen pe
 WHERE pe.nachname like 'Wei_';
```

```
-------------------------------------------------------------------------------
| Id  | Operation          | Name     | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |          | 55247 |  1402K| 19099   (2)| 00:00:01 |
|*  1 |  TABLE ACCESS FULL| PERSONEN | 55247 |  1402K| 19099   (2)| 00:00:01 |
-------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   1 - filter("PE"."NACHNAME" LIKE 'Wei_')


Statistics
----------------------------------------------------------
          7  recursive calls
          0  db block gets
      26324  consistent gets
      26288  physical reads
          0  redo size
        841  bytes sent via SQL*Net to client
        552  bytes received via SQL*Net from client
          2  SQL*Net roundtrips to/from client
          1  sorts (memory)
          0  sorts (disk)
          2  rows processed
```

Versagen des Index!

Indexversagen

- Wo ist das Problem?

```
SQL> SELECT anrede, vorname, nachname
  2    FROM tuk.personen pe
  3   WHERE pe.nachname LIKE 'wei_'
  4     AND pe.vorname LIKE 'Martin';

ANRED VORNAME              NACHNAME
----- -------------------- --------------------
Herr  Martin               Weiß
Herr  Martin               Weis
Herr  Martin               Weiz

3 Zeilen ausgewählt.
```

Warum?

```
-------------------------------------------------------------------------------
| Id  | Operation         | Name     | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT  |          |     3 |    78 | 19102   (2)| 00:00:01 |
|*  1 |  TABLE ACCESS FULL| PERSONEN |     3 |    78 | 19102   (2)| 00:00:01 |
-------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   1 - filter("PE"."NACHNAME" LIKE 'Wei_' AND
              NLSSORT("PE"."VORNAME",'nls_sort=''BINARY_CI''')=HEXTORAW('6D617274696E00') AND
              NLSSORT("PE"."ANREDE",'nls_sort=''BINARY_CI''')=HEXTORAW('6865727200'))

Statistics
----------------------------------------------------------
         81  recursive calls
          3  db block gets
      26386  consistent gets
      26294  physical reads
        568  redo size
        851  bytes sent via SQL*Net to client
        552  bytes received via SQL*Net from client
          2  SQL*Net roundtrips to/from client
          8  sorts (memory)
          0  sorts (disk)
          3  rows processed
```

Linguistische Suche

- `ALTER SESSION SET nls_sort=binary_ci;`
  - Sortierung ist unabhängig von Groß- / Kleinschreibung aber abhängig von Akzenten
  - Alternativen:
    - binary_ai → Case und Akzent insensitiv
    - german_ai → Deutsche Sortierung, Case und Akzent insensitiv
    - …
- `ALTER SESSION SET nls_comp=linguistic;`
  - Filter verwenden die gleiche Funktion wie `NLS_SORT`, d.h. in diesem Fall ist die WHERE-Clausel unabhängig von Groß- / Kleinschreibung und von Akzenten
  - Alternativen:
    - BINARY oder ANSI

CARAJAN DB

Lösung

- Index auf Linguistische Suche

```
CREATE INDEX idx_name2
   ON personen (NLSSORT (vorname, 'nls_sort=binary_ci'),
                NLSSORT (nachname, 'nls_sort=binary_ci'));
```

- Aber jetzt sind z.B. keine Bitmapped Indizes auf den Spalten möglich
- Anwendungen, die diese Einstellungen nicht verwenden, können den Index nicht benutzen
- Bei unscharfer Suche ist der Index nur sehr eingeschränkt nutzbar
- Besser: Linguistische Suche ausschalten

Linguistische Suche

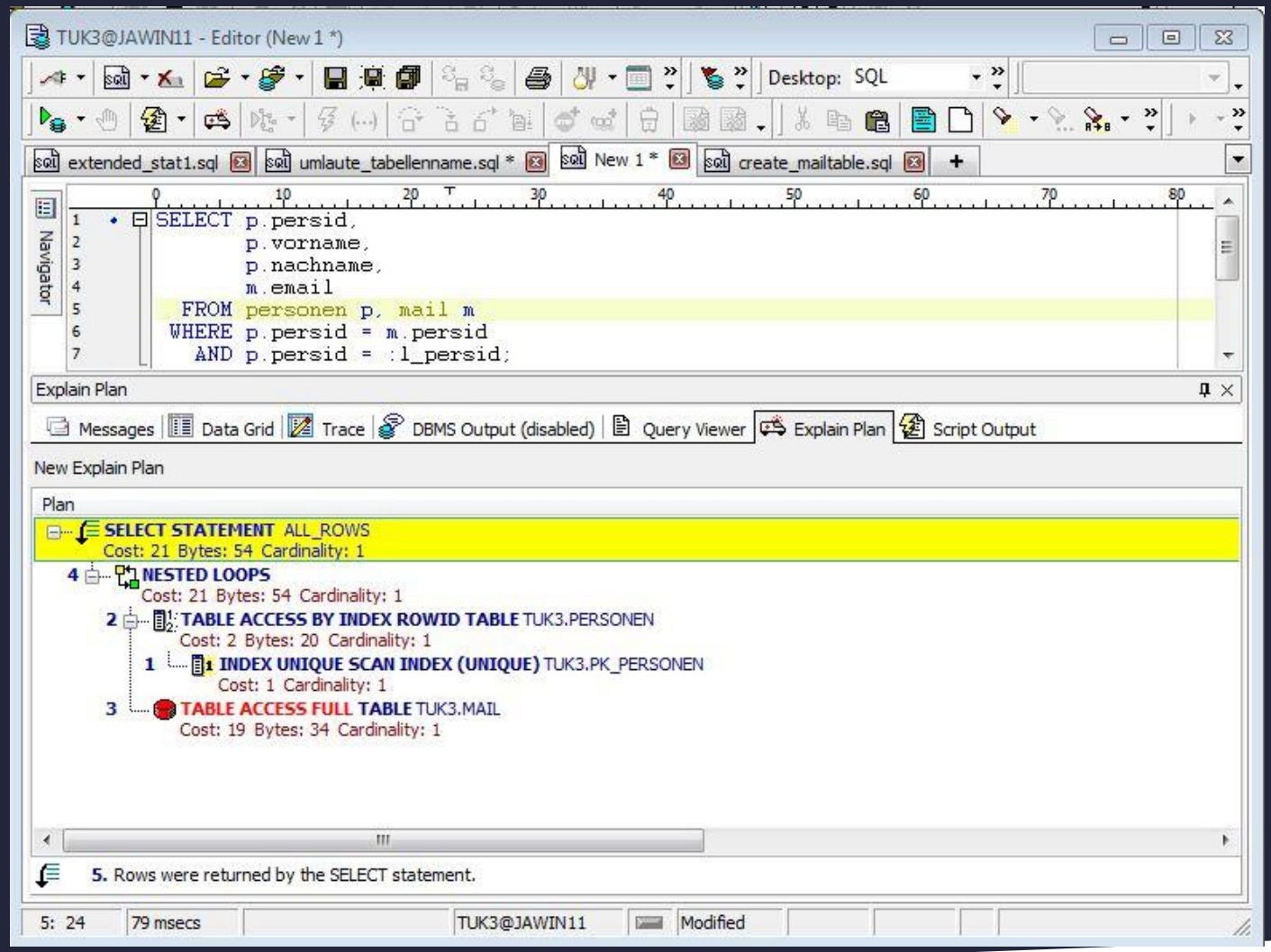
```
-------------------------------------------------------------------------------------------------
| Id  | Operation                             | Name     | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                      |          |   233 | 11417 | 18828   (1)| 00:00:01 |
|*  1 |  TABLE ACCESS BY INDEX ROWID BATCHED| PERSONEN |   233 | 11417 | 18828   (1)| 00:00:01 |
|*  2 |   INDEX RANGE SCAN                    | IDX_NAME | 20000 |       |    80   (2)| 00:00:01 |
-------------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   1 - filter("PE"."NACHNAME" LIKE 'Wei_' AND
              NLSSORT("PE"."ANREDE",'nls_sort=''BINARY_CI''')=HEXTORAW('6865727200'))
   2 - access(NLSSORT("VORNAME",'nls_sort=''BINARY_CI''')=HEXTORAW('6D617274696E00'))


Statistics
----------------------------------------------------------
         16  recursive calls
          0  db block gets
        926  consistent gets
        549  physical reads
          0  redo size
        851  bytes sent via SQL*Net to client
        552  bytes received via SQL*Net from client
          2  SQL*Net roundtrips to/from client
          2  sorts (memory)
          0  sorts (disk)
          3  rows processed
```

Indexversagen

- Beide Tabellen haben *persid* als Primary Key!

```
SELECT p.persid,
       p.vorname,
       p.nachname,
       m.email
  FROM personen p, mail m
 WHERE p.persid = m.persid
   AND p.persid = :1_persid;
```

Warum kein Index Zugriff?



Datentyp Konvertierung

```
-------------------------------------------------------------------------------------------
| Id  | Operation                     | Name          | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT              |               |   605 | 81675 | 39535   (2)| 00:00:02 |
|   1 |  NESTED LOOPS                 |               |   605 | 81675 | 39535   (2)| 00:00:02 |
|   2 |   TABLE ACCESS BY INDEX ROWID| PERSONEN      |     1 |    21 |     3   (0)| 00:00:01 |
|*  3 |    INDEX UNIQUE SCAN          | PK_PERSONEN   |     1 |       |     2   (0)| 00:00:01 |
|*  4 |   TABLE ACCESS FULL           | EMAILADRESSEN |   605 | 68970 | 39532   (2)| 00:00:02 |
-------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   3 - access("P"."PERSID"=12345678)
   4 - filter(TO_NUMBER("M"."PERSID")=12345678)

Statistics
----------------------------------------------------------
       1070  recursive calls
          0  db block gets
       1354  consistent gets
        408  physical reads
        132  redo size
        555  bytes sent via SQL*Net to client
        541  bytes received via SQL*Net from client
          1  SQL*Net roundtrips to/from client
         85  sorts (memory)
          0  sorts (disk)
          0  rows processed
```

Warum kein Index Zugriff?

```
-------------------------------------------------------------------------------------------
| Id  | Operation                     | Name          | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT              |               |   605 | 81675 | 39535   (2)| 00:00:02 |
|   1 |  NESTED LOOPS                 |               |   605 | 81675 | 39535   (2)| 00:00:02 |
|   2 |   TABLE ACCESS BY INDEX ROWID| PERSONEN      |     1 |    21 |     3   (0)| 00:00:01 |
|*  3 |    INDEX UNIQUE SCAN          | PK_PERSONEN   |     1 |       |     2   (0)| 00:00:01 |
|*  4 |   TABLE ACCESS FULL           | EMAILADRESSEN |   605 | 68970 | 39532   (2)| 00:00:02 |
-------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------
   3 - access("P"."PERSID"=12345678)
   4 - filter(TO_NUMBER("M"."PERSID")=12345678)
```

- PERSID ist in der Personen Tabelle als Number und in der Email-Tabelle als VARCHAR2 definiert
- D.h. es muss eine Datentypkonvertierung stattfinden

Abfragen eines Satzes

- Abfragen eines Satzes über den Primary Key

```
SELECT p.persid,
       p.vorname,
       p.nachname
  FROM personen p
 WHERE p.persid = 12345678;
```

Ausführungsplan

```
-------------------------------------------------------------------------------------------
| Id  | Operation                    | Name        | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |             |     1 |    21 |     3   (0)| 00:00:01 |
|   1 |  TABLE ACCESS BY INDEX ROWID| PERSONEN    |     1 |    21 |     3   (0)| 00:00:01 |
|*  2 |   INDEX UNIQUE SCAN          | PK_PERSONEN |     1 |       |     2   (0)| 00:00:01 |
-------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   2 - access("P"."PERSID"=12345678)


Statistics
----------------------------------------------------------
          1  recursive calls
          0  db block gets
          4  consistent gets
          0  physical reads
          0  redo size
        697  bytes sent via SQL*Net to client
        552  bytes received via SQL*Net from client
          2  SQL*Net roundtrips to/from client
          0  sorts (memory)
          0  sorts (disk)
          1  rows processed
```

Löschen eines Satzes

- Versicherungsbranche
- Löschen eines Satzes über den Primary Key

```
DELETE FROM personen p
 WHERE p.persid = 12345678;
```

Ausführungsplan

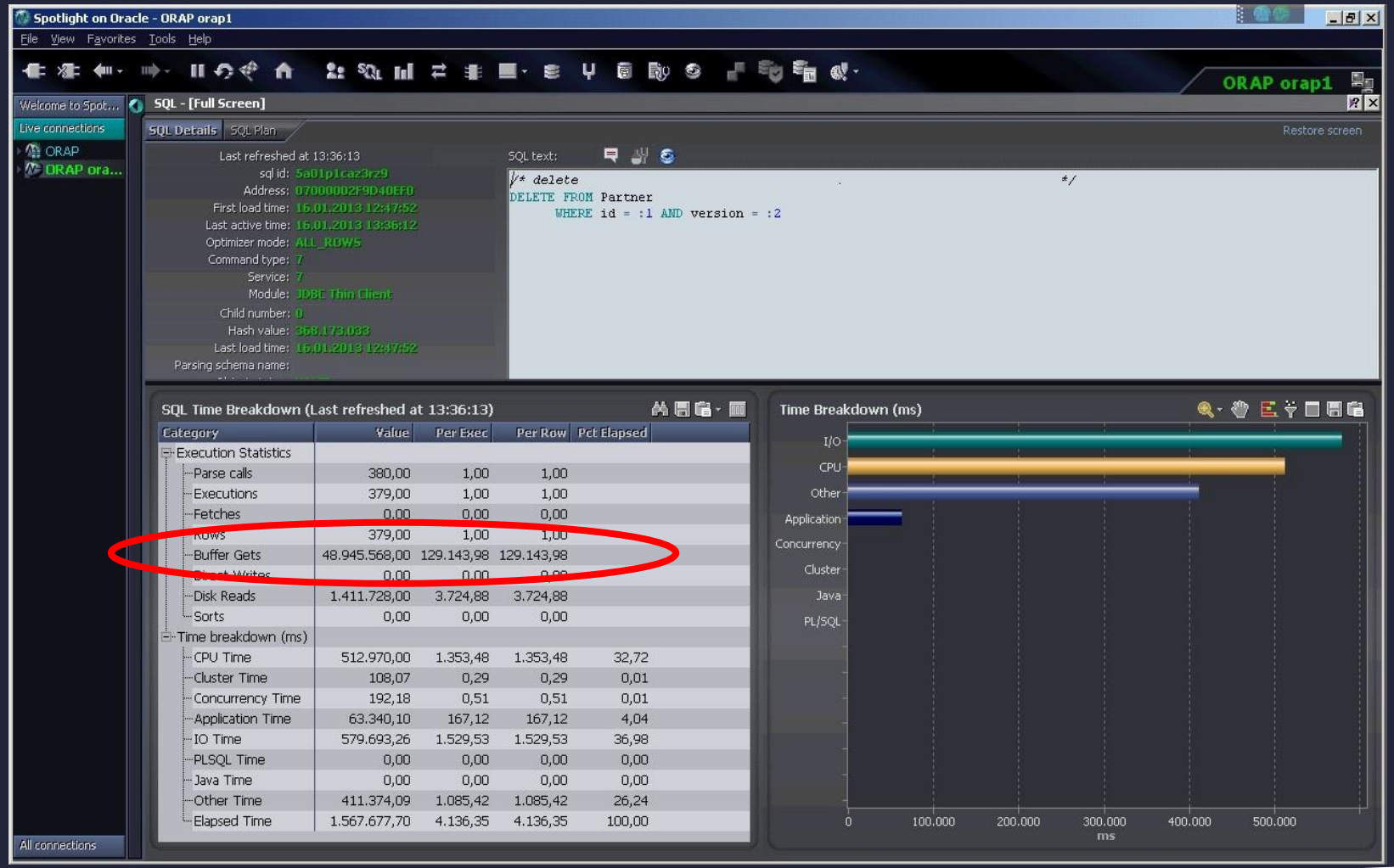
```
-------------------------------------------------------------------------------
| Id  | Operation          | Name        | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | DELETE STATEMENT   |             |     1 |    21 |     2   (0)| 00:00:01 |
|   1 |  DELETE            | PERSONEN    |       |       |            |          |
|*  2 |   INDEX UNIQUE SCAN| PK_PERSONEN |     1 |    21 |     2   (0)| 00:00:01 |
-------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   2 - access("P"."PERSID"=12345678)


Statistics
----------------------------------------------------------
         38  recursive calls
         19  db block gets
     200981  consistent gets
     124756  physical reads
       1084  redo size
        865  bytes sent via SQL*Net to client
        858  bytes received via SQL*Net from client
          3  SQL*Net roundtrips to/from client
          5  sorts (memory)
          0  sorts (disk)
          1  rows processed
```

CARAJANDB

Index auf Foreign Key

- Kein Index:
  - Table Lock auf die Detail Tabelle, wenn Update auf Primary Key der Master Tabelle
- Updates auf den Primary Key sind selten
  - Index auf Foreign Key nicht erforderlich
- Aber:
- DELETE FROM *Master Tabelle*
  ➔ FULL TABLE SCAN auf die *Detail Tabelle*

Index auf Foreign Key



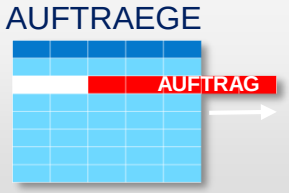
* 129.143,98 * 8192 ~ 1 GByte

In-Memory

Spalten gegen Zeilenorientierung

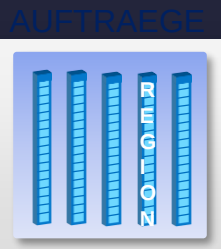
- Optimierung für Transaktionen und Query Performance

Zeile



- **Transaktionen sind schneller bei Zeilenverarbeitung**
  - Schnell bei wenigen Zeilen und vielen Spalten
  - Beispiel: Einfügen oder Abfragen von Aufträgen

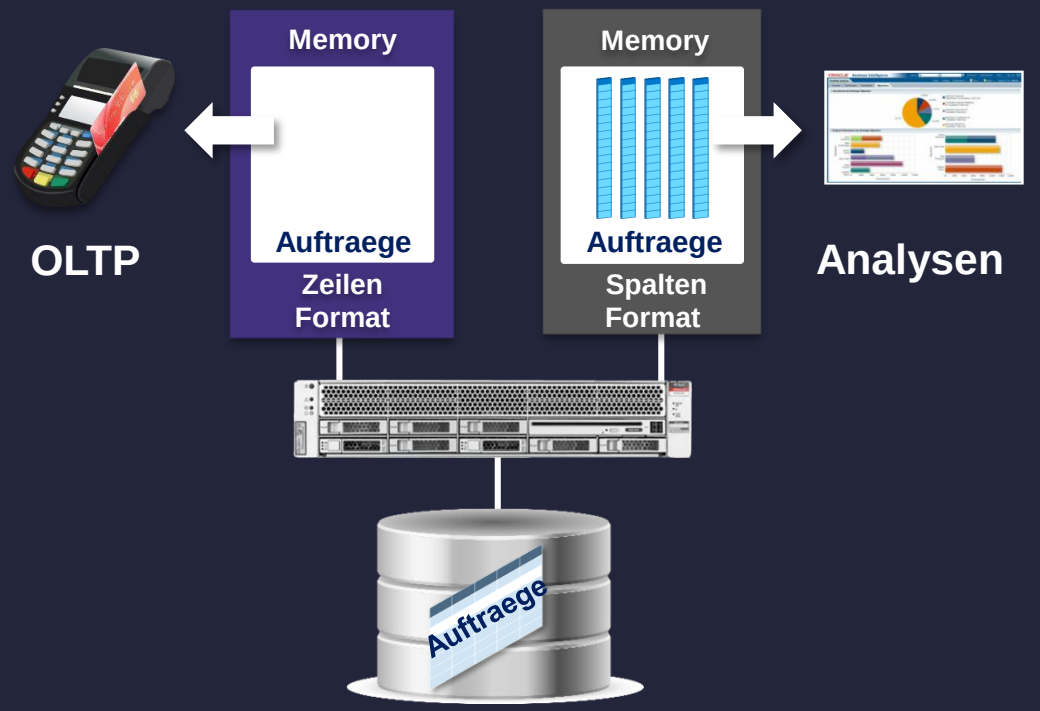
Spalte



- **Analysen sind schneller bei Spaltenorientierung**
  - Schnell bei wenigen Spalten und vielen Zeilen
  - Beispiel: Report für die Auftragssumme pro Region

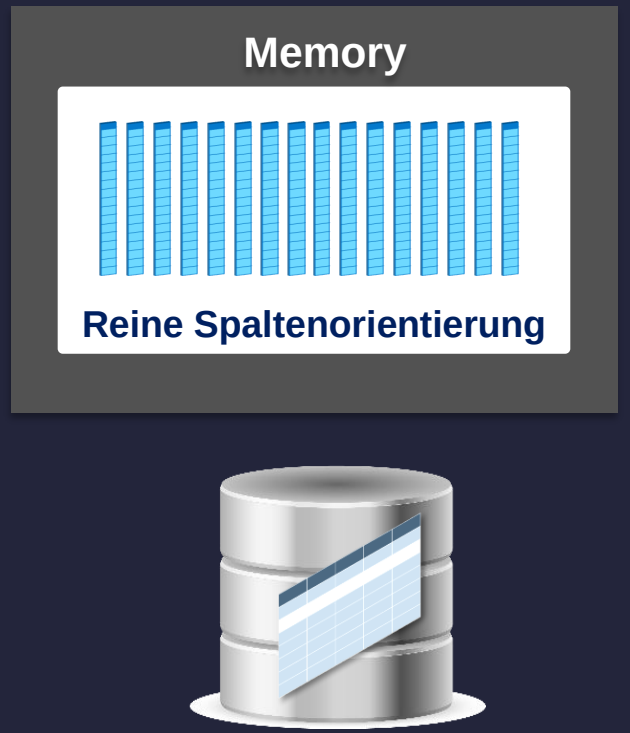
In-Memory Datenbank für beide Formate

- BEIDE Formate (Zeile und Spalte) für die selbe Tabelle
- Gleichzeitig aktiv und transaktionskonsistent
- Analysen und Reports benutzen das NEUE Spaltenformat
- OLTP benutzt das Zeilenformat



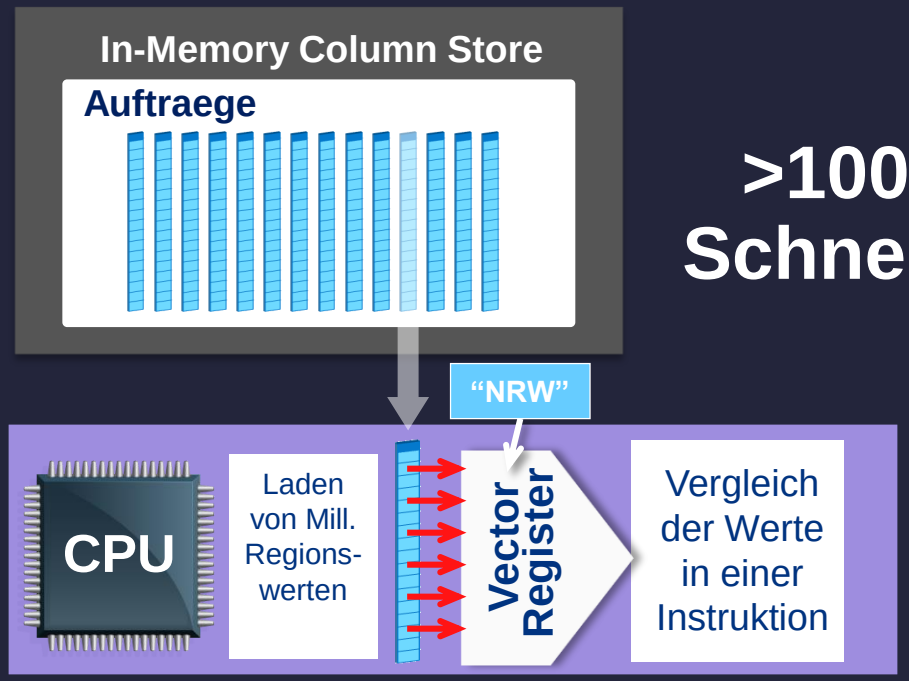
Oracle In-Memory Spalten Technologie

- Reines In-Memory Format mit Nologging
- Fast kein Overhead bei Änderungen
- Memoryoptimierte Komprimierung (2x bis 10x)
- Daten werden bei Startup oder ersten Zugriff geladen
- Bei In-Memory wird ca. 90% des Speichers für das Zeilenformat benutzt



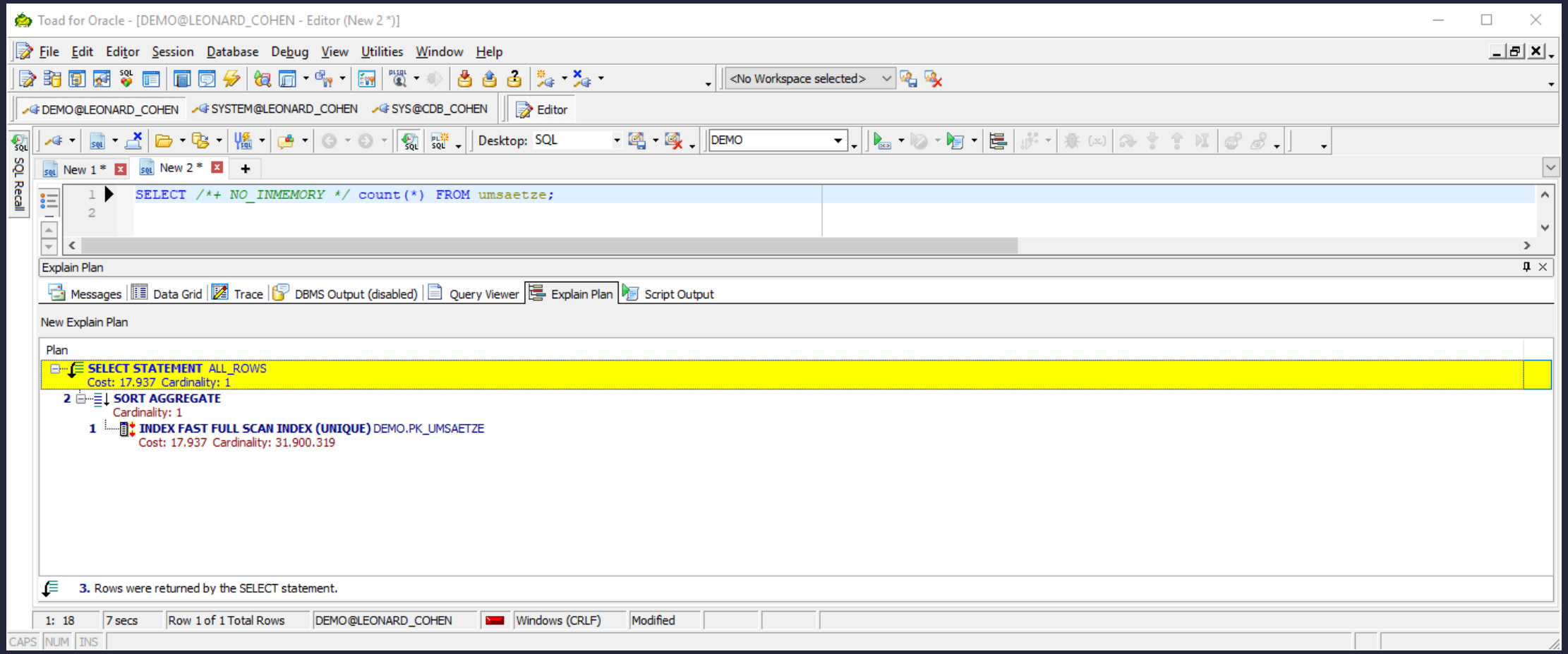
Extrem schnelle Verarbeitung

- Jede CPU scant lokale In-memory Spalten
- Scans verwenden SIMD Vector Instruktionen
- Milliarden Spalten pro Sekunde pro CPU Core

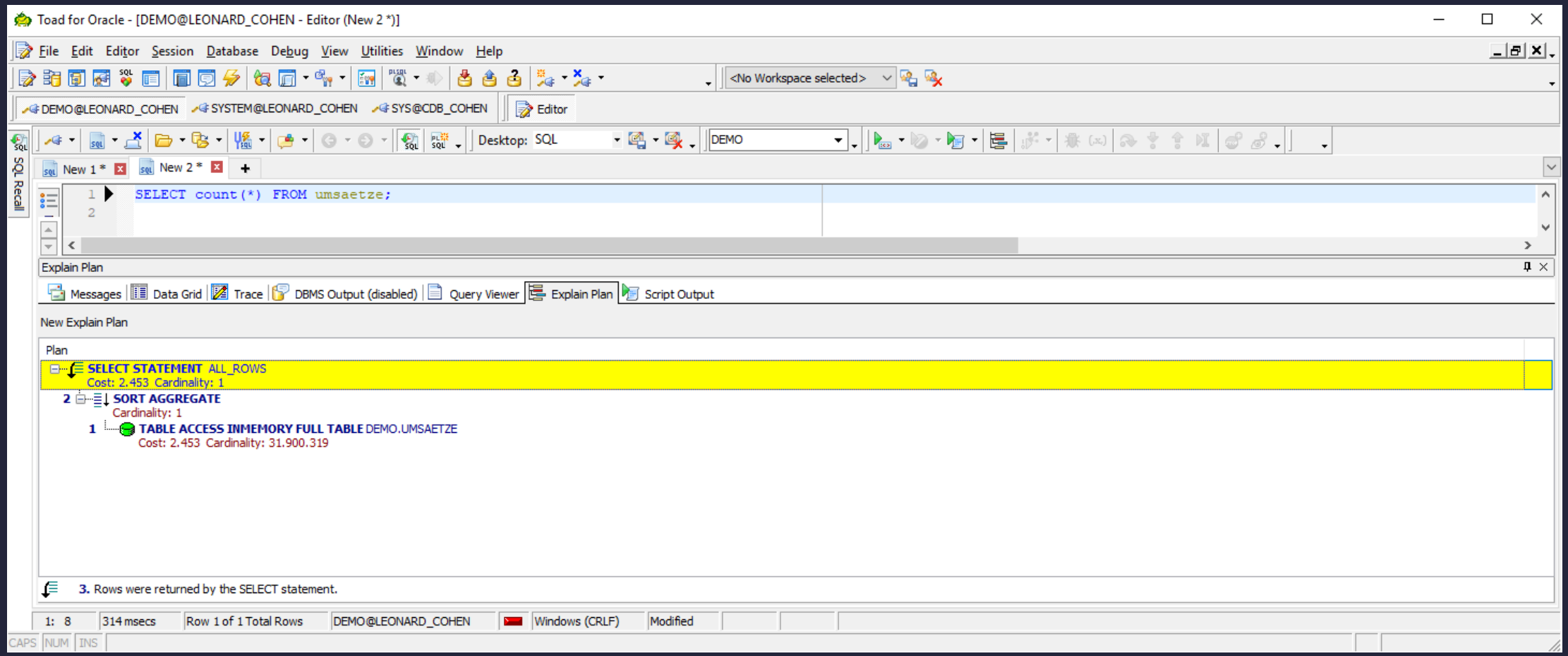


>100X Schneller

Test 1: mit Hint



Test 2: ohne Hint → InMemory



Oracle 18c XE

- Kostenlos
- 2 Cores
- Eine Installation pro Umgebung (z.B. VM)
- 12 GB Userdata
- 2 GB RAM
- 3 PDBs
- Datenmigration über Data Pump
  - Ausnahme: nach Oracle EE über PDB Unplug / Plug
  - Migration nach Oracle SE2 ist nicht supported!

DOAG

DOAG 2019 **Datenbank**

3. und 4. Juni 2019 in Düsseldorf

datenbank.doag.org

DOAG 2019 Datenbank: Call for Papers gestartet!

CarajanDB

- Experten mit über 30 Jahren Datenbank Erfahrung
- Spezialisten für
  - Datenbank Administration (Oracle und PostgreSQL)
  - Hochverfügbarkeit (RAC, Data Guard, Replication, etc.)
  - Migrationen (Unicode, PostgreSQL)
  - Performance Optimierung
  - Monitoring (OEM, Foglight, CheckMK, PEM)
- Fernwartung
- Schulung und Workshops
  - PostgreSQL
  - Oracle
  - Toad

Quest Silver Partner

ORACLE Gold Partner

EDB POSTGRES ECOSYSTEM PARTNER GOLD

Dbvisit GOLD RESELLER

Kontakt

- E-Mail: johannes.ahrends@carajandb.com
- Homepage: www.carajandb.com
- Adresse:
  - CarajanDB GmbH
    Siemensstraße 25
    50374 Erftstadt
- Telefon:
  - +49 (22 35) 1 70 91 84
  - +49 (1 70) 4 05 69 36
- Twitter: carajandb
- Facebook: johannes.ahrends
- Blogs:
  - blog.carajandb.com
  - www.toadworld.com