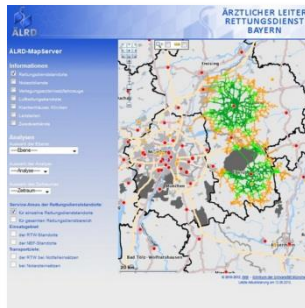


ORACLE 12c AUTOMATIC BIG TABLE CACHE

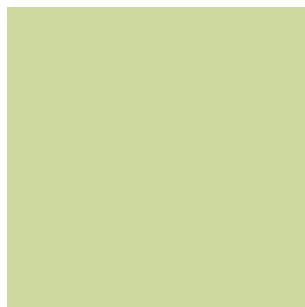
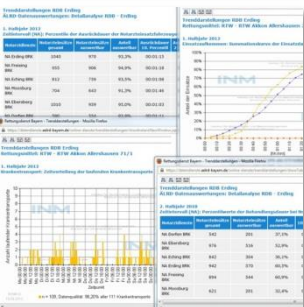
Markus Geis

18.11.2015



Freizeitaktivitäten 2014 Erlang

Aktivität	2012	2013	2014	2015	2016	2017	2018	2019	2020
...



AGENDA

- **INM**
- **automatic big table cache**
 - **Technik**
 - **Einrichtung**
 - **erste Erfahrungen**
 - **Zusammenfassung / Fazit**

INM

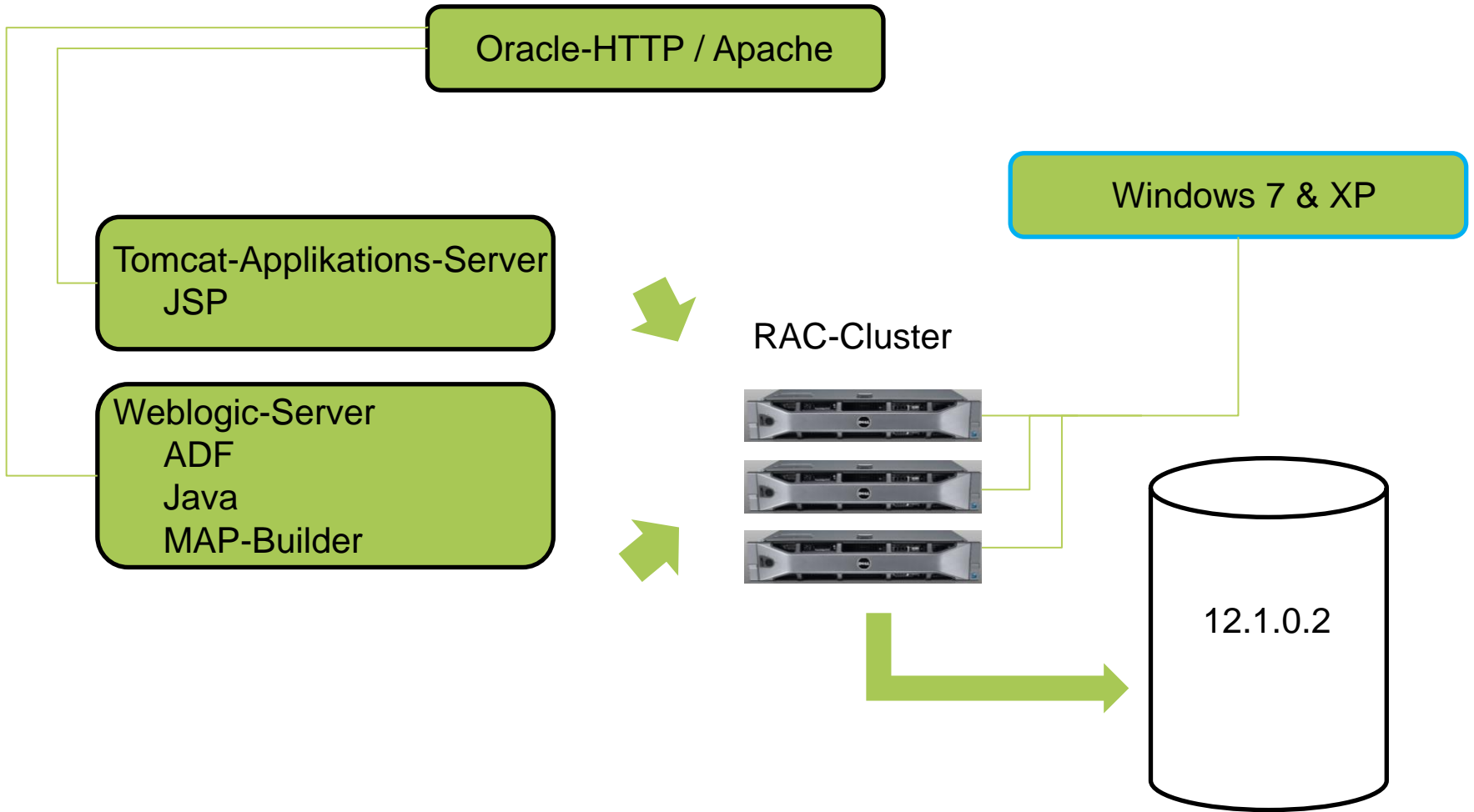
- Institut für Notfallmedizin und Medizinmanagement
 - Teil des Klinikums der Universität München (LMU)
- Qualitätsmanagement in der Notfallmedizin und im Rettungswesen
- Interdisziplinäre Forschungseinrichtung
- DWH: 2 MIO Rettungsdienst-Daten / pro Jahr
Verarbeitung medizinischer- / GEO- Daten
- www.inm-online.de



AUSGANGSSITUATION

- Betrieb eines 12c 3 Knoten Clusters (12.1.0.2 EE)
 - > 3 Cluster-DB's mit ca. 3TB / 1.200 DB-Usern
- Netapp Metro Cluster
- Shared Disk Spaces sind über NFS eingebunden (kein ASM)
- Optionen:
 - Partitioning
 - Spatial (Routenberechnung und Geo-Analysen)

AUSGANGSSITUATION



AUTOMATIC BIG TABLE CACHE

- Teil der Oracle „inMemory“ Initiative
- verfügbar ab der Version 12.1.0.2
- Feature (Poor Man's In-Memory Caching)
- in allen Editionen verfügbar:
 - SI (EE): bei sequentiellen Abfragen (PX)
 - RAC / (EE): nur parallel-query

AUTOMATIC BIG TABLE CACHE

- Performance der DB wird durch den CACHE bestimmt
 - der Zugriff auf die DB-Daten sollte über den CACHE realisiert werden (DB-Blöcke)
 - Plattenzugriffe sollten vermieden werden
- Datenmengen werden größer
- wichtige/unwichtige Daten

AUTOMATIC BIG TABLE CACHE

- bisherige Technologie für die Verarbeitung großer Tables (Full Table Scans)
 - direct path read
 - `_small_table_threshold`
 - COMPRESSED
 - PARTIONING

AUTOMATIC BIG TABLE CACHE

- direct path read
 - liest große Tables bei einem FULL-TABLE-SCAN direkt unter Umgehung des BUFFER-Caches aus den Data-Files in die PGA des Users
 - BUFFER-Cache wird nicht geflutet
 - Nachteil: `select` immer von der Platte

AUTOMATIC BIG TABLE CACHE

- `_small_table_threshold` (Schwellenwert):
beschreibt eine Table als „SMALL“
 - HIDDEN
 - eine Table gilt als SMALL, bei einer Größe $< 2\%$ des
buffer-caches
 - Infos werden über den DB-BLOCK-BUFFER zur Verfügung
gestellt
 - Info wird auch später für den „automatic big table cache“
benötigt

AUTOMATIC BIG TABLE CACHE

■ `_small_table_threshold` (Anzahl Blöcke)

```
SELECT a.kspbinm,b.kspbstvl ,c.kspbstvl,a.kspdesc
FROMx$ksppi a ,x$ksppcv b ,x$ksppsv c
WHEREa.indx = b.indx AND a.indx = c.indx
AND a.kspbinm LIKE '/_%' escape '/'
and a.kspbinm like '_small%'
ORDER BY 1
```

PARAM	SESSIONVAL	INSTANCEVAL	DESCR
<code>_small_table_threshold</code>	76943	76943	lower threshold level of table size for direct reads

- `gv$sgainfo` (Buffer Cache Size 26306674688 bytes ca.26GB)
- $76.943 * \text{DB-Block (8K)} / \text{DB-Buffer-CACHE}$ ca. 2%
- Größe der SMALL-Table in der INM-Umgebung: ca. 600 MB

AUTOMATIC BIG TABLE CACHE

- bisheriges CACHE-Verhalten des DB-BLOCK-BUFFERS:
 - LRU (last recently used)-> Algorithmus
 - LRU-Liste verwaltet den CACHE (DB-BUFFER)
 - bei vollem CACHE, werden die am längsten nicht genutzten Blöcke aus dem CACHE entfernt, um Platz für neue Blöcke zu schaffen
 - **Problem:** wichtige Objekte/Blöcke können so aus dem CACHE herausfallen (Abfragen werden langsam / lesen von Disk)

AUTOMATIC BIG TABLE CACHE

- bisheriges CACHE-Verhalten des DB-BLOCK-BUFFERS:
 - COLD- / HOT-AREA
 - Blöcken werden in der Mitte der LRU-Liste eingefügt
 - Je nach Nutzungsfrequenz wandern die Blöcke in den COLD oder HOT Bereich
 - bei Platzbedarf fallen dann mit der Zeit Blöcke aus dem CACHE heraus

AUTOMATIC BIG TABLE CACHE

- eine Möglichkeit in der Vergangenheit, um das Herausfallen von Objekten aus dem CACHE zu verhindern -> KEEP-Mechanismus
- bestimmte Tables konnten in den Hauptspeicher gepinnt werden
- es war eine Technologie für kleine Tables
- es konnten keine „Partitionen“ angesprochen werden
- der Workload spielte keine Rolle / die Technik wurde durch den Administrator bestimmt

AUTOMATIC BIG TABLE CACHE

- **Automatic big table cache: „ABTC“**
 - die Technologie soll verhindern, daß oft genutzte Objekte aus dem CACHE herausfallen
 - der „*workload*“ soll die bestimmende Größe für das CACHE-Verhalten sein
 - ein bestimmter Teil des DB-Block-Buffers wird für diese Technologie reserviert (bis 90%)

AUTOMATIC BIG TABLE CACHE

- Wie berechnet sich der Workload?
 - Algorithmus, welcher über die Häufigkeit des Zugriffes bestimmt ob und wie ein Objekt (z.B. Table) in den CACHE geladen wird und dort verbleibt
 - Oracle bezeichnet dies als "temperature"
 - je höher die "temparture" desto eher verbleiben die Objekte im CACHE
 - Es werden nur große Tables gemessen;
keine kleinen Tables -> `_small_table_threshold`

AUTOMATIC BIG TABLE CACHE

- **zwei Einstellungen für die Aktivierung:**

```
ALTER SYSTEM SET db_big_table_cache_percent_target=55;
```

```
ALTER SYSTEM SET compatible='12.1.0.2.0'
```

- RAC-Cluster: „parallel query“ eingeschaltet
(automatic / adaptive)

```
ALTER SYSTEM SET PARALLEL_DEGREE_POLICY=AUTO;
```

- single instance : auch Verarbeitung von sequentiellen Abfragen

UPGRADE 12C – TECHNIK

- Es können folgende Objekte in den „ABTC“ geladen werden:

OBJEKTE	OBJECT-TYPE (v\$bt_scan_obj)
Table	TABLE PARTITION
Table-Partition	TABLE SUBPARTITION
Index	INDEX
Index-Partition	INDEX SUBPARTITION

AUTOMATIC BIG TABLE CACHE - TECHNIK

- Zwei SYS-Views für die Verwaltung des "ABTC":
 - Single-Instance
 - V\$BT_SCAN_CACHE -> CACHE-Infos
 - V\$BT_SCAN_OBJ_TEMPS -> Objekte im "ABTC"
 - RAC:
 - gV\$BT_SCAN_CACHE
 - gV\$BT_SCAN_OBJ_TEMPS

AUTOMATIC BIG TABLE CACHE – TECHNIK

■ Beispiel: Platzbedarf und Infos über den „ABTC“

```
SELECT bt_cache_alloc, bt_cache_target, object_count, memory_buf_alloc
FROM v$bt_scan_cache;
```

BT_CACHE_ALLOC	BT_CACHE_TARGET	OBJECT_COUNT	MEMORY_BUF_ALLOC
.400005755	40	3	50685

```
SELECT * from gv$BT_SCAN_CACHE
```

INST_ID	BT_CACHE_ALLOC	BT_CACHE_TARGET	OBJECT_COUNT	MEMORY_BUF_ALLOC
2	0,550005216490914	55	204	1706599
1	0,550000710281341	55	2563	1659568
3	0,550000788577281	55	234	1732373

AUTOMATIC BIG TABLE CACHE – TECHNIK

■ Objekte im „ABTC“:

```
SELECT distinct obj.object_name,ts.name ts_name,subobject_name,object_type,
temperature,policy, (cached_in_mem*8192)/1024/1024 cached_mb, dataobj# , btt.inst_id
FROM gv$tablespace ts,gv$bt_scan_obj_temps btt,dba_objects obj
WHERE ts.ts# = btt.ts#
AND obj.object_id = DATAOBJ#
order by temperature desc, dataobj# ;
```

OBJECT_NAME	TS_NAME	SUBOBJECT_NAME	OBJECT_TYPE	TEMPERATURE	POLICY	CACHED_MB	DATAOBJ#	ID
ILS_EINSATZMITTEL	ARLIS_DATA	RDB_303	TABLE PARTITION	405000	MEM_ONLY	64,4140625	1251086	2
ARLIS_DATA	ARLIS_DATA	ED_2012_RDB_303	TABLE SUBPARTITION	9813	MEM_ONLY	44,7109375	1029063	3
ARLIS_DATA_FA_EDAT	ARLIS_INDEX	ED_2008_RDB_302	INDEX SUBPARTITION	2000	MEM_ONLY	1,3671875	1046850	1

AUTOMATIC BIG TABLE CACHE - TECHNIK

- im RAC werden die Objekte /oder dessen Fragmente über die Nodes verteilt
- jeder Node verwaltet seinen CACHE
- im Gegensatz zu „cache-fusion“ werden keine Blöcke/Daten über den interconnect übertragen (nur Ergebnisse von entfernten Knoten über PX erzeugt und übertragen)
- bei: `PARALLEL_FORCE_LOCAL=TRUE` (interconnect / PX bleiben auf einer Instance)
- Objekte können dann mehrfach auf den Instanzen allokiert werden

AUTOMATIC BIG TABLE CACHE - TECHNIK

```

SELECT  distinct obj.object_name,ts.name ts_name,subobject_name,object_type,
        temperature,policy, (cached_in_mem*8192)/1024/1024 cached_mb, dataobj# , btt.inst_id
FROM gv$tablespace ts,gv$bt_scan_obj_temps btt,dba_objects obj
WHERE ts.ts# = btt.ts#
AND obj.object_id = DATAOBJ#
and subobject_name='ED_2013_RDB_501'
order by  obj.object_name,btt.inst_id

```

OBJECT_NAME	SUBOBJECT_NAME	OBJECT_TYPE	TEMPERATURE	POLICY	CACHED_MB	DATAOBJ#	INST_ID
ARLIS_DATA	ED_2013_RDB_501	TABLE SUBPARTITION	2000	MEM_ONLY	130,3203125	1058127	1
ARLIS_DATA	ED_2013_RDB_501	TABLE SUBPARTITION	1000	MEM_ONLY	130,3203125	1058127	2
ARLIS_DATA	ED_2013_RDB_501	TABLE SUBPARTITION	60250	MEM_ONLY	130,3203125	1058127	3

AUTOMATIC BIG TABLE CACHE – TECHNIK

■ Ausprägungen des „ABTC“ (POLICY)

MEM_ONLY	Objekt wird komplett im Cache gehalten
MEM_PART	Teile des Objektes werden im CACHE gehalten
DISK	Objekt wird komplett von DISK gelesen

OBJECT_NAME	TS_NAME	TEMPERATURE	POLICY	CACHED_MB
ARLIS_DATA_FIRMA_P_O_IDX	ARLIS_INDEX	2000	DISK	0
ARLIS_DATA_FIRMA_P_O_IDX1	ARLIS_INDEX	2000	MEM_PART	1,2578125
DM_TG_RDS_BODEN	ARLIS_DATA	26563	MEM_ONLY	888,8046875

AUTOMATIC BIG TABLE CACHE - TECHNIK

```
SQL> select object_count from V$BT_SCAN_CACHE;
```

```
OBJECT_COUNT
```

```
-----
```

```
0
```

```
SQL> set autotrace on
```

```
SQL> select count(*) from scott.t1;
```

```
COUNT(*)
```

```
-----
```

```
2000000
```

```
Execution Plan
```

```
-----
```

```
Plan hash value: 227768458
```

```
-----
```

Id	Operation	Name	Rows	Cost (%CPU)	Time
0	SELECT STATEMENT		1	211 (6)	00:00:01
1	SORT AGGREGATE		1		
12	INDEX FAST FULL SCAN	SYS_C0011355	2565K	211 (6)	00:00:01

```
-----
```

```
Statistiken
```

```
-----
```

```
947 recursive calls
```

```
0 db block gets
```

```
11489 consistent gets
```

```
4191 physical reads
```

AUTOMATIC BIG TABLE CACHE - TECHNIK

```
select object_count from V$BT_SCAN_CACHE;
```

```
OBJECT_COUNT
```

```
-----
```

```
1
```

```
select ts#,dataobj#,policy from V$BT_SCAN_OBJ_TEMPS;
```

```
TS#      DATAOBJ# POLICY
```

```
-----
```

```
196612   96464  MEM_ONLY
```

```
select owner,object_name,object_type
from dba_objects
where data_object_id=96464;
```

```
OWNER OBJECT_NAME OBJECT_TYPE
```

```
-----
```

```
SCOTT T1          TABLE
```

AUTOMATIC BIG TABLE CACHE - TECHNIK

```
select count(*) from scott.t1;
```

Execution Plan

Plan hash value: 227768458

Id	Operation	Name	Rows	Cost (%CPU)	Time
0	SELECT STATEMENT		1	211 (6)	00:00:01
1	SORT AGGREGATE		1		
2	INDEX FAST FULL SCAN	SYS_C0011355	2565K	211 (6)	00:00:01

Statistiken

3911 consistent gets
0 physical reads

AUTOMATIC BIG TABLE CACHE - TECHNIK

```
select /*+ full(t1) */ count(*) from scott.t1;  
COUNT(*)
```

```
-----  
2000000
```

```
Elapsed: 00:00:00.03
```

```
Execution Plan
```

```
-----  
Plan hash value: 3724264953  
-----
```

Id	Operation	Name	Rows	Cost (%CPU)	Time
0	SELECT STATEMENT		1	596 (33)	00:00:01
1	SORT AGGREGATE		1		
2	TABLE ACCESS FULL	T1	2565K	596 (33)	00:00:01

```
-----  
Statistiken
```

```
-----  
7883 consistent gets
```

```
0 physical reads
```

AUTOMATIC BIG TABLE CACHE - TECHNIK

■ Messungen INM:

- Problem: Testumgebung (neuer Test jeweils nach ein paar Tagen Normalbetrieb)
- 55% „ABTC“ bei ca. 26 GB DB-BLOCK-BUFFERS
- regelmäßige einzelne selects auf eine Table „distanz_matrix“ mit ca. 1,6 MRD Rows, um den Cache und die Maschine zu belasten (12c mit ABTC)
- Restart der Statments jeweils nach PLUS einem Tag
- Table mit Partitionen und SUB-Partitionen
- SIZE: 22,6 GB / 30,5 MIO Rows

AUTOMATIC BIG TABLE CACHE - TECHNIK

- **Select-INM:**
- **Test: 29.10.2015:**

```
SELECT distinct obj.object_name,ts.name ts_name,subobject_name,object_type,
  temperature,policy, (cached_in_mem*8192)/1024/1024 cached_mb, dataobj# , btt.inst_id
FROM gv$tablespace ts,gv$bt_scan_obj_temps btt,dba_objects obj
WHERE ts.ts# = btt.ts#
AND obj.object_id = DATAOBJ# order by temperature desc, dataobj# ;
```

OBJECT_NAME	TS_NAME	SUBOBJECT_NAME	OBJECT_TYPE	TEMPERATURE	POLICY	CACHED_MB	DATAOBJ#	INST_ID
ARLIS_DATA	ARLIS_DATA	ED_2013_RDB_308	TABLE SUBPARTITION	56250	MEM_ONLY	43,4609375	1058112	3
ARLIS_DATA	ARLIS_DATA	ED_2013_RDB_505	TABLE SUBPARTITION	56250	MEM_ONLY	59,8359375	1058114	3
ARLIS_DATA	ARLIS_DATA	ED_2013_RDB_206	TABLE SUBPARTITION	56250	MEM_ONLY	63,90625	1058116	3
ILS_EINSATZMITTEL	ARLIS_DATA	RDB_303	TABLE PARTITION	17010	MEM_ONLY	69,3203125	1251086	2
ILS_EINSATZMITTEL	ARLIS_DATA	RDB_314	TABLE PARTITION	16010	MEM_ONLY	400,984375	1251097	2

AUTOMATIC BIG TABLE CACHE - TECHNIK

■ Messungen INM:

```
select /*+ full(t1) */ distinct to_char(einsatz_datum,'dd.mm.yyyy') TAG, count(*)
from oracle.arlis_data t1
where einsatz_datum >=to_date('01.01.2013','dd.mm.yyyy')
and einsatz_datum < to_date('01.01.2014','dd.mm.yyyy')
and firma=308
group by to_char(einsatz_datum,'dd.mm.yyyy')
```

```
TAG                COUNT(*)
-----
23.10.2013         157
22.10.2013         140
09.05.2013          97
```

...

365 Zeilen ausgewählt.

Abgelaufen: 00:00:05.17

AUTOMATIC BIG TABLE CACHE - TECHNIK

■ Messungen INM:

Ausführungsplan

 Plan hash value: 1211467257

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time	Pstart	Pstop
0	SELECT STATEMENT		365	4380	1260 (4)	00:00:01		
1	HASH GROUP BY		365	4380	1260 (4)	00:00:01		
2	PARTITION RANGE SINGLE		62513	732K	1249 (3)	00:00:01	16	16
3	PARTITION LIST SINGLE		62513	732K	1249 (3)	00:00:01	KEY	KEY
4	TABLE ACCESS FULL	ARLIS_DATA	62513	732K	1249 (3)	00:00:01	401	401

 Statistiken

55359 consistent gets
 362 physical reads

AUTOMATIC BIG TABLE CACHE - TECHNIK

■ Messungen INM:

- Nutzung von Partitioning in Kombination mit "ABTC"
- Partition reduziert Datenmenge erheblich
- "ABTC" hält die Partition durch den errechneten workflow im CACHE

AUTOMATIC BIG TABLE CACHE - TECHNIK

■ Messungen INM / Zeitverlauf:

```
select count(*), firma, kfzart  
from oracle.arlis_data  
group by firma, kfzart
```

Statement	12.1.0.2 / SI / ohne ABTC	11gCluster	12c Cluster + parallel /ABTC
2	00:00:20.64	00:05:39.64	00:00:13.67
			00:00:18.31 (Tage später)
			00:00:26.97 (Tage später)

AUTOMATIC BIG TABLE CACHE – ERSTE ERFAHRUNGEN

- einfach einzurichten / keine Änderung an DB-Strukturen u. Applikationen
- besonders gut für partitionierte Tables verwendbar
- bei Simulationsberechnungen mit wiederkehrenden SQL-Abfragen auf gleiche Table-Partitionen
- der workload bestimmt den Inhalt des CACHES
- es benötigt einige Zeit, damit sich der „workload“ richtig einpendelt
- erheblicher Performance-Gewinn in unserer Umgebung
z.B. Procedure vor „ABTC“ 45 Minuten / jetzt 12 Minuten
- produktiv seit ca. 3 Monaten

VIELEN DANK FÜR IHRE AUFMERKSAMKEIT

ANSPRECHPARTNER:

Markus Geis

Klinikum der Universität München

INM - Institut für Notfallmedizin
und Medizinmanagement

Telefon: 089 / 4400-57101

E-Mail: markus.geis@med.uni-muenchen.de

Internet: www.inm-online.de



