F A C H L I T E R A T U R

Oracle In-Memory

Workshopbegleitende Unterlage

MD Consulting & Informationsdienste GmbH

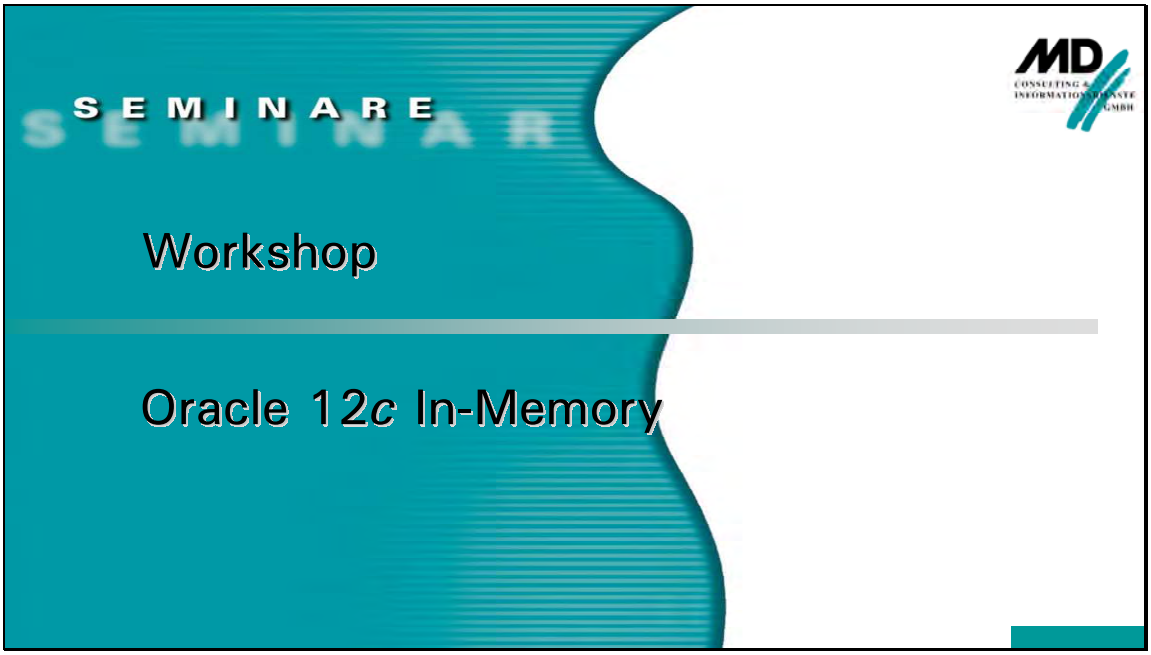
www.md-consulting.de

Michaelisstraße 13 a
99084 Erfurt
03 61 / 5 65 93-0

Berghamer Straße 14
85435 Erding
0 81 22 / 97 40-0

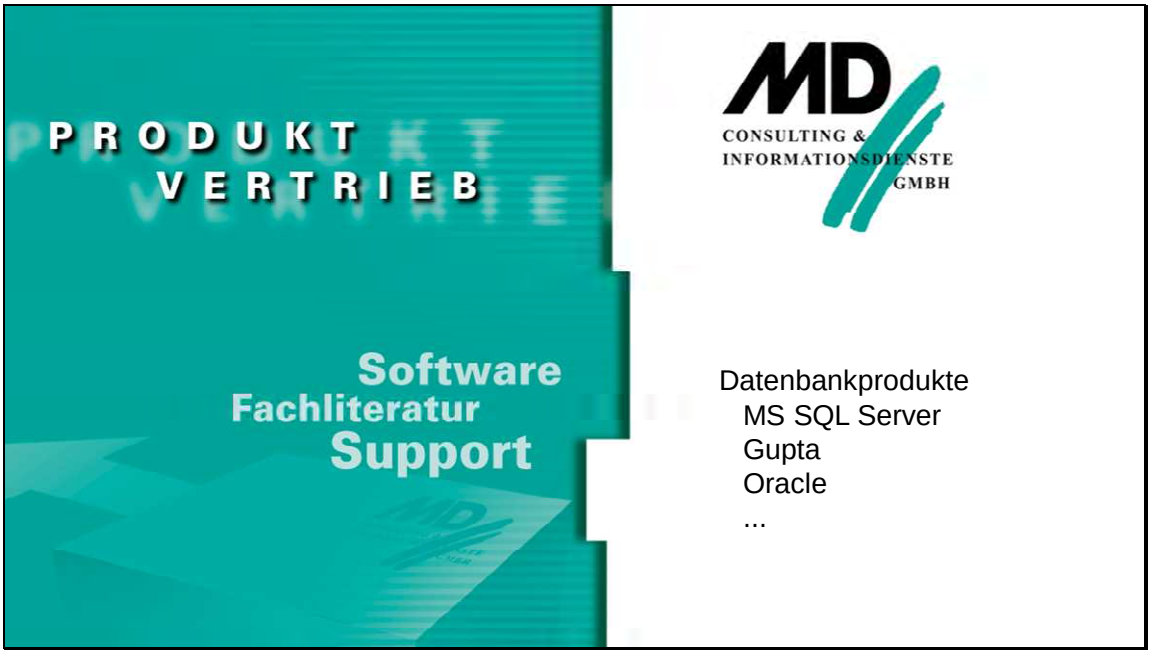
info@md-consulting.de

Folie 1



Folie 2

Folie 3

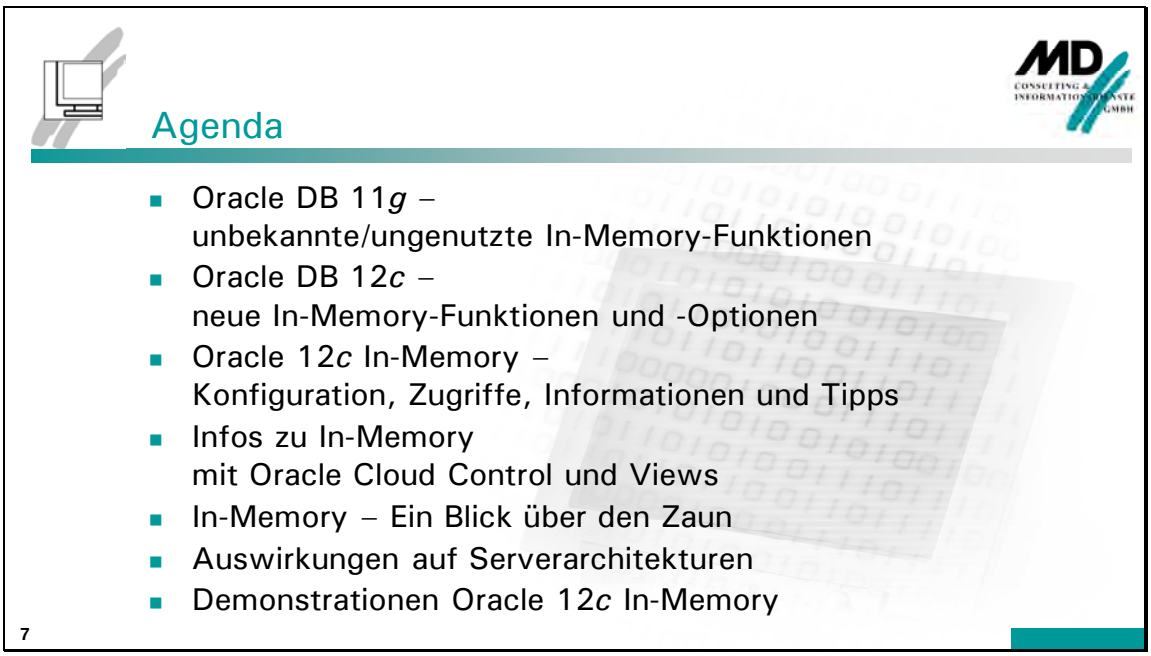


Folie 4

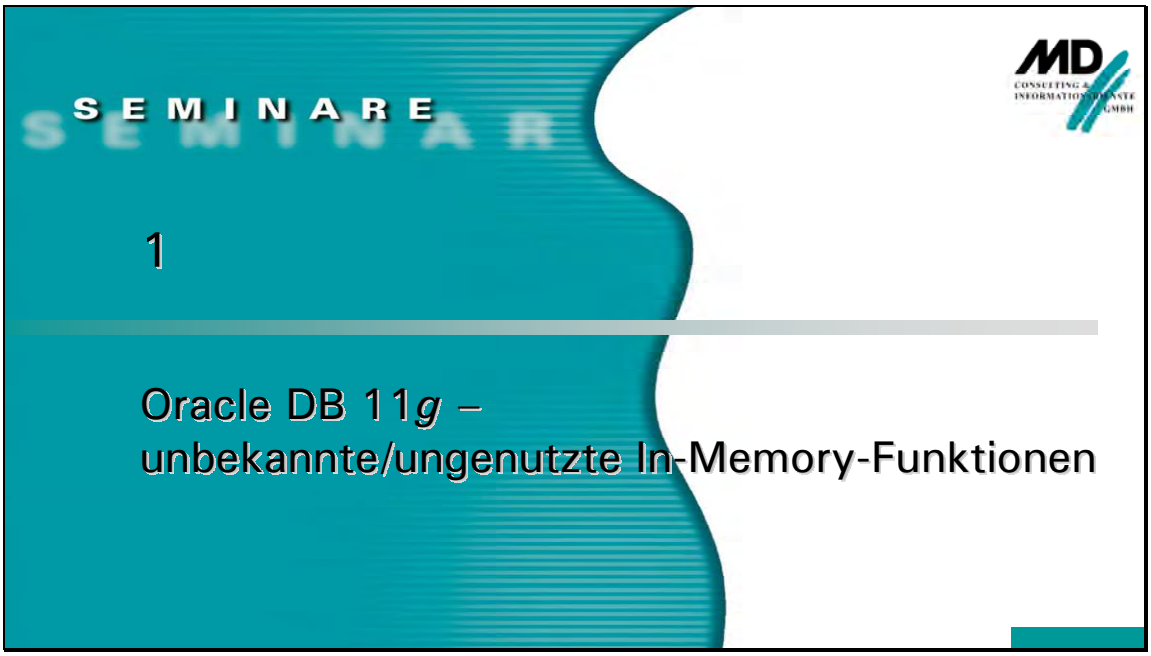
Folie 5

Folie 6

Folie 7



Folie 8



Folie 9

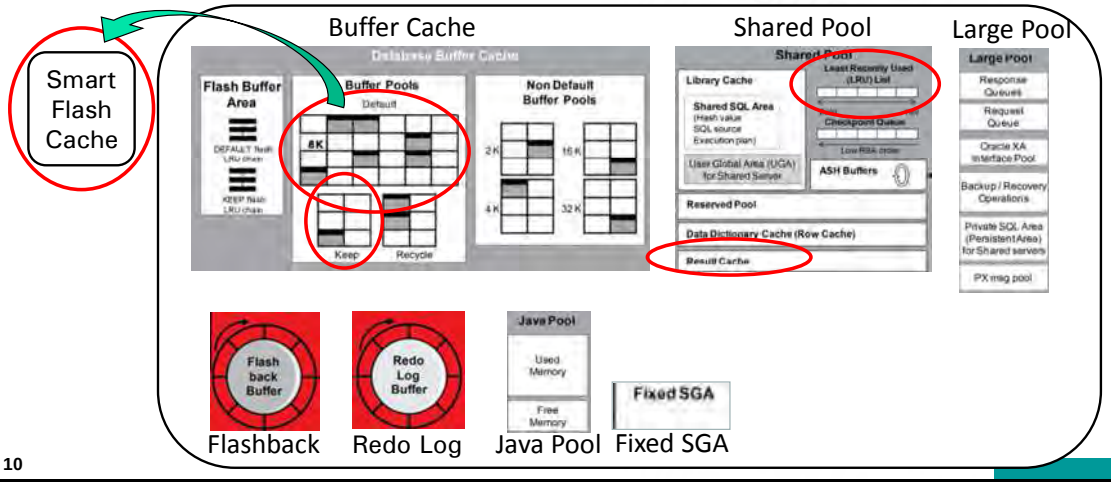
In-Memory-Technologien in 11*g* und 12.1.0.1

In-Memory-Technologien in 11*g* und 12.1.0.1

- ***Großer DB-Cache***
- Row-Caches
- Statement Caches
- ***Pinnen von ganzen Tabellen***
- Oracle Text Index im Memory
- ***Query Result Cache***
- ***In-Memory Parallel Query (RAC)***
- Cache Fusion (RAC)
- Cache Hierarchie
- ***Smart Flash Cache***

Folie 10

Oracle System Global Area (SGA) Komponenten



Folie 11

Pinning von Objekten

- Alle Segmente (Tabellen, Indexes, LOBs) – Ausschalten LRU

```
ALTER SYSTEM SET db_keep_cache_size = 1G; -- Extra Pool LRU
ALTER TABLE name STORAGE (BUFFER_POOL KEEP);
```

```
ALTER TABLE name CACHE;     -- Default Pool, LRU für Objekt
```

  - Segment wird erst bei Zugriff geladen
    deshalb Preload durch Trigger (Startup, ...)
- Stored Procedures
  - Pinnen über **DBMS_SHARED_POOL.KEEP**

11

Folie 12

Result Cache – Ergebnisse im Memory
Einmal rechnen, mehrfach nutzen!

- Server Result Cache und Client Site Cache (OCI Anwendungen)
- Server Result Cache für Query und PL/SQL
  - eigener Bereich der SGA – Steuerung mittels Parametern
  - Automatische Cache-Verwaltung und Invalidierung
- Mögliche Anwendungsbeispiele:
  - Langlaufende und rechenintensive SQL-Abfragen und PL/SQL-Funktionen
  - Vorhersehbare SQL-Abfragen oder gleichbleibende deterministische Ergebnismengen
- Kriterien:
  - Kleine Ergebnismengen bzw. ausreichendes Memory
  - Geringe DML-Aktivität auf den zugrundeliegenden Tabellen

12

Folie 13

Nutzung des Result Cache

- Nutzung des Result Cache wird bestimmt durch drei Faktoren
  - Auf SQL Query Ebene durch den Hint RESULT_CACHE

```
SELECT /*+ RESULT_CACHE*/ * FROM tabelle;
```

  - Einstellung RESULT_CACHE in der Tabellendefinition

```
ALTER TABLE tabelle RESULT_CACHE FORCE;
```

  - Session-Parameter **RESULT_CACHE_MODE**

```
ALTER SESSION SET RESULT_CACHE_MODE = FORCE;
```

- Result Cache für PL/SQL Function Results
  - Aktiv, wenn PL/SQL-Funktionen von PL/SQL aus aufgerufen werden

13

Folie 14

Database Smart Flash Cache

- Hardwarevoraussetzung: Flashspeicher (PCI)
- Softwareanforderungen
  - Datenbank EE (11*g*R2 od. 12*c*)
  - Betriebssysteme: Oracle Linux, Solaris
- Erweiterung des Database Buffer Cache
  - SSD ist wie ein Level 2 Cache (SGA ist Level 1)
    - Schneller als Platte (bis 100x schneller bei Leseoperationen)
    - Günstiger als Memory
    - Größere Kapazität

14

Folie 15

Fazit

- In-Memory-Technologien und Caches der Oracle-Datenbank
  - Buffer Cache: KEEP und RECYCLE
  - Flash Cache
  - Result Cache
- Mehrwert liegt in der **Kombination der Technologien**
  - Aggregate auf normale, relationale Tabellen: **„In Memory DB“**
  - LOBs, JSON, XML, Geodaten: **Buffer Cache KEEP Pool**
  - **Query Result Cache**

Folie 16

SEMINARE

2

Oracle DB 12c –
neue In-Memory-Funktionen und -Optionen

Folie 17

Oracle DB und In-Memory

In-Memory-Technologien in 11g u. 12.1.0.1

. ***Großer DB-Cache***
. Row-Caches
. Statement Caches
. ***Pinnen von ganzen Tabellen***
. Oracle Text Index im Memory
. ***Query Result Cache***
. ***In-Memory Parallel Query (RAC)***
. Cache Fusion (RAC)
. Cache Hierarchie
. ***Smart Flash Cache***

In-Memory-Technologien in 11g und 12.1.0.2

. Großer DB-Cache
. Row-Caches
. Statement Caches
. Pinnen von ganzen Tabellen
. Oracle Text Index im Memory
. Query Result Cache
. In-Memory Parallel Query (RAC)
. Cache Fusion (RAC)
. Cache Hierarchie
. Smart Flash Cache
. ***Full DB in Memory***
. ***Automatic Big Table Caching***
. ***In-Memory*** *(Columnar)* ***Option***

17

Folie 18

Full Database Caching (12.1.0.2)

- Ziel: Datenbank komplett in der SGA (Buffer Cache) halten
- Force Full Caching Mode
  Die Datenbank lädt alle Objekte in den Buffer Cache und hält sie dort (keine Auslagerung mehr). Auch **NOCACHE**-Objekte sind betroffen.
- Information in Dictionary View **V$DATABASE**

```
SELECT FORCE_FULL_DB_CACHING FROM V$DATABASE;
```

```
FORCE_FULL_DB_CACHING
------------------------
NO
```

18

Folie 19

Full Database Caching: Vorgehensweise

- Sicherstellen, dass der Buffer Cache tatsächlich groß genug ist
- Force Full Database Caching aktivieren mit …

```
ALTER DATABASE MOUNT;
ALTER DATABASE FORCE FULL DATABASE CACHING ;
ALTER DATABASE OPEN;
```

- Full DB Caching wird auf Ebene der Containerdatenbank eingestellt
  - Alle Pluggable Databases betroffen
- Objekte werden bei Zugriff (*Access*) geladen, nicht sofort
  - Es findet kein Pre-Loading statt

19

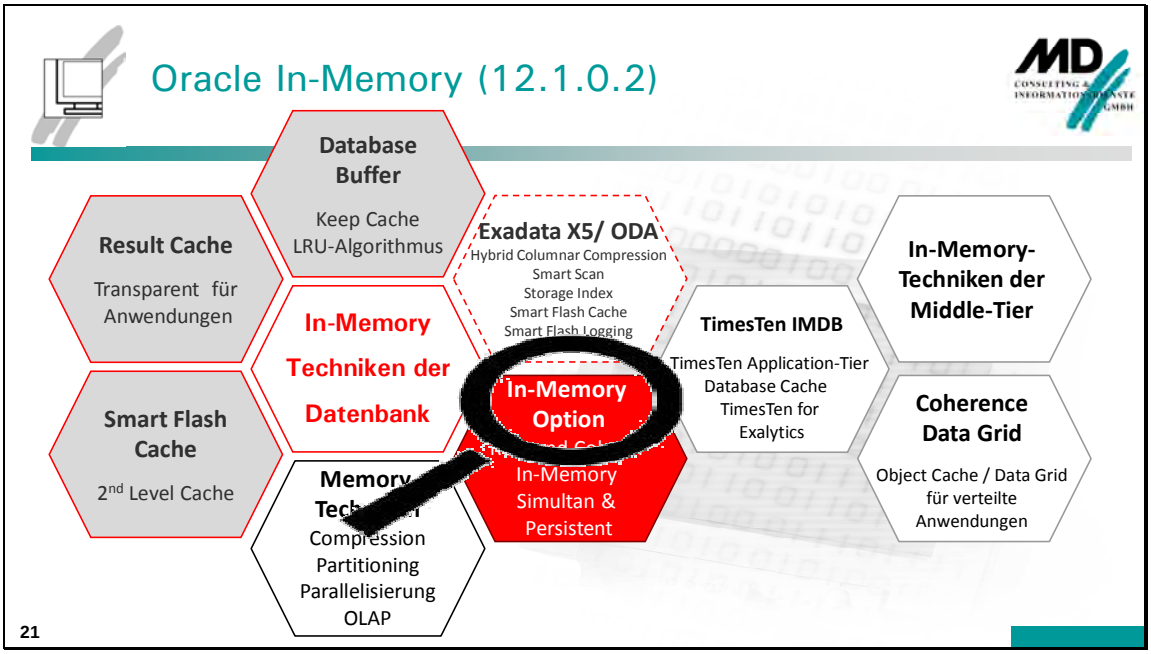
Folie 20

Automatic Big Table Caching (12.1.0.2)

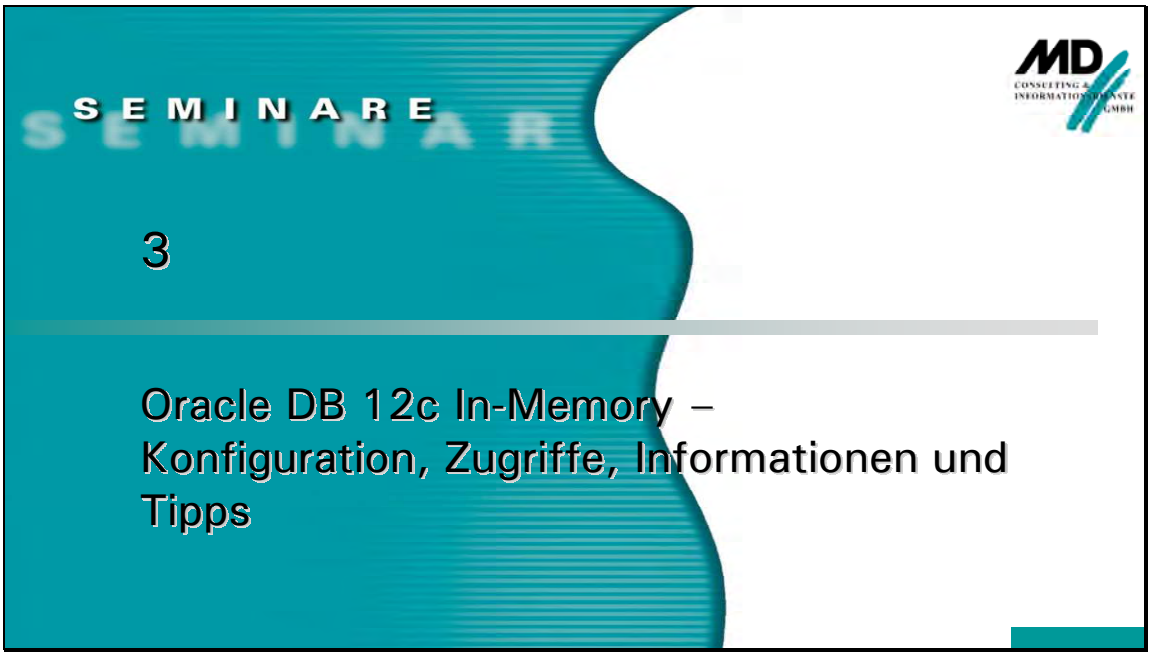
- unterstützt Situationen, in denen die gesamte Datenbank nicht in den Speicher passen würde, aber einige große Objekte
- Blöcke werden nicht aus dem Cache geworfen, sondern bleiben als Objekte (Tabellen) erhalten
- Einschalten des Features durch **`db_big_table_cache_percent_target`** = <Prozentsatz des Buffer Cache>
- bei einfachen Table Scan Workloads kann die Latenz um Faktor 4 reduziert werden
- bei TPCH Workload kann der Durchsatz um 15% erhöht und I/O um Faktor 3 reduziert werden

20

Folie 21



Folie 22



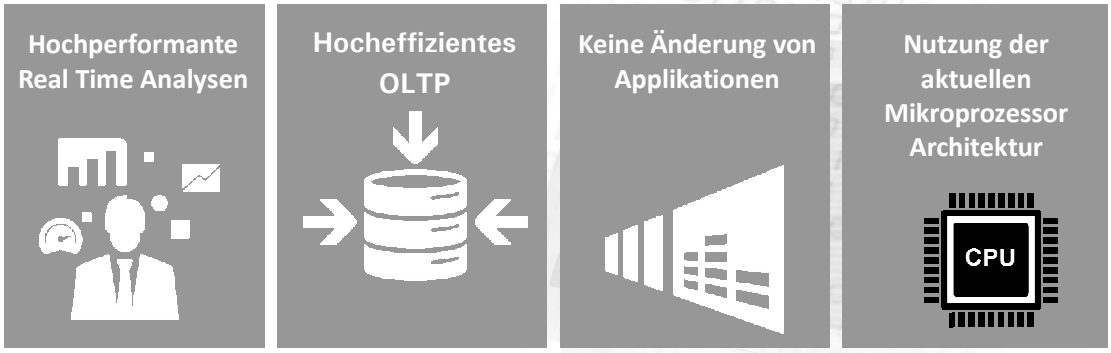
Folie 23

Oracle 12c In-Memory DB Option

- Konzept
- Konfiguration
- Funktionsweise des Column-Stores
- Zugriffe
- Fazit, Tipps und Informationen

Folie 24

Oracle Database In-Memory-Ziele



Folie 25

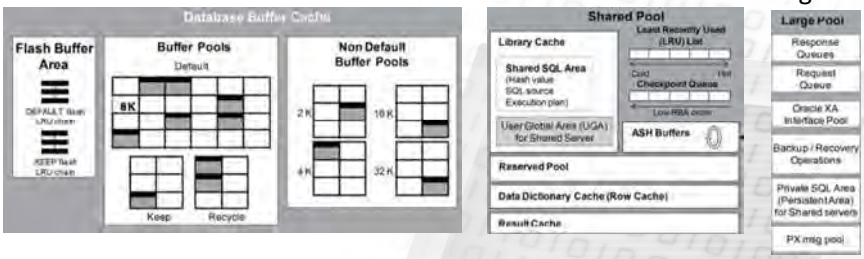
Oracle Datenbank 12c In-Memory – Designziele

- Extreme Performance-Steigerungen für
  - Berichte, Analysen, Ad-hoc-Reporting
  - OLTP, Batch und Data-Warehousing
- Nahtlose Integration in vorhandene Infrastrukturen
  - Administration bleibt unverändert
  - Betriebsprozesse bleiben unverändert (z.B. Backup, Monitoring, gleiche Tools, keine Migration notwendig)
  - Partitionierungskonzepte und Aggregationsstrategien bleiben unverändert
  - Bedeutet ein Höchstmaß an Investitionsschutz
- Transaktional konsistent (SCN-aware)
- Transparent für alle Applikationen
- Einfache Aktivierung in der Datenbank

Folie 26

Standard System Global Area (SGA) Komponenten

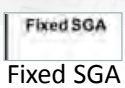
Buffer Cache | Shared Pool | Large Pool



Flashback

Redo Log

Java Pool



Fixed SGA

Folie 27

Einfaches Setup und Nutzung

- In-Memory Größe definieren – Systemparameter

```
ALTER SYSTEM SET inmemory_size = xxxx M ;
```

  - Default Value 0
  - Mindestwert, falls > 0 : 100 M
- Tabellen oder Partitionen festlegen

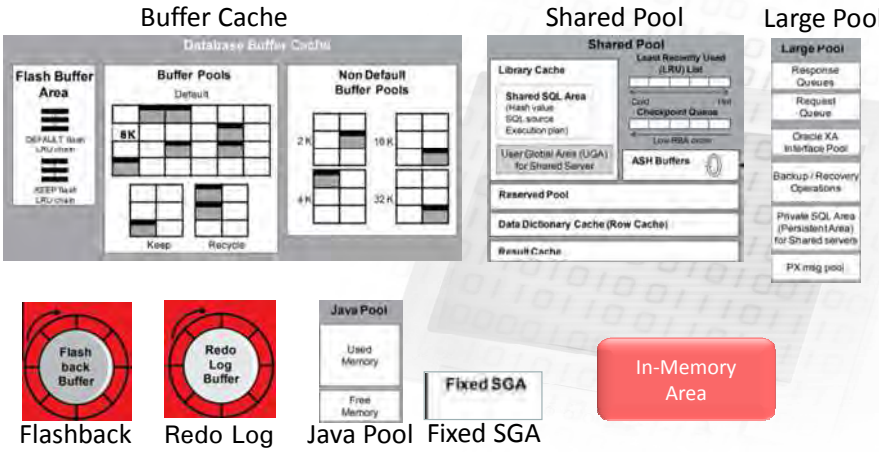
```
ALTER TABLE tablename INMEMORY ;
```

```
ALTER PARTITION partitionname INMEMORY ;
```

- Analytische Indizes löschen/invisible setzen (optional)

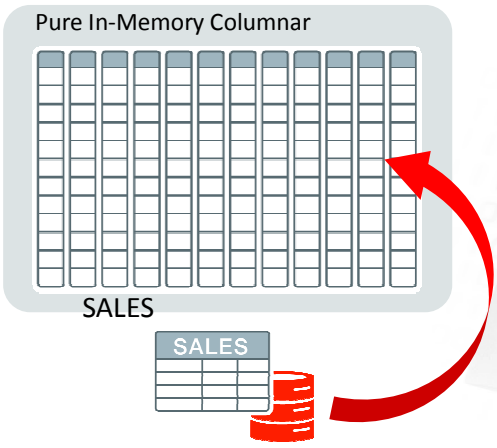
Folie 28

Neue SGA Komponente – In-Memory Area



Folie 29

In-Memory Column Store – Kurzbeschreibung

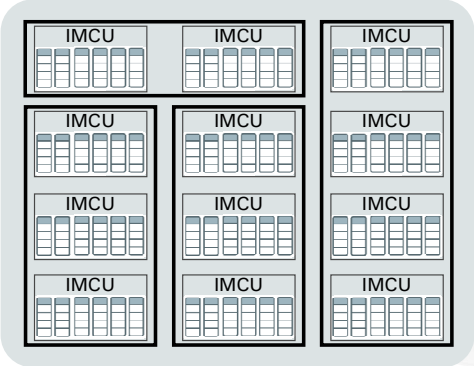


- Reines In-Memory-Format
- Komprimierte Bereiche mit unterschiedlichen Raten (2x bis 20x)
- Einschaltbar auf Objektebene
- Verfügbar auf allen Plattformen
- Voraussetzungen
  - Datenbank Release 12.1.0.2 (2015)
  - Manuelle Aktivierung
  - Gebunden an In-Memory-Option
- Läuft auf jeder Hardware

29

Folie 30

In-Memory Column Store – Im Detail

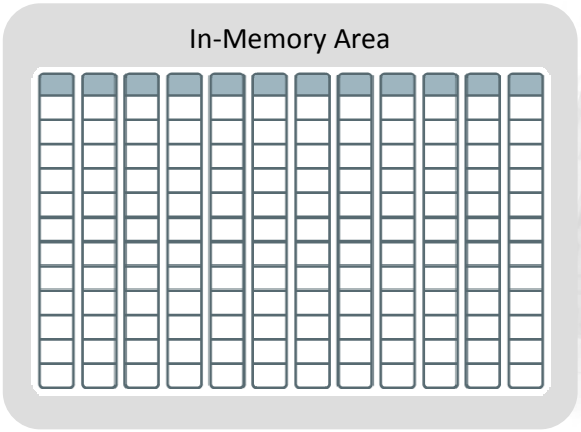


- jedes Objekt im Column Store besteht aus mehreren **I**n-**M**emory **C**ompression **U**nits (IMCU)
- Jede IMCU enthält einen Teil der Spalteneinträge eines Objekts
- IMCUs können in der Größe variieren – abhängig vom Datentyp und Komprimierungsrate

30

Folie 31

Aufbau der In-Memory Area



- 2 Subpools:
  - **IMCU** Pool: enthält **I**n **M**emory **C**ompression **U**nits (IMCUs)
  - **SMU** Pool: enthält **S**napshot **M**etadata **U**nits (SMUs)
- IMCUs enthalten Daten im Column Data Format
- SMUs enthalten Metadaten und Informationen über Transaktionen

Folie 32

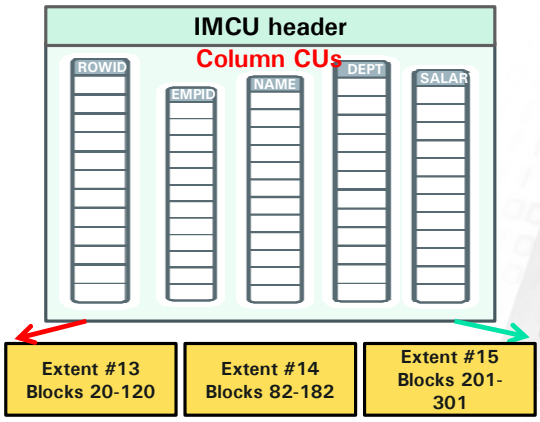
Info über **V$INMEMORY_AREA**

```
SELECT * FROM v$inmemory_area;
```

```
POOL        ALLOC_BYTES USED_BYTES POPULATE_STATUS                CON_ID
----------- ----------- ---------- ------------------------- ---------
MB POOL       854589440  594542592 DONE                              0
64KB POOL     201326592    2097152 DONE                              0
```

Folie 33

Aufbau der In-Memory Compression Unit (IMCU)



- Unit of column store allocation
  - Repräsentation der Spalten von einer als separat fortlaufende großen Anzahl von Zeilen eines Objektes
  - Zeilen von einem oder mehreren Table Extents
- Größe abhängig von Zeilengröße, Komprimierungsfaktor etc.
- jede Spalte gespeichert als **Column Compression Unit** (Column CU)
  - auch RowIds als Column CU

33

Folie 34

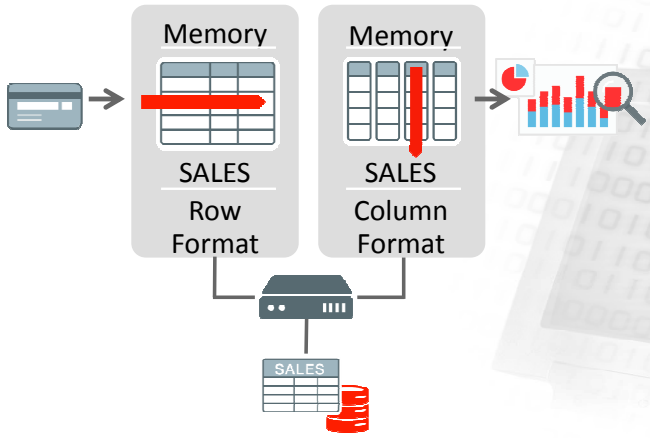
Prinzipielle Unterschiede zum traditionellen Buffer Cache

- Der Column Store ist statisch und folgt keinem LRU (Least Recently Used) Algorithmus
- Der Column Store ist ein reines Memory-Format
- Der Column Store ist geeignet für einheitliche Zugriffe (alle Zeilen einer Tabelle) => typisch für analytische Abfragen
- Der Buffer Cache hingegen ist vorgesehen für nicht einheitliche Zugriffe (z.B. einige Zeilen einer Tabelle) => typisch für OLTP
- Der Column Store ist mit zusätzlichen neuen Optimizerzugriffen und Komprimierungsalgorithmen ausgestattet

34

Folie 35

Das duale Format in der Datenbank



- **Beides** – Row und Column In-Memory-Formate – für die gleiche Tabelle
- Real-Time-Analysen nutzen das **neue** Column Format
- OLTP Applikationen nutzen das Row-Format
- Erhalt der Transaktionskonsistenz

35

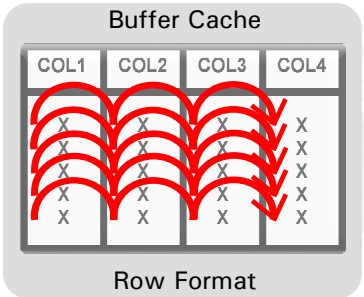
Folie 36

Buffer Cache versus In-Memory Column Store?

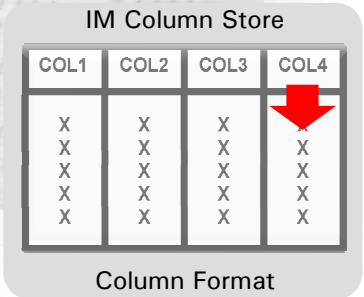
```
SELECT column4 FROM mytable;
```

RESULT

Mit Buffer Cache



Mit Column Store



36

Folie 37

Konfiguration in 2 Schritten

1. Memory Allokation in der SGA über neuen Initialisierungsparameter und danach Start der Datenbank
2. Auswahl der Objekte (Spalten, Tabellen, Materialized Views, Tablespace oder Partitionen) und Belegung mit den entsprechenden Eigenschaften
   - Über CREATE oder ALTER Kommando
   - u.U. überflüssige analytische Indizes löschen!
   - => Nicht alle Objekte müssen im Column Store vorliegen!

Folie 38

Beispiel für In-Memory Column Store Konfiguration

- Setzen Parameter **INMEMORY_SIZE** (mindestens 100 MB)
  - Default ist 0
- Gesamt SGA muss groß genug sein
  - **SGA_TARGET**
- Statischer Pool
  - Aufteilung in 64K und 1MB Pool
  - Monitoring über **V$INMEMORY_AREA** (Aufteilung der Pools)

```
SQL> show sga

Total System Global Area 2415919104
bytes
Fixed Size                  2927240 bytes
Variable Size             436209016 bytes
Database Buffers          889192448 bytes
Redo Buffers               14848576 bytes
In-Memory Area           1073741824 bytes
```

Folie 39

Festlegung der Objekte – Neue Schlüsselworte

- Ein- und Ausschalten einer gesamten Tabelle

```
ALTER TABLE sales INMEMORY ;
```

- Auf Spaltenebene

```
ALTER TABLE sales INMEMORY NO INMEMORY(prod_id, cust_id));
```

- Auf Partitionsebene

```
CREATE TABLE customers ...
PARTITION BY LIST
  (PARTITION p1 ... INMEMORY ,
  (PARTITION p2 ... NO INMEMORY );
```

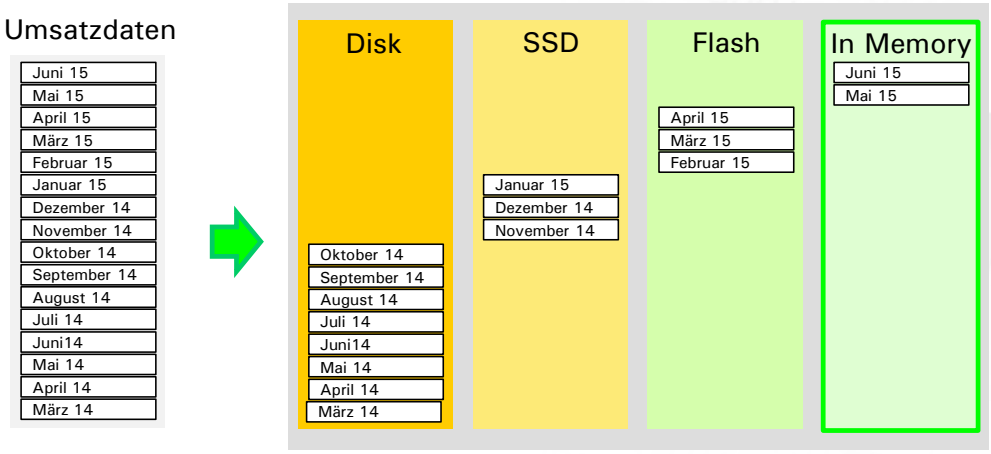
- Auf Tablespaceebene

```
CREATE TABLESPACE tbs1 DATAFILE 'tbs1.dbf' SIZE 400M
DEFAULT INMEMORY ;
```

39

Folie 40

Verteilung von Tabellendaten über Speicherhierarchien



40

Folie 41

Festlegung der Objekte mit Prioritätseinstellung

- Beeinflussung der Reihenfolge, wann Objekte geladen werden
  - **PRIORITY**
    - **CRITICAL**, **HIGH**, **MEDIUM**, **LOW** und **NONE**
    - **NONE** bedeutet beim ersten Zugriff
  - Beispiel:

```
ALTER TABLE lineorder
INMEMORY MEMCOMPRESS FOR QUERY LOW
PRIORITY CRITICAL ;
```

41

Folie 42

Wie funktioniert der Column Store?

- Wann wird im Column Store gespeichert?
  - nach dem ersten Zugriff oder gleich nach dem Hochfahren der Datenbank gemäß der Priority-Einstellung
- Welche Daten werden gespeichert?
  - ausgewählte Daten werden im optimierten und komprimierten Spaltenformat (siehe **MEMCOMPRESS**) gespeichert.
- Wer erledigt diese Aufgaben?
  - ein oder mehrere Hintergrundprozesse (gemäß Parameter **INMEMORY_MAX_POPULATE_SERVERS**) wie z.B. ORA_W001_orcl

42

Folie 43

Festlegung der Objekte mit Komprimierungsstufe

- Komprimierungs-einstellungen mit **MEMCOMPRESS FOR**
  - **DML**
  - **QUERY [LOW|HIGH]**
  - **CAPACITY [LOW|HIGH]**

```
CREATE TABLE ORDERS ...
PARTITION BY RANGE ...
 (PARTITION p1 ...
  INMEMORY NO MEMCOMPRESS ,
  PARTITION p2 ...
  INMEMORY MEMCOMPRESS FOR DML ,
  PARTITION p3 ...
  INMEMORY MEMCOMPRESS FOR QUERY ,
  ,... ,
  PARTITION p200 ...
  INMEMORY MEMCOMPRESS FOR
  CAPACITY HIGH
  );
```

43

Folie 44

Monitoring des Column Store

```
SELECT v.segment_name        AS "Name"
  , bytes/1024/1024          AS "Orig_Size MB"
  , inmemory_size/1024/1024  AS "InMem_Size MB"
  , bytes/v.inmemory_size    AS "Comp_Ratio"
  , inmemory_compression     AS "Compression"
  , bytes_not_populated      AS "not populated"
  , inmemory_priority        AS "Priority"
FROM v$im_segments ORDER by 4;
```

Name	Orig_Size MB	InMem_Size MB	Comp_Ratio	Compression	not populated	Priority
LINEORDER	1216	553.5	2.19692864	FOR QUERY LOW	0	CRITICAL
PART	64	8.125	7.87692308	FOR QUERY LOW	0	NONE
CUSTOMER	64	5.125	12.4878049	FOR QUERY LOW	0	NONE
DATE_DIM	64	1.125	56.8888889	FOR CAPACITY HIGH	0	NONE

44

Folie 45

Beispiele für verschiedene Komprimierungsstufen

- Tabelle **LINEORDER**
  - 1216 MB
  - 11 Mio. Zeilen

Compression	In Memory (MB)	Ratio
DML	1032	1.2
FOR QUERY LOW	554	2.2
FOR QUERY HIGH	401.5	3.0
CAPACITY LOW	357.5	3.4
CAPACITY HIGH	277.5	4.3

- Tabelle **CUSTOMER**
  - 64 MB
  - 60 000 Zeilen

Compression	In Memory (MB)	Ratio
DML	6.1	10.4
FOR QUERY LOW	5.1	12.4
FOR QUERY HIGH	4.1	15.5
CAPACITY LOW	2.12	30.1
CAPACITY HIGH	2.12	30.1

45

Folie 46

Komprimierungsstufen aus dem Handbuch

CREATE/ALTER Syntax	Description
NO MEMCOMPRESS	The data is not compressed.
MEMCOMPRESS FOR DML	This method optimizes the data for DML operations and compresses IM column store data the least (excluding NO MEMCOMPRESS).
MEMCOMPRESS FOR QUERY LOW	This method results in the best query performance. This method compresses IM column store data more than MEMCOMPRESS FOR DML but less than MEMCOMPRESS FOR QUERY HIGH. This method is the default when the INMEMORY clause is specified without a compression method in a CREATE or ALTER SQL statement or when MEMCOMPRESS FOR QUERY is specified without including either LOW or HIGH.
MEMCOMPRESS FOR QUERY HIGH	This method results in excellent query performance. This method compresses IM column store data more than MEMCOMPRESS FOR QUERY LOW but less than MEMCOMPRESS FOR CAPACITY LOW.
MEMCOMPRESS FOR CAPACITY LOW	This method results in good query performance. This method compresses IM column store data more than MEMCOMPRESS FOR QUERY HIGH but less than MEMCOMPRESS FOR CAPACITY HIGH. This method is the default when MEMCOMPRESS FOR CAPACITY is specified without including either LOW or HIGH.
MEMCOMPRESS FOR CAPACITY HIGH	This method results in fair query performance. This method compresses IM column store data the most.

46

Folie 47

Weitere Hinweise

- Handhabung im RAC-Umfeld
  - Column Store kann auf jedem RAC-Knoten existieren
  - Die gleiche Tabelle kann in jedem Column Store existieren (Option **DUPLICATE**)
  - Größere Tabellen, die nicht in den Column Store passen, können über die Knoten verteilt werden (Option **DISTRIBUTE**)
- Generelle Einschränkungen
  - Objekte, die kleiner als 64 KB sind, werden nicht berücksichtigt
  - Kein SYSTEM und SYS Objekte
  - Objekte wie IOTs, Cluster Tables, Global Temporary Tables und out-of-line LOBs
  - Option **DUPLICATE** ist nur auf Engineered System verfügbar

Folie 48

Ermittelung In-Memory Compression

- 2 Möglichkeiten: Compression Advisor oder In-Memory Advisor
- Compression Advisor:
  - Column Store muss vorab konfiguriert sein
  - Tests nur in 12.1.0.2 Umgebungen möglich
- In Memory Advisor:
  - nutzbar schon mit 11.2.0.3 Umgebungen
  - separate Installation erforderlich
  - Auswertung von ASH und AWR Daten

Folie 49

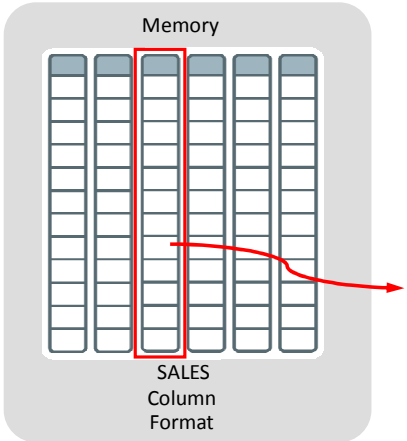
Zugriffe

- Was passiert beim Absetzen einer Abfrage mit Objekten im Column Store?
- Welche neuen Optimizerzugriffe gibt es?
- Was passiert im Fall von Mixed Workloads – OLTP und Abfragen?
  Wie kann die Transaktionskonsistenz beibehalten werden?
- Wie kann man die Nutzung monitoren?
- Was passiert, wenn nicht alle Objekte Platz im Column Store finden?
  Wird die Abfrage trotzdem optimiert?

49

Folie 50

Einfacher Zugriff mit In-Memory Scans



- Beispiel: Abfrage auf eine Spaltenuntermenge mit Filterung
  (z.B. IN, =, > <)

⇒ Direkter Zugriff auf entsprechende Spalte

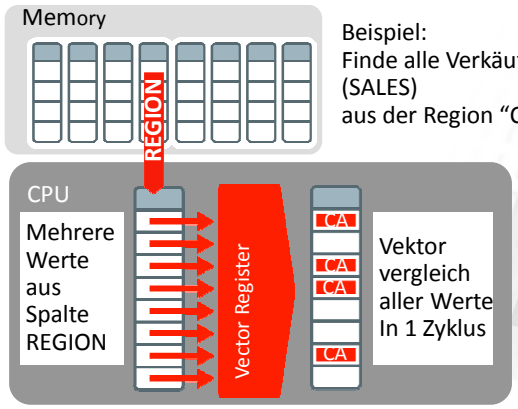
⇒ Scannen von geringerem Datenvolumen

⇒ Nutzen einer Art Spaltenindex
(Min und Max Wert)
=> Pruning

50

Folie 51

Zusätzliche Performance durch Nutzung von SIMD



Beispiel:
Finde alle Verkäufe (SALES)
aus der Region "CA"

- Jede CPU Core durchsucht lokal die In-Memory-Spalten
- Scans nutzen dabei "**S**ingle **I**nstruction **m**ultiple **D**ata Values" (SIMD) Vektor Operationen
- Hohe Scan Raten pro CPU Core => **Milliarden von Zeilen pro Sekunde**

> 100x Faster

51

Folie 52

Monitoring von In-Memory Scans – Ausführungsplan

- Schlüsselwort **INMEMORY FULL**

```
SELECT * FROM
table(dbms_xplan.display_cursor(format=>'BASIC'));
```

```
PLAN_TABLE_OUTPUT
------------------------------------------------------------------------
EXPLAINED SQL STATEMENT:
------------------------
select count(*) from customers_mem where cust_gender='F'
Plan hash value: 1973973873
-----------------------------------------------------
| Id  | Operation                   | Name          |
-----------------------------------------------------
|   0 | SELECT STATEMENT            |               |
|   1 |  SORT AGGREGATE             |               |
|   2 |   TABLE ACCESS INMEMORY FULL| CUSTOMERS_MEM |
-----------------------------------------------------
```

52

Folie 53

Monitoring – INMEMORY Table Scan

```
IM scan CUs memcompress basic
IM scan CUs memcompress for query
IM scan CUs memcompress for capacity low
IM scan CUs memcompress for capacity high
IM scan segments disk
IM scan bytes in-memory
IM scan bytes uncompressed
IM scan CUs columns accessed
IM scan CUs columns decompressed
IM scan CUs columns theoretical max
IM scan rows
IM scan rows valid
IM scan rows cache
IM scan blocks cache
IM scan fetches journal
IM scan rows journal
IM scan CUs split pieces
IM scan CUs predicates received
IM scan CUs predicates applied
IM scan CUs predicates optimized
IM scan CUs optimized read
IM scan CUs pruned
IM scan segments minmax eligible
IM scan CUs column not in memory
```

- Neue Session Level Statistiken mit Präfix IM
- Zeigen, ob und wie Column Store verwendet wird

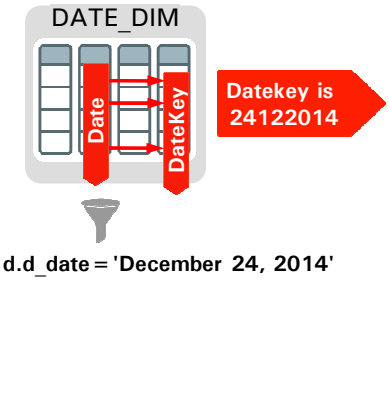
```
SELECT display_name, value
FROM   v$mystat m,
       v$statname n
WHERE  m.STATISTIC# =
             n.STATISTIC#
AND    display_name IN (
 'IM scan CUs columns access',
 'IM scan segments minmax
  eligible',
 'IM scan CUs pruned');
```

```
DISPLAY_NAME                          VALUE
------------------------------- -------
IM scan CUs columns accessed           46
IM scan CUs pruned                      0
IM scan segments minmax eligible       23
```

53

Folie 54

Memory Operationen – In-Memory Joins



- In-Memory Joins mit schnellen Scans und speziellen Filtern

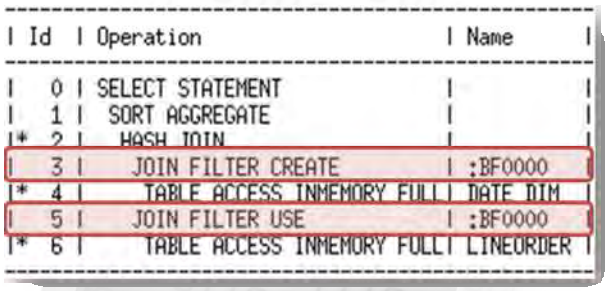
```
SELECT
  SUM(lo_extendedprice
      * lo_discount)
FROM  lineorder l,
      date_dim d
WHERE l.lo_orderdate =
      d.d_datekey
AND   l.lo_discount
      BETWEEN 2 AND 3
AND   d.d_date =
      'December 24, 2014';
```

54

Folie 55

Monitoring In-Memory Joins

- Spezieller Filter – Bloom Filter
  - während eines Hash Joins: Erzeugung eines Bitvektors (Bloomfilter) basierend auf Join-Bedingung
  - Bitvektor dient als zusätzliches Prädikat für den Scan der zweiten Tabelle



```
| Id  | Operation                    | Name        |
|   0 | SELECT STATEMENT             |             |
|   1 |  SORT AGGREGATE              |             |
|*  2 |   HASH JOIN                  |             |
|   3 |    JOIN FILTER CREATE        | :BF0000     |
|*  4 |     TABLE ACCESS INMEMORY FUL| DATE_DIM    |
|   5 |    JOIN FILTER USE           | :BF0000     |
|*  6 |     TABLE ACCESS INMEMORY FUL| LINEORDER   |
```

55

Folie 56

In-Memory Aggregat Operations

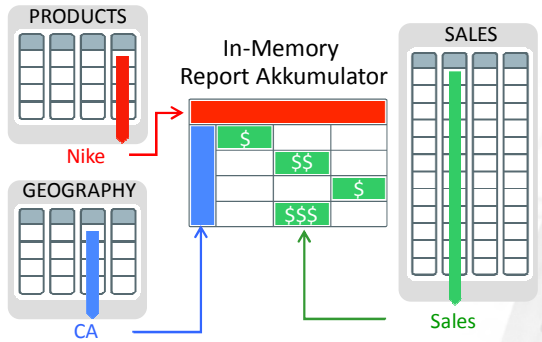
- optimiert Aggregat und Join-Operationen in einem typischen Star Schema
- nutzt KEY VECTOR und VECTOR GROUP BY Operationen
- Beispiel:

```
SELECT  p.category, p.subcategory, g.country
      , g.state, SUM(s.amount)
FROM  sales s, products p, geography g
WHERE s.geog_id = g.geog_id AND s.prod_id = p.prod_id
AND   g.state ='CA' AND p.pname = 'Nike'
GROUP BY category, subcategory, country, state;
```

56

Folie 57

In-Memory Aggregat Operations

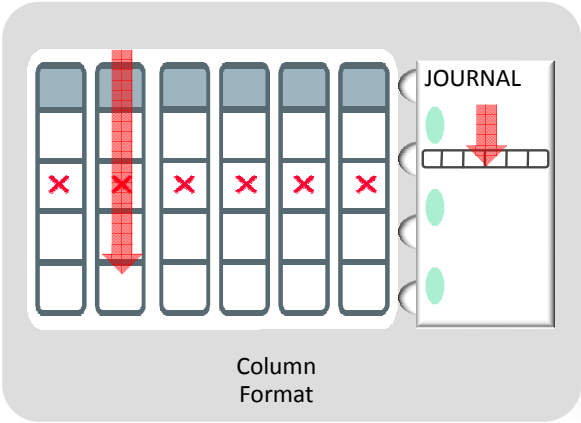


1. Phase
   - Scan der Dimension Tables
   - Key Vectors werden berechnet
   - Key Vectors bilden Multi- dimensionale Arrays – der In-Memory-Akkumulator
   - Temporäre Tabellen mit SELECT Liste
2. Phase
   - Scan der Fact Table und Anwendung der Key Vectors und Befüllung des Akkumalators
   - Join Back auf die temporäre Tabellen

57

Folie 58

DML und In-Memory Column Store



- Änderung im Buffer Cache löst automatisch Änderung im Column Store aus
  => Konsistenz
- Eintrag in IMCU wird als Stale markiert und eine Kopie des neuen Werts wird im In-Memory Transactions Journal gespeichert
- Aktualisierung der IMCU über einen Hintergrundprozess

58

Folie 59

Performance-Funktionen für In-Memory

- In-Memory SCANS: Where-Bedingung auf relevante Spalten (ohne decompress)
  In-Memory STORAGE INDEX: max/min (pruning), count, in, not in, between, ...
  In-Memory STORAGE INDEX: Metadata Dictionay-Pruning (Equality, In-List, ...)
- SIMD Vector Processing (Vector Register der CPU)
- In-Memory FILTER aus JOIN: BLOOM FILTER / VECTOR JOIN
- AGGREGATE (aus KEY VEKTOREN): VECTOR GROUP BY, VECTOR ORDER BY
- Ein Analyseschritt: SCAN(SIMD).FILTER.AGGREGATE

59

Folie 60

Der Oracle Optimizer entscheidet

- Optimizer erstellt den Ausführungsplan für die Abfragen und entscheidet, ob der In-Memory Column Store genutzt wird oder nicht
- beim traditionellen Kostenmodell wird davon ausgegangen, dass bei den Scan-Vorgängen die Daten von Platte gelesen werden
- Erweiterung des Kostenmodells mit 12.1.0.2 um In-Memory-Scans
- Kosten werden berechnet basierend auf den Statistiken:
  - Objekten: Tabellen, Spalten, Indizes, Partitionen etc.
  - System: CPU Speed, IO Durchsatz etc.
  - In-Memory Tabellen: In-Memory spezifische Statistiken

Statistiken müssen aktuell sein! Verantwortung des DBA

60

Folie 61

Neue In-Memory Statistiken

- automatische Berechnung während des Hard Parses on-the-fly
- auf Segment Level berechnet – Tabelle oder (Sub)Partition
  - **# IMCUs**
  - **# IM Blocks**
  - **IM Quotient**
    - Teil der Tabelle, die im In-Memory Column Store gespeichert ist
    - Wert zwischen 0 und 1
  - **# IM Rows**
  - **# IM Transaction Journal Rows**
- In-Memory Statistiken sind RAC-aware (DUPLICATE und DISTRIBUTE)

61

Folie 62

Kostenmodell umfasst auch In-Memory

- Kosten der In-Memory Scans
  - I/O-Kosten: Umfassen die Lesekosten:
    - Rows auf Disk
    - Extent Map
  - CPU-Kosten: Umfassen
    - Durchlaufen der IMCUs
    - IMCU Pruning mittels Nutzung der Storage Indizes
    - Dekomprimierung der IMCUs
    - Überprüfung der Prädikate
    - Zusammenbauen der Rows
    - Scannen der Zeilen im Transaction Journal

62

Folie 63

Prädikate werden bei den In-Memory Scans angewendet

- viele Typen der Filterprädikate können besser **während** des In-Memory-Scans evaluiert werden als **danach**.
  - nur die für die Abfrage relevanten Spalten werden benötigt
  - Prune IMCUs nutzen Storage Indexes und Dictionary-basierte Kompression Metadaten
  - Prädikate werden direkt gegen die komprimierte Column-Daten angewendet
  - SIMD wird beim Evaluieren der Prädikate bei mehreren Spaltenwerten verwendet
- Welche Prädikate werden bei der Überprüfung für In Memory verwendet?
  - Prädikate aus der WHERE-Clause
  - Prädikate, die sich aus der WHERE-Clause ableiten

63

Folie 64

Prädikate werden beim Scan auf In-Memory angewendet (Prädikat Push Down)

- Implizierte Prädikate-Generierung
  - Beispiele:
    col1 = 2 OR (col1 = 1 AND col2 > 1) ⇒ col1 IN (1, 2)
    col1 IN (1, 2) wird beim In-Memory-Scan angewendet
  - Anwendung wie normale Prädikate für Pruning & Evaluation während des In-Memory Scans
- Prädikat Push Down ersichtlich in 10053 Trace
- Joins werden auch “gepushed” via Bloom Filter bei seriellen & parallelen Abfragen

64

Folie 65

Abfragekriterien – 1

Abfrage-Kriterium	Beschreibung	Beispielabfrage 1 unvorteilhaft	Beispielabfrage 2 vorteilhaft
Anzahl der selektieren Spalten	Je mehr Tabellenspalten selektiert werden umso mehr übersteigen die Verarbeitungskosten die Abfragezeit.	SELECT * FROM Sales;	SELECT revenue FROM Sales;
		Unvorteilhaft, da die Abfrage alle Spalten selektiert und viel Zeit für die Spaltenverarbeitung verwendet wird im Verhältnis zu den Zeilen, die die Abfrage u.U. zurückliefert.	Vorteilhaft, da nur eine Spalte selektiert wird, wenig Zeit für die Spaltenverarbeitung verwendet in Relation zu den Zeilen, die zurückgeliefert werden.

65

Folie 66

Abfragekriterien – 2

Abfrage-Kriterium	Beschreibung	Beispielabfrage 1 unvorteilhaft	Beispielabfrage 2 vorteilhaft
Anzahl der Rückgabewerte	Je größer die Anzahl der Rückgabewerte einer Abfrage ist, umso geringer ist der Benefit des IM Column Stores. Die Übergabe der Daten an den Client übersteigt die „Kosten“.	SELECT revenue FROM sales;	SELECT sum(revenue) FROM sales
		Unvorteilhaft, da die Abfrage jeden Wert einer Zeile einer Tabelle zurückgibt.	Vorteilhaft, da die Abfrage nur einen Wert zurückgibt, obwohl sie jeden Wert innerhalb der Spalte verarbeitet.

66

Folie 67

Weitere Abfragekriterien...

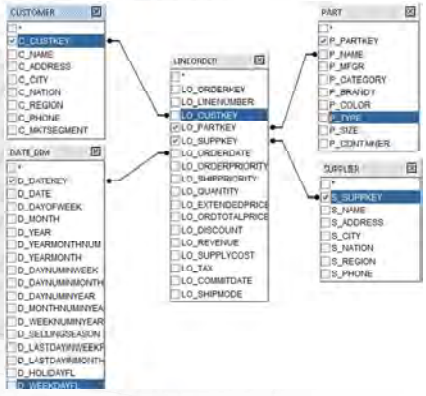
- Selektivität von Spaltenprädikaten
- Selektivität von Join-Bedingungen
- Anzahl der Tabellen, die gejoined werden
- Komplexität von SQL-Funktionen
- siehe im White Paper
  „When to Use Oracle Database In-Memory“

Folie 68

Oracle Database In-Memory – Beispiel

- Abfragen nach ...
  - höchster Bestellwert
  - Umsatz nach Kunde, Versandsart, Priorität
  - Summe Kundenumsätze, Monat, Rabatt, Menge
  - ...

LINEORDER	3.000.028.242 Rows
ROW STORE COMPRESS BASIC	23.101.440 Blocks a' 8KB
LINEORDER Size	180 GB
Column Store Size (query_low)	163 GB



Folie 69

Oracle Database In-Memory – Beispiel: 2 SELECTs – Ausführungszeiten

```
SELECT count(*) FROM lineorder;
```

SQL | Alle Zeilen abgerufen: 1 in 0,639 Sekunden

	COUNT(*)
1	3000028242

```
SELECT /*+ PARALLEL (1,60)*/
sum(lo_revenue), max(lo_discount)
FROM lineorder
WHERE lo_shipmode = 'RAIL' AND lo_priority = '2-HIGH' ;
```

SQL | Alle Zeilen abgerufen: 1 in 0,796 Sekunden

	SUM(LO_REVENUE)	MAX(LO_DISCOUNT)
1	311286408795491	10

69

Folie 70

Oracle Database In-Memory – Beispiel: Ausführungsplan 2. Select

OPERATION	OBJECT_NAME	CARDINALITY	COST	LAST_CR_BUFFER_GETS
SELECT STATEMENT			1105	
SORT (AGGREGATE)		1		25
PX COORDINATOR				25
PX SEND (QC (RANDOM))	:TQ10000	1		0
SORT (AGGREGATE)		1		0
PX BLOCK (ITERATOR)		85651955	1105	0
TABLE ACCESS (INMEMORY FULL)	LINEORDER	85651955	1105	0
Access Predicates				
AND				
:Z>=:Z				
:Z<=:Z				
AND				
LO_SHIPMODE='RAIL'				
LO_ORDERPRIORITY='2-HIGH'				
Filter Predicates				
AND				
LO_SHIPMODE='RAIL'				
LO_ORDERPRIORITY='2-HIGH'				

70

Folie 71

Fazit

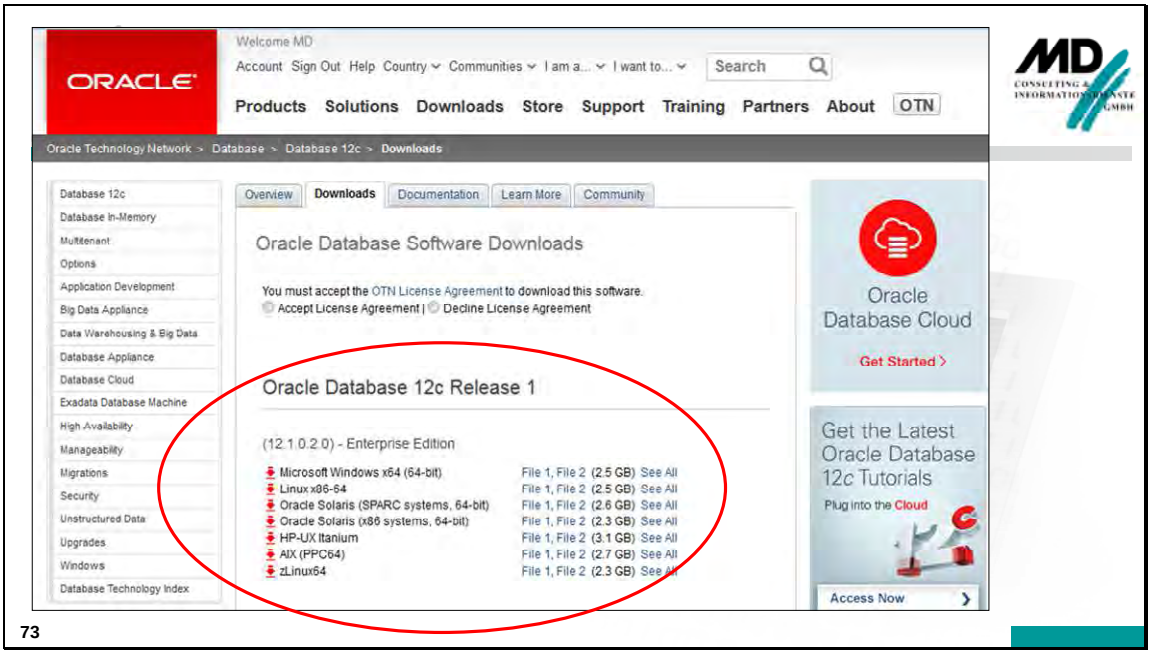
- kostenintensive Abfragen, die aus Table Scans resultieren, identifizieren (Anwender, AWR, Advisor)
- einige Beispiele für typische Abfragen, die vom Im Column Store profitieren:
  - Scannen von vielen Zeilen und Anwendung von Filtern wie z.B. =, <, >, und IN etc.
  - Abfragen einer Teilmenge von Spalten einer Tabelle (5 von 100 Spalten)
  - Joins von kleinen Tabellen mit einer großen Tabelle, wobei die Filter viele Zeilen herausfiltern
  - Abfragen die Aperationen/Aggregationen verwenden wie min, max und sum, group by, distinct und count(distinct())
- unbedingt auch weiteren Tuning Techniken mitberücksichtigen
  - Statistikmanagement, Plan Baseline, Result Cache etc.

Folie 72

Oracle Datenbank 12*c* In-Memory Option

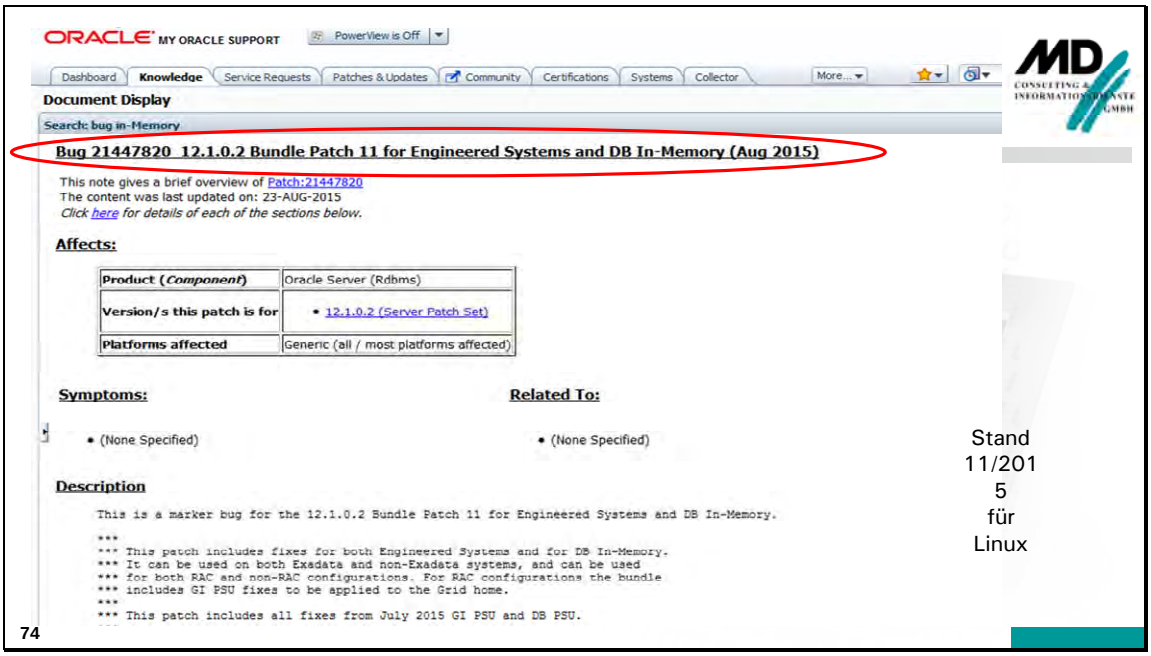
- ✓ Tiefe Integration in den Oracle Kernel
- ✓ In-Memory Area ist Teil der System Global Area (SGA)
- ✓ Integraler Teil des Oracle Eco-Systems
- ✓ Einfache Aktivierung und De-Aktivierung
- ✓ Dualer Ansatz (Row und Column) In-Memory Repräsentation der Objekte
- ✓ Zusätzliche Zugriffs-/Verarbeitungs-Operationen (SIMD, Vektor-Operationen)
- ✓ Analytic/Reporting sowie OLTP und Analytic/Reporting profitieren

Folie 73



73

Folie 74



Stand 11/2015 für Linux

74

Folie 75

Informationen

- Handbücher
  - Concepts Guide
  - Administrator's Guide
  - Performance Tuning Guide
- Oracle Database In-Memory An Oracle White Paper (July 2015)
  - www.oracle.com_technetwork_database_in-memory_overview_twp-oracle-database-in-memory-2245633.pdf
- In Memory Blog von Maria Colgan
  - https://blogs.oracle.com/in-memory/

Folie 76

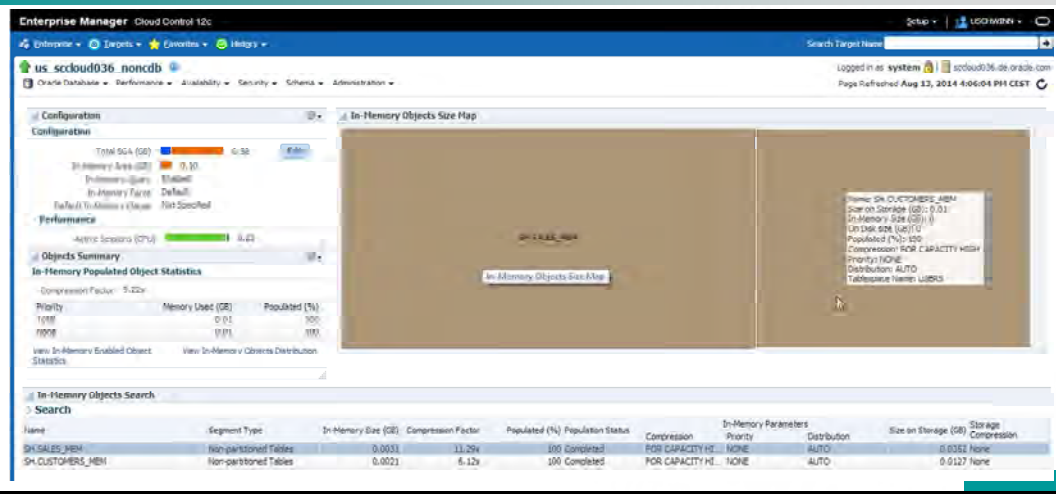
SEMINARE

4

Infos zu In-Memory
mit Oracle Cloud Control

Folie 77

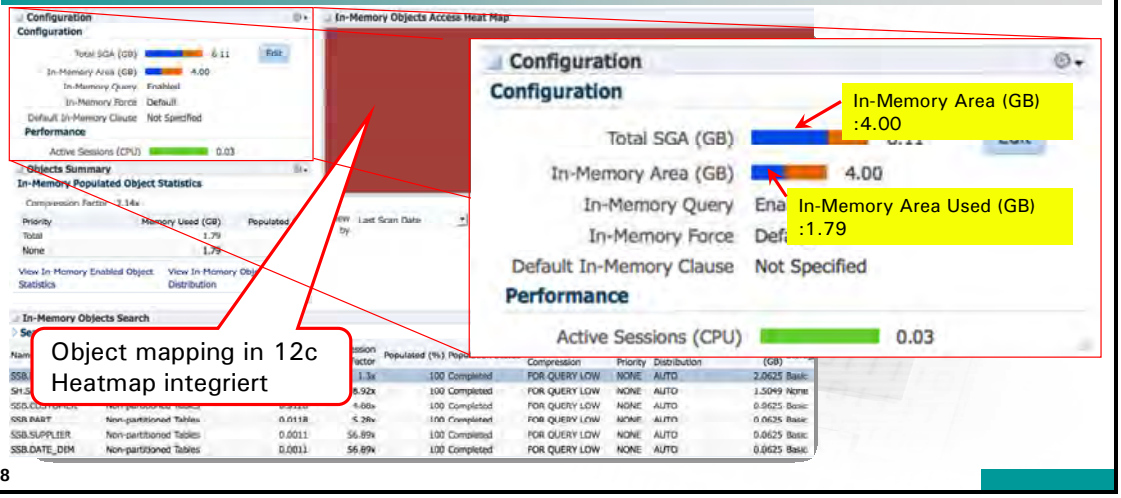
Monitoring im Cloud Control – 1



77

Folie 78

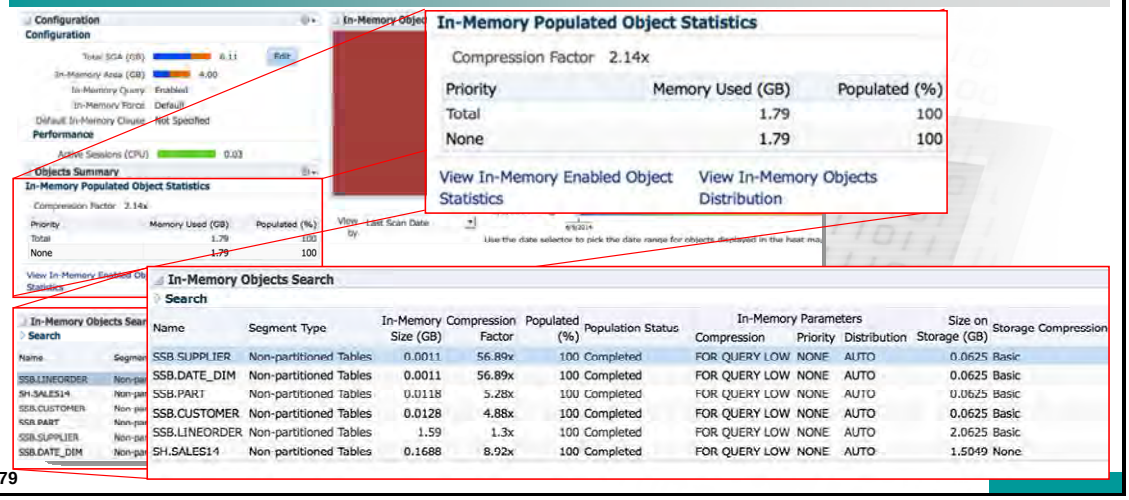
Monitoring im Cloud Control – 2



78

Folie 79

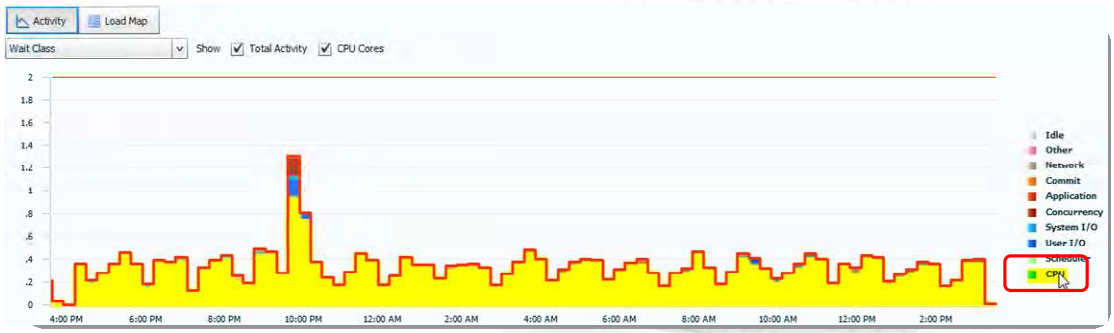
Monitoring im Cloud Control – 3



79

Folie 80

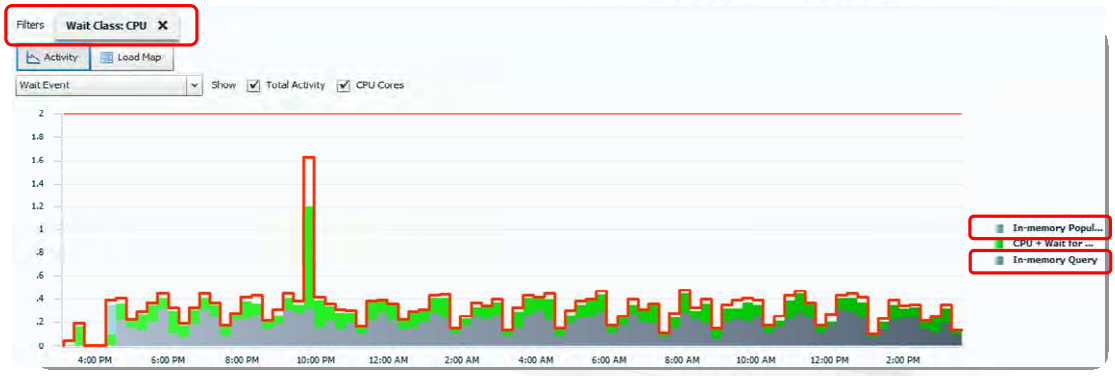
Neue In-Memory Einträge in CPU Wait Class



80

Folie 81

Neue In-Memory Einträge in CPU Wait Class



81

Folie 82

SQL Monitoring

Line ID	Operation	Name	Estimated Rows	Memory...	IO...	Activity %	Cost	Timeline(3s)
0	SELECT STATEMENT							
1	SORT GROUP BY		280	24KB			3,314	
2	HASH JOIN		230K	2MB		50	3,308	
3	TABLE ACCESS INMEMORY FULL	DATE_DIM	2,556				1	
4	HASH JOIN		230K	1MB			3,307	
5	JOIN FILTER CREATE	:BF0000	2,446				2	
6	TABLE ACCESS INMEMORY FULL	SUPPLIER	2,446				2	
7	HASH JOIN		1,129K	2MB			3,301	
8	JOIN FILTER CREATE	:BF0001	24K				62	
9	TABLE ACCESS INMEMORY FULL	PART	24K				62	
10	JOIN FILTER USE	:BF0000	36M				3,146	
11	JOIN FILTER USE	:BF0001	36M				3,146	
12	TABLE ACCESS INMEMORY FULL	LINEORDER	36M			50	3,146	

Cpu: in memory: 1 samples (50%)

82

Folie 83

SEMINARE

5

In-Memory – Ein Blick über den Zaun

Folie 84

SAP HANA

- Compatibility
  - Applikationen müssen umgeschrieben werden
  - Dynamic Tiering erfordert das Management zur Datenplazierung und des Zugriffs
- OLTP Scalability
  - noch keine Performance Benchmarks
- High Availability/Disaster Recovery
  - mit Dynamic Tiering ist HA manuell / DR über Backup und Restore
- VLDB Support
  - Scale out clustering erfordert weiterhin Connection Management und Application Co-Location

Folie 85

Microsoft SQL Server

- Compatibility
  - Applikationen müssen umgeschrieben werden zur Behandlung des Optimistic Locking Model und der Restriktionen ref. Integrität
- OLTP Scalability
  - In-Memory OLTP Option nicht ausgelegt für Real Time Analytics
  - Columnstore Indexes ausgelegt für DW, nicht für Real Time Analytics
- High Availability/Disaster Recovery
  - erreicht nicht Oracle's Maximum Availability Architecture Feature
- VLDB Support
  - In-Memory OLTP Option ist begrenzt bzgl. Memory und CPU

85

Folie 86

IBM DB2 BLU

- Compatibility
  - Application Code muss nicht geändert werden
- OLTP Scalability
  - Limits bei Verwendung column-organized Tables aufgrund Columnar Format
  - Replication Overhead bei Nutzung Shadow Tables
- High Availability/Disaster Recovery
  - HA eingeschränkt seit unsupported DB2 pureScale
- VLDB Support
  - Begrenzung Scale Out auf einzelnen Knoten seit unsupported Clustering mit DB2 pureScale

86

Folie 87

Zusammenfassung des Vergleichs

- nur Oracle Database In-Memory unterstützt voll alle 4 Vergleichskriterien
  - Compatibility
    - Applikationen komplett transparent
  - OLTP Scalability
    - Das duale Format des Database In-Memory liefert die höchste OLTP-Skalierbarkeit und Real Time Analytics
  - High Availability/Disaster Recovery
    - Oracle hält Spitzenplatz der Lösungen zu HA und DR
  - VLDB Support
    - ausschließlich Nutzung der Datenbank zum transparenten Scale Out der Applikationen über Cluster-Knoten

Folie 88

SEMINARE

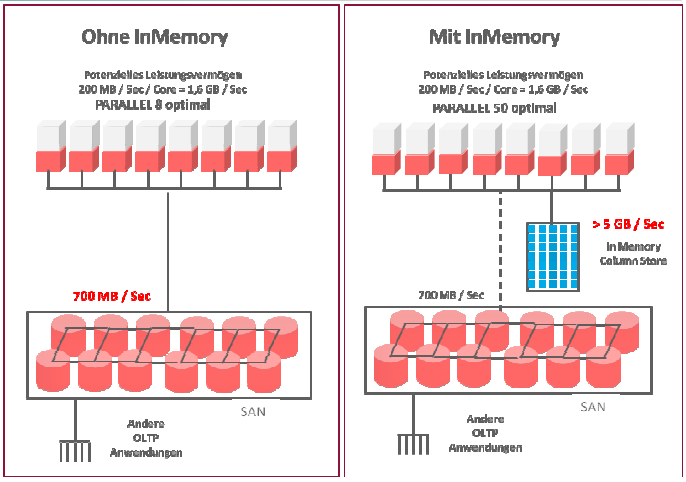
6

Auswirkungen auf Serverarchitekturen

Folie 89

Ist In-Memory nur etwas für große Systeme?

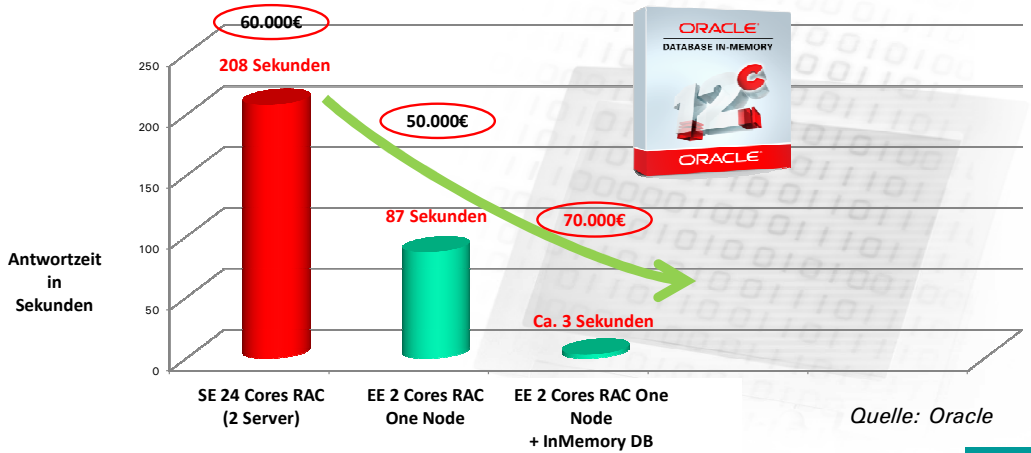
- **Nein**
- vor allem kleinere Umgebungen profitieren
- sie haben oft kein eigenes Storage-System, sondern hängen am SAN und konkurrieren mit vielen OLTP-Anwendungen
- das SAN wirkt oft als „IO-Bremse“
- mit In-Memory machen sich kleinere Systeme IO-unabhängig!



Quelle: Oracle

Folie 90

Kapitalkosten –
Vergleich DB 11gR2/12c Ed. SE - EE (ODA)
Wirkungen durch Parallelisierung, … und In-Memory DB Option



Quelle: Oracle

Folie 91

- Datenbank wahlweise im Hauptspeicher
- Spaltenorientiert
- Komprimiert
- Parallelisierung
- Nutzt bestehende DB-Architektur und Funktionalität
- Keine Änderung der Anwendung

Ein Schalter und es funktioniert!!

91

Folie 92

SEMINARE

7

Demonstrationen Oracle 12c In-Memory

Abschluss

- Abschlussbesprechung
- Urkundenausgabe
- Beurteilung des Workshop