

Senkrecht oder waagerecht?

Row Store vs. Column Store

Eero Mattila
Principal Systems Consultant
Quest Software GmbH, Köln

Agenda

- Wer bin ich?
- Begrifflichkeiten
- Oracle In-Memory Column Store – Architektur und Handhabung
- Was habe ich von der In-Memory Option?
- Anwendungsfälle für In-Memory Column Store
- Fazit

Wer bin ich?

- **Angefangen mit Oracle V6 im Jahr 1991**
 - DBA, Forms-/Reports-Entwickler
- **Oracle Deutschland 1995**
 - RDBMS, Forms, Reports, Designer
- **Quest seit 2005**
 - Oracle und SQL Server Tools – Toad, Spotlight, SQL Optimizer
 - Datenbankreplikation – SharePlex
 - Performance Monitoring – Foglight

Begrifflichkeiten

- Verwirrung durch mehrfach besetzte Begriffe
- Row Store / Column Store (zeilen- bzw. spaltenorientiert)
 - Art der physikalischen Speicherung
 - Art der Verarbeitung im Hauptspeicher
 - Store = Massenspeicher oder flüchtiger Speicherbereich
- In-Memory
 - Gesamte Datenbank oder nur ein Teil der Daten
 - Zeilenorientiert oder spaltenorientiert – oder teils, teils

Begrifflichkeiten (2)

- Zeilenorientierte Tabellen

- Meist mehrere Spalten
- Ein Datensatz (Zeile, Row, Record) enthält unterschiedliche Informationen
- Alle Attribute zusammen gespeichert -> schneller Zugriff auf alle Spalten
- Ideal, wenn viele Spalten aber wenige Zeilen verarbeitet werden
- Effizientes DML -> optimal für transaktionsorientierte Anwendungen

ID	VORNAME	NACHNAME	STRASSE	ORT	EMAIL
10001	Andi	Arbeit	Hauptstraße	Köln	andi.arbeit@firma.com
10002	Ellen	Lang	Holzweg	Bonn	ellen.lang@laden.org
10003	Lilly	Puth	Sackgasse	Oberpleis	lilly.puth@schuppen.de

Begrifflichkeiten (3)

- Spaltenorientierte Speicherung

- Jede Spalte wird in einer eigenen Struktur gespeichert
- Effektive Komprimierungsmöglichkeiten
- Ideal, wenn wenige Attribute, aber viele Zeilen verarbeitet werden
- Optimal für analytische Anwendungen (Data Warehouse/Mart/Mining)
- DML aufwendig, weil alle Spaltenstrukturen einzeln angefasst werden müssen

ID	VORNAME	NACHNAME	STRASSE	ORT	EMAIL
10001	Andi	Arbeit	Hauptstraße	Köln	andi.arbeit@firma.com
10002	Ellen	Lang	Holzweg	Bonn	ellen.lang@laden.org
10003	Lilly	Puth	Sackgasse	Oberpleis	lilly.puth@schuppen.de

Begrifflichkeiten (4)

- RDMBS wie Oracle verwenden traditionell Zeilenorientierte Speicherung
 - Optimiert für Transaktionen
- Erste Spaltenorientierte Datenbanken bereits in den 1970-er Jahren
 - RAPID der kanadischen Bundesbehörde für Statistiken
- Erstes kommerzielles Produkt wohl Sybase IQ (heute SAP IQ) um 1995
 - Mittlerweile zahlreiche sowohl proprietäre als auch Open Source Lösungen
- In-Memory Datenbanken (= alles in Memory!) auch nichts Neues
 - z.B. Oracle TimesTen
 - zeilenorientiert!
 - für hochfrequentierte OLTP-Anwendungen
 - extrem schnelle Antwortzeiten

Oracle In-Memory Column Store

- Anderer Ansatz
 - Persistente Speicherung bleibt zeilenorientiert
 - Buffer Cache wie gehabt (blockorientiert)
 - In-Memory Column Store ein neuer, zusätzlicher Bereich in der SGA
 - Optimierung analytischer Abfragen
 - Flüchtig
- Nicht alle Tabellen müssen (und gehören) in den Column Store
- Kein zusätzlicher Massenspeicher erforderlich
 - Optional möglich – dazu später mehr
- Zusätzlicher Hauptspeicher für die SGA im Zweifel nötig

Oracle In-Memory Column Store - Handhabung

- Immer installiert, aber nicht automatisch aktiviert
 - schließlich kostenpflichtig...
- Im Zweifel SGA erweitern
- `alter system set inmemory_size = 250G scope=spfile;`
- Restart der Instanz
- Ab Oracle12g R2 kann der Column Store dynamisch vergrößert (sofern Platz in der SGA), aber nicht verkleinert werden

Oracle In-Memory Column Store – Handhabung (2)

```
select name, value from v$sga;
```

NAME	VALUE
-----	-----
Fixed Size	87612324
Variable Size	8388609129
Database Buffers	32184563284
Redo Buffers	801587242
In-Memory Area	268435455328

Oracle In-Memory Column Store – Handhabung (3)

- Bei AMM (mit `memory_target`) oder ASMM (mit `sga_target`) wird die Größe des Column Store nicht angepasst
- Kein Cache – kein LRU Algorithmus – kein *aging out*!
- Ist der Column Store voll belegt, wird eine neue Tabelle nicht (oder nicht komplett) geladen
 - Alert.log: Insufficient memory to populate table to inmemory area
 - ABER: welche Tabelle denn?

Oracle In-Memory Column Store – Handhabung (4)

```
select v.segment_name name,  
       v.populate_status pop_status,  
       v.bytes_not_populated missing_bytes  
from v$im_user_segments v  
order by 1;
```

NAME	POP_STATUS	MISSING_BYTES
ADRESSEN	COMPLETED	0
AUFTRAEGE	COMPLETED	0
MIST	OUT OF MEMORY	1034231808
PERSONEN	COMPLETED	0
POSITIONEN	COMPLETED	0

Oracle In-Memory Column Store – Handhabung (5)

- Was kann in den Column Store geladen werden?
 - Tabellen, Partitionen, Subpartitionen, Materialized Views
- Wie werden Objekte in den Column Store geladen?
 - Am einfachsten (wenn auch nicht unbedingt am besten) so:
 - `create table <tabelle> [...] INMEMORY;`
 - `alter table <tabelle> INMEMORY;`

Oracle In-Memory Column Store – Handhabung (6)

- Die einfache Syntax: Oracle entscheidet, wann und in welcher Reihenfolge die Objekte in den Column Store geladen werden
 - Beim ersten Zugriff (SELECT oder DML)
- Lieber immer den Parameter PRIORITY verwenden
 - `[...] INMEMORY CRITICAL;`
 - `[...] INMEMORY HIGH;`
 - `[...] INMEMORY MEDIUM;`
 - `[...] INMEMORY LOW;`
 - `[...] INMEMORY NONE; -- STANDARD`

Oracle In-Memory Column Store – Handhabung (7)

- Mit `PRIORITY != NONE` wird ein Objekt sofort in den Column Store geladen
- Zuerst alle Objecte mit `CRITICAL`, dann `HIGH` usw.
- Beim Neustart der Instanz werden alle Objekte mit `PRIORITY != NONE` automatisch in den Column Store geladen
- Sinnvoll: Die wichtigsten Objekte mit `CRITICAL` und die dazugehörigen mit `HIGH` versehen.

Oracle In-Memory Column Store – Handhabung (8)

- Zusätzliche Parameter für
 - Komprimierung
 - Optimierung für Abfrage-Performance oder Größe (*Capacity*)
 - RAC und Active Data Guard
 - RAC: DISTRIBUTE und DUPLICATE (nur Engineered Systems!)
 - Obacht – Verteilung ist Standard und potentieller Performance-Killer
 - ADG: Column Store für sowohl Primary als auch Standby möglich

Oracle In-Memory Column Store – Handhabung (9)

- Nicht alle Spalten einer Tabelle müssen in den Column Store!
- `alter table TABELLE inmemory (sp1, sp2) no inmemory (sp3, sp4);`
- Wir erinnern uns: Die Größe des Column Stores ist nicht dynamisch. Sparsamer Umgang ist sinnvoll...
- Nur Spalten, die oft abgefragt werden, ob in SELECT oder WHERE-Clause, gehören in den Column Store. Eigentlich nichts Neues...

Oracle In-Memory Column Store – Handhabung (10)

- Der Column Store ist flüchtig und muss nach Neustart befüllt werden. Das ist CPU-intensiv!
- Ab Oracle12c R2 kann der Column Store auch persistiert werden.
- `exec dbms_inmemory_admin.faststart_enable('FS_TBS');`
- Der Inhalt des Column Store wird in einem dedizierten Tablespace gespeichert.
- Nach Neustart ist die Befüllung wesentlich schneller, da die CPU-intensive Umformatierung von Row auf Column Format entfällt

Oracle In-Memory Column Store – Handhabung (11)

- Derzeit 19 V\$-Views für die In-Memory Option. Zum Beispiel:

```
select v.segment_name name,  
       v.populate_status pop_status,  
       v.bytes/1024/1024/1024 orig_GB,  
       v.inmemory_size/1024/1024/1024 in_mem_GB,  
       v.bytes / v.inmemory_size comp_ratio,  
       v.bytes_not_populated missing_bytes  
from v$im_user_segments v  
order by 1;
```

NAME	POP_STATUS	ORIG_GB	IN_MEM_GB	COMP_RATIO	MISSING_BYTES
ADRESSEN	COMPLETED	0,01953125	0,008056640625	2,42424242424242	0
AUFTRAEGE	COMPLETED	0,2705078125	0,186767578125	1,4483660130719	0
PERSONEN	COMPLETED	0,0087890625	0,004150390625	2,11764705882353	0
POSITIONEN	COMPLETED	1,5	0,45611572265625	3,28863910076275	0

Was habe ich von der In-Memory Option?

- Enorme Steigerung der Abfrage-Performance
- Winziges Beispiel: Auftragsverwaltungssystem, „größte“ Tabelle mit ca. 25 Mio Zeilen und ca. 1,5 GB auf Festplatte.
- Quick and Dirty Abfrage: Wieviel Umsatz hatten wir mit Bademoden?

```
SELECT produktname, SUM (auftragssumme) Umsatz
  FROM ( SELECT a.aufid, produktname, menge * einzelpreis auftragssumme
         FROM doag.positionen pos,
              doag.auftraege a,
              doag.produkte pro,
              doag.status s
        WHERE      a.aufstatus = s.statusid
                 AND (      (      (RTRIM (TO_CHAR (aufdatum, 'Month')) IN
                                   ('Juli', 'August', 'September'))
                           AND (TO_CHAR (aufdatum, 'yyyy') IN ('2016', '2017'))))
                 AND (produktname IN ('Badehose', 'Badeanzug'))
                 AND S.kurzbeschreibung = ('GELIEFERT')
                 AND pos.aufid = a.aufid
                 AND pos.prodid = pro.prodid
        GROUP BY a.aufid, produktname, menge * einzelpreis)
GROUP BY produktname;
```

Was habe ich von der In-Memory Option? (2)

- Zum Testen: Hint `/*+ NO_INMEMORY */`
- Mit dem Hinweis braucht die Abfrage ca. 7 Sekunden

Database	Start Time	Stop Time	Execution Time	SQL
DOAG@EM12	20/11/2017 17:27:55	20/11/2017 17:28:03	7 secs	SELECT /*+ NO_INMEMORY */ produktname, SUM (auftragssumme) Umsatz

- Ohne den Hinweis, d.h. mit Nutzung des Column Store knapp eine halbe Sekunde

Database	Start Time	Stop Time	Execution Time	SQL
DOAG@EM12	20/11/2017 17:26:05	20/11/2017 17:26:05	392 msecs	SELECT produktname, SUM (auftragssumme) Umsatz

Was habe ich von der In-Memory Option? (3)

- Der Ausführungsplan zeigt eine neue Zugriffsmethode, **TABLE ACCESS INMEMORY FULL**

```
-----  
| Id | Operation | Name | Rows | Bytes | TempSpc | Cost (%CPU) | Time |  
-----  
| 0 | SELECT STATEMENT | | 2 | 52 | | 5778 (16) | 00:00:01 |  
| 1 | HASH GROUP BY | | 2 | 52 | | 5778 (16) | 00:00:01 |  
| 2 | VIEW | VM_NWVW_0 | 3253 | 84578 | | 5778 (16) | 00:00:01 |  
| 3 | HASH GROUP BY | | 3253 | 231K | 280K | 5778 (16) | 00:00:01 |  
|* 4 | HASH JOIN | | 3253 | 231K | | 5718 (16) | 00:00:01 |  
|* 5 | TABLE ACCESS FULL | STATUS | 1 | 11 | | 3 (0) | 00:00:01 |  
|* 6 | HASH JOIN | | 16264 | 984K | 7312K | 5715 (16) | 00:00:01 |  
| 7 | JOIN FILTER CREATE | :BF0000 | 16264 | 984K | | 5715 (16) | 00:00:01 |  
|* 8 | HASH JOIN | | 138K | 5683K | | 3364 (15) | 00:00:01 |  
| 9 | JOIN FILTER CREATE | :BF0001 | 2 | 36 | | 3 (0) | 00:00:01 |  
|* 10 | TABLE ACCESS INMEMORY FULL | PRODUKTE | 2 | 36 | | 3 (0) | 00:00:01 |  
| 11 | JOIN FILTER USE | :BF0001 | 25M | 593M | | 3295 (13) | 00:00:01 |  
|* 12 | TABLE ACCESS INMEMORY FULL | POSITIONEN | 25M | 593M | | 3295 (13) | 00:00:01 |  
| 13 | JOIN FILTER USE | :BF0000 | 824K | 15M | | 745 (58) | 00:00:01 |  
|* 14 | TABLE ACCESS INMEMORY FULL | AUFTRAEGE | 824K | 15M | | 745 (58) | 00:00:01 |  
-----
```

Was habe ich von der In-Memory Option? (4)

- Bereits das winzige Beispiel (25 Mio Zeilen, 1,5 GB) erweist eine Performance-Steigerung um weit mehr als Faktor 10
- Berichte über realistische Umgebungen belegen einhellig Steigerungen von mindestens Faktor 10, im Einzelfall 50 und mehr
- Keine Änderung der bestehenden Anwendung erforderlich
- Analytische Indizes sind nicht mehr nötig

Anwendungsfälle für In-Memory Column Store

- Primäres Ziel: Optimierung von analytischen Abfragen
 - Data Warehouse, Data Mart, Data Mining usw.
- Volatile Anwendungen mit viel OLTP und Bedarf an Analyse
- Faustregel: Je mehr Daten *gelesen* werden im Verhältnis zu den Daten, die *verarbeitet* (oder einfach zurückgegeben) werden, umso größer der potenzielle Nutzen!

Anwendungsfälle für In-Memory Column Store (2)

- **Anzahl der zu selektierenden Spalten.** Je mehr Spalten abgefragt werden, umso aufwändiger ist die Verarbeitung und geringer der Gewinn.

```
SELECT * ist schlecht, SELECT spalte ist besser.
```

- **Anzahl der zurückgegebenen Zeilen.** Je mehr Zeilen zurückgegeben werden, umso aufwändiger ist die Verarbeitung und geringer der Gewinn.

```
SELECT spalte ist schlecht, SELECT SUM(spalte) ist besser.
```

- **Selektivität der Abfrageprädikate.** Je mehr Zeilen die Abfragekriterien erfüllen, umso aufwändiger ist die Verarbeitung und geringer der Gewinn.

```
SELECT AVG(zahl) ... WHERE zahl > 2 ist schlecht,  
SELECT AVG(zahl) ... WHERE zahl < 2 ist besser (wenn „zahl“ meist größer als 2 ist...)
```

- **Anzahl der Tabellen und Joins in der Abfrage.** Je komplexer die Abfrage, umso aufwändiger ist die Verarbeitung und geringer der Gewinn.

```
SELECT ... from a,b,c,d,e,f,g WHERE <JOIN-Bedingungen) ist schlecht,  
SELECT ... from a,b,c WHERE <JOIN-Bedingungen) ist besser.
```

Fazit

- Row Store oder Column Store? Sowohl als auch!
- Keine Änderung der physikalischen Speicherung
- Hohes Potenzial an Performance-Gewinn für **analytische** Abfragen
- Keine Änderung der bestehenden Anwendung erforderlich
- Verzicht auf analytische Indizes -> Bessere Performance der OLTP-Leistung
- Anwendung verstehen – Testen!

Thank you!

eero.mattila@quest.com

Quest™