# Oracle In-Memory Database Option und die Folgend für das Data Warehouse

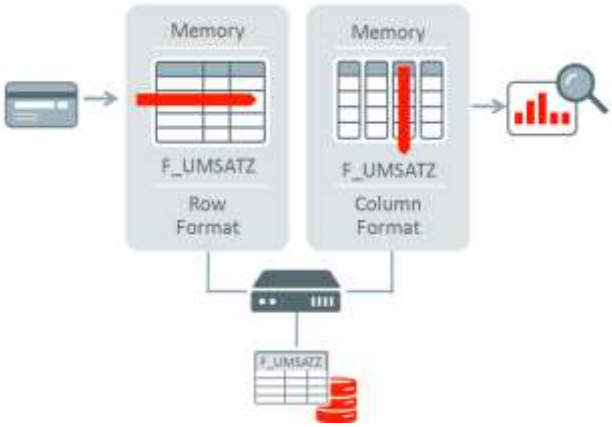
Seit einigen Jahren treibt die In-Memory Diskussion schon ihre Blühten. Auf Konferenzen wurde sogar ein eigenes Produktsegment – Analytische Datenbanken – kreiert und an manchen Stellen das Ende der klassischen Disk-basierten Datenbank gefeiert. Daneben sahen wir, wie sich viele Business-Intelligence Werkzeugen zu verzweigten Plattformen mit zusätzlichen In-Memory-Datenhaltungen und Caches verselbständigten. Das Marketing der Hersteller ließ glauben, dass allein durch ein Stück Technik - „In-Memory“ – ein glücklicher Endanwender entsteht.
Wer die Diskussion genauer analysiert entdeckt viel Unwissenheit vor allem bei denen, die sich nie mit Data Warehouse – Theorie beschäftigten. In-Memory – Technologie muss gezielt dort eingesetzt werden, wo sie nützt. Das setzt Kenntnisse über die innere Struktur und Prozesse eines Data Warehouse – Systems voraus

## In-Memory mit Oracle

Seit Sommer 2014 steht die Oracle – In-Memory Option  mit dem Datenbank-Release 12.1.0.2 zur Verfügung. Bei den meisten bislang bestehenden In-Memory-Techniken kann nur der komplette Daten-Stack des Auswertesystems In-Memory liegen. Oracle geht dagegen auf die spezifischen Anforderungen eines unternehmensweiten Data Warehouse ein: Je nach Bedarf lassen sich Tabellen, einzelne Spalten, Materialized Views oder Indexe in den Hauptspeicher packen. Und das ohne die Data Warehouse-Anwendung zu verändern.
Ein

„ALTER TABLE f_umsatz INMEMORY“

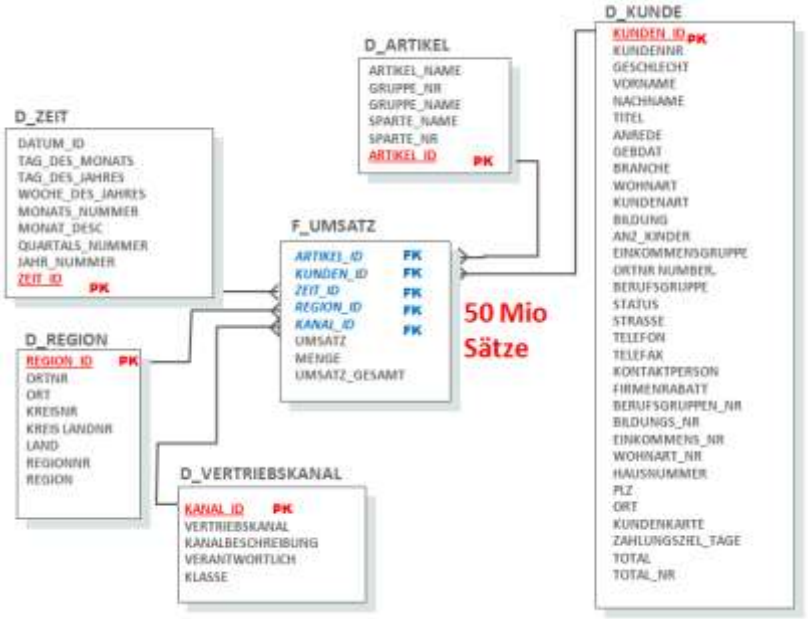
markiert die Tabelle F_UMSATZ, und das Datenbanksystem lädt Datenobjekte über einen Hintergrundprozess in die neue In-Memory Area im Hauptspeicher (SGA). Ein Initialisierungs-Paramter legt die Größe der In-Memory-Area fest („INMEMORY_SIZE“ und Achtung auch „SGA_TARGET“ als Gesamt-SGA-Größe).
Bei diesem Ladevorgang zerlegt das System die Tabellen zunächst in seine Spalten und komprimiert sie mit unterschiedlichen Optionen auf einen Faktor von 4 – 20.
Damit koexistieren beide bekannten Speicher-Konzepte in einer Datenbank: ROW-BASED für die klassische Disk-Speicherung bzw. den Buffer-Pool und COLUMN-BASED für die neue In-Memory Area.
Das Konzept ist wie geschaffen für ein hochvolumiges Data Warehouse, in dem eventuell nur 10 % der Daten ad hoc sekundenschnell zu lesen sind, während die Masse der Daten oft nur für den „Fall der Fälle“ oder eine Jahresvergleichsabfrage dahin schlummert.




*Tabellen liegen im ROW- und COLUMN-Format vor.*

### Welche Antwortzeiten entstehen?

Offiziellen Aussagen nach erreicht die Oracle In-Memory Technologie mehrere Milliarden Zeilen pro Sekunde pro CPU-Core. Diese Werte sind auf eine Spalte bezogen und mit schneller Hardware durchgeführt. Aber bereits auf einem einfachen Laptop und mit realistischeren, weil komplexeren Abfragen lassen sich erstaunliche Performancewerte erzielen. Das folgende, aus 5 Dimensionen bestehende Beispiel-Star-Schema (aus den Seminaren der Oracle Data Warehouse Community), ist ein praktisches Testszenario und erlaubt  Beispielabfragen mittlerer Komplexität.




Beispiel Star Schema als Testszenario.

Die Belegung im Speicher erhält man über den Dictionary View  v$im_segments. Erkennbar ist die Kompression der größten Tabelle (50 Mill. Sätze) mit Faktor 5,62. Aus 2,2GB wurden 0,41GB.

| NAME | NUM_ROWS | DISK_GB | BYTES | REST_BYTES | STATUS | IN_MEM_GB | COMP_RATIO |
|---|---|---|---|---|---|---|---|
| D_ARTIKEL | 129 | .00004 | 65536 | 0 | COMPLETED | .00118 | .06 |
| D_KUNDE | 1029 | .000224 | 262144 | 0 | COMPLETED | .00118 | .22 |

```
D_REGION                7202     .000584      655360          0 COMPLETED      .00118      .56
D_VERTRIEBSKANAL           7      .00004       65536          0 COMPLETED      .00118      .06
D_ZEIT                  5844     .000344      393216          0 COMPLETED      .00118      .33
F_UMSATZ              51200000   2.214512  2310012928          0 COMPLETED     .410584     5.626
```

*Auflisten der In-Memory-Objekte*

Ohne In-Memory läuft eine etwas aufwendigere Abfrage nach den Top 10 Artikeln pro Bundesland im Jahr 2010 läuft auch nach mehrmaligem Aufruf (also auf bereits im Buffer-Pool befindliche Daten - Cache) nicht schneller als 21 Sekunden.

```
SELECT * FROM
(SELECT  a.Artikel_Name       Artikel,
         r.Land               Bundesland,
         z.Jahr_nummer        Jahr,
         sum(U.umsatz)        Wert,
         sum(U.Menge)         Menge,
          round(sum(U.umsatz) / sum(U.Menge),2) Ums_pro_Art,
       RANK() OVER  (ORDER  BY sum(U.umsatz) DESC ) AS Rangfolge
from  F_umsatz U,  D_Artikel A ,  D_Zeit z, d_region r
WHERE
       U.artikel_id =  a.artikel_id   and
       U.REGION_ID =   R.REGION_ID    AND
       U.zeit_id       = z.zeit_id    AND
       z.jahr_nummer   = 2010
group by
      a.artikel_name,r.Land,z.Jahr_nummer)
WHERE rownum < 11;
```

*Beispielabfrage mit Sub-Select, 4 Join-Tabellen, 3 Gruppierungen und mehreren Filtern*

Schaltet man die In-Memory-Funktion für die Datenbank frei (ALTER SESSION set inmemory_query = enable;), so reduziert sich diese 50 Millionen-Sätze-Abfrage auf unter 1,5 Sekunden bei parallel 4 und das auf einem Windows-Laptop mit einfachem Dual Core – Prozessor.

### Was macht die Abfragen so schnell?

Die Abfrage der In-Memory-Daten ist um ein vielfaches schneller als eine Abfrage von Daten, die sich im Buffer-Pool der Datenbank befinden. Da es sich in beiden Fällen um eine Hauptspeicherablage handelt, muss es noch andere Performance-Faktoren geben:

- Ein Grund ist die **Spalten-bezogene Speicherung** der In-Memory-Option. Sie hat bei Data Warehouse-typischen Abfragen auf Wertebereiche und mit wenigen abgefragten Spalten Vorteile gegenüber der Satz-bezogenen Speicherung, die dafür bei OLTP-typischen Einzelsatzabfragen und vielen Spalten punkten kann.
- Hinzu kommt der höhere **Komprimierungsfaktor**, sodass geringere Datenmengen aus dem In-Memory-Speicher zu lesen sind. Die Filterprüfung (WHERE-Klausel und Join-Prüfungen) wendet das System direkt auf die komprimierten Daten an, ohne sie zuvor zu dekomprimieren. Oracle nutzt hierfür ein neu entwickeltes Komprimierungsverfahren, bei dem die Daten im komprimierten Zustand interpretiert werden können.
- Die Join-Bedingungen löst das System mittels **Bloom-Filtering**. Kleine Tabellen – wie die meisten Dimensionstabellen in einem Star Schema – wandelt das System in einen Bloom-Filter (eine Art Hash-Tabelle) um und sendet sie direkt zur größeren Faktentabelle. Schon bei dem Zugriff auf die Faktentabelle zieht der Join-Filter.
- Verfügt das System über mehrere CPU-Cores, so erhält jeder Core ein eigenes Bloom-Filter-Objekt und arbeitet direkt die ihm zugeordneten Faktentabellen-Sätze ab (**SIMD-Verfahren**).

- Aggregationen (GROUP BY) löst das System mit sog. „**Report Outlines**“. Ein einmal berechnetes Aggregationsergebnis merkt sich das System dynamisch für folgende Abfragen. Praktisch entsteht „On-The-Fly“ ein OLAP-Würfel im Speicher.

### Parallelisierung und In-Memory

In vielen gerade kleineren Data Warehouse-Systemen wirkt Parallelisierung aufgrund der geringen IO-Leistung des Storage-Systems nicht besonders gut. Kleinere Systeme haben oft keine nur ihnen zugeordnete Speicherplatten und sind an ein SAN angeschlossen. Sie konkurrieren mit der Masse OLTP-Anwendungen. die eine grundsätzlich andere Art der Storage-Nutzung praktizieren (Einzelsatzlesen vs. Bereichs-bezogenes Lesen). IO-Leistungen von wenigen 100MB / Sekunde sind die Folge. Parallel arbeitende CPUs einer Data Warehouse – Maschine erhalten nicht genügend Daten. Mit In-Memory wächst die IO-Leistung dramatisch in den Bereich von mehrere GB/Sekunde. Damit können die CPUs unter Volllast arbeiten. Ein effektives Parallelisieren ist möglich.

### Welche Tabellen sollen in den In-Memory-Speicher?

Wir haben die Wahl: Daten können In-Memory oder auf der Festplatte liegen. Welche sollen jetzt permanent in den Hauptspeicher? Generell solle ein Data Warehouse-Administrator auch ohne fremde Hilfe wissen, welche Daten dafür infrage kommen: Es sollten die meist genutzten Daten sein. Das System bietet hierfür einen Adviser an, der aufgrund der gesammelten Verwendungsinformationen des AWR (Automatic Workload Repository) und des ASH (Active Session History) Vorschläge für die lohnenswertesten Objekte liefert.

| Object Type | Object | Estimated In-Memory Size | Analytics Processing Seconds | Estimated Reduced Analytics Processing Seconds | Estimated Analytics Processing Performance Improvement Factor | Benefit / Cost Ratio (Improvement Factor / In-Memory Size) |
|---|---|---|---|---|---|---|
| Table | SOE.LOGON | 451.76MB | 2114 | 1,887 | 9.3X | 20.586 |
| Table | SOE.CARD_DETAILS | 607.32MB | 8346 | 7,248 | 7.6X | 12.514 |
| Table | SOE.ADDRESSES | 1.09GB | 5237 | 4,621 | 8.5X | 7.798 |
| Partition | SOE.PRODUCT_MOCKUP.Y2014Q1 | 812.6MB | 2003 | 1,489 | 3.9X | 4.799 |
| Table | SOE.CUSTOMERS | 1.10GB | 108 | 95 | 8.2X | 7.455 |
| Table | SOE.ORDER_ITEMS | 2.19GB | 7128 | 6,393 | 9.7X | 4.429 |
| Table | SOE.ORDERS | 1.34GB | 3512 | 2,917 | 5.9X | 4.403 |
| Table | SOE.PRODUCT_INFORMATION | 1.78MB | 2873 | 2,205 | 4.3X | 2.416 |
| Partition | SOE.PRODUCT_MOCKUP.Y2013Q4 | 1.62GB | 97 | 1,489 | 3.7X | 2.284 |
| Partition | SOE.PRODUCT_MOCKUP.Y2014Q2 | 3.37GB | 642 | 493 | 4.3X | 1.276 |

### Wie werden die Daten aktuell gehalten?

Oracle führt INSERTS und UPDATES nach dem bekannten Verfahren durch. Es können also auch lesende und schreibende Zugriffe parallel durchgeführt werden, ohne einen Lock-Zustand oder inkonsistente Daten zu erzeugen.

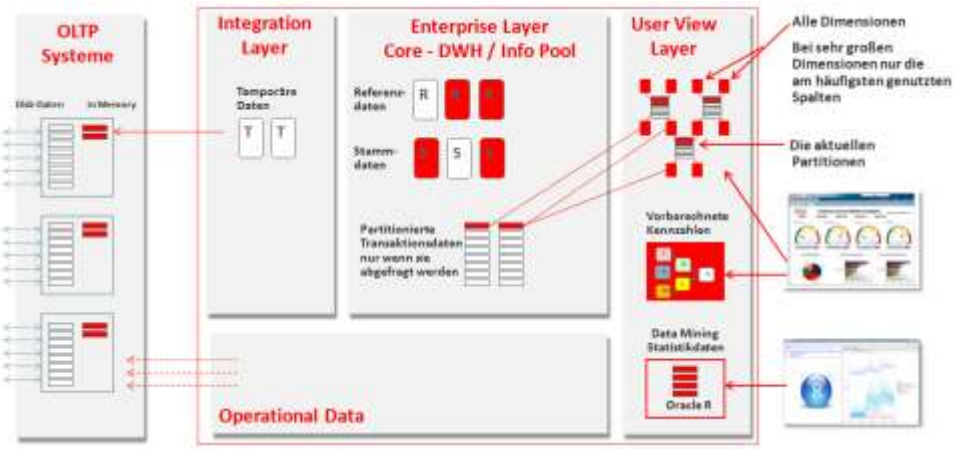
D. h. aber auch umgekehrt, dass das Lesen von Daten aus dem In-Memory-Speicher die zuletzt mit COMMIT bestätigten Daten widerspiegelt. Oracle hat zur Synchronisierung zusätzliche Prozesse implementiert. Dies gilt für einzelne INSERTS und UPDATES, wie sie etwa in einem OLTP-System vorkommen.

Diese Technik ersetzt jedoch nicht die bekannten Massenladeverfahren, die wir aus dem Data Warehouse kennen. D. h. also die Vorgehensweise, nach der wir z. B. durch einen Direct Path Load

zunächst temporäre Tabellen erzeugen, deren Inhalte prüfen und schließlich mit einem „Partition Exchange and Load“ in kürzester Zeit riesige Datenmengen in ein Data Warehouse schieben. Bei dieser gewählten Vorgehensweise „populiert“ man den In-Memory-Speicher dezidiert in einem nachgelagerten Schritt z. B. mit dem Package DBMS_INMEMORY.repopulate(schema,table,partition).
Diese Vorgehensweise kennen wir bereits von anderen Themen wie dem Aktualisieren von Statistiken, Indexen oder Materialized Views. Hier wird nachgelagert gehandelt, um ETL-Läufe möglichst kurz zu halten.

### In Memory im Data Warehouse

Gerade für Warehouse-Systeme mit sehr vielen historischen Daten ist diese Wahlfreiheit wichtig. In dem Oracle – Data Warehouse sind große historisierte Tabellen partitioniert. Während aktuelle und sehr oft genutzte Daten den Vorzug einer In-Memory-Speicherung genießen, lagern ältere und kaum gelesene Daten auf dem klassischen Plattenspeicher. Statistiken belegen, dass weniger als 10% der in einem Data Warehouse gespeicherten Daten mehr als 90% der Datenzugriffe erfahren. Diese Statistik berücksichtigt sowohl Satz- als auch Spaltenlevel, denn nicht nur Partitionen, sondern auch nur einzelne Spalten einer Tabelle lassen sich In-Memory vorhalten.
In einer klassischen 3-Schichten-Architektur nach Inmon würde man danach folgende Objekte In-Memory laden:



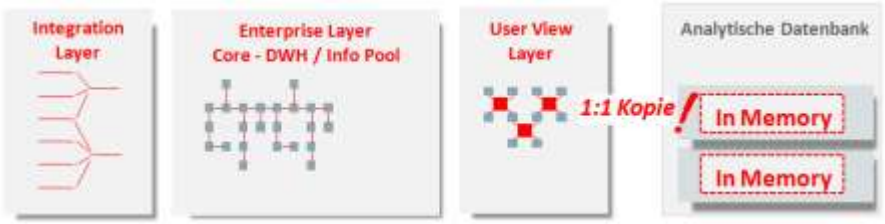

*Vorschlag für die „In-Memory-Objekte“ in einem Data Warehouse.*

- Die meisten Dimensionen in den Data Marts (User View Layer) sofern man sie überhaupt genutzt.
- Die jüngsten Partitionen der Faktentabellen und auch nur die genutzten Spalten.
- Data Mining-Tabellen bzw. Tabellen, die über R-Skripte in der Datenbank analysiert werden. R hat durch die Integration in die Oracle-Datenbank (ORE) an sich schon einen erheblichen Leistungsschub erfahren. Wandern die Analyse-Tabellen in den In-Memory-Column-Store, so potenzieren sich die Leistungsmerkmale.
- Kennzahlensysteme (basierend auf Materialized Views). Viele Abfragen in Warehouse-Systemen sind vorhersehbar. Zudem gibt es eine Grundmenge an Kennzahlen mit exakt definierten Herleitungsalgorithmen. Solche Objekte würden man weiterhin mit Materialized Views herleiten und auch diese in den In-Memory-Speicher legen.

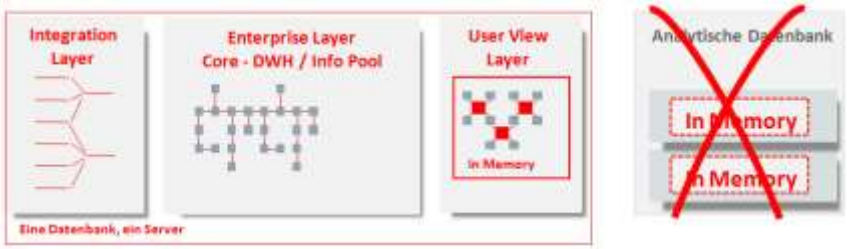
- In der Kern-Warehouse-Schicht (Enterprise Layer) sind die am häufigsten genutzten Stamm (S)- und Referenz(R)-Datentabellen im Speicher. Die Masse der Stamm- und Referenzdaten sind allerdings meist kleine Tabellen, die das System im „wie im Vorbeiflug“ auch schnell von der Festplatte lesen kann.
- Transaktionale Daten (Bewegungsdaten) müssen in dieser Schicht nur dann in den Speicher, wenn man sie direkt aus dem User-View-Layer heraus referenziert. Diese Schichten übergreifende Referenzierung mag einige verwundern. Aber warum sollte man große Bewegungsdatentabellen als Fakten in die Data Marts kopieren, wenn sich strukturell an den Daten nichts ändert?
- Operational Data Stores verlieren bei In-Memory zunehmend an Bedeutung. Liegen die operativen Anwendungen auf Oracle, dann kann man die benötigten OLTP-Tabellen in den Speicher laden und diese In-Memory-Version für Analysen freigeben. Die operative Anwendung wird durch das In-Memory -Lesen nicht behindert.

## Analytische Datenbanken sind passé!

Diese Liste zeigt, dass man doch mit etwas mehr Grips im Kopf an die In-Memory-Thematik im Warehouse herangehen sollte. Mit dem Begriff analytische Datenbanken wird man aufräumen müssen, wenn diese Datenbanken im Kern einfach nur mehr Performance versprechen. Der Preis ist hoch: Jede separate Datenhaltung neben dem zentralen Data Warehouse zersplittert die Informationslandschaft und provoziert unnötige Ressourcen-Aufwände, Kosten und die Gefahr von Doppelarbeit mit nicht abgestimmten Ergebnissen.
Es bewahrheitet sich immer wieder:
Wer die Daten an einer Stelle zusammenhält, schafft Übersichtlichkeit und gleichzeitig Flexibilität. Mit In-Memory wird er zusätzlich belohnt.




*Separate analytische Datenbanken verlängern den Informationsbeschaffungsprozess und schaffen unkontrollierbare Inseln.*




*Kompakte Analyse-Umgebungen profitieren am meisten von In-Memory direkt in der Datenbank.*

## Auch BI-Werkzeuge und BI-Plattformen müssen umdenken

In den letzten Jahren haben BI-Tool-Hersteller ihre Analyse-Umgebungen zu eigenständigen Plattformen weiterentwickelt. Mit dem Ziel besonders performante Auswertungen zu ermöglichen, wurden In-Memory-Caches und sogar spezielle Analysedatenhaltungen (Files oder OLAP-Cubes) in die Werkzeuge integriert. Ähnlich wie bei den Analytischen Datenbanken kam es auch hier zu zusätzlichen Datentransporten aus der zentralen Warehouse-Datenbank heraus und in die „dezentralen Caches" hinein. Unschwer zu erkennen: In Zeiten von In-Memory in der zentralen Datenbank ist das alles andere als ein schlankes smartes Analysesystem. Von In-Memory in der Datenbank profitieren heute diejenigen BI-Werzeuge, die mit SQL direkt aus der Datenbank lesen. Der Betrachtungs-Horizont der BI-Administratoren sollte sich auch auf die Möglichkeiten in der Datenbank erstrecken, anstatt alle Kalkulationen und Aufbereitungen innerhalb der BI-Plattform machen zu wollen. Auch ein BI-Administrator sollte sich die Frage stellen: Welche Benutzerabfragen und welche Kennzahlen sind bekannt und lassen sich bereits innerhalb der zentralen Datenbank vorberechnen. Mit In-Memory in der Datenbank wird dieses Denken plötzlich attraktiv.

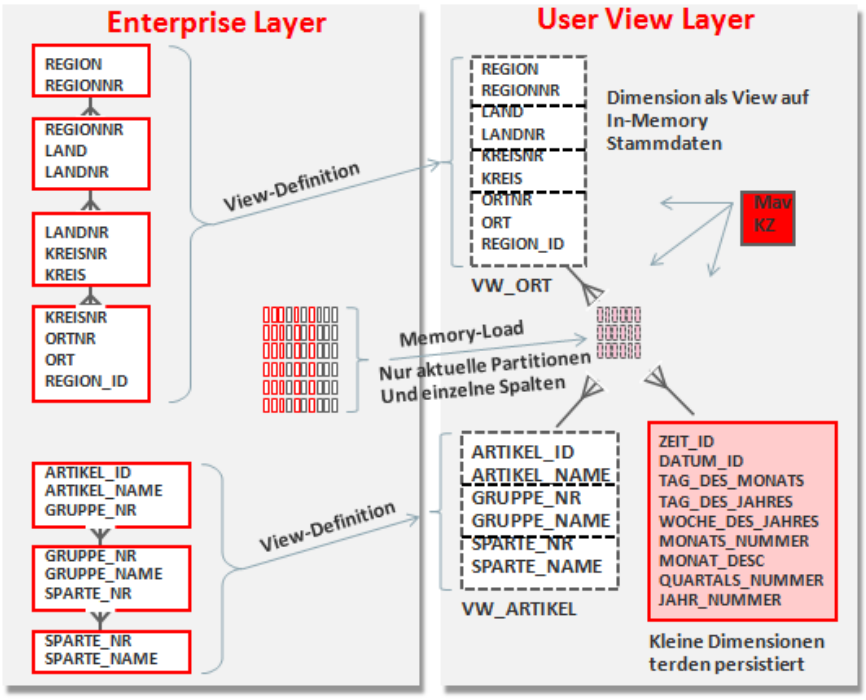
### Virtualisierte Data Marts mit In-Memory und schlanke Architekturen

Soweit so gut. Wir können In-Memory aber auch nutzen, um unsere Data Warehouse Architekturen neu zu überdenken. Warehouse-Architekten kennen die von Inmon vorgeschlagene 3-Schichten-Architektur, bestehend aus Stage / Kern-Warehouse (Enterprise Layer) / Data Marts (User View Layer). Dieses in den meisten großen Warehouse – Systemen genutzte Verfahren ist in jüngster zeit vor allem wegen der langen Durchlaufzeiten und fehlender Flexibilität kritisiert worden. Sinnvolle Pluspunkte an diesem Verfahren sind aber:

1) Data Mart-übergreifende zentrale Synchronisation aller (!) Daten in einer standardisierten zentralen Schicht (Kern-Warehouse / Enterprise Layer)
2) Analyse-spezifische Aufbereitung von Daten in den Data Marts (oft multidimensional)

Wenn wir das Architekturkonzept verändern, um es effektiver zu machen, dann sollten diese beiden Pluspunkte nicht verloren gehen.

Mit In-Memory können wir die 3. Schicht die Data Marts (User View Layer) in Teilen auflösen (virtualisieren). Z. B. können wir Star-Schemen, die bisher aus physischen Fakten- und Dimensionstabellen bestanden, durch In-Memory-Daten ersetzen. Für multidimensional angelegte Abfragen ändert sich nichts. Aber es gibt keine physikalischen Tabellen mehr für ein Star Schema und damit auch keine ETL-Prozesse und keinen Zeitverzug zur Beschaffung der Daten. Die Daten des Data Marts lassen sich ad hoc verändern, nachdem sich die Basis in dem Kern-Data Warehouse (Enterprise Layer) geändert hat. Auch die Strukturen eines Star Schemas lassen sich schnell ändern, indem man geänderte Objekte in den Column-Store lädt.




Eine weitere Überlegung ist der Ersatz von Materialized Views, die nur aus Performance-Gründen in das System eingebaut wurden durch reine View-Definitionen. Das „SELECT“ der Views wird erst zum Abfragezeitpunkt ausgeführt und ist schnell, wenn die Basis-Tabellen der View in-Memory liegen.
(Wir empfehlen natürlich das Materialized View Konzept auch zum Aufbau von Kennzahlen-Systemen. Diese Materialized Views sollte man dann nicht ersetzen).

Alfred Schlaucher September 2014-07-11

Alfred Schlaucher
Gestartet Mitte der 80er mit Datenbankprogrammierung auf Großrechnern von Siemens und IBM. Beschäftigte sich in den 90er mehrere Jahre mit Datenmodellierung und Metadatenmanagement und seit Mitte der 90er mit Data Warehouse. Seit Ende der 90er ist er für Oracle im Bereich Data Warehouse aktiv.