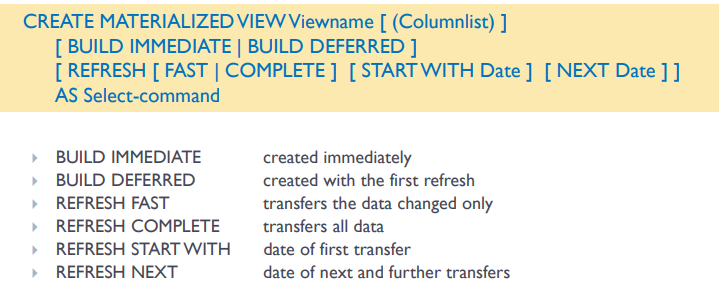
**Einführung**

* Data Warehouse:
  + Die Daten werden regelmäßig von der Produktivdatenbank abgefragt
  + Daten werden zum Lesen vorbereitet
  + Daten auf Dauer speichern 🡪 Archiv
  + erlaubt Fragen mit einer breiteren Antwortmöglichkeit (z.B. Kunden die x kauften, kauften auch y)
  + Vorteile:
    - unabhängig von der Datenbank
    - keine zusätzliche Last für die Datenbank
    - Direkter OLAP Zugriff von mehreren Usern stresst nur DW und nicht Datenbank
  + Nachteile:
    - Eigenes System zum Verwalten
    - Data Warehouse muss geupdated werden, meistens über Nacht
* **Gründe für DW:**
  + Keine zusätzliche Last auf Produktivsystem
  + Daten werden für Jahre gespeichert zum vergleichen, diese sind nicht im Produktivsystem vorhanden
  + Ein Bild, das Tisch enthält.

    Automatisch generierte BeschreibungProduktivsystem ist für Transaktionen optimiert und erlaubt keine Redundanz
* **OLTP: Online Transaction Processing**
  + Produktivsystem mit Transaktionen
  + Transaktionssicherheit, Hoher Durchsatz, Kurze Antwortzeit, Paralleles Verarbeiten
* **OLAP: Online Analytical Processing ( 🡪 Data Warehouse)**
  + nur zum Auslesen, Analyse der Daten im Vordergrund
* **FASMI: Fast Analysis of Shader Multidimensional Information**
  + Fast: die meisten Antworten kommen innerhalb von 5 Sekunden
  + Analysis: Analyse ist einfach genug für den Zieluser
  + Shared: Gleichzeitige Schreib- & Lesezugriffe können verwaltet werden
  + Multidimensional: Multidimensionale, konzeptionelle Sicht auf die Daten
  + Information: System verwaltet alle Eingabedaten und speichert diese

**Datenbank zu Data Warehouse**

* Characteristics: Merkmale
  + Basisdaten, die für Abfragen verwendet werden (z. B. Produkt, Kunde, Datum)
* Key figures: Kennzahlen
  + Resultate, die für Basisdaten erwartet werden, können sich in DB ändern, aber werden in DW festgehalten
    - z.B. Verkaufspreis, Anzahl, Wachstum ...
* Fact Table: optimiert Datenzugriff über verschiedene Tabellen
  + beinhaltet Characteristics (OrderID, CustID, ArtID, Salesdate), Key figures (Quantity, Price) und Zusatzinformationen (Monat, Jahr)
  + meistens Join von verschiedenen Tabellen
  + Vorteile:
    - Ein Join weniger
    - Optimisierung ist unabhängig von Datenbank
    - Anfragen brauchen keinen Zugriff auf einzelne Tabellen
  + Nachteile:
    - Zusätzliche Tabelle (Verwaltung, Laden der aktuellen Daten)
  + Aggregationen:
    - Daten werden in einzelnen Tabellen angezeigt, z.B. pro Monat



* Materialized View: aktualisiert Daten nebenbei, um gleich den Originaldaten zu sein
  + wird mit Daten nach dem „START WITH“ date befüllt
  + wird upgedated mit dem „NEXT“ date

Daten in Datenbank

* Master Data: Alle Informationen, die eine wichtige Rolle in der Ausführung eines Geschäfts spielen
  + z.B. customers, supplier, region , country, articles, stock location ...
* Transactional Data: Datensatz, der unvorhersehbar und zufällig während der Ausführung ist
  + z.B. order, account, production, sale, purchase

Beispiel INSERT INTO Facttab\_Month mit Group by:

***INSERT INTO*** fact1\_month (mid, artid, month, year, price, quantity)

***SELECT***

        mid\_counter.NEXTVAL,

        FM.\*

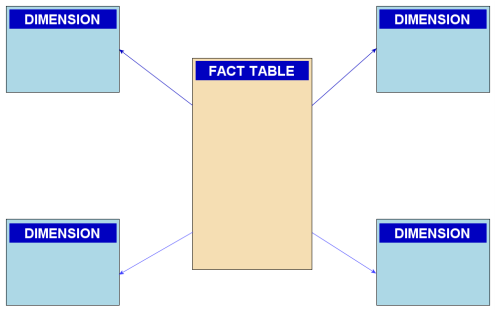
***FROM***

        ( ***SELECT*** artid, month, year, round(AVG(price), 2), SUM(quantity)

***FROM*** fact1 *//mit JOIN ... USING (SSID)*

***GROUP*** BY artid,  month, year ) as FM;

**Konstruktion eines Data Warehouse**

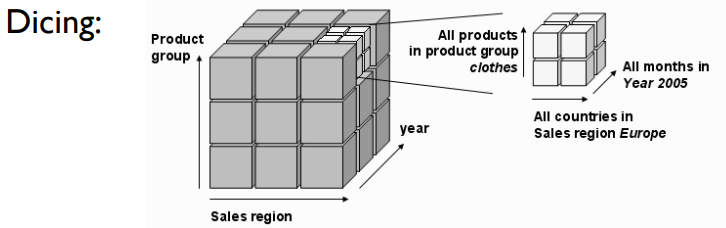
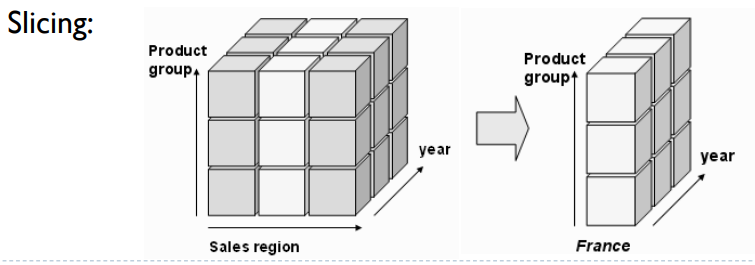
* **Star Model:** 
  + Fakttabelle, umgeben von Dimensionen (Master Data)
  + Faktttabelle beinhaltet IDs und Key Figures
* Snowflake Model
  + Fakttabelle, umgeben von Dimensionen, mit zusätzlichen Dimensionen
  + Weniger Redundanz, weil Dimensionen mit IDs auf Daten zugreifen (z.B. ID Day of Week...)
* Multi Dimensional Model
  + Fakttabelle mit Dimensionen ist ein „Cube“, mehrere Cubes miteinander sin eine Galaxy

**zum Star Model:**

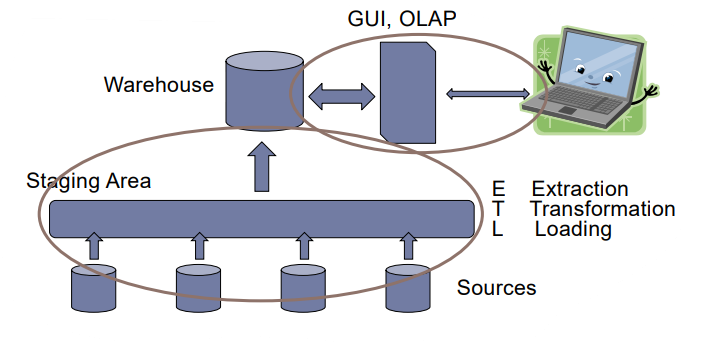
* ROLAP: Relational OLAP
  + Daten werden relational gespeichert, Fakttabelle und Dimensionen sind Relationen, verbunden durch FK
  + Vorteile:
    - Existierende, high-performance Relationale Datenbanken können benutzt werden
    - Wir können SQL zum abfragen nutzen
  + Nachteile:
    - viele joins 🡪 Performance Probleme:
      * Star Schema mit n Dimensionen brauch n Joins
    - wir brauchen zusätzliche, High-Performance SQL-Operatoren
* MOLAP: Multidimensional OLAP
  + fixiertes und statisches, multidimensionales Array (keine Datenbank), Einträge zu allen Kombinationen von Characteristics
  + Vorteile:
    - wenige Joins, sehr performant
  + Nachteile:
    - proprietäres System, ungewöhnliche Vorgehensweise für Abfragen, viel Speicherverbrauch
  + nur für kleine Würfel geeignet
* HOLAP: Hybrid OLAP
  + Mix zwischen ROLAP und MOLAP, benutzt Vorteile beider Modelle
    - Große Würfel in ROLAP, kleine Würfel und aggregierte Würfel mit in MOLAP
* Oracle: ROLAP, zusätzlich zu memory datenbanken
* MS SQL Server: MOLAP ist zusätzlich implementiert
* SAP BW on HANA: ROLAP mit vielen Spüalten und vielen Speichertechniken

**Abfragen:**

* Definitionen:
  + Drill Down: von gesammelten Informationen zu Detail durch Fokus auf ein Detail
  + Roll Up: Gegenteil von Drill Down, Details werden zu gesammelten Informationen aggregiert
* Slicing und Dicing:
  + Slicing: Scheibe aus einem Würfel schneiden, also nur eine Bedingung
  + Dicing: kleinen Würfel aus dem großen Würfel mit gleichen Dimensionen schneiden



**ETL-Prozess**

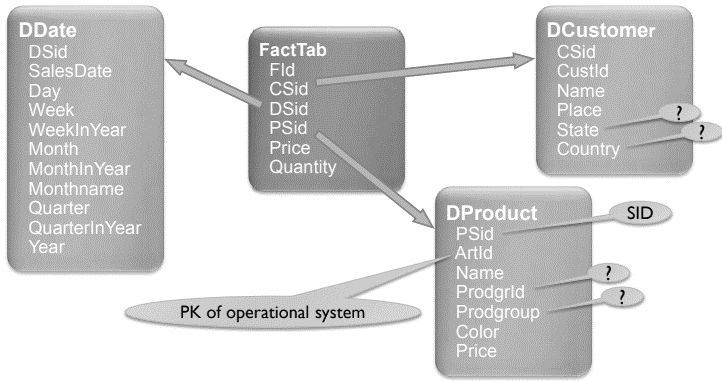
* **ETL-Prozess** Extraction, Transforming, Loading
  + Daten werden in Staging Area geladen, transformiert & persistiert, erst zu günstiger Zeit werden Daten in Data Warehouse geladen
  + Sources:
    - Relationale Datenbanken, Daten sollen am besten ohne Redundanz geladen werden
  + Staging Area:

Data Flow

* + - Interface für die Sources (meist relationale DB)
    - persistent, weil Laden des Warehouses fehlschlagen kann
    - Sicheres Interface zwischen OLTB DB und Warehouse
  + Warehouse
    - NoSQL Datenbank oder propertiäres System
    - Performance ist extrem wichtig
    - Redundanz ist nicht so wichtig
* Extraction
  + Staging Area: die externen Daten werden für das Data Warehouse vorbereitet
    - Quellen: aus Datenbanken, Excel-Files, Text-Files, ...
* Transformation
  + Transformation der Daten in der Staging Area
  + Datentypen adaptieren, konvertieren von speziell gecodeten Daten (ASCII, Unicode), unifying of Strings and Dates, Messeinheiten konvertieren
  + Probleme: Extrahierte Daten können out of Date, inkorrekt, inkosistent, unvollständig sein
* Loading
  + Laden der Daten aus der Staging Area in Data Warehouse (erhöht response time)
  + Laden während einer Zeit mit wenig Anfragen, z.B. Nachts
  + Abfragen werden limitiert während dem Laden

GUI:

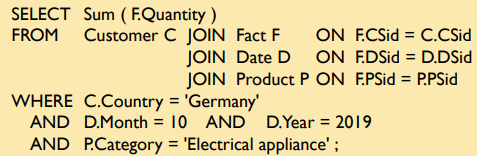
* soll Abfragen auf alle Daten ermöglichen
* Einfach und intuitiv
* Technische Details werden versteckt
* Automatische Erstellung von Instruktionen mit hoher Performanz
* Zugriff für Idioten die kein SQL können soll möglichst einfach sein

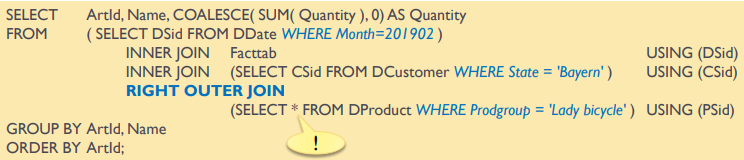


**Arbeiten mit dem Data Warehouse**

Star Join:

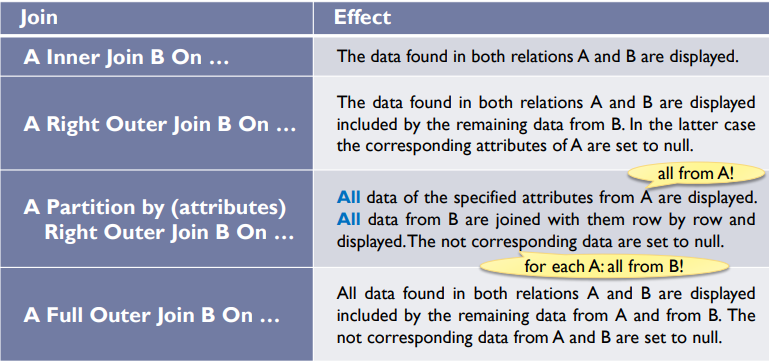
ON, USING, NATURAL

* 3. Normalform: Redundanz wird verhindert durch Snowflake Modell
* Join mit Join mit Join ... 🡪 durch IDs wird von einer Tabelle zur nächsten gegangen
* Surrogat Key:
  + Enumeration
  + z.B. Csid 🡪 Customer Surrogat ID: Primärschlüssel in den Dimensionen
* Join zwischen Dimensionen und Fakttabellen

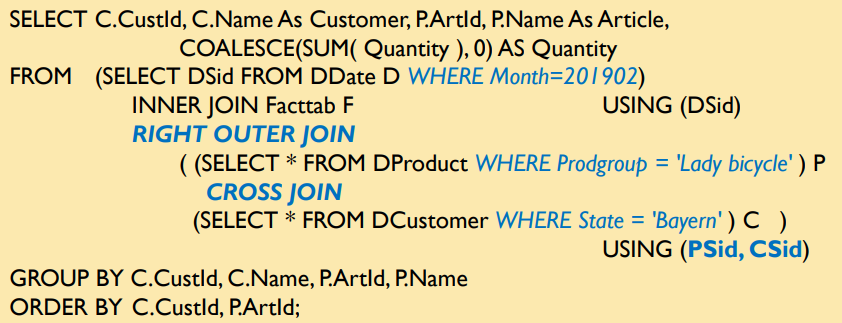
Outer Join:

* Vorziehen von SELECT-Befehlen in den FROM-Teil: Ergebnis wird erst gefiltert und dann gejoined, dadurch kann man auch Ergebnisse anzeigen, die bestimmte Filter sonst nicht erfüllen

Cross Join:

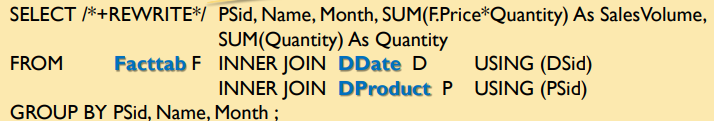
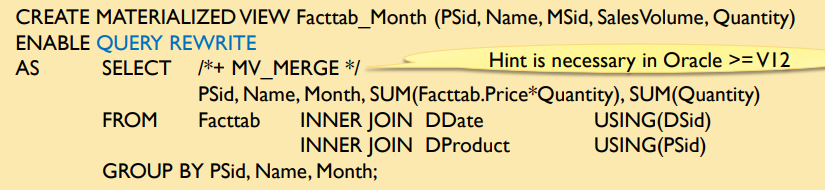
* A **PARTITION BY** (Attribut1, Attribut2) **RIGHT OUTER JOIN** B ON ...
  + Attribute in Partition By: diese sollen gecrossed werden mit B
    - alle spezifizierten Attribute werden einmal mit komplett B gekreuzt
* A RIGHT OUTER JOIN ( SELECT B CROSS JOIN C ) USING ...
  + CROSS JOIN: alle Zeilen von B mit allen Zeilen von C kreuzen
  + Reihenfolge:

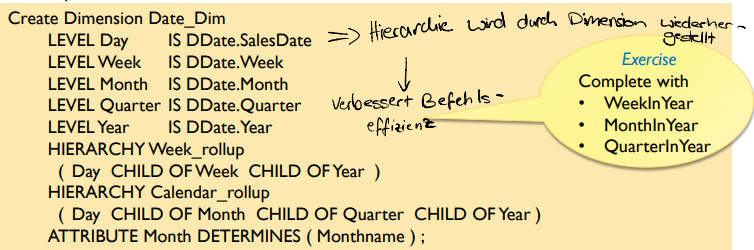
1. INNER JOINS
2. RIGHT OUTER JOIN
3. CROSS JOIN
4. 2 USINGs nach CROSS JOIN 🡪 sind ja 2 Tabellen die gekreuzt werden



Query Rewrite (Materialized View):

* Materialized View kann mit der Option „ENABLE QUERY REWRITE“ erstellt werden
* Dadurch nutzt Oracle automatisch die MV, wenn es sich anbietet, um die Performance zu verbessern
  + auch wenn eigentlich auf die Facttab zugegriffen wird, wenn die Option Rewrite eingeschaltet ist, optimiert Oracle automatisch die Abfrage



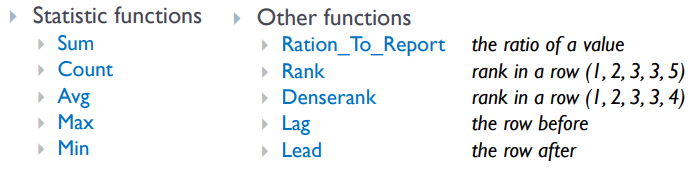
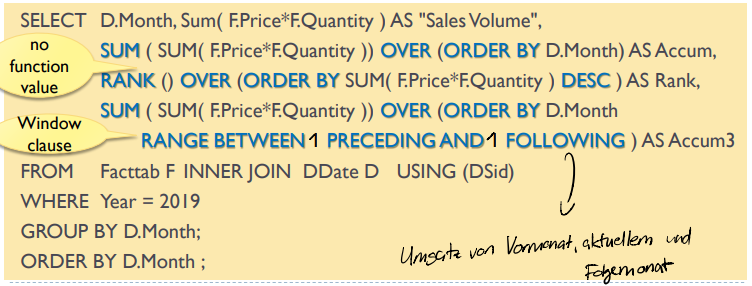
Dimensions:

* CREATE DIMENSION stellt Hierarchie wieder her (vgl. DDate: Jahr, Quartal, Monat, Tag)
* sonst nimmt QUERY REWRITE für z.B. Jahr nicht die erstellte MV her, sondern berechnet es einzeln 🡪 Performanceeinbußen
  + LEVEL: Hierarchie Level werden bezeichnet und mit den Spalten verknüpft
  + HIERARCHY: Beschreibung der Hierarchie: *m to n* Relation (Foreign Keys von Tabellen der 3. NF)
  + ATTRIBUTE: Beschreibung von zusammenhängenden Spalten innerhalb der Tabelle
* Problem: Im Star Model wird die 3. Normalform aufgegebene
  + Hierarchien existieren nicht mehr, Oracle braucht Anweisungen
* Validierung: EXECUTE DBMS\_DIMENSION.VALIDATE\_DIMENSION(‚DATE\_DIM‘, FALSE, TRUE)
  + Nur nach Validierung wird Dimension mit Query Rewrite benutzt

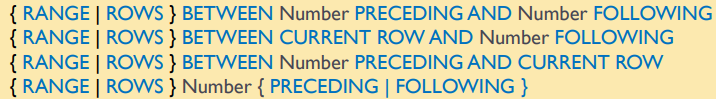
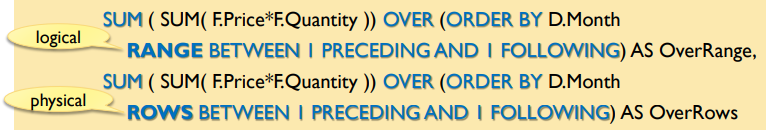
Optimizer:

* nimmt aggregierte Tabellen mit Query Rewrite, durch Dimensions wird auch z.B. für Jahresabfrage die aggregierte Monats-View benutzt
* Average of Average ist ein statistisches Problem 🡪 für Funktion Average wird nicht die MV benutzt

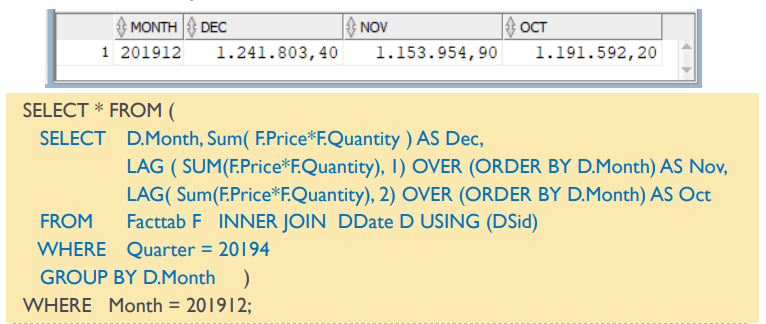
Analytische Funktionen:

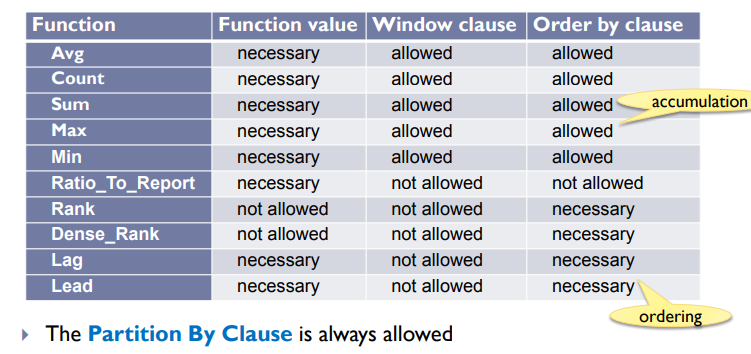
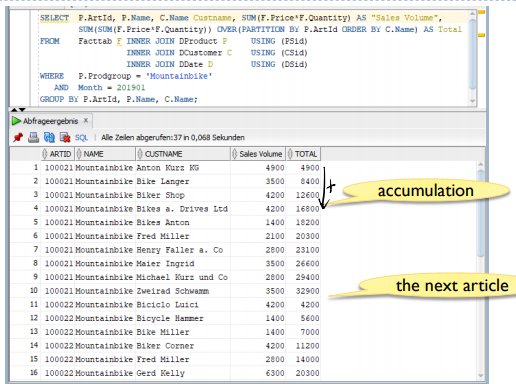
* ersetzen komplexe Anfragen durch einfache Funktionen, erhöhen die Performance
* Tabellen werden nur einmal gescannt und die Daten dann gebuffered
* Summieren:
  + SUM ( ... ) OVER ( ... )
    - SUM über den kompletten gewählten Bereich
* Prozent:
  + RATIO\_TO\_REPORT ( ... ) OVER ( ... )
    - prozentualer Anteil über alle gewählten Spalten
* Syntax: 
  + Partition By: z.B. PARTITION BY ArtID 🡪 immer summieren über Artikel
  + Order By: z.B. ORDER BY Name 🡪 Sortierung nach Name, Akkumulation in Partition
  + Window Clause: z.B. RANGE BETWEEN 1 PRECEEDING AND 1 FOLLOWING
    - Summieren über eins vorher und eins nachher, **nur erlaubt mit order by Klausel**!
    - z.B. mit ORDER BY Month 🡪 immer Addition mit einem Monat vorher & nachher
* Rang:
  + RANK () OVER (ORDER BY SUM (price \* quantity) DESC)
    - Rangliste mit Order und Desc 🡪 Ausdruck hinten wird von 1 - ... geranked

Window Clauses für analytische Funktionen:

* Syntax:
* Range: zählt logisch in der ORDER BY Spalte
  + z.B. bei Monat: wenn April nicht inbegriffen, werden nur März und Februar zusammengerechnet
* Row: zählt physisch in der ORDER BY Spalte
  + z.B. bei Monat: wenn April nicht inbegriffen, werden März, Februar und Mai zusammengerechnet

Lag und Lead:

* Syntax: {LAG|LEAD}(<argument>, <count>, <alternative>) OVER (**Order by** ...)
  + argument: was soll mit der Zeile gemacht werden
  + count: die wievielte Zeile vorher / nachher soll genommen werden
  + alternative: Alternative für NULL, also meistens 0
  + ORDER BY: wird benötigt, ohne geht’s nicht
* Lag: vorherige Zeilen werden abgefragt
* Lead: nachfolgende Zeilen werden abgefragt



Beispiel für SUM () OVER ()

Ein Bild, das Text enthält.

Automatisch generierte BeschreibungPartitions:

* große Tabellen sollen in mehrere Einzelteile aufgeteilt werden
* Wichtige Eigenschaften:
  + Transparenz: unsichtbar für den User
  + Performanz: Antwortzeit reduzieren
  + Concurrency: gleichzeitige Zugriffe erlauben
  + Allocation: Tabellen auf mehrere Disks verteilen
  + Ladezeiten: Schnellerer ETL-Prozess
* Vorgehensweisen:
  + BY LIST (VALUES ...)
  + BY RANGE (VALUES LESS THAN ... oder MORE THAN ...)
* Vorteile:
  + Nicht sichtbar für den User
  + Der Optimizer kennt die Partitionen (gute Performance)
  + Meistens nur Zugriff auf eine oder wenige Partitionen
  + Parallele Zugriffe möglich
  + ETL-Prozess braucht nur die aktive Partition (z.B. von diesem Monat)
* Beispiel: in SAP BW: jeder neue ETL-Prozess erzeugt eine neue Partition

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

Bitmap Index:

* B-Baum: optimiert für Laufwerkzugriff, average filling degree der Blöcke von 70 – 80%
  + Nachteil: B-Baum ist nicht gut für Abfragen mit Aggregationen
  + Erfahrung: Index sinnvoll, wenn weniger als 7% der Daten gefragt sind
* Selektivität: Anteil der selektierten Daten an allen Daten z.B. Gender Mann = 33% (bei gleicher Verteilung)
  + Problem: hohe Selektivität verringert die Vorteile von B-Bäumen
  + Lösung #1: B\*-Baum
    - B-Baum mit zusätzlichen Merkmalen
      * In der letzten Hierarchieebene auf den Einträgen Link zu den Daten & zum nächsten Eintrag
      * die anderen Hierarchieebenen haben nur Links auf das nächste Level
    - B\*-Bäume sind „state of the art“ 🡪 implementiert in allen Datenbanken
    - Hohe Selektivität führt zu Zerfall des B\*-Baums
  + Lösung #2: Bitmap-Index
* Bitmap-Index:
  + Vorteile: Lesezugriffe sehr schnell, Gruppenzugriffe sehr effizient
  + Nachteil: Schreibzugriff 🡪 schon eine kleine Schreibrate von 1-2% ist ein großes Problem
    - deshalb: nur Verwendung in OLAP interessant, für OLTP nicht benutzbar
  + z.B. bei Prouktgruppen: nur 1 Bit pro Produktgruppe als Index, 6 Bits dann pro Indexeintrag (weil 6 Prodgru.)
  + Performance:
    - Bit-Poeratoren (AND, OR, NOT) sind sehr schnell
    - Hohe Effizienz und niedriger Speicherverbrauch
  + Erfahrung: Im Data Warehouse sind Bitmap Indizes eine effizientere Alternative zu B\*-Bäumen
  + in Oracle:
  + Optimizer benutzt automatisch Bitmap Indizes zum suchen von Ergebnissen, deutlich erhöhte Performance

**Operating a Data Warehouse**

ODS (Operational Data Store):

* Bereich im DW, wo zusätzliche Daten gespeichert werden (ähnliche / gleiche Struktur wie in der Quelle)
* 2 Hauptaufgaben:
  + offene Daten (z.B. offene Bestellungen)
  + detaillierte Daten (verschiedene Dokumente)
    - z.B. im Würfel monatliche Daten, in ODS tägliche Daten
* integriert im DW, parallel zu Würfeln
* in SAP BW: DSO parallel zu cubes, aDSO ist standard Cube in SAP BW on Hana

Name Conventions:

* für main identifier, classifiyng und qualifying expressions
* Abbrevation Tables für Bezeichner in verschiedenen Längen (Abkürzungen)

Vorbereitung von Daten:

* Aggregation entweder über Export und Vorbereitung in Excel (Pivot Tabellen), oder mit einer Datenbank (SQL)
  + in Oracle:
    - Holzhammermethode: mit mehreren SQL-Statements und UNION ALL
      * schlechte Performance, riesige SQL-Statements, fast nicht lösbar
    - Besser: Grouping Sets, Rollup, Cube
* Ein Bild, das Text enthält.

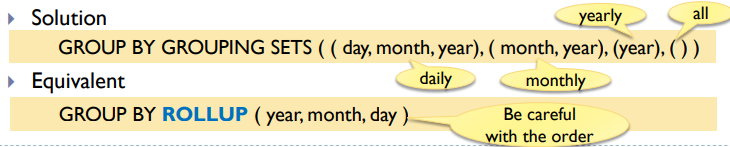
  Automatisch generierte Beschreibungin SAP BW sind beide Möglichkeiten gegeben

**=**

Grouping Sets:

* Gruppierung der Daten nach bestimmten „Sets“: Code rechts macht in beiden Fällen das gleiche
* Vorteile:
  + Gute Performanz (nur ein Scan von Fact-Table)
  + Oracle speichert die aggregierten Daten für zukünfitige Aggregationen
* Da in letzter Zeile (null)-Values ausgegeben werden 🡪 Nutzung von COALESCE
* Ein Bild, das Text enthält.

  Automatisch generierte BeschreibungBsp.: Ausgabe von einzelnen Artikeln, dann von einzelnen Kunden und dann vom Umsatz zusammengerechnet

Rollup:

* Partition wie bei Grouping Sets, aber in hierarchischer Reihenfolge
* Bsp.: erst Report von Tagen, dann vom Monat, dann vom Jahr:
* zuerst Gruppierung nach Jahr, dann Gruppierung nach Monat, dann Gruppierung nach Tag (von links nach rechts) 🡪 alle einzeln aufgeführten Elemente werden wieder einzeln gruppiert, zusammenfassen mit Klammern

**Nach jedem Monat wird die Monatszusammenfassung ausgegeben 🡪 ORDER BY wichtig!**

Cube:

* Cube-Operator zeigt alle verschiedenen Möglichkeiten von Grouping Sets:

**=**



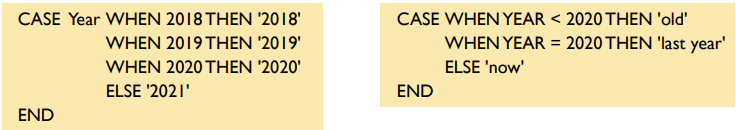
Grouping:

* Abfrage: Wenn die Spalte gruppiert wurde, dann schreib Ausdruck rein
  + CASE WHEN GROUPING (column) = 1 THEN ‚...‘
  + Ein Bild, das Tisch enthält.

    Automatisch generierte BeschreibungBenennung der aktuellen Aggregation so besser möglich, z.B. Monthly Sum, Yearly Sum, ...

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

* Allgemein:
  + Ein Bild, das Text enthält.

    Automatisch generierte Beschreibung2 verschiedene Möglichkeiten:

**=**

Beispiel für Cross Join:

***SELECT COALESCE*** (TO\_CHAR (salesdate), 'SUM') as *Day*,

***COALESCE*** (dproduct.name, 'SUM') as *Name*,

***COALESCE*** (color, 'SUM') as *Color*,

***COALESCE*** (***SUM*** (f.price\*quantity), 0) as *Revenue*

***FROM*** facttab *f*

***INNER*** ***JOIN*** (***SELECT*** \* ***FROM*** dcustomer ***WHERE*** state = 'Hessen') ***USING*** (Csid)

***RIGHT*** ***OUTER*** ***JOIN*** (***SELECT*** \* ***FROM*** ddate ***WHERE*** monat = 202011) ***USING*** (Dsid)

***PARTITION*** ***BY*** (salesdate)

***RIGHT*** ***OUTER*** ***JOIN*** (***SELECT*** \* ***FROM*** dproduct ***WHERE*** prodgroup = 'mtbnbike') ***USING*** (Psid)

***GROUP*** ***BY*** ***Grouping*** ***Sets*** ((salesdate, name, color), ())

Beispiel für LAG / LEAD Abfrage:

***SELECT*** month, rev20, rev19, rev18

***FROM*** (

***SELECT*** year, ***COALESCE***(monthinyear, 0) *month*,

***COALESCE***(sum(f.price\*quantity), 0) *rev20*,

***LAG***(***COALESCE***(***SUM***(f.price\*quantity),0), 12, 0) ***OVER*** (***ORDER*** ***BY*** year, monthinyear) *rev19*,

***LAG***(***COALESCE***(***SUM***(f.price\*quantity),0), 24, 0) ***OVER*** (***ORDER*** ***BY*** year, monthinyear) *rev18*

***FROM*** facttab *f*

***INNER*** ***JOIN*** (***SEELCT*** \* ***FROM*** dproduct ***WHERE*** name = 'mtnbike') ***USING*** (Psid)

***RIGHT*** ***OUTER*** ***JOIN*** (***SELECT*** \* ***FROM*** ddate ***WHERE*** year ***IN*** (2018, 2019, 2020)) ***USING*** (Dsid)

***GROUP*** ***BY*** (year, monthinyear)

)

***WHERE*** year = 2020;

Materialized View:

* REFRESH FAST ON COMMIT 🡪 nach jedem Commit wir dein schneller refresh durchgeführt

Metadata:

* Informationen um Konstruktion, Wartung und Administration zu vereinfachen
  + descriptive Information über Inhalt, Struktur, Kontext und Sinn der Daten
  + process relational Information über das Data Processing
* gibt Information darüber, wo die Daten im DW herkommen, wie die Daten transformiert wurden, wann und wo die Daten in die Würfel geladen wurden, über die Struktur und den Inhalt der Würfel

History:

* DW wird ständig geändert, aber es gibt auch seltene Änderungen (Umzug von Kunden, neue Produktgruppe,...)
  + Facttable (transaktionelle Daten): die betroffenen Daten werden gelöscht und neu angelegt
  + Dimesion table (master Daten): Duplikat mit Änderung wird angelegt, neue ID, neue darauf refernzierte Daten bekommen neue ID mit
* SCD: slowly changing dimensions

Date and time:

* Datum und Zeit sind separiert, Datum hat eigene Dimesion Table
* Zeit wird extra gespeichert wenn sie benötigt wird (neue Dimension)
* Ein Bild, das Text enthält.

  Automatisch generierte BeschreibungProbleme:
  + Wochen 1 und 53 sind nicht vollständig
  + Zeitzonen in verschiedenen Ländern spielen eine Rolle
  + Sommer- und Winterzeit

Währung:

* Geldsumme wird in lokale Währung umberechnet und dann im Data Warehouse gespeichert

Hierarchien:

* in manchen Tabellen hat die Hierarchie ein festes Limit (vgl. DDate)
  + Ein Bild, das Tisch enthält.

    Automatisch generierte Beschreibungin manch anderen nicht (z.B. Articlestructure)
* Bridge Tables:
  + gibt Hierarchie (Father – Son) an und Hierarchieebenen (Levels between & lowest Level)

Junk Dimensions:

* alle Attribute, die nicht in andere, existierende Dimensionstabellen passen, werden in einer Junk Dimension zusammengefasst
* z.B. Indikator zum analysieren und verfizieren von verkauften Produkten

Fact tables without facts:

* in manchen Würfeln exisiteren keine Fakttabellen
* z.B. bei einer Auktion wird nichts verkauft, trotzdem sind Angebote interessant

Protocoll Dimension:

* enthält alle Ladeprozesse für die Würfel
* enthält alle durchgeführten Transaktionen

**in SAP BW immer 3 gegebene Dimension Tables:**

* + Time: für Zeit- und Datumswerte
  + Unit: für Währungs- und Messeinheiten
  + Data Package: als Protokolldimension für das Laden von packages

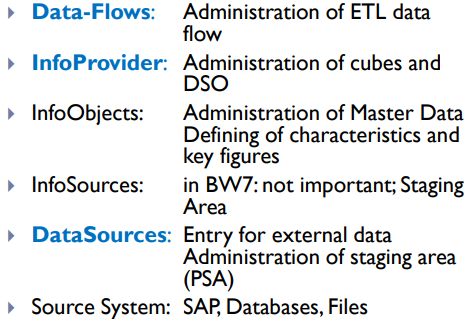
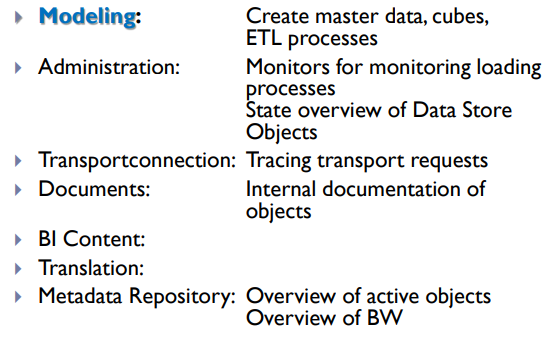
**SAP BW on Hana**

SAP HANA:

* ERP = system supporting enterprise resource planning
* Kompatibel zu fast jedem enterprise system durch die Nutzung von über 1000 Regulierungsschrauben

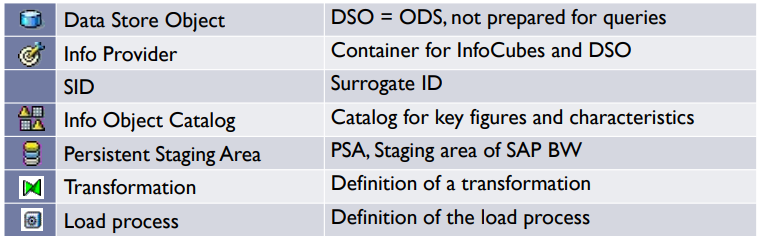
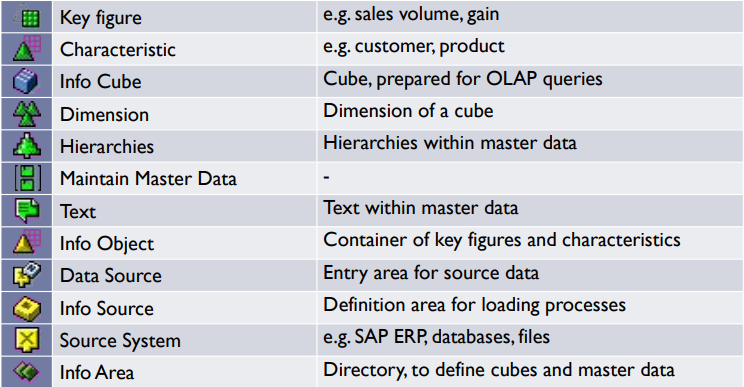
SAP BW:

* Data Warehouse von SAP mit SAP GUI für den Client
* Applikationsserver und relationale Datenbank in Speicherdatenbank (SAP BW on HANA)



Areas in SAP: Modeling:

Bezeichnungen:



Vorgehensweise:

* Anfang:
  + Info Object wird erstellt, definieren von Info Object Catalog, Key figures und Characteristics
  + danach Info Provider, definieren von Master Data, Info Cube und DSO (Data Store Object)
  + als letztes zur Data Source, wo input prozesse, data transfer prozesse und transformationsprozesse definiert werden

SAP HANA Studio:

* definieren von Strukturen im DW, vor allem die benötigten Objekte und aDSO (NUR in Hana Studio)
* Queries für die Daten

SAP Workbench:

* Objekte von HANA sind sichtbar
* externen Inhalt in die Objekte laden (externer Inhalt ist keine Datenbank, sondern CSV-Datei)
* für den Ladeprozess kein Unterschied ob Datenbank oder Dateien
* kompletter ETL-Prozess in der Workbench

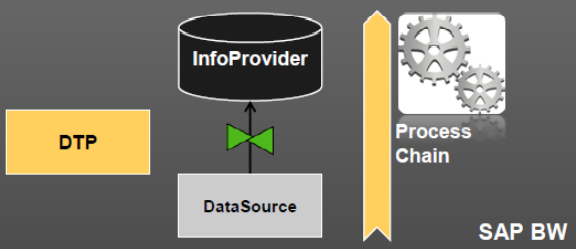
Transaction RSRT:

* RSRT zum analyisieren und debuggen der Abfragen in SAP

InfoObject: Fokus auf business requirements

* Datenstruktur wird auf Infoobjects gebaut, technisch wie eine Datenbank mit Eigenschaften
* am wichtigsten: Charakteristics und Key Figures
  + Charakteristics haben zusätzlich zu Key und Beschreibung weitere Details
    - z.B. „Customer“ hat noch „Location“, „Sales Organisation“, „Country“ ...
    - oft Attribute die eine Hierarchie formen (Hierarchie wird nicht von selbst angezeigt, kann einzeln angezeigt werden)
* InfoObjects sind unique in SAP BW, bauen Datenstruktur für Analyseanforderungen
  + conceptionally unique: Semantik ist einzigartig systemweit definiert
  + logically unique: Datentyp, Value range und field length sind einzigartig systemweit definiert
* Business Transaction: Datensatz, der aus einer Kombination von Characteristics und key figures besteht

InfoArea:

* Verzeichnis zu Definition von Cubes und Master Data

InfoProviders:

* Datencontainer für DSOs und Cubes, speichern Daten physisch
* physische Daten: aDSO (advanced data store objects) 🡪 Daten werden optimiert
* virtuelle Daten: composite providers
* hierarchische Darstellung des gesamten Ladeprozesses von der Quelle bis zum Cube

InfoObjects:

* zu Administration der Master Data, Container für Characteristics und Key Figures
* InfoObjectCatalog: Tabellenspalten für Characteristics und Key Figures

Advanced Data Store Objects:

* können aus InfoObjects oder Datenbankfeldern bestehen
* besteht maximal aus 3 Tabellen:
  + Inbound Table, Change Log und Active Data
  + Daten werden immer auf inbound table geschrieben, im Modus „Activate Data“ werden sie in die Active Data Tabelle geschrieben (während Kompressionsprozess)
  + Direct update: Daten werden direkt auf Active Data Tabelle geschrieben
  + Plannin Mode: Objekt kann zum planen genutzt werden

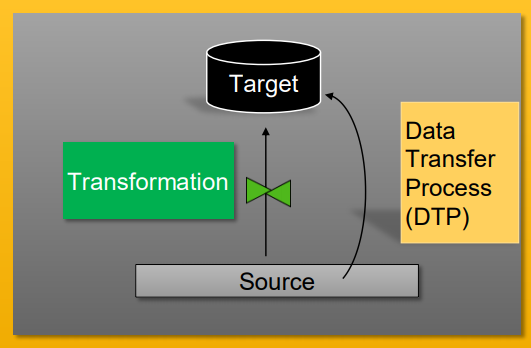
Composite Provider:

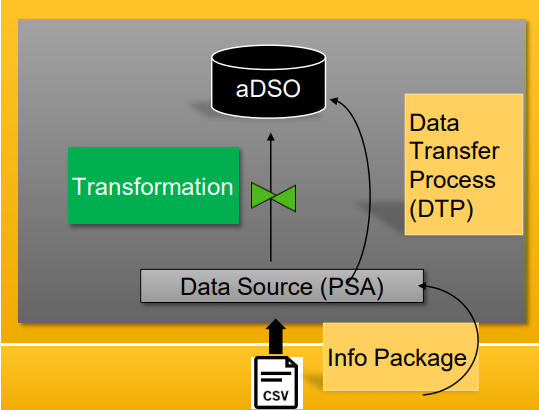
* persistiert keine Daten, kann Daten von verschiedenen InfoProviders zusammenführen
  + UNION und verschieden JOIN Typen werden benutzt
* wird in Queries verwendet, eine Art Sicht auf das aDSO Objekt
* Zwischenschicht zwischen aDSO und Queries

Conversion Routine:

* transformiert externe Daten in internes Format, z.B. Datum YYYYMMDD zu DD.MM.YYYY
* ALPHA Conversion Routine: nimmt Leerzeichen & 0er am Anfang raus („ 1000“ wird zu „1000“)

**ETL-Prozess in SAP:**

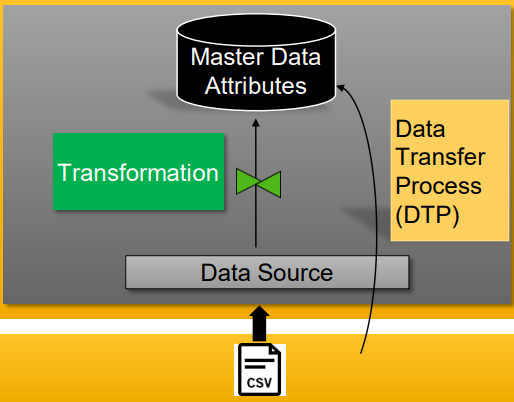
Datenfluss:

* Data Source: interface object zum source system, repräsentiert die source data structure, besteht aus Feldern und adapter settings
* Transformation: transformiert die Daten nach den definierten Regeln
* Data Transfer Process (DTP): triggered Datentransfer von Quelle zu Ziel
* Process Chain: Sequenz von Prozessen, die in einer bestimmten Reihenfolge getriggered werden

Transaktionsdaten:

* Die Daten werden von der Quelldatei zur PSA durch ein **InfoPackage** transferiert
* Data Source hat Persistenz (PSA / Persistent Staging Area)
  + temporäre Tabelle, die Daten im data acquisiton layer speichert, bis sie in die oberen Layer des DW transferiert werden
  + für Transformation, Umwandlung und Transfer der Daten in Cube zuständig

Master Data:

* 2-Schritt-Prozess: Attribute und Beschreibungen (Text) müssen geladen werden
* Data Transfer Process (DTP) kann Datei übertragen, ohne PSA zu betreten (bypassing PSA)
* alle Kennzahlen (Key Figures) und alle Merkmale (Characteristics)
* Master Data Tables gehören zu InfoObjects, wird separat zu transaktionellen Daten geladen
* können Hierarchien und komplexe Strukturen enthalten

Full vs. Delta Data Upload:

* Full: Alle Daten aus der Data Source extrahieren
* Delta: Nur Daten aus Data Soruce extrahieren, die sich seit dem letzten Update geändert haben

Data Transfer Process:

* Monitor: in InfoProvider Management & DataTransferProcess Management integriert
  + Error Stack wird im Monitor gezeigt
* Error Handling:
  + Valid Records Update, No Reporting at all (FAIL)
  + No Update, No Reporting (FAIL)
  + Valid Records Update, No Reporting (FAIL)
  + Valid Records Update, Reporting Possible (OK)

DSOs 🡪 siehe S. 9 oben

Cube:

* aufbereitete Daten für OLAP-Queries
* 3 MUSS-Dimensionen: Data Package, Time, Unit + Modelling Dimension
* extended Star Model: keine Master Data in Dimensionstabellen, sondern nur SID auf Master Data

Transformation von Währungen & Messeinheiten:

* automatisches Hinzufügen von InfoObjects (0Currency, 0Baseum) 🡪 alte Daten aus Data Source müssen gelöscht werden, bevor DataStore Objekt aktiviert wird (sonst inkonsistent)
* alle Währungskonvertierungen werden in einem „conversion type“ gespeichert, kann zur query time konvertiert werden

Selections & Filter:

* mit Selections können Filter direkt an eine Key Figure gebunden werden
  + Einschränkung der Kennzahl
* Filter schränken eine komplette Query ein

Navigationsattribut:

* ein Attribut eines Merkmals, dass in Abfragen aktiv verwendet werden kann
* wird definiert in einem InfoObject (bzw. in einem Merkmal) und entsprechend gekennzeichnet
* wird aktiviert im Composite Provider

Query Designer:

* vorgegebene und frei auswählbare Characteristics:
  + vorgegeben: in Rows und Columns ablegen, werden in Query beim Start sofort verwendet
  + frei auswählbar: in „free Characteristics“ ablegen, **können** in Query verwendet werden

Query:

* liest Daten von einem InfoProvider (CompositeProvider), der auf einem aDSO basiert
* Definiert ein subset von den Characteristics und KeyFigures des Infor Providers und filtert diese
* ist ein multidimensionales Objekt
* wir definieren eine Start View: welche Elemente sind auf Spalten & Zeilen
* kann Formeln, Konditionen, Exceptions, Hierarchien und mehr enthalten
  + Formeln werden lokal in der Query als Teil der KeyFigure Struktur definiert 🡪 nicht wiederverwendbar
  + calculated key figures werden in der query erstellt, aber separat gespeichert und können in anderen queries auf dem selben Inforprovieder wiederverwendet werden
* die Metadaten paramterize den Analytic Manager des BW (also die OLAP Engine)
* dediziertes Tool um die Querys wiederverwendbar zu designen

