Data-Warehouse-Technologien

Prof. Dr.-Ing. Kai-Uwe Sattler[1]

Prof. Dr. Gunter Saake[2]
Dr. Veit Köppen[2]

[1]TU Ilmenau
FG Datenbanken & Informationssysteme

[2]Universität Magdeburg
Institut für Technische und Betriebliche Informationssysteme

Letzte Änderung: 18.10.2019

Teil V

Anfragen an Data Warehouse

Anfragen an DW

Einführung

- Typische Anfragen an Data Warehouses beinhalten Aggregationen, z.B.

> *Wie viele Einheiten wurden in den Produktgruppen Softdrinks und Wein in den Bundesländern Sachsen-Anhalt und Thüringen pro Monat und Ort in den Jahren 2010 und 2011 verkauft und welche Umsätze sind dabei angefallen?*

- Charakteristik typischer Data-Warehouse-Anfragen:
  - Aus der großen Menge vorhandener Fakten wird nur ein bestimmter, in den meisten Dimensionen beschränkter Datenbereich angefragt

Einführung (2)

- Multidimensionale Anfrage:
  - Restriktion, die sich i.d.R. je Dimension auf einfache Klassifikationsknoten bezieht
- Spezielle Optimierungstechniken sinnvoll!
- Problem: Aggregationen auf großen Datenmengen
- Beispiel Getränkehandelskette
  - 2.000 Filialen, pro Filiale: 1.000 Kunden täglich mit je 5 Artikelm
  - pro Einkauf: 1 Artikel Softdrink, 0,5 Artikel Wein
  - pro Tag: 10.000.000 Datensätze in Faktentabelle `Verkauf`, Satzgröße 63 Byte
  - Faktentabelle: ca. 600 MB/Tag, bei 310 Einkaufstagen 182 GB/Jahr
  - Scan der Faktentabelle über 10 Jahre: 6,5 Stunden bei 80 MB/s !!

Relationale Umsetzung multidimensionaler Anfragen

- Grundsätzlich abhängig von Abbildung für Schema
  - Star- vs. Snowflake-Schema
  - Klassifikationshierarchien
- Häufiges Anfragemuster
  - $(n+1)$-Wege-Verbund zwischen n Dimensionstabellen und der Faktentabelle sowie
  - Restriktionen über Dimensionstabellen

Star-Join: Beispiel

```
SELECT O_Stadt, YEAR_MONTH(Z_Datum),
    SUM(V_Anzahl) AS Einheiten,
    SUM(V_Anzahl * P_Verkaufspreis) AS Umsatz
FROM Verkauf, Zeit, Produkt, Ort
WHERE V_Zeit_ID = Z_ID AND V_Produkt_ID = P_ID AND
    V_Ort_ID = O_ID AND
    YEAR(Z_Datum) BETWEEN 2010 AND 2011 AND
    O_Bundesland IN ('Sachsen-Anhalt', 'Thüringen') AND
    P_Produktgruppe IN ('Softdrink', 'Wein')
GROUP BY O_Stadt, YEAR_MONTH(Z_Datum)
```

Star-Join: Aufbau

- **SELECT**-Klausel
  - Kenngrößen mit Aggregatfunktion
  - Ergebnisgranularität (z.B. Monat, Region)
- **FROM**-Klausel
  - Fakten- und Dimensionstabellen
- **WHERE**-Klausel
  - Verbundbedingungen
  - Restriktionen (z.B.: P_Produktgruppe **IN** ('Softdrink', 'Wein') **AND** O_Bundesland **IN** ('Sachsen-Anhalt', 'Thüringen') **AND** **YEAR**(Z_Datum) **BETWEEN** 2010 **AND** 2011)

Gruppierung und Aggregation

- Datenanalyse: Aggregation mehrdimensionaler Daten
- Aggregatfunktion: „dimensionsfreie“ Antwort
  - Standard: **`SUM`**, **`MIN`**, **`MAX`**, **`COUNT`**
  - Erweiterungen: statistische, physikalische und Finanzfunktionen
  - Benutzerdefinierte Aggregatfunktionen
- Gruppierung: „1-dimensionale“ Antwort
  - Ergebnis: Tabelle mit Aggregatwerten indiziert durch Menge von Attributen

Gruppierung und Aggregation (2)

- SQL: **GROUP BY** attrib_liste [**HAVING** bedingung]
  - Gruppierung bzgl. gleicher Werte der Gruppierungsattribute
  - Abschließende Projektion nur über Gruppierungsattribute oder Aggregationen
- Einschränkungen
  - Berechnung von Histogrammen: Aggregationen über berechnete Kategorien

    ```
    ... GROUP BY func(Zeit) AS Woche ...
    ```
  - Berechnung von Zwischen- und Gesamtsummen
  - Berechnung von Kreuztabellen

Aggregatfunktionen

- Standard-SQL-Funktionen wie **`MIN`**, **`MAX`**, **`SUM`**, **`COUNT`**, **`AVG`**
- neue Funktionen in SQL:2003 für Varianz **`VAR_POP`** (x), Standardabweichung **`STDDEV_POP`** (x), Kovarianz **`COVAR_POP`** (x, y) und Korrelationskoeffizienten **`CORR`** (x, y)
- jeweils für die gesamte Population (**`_POP`**) bzw. mit Bessel-Korrektur (**`_SAMP`**)

Aggregatfunktionen: Beispiele

- Existiert ein (linearer) Zusammenhang zwischen Anzahl der verkauften Produkte und deren Verkaufspreis?

```
SELECT opCOVAR_POP(V_Anzahl, P_Verkaufspreis)
FROM Verkauf, Produkt
WHERE V_Produkt_ID = P_ID
```

- Werte nahe Null $\approx$ nicht stärker als statistischer Zufall

Aggregatfunktionen: Beispiele

- Kovarianz gibt keinen Aufschluss über Stärke der Korrelation, besser Korrelationskoeffizient

```
SELECT CORR(P_Verkaufspreis, P_Einkaufspreis),
   P_Produktgruppe
FROM Verkauf, Produkt
WHERE V_Produkt_ID = P_ID
GROUP BY P_Produktgruppe
```

- Werte ab $0{,}5$ deuten auf mittlere bis starke Korrelation

Aggregatfunktionen: Beispiele

- Regressionsanalyse für Zusammenhang zwischen Anzahl und Verkaufspreis
- Berechnung von Geradenanstieg **REGR_SLOPE**, Regressionskoeffizienten **REGR_R2**, mittleren Preis **REGR_AVGX** und mittlere Anzahl **REGR_AVGY**

```
SELECT V_Kanal,
   REGR_SLOPE(V_Anzahl, P_Verkaufspreis) AS Anstieg,
   REGR_R2(V_Anzahl, P_Verkaufspreis) AS Koeff,
   REGR_COUNT(V_Anzahl, P_Verkaufspreis) AS Anzahl,
   REGR_AVGX(V_Anzahl, P_Verkaufspreis) AS MPreis,
   REGR_AVGY(V_Anzahl, P_Verkaufspreis) AS MAnzahl
FROM Verkauf, Produkt, Zeit
WHERE V_Produkt_ID = P_ID AND
V_Zeit_ID = Z_ID AND YEAR(Z_Datum) = 2011
GROUP BY V_Kanal
```

Berechnung von Zwischen- und Gesamtsummen

PGruppe	Jahr	Bundesland	Umsatz PGruppe-Jahr-Bundesland	Umsatz PGruppe-Jahr	Umsatz PGruppe	Umsatz
Wein	2010	Sachsen-Anhalt Thüringen	45 43			
				88		
	2011	Sachsen-Anhalt	47			
				47		
					135	
Bier	2011	Thüringen	42			
				42		
					42	
						177

Berechnung von Zwischen- und Gesamtsummen (2)

```
-- Zwischensumme (1) über alle Produktgruppen, Jahre und Bundesländer
SELECT P_Produktgruppe AS PGruppe, YEAR(Z_Datum), O_Bundesland,
   SUM(V_Anzahl * P_Verkaufspreis) AS Umsatz
FROM Verkauf, Zeit, Produkt, Ort
WHERE V_Zeit_ID = Z_ID AND V_Produkt_ID = P_ID AND V_Ort_ID = O_ID
GROUP BY P_Produktgruppe, YEAR (Z_Datum), O_Bundesland
UNION ALL
-- Zwischensumme (2) über alle Produktgruppen und Jahre
SELECT P_Produktgruppe AS PGruppe, YEAR (Z_Datum),
   CAST(NULL AS VARCHAR(50)),
   SUM(V_Anzahl * P_Verkaufspreis) AS Umsatz
FROM Verkauf, Zeit, Produkt, Ort
WHERE V_Zeit_ID = Z_ID AND V_Produkt_ID = P_ID AND V_Ort_ID = O_ID
GROUP BY P_Produktgruppe, YEAR(Z_Datum)
UNION ALL
```

Berechnung von Zwischen- und Gesamtsummen (3)

```
-- Zwischensumme (3) über alle Produktgruppen
SELECT P_Produktgruppe AS PGruppe, CAST(NULL AS INT),
   CAST(NULL AS VARCHAR(50)),
   SUM(V_Anzahl * P_Verkaufspreis) AS Umsatz
FROM Verkauf, Zeit, Produkt, Ort
WHERE V_Zeit_ID = Z_ID AND V_Produkt_ID = P_ID AND V_Ort_ID = O_ID
GROUP BY P_Produktgruppe
UNION ALL
-- Gesamtsumme
SELECT CAST(NULL AS VARCHAR(50)) AS PGruppe, CAST(NULL AS INT),
   CAST(NULL AS VARCHAR(50)),
   SUM(V_Anzahl * P_Verkaufspreis) AS Umsatz
FROM Verkauf, Zeit, Produkt, Ort
WHERE V_Zeit_ID = Z_ID AND V_Produkt_ID = P_ID AND V_Ort_ID = O_ID
```

Ausschnitt der Zwischen- und Gesamtsummen

PGruppe	Jahr	O_Bundesland	Umsatz
Wein	2010	Sachsen-Anhalt	45
Wein	2010	Thüringen	43
Wein	2011	Sachsen-Anhalt	47
Bier	2011	Thüringen	42
Wein	2010	*NULL*	88
Wein	2011	*NULL*	47
Bier	2011	*NULL*	42
Wein	0	*NULL*	135
Bier	0	*NULL*	42
NULL	0	*NULL*	177

Nachteile der UNION-Variante

- Hoher Aufwand:
  - Berechnung aller Teilsummen für n Gruppierungsattribute erfordert 2^n Teilanfragen
  - Eventuelle Verbundoperationen müssen mehrfach wiederholt werden
- Aufwendige Formulierung:
  - Jedoch eventuell Generierung durch OLAP-Werkzeuge
  - Einhalten der Struktur

Berechnung von Kreuztabellen

- Symmetrische Aggregation
- Auch Pivot-Tabellen

Verkäufe	2010	2011	Gesamt
Thüringen	120	135	255
Sachen-Anhalt	135	140	275
Gesamt	255	275	530

PIVOT in SQL Server

```
SELECT Jahr, [THÜR] AS Thüringen,
    [SANH] AS Sachsen-Anhalt
FROM Verkauf
PIVOT (SUM(Verkäufe) FOR
    Region IN ([THÜR], [SANH]))
```

Jahr	Thüringen	Sachsen-Anhalt
2010	135	120
2011	140	135

Cube-Operator

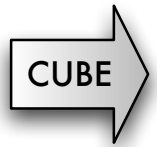
- „Kurzform“ für Anfragemuster zur Berechnung von Teil- und Gesamtsummen
- Generierung aller möglichen Gruppierungskombinationen aus gegebener Menge von Gruppierungsattributen
- Ergebnis: Tabelle mit aggregierten Werten
- Gesamtaggregat:

$$\texttt{NULL}, \texttt{NULL}, \ldots, \texttt{NULL}, f(*)$$

- Höherdimensionale Ebenen mit weniger `NULL`-Werten

Cube-Operator: Beispiel

PGruppe	Bundesland	Jahr	Umsatz
Wein	Sachsen-Anhalt	2010	45
Wein	Thüringen	2010	43
Wein	Sachsen-Anhalt	2011	47
Bier	Thüringen	2011	42



PGruppe	Jahr	Bundesland	Umsatz
Wein	2010	Sachsen-Anhalt	45
Wein	2010	Thüringen	43
...	...	...	...
Wein	2010	NULL	88
Wein	2011	NULL	47
Bier	2011	NULL	42
Wein	NULL	Sachsen-Anhalt	92
Wein	NULL	Thüringen	43
Bier	NULL	Thüringen	42
Wein	NULL	NULL	135
Bier	NULL	NULL	42
NULL	2010	Sachsen-Anhalt	45
...	...	...	...
NULL	NULL	Sachsen-Anhalt	92
NULL	NULL	Thüringen	85
...	...	...	...
NULL	2010	NULL	88
NULL	2011	NULL	89
NULL	NULL	NULL	177

Cube: Details

- Kardinalität
  - N Attribute mit Kardinalität $C_1, C_2, ..., C_N$
  - Gesamtkardinalität des **`CUBE`**:

$$\prod_{i=1}^{N}(C_i + 1)$$

- Anzahl der Super-Aggregatwerte
  - N Attribute in der **`SELECT`**-Klausel
  - Super-Aggregate: $2^N - 1$

Cube-Operator: SQL-Syntax

- Implementierung in SQL Server, DB2, Oracle
- Syntax ORACLE:

```
SELECT P_Produktgruppe AS PGruppe,
   O_Bundesland, YEAR(Z_Datum),
   SUM(V_Anzahl * P_Verkaufspreis) AS Umsatz
FROM Verkauf, Zeit, Produkt, Ort
WHERE ...
GROUP BY CUBE(P_Produktgruppe, O_Bundesland,
   YEAR(Z_Datum))
```

- Funktion **GROUPING**(Attribut)
  - Liefert Wert = 1, wenn über Attribut aggregiert wurde
  - Liefert Wert = 0, wenn nach Attribut gruppiert wurde
- Unterdrückung von Teilsummen, z.B. der Gesamtsumme

```
... HAVING NOT (GROUPING(P_Produktgruppe) = 1 AND
                GROUPING(O_Bundesland) = 1 AND
                GROUPING(YEAR(Z_Datum)) = 1)
```

Rollup-Operator

- **`CUBE`**-Operator: interdimensional
  - anwendbar für Attribute aus unterschiedlichen Dimensionen
  - Für Roll-Up oder Drill-Down-Operationen zu aufwendig
- **`ROLLUP`**-Operator: intradimensional
  - Generierung der Attributkombinationen

$$(A_1, ..., A_N), (A_1, ..., A_{N-1}), (A_1, A_2), (A_1), ()$$

für gegebene Attribut*liste* $A_1, ..., A_N$

ROLLUP-Operator: Beispiel (einfach)

- Anfrage:

```
SELECT P_Gruppe, Z_Tag, Z_Monat, Z_Jahr,
   SUM(V_Anzahl * P_Verkaufspreis) AS Umsatz
FROM Verkauf, Zeit, Produkt, Ort
WHERE V_Produkt_ID = P_ID AND V_Ort_ID = O_ID AND
   V_Zeit_ID = Z_ID AND YEAR(Z_Datum) = 2011 AND
   P_Produktgruppe = 'Rotwein'
GROUP BY ROLLUP(Z_Jahr, Z_Monat, Z_Tag)
```

- Auswertung:
  - Rollup: `(Z_Jahr,Z_Monat,Z_Tag),`
    `(Z_Jahr,Z_Monat),(Z_Jahr),()`

ROLLUP-Operator: Beispiel (einfach)

Gruppe	Tag	Monat	Jahr	Umsatz
Rotwein	1	Januar	2011	100
Rotwein	2	Januar	2011	100
...	...	...	...	...
Rotwein	31	Januar	2011	100
Rotwein	NULL	Januar	2011	2000
Rotwein	1	Februar	2011	100
Rotwein	2	Februar	2011	100
...	...	...	...	...
Rotwein	28	Februar	2011	100
Rotwein	NULL	Februar	2011	2000
...	...	...	...	...
Rotwein	NULL	NULL	2011	24000
Rotwein	NULL	NULL	NULL	24000

ROLLUP-Operator: Beispiel (zusammengesetzt)

- Anfrage:

```
SELECT P_Kategorie, P_Gruppe, O_Land, O_Region
   SUM(V_Anzahl) AS Verkäufe
FROM Verkauf, Zeit, Produkt, Ort
WHERE V_Zeit_ID = Z_ID AND V_Produkt_ID = P_ID
   AND V_Ort_ID = O_ID AND YEAR(Z_Datum) = 2011
GROUP BY ROLLUP(P_Kategorie, P_Gruppe),
   ROLLUP(O_Land, (O_Region))
```

- Auswertung:
  - 1. Rollup: `(P_Kategorie,P_Gruppe), (P_Kategorie),()`
  - 2. Rollup: `(O_Land,O_Region),(O_Land),()`
  - Kreuzprodukt beider Kombination

ROLLUP-Operator: Beispiel (zusammengesetzt)

P_Kategorie	P_Gruppe	Land	Region	Verkäufe
Wein	Weißwein	D	SANH	102
Wein	Rotwein	D	SANH	98
Wein	NULL	D	SANH	200
...	...	...	...	...
Wein	Weißwein	D	NULL	541
Wein	Rotwein	D	NULL	326
Wein	NULL	D	NULL	867
...	...	...	...	...
Wein	NULL	D	NULL	1232
...	...	...	...	...
NULL	NULL	D	NULL	1432
...	...	...	...	...
NULL	NULL	NULL	NULL	3456

CUBE- vs. ROLLUP-Operator

- CUBE-Operator:
  - Generiert alle 2^n Kombinationen:
    - z.B. für 4 Gruppierungsattribute 16 Kombinationen
- ROLLUP-Operator:
  - Generiert nur Kombinationen mit Superaggregaten:
    - $(f_1, f_2, ..., f_n)$,
    - ...
    - $(f_1, NULL, ..., NULL)$,
    - $(NULL, NULL, ..., NULL)$
  - $n + 1$ Kombinationen

GROUPING SETS

- SQL: 2003-Gruppierung

```
GROUP BY ... GROUPING SETS (gruppierung)
```

- Gruppierung:
  - Einfache Gruppierungskombination, z.B.: `(O_Bundesland, O_Stadt)`
  - Komplexe Gruppierungsbedingung mit **`CUBE`** oder **`ROLLUP`**

GROUPING SETS: Beispiel

- Anfrage

```
...
GROUP BY
    ROLLUP(P_Produktgruppe, P_Produktkategorie),(1)
    GROUPING SETS((O_Stadt), (O_Bundesland)),(2)
    GROUPING SETS(
        ROLLUP(Jahr, Quartal, Monat), (Woche))(3)
```

- Bedeutung
  - (1) entlang der Klassifikationshierarchie
  - (2) nur für Städte und Bundesländer
  - (3) Nutzung der Parallelhierarchie `(Jahr→Quartal→Monat)` sowie `(Woche)`

Iceberg-Cube: Motivation

- Probleme der CUBE-Berechnung (Beispiel)
  - 9-dimensionaler Datensatz (Daten von Wetterstationen)
  - $1.015.367$ Tupel (ca. $39MB$)
  - CUBE: $210.343.580$ Tupel (ca. 8 GB $\approx 200\times$ Eingangsdaten)
  - 20% aller GROUP-BYs nahezu ohne Aggregation (Größe: ca. 1)
  - Berechnung der GROUP-BYs mit mind. 2 Eingangstupeln: nur $50\times$ Eingangsdaten!
  - Für mind. 10 Tupel: nur $5\times$ Eingangsdaten!
- Idee Iceberg-Cube: Berechne nur Aggregationen, die einen minimalen Support haben

Iceberg-Cube

- Berechnung der Gruppierungen (Partitionen), die Aggregatselektionsbedingung erfüllen

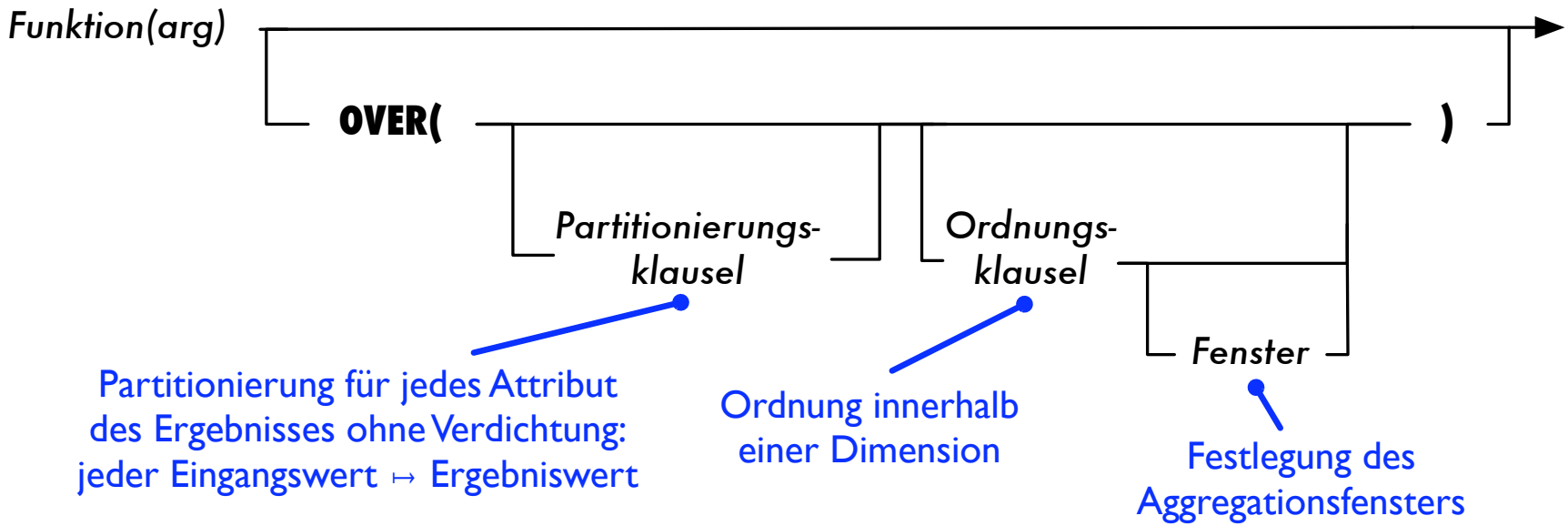
```
SELECT A, B, C, COUNT(*), SUM(X)
FROM R
GROUP BY CUBE(A, B, C)
HAVING COUNT(*) >= N
```

- N: minimaler Support einer Partition
- Spezielle Optimierung möglich
  - Pruning: „Abschneiden“ von Partitionen, die minimalen Support nicht erfüllen

SQL:2003 – Sequenzbasierte Operationen

- Seit SQL:1999 – Erweiterung um OLAP-Funktionen zur attribut- und sequenzbasierten Auswertung
- Attribut- und tupelbasierte Aggregation
- Umsetzung u.a. in Oracle und DB2
- Unterstützte Anfragetypen
  - Ratio-To-Total
  - Laufende Summen (Kumulation)
  - Gleitender Durchschnitt
  - Ranking-Analyse

OLAP-Funktionen: Syntax



Sichtdefinition für folgende Beispiele

```
CREATE VIEW TagesUmsatz AS
    SELECT P_Produktgruppe, Z_Datum,
        SUM(V_Anzahl * P_Verkaufspreis) AS Umsatz
    FROM Verkauf, Zeit, Produkt
    WHERE V_Zeit_Id = Z_Id AND V_Produkt_Id = P_Id
    GROUP BY P_Produktgruppe, Z_Datum
```

OLAP-Funktionen: Motivation

- Ratio-To-Total-Analyse
  - Berechnung des Tagesumsatzes am Gesamtumsatz des Monats
  - Klassische SQL-Anfrage:

```
SELECT Z_Datum, Umsatz,GesamtUmsatz AS MonatGesamt
   100.0*Umsatz/GesamtUmsatz AS Anteil,
FROM TagesUmsatz,
   (SELECT SUM(Umsatz) AS GesamtUmsatz
   FROM TagesUmsatz
   WHERE P_Produktgruppe = 'Wein' AND
      YEAR_MONTH(Z_Datum) = 201108) Gesamt
WHERE P_Produktgruppe = 'Wein' AND
   YEAR_MONTH(Z_Datum) = 201108
```

  - Innere Unteranfrage berechnet die Gesamtmenge für die Anteilsberechnung:

```
( SELECT SUM(Umsatz) AS GesamtUmsatz
  FROM TagesUmsatz WHERE ...)
```

Formulierung mittels OLAP-Funktion

- Anfrage:

```
SELECT Z_Datum, Umsatz,
    100.0*Umsatz/SUM(Umsatz) OVER() AS Anteil,
    SUM(Umsatz) OVER() AS MonatGesamt
FROM TagesUmsatz
WHERE P_Produktgruppe = 'Wein' AND
    YEAR_MONTH(Z_Datum) = 201108
```

- OLAP-Funktion

```
SUM(Umsatz) OVER()
```

  - Aggregation über gesamten Eingangsbereich
  - Partition für Aggregation wird lokal für jeden Eintrag generiert

Ergebnisrelation

Datum	Umsatz	Anteil	MonatGesamt
01-AUG-2011	58	4,669	1242
02-AUG-2011	52	4,186	1242
03-AUG-2011	64	5,152	1242
04-AUG-2011	0	0,000	1242
	...		
31-AUG-2011	47	3,784	1242
	...		

Attributlokale Partitionierung

- Partitionierung des Eingabestroms einer OLAP-Funktion (ähnlich Gruppierung)
- Aber: Partitionierung erfolgt pro Attribut/Anweisung der Aggregrationsoperation
  - Ermöglicht Nachgruppierung
- Beispiel: Ermittlung der Anteile der Tagesumsätze im Vergleich zum Monatsumsatz

```
SELECT P_Produktgruppe, Z_Datum, Umsatz,
   100.0*Umsatz/SUM(Umsatz)
      OVER( PARTITION BY YEAR_MONTH(Z_Datum),
         P_Produktgruppe) AS MonatAnteil,
   SUM(Umsatz)
      OVER( PARTITION BY YEAR_MONTH(Z_Datum),
         P_Produktgruppe) AS MonatGesamt
FROM TagesUmsatz
```

Attributlokale Partitionierung: Details

- Prinzip:

```
SUM(Menge) OVER(PARTITION BY MONTH(Z_Datum))
```

- Spezifikationstext hinter **OVER** heisst Partitionierungsschema
- Keine Konflikte durch unterschiedliche Partitionierungsschemata innerhalb einer Anfrage
  - Jeweils alle Einträge einer Partition in Berechnung einbezogen

Sequenzorientierte Analyse

- Spezifikation einer attributlokalen Ordnung für Partitionen
- Anwendung: laufende Summe, gleitender Durchschnitt, etc.
- Beispiel: kumulierte Umsatzzahlen der Weine über Gesamtzeitraum und pro Monat

```
SELECT Z_Datum,
    SUM(Umsatz) OVER(
          ORDER BY Z_Datum) AS SummeGesamt,
    SUM(Umsatz) OVER(
          PARTITION BY YEAR_MONTH(Z_Datum)
          ORDER BY Z_Datum) AS SummeMonat
FROM TagesUmsatz
WHERE P_Produktkategorie = 'Wein'
```

Sequenzorientierte Analyse: Prinzip

- Anzahl der Tupel, die in ein Ergebnistupel eingehen entspricht Position des Tupels bzgl. gegebener Ordnung
- Eingangstupel t_i, Ergebnistupel s_i

$$
\begin{array}{lcll}
t_1 & \longrightarrow & SUM(\{t_1\}) & \longrightarrow & s_1 \\
t_2 & \longrightarrow & SUM(\{t_1, t_2\}) & \longrightarrow & s_2 \\
t_3 & \longrightarrow & SUM(\{t_1, t_2, t_3\}) & \longrightarrow & s_3 \\
 & & \ldots & &
\end{array}
$$

- Schrittweise Vergrößerung des Analysefensters

Nutzung für Ranking-Analysen

- Funktionen
  - **RANK**() : liefert Rang eines Tupels bzgl. vorgegebener Ordnung innerhalb der Partition
    - ★ Bei Duplikaten gleicher Rang (mit Lücken)
  - **DENSE_RANK**() : wie **RANK**(), jedoch ohne Lücken
- Beispiel: Ranking nach Umsatz

```
SELECT Z_Datum, RANK()
   OVER(ORDER BY Umsatz DESC) AS Rang
FROM TagesUmsatz
WHERE P_Produktgruppe = 'Wein'
```

Ranking-Analyse: Beispiel

- Beschränkung von „Hitlisten“
- Beispiel: Top-3 der Tage mit den höchsten Umsatzzahlen pro Monat
- Anfrage:

```
SELECT P.Z_Datum, P.TopMonat
FROM (SELECT Z_Datum, P_Produktgruppe,
       RANK() OVER(
          PARTITION BY YEAR_MONTH(Z_Datum)
          ORDER BY Umsatz DESC) AS TopMonat
       FROM TagesUmsatz) P
WHERE P.TopMonat <= 3 AND
    P.P_Produktgruppe = 'Wein'
ORDER BY P.TopMonat DESC
```

Bildung dynamischer Fenster

- Bisher: nur wachsende Fenstergröße für Partition
- Jetzt: explizite Angabe des Fensters
  - **`ROWS`**: Anzahl der Tupel
  - **`RANGE`**: Anzahl der wertmäßig verschiedenen Tupel
- Anwendung: gleitender Durchschnitt
- Ausgehend von definierten Startpunkt bis zum aktuellen Tupel
  - **`UNBOUNDED PRECEDING`**: erstes Tupel der jeweiligen Partition
  - `n` **`PRECEDING`**: n-ter Vorgänger relativ zur aktuellen Position
  - **`CURRENT ROW`**: aktuelles Tupel (nur mit **`RANGE`** und Duplikaten sinnvoll)

Bildung dynamischer Fenster (2)

- Angabe der unteren und oberen Schranken

```
BETWEEN untereGrenze AND obereGrenze
```

- Spezifikation der Grenzen
  - **UNBOUNDED PRECEDING**
  - **UNBOUNDED FOLLOWING**
  - `n` **PRECEDING**
  - `n` **FOLLOWING**
  - **CURRENT ROW**
- `obereGrenze` muss höhere Position als `untereGrenze` spezifizieren

Dynamische Fenster: Beispiel

- Gleitender Durchschnitt mit 5-Tage-Fenster auf Monatsebene

```
SELECT Z_Datum, AVG(Umsatz) OVER(
       PARTITION BY YEAR_MONTH(Z_Datum)
       ORDER BY Z_Datum
       ROWS BETWEEN 2 PRECEDING
       AND 2 FOLLOWING) AS Durch5Tage
FROM TagesUmsatz
WHERE P_Produktkategorie = 'Wein'
```

Multidimensionale Erweiterungen: MDX

- OLE DB for OLAP (Microsoft)
- OLE DB
  - COM-Objekte und -Schnittstellen für Datenbankzugriff
  - Kommunikation zwischen Datenprovider und -konsumenten (Clients)
  - „Treiber“-Konzept
- OLE DB for OLAP
  - Zugriff auf multidimensionale Daten

OLE DB for OLAP: Schema und Sprache

- Multidimensionales Schema
  - Würfel
  - Dimensionen mit Ebenen
  - Kenngrößen als Dimensionen
- MDX: Multidimensional Expressions
  - Spezifikation von multidimensionalen Datensätzen mit Hilfe OLAP-typischen Anfragen
  - Makros auf Basis von SQL
  - Provider
    - RDBMS: Abbildung auf SQL
    - Multidimensionale DBMS: Abbildung auf eigene Sprache

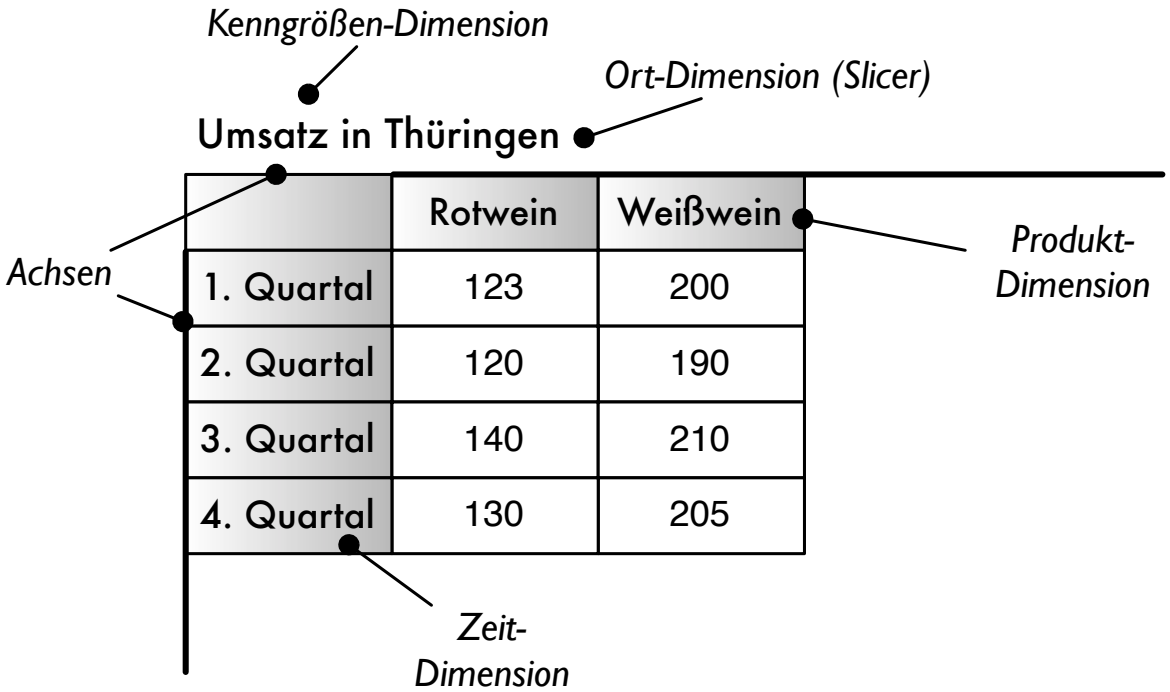
MDX: Statement

- Spezifikation von
  - Würfel
  - Anzahl der Achsen
  - Dimensionen, die auf Achsen projiziert werden und deren Schachtelung
  - Dimensionselemente und deren Sortierreihenfolge
  - Dimensionselemente von nicht-projizierten Dimensionen zur Filterung der Daten („Slicer“)
- Syntax

```
SELECT achsen-spez [, achsen-spez ]
FROM cube-spez WHERE slicer-spez
```

Aufteilung der Dimensionen

- Achsen-Dimensionen: Daten für mehrere Elemente
- Slicer-Dimensionen: Daten für ein Element
- Beispiel: Dimensionen Geographie, Produkt, Zeit (Quartale), Kenngrößen



	Rotwein	Weißwein
1. Quartal	123	200
2. Quartal	120	190
3. Quartal	140	210
4. Quartal	130	205

Beispiel CUBE

Umsätze 2011

		Rotwein			Weißwein		
		S.-A.		Thür.	S.-A.		Thür.
		Magdeb.	Halle		Magdeb.	Halle	
1. Quart.	**Jan**	14	12	25	12	9	22
	Feb	13	10	22	11	9	21
	Mär	15	14	23	11	10	22
2. Quart.		42	40	82	39	37	75
3. Quart.		44	42	80	37	35	73
4. Quart.	**Okt**	13	12	23	10	10	22
	Nov	14	12	24	9	10	21
	Dez	16	14	26	12	11	20

Spezifikation der Achsen

- Achse: Menge von Tupeln
  - 1 Dimension: Tupel aus einem Element
  - n Dimensionen: Tupel aus n Elementen
  - Beispiel:

    ```
    (Rotwein, Sachsen-Anhalt), (Weißwein, Thüringen),
    (Rotwein, Magdeburg), (Rotwein, Halle),
    (Weißwein, Erfurt)
    ```
- Spezifikation
  - Ausdruck zur Erzeugung von Elementen
  - Explizite Angabe der Elemente
  - Kreuzprodukt von Elementen

Spezifikation der Achsen: Beispiele

- Aufzählung

  `{Rotwein,Weißwein}`
- Mengenausdrücke

  `Sachsen-Anhalt.CHILDREN` liefert `{Magdeburg,Halle}`
- Kreuzprodukt

```
CROSSJOIN({Rotwein,Weißwein},
          {Sachsen-Anhalt.CHILDREN,Thüringen})
```

Projektion auf Achsen

- Zuordnung der Tupelmengen der Dimensionen zu Achsen
- Notation:

  `set` **`ON`** `achsen-bezeicher`
- *achsen-bezeichner*
  - **`ROWS`**
  - **`COLUMNS`**
  - **`AXIS`**(index)
- Beliebige Anzahl von Achsen möglich

Spezifikation der Slicer

- Slicer-Dimensionen:
  - Dimensionen, die nicht zu Achsen zugeordnet sind
  - Filterung bezüglich dieser Dimensionen
- Slicer: Tupel
  - Beispiel: `WHERE (Verkauf, [2010], Produkte.[All])`
  - Auswahl der Kenngröße „Verkäufe“, der Zahlen aus 2010 und aller Produkte

Gesamtanfrage

```
SELECT CROSSJOIN(
       {Produkt.Kategorie.Gruppe.[Rotwein],
       Produkt.Kategorie.Gruppe.[Weißwein]},
    {Ort.[Sachsen-Anhalt].CHILDREN,Ort.[Thüringen]})
       ON COLUMNS,
    {Zeit.[2011].[Q1].CHILDREN,Zeit.[2011].[Q2],
       Zeit.[2011].[Q3],Zeit.[2011].[Q4].CHILDREN}
       ON ROWS
FROM Verkauf
WHERE (Measures.[Umsatz])
```

Weitere Funktionen

- Filterung von Mengen basierend auf Bedingung
- weitere Mengenoperationen: **UNION**, **EXCEPT**, **INTERSECT**
- Beispiel: Umsatzzahlen pro Produkt, jedoch nur Städte, bei denen der Umsatz höher als im Dezember des Vorjahres war:

```
SELECT Produkt.Kategorie.Gruppe.CHILDREN
       ON COLUMNS,
   FILTER(Ort.[Thüringen].CHILDREN,
          (Measures.[Umsatz], Zeit.[2011].CHILDREN) >
          (Measures.[Umsatz],
          Zeit.[2010].[Dezember])) ON ROWS
FROM Verkauf
WHERE (Measures.[Umsatz])
```

Weitere Funktionen (2)

- Einschränkung bzgl. Kennzahlen über **TOPCOUNT**-Operator
  - „Top10-Städte nach Verkaufszahlen“

```
SELECT ...,
{TOPCOUNT(Ort.[Sachsen-Anhalt].CHILDREN, 10,
Verkauf)} ON ROWS FROM ...
```

- Zeitreihen
  - **PERIODSTODATE**`(Quartal, [22-Nov-10])`:
    liefert Zeitintervall `01.10.10-22.11.10`
  - **LASTPERIODS**`(2, [Nov-10])`:
    liefert `{[Sep-10], [Okt-10]}`
  - **PARALLELPERIOD**`(Jahr,2, [Nov-10])`:
    liefert `[Nov-08]`

Calculated Members: Berechnung eigener Kennzahlen

- Beispiel: prozentualer Anteil einer Filiale am Umsatz des Ortes

```
WITH MEMBER Measures.FilialAnteil AS
   '(Measures.[Umsatz],
      Ort.[Thüringen].Stadt.Filiale) /
      (Measures.[Umsatz], Ort.[Thüringen].Stadt)',
   FORMAT_STRING = '0.00%'
SELECT Ort.[Thüringen].Stadt.Filiale ON COLUMNS,
   Zeit.[2011].Quartal.Monat.MEMBERS ON ROWS
FROM Verkauf
WHERE (Measures.[FilialAnteil])
```

Calculated Members: Berechnung eigener Dimensionselemente

- Beispiel: Umsatz pro Produktkategorie und Quartal für die Region Mitteldeutschland

```
WITH MEMBER Ort.[Mitteldeutschland] AS
   'SUM({ Ort.[Sachsen-Anhalt], Ort.[Thüringen],
          Ort.[Sachsen]})',
SELECT Produkt.Kategorie.MEMBERS ON COLUMNS,
   Zeit.[2011].CHILDREN ON ROWS
FROM Verkauf
WHERE (Measures.[Umsatz], Ort.[Mitteldeutschland])
```

Zusammenfassung

- Standard-Anfragesprachen für Data-Warehouse-Datenbanken: SQL und MDX
- Star-Join-Anfragen als typisches Muster einer SQL-Anfrage
- spezielle SQL-Erweiterungen für Gruppierungen und Aggregationen
- MDX als „multidimensionale“ Anfragesprache auf Kennzahlen und Dimensionen ausgerichtet