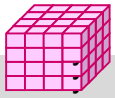
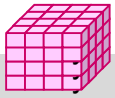
3. Mehrdimensionale Datenmodellierung und Operationen

- Grundlagen
  - Kennzahlen, Dimensionen, Cube
  - Cuboide / **Aggregationsgitter**
  - hierarchische Dimensionen / Cube-Operationen
- multi-dimensionale Speicherung (MOLAP)
  - MDX-Abfragen
- relationale Repräsentation mehrdimensionaler Daten (ROLAP)
  - Star-Schema
  - Varianten: Snowflake-, Galaxien-Schema
- mehrdimensionale Gruppierungen in SQL
  - Star Join
  - Group-By-Erweiterungen**:** Cube, Rollup, Grouping Sets
- Auswertung von Zeitreihen/Sequenzen mit SQL
  - RANK-, WINDOW-Queries

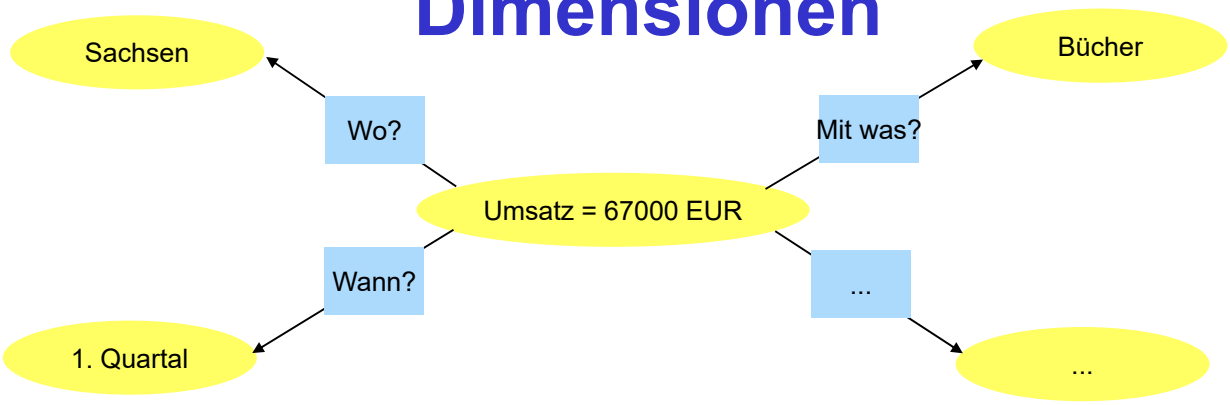


Kennzahlen

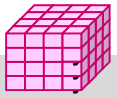
- Kennzahl ist numerische Größe mit konzentrierter Aussagekraft zur Diagnose, Überwachung und Steuerung eines Systems
  - auch: Fakten, Meßgrößen, Measures, Key Performance Indicators (KPI)
  - meist betriebswirtschaftliche Größen, z.B. Umsatz / Gewinn / Rentabilität
  - KPIs oft aus einfacheren Kennzahlen abgeleitet: Umsatz pro Kunde, Liefertreue, Anlagenauslastung, ROI
- Kennzahlen besitzen beschreibende Attribute
  - z.B. Einheit, Wertebereich, Berechnungsvorschrift
- Arten von Kennzahlen
  - *additive Kennzahlen* (z.B. Umsatz) bzw. *semi-additive* Kennzahlen, für die additive Aggregation nur bzgl ausgewählter Dimensionen möglich ist (z.B. Produktbestand)
  - *nicht-additive* Kennzahlen wir Durchschnittswerte oder Prozentanteile



Dimensionen

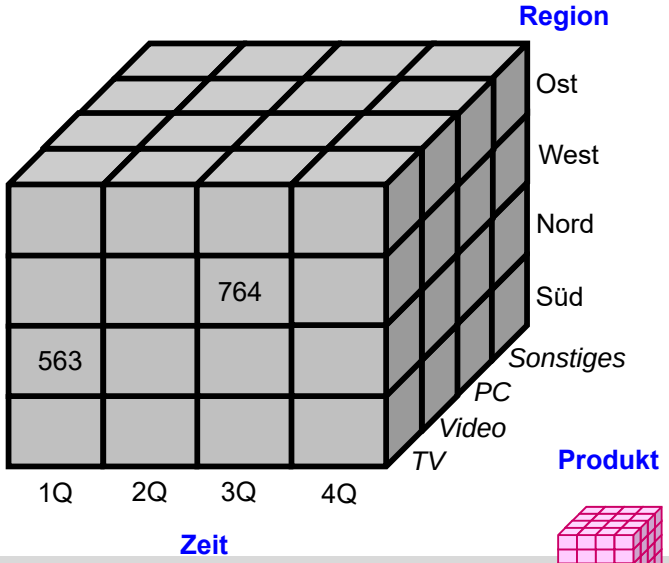


- Zahlenwert einer Kennzahl ist ohne semantischen Bezug nichtssagend
  - Dimensionen setzen Kennzahlen in Bezug zu Eigenschaften / sachlichen Kriterien
- *Dimension*: Datentyp, i.a. endlich (z.B. Aufzählung)
  - Beispiele: Menge aller Produkte, Regionen, Kunden, Zeitperioden etc.
  - *Dimensionselement*: Element / Ausprägung / Wert zu einer Dimension
  - *Klassifikations-/Kategorienattribute* (inkl. eines *Primärattributs* für detaillierteste Stufe)
  - *dimensionale Attribute* : zusätzliche beschreibende Eigenschaften, z.B. Produktfarbe / Gewicht

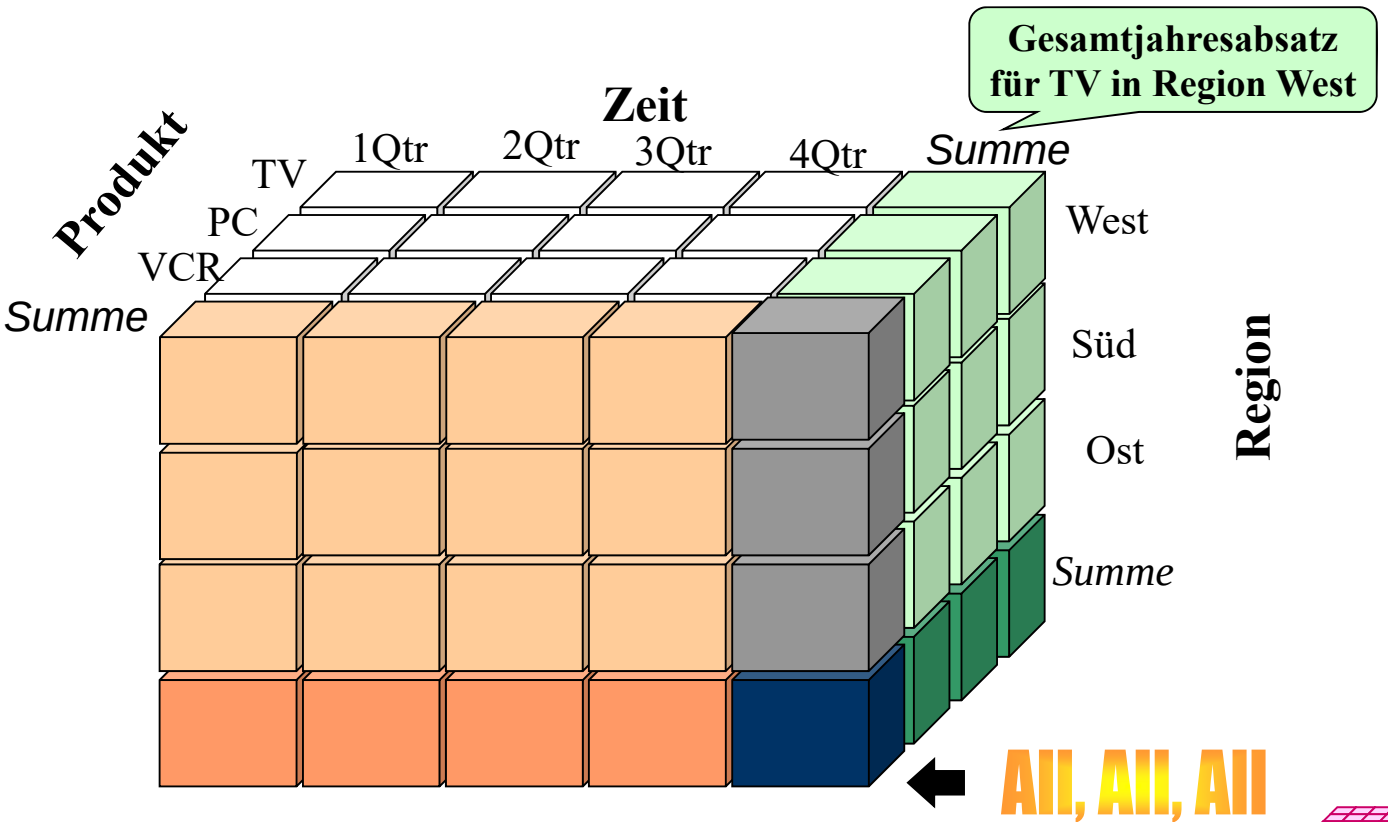


Data Cube

- Datenwürfel bzw. OLAP-Würfel (Cube), Data Cube
  - Dimensionen: Koordinaten
  - Kennzahlen: Zellen im Schnittpunkt der Koordinaten
- Cube bezüglich Dimensionen D_1, ...D_n und k Kennzahlen (Fakten):
  - W = { (d_1, ... d_n), (f_1, ... f_k), Dimensionselement d_i aus D_i, i= 1..n, Kennzahlen f_j, j = 1..k)}
  - eindeutige Zellen-Adresse: (d_1, ... d_n)
  - Zellen-Inhalt: (f_1, ... f_k)
- n: Dimensionalität des Cube
- Alternative: k Cubes mit je einer Kennzahl pro Zelle (Multi-Cube)
- typischerweise 4 - 12 Dimensionen
  - Zeitdimension fast immer dabei
  - weitere Standarddimensionen: Produkt, Kunde, Verkäufer, Region, Lieferant, ...

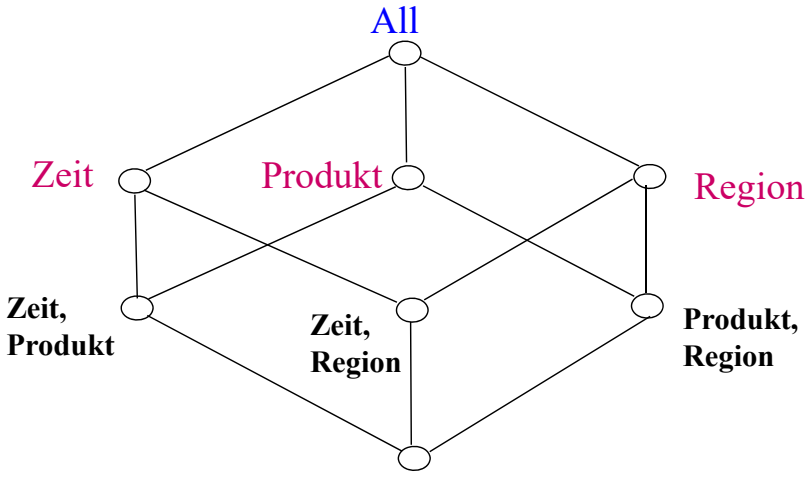


Data Cube: 3D-Beispiel mit Aggregation

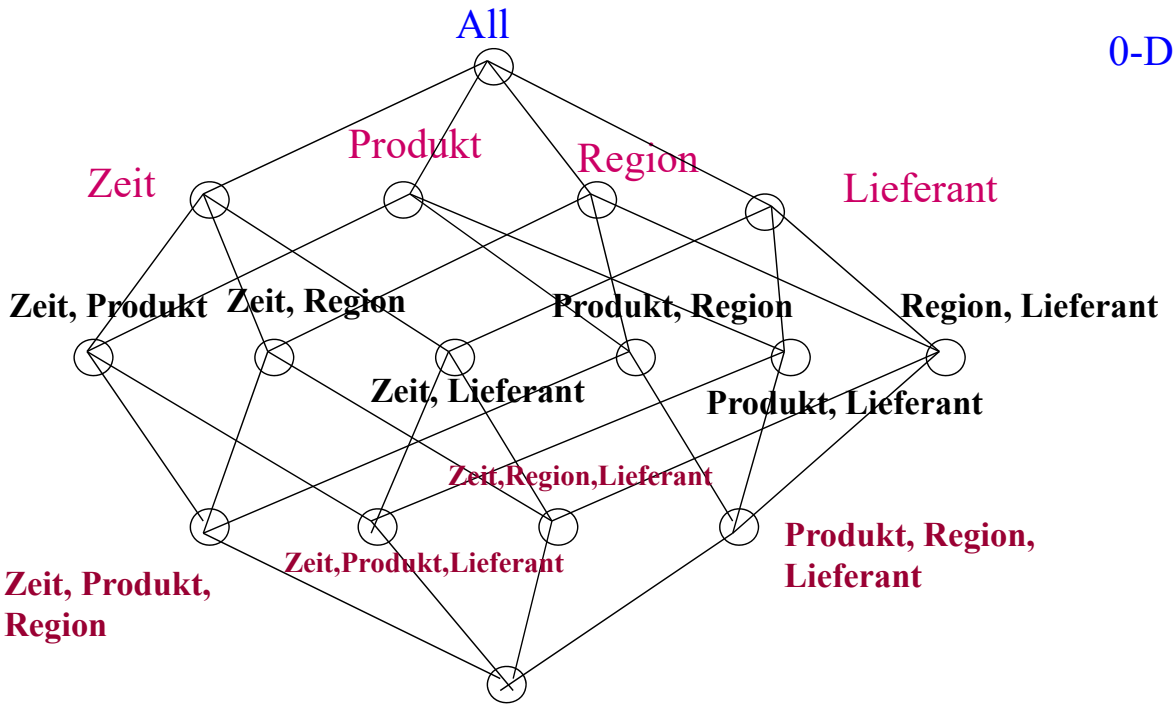


Cube mit Aggregationen

- Aggregationen von Kennzahlen für jede Dimension und Kombination von Dimensionen möglich -> *Cuboide*
  - Basis-Cuboid: N-dimensionaler Cube
  - Hieraus lassen sich Cuboiden geringerer Dimensionsanzahl ableiten -> Data Cube entspricht Verband (Lattice) von Cuboiden (**Aggregationsgitte**r)
  - N-dimensionaler Cube hat 2^N Cuboiden inkl. Basis-Cuboid (ohne Dimensionshierarchien)
  - Scheitel-Cuboid: 0-dimensionale Aggregation über alle Dimensionen



Cuboid-Verband für 4D Cube



0-D(Scheitel) Cuboid

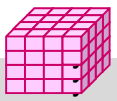
1-D Cuboide

2-D Cuboide

3-D Cuboide

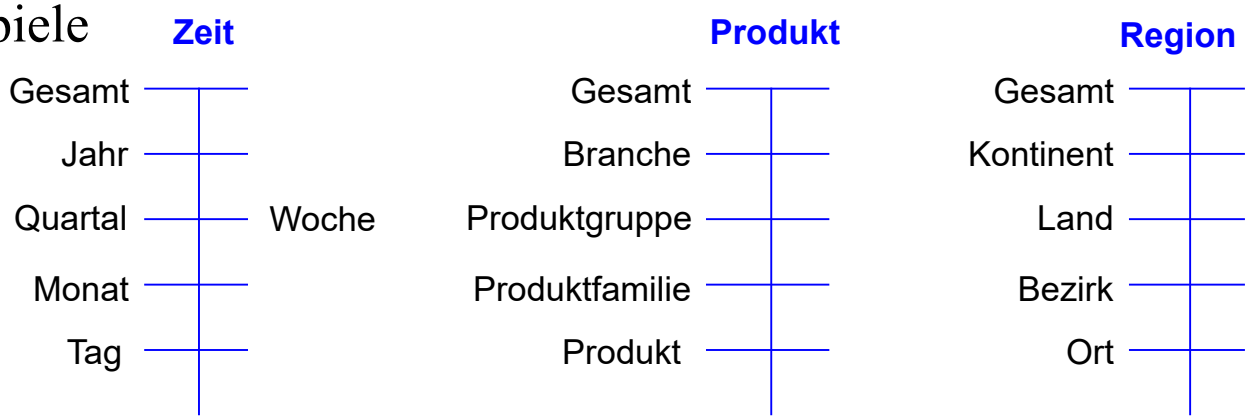
Zeit, Produkt, Region, Lieferant

4-D (Basis)-Cuboid

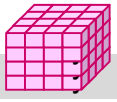


Dimensionshierarchien (Konzepthierarchien)

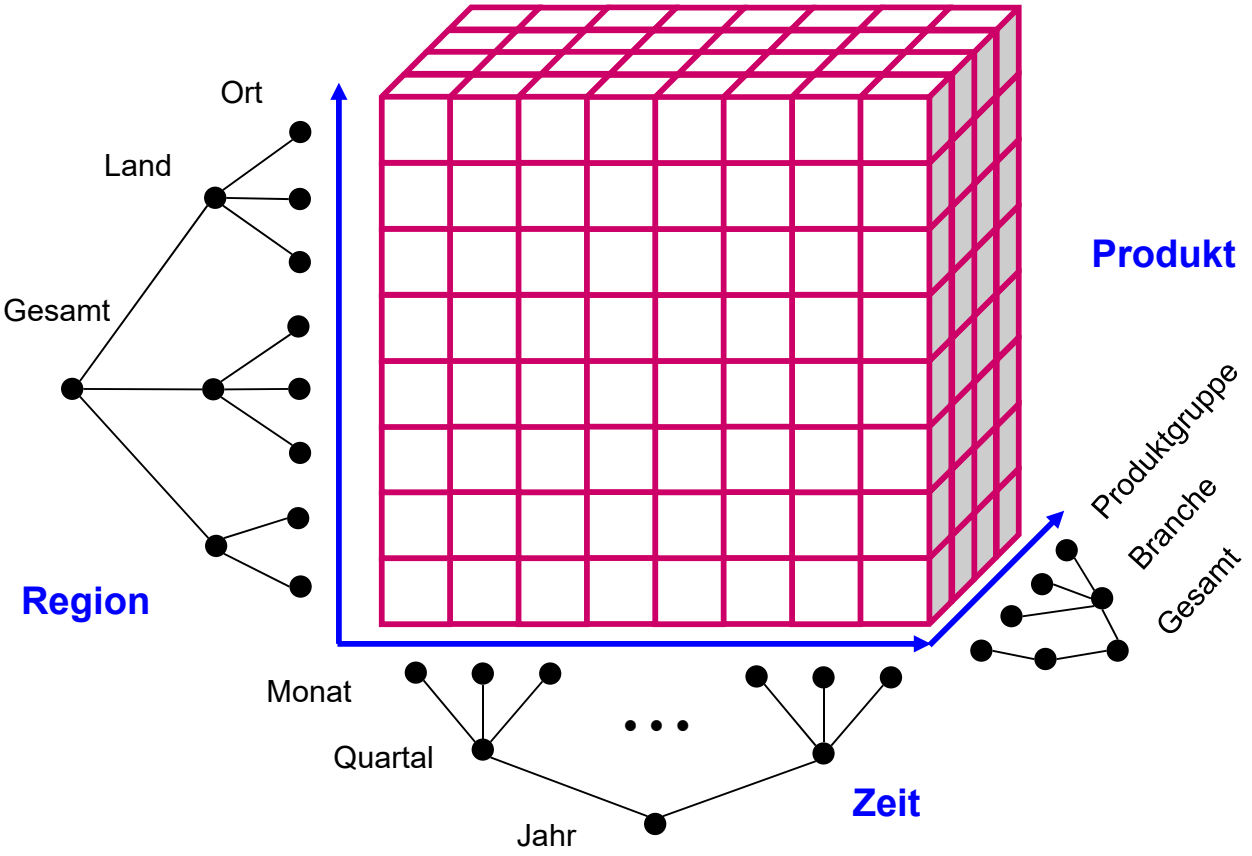
- häufig hierarchische Beziehungen zwischen Dimensionsobjekten
  - Top-Level pro Hierarchie für alle Dimensionselemente (Gesamt, Top, All)
  - *Primärattribut*: unterste (genaueste) Stufe
  - funktionale Abhängigkeiten zwischen Primärattribut und *Klassifikationsattributen* höherer Stufen
  - *einfache Hierarchie* (pro Element höchstens ein übergeordnetes Element) vs. *parallele Hierarchie* bzw. Halbordnung
- Beispiele



- Dimensionen können neben Klassifikations(Hierarchie)attributen noch beschreibende *dimensionale Attribute* aufweisen

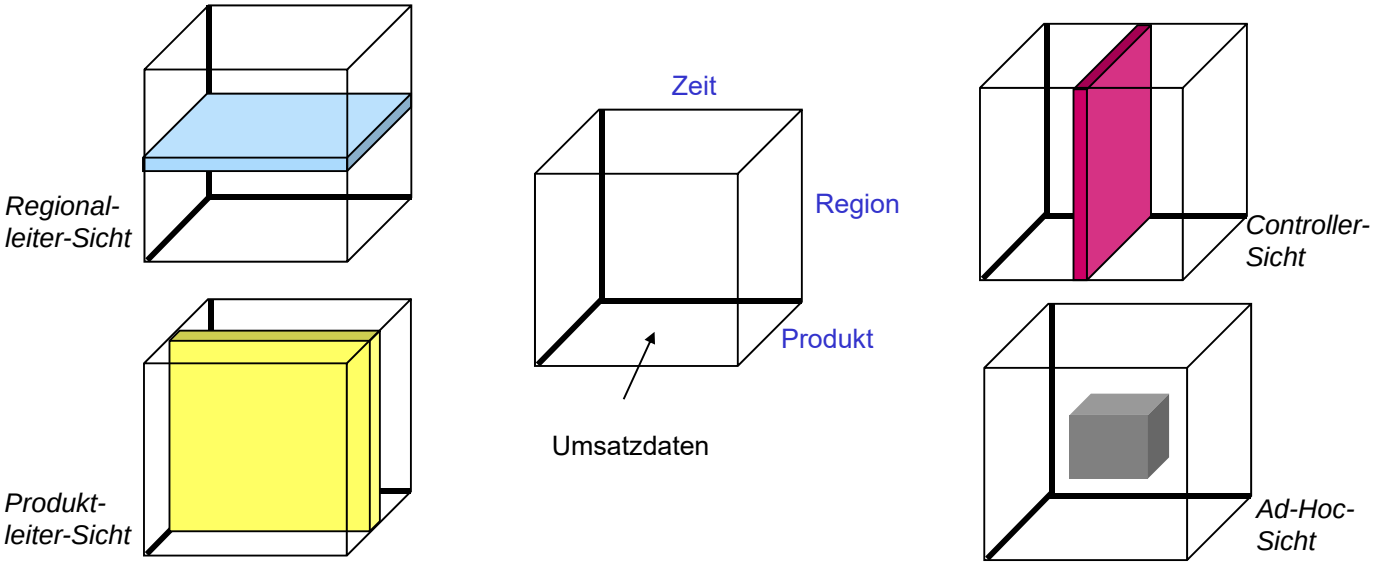


Cube mit hierarchischen Dimensionen

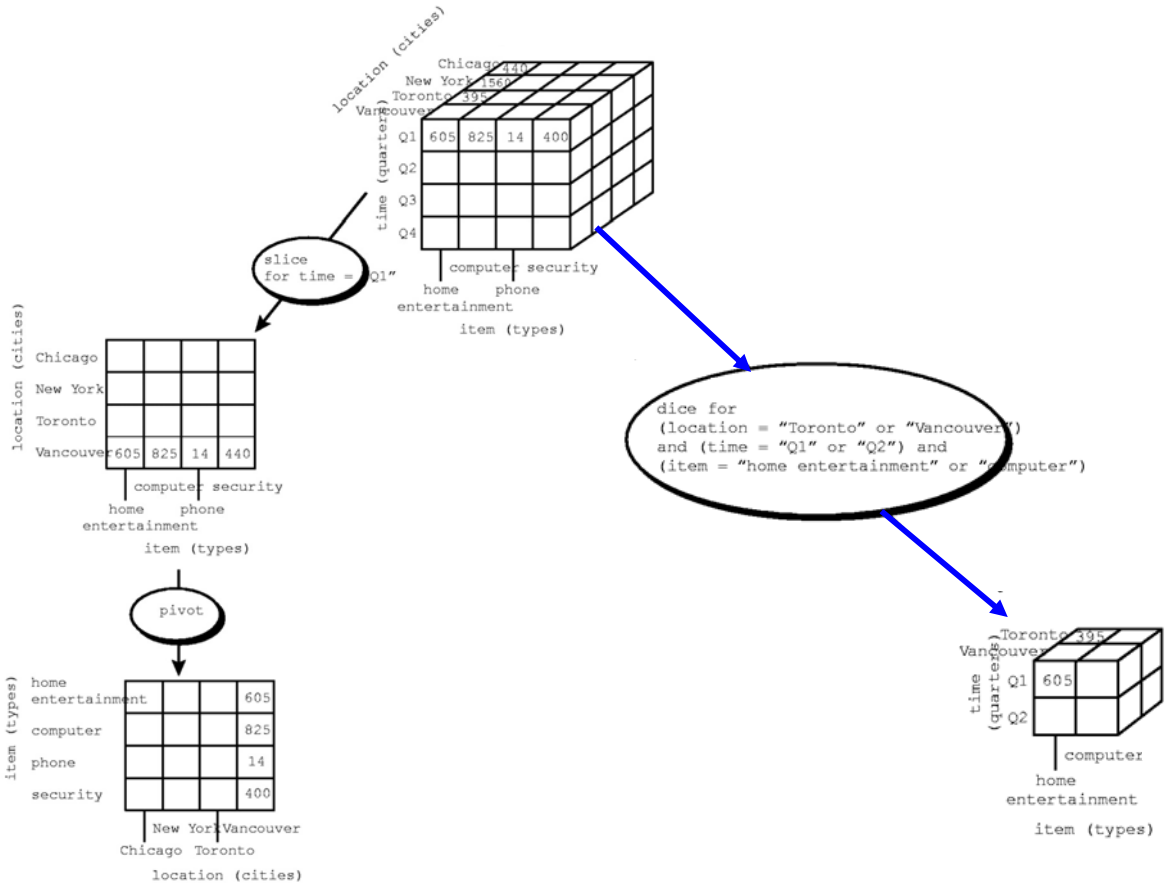


Operationen auf Cubes

- Slice: Herausschneiden von „Scheiben“ aus dem Würfel durch Einschränkung (Selektion) auf einer Dimension
  - Verringerung der Dimensionalität
- Dice: Herausschneiden einen „Teilwürfels“ durch Selektion auf mehreren Dimensionen
- unterschiedlichste mehrdimensionale Aggregationen / Gruppierungen
- Pivot (Austausch von Dimensionen), Sortierung, Top-n-Anfragen, ...

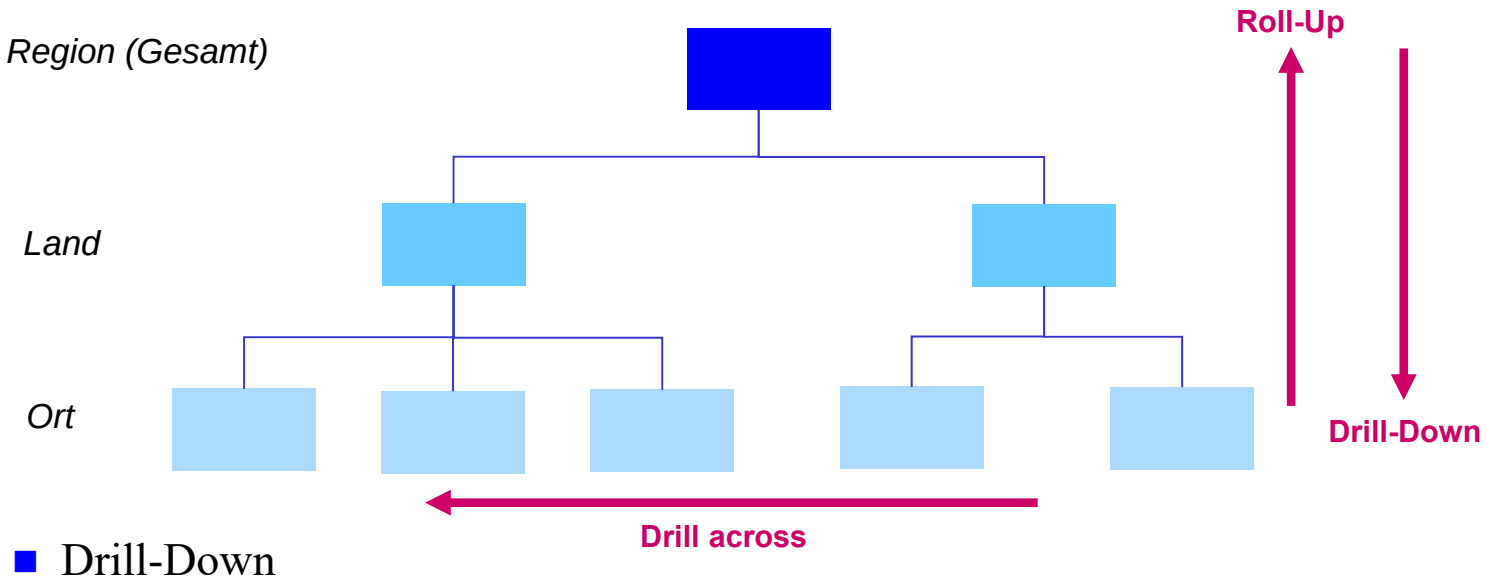


Beispiele



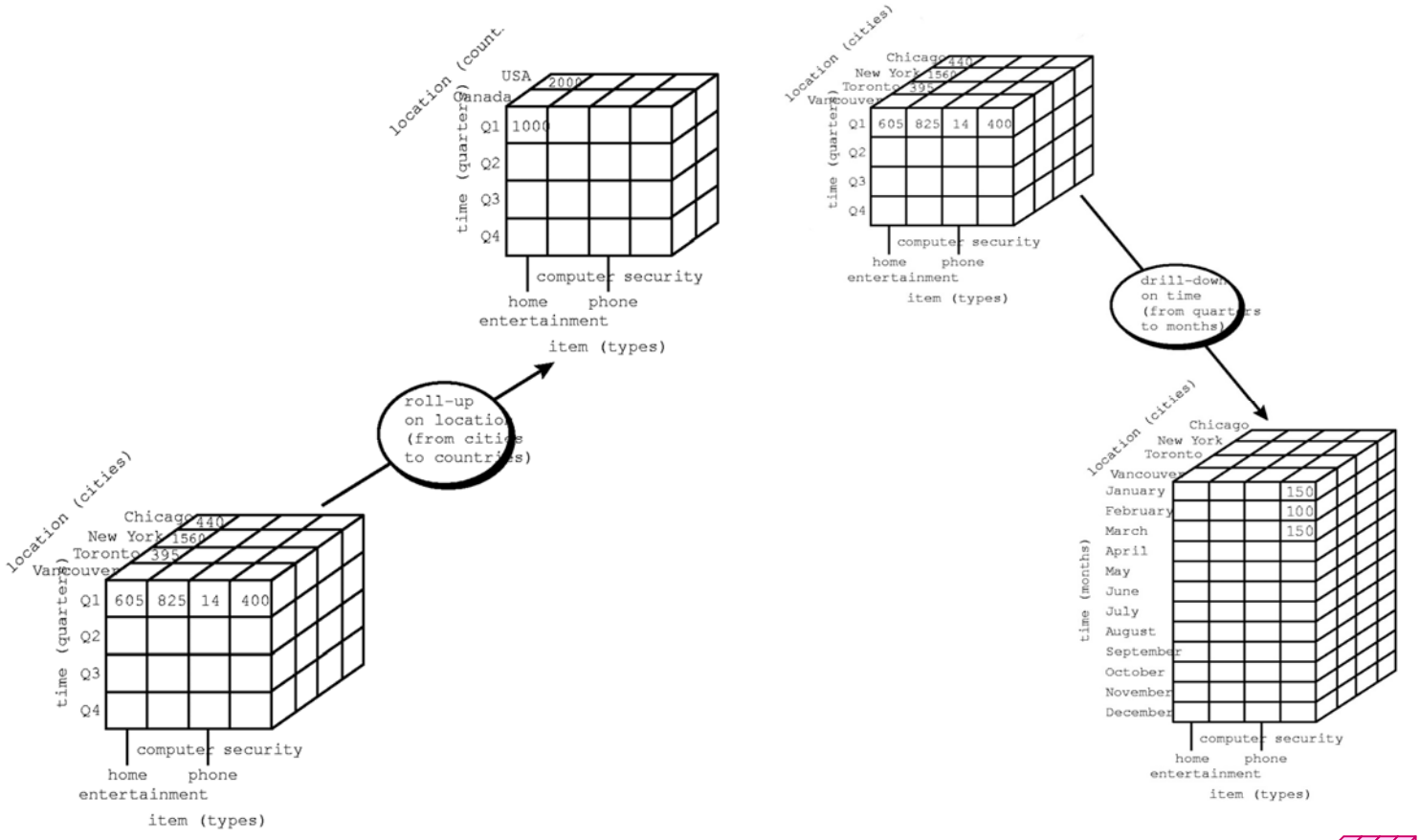
Quelle: J. Han, M. Kamber: Data Mining, Morgan Kaufmann

Navigation in Hierarchien

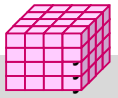


- Drill-Down
  - Navigation nach „unten“ in der Hierarchie
  - Erhöhung des Detailgrad: von verdichteten Daten zu weniger verdichteten/aggregierten Daten
- Roll-Up (Drill-Up)
  - Navigation nach „oben“ in der Hierarchie
  - von weniger verdichteten (aggregierten) Daten zu stärker verdichteten Daten

Beispiele Roll-Up, Drill-Down



Quelle: J. Han, M. Kamber: Data Mining, Morgan Kaufmann

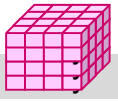


Aggregation: 2D-Beispiel

- Summenbildung

Produktgruppe	*Ost*	*Süd*	*Nord*	*West*	**Summe**
Elektronik	1800	1500	1450	2000	6750
Spielwaren	500	1700	600	1500	4300
Kleidung	1200	1200	400	1000	3800
Summe	3500	4400	2450	4500	14850

- Vorberechnung (Materialisierung) der Aggregationen zur schnellen Beantwortung von Aggregationsanfragen
- hoher Speicher- und Aktualisierungsaufwand bei vielen Dimensionen und vielen Dimensionselementen
  - i.a. kann nur kleiner Teil benötigter Aggregationen vorberechnet werden

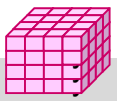


Größe der Cubes

- Größe der Basis-Cuboids
  - Anzahl der Zellen entspricht Produkt der Dimensionskardinalitäten D_i , i=1..n
  - Beispiel: 1.000 Tage, 100.000 Produkte, 1 Million Kunden -> **10^{14} Kombinationen** bei nur 3 Dimensionen
  - jede weitere Dimension, z.B. Region oder Verkäufer, führt zu einer Vervielfachung des Datenraumes
- Vorberechnung von (aggregierten) Cuboiden erhöht Speicherbedarf
- Größe eines hierarchisch aggregierten Cubes
  - für jedes Dimensionselement ist Aggregierung auf einer höheren Hierarchiestufe möglich
  - Kombinationsmöglichkeit mit jedem Element auf einer der Hierarchiestufen der anderen n-1 Dimensionen
- Anzahl Cuboiden bei n-dimensionalem Cube: $T = \prod_{i=1}^{n} (L_i + 1)$

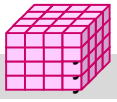
  L_i: #Ebenen von Dimension i (ohne Top-Level)

Zeit		Produkt		Kunde	
Gesamt	1	Gesamt	1	Gesamt	1
Quartal	12	Branche	50	Kundengruppe	10.000
Monat	36	Produktgruppe	5.000	Einzelkunde	1 Million
Tag	1000	Produkt	100.000		



Umsetzung des multi-dimensionalen Modells

- betrifft Speicherung der Daten und Formulierung / Ausführung der Operationen
- MOLAP: direkte Speicherung in multi-dimensionalen Speicherungsstrukturen
  - Cube-Operationen einfach formulierbar und effizient ausführbar
  - begrenzte Skalierbarkeit auf große Datenmengen da riesige Anzahl von Zellen
- ROLAP: relationale Speicherung der Daten in Tabellen
  - effiziente Speicherung sehr großer Datenmengen
  - eher umständliche Anfrageformulierung mit SQL
  - Standard-SQL nicht ausreichend (nur 1-dimensionale Gruppierung, ...)
- HOLAP: hybride Lösung
  - relationale Speicherung der Detail-Daten, multidimensionale Zugriffsschnittstelle
  - unterschiedliche Kombinationen mit multidimensionaler Speicherung / Auswertung von aggregierten Daten
- (teilweise) Vorberechnung von Aggregationen wesentlich für gute Leistung



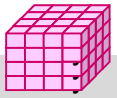
Multi-dimensionale Datenspeicherung

- Datenspeicherung in multi-dimensionaler Matrix
  - direkte Umsetzung der logischen Cube-Sicht
  - Vorab-Berechnung und Speicherung der Kennzahlen basierend auf dem Kreuzprodukt aller Wertebereiche der Dimensionen
  - schneller direkter Zugriff auf jede Kennzahl über Indexposition (x_1, x_2, ... x_n)

multi-dimensional (Kreuztabelle)

	Sachsen	Brandenburg	Thüringen
Smartphone	100	150	200
TV	50	170	150
Camcorder	20	120	100

Anfragen:
- wie hoch ist der Umsatz für TV in Thüringen
- wie hoch ist der Gesamtumsatz für Camcorder?



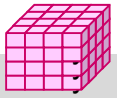
Multi-dimensionale Datenspeicherung (2)

- mehrdimensionale Speicherung führt oft zu dünn besetzten Matrizen
- Beispiel (Kundenumsätze nach Regionen)

REGION

Kunde	*B*	*S*	*NRW*	*SH*	*BW*	*SA*	*MVP*	*HH*	*TH*
Kunde 1	100	-	-	-	-	-	-	-	-
Kunde 2	-	-	150	-	-	-	-	-	-
Kunde 3	-	-	-	-	200	-	-	-	-
Kunde 4	-	50	-	-	-	-	-	-	-
Kunde 5	-	-	-	170	-	-	-	-	-
Kunde 6	-	-	-	-	-	-	-	-	100
Kunde 7	-	-	-	-	-	20	-	-	-
Kunde 8	-	-	-	-	-	-	120	-	-
Kunde 9	-	-	-	-	-	-	-	100	-

- vollständig besetzte Matrizen i.a. nur für höhere Dimensionsebenen
- Unterstützung dünn besetzter Matrizen erforderlich (Leistungseinbussen)
  - Zerlegung eines Cubes in Sub-Cubes („chunks“), die in Hauptspeicher passen
  - zweistufige Adressierung: Chunk-Id, Zelle innerhalb Chunk



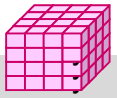
Sprachansatz MDX*

- MDX: MultiDimensional eXpressions
  - Microsoft-Spezifikation für Cube-Zugriffe / Queries
  - an SQL angelehnt
  - Extraktion von aggregierten Sub-Cubes / Cuboiden aus Cubes
- Unterstützung durch Microsoft und zahlreiche Tool-Anbieter
- Hauptanweisung

```
SELECT    [<axis_specification> [, <axis_specification>...]]
FROM      [<cube_specification>]
[WHERE [<slicer_specification>]]
```

  - Axis_specification: Auszugebende Dimensionselemente
  - 5 vordefinierte Achsen: columns, rows, pages, chapters, and sections
  - Slicer: Auswahl der darzustellenden Werte

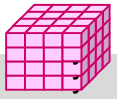
* http://msdn.microsoft.com/en-us/library/ms145506.aspx



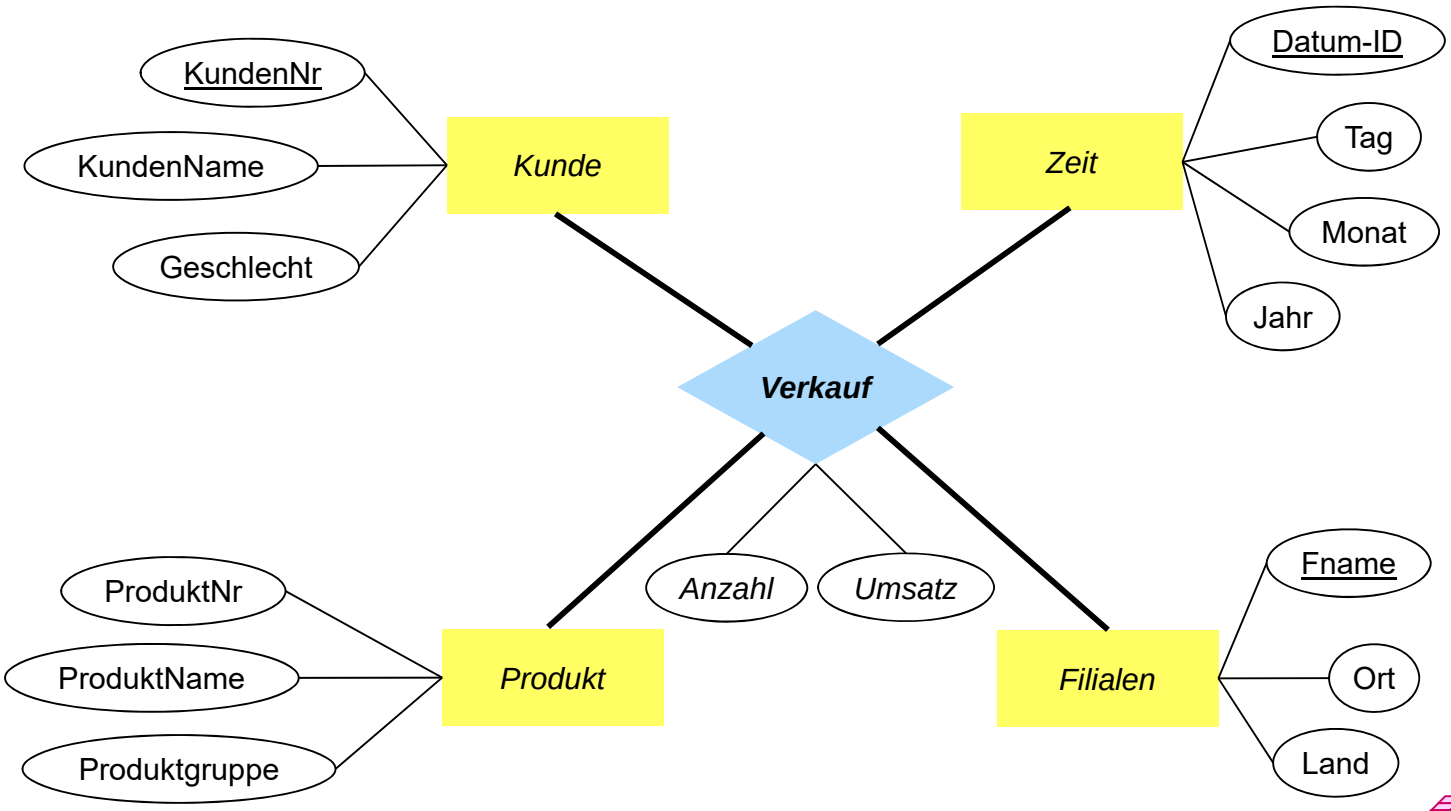
MDX: Beispiele

```
SELECT  Region.CHILDREN ON COLUMNS,
        Produkt.CHILDREN ON ROWS
FROM    Verkauf
WHERE   (Umsatz, Zeit.[2019])
```

```
SELECT  Measures.MEMBERS ON COLUMNS,
        TOPCOUNT(Filiale.Ort.MEMBERS, 10, Measures.Anzahl) ON ROWS
FROM    Verkauf
```

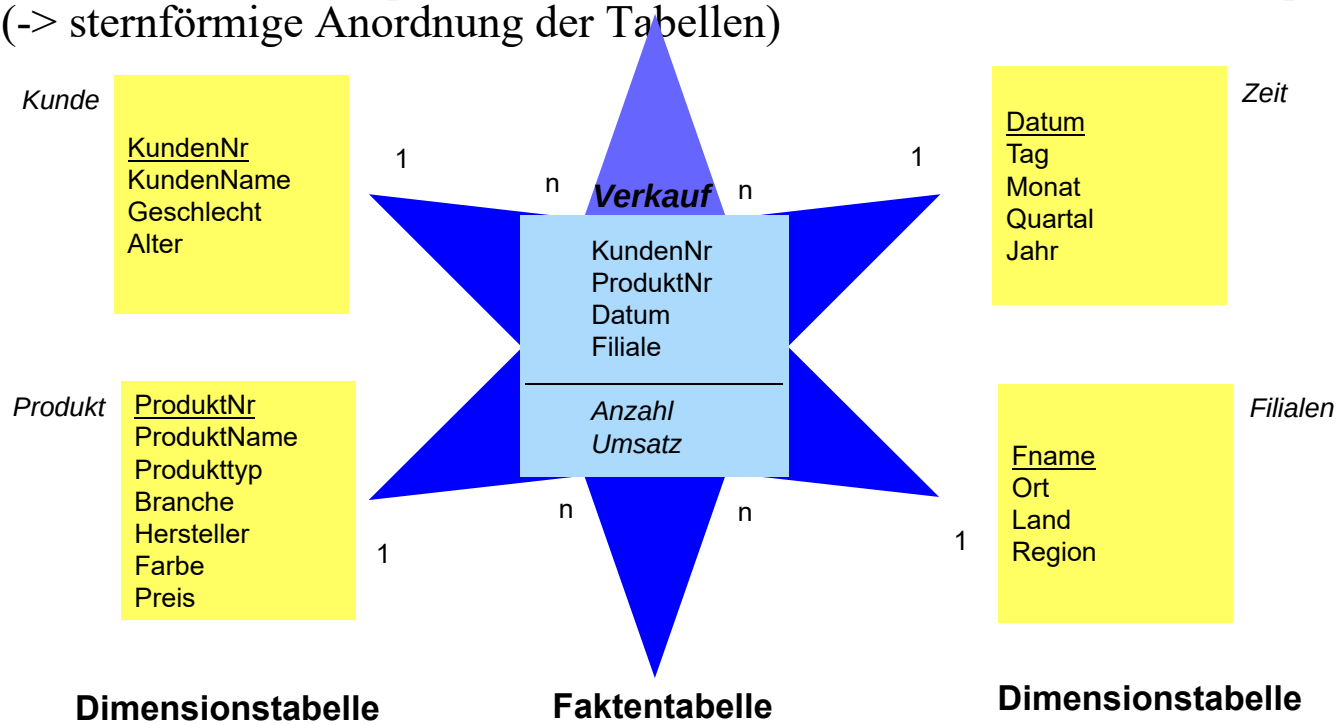


ER-Diagramm eines multi-dimensionalen Datenmodells



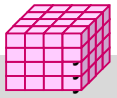
Relationale Speicherung: Star-Schema

- Faktentabelle bildet Zentrum des Star-Schemas und enthält die Detail-Daten mit den zu analysierenden Kennzahlen
- 1 Dimensionstabelle pro Dimension, die nur mit Faktentabelle verknüpft ist (-> sternförmige Anordnung der Tabellen)



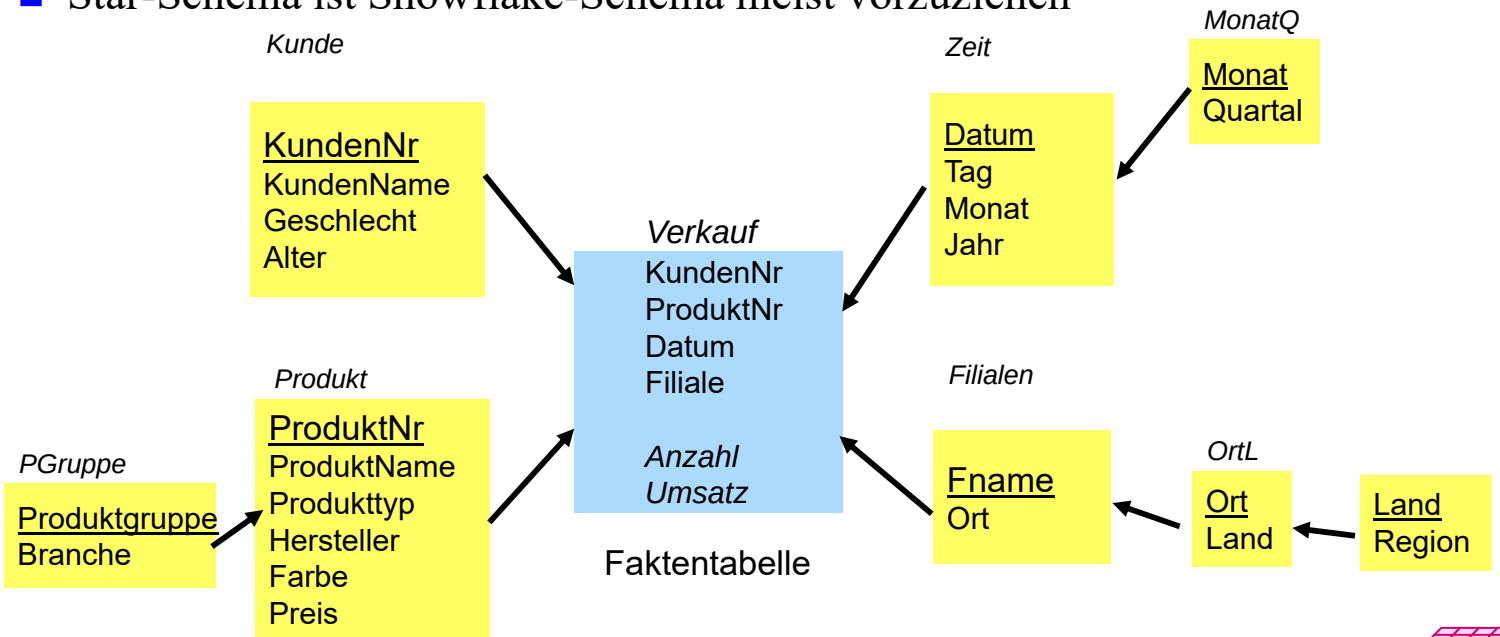
Star-Schema (2)

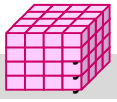
- formale Definition: Star-Schema besteht aus einer Menge von Tabellen D_1, ...D_n, F mit
  - Dimensionstabellen D_i bestehend aus (i.a. künstlichen) Primärschlüssel d_i und Dimensionsattributen
  - Faktentabelle F bestehend aus Fremdschlüsseln d_1, ... d_n sowie Meßgrößen (Kennzahlen) als weiteren Attributen
  - Dimensionstabellen sind i.a. *denormalisiert*, d.h. nicht in dritter Normalform
- Beobachtungen
  - Anzahl der Datensätze in Faktentabelle entspricht Anzahl der belegten Zellen einer multi-dimensionalen Matrix
  - leere Dimensionskombinationen unproblematisch, da nur relevante Kombinationen in der Faktentabelle auftreten.
  - dennoch oft riesige Faktentabellen
  - Dimensionstabellen vergleichsweise klein, teilweise jedoch auch umfangreich (Kunden, Artikel etc.)



Snowflake-Schema

- explizite Repräsentation der Dimensionshierarchien
- normalisierte Dimensionstabellen
  - leicht geringere Redundanz, geringerer Änderungsaufwand
  - erhöhte Zugriffskosten (höherer Join-Aufwand)
- Star-Schema ist Snowflake-Schema meist vorzuziehen

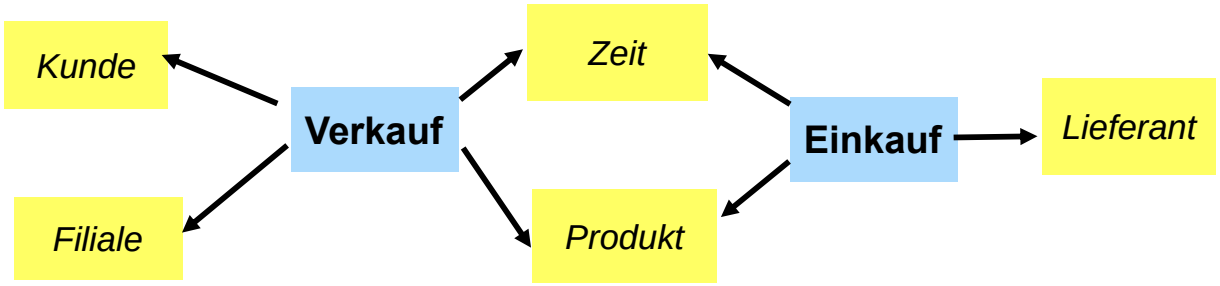




Galaxien-Schema

- Data Warehouses benötigen meist mehrere Faktentabellen

  -> Multi-Star-Schema (Galaxien-Schema, „Fact Constellation Schema“)
- gemeinsame Nutzung von Dimensionstabellen
- Speicherung vorberechneter Aggregate
  - separate Faktentabelle
  - im Rahmen der Faktentabelle mit Detail-Daten



Übungsaufgabe : Warehouse-Entwurf

Erstellen Sie ein Star-Schema zur Analyse einer global auftretenden Viruserkrankung

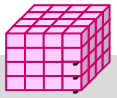
- es sollen tagesweise die nachgewiesenen Infektionen und Todesfälle zusammen mit dem Ort des Auftretens/Todes erfasst werden
- pro Infektion/Todesfall sollen anonyme Personenmerkmale wie Alter und Geschlecht erfasst werden
- Auswertungen über Zahl der Infektionen und Todesfälle sollen für unterschiedliche Zeiträume (Tag, Monate, Jahre) sowie für Länder und deren Kontinente möglich sein. Pro Land sollen auch die relativen Häufigkeiten bezogen auf je 1 Million Einwohner berechnet werden können.

Dimensionen:

Faktentabelle(n):

3. Mehrdimensionale Datenmodellierung und Operationen

- Grundlagen
  - Kennzahlen, Dimensionen, Cube
  - Cuboide / **Aggregationsgitter**
  - hierarchische Dimensionen / Cube-Operationen
- multi-dimensionale Speicherung (MOLAP)
  - MDX-Abfragen
- relationale Repräsentation mehrdimensionaler Daten (ROLAP)
  - Star-Schema
  - Varianten: Snowflake-, Galaxien-Schema
- mehrdimensionale Gruppierungen in SQL
  - Star Join
  - Group-By-Erweiterungen**:** Cube, Rollup, Grouping Sets
- Auswertung von Zeitreihen/Sequenzen mit SQL
  - RANK-, WINDOW-Queries



Anfragen auf dem Star-Schema

- Star-Join
  - sternförmiger Join der (relevanten) Dimensionstabellen mit der Faktentabelle
  - Einschränkung der Dimensionen
  - Verdichtung der Kennzahlen durch Gruppierung und Aggregation
- allgemeine Form

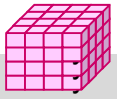
aggregierte Kennzahlen

```
select      g1, ... gk, agg(f1), ... agg (fm)
from   D1, ..., Dn, F
where  <Selektionsbedingung auf D1> and
            ... and
       <Selektionsbedingung auf Dn> and
            D1.d1 = F.d1 and
             ... and
            Dn.dn = F.dn
group by    g1, ... gk
sort by     ...;
```

Relationen des Star-Schemas

Join-Bedingungen

Ergebnis-Dimensionalität

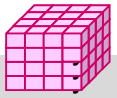


Beispiel eines Star-Join

- in welchen Jahren wurden von weiblichen Kunden in Sachsen im 1. Quartal die meisten Autos gekauft?

```
select    z.Jahr as Jahr, sum (v.Anzahl) as Gesamtzahl
from      Filialen f, Produkt p, Zeit z, Kunden k, Verkauf v
where     z.Quartal = 1 and k.Geschlecht = 'W' and
                p.Produkttyp = 'Auto' and f.Land = 'Sachsen' and
                v.Datum = z.Datum and v.ProduktNr = p.ProduktNr and
                v.Filiale = f. FName and v.KundenNr = k.KundenNr
group by  z.Jahr
order by  Gesamtzahl descending;
```

Jahr	Gesamtzahl
2018	745
2019	710
2017	650



Mehrdimensionale Aggregationen mit Group-By

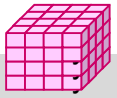
- Attributanzahl in **group by**-Klausel bestimmt Dimensionalität

```
select Hersteller, Jahr,
       sum (Anzahl) as Anzahl
from Verkauf v, Produkt p, Zeit z
where v.ProduktNr = p.ProduktNr and
v.Datum= z.Datum  and p.Produkttyp = 'Auto'
group by Hersteller, Jahr;

select Hersteller, sum (Anzahl) as Anzahl
from Verkauf v, Produkt p
where v. Produkt = p. ProduktNr and
and p. Produkttyp = 'Auto'
group by Hersteller;

select sum (Anzahl) as Anzahl
from Verkauf v, Produkt p
where v. Produkt = p. ProduktNr and
p. Produkttyp = 'Auto';
```

Hersteller	Jahr	*Anzahl*
VW	2017	*2.000*
VW	2018	*3.000*
VW	2019	*3.500*
Opel	2017	*1.000*
...	...	...
BMW	2019	*1.500*
Ford	2017	*1.000*
Ford	2018	*1.500*
Ford	2019	*2.000*

Hersteller	*Anzahl*
VW	*8.500*
Opel	*3.500*
Ford	*4.500*
BMW	*3.000*

Anzahl
19.500



Beispiel Covid-19-Daten

Datenquelle https://github.com/owid/covid-19-data/tree/master/public/data
eingebunden in LOTS https://lots.uni-leipzig.de/sql-training/ (Stand 07.04.2022)

alternativ: CSV-Import in relationale DBS (MySql, PostgresSQL …)

datum	land	faelle	tote	population	kontinent
2020-04-27	Afghanistan	68	1	39835428	Asia
2020-04-27	Albania	10	0	2872934	Europe
2020-04-27	Algeria	135	7	44616626	Africa
2020-04-27	Andorra	5	0	77354	Europe
2020-04-27	Angola	1	0	33933611	Africa
2020-04-27	Anguilla	0		15125	North_America
2020-04-27	Antigua_and_Barbuda	0	0	98728	North_America
2020-04-27	Argentina	111	5	45605823	South_America

SQL Anfrage

```
select kontinent, to_char(SUM(faelle), 'FM999,999,999')  AS faelle, to_char(SUM(todesfaelle),
'FM999,999,999') as tote
from cov natural join land
group by kontinent
order by 1
```

DB Auswahl: Covid, Go, DB Schema

land

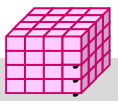
Name	Typ
landname	TEXT(2147483647) NOT NULL
kontinent	TEXT(2147483647) NOT NULL
population	INT4(10) NOT NULL

cov

Name	Typ
datum	DATE(13) NOT NULL
landname	TEXT(2147483647) NOT NULL
faelle	INT4(10)
todesfaelle	INT4(10)

zeit

Name	Typ
datum	DATE(13) NOT NULL
jahr	INT4(10) NOT NULL
monat	INT4(10) NOT NULL
tag	INT4(10) NOT NULL



Beispiel Covid-19-Daten

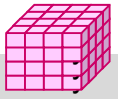
Group-by kontinent (1-dim. Aggregation)

kontinent	faelle	tote
Africa	11,564,707	251,998
Asia	142,468,173	1,407,217
Europe	184,040,834	1,783,116
North_America	94,924,083	1,493,299
Oceania	6,022,156	9,573
South_America	56,160,167	1,266,766

Datenstand: 07.04.2022

Group-by kontinent + jahr (drill-down, 2-dimens.)

kontinent	jahr	faelle	tote
Africa	2020	2,760,926	65,507
Africa	2021	6,977,423	163,031
Africa	2022	1,826,358	23,460
Asia	2020	19,898,890	338,282
Asia	2021	64,274,703	914,735
Asia	2022	58,294,580	154,200
Europe	2020	23,928,285	546,454
Europe	2021	65,198,522	985,155
Europe	2022	94,914,027	251,507
North_America	2020	23,180,679	509,168
North_America	2021	41,865,982	770,693
North_America	2022	29,877,422	213,438
Oceania	2020	48,428	1,060
Oceania	2021	540,424	3,462
Oceania	2022	5,433,304	5,051



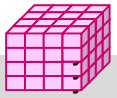
Cube-Operator

- SQL- CUBE-Operator für n-dimensionale Gruppierung und Aggregation
  - Syntax: *Group By CUBE (D_1, D_2, ... D_n)*
  - n-dimensionale Gruppierung/Aggregation + alle Gruppierungen geringerer Dimensionaltät; **2^n Aggregationen bei n Attributen** (4 bei n=2, 8 bei n=3 etc.)
  - bei niedriger Dimensionalität fasst ALL alle Werte einer Dimension zusammen
  - implementiert in MS SQL-Server, DB2, Oracle, PostgresSQL (ab V9.5), …

- Beispiele Auto-Verkauf (2D-Cube)

```
select p.Hersteller as Hersteller,
  z.Jahr as Jahr, sum(v.Anzahl)as Anzahl
from Verkauf v, Produkt p,  Zeit z
where p.Produkttyp = 'Auto' and
  v.ProduktNr = p.ProduktNr and
  v.Datum = z.Datum
group by cube (p.Hersteller, z.Jahr);
```

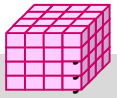
Hersteller	Jahr	*Anzahl*
VW	2017	*2.000*
VW	2018	*3.000*
VW	2019	*3.500*
Opel	2017	*1.000*
Opel	2018	*1.000*
Opel	2019	*1.500*
BMW	2017	*500*
BMW	2018	*1.000*
BMW	2019	*1.500*
Ford	2017	*1.000*
Ford	2018	*1.500*
Ford	2019	*2.000*
VW	**ALL**	*8.500*
Opel	**ALL**	*3.500*
BMW	**ALL**	*3.000*
Ford	**ALL**	*4.500*
ALL	2017	*4.500*
ALL	2018	*6.500*
ALL	2019	*8.500*
ALL	**ALL**	*19.500*



3D-Cube

```
select p. Hersteller as
  Hersteller, z. Jahr as Jahr,
  k.Geschlecht as Geschlecht,
  sum (v. Anzahl)as Anzahl
from Verkauf v, Produkt p,
     Zeit z, Kunde k
where p.Produkttyp = 'Auto' and
  v.ProduktNr = p.ProduktNr and
  v.Datum = z.Datum and
  v.KundenNr = k.KundenNr
group by cube (p.Hersteller,
  z.Jahr, k.Geschlecht);
```

Hersteller	Jahr	Geschlecht	Anzahl
VW	2017	m	1300
VW	2017	w	700
...	...	...	...
VW	2017	ALL	2.000
...	...	ALL	...
Ford	2019	ALL	2.000
VW	ALL	m	5.400
...	...	...	...
Ford	ALL	w	...
ALL	2017	m	...
...	...		
VW	ALL	ALL	8.500
...	...		
ALL	2017	ALL	...
...			
ALL	ALL	m	...
...			
ALL	ALL	ALL	19.500

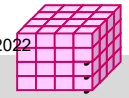


Cube-Beispiel Covid-19-Daten

- Anfrage auf den Trainingdaten im LOTS (PostgresSQL)

```
SELECT landname AS land, jahr, to_char(SUM(faelle), 'FM999,999,999')  AS faelle,
       to_char(SUM(todesfaelle), 'FM999,999,999') AS tote
FROM (cov NATURAL JOIN land NATURAL JOIN zeit)
WHERE landname='Germany' OR landname='France'
GROUP BY CUBE (land, jahr) ORDER BY land, jahr
```

land	jahr	faelle	tote
France	2020	2,735,590	65,031
France	2021	7,706,191	59,161
France	2022	16,584,807	19,281
France		27,026,588	143,473
Germany	2020	1,719,737	33,079
Germany	2021	5,430,685	78,854
Germany	2022	15,328,337	19,458
Germany		22,478,759	131,391
	2020	4,455,327	98,110
	2021	13,136,876	138,015
	2022	31,913,144	38,739
		49,505,347	274,864



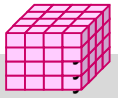
Datenstand: 07.04.2022

ROLLUP-Operator

- CUBE-Operator: inter-dimensionale Gruppierung / Aggregierung
  - generiert Aggregate für alle 2^n Kombinationsmöglichkeiten bei n Dimensionen
  - zu aufwendig für Roll-Up / Drill-Down innerhalb einer Dimension, da i.d.R. funktionale Abhängigkeiten (Land -> Kontinent, Datum -> Monat …)
- ROLLUP-Operator: intra-dimensionale Aggregierung
  - Syntax: *Group By ROLLUP (D_1, D_2, ... D_n)*
  - liefert nur die n+1 Gruppierungen

$d_1, d_2, \ldots, d_{n-1}, d_n, f()$, (Aggregationsfunktion f)
$d_1, d_2, \ldots, d_{n-1}$, ALL, $f()$,
...
d_1, ALL, ... , ALL, $f()$,
ALL, ALL, ... , ALL, $f()$

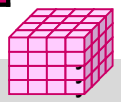
- Reihenfolge der Attribute relevant!



ROLLUP-Operator: Beispiel

```
select p.Hersteller as Hersteller,
  p. Marke as Marke, p.Farbe as
  Farbe, sum(v.Anzahl) as Anzahl
from Verkauf v, Produkt p
where v.ProduktNr= p. ProduktNr
  and p.Hersteller in(„VW“,“Opel“)
group by rollup (p.Hersteller,
                 p.Marke,
                 p.Farbe);
```

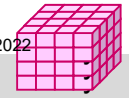
Hersteller	Marke	Farbe	Anzahl
VW	Passat	rot	800
VW	Passat	weiß	600
VW	Passat	blau	600
VW	Golf	rot	1.200
VW	Golf	weiß	800
VW	Golf	blau	1.000
VW	...	rot	1.400
...	...	...	...
Opel	Vectra	rot	400
Opel	Vectra	weiß	300
Opel	Vectra	blau	300
...	...		
VW	Passat	ALL	2.000
VW	Golf	ALL	3.000
VW	...	ALL	3.500
Opel	Vectra	ALL	1.600
Opel	...	ALL	...
VW	ALL	ALL	8.500
Opel	ALL	ALL	3.500
ALL	ALL	ALL	12.000



Rollup-Beispiel Covid-19-Daten

```
SELECT kontinent, landname AS land, SUM(faelle) AS faelle, SUM(todesfaelle) AS tote,
ROUND( ( (SUM(todesfaelle) * 1000000.0) / (SELECT SUM(population) FROM land l
    WHERE (cl.kontinent IS NULL) OR (cl.landname IS NULL AND l.kontinent = cl.kontinent)
          OR (l.landname = cl.landname) ) ), 1) AS "ToteProMillion"
FROM (cov NATURAL JOIN land) AS cl
GROUP BY ROLLUP (kontinent, land) ORDER BY kontinent DESC, land DESC
```

kontinent	land	faelle	tote	ToteProMillion
		495,180,120	6,211,969	791.5
South_America		56,160,167	1,266,766	2,919.1
South_America	Venezuela	521,186	5,697	198.5
South_America	Uruguay	892,475	7,179	2,059.9
South_America	Suriname	79,241	1,325	2,238.9
South_America	Peru	3,550,240	213,129	6,388.9
South_America	Paraguay	648,353	18,731	2,594.5
South_America	Guyana	63,320	1,226	1,551.3
South_America	Falkland_Islands	130		
South_America	Ecuador	874,871	22,823	1,275.8
South_America	Colombia	6,087,123	139,693	2,724.9
South_America	Chile	3,500,670	45,678	2,377.5
South_America	Brazil	29,990,246	661,228	3,089.9
South_America	Bolivia	903,062	21,899	1,850.7
South_America	Argentina	9,049,250	128,158	2,810.1
Oceania		6,022,156	9,573	220.4
Oceania	Wallis_and_Futuna	454	7	631.0



Datenstand: 07.04.2022

Grouping Sets

- mehrere Gruppierungen pro Anfrage

  GROUP BY GROUPING SETS (<Gruppenspezifikationsliste>)

  Gruppenspezifikation: (<Gruppenspezifikationsliste>) |
  CUBE <Gruppenspezifikationsliste> |
  ROLLUP <Gruppenspezifikationsliste>

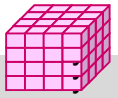
  leere Spezifikationsliste ( ) möglich: Aggregation über gesamte Tabelle

- Beispiel

```
select p.Hersteller, p.Farbe,
          sum (v.Anzahl)
from Verkauf v, Produkt p
where v.ProduktNr = p. ProduktNr and
  p.Hersteller in („VW“,“Opel“)
group by grouping sets
  ((p.Hersteller), (p.Farbe));
```

Hersteller	Farbe	*Anzahl*
VW	**ALL**	*8500*
Opel	**ALL**	*3500*
ALL	blau	*3100*
ALL	rot	*6200*
ALL	weiß	*2700*

- CUBE, ROLLUP, herkömmliches Group-By entsprechen speziellen Grouping-Sets
  - GROUP BY A,B -> GROUP BY GROUPING SETS (
  - GROUP BY ROLLUP (A,B) -> GROUP BY GROUPING SETS (
  - GROUP BY CUBE (A,B) -> GROUP BY GROUPING SETS (

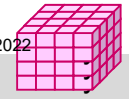


Grouping Sets: Covid-Beispiel

- Umsetzung des Cube-Beispiels mit Grouping Sets

```
SELECT landname AS land, jahr, SUM(faelle) AS faelle, SUM(todesfaelle) AS tote
FROM (cov NATURAL JOIN land NATURAL JOIN zeit)
WHERE landname='Germany' OR landname='France'
GROUP BY GROUPING SETS ((land, jahr), (land), (jahr)) ORDER BY land, jahr
```

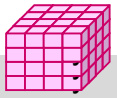
land	jahr	faelle	tote
France	2020	2,735,590	65,031
France	2021	7,706,191	59,161
France	2022	16,584,807	19,281
France		27,026,588	143,473
Germany	2020	1,719,737	33,079
Germany	2021	5,430,685	78,854
Germany	2022	15,328,337	19,458
Germany		22,478,759	131,391
	2020	4,455,327	98,110
	2021	13,136,876	138,015
	2022	31,913,144	38,739
		49,505,347	274,864



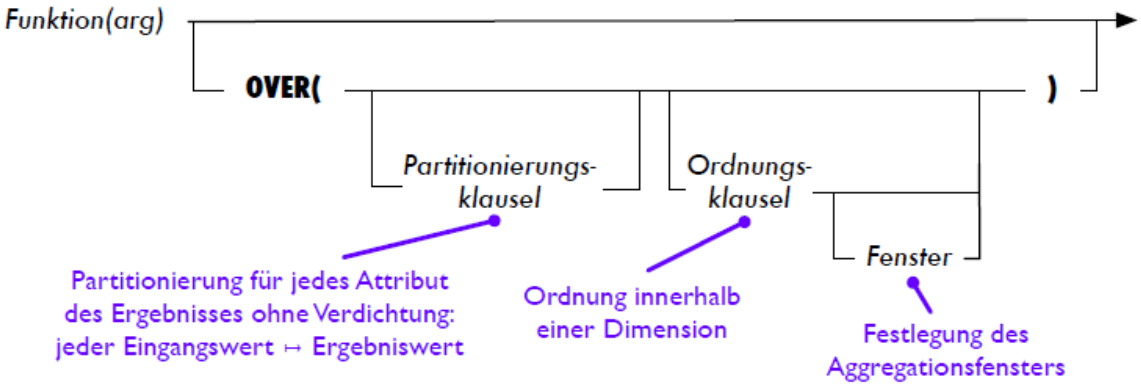
Datenstand: 07.04.2022

Zeitreihen

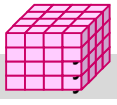
- viele Data-Warehouse-Fakten entsprechen Zeitreihen
  - Sequenz von Sätzen mit Zeit/Datumsangabe
  - Bsp.: Produktverkäufe, Erkrankungsfälle, Temperaturmesswerte, etc.
- Bedarf für Auswertungen zu bestimmten Zeitbereichen etc.
- SQL kann Auswertung auf Fenster/Ausschnitte von Satzsequenzen ausführen
  - **OVER**-Prädikat in Select-Klausel oder **WINDOW**-Klausel
- unterstützt erweiterte Analysemöglichkeiten auf Sequenzen
  - Ranking: Berechnung von Rangfolgen / Top-N
  - kumulierte Häufigkeiten/Anteile (z.B. bezüglich eines Jahres/Monats)
  - fortgeschrittene Vergleiche, z.B.
    - Tagesinfektionen gegenüber gleitenden Wochendurchschnitt,
    - Monatsumsatz gegenüber gleitendem 3-Monatsdurchschnitt …
  - Unterstützung statistischer Funktionen wie Varianz (Funktionen `VAR_POP, VAR_SAMP)`, Standardabweichung (`STDEV_POP, …)` etc.



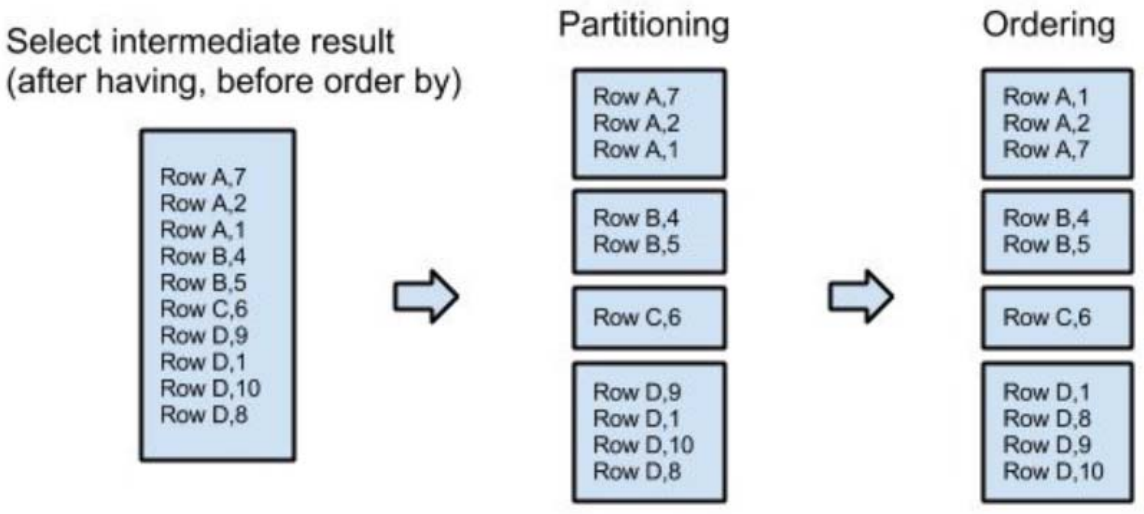
OVER-Prädikat



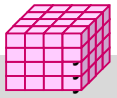
- Funktionen: `SUM, AVG,` **`RANK, DENSE_RANK,`** …
  - Berechnung bezüglich durch OVER spezifiziertes Intervall
- PARTITION BY: Zerlegung in mehrere Datensequenzen/ströme (optional)
- ORDER BY: Sortierreigenfolge pro Datensequenz (optional)
  - optionale Fensterangabe bei ORDER BY:
    tupelweise (ROWS) oder wertebereichsweise (RANGE) Einschränkung für Aggregationsfenster



Window-Verarbeitung



https://blog.matters.tech/sql-window-functions-basics-e9a9fa17ce7e



Rank-Funktion

Tabelle AVerkauf (Hersteller, Jahr, Anzahl)

```
select Hersteller, Anzahl,
  rank() over (order by Anzahl desc)
                    as Rang,
  dense_rank() over (order by Anzahl desc)
                    as DRang
from AVerkauf
where  Jahr=2017
order by Anzahl desc, Hersteller
```

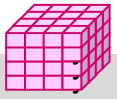
alternative Formulierung mit WINDOW-Klausel

```
select Hersteller, Anzahl,
   rank() over w as Rang,
   dense_rank() over w as DRang
from AVerkauf
where  Jahr=2017
order by Anzahl desc, Hersteller
window w as (order by Anzahl desc)
```

Hersteller	Anzahl	Rang	DRang
VW	2000	1	1
Ford	1000	2	2
Opel	1000	2	2
BMW	500	4	3

DENSE_RANK überspringt keine Nummer

44

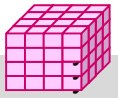


Rank-Beispiel (Covid-Datenbank)

```
SELECT landname AS land, SUM(COALESCE(faelle, 0)) AS faelle,
SUM(COALESCE(todesfaelle, 0)) AS tote,
RANK() OVER w1 AS "RankCases", RANK() OVER w2 AS "RankDeaths"
FROM (cov NATURAL JOIN land)
WHERE NOT (faelle IS NULL AND todesfaelle IS NULL) GROUP BY land
WINDOW w1 AS (ORDER BY SUM(COALESCE(faelle, 0)) DESC),
       w2 AS (ORDER BY SUM(COALESCE(todesfaelle, 0)) DESC) LIMIT 16
```

land	faelle	tote	RankCases	RankDeaths
United_States	80,289,236	1,065,464	1	1
India	43,033,067	514,200	2	3
Brazil	29,990,246	661,228	3	2
France	27,026,588	143,473	4	10
Germany	22,478,759	131,391	5	13
United_Kingdom	20,737,905	165,501	6	7
Russia	17,693,468	363,455	7	4
Italy	15,106,214	160,433	8	8
South_Korea	14,983,693	18,754	9	46
Turkey	14,116,034	98,311	10	19
Spain	11,636,410	103,456	11	17
Vietnam	10,070,692	42,918	12	24
Argentina	9,049,250	128,158	13	14
Netherlands	8,054,934	22,197	14	39
Iran	7,183,808	140,492	15	11
Japan	6,885,874	28,531	16	32

Datenstand: 07.04.2022



Rank-Funktion (2)

- partitionsweises Ranking

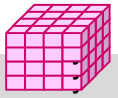
```
select Hersteller, Jahr, Anzahl,  rank() over
       (partition by Jahr order by Anzahl desc) as Rang,
from AVerkauf
order by Jahr, Rang
```

- COVID-Beispiel: Ranking eines Landes pro Kontinent

```
SELECT landname AS land, kontinent, SUM(faelle) AS faelle,
RANK() OVER (PARTITION BY kontinent ORDER BY SUM(faelle) DESC) AS "ContRank"
FROM (cov NATURAL JOIN land) WHERE faelle IS NOT NULL
GROUP BY land, kontinent ORDER BY faelle DESC
```

land	kontinent	faelle	ContRank
United_States	North_America	80,289,236	1
India	Asia	43,033,067	1
Brazil	South_America	29,990,246	1
France	Europe	27,026,588	1
Germany	Europe	22,478,759	2
United_Kingdom	Europe	20,737,905	3
Russia	Europe	17,693,468	4
Italy	Europe	15,106,214	5
South_Korea	Asia	14,983,693	2
Turkey	Asia	14,116,034	3
Spain	Europe	11,636,410	6
Vietnam	Asia	10,070,692	4
Argentina	South_America	9,049,250	2
Netherlands	Europe	8,054,934	7

Datenstand: 07.04.2022



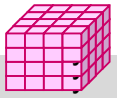
Partitionsweises Ranking: Covid-Beispiel

Top-3 pro Kontinent

```
WITH CRank AS (SELECT landname AS land, kontinent, SUM(faelle) AS faelle,
RANK() OVER w1 AS "GlobalRank",
RANK() OVER (PARTITION BY kontinent ORDER BY SUM(faelle) DESC) AS "ContRank"
FROM (cov NATURAL JOIN land) WHERE faelle IS NOT NULL GROUP BY land, kontinent
WINDOW w1 AS (ORDER BY SUM(faelle) DESC) )
SELECT * FROM CRank WHERE "ContRank" <=3 ORDER BY "GlobalRank"
```

land	kontinent	faelle	GlobalRank	ContRank
United_States	North_America	80,289,236	1	1
India	Asia	43,033,067	2	1
Brazil	South_America	29,990,246	3	1
France	Europe	27,026,588	4	1
Germany	Europe	22,478,759	5	2
United_Kingdom	Europe	20,737,905	6	3
South_Korea	Asia	14,983,693	9	2
Turkey	Asia	14,116,034	10	3
Argentina	South_America	9,049,250	13	2
Colombia	South_America	6,087,123	17	3
Mexico	North_America	5,715,504	20	2
Australia	Oceania	5,000,730	22	1
South_Africa	Africa	3,710,103	29	1
Canada	North_America	3,533,901	33	3
Morocco	Africa	1,163,806	52	2
Tunisia	Africa	1,037,358	59	3
New_Zealand	Oceania	741,992	75	2
French_Polynesia	Oceania	72,482	132	3

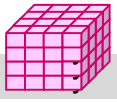
Datenstand: 07.04.2022



Weitere Ranking-Funktionen

- **row_number():** Position des Satzes in Sequenz
- **percent_rank()**
  - relativer Anteil pro Partition (zwischen 0 und 1)
- **percentile_cont(p), percentile_disc(p)**
  - Perzentile (Prozentränge) für kontinuierliche bzw. gleichmäßige Verteilung der Attributwerte innerhalb einer Gruppe (WITHIN GROUP- und ORDER BY-Klauseln)
- Beispiel: Median der Verkaufspreise pro Hersteller

```
select Hersteller, percentile_disc(0.5) within
                        group (order by Verkaufspreis)
          over (partition by Hersteller)  as Preismedian,
from verkauf natural join produkt
```

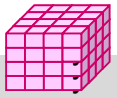


Median-Beispiel

```
SELECT jahr, MIN(faelle) AS minFaelle, MAX(faelle) AS maxfaelle,
       COUNT(*) AS Tage,  AVG(faelle) AS durchschnitt,
       PERCENTILE_DISC(0.5) WITHIN GROUP (ORDER BY faelle) AS Fmedian
FROM (cov NATURAL JOIN land NATURAL JOIN zeit)
WHERE  faelle IS NOT NULL and landname='Germany‘
GROUP BY jahr
```

jahr	minfaelle	maxfaelle	tage	durchschnitt	fmedian
2020	0	33,777	340	5,058	1,251
2021	212	76,414	365	14,879	10,303
2022	0	565,139	96	159,670	156,799

Datenstand: 07.04.2022



Aggregatberechnung auf Windows

- Nutzung von SUM, AVG etc. in Verbindung mit OVER
- Anwendungsbeispiel für Tabelle *sales (sdate, value)*

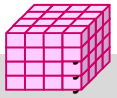
Verkäufe pro Tag sowie Anteil an Gesamtsumme

```
select date, value, all_sum,100.0*value/all_sum as anteil
from (select sdate as date, value,
      sum(value) over () as all_sum from sales) as sales2
order by 1
```

sales

sdate	value
1.2.2019	500
1.2.2019	300
5.7.2019	200
2.3.2020	400
3.4.2020	100
9.6.2021	500

date	value	all_sum	anteil
1.2.2019	500	2000	25
1.2.2019	300	2000	15
5.7.2019	200	2000	10
2.3.2020	400	2000	20
3.4.2020	100	2000	5
9.6.2021	500	2000	25



Aggregatberechnung auf Windows mit Groupin

- Group-By resultiert in Datenstrom mit 1 Satz pro Gruppe
- Bestimmung der Gesamtsumme über alle Sätze erfordert Summe über Gruppensummen -> sum(sum(…)

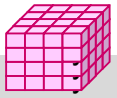
Summe der Verkäufe eines Tages im Verhältnis zu Verkäufen des Jahres

```
select sdate as date,day_sum, year_sum,
       100.0*day_sum/year_sum as janteil
from (select sdate, sum(value) as day_sum,
  sum(sum(value)) over (partition by year(sdate)) as year_sum
      from sales  group by sdate) as sales2 order by 1
```

sales

sdate	value
1.2.2019	500
1.2.2019	300
5.7.2019	200
2.3.2020	400
3.4.2020	100
9.6.2021	500

date	day_sum	year_sum	janteil
1.2.2019	800	1000	80
5.7.2019	200	1000	20
2.3.2020	400	500	80
3.4.2020	100	500	20
9.6.2021	500	500	100



Bsp.: Kumulierte Summe

- Aggregatfunktion vor OVER aggregiert bei Order By vom ersten bis zum aktuellen Tupel
- nutzbar zur Berechnung einer kumulierten Summe

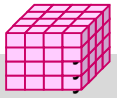
Summe der Verkäufe pro Tag sowie die kumulierten Gesamtverkäufe nach Tagen und die kumulierten Verkäufe im jeweiligen Jahr nach Tagen sortiert

```
select sdate as date, sum(value) AS day_sum,
  sum(sum(value))over(order by sdate) as cum_sum,
  sum(sum(value))over(partition by year(sdate) order by sdate)
            as cumy_sum
from sales
group by sdate
order by sdate
```

sdate	value
1.2.2019	500
1.2.2019	300
5.7.2019	200
2.3.2020	400
3.4.2020	100
9.6.2021	500

sales

date	day_sum	cum_sum	cumy_sum
1.2.2019	800	800	800
5.7.2019	200	1000	1000
2.3.2020	400	1400	400
3.4.2020	100	1500	500
9.6.2021	500	2000	500



Covid-Beispiel für kumulierte Summen

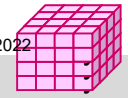
Tagesfälle sowie kumulierte Summen insgesamt und pro Monat

```
SELECT tag, monat, jahr, faelle, SUM(faelle) OVER w1 AS cum_sum,
SUM(faelle) OVER w2 AS cum_msum
FROM (cov NATURAL JOIN land NATURAL JOIN zeit)
WHERE landname='Germany' and faelle<>0
WINDOW w1 AS (ORDER BY jahr, monat, tag),
       w2 AS (PARTITION BY jahr, monat ORDER BY jahr, monat, tag)
```

tag	monat	jahr	faelle	cum_sum	cum_msum
27	1	2020	1	1	1
28	1	2020	3	4	4
31	1	2020	1	5	5
1	2	2020	3	8	3
2	2	2020	2	10	5
3	2	2020	2	12	7
7	2	2020	1	13	8
9	2	2020	1	14	9
11	2	2020	2	16	11
26	2	2020	5	21	16
27	2	2020	5	26	21
28	2	2020	27	53	48
29	2	2020	13	66	61
1	3	2020	51	117	51
2	3	2020	33	150	84
3	3	2020	38	188	122
4	3	2020	52	240	174
5	3	2020	109	349	283
6	3	2020	185	534	468
7	3	2020	150	684	618

tag	monat	jahr	faelle	cum_sum	cum_msum
8	3	2020	163	847	781
9	3	2020	265	1,112	1,046
10	3	2020	184	1,296	1,230
11	3	2020	271	1,567	1,501
12	3	2020	802	2,369	2,303
13	3	2020	693	3,062	2,996
14	3	2020	733	3,795	3,729
15	3	2020	1,043	4,838	4,772
16	3	2020	1,174	6,012	5,946
17	3	2020	1,144	7,156	7,090
18	3	2020	1,042	8,198	8,132
19	3	2020	2,801	10,999	10,933
20	3	2020	2,958	13,957	13,891
21	3	2020	2,705	16,662	16,596
22	3	2020	1,948	18,610	18,544
23	3	2020	4,062	22,672	22,606
24	3	2020	4,764	27,436	27,370
25	3	2020	4,118	31,554	31,488
26	3	2020	4,954	36,508	36,442
27	3	2020	5,780	42,288	42,222

Datenstand: 07.04.2022



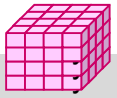
Windowing mit expliziter Fensterangabe

- Beispiel Moving Average für Tabelle *sales (date, value)*

  Berechne pro Datum durchschnittlichen Umsatz für diesen Tag, den vorhergehenden Tag sowie den nächsten Tag

```
select sdate, avg(value) over
        (order by sdate rows between 1 preceding and 1 following)
 from sales
```

- weitere dynamische Windows-Spezifikationen
  - **rows unbounded preceding** (alle Vorgänger inkl. aktuellem Tupel)
  - **rows unbounded following** (alle Nachfolger inkl. aktuellem Tupel)
  - **rows between 2 preceding and 2 following**
    (Fenster von 5 Sätzen, z.B. 5 Tage, Monate etc.)
  - **rows 3 following** (aktueller Satz und maximal 3 nachfolgende Sätze)
  - **range between interval '10' day preceding and current row**
    (wertebasierter Bereich)



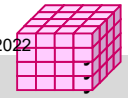
Beispiel Moving Average (Covid)

Infektionen pro Tag vs. Tagesdurchschnitt sowie Inzidenz (pro 100000) der letzten Woche

```
SELECT tag, monat, jahr, COALESCE(faelle, 0) AS faelle,
ROUND(AVG(COALESCE(faelle, 0)) OVER w1, 0) AS "avgCasesWeek",
ROUND(100000 * SUM(COALESCE(faelle, 0)) OVER w1 / population, 0) AS "WeekIncidence"
FROM (cov NATURAL JOIN land NATURAL JOIN zeit) WHERE landname='Germany'
WINDOW w1 AS (ORDER BY jahr, monat, tag ROWS BETWEEN 6 PRECEDING AND 0 FOLLOWING)
```

tag	monat	jahr	faelle	avgCasesWeek	WeekIncidence
19	3	2022	260,239	221,926	1,851
20	3	2022	131,792	219,809	1,833
21	3	2022	92,314	219,800	1,833
22	3	2022	222,080	223,113	1,861
23	3	2022	283,732	226,133	1,886
24	3	2022	318,387	229,484	1,914
25	3	2022	296,498	229,292	1,913
26	3	2022	252,026	228,118	1,903
27	3	2022	111,224	225,180	1,878
28	3	2022	67,501	221,635	1,849
29	3	2022	237,352	223,817	1,867
30	3	2022	268,477	221,638	1,849
31	3	2022	565,139	256,888	2,143
1	4	2022	0	214,531	1,789
2	4	2022	196,456	206,593	1,723
3	4	2022	115,182	207,158	1,728
4	4	2022	0	197,515	1,647
5	4	2022	180,397	189,379	1,580
6	4	2022	416,714	210,555	1,756
7	4	2022	175,263	154,859	1,292

Datenstand: 07.04.2022



Explizites Windowing (2)

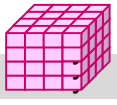
- partitionsweises Windowing

  Tabelle *transaction (account-number, date-time, amount)*

  Amount ist positiv für Zubuchung, negativ für Abbuchung

Bestimme Kontostand (kumulierte Summe) pro Konto nach jeder Kontobewegung

```
select account-number, date-time,
       sum (amount) over
        (partition by account-number   order by date-time
          rows unbounded preceding ) as balance
from transaction
order by account-number, date-time
```



Zusammenfassung

- Einfachheit des mehrdimensionalen Modellierungsansatzes wesentlich für Erfolg von Data Warehousing
  - Cube-Repräsentation mit Kennzahlen und hierarchischen Dimensionen
  - Operationen: Slice and Dice, Roll-Up, Drill-Down, ...
- multidimensionale Speicherung
  - primär für aggregierte Daten relevant, weniger zur Verwaltung von Detail-Fakten
- relationale Speicherung auf Basis von Star-Schemas
  - Unterstützung großer Datenmengen, Skalierbarkeit
  - neue Anforderungen bezüglich effizienter Verarbeitung von Star-Joins, mehrdimensionale Gruppierung und Aggregation ...
- Vorberechnung aggregierter Daten wesentlich für ausreichende Leistung
- Sprachansätze: MDX für Cubes bzw. SQL-Erweiterungen
  - Mehrdimensionale Gruppierung/Aggregation: CUBE-, ROLLUP, GROUPING SETS
  - Windowing-Funktionen für Zeitreihen / Datenströme