# 5. Fortgeschrittene SQL-Themen

- Rekursive Anfragen
  - WITH-Anweisung
  - RECURSIVE UNION
- Fortgeschrittene Datenanalysen
  - Star-Joins
  - mehrdimensionale Gruppierungen (ROLLUP, CUBE, Grouping Sets)
  - Zeitreihenauswertungen (Ranking, Windowing)
- Temporale Datenbanken
  - anwendungsversionierte Tabellen und Zeitprädikate
  - systemversionierte Tabellen
  - bitemporale Tabellen




# With-Anweisung

- Vergabe von Namen für Anfrageausdruck (benannte Anfrage)
  - entspricht temporärem View (für eine Anfrage)
  - u.a. nützlich bei mehrfacher Referenzierung von Teilanfragen (gemeinsamer Tabellenausdruck) oder Vorberechnung komplexer Teilergebnisse
- Spezifikation: WITH <R1> [(<Attributliste>) ] AS ( <Anfrageausdruck>) [,
  <R2> [(<Attributliste>) ] AS ( <Anfrageausdruck>), …]
  SELECT … *(Bezug auf R1,R2 und andere Tabellen)*
- **Beispiel** für Tabelle Verkauf (Kunde, Produkt, Ort, Anzahl, Betrag)

```
WITH Umsatz_O AS (SELECT Ort, SUM(Betrag) AS Umsatz FROM Verkauf
                  GROUP BY Ort),
     top_Orte  AS (SELECT Ort FROM Umsatz_O
                   WHERE Umsatz > (SELECT AVG(Umsatz)*3 FROM Umsatz_O)
SELECT Ort, Produkt, SUM(Anzahl) AS Anzahl, SUM(Betrag) AS Umsatz
FROM Verkauf
WHERE Ort IN (SELECT Ort FROM top_Orte)
GROUP BY Ort, Produkt;
```

# Rekursion

- Berechnung rekursiver Anfragen (z. B. transitive Hülle) über rekursiv definierte Sichten (Tabellen)
- Anwendungsfälle
  - Stücklistenauflösung
  - Routenberechnung aus Einzelverbindungen
  - Auswertung über Vorgesetztenverhältnisse, Vorfahrenbeziehungen etc.
- Grundgerüst seit SQL:1999

```
WITH RECURSIVE RekursiveTabelle (...) AS
( SELECT ... FROM Tabelle WHERE ...
  UNION
  SELECT ... From Tabelle, RekursiveTabelle     WHERE ...)

SELECT ... From RekursiveTabelle WHERE ...
```




# Rekursion: Beispiel

*Regeln:* *Vorfahr (K,V)* <- *Elternteil (K,V)*
*Vorfahr (K,V)* <- *Vorfahr (K,X), Elternteil (X,V)*

```
CREATE TABLE Eltern(Kind   CHAR (20),
                    Elternteil  CHAR (20));
```

**Eltern**

| *Kind* | *Elternteil* |
|---|---|
| John | Sabrina |
| Mary | Sabrina |
| Sabrina | Helmut |
| Helmut | Jennifer |
| ... | ... |

**Alle Vorfahren von „John“ ?**

```
WITH RECURSIVE Vorfahren (Kind,Vorfahr,Generation) AS
( SELECT Kind, Elternteil, 1
   FROM Eltern
UNION
SELECT V.Kind, E.Elternteil, V.Generation+1
   FROM Vorfahren V, Eltern E
   WHERE V.Vorfahr = E.Kind)

SELECT Generation, Vorfahr  FROM Vorfahren
   WHERE Kind=“John“
```

**Komplette transitive Hülle, danach Einschränkung auf „John“**

**Vorfahren**

| *Kind* | *Vorfahr* | *Gen.* |
|---|---|---|
| John | Sabrina | 1 |
| Mary | Sabrina | 1 |
| Sabrina | Helmut | 1 |
| Helmut | Jennifer | 1 |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |

# Rekursion: Beispiel (Forts.)

```
CREATE TABLE Eltern  (Kind   CHAR (20),
                      Elternteil  CHAR (20));
```

| *Kind* | *Elternteil* |
|---|---|
| John | Sabrina |
| Mary | Sabrina |
| Sabrina | Helmut |
| Helmut | Jennifer |
| ... | ... |

**Alle Vorfahren von „John“ ?**

```
WITH RECURSIVE Vorfahren (Kind, Vorfahr, Generation) AS
( SELECT Kind, Elternteil, 1
    FROM Eltern
    WHERE Kind=“John“
UNION
SELECT V.Kind, E.Elternteil, V.Generation+1
    FROM Vorfahren V, Eltern E
    WHERE V.Vorfahr = E.Kind)

SELECT Generation, Vorfahr  FROM Vorfahren
```

**Transitive Hülle nur für „John“**

| *Kind* | *Vorfahr* | *Gen.* |
|---|---|---|
| John | Sabrina | 1 |
| | | |
| | | |
| | | |
| | | |
| | | |

**Vorfahren**



# Rekursionsbeispiele Postgres (LOTS)

SQL Anfrage

```
WITH RECURSIVE t(n) AS (
    VALUES (1)
  UNION
    SELECT n+1 FROM t WHERE n < 100
)
SELECT sum(n) FROM t;
```

| sum |
|---|
| 5050 |

SQL Anfrage

```
WITH RECURSIVE Fak(n, r) AS (
    SELECT 0::NUMERIC, 1::NUMERIC
UNION ALL
    SELECT n+1, (n+1) * r
    FROM Fak
)
SELECT *
FROM Fak
LIMIT 100;
```

| n | r |
|---|---|
| 80 | 71569457046263802294811533723186532165584657342365752577109445058227039255480148842668944867280814080000000000000000000 |
| 81 | 57971260207473679858797342315781091054123572447316259587458650497163901796938920562561845342497459404800000000000000000000 |
| 82 | 47536433370128417484213820698940494664381329406799332861716093407674399473489914861300713180847916711936000000000000000000000 |
| 83 | 39455239697206586511897471180120610571436503407643446275224357528369751562996629334879591940103770870906880000000000000000000 |
| 84 | 33142401345653532669993875791301312880006662862420494871188460323830591312917168641298857229687167531561779200000000000000000000 |
| 85 | 28171041143805502769494794422606115948005663433057420640510191275256002615979593345104028645234092401827512320000000000000000000 |
| 86 | 24227095383672732381765523203441259715284870552429381750838764496720162249742450276789464634901319465571660595200000000000000000000 |
| 87 | 21077572983795277172136005186993895952297837380613562123229725112146541157275931740806834232364147935047344717824000000000000000000000 |
| 88 | 18548264225739843911479684564554628438022096894939934668442158098688956218402819931910014124480450182841663351685120000000000000000000000 |
| 89 | 16507955160908461081216919262453619309839666236496541854913520707833171034378509739399125707876006627290803829997568000000000000000000000 |
| 90 | 14857159644817614973095227336208257378855699612846887669422168637049853930940658765459921313708840596456172344699781120000000000000000000000 |
| 91 | 13520015276784029625516656875949514214758686647690667779174173459715367077155999476568528395475044942775116833676800819200000000000000000000000 |
| 92 | 12438414054641307255475324325873553077577991715875414356840239589283177109835195184430461238370413473531074869826567536640000000000000000000000 |
| 93 | 11567725070816415747592051623062404362147532295764135351861422812132468071214673152152032895168448453038389962893870780907520000000000000000000000 |
| 94 | 10873661566567430802736528525678660100418680358018287230749737443404519986941792763022910921458341545856086565120238534053068800000000000000000000000 |
| 95 | 10329978488239059262599702099394727095397746340117372869212250571234293987594703124871765375385424468563282236864226607350415360000000000000000000000 |
| 96 | 99167793487094968920957140154189380115818364865126779544437605483849222280909149998768947603700074898207509473896575430563987456000000000000000000000000 |
| 97 | 96192759682482119853328425949563698712343813919172976158104477319333745612481875498805879175589072651261284189679678167647067832320000000000000000000000 |
| 98 | 94268904488832477456261857430572424738096937640789516634942387772947070700232237988829761592077291198236058505886084604294126475673600000000000000000000000 |
| 99 | 93326215443944152681699238856266700490715968264381621468592963895217599993229915608941463976156518286253697920827223758251185210916864000000000000000000000000 |

# Rekursionsbeispiele Postgres (2)

Direkte und indirekte Koautorschaften von Autor 68

**SQL Anfrage**

```
WITH RECURSIVE koautoren(autor) as (
  SELECT DISTINCT autorid
  FROM buch_aut
  WHERE buchid IN
      (SELECT buchid FROM buch_aut WHERE autorid=68)
      AND autorid<>68
UNION ALL
   SELECT DISTINCT ba.autorid
   FROM buch_aut ba, koautoren k
   WHERE ba.autorid<> k.autor AND ba.buchid IN (SELECT buchid FROM buch_aut WHERE autorid=k.autor) )

SELECT DISTINCT autor FROM koautoren
WHERE autor<>68
```

**Fehlermeldung**

Ihre Anfrage benötigt mehr als die erlaubten 30 Sekunden.



# Rekursionsbeispiele Postgres (3)

Direkte und indirekte Koautorschaften von Autor 68

**SQL Anfrage**

```
WITH RECURSIVE koautoren(autor,afeld) as (
  SELECT DISTINCT autorid, ARRAY[autorid]
  FROM buch_aut
  WHERE buchid IN
      (SELECT buchid FROM buch_aut WHERE autorid=68)
      AND autorid<>68
UNION
   SELECT DISTINCT ba.autorid, k.afeld || ba.autorid
   FROM buch_aut ba, koautoren k
   WHERE ba.autorid <> k.autor AND (NOT ba.autorid = ANY(k.afeld)) AND ba.buchid IN (SELECT buchid FROM buch_aut WHERE autorid=k.autor) )

SELECT DISTINCT autor FROM koautoren
WHERE autor<>68
```

zeige Datensätze 1 - 4 (4 insgesamt)

Zeige: 4 Datensätze, beginnend ab 1

| autor |
|---|
| 3966 |
| 1761 |
| 67 |
| 407 |

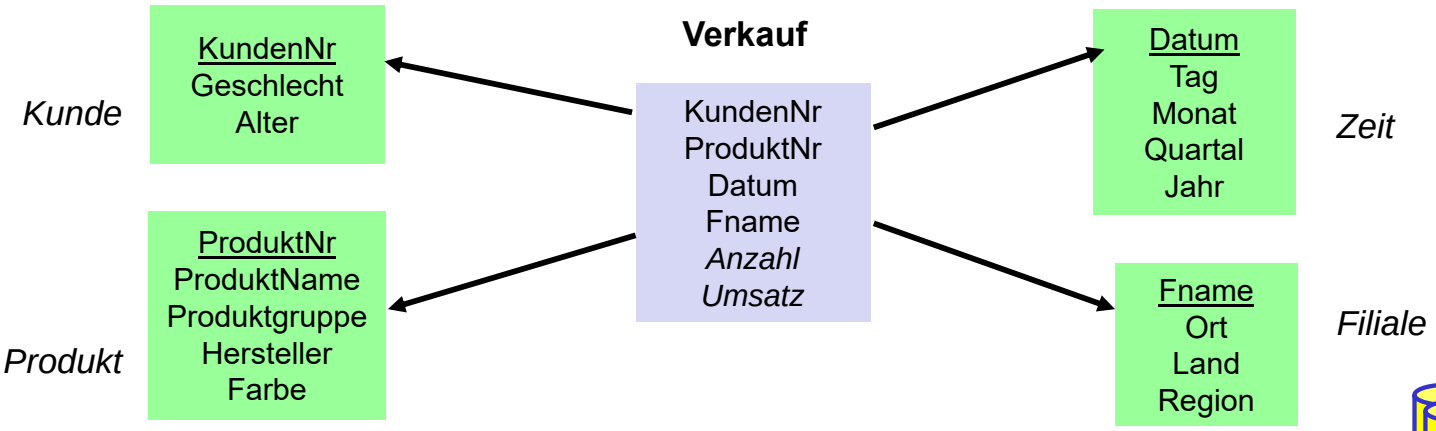
# 5. Fortgeschrittene SQL-Themen

- Rekursive Anfragen
  - WITH-Anweisung
  - RECURSIVE UNION
- Fortgeschrittene Datenanalysen
  - Star-Joins
  - mehrdimensionale Gruppierungen (ROLLUP, CUBE, Grouping Sets)
  - Zeitreihenauswertungen (Ranking, Windowing)
- Temporale Datenbanken
  - anwendungsversionierte Tabellen und Zeitprädikate
  - systemversionierte Tabellen
  - bitemporale Tabellen



# Mehrdimensionale Auswertungen (OLAP)

- OLAP (Online Analytical Processing)
  - Datenanalysen für Entscheidungsunterstützung
  - Nutzung u.a. in Data Warehouses
  - Flexible Auswertungen entlang mehrerer „Dimensionen“, z.B. Zeit, Region, Produkt, etc.
- Relationale Warehouses nutzen oft Stern-Schema (star schema) mit zentraler Faktentabelle und mehreren Dimensionstabellen
  - Faktentabelle hat Kennzahlen (z.B. Umsatz) und Fremdschlüssel auf beschreibende Dimensionen



# 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

```
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     ...;
```

*aggregierte Kennzahlen*

*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 |
|---|---|
| 2020 | 745 |
| 2021 | 710 |
| 2019 | 650 |

# 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

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

**land**

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

**cov**

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

**zeit**



# 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 | 2019 | m | 1300 |
| VW | 2019 | w | 700 |
| ... | ... | ... | ... |
| VW | 2019 | ALL | 2.000 |
| ... | ... | ALL | ... |
| Ford | 2021 | ALL | 2.000 |
| VW | ALL | m | 5.400 |
| ... | ... | ... | ... |
| Ford | ALL | w | ... |
| ALL | 2019 | m | ... |
| ... | ... | | |
| VW | ALL | ALL | 8.500 |
| ... | ... | | |
| ALL | 2019 | ALL | ... |
| ... | | | |
| ALL | ALL | m | ... |
| ... | | | |
| ALL | ALL | ALL | 19.500 |

# Cube-Beispiel Covid-19-Daten

- Anfrage auf den LOTS-Daten (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

- ermöglichen mehrere Gruppierungen pro Anfrage
  - GROUP BY GROUPING SETS ( <Gruppenspezifikationsliste> )
  - Gruppenspezifikation: (<Gruppenspezif.liste> ) | [CUBE | ROLLUP] <Gruppenspezif.liste>
  - 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
```

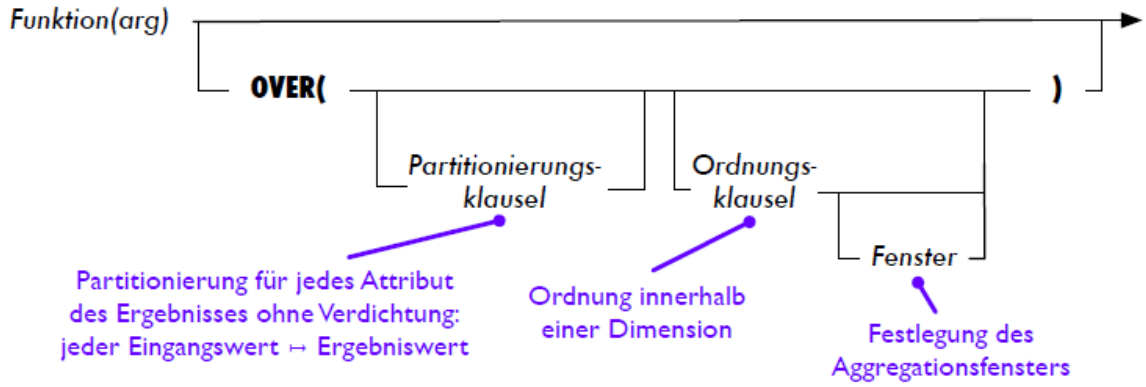
| land | jahr | faelle | tote |
|---|---|---|---|
| France | 2020 | 2,735,590 | 65,031 |
| France | 2021 | 7,706,191 | 59,161 |
| France | 2022 | 16,584,807 | 19,281 |
| France | | 27,026,588 | 143,473 |
| Germany | 2020 | 1,719,737 | 33,079 |
| Germany | 2021 | 5,430,685 | 78,854 |
| Germany | 2022 | 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 |

# Zeitreihen

- viele Data-Warehouse-Fakten entsprechen Zeitreihen
  - Sequenz von Sätzen mit Zeit/Datumsangabe
  - Bsp.: Produktverkäufe, Erkrankungsfälle, Temperaturmesswerte, etc.
- Notwendigkeit Auswertungen auf bestimmte Zeitbereiche etc. zu begrenzen
- SQL kann daher 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 Monats/Jahres)
  - 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



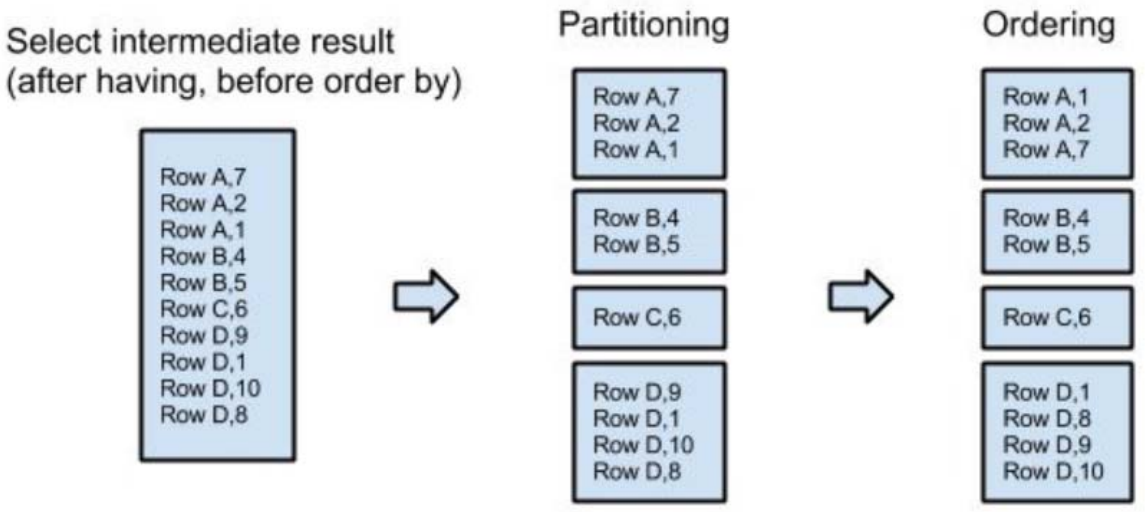

# OVER-Prädikat




- 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: Einschränkung für Aggregationsfenster auf Satzbasis (ROWS) oder für Wertebereiche (RANGE)



# 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=2019
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=2019
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

# 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




# 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 |

SQL Anfrage

```
with sales (sdate,value) as (Values ('2019-02-01',500),('2019-02-01',300),('2019-07-05',200),('2020-03-02',400),('2020-04-03',100),('2021-08-09',500))
select date, value, all_sum,   100*value/all_sum as anteil
from  (select sdate as date, value, sum(value) over () as all_sum from sales) as sales2
order by 1
```

zeige Datensätze 1 - 6 (6 insgesamt)

Zeige: 6 Datensätze, beginnend ab 1

| date | value | all_sum | anteil |
|---|---|---|---|
| 2019-02-01 | 500 | 2000 | 25 |
| 2019-02-01 | 300 | 2000 | 15 |
| 2019-07-05 | 200 | 2000 | 10 |
| 2020-03-02 | 400 | 2000 | 20 |
| 2020-04-03 | 100 | 2000 | 5 |
| 2021-08-09 | 500 | 2000 | 25 |

Zeige: 6 Datensätze, beginnend ab 1



# Aggregatberechnung auf Windows mit Grouping

- 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 |

SQL Anfrage

```
with sales (sdate,value) as (Values ('2019-02-01',500),('2019-02-01',300),('2019-07-05',200),('2020-03-02',400),('2020-04-03',100),('2021-08-09',500))

select sdate as date,day_sum, year_sum, round(100.0*day_sum/year_sum,0) as janteil
from (select sdate, sum(value) as day_sum, sum(sum(value)) over (partition by date_part('year',sdate::date)) as year_sum
      from sales
      group by sdate) as sales2
order by 1
```

zeige Datensätze 1 - 5 (5 insgesamt)

Zeige: 5 Datensätze, beginnend ab 1

| date | day_sum | year_sum | janteil |
|---|---|---|---|
| 2019-02-01 | 800 | 1000 | 80 |
| 2019-07-05 | 200 | 1000 | 20 |
| 2020-03-02 | 400 | 500 | 80 |
| 2020-04-03 | 100 | 500 | 20 |
| 2021-08-09 | 500 | 500 | 100 |

Zeige: 5 Datensätze, beginnend ab 1



# 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
```

**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 | 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 100.000) 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




# 5. Fortgeschrittene SQL-Themen

- Rekursive Anfragen
  - WITH-Anweisung
  - RECURSIVE UNION
- Fortgeschrittene Datenanalysen
  - Star-Joins
  - mehrdimensionale Gruppierungen (ROLLUP, CUBE, Grouping Sets)
  - Zeitreihenauswertungen (Ranking, Windowing)
- Temporale Datenbanken
  - anwendungsversionierte Tabellen und Zeitprädikate
  - systemversionierte Tabellen
  - bitemporale Tabellen

# Temporale Datenbanken

- systemseitige Unterstützung zeitbehafteter / versionierter Datenbanken
- unterschiedliche Zeiten bzw. Zeitintervalle
  - Gültigkeitszeit (valid time, „business time“)
    - kann in Vergangenheit, Gegenwart oder Zukunft liegen
    - durch Anwendung / Nutzer festzulegen
  - Transaktionszeit: Zeitpunkt der Änderung („system time“)
    - kann nicht in Zukunft liegen
    - automatische Festlegung durch DBS
  - *Bitemporale Datenbanken* unterstützen beide Zeitarten (bei Speicherung, Änderung und Abfragen)
- Zeitunterstützung seit SQL:2011 standardisiert*
  - anwendungsversionierte Tabellen (application-time period table) für Gültigkeitszeit
  - systemversionierte Tabellen für Transaktionszeit
  - anwendungs- und systemversionierte (bitemporale) Tabellen

* K. Kulkarni, J. Michels: *Temporal features in SQL:2011.* Sigmod Record, Sep. 2012




# Anwendungsversionierte Tabellen

- Verwendung von Zeitintervallen (PERIOD) mit Start- und Endzeitpunkt vom Typ DATE oder TIMESTAMP
  - halboffene Intervalle (Startzeitpunkt ist Teil des Intervalls, Endzeitpunkt nicht)
  - Primärschlüssel erfordert Hinzunahme des Zeitintervalls
  - Fremdschlüssel müssen für Bezugsintervall korrekt sein

```
CREATE TABLE PERS
(Pname VARCHAR(50) NOT NULL,
 ANR VARCHAR(10),
 Starttermin  DATE NOT NULL,
 Endtermin DATE NOT NULL,
 PERIOD FOR Pzeitraum (Starttermin, Endtermin),
 PRIMARY KEY(Pname, Pzeitraum WITHOUT OVERLAPS),
 FOREIGN KEY(ANR,PERIOD Pzeitraum)REFERENCES ABT(ANR, PERIOD Azeitraum));
```

| Pname | ANR | Starttermin | Endtermin |
|---|---|---|---|
| John | J13 | 15/11/2019 | 16/11/2020 |
| John | J14 | 16/11/2020 | 01/05/2021 |
| Tracy | K25 | 01/01/2020 | 31/12/9999 |

# Anwendungsversionierte Tabellen (2)

- Intervallanpassungen bei Änderungen (UPDATE, DELETE) möglich

```
INSERT INTO PERS (Pname, ANR, Starttermin, Endtermin)
       VALUES ('John', 'J12', DATE '15-11-2015', DATE '15-11-2019'')

UPDATE PERS FOR PORTION OF PZeitraum
    FROM DATE '01-03-2020' TO DATE '01-07-2020'
SET  ANR = 'M12'' WHERE Pname = 'John'

DELETE FROM PERS FOR PORTION OF PZeitraum
   FROM DATE '01-08-2019' TO DATE '01-09-2020'
WHERE Pname = 'Tracy'
```

| Pname | ANR | Starttermin | Endtermin | |
|---|---|---|---|---|
| *John* | *J12* | *15/11/2015* | *15/11/2019* | |
| John | J13 | 15/11/2019 | ~~16/11/2020~~ - | *01/03/2020* |
| John | J14 | 16/11/2020 | 01/05/2021 | |
| Tracy | K25 | 01/01/2019 | ~~31/12/9999~~ - | *01/08/2019* |
| John | M12 | 01/03/2020 | 01/07/2020 | |
| John | J13 | 01/07/2020 | 16/11/2020 | |
| Tracy | K25 | 01/09/2020 | 31/12/9999 | |




# Anwendungsversionierte Tabellen (3)

- Ansatz ermöglicht zeitbezogene Auswertungen über Start/Endzeitpunkte

```
SELECT ANR FROM PERS
WHERE Pname = 'John' AND Starttermin >= DATE '01-12-2020'
      AND Endttermin <= DATE '01-12-2020';
```

- oft einfacher mit Suchprädikaten für PERIOD-Intervalle:
  - CONTAINS
  - OVERLAPS
  - EQUALS
  - PRECEDES, SUCCEEDS, IMMEDIATELY PRECEDES, IMMEDIATELY SUCCEEDS

***Abteilung von John am 1. Dezember 2020:***

```
SELECT ANR FROM PERS
WHERE Pname = 'John' AND Pzeitraum CONTAINS DATE '01-12-2020';
```

***In wie vielen Abteilungen war John seit 2015 beschäftigt:***

```
SELECT count (distinct ANR) FROM PERS
WHERE Pname = 'John' AND
      Pzeitraum OVERLAPS PERIOD (DATE '01-01-2015', CURRENT_DATE)
```

# Systemversionierte Tabellen

- automatische Erzeugung und Anpassung von Zeitintervallen bzgl. Änderungszeit
  - obligatorische Attribute für Start- und Endzeitpunkte
  - nur aktuelle (nicht historische) Versionen von Tupeln können geändert/gelöscht werden
  - Integritätsbedingungen werden nur auf aktuellen Tupelversionen überwacht

CREATE TABLE **PERS**

```
(Pname VARCHAR(50) NOT NULL PRIMARY KEY,
 ANR VARCHAR(10),
 sys_start TIMESTAMP(6) GENERATED ALWAYS AS ROW START,
 sys_end TIMESTAMP(6) GENERATED ALWAYS AS ROW END,
 PERIOD FOR SYSTEM_TIME (sys_start, sys_end),
 FOREIGN KEY (ANR) REFERENCES  ABT (ANR);
)
WITH SYSTEM VERSIONING;
```

| Pname | ANR | sys_start | sys_end |
|---|---|---|---|
| John | J13 | 15/11/2019 | 31/12/9999 |
| Tracy | K25 | 15/11/2019 | 31/12/9999 |




# Systemversionierte Tabellen (2)

- UPDATE führt zu historischem Tupel und aktuellem Tupel

Änderung am 15.5.2021:

```
UPDATE PERS  SET  ANR = 'M12' WHERE Pname = 'John'
```

- DELETE setzt Endzeitpunkt auf Transaktionszeitpunkt

Löschen am 15.5.2021:

```
DELETE FROM PERS  WHERE Pname = 'Tracy'
```

| Pname | ANR | sys_start | sys_end | |
|---|---|---|---|---|
| John | J13 | 15/11/2019 | ~~31/12/9999~~ | *15/05/2021* |
| *John* | *M12* | *15/05/2021* | *31/12/9999* | |
| Tracy | K25 | 15/11/2019 | ~~31/12/9999~~ | *15/05/2021* |

# Anfragen auf systemversionierten Tabellen

- neue Suchprädikate
  - FOR SYSTEM_TIME AS OF <datetime value expression>
  - FOR SYSTEM_TIME BETWEEN < datetime 1> AND < datetime 2>
  - FOR SYSTEM_TIME FROM < datetime 1> TO < datetime 2>

```
SELECT ANR
FROM PERS FOR SYSTEM_TIME AS OF DATE '01-01-2020'
WHERE Pname = 'John'
```

```
SELECT count (distinct ANR)
FROM PERS FOR SYSTEM_TIME BETWEEN DATE '01-01-2019' AND CURRENT_DATE
WHERE Pname = 'John'
```




# Bitemporale Tabellen

```
CREATE TABLE PERS
(Pname VARCHAR(50) NOT NULL,
 ANR VARCHAR(10),
 Starttermin  DATE NOT NULL,
 Endtermin DATE NOT NULL,
 sys_start DATE GENERATED ALWAYS AS ROW START,
 sys_end DATE GENERATED ALWAYS AS ROW END,
 PERIOD FOR Pzeitraum (Starttermin, Endtermin),
 PERIOD FOR SYSTEM_TIME (system_start, system_end),
 PRIMARY KEY (Pname, Pzeitraum WITHOUT OVERLAPS),
 FOREIGN KEY (ANR, PERIOD Pzeitraum)
               REFERENCES  ABT (ANR, PERIOD Azeitraum)
 ) WITH SYSTEM VERSIONING;
```

| Pname | ANR | Starttermin | Endtermin | sys_start | sys_end |
|---|---|---|---|---|---|
| John | J13 | 15/11/2019 | 16/11/2020 | 15/11/2019 | 15/11/2020 |
| John | J14 | 16/11/2020 | 31/12/9999 | 15/11/2020 | 31/12/9999 |
| Tracy | K25 | 01/01/2020 | 31/12/9999 | 15/01/2020 | 31/12/9999 |

# Bitemporale Tabellen (2)

Änderung am 15.5.2021 (Korrektur falsche Abteilungszugehörigkeit):

```
UPDATE PERS  SET  ANR = 'M12' WHERE Pname = 'John'
```

Am 1.6.2021 wird John für den Zeitraum 1.10.2021 bis 31.3.2022 an Abteilung M13 ausgeliehen

```
UPDATE PERS FOR PORTION OF Pzeitraum FROM DATE '01-10-2021'
                                     TO DATE '01-04-2022'
SET ANR = 'M13' WHERE Pname = 'John'
```

In welcher Abteilung war John am 1.1.2021 gemäß Datenbankstand vom 1.5.2021?

```
SELECT ANR  FROM PERS FOR SYSTEM_TIME AS OF DATE '01-05-2021'
WHERE Pname = 'John' AND  Pzeitraum CONTAINS DATE '01-01-2021'
```

| Pname | ANR | Starttermin | Endtermin | sys_start | sys_end |
|---|---|---|---|---|---|
| John | M12 | 16/11/2020 | 01/10/2021 | 01/06/2021 | 31/12/9999 |
| John | M13 | 01/10/2021 | 01/04/2022 | 01/06/2021 | 31/12/9999 |
| John | M12 | 01/04/2022 | 31/12/9999 | 01/06/2021 | 31/12/9999 |
| John | M12 | 16/11/2020 | 31/12/9999 | 15/05/2021 | ~~31/12/9999~~ *01/06/2021* |
| John | J14 | 16/11/2020 | 31/12/9999 | 15/11/2020 | ~~31/12/9999~~ *15/05/2021* |
| Tracy | K25 | 01/01/2020 | 31/12/9999 | 15/11/2020 | 31/12/9999 |




# Zusammenfassung

- zahlreiche mächtige SQL-Erweiterungen
- With-Statement
  - Nutzung von benannten Zwischenergebnissen in Anfrage
- Rekursion mit With-Statement und rekursivem UNION
- Datenanalysen
  - Star-Joins für Data Warehouses mit Star Schema
  - mehrdimensionale Gruppierungen/Aggregationen (CUBE, Rollup, Grouping Sets)
  - Window-Anfragen mit Aggregationen und Ranking
- Temporale Datenbanken seit SQL:2011
  - Zeitintervalle (PERIOD) für Gültigkeits- und Transaktionszeit
  - Gültigkeitszeit in anwendungsversionierten Tabellen; Transaktionszeit in systemversionierten bzw. bitemporalen Tabellen wird automatisch gesetzt
  - neue Zeitprädikate (contains, overlaps, …)