

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 Vorbereitung 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: $Vorfahr(K,V) \leftarrow Elternteil(K,V)$
 $Vorfahr(K,V) \leftarrow Vorfahr(K,X), Elternteil(X,V)$

```

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

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"
    
```

Eltern

Kind	Elternteil
John	Sabrina
Mary	Sabrina
Sabrina	Helmut
Helmut	Jennifer
...	...

Kind	Vorfahr	Gen.
John	Sabrina	1
Mary	Sabrina	1
Sabrina	Helmut	1
Helmut	Jennifer	1

Komplette transitive Hülle, danach Einschränkung auf „John“

Vorfahren



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
```

Kind	Vorfahr	Gen.
John	Sabrina	1

Vorfahren

Transitive Hülle nur für „John“



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



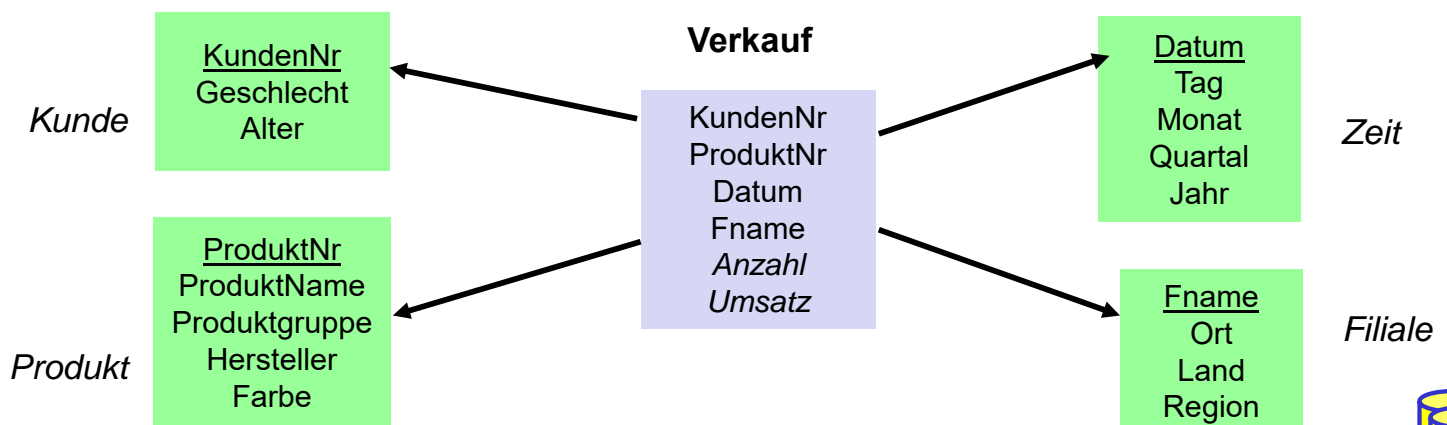
Mehrdimensionale Auswertungen (OLAP)

■ 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, aggregierte Kennzahlen agg(f1), ... agg (fm)
from       D1, ..., Dn, F Relationen des Star-Schemas
where      <Selektionsbedingung auf D1> and
              ... and
              <Selektionsbedingung auf Dn> and
              D1.d1 = F.d1 and Join-Bedingungen
              ... and
              Dn.dn = F.dn
group by   g1, ... gk Ergebnis-Dimensionalität
sort by    ... i
    
```



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



Beispiel Covid-19-Daten

Download z.B. <https://opendata.ecdc.europa.eu/covid19/casedistribution/csv> (bis 14.12.2020)

CSV-Import in relationale DB (MySQL, PostgreSQL ...)

day	month	year	country	cases	deaths	pop	cont
27	4	2020	Finland	101	4	5518050	Europe
27	4	2020	France	461	242	66987244	Europe
27	4	2020	French_Poly...	0	0	277679	Oceania
27	4	2020	Gabon	0	0	2119275	Africa
27	4	2020	Gambia	0	0	2280102	Africa
27	4	2020	Georgia	30	1	3731000	Europe
27	4	2020	Germany	1018	110	82927922	Europe
27	4	2020	Ghana	271	1	29767108	Africa
27	4	2020	Gibraltar	5	0	33718	Europe
27	4	2020	Greece	0	0	10727668	Europe

Group-by continent + month (drill-down, 2-dimens.)

Group-by continent (1-dim. Aggregation)

cont	cases	deaths
Asia	16782046	290129
Europe	21400012	479789
Africa	2379827	56334
America	30887593	785420
Oceania	53437	1154

cont	month	cases	deaths
Asia	10	2948487	45952
Asia	11	2606038	42872
Asia	12	1087897	17759
Europe	3	448271	28117
Europe	4	964192	108348
Europe	5	668611	40296
Europe	6	516302	16573
Europe	7	517419	11291
Europe	8	810756	10736
Europe	9	1478063	16456
Europe	10	4868094	44565
Europe	11	7854563	132821
Europe	12	3272587	70563
Oceania	3	5298	21
Oceania	4	2812	95



Cube-Operator

■ SQL- CUBE-Operator für n-dimensionale Gruppierung und Aggregation

- Syntax: *Group By CUBE (D₁, D₂, ... D_n)*
- n-dimensionale Gruppierung/Aggregation + alle Gruppierungen geringerer Dimensionalität;
2ⁿ 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, PostgreSQL (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



ROLLUP-Operator

- CUBE-Operator: inter-dimensionale Gruppierung / Aggregation
 - 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 Aggregation
 - Syntax: *Group By ROLLUP (D₁, D₂, ... D_n)*
 - liefert nur die n+1 Gruppierungen

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

- 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

■ MySQL-Anfrage (leicht abweichende Syntax)

```
SELECT cont, country, sum(cases) as cases, sum(deaths) as deaths,
FORMAT(((SUM(`cov-cases`.`deaths`) * 1000000) / `cov-cases`.`pop`),1) AS `DeathsPerMillion` FROM covid.`cov-cases`
group by cont, country with rollup;
```

cont	country	cases	deaths	DeathsPerMillion
Europe	Poland	1135676	22864	602.1
Europe	Portugal	348744	5559	540.9
Europe	Romania	556335	13385	689.4
Europe	Russia	2653928	46941	321.8
Europe	San_Marino	1947	51	1,480.3
Europe	Serbia	266432	2331	334.7
Europe	Slovakia	132984	1175	215.6
Europe	Slovenia	96314	1459	701.1
Europe	Spain	1730575	47624	1,014.6
Europe	Sweden	320098	7514	734.5
Europe	Switzerland	372329	5378	629.4
Europe	Turkey	995471	16199	197.5
Europe	Ukraine	900666	15247	346.6
Europe	United_Kingdom	1849403	64170	962.8
Europe	NULL	21400012	479789	
Oceania	Australia	28031	908	36.0
Oceania	Fiji	46	2	2.2
Oceania	French_Polynesia	15618	91	325.8
Oceania	NULL	53437	1154	
NULL	NULL	71502915	1612826	

Datenstand: 14.12.2020



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 (

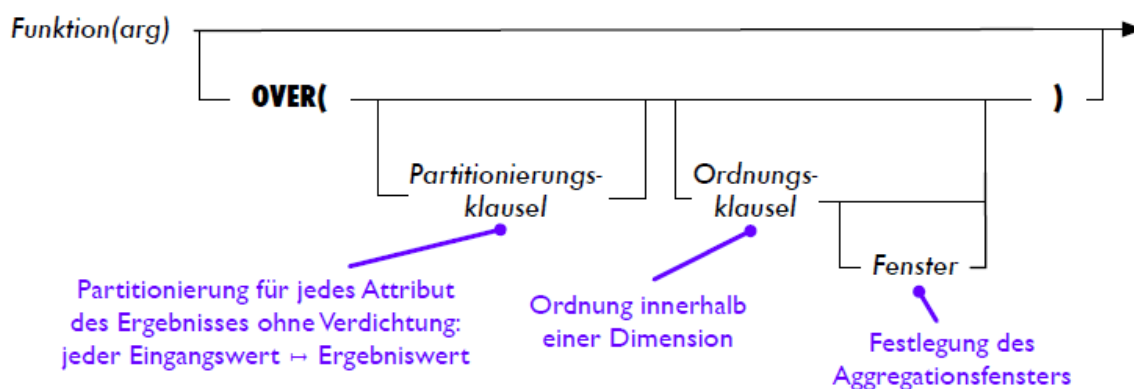


Zeitreihen

- 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



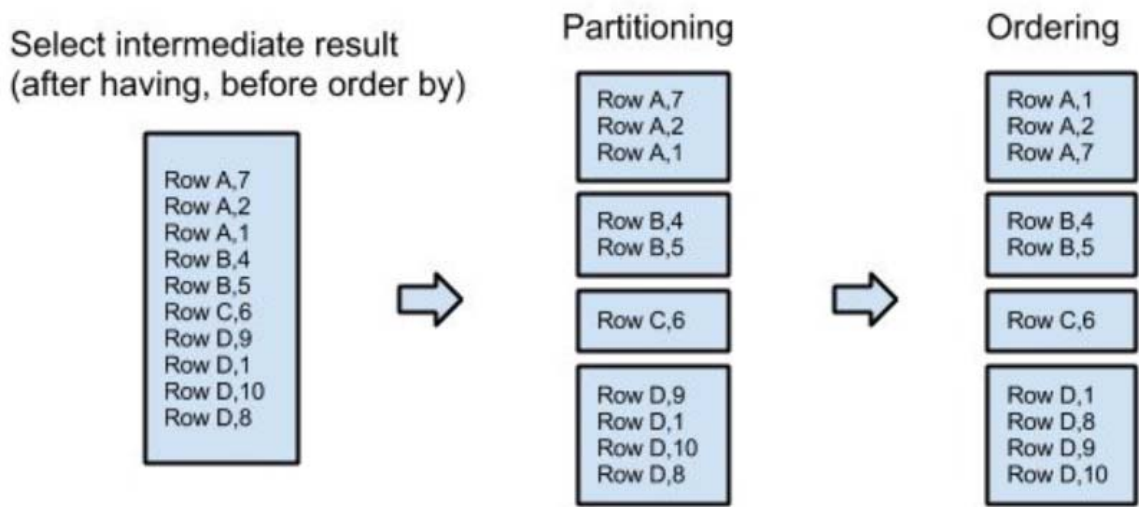
OVER-Prädikat



- Funktionen: `SUM`, `AVG`, `RANK`, `DENSE_RANK`, ...
 - Berechnung bezüglich durch OVER spezifiziertes Intervall
- `PARTITION BY`: Zerlegung in mehrere Datensequenzen/ströme (optional)
- `ORDER BY`: Sortierreihenfolge 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=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



Rank-Beispiel (Covid-Datenbank)

```
select `covid`.`cov-cases`.`country` AS `country`, sum(`covid`.`cov-cases`.`cases`) AS `cases`,
sum(`covid`.`cov-cases`.`deaths`) AS `deaths`, rank() OVER `w1` AS `RankCases`, rank() OVER `w2` AS `RankDeaths`
from `covid`.`cov-cases` group by `covid`.`cov-cases`.`country`
window `w1` AS (ORDER BY sum(`covid`.`cov-cases`.`cases`) desc ) ,
`w2` AS (ORDER BY sum(`covid`.`cov-cases`.`deaths`) desc ) limit 16
```

country	cases	deaths	RankCases	RankDeaths
United_States_of_America	16256754	299177	1	1
Brazil	6901952	181402	3	2
India	9884100	143355	2	3
Mexico	1250044	113953	12	4
Italy	1843712	64520	7	5
United_Kingdom	1849403	64170	6	6
France	2376852	57911	5	7
Iran	1108269	52196	14	8
Spain	1730575	47624	8	9
Russia	2653928	46941	4	10
Argentina	1498160	40766	9	11
Colombia	1425774	39053	10	12
Peru	984973	36677	16	13
South_Africa	860964	23276	18	14
Poland	1135676	22864	13	15
Germany	1337078	21975	11	16

Datenstand: 14.12.2020



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 country, cont, sum(cases) as cases,
rank()over(partition by cont order by(sum(cases)desc) as ContRank,
from cov-cases
group by country order by 3 desc
```

country	cont	cases	ContRank
United_States_of_America	America	16256754	1
India	Asia	9884100	1
Brazil	America	6901952	2
Russia	Europe	2653928	1
France	Europe	2376852	2
United_Kingdom	Europe	1849403	3
Italy	Europe	1843712	4
Spain	Europe	1730575	5
Argentina	America	1498160	3
Colombia	America	1425774	4
Germany	Europe	1337078	6
Mexico	America	1250044	5
Poland	Europe	1135676	7

Datenstand: 14.12.2020



Partitionsweises Ranking: Covid-Beispiel

Top-3 pro
Kontinent

```
with CRank as (select `cov-cases`.`country` AS `country`, `cov-cases`.`cont` AS `cont`,
sum(`cov-cases`.`cases`) AS `cases`, rank() OVER `w1` AS `GlobalRank`,
rank() OVER (PARTITION BY `cov-cases`.`cont` ORDER BY sum(`cov-cases`.`cases`) desc )
AS `ContRank` from `covid`.`cov-cases` group by `cov-cases`.`country`
window `w1` AS (ORDER BY sum(`cov-cases`.`cases`) desc ) )
Select * FROM CRank where ContRank <=3 order by GlobalRank
```

country	cont	cases	GlobalRank	ContRank
United_States_of_America	America	16256754	1	1
India	Asia	9884100	2	1
Brazil	America	6901952	3	2
Russia	Europe	2653928	4	1
France	Europe	2376852	5	2
United_Kingdom	Europe	1849403	6	3
Argentina	America	1498160	9	3
Iran	Asia	1108269	14	2
South_Africa	Africa	860964	18	1
Indonesia	Asia	617820	19	3
Morocco	Africa	399609	30	2
Egypt	Africa	121575	64	3
Australia	Oceania	28031	99	1
French_Polynesia	Oceania	15618	112	2
Guam	Oceania	7106	132	3

Datenstand: 14.12.2020



Aggregatberechnung auf Windows

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

Summe der Verkäufe pro Tag sowie Anteil an Gesamtsumme

```
select date, sum(value) as day_sum, sum(value) over () as all_sum,
100.0*day_sum/all_sum as anteil
from sales
group by date
```

sales

date	value
1.2.2017	500
1.2.2017	300
5.7.2017	200
2.3.2018	400
3.4.2018	100
9.6.2019	500

date	day_sum	all_sum	anteil
1.2.2017	800	2000	40
5.7.2017	200	2000	10
2.3.2018	400	2000	20
3.4.2018	100	2000	5
9.6.2019	500	2000	25



Aggregate (2)

Summe der Verkäufe eines Tages im Verhältnis zu Verkäufen des Jahres

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

sales

date	value
1.2.2017	500
1.2.2017	300
5.7.2017	200
2.3.2018	400
3.4.2018	100
9.6.2019	500

date	day_sum	year_sum	janteil
1.2.2017	800	1000	80
5.7.2017	200	1000	20
2.3.2018	400	500	80
3.4.2018	100	500	20
9.6.2019	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 date, sum(value) AS day_sum,  
       sum(value)over(order by date) as cum_sum,  
       sum(value)over(partition by year(date) order by date)as cumy_sum  
from sales  
group by date  
order by date
```

date	value
1.2.2017	500
1.2.2017	300
5.7.2017	200
2.3.2018	400
3.4.2018	100
9.6.2019	500

sales

date	day_sum	cum_sum	cumy_sum
1.2.2017	800	800	800
5.7.2017	200	1000	1000
2.3.2018	400	1400	400
3.4.2018	100	1500	500
9.6.2019	500	2000	500



Covid-Beispiel Kum. Summen

Tagesfälle sowie kumulierte Summen insgesamt und pro Monat

```
SELECT day,month, cases, sum(cases) over w1 as cum_sum, sum(cases) over w2 as cum_msum
FROM covid.`cov-cases` where country="Germany" and cases<>0
window w1 as (order by month,day),
w2 as (partition by month order by month,day);
```

day	month	cases	cum_sum	cum_msum
28	1	1	1	1
29	1	3	4	4
31	1	1	5	5
1	2	2	7	2
2	2	1	8	3
3	2	1	9	4
4	2	2	11	6
7	2	1	12	7
8	2	1	13	8
12	2	2	15	10
26	2	2	17	12
27	2	4	21	16
28	2	26	47	42
29	2	10	57	52
1	3	54	111	54
2	3	18	129	72
3	3	28	157	100
4	3	39	196	139
5	3	66	262	205

day	month	cases	cum_sum	cum_msum
28	3	6294	48582	48525
29	3	3965	52547	52490
30	3	4751	57298	57241
31	3	4615	61913	61856
1	4	5453	67366	5453
2	4	6156	73522	11609
3	4	6174	79696	17783
4	4	6082	85778	23865
5	4	5936	91714	29801
6	4	3677	95391	33478
7	4	3834	99225	37312
8	4	4003	103228	41315
9	4	4974	108202	46289
10	4	5323	113525	51612
11	4	4133	117658	55745
12	4	2821	120479	58566
13	4	2537	123016	61103
14	4	2082	125098	63185
15	4	2486	127584	65671
16	4	2866	130450	68537
17	4	3380	133830	71917
18	4	3609	137439	75526
19	4	2458	139897	77984
20	4	1775	141672	79759
21	4	1785	143457	81544
22	4	2237	145694	83781
23	4	2352	148046	86133
24	4	2337	150383	88470



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 date, avg(value) over
      (order by date 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, zB 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 `day`,`month`,`cases`,  
format(avg(`cases`) OVER `w1`,0) AS `avgCasesWeek`,  
format( 100000/pop*sum(`cases`) OVER `w1`,0) AS `WeekIncidence`  
from `covid`.`cov-cases` where (`country` = 'Germany')  
window `w1` AS (ORDER BY `month`,`day` ROWS BETWEEN 6 PRECEDING AND 0 FOLLOWING)
```

day	month	cases	avgCasesWeek	WeekIncidence
27	11	22806	18,119	152
28	11	21695	17,937	151
29	11	14611	17,776	149
30	11	11169	17,819	150
1	12	13604	17,827	150
2	12	17270	17,632	148
3	12	22046	17,600	148
4	12	23448	17,692	149
5	12	23318	17,924	151
6	12	17767	18,375	154
7	12	12332	18,541	156
8	12	14055	18,605	156
9	12	20815	19,112	161
10	12	23679	19,345	162
11	12	29875	20,263	170
12	12	28438	20,994	176
13	12	20200	21,342	179
14	12	16362	21,918	184



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	15/11/2017	16/11/2018
John	J14	16/11/2018	01/05/2019
Tracy	K25	01/01/2018	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 '15-11-2017')
```

```
UPDATE PERS FOR PORTION OF PZeitraum
FROM DATE '01-03-2018' TO DATE '01-07-2018'
SET ANR = 'M12' WHERE Pname = 'John'
```

```
DELETE FROM PERS FOR PORTION OF PZeitraum
FROM DATE '01-08-2017' TO DATE '01-09-2018'
WHERE Pname = 'Tracy'
```

Pname	ANR	Starttermin	Endtermin	
John	J12	15/11/2015	15/11/2017	
John	J13	15/11/2017	16/11/2018	01/03/2018
John	J14	16/11/2018	01/05/2019	
Tracy	K25	01/01/2017	31/12/9999	01/08/2017
John	M12	01/03/2018	01/07/2018	
John	J13	01/07/2018	16/11/2018	
Tracy	K25	01/09/2018	31/12/9999	



Anwendungsversionierte Tabellen (3)

- Ansatz ermöglicht zeitbezogene Auswertungen über Start/Endzeitpunkte

```
SELECT ANR FROM PERS
WHERE Pname = 'John' AND Starttermin >= DATE '01-12-2018'
AND Endttermin <= DATE '01-12-2018';
```

- oft einfacher mit Suchprädikaten für PERIOD-Intervalle:

- CONTAINS
- OVERLAPS
- EQUALS
- PRECEDES, SUCCEEDS, IMMEDIATELY PRECEDES, IMMEDIATELY SUCCEEDS

Abteilung von John am 1. Dezember 2018:

```
SELECT ANR FROM PERS
WHERE Pname = 'John' AND PZeitraum CONTAINS DATE '01-12-2018';
```

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	15/11/2017	31/12/9999
Tracy	K25	15/11/2017	31/12/9999



Systemversionierte Tabellen (2)

- UPDATE führt zu historischem Tupel und aktuellem Tupel

Änderung am 15.5.2019:

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

- DELETE setzt Endzeitpunkt auf Transaktionszeitpunkt

Löschen am 15.5.2019:

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

Pname	ANR	sys_start	sys_end
John	J13	15/11/2017	31/12/9999
<i>John</i>	<i>M12</i>	<i>15/05/2019</i>	<i>31/12/9999</i>
Tracy	K25	15/11/2017	31/12/9999

15/05/2019

15/05/2019



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 '01-01-2018'
WHERE Pname = 'John'
```

```
SELECT count (distinct ANR)
FROM PERS FOR SYSTEM_TIME BETWEEN DATE '01-01-2017' 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	15/11/2017	16/11/2018	15/11/2017	15/11/2018
John	J14	16/11/2018	31/12/9999	15/11/2018	31/12/9999
Tracy	K25	01/01/2018	31/12/9999	15/01/2018	31/12/9999



Bitemporale Tabellen (2)

Änderung am 15.5.2019 (Korrektur falsche Abteilungszugehörigkeit):

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

Am 1.6.2019 wird John für den Zeitraum 1.10.2019 bis 31.3.2020 an Abteilung M13 ausgeliehen

```
UPDATE PERS FOR PORTION OF Pzeitraum FROM DATE '01-10-2019'
    TO DATE '01-04-2020'
SET ANR = 'M13' WHERE Pname = 'John'
```

In welcher Abteilung war John am 1.1.2019 gemäß Datenbankstand vom 1.5.2019?

```
SELECT ANR FROM PERS FOR SYSTEM_TIME AS OF DATE '01-05-2019'
WHERE Pname = 'John' AND Pzeitraum CONTAINS DATE '01-01-2019'
```

Pname	ANR	Starttermin	Endtermin	sys_start	sys_end
John	M12	16/11/2018	01/10/2019	01/06/2019	31/12/9999
John	M13	01/10/2019	01/04/2020	01/06/2019	31/12/9999
John	M12	01/04/2020	31/12/9999	01/06/2019	31/12/9999
John	M12	16/11/2018	31/12/9999	15/05/2019	31/12/9999 01/06/2019
John	J14	16/11/2018	31/12/9999	15/11/2018	31/12/9999 15/05/2019
Tracy	K25	01/01/2018	31/12/9999	15/11/2018	31/12/9999



Zusammenfassung

- zahlreiche mächtige SQL-Erweiterungen
- With-Statement
 - Nutzung von benannten Zwischenergebnissen in Anfrage
- Rekursion mit With-Statement und Recursive 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, ...)

