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

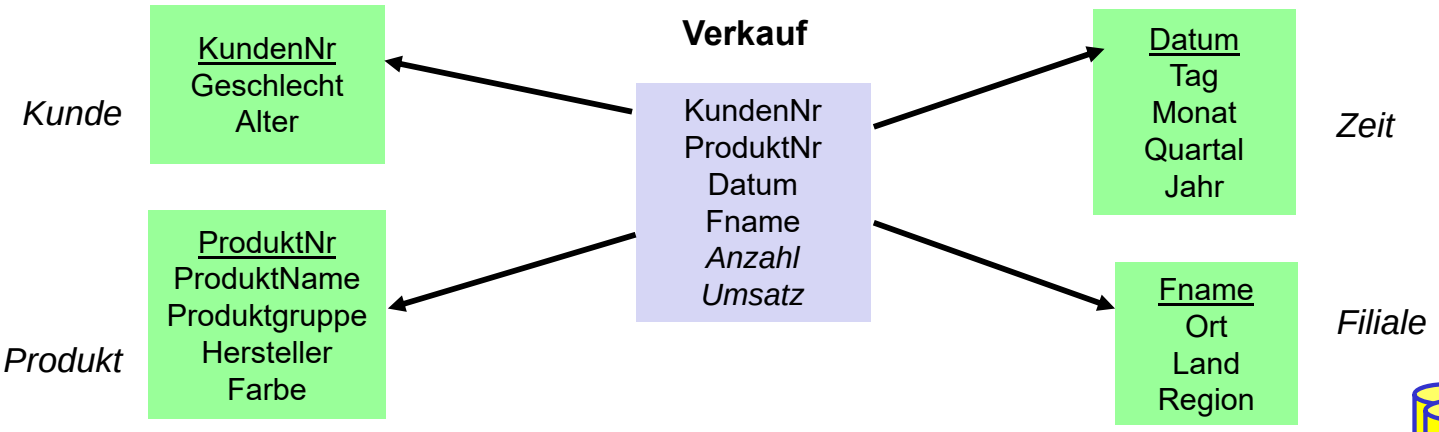
Vorfahren

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

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

Beispiel Covid-19-Daten

Download z.B. https://opendata.ecdc.europa.eu/covid19/casedistribution/csv

CSV-Import in relationale DB (MySql, PostgresSQL …)

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 (1-dim. Aggregation)

cont	cases	deaths
Africa	60,616	2,223
America	1,736,688	103,260
Asia	645,103	21,770
Europe	1,534,605	151,429
Oceania	8,344	125
Other	696	7

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

Africa	3	5,122	166
Africa	4	31,565	1,425
Africa	5	23,926	632
America	3	188,658	3,687
America	4	1,104,842	70,904
America	5	443,103	28,669
Asia	3	85,873	3,990
Asia	4	333,821	11,321
Asia	5	142,148	3,567
Europe	3	407,028	27,125
Europe	4	881,207	105,679
Europe	5	245,247	18,602
Oceania	3	5,298	21
Oceania	4	2,812	95
Oceania	5	208	9

Datenstand: 10.5.2020

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

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

- 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	Portugal	25056	989	96.2
Europe	Romania	12240	717	36.8
Europe	Russia	106498	1073	7.4
Europe	Serbia	9009	179	25.6
Europe	Slovakia	1396	23	4.2
Europe	Slovenia	1429	91	44.0
Europe	Spain	213435	24543	525.3
Europe	Sweden	21092	2586	253.9
Europe	Switzerland	29503	1422	167.0
Europe	Ukraine	9866	250	5.6
Europe	United_Kingdom	171253	26771	402.6
Europe	NULL	1302143	134421	
Oceania	Australia	6762	92	3.7
Oceania	New_Zealand	1132	19	3.9
Oceania	Papua_New_Guinea	8	0	0.0
Oceania	NULL	7902	111	
NULL	NULL	3201470	232231	

Datenstand: 1.5.2020

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

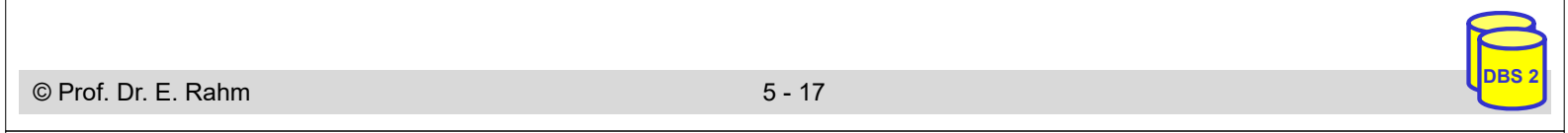
```
select p.Hersteller, p.Farbe,
          sum (v.Anzahl)
from Verkauf v, Produkt p
where v.ProduktNr = p. ProduktNr and
  p.Hersteller in („VW“,“Opel“)
group by grouping sets
  ((p.Hersteller), (p.Farbe));
```

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

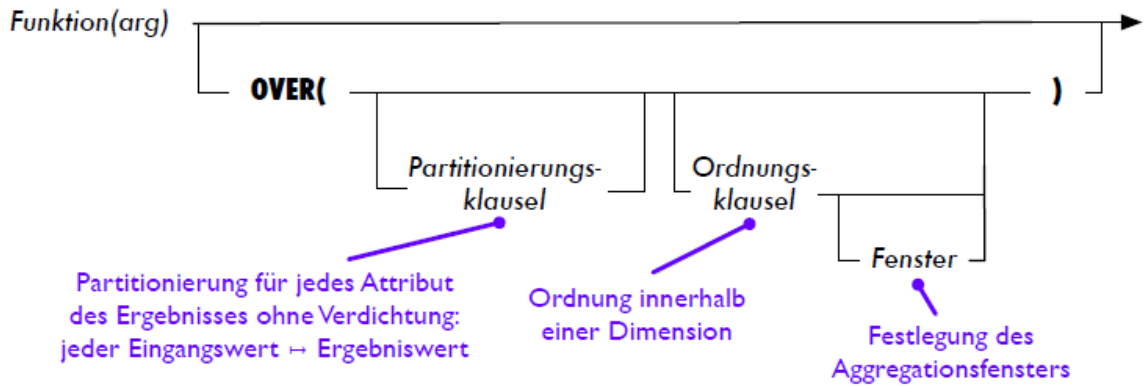
- CUBE, ROLLUP, herkömmliches Group-By entsprechen speziellen Grouping-Sets
  - GROUP BY A,B -> GROUP BY GROUPING SETS (
  - GROUP BY ROLLUP (A,B) -> GROUP BY GROUPING SETS (
  - GROUP BY CUBE (A,B) -> GROUP BY GROUPING SETS (

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

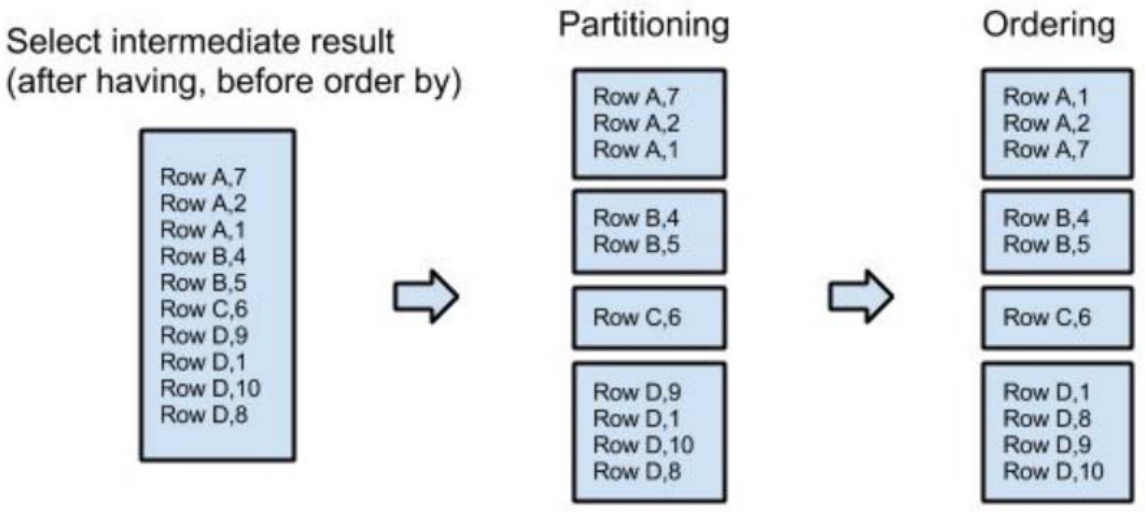


OVER-Prädikat



- Funktionen: `SUM`, `AVG`, **`RANK`**, **`DENSE_RANK`**, …
  - Berechnung bezüglich durch OVER spezifiziertes Intervall
- PARTITION BY: Zerlegung in mehrere Datensequenzen/ströme (optional)
- ORDER BY: 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=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 country, sum(cases) as cases, sum(deaths) as deaths,
rank() over w1    as RankCases, rank() over w2    as RankDeaths
FROM covid.`cov-cases`
group by country
window w1 as (order by sum(cases) desc), w2 as  (order by sum(deaths) desc)
Limit 12
```

country	cases	deaths	RankCases	RankDeaths
United_States_of_America	1309541	78794	1	1
United_Kingdom	215260	31587	4	2
Italy	218268	30395	3	3
Spain	223578	26478	2	4
France	138854	26310	8	5
Brazil	155939	10627	7	6
Belgium	52596	8581	15	7
Germany	169218	7395	6	8
Iran	106220	6589	10	9
Netherlands	42382	5422	16	10
Canada	67702	4693	12	11
China	83991	4637	11	12

Datenstand: 10.5.2020

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 ContRang,
from cov-cases
group by country order by 3 desc
```

country	cont	cases	contRank
United_States_of_America	America	1309541	1
Spain	Europe	223578	1
Italy	Europe	218268	2
United_Kingdom	Europe	215260	3
Russia	Europe	198676	4
Germany	Europe	169218	5
Brazil	America	155939	2
France	Europe	138854	6
Turkey	Asia	137115	1
Iran	Asia	106220	2
China	Asia	83991	3

Datenstand: 10.5.2020

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	1309541	1	1
Spain	Europe	223578	2	1
Italy	Europe	218268	3	2
United_Kingdom	Europe	215260	4	3
Brazil	America	155939	7	2
Turkey	Asia	137115	9	1
Iran	Asia	106220	10	2
China	Asia	83991	11	3
Canada	America	67702	12	3
South_Africa	Africa	9420	44	1
Egypt	Africa	8964	45	2
Australia	Oceania	6929	50	1
Morocco	Africa	5910	52	3
New_Zealand	Oceania	1144	90	2
Guam	Oceania	151	147	3

Datenstand: 10.5.2020

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
```

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	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 der letzten Woche

```
SELECT day,month, cases, avg (cases) over w1 as avgCasesWeek
FROM covid.`cov-cases` where country="Germany"
window w1 as (order by month,day rows 6 preceding);
```

day	month	Cases	avgCasesWeek
5	4	5,936	5,595
6	4	3,677	5,442
7	4	3,834	5,330
8	4	4,003	5,123
9	4	4,974	4,954
10	4	5,323	4,833
11	4	4,133	4,554
12	4	2,821	4,109
13	4	2,537	3,946
14	4	2,082	3,696
15	4	2,486	3,479
16	4	2,866	3,178
17	4	3,380	2,901
18	4	3,609	2,826
19	4	2,458	2,774
20	4	1,775	2,665
21	4	1,785	2,623
22	4	2,237	2,587
23	4	2,352	2,514
24	4	2,337	2,365
25	4	2,055	2,143
26	4	1,737	2,040
27	4	1,018	1,932
28	4	1,144	1,840
29	4	1,304	1,707
30	4	1,478	1,582
1	5	0	1,248
2	5	2,584	1,324
3	5	793	1,189
4	5	679	1,140
5	5	685	1,075
6	5	1,037	1,037
7	5	1,194	996

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~~	*15/05/2019*
John	*M12*	*15/05/2019*	*31/12/9999*	
Tracy	K25	15/11/2017	~~31/12/9999~~	*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, …)