Thomas Studer
Relationale Datenbanken:
Von den theoretischen Grundlagen zu Anwendungen mit PostgreSQL
Springer, 2016
ISBN 978-3-662-46570-7

Datenbanken

—

SQL

Thomas Studer

Institut für Informatik
Universität Bern

CREATE TABLE

```
CREATE TABLE TabellenName (
                Attribut_1 Domäne_1,
                Attribut_2 Domäne_2,
                Attribut_3 Domäne_3 )
```

Damit wird eine leere Instanz über dem Schema

$$(\texttt{Attribut\_1} : \texttt{Domäne\_1},\ \texttt{Attribut\_2} : \texttt{Domäne\_2},\ \texttt{Attribut\_3} : \texttt{Domäne\_3})$$

erzeugt. Dieser Instanz wird der Name `TabellenName` gegeben.

Beispiel

```
CREATE TABLE Personen (
    PersNr INTEGER,
    Name VARCHAR(30),
    GebDatum INTEGER )
```

INSERT

```
INSERT INTO TabellenName
  VALUES (Wert_1 , Wert_2, Wert_3);
```

Dabei wird das Tupel

```
( Wert_1, Wert_2, Wert_3 )
```

in die Tabelle `TabellenName` eingefügt.

Beispiele

```
INSERT INTO  Personen VALUES (1, 'Tom', 19720404)
```

```
INSERT INTO  Personen VALUES (2, 'Eva', null)
```

SELECT

```
SELECT * FROM Personen
```

Dieses Statement liefert folgende Tabelle (nach dem Ausführen der vorherigen beiden `INSERT` Statements):

PersNr	Name	GebDatum
1	Tom	19720404
2	Eva	Null

Gross-/Kleinschreibung

PostgreSQL setzt alle Identifier (Namen von Tabellen, Attributen, etc.) automatisch in Kleinschreibung. Mit dem `CREATE TABLE` Statement weiter oben wird also intern eine Tabelle `personen` erzeugt, welche die Attribute `persnr`, `name` und `gebdatum` enthält.

Die obige Abfrage wird also übersetzt zu

```
SELECT * FROM personen
```

DROP TABLE

```
DROP TABLE Personen
```

SELECT

$$\begin{aligned}&\texttt{SELECT } \mathtt{A}_1\texttt{, } \mathtt{A}_2\texttt{, } \ldots\texttt{, } \mathtt{A}_m\\&\texttt{FROM } \mathtt{R}_1\texttt{, } \mathtt{R}_2\texttt{, } \ldots\texttt{, } \mathtt{R}_n\\&\texttt{WHERE } \Theta\end{aligned}$$

Diese Abfrage ist äquivalent zum Ausdruck

$$\pi_{\mathtt{A}_1,\ldots,\mathtt{A}_m}(\sigma_\Theta(\mathtt{R}_1 \times \cdots \times \mathtt{R}_n))$$

der relationalen Algebra.
Wir nennen diesen Ausdruck die *kanonische Übersetzung* der SQL Abfrage.
Dabei arbeitet SQL mit Multimengen und nicht mit gewöhnlichen Mengen.

Beispiel

Autos

Marke	Farbe	Baujahr	FahrerId
Opel	silber	2010	1
Opel	schwarz	2010	2
VW	rot	2014	2
Audi	schwarz	2014	3

```
SELECT Marke, Baujahr
FROM Autos
```

Marke	Baujahr
Opel	2010
Opel	2010
VW	2014
Audi	2014

SELECT DISTINCT

Autos

Marke	Farbe	Baujahr	FahrerId
Opel	silber	2010	1
Opel	schwarz	2010	2
VW	rot	2014	2
Audi	schwarz	2014	3

```
SELECT DISTINCT Marke, Baujahr
FROM Autos
```

Marke	Baujahr
Opel	2010
VW	2014
Audi	2014

Arithmetischer Ausdruck

Die SELECT Klausel kann auch arithmetische Ausdrücke mit Konstanten und den Operatoren $+$, $-$, $*$ und $/$ enthalten. Zum Beispiel liefert die SQL-Anfrage

```
SELECT DISTINCT Marke, Baujahr+1
FROM Autos
```

die Tabelle

Marke	Baujahr
Opel	2011
VW	2015
Audi	2015

SELECT *

In einer `SELECT` Klausel bedeutet das Symbol $*$ anstelle der Liste mit Attributnamen *alle Attribute*. Daher liefert

```
SELECT *
FROM Autos
```

die Tabelle

Marke	Farbe	Baujahr	FahrerId
Opel	silber	2010	1
Opel	schwarz	2010	2
VW	rot	2014	2
Audi	schwarz	2014	3

WHERE

Prädikate sind aufgebaut aus
Vergleichsoperatoren: $<$, $<=$, $>$, $>=$, $=$ und $<>$
Schlüsselwörter: `NOT`, `OR` und `AND`

Mit der Abfrage

```
SELECT *
FROM Autos
WHERE Marke = 'OPEL' AND Baujahr > 2010
```

erhalten wir alle Autos der Marke `OPEL`, welche später als 2010 gebaut worden sind.

BETWEEN

```
SELECT *
FROM Autos
WHERE Marke = 'OPEL' AND
  Baujahr BETWEEN 2000 AND 2010
```

ist äquivalent zu

```
SELECT *
FROM Autos
WHERE Marke = 'OPEL' AND
  Baujahr >= 2000 AND Baujahr <= 2010
```

LIKE

Personen

Id	Name
1	Tom
2	Eva
3	-
4	Tim
5	
6	Thom

```
SELECT Id
FROM Personen
WHERE Name LIKE 'T%'
```

liefert die `Id`s aller Personen, deren Name mit `T` beginnt: 1, 4 und 6.

LIKE 2

Personen

Id	Name
1	Tom
2	Eva
3	-
4	Tim
5	
6	Thom

```
SELECT Id
FROM Personen
WHERE Name LIKE 'T_m'
```

liefert die `Ids` aller Personen, deren Namen mit `T` beginnt, dann genau ein Zeichen enthält und dann mit `m` endet: 1 und 4.

LIKE 3

Personen

Id	Name
1	Tom
2	Eva
3	-
4	Tim
5	
6	Thom

```
SELECT Id
FROM Personen
WHERE Name LIKE '%'
```

liefert die `Id`s aller Personen, die einen beliebigen (ev. leeren) Namen haben: 1, 2, 4, 5 und 6.

IS NULL

Personen

Id	Name
1	Tom
2	Eva
3	-
4	Tim
5	
6	Thom

```
SELECT Id
FROM Personen
WHERE Name IS NULL
```

gibt die `Id`s derjenigen Personen, deren Name nicht bekannt ist: 3.

IS NOT NULL

Personen

Id	Name
1	Tom
2	Eva
3	-
4	Tim
5	
6	Thom

```
SELECT Id
FROM Personen
WHERE Name IS NOT NULL
```

gibt die `Id`s derjenigen Personen, deren Name bekannt ist (es kann auch der leere String sein): 1, 2, 4, 5, 6.

IS NOT NULL

Bemerkung

Tests auf `IS NOT NULL` können üblicherweise nicht effizient durchgeführt werden.

FROM

Autos

Marke	Farbe	Baujahr	FahrerId
Opel	silber	2010	1
Opel	schwarz	2010	2
VW	rot	2014	2
Audi	schwarz	2014	3

Personen

PersId	Vorname	Nachname
1	Tom	Studer
2	Eva	Studer
3	Eva	Meier

```
SELECT Vorname, Nachname, Marke
FROM Autos, Personen
WHERE FahrerId = PersId
```

Vorname	Nachname	Marke
Tom	Studer	Opel
Eva	Studer	Opel
Eva	Studer	VW
Eva	Meier	Audi

Mehrfache Attribute

Autos

Marke	Jahrgang	PersId
Opel	2010	1
VW	1990	1
Audi	2014	-
Skoda	2014	2

Personen

PersId	Name
1	Studer
2	Meier

```
SELECT DISTINCT Autos.PersId, Marke, Jahrgang, Name
FROM Autos, Personen
WHERE Autos.PersId = Personen.PersId
```

AS - Attribute

R

a	b
2	4
3	5

```
SELECT 1 AS Konstante, a AS Argument, b, a+b AS Summe
FROM R
```

Konstante	Argument	b	Summe
1	2	4	6
1	3	5	8

Es gilt zu beachten, dass die Umbenennung erst am Schluss erfolgt.

AS - Relationen

VaterSohn

Vater	Sohn
Bob	Tom
Bob	Tim
Tim	Rob
Tom	Ted
Tom	Rik
Ted	Nik

```
SELECT N1.Vater AS Grossvater,
       N2.Sohn AS Enkel
FROM VaterSohn AS N1,
     VaterSohn AS N2
WHERE N1.Sohn = N2.Vater
```

Grossvater	Enkel
Tom	Nik
Bob	Ted
Bob	Rik
Bob	Rob

UNION

```
SELECT 1 AS Gen,
    Vater AS Vorfahre,
    Sohn AS Nachfolger
FROM VaterSohn

UNION

SELECT 2, N1.Vater ,
    N2.Sohn
FROM VaterSohn AS N1,
    VaterSohn AS N2
WHERE N1.Sohn = N2.Vater
```

Gen	Vorfahre	Nachfolger
1	Bob	Tom
1	Bob	Tim
1	Tim	Rob
1	Tom	Ted
1	Tom	Rik
1	Ted	Nik
2	Tom	Nik
2	Bob	Ted
2	Bob	Rik
2	Bob	Rob

Keine Duplikate

U1
A
1
1
2

U2
A
1
3

```
  SELECT A FROM U1
UNION
  SELECT A FROM U2
```

A
1
2
3

UNION ALL

U1
A
1
1
2

U2
A
1
3

```
  SELECT A FROM U1
UNION ALL
  SELECT A FROM U2
```

A
1
1
2
1
3

ORDER BY

Test

Id	Name
1	Tom
2	-
3	Eva

```
SELECT *
FROM Test
ORDER BY Name ASC
```

Id	Name
3	Eva
1	Tom
2	-

Mit dem Schlüsselwort `ORDER BY · DESC` können wir auch absteigend sortieren. In diesem Fall kommen die `Null`-Werte zuerst.

Subqueries in WHERE Klauseln

Sei θ eine Vergleichsoperation und S eine vollständige Query / eine Relation.

$$
\begin{aligned}
A \texttt{ IN } S &: A \in S \\
A \texttt{ NOT IN } S &: A \notin S \\
A\ \theta \texttt{ ANY } S &: \exists x(x \in S \wedge A\theta x) \\
A\ \theta \texttt{ ALL } S &: \forall x(x \in S \rightarrow A\theta x)\ .
\end{aligned}
$$

Beispiel

Autos

Marke	Farbe	Baujahr	FahrerId
Opel	silber	2010	1
Opel	schwarz	2010	2
VW	rot	2014	2
Audi	schwarz	2014	3

Personen

PersId	Vorname	Nachname
1	Tom	Studer
2	Eva	Studer
3	Eva	Meier

Finde diejenigen Personen, die kein Auto haben:

```
SELECT *
FROM Personen
WHERE PersId NOT IN (
  SELECT FahrerId
  FROM Autos)
```

Beispiel

Autos

Marke	Farbe	Baujahr	FahrerId
Opel	silber	2010	1
Opel	schwarz	2010	2
VW	rot	2014	2
Audi	schwarz	2014	3

Finde die ältesten Autos:

```
SELECT *
FROM Autos
WHERE Baujahr <= ALL (
  SELECT Baujahr
  FROM Autos)
```

Korrelierte Subquery

Personen

PersId	Vorname	Nachname
1	Tom	Studer
2	Eva	Studer
3	Eva	Meier

Finde alle Personen, zu denen es eine zweite Person mit demselben Vornamen gibt:

```
SELECT *
FROM Personen
WHERE EXISTS (
  SELECT P2.*
  FROM Personen AS P2
  WHERE Personen.Vorname = P2.Vorname AND
        Personen.PersId <> P2.PersId)
```

All-Aussagen

Um Allaussagen zu behandeln, nützen wir die logische Äquivalenz

$$\forall x \varphi(x) \quad \Longleftrightarrow \quad \neg \exists x \neg \varphi(x)$$

aus und arbeiten mit dem Prädikat `NOT EXISTS`, das zur Negation von Existenzaussagen dient.

Gesucht wurden diejenigen Mechaniker, welche *alle* Automarken, die in der Garage vorkommen, reparieren können. Dazu müssen wir die Division

$$\texttt{Mechaniker} \div \texttt{Garage}$$

berechnen.

Division

$$(R \div S) = \pi_{A_{i_1},\ldots,A_{i_{m-n}}}(R) \setminus \pi_{A_{i_1},\ldots,A_{i_{m-n}}}((\pi_{A_{i_1},\ldots,A_{i_{m-n}}}(R) \times S) \setminus R)$$

Finde alle Mechaniker,

für die es keine Automarke in der Garage gibt,

die sie nicht reparieren können.

```
SELECT DISTINCT Name
FROM Mechaniker AS m1
WHERE NOT EXISTS (
  SELECT *
  FROM Garage
  WHERE NOT EXISTS (
    SELECT *
    FROM Mechaniker AS m2
    WHERE (m1.Name = m2.Name) AND
          (m2.Marke = Garage.Marke)))
```

Beispiel

Mechaniker

Name	Marke
Studer	Opel
Meier	Opel
Meier	VW
Meier	Audi

Garage

Marke
Opel
Audi

```
SELECT DISTINCT Name FROM Mechaniker AS m1
WHERE NOT EXISTS (
  SELECT * FROM Garage
  WHERE NOT EXISTS (
    SELECT *  FROM Mechaniker AS m2
    WHERE (m1.Name = m2.Name) AND (m2.Marke = Garage.Marke)))
```

Natural Join

Links

A	B
a1	b1
a2	b2

Rechts

B	C
b1	c1
b3	c3

```
SELECT A, B, C
FROM Links NATURAL JOIN Rechts
```

A	B	C
a1	b1	c1

Inner Join

Links

A	B
a1	b1
a2	b2

Rechts

B	C
b1	c1
b3	c3

```
SELECT A, Links.B, C
FROM Links INNER JOIN Rechts ON Links.B = Rechts.B
```

A	B	C
a1	b1	c1

Inner Join entspricht Θ-Join. Das Join-Kriterium kann explizit angegeben werden.

USING

```
SELECT A, Links.B, C
FROM Links INNER JOIN Rechts ON Links.B = Rechts.B
```

ist äquivalent zu

```
SELECT A, B, C
FROM Links INNER JOIN Rechts USING (B)
```

Left Outer Join

Links

A	B
a1	b1
a2	b2

Rechts

B	C
b1	c1
b3	c3

```
SELECT A, B, C
FROM Links LEFT OUTER JOIN Rechts USING (B)
```

A	B	C
a1	b1	c1
a2	b2	-

Right Outer Join

Links

A	B
a1	b1
a2	b2

Rechts

B	C
b1	c1
b3	c3

```
SELECT A, B, C
FROM Links RIGHT OUTER JOIN Rechts USING (B)
```

A	B	C
a1	b1	c1
-	b3	c3

Full Outer Join

Links

A	B
a1	b1
a2	b2

Rechts

B	C
b1	c1
b3	c3

```
SELECT A, B, C
FROM Links FULL OUTER JOIN Rechts USING (B)
```

A	B	C
a1	b1	c1
a2	b2	-
-	b3	c3

GROUP BY

Aggregatsfunktionen: `COUNT`, `SUM`, `MIN`, `MAX`, `AVG`

Gesucht: Durchschnittslänge der Filme pro Jahr

```
SELECT Jahr, AVG(Dauer)
FROM Filme
GROUP BY Jahr
```

Bei Abfragen mit einer `GROUP BY` Klausel gilt es zu beachten, dass in der `SELECT` Klausel nur Attribute vorkommen dürfen, welche entweder in der `GROUP BY` Klausel vorkommen oder durch eine Aggregatsfunktion berechnet werden.

COUNT(*)

Die Funktion `COUNT(*)` bestimmt die Anzahl Elemente jeder Gruppe. Dabei werden auch Tupel mitgezählt, die `Null`-Werte enthalten. Mit der Abfrage

```
SELECT Jahr, COUNT(*)
FROM Filme
GROUP BY Jahr
```

können wir somit für jedes Jahr die Anzahl Filme bestimmen.

GROUP BY mit WHERE

Filme

FId	Jahr	Dauer
1	2010	110
2	2012	90
3	2012	120
4	2010	100
5	2012	120
6	2011	95
7	2010	12
8	-	140
9	2012	16
10	-	130

```
SELECT Jahr, AVG(Dauer)
FROM Filme
WHERE Dauer >= 30
GROUP BY Jahr
```

Jahr	AVG(Dauer)
2010	105
2011	95
2012	110
-	135

Kurzfilme werden nicht in die Gruppen aufgenommen.
Filme, bei denen das Attribut `Jahr` den Wert `Null` hat, bilden eine Gruppe.

GROUP BY mit HAVING

Filme

FId	Jahr	Dauer
1	2010	110
2	2012	90
3	2012	120
4	2010	100
5	2012	120
6	2011	95
7	2010	12
8	-	140
9	2012	16
10	-	130

```
SELECT Jahr, AVG(Dauer)
FROM Filme
WHERE Dauer >= 30
GROUP BY Jahr
HAVING COUNT(*)>1
  AND Jahr IS NOT NULL
```

Jahr	AVG(Dauer)
2010	105
2012	110

Wir verlangen, dass keine Jahre ins Resultat aufgenommen werden, in denen nur *ein* Film erschienen ist oder bei denen das Jahr unbekannt ist.

Allgemeine Form

$$\texttt{SELECT } A_1, A_2, \ldots, A_m, \texttt{AGG}_1(A_{m+1}), \ldots, \texttt{AGG}_k(A_{m+k})$$
$$\texttt{FROM } R_1, R_2, \ldots, R_n$$
$$\texttt{WHERE } \Theta_1$$
$$\texttt{GROUP BY } A_{r_1}, \ldots, A_{r_s}$$
$$\texttt{HAVING } \Theta_2$$
$$\texttt{ORDER BY } \ldots$$

Dabei muss gelten: $\{A_1, A_2, \ldots, A_m\} \subseteq \{A_{r_1}, \ldots, A_{r_s}\}$

Sequentielle Abarbeitung:

1. Bedingung Θ_1 der `WHERE` Klausel auswerten.
2. Gruppierung gemäss `GROUP BY` auswerten.
3. Bedingung Θ_2 der `HAVING` Klausel auswerten.
4. Ergebnis in der Sortierfolge gemäss `ORDER BY` ausgeben.

Aggregatsfunktion ohne Gruppierung

Gesucht: Laufzeit des kürzesten Films

Filme

FId	Jahr	Dauer
1	2010	110
2	2012	90
3	2012	120
4	2010	100
5	2012	120
6	2011	95
7	2010	12
8	-	140
9	2012	16
10	-	130

```
SELECT MIN(Dauer)
FROM Filme
```

Das Resultat ist 12.

Aggregatsfunktion ohne Gruppierung 2

Gesucht: Film-Id und Laufzeit des kürzesten Films

Filme

FId	Jahr	Dauer
1	2010	110
2	2012	90
3	2012	120
4	2010	100
5	2012	120
6	2011	95
7	2010	12
8	-	140
9	2012	16
10	-	130

```
SELECT FId, Dauer
FROM Filme
WHERE Dauer = ( SELECT MIN(Dauer)
                FROM Filme )
```

COUNT und NULL

Filme

FId	Jahr	Dauer
1	2010	110
2	2012	90
3	2012	120
4	2010	100
5	2012	120
6	2011	95
7	2010	12
8	-	140
9	2012	16
10	-	130

```
SELECT COUNT(Jahr)
FROM Filme
```

liefert das Resultat 8.

COUNT DISTINCT

Filme

FId	Jahr	Dauer
1	2010	110
2	2012	90
3	2012	120
4	2010	100
5	2012	120
6	2011	95
7	2010	12
8	-	140
9	2012	16
10	-	130

```
SELECT COUNT(DISTINCT Jahr)
FROM Filme
```

liefert das Resultat 3.

Gruppe mit NULL mitzählen

Filme

FId	Jahr	Dauer
1	2010	110
2	2012	90
3	2012	120
4	2010	100
5	2012	120
6	2011	95
7	2010	12
8	-	140
9	2012	16
10	-	130

```
SELECT COUNT(*)
FROM (
  SELECT Jahr
  FROM Filme
  GROUP BY Jahr ) AS G
```

liefert das Resultat 4.

Division mit COUNT

Mechaniker

Name	Marke
Studer	Opel
Meier	Opel
Meier	VW
Meier	Audi

Garage

Marke
Opel
Audi

```
SELECT DISTINCT Name
FROM Mechaniker, Garage
WHERE Mechaniker.Marke = Garage.Marke
GROUP BY Name
HAVING COUNT(Mechaniker.Marke) = ( SELECT COUNT(Marke)
                                   FROM Garage )
```

Vergleich

```
SELECT DISTINCT Name FROM Mechaniker AS m1
WHERE NOT EXISTS (
  SELECT * FROM Garage
  WHERE NOT EXISTS (
    SELECT *  FROM Mechaniker AS m2
    WHERE (m1.Name = m2.Name) AND (m2.Marke = Garage.Marke)))
```

```
SELECT DISTINCT Name
FROM Mechaniker, Garage
WHERE Mechaniker.Marke = Garage.Marke
GROUP BY Name
HAVING COUNT(Mechaniker.Marke) = ( SELECT COUNT(Marke)
                                   FROM Garage )
```

Was passiert, wenn `Garage` leer ist?

Die erste Abfrage liefert in diesem Fall alle Mechaniker als Resultat.
Hingegen liefert die zweite Abfrage die leere Menge als Resultat.

Ranglisten

Gesucht: Die zwei kürzesten Filme

Filme

FId	Jahr	Dauer
6	2011	95
7	2010	12
8	-	140
9	2012	16
10	-	130
11	2011	16

```
SELECT *
FROM Filme
WHERE 2 >
  ( SELECT count(*)
    FROM Filme AS T
    WHERE T.Dauer<Filme.Dauer )
```

FId	Jahr	Dauer
7	2010	12
9	2012	16
11	2011	16

LIMIT

Gesucht: Die zwei kürzesten Filme

Filme

FId	Jahr	Dauer
6	2011	95
7	2010	12
8	-	140
9	2012	16
10	-	130
11	2011	16

```
SELECT *
FROM Filme
ORDER BY Dauer ASC
LIMIT 2
```

FId	Jahr	Dauer
7	2010	12
9	2012	16

Die ersten n-Tupel, welche die Abfrage liefert, werden in die Resultat-Tabelle aufgenommen.

WITH - Auslagern von Subqueries

```
SELECT COUNT(*)
FROM (
  SELECT Jahr
  FROM Filme
  GROUP BY Jahr ) AS G
```

```
WITH G AS (
  SELECT Jahr
  FROM FILME
  GROUP BY Jahr )
SELECT COUNT(*)
FROM G
```

WITH RECURSIVE

VaterSohn

Vater	Sohn
Bob	Tom
Bob	Tim
Tim	Rob
Tom	Ted
Tom	Rik
Ted	Nik

```
WITH RECURSIVE t(v,n) AS (
    SELECT Vater, Sohn
    FROM VaterSohn
  UNION ALL
    SELECT t.v, VaterSohn.Sohn
    FROM t INNER JOIN
      VaterSohn ON t.n = VaterSohn.Vater
  )
SELECT v AS Vorfahre, n AS Nachfolger
FROM t
```

Resultat

Vorfahre	Nachfolger
Bob	Tom
Bob	Tim
Tim	Rob
Tom	Ted
Tom	Rik
Ted	Nik
Bob	Rik
Bob	Ted
Bob	Rob
Tom	Nik
Bob	Nik

Window Funktionen

Wein

WId	Sorte	Jahr	Menge	Bewertung
1	weiss	2014	20000	-
2	rot	2014	7000	-
3	weiss	2013	18000	7
4	rot	2013	9000	6
5	weiss	2012	18000	8
6	rot	2012	5000	10
7	weiss	2011	14000	5
8	rot	2011	8000	6
9	weiss	2010	19000	7
10	rot	2010	6000	5

Rangliste nach Bewertung

```
SELECT
  Wid,
  Bewertung,
  RANK() OVER
    (ORDER BY Bewertung DESC NULLS LAST)
    AS Rang
FROM Wein
```

WId	Bewertung	Rang
6	10	1
5	8	2
9	7	3
3	7	3
8	6	5
4	6	5
10	5	7
7	5	7
2	-	9
1	-	9

Rangliste nach Bewertung mit DENSE_RANK

```
SELECT
  Wid,
  Bewertung,
  DENSE_RANK() OVER
    (ORDER BY Bewertung DESC NULLS LAST)
    AS Rang
FROM Wein
```

WId	Bewertung	Rang
6	10	1
5	8	2
9	7	3
3	7	3
8	6	4
4	6	4
10	5	5
7	5	5
2	-	6
1	-	6

Rang pro Sorte

```
SELECT
  Wid,
  Sorte,
  Bewertung,
  RANK() OVER (
    PARTITION BY Sorte
    ORDER BY Bewertung
       DESC NULLS LAST)
    AS Rang
FROM Wein
```

WId	Sorte	Bewertung	Rang
6	rot	10	1
8	rot	6	2
4	rot	6	2
10	rot	5	4
2	rot	-	5
5	weiss	8	1
9	weiss	7	2
3	weiss	7	2
7	weiss	5	4
1	weiss	-	5

Summen bilden

```
SELECT
  Jahr,
  Sorte,
  SUM(Menge) OVER (
    PARTITION BY Sorte
    ORDER BY Jahr ASC
    ROWS UNBOUNDED PRECEDING)
    AS Summe
FROM Wein
```

Jahr	Sorte	Summe
2010	rot	6000
2011	rot	14000
2012	rot	19000
2013	rot	28000
2014	rot	35000
2010	weiss	19000
2011	weiss	33000
2012	weiss	51000
2013	weiss	69000
2014	weiss	89000

Durchschnitt der letzten drei Jahre

```
SELECT
  Jahr,
  Sorte,
  AVG(Bewertung)
    OVER (PARTITION BY Sorte
    ORDER BY Jahr ASC
    ROWS 2 PRECEDING)
    AS Schnitt
FROM Wein
```

Jahr	Sorte	Schnitt
2010	rot	5
2011	rot	5.5
2012	rot	7
2013	rot	7.333
2014	rot	8
2010	weiss	7
2011	weiss	6
2012	weiss	6.666
2013	weiss	6.666
2014	weiss	7.5

Rangliste der Jahrgänge

```
SELECT
 Jahr,
 AVG(Bewertung),
 RANK() OVER (
   ORDER BY AVG(Bewertung)
      DESC NULLS LAST)
   AS Rang
FROM Wein
GROUP BY Jahr
```

Jahr	Avg	Rang
2012	9	1
2013	6.5	2
2010	6	3
2011	5.5	4
2014	-	5

Kombiniert

```
SELECT
 Jahr,
 AVG(AVG(Bewertung)) OVER (
   ORDER BY Jahr ASC
   ROWS 2 PRECEDING )
   AS Schnitt
FROM Wein
GROUP BY Jahr
```

Jahr	Schnitt
2010	6
2011	5.75
2012	6.833
2013	7
2014	7.75