

# SQL Fensterfunktionen Spickzettel

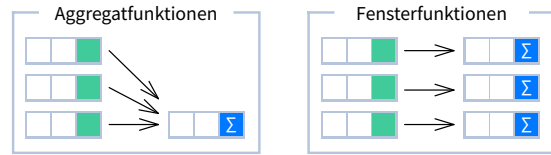
## FENSTERFUNKTIONEN

**Fensterfunktionen** berechnen ihr Ergebnis auf der Grundlage eines gleitenden Fensterrahmens, einer Reihe von Zeilen, die in irgendeiner Weise mit der aktuellen Zeile verbunden sind.



## AGGREGATFUNKTIONEN VS. FENSTERFUNKTIONEN

Im Gegensatz zu Aggregatfunktionen werden bei Fensterfunktionen die Zeilen nicht zusammenfasst.



## SYNTAX

```
SELECT Stadt, Monat,
SUM(Verkauft) OVER (
PARTITION BY Stadt
ORDER BY Monat
RANGE UNBOUNDED PRECEDING) Gesamt
FROM Verkauf;
```

```
SELECT <Spalte_1>, <Spalte_2>,
<Fensterfunktion> OVER (
PARTITION BY <...>
ORDER BY <...>
<Fensterrahmen>) <Fenster_Spalte_Alias>
FROM <Tabellenname>;
```

## DEFINITION BENANNTER FENSTER

```
SELECT Land, Stadt,
RANK() OVER Durchschnitt_Land_Verkauft
FROM Verkauf
WHERE Monat BETWEEN 1 AND 6
GROUP BY Land, Stadt
HAVING sum(Verkauft) > 10000
WINDOW Durchschnitt_Land_Verkauft AS (
PARTITION BY Land
ORDER BY avg(Verkauft) DESC)
ORDER BY Land, Stadt;
```

```
SELECT <Spalte_1>, <Spalte_2>,
<Fensterfunktion>() OVER <Fenstername>
FROM <Tabellenname>
WHERE <...>
GROUP BY <...>
HAVING <...>
WINDOW <Fenstername> AS (
PARTITION BY <...>
ORDER BY <...>
<window_frame>)
ORDER BY <...>;
```

PARTITION BY, ORDER BY und die Definition des Fensterrahmens sind alle optional.

## LOGISCHE REIHENFOLGE DER OPERATIONEN IN SQL

1. FROM, JOIN
2. WHERE
3. GROUP BY
4. Aggregatfunktionen
5. HAVING
6. **Fensterfunktionen**
7. SELECT
8. DISTINCT
9. UNION/INTERSECT/EXCEPT
10. ORDER BY
11. OFFSET
12. LIMIT/FETCH/TOP

Sie können Fensterfunktionen in SELECT und ORDER BY verwenden. Sie können jedoch keine Fensterfunktionen in den Klauseln FROM, WHERE, GROUP BY oder HAVING einfügen.

## PARTITION BY

unterteilt Zeilen in mehrere Gruppen, die **Partitionen** genannt werden und auf die die Fensterfunktion angewendet wird.

PARTITION BY Stadt			PARTITION BY Stadt			
Monat	Stadt	Verkauft	Monat	Stadt	Verkauft	sum
1	Rom	200	1	Paris	300	800
2	Paris	500	2	Paris	500	800
1	London	100	1	Rom	200	900
1	Paris	300	2	Rom	300	900
2	Rom	300	3	Rom	400	900
2	London	400	1	London	100	500
3	Rom	400	2	London	400	500

**Standard-Partition:** Ohne PARTITION BY-Klausel ist die gesamte Ergebnismenge die Partition.

## ORDER BY

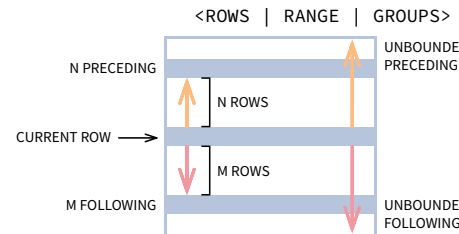
gibt die Reihenfolge der Zeilen in jeder Partition an, auf die die Fensterfunktion angewendet wird.

PARTITION BY Stadt			PARTITION BY Stadt		
Verkauft	Stadt	Monat	Verkauft	Stadt	Monat
200	Rom	1	300	Paris	1
500	Paris	2	500	Paris	2
100	London	1	200	Rom	1
300	Paris	1	300	Rom	2
300	Rom	2	400	Rom	3
400	London	2	100	London	1
400	Rom	3	400	London	2

**Standard ORDER BY:** Ohne ORDER BY Klausel ist die Reihenfolge der Zeilen innerhalb jeder Partition willkürlich.

## FENSTERRAHMEN

Ein **Fensterrahmen** ist ein Satz von Zeilen, die in irgendeiner Weise mit der aktuellen Zeile zusammenhängen. Der Fensterrahmen wird in jeder Partition separat ausgewertet.



Die Grenzen können eine der fünf Optionen sein:

- UNBOUNDED PRECEDING
- n PRECEDING
- CURRENT ROW
- n FOLLOWING
- UNBOUNDED FOLLOWING

Die Untergrenze muss VOR der Obergrenze stehen.

**ROWS BETWEEN 1 PRECEDING AND 1 FOLLOWING**

Stadt	Verkauft	Monat
Paris	300	1
Rom	200	1
Paris	500	2
Rom	100	4
Paris	200	4
Paris	300	5
Rom	200	5
London	200	5
London	100	6
Rom	300	6

1 Zeile vor der aktuellen Zeile und 1 Zeile nach der aktuellen Zeile

**RANGE BETWEEN 1 PRECEDING AND 1 FOLLOWING**

Stadt	Verkauft	Monat
Paris	300	1
Rom	200	1
Paris	500	2
Rom	100	4
Paris	200	4
Paris	300	5
Rom	200	5
London	200	5
London	100	6
Rom	300	6

Werte im Bereich zwischen 3 und 5 ORDER BY muss einen einzigen Ausdruck enthalten

**GROUPS BETWEEN 1 PRECEDING AND 1 FOLLOWING**

Stadt	Verkauft	Monat
Paris	300	1
Rom	200	1
Paris	500	2
Rom	100	4
Paris	200	4
Paris	300	5
Rom	200	5
London	200	5
London	100	6
Rom	300	6

1 Gruppe vor der aktuellen Zeile und 1 Gruppe nach der aktuellen Zeile, unabhängig vom Wert

Stand 2024 wird GROUPS nur in PostgreSQL 11 und höher unterstützt.

## ABKÜRZUNGEN

ABKÜRZUNG	BEDEUTUNG
UNBOUNDED PRECEDING	BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW
n PRECEDING	BETWEEN n PRECEDING AND CURRENT ROW
CURRENT ROW	BETWEEN CURRENT ROW AND CURRENT ROW
n FOLLOWING	BETWEEN CURRENT ROW AND n FOLLOWING
UNBOUNDED FOLLOWING	BETWEEN CURRENT ROW AND UNBOUNDED FOLLOWING

## STANDARD-FENSTERRAHMEN

Wenn ORDER BY angegeben ist, lautet der Rahmen RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW.

Ohne ORDER BY lautet die Rahmenangabe ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING.

## LISTE DER FENSTERFUNKTIONEN

### Aggregat-Funktionen

- avg()
- count()
- max()
- min()
- sum()

### Ranking-Funktionen

- row\_number()
- rank()
- dense\_rank()

### Verteilungs-Funktionen

- percent\_rank()
- cume\_dist()

### Analytische Funktionen

- lead()
- lag()
- ntile()
- first\_value()
- last\_value()
- nth\_value()

## AGGREGATFUNKTIONEN

- **avg**(expr) – Durchschnittswert für Zeilen innerhalb des Fensterrahmens
- **count**(expr) – Anzahl der Werte für Zeilen innerhalb des Fensterrahmens
- **max**(expr) – Maximalwert innerhalb des Fensterrahmens
- **min**(expr) – Mindestwert innerhalb des Fensterrahmens
- **sum**(expr) – Summe der Werte innerhalb des Fensterrahmens

**ORDER BY und Fensterrahmen:** Für die Aggregatfunktionen ist kein ORDER BY erforderlich. Sie akzeptieren die Definition eines Fensterrahmens (ROWS, RANGE, GROUPS).

## RANKING-FUNKTIONEN

- **row\_number**() – eindeutige Nummer für jede Zeile innerhalb der Partition, mit unterschiedlichen Nummern für gleiche Werte
- **rank**() – Rangfolge innerhalb der Partition, mit Lücken und gleicher Rangfolge für gleiche Werte
- **dense\_rank**() – Rangfolge innerhalb der Partition, ohne Lücken und mit gleicher Rangfolge für gleiche Werte

Stadt	Preis	row_number	rank	dense_rank
Paris	7	1	1	1
Rom	7	2	1	1
London	8.5	3	3	2
Berlin	8.5	4	3	2
Moskau	9	5	5	3
Madrid	10	6	6	4
Oslo	10	7	6	4

**ORDER BY und Fensterrahmen:** rank() und dense\_rank() erfordern ORDER BY, aber row\_number() erfordert kein ORDER BY. Ranking-Funktionen akzeptieren keine Fensterrahmen-Definition (ROWS, RANGE, GROUPS).

## ANALYTISCHE FUNKTIONEN

- **Lead**(expr, offset, default) – der Wert für den Zeilen-Offset: Zeilen nach der aktuellen Zeile; offset und default sind optional; Default-Werte: offset = 1, default = NULL
- **lag**(expr, offset, default) – der Wert für den Zeilen-Offset: Zeilen vor der aktuellen Zeile; offset und default sind optional; Default-Werte: offset = 1, default = NULL

Lead(Verkauf) OVER(ORDER BY Monat)

Monat	Verkauf	lead
1	500	300
2	300	400
3	400	100
4	100	500
5	500	NULL

lag(Verkauf) OVER(ORDER BY Monat)

Monat	Verkauf	lag
1	500	NULL
2	300	500
3	400	300
4	100	400
5	500	100

Lead(Verkauf, 2, 0) OVER(ORDER BY Monat)

Monat	Verkauf	lead
1	500	400
2	300	100
3	400	500
4	100	0
5	500	0

lag(Verkauf, 2, 0) OVER(ORDER BY Monat)

Monat	Verkauf	lag
1	500	0
2	300	0
3	400	500
4	100	300
5	500	400

- **ntile**(n) – Zeilen innerhalb einer Partition so gleichmäßig wie möglich in n Gruppen aufteilen und jeder Zeile eine Gruppennummer zuweisen.

ntile(3)

Stadt	Verkauf	ntile
Rom	100	1
Paris	100	1
London	200	1
Moskau	200	2
Berlin	200	2
Madrid	300	2
Oslo	300	3
Dublin	300	3

**ORDER BY und Fensterrahmen:** ntile(), lead() und lag() erfordern ein ORDER BY. Sie akzeptieren keine Fensterrahmen-Definition (ROWS, RANGE, GROUPS).

## VERTEILUNGSFUNKTIONEN

- **percent\_rank**() – die Perzentil-Rangzahl einer Zeile – ein Wert im Intervall [0, 1] (Rang-1) / (Gesamtzahl der Zeilen - 1)
- **cume\_dist**() – die kumulative Verteilung eines Wertes innerhalb einer Gruppe von Werten, d.h. die Anzahl der Zeilen mit Werten, die kleiner oder gleich dem Wert der aktuellen Zeile sind, geteilt durch die Gesamtzahl der Zeilen; ein Wert im Intervall (0, 1]

percent\_rank() OVER(ORDER BY Verkauft)

Stadt	Verkauft	percent_rank
Paris	100	0
Berlin	150	0.25
Rom	200	0.5
Moskau	200	0.5
London	300	1

★ ohne diese Zeile sind 50% der Werte kleiner als der Wert dieser Zeile

cume\_dist() OVER(ORDER BY Verkauft)

Stadt	Verkauft	cume_dist
Paris	100	0.2
Berlin	150	0.4
Rom	200	0.8
Moskau	200	0.8
London	300	1

★ 80 % der Werte sind kleiner als oder gleich diesem Wert

**ORDER BY und Fensterrahmen:** Die Verteilungsfunktionen erfordern ORDER BY. Sie akzeptieren keine Fensterrahmen-Definition (ROWS, RANGE, GROUPS).

- **first\_value**(expr) – den Wert für die erste Zeile innerhalb des Fensterrahmens
- **last\_value**(expr) – der Wert für die letzte Zeile innerhalb des Fensterrahmens

first\_value(Verkauft) OVER (PARTITION BY Stadt ORDER BY Monat)

Stadt	Monat	Verkauf	first_value
Paris	1	500	500
Paris	2	300	500
Paris	3	400	500
Rom	2	200	200
Rom	3	300	200
Rom	4	500	200

last\_value(Verkauft) OVER (PARTITION BY Stadt ORDER BY Monat RANGE BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING)

Stadt	Monat	Verkauf	last_value
Paris	1	500	400
Paris	2	300	400
Paris	3	400	400
Rom	2	200	500
Rom	3	300	500
Rom	4	500	500

**Hinweis:** Normalerweise sollten Sie RANGE BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING mit last\_value() verwenden. Mit dem Standardfensterrahmen für ORDER BY, RANGE UNBOUNDED PRECEDING gibt last\_value() den Wert für die aktuelle Zeile zurück.

- **nth\_value**(expr, n) – der Wert für die n-te Zeile innerhalb des Fensterrahmens; n muss eine ganze Zahl sein

Stadt	Monat	Verkauf	nth_value
Paris	1	500	300
Paris	2	300	300
Paris	3	400	300
Rom	2	200	300
Rom	3	300	300
Rom	4	500	300
Rom	5	300	300
London	1	100	NULL

**ORDER BY und Fensterrahmen:** first\_value(), last\_value() und nth\_value() erfordern kein ORDER BY. Sie akzeptieren die Definition von Fensterrahmen (ROWS, RANGE, GROUPS).