stackfuel

SQL Codeübersicht

Hier findest du die wichtigsten Befehle des vierten Kapitels mit Beispielcode. Wenn du dir den Output jeder Query noch einmal anschauen willst, kannst du einen Blick in die Notebooks der Lektion werfen.

Analytische Funktionen/Fensterfunktionen

Die `OVER`-Klausel macht aus einer Aggregationsfunktion eine analytische Funktion, auch Fensterfunktion genannt. Mittels `PARTITION BY` kann bestimmt werden, auf welche Zeilen sich ein Fenster beziehen soll.

```
SELECT WEEK(rental_date) AS week,
       DAY(rental_date) AS day_of_month,
       COUNT(*) AS daily_rentals,
       MAX(COUNT(*)) OVER () AS best_day_overall,
       MAX(COUNT(*)) OVER (PARTITION BY WEEK(rental_date))
           AS best_day_per_week
FROM rental
WHERE MONTHNAME(rental_date) = "July"
GROUP BY week,
         weekday
ORDER BY week;
```

Rankings sind nicht-aggregierende analytische Funktionen. Sie erstellen Rangfolgen und benötigen für sinnvolle Ergebnisse eine `ORDER BY`-Klausel in der Fensterdefinition. Es gibt mehrere unterschiedliche Rankingfunktionen.

```
SELECT WEEK(rental_date) AS week,
       DAYNAME(rental_date) AS weekday,
       COUNT(*) AS daily_rentals,
       ROW_NUMBER() OVER(ORDER BY COUNT(*) DESC) AS rent_row_num,
       RANK()       OVER(ORDER BY COUNT(*) DESC) AS rent_rank,
       DENSE_RANK() OVER(ORDER BY COUNT(*) DESC) AS rent_dense_rank
FROM rental
WHERE MONTHNAME(rental_date) = "July"
GROUP BY week,
         weekday
ORDER BY week;
```

`WINDOW` erstellt eine Fensterdefinition mit Alias, die in analytischen Funktionen beliebig wiederverwendet werden kann.

```
SELECT WEEK(rental_date) AS week,
       DAYNAME(rental_date) AS weekday,
       COUNT(*) AS daily_rentals,
       RANK() OVER week_window AS rentals_weekly_rank
FROM rental
WHERE MONTHNAME(rental_date) = "July"
GROUP BY week,
         weekday
WINDOW week_window AS (PARTITION BY WEEK(rental_date)
                       ORDER BY COUNT(*) DESC)
ORDER BY week;
```

stackfuel

`LAG()` und `LEAD()` geben Zugriff auf Zeilenwerte innerhalb eines Fensters, die vor bzw. nach der aktuellen Zeile stehen.

```
SELECT WEEK(rental_date) AS week,
       DAY(rental_date) AS weekday,
       COUNT(*) AS daily_rentals,
       LAG(COUNT(*), 1, NULL) OVER
           week_window - COUNT(*) AS diff_prev_line,
       LEAD(COUNT(*), 1, NULL) OVER
           week_window - COUNT(*) AS diff_next_line
FROM rental
WHERE MONTHNAME(rental_date) = "July"
GROUP BY week,
         weekday
WINDOW week_window AS (PARTITION BY WEEK(rental_date)
                       ORDER BY DAY(rental_date))
ORDER BY week;
```

Eine Fensterdefinition kann eine *frame*-Subklausel beinhalten. Sie ermöglicht dynamischen Zugriff auf Werte in Vorgänger- und Nachfolgerzeilen. Das jeweilige Fenster einer Zeile kann sich nicht nur auf konkrete Zeilen beziehen, sondern auch auf zeitliche Intervalle.

```
SELECT WEEK(rental_date) AS week,
       DAY(rental_date) AS weekday,
       COUNT(*) AS daily_rentals,
       SUM(COUNT(*)) OVER all_prev_window AS all_prev_rolling_sum,
       AVG(COUNT(*)) OVER plus_minus_three_days AS 7_day_avg
FROM rental
WHERE MONTHNAME(rental_date) = "July"
GROUP BY week,
         weekday,
         DATE(rental_date)
WINDOW all_prev_window AS (ORDER BY DAY(rental_date)
                           ROWS BETWEEN UNBOUNDED PRECEDING
                                    AND 1 PRECEDING),
       plus_minus_three_days AS (
           ORDER BY DATE(rental_date)
           RANGE BETWEEN INTERVAL 3 DAY PRECEDING
                     AND INTERVAL 3 DAY FOLLOWING)
ORDER BY week;
```

Zwischensummen berechnen

`GROUP BY ... WITH ROLLUP` fügt zusätzliche Ergebniszeilen hinzu mit Zwischensummen pro Gruppe der gruppierten Spalten.

```
SELECT WEEK(rental_date) AS week,
       DAY(rental_date) AS day,
       COUNT(*) AS daily_rentals
FROM rental
WHERE MONTHNAME(rental_date) = "July"
GROUP BY week,
         day WITH ROLLUP;
```

stackfuel

`GROUPING()` identifiziert Zwischensummenzeilen und ermöglicht es, diese individuell umzubenennen.

```
SELECT IF(GROUPING(staff_id), 'all_staff', staff_id) AS staff_id,
       COUNT(*) AS total_rentals_july
FROM rental
WHERE MONTHNAME(rental_date) = "July"
GROUP BY staff_id WITH ROLLUP;
```

Gruppierungswerte zu Liste konkatenieren

`GROUP_CONCAT()` ist eine Aggregationsfunktion, die beim Gruppieren die Werte pro Gruppe zu einer Liste zusammenfasst.

```
SELECT inventory_id,
       GROUP_CONCAT(customer_id
           ORDER BY customer_id
           SEPARATOR ' / ') AS customers
FROM rental
WHERE inventory_id IN (2, 3, 5, 7, 11, 13, 17, 19, 23, 29)
GROUP BY inventory_id
HAVING COUNT(*) > 2;
```

CTEs

CTEs funktionieren ähnlich zu Subqueries. Beachte das sie immer benannt sein müssen. CTEs können nach ihrer Definition beliebig oft innerhalb einer Query genutzt werden.

```
WITH min_rental_duration AS
(SELECT MIN(DATEDIFF(return_date, rental_date)) AS min_duration
 FROM rental)
```