Analytische Funktionen in SQL für Einsteiger

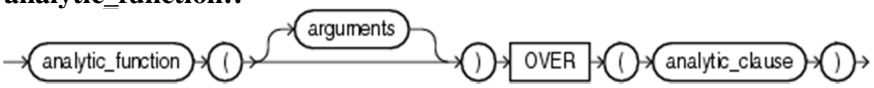
Jürgen Habdank
Ewald GmbH
Miesbacher Str. 38a
D-83620 Feldkirchen-Westerham

Schlüsselworte
ORACLE, ANSI SQL 99, Analytische Funktionen, analytic functions, SQL, Einsteiger.

Einleitung
Überblick über die analytischen Funktionen in SQL und Vorstellung von einzelnen analytischen Funktionen in SQL an Hand von konkreten Beispielen für die ORACLE Datenbank.
Es handelt sich bei den analytischen Funktionen um mächtige Funktionen, die es erlauben, Abfragen zu erstellen, für die man ansonsten prozeduralen oder sehr umständlichen SQL-Code programmieren müsste. Der Vortrag richtet sich an Anwender und Programmierer, die einen ersten Einblick in die Möglichkeiten der analytischen Funktionen erhalten möchten.

Syntax der analytischen Funktionen

analytic_function::=



Hinweis: Das Syntaxdiagramm wurde aus der „Oracle® Database SQL Language Reference 11g Release 2 (11.2)“ übernommen.

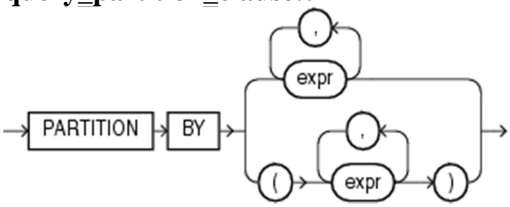
- Die analytische (Aggregats-)Funktion wird für jede Zeile der Eingabemenge berechnet, liefert also die gleiche Anzahl von Zeilen wie die Eingabemenge für die Funktion.
- Analytische Funktionen können im „**Select**“ (auch in einer Sub-Query) und im „**Order By**“ verwendet werden.
- Analytische Funktionen können **nicht** im „**Where**“ verwendet werden.
- Analytische Funktionen können **nicht** ineinander **geschachtelt** werden.
- Es können auch anwenderdefinierte analytische Funktionen verwendet werden (nicht Bestandteil dieses Dokuments).

analytic_clause::=

Hinweis: Das Syntaxdiagramm wurde aus der „Oracle® Database SQL Language Reference 11g Release 2 (11.2)“ übernommen.

- Anders als bei den Standard-(Aggregats-)Funktionen, bei denen die Gruppierung mit „**Group By**“ erfolgt, wird diese bei den analytischen (Aggregats-)Funktionen durch die „**query_partition_clause**“ und die „**windowing_clause**“ durchgeführt.
- Die einzelnen Partitionen können durch die „**order_by_clause**“ sortiert werden. Diese Sortierung bezieht sich aber nur auf die Eingabemenge der Funktion, nicht auf die Sortierung des gesamten Result-Sets.
- Die drei Komponenten „**query_partition_clause**“, „**order_by_clause**“ und „**windowing_clause**“, die das Fenster („window“) bestimmen, auf dem die Funktion operiert, werden in der „**analytic_clause**“, d.h. im SQL „**Over(..)**“, definiert.
- Das Fenster wird in Abhängigkeit von der aktuellen Zeile des gesamten Result-Sets berechnet und bewegt sich somit. Die Größe dieses Fensters basiert auf logischen Intervallen (z.B. Datum, Zeit) oder auf physikalischen Rows.
- Eine “**windowing-clause**” kann nur angegeben werden, wenn auch eine “**order_by_clause**” angegeben wird.
- Die „**analytic_clause**“ wird nach „**From**“, „**Where**“, „**Group By**“ und „**Having**“ des gesamten Result-Sets ausgeführt.
- Die „**analytic_clause**“ wird vor „**Order By**“ des gesamten Result-Sets ausgeführt.

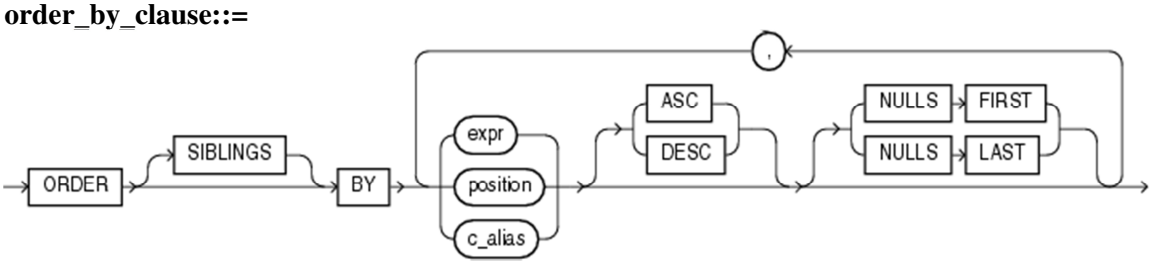
query_partition_clause::=



Hinweis: Das Syntaxdiagramm wurde aus der „Oracle® Database SQL Language Reference 11g Release 2 (11.2)“ übernommen.

- Für analytische Funktionen gilt der obere Zweig des Syntaxdiagramms, d.h. die einzelnen Ausdrücke (z.B. Spalten, Konstanten, nicht-analytische Funktionen, …) werden mit Komma getrennt, aber ohne umschließende Klammern angegeben.
- Die „**query_partition_clause**“ unterteilt die Zeilen der Eingabemenge durch die angegebenen Spalten oder Ausdrücke in Partitionen.
- Wird die „**query_partition_clause**“ weggelassen, werden alle Zeilen der Eingabemenge als eine Partition behandelt.
- Das Ergebnis der analytischen Funktion wird je Partition berechnet und wird für jede neue Partition wieder zurückgesetzt (z.B. Summe, Durchschnitt, Minimum, Maximum, …).

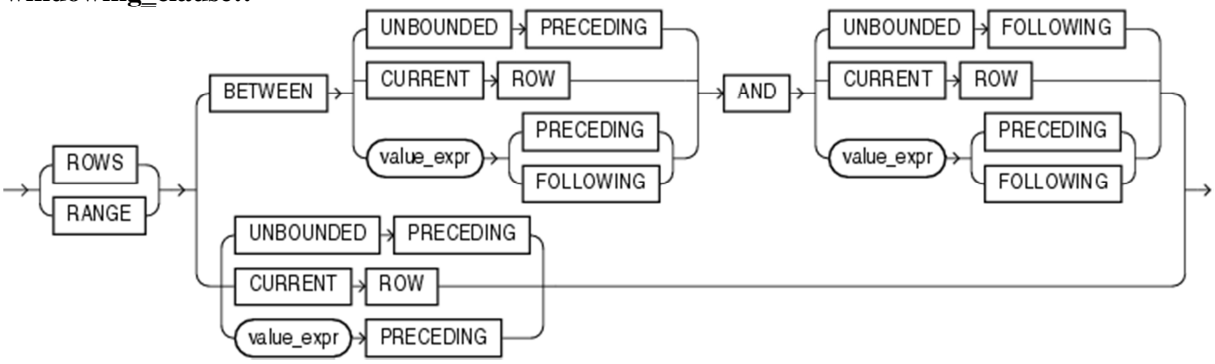
order_by_clause::=



Hinweis: Das Syntaxdiagramm wurde aus der „Oracle® Database SQL Language Reference 11g Release 2 (11.2)“ übernommen.

- Für analytische Funktionen können nur Ausdrücke (z.B. Spalten, Konstanten, nicht-analytische Funktionen, …) verwendet werden. „SIBLINGS“ kann für analytische Funktionen nicht verwendet werden.
- Die „**order_by_clause**“ sortiert die, durch die „**query_partition_clause**“ festgelegten Zeilen und liefert diese als sortierte Menge für die analytische Funktion.
- Die „**order_by_clause**“ liefert keine Sortierung für das gesamte Result-Set. Für das gesamte Result-Set muss deshalb eine eigene Sortierung angegeben werden.
- Wird die „**windowing_clause**“ für einen logischen Bereich gebildet („RANGE“), dann sind mehrere Sortierkriterien nur für „**RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW**“ und „**RANGE BETWEEN CURRENT ROW AND UNBOUNDED FOLLOWING**“ erlaubt.
- Wird die „**windowing_clause**“ für einen physikalischen Bereich gebildet („ROWS“), dann sind immer mehrere Sortierkriterien erlaubt.
- Achtung bei “**Null**”-Werten: Diese haben den **höchsten Wert**, also bei aufsteigender Sortierung am Ende, bei absteigender Sortierung am Anfang. “Null”-Werte am besten ausschließen.

windowing_clause::=



Hinweis: Das Syntaxdiagramm wurde aus der „Oracle® Database SQL Language Reference 11g Release 2 (11.2)“ übernommen.

- Die „**windowing_clause**“ legt den Anfang und das Ende des Fensters, bezogen auf die aktuelle Row des gesamten Result-Sets, fest.
- Die analytische Funktion wird basierend auf den Daten in diesem Fenster berechnet.
- „**CURRENT ROW**“ repräsentiert den logischen Wert bzw. die physikalische Row, für die die Funktion das Ergebnis berechnet.
- „**UNBOUNDED PRECEDING**“ ist die erste Zeile der aktuellen Partition.
- „**UNBOUNDED FOLLOWING**“ ist die letzte Zeile der aktuellen Partition.
- Wird der Start- oder Endpunkt weggelassen (unterer Zweig) so wird der fehlende Start- bzw. Endpunkt mit “**CURRENT ROW**” angenommen.
- Der **Standard** für die “**windowing-clause**” ist “**RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW**” und kann auch weggelassen werden.
- Ist **keine** “**order_by_clause**” angegeben, gibt es auch **keinen Standard** für die “**windowing_clause**”.

- Eine "**windowing-clause**" kann nur angegeben werden, wenn auch eine "**order_by_clause**" angegeben wird.
- "**windowing_clause**" mit "**ROWS**": Zeilen ermitteln auf Basis der **physikalischen Rows**. Der Bezug ist also die Position der Zeile.
- "**windowing_clause**" mit "**RANGE**": Zeilen ermitteln auf Basis einer "**where_clause**". Der Bezug ist also der Wert der Zeile.
- "**windowing_clause**" mit "**RANGE**": Intervalle sind **numerisch** oder "**Date**".
- "**windowing_clause**" mit "**RANGE**":"**CURRENT ROW**" ist der aktuelle Wert der Zeile nicht die physikalische Row.
- Mit "<Ausdruck> PRECEDING" bzw. "<Ausdruck> FOLLOWING" kann ein logisches bzw. physikalisches Intervall definiert werden.
- Die „**windowing_clause**" ist nicht bei allen analytischen Funktionen erlaubt (siehe unten).

Liste der wichtigsten analytischen Funktionen
Die folgenden Funktionen können als analytische Funktionen verwendet werden. Eine nähere Beschreibung der analytischen Funktionen findet sich in der ORACLE-Dokumentation (z.B. „Oracle® Database SQL Language Reference 11g Release 2 (11.2): Analytic Functions“):

```
AVG
CORR
COUNT
COVAR_POP
COVAR_SAMP
CUME_DIST                    (keine "windowing_clause" möglich)
DENSE_RANK                   (keine "windowing_clause" möglich)
FIRST_VALUE
LAG                          (keine "windowing_clause" möglich)
LAST_VALUE
LEAD                         (keine "windowing_clause" möglich)
MAX
MIN
NTILE                        (keine "windowing_clause" möglich)
PERCENT_RANK                 (keine "windowing_clause" möglich)
RANK                         (keine "windowing_clause" möglich)
RATIO_TO_REPORT              (keine "windowing_clause" möglich)
REGR_ Functions
ROW_NUMBER                   (keine "windowing_clause" möglich)
STDDEV
STDDEV_POP
STDDEV_SAMPS
SUM
VAR_POP
VAR_SAMP
VARIANCE
```

Vorbemerkungen zu den Beispielen
Alle folgenden Beispiele wurden auf dem „HR“-Schema einer ORACLE 11 XE entwickelt und getestet und sind auch auf diesem Schema ablauffähig.
Evtl. muss das „HR“-Schema zunächst mit folgendem SQL-Befehl aus einem anderen Schema (z.B. „SYSTEM“) freigeschaltet werden:

```
ALTER USER HR ACCOUNT UNLOCK
```

Für die Ausführung der Beispiele wurde der SQL Developer von ORACLE in der Version 4.0.2 verwendet.

Beispiel 01: Laufende Summierung über Angestellte
Die Angestellten werden nach Gehalt und Nachname sortiert. Die folgenden Werte werden mit einer analytischen Funktion berechnet:

- „CurrSumEmp“: Zu jedem Angestellten wird die Summe der Gehälter aller Angestellten aus den vorhergehenden Zeilen (inkl. des Angestellten selbst) ausgegeben.

```
Select
  EMPLOYEES.LAST_NAME,
  EMPLOYEES.SALARY,
  Sum(EMPLOYEES.SALARY)
    Over(Order By
           EMPLOYEES.SALARY,
           EMPLOYEES.LAST_NAME
         Range Between Unbounded Preceding And Current Row
        ) CurrSumEmp
From
  EMPLOYEES
Order By
  EMPLOYEES.SALARY,
  EMPLOYEES.LAST_NAME;
```

Beispiel 02: Laufende Summierung über Angestellte je Abteilung
Die Angestellten werden nach Abteilung und Nachname sortiert. Die folgenden Werte werden mit einer analytischen Funktion berechnet:

- „CurrSumEmpPerDep“: Zu jedem Angestellten wird die Summe der Gehälter aller Angestellten der Abteilung aus den vorhergehenden Zeilen (inkl. des Angestellten selbst) ausgegeben.

```
Select
  DEPARTMENTS.DEPARTMENT_NAME,
  EMPLOYEES.LAST_NAME,
  EMPLOYEES.SALARY,
  Sum(EMPLOYEES.SALARY)
    Over(Partition By
           DEPARTMENTS.DEPARTMENT_NAME
         Order By
           EMPLOYEES.LAST_NAME
         Range Between Unbounded Preceding And Current Row
        ) CurrSumEmpPerDep
From
  EMPLOYEES,
  DEPARTMENTS
Where
  EMPLOYEES.DEPARTMENT_ID = DEPARTMENTS.DEPARTMENT_ID
Order By
  DEPARTMENTS.DEPARTMENT_NAME,
  EMPLOYEES.LAST_NAME;
```

Beispiel 03: Mehrere Summierungen in einer Anweisung
Die Angestellten werden nach Abteilung und Gehalt sortiert. Die folgenden Werte werden mit einer analytischen Funktion berechnet:

- „CurrSumEmpPerDep“: Zu jedem Angestellten wird die Summe der Gehälter aller Angestellten der Abteilung aus den vorhergehenden Zeilen (inkl. des Angestellten selbst) ausgegeben.
  Liefert fehlerhaftes Ergebnis, wenn Gehaltswerte mehrfach in der Abteilung vorkommen, da hier „CURRENT ROW“ dem Wert des aktuellen Angestellten entspricht.
  Lösung: Zusätzlicher „Order By …, EMPLOYEES.LAST_NAME“.
- „SumEmpPerDep“: Zu jedem Angestellten wird die Summe der Gehälter aller Angestellten der Abteilung ausgegeben.
- „CurrSumEmpAllDep“: Zu jedem Angestellten wird die Summe der Gehälter aller Angestellten aus den vorhergehenden Zeilen (inkl. des Angestellten selbst) ausgegeben.
  Liefert fehlerhaftes Ergebnis, wenn Gehaltswerte mehrfach in der Abteilung vorkommen, da hier „CURRENT ROW“ dem Wert des aktuellen Angestellten entspricht.
  Lösung: Zusätzlicher „Order By …,EMPLOYEES.LAST_NAME“.
- „SumEmpAllDep“: Zu jedem Angestellten wird die Summe der Gehälter aller Angestellten ausgegeben.

```
Select
  EMPLOYEES.DEPARTMENT_ID,
  EMPLOYEES.LAST_NAME,
  EMPLOYEES.SALARY,
  Sum(EMPLOYEES.SALARY)
    Over(Partition By
           EMPLOYEES.DEPARTMENT_ID
         Order By
           EMPLOYEES.SALARY
        ) CurrSumEmpPerDep,
  Sum(EMPLOYEES.SALARY)
    Over(Partition By
           EMPLOYEES.DEPARTMENT_ID
        ) SumEmpPerDep,
  -- Kein Standard für -'windowing-clause', da keine 'order_by_clause'
  Sum(EMPLOYEES.SALARY)
    Over(Order By
           EMPLOYEES.DEPARTMENT_ID,
           EMPLOYEES.SALARY
        ) CurrSumEmpAllDep,
  Sum(EMPLOYEES.SALARY)
    Over(
        ) SumEmpAllDep
  -- Kein Standard für -'windowing-clause', da keine 'order_by_clause'
From
  EMPLOYEES
Order By
  EMPLOYEES.DEPARTMENT_ID,
  EMPLOYEES.SALARY;
```

Beispiel 04: Nachfolgende oder vorhergehende Werte
Die Angestellten werden nach Gehalt sortiert. Die folgenden Werte werden mit einer analytischen Funktion berechnet:

- „NextEmp“: Zu jedem Angestellten wird der Name des Angestellten mit dem nächst höheren Gehalt ausgegeben. Falls es diesen nicht gibt, wird „nicht vorhanden“ ausgegeben.
- „AfterNextSal“: Zu jedem Angestellten wird das Gehalt des Angestellten mit dem übernächst höheren Gehalt ausgegeben.
- „PrevEmp“: Zu jedem Angestellten wird der Name des Angestellten mit dem nächst niedrigeren Gehalt ausgegeben. Falls es diesen nicht gibt, wird „nicht vorhanden“ ausgegeben.

```
Select
  EMPLOYEES.LAST_NAME,
  EMPLOYEES.SALARY,
  Lead(EMPLOYEES.LAST_NAME,
       1,
       'nicht vorhanden'
      )
    Over(Order By
           EMPLOYEES.SALARY
         ) NextEmp,
  -- Keine 'windowing_clause' für 'Lead' erlaubt => Komplette Menge
  Lead(EMPLOYEES.SALARY,
       2,
       Null
      )
    Over(Order By
           EMPLOYEES.SALARY
         ) AfterNextSal,
  -- Keine 'windowing_clause' für 'Lead' erlaubt => Komplette Menge
  Lag(EMPLOYEES.LAST_NAME,
       1,
       'nicht vorhanden'
      )
    Over(Order By
           EMPLOYEES.SALARY
         ) PrevEmp
  -- Keine 'windowing_clause' für 'Lag' erlaubt => Komplette Menge
From
  EMPLOYEES
Order By
  EMPLOYEES.SALARY;
```

Beispiel 05: Nachfolgende oder vorhergehende Werte
Die Angestellten werden nach Abteilung und Gehalt sortiert. Die folgenden Werte werden mit einer analytischen Funktion berechnet:

- „NextEmp“: Zu jedem Angestellten wird der Name des Angestellten aus der gleichen Abteilung mit dem nächst höheren Gehalt ausgegeben. Falls es diesen nicht gibt, wird „nicht vorhanden“ ausgegeben.
- „PrevEmp“: Zu jedem Angestellten wird der Name des Angestellten aus der gleichen Abteilung mit dem nächst niedrigeren Gehalt ausgegeben. Falls es diesen nicht gibt, wird „nicht vorhanden“ ausgegeben.

```
Select
  EMPLOYEES.DEPARTMENT_ID,
  EMPLOYEES.LAST_NAME,
  EMPLOYEES.SALARY,
  Lead(EMPLOYEES.LAST_NAME,
       1,
       'nicht vorhanden'
      )
    Over(Partition By
           EMPLOYEES.DEPARTMENT_ID
         Order By
           EMPLOYEES.SALARY
        ) NextEmp,
  -- Keine 'windowing_clause' für 'Lead' erlaubt => Komplette Menge
  Lag(EMPLOYEES.LAST_NAME,
       1,
       'nicht vorhanden'
      )
    Over(Partition By
           EMPLOYEES.DEPARTMENT_ID
         Order By
           EMPLOYEES.SALARY
        ) PrevEmp
  -- Keine 'windowing_clause' für 'Lag' erlaubt => Komplette Menge
From
  EMPLOYEES
Order By
  EMPLOYEES.DEPARTMENT_ID,
  EMPLOYEES.SALARY;
```

Beispiel 06: Rangfolge der Werte
Die Angestellten werden nach Einstellungsdatum sortiert. Die folgenden Werte werden mit einer analytischen Funktion berechnet:

- „RankWithGaps“: Zu jedem Angestellten wird die Platzierung des Angestellten bezogen auf das Einstellungsdatum ausgegeben. Haben zwei oder mehr Angestellte das gleiche Einstellungsdatum, so haben diese die gleiche Platzierung. Für den Angestellten mit dem nächst höherem Einstellungsdatum werden entsprechend viele Platzierungen ausgelassen (Platzierungen wie in Sporttabellen).
- „RankWithoutGaps“: Zu jedem Angestellten wird die Platzierung des Angestellten bezogen auf das Einstellungsdatum ausgegeben. Haben zwei oder mehr Angestellte das gleiche Einstellungsdatum, so haben diese die gleiche Platzierung. Für den Angestellten mit dem nächst höherem Einstellungsdatum wird die nächste Platzierung verwendet.

```
Select
  EMPLOYEES.LAST_NAME,
  EMPLOYEES.HIRE_DATE,
  Rank()
    Over(Order By
           EMPLOYEES.HIRE_DATE
         ) RankWithGaps,
  Dense_Rank()
    Over(Order By
           EMPLOYEES.HIRE_DATE
         ) RankWithoutGaps
From
  EMPLOYEES
Order By
  EMPLOYEES.HIRE_DATE;
```

Beispiel 07: Rangfolge der Werte
Die Angestellten werden nach Einstellungsdatum sortiert. Die folgenden Werte werden mit einer analytischen Funktion berechnet:

- „RankYearWithGaps“: Zu jedem Angestellten wird die Platzierung des Angestellten bezogen auf das Einstellungsdatum aller Angestellten, die im gleichen Jahr eingestellt wurden, ausgegeben. Haben zwei oder mehr Angestellte das gleiche Einstellungsdatum, so haben diese die gleiche Platzierung. Für den Angestellten mit dem nächst höherem Einstellungsdatum werden entsprechend viele Platzierungen ausgelassen (Platzierungen wie in Sporttabellen).
- „ConsNumbering“: Zu jedem Angestellten wird die fortlaufende Nummerierung des Angestellten bezogen auf das Einstellungsdatum ausgegeben.

```
Select
  EMPLOYEES.LAST_NAME,
  EMPLOYEES.HIRE_DATE,
  Rank()
    Over(Partition By
           To_Char(EMPLOYEES.HIRE_DATE,'YYYY')
         Order By
           EMPLOYEES.HIRE_DATE
        ) RangYearWithGaps,
  Row_Number()
    Over(Order By
           EMPLOYEES.HIRE_DATE
        ) ConsNumbering
From
  EMPLOYEES
Order By
  EMPLOYEES.HIRE_DATE;
```

Beispiel 08: Erster bzw. letzter Wert einer Gruppe
Die Angestellten werden nach Abteilung und Gehalt sortiert. Die folgenden Werte werden mit einer analytischen Funktion berechnet:

- „DiffLowSalDep“: Zu jedem Angestellten wird der Unterschied zum niedrigsten Gehalt der Abteilung ausgegeben.
- „HighSalDepWrong“: Zu jedem Angestellten wird das höchste Gehalt der Abteilung ausgegeben. Führt hier zu einem fehlerhaften Ergebnis, da hier als Bereich nur die Angestellten mit einem niedrigeren oder gleichen Gehalt wie der Angestellte betrachtet werden.
- „HighSalDepRight“: Zu jedem Angestellten wird das höchste Gehalt der Abteilung ausgegeben.

```
Select
  EMPLOYEES.LAST_NAME,
  EMPLOYEES.DEPARTMENT_ID,
  EMPLOYEES.SALARY,
  EMPLOYEES.SALARY -
  First_Value(EMPLOYEES.SALARY)
    Over(Partition By
           EMPLOYEES.DEPARTMENT_ID
         Order By
           EMPLOYEES.SALARY
        ) DiffLowSalDep,
  Last_Value(EMPLOYEES.SALARY)
    Over(Partition By
           EMPLOYEES.DEPARTMENT_ID
         Order By
           EMPLOYEES.SALARY
        ) HighSalDepWrong,
  -- Werte nicht wie erwartet, da Standard für 'windowing_clause' gleich
  -- 'Range Between Unbounded Preceding And Current Row' !!!
  Last_Value(EMPLOYEES.SALARY)
    Over(Partition By
           EMPLOYEES.DEPARTMENT_ID
         Order By
           EMPLOYEES.SALARY
         Range Between Unbounded Preceding And Unbounded Following
        ) HighSalDepRight
From
  EMPLOYEES
Order By
  EMPLOYEES.DEPARTMENT_ID,
  EMPLOYEES.SALARY;
```

Beispiel 09: Einfache „windowing_clause“ über physikalische Rows
Die Angestellten werden nach Abteilung und Nachname sortiert. Die folgenden Werte werden mit einer analytischen Funktion berechnet:

- „SumSalPerDep“: Zu jedem Angestellten wird die Summe der Gehälter der beiden vorherigen und des aktuellen Angestellten der Abteilung ausgegeben.

```
Select
  EMPLOYEES.DEPARTMENT_ID,
  EMPLOYEES.LAST_NAME,
  EMPLOYEES.SALARY,
  Sum(EMPLOYEES.SALARY)
    Over(Partition By
           EMPLOYEES.DEPARTMENT_ID
         Order By
           EMPLOYEES.LAST_NAME
         Rows 2 Preceding
         -- Die zwei vorhergehende Rows und die aktuelle Row
         -- Kurz für 'Rows Between 2 Preceding And Current Row'
        ) SumSalPerDep
From
  EMPLOYEES
Order By
  EMPLOYEES.DEPARTMENT_ID,
  EMPLOYEES.LAST_NAME;
```

Beispiel 10: Einfache „windowing_clause“ über logischen Range
Die Angestellten werden nach Einstellungsdatum und Nachname sortiert. Die folgenden Werte werden mit einer analytischen Funktion berechnet:

- „EmpMax100DaysBefore“: Zu jedem Angestellten wird der am frühesten eingestellte Angestellte ausgegeben, der höchstens 100 Tage vor dem aktuellen Angestellten eingestellt wurde.
- „EmpMax100DaysAfter“: Zu jedem Angestellten wird der am spätesten eingestellte Angestellte ausgegeben, der höchstens 100 Tage nach dem aktuellen Angestellten eingestellt wurde.

```
Select
  EMPLOYEES.LAST_NAME,
  EMPLOYEES.HIRE_DATE,
  First_Value(EMPLOYEES.LAST_NAME)
    Over(Order By
           EMPLOYEES.HIRE_DATE
         Range Between 100 Preceding And 100 Following
        ) EmpMax100DaysBefore,
  -- Angestellter (kleinster Datums-Wert) der höchstens hundert Tage vorher
  -- eingestellt wurde
  Last_Value(EMPLOYEES.LAST_NAME)
    Over(Order By
           EMPLOYEES.HIRE_DATE
         Range Between 100 Preceding And 100 Following
        ) EmpMax100DaysAfter
  -- Angestellter (größter Datums-Wert) der höchstens hundert Tage nachher
  -- eingestellt wurde
From
  EMPLOYEES
Order By
  EMPLOYEES.HIRE_DATE,
  EMPLOYEES.LAST_NAME;
```

Beispiel 11: Durchschnittsgehalt über logischen Range
Die Angestellten werden nach Einstellungsdatum und Nachname sortiert. Die folgenden Werte werden mit einer analytischen Funktion berechnet:

- „AvgSal2MonthBefore2MonthAfter“: Für jeden Angestellten wird das Durchschnittsgehalt aller Angestellten ausgegeben, die maximal zwei Monate vorher und maximal zwei Monate nachher eingestellt wurden.

```
Select
  EMPLOYEES.LAST_NAME,
  EMPLOYEES.HIRE_DATE,
  EMPLOYEES.SALARY,
  Round(Avg(EMPLOYEES.SALARY)
          Over(Order By
                 EMPLOYEES.HIRE_DATE
               Range Between Numtoyminterval(2,'Month') Preceding And
               Numtoyminterval(2,'Month') Following
              ),2
       ) AvgSal2MonthBefore2MonthAfter
From
  EMPLOYEES
Order By
  EMPLOYEES.HIRE_DATE,
  EMPLOYEES.LAST_NAME;
```

Beispiel 12: Liste mit doppelten Werten
Die Angestellten werden nach Nachname und Vorname sortiert. Die folgenden Werte werden mit einer analytischen Funktion berechnet:

- Die Liste enthält alle Angestellten, deren Nachname mehrfach vorkommt.

```
Select
  EMPLOYEES_LIST.LAST_NAME,
  EMPLOYEES_LIST.FIRST_NAME,
  EMPLOYEES_LIST.LAST_NAME_NEXT,
  EMPLOYEES_LIST.FIRST_NAME_NEXT
From
  (
    Select
      EMPLOYEES.FIRST_NAME,
      EMPLOYEES.LAST_NAME,
      Lead(EMPLOYEES.FIRST_NAME,1,'nicht vorhanden')
        Over(Order By
               EMPLOYEES.LAST_NAME
            ) FIRST_NAME_NEXT,
      Lead(EMPLOYEES.LAST_NAME,1,'nicht vorhanden')
        Over(Order By
               EMPLOYEES.LAST_NAME
            ) LAST_NAME_NEXT
    From
      EMPLOYEES
  ) EMPLOYEES_LIST
Where
  EMPLOYEES_LIST.LAST_NAME =
  EMPLOYEES_LIST.LAST_NAME_NEXT
  -- In der 'where_clause' sind analytische Funktionen nicht erlaubt
Order By
  EMPLOYEES_LIST.LAST_NAME,
  EMPLOYEES_LIST.FIRST_NAME;
```

Beispiel 13: Kleinster Wert einer Menge
Die Angestellten werden nach Manager, Nachname, Einstellungsdatum und Gehalt sortiert. Die folgenden Werte werden mit einer analytischen Funktion berechnet:

- „MinSalEmp“: Für jeden Angestellten wird das kleinste Gehalt aller Angestellten ausgegeben, die zum gleichen Manager gehören und die vorher eingestellt wurden.

```
Select
  EMPLOYEES.MANAGER_ID,
  EMPLOYEES.LAST_NAME,
  EMPLOYEES.HIRE_DATE,
  EMPLOYEES.SALARY,
  Min(EMPLOYEES.SALARY)
    Over(Partition By
           EMPLOYEES.MANAGER_ID
         Order By
           EMPLOYEES.HIRE_DATE
         Range Between Unbounded Preceding And Current Row
        ) MinSalEmp
From
  EMPLOYEES
Order By
  EMPLOYEES.MANAGER_ID,
  EMPLOYEES.LAST_NAME,
  EMPLOYEES.HIRE_DATE,
  EMPLOYEES.SALARY;
```

Betrachtung der Performance (Vorbereitung)
Für die Betrachtung der Performance wird eine Datenbanktabelle benötigt, die eine größere Anzahl von Einträgen hat, da sonst die Unterschiede zu gering bzw. nicht korrekt messbar wären.
Zu diesem Zweck wird zunächst eine Datenbanktabelle mit 5000 „Dummy"-Angestellten erzeugt, auf der dann die Performance-Messungen durchgeführt werden.

```
Drop Table
  BIG_EMPLOYEES;

Create Table
  BIG_EMPLOYEES
As
Select
  ALL_OBJECTS.OBJECT_NAME LAST_NAME,
  Mod(ALL_OBJECTS.OBJECT_ID,50) DEPARTMENT_ID,
  ALL_OBJECTS.OBJECT_ID SALARY
From
  ALL_OBJECTS
Where
  Rownum <= 5000;
  -- Testtabelle mit 5000 Einträgen
```

Evtl. muss in den Optionen eingestellt werden, dass alle Ergebniszeilen des „Select"-Statements ausgegeben werden sollen, da ansonsten die Performance-Messung nicht möglich ist.
Im SQL Developer von ORACLE kann dies unter „Extras -> Voreinstellungen -> Datenbank -> Erweitert" erfolgen. Dort den Wert von „SQL-Arrayabrufgröße" entsprechend (hier auf 5.000) setzen.

Betrachtung der Performance (mit analytischen Funktionen)
Die Angestellten werden nach Abteilung und Nachname sortiert. Die folgenden Werte werden mit einer analytischen Funktion berechnet:

- „CurrSumEmpSal“: Für jeden Angestellten wird die laufende Summe der Gehälter aller Angestellten ausgegeben, die vorher in der Liste ausgegeben wurden.
- „CurrSumDepSal“: Für jeden Angestellten wird die laufende Summe der Gehälter aller Angestellten der gleichen Abteilung ausgegeben, die vorher in der Liste ausgegeben wurden.
- „CurrDepCount“: Für jeden Angestellten wird die laufende Nummer innerhalb der Abteilung ausgegeben.

Wie man dem “Autotrace” entnehmen kann, sind bei Verwendung von analytischen Funktionen nur 25 Zugriffe notwendig. Das Ergebnis wird nach ca. 0,14 Sekunden zur Verfügung gestellt.

```
Set Autotrace Off;
Set Autotrace On Statistics;

Select
  BIG_EMPLOYEES.LAST_NAME,
  BIG_EMPLOYEES.DEPARTMENT_ID,
  BIG_EMPLOYEES.SALARY,
  Sum(BIG_EMPLOYEES.SALARY)
    Over(Order By
           BIG_EMPLOYEES.DEPARTMENT_ID,
           BIG_EMPLOYEES.LAST_NAME
         ) CurrSumEmpSal,
  Sum(BIG_EMPLOYEES.SALARY)
    Over(Partition By
           BIG_EMPLOYEES.DEPARTMENT_ID
          Order By
           BIG_EMPLOYEES.LAST_NAME
         ) CurrSumDepSal,
  Row_Number()
    Over(Partition By
           BIG_EMPLOYEES.DEPARTMENT_ID
          Order By
           BIG_EMPLOYEES.LAST_NAME
         ) CurrDepCount
From
  BIG_EMPLOYEES
Order By
  BIG_EMPLOYEES.DEPARTMENT_ID,
  BIG_EMPLOYEES.LAST_NAME;
-- Autotrace: Abfragezeit ca. 0,14s - 25 Zugriffe
```

Betrachtung der Performance (ohne analytische Funktionen)
Die Angestellten werden nach Abteilung und Nachname sortiert. Die folgenden Werte werden mit einer analytischen Funktion berechnet:

- „CurrSumEmpSal“: Für jeden Angestellten wird die laufende Summe der Gehälter aller Angestellten ausgegeben, die vorher in der Liste ausgegeben wurden.
- „CurrSumDepSal“: Für jeden Angestellten wird die laufende Summe der Gehälter aller Angestellten der gleichen Abteilung ausgegeben, die vorher in der Liste ausgegeben wurden.
- „CurrDepCount“: Für jeden Angestellten wird die laufende Nummer innerhalb der Abteilung ausgegeben.

Wie man dem “Autotrace” entnehmen kann, sind ohne Verwendung von analytischen Funktionen insgesamt 325.075 Zugriffe notwendig. Das Ergebnis wird nach ca. 6,2 Sekunden zur Verfügung gestellt.

```
Set Autotrace Off;
Set Autotrace On Statistics;

Select
  BIG_EMPLOYEES.LAST_NAME,
  BIG_EMPLOYEES.DEPARTMENT_ID,
  BIG_EMPLOYEES.SALARY,
  (
    Select
      Sum(BIG_EMPLOYEES_2.SALARY)
    From
      BIG_EMPLOYEES BIG_EMPLOYEES_2
    Where
      BIG_EMPLOYEES_2.DEPARTMENT_ID < BIG_EMPLOYEES.DEPARTMENT_ID Or
      (
        BIG_EMPLOYEES_2.DEPARTMENT_ID = BIG_EMPLOYEES.DEPARTMENT_ID And
        BIG_EMPLOYEES_2.LAST_NAME <= BIG_EMPLOYEES.LAST_NAME
      )
  ) CurrSumEmpSal,
  (
    Select
      Sum(BIG_EMPLOYEES_3.SALARY)
    From
      BIG_EMPLOYEES BIG_EMPLOYEES_3
    Where
      BIG_EMPLOYEES_3.DEPARTMENT_ID = BIG_EMPLOYEES.DEPARTMENT_ID And
      BIG_EMPLOYEES_3.LAST_NAME <= BIG_EMPLOYEES.LAST_NAME
  ) CurrSumDepSal,
  (
    Select
      Count(BIG_EMPLOYEES_4.LAST_NAME)
    From
      BIG_EMPLOYEES BIG_EMPLOYEES_4
    Where
      BIG_EMPLOYEES_4.DEPARTMENT_ID = BIG_EMPLOYEES.DEPARTMENT_ID And
      BIG_EMPLOYEES_4.LAST_NAME <= BIG_EMPLOYEES.LAST_NAME
  ) CurrDepCount
From
  BIG_EMPLOYEES
Order By
```

```
  BIG_EMPLOYEES.DEPARTMENT_ID,
  BIG_EMPLOYEES.LAST_NAME;
-- Autotrace: Abfragezeit ca. 6,2s - 375025 Zugriffe
```

Jürgen Habdank
Ewald GmbH
Miesbacher Str. 38a
D-83620 Feldkirchen-Westerham

Telefon: +49 (0) 8063-256 8063
Fax: +49 (0) 8063-256 8064
E-Mail juergen.habdank@ewald.de
Internet: www.ewald.de