Oracle10gR2 分析函数

(Translated By caizhuoyi 2008-9-19)

说明：

1、 原文中底色为黄的部分翻译存在商榷之处，请大家踊跃提意见；

2、 原文中淡蓝色字体的文字，不宜翻译，保持原样。

1. ANALYTIC FUNCTIONS

Analytic functions compute an aggregate value based on a group of rows. They differ from aggregate functions in that they return multiple rows for each group. The group of rows is called a **window** and is defined by the *analytic_clause*. For each row, a sliding window of rows is defined. The window determines the range of rows used to perform the calculations for the current row. Window sizes can be based on either a physical number of rows or a logical interval such as time.

分析函数通过将行分组后，再计算这些分组的值。它们与聚集函数不同之处在于能够对每一分组返回多行值。分析函数根据analytic_claues（分析子句）将行分组，一个分组称为一个窗口。每一行都对应有一个在行上滑动的窗口。该窗口确定当前行的计算范围。窗口大小可以用多个物理行进行度量，也可以使用逻辑区间进行度量，比如时间。

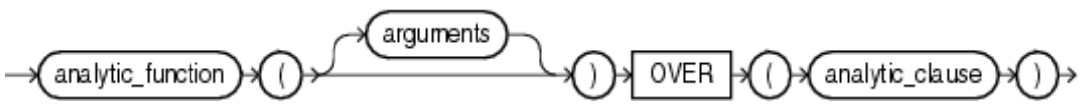
Analytic functions are the last set of operations performed in a query except for the final ORDER BY clause. All joins and all WHERE, GROUP BY, and HAVING clauses are completed before the analytic functions are processed. Therefore, analytic functions can appear only in the select list or ORDER BY clause.

分析函数是查询中除需要在最终处理的order by子句之外最后执行的操作。所有连接和所有where，group by，和having子句都要在处理分析函数之前进行计算。因此，分析函数只能用于选择列或order by子句中。

Analytic functions are commonly used to compute cumulative, moving, centered, and reporting aggregates.

分析函数通常用于计算数据累积值，数据移动值、数据中间值，和输出集合报表。

analytic_function::=



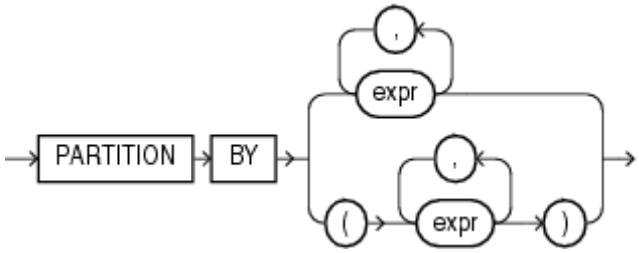

```
analytic_function([ arguments ])
```

```
OVER (analytic_clause)
```

analytic_clause::=

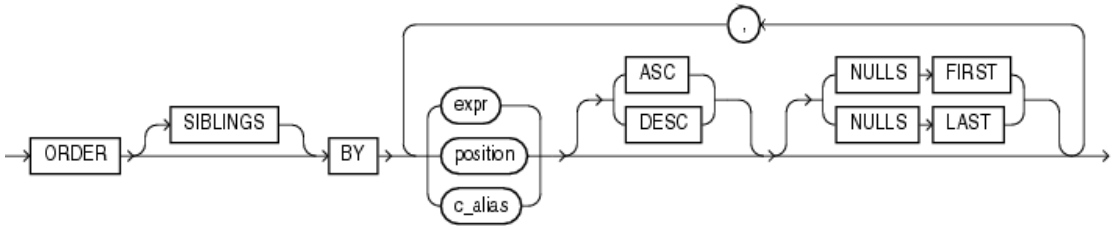

```
[ query_partition_clause ]
[ order_by_clause [ windowing_clause ] ]
```

query_partition_clause::=



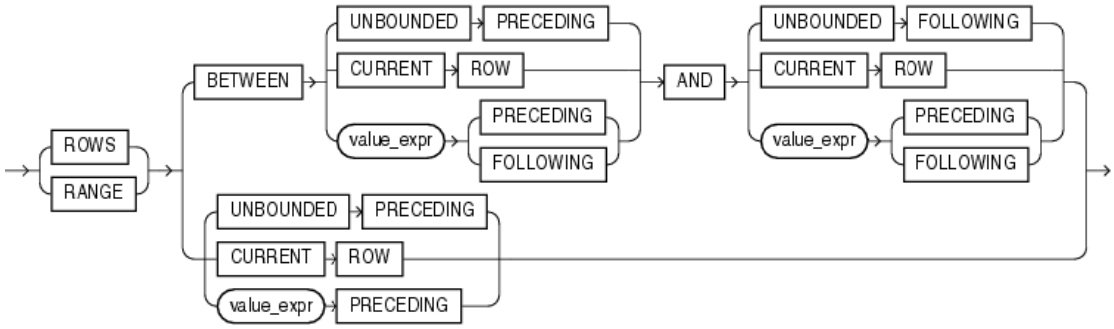

```
PARTITION BY
  { value_expr[, value_expr ]...
  | ( value_expr[, value_expr ]... )
  }
```

order_by_clause::=




```
ORDER [ SIBLINGS ] BY
{ expr | position | c_alias }
[ ASC | DESC ]
[ NULLS FIRST | NULLS LAST ]
  [, { expr | position | c_alias }
     [ ASC | DESC ]
     [ NULLS FIRST | NULLS LAST ]
  ]...
```

windowing_clause ::=




```
{ ROWS | RANGE }
{ BETWEEN
  { UNBOUNDED PRECEDING
  | CURRENT ROW
  | value_expr { PRECEDING | FOLLOWING }
  }
  AND
  { UNBOUNDED FOLLOWING
  | CURRENT ROW
  | value_expr { PRECEDING | FOLLOWING }
  }
| { UNBOUNDED PRECEDING
  | CURRENT ROW
  | value_expr PRECEDING
  }
}
```

The semantics of this syntax are discussed in the sections that follow.

以下各节将讨论分析函数语法的语义。

1.1 analytic_function

Specify the name of an analytic function (see the listing of analytic functions following this discussion of semantics).

Analytic_function指定分析函数的名称。(请参阅以下语义论述中的分析函数列表)

1.2 Arguments

Analytic functions take 0 to 3 arguments. The arguments can be any numeric datatype or any nonnumeric datatype that can be implicitly converted to a numeric datatype. Oracle determines the argument with the highest numeric precedence and implicitly converts the remaining arguments to that datatype. The return type is also that datatype, unless otherwise noted for an individual function.

分析函数可取0-3个参数。参数可以是任何数字类型或是可以隐式转换为数字类型的数据类型。Oracle根据最高数字优先级别确定函数参数，并且隐式地将需要处理的参数转换为数字类型。函数的返回类型也为数字类型，除非此函数另有说明。

See Also:
"Numeric Precedence" for information on numeric precedence and Table 2-10, "Implicit Type Conversion Matrix" for more information on implicit conversion

请参阅：
"Numeric Precedence"可获取数字优先级的相关信息，参阅表2-10—-隐式类型转换矩阵，可获取隐式转换的更多信息。

1.3 analytic_clause

Use OVER *analytic_clause* to indicate that the function operates on a query result set. That is, it is computed after the FROM, WHERE, GROUP BY, and HAVING clauses. You can specify analytic functions with this clause in the select list or ORDER BY clause. To filter the results of a query based on an analytic function, nest these functions within the parent query, and then filter the results of the nested subquery.

Over Analytic_clause用以指明函数操作的是一个查询结果集。也就是说分析函数是在**from,where,group by,**和**having**子句之后才开始进行计算的。因此在选择列或order by子句中可以使用分析函数。为了过滤分析函数计算的查询结果，可以将它作为子查询嵌套在外部查询中，然后在外部查询中过滤其查询结果。

Notes on the analytic_clause: The following notes apply to the *analytic_clause*:

Analytic_clause注意事项：使用分析子句注意事项如下：

- You cannot specify any analytic function in any part of the *analytic_clause*. That is, you cannot nest analytic functions. However, you can specify an analytic function in a subquery and compute another analytic function over it.

  Analytic_clause中不能包含其他任何分析函数。也就是说，分析函数不能嵌套。然而，可以在一个子查询中应用分析函数，并且通过它计算另外的分析函数。

- You can specify OVER *analytic_clause* with user-defined analytic functions as well as built-in analytic functions. See CREATE FUNCTION.

  用户自定义分析函数和内置函数分析函数都可以使用over analytic_clause。参见create function。

1.4 query_partition_clause

Use the PARTITION BY clause to partition the query result set into groups based on one or more *value_expr*. If you omit this clause, then the function treats all rows of the query result set as a single group.

Partition by子句根据一个或多个value_expr将查询结果集分成若干组。若不使用该子句，那末函数将查询结果集的所有行当作一个组。

To use the *query_partition_clause* in an analytic function, use the upper branch of the syntax (without parentheses). To use this clause in a model query (in the *model_column_clauses*) or a partitioned outer join (in the *outer_join_clause*), use the lower branch of the syntax (with parentheses).

在分析函数中使用query_partition_clause，应该使用语法图中上分支中的语法(不带圆括号).在model查询(位于model_column_clauses中)或被分隔的外部连接(位于outer_join_clause中)中使用该子句,应该使用语法图中下分支中的语法(带有圆括号)。

You can specify multiple analytic functions in the same query, each with the same or different PARTITION BY keys.

在同一查询中可以使用多个分析函数，它们可以有相同或不同的partition by键值。

If the objects being queried have the parallel attribute, and if you specify an analytic function with the *query_partition_clause*, then the function computations are parallelized as well.

若被查询的对象具有并行特性，并且分析函数中包含query_partition_clause，那末函数的计算也是并行的。

Valid values of *value_expr* are constants, columns, nonanalytic functions, function expressions, or expressions involving any of these.

value_expr的有效值包括常量，表列，非分析函数，函数表达式，或者前面这些元素的任意组合表达式。

1.5 order_by_clause

Use the *order_by_clause* to specify how data is ordered within a partition. For all analytic functions except PERCENTILE_CONT and PERCENTILE_DISC (which take only a single key), you can order the values in a partition on multiple keys, each defined by a *value_expr* and each qualified by an ordering sequence.

Order_by_clause用以指定分组中数据的排序形式。除PERCENTILE_CONT和PERCENTILE_DISC之外(它们只能取唯一的键值)外的分析函数，分组中可以使用多个键值对值进行排序，每个键值在value_expr中定义，并且被排序序列限定。

Within each function, you can specify multiple ordering expressions. Doing so is especially useful when using functions that rank values, because the second expression can resolve ties between identical values for the first expression.

每个函数内可以指定多个排序表达式。当使用函数给值排名时，尤其显得意义非凡，因为第二个表达式能够解决按照第一个表达式排序后仍然存在相同排名的问题。

Whenever the *order_by_clause* results in identical values for multiple rows, the function returns the same result for each of those rows. Please refer to the analytic example for SUM for an illustration of this behavior.

只要使用order_by_clause后，仍存在值相同的行，则每一行都会返回相同的结果。相关行为的例子请参阅考sum分析函数的例子。

Restrictions on the ORDER BY Clause The following restrictions apply to the ORDER BY clause:

Order by子句的限制：下面是使用order by子句的一些限制：

- When used in an analytic function, the *order_by_clause* must take an expression (*expr*). The SIBLINGS keyword is not valid (it is relevant only in hierarchical queries). Position (*position*) and column aliases (*c_alias*) are also invalid. Otherwise this *order_by_clause* is the same as that used to order the overall query or subquery.

  分析函数中的order_by_clause必须是一个表达式(expr)。Sibling关键字在此处是非法的(它仅仅与层次查询有关)。位置(position)和列别名（c_alias）也是非法的。除此之外，order_by_clause的用法与整个查询或子查询中的相同。

- An analytic function that uses the RANGE keyword can use multiple sort keys in its ORDER BY clause if it specifies either of these two windows:

  当分析函数使用range关键字限定窗口时，若使用的窗口是下列两个窗口之一，那末可以在分析函数的order by子句中使用多个排序键值。

  - RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW. The short form of this is RANGE UNBOUNDED PRECEDING.

    RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW.

简写成 **range unbounded preceding**

- RANGE BETWEEN CURRENT ROW AND UNBOUNDED FOLLOWING. The short form of this is RANGE UNBOUNDED FOLLOWING.

  RANGE BETWEEN CURRENT ROW AND UNBOUNDED FOLLOWING.

  简写成：**range unbounded following**

Window boundaries other than these two can have only one sort key in the ORDER BY clause of the analytic function. This restriction does not apply to window boundaries specified by the ROW keyword.

若窗口范围由range关键字指定的分析函数中指定的不是这两个窗口范围，那末order by子句中仅能使用一个排序键值。若分析函数的窗口范围由row关键字指定，order by子句中排序键值的使用没有这个限制。

ASC | DESC Specify the ordering sequence (ascending or descending). ASC is the default.

asc | desc 指定排序顺序(升序或降序)。asc是默认值。

NULLS FIRST | NULLS LAST Specify whether returned rows containing nulls should appear first or last in the ordering sequence.

nulls first | nulls last 指定若返回行包含空值，该值应该出现在排序序列的开始还是末尾。

NULLS LAST is the default for ascending order, and NULLS FIRST is the default for descending order.

升序排序的默认值是**nulls last**，降序排序的默认值是**nulls first**。

Analytic functions always operate on rows in the order specified in the *order_by_clause* of the function. However, the *order_by_clause* of the function does not guarantee the order of the result. Use the *order_by_clause* of the query to guarantee the final result ordering.

分析函数总是按order_by_clause对行排序。然而，分析函数中的order_by_clause只对各个分组进行排序，而不能保证查询结果有序。要保证最后的查询结果有序，可以使用查询的order_by_clause。

See Also:
order_by_clause of SELECT for more information on this clause

请参阅：
select中的order_by_clause获取该子句的更多信息。

1.6 windowing_clause

Some analytic functions allow the *windowing_clause*. In the listing of analytic functions at the end of this section, the functions that allow the *windowing_clause* are followed by an asterisk (*).

有些分析函数允许使用windowing_clause。在此节末尾的分析函数列表中，带有星号(*)的函数都允许使用windowing_clause。

ROWS | RANGE These keywords define for each row a window (a physical or logical set of rows) used for calculating the function result. The function is then applied to all the rows in the window. The window moves through the query result set or partition from top to bottom.

row | range 这些关键字为每一行定义一个窗口，该窗口用于计算函数结果(物理或逻辑的行的集合).然后对窗口中的每一行应用分析函数。窗口在查询结果集或分组中从上至下移动。

- ROWS specifies the window in physical units (rows).

  rows 指定窗口以物理单位(行)构成。

- RANGE specifies the window as a logical offset.

  range 指定窗口以逻辑偏移量构成。

You cannot specify this clause unless you have specified the *order_by_clause*. Some window boundaries defined by the RANGE clause let you specify only one expression in the *order_by_clause*. Please refer to "Restrictions on the ORDER BY Clause".

只有指定order_by_clause后才能指定windowing_clause。有些range子句定义的窗口范围只能在order_by_clause中指定一个排序表达式。请参阅Restrictions on order by Clause。

The value returned by an analytic function with a logical offset is always deterministic. However, the value returned by an analytic function with a physical offset may produce nondeterministic results unless the ordering expression results in a unique ordering. You may have to specify multiple columns in the *order_by_clause* to achieve this unique ordering.

一个带逻辑偏移量的分析函数的返回值总是确定的。然而，除非排序表达式能产生唯一的排序，否则带有物理偏移量的分析函数的返回值可能会产生不确定的结果。为了解决此问题，你可能不得不在order_by_clause中指定多个列以获得唯一的排序。

BETWEEN ... AND Use the BETWEEN ... AND clause to specify a start point and end point for the window. The first expression (before AND) defines the start point and the second expression (after AND) defines the end point.

between ... and between … and子句用来指定窗口的起点和终点。第一个表达式(位于and之前)定义起点，第二个表达式(位于and之后)定义终点。

If you omit BETWEEN and specify only one end point, then Oracle considers it the start point, and the end point defaults to the current row.

若不使用between而仅指定一个终点，那末oracle认为它是起点，终点默认为当前行。

UNBOUNDED PRECEDING Specify UNBOUNDED PRECEDING to indicate that the window starts at the first row of the partition. This is the start point specification and cannot be used as an end point specification.

unbounded preceding 指定unbounded preceding 指明窗口开始于分组的第一行。它只用于指定起点而不能用于指定终点。

UNBOUNDED FOLLOWING Specify UNBOUNDED FOLLOWING to indicate that the window ends at the last row of the partition. This is the end point specification and cannot be used as a start point specification.

unbounded following 指定unbounded following 指明窗口结束于分组的最后一行。它只用于指定终点而不能用于指定起点。

CURRENT ROW As a start point, CURRENT ROW specifies that the window begins at the current row or value (depending on whether you have specified ROW or RANGE, respectively). In this case the end point cannot be *value_expr* PRECEDING.

current row 用作起点，current row 指定窗口开始于当前行或当前值(依赖于是否分别指定row 或range)。在这种情况下终点不能为value_expr preceding。

As an end point, CURRENT ROW specifies that the window ends at the current row or value (depending on whether you have specified ROW or RANGE, respectively). In this case the start point cannot be *value_expr* FOLLOWING.

用作终点，current row 指定窗口结束于当前行或当前值(依赖于是否分别指定row 或range)。在这种情况下起点不能为value_expr following。

value_expr PRECEDING or value_expr FOLLOWING For RANGE or ROW:

range或row中的value_expr preceding 或 value_expr following：

- If *value_expr* FOLLOWING is the start point, then the end point must be *value_expr* FOLLOWING.

  若value_expr FOLLOWING是起点，那末终点必须是value_expr FOLLOWING。

- If *value_expr* PRECEDING is the end point, then the start point must be *value_expr* PRECEDING.

  若value_expr PRECEDING是终点，那末起点必须是value_expr PRECEDING。

If you are defining a logical window defined by an interval of time in numeric format, then you may need to use conversion functions.

若要定义一个数字格式的时间间隔的逻辑窗口，那末可能需要用到转换函数。

See Also:
NUMTOYMINTERVAL and NUMTODSINTERVAL for information on converting numeric times into intervals

请参阅：
NUMTOMINTERVAL和NUMTODSINTERVAL获取关于数次转换为时间间隔的信息。

If you specified ROWS:

若*windowing_clause*由rows指定：

- *value_expr* is a physical offset. It must be a constant or expression and must evaluate to a positive numeric value.

  value_expr是一个物理偏移量，它必须是一个常量或表达式，并且表达式的值必须为正数值。

- If *value_expr* is part of the start point, then it must evaluate to a row before the end point.

  若 value_expr 是起点的一部分，那末它必须在终点之前对行求值。

If you specified RANGE:

若*windowing_clause*由range指定：

- *value_expr* is a logical offset. It must be a constant or expression that evaluates to a positive numeric value or an interval literal. Please refer to "Literals" for information on interval literals.

  value_expr是一个逻辑偏移量。它必须是常量，或值为正数值的表达式，或时间间隔文字常量。请参阅Literals获取有关时间间隔文字常量的信息。

- You can specify only one expression in the *order_by_clause*

  只能在order_by_clause中指定一个表达式。

- If *value_expr* evaluates to a numeric value, then the ORDER BY *expr* must be a numeric or DATE datatype.

  若 value_expr 求值为一个数字值，那末 order by expr 必须为数字或 date 类型。

- If *value_expr* evaluates to an interval value, then the ORDER BY *expr* must be a DATE datatype.

  若value_expr求值为一个间隔值，那末order by expr必须是一个date类型。

If you omit the *windowing_clause* entirely, then the default is RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW.

若完全忽略windowing_clause，那末默认值为 range between unbounded preceding and current row 。

Analytic functions are commonly used in data warehousing environments. In the list of analytic functions that follows, functions followed by an asterisk (*) allow the full syntax, including the *windowing_clause*.

分析函数通常用于数据仓库环境中。下面是分析函数列表，带有星号的函数可以包含windowing_clause。

AVG *
CORR *
COVAR_POP *
COVAR_SAMP *
COUNT *
CUME_DIST

DENSE_RANK
FIRST
FIRST_VALUE *
LAG
LAST
LAST_VALUE *
LEAD
MAX *
MIN *
NTILE
PERCENT_RANK
PERCENTILE_CONT
PERCENTILE_DISC
RANK
RATIO_TO_REPORT
REGR_ (Linear Regression) Functions *
ROW_NUMBER
STDDEV *
STDDEV_POP *
STDDEV_SAMP *
SUM *
VAR_POP *
VAR_SAMP *
VARIANCE *

See Also:
Oracle Data Warehousing Guide for more information on these functions and for scenarios illustrating their use

请参阅：
Oracle Data Warehousing Guide获取关于这些函数及其方案使用说明的更多信息。

2. AVG

2.1 Syntax

AVG([DISTINCT | ALL] expr)

[OVER(analytic_clause)]

> **See Also:**
> "Analytic Functions" for information on syntax, semantics, and restrictions

2.2 Purpose

AVG returns average value of *expr*.

Avg函数返回expr的平均值。

This function takes as an argument any numeric datatype or any nonnumeric datatype that can be implicitly converted to a numeric datatype. The function returns the same datatype as the numeric datatype of the argument.

函数参数可取任何数字类型或任何可以隐式转换为数字类型的非数字类型。函数返回类型与参数类型相同，都为数字类型。

> **See Also:**
> Table 2-10, "Implicit Type Conversion Matrix" for more information on implicit conversion

If you specify DISTINCT, then you can specify only the *query_partition_clause* of the *analytic_clause*. The *order_by_clause* and *windowing_clause* are not allowed.

Distinct关键字仅能在analytic_clause的query_partition_clause中使用。在order_by_clause和windowing_clause中不允许使用distinct。

> **See Also:**
> "About SQL Expressions" for information on valid forms of *expr* and "Aggregate Functions"

2.3 Aggregate Example

The following example calculates the average salary of all employees in the hr.employees table:

下面的例子计算hr.employees表中所有雇员的平均薪水：

```
SELECT AVG(salary) "Average" FROM employees;

 AVERAGE
--------
    6425
```

2.4 Analytic Example

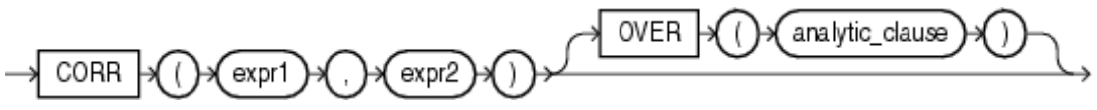
The following example calculates, for each employee in the `employees` table, the average salary of the employees reporting to the same manager who were hired in the range just before through just after the employee:

下面的例子计算，employees表中相同经理下的每一雇员和雇佣日期正好位于该雇员正前后的雇员的平均薪水：

```
SELECT manager_id,
       last_name,
       hire_date,
       salary,
       AVG(salary) over(PARTITION BY manager_id ORDER BY hire_date rows
BETWEEN 1 preceding AND 1 following) AS c_mavg
  FROM employees;

MANAGER_ID LAST_NAME                 HIRE_DATE     SALARY     C_MAVG
---------- ------------------------- --------- ---------- ----------
       100 Kochhar                   21-SEP-89      17000      17000
       100 De Haan                   13-JAN-93      17000      15000
       100 Raphaely                  07-DEC-94      11000 11966.6667
       100 Kaufling                  01-MAY-95       7900 10633.3333
       100 Hartstein                 17-FEB-96      13000 9633.33333
       100 Weiss                     18-JUL-96       8000 11666.6667
       100 Russell                   01-OCT-96      14000 11833.3333
. . .
```

3. CORR

3.1 Syntax



```
CORR(expr1, expr2)
   [ OVER (analytic_clause) ]
```

> **See Also:**
> "Analytic Functions" for information on syntax, semantics, and restrictions

3.2 Purpose

CORR returns the coefficient of correlation of a set of number pairs. You can use it as an aggregate or analytic function.

Corr返回一组数值对的相关系数。它可以用作聚集或分析函数。

This function takes as arguments any numeric datatype or any nonnumeric datatype that can be implicitly converted to a numeric datatype. Oracle determines the argument with the highest numeric precedence, implicitly converts the remaining arguments to that datatype, and returns that datatype.

函数参数可取任何数字类型或任何可以隐式转换为数字类型的非数字类型。Oracle根据最高数字优先级确定参数，隐式地将需要处理的参数转换为数字类型，并返回数字类型。

> **See Also:**
> Table 2-10, "Implicit Type Conversion Matrix" for more information on implicit conversion and "Numeric Precedence" for information on numeric precedence

Oracle Database applies the function to the set of (*expr1*, *expr2*) after eliminating the pairs for which either *expr1* or *expr2* is null. Then Oracle makes the following computation:

Oracle数据库使用该函数前先排除(expr1,expr2)集中所有expr1或expr2为null的数值对。然后作如下的计算：

```
COVAR_POP(expr1, expr2) / (STDDEV_POP(expr1) * STDDEV_POP(expr2))
```

The function returns a value of type NUMBER. If the function is applied to an empty set, then it returns null.

函数返回一个number类型的值。若函数应用在一个空集上，那末它将返回null。

> **Note:**
> The CORR function calculates the Pearson's correlation coefficient, which requires numeric expressions as arguments. Oracle also provides the CORR_S (Spearman's rho coefficient) and CORR_K (Kendall's tau-b coefficient) to support nonparametric or rank correlation.

注意：
Corr函数计算Pearson关系系数时，需要用数字表达式作为参数。Oracle也提供了corr_s(Spearman's rho系数)和corr_k(Kendall's tau-b系数)来支持非参数或排名相关性。

> **See Also:**
> "Aggregate Functions", "About SQL Expressions" for information on valid forms of *expr*, and CORR_* and CORR_S

3.3 Aggregate Example

The following example calculates the coefficient of correlation between the list prices and minimum prices of products by weight class in the sample table oe.product_information:

下面的列子，计算oe.product_information表中不同重量等级产品订价和最低价格之间的相关系数：

```
SELECT weight_class, corr(list_price, min_price)
  FROM product_information
 GROUP BY weight_class;
```

```
WEIGHT_CLASS CORR(LIST_PRICE,MIN_PRICE)
------------ -------------------------
           1                .99914795
           2               .999022941
           3               .998484472
           4               .999359909
           5               .999536087
```

补充：
这个查询与下面的查询等价：

```
SELECT weight_class,
       covar_pop(list_price, min_price) /
```

```
      (stddev_pop(list_price) * stddev_pop(min_price))
  FROM product_information
 WHERE list_price IS NOT NULL
   AND min_price IS NOT NULL
 GROUP BY weight_class;
```

3.4 Analytic Example

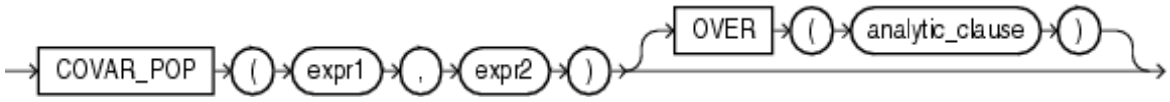
The following example shows the correlation between duration at the company and salary by the employee's position. The result set shows the same correlation for each employee in a given job:

下面的例子显示了不同职务的雇员的工龄与薪水之间的相关性。结果表明职务相同的雇员有相同的相关性：

```
SELECT employee_id,
       job_id,
       to_char((SYSDATE - hire_date) YEAR TO MONTH) "Yrs-Mns",
       salary,
       corr(SYSDATE - hire_date, salary) over(PARTITION BY job_id) AS
"Correlation"
  FROM employees
 WHERE department_id IN (50, 80)
 ORDER BY job_id, employee_id;

EMPLOYEE_ID JOB_ID     Yrs-Mns     SALARY Correlation
----------- ---------- ------- ---------- -----------
        145 SA_MAN     +08-07       14000  .912385598
        146 SA_MAN     +08-04       13500  .912385598
        147 SA_MAN     +08-02       12000  .912385598
        148 SA_MAN     +05-07       11000  .912385598
        149 SA_MAN     +05-03       10500  .912385598
        150 SA_REP     +08-03       10000   .80436755
        151 SA_REP     +08-02        9500   .80436755
        152 SA_REP     +07-09        9000   .80436755
        153 SA_REP     +07-01        8000   .80436755
        154 SA_REP     +06-05        7500   .80436755
        155 SA_REP     +05-06        7000   .80436755
...
```

4. COVAR_POP

4.1 Syntax




```
COVAR_POP(expr1, expr2)
   [ OVER (analytic_clause) ]
```

> **See Also:**
> "Analytic Functions" for information on syntax, semantics, and restrictions

4.2 Purpose

COVAR_POP returns the population covariance of a set of number pairs. You can use it as an aggregate or analytic function.

Covar_pop返回一组数值对的总体协方差。它可以用作聚集或分析函数。

This function takes as arguments any numeric datatype or any nonnumeric datatype that can be implicitly converted to a numeric datatype. Oracle determines the argument with the highest numeric precedence, implicitly converts the remaining arguments to that datatype, and returns that datatype.

函数参数可取任何数字类型或任何可以隐式转换为数字类型的非数字类型。Oracle根据最高数字优先级确定参数，隐式地将需要处理的参数转换为数字类型，并返回数字类型。

> **See Also:**
> Table 2-10, "Implicit Type Conversion Matrix" for more information on implicit conversion and "Numeric Precedence" for information on numeric precedence

Oracle Database applies the function to the set of (*expr1*, *expr2*) pairs after eliminating all pairs for which either *expr1* or *expr2* is null. Then Oracle makes the following computation:

Oracle数据库使用该函数前先排除(expr1,expr2)集中所有expr1或expr2为null的数值对。然后作如下的计算：

```
(SUM(expr1 * expr2) - SUM(expr2) * SUM(expr1) / n) / n
```

where *n* is the number of (*expr1*, *expr2*) pairs where neither *expr1* nor *expr2* is null.

这里n是(expr1,expr2)数值对的个数，expr1和expr2都不能为null。

The function returns a value of type NUMBER. If the function is applied to an empty set, then it returns null.

函数返回一个number类型的值。若将此函数应用在一个空集上，那末它将返回null。

> See Also:
> "About SQL Expressions" for information on valid forms of *expr* and "Aggregate Functions"

4.3 Aggregate Example

The following example calculates the population covariance and sample covariance for time employed (SYSDATE - hire_date) and salary using the sample table hr.employees:

下面的例子计算hr.employees表中不同职务雇员的雇佣时间和薪水的总体协方差和样本协方差：

```
SELECT job_id,
      covar_pop(SYSDATE - hire_date, salary) AS covar_pop,
      covar_samp(SYSDATE - hire_date, salary) AS covar_samp
  FROM employees
 WHERE department_id IN (50, 80)
 GROUP BY job_id;

JOB_ID       COVAR_POP  COVAR_SAMP
---------- ---------- ----------
ST_MAN       436092.000  545115.000
SH_CLERK     782717.500  823913.158
```

```
SA_MAN       660700.000   825875.000
SA_REP       579988.466   600702.340
ST_CLERK     176577.250   185870.789
```

4.4 Analytic Example

The following example calculates cumulative sample covariance of the list price and minimum price of the products in the sample schema oe:

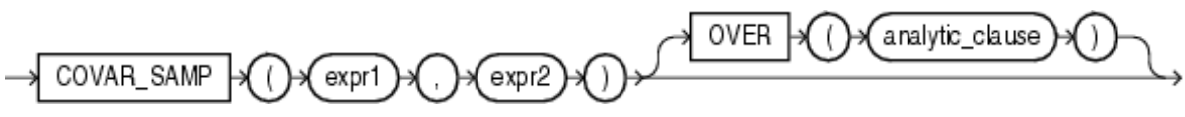
下面的例子计算oe模式中不同产品的订价和最低价格的累计样本协方差：

```
SELECT product_id,
       supplier_id,
       covar_pop(list_price, min_price) over(ORDER BY product_id,
supplier_id) AS cum_covp,
       covar_samp(list_price, min_price) over(ORDER BY product_id,
supplier_id) AS cum_covs
  FROM product_information p
 WHERE category_id = 29
 ORDER BY product_id, supplier_id;
```

```
PRODUCT_ID SUPPLIER_ID   CUM_COVP   CUM_COVS
---------- ----------- ---------- ----------
      1774      103088          0
      1775      103087    1473.25     2946.5
      1794      103096 1702.77778 2554.16667
      1825      103093    1926.25 2568.33333
      2004      103086     1591.4    1989.25
      2005      103086     1512.5       1815
      2416      103088 1475.97959 1721.97619
. . .
```

5. COVAR_SAMP

5.1 Syntax




```
COVAR_SAMP(expr1, expr2)
  [ OVER (analytic_clause) ]
```

> **See Also:**
> "Analytic Functions" for information on syntax, semantics, and restrictions

5.2 Purpose

COVAR_SAMP returns the sample covariance of a set of number pairs. You can use it as an aggregate or analytic function.

Covar_samp返回一组数值对的样本协方差。它可用作聚集或分析函数。

This function takes as arguments any numeric datatype or any nonnumeric datatype that can be implicitly converted to a numeric datatype. Oracle determines the argument with the highest numeric precedence, implicitly converts the remaining arguments to that datatype, and returns that datatype.

函数参数可取任何数字类型或任何可以隐式转换为数字类型的非数字类型。Oracle根据最高数字优先级确定参数，隐式地将需要处理的参数转换为数字类型，并返回数字类型。

> **See Also:**
> Table 2-10, "Implicit Type Conversion Matrix" for more information on implicit conversion and "Numeric Precedence" for information on numeric precedence

Oracle Database applies the function to the set of (*expr1*, *expr2*) pairs after eliminating all pairs for which either *expr1* or *expr2* is null. Then Oracle makes the following computation:

Oracle数据库使用该函数前先排除(expr1,expr2)集中所有expr1或expr2为null的数值对。然后作如下的计算：

```
(SUM(expr1 * expr2) - SUM(expr1) * SUM(expr2) / n) / (n-1)
```

where *n* is the number of (*expr1*, *expr2*) pairs where neither *expr1* nor *expr2* is null.

这里n是(expr1,expr2)数值对的个数，expr1和expr2都不能为null。

The function returns a value of type NUMBER. If the function is applied to an empty set, then it returns null.

函数返回一个number类型的值。若将此函数应用在一个空集上，那末它将返回null。

> **See Also:**
> "About SQL Expressions" for information on valid forms of *expr* and "Aggregate Functions"

5.3 Aggregate Example

Please refer to the aggregate example for COVAR_POP.

请参阅covar_pop聚集函数例子。

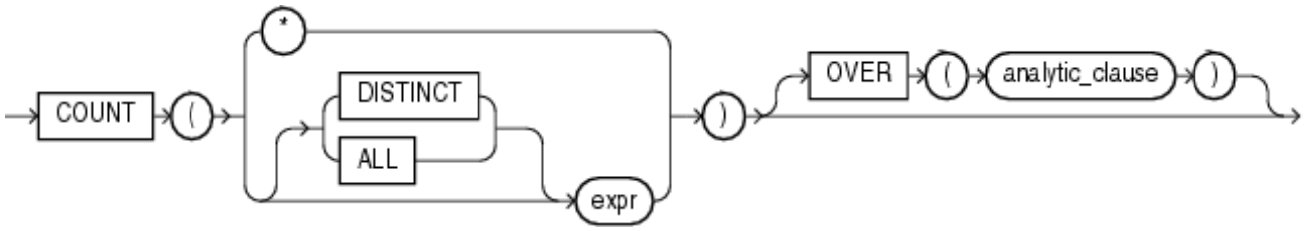
5.4 Analytic Example

Please refer to the analytic example for COVAR_POP.

请参阅cova_pop分析函数例子。

6. COUNT

6.1 Syntax




```
COUNT({ * | [ DISTINCT | ALL ] expr })
   [ OVER (analytic_clause) ]
```

> **See Also:**

"Analytic Functions" for information on syntax, semantics, and restrictions

6.2 Purpose

COUNT returns the number of rows returned by the query. You can use it as an aggregate or analytic function.

Count返回查询结果集的行数。它可以用作聚集或分析函数。

If you specify DISTINCT, then you can specify only the *query_partition_clause* of the *analytic_clause*. The *order_by_clause* and *windowing_clause* are not allowed.

Distinct关键字仅能在analytic_clause的query_partition_clause中使用。在order_by_clause和windowing_clause中不允许使用distinct。

If you specify *expr*, then COUNT returns the number of rows where *expr* is not null. You can count either all rows, or only distinct values of *expr*.

若expr作为函数参数，那末count不计算expr为null的行。函数要么计算所有行，要么仅计算expr的不同值。

If you specify the asterisk (*), then this function returns all rows, including duplicates and nulls. COUNT never returns null.

若星号(*)作为函数参数，那末函数返回包括数据重复的行和数据为null的行在内的所有行数。Count绝不返回null。

See Also:
"About SQL Expressions" for information on valid forms of *expr* and "Aggregate Functions"

6.3 Aggregate Examples

The following examples use COUNT as an aggregate function:

下面是count用作聚集函数的若干例子：

```
SELECT COUNT(*) "Total" FROM employees;
```

```
     Total
----------
       107
```

```
SELECT COUNT(*) "Allstars" FROM employees WHERE commission_pct > 0;
```

```
 Allstars
---------
       35
```

```
SELECT COUNT(commission_pct) "Count" FROM employees;
```

```
     Count
----------
        35
```

```
SELECT COUNT(DISTINCT manager_id) "Managers" FROM employees;
```

```
  Managers
----------
        18
```

6.4 Analytic Example

The following example calculates, for each employee in the `employees` table, the moving count of employees earning salaries in the range 50 less than through 150 greater than the employee's salary.

下面的例子计算`employees`表每个雇员与雇员自己薪水相差在`50`至`150`之间的雇员的个数。

```
SELECT last_name,
      salary,
      COUNT(*) over(ORDER BY salary RANGE BETWEEN 50 preceding AND 150
following) AS mov_count
  FROM employees;
```

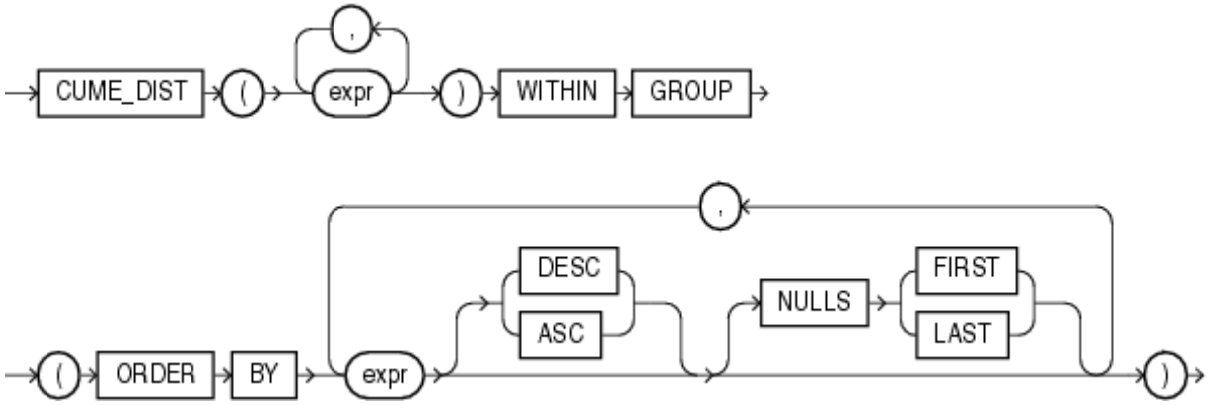
```
LAST_NAME                     SALARY  MOV_COUNT
------------------------- ---------- ----------
Olson                           2100          3
Markle                          2200          2
Philtanker                      2200          2
Landry                          2400          8
Gee                             2400          8
Colmenares                      2500         10
```

```
Patel                          2500          10
. . .
```

7. CUME_DIST

7.1 Aggregate Syntax

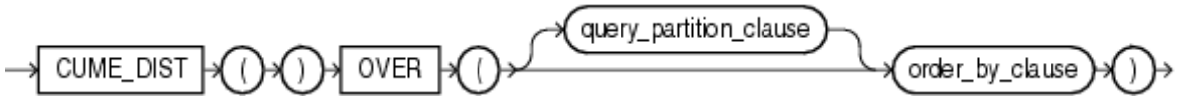
cume_dist_aggregate::=




```
CUME_DIST(expr[,expr ]...)
   WITHIN GROUP
   (ORDER BY expr [ DESC | ASC ]
                  [ NULLS { FIRST | LAST } ]
             [, expr [ DESC | ASC ]
                     [ NULLS { FIRST | LAST } ]
             ]...
   )
```

7.2 Analytic Syntax

cume_dist_analytic::=




```
CUME_DIST( )
   OVER ([ query_partition_clause ] order_by_clause)
```

> **See Also:**
> "Analytic Functions" for information on syntax, semantics, and restrictions

7.3 Purpose

CUME_DIST calculates the cumulative distribution of a value in a group of values. The range of values returned by CUME_DIST is >0 to <=1. Tie values always evaluate to the same cumulative distribution value.

Cume_dist计算一个值在一组值中的累计分布。Cume_dist返回值的范围为(0,1]。连接值总是对相同的累积值进行求值。

This function takes as arguments any numeric datatype or any nonnumeric datatype that can be implicitly converted to a numeric datatype. Oracle Database determines the argument with the highest numeric precedence, implicitly converts the remaining arguments to that datatype, makes the calculation, and returns NUMBER.

函数参数可取任何数字类型或任何可以隐式转换为数字类型的非数字类型。Oracle根据最高数字优先级确定参数，隐式地将需要处理的参数转换为数字类型，然后进行计算，并返回number类型的值。

> **See Also:**
> Table 2-10, "Implicit Type Conversion Matrix" for more information on implicit conversion and "Numeric Precedence" for information on numeric precedence

- As an aggregate function, CUME_DIST calculates, for a hypothetical row *r* identified by the arguments of the function and a corresponding sort specification, the relative position of row *r* among the rows in the aggregation group. Oracle makes this calculation as if the hypothetical row *r* were inserted into the group of rows to be aggregated over. The arguments of the function identify a single hypothetical row within each aggregate group. Therefore, they must all evaluate to constant expressions within each aggregate group. The constant argument expressions and the expressions in the ORDER BY clause of the aggregate match by position. Therefore, the number of arguments must be the same and their types must be compatible.

  CUME_DIST 用作聚集函数时，对于一个被函数参数和相应排序规则确定的假定行 r，cume_dist 计算此假定行 r 在聚集分组行中的相对位置。Oracle 对此进行计算时，

就好像假定行 r 插入了被聚集的行组中一样。函数参数只确定聚集分组内的一个假定行。因此，它们必须对每个聚集分组中的常量表达式全部求值。常量参数表达式和聚集的 order by 子句中的表达式按位置进行匹配。因此，两者参数个数必须相同，类型必须兼容。

As an analytic function, CUME_DIST computes the relative position of a specified value in a group of values. For a row *r*, assuming ascending ordering, the CUME_DIST of *r* is the number of rows with values lower than or equal to the value of *r*, divided by the number of rows being evaluated (the entire query result set or a partition).

CUME_DIST 用作分析函数时，用于计算一个值在一组值中的相对位置。假定按升序排序的一个结果集或分组中存在一行 r，cume_dist()在 r 上结果如是求得：值小于等于行 r 上值的行的行数，除以整个查询结果集或分组的行数。

7.4 Aggregate Example

The following example calculates the cumulative distribution of a hypothetical employee with a salary of $15,500 and commission rate of 5% among the employees in the sample table oe.employees:

下面的例子计算oe.employees表中薪水达到$15500并且佣金率达到5%的假定雇员的累计分布值：

```
SELECT cume_dist(15500, .05) within
 GROUP(
 ORDER BY salary, commission_pct) "Cume-Dist of 15500"
  FROM employees;

Cume-Dist of 15500
------------------
        .972222222
```

7.5 Analytic Example

The following example calculates the salary percentile for each employee in the purchasing division. For example, 40% of clerks have salaries less than or equal to Himuro.

下面的例子计算每个采购科雇员的薪水百分点。例如，40%的职员的薪水少于或等于 Himur。

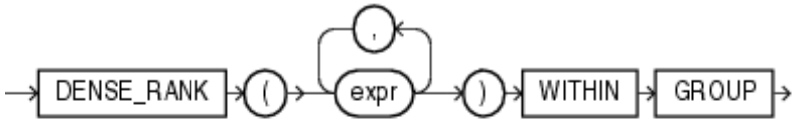
```
SELECT job_id,
       last_name,
       salary,
```

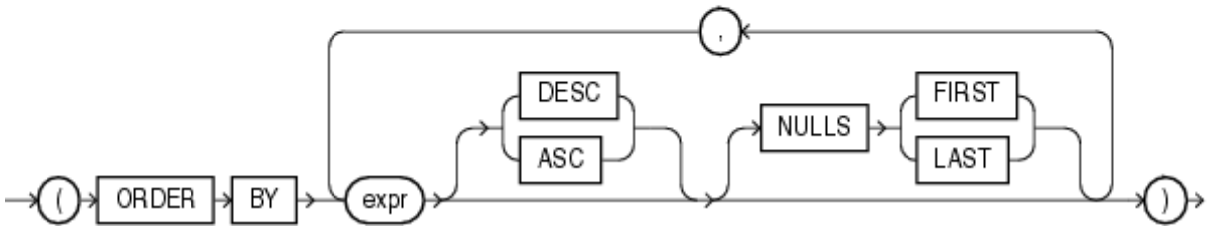
```
      cume_dist() over(PARTITION BY job_id ORDER BY salary) AS
cume_dist
  FROM employees
 WHERE job_id LIKE 'PU%';

JOB_ID     LAST_NAME                 SALARY  CUME_DIST
---------- ------------------------- ---------- ----------
PU_CLERK   Colmenares                  2500         .2
PU_CLERK   Himuro                      2600         .4
PU_CLERK   Tobias                      2800         .6
PU_CLERK   Baida                       2900         .8
PU_CLERK   Khoo                        3100          1
PU_MAN     Raphaely                   11000          1
```

8. DENSE_RANK

8.1 Aggregate Syntax

dense_rank_aggregate::=






```
DENSE_RANK(expr [, expr ]...) WITHIN GROUP
  (ORDER BY expr [ DESC | ASC ]
                 [ NULLS { FIRST | LAST } ]
             [,expr [ DESC | ASC ]
                    [ NULLS { FIRST | LAST } ]
             ]...
  )
```

8.2 Analytic Syntax

dense_rank_analytic::=


```
DENSE_RANK( )
   OVER([ query_partition_clause ] order_by_clause)
```

> **See Also:**
> "Analytic Functions" for information on syntax, semantics, and restrictions

8.3 Purpose

DENSE_RANK computes the rank of a row in an ordered group of rows and returns the rank as a NUMBER. The ranks are consecutive integers beginning with 1. The largest rank value is the number of unique values returned by the query. Rank values are not skipped in the event of ties. Rows with equal values for the ranking criteria receive the same rank. This function is useful for top-N and bottom-N reporting.

Dense_rank 计算有序组中行的排名，返回的排名是一个 number 数值。排名是从 1 开始的连续整数。排名的最大值是查询返回的唯一值的个数。排名一旦与行关联就不会产生跳跃的值。值相等的行排名相同。此函数对于计算 top-N 和 bottom-N 报表十分有用。

This function accepts as arguments any numeric datatype and returns NUMBER.

函数接受任何数字类型的参数并返回 number 类型。

- As an aggregate function, DENSE_RANK calculates the dense rank of a hypothetical row identified by the arguments of the function with respect to a given sort specification. The arguments of the function must all evaluate to constant expressions within each aggregate group, because they identify a single row within each group. The constant argument expressions and the expressions in the *order_by_clause* of the aggregate match by position. Therefore, the number of arguments must be the same and types must be compatible.

  Dense_rank 用作聚集函数时，它计算由一个带有排序规则的函数参数确定的假定行的密集排名。函数参数必须对每个聚集分组中的常量表达式全部求值。常量参数表达式和聚集的 order by 子句中的表达式按位置进行匹配。因此，参数个数必须相同，参数类型必须兼容。

- As an analytic function, DENSE_RANK computes the rank of each row returned from a query with respect to the other rows, based on the values of the *value_exprs* in the *order_by_clause*.

  Dense_rank 用作分析函数时，它计算按照 order_by_clause 中 value_exprs 值排序返回的查询结果中，每一行相对于其他行的排名。

8.4 Aggregate Example

The following example computes the ranking of a hypothetical employee with the salary $15,500 and a commission of 5% in the sample table oe.employees:

下面的例子计算 oe.employees 表中薪水达到$15500 并且佣金达到 5%的假定雇员的排名：

```
SELECT dense_rank(15500, .05) within
 GROUP(
 ORDER BY salary DESC, commission_pct) "Dense Rank"
  FROM employees;

        Dense Rank
------------------
                 3
```

8.5 Analytic Example

The following statement selects the department name, employee name, and salary of all employees who work in the human resources or purchasing department, and then computes a rank for each unique salary in each of the two departments. The salaries that are equal receive the same rank. Compare this example with the example for RANK.

下面的语句在在人力资源或采购部门中，选择部门名称，雇员名称，雇员薪水，然后对这两个部门中每个唯一的薪水值排名。薪水相等则排名相同。请将本例与 rank 示例比较。

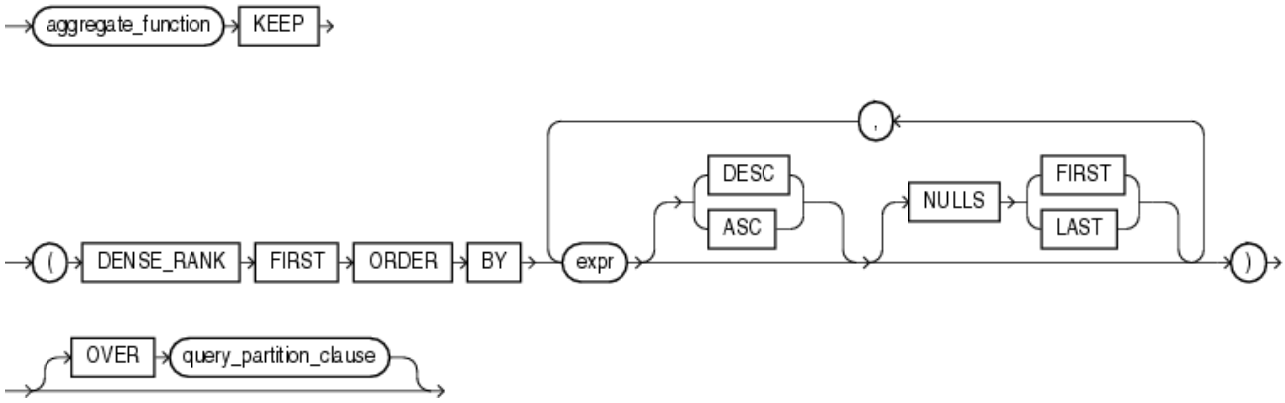
```
SELECT d.department_name,
      e.last_name,
      e.salary,
      dense_rank() over(PARTITION BY e.department_id ORDER BY
e.salary) AS drank
  FROM employees e, departments d
 WHERE e.department_id = d.department_id
   AND d.department_id IN ('30', '40');

DEPARTMENT_NAME          LAST_NAME                 SALARY       DRANK
```

```
--------------------- ----------------- ---------- ----------
Purchasing            Colmenares              2500          1
Purchasing            Himuro                  2600          2
Purchasing            Tobias                  2800          3
Purchasing            Baida                   2900          4
Purchasing            Khoo                    3100          5
Purchasing            Raphaely               11000          6
Human Resources       Marvis                  6500
```

9. FIRST

9.1 Syntax

first::=




```
aggregate_function
   KEEP
   (DENSE_RANK FIRST ORDER BY
    expr [ DESC | ASC ]
         [ NULLS { FIRST | LAST } ]
    [, expr [ DESC | ASC ]
            [ NULLS { FIRST | LAST } ]
    ]...
   )
   [ OVER query_partition_clause ]
```

See Also:

"Analytic Functions" for information on syntax, semantics, and restrictions of the ORDER BY clause and OVER clause

9.2 Purpose

FIRST and LAST are very similar functions. Both are aggregate and analytic functions that operate on a set of values from a set of rows that rank as the FIRST or LAST with respect to a given sorting specification. If only one row ranks as FIRST or LAST, the aggregate operates on the set with only one element.

First 和 last 是非常类似的函数。它们都可用作聚集和分析函数，操作按排序规则排名后的行组中排名为 first 或 last 的值。若分组中只有排名为 first 或 last 的行，那末只对这个唯一元素进行聚集操作(意思是说，当分组中只有一行记录时，不论 first 或 last 都取这一行——译者注)。

This function takes as an argument any numeric datatype or any nonnumeric datatype that can be implicitly converted to a numeric datatype. The function returns the same datatype as the numeric datatype of the argument.

函数参数可取任何数字类型或者是任何可以隐式转换为数字类型的非数字类型。函数返回类型与参数类型相同，都为数字类型。

When you need a value from the first or last row of a sorted group, but the needed value is not the sort key, the FIRST and LAST functions eliminate the need for self-joins or views and enable better performance.

当已排序组中第一行或最后一行的值不是排序键值时，为了获得更好的性能，first 和 last 函数不会进行自连接或产生视图。

- The *aggregate_function* is any one of the MIN, MAX, SUM, AVG, COUNT, VARIANCE, or STDDEV functions. It operates on values from the rows that rank either FIRST or LAST. If only one row ranks as FIRST or LAST, the aggregate operates on a singleton (nonaggregate) set.

  aggregate_function 可以是 min,max,sum,avg,count,variance,或 stddev 函数中的任一个。它操作组中排名为 first 或 last 值。若分组中只有排名为 first 或 last 的行，那末只对这个唯一元素进行聚集操作。

- The KEEP keyword is for semantic clarity. It qualifies *aggregate_function*, indicating that only the FIRST or LAST values of *aggregate_function* will be returned.

使用 Keep 关键字是为了保持语义清晰。它限制 aggregate_function，表示仅返回 aggregate_function 的 first 或 last 值。

- DENSE_RANK FIRST or DENSE_RANK LAST indicates that Oracle Database will aggregate over only those rows with the minimum (FIRST) or the maximum (LAST) dense rank (also called olympic rank).

  Dense_rank_first 或 dense_rank_last 表明 Oracle 数据库仅将排名为最小(first)或最大(last)的行聚集在一起。

You can use the FIRST and LAST functions as analytic functions by specifying the OVER clause. The *query_partitioning_clause* is the only part of the OVER clause valid with these functions.

First 和 last 函数中指定 over 子句可用作分析函数。在这两个分析函数的 over 子句中仅能使用 query_partitioning_clause。

> **See Also:**
> Table 2-10, "Implicit Type Conversion Matrix" for more information on implicit conversion and LAST

9.3 Aggregate Example

The following example returns, within each department of the sample table hr.employees, the minimum salary among the employees who make the lowest commission and the maximum salary among the employees who make the highest commission:

下面的例子返回 hr.employees 表中每个部门佣金最少雇员的最低薪水以及佣金最高雇员的最高薪水：

```
SELECT department_id,
      MIN(salary) keep(dense_rank FIRST ORDER BY commission_pct)
"Worst",
      MAX(salary) keep(dense_rank LAST ORDER BY commission_pct)
"Best"
  FROM employees
 GROUP BY department_id;

DEPARTMENT_ID      Worst       Best
------------- ---------- ----------
           10       4400       4400
```

```
        20       6000      13000
        30       2500      11000
        40       6500       6500
        50       2100       8200
        60       4200       9000
        70      10000      10000
        80       6100      14000
        90      17000      24000
       100       6900      12000
       110       8300      12000
                 7000       7000
```

9.4 Analytic Example

The next example makes the same calculation as the previous example but returns the result for each employee within the department:

接下的例子对前例作相同的计算，但返回的是部门中每个雇员的薪水：

```
SELECT last_name,
      department_id,
      salary,
      MIN(salary) keep(dense_rank FIRST ORDER BY commission_pct)
over(PARTITION BY department_id) "Worst",
      MAX(salary) keep(dense_rank LAST ORDER BY commission_pct)
over(PARTITION BY department_id) "Best"
  FROM employees
 ORDER BY department_id, salary;
```

```
LAST_NAME           DEPARTMENT_ID     SALARY      Worst       Best
------------------- ------------- ---------- ---------- ----------
Whalen                         10       4400       4400       4400
Fay                            20       6000       6000      13000
Hartstein                      20      13000       6000      13000
. . .
Gietz                         110       8300       8300      12000
Higgins                       110      12000       8300      12000
Grant                                   7000       7000       7000
```

10. FIRST_VALUE

10.1 Syntax


```
FIRST_VALUE (expr [ IGNORE NULLS ])
   OVER (analytic_clause)
```

> **See Also:**
> "Analytic Functions" for information on syntax, semantics, and restrictions, including valid forms of *expr*

10.2 Purpose

FIRST_VALUE is an analytic function. It returns the first value in an ordered set of values. If the first value in the set is null, then the function returns NULL unless you specify IGNORE NULLS. This setting is useful for data densification. If you specify IGNORE NULLS, then FIRST_VALUE returns the fist non-null value in the set, or NULL if all values are null. Please refer to "Using Partitioned Outer Joins: Examples" for an example of data densification.

First_value 只用作分析函数。它返回已排序集的第一个值。若集合中的第一个值为 null，除非指定 ignor nulls 那末函数返回 null。忽略空值的限定对稠化数据很有用处。若指定 ignor nulls，那末 first_value 函数返回集合中第一个不为 null 的值，或若值全为 null 则返回 null。请参阅 Using Partioned Outer Joins:Examples 中关于稠化数据的例子。

You cannot use FIRST_VALUE or any other analytic function for *expr*. That is, you cannot nest analytic functions, but you can use other built-in function expressions for *expr*. Please refer to "About SQL Expressions" for information on valid forms of *expr*.

不能在 expr 中使用 first_value 或其他任何分析函数。也就是说，此处分析函数不能嵌套，但可以在 expr 中使用内置函数表达式。请参阅 About SQL Expressions 获取合法 expr 的相关信息。

10.3 Examples

The following example selects, for each employee in Department 90, the name of the employee with the lowest salary.

下面的例子，选出部门 90 中薪水最低的每一雇员的名字：

```
SELECT department_id,
       last_name,
       salary,
       first_value(last_name) over(ORDER BY salary ASC rows unbounded
preceding) AS lowest_sal
  FROM (SELECT *
          FROM employees
         WHERE department_id = 90
         ORDER BY employee_id);
```

```
DEPARTMENT_ID LAST_NAME          SALARY LOWEST_SAL
------------- ------------- ---------- -------------------------
           90 Kochhar            17000 Kochhar
           90 De Haan            17000 Kochhar
           90 King               24000 Kochhar
```

The example illustrates the nondeterministic nature of the `FIRST_VALUE` function. Kochhar and DeHaan have the same salary, so are in adjacent rows. Kochhar appears first because the rows returned by the subquery are ordered by `employee_id`. However, if the rows returned by the subquery are ordered by `employee_id` in descending order, as in the next example, then the function returns a different value:

这个例子表明了 `first_name` 函数的不确定性。`Kochhar` 和 `De Haan` 有相同的薪水，因此在行中位置相邻。`Kochhar` 出现在第一行，因为行是通过按 `employee_id` 排序的子查询返回的。然而，若行是通过按 `employee_id` 降序排序的子查询返回的，正如下面的例子一样，那末函数返回的值不同：

```
SELECT department_id,
       last_name,
       salary,
       first_value(last_name) over(ORDER BY salary ASC rows unbounded
preceding) AS fv
  FROM (SELECT *
          FROM employees
         WHERE department_id = 90
         ORDER BY employee_id DESC);
```

```
DEPARTMENT_ID LAST_NAME          SALARY FV
------------- ------------- --------- ----------------------
           90 De Haan             17000 De Haan
           90 Kochhar             17000 De Haan
           90 King                24000 De Haan
```

The following example shows how to make the `FIRST_VALUE` function deterministic by ordering on a unique key.

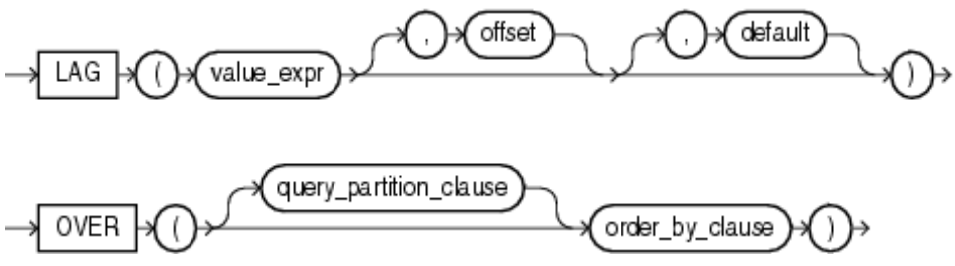
下面的例子说明怎样通过一个唯一键值排序使 `first_value` 函数具有确定性：

```
SELECT department_id,
       last_name,
       salary,
       hire_date,
       first_value(last_name) over(ORDER BY salary ASC, hire_date rows
unbounded preceding) AS fv
  FROM (SELECT *
          FROM employees
         WHERE department_id = 90
         ORDER BY employee_id DESC);
```

```
DEPARTMENT_ID LAST_NAME          SALARY HIRE_DATE FV
------------- ------------- --------- --------- ---------------
           90 Kochhar             17000 21-SEP-89 Kochhar
           90 De Haan             17000 13-JAN-93 Kochhar
           90 King                24000 17-JUN-87 Kochhar
```

11. LAG

11.1 Syntax



```
LAG(value_expr [, offset ] [, default ])
   OVER ([ query_partition_clause ] order_by_clause)
```

> **See Also:**
> "Analytic Functions" for information on syntax, semantics, and restrictions, including valid forms of *value_expr*

11.2 Purpose

LAG is an analytic function. It provides access to more than one row of a table at the same time without a self join. Given a series of rows returned from a query and a position of the cursor, LAG provides access to a row at a given physical offset prior to that position.

Lag 只能用作分析函数。它提供在不使用自连接的情况下访问表中多个行的途径。给定要查询的行组和一个位置指针，lag 能根据给定的物理偏移量访问前面位置的行。

If you do not specify *offset*, then its default is 1. The optional *default* value is returned if the offset goes beyond the scope of the window. If you do not specify *default*, then its default is null.

若不指定 offset，那末其默认为 1。若偏移量超出窗口范围，则返回可选的 default 值。若没有指定 default，那末其默认为 null。

You cannot use LAG or any other analytic function for *value_expr*. That is, you cannot nest analytic functions, but you can use other built-in function expressions for *value_expr*.

不能在 value_expr 中使用 lag 或其他任何分析函数。也就是说，此处分析函数不能嵌套。但是可以在 value_expr 中使用内置函数表达式。

> **See Also:**
> "About SQL Expressions" for information on valid forms of *expr* and LEAD

11.3 Examples

The following example provides, for each salesperson in the employees table, the salary of the employee hired just before:

下面的例子提供 employees 表中每个雇员正前一个被雇佣的雇员薪水：

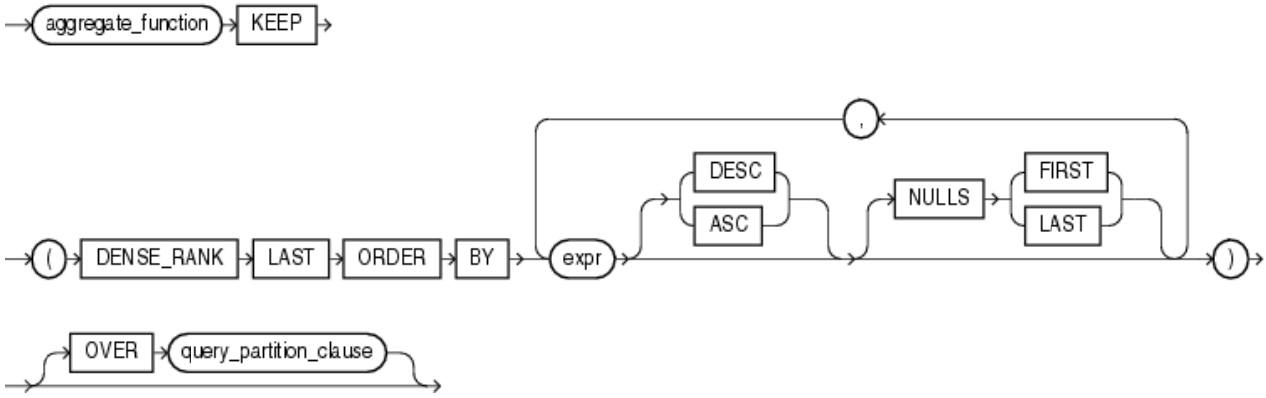
```
SELECT last_name,
       hire_date,
       salary,
       lag(salary, 1, 0) over(ORDER BY hire_date) AS prev_sal
  FROM employees
 WHERE job_id = 'PU_CLERK';
```

LAST_NAME	HIRE_DATE	SALARY	PREV_SAL
Khoo	18-MAY-95	3100	0
Tobias	24-JUL-97	2800	3100
Baida	24-DEC-97	2900	2800
Himuro	15-NOV-98	2600	2900
Colmenares	10-AUG-99	2500	2600

12. LAST

12.1 Syntax

last::=




```
aggregate_function KEEP
   (DENSE_RANK LAST ORDER BY
    expr [ DESC | ASC ]
         [ NULLS { FIRST | LAST } ]
    [, expr [ DESC | ASC ]
            [ NULLS { FIRST | LAST } ]
    ]...
```

```
)
[ OVER query_partition_clause ]
```

> **See Also:**
> "Analytic Functions" for information on syntax, semantics, and restrictions of the *query_partitioning_clause*

12.2 Purpose

FIRST and LAST are very similar functions. Both are aggregate and analytic functions that operate on a set of values from a set of rows that rank as the FIRST or LAST with respect to a given sorting specification. If only one row ranks as FIRST or LAST, the aggregate operates on the set with only one element.

First 和 last 是非常类似的函数。它们都可以用作聚集和分析函数，操作按排序规则排名后行组中排名为 first 或 last 的值。若分组中只有排名为 first 或 last 的行，那末只对这个唯一元素进行聚集操作(意思是说，当分组中只有一行记录时，不论 first 或 last 都取这一行——译者注)。

This function takes as an argument any numeric datatype or any nonnumeric datatype that can be implicitly converted to a numeric datatype. The function returns the same datatype as the numeric datatype of the argument.

函数参数可取任何数字类型或者是任何可以隐式转换为数字类型的非数字类型。函数返回类型与参数类型相同，都为数字类型。

> **See Also:**
> Table 2-10, "Implicit Type Conversion Matrix" for more information on implicit conversion

Please refer to FIRST for complete information on this function and for examples of its use.

函数使用的全部信息和用例请参阅first函数。

13. LAST_VALUE

13.1 Syntax


```
LAST_VALUE(expr [ IGNORE NULLS ])
   OVER (analytic_clause)
```

> **See Also:**
> "Analytic Functions" for information on syntax, semantics, and restrictions, including valid forms of *expr*

13.2 Purpose

LAST_VALUE is an analytic function. It returns the last value in an ordered set of values. If the last value in the set is null, then the function returns NULL unless you specify IGNORE NULLS. This setting is useful for data densification. If you specify IGNORE NULLS, then LAST_VALUE returns the fist non-null value in the set, or NULL if all values are null. Please refer to "Using Partitioned Outer Joins: Examples" for an example of data densification.

Last_value 只用作分析函数。它返回已排序集的最后一个值。若集合中最后一个值为空，除非指定为 ignore nulls，否则返回 null。忽略空值的限定对稠化数据很有用处。若指定 ignor nulls，那末 last_value 返回集合中最后一个不为 null 的值，或若值全为 null 则返回 null。请参阅 Using Partioned Outer Joins:Examples 中关于稠化数据的例子。

You cannot use LAST_VALUE or any other analytic function for *expr*. That is, you cannot nest analytic functions, but you can use other built-in function expressions for *expr*. Please refer to "About SQL Expressions" for information on valid forms of *expr*.

不能在 expr 中使用 last_value 或其他任何分析函数。也就是说，此处分析函数不能嵌套，但可以在 expr 中使用内置函数表达式。请参阅 About SQL Expressions 获取合法 expr 的相关信息。

13.3 Examples

The following example returns, for each row, the hire date of the employee earning the highest salary:

下面的例子返回雇佣期间薪水最高的雇员：

```
SELECT last_name,
       salary,
       hire_date,
       last_value(hire_date) over(ORDER BY salary rows BETWEEN
unbounded preceding AND unbounded following) AS lv
  FROM (SELECT * FROM employees WHERE department_id = 90 ORDER BY
hire_date);
```

```
LAST_NAME                 SALARY HIRE_DATE LV
------------------------- ---------- --------- ---------
Kochhar                        17000 21-SEP-89 17-JUN-87
De Haan                        17000 13-JAN-93 17-JUN-87
King                           24000 17-JUN-87 17-JUN-87
```

This example illustrates the nondeterministic nature of the `LAST_VALUE` function. Kochhar and De Haan have the same salary, so they are in adjacent rows. Kochhar appears first because the rows in the subquery are ordered by `hire_date`. However, if the rows are ordered by `hire_date` in descending order, as in the next example, then the function returns a different value:

这个例子表明了 `last_name` 函数的不确定性。`Kochhar` 和 `De Haan` 有相同的薪水，因此在行中位置相邻。`Kochhar` 出现在第一行，因为行是通过按 `hire_date` 排序的子查询返回的。然而若行是通过按 `hire_date` 降序排序的子查询返回的，正如下面的例子一样，那末函数返回的值不同：

```
SELECT last_name,
       salary,
       hire_date,
       last_value(hire_date) over(ORDER BY salary rows BETWEEN
unbounded preceding AND unbounded following) AS lv
  FROM (SELECT *
          FROM employees
         WHERE department_id = 90
         ORDER BY hire_date DESC);
```

```
LAST_NAME                 SALARY HIRE_DATE LV
```

```
-------------------- --------- --------- ---------
De Haan                 17000 13-JAN-93 17-JUN-87
Kochhar                 17000 21-SEP-89 17-JUN-87
King                    24000 17-JUN-87 17-JUN-87
```

The following two examples show how to make the `LAST_VALUE` function deterministic by ordering on a unique key. By ordering within the function by both `salary` and `hire_date`, you can ensure the same result regardless of the ordering in the subquery.

下面的两个例子说明怎样通过一个唯一键值排序使 `last_value` 函数具有确定性。通过在函数中使用`salary`和`hire_date`排序，不论子查询中排序如何，都能确保查询结果一致。

```
SELECT last_name,
       salary,
       hire_date,
       last_value(hire_date) over(ORDER BY salary, hire_date rows
BETWEEN unbounded preceding AND unbounded following) AS lv
  FROM (SELECT * FROM employees WHERE department_id = 90 ORDER BY
hire_date);
```

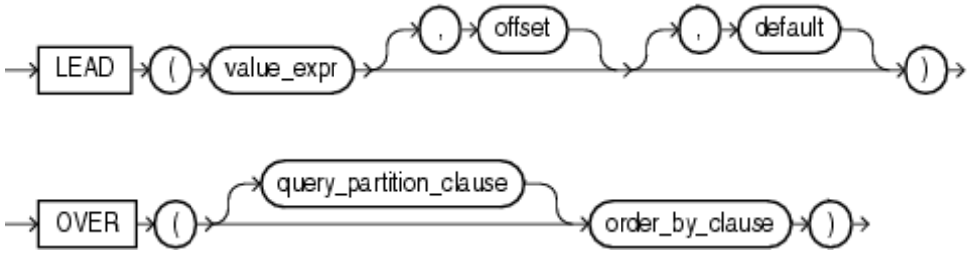
```
LAST_NAME                 SALARY HIRE_DATE LV
-------------------- --------- --------- ---------
Kochhar                 17000 21-SEP-89 17-JUN-87
De Haan                 17000 13-JAN-93 17-JUN-87
King                    24000 17-JUN-87 17-JUN-87
```

```
SELECT last_name,
       salary,
       hire_date,
       last_value(hire_date) over(ORDER BY salary, hire_date rows
BETWEEN unbounded preceding AND unbounded following) AS lv
  FROM (SELECT *
          FROM employees
         WHERE department_id = 90
         ORDER BY hire_date DESC);
```

```
LAST_NAME                 SALARY HIRE_DATE LV
-------------------- --------- --------- ---------
Kochhar                 17000 21-SEP-89 17-JUN-87
De Haan                 17000 13-JAN-93 17-JUN-87
King                    24000 17-JUN-87 17-JUN-87
```

14. LEAD

14.1 Syntax




```
LEAD(value_expr [, offset ] [, default ])
   OVER ([ query_partition_clause ] order_by_clause)
```

> **See Also:**
> "Analytic Functions" for information on syntax, semantics, and restrictions, including valid forms of *value_expr*

14.2 Purpose

LEAD is an analytic function. It provides access to more than one row of a table at the same time without a self join. Given a series of rows returned from a query and a position of the cursor, LEAD provides access to a row at a given physical offset beyond that position.

Lead 只能用作分析函数。它提供在不使用自连接的情况下访问表中多个行的途径。给定要查询的行组和一个位置指针，lead 能访问距离给定物理偏移量的行。

If you do not specify *offset*, then its default is 1. The optional *default* value is returned if the offset goes beyond the scope of the table. If you do not specify *default*, then its default value is null.

若不指定 offset，那末其默认为 1。若偏移量超出表范围，则返回可选的 default 值。若没有指定 default，那末其默认为 null。

You cannot use LEAD or any other analytic function for *value_expr*. That is, you cannot nest analytic functions, but you can use other built-in function expressions for *value_expr*.

不能在 value_expr 中使用 lead 或其他任何分析函数。也就是说，此处不能分析函数不能嵌套。但是可以在 value_expr 中使用内置函数表达式。

> **See Also:**
> "About SQL Expressions" for information on valid forms of *expr* and LAG

14.3 Examples

The following example provides, for each employee in the employees table, the hire date of the employee hired just after:

下面的列子提供 employees 表中雇佣日期恰好在当前雇员后的雇员：

```
SELECT last_name,
       hire_date,
       lead(hire_date, 1) over(ORDER BY hire_date) AS "NextHired"
  FROM employees
 WHERE department_id = 30;
```

```
LAST_NAME                 HIRE_DATE NextHired
------------------------- --------- ---------
Raphaely                  07-DEC-94 18-MAY-95
Khoo                      18-MAY-95 24-JUL-97
Tobias                    24-JUL-97 24-DEC-97
Baida                     24-DEC-97 15-NOV-98
Himuro                    15-NOV-98 10-AUG-99
Colmenares                10-AUG-99
```

15. MAX

15.1 Syntax


```
MAX([ DISTINCT | ALL ] expr)
   [ OVER (analytic_clause) ]
```

> **See Also:**
> "Analytic Functions" for information on syntax, semantics, and restrictions

15.2 Purpose

MAX returns maximum value of *expr*. You can use it as an aggregate or analytic function.

Max 返回 expr 的最大值。它可用作聚集或分析函数。

> **See Also:**
> "About SQL Expressions" for information on valid forms of *expr*, "Floating-Point Numbers" for information on binary-float comparison semantics, and "Aggregate Functions"

15.3 Aggregate Example

The following example determines the highest salary in the hr.employees table:

下面的例子确定 hr.employees 表中的最高薪水：

```
SELECT MAX(salary) "Maximum" FROM employees;
```

```
   Maximum
----------
      24000
```

15.4 Analytic Example

The following example calculates, for each employee, the highest salary of the employees reporting to the same manager as the employee.

下面的例子计算具有相同经理的雇员的最高薪水：

```
SELECT manager_id,
       last_name,
       salary,
       MAX(salary) over(PARTITION BY manager_id) AS mgr_max
  FROM employees;
```

```
MANAGER_ID LAST_NAME                 SALARY    MGR_MAX
---------- ------------------------- ---------- ----------
       100 Kochhar                        17000      17000
       100 De Haan                        17000      17000
       100 Raphaely                       11000      17000
       100 Kaufling                        7900      17000
       100 Fripp                           8200      17000
       100 Weiss                           8000      17000
. . .
```

If you enclose this query in the parent query with a predicate, then you can determine the employee who makes the highest salary in each department:

若在父查询中使用谓词，那末可以确定各部门中哪个雇员薪水最高：

```
SELECT manager_id, last_name, salary
  FROM (SELECT manager_id,
               last_name,
               salary,
               MAX(salary) over(PARTITION BY manager_id) AS rmax_sal
          FROM employees)
 WHERE salary = rmax_sal;
```

```
MANAGER_ID LAST_NAME                 SALARY
---------- ------------------------- ----------
```

100	Kochhar	17000
100	De Haan	17000
101	Greenberg	12000
101	Higgens	12000
102	Hunold	9000
103	Ernst	6000
108	Faviet	9000
114	Khoo	3100
120	Nayer	3200
120	Taylor	3200
121	Sarchand	4200
122	Chung	3800
123	Bell	4000
124	Rajs	3500
145	Tucker	10000
146	King	10000
147	Vishney	10500
148	Ozer	11500
149	Abel	11000
201	Goyal	6000
205	Gietz	8300
	King	24000

16. MIN

16.1 Syntax


```
MIN([ DISTINCT | ALL ] expr)
   [ OVER (analytic_clause) ]
```

See Also:
"Analytic Functions" for information on syntax, semantics, and restrictions

16.2 Purpose

`MIN` returns minimum value of *`expr`*. You can use it as an aggregate or analytic function.

`Min` 返回 `expr` 的最小值。它可以用作聚集或分析函数。

> **See Also:**
> "About SQL Expressions" for information on valid forms of *`expr`*, "Floating-Point Numbers" for information on binary-float comparison semantics, and "Aggregate Functions"

16.3 Aggregate Example

The following statement returns the earliest hire date in the `hr.employees` table:

下面的语句返回 `hr.employees` 表中雇员最早入职时间：

```
SELECT MIN(hire_date) "Earliest" FROM employees;

Earliest
---------
17-JUN-87
```

16.4 Analytic Example

The following example determines, for each employee, the employees who were hired on or before the same date as the employee. It then determines the subset of employees reporting to the same manager as the employee, and returns the lowest salary in that subset.

下面的例子确定，同一经理下的各雇员，雇佣日期不晚于自己的雇员的最低薪水：

```
SELECT manager_id,
       last_name,
       hire_date,
       salary,
       MIN(salary) over(PARTITION BY manager_id ORDER BY hire_date
RANGE unbounded preceding) AS p_cmin
  FROM employees;
```

```
MANAGER_ID LAST_NAME                 HIRE_DATE      SALARY      P_CMIN
---------- ------------------------- --------- ---------- ----------
       100 Kochhar                   21-SEP-89      17000       17000
       100 De Haan                   13-JAN-93      17000       17000
       100 Raphaely                  07-DEC-94      11000       11000
       100 Kaufling                  01-MAY-95       7900        7900
       100 Hartstein                 17-FEB-96      13000        7900
       100 Weiss                     18-JUL-96       8000        7900
       100 Russell                   01-OCT-96      14000        7900
       100 Partners                  05-JAN-97      13500        7900
       100 Errazuriz                 10-MAR-97      12000        7900
. . .
```

17. NTILE

17.1 Syntax


```
NTILE(expr)
   OVER ([ query_partition_clause ] order_by_clause)
```

> **See Also:**
> "Analytic Functions" for information on syntax, semantics, and restrictions, including valid forms of *expr*

17.2 Purpose

NTILE is an analytic function. It divides an ordered data set into a number of buckets indicated by *expr* and assigns the appropriate bucket number to each row. The buckets are numbered 1 through *expr*. The *expr* value must resolve to a positive constant for each partition. Oracle Database expects an integer, and if *expr* is a noninteger constant, then Oracle truncates the value to an integer. The return value is NUMBER.

Ntile 只能用作分析函数。它将一个有序数集分成 expr 数目的桶，并且对每一行赋值适当的桶数。桶的取值范围为 1…expr。对每一行的位置来说 expr 值必须是一个正常数。Oracle 数据库期望它是一个整数，若 expr 不是一个整数常量，那末 Oracle 将 expr 值截断为一个整数。函数返回值是 number 类型。

The number of rows in the buckets can differ by at most 1. The remainder values (the remainder of number of rows divided by buckets) are distributed one for each bucket, starting with bucket 1.

桶中的行数至多相差 1.余值(行数除以桶数得到的余数)从 1 号桶开始，与桶号相同的分布在每个桶中(直到等于余值的桶为止)。

If *expr* is greater than the number of rows, then a number of buckets equal to the number of rows will be filled, and the remaining buckets will be empty.

若 expr 比行数要大，那末行数将用作桶数，并且剩余的桶数将被置为空(结果集中也不会显示)。

You cannot use NTILE or any other analytic function for *expr*. That is, you cannot nest analytic functions, but you can use other built-in function expressions for *expr*.

不能在 expr 中使用 ntile 或其他任何分析函数。也就是说，此处分析函数不能嵌套，但是可以在 expr 中使用内置函数表达式。

> **See Also:**
> "About SQL Expressions" for information on valid forms of *expr* and Table 2-10, "Implicit Type Conversion Matrix" for more information on implicit conversion

17.3 Examples

The following example divides into 4 buckets the values in the salary column of the oe.employees table from Department 100. The salary column has 6 values in this department, so the two extra values (the remainder of 6 / 4) are allocated to buckets 1 and 2, which therefore have one more value than buckets 3 or 4.

下面的例子,在 oe.employees 表中,部门 100 的薪水值被分成 4 个桶。此部门的 Salary 列有 6 个值，因此两个额外的值(6/4 的余数)被分配给桶 1 和桶 2，因此它们比桶 3 和桶 4 多一个值。

```
SELECT last_name, salary, ntile(4) over(ORDER BY salary DESC) AS
quartile
```

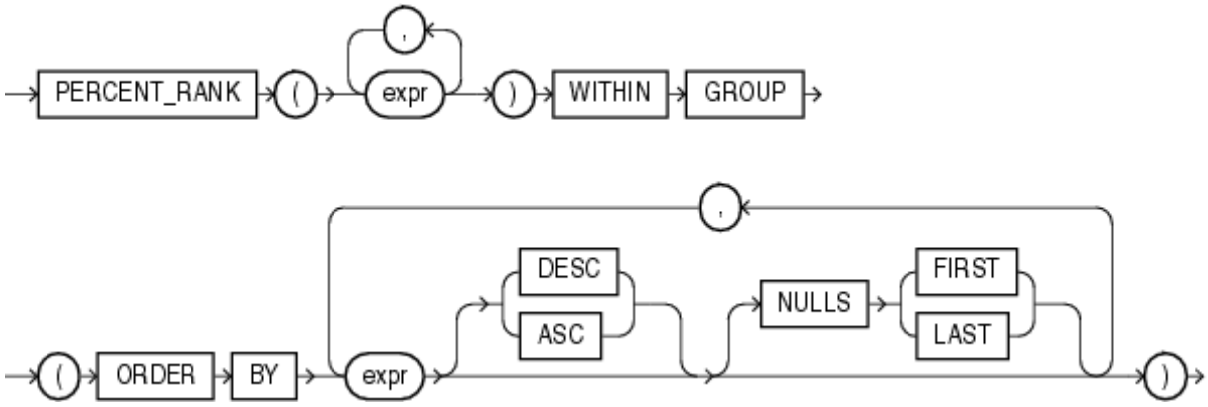
```
  FROM employees
 WHERE department_id = 100;
```

```
LAST_NAME                      SALARY   QUARTILE
------------------------- ---------- ----------
Greenberg                       12000          1
Faviet                           9000          1
Chen                             8200          2
Urman                            7800          2
Sciarra                          7700          3
Popp                             6900          4
```

18. PERCENT_RANK

18.1 Aggregate Syntax

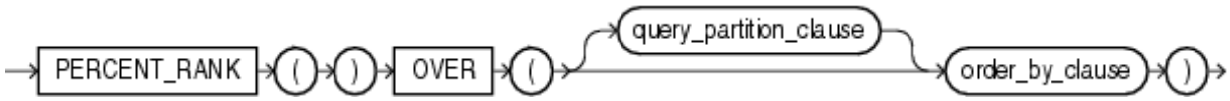
percent_rank_aggregate::=




```
PERCENT_RANK(expr [, expr ]...) WITHIN GROUP
   (ORDER BY
    expr [ DESC | ASC ]
         [NULLS { FIRST | LAST } ]
    [, expr [ DESC | ASC ]
            [NULLS { FIRST | LAST } ]
    ]...
   )
```

18.2 Analytic Syntax

percent_rank_analytic::=




```
PERCENT_RANK( )
   OVER ([ query_partition_clause ] order_by_clause)
```

> **See Also:**
> "Analytic Functions" for information on syntax, semantics, and restrictions

18.3 Purpose

PERCENT_RANK is similar to the CUME_DIST (cumulative distribution) function. The range of values returned by PERCENT_RANK is 0 to 1, inclusive. The first row in any set has a PERCENT_RANK of 0. The return value is NUMBER.

Percent_rank 与 cume_dist 是类似的(累计分布)函数。Percent_rank 返回 0 到 1 的数值。任意集合的第一行 percent_rank 值都为 0。函数返回一个 number 值。

> **See Also:**
> Table 2-10, "Implicit Type Conversion Matrix" for more information on implicit conversion

- As an aggregate function, PERCENT_RANK calculates, for a hypothetical row *r* identified by the arguments of the function and a corresponding sort specification, the rank of row *r* minus 1 divided by the number of rows in the aggregate group. This calculation is made as if the hypothetical row *r* were inserted into the group of rows over which Oracle Database is to aggregate. The arguments of the function identify a single hypothetical row within each aggregate group. Therefore, they must all evaluate to constant expressions within each aggregate group. The constant argument expressions and the expressions in the ORDER BY clause of the aggregate match by position. Therefore the number of arguments must be the same and their types must be compatible.

  当 percent_rank 用作聚集函数时，它计算，对一个由函数参数和相应排序规则确定的假定行 r，用行 r 在聚集分组中的排名减去 1，再除以行数。Oracle 对此

聚集组进行计算时，就像假定行 r 被插入计算的行组中一样。函数参数确定了各个聚集分组中唯一的假定行。常量参数表达式与聚集分组的 order by 子句中的表达式根据位置匹配。因此，参数个数必须相同且类型必须兼容。

- As an analytic function, for a row *r*, PERCENT_RANK calculates the rank of *r* minus 1, divided by 1 less than the number of rows being evaluated (the entire query result set or a partition).

  当 percent_rank 用作分析函数时，对行 r，percent_rank 计算 r 的排名，再用排名减去 1，若得到的结果小于(整个查询结果集或一个分组)行数，再用 1 除以此结果。

18.4 Aggregate Example

The following example calculates the percent rank of a hypothetical employee in the sample table hr.employees with a salary of $15,500 and a commission of 5%:

下面的例子计算 hr.exployees 表中薪水为$15500 并且佣金为 5%的假想雇员百分比排名：

```
SELECT percent_rank(15000, .05) within
 GROUP(
 ORDER BY salary, commission_pct) "Percent-Rank"
  FROM employees;

Percent-Rank
------------
  .971962617
```

18.5 Analytic Example

The following example calculates, for each employee, the percent rank of the employee's salary within the department:

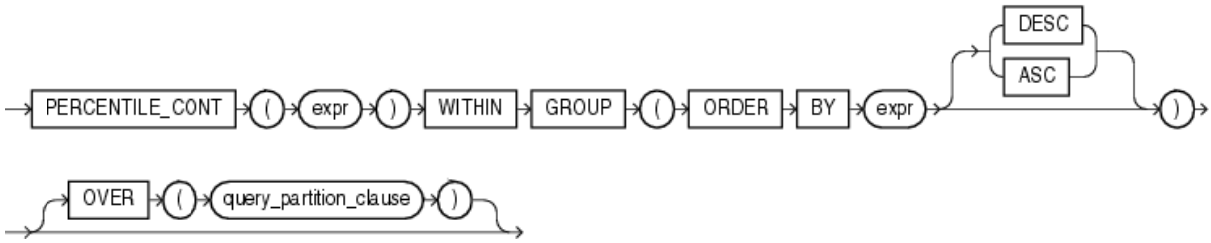
下面的例子计算各部门中雇员薪水的百分比排名：

```
SELECT department_id,
       last_name,
       salary,
       percent_rank() over(PARTITION BY department_id ORDER BY salary
DESC) AS pr
  FROM employees
 ORDER BY pr, salary;
```

```
DEPARTMENT_ID LAST_NAME                    SALARY         PR
------------- ------------------------ ---------- ----------
           10 Whalen                         4400          0
           40 Marvis                         6500          0
. . .
           80 Vishney                       10500 .176470588
           50 Everett                        3900 .181818182
           30 Khoo                           3100         .2
. . .
           80 Johnson                        6200 .941176471
           50 Markle                         2200 .954545455
           50 Philtanker                     2200 .954545455
           50 Olson                          2100          1
. . .
```

19. PERCENTILE_CONT

19.1 Syntax




```
PERCENTILE_CONT(expr) WITHIN GROUP
   (ORDER BY expr [ DESC | ASC ])
   [ OVER (query_partition_clause) ]
```

> **See Also:**
> "Analytic Functions" for information on syntax, semantics, and restrictions of the OVER clause

19.2 Purpose

PERCENTILE_CONT is an inverse distribution function that assumes a continuous distribution model. It takes a percentile value and a sort specification, and returns an interpolated value that would fall into that percentile value with respect to the sort specification. Nulls are ignored in the calculation.

Percentile_cont 是一个采用连续分布模型的反分布函数。它输入一个百分点值并根据相关的排序规则，计算后返回一个内插值，这个值由排序规则相关的百分点值确定。函数计算时忽略空值。

This function takes as an argument any numeric datatype or any nonnumeric datatype that can be implicitly converted to a numeric datatype. The function returns the same datatype as the numeric datatype of the argument.

该函数参数可取任何数字类型或是任何能隐式转换成数字类型的非数字类型。函数返回类型与函数参数类型相同，都为数字类型。

> **See Also:**
> Table 2-10, "Implicit Type Conversion Matrix" for more information on implicit conversion

The first *expr* must evaluate to a numeric value between 0 and 1, because it is a percentile value. This *expr* must be constant within each aggregation group. The ORDER BY clause takes a single expression that must be a numeric or datetime value, as these are the types over which Oracle can perform interpolation.

第一个 expr 的值必须是 0 到 1 的数字，因为它是百分点值。在各聚集分组中 expr 必须为常数。Order by 子句中只能使用一个数字或日期表达式，因为 Oracle 只支持这些数据类型的内插。

The result of PERCENTILE_CONT is computed by linear interpolation between values after ordering them. Using the percentile value (P) and the number of rows (N) in the aggregation group, we compute the row number we are interested in after ordering the rows with respect to the sort specification. This row number (RN) is computed according to the formula RN = (1+ (P*(N-1)). The final result of the aggregate function is computed by linear interpolation between the values from rows at row numbers CRN = CEILING(RN) and FRN = FLOOR(RN).

Percentile_cont 结果是由值和排序后的值之间线性内插计算出来的。使用百分点值(P)与聚集分组的行数(N)，计算按照排序规则排序后的行号。行号(RN)根据 RN = (1+

(P*(N-1))计算而得。聚集函数最终的结果是通过行号的行值 CRN = CEILING(RN)与 FRN = FLOOR(RN)之间线性内插计算而得。

The final result will be:

最终的结果将是：

```
If (CRN = FRN = RN) then the result is
    (value of expression from row at RN)
  Otherwise the result is
    (CRN - RN) * (value of expression for row at FRN) +
    (RN - FRN) * (value of expression for row at CRN)
```

You can use the PERCENTILE_CONT function as an analytic function. You can specify only the *query_partitioning_clause* in its OVER clause. It returns, for each row, the value that would fall into the specified percentile among a set of values within each partition.

Percentile_cont 可用作分析函数。可以在 over 子句中仅指定 query_partitioning_clause。对每一行，它返回属于每个分组中的一组值的指定百分点。

The MEDIAN function is a specific case of PERCENTILE_CONT where the percentile value defaults to 0.5. For more information, please refer to MEDIAN.

Median 函数是 percentile_count 函数的特例，百分点值默认为 0.5。更多的信息请参阅 median。

19.3 Aggregate Example

The following example computes the median salary in each department:

下面的例子计算每个部门的中值薪水：

```
SELECT department_id, percentile_cont(0.5) within
 GROUP(
 ORDER BY salary DESC) "Median cont", percentile_disc(0.5) within
 GROUP(
 ORDER BY salary DESC) "Median disc"
  FROM employees
 GROUP BY department_id;

DEPARTMENT_ID Median-cont Median-disc
------------- ----------- -----------
```

10	4400	4400
20	9500	13000
30	2850	2900
40	6500	6500
50	3100	3100
60	4800	4800
70	10000	10000
80	8800	8800
90	17000	17000
100	8000	8200
110	10150	12000

`PERCENTILE_CONT` and `PERCENTILE_DISC` may return different results. `PERCENTILE_CONT` returns a computed result after doing linear interpolation. `PERCENTILE_DISC` simply returns a value from the set of values that are aggregated over. When the percentile value is 0.5, as in this example, `PERCENTILE_CONT` returns the average of the two middle values for groups with even number of elements, whereas `PERCENTILE_DISC` returns the value of the first one among the two middle values. For aggregate groups with an odd number of elements, both functions return the value of the middle element.

`Percentile_count` 和 `percentile_disc` 可能返回不同的结果。`Percentile_count` 返回线性内插后的计算结果。`Percentile_disc` 仅从聚集的一组值中返回一个值。当百分点值为 `0.5` 时，正如下面的例子，`percentile_cont` 返回分组中偶数位元素两个中值的平均值，然而 `percentile_disc` 返回这两个中值的第一个值。对于奇数位元素的聚集分组，两个函数都返回中值元素的值。

19.4 Analytic Example

In the following example, the median for Department 60 is 4800, which has a corresponding percentile (`Percent_Rank`) of 0.5. None of the salaries in Department 30 have a percentile of 0.5, so the median value must be interpolated between 2900 (percentile 0.4) and 2800 (percentile 0.6), which evaluates to 2850.

在下面的例子，部门 `60` 的中值为 `4800`，与之相应的百分点值为 `0.5`。在部门 `30` 中没有薪水百分点为 `0.5` 的人，因此必须插入一个 `2900` 到 `2800` 的中值，这个值为 `2850`。

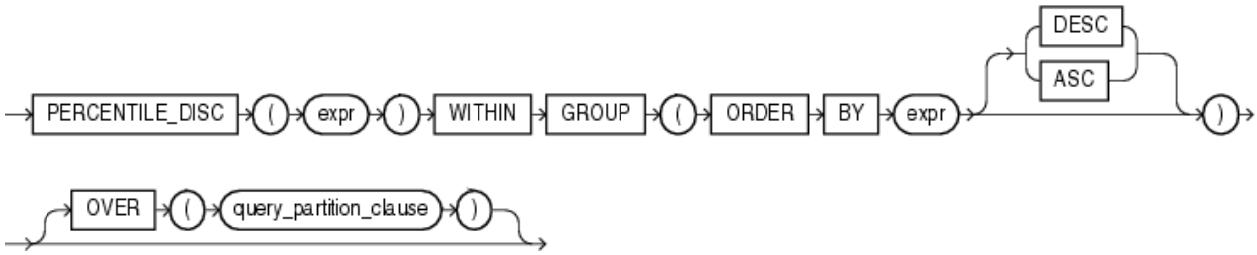
```
SELECT last_name, salary, department_id, percentile_cont(0.5) within
 GROUP(
 ORDER BY salary DESC) over(PARTITION BY department_id)
"Percentile_Cont", percent_rank() over(PARTITION BY department_id
 ORDER BY salary DESC) "Percent_Rank"
  FROM employees
```

```
 WHERE department_id IN (30, 60);
```

LAST_NAME	SALARY	DEPARTMENT_ID	Percentile_Cont	Percent_Rank
Raphaely	11000	30	2850	0
Khoo	3100	30	2850	.2
Baida	2900	30	2850	.4
Tobias	2800	30	2850	.6
Himuro	2600	30	2850	.8
Colmenares	2500	30	2850	1
Hunold	9000	60	4800	0
Ernst	6000	60	4800	.25
Austin	4800	60	4800	.5
Pataballa	4800	60	4800	.5
Lorentz	4200	60	4800	1

20. PERCENTILE_DISC

20.1 Syntax




```
PERCENTILE_DISC(expr) WITHIN GROUP
   (ORDER BY expr [ DESC | ASC ])
   [ OVER (query_partition_clause) ]
```

> **See Also:**
> "Analytic Functions" for information on syntax, semantics, and restrictions of the OVER clause

20.2 Purpose

PERCENTILE_DISC is an inverse distribution function that assumes a discrete distribution model. It takes a percentile value and a sort specification and returns an element from the set. Nulls are ignored in the calculation.

Percentile_disc 是一个采用连续分布模型的反分布函数。它输入一个百分点值并根据相关的排序规则，计算后返回一个内插值，这个值由排序规则相关的百分点值确定。函数计算时忽略空值。

This function takes as an argument any numeric datatype or any nonnumeric datatype that can be implicitly converted to a numeric datatype. The function returns the same datatype as the numeric datatype of the argument.

该函数参数可取任何数字类型或是任何能隐式转换成数字类型的非数字类型。函数返回类型与函数参数类型相同，都为数字类型。

> **See Also:**
> Table 2-10, "Implicit Type Conversion Matrix" for more information on implicit conversion

The first *expr* must evaluate to a numeric value between 0 and 1, because it is a percentile value. This expression must be constant within each aggregate group. The ORDER BY clause takes a single expression that can be of any type that can be sorted.

第一个 expr 的值必须是 0 到 1 的数字，因为它是百分点值。在各聚集分组中 expr 必须为常数。Order by 子句中只能使用一个表达式，该表达式值的类型可以是任何可排序类型。

For a given percentile value P, PERCENTILE_DISC sorts the values of the expression in the ORDER BY clause and returns the value with the smallest CUME_DIST value (with respect to the same sort specification) that is greater than or equal to P.

对于一个假定的百分点值 P，percentile_dist 根据 order by 子句中的表达式值排序，并返回一个大于等于 P 的最小的 cume_dist 值(与相同的排序规则相关)。

20.3 Aggregate Example

See aggregate example for PERCENTILE_CONT.

参见 percentile_count 聚集函数的例子。

20.4 Analytic Example

The following example calculates the median discrete percentile of the salary of each employee in the sample table `hr.employees`:

下面的例子计算 `hr.employees` 表中每个雇员薪水的中值离散百分点：

```
SELECT last_name, salary, department_id, percentile_disc(0.5) within
 GROUP(
 ORDER BY salary DESC) over(PARTITION BY department_id)
"Percentile_Disc", cume_dist() over(PARTITION BY department_id
 ORDER BY salary DESC) "Cume_Dist"
  FROM employees
 WHERE department_id IN (30, 60);
```

```
LAST_NAME         SALARY DEPARTMENT_ID Percentile_Disc  Cume_Dist
------------- ---------- ------------- --------------- ----------
Raphaely           11000            30            2900 .166666667
Khoo                3100            30            2900 .333333333
Baida               2900            30            2900         .5
Tobias              2800            30            2900 .666666667
Himuro              2600            30            2900 .833333333
Colmenares          2500            30            2900          1
Hunold              9000            60            4800         .2
Ernst               6000            60            4800         .4
Austin              4800            60            4800         .8
Pataballa           4800            60            4800         .8
Lorentz             4200            60            4800          1
```

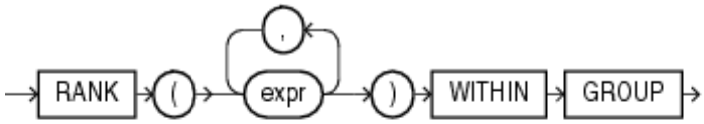
The median value for Department 30 is 2900, which is the value whose corresponding percentile (`Cume_Dist`) is the smallest value greater than or equal to 0.5. The median value for Department 60 is 4800, which is the value whose corresponding percentile is the smallest value greater than or equal to 0.5.

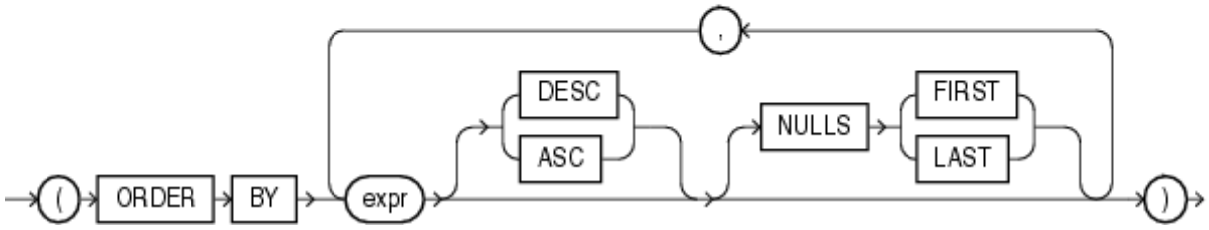
部门`30`的中值为`2900`，相应的百分点`(Cume_dist)`是大于等于`0.5`的最小值。部门`60`的中值为`4800`，相应的百分点是大于等于`0.5`的最小值。

21. RANK

21.1 Aggregate Syntax

rank_aggregate::=

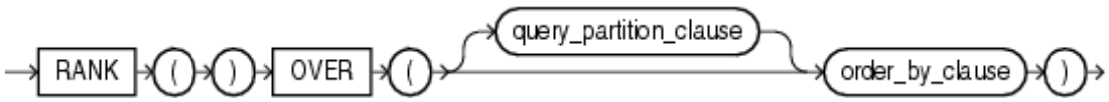





```
RANK(expr [, expr ]...) WITHIN GROUP
   (ORDER BY
    expr [ DESC | ASC ]
         [ NULLS { FIRST | LAST } ]
    [, expr [ DESC | ASC ]
            [ NULLS { FIRST | LAST } ]
    ]...
   )
```

21.2 Analytic Syntax

rank_analytic::=




```
RANK( )
   OVER ([ query_partition_clause ] order_by_clause)
```

See Also:

"Analytic Functions" for information on syntax, semantics, and restrictions

21.3 Purpose

RANK calculates the rank of a value in a group of values. The return type is NUMBER.

Rank 计算一组值的排名。它的返回类型为 number。

See Also:
Table 2-10, "Implicit Type Conversion Matrix" for more information on implicit conversion and "Numeric Precedence" for information on numeric precedence

Rows with equal values for the ranking criteria receive the same rank. Oracle Database then adds the number of tied rows to the tied rank to calculate the next rank. Therefore, the ranks may not be consecutive numbers. This function is useful for top-N and bottom-N reporting.

相等值的行排名相同。Oracle 数据库计算当前排名占据的行数加上当前排名得到下一个排名。因此，排名可能是不连续的数字。此函数对于求 top-N 和 bottom-N 报表很有用处。

- As an aggregate function, RANK calculates the rank of a hypothetical row identified by the arguments of the function with respect to a given sort specification. The arguments of the function must all evaluate to constant expressions within each aggregate group, because they identify a single row within each group. The constant argument expressions and the expressions in the ORDER BY clause of the aggregate match by position. Therefore, the number of arguments must be the same and their types must be compatible.

  Rank 用作聚集函数时，它计算一个被带有排序规则的函数参数确定的假定行的排名。函数参数必须对每个聚集组中的常量表达式全部求值。常量参数表达式和聚集中的 order by 子句中的表达式通过位置匹配。因此，参数个数必须相等且类型必须兼容。

- As an analytic function, RANK computes the rank of each row returned from a query with respect to the other rows returned by the query, based on the values of the *value_exprs* in the *order_by_clause*.

  Rank 用作分析函数时，它计算由 order_by_clause 中的 value_expr 值确定返回的查询结果的每一行的排名。

21.4 Aggregate Example

The following example calculates the rank of a hypothetical employee in the sample table `hr.employees` with a salary of $15,500 and a commission of 5%:

下面的例子计算 `hr.employees` 表中薪水为`$15500` 和佣金为 `5%`的雇员的排名：

```
SELECT rank(15500, .05) within
 GROUP(
 ORDER BY salary, commission_pct) "Rank"
  FROM employees;

      Rank
----------
       105
```

Similarly, the following query returns the rank for a $15,500 salary among the employee salaries:

相似地，下面的查询返回薪水为`$15500` 的雇员的排名：

```
SELECT rank(15500) within
 GROUP(
 ORDER BY salary DESC) "Rank of 15500"
  FROM employees;

Rank of 15500
-------------
            4
```

21.5 Analytic Example

The following statement ranks the employees in the sample `hr` schema in department 80 based on their salary and commission. Identical salary values receive the same rank and cause nonconsecutive ranks. Compare this example with the example for DENSE_RANK.

下面的语句对 `hr` 模式 `emplooyes` 表中部门 `80`，以薪水和佣金排名。相同的薪水值有相同的排名，并且导致排名的不连续。请比较 `densce_rank` 例子。

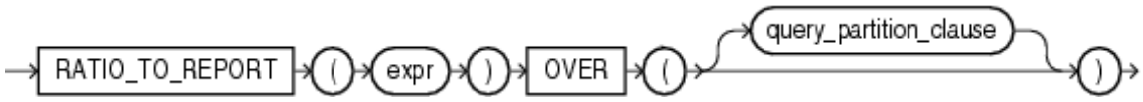
```
SELECT department_id,
      last_name,
      salary,
      commission_pct,
```

```
      rank() over(PARTITION BY department_id ORDER BY salary DESC,
commission_pct) "Rank"
  FROM employees
 WHERE department_id = 80;
```

DEPARTMENT_ID	LAST_NAME	SALARY	COMMISSION_PCT	Rank
80	Russell	14000	.4	1
80	Partners	13500	.3	2
80	Errazuriz	12000	.3	3
80	Ozer	11500	.25	4
80	Cambrault	11000	.3	5
80	Abel	11000	.3	5
80	Zlotkey	10500	.2	7
80	Vishney	10500	.25	8
80	Bloom	10000	.2	9
80	Tucker	10000	.3	10
80	King	10000	.35	11
80	Fox	9600	.2	12
80	Greene	9500	.15	13
80	Bernstein	9500	.25	14
80	Sully	9500	.35	15
80	Hall	9000	.25	16
80	McEwen	9000	.35	17
80	Hutton	8800	.25	18
80	Taylor	8600	.2	19
80	Livingston	8400	.2	20
80	Olsen	8000	.2	21
80	Smith	8000	.3	22
80	Cambrault	7500	.2	23
80	Doran	7500	.3	24
80	Smith	7400	.15	25
80	Bates	7300	.15	26
80	Marvins	7200	.1	27
80	Tuvault	7000	.15	28
80	Sewall	7000	.25	29
80	Lee	6800	.1	30
80	Ande	6400	.1	31
80	Banda	6200	.1	32
80	Johnson	6200	.1	32
80	Kumar	6100	.1	34

22. RATIO_TO_REPORT

22.1 Syntax




```
RATIO_TO_REPORT(expr)
   OVER ([ query_partition_clause ])
```

> **See Also:**
> "Analytic Functions" for information on syntax, semantics, and restrictions, including valid forms of *expr*

22.2 Purpose

RATIO_TO_REPORT is an analytic function. It computes the ratio of a value to the sum of a set of values. If *expr* evaluates to null, then the ratio-to-report value also evaluates to null.

Ratio_to_report 只能用作分析函数。它计算一个值在一组值总和中占的比率。若 expr 求值为 null，那末 ratio_to_report 值也为 null。

The set of values is determined by the *query_partition_clause*. If you omit that clause, then the ratio-to-report is computed over all rows returned by the query.

组值由 query_partition_clause 确定。若忽略该子句，那末 ratio_to_report 计算查询返回的所有行。

You cannot use RATIO_TO_REPORT or any other analytic function for *expr*. That is, you cannot nest analytic functions, but you can use other built-in function expressions for *expr*. Please refer to "About SQL Expressions" for information on valid forms of *expr*.

Expr 中不能使用 ratio_to_report 或其他任何分析函数。也就是说，此处分析函数不能嵌套，但是在 expr 中可以使用内置函数表达式。请参阅 About SQL Expressions 获取合法 expr 的更多信息。

22.3 Examples

The following example calculates the ratio-to-report value of each purchasing clerk's salary to the total of all purchasing clerks' salaries:

下面的例子计算每个采购员薪水在整个采购员薪水总和中的 `ratio-to-report` 值：

```
SELECT last_name, salary, ratio_to_report(salary) over() AS rr
  FROM employees
 WHERE job_id = 'PU_CLERK';
```

```
LAST_NAME                      SALARY         RR
------------------------- ---------- ----------
Khoo                             3100 .223021583
Baida                            2900 .208633094
Tobias                           2800 .201438849
Himuro                           2600  .18705036
Colmenares                       2500 .179856115
```

23. REGR_ (LINEAR REGRESSION) FUNCTIONS

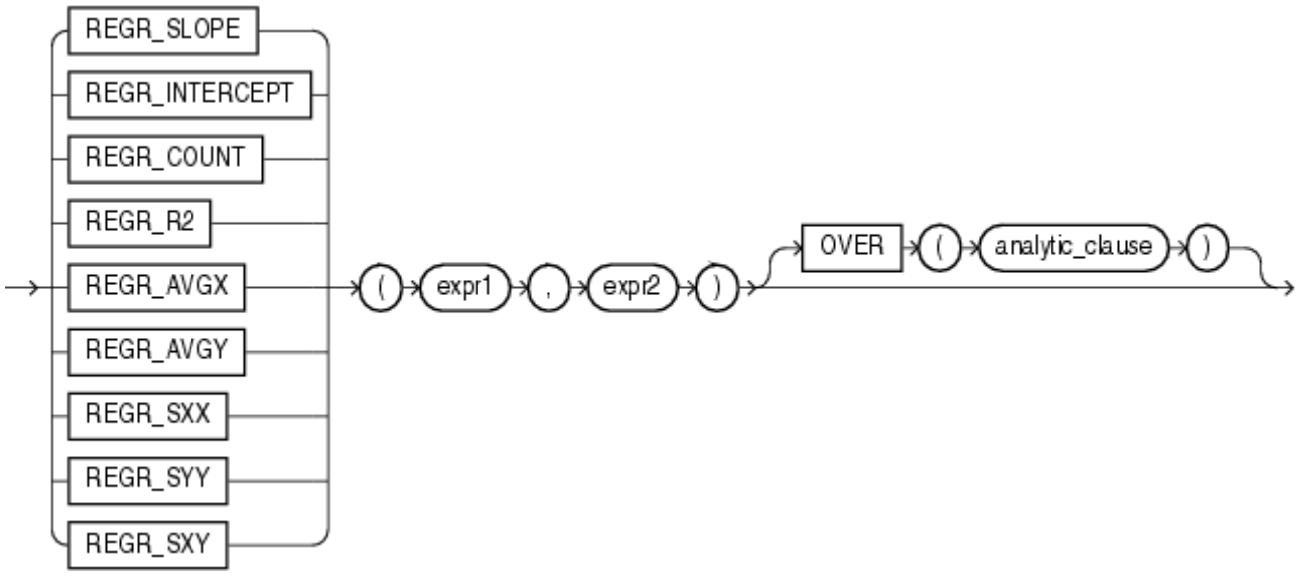
The linear regression functions are:

线性回归函数包括：

- REGR_SLOPE
- REGR_INTERCEPT
- REGR_COUNT
- REGR_R2
- REGR_AVGX
- REGR_AVGY
- REGR_SXX
- REGR_SYY
- REGR_SXY

23.1 Syntax

linear_regr::=




```
{ REGR_SLOPE
| REGR_INTERCEPT
| REGR_COUNT
| REGR_R2
| REGR_AVGX
| REGR_AVGY
| REGR_SXX
| REGR_SYY
| REGR_SXY
}
(expr1 , expr2)
[ OVER (analytic_clause) ]
```

> **See Also:**
> "Analytic Functions" for information on syntax, semantics, and restrictions

23.2 Purpose

The linear regression functions fit an ordinary-least-squares regression line to a set of number pairs. You can use them as both aggregate and analytic functions.

线性回归函数适用于一组数值对的最小二乘法回归线。它可用作聚集和分析函数。

> **See Also:**
> "Aggregate Functions" and "About SQL Expressions" for information on valid forms of *expr*

These functions take as arguments any numeric datatype or any nonnumeric datatype that can be implicitly converted to a numeric datatype. Oracle determines the argument with the highest numeric precedence, implicitly converts the remaining arguments to that datatype, and returns that datatype.

函数参数可取任何数字类型或任何可以隐式转换为数字类型的非数字类型。Oracle根据最高数字优先级确定参数，隐式地将需要处理的参数转换为数字类型，并返回数字类型。

> **See Also:**
> Table 2-10, "Implicit Type Conversion Matrix" for more information on implicit conversion and "Numeric Precedence" for information on numeric precedence

Oracle applies the function to the set of (*expr1*, *expr2*) pairs after eliminating all pairs for which either *expr1* or *expr2* is null. Oracle computes all the regression functions simultaneously during a single pass through the data.

Oracle 使用该函数前先排除(expr1,expr2)中 expr1 或 expr2 不为 null 数值对。Oracle 在数据单个传递期间计算所有的回归函数。

expr1 is interpreted as a value of the dependent variable (a y value), and *expr2* is interpreted as a value of the independent variable (an x value).

Expr1 是因变量(作为 y 值)，expr2 是自变量(作为 x 值)。

- REGR_SLOPE returns the slope of the line. The return value is a numeric datatype and can be null. After the elimination of null (*expr1*, *expr2*) pairs, it makes the following computation:

  REGR_SLOPE 返回行的斜率。返回值是一个数字类型并能为空。在排除(expr1，expr2) 中的 null 对后，它作如下的计算：

  COVAR_POP(expr1, expr2) / VAR_POP(expr2)

- REGR_INTERCEPT returns the y-intercept of the regression line. The return value is a numeric datatype and can be null. After the elimination of null (*expr1*, *expr2*) pairs, it makes the following computation:

  REGR_INTERCEPT 返回回归线的 y 轴截距。 返回值是一个数字类型并能为空。 在排除 (expr1，expr2) 中的 null 对后，它作如下的计算：

  AVG(expr1) - REGR_SLOPE(expr1, expr2) * AVG(expr2)

- REGR_COUNT returns an integer that is the number of non-null number pairs used to fit the regression line.

  REGR_COUNT 返回用于匹配回归线的非空的数字对的个数。

- REGR_R2 returns the coefficient of determination (also called R-squared or goodness of fit) for the regression. The return value is a numeric datatype and can be null. VAR_POP(*expr1*) and VAR_POP(*expr2*) are evaluated after the elimination of null pairs. The return values are:

  REGR_R2 返回回归确定系数（也称为 R_squared 或完全匹配）。返回值是一个数字类型并能为空。 VAR_POP(expr1) 和 VAR_POP(expr2)在 null 对排除后进行求值。 返回值是：

  NULL if VAR_POP(expr2) = 0

  若 VAR_POP(expr2) = 0，值为 null

  1 if VAR_POP(expr1) = 0 and VAR_POP(expr2) != 0

  若 VAR_POP(expr1) = 0 且 VAR_POP(expr2) != 0，值为 1

  POWER(CORR(expr1,expr),2) if VAR_POP(expr1) > 0 and VAR_POP(expr2) != 0

  若 VAR_POP(expr1) > 0 且 VAR_POP(expr2) != 0，值为 POWER(CORR(expr1,expr),2)

All of the remaining regression functions return a numeric datatype and can be null:

所有余下的回归函数返回数字类型并能为空：

- REGR_AVGX evaluates the average of the independent variable (*expr2*) of the regression line. It makes the following computation after the elimination of null (*expr1*, *expr2*) pairs:

REGR_AVGX 计算回归直线自变量 (expr2) 的平均值。它排除 (expr1，expr2) 中的 null 对后作如下计算：

```
AVG(expr2)
```

- REGR_AVGY evaluates the average of the dependent variable (*expr1*) of the regression line. It makes the following computation after the elimination of null (*expr1*, *expr2*) pairs:

  REGR_AVGY 计算回归直线因变量 (expr1) 的平均值。它排除(expr1，expr2) 中的 null 对后作如下计算：

  ```
  AVG(expr1)
  ```

REGR_SXY, REGR_SXX, REGR_SYY are auxiliary functions that are used to compute various diagnostic statistics.

REGR_SXY，REGR_SXX，REGR_SYY 是用来计算各种诊断统计的辅助函数。

- REGR_SXX makes the following computation after the elimination of null (*expr1*, *expr2*) pairs:

  REGR_SXX 在排除(expr1，expr2)中的 null 对后作如下计算：

  ```
  REGR_COUNT(expr1, expr2) * VAR_POP(expr2)
  ```

- REGR_SYY makes the following computation after the elimination of null (*expr1*, *expr2*) pairs:

  REGR_SXY 在排除(expr1，expr2)中的 null 对后作如下计算：

  ```
  REGR_COUNT(expr1, expr2) * VAR_POP(expr1)
  ```

- REGR_SXY makes the following computation after the elimination of null (*expr1*, *expr2*) pairs:

  REGR_SXY 在排除(expr1，expr2)中的 null 对后作如下计算：

  ```
  REGR_COUNT(expr1, expr2) * COVAR_POP(expr1, expr2)
  ```

The following examples are based on the sample tables sh.sales and sh.products.

下面的例子基于示例表 sh.sales 和 sh.products。

23.3 General Linear Regression Example

The following example provides a comparison of the various linear regression functions used in their analytic form. The analytic form of these functions can be useful when you want to use regression statistics for calculations such as finding the salary predicted for each employee by the model. The sections that follow on the individual linear regression functions contain examples of the aggregate form of these functions.

下面的示例比较了具有各种不同分析形式的线性回归函数。对计算诸如使用模型预测雇员薪水增长之类的回归统计，这些分析函数十分有用。下面单独的线性回归函数部分包含这些函数的聚合形式的示例。

```
SELECT job_id,
       employee_id id,
       salary,
       regr_slope(SYSDATE - hire_date, salary) over(PARTITION BY
job_id) slope,
       regr_intercept(SYSDATE - hire_date, salary) over(PARTITION BY
job_id) intcpt,
       regr_r2(SYSDATE - hire_date, salary) over(PARTITION BY job_id)
rsqr,
       regr_count(SYSDATE - hire_date, salary) over(PARTITION BY
job_id) COUNT,
       regr_avgx(SYSDATE - hire_date, salary) over(PARTITION BY job_id)
avgx,
       regr_avgy(SYSDATE - hire_date, salary) over(PARTITION BY job_id)
avgy
  FROM employees
 WHERE department_id IN (50, 80)
 ORDER BY job_id, employee_id;
```

```
JOB_ID         ID     SALARY SLOPE     INTCPT  RSQR  COUNT       AVGX       AVGY
---------- ----- ---------- ----- ---------- ----- ------ ---------- ----------
SA_MAN        145      14000  .355 -1707.035  .832      5  12200.000  2626.589
SA_MAN        146      13500  .355 -1707.035  .832      5  12200.000  2626.589
SA_MAN        147      12000  .355 -1707.035  .832      5  12200.000  2626.589
SA_MAN        148      11000  .355 -1707.035  .832      5  12200.000  2626.589
SA_MAN        149      10500  .355 -1707.035  .832      5  12200.000  2626.589
SA_REP        150      10000  .257   404.763  .647     29   8396.552  2561.244
SA_REP        151       9500  .257   404.763  .647     29   8396.552  2561.244
SA_REP        152       9000  .257   404.763  .647     29   8396.552  2561.244
SA_REP        153       8000  .257   404.763  .647     29   8396.552  2561.244
SA_REP        154       7500  .257   404.763  .647     29   8396.552  2561.244
SA_REP        155       7000  .257   404.763  .647     29   8396.552  2561.244
```

```
SA_REP        156       10000   .257    404.763   .647      29    8396.552   2561.244
...
```

23.4 REGR_SLOPE and REGR_INTERCEPT Examples

The following example calculates the slope and regression of the linear regression model for time employed (`SYSDATE` - `hire_date`) and salary using the sample table `hr.employees`. Results are grouped by `job_id`.

下面的例子计算表 `hr.employees` 中雇佣时间 `(SYSDATE-hire_date)` 和薪水的斜率和线性回归模型的回归值。 结果按 `job_id` 分组。

```
SELECT job_id,
      regr_slope(SYSDATE - hire_date, salary) slope,
      regr_intercept(SYSDATE - hire_date, salary) intercept
  FROM employees
 WHERE department_id IN (50, 80)
 GROUP BY job_id
 ORDER BY job_id;
```

```
JOB_ID     SLOPE     INTERCEPT
---------- ----- ------------
SA_MAN      .355 -1707.030762
SA_REP      .257   404.767151
SH_CLERK    .745   159.015293
ST_CLERK    .904   134.409050
ST_MAN      .479  -570.077291
```

23.5 REGR_COUNT Examples

The following example calculates the count of by `job_id` for time employed (`SYSDATE` - `hire_date`) and salary using the sample table `hr.employees`. Results are grouped by `job_id`.

下面的例子计算，`hr.employees` 表中，雇佣时间 `(SYSDATE-hire_date)`和薪水按 `job_id` 计数。 结果按 `job_id` 分组。

```
SELECT job_id, regr_count(SYSDATE - hire_date, salary) COUNT
  FROM employees
 WHERE department_id IN (30, 50)
 GROUP BY job_id;
```

```
JOB_ID      COUNT
---------- ------
```

```
ST_MAN           5
PU_MAN           1
SH_CLERK        20
PU_CLERK         5
ST_CLERK        20
```

23.6 REGR_R2 Examples

The following example calculates the coefficient of determination the linear regression of time employed (`SYSDATE` - `hire_date`) and salary using the sample table `hr.employees`:

下面的例子计算，`hr.employees` 表中，雇佣时间 `(SYSDATE-hire_date)`和薪水线性回归的确定系数：

```
SELECT job_id, regr_r2(SYSDATE - hire_date, salary) regr_r2
  FROM employees
 WHERE department_id IN (80, 50)
 GROUP BY job_id;
```

```
JOB_ID         REGR_R2
---------- -----------
ST_MAN      .694185080
SH_CLERK    .879799698
SA_MAN      .832447480
SA_REP      .647007156
ST_CLERK    .742808493
```

23.7 REGR_AVGY and REGR_AVGX Examples

The following example calculates the average values for time employed (`SYSDATE` - `hire_date`) and salary using the sample table `hr.employees`. Results are grouped by `job_id`:

下面的例子计算，`hr.employees` 表中，雇佣时间 `(SYSDATE-hire_date)`和薪水的平均值。结果按 `job_id` 分组：

```
SELECT job_id,
       regr_avgy(SYSDATE - hire_date, salary) avgy,
       regr_avgx(SYSDATE - hire_date, salary) avgx
  FROM employees
 WHERE department_id IN (30, 50)
 GROUP BY job_id;
```

```
JOB_ID                AVGY    AVGX
--------- --------------- ------
ST_MAN      2899.055555556   7280
PU_MAN      3785.455555556  11000
SH_CLERK    2531.955555556   4925
PU_CLERK    2709.255555556   2780
ST_CLERK    2631.605555556   2785
```

23.8 REGR_SXY, REGR_SXX, and REGR_SYY Examples

The following example calculates three types of diagnostic statistics for the linear regression of time employed (`SYSDATE - hire_date`) and salary using the sample table `hr.employees`:

下面的列子计算，`hr.employees` 表中，雇佣时间 `(SYSDATE-hire_date)`和薪水的三种线性回归类型的诊断统计：

```
SELECT job_id,
       regr_sxy(SYSDATE - hire_date, salary) regr_sxy,
       regr_sxx(SYSDATE - hire_date, salary) regr_sxx,
       regr_syy(SYSDATE - hire_date, salary) regr_syy
  FROM employees
 WHERE department_id IN (80, 50)
 GROUP BY job_id
 ORDER BY job_id;
```

```
JOB_ID       REGR_SXY    REGR_SXX   REGR_SYY
--------- ---------- ---------- ----------
SA_MAN        3303500   9300000.0    1409642
SA_REP     16819665.5  65489655.2 6676562.55
SH_CLERK      4248650   5705500.0    3596039
ST_CLERK      3531545   3905500.0 4299084.55
ST_MAN        2180460   4548000.0  1505915.2
```

24. ROW_NUMBER

24.1 Syntax

```
ROW_NUMBER( )
   OVER ([ query_partition_clause ] order_by_clause)
```

> **See Also:**
> "Analytic Functions" for information on syntax, semantics, and restrictions

24.2 Purpose

ROW_NUMBER is an analytic function. It assigns a unique number to each row to which it is applied (either each row in the partition or each row returned by the query), in the ordered sequence of rows specified in the *order_by_clause*, beginning with 1.

Row_number 只可用作分析函数。它作用于按 order_by_claus 排序的已排序行组中，从 1 开始为每一行(或者为一个分组中的各行，或者为查询返回的各行)分配一个唯一的数字。

By nesting a subquery using ROW_NUMBER inside a query that retrieves the ROW_NUMBER values for a specified range, you can find a precise subset of rows from the results of the inner query. This use of the function lets you implement top-N, bottom-N, and inner-N reporting. For consistent results, the query must ensure a deterministic sort order.

在查询内部嵌套包含 row_number 的子查询，并在外部查询中限定 row_number 值的范围，可以得到内部查询的精确结果。 此函数可用于实现 top-N、 bottom-N 和 inner-N 报表输出。为了使结果一致，查询必须确保排序后结果唯一。

You cannot use ROW_NUMBER or any other analytic function for *expr*. That is, you cannot nest analytic functions, but you can use other built-in function expressions for *expr*. Please refer to "About SQL Expressions" for information on valid forms of *expr*.

在 expr 中不能使用 row_number 或其他任何分析函数。也就是此处分析函数不能嵌套。但是可以在 expr 中使用内置函数表达式。请参阅 About SQL Expressions 获取合法 expr 的更多信息。

24.3 Examples

For each department in the sample table oe.employees, the following example assigns numbers to each row in order of employee's hire date:

下面的例子在 oe.employees 表中，各部门按雇员雇佣日期排序后赋予每行一个数：

```
SELECT department_id,
      last_name,
      employee_id,
      row_number() over(PARTITION BY department_id ORDER BY
employee_id) AS emp_id
  FROM employees;

DEPARTMENT_ID LAST_NAME                 EMPLOYEE_ID     EMP_ID
------------- ------------------------- ----------- ----------
           10 Whalen                            200          1
           20 Hartstein                         201          1
           20 Fay                               202          2
           30 Raphaely                          114          1
           30 Khoo                              115          2
           30 Baida                             116          3
           30 Tobias                            117          4
           30 Himuro                            118          5
           30 Colmenares                        119          6
           40 Mavris                            203          1
. . .
          100 Popp                              113          6
          110 Higgins                           205          1
          110 Gietz                             206          2
```

ROW_NUMBER is a nondeterministic function. However, employee_id is a unique key, so the results of this application of the function are deterministic.

Row_number 是非确定性函数。然而，这里 exployee_id 是唯一键值，因此应用函数的结果是确定的。

> **See Also:**
> FIRST_VALUE and LAST_VALUE for examples of nondeterministic behavior

The following inner-N query selects all rows from the employees table but returns only the fifty-first through one-hundredth row:

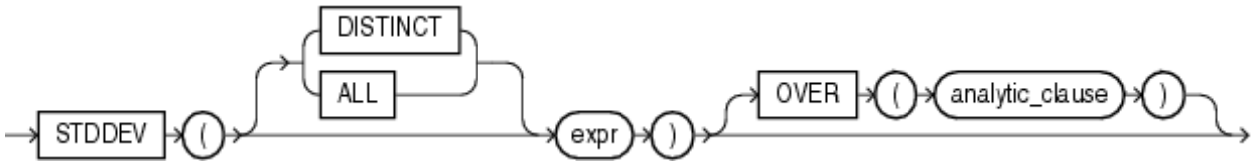
下面的 inner-N 查询，选择 employees 表中所有行，但是结果集只返回 51 到 100 行：

```
SELECT last_name
```

```
  FROM (SELECT last_name, row_number() over(ORDER BY last_name) r
        FROM employees)
 WHERE r BETWEEN 51 AND 100;
```

25. STDDEV

25.1 Syntax




```
STDDEV([ DISTINCT | ALL ] expr)
   [ OVER (analytic_clause) ]
```

> **See Also:**
> "Analytic Functions" for information on syntax, semantics, and restrictions

25.2 Purpose

`STDDEV` returns the sample standard deviation of *`expr`*, a set of numbers. You can use it as both an aggregate and analytic function. It differs from `STDDEV_SAMP` in that `STDDEV` returns zero when it has only 1 row of input data, whereas `STDDEV_SAMP` returns null.

`Stddev` 返回 `expr` 的样本标准偏差。它可用作聚集和分析函数。它与 `stddev_samp` 的不同之处在于，当计算的输入数据只有一行时，`stddev` 返回 `0`，而 `stddev_samp` 返回 `null`。

Oracle Database calculates the standard deviation as the square root of the variance defined for the `VARIANCE` aggregate function.

`Oracle` 数据库中，标准偏差计算结果与 `variance` 用作集聚函数计算结果的平方根相等。

This function takes as an argument any numeric datatype or any nonnumeric datatype that can be implicitly converted to a numeric datatype. The function returns the same datatype as the numeric datatype of the argument.

该函数参数可取任何数字类型或是任何能隐式转换成数字类型的非数字类型。函数返回类型与函数参数类型相同，都为数字类型。

> **See Also:**
> Table 2-10, "Implicit Type Conversion Matrix" for more information on implicit conversion

If you specify `DISTINCT`, then you can specify only the *`query_partition_clause`* of the *`analytic_clause`*. The *`order_by_clause`* and *`windowing_clause`* are not allowed.

`Distinct`关键字仅能在`analytic_clause`的`query_partition_clause`中使用。在`order_by_clause`和`windowing_clause`中不允许使用`distinct`。

> **See Also:**
>
> - "Aggregate Functions", VARIANCE, and STDDEV_SAMP
> - "About SQL Expressions" for information on valid forms of *`expr`*

25.3 Aggregate Examples

The following example returns the standard deviation of the salaries in the sample `hr.employees` table:

下面的例子返回 `hr.employees` 表中薪水的标准偏差：

```
SELECT STDDEV(salary) "Deviation" FROM employees;

 Deviation
----------
3909.36575
```

25.4 Analytic Examples

The query in the following example returns the cumulative standard deviation of the salaries in Department 30 in the sample table `hr.employees`, ordered by `hire_date`:

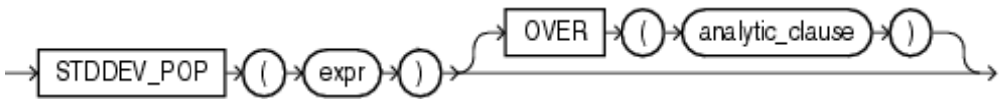
下面的查询示例返回 `hr.employees` 表中部门 `30` 按照 `hire-date` 排序后薪水的累积标准偏差：

```
SELECT last_name, salary, STDDEV(salary) over(ORDER BY hire_date)
"StdDev"
  FROM employees
 WHERE department_id = 30;
```

LAST_NAME	SALARY	StdDev
Raphaely	11000	0
Khoo	3100	5586.14357
Tobias	2800	4650.0896
Baida	2900	4035.26125
Himuro	2600	3649.2465
Colmenares	2500	3362.58829

26. STDDEV_POP

26.1 Syntax




```
STDDEV_POP(expr)
   [ OVER (analytic_clause) ]
```

> **See Also:**
> "Analytic Functions" for information on syntax, semantics, and restrictions

26.2 Purpose

STDDEV_POP computes the population standard deviation and returns the square root of the population variance. You can use it as both an aggregate and analytic function.

Stddev_pop 计算总体标准偏差并返回总体方差的平方根。它可用作聚集和分析函数。

This function takes as an argument any numeric datatype or any nonnumeric datatype that can be implicitly converted to a numeric datatype. The function returns the same datatype as the numeric datatype of the argument.

该函数参数可取任何数字类型或是任何能隐式转换成数字类型的非数字类型。函数返回类型与函数参数类型相同，都为数字类型。

> **See Also:**
> Table 2-10, "Implicit Type Conversion Matrix" for more information on implicit conversion

This function is the same as the square root of the VAR_POP function. When VAR_POP returns null, this function returns null.

该函数与 var_pop 函数的平方根等价。当 var_pop 返回 null 时，该函数也返回 null。

> **See Also:**
>
> - "Aggregate Functions" and VAR_POP
> - "About SQL Expressions" for information on valid forms of *expr*

26.3 Aggregate Example

The following example returns the population and sample standard deviations of the amount of sales in the sample table sh.sales:

下面的例子返回 sh.sales 表中销售总量的总体和样本标准偏差：

```
SELECT stddev_pop(amount_sold) "Pop", stddev_samp(amount_sold)
"Samp"
  FROM sales;

       Pop        Samp
---------- ----------
896.355151 896.355592
```

26.4 Analytic Example

The following example returns the population standard deviations of salaries in the sample `hr.employees` table by department:

下面的例子返回 `hr.employees` 表中部门薪水的总体标准偏差：

```
SELECT department_id,
       last_name,
       salary,
       stddev_pop(salary) over(PARTITION BY department_id) AS pop_std
  FROM employees;
```

```
DEPARTMENT_ID LAST_NAME                    SALARY    POP_STD
------------- ---------------------- --------- ---------
           10 Whalen                       4400         0
           20 Hartstein                   13000      3500
           20 Goyal                        6000      3500
. . .
          100 Sciarra                      7700 1644.18166
          100 Urman                        7800 1644.18166
          100 Popp                         6900 1644.18166
          110 Higgens                     12000      1850
          110 Gietz                        8300      1850
```

27. STDDEV_SAMP

27.1 Syntax


```
STDDEV_SAMP(expr)
   [ OVER (analytic_clause) ]
```

> **See Also:**
> "Analytic Functions" for information on syntax, semantics, and

restrictions

27.2 Purpose

STDDEV_SAMP computes the cumulative sample standard deviation and returns the square root of the sample variance. You can use it as both an aggregate and analytic function.

Stddev_samp 计算累积样本标准偏差并返回样本方差的平方根。 它可以用作聚集和分析函数。

This function takes as an argument any numeric datatype or any nonnumeric datatype that can be implicitly converted to a numeric datatype. The function returns the same datatype as the numeric datatype of the argument.

该函数参数可取任何数字类型或是任何能隐式转换成数字类型的非数字类型。函数返回类型与函数参数类型相同，都为数字类型。

See Also:
Table 2-10, "Implicit Type Conversion Matrix" for more information on implicit conversion

This function is same as the square root of the VAR_SAMP function. When VAR_SAMP returns null, this function returns null.

该函数与 var_samp 函数的平方根等价。当 var_samp 返回 null 时，此函数也返回 null。

See Also:

- "Aggregate Functions" and VAR_SAMP
- "About SQL Expressions" for information on valid forms of *expr*

27.3 Aggregate Example

Please refer to the aggregate example for STDDEV_POP.

请参阅 stddev_pop 聚集函数的例子。

27.4 Analytic Example

The following example returns the sample standard deviation of salaries in the `employees` table by department:

下面的例子返回 `employees` 表中部门薪水的样本标准偏差：

```
SELECT department_id,
       last_name,
       hire_date,
       salary,
       stddev_samp(salary) over(PARTITION BY department_id ORDER BY
hire_date rows BETWEEN unbounded preceding AND CURRENT ROW) AS cum_sdev
  FROM employees;
```

```
DEPARTMENT_ID LAST_NAME       HIRE_DATE     SALARY   CUM_SDEV
------------- --------------- --------- ---------- ----------
           10 Whalen          17-SEP-87       4400
           20 Hartstein       17-FEB-96      13000
           20 Goyal           17-AUG-97       6000 4949.74747
           30 Raphaely        07-DEC-94      11000
           30 Khoo            18-MAY-95       3100 5586.14357
           30 Tobias          24-JUL-97       2800  4650.0896
           30 Baida           24-DEC-97       2900 4035.26125
. . .
          100 Chen            28-SEP-97       8200 2003.33056
          100 Sciarra         30-SEP-97       7700 1925.91969
          100 Urman           07-MAR-98       7800 1785.49713
          100 Popp            07-DEC-99       6900 1801.11077
          110 Higgens         07-JUN-94      12000
          110 Gietz           07-JUN-94       8300 2616.29509
```

28. SUM

28.1 Syntax

```
SUM([ DISTINCT | ALL ] expr)
   [ OVER (analytic_clause) ]
```

> **See Also:**
> "Analytic Functions" for information on syntax, semantics, and restrictions

28.2 Purpose

SUM returns the sum of values of *expr*. You can use it as an aggregate or analytic function.

Sum 返回 expr 的和值。它可用作聚集或分析函数：

This function takes as an argument any numeric datatype or any nonnumeric datatype that can be implicitly converted to a numeric datatype. The function returns the same datatype as the numeric datatype of the argument.

该函数参数可取任何数字类型或是任何能隐式转换成数字类型的非数字类型。函数返回类型与函数参数类型相同，都为数字类型。

> **See Also:**
> Table 2-10, "Implicit Type Conversion Matrix" for more information on implicit conversion

If you specify DISTINCT, then you can specify only the *query_partition_clause* of the *analytic_clause*. The *order_by_clause* and *windowing_clause* are not allowed.

Distinct关键字仅能在analytic_clause的query_partition_clause中使用。在order_by_clause和windowing_clause中不允许使用distinct。

> **See Also:**
> "About SQL Expressions" for information on valid forms of *expr* and "Aggregate Functions"

28.3 Aggregate Example

The following example calculates the sum of all salaries in the sample `hr.employees` table:

下面的例子计算 `hr.employees` 表中雇员薪水之和：

```
SELECT SUM(salary) "Total" FROM employees;
```

```
     Total
----------
    691400
```

28.4 Analytic Example

The following example calculates, for each manager in the sample table `hr.employees`, a cumulative total of salaries of employees who answer to that manager that are equal to or less than the current salary. You can see that Raphaely and Cambrault have the same cumulative total. This is because Raphaely and Cambrault have the identical salaries, so Oracle Database adds together their salary values and applies the same cumulative total to both rows.

下面的例子计算在 `hr.employees` 表中具有同一经理的雇员中薪水小于等于当前雇员的薪水的雇员薪水的累积总值。可以看到 `Rahaely` 和 `Cambrault` 有相同的累积总值，这是因为 `Raphaely` 和 `Cambrault` 有相同的薪水，故 `Oracle` 数据库将它们的薪水值同时累加得到这两行的累积总值。

```
SELECT manager_id,
       last_name,
       salary,
       SUM(salary) over(PARTITION BY manager_id ORDER BY salary RANGE
unbounded preceding) l_csum
  FROM employees;
```

```
MANAGER_ID LAST_NAME                 SALARY     L_CSUM
---------- ------------------------- ---------- ----------
       100 Mourgos                         5800       5800
       100 Vollman                         6500      12300
       100 Kaufling                        7900      20200
       100 Weiss                           8000      28200
       100 Fripp                           8200      36400
       100 Zlotkey                        10500      46900
       100 Raphaely                       11000      68900
       100 Cambrault                      11000      68900
```

```
        100 Errazuriz              12000       80900
. . .
        149 Taylor                  8600       30200
        149 Hutton                  8800       39000
        149 Abel                   11000       50000
        201 Fay                     6000        6000
        205 Gietz                   8300        8300
            King                   24000       24000
```

29. VAR_POP

29.1 Syntax


```
VAR_POP(expr) [ OVER (analytic_clause) ]
```

> **See Also:**
> "Analytic Functions" for information on syntax, semantics, and restrictions

29.2 Purpose

VAR_POP returns the population variance of a set of numbers after discarding the nulls in this set. You can use it as both an aggregate and analytic function.

Var_pop 返回非空数集的总体方差。它可用作聚集和分析函数。

This function takes as an argument any numeric datatype or any nonnumeric datatype that can be implicitly converted to a numeric datatype. The function returns the same datatype as the numeric datatype of the argument.

该函数参数可取任何数字类型或是任意能隐式转换成数字类型的非数字类型。函数返回类型与函数参数类型相同，都为数字类型。

> **See Also:**
> Table 2-10, "Implicit Type Conversion Matrix" for more information on implicit conversion

If the function is applied to an empty set, then it returns null. The function makes the following calculation:

若函数作用于一个空集，那末它返回 null。函数按照下面的公式进行计算：

$$(SUM(expr^2) - SUM(expr)^2 / COUNT(expr)) / COUNT(expr)$$

> **See Also:**
> "About SQL Expressions" for information on valid forms of *expr* and "Aggregate Functions"

29.3 Aggregate Example

The following example returns the population variance of the salaries in the `employees` table:

下面的例子返回 employees 表中薪水的总体方差：

```
SELECT VAR_POP(salary) FROM employees;

VAR_POP(SALARY)
---------------
     15140307.5
```

29.4 Analytic Example

The following example calculates the cumulative population and sample variances in the `sh.sales` table of the monthly sales in 1998:

下面的例子计算 sh.sales 表中 1998 年月度销售量的累计总体和样本方差：

```
SELECT t.calendar_month_desc,
      var_pop(SUM(s.amount_sold)) over(ORDER BY
t.calendar_month_desc) "Var_Pop",
      var_samp(SUM(s.amount_sold)) over(ORDER BY
```

```
t.calendar_month_desc) "Var_Samp"
  FROM sales s, times t
 WHERE s.time_id = t.time_id
   AND t.calendar_year = 1998
 GROUP BY t.calendar_month_desc;
```

```
CALENDAR    Var_Pop   Var_Samp
-------- ---------- ----------
1998-01            0
1998-02  6.1321E+11 1.2264E+12
1998-03  4.7058E+11 7.0587E+11
1998-04  4.6929E+11 6.2572E+11
1998-05  1.5524E+12 1.9405E+12
1998-06  2.3711E+12 2.8453E+12
1998-07  3.7464E+12 4.3708E+12
1998-08  3.7852E+12 4.3260E+12
1998-09  3.5753E+12 4.0222E+12
1998-10  3.4343E+12 3.8159E+12
1998-11  3.4245E+12 3.7669E+12
1998-12  4.8937E+12 5.3386E+12
```

30. VAR_SAMP

30.1 Syntax


```
VAR_SAMP(expr) [ OVER (analytic_clause) ]
```

> **See Also:**
> "Analytic Functions" for information on syntax, semantics, and restrictions

30.2 Purpose

VAR_SAMP returns the sample variance of a set of numbers after discarding the nulls in this set. You can use it as both an aggregate and analytic function.

Var_samp 返回非空数集的样本方差。它可用作聚集和分析函数。

This function takes as an argument any numeric datatype or any nonnumeric datatype that can be implicitly converted to a numeric datatype. The function returns the same datatype as the numeric datatype of the argument.

该函数参数可取任何数字类型或是任何能隐式转换成数字类型的非数字类型。函数返回类型与函数参数类型相同，都为数字类型。

> **See Also:**
> Table 2-10, "Implicit Type Conversion Matrix" for more information on implicit conversion

If the function is applied to an empty set, then it returns null. The function makes the following calculation:

若函数作用于一个空集，那末它返回 null。函数按照下面的公式进行计算：

$$(\text{SUM}(\text{expr}^2) - \text{SUM}(\text{expr})^2 / \text{COUNT}(\text{expr})) / (\text{COUNT}(\text{expr}) - 1)$$

This function is similar to VARIANCE, except that given an input set of one element, VARIANCE returns 0 and VAR_SAMP returns null.

该函数与 variance 类似，不同的是，若输入集合只有一个元素，variance 返回 0，而 var_samp 返回 null。

> **See Also:**
> "About SQL Expressions" for information on valid forms of *expr* and "Aggregate Functions"

30.3 Aggregate Example

The following example returns the sample variance of the salaries in the sample `employees` table.

下面的例子返回 `employees` 表中薪水值的样本方差：

```
SELECT var_samp(salary) FROM employees;

VAR_SAMP(SALARY)
----------------
      15283140.5
```

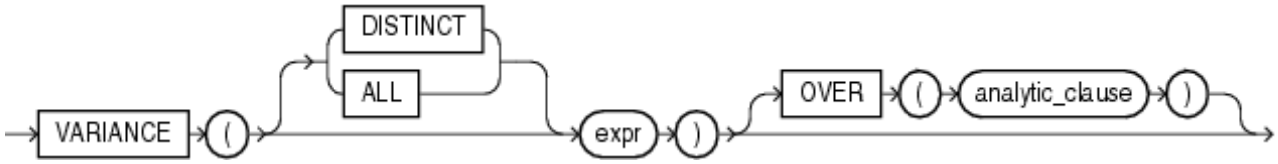
30.4 Analytic Example

Please refer to the analytic example for VAR_POP.

请参阅`var_pop`分析函数的例子。

31. VARIANCE

31.1 Syntax




```
VARIANCE([ DISTINCT | ALL ] expr)
        [ OVER (analytic_clause) ]
```

> **See Also:**
> "Analytic Functions" for information on syntax, semantics, and restrictions

31.2 Purpose

VARIANCE returns the variance of *expr*. You can use it as an aggregate or analytic function.

Variance 返回 expr 的方差。它可用作聚集或分析函数。

Oracle Database calculates the variance of *expr* as follows:

Oracle 数据库按照下面原则计算 expr 的方差：

- 0 if the number of rows in *expr* = 1

  若符合 expr 的行数为 1，则返回 0

- VAR_SAMP if the number of rows in *expr* > 1

  若符合 expr 的行数大于 1，则返回 var_samp。

If you specify DISTINCT, then you can specify only the *query_partition_clause* of the *analytic_clause*. The *order_by_clause* and *windowing_clause* are not allowed.

Distinct关键字仅能在analytic_clause的query_partition_clause中使用。在order_by_clause和windowing_clause中不允许使用distinct。

This function takes as an argument any numeric datatype or any nonnumeric datatype that can be implicitly converted to a numeric datatype. The function returns the same datatype as the numeric datatype of the argument.

该函数参数可取任何数字类型或是任何能隐式转换成数字类型的非数字类型。函数返回类型与函数参数类型相同，都为数字类型。

> **See Also:**
> Table 2-10, "Implicit Type Conversion Matrix" for more information on implicit conversion, "About SQL Expressions" for information on valid forms of *expr* and "Aggregate Functions"

31.3 Aggregate Example

The following example calculates the variance of all salaries in the sample employees table:

下面的例子计算 employees 表中所有薪水的方差：

```
SELECT VARIANCE(salary) "Variance" FROM employees;
```

```
  Variance
----------
15283140.5
```

31.4 Analytic Example

The following example returns the cumulative variance of salary values in Department 30 ordered by hire date.

下面的例子返回部门 30 按雇佣日期排序的薪水值的累计方差：

```
SELECT last_name,
       salary,
       VARIANCE(salary) over(ORDER BY hire_date) "Variance"
  FROM employees
 WHERE department_id = 30;
```

```
LAST_NAME           SALARY   Variance
--------------- ---------- ----------
Raphaely             11000          0
Khoo                  3100   31205000
Tobias                2800 21623333.3
Baida                 2900 16283333.3
Himuro                2600   13317000
Colmenares            2500   11307000
```