Analytic Functions

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

Analytic Functions

- The analytic functions enable you to calculate:
  - Rankings and percentiles
  - Moving window calculations
  - Lag/Lead analysis
  - First/last analysis
  - Linear regression statistics
- Ranking functions include cumulative distributions, percent rank, and N-tiles.
- Moving window calculations allow you to find moving and cumulative aggregations, such as sums and averages.
- Lag/lead analysis enables direct inter-row references so you can calculate period-to-period changes.
- First/last analysis enables you to find the first or last value in an ordered group.

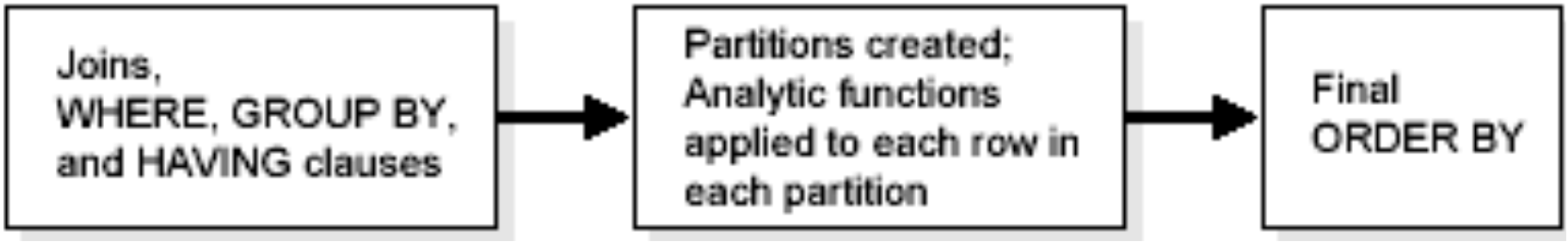
Analytic Functions Categories

Type	Used for
Ranking	Calculating ranks, percentiles, and n-tiles of the values in a result set.
Windowing	Calculating cumulative and moving aggregates. Works with these functions: SUM, AVG, MIN, MAX, COUNT, VARIANCE, STDDEV, FIRST_VALUE, LAST_VALUE.
Reporting	Calculating shares, for example, market share. Works with these functions: SUM, AVG, MIN, MAX, COUNT, VARIANCE, STDDEV, RATIO_TO_REPORT.
LAG/LEAD	Finding a value in a row a specified number of rows from a current row.
FIRST/LAST	First or last value in an ordered group.
Linear Regression	Calculating linear regression and other statistics (slope, intercept, and so on).
Inverse Percentile	The value in a data set that corresponds to a specified percentile.
Hypothetical Rank & Distribution	The rank or percentile that a row would have if inserted into a specified data set.

List of Analytic Functions

AVG	LAST_VALUE	REGR_ (linear regression) functions
CORR	LEAD	STDDEV
COVAR_POP	MAX	STDDEV_POP
COVAR_SAMP	MIN	STDDEV_SAMP
COUNT	NTILE	SUM
CUME_DIST	PERCENT_RANK	VAR_POP
DENSE_RANK	PERCENTILE_CONT	VAR_SAMP
FIRST	PERCENTILE_DISC	VARIANCE *
FIRST_VALUE	RANK	
LAG	RATIO_TO_REPORT	
LAST	ROW_NUMBER	

Essential concepts



- **Processing Order**

  Query processing using analytic functions takes place in three stages.

  First, all joins, WHERE, GROUP BY and HAVING clauses are performed.

  Second, the result set is made available to the analytic functions, and all their calculations take place.

  Third, if the query has an ORDER BY clause at its end, the ORDER BY is processed to allow for precise output ordering. The processing order is shown above.

Essential concepts

- **Result Set Partitions**

  The analytic functions allow users to divide query result sets into groups of rows called ***partitions***.

  Partitions are created after the groups defined with GROUP BY clauses, so they are available to any aggregate results such as sums and averages.

  Partition divisions may be based upon any desired columns or expressions.

  A query result set may be partitioned into just one partition holding all the rows, a few large partitions, or many small partitions holding just a few rows each.

Essential concepts

- **Window**

  For each row in a partition, you can define a sliding window of data.

  This 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.

  The window has a starting row and an ending row. Depending on its definition, the window may move at one or both ends.

  For instance, a window defined for a cumulative sum function would have its starting row fixed at the first row of its partition, and its ending row would slide from the starting point all the way to the last row of the partition.

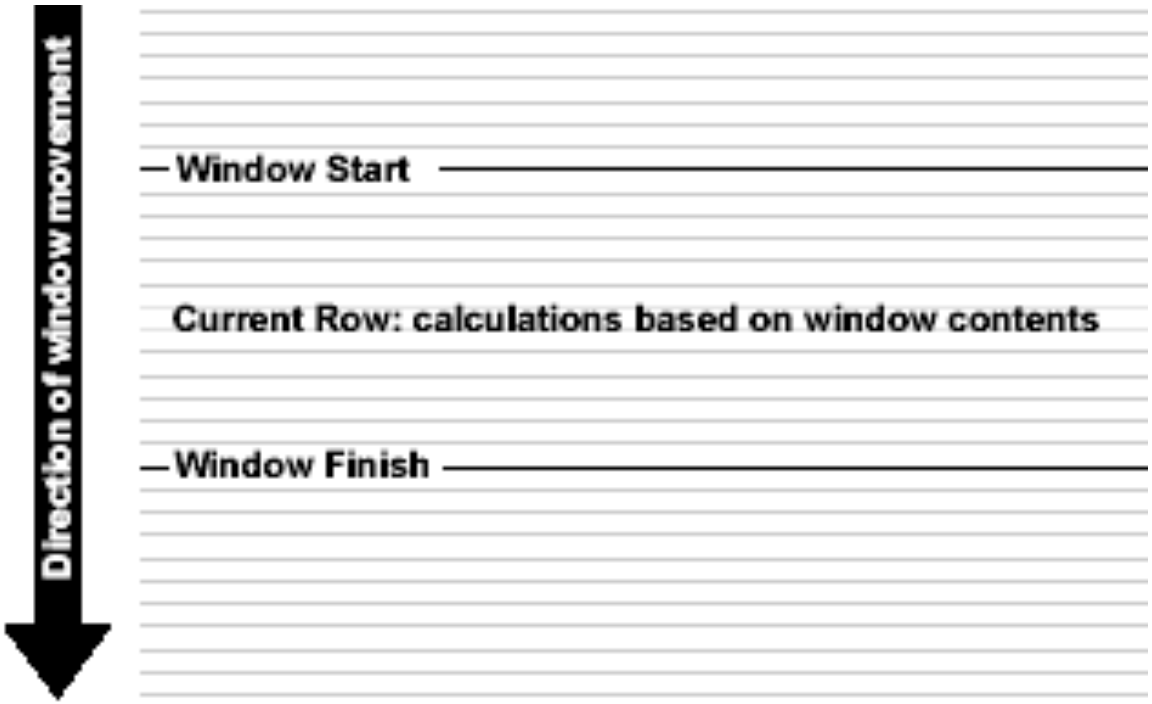
  In contrast, a window defined for a moving average would have both its starting and end points slide so that they maintain a constant physical or logical range.

Essential concepts

- **Current Row**

  Each calculation performed with an analytic function is based on a current row within a partition. The current row serves as the reference point determining the start and end of the window.

  For instance, a centered moving average calculation could be defined with a window that holds the current row, the six preceding rows, and the following six rows. This would create a sliding window of 13 rows, as shown below.



Ranking Functions

- A ranking function computes the rank of a record compared to other records in the dataset based on the values of a set of measures.
- The ranking function are:
  - RANK and DENSE_RANK
  - CUME_DIST and PERCENT_RANK
  - NTILE
  - ROW_NUMBER

RANK and DENSE_RANK

- **RANK**

RANK calculates the rank of a value in a group of values.

Rows with equal values for the ranking criteria receive the same rank. Oracle then adds the number of tied rows to the tied rank to calculate the next rank. Therefore, **the ranks may not be consecutive numbers.**

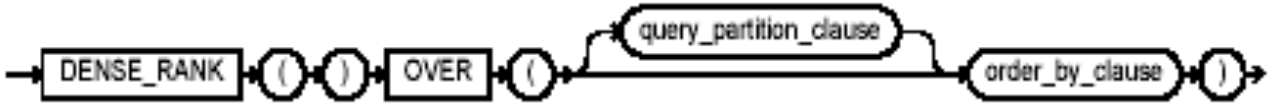
As an analytic function, RANK computes the rank of each row returned from a query with respect to the other rows returned by the query, based on the values of the *value_exprs* in the *order_by_clause.*

The following statement ranks the employees in the sample hr schema within each department based on their salary and commission.

```
SELECT…… RANK() OVER (PARTITION BY department_id
ORDER BY salary DESC,commission_pct) "Rank‟ FROM emp;
DEPT LNAME      SALARY COMMISSION_PCT Rank
10   Whalen      4400                    1
20   Hartstein  13000                     1
20   Goyal       6000                     2
30   Raphaely   11000                     1
30   Khoo        3100                     2
30   Baida       2900                     3
30   Tobias      2800                     4
```

RANK and DENSE_RANK

- **DENSE_RANK**



The DENSE_RANK function computes the rank of a row in an ordered group of rows.

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.

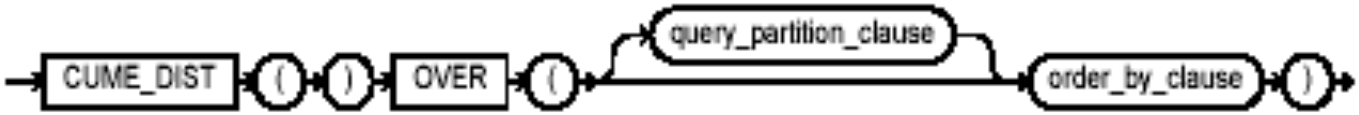
DENSE_RANK computes the rank of each row returned from a query with respect to the other rows, based on the values of the *value_exprs* in the *order_by_clause*.

The following statement selects details 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.

```
SELECT……., DENSE_RANK() OVER (PARTITION BY e.dept_id ORDER BY
  e.salary) as drank FROM employees e, departments d
WHERE e.dept_id = d.dept_id AND d.dept_id IN ('30', '40');
DEPT            LNAME        SALARY     DRANK
Purchasing      Colmenares    2500        1
Purchasing      Himuro        2600        2
Purchasing      Tobias        2800        3
Purchasing      Baida         2900        4
```

CUME_DIST and PERCENT_RANK

- **CUME_DIST**



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.

To compute the CUME_DIST of a value x in a set S of size N, you use the formula:

CUME_DIST(x) = number of values in S before and including x in the specified order/ N

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).

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

```
SELECT ……..,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 Tobias       2800    .6
PU_CLERK Khoo         3100     1
PU_MAN   Raphaely     11000    1
```

CUME_DIST and PERCENT_RANK

- **PERCENT_RANK**

PERCENT_RANK is similar to CUME_DIST, but it uses rank values rather than row counts in its numerator. Therefore, it returns the percent rank of a value relative to a group of values. The function is available in many popular spreadsheets. PERCENT_RANK of a row is :

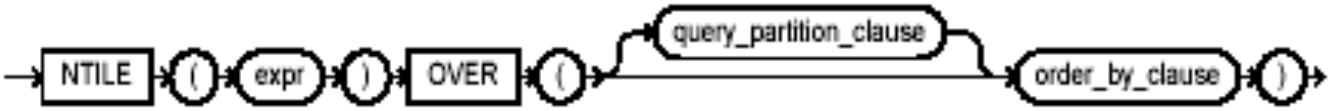
(rank of row in its partition - 1) / (number of rows in the partition - 1)

PERCENT_RANK returns values in the range zero to one. The row(s) with a rank of 1 will have a PERCENT_RANK of zero.

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

```
SELECT ……..,PERCENT_RANK() OVER (PARTITION BY department_id ORDER BY
   salary DESC) AS pr FROM employees ORDER BY pr, salary;
DEPT_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
```

NTILE



NTILE allows easy calculation of tertiles, quartiles, deciles and other common summary statistics. This function divides an ordered partition into a specified number of groups called **buckets** and assigns a bucket number to each row in the partition.

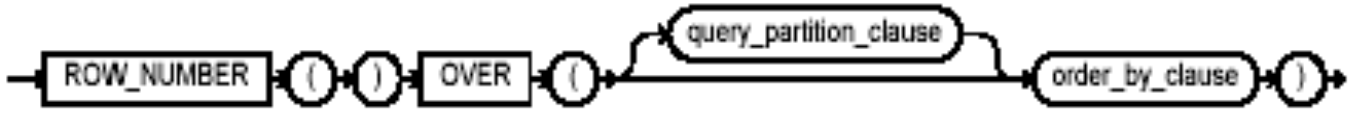
NTILE is a very useful calculation because it lets users divide a data set into fourths, thirds, and other groupings. The buckets are numbered 1 through expr, and expr must resolve to a positive constant for each partition.

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.

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

```
SELECT …,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
```

ROW_NUMBER



The ROW_NUMBER function assigns a unique number (sequentially, starting from 1, as defined by ORDER BY) to each row within the partition.

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

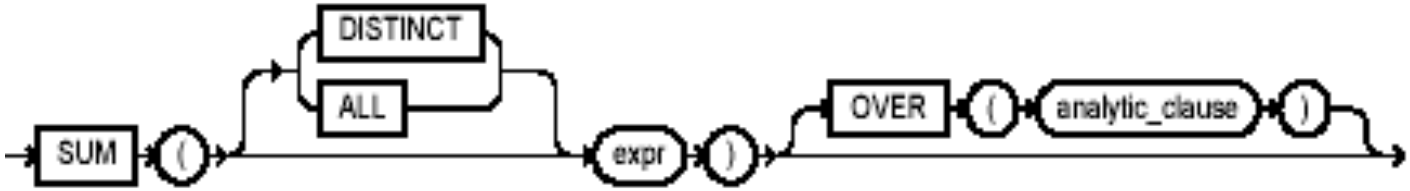
```
SELECT……, ROW_NUMBER() OVER (PARTITION BY department_id ORDER
  BY employee_id) AS emp_id FROM employees;
DEPTID LNAME  EMPLOYEE_ID EMP_ID
10      Whalen    200       1
20       Hartstein 201       1
20      Goyal     202       2
30      Raphaely  114       1
30      Khoo      115       2
30      Baida     116       3
30      Tobias    117       4
30      Himuro    118       5
30      Colmenares119       6
40      Marvis    203       1
```

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

Windowing Aggregate Functions

- Windowing functions can be used to compute cumulative, moving, and centered aggregates.
- They return a value for each row in the table, which depends on other rows in the corresponding window.
- They can be used only in the SELECT and ORDER BY clauses of the query.
- Two other functions are available: FIRST_VALUE, which returns the first value in the window; and LAST_VALUE, which returns the last value in the window.
- These functions provide access to more than one row of a table without a self-join.
- Aggregate Functions include:
  - SUM
  - AVG
  - MAX and MIN
  - COUNT
  - STDDEV
  - VARIANCE
  - FIRST_VALUE and LAST_VALUE

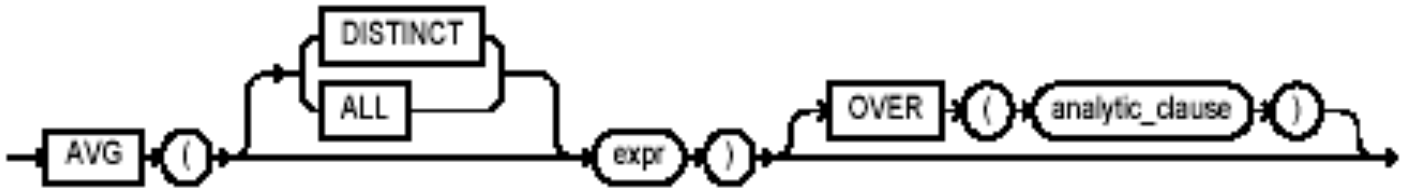
SUM



- SUM returns sum of values of expr. You can use it as an aggregate or analytic function.
- If you specify DISTINCT, you can specify only the query_partition_clause of the analytic_clause. The order_by_clauseand windowing_clauseare not allowed.
- The following example calculates, for each manager in the sample table 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 Zlotkey have the same cumulative total. This is because Raphaely and Cambrault have the identical salaries, so Oracle adds together their salary values and applies the same cumulative total to both rows.

```
SELECT ……,SUM(salary) OVER (PARTITION BY manager_id ORDER
  BY salary RANGE UNBOUNDED PRECEDING) l_csum FROM emp;

MANAGER_ID LNAME      SALARY L_CSUM
100        Mourgos      5800  5800
100        Vollman      6500 12300
100        Kaufling     7900 20200
……..
201        Goyal        6000  6000
201        Gietz        8300 14300
```

AVG

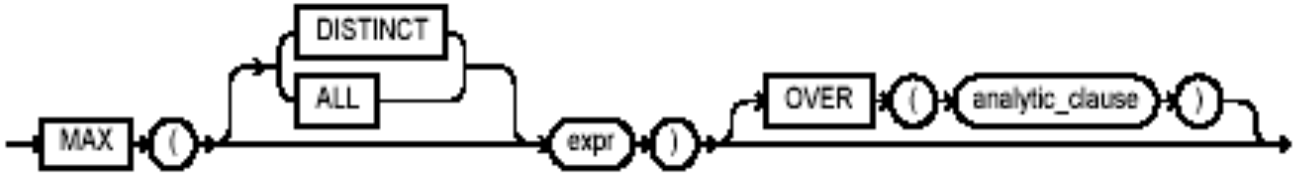


- AVG returns average value of expr.You can use it as an aggregate or analytic function.
- If you specify DISTINCT, you can specify only the query_partition_clause of the analytic_clause. The order_by_clauseand windowing_clauseare not allowed.
- The following example calculates, for each employee in 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:

```
SELECT……,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 LNAME    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
```

MAX and MIN

MAX

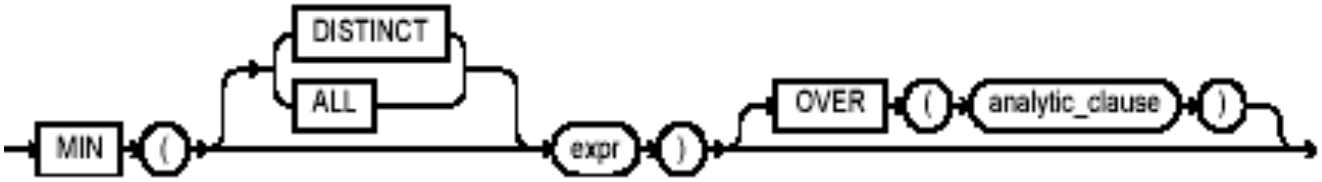


- MAX returns maximum value of expr. You can use it as an aggregate or analytic function.
- If you specify DISTINCT, you can specify only the query_partition_clause of the analytic_clause. The order_by_clauseand windowing_clauseare not allowed.
- The following example calculates, for each employee, the highest salary of the employees reporting to the same manager as the employee.

```
SELECT……, 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
```

MAX and MIN

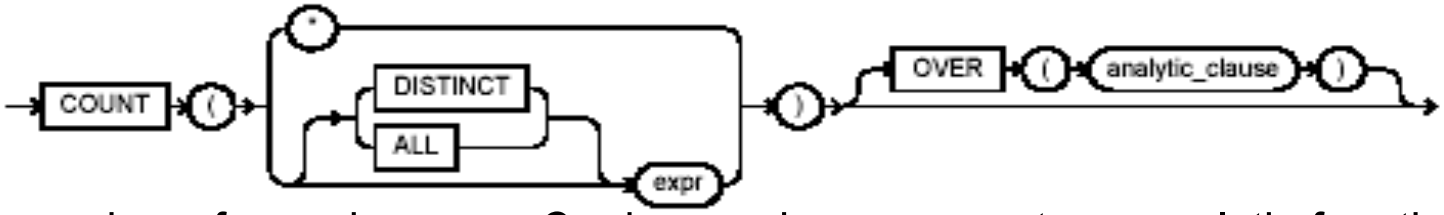
MIN



- MIN returns minimum value of expr. You can use it as an aggregate or analytic function.
- If you specify DISTINCT, you can specify only the query_partition_clause of the analytic_clause. The order_by_clauseand windowing_clauseare not allowed.
- 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.

```
SELECT …,MIN(salary) OVER(PARTITION BY manager_id ORDER BY
  hire_date RANGE UNBOUNDED PRECEDING) as p_cmin FROM emp;
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
```

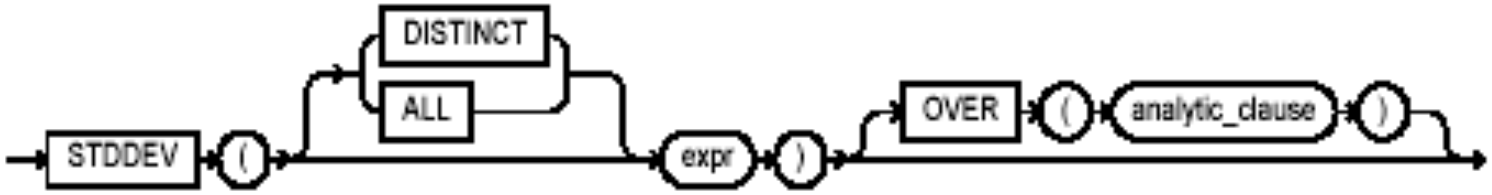
COUNT



- COUNT returns number of rows in query. Canbe used as aggregate or analytic function.
- If you specify DISTINCT, you can specify only the query_partition_clause of the analytic_clause. The order_by_clauseand windowing_clauseare not allowed.
- If you specify expr, COUNT returns the number of rows where exprIs not null. You can count either all rows, or only distinct values of expr.
- If you specify the asterisk (*), this function returns all rows, including duplicates and nulls.
- 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.

```
SELECT…, 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
```

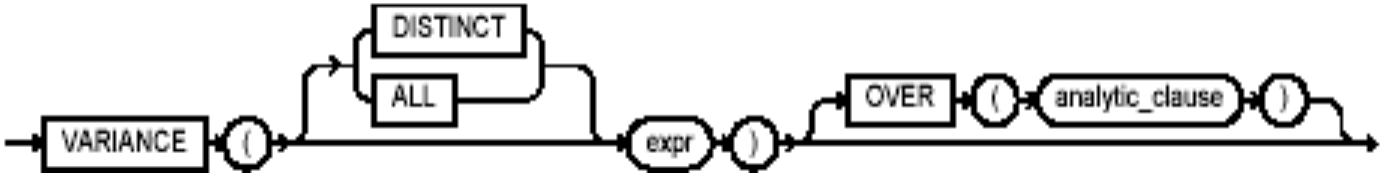
STDDEV



- STDDEV returns sample standard deviation of expr, a set of numbers. You can use it as both an aggregate and analytic function.
- Oracle calculates the standard deviation as the square root of the variance defined for the VARIANCE aggregate function.
- If you specify DISTINCT, you can specify only the query_partition_clause of the analytic_clause. The order_by_clauseand windowing_clauseare not allowed.
- The query in the following example returns the cumulative standard deviation of salary values in Department 80 in the sample table hr.employees, ordered by hire_date:

```
SELECT ……,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
```

VARIANCE



- VARIANCE returns variance of expr. You can use it as an aggregate or analytic function.
- Oracle calculates the variance of expr as follows:
  - 0 if the number of rows in expr = 1
  - VAR_SAMP if the number of rows in expr > 1
- The query returns the cumulative variance of salary in Dept 30 ordered by hiredate.

```
SELECT ……, 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
```

FIRST_VALUE and LAST_VALUE

- The FIRST_VALUE and LAST_VALUE functions allow you to select the first and last rows from a window.

FIRST_VALUE

- FIRST_VALUE is an analytic function. It returns the first value in an ordered set of values.
- The following example selects, for each employee in Department 90, the name of the employee with the lowest salary.

```
SELECT……, 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);
    DEPT_ID LAST_NAME SALARY LOWEST_SAL
    90         Kochhar 17000 Kochhar
    90         De Haan 17000 Kochhar
    90            King 24000 Kochhar
```

FIRST_VALUE and LAST_VALUE

LAST_VALUE

- LAST_VALUE is an analytic function. It returns the last value in an ordered set of values.
- The following example returns, for each row, the hiredate of the employee earning the highest salary.

```
SELECT ……, LAST_VALUE(hire_date) OVER (ORDER BY
  salary ROWS BETWEEN UNBOUNDED PRECEDING AND
  UNBOUNDED FOLLOWING) AS lv FROM (SELECT * FROM
  employees WHERE dept_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
```

Reporting Aggregate Functions

- After a query has been processed, aggregate values like the number of resulting rows or an average value in a column can be easily computed within a partition and made available to other reporting functions. Reporting aggregate functions return the same aggregate value for every row in a partition
- The syntax is:

```
{SUM | AVG | MAX | MIN | COUNT | STDDEV | VARIANCE}
([ALL | DISTINCT] {<value expression1> | *})
OVER ([PARTITION BY <value expression2>[,...]])
```

where

- An asterisk (*) is only allowed in COUNT(*)
- DISTINCT is supported only if corresponding aggregate functions allow it
- <value expression1> and <value expression2> can be any valid expression involving column references or aggregates.
- The PARTITION BY clause defines the groups on which the windowing functions would be computed. If the PARTITION BY clause is absent, then the function is computed over the whole query result set.
- Reporting functions can appear only in the SELECT clause or the ORDER BY clause.
- The major benefit of reporting functions is their ability to do multiple passes of data in a single query block and speed up query performance. .

RATIO_TO_REPORT

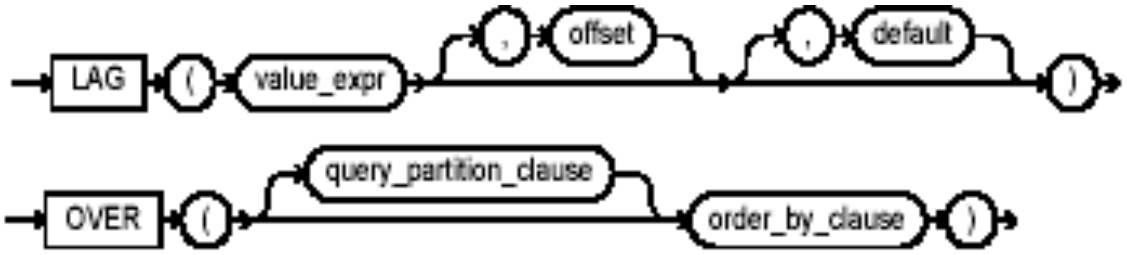
- RATIO_TO_REPORT is an analytic function.
- It computes the ratio of a value to the sum of a set of values.
- If expr evaluates to null, the ratio-to-report value also evaluates to null.
- The set of values is determined by the query_partition_clause. If you omit that clause, the ratio-to-report is computed over all rows returned by the query.
- The following example calculates the ratio-to-report of each purchasing clerk's salary to the total of all purchasing clerks' salaries:

```
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
```

LAG/LEAD Functions

- The LAG and LEAD functions are useful for comparing values when the relative positions of rows can be known reliably.
- They work by specifying the count of rows which separate the target row from the current row.
- Since the functions provide access to more than one row of a table at the same time without a self-join, they can enhance processing speed.
- The LAG function provides access to a row at a given offset prior to the current position, and the LEAD function provides access to a row at a given offset after the current position.

LAG/LEAD Functions

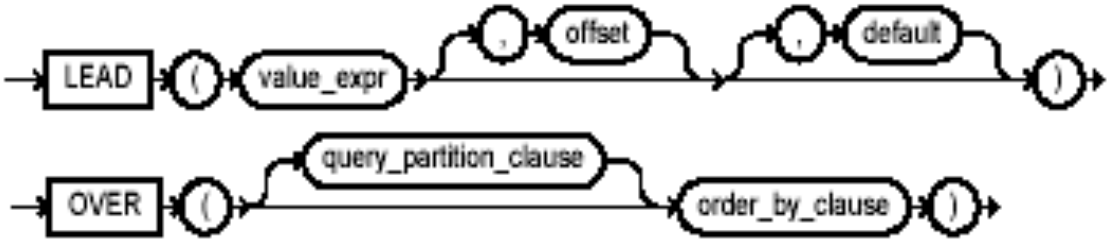


LAG

- LAG is an analytic function. 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.
- If you do not specify offset, 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, its default value is null.
- The following example provides, for each salesperson in the employees table, the salary of the employee hired just before:

```
SELECT ……, 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
```

LAG/LEAD Functions



LEAD

- LEAD is an analytic function.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.
- If you do not specify offset, 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, its default value is null.
- The following example provides, for each employee in the employees table, the hiredate of the employee hired just after:

```
SELECT…,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
```

Linear Regression

- The regression functions support the fitting of an ordinary-least-squares regression line to a set of number pairs. You can use them as both aggregate functions or windowing or reporting functions.
- The functions are:

  REGR_COUNT

  REGR_AVGX and REGR_AVGY

  REGR_SLOPE and REGR_INTERCEPT

  REGR_R2

  REGR_SXX and REGR_SYY and REGR_SXY
- Oracle applies the function to the set of (e1, e2) pairs after eliminating all pairs for which either of e1 or e2 is null. e1 is interpreted as a value of the dependent variable (a "y value"), and e2 is interpreted as a value of the independent variable (an "x value"). Both expressions must be numbers.

Linear Regression

- **REGR_COUNT**

  REGR_COUNT returns the number of non-null number pairs used to fit the regression line. If applied to an empty set (or if there are no (e1, e2) pairs where neither of e1 or e2 is null), the function returns 0.

- **REGR_AVGY and REGR_AVGX**

  REGR_AVGY and REGR_AVGX compute the averages of the dependent variable and the independent variable of the regression line, respectively. REGR_AVGY computes the average of its first argument (e1) after eliminating (e1, e2) pairs where either of e1 or e2 is null. Similarly, REGR_AVGX computes the average of its second argument (e2) after null elimination. Both functions return NULL if applied to an empty set.

- **REGR_SLOPE and REGR_INTERCEPT**

  The REGR_SLOPE function computes the slope of the regression line fitted to non-null (e1, e2) pairs. The REGR_INTERCEPT function computes the y-intercept of the regression line. REGR_INTERCEPT returns NULL whenever slope or the regression averages are NULL.

Linear Regression

- **REGR_R2**

  The REGR_R2 function computes the coefficient of determination (usually called "R-squared" or "goodness of fit") for the regression line. REGR_R2 returns values between 0 and 1 when the regression line is defined (slope of the line is not null), and it returns NULL otherwise. The closer the value is to 1, the better the regression line fits the data.

- **REGR_SXX, REGR_SYY, and REGR_SXY**

  REGR_SXX, REGR_SYY and REGR_SXY functions are used in computing various diagnostic statistics for regression analysis. After eliminating (e1, e2) pairs where either of e1 or e2 is null, these functions make the following computations:

  REGR_SXX: REGR_COUNT(e1,e2) * VAR_POP(e2)

  REGR_SYY: REGR_COUNT(e1,e2) * VAR_POP(e1)

  REGR_SXY: REGR_COUNT(e1,e2) * COVAR_POP(e1, e2)

CORR

- CORR returns the coefficient of correlation of a set of number pairs. Both expr1 and expr2 are number expressions. Oracle 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:

  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, it returns null.
- The following example returns the cumulative coefficient of correlation of monthly sales revenues and monthly units sold from the sample tables sales and times for year 1998:

```
SELECT ……,CORR (SUM(s.amount_sold), SUM(s.quantity_sold))
OVER (ORDER BY t.calendar_month_number) as CUM_CORR FROM
  sales s, times t WHERE s.time_id = t.time_id AND
  calendar_year = 1998 GROUP BY t.calendar_month_number
ORDER BY t.calendar_month_number;
CALENDAR_MONTH_NUMBER CUM_CORR
2                              1
3                              .994309382
4                              .852040875
5                              .846652204
6                              .871250628
```

COVAR_POP

- COVAR_POP returns the population covariance of a set of number pairs. Both expr1 and expr2 are number expressions. Oracle 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:

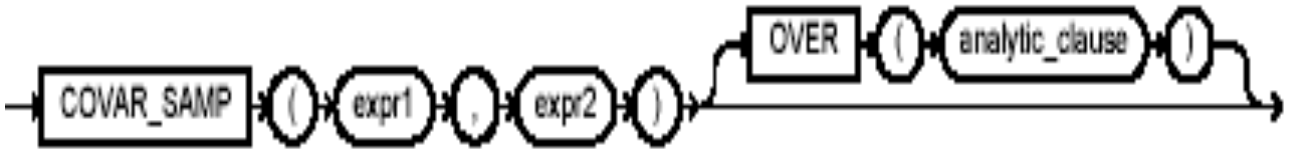
  (SUM(expr1 * expr2) - SUM(expr2) * SUM(expr1) / n) / n

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

- The following example calculates cumulative sample covariance of the list price and minimum price of the products:

```
SELECT……,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
```

COVAR_SAMP



- COVAR_SAMP returns the sample covariance of a set of number pairs. Both expr1 and expr2 are number expressions. Oracle 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:

  (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.

- The following example calculates cumulative sample covariance of the list price and minimum price of the products:

```
SELECT ....., 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            03088       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
```

FIRST and LAST

- The FIRST and LAST functions are very similar. They 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. 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.
- The following example returns, within each department of the demo 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 but returns the result for each employee within the department:

```
SELECT ……, 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;
LNAME DEPT_ID SALARY Worst Best
Whalen    10   4400  4400  4400
Goyal     20   6000  6000 13000
.
Greenberg 100 12000  6900 12000
Gietz     110 8300   8300 12000
```

Hypothetical Rank and Distribution Functions

- These functions provide functionality useful for what-if analysis.
- As an example, what would be the rank of a row, if the row was **hypothetically** inserted into a set of other rows?
- This family of aggregates takes one or more arguments of a hypothetical row and an ordered group of rows, returning the RANK, DENSE_RANK, PERCENT_RANK or CUME_DIST of the row as if it was hypothetically inserted into the group.
- **Hypothetical Rank and Distribution Syntax**

```
[RANK | DENSE_RANK | PERCENT_RANK | CUME_DIST](
 <constant expression> [, ...] ) WITHIN GROUP (
 ORDER BY <order by expression> [ASC|DESC] [NULLS
 FIRST|NULLS LAST][, ...] )
```

- Here, <constant expression> refers to an expression that evaluates to a constant, and there may be more than one such expressions that are passed as arguments to the function. The ORDER BY clause can contain one or more expressions that define the sorting order on which the ranking will be based. ASC, DESC, NULLS FIRST, NULLS LAST options will be available for each expression in the ORDER BY.

Hypothetical Rank and Distribution Functions

- Using the list price data from the products table, you can calculate the RANK, PERCENT_RANK and CUME_DIST for a hypothetical sweater with a price of $50 for how it fits within each of the sweater subcategories. The query and results are:

```
SELECT……, RANK(50) WITHIN GROUP (ORDER BY
  prod_list_price DESC) as HRANK,
  TO_CHAR(PERCENT_RANK(50) WITHIN GROUP
(ORDER BY prod_list_price),'9.999') AS HPERC_RANK,
  TO_CHAR(CUME_DIST (50) WITHIN GROUP (ORDER BY
  prod_list_price),'9.999') AS HCUME_DIST FROM products
WHERE prod_subcat LIKE 'Sweater%' GROUP BY prod_subcat;
```

PROD_SUBCAT	HRANK	HPERC_RANK	HCUME_DIST
Sweaters- Boys	16	.911	.912
Sweaters- Girls	1	1.000	1.000
Sweaters- Men	240	.351	.352
Sweaters- Women	21	.783	.785

Other Functions

ABS

- ABS returns the absolute value of n.

- The following example returns the absolute value of -15:

```
SELECT ABS(-15) "Absolute" FROM DUAL;

Absolute
----------
15
```

ADD_MONTHS

- ADD_MONTHS returns the date d plus n months.
- The argument n can be any integer.
- If d is the last day of the month or if the resulting month has fewer days than the day component of d, then the result is the last day of the resulting month.
- Otherwise, the result has the same day component as d.
- The following example returns the month after the hire_date in the sample table employees:

```
SELECT TO_CHAR( ADD_MONTHS(hire_date,1), 'DD-MON-
  YYYY') "Next month"
FROM employees
WHERE last_name = 'Baer';

Next Month
-----------
07-JUL-1994
```

ASCII

- ASCII returns the decimal representation in the database character set of the first character of char.
- char can be of datatype CHAR, VARCHAR2, NCHAR, or NVARCHAR2.
- The value returned is of datatype NUMBER.
- If your database character set is 7-bit ASCII, this function returns an ASCII value. If your database character set is EBCDIC Code, this function returns an EBCDIC value. There is no corresponding EBCDIC character function.

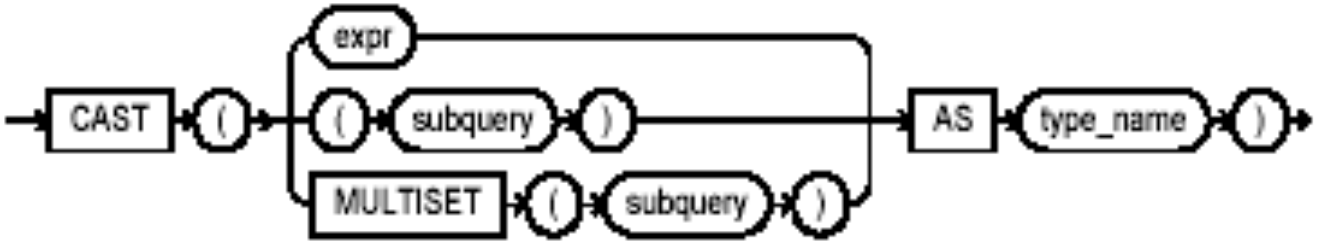
```
SELECT ASCII('Q') FROM DUAL;

ASCII('Q')
----------
81
```

ASCIISTR

- ASCIISTR takes as its argument a string in any character set and returns an ASCII string in the database character set.
- The value returned contains only characters that appear in SQL, plus the forward slash (/).

```
SELECT ASCIISTR('flauwekul') FROM
 DUAL;

ASCIISTR('FLAUW
---------------
\6<65\756<\6700
```

CAST



- A CAST function converts one built-in datatype or collection-typed value into another built-in datatype or collection-typed value.
- You can cast an unnamed operand (such as a date or the result set of a subquery) or a named collection (such as a varray or a nested table) into a type-compatible datatype or named collection.
- The *type_name* must be the name of a built-in datatype or collection type and the operand must be a built-in datatype or must evaluate to a collection value.
- For the operand, *expr* can be either a built-in datatype or a collection type, and *subquery* must return a single value of collection type or built-in type. MULTISET informs Oracle to take the result set of the subquery and return a collection value.
- If you want to cast a named collection type into another named collection type, the elements of both collections must be of the same type.
- If the result set of subquery can evaluate to multiple rows, you must specify the MULTISET keyword.
- The rows resulting from the subquery form the elements of the collection value into which they are cast. Without the MULTISET keyword, the subquery is treated as a scalar subquery.

CAST

- The following examples use the CAST function with scalar datatypes:

```
SELECT CAST('22-OCT-1997' AS DATE) FROM dual;
```

From/ To	CHAR, VARCHAR2	NUMBER	DATETIME / INTERVAL[b]	RAW	ROWID, UROWID	NCHAR, NVARCHAR2
CHAR, VARCHAR2	X	X	X	X	X	
NUMBER	X	X				
DATE, TIMESTAMP, INTERVAL	X		X			
RAW	X			X		
ROWID, UROWID	X				X[a]	
NCHAR, NVARCHAR2		X	X	X	X	X

[a] You cannot cast a UROWID to a ROWID if the UROWID contains the value of a ROWID of an index-organized table.

[b] Datetime/Interval includes DATE, TIMESTAMP, TIMESTAMP WITH TIMEZONE, INTERVAL DAY TO SECOND, and INTERVAL YEAR TO MONTH.

CAST

From/ To	CHAR, VARCHAR2	NUMBER	DATETIME / INTERVAL[b]	RAW	ROWID, UROWID	NCHAR, NVARCHAR2
CHAR, VARCHAR2	X	X	X	X	X	
NUMBER	X	X				
DATE, TIMESTAMP, INTERVAL	X		X			
RAW	X			X		
ROWID, UROWID	X				X[a]	
NCHAR, NVARCHAR2		X	X	X	X	X

[a] You cannot cast a UROWID to a ROWID if the UROWID contains the value of a ROWID of an index-organized table.

[b] Datetime/Interval includes DATE, TIMESTAMP, TIMESTAMP WITH TIMEZONE, INTERVAL DAY TO SECOND, and INTERVAL YEAR TO MONTH.

CEIL

- CEIL returns smallest integer greater than or equal to n.
- The following example returns the smallest integer greater than or equal to 15.7:

```
SELECT CEIL(15.7) "Ceiling" FROM DUAL;

Ceiling
----------
16
```

CHARTOROWID

- CHARTOROWID converts a value from CHAR, VARCHAR2, NCHAR, or NVARCHAR2 datatype to ROWID datatype.
- The following example converts a character rowid representation to a rowid. (The function will return a different rowid on different databases).

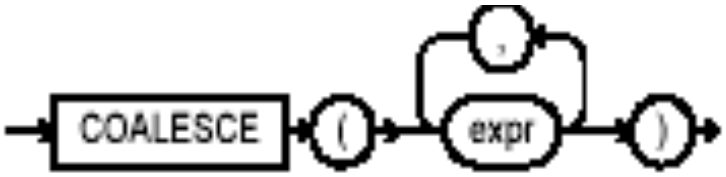
```
SELECT last_name FROM employees
WHERE ROWID =
  CHARTOROWID('AAAFYmAAFAAAAFEAAP');
LAST_NAME
-------------------------
Greene
```

CHR

- CHR returns the character having the binary equivalent to n in either the database character set or the national character set.
- If USING NCHAR_CS is not specified, this function returns the character having the binary equivalent to n as a VARCHAR2 value in the database character set.
- If USING NCHAR_CS is specified, this function returns the character having the binary equivalent to n as a NVARCHAR2 value in the national character set.

```
SELECT CHR(67)||CHR(65)||CHR(84) "Dog" FROM
  DUAL;

Dog
---
CAT
```

COALESCE



- The COALESCE function returns the first non-null expr in the expression list.
- At least one expr must not be the literal NULL. If all occurrences of expr evaluate to null, the function returns null.
- This function is a generalization of the NVL function.
- You can also use COALESCE as a variety of the CASE expression. For example,

`COALESCE (expr1, expr2)` is equivalent to:

```
CASE WHEN expr1 IS NOT NULL THEN expr1 ELSE expr2 END
```

`COALESCE (expr1, expr2, ..., expr n), for n>=3` is equivalent to:

```
CASE WHEN expr1 IS NOT NULL THEN expr1
ELSE COALESCE (expr2, ..., expr n) END
```

- The following example gives a 10% discount to all products with a list price. It there is no list price, the sale price is the minimum price, and if there is no minimum price, the sale price is "5":

```
SELECT ……, COALESCE(0.9*list_price, min_price, 5) "Sale"
FROM product_information WHERE supplier_id = 102050;
PRODUCT_ID LIST_PRICE MIN_PRICE Sale
3355                              5
1770         73         73
2378        305        247     274.5
```

CONCAT

- CONCAT returns char1 concatenated with char2.
- Both char1 and char2 can be any of the datatypes CHAR, VARCHAR2, NCHAR, NVARCHAR2, CLOB, or NCLOB.
- The string returned is of VARCHAR2 datatype and is in the same character set as char1.
- This function is equivalent to the concatenation operator (||).
- This example uses nesting to concatenate three character strings:

```
SELECT
CONCAT(CONCAT(last_name, '''s job category is '),
job_id) "Job‟ FROM employees WHERE employee_id = 152;

Job
----------------------------------------------------
Hall's job category is SA_REP
```

CURRENT_DATE

- CURRENT_DATE returns the current date in the session time zone, in a value in the Gregorian calendar of datatype DATE.
- The following example illustrates that CURRENT_DATE is sensitive to the session time zone:

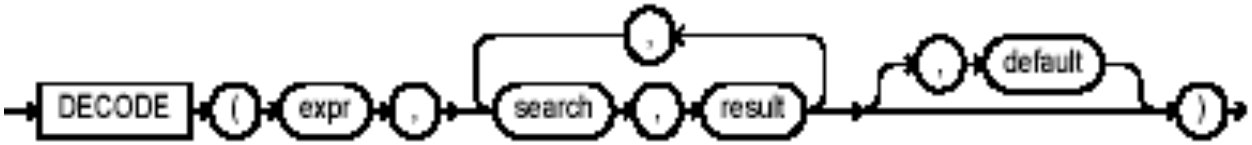
```
ALTER SESSION SET TIME_ZONE = '-5:0';
ALTER SESSION SET NLS_DATE_FORMAT = 'DD-MON-YYYY
   HH24:MI:SS';
SELECT SESSIONTIMEZONE, CURRENT_DATE FROM DUAL;

SESSIONTIMEZONE CURRENT_DATE
--------------- --------------------
-05:00 04-APR-2000 13:14:03
```

CURRENT_TIMESTAM

- CURRENT_TIMESTAMP returns the current date and time in the session time zone, in a value of datatype TIMESTAMP WITH TIME ZONE.
- The time zone displacement reflects the current local time of the SQL session. If you omit precision, the default is 6. The difference between this function and LOCALTIMESTAMP is that CURRENT_TIMESTAMP returns a TIMESTAMP WITH TIME ZONE value while LOCALTIMESTAMP returns a TIMESTAMP value.
- In the optional argument, precision specifies the fractional second precision of the time value returned.
- The following example illustrates that CURRENT_TIMESTAMP is sensitive to the session time zone:

```
ALTER SESSION SET TIME_ZONE = '-5:0';
ALTER SESSION SET NLS_DATE_FORMAT = 'DD-MON-YYYY HH24:MI:SS';
SELECT SESSIONTIMEZONE, CURRENT_TIMESTAMP FROM DUAL;

SESSIONTIMEZONE CURRENT_TIMESTAMP
--------------- ---------------------------------------------
-05:00 04-APR-00 01.17.56.917550 PM -05:00
```

DECODE



- A DECODE function compares *expr* to each *search* value one by one.
- If *expr* is equal to a *search*, Oracle returns the corresponding *result*. If no match is found, Oracle returns *default*, or, if *default* is omitted, returns null. If *expr* and *search* contain character data, Oracle compares them using nonpadded comparison semantics.
- The *search, result*, and *default* values can be derived from expressions. Oracle evaluates each *search* value only before comparing it to *expr*, rather than evaluating all *search* values before comparing any of them with *expr*. Oracle automatically converts *expr* and each *search* value to the datatype of the first *search* value before comparing.
- In a DECODE function, Oracle considers two nulls to be equivalent. If *expr* is null, Oracle returns the *result* of the first *search* that is also null.
- The maximum number of components in the DECODE function, including *expr*, *searches*, *results*, and *default* is 255.
- This example decodes the value warehouse_id. If warehouse_id is 1, the function returns 'Southlake'; if warehouse_id is 2, it returns 'San Francisco'; etc. If warehouse_id is not 1, 2, 3, or 4, the function returns 'Non-domestic'.

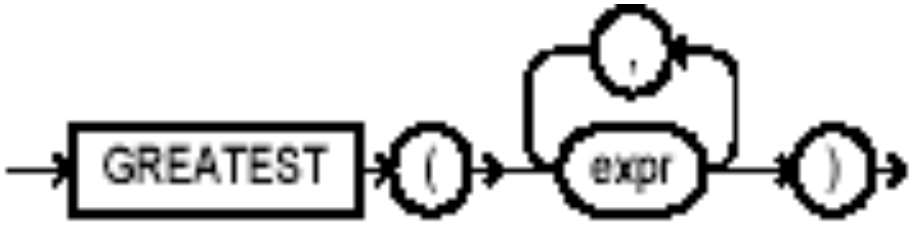
```
SELECT product_id, DECODE (warehouse_id, 1, 'Southlake',
  2, 'San Francisco', 3, 'New Jersey', 4, 'Seattle',
  'Non-domestic') quantity_on_hand FROM inventories;
```

FLOOR

- FLOOR returns largest integer equal to or less than n.
- The following example returns the largest integer equal to or less than 15.7:

```
SELECT FLOOR(15.7) "Floor" FROM DUAL;

Floor
----------
15
```

GREATEST



- GREATEST returns the greatest of the list of exprs.
- All exprs after the first are implicitly converted to the datatype of the first expr before the comparison. Oracle compares the exprs using nonpadded comparison semantics. Character comparison is based on the value of the character in the database character set. One character is greater than another if it has a higher character set value.
- If the value returned by this function is character data, its datatype is always VARCHAR2.
- The following statement selects the string with the greatest value:

SELECT GREATEST (’HARRY’, ’HARRIOT’, ’HAROLD’)

"Greatest" FROM DUAL;

Greatest

HARRY

GROUP_ID

- The GROUP_ID function distinguishes duplicate groups resulting from a GROUP BY specification.
- It is therefore useful in filtering out duplicate groupings from the query result.
- It returns an Oracle NUMBER to uniquely identify duplicate groups.
- This function is applicable only in a SELECT statement that contains a GROUP BY clause.
- If n duplicates exist for a particular grouping, it returns numbers in the range 0 to n-1.
- The following example assigns the value "1" to the duplicate co.country_region grouping from a query on the sample tables sh.countries and sh.sales:

```
SELECT ……, GROUP_ID() g FROM sales s, customers c, countries
  co WHERE s.cust_id = c.cust_id AND c.country_id =
  co.country_id AND s.time_id = '1-JAN-00' AND
  co.country_region IN ('Americas', 'Europe')
GROUP BY co.country_region;
COUNTRY_REGION COUNTRY_SUBREGION     Revenue  G
Americas         Northern America    220844  0
Americas         Southern America     10872  0
Europe             Eastern Europe     12751  0
Americas                            231716  0
Americas                            231716  1
Europe                              571437  1
```

INITCAP

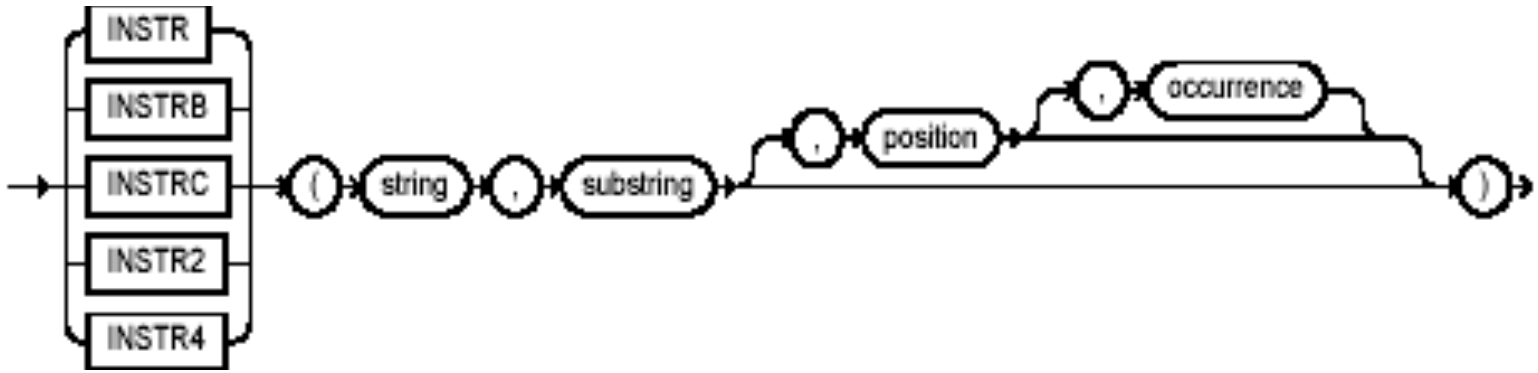
- INITCAP returns char, with the first letter of each word in uppercase, all other letters in lowercase.
- Words are delimited by white space or characters that are not alphanumeric.
- char can be of any of the datatypes CHAR, VARCHAR2, NCHAR, or NVARCHAR2.
- The return value is the same datatype as char.
- The following example capitalizes each word in the string:

```
SELECT INITCAP('the soap') "Capitals" FROM DUAL;

Capitals
---------
The Soap
```

INSTR

- The "instring" functions search *string* for *substring*.
- The function returns an integer indicating the position of the character in string that is the first character of this occurrence. INSTR calculates strings using characters as defined by the input character set. INSTRB uses bytes instead of characters. INSTRC uses unicode complete characters. INSTR2 uses UCS2 codepoints. INSTR4 uses UCS4 codepoints.
- Both string and substring can be any of the datatypes CHAR, VARCHAR2, NCHAR, NVARCHAR2, CLOB, or NCLOB. The value returned is of NUMBER datatype.
- If the search is unsuccessful (if substring does not appear occurrence times after the position character of string), the return value is 0.
- The following example searches the string "CORPORATE FLOOR", beginning with the third character, for the string "OR". It returns the position in CORPORATE FLOOR at which the second occurrence of "OR" begins:

```
SELECT INSTR('CORPORATE FLOOR','OR', 3, 2) "Instring"
   FROM DUAL;

Instring
------------
14
```



LAST_DAY

- LAST_DAY returns the date of the last day of the month that contains date.
- The following statement determines how many days are left in the current month.

```
SELECT SYSDATE, LAST_DAY(SYSDATE) "Last",
LAST_DAY(SYSDATE) - SYSDATE "Days Left"
FROM DUAL;
SYSDATE Last Days Left
--------- --------- ----------
23-OCT-97 31-OCT-97 8
```

LENGTH

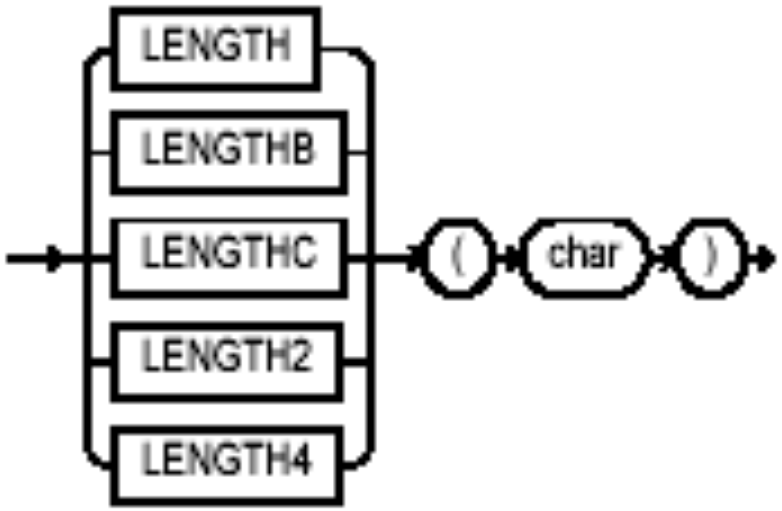
- The length functions return the length of char.
- LENGTH calculates length using characters as defined by the input character set. LENGTHB uses bytes instead of characters. LENGTHC uses unicode complete characters. LENGTH2 uses UCS2 codepoints. LENGTH4 uses UCS4 codepoints..
- The return value is of datatype NUMBER. If char has datatype CHAR, the length includes all trailing blanks.
- If char is null, this function returns null.
- The following examples use the LENGTH function using single- and multibyte database character set.

SELECT LENGTH('CANDIDE') "Length in characters"

FROM DUAL;

Length in characters

7



LOWER

- LOWER returns char, with all letters lowercase.
- The following example returns a string in lowercase:

```
SELECT LOWER('MR. SCOTT MCMILLAN') "Lowercase"
FROM DUAL;
Lowercase
--------------------
mr. scott mcmillan
```

LPAD

- LPAD returns char1, left-padded to length n with the sequence of characters in char2; char2 defaults to a single blank.
- If char1 is longer than n, this function returns the portion of char1 that fits in n.
- Both char1 and char2 can be any of the datatypes CHAR, VARCHAR2, NCHAR, NVARCHAR2, CLOB, or NCLOB. The string returned is of VARCHAR2 datatype and is in the same character set as char1.
- The argument n is the total length of the return value as it is displayed on your terminal screen.
- In most character sets, this is also the number of characters in the return value. However, in some multibyte character sets, the display length of a character string can differ from the number of characters in the string.
- The following example left-pads a string with the characters "*.":

SELECT LPAD('Page 1',15,'*.') "LPAD example"

FROM DUAL;

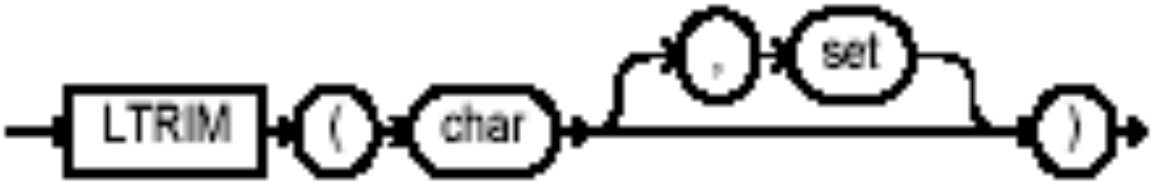
LPAD example

..*.*.*Page 1

LTRIM

- LTRIM removes characters from the left of char, with all the leftmost characters that appear in set removed; set defaults to a single blank.
- If char is a character literal, you must enclose it in single quotes.
- Oracle begins scanning char from its first character and removes all characters that appear in set until reaching a character not in set and then returns the result.
- Both char and set can be any of the datatypes CHAR, VARCHAR2, NCHAR, NVARCHAR2, CLOB, or NCLOB.
- The string returned is of VARCHAR2 datatype and is in the same character set as char.
- The following example trims all of the left-most x's and y's from a string:

```
SELECT LTRIM('xyxXxyLAST WORD','xy') "LTRIM example"
FROM DUAL;

LTRIM example
------------
XxyLAST WORD
```



MONTHS_BETWEEN

- MONTHS_BETWEEN returns number of months between dates date1 and date2.
- If date1 is later than date2, result is positive; if earlier, negative.
- If date1 and date2 are either the same days of the month or both last days of months, the result is always an integer.
- Otherwise Oracle calculates the fractional portion of the result based on a 31-day month and considers the difference in time components date1 and date2.
- The following example calculates the months between two dates:

```
SELECT MONTHS_BETWEEN
(TO_DATE('02-02-1995','MM-DD-YYYY'),
TO_DATE('01-01-1995','MM-DD-YYYY') ) "Months"
FROM DUAL;
Months
----------
1.03225806
```

NULLIF

- The NULLIF function compares expr1 and expr2.
- If they are equal, the function returns null.
- If they are not equal, the function returns expr1.
- You cannot specify the literal NULL for expr1.
- The NULLIF function is logically equivalent to the following CASE expression:

```
CASE WHEN expr1 = expr 2 THEN NULL ELSE expr1 END
```

- The following example selects those employees from the sample schema hr who have changed jobs since they were hired, as indicated by a job_id in the job_history table different from the current job_id in the employees table:

```
SELECT e.last_name, NULLIF(e.job_id, j.job_id) "Old Job
   ID‟ FROM employees e, job_history j
WHERE e.employee_id = j.employee_id;
LAST_NAME Old Job ID
------------------------- ----------
De Haan AD_VP
Kochhar AD_VP
Kochhar AD_VP
```

NVL

- If expr1 is null, NVL returns expr2.
- If expr1 is not null, NVL returns expr1. The arguments expr1 and expr2 can have any datatype. If their datatypes are different, Oracle converts expr2 to the datatype of expr1 before comparing them.
- The datatype of the return value is always the same as the datatype of expr1, unless expr1 is character data, in which case the return value's datatype is VARCHAR2 and is in the character set of expr1.
- The following example returns a list of employee names and commissions, substituting "Not Applicable" if the employee receives no commission:

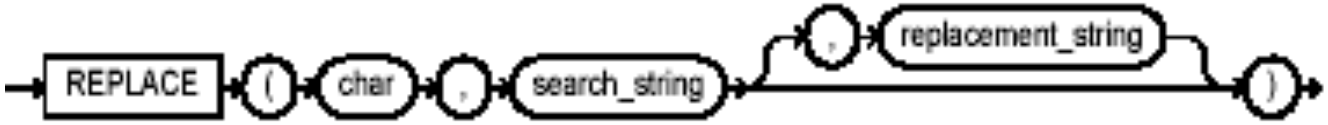
```
SELECT NVL(TO_CHAR(commission_pct), 'Not Applicable')
"COMMISSION" FROM employees WHERE lname LIKE 'B%';
LAST_NAME COMMISSION
Baer      Not Applicable
Banda       .11
Bates       .16
```

NVL2

- If expr1 is not null, NVL2 returns expr2.
- If expr1 is null, NVL2 returns expr3.
- The argument expr1can have any datatype. The arguments expr2and expr3can have any datatypes except LONG.
- If the datatypes of expr2 and expr3 are different, Oracle converts expr3 to the datatype of expr2 before comparing them unless expr3 is a null constant.
- The datatype of the return value is always the same as the datatype of expr2, unless expr2 is character data, in which case the return value's datatype is VARCHAR2.
- The following example shows whether the income of some employees is made up of salary plus commission, or just salary, depending on whether the commission_pct column of employees is null or not.

```
SELECT last_name, salary, NVL2(commission_pct, salary +
   (salary * commission_pct), salary) income FROM
   employees WHERE last_name like 'B%';
LAST_NAME SALARY INCOME
Baer       10000 10000
Baida       2900 2900
Banda       6200 6882
```

REPLACE



- REPLACE returns char with every occurrence of search_string replaced with replacement_string.
- If replacement_string is omitted or null, all occurrences of search_string are removed.
- If search_string is null, char is returned.
- This function provides a superset of the functionality provided by the TRANSLATE function. TRANSLATE provides single-character, one-to-one substitution. REPLACE lets you substitute one string for another as well as to remove character strings.
- The following example replaces occurrences of "J" with "BL":

```
SELECT REPLACE('JACK and JUE','J','BL') "Changes"
FROM DUAL;
Changes
--------------
BLACK and BLUE
```

ROUND (number)

- ROUND returns number rounded to integer places right of the decimal point.
- If integer is omitted, number is rounded to 0 places. integer can be negative to round off digits left of the decimal point. integer must be an integer.
- The following example rounds a number to one decimal point:

```
SELECT ROUND(15.193,1) "Round" FROM DUAL;
Round
----------
15.2
```

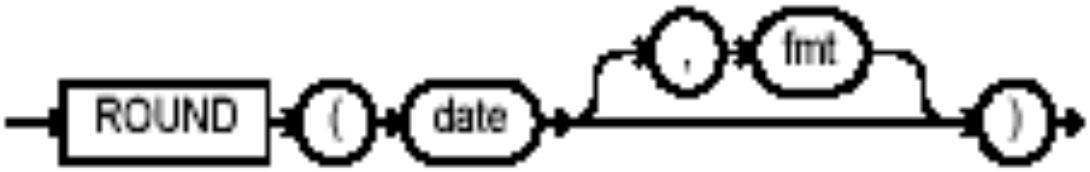
- The following example rounds a number one digit to the left of the decimal point:

```
SELECT ROUND(15.193,-1) "Round" FROM DUAL;
Round
----------
20
```

ROUND (date)

- ROUND returns date rounded to the unit specified by the format model fmt.
- If you omit fmt, date is rounded to the nearest day.
- The following example rounds a date to the first day of the following year:

```
SELECT ROUND (TO_DATE ('27-OCT-00'),'YEAR')
"New Year" FROM DUAL;
New Year
---------
01-JAN-01
```



RPAD

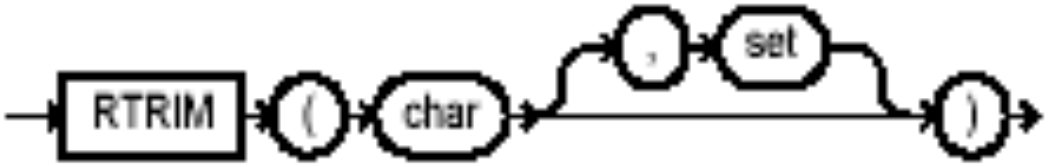
- RPAD returns char1, right-padded to length n with char2, replicated as many times as necessary; char2 defaults to a single blank.
- If char1 is longer than n, this function returns the portion of char1 that fits in n.
- The argument n is the total length of the return value as it is displayed on your terminal screen.
- The following example rights-pads a name with the letters "ab" until it is 12 characters long:

```
SELECT RPAD('MORRISON',12,'ab') "RPAD example"
FROM DUAL;
RPAD example
------------
MORRISONabab
```

RTRIM

- RTRIM returns char, with all the rightmost characters that appear in set removed; set defaults to a single blank.
- If char is a character literal, you must enclose it in single quotes. RTRIM works similarly to LTRIM.
- The following example trims the letters "xy" from the right side of a string:

```
SELECT RTRIM('BROWNINGyxXxy','xy') "RTRIM e.g."
FROM DUAL;
RTRIM e.g
-------------
BROWNINGyxX
```

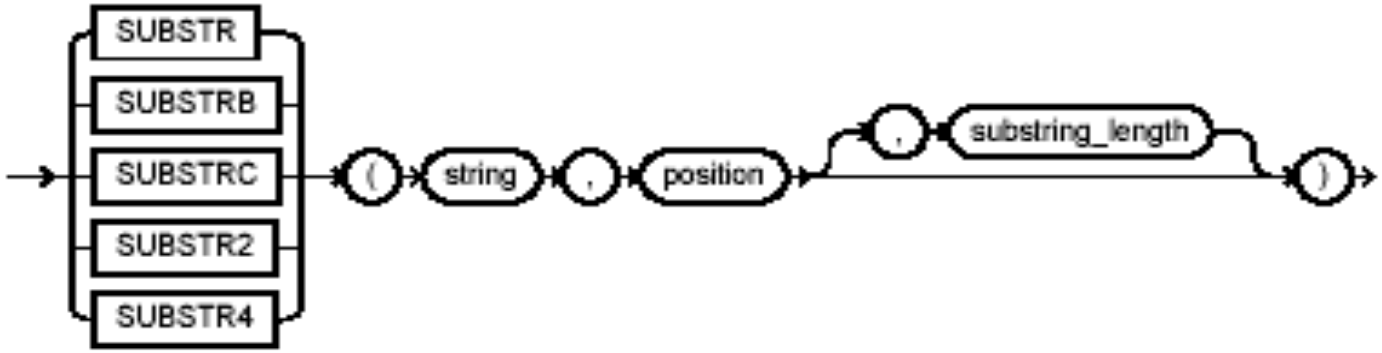


SOUNDEX

- SOUNDEX returns a character string containing the phonetic representation of char.
- This function lets you compare words that are spelled differently, but sound alike in English.
- The following example returns the employees whose last names are a phonetic representation of "Smyth":

```
SELECT last_name, first_name FROM hr.employees
WHERE SOUNDEX(last_name) = SOUNDEX('SMYTHE');
LAST_NAME FIRST_NAME
---------- ----------
Smith Lindsey
Smith William
```

SUBSTR



- The substring functions return a portion of string, beginning at character position, substring_length characters long.
- SUBSTR calculates lengths using characters as defined by the input character set. SUBSTRB uses bytes instead of characters. SUBSTRC uses unicode complete characters. SUBSTR2 uses UCS2 codepoints. SUBSTR4 uses UCS4 codepoints.
  - If position is 0, it is treated as 1.
  - If position is positive, Oracle counts from the beginning of string to find the first character.
  - If position is negative, Oracle counts backwards from the end of string.
  - If substring_length is omitted, Oracle returns all characters to the end of string. If substring_length is less than 1, a null is returned.
- The following example returns several specified substrings of "ABCDEFG":

```
SELECT SUBSTR('ABCDEFG',3,4) "Substring" FROM DUAL;
    Substring
    CDEF
```

SYSDATE

- SYSDATE returns the current date and time. Requires no arguments.
- In distributed SQL statements, this function returns the date and time on your local database.
- The following example returns the current date and time:

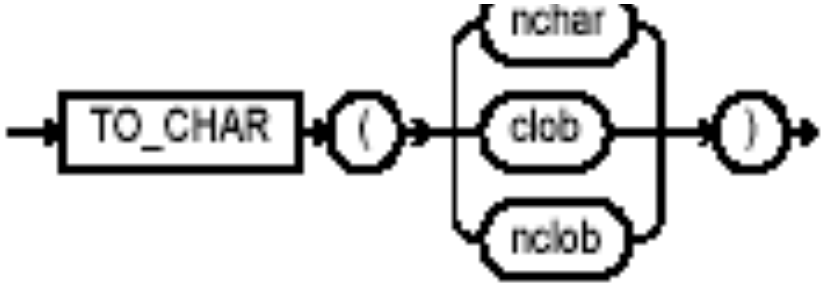
```
SELECT TO_CHAR
(SYSDATE, 'MM-DD-YYYY HH24:MI:SS')"NOW"
FROM DUAL;
NOW
--------------------
04-13-2001 09:45:51
```

SYSTIMESTAMP

- The SYSTIMESTAMP function returns the system date, including fractional seconds
- and time zone of the database. The return type is TIMESTAMP WITH TIME ZONE.
- The following example returns the system date:

```
SELECT SYSTIMESTAMP FROM DUAL;

SYSTIMESTAMP

28-MAR-00 12.38.55.538741 PM -08:00
```

TO_CHAR (character)



- The TO_CHAR (character) function converts NCHAR, NVARCHAR2, CLOB, or NCLOB data to the database character set.
- The following example converts some CLOB data from the pm.print_media table to the database character set:

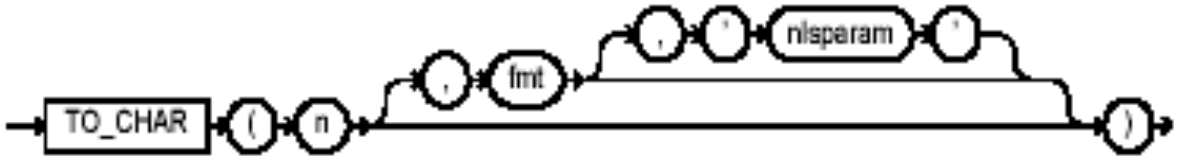
```
SELECT TO_CHAR(ad_sourcetext) FROM print_media
WHERE product_id = 2268;


TO_CHAR(AD_SOURCETEXT)
******************************
TIGER2 2268...Standard Hayes Compatible Modem
Product ID: 2268
The #1 selling modem in the universe! Tiger2's modem
  includes call
management and Internet voicing. Make real-time full
  duplex phone
calls at the same time you're online.
**********************************
```

TO_CHAR (datetim

- TO_CHAR converts date o… TIMESTAMP WITH LOCAL TIME ZONE datatype to a value of VARCHAR2 datatype in the format specified by the date format fmt.
- The ' nlsparams' specifies the language in which month and day names and abbreviations are returned. This argument can have this form:
- 'NLS_DATE_LANGUAGE = language'
- If you omit nlsparams, this function uses the default date language for your session.

```
SELECT TO_CHAR(ts_col, 'DD-MON-YYYY HH24:MI:SSxFF'),
TO_CHAR(tstz_col, 'DD-MON-YYYY HH24:MI:SSxFF
  TZH:TZM')
FROM my_tab;
TO_CHAR(TS_COL,'DD-MON-YYYYHH2 TO_CHAR(TSTZ_COL,'DD-MON-
  YYYYHH24:MI:
01-DEC-1999 10:00:00 01-DEC-1999 10:00:00.000000 -08:00
02-DEC-1999 10:00:00 02-DEC-1999 10:00:00.000000 -08:00
```

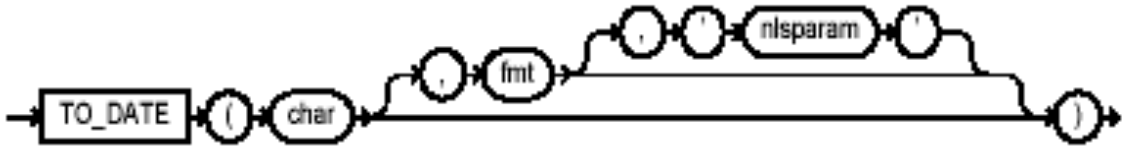
TO_CHAR (number)



- TO_CHAR converts n of NUMBER datatype to a value of VARCHAR2 datatype, using the optional number format fmt.
- The nlsparam specifies these characters that are returned by number format elements:
  - Decimal character
  - Group separator
  - Local currency symbol
  - International currency symbol
- This argument can have this form:
  - ’NLS_NUMERIC_CHARACTERS = ’’dg’’
  - NLS_CURRENCY = ’’text’’
  - NLS_ISO_CURRENCY = territory ’
- The characters d and g represent the decimal character and group separator, respectively. They must be different single-byte characters.
- In this example, the output is blank padded to the left of the currency symbol.

```
SELECT TO_CHAR(-10000,'L99G999D99MI') "Amount" FROM
   DUAL;
Amount
$10,000.00-
```

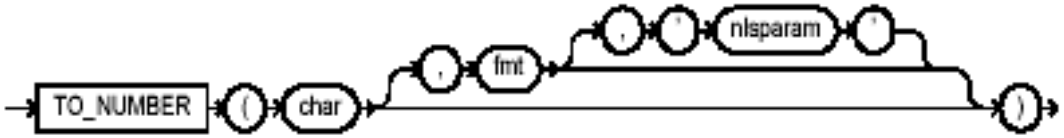
TO_DATE



- TO_DATE converts char of CHAR, VARCHAR2, NCHAR, or NVARCHAR2 datatype to a value of DATE datatype.
- The fmt is a date format specifying the format of char.
- The nlsparam has the same purpose in this function as in the TO_CHAR function for date conversion.
- The following example converts character strings into dates:

```
SELECT TO_DATE( 'January 15, 1989, 11:00 A.M.',
'Month dd, YYYY, HH:MI A.M.', 'NLS_DATE_LANGUAGE =
  American') FROM DUAL;


TO_DATE('
---------
15-JAN-89
```

TO_NUMBER



- TO_NUMBER converts char, a value of CHAR, VARCHAR2, NCHAR, or NVARCHAR2 datatype containing a number in the format specified by the optional format model fmt, to a value of NUMBER datatype.
- The following example converts character string data into a number:

```
UPDATE employees SET salary = salary +
  TO_NUMBER('100.00', '9G999D99')
WHERE last_name = 'Perkins';
```

- The nlsparam string in this function has the same purpose as it does in the TO_CHAR function for number conversions.

```
SELECT TO_NUMBER('-AusDollars100','L9G999D99', '
  NLS_NUMERIC_CHARACTERS = '',.'' NLS_CURRENCY =
  ''AusDollars'' ') "Amount" FROM DUAL;

Amount
----------
-100
```

TRANSLATE

- TRANSLATE returns char with all occurrences of each character in from_string replaced by its corresponding character in to_string. Characters not in from_string are not replaced.
- The argument from_string can contain more characters than to_string. In this case, the extra characters at the end of from_string have no corresponding characters in to_string.
- If these extra characters appear in char, they are removed from the return value.
- The following statement translates a license number. All letters 'ABC...Z' are translated to 'X' and all digits '012 . . . 9' are translated to '9':

```
SELECT TRANSLATE('2KRW229','0123456789AKLMNOPQRSTUWXYZ',
  '9999999999XXXXXXXXXXXXXXXXXX') "License" FROM DUAL;

      License
      9XXX999
```

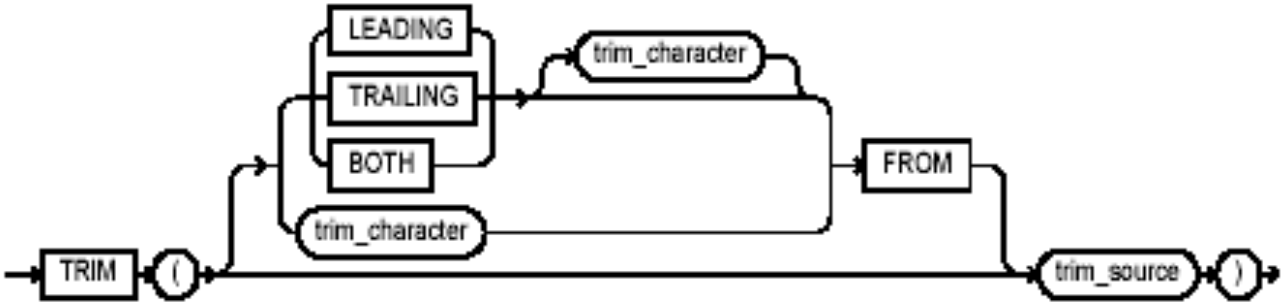
- The following statement returns a license number with the characters removed and the digits remaining:

```
SELECT TRANSLATE('2KRW229', '0123456789AKLMNOQRSVWXZ',
  '0123456789') "Translate example" FROM DUAL;

Translate example
-----------------
2229
```

TRIM

- TRIM enables you to trim leading or trailing characters (or both) from a character string.
- If trim_character or trim_source is a character literal, you must enclose it in single quotes.
- If you specify LEADING, Oracle removes any leading characters equal to trim_character.
- If you specify TRAILING, Oracle removes any trailing characters equal to trim_character.
- If you specify BOTH or none of the three, Oracle removes leading and trailing characters equal to trim_character.
- If you do not specify trim_character, the default value is a blank space.
- If you specify only trim_source, Oracle removes leading and trailing blank spaces.
- If either trim_source or trim_character is null then the TRIM function returns a null value.
- This example trims leading and trailing zeroes from a number:

```
SELECT TRIM (0 FROM 0009872348900) "TRIM Example"
FROM DUAL;
TRIM example
------------
98723489
```



TRUNC (number)

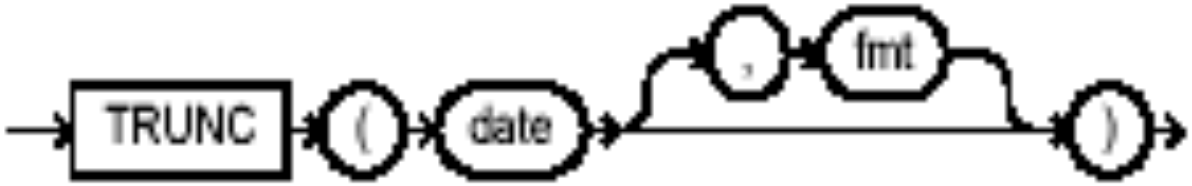
- TRUNC returns n truncated to m decimal places.
- If m is omitted, n is truncated to 0 places. m can be negative to truncate (make zero) m digits left of the decimal point.
- The following example truncate numbers:

```
SELECT TRUNC(15.79,1) "Truncate" FROM DUAL;
Truncate
----------
15.7
SELECT TRUNC(15.79,-1) "Truncate" FROM DUAL;
Truncate
----------
10
```

TRUNC (date)

- TRUNC returns datewith the time portion of the day truncated to the unit specified by the format model fmt.
- If you omit fmt, date is truncated to the nearest day.
- The following example truncates a date:

```
SELECT TRUNC(TO_DATE('27-OCT-92','DD-MON-YY'),
  'YEAR')
"New Year" FROM DUAL;
New Year
---------
01-JAN-92
```



UPPER

- UPPER returns char, with all letters uppercase. char can be any of the datatypes CHAR, VARCHAR2, NCHAR, NVARCHAR2, CLOB, or NCLOB.
- The return value is the same datatype as char.
- The following example returns a string in uppercase:

```
SELECT UPPER('Large') "Uppercase" FROM DUAL;

Upper
-----
LARGE
```

VSIZE

- VSIZE returns the number of bytes in the internal representation of expr.
- If expris null, this function returns null.
- The following example returns the number of bytes in the last_name of the employee in department 10:

```
SELECT last_name, VSIZE (last_name) "BYTES"
FROM employees
WHERE department_id = 10;
LAST_NAME        BYTES
---------------- ----------
Whalen                6
```

Format Models used in Functions like TO_CHAR(number) etc.

- A **format model** is a character literal that describes the format of DATE or NUMBER data stored in a character string.
- When you convert a character string into a date or number, a format model tells Oracle how to interpret the string.
- In SQL statements, you can use a format model as an argument of the TO_CHAR and TO_DATE functions:
  - To specify the format for Oracle to use to return a value from the database
  - To specify the format for a value you have specified for Oracle to store in the database

Format Models used in Functions like TO_CHAR(number) etc.

- You can use a format model to specify the format for Oracle to use to return values from the database to you.
- The following statement selects the salaries of the employees in Department 80 and uses the **TO_CHAR** function to convert these salaries into character values with the format specified by the number format model '**$9,990.99**':

```
SELECT last_name employee, TO_CHAR(salary,
  '$99,990.99')
FROM employees
WHERE department_id = 80;
```

- Because of this format model, Oracle returns salaries with leading dollar signs, commas every three digits, and two decimal places.
- Different categories of format models include:
  1. Number Format Models
  2. Date Format Models
  3. Format Model Modifiers

Number Format Models

- You can use number format models:
  - In the TO_CHAR function to translate a value of NUMBER datatype to VARCHAR2 datatype
  - In the TO_NUMBER function to translate a value of CHAR or VARCHAR2 datatype to NUMBER datatype

Table 2–10 Number Format Elements

Element	Example	Description
. (comma)	9,999	Returns a comma in the specified position. You can specify multiple commas in a number format model. Restrictions: ▪ A comma element cannot begin a number format model. ▪ A comma cannot appear to the right of a decimal character or period in a number format model.
. (period)	99.99	Returns a decimal point, which is a period (.) in the specified position. Restriction: You can specify only one period in a number format model.
$	$9999	Returns value with a leading dollar sign.
0	0999	Returns leading zeros.
	9990	Returns trailing zeros.
9	9999	Returns value with the specified number of digits with a leading space if positive or with a leading minus if negative. Leading zeros are blank, except for a zero value, which returns a zero for the integer part of the fixed-point number.
B	B9999	Returns blanks for the integer part of a fixed-point number when the integer part is zero (regardless of "0"s in the format model).
C	C999	Returns in the specified position the ISO currency symbol (the current value of the NLS_ISO_CURRENCY parameter).
D	99D99	Returns in the specified position the decimal character, which is the current value of the NLS_NUMERIC_CHARACTER parameter. The default is a period (.). Restriction: You can specify only one decimal character in a number format model.

Table 2–10 Number Format Elements

Element	Example	Description
EEEE	`9.9EEEE`	Returns a value using in scientific notation.
FM	`FM90.9`	Returns a value with no leading or trailing blanks.
G	`9G999`	Returns in the specified position the group separator (the current value of the `NLS_NUMERIC_CHARACTER` parameter). You can specify multiple group separators in a number format model. Restriction: A group separator cannot appear to the right of a decimal character or period in a number format model.
L	`L999`	Returns in the specified position the local currency symbol (the current value of the `NLS_CURRENCY` parameter).
MI	`9999MI`	Returns negative value with a trailing minus sign (-). Returns positive value with a trailing blank. Restriction: The MI format element can appear only in the last position of a number format model.
PR	`9999PR`	Returns negative value in <angle brackets>. Returns positive value with a leading and trailing blank. Restriction: The PR format element can appear only in the last position of a number format model.
RN	`RN`	Returns a value as Roman numerals in uppercase.
rn	`rn`	Returns a value as Roman numerals in lowercase. Value can be an integer between 1 and 3999.
S	`S9999`	Returns negative value with a leading minus sign (-). Returns positive value with a leading plus sign (+).
	`9999S`	Returns negative value with a trailing minus sign (-). Returns positive value with a trailing plus sign (+). Restriction: The S format element can appear only in the first or last position of a number format model.

Table 2–10 Number Format Elements

Element	Example	Description
TM	`TM`	"Text minimum". Returns (in decimal output) the smallest number of characters possible. This element is case-insensitive. The default is TM9, which returns the number in fixed notation unless the output exceeds 64 characters. If output exceeds 64 characters, Oracle automatically returns the number in scientific notation. Restrictions: ▪ You cannot precede this element with any other element. ▪ You can follow this element only with 9 or E (only one) or e (only one).
U	`U9999`	Returns in the specified position the "Euro" (or other) dual currency symbol (the current value of the `NLS_DUAL_CURRENCY` parameter).
V	`999V99`	Returns a value multiplied by 10^n (and if necessary, round it up), where n is the number of 9's after the "V".

X	`XXXX`
	
	`xxxx`
		Returns the hexadecimal value of the specified number of digits. If the specified number is not an integer, Oracle rounds it to an integer.
		
		Restrictions:
		
		▪ This element accepts only positive values or 0. Negative values return an error.
		
		▪ You can precede this element only with 0 (which returns leading zeroes) or FM. Any other elements return an error. If you specify neither 0 nor FM with X, the return always has 1 leading blank.

Date Format Models

- You can use date format models:
  - In the `TO_DATE` function to translate a character value that is in a format other than the default date format into a `DATE` value
  - In the `TO_CHAR` function to translate a `DATE` value that is in a format other than the default date format into a string (for example, to print the date from an application)

Table 2–12 Datetime Format Elements

Element	Specify in TO_* datetime functions?[a]	Meaning
- / , . ; : "text"	Yes	Punctuation and quoted text is reproduced in the result.
AD A.D.	Yes	AD indicator with or without periods.
AM A.M.	Yes	Meridian indicator with or without periods.
BC B.C.	Yes	BC indicator with or without periods.
CC SCC	No	One greater than the first two digits of a four-digit year; "S" prefixes BC dates with "-". For example, '20' from '1900'.
D	Yes	Day of week (1-7).
DAY	Yes	Name of day, padded with blanks to length of 9 characters.
DD	Yes	Day of month (1-31).
DDD	Yes	Day of year (1-366).
DY	Yes	Abbreviated name of day.
E	No	Abbreviated era name (Japanese Imperial, ROC Official, and Thai Buddha calendars).
EE	No	Full era name (Japanese Imperial, ROC Official, and Thai Buddha calendars).
FF		Fractional seconds; no radix printed (see X format element below). Example: 'HH:MI:SS.FF'.
HH	Yes	Hour of day (1-12).

HH12	No	Hour of day (1-12).
HH24	Yes	Hour of day (0-23).
IW	No	Week of year (1-52 or 1-53) based on the ISO standard.
IYY IY I	No	Last 3, 2, or 1 digit(s) of ISO year.
IYYY	No	4-digit year based on the ISO standard.
J	Yes	Julian day; the number of days since January 1, 4712 BC. Number specified with 'J' must be integers.
MI	Yes	Minute (0-59).
MM	Yes	Month (01-12; JAN = 01).
MON	Yes	Abbreviated name of month.
MONTH	Yes	Name of month, padded with blanks to length of 9 characters.
PM P.M.	No	Meridian indicator with or without periods.
Q	No	Quarter of year (1, 2, 3, 4; JAN-MAR = 1).
RM	Yes	Roman numeral month (I-XII; JAN = I).
RR	Yes	Given a year with 2 digits: ▪ If the year is <50 and the last 2 digits of the current year are >=50, the first 2 digits of the returned year are 1 greater than the first 2 digits of the current year. ▪ If the year is >=50 and the last 2 digits of the current year are <50, the first 2 digits of the returned year are 1 less than the first 2 digits of the current year.

HH12	No	Hour of day (1-12).
HH24	Yes	Hour of day (0-23).
IW	No	Week of year (1-52 or 1-53) based on the ISO standard.
IYY IY I	No	Last 3, 2, or 1 digit(s) of ISO year.
IYYY	No	4-digit year based on the ISO standard.
J	Yes	Julian day; the number of days since January 1, 4712 BC. Number specified with 'J' must be integers.
MI	Yes	Minute (0-59).
MM	Yes	Month (01-12; JAN = 01).
MON	Yes	Abbreviated name of month.
MONTH	Yes	Name of month, padded with blanks to length of 9 characters.
PM P.M.	No	Meridian indicator with or without periods.
Q	No	Quarter of year (1, 2, 3, 4; JAN-MAR = 1).
RM	Yes	Roman numeral month (I-XII; JAN = I).
RR	Yes	Given a year with 2 digits: ▪ If the year is <50 and the last 2 digits of the current year are >=50, the first 2 digits of the returned year are 1 greater than the first 2 digits of the current year. ▪ If the year is >=50 and the last 2 digits of the current year are <50, the first 2 digits of the returned year are 1 less than the first 2 digits of the current year.

YEAR SYEAR	No	Year, spelled out; "S" prefixes BC dates with "-".
YYYY SYYYY	Yes	4-digit year; "S" prefixes BC dates with "-".
YYY YY Y	Yes	Last 3, 2, or 1 digit(s) of year.