Oracle官方文档SQL Language Reference阅读笔记 Functions

Functions :

SQL Functions :

Single-Row Functions : return a single result row for every row of a queried table or view.
Numeric Functions : accept numeric input and return numeric values.
abs : returns the absolute value of n.
eg :
selectabs(-15) "Absolute" from dual;

acos : returns the arc cosine of n. the argument n must be in the range of -1 to 1.
eg :
select acos(.3) "Arc_Cosine" from dual;

asin : returns the arc sin of n. The argument n must be in the range of -1 to 1.
eg :
select asin(.3) "Arc_Sine" from dual;

atan : returns the arc tangent of n. The argument n can be in an unbounded range.
eg :
seect atan(.3) "Arc_Tangent" from dual;

atan2 : returns the arc tangent of n1 and n2. The argument n1 can be in an unbounded range.
eg :
select atan(.3, .2) "Arc_Tangent" from dual;

bitand : treats its input and its output as vectors of bits; the output is the bitwise and of the input.
eg
select bitand(6,3) from dual;
select bitand(bin_to_num(1,1,0), bin_to_num(0,1,1)) "Binary" from dual;

ceil : returns the smallest integer that is greater than or equal to n.
eg :
select order_total, ceil(order_total) from orders where order_id = 2434;

cos : returns the cosine of n.
eg :
select cos(180 * 3.14159265359/180) "Cosine of 180 degress" from dual;

cosh : returns the hyperbolic cosine of n.
eg :
select cosh(0) "Hyperbolic cosine of 0" from dual;

exp : returns e raised to the nth power, where e = 2.71828183... The function returns a value of the same type as the argument.
eg :
select exp(4) "e to the 4th power" from dual.

floor : returns the largest integer equal to or less than n.
eg :
select floor(15.7) "Floor" from dual;

ln : returns the natural logarithm of n, where n is greater than 0.
eg :
select ln(95) "Natural log of 95" from dual;

log : returns the logarithm, base n2 of n1.
eg :
select log(10,100) "Log base 10 of 100" from dual;

mod : returns the remainder of n2 devided by n1. Returns n2 if n1 is 0.
eg :
select mod(11,4) "Modulus" from dual;

nanvl : returns an alternative value n1 if the input value n2 is NaN (not a number). If n2 is not NaN, then it returns n2.
eg :
insert into float_point_demo values (0, 'NaN', 'NaN');
select * from float_point_demo;
select bin_float, NANVL(bin_float, 0) from float_point_demo;

power : returns n2 raised to n1 power. The base n2 and the exponent n1 can be any numbers, but if n2 is negative, then n1 must be an integer.
eg :
select power(3 2) "Raised" from dual;

remainder : returns the remainder of n2 divided by n1.
eg :
select bin_float, bin_double, remainder(bin_float, bin_double) from float_point_demo;

round(number) : returns n rounded to integer places to the right of the decimal point.
eg :
select round(15.193, 1) "Round" from dual;
select round(15.193, -1) "Round" from dual;

sign : returns the sign of n.
eg :
select sign(-15) "Sign" from dual;

sin : returns the sine of n(an angle expressed in radians).
eg :
select sin(30 * 3.14159265259/180) "Sine of 30 degrees" from dual;

sinh : returns the hyperbolic sine of n.
eg :
select sinh(1) "Hyperbolic sine of 1" from dual;

sqrt : returns the square root of n.
eg :
select sqrt(16) "Square root" from dual;

tan : returns the tangent of n(an angle expressed in radians).
eg :
select tan(135 * 3.14159265359/180) "Tangent of 135 degrees" from dual;

tanh : returns the hyperbolic tangent of n.
eg :
select tanh(.5) "Hyperbolic tangent of .5" from dual;

trunc(number) : returns n1 truncated to n2 decimal places. If n2 is omitted, then n1 is truncated to 0 places.
eg :
select trunc(15.79, 1) "Truncate" from dual;
select trunc(15.79, -1) "Truncate" from dual;

width_bucket : It lets you construct equiwidth histograms, in which the histogram range is divided into intervals that have

identical size. Ideally each bucket is a closed-open interval of the real number line. For example, a bucket can be assigned to scores between 10.00 and 19.999... to indicate that 10 is included in the interval and 20 is excluded. This is sometimes denoted [10, 20).
eg :
width_bucket(expr, min_value, max_value, num_buckets)
note :
expr : the expression for which the histogram is being created.
min_value and max_value are expressions that resolve to the end points of the acceptable range for expr.
num_buckets : an expression that resolves to a constant indicating the number of buckets.
eg : create a ten_bucket histogram on the credit_limit column for customers in Switzerland in the sample table oe.customers and returns the bucket number ("Credit Group") for each customer. Customers with credit limits greater than or equal to the maximum value are assigned to the overflow bucket, 11
select customer_id, cust_last_name, credit_limit, width_bucket(credit_limit, 100, 5000, 10) "Credit Group" from customer where nls_territory = "SWITZERLAND' order by "Credit Group", customer_id, cust_last_name, credit_limit;

Character Functions Returning Character Values : return character values
chr : returns the character having the binary equivalent to n as a varchar2 value in either the database character set or, if ou specify using nchar_cs, the national character set.
eg : the following example is run on an ASCII-based machine with the database character set defined as WE8ISO8859P1
select chr(67) || chr(65) || chr(84) "Dog" from dual;
eg : To produce the same results on an EBCDIC-based machine with the WE8EBCDIC1047 character set, the preceding example would have to be modified as follows:
select chr(195) || chr(193) || chr(227) "Dog" from dual;

concat : returns chr1 concatenated with char2.
eg :
select concat(concat(last_name, '''s job category is '), job_id) "Job" from employees where employee_id = 152;

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.
eg : capitalizes each word in the string
select initcap('the soap') "Capitals" from dual;

lower : returns char, with all letters lowercase.
eg :
select lower('MR. SCOTT MCMILIAN') "Lowercase" from dual;

lpad : returns expr1, left-padded to length n characters with the sequence of characters in expr2. This function is useful for formatting the output of a query.
eg :
lpad(expr1, n, expr2)
eg :
select lpad('Page 1', 15, '*.') "LPAD example" from dual;

ltrim : removes from the left end of char all of the characters contained in set.
eg :
ltrim(char, set)
eg :
select ltrim('<======>EROWNING<===>','<>=') "LTRIM Example" from dual;
nchr : returns the character having the binary equivalent to number in the national character set.
eg :
select nchr(187) from dual;
select chr(187 using nchr_cs) from dual;

nls_initcap : returns char, with the first letter of each word in uppercase, all other letters in lowercase.
eg :
select nls_initcap('ijsland') "IniCap" from dual;

select nls_INITCAP('IJSLAND','NLS_SORT = XDutch') "InitCap" from dual;

nls_lower : returns char, with all letters lowercase.
eg :
select nls_lower('NOKTASINDA', 'NLS_SORT = XTurkish') "Lowercase" from dual;

nlssort : returns the string of bytes used to sort char.
eg :
select * from test order by nlssort(name, 'NLS_SORT = XDanish');
eg :
select * from test where name 'Gaberd' order by name;
select * from test where NLSSORT(name, 'NLS_SORT = XDanish') > NLSSORT('Gaberd', 'NLS_SORT = XDanish) order by name;
eg :
alter session set nls_comp = 'LINGUISTIC';
alter session set nls_sort = 'XDanish';
select * from test where name > 'Gaberd' order by name;

nls_upper : returns char, with all letters uppercase.
eg :
select6 nls_upper('große') "Uppercase" from dual;
select NLS_UPPER('große', 'NLS_SORT = XGerman') "Uppercase" from dual;

regexp_replace : extends the functionality of the REPLACE function by letting you search a string or a regular expression pattern.
eg :
select regexp_replace(phone_number, '([[:digit:]]{3}\,([[:digit:]]{3}\,.([[:digit:]]{4})', '(\1) \2-\3') "REGEXP_REPLACE" from employees order by 'REGEXP_REPLACE";

regexp_substr : extends the functionality of the SUBSTR function by letting you search a strng for a regular expression pattern.
eg :
select regexp_substr('500 Oracle Parkway, Redwood Shored, CA', ',[^.]+,') "REGEXPR_SUBSTR" from dual;
select regexp_substr('http://www.example.com/products', 'heep://([[:alnum:]]+\.?) {3,4}/?') "REGEXP_SUBSTR" from dual;

replace : returns char with every occurrence of search_string replaced with replacement_string. If replacement_string is omitted or null, then all occurrences of search_string are removed. If search_string is null, then char is returned.
eg :
replace(char, search_string, replacement_string)
eg :
select replace('JACK and JUE', 'J', 'BL') "Changes" from dual;

rpad : returns exp1, right-padded to length n characters with expr2, replicated as many times as necessary. This function is useful for formatting the output of a query.
eg :
rpad(expr1, n, expr2)
eg : creates a simple chart of salary amounts by padding a single space with asterisks.
select last_name, RPAD(' ', salary/1000/1, '*') "Salary" from employees where department_id = 80 order by last_name, "Salary";

rtrim : removes from the right end of char all of the characters that appear in set. This function is useful for formatting to the output of a query.
eg :
select rtrim('<======>EROWNING<===>','<>=') "RTRIM Example" from dual;

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.
eg :
select last_name, first_name from hr.employees where soundex(last_name) = soundex('SMYTHE') order by last_name, first_name;

substr : return a portion of char, beginning at character position, substring_length characters long.

eg :
select substr('ABCDEFG', 3, 4) "Substring" from dual;
select substr('ABCDEFG', -5, 4) "Substring" from dual;
select substrb('ABCDEFG', 5,4.2) "Substring with bytes" from dual;

translate : returns expr with all occurrences of each character in from_string replaced by its corresponding character in to_string.
eg :
select translate('SQL*Plus User''s Guide', '*/''','__') from dual;

translate ... using : converts char into the character set specified for conversions between the database character set and the national character set.
eg :
update translate_tab set char_col = translate(nchar_col using char_cs);

trim : enables you to trim leading or trailing characters (or both) from a character string.
eg :
select employee_id, to_char(trim(leading 0 from hiredate)) from employees where department_id = 60 order by employee_id;

upper : returns char, with all letters uppercase.
eg :
select upper(last_name) "Uppercase" from employees;

NLS Character Functions : return information about the character set.
nls_charset_decl_len : returns the declaration length (in number of characters) of an NCHAR column. The byte_count argument is the width of the column. The char_set_id argument is the character set ID of the column.
eg :
nls_charset_decl_len(byte_count, char_set_id')
eg : returns the number of characters that are in a 200-byte column when you are using a multibyte character set
select nls_charset_decl_len(200, nls_charset_id('ja16eucfixed')) from dual;

nls_charset_id : returns the character set ID number corresponding to character set name string.
eg : returns the character set ID of a character set
select nls_charset_id('ja16euc') from dual;

nls_charset_name : returns the name of the character set corresponding to ID number number.
eg :
select nls_charset_name(2) from dual;

Character Functions Returning Number Values :
ASCII : returns the decimal representation in the database character set of the first character of char.
eg :
select last_name from employees where ASCII(SUBSTR(last_name, 1, 1)) = 76 order by last_name;

instr : search string for substring.
eg : 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;

length : return the length of char.
eg :
select length('CANDIDE') "Length in characters" from dual;

regexp_count : complements the functionality of the REGEXP_INSTR function by returning the number of times a pattern occurs in a source string.
eg : subexpressions parentheses in pattern are ignored
select regexp_count(123123123123123', '(12)3', 1, 'i') regexp_count from dual;
eg : the function begins to evaluate the source string at the third character, so skips over the first occurrence of pattern.

select regexp_count('123123123123', '123', 3, 'i') count from dual;

regexp_instr : extends the functionality of the INSTR function by letting you search a string for a regular expression pattern.
eg : examines the string, looking for occurrences of one or more none-blank characters. Oracle begins searching at the first character in the string and returns the starting position (default) of the sixth occurrence of one or more non-blank characters.
select regexp_instr('500 Oracle Parkway, Redwood Shores, CA', '[^]+', 1, 6) "REGEXP_INSTR" from dual;
eg : examines the string, looking for occurrences of words beginning with s, r, or p, regrardless of case, followed by any six alphabetic characters. Oracle begins searching at the third character in the string and returns the position in the string of the character following the second occurrence of a seven-letter word beginning with s, r, or p, regardless of case.
select regexp_instr('500 Oracle Parkway, Redwood Shores, CA' , '[s|r|p][[:alpha:]] {6}', 3, 2, 1, 'i') "REGEXP_INSTR" from dual;

Datetime Functions : operate on date, timestamp, and interval values.
add_months : returns the date date plus integer months.
eg :
select to_char(add_months(hire_date, 1), 'DD-MON-YYYY') "Next month" from employees where last_name = 'Baer';

current_date : returns the current date in the session time zone, in a value in the Gregorian calendar of data type DATE.
eg :
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;
alter session set time_zone = '-8:0';
select sessiontimezone, current_date from dual;

current_timestamp : returns the current date and time in the session time zone, in a value of data type TIMESTAMP WITH TIME ZONE. The time zone offset reflects the current local time of the SQL session. If you omit precision, then the default is 6.
eg :
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;
alter session set time_zone = '-8:0';
select sessiontimezone, current_timestamp from dual;

dbtimezone : returns the value of the database time zone. The return type is a time zone offset (a character type in the format '[+|-]TZH:TZM') or a time zone region name, depending on how the user specified the database time zone value in the most recent create database or alter database statement.
eg : the following example assumes that the database time zone is set to UTC time zone
select dbtimezone from dual;

extract(datetime) : extracts and returns the value of a specified datetime field from a datetime or interval expression.
eg :
select extract(month from order_date) "Month", count(order_date) "No. of Orders" from orders group by extract(month from order_date) order by "No. of Orders" desc, "Month";
eg :
select extract(year from date '1998-03-07') from dual;
select last_name, employee_id, hiredate from employees where extract(year from to_date(hire_date, 'DD-MON-RR')) > 2007 order by hire_date;
eg : the following example results in ambiguity, so Oracle returns UNKNOWN
select extract(timezone_region from timestamp '1999-01-01 10:00:00 -08:00') from dual;

from_tz : converts a timestamp value and a time zone to a TIMESTAMP WITH TIME ZONE value.
eg : the following example returns a timestamp value to TIMESTAMP WITH TIME ZONE
select from_tz(timestamp '2000-03-28 08:00:00', '3:00') from dual;

last_day : returns the date of the last day of the month that contains date. The last day of the month is defined by the session parameter NLS_CALENDAR. The return type is always DATE, regardless of the data type of date.
eg : determines how many days are left in the current month.
select sysdate, last_day(sysdate) "Last", last_day(sysdate) - sysdate "Days Left" from dual;

eg : adds 5 months to the hire date of each employee to give an evaluation date.
select last_name, hire_date, to_char(add_months(last_day(hire_date), 5)) "Eval Date" from employees order by last_name, hire_date;

localtimestamp : returns the current date and time in the session time zone in a value of data type TIMESTAMP.
eg : the following example illustrates the difference between LOCALTIMESTAMP and CURRENT_TIMESTAMP.
alter session set time_zone = '-5:00';
select current_timestamp, localtimestamp from dual;
alter session set time_zone = '-8:00';
select current_timestamp, localtimestamp from dual;
eg :
create table local_test(col1 timestamp with local time zone);
eg : this fails
insert into local_test values (to_timestamp(localtimestamp, 'DD-MON-RR HH.MI.SSXFF')):
eg : this succeeds
insert into local_test values (to_timestamp(localtimestamp, 'DD-MON-RR HH.MI.SSXFF PM'));

months_between : returns number of months between dates date1 and date2.
eg :
select months_between (to_date(02-02-1995', 'MM-DD-YYYY'), to_date('01-01-1995', 'MM-DD-YYYY')) "Months" from dual;

new_time : returns the date and time in time zone timezone2 when date and time in time zone timezone1 are date. Before using this function, you must set the NLS_DATE_FORMAT parameter to display 24-hour time.
eg : returns an Atlantic Standard time, given the Pacific Standard time equivalent
alter session set nls_date_format = 'DD-MON-YYYY HH24:MI:SS';
select new_time(to_date('11-10-09 01:23:45', 'MM-DD-YY HH24:MI:SS'), 'AST', 'PST') "New Date and Time" from dual;

next_day : returns the date of the first weekday named by char that is later than the date date.
eg : returns the date of the next Tuesday after October 15, 2009.
select next_day('15-OCT-2009', 'TUESDAY') "NEXT DAY" from dual;

numtodsinterval : converts n to an INTERVAL DAY TO SECOND literal.
eg : uses NUMTODSINTERVAL in a COUNT analytic function to calculate, for each employee, the number of employees hired by the same manager within the past 100 days from his or her hire date.
select manager_id, last_name, hire_date, count(*) over (partition by manager_id order by hire_date range numtodsinterval(100, 'day') preceding) as t_count from employees order by last_name, hire_date;

numtoyminterval : converts number n to an INTERVAL YEAR TO MONTH literal.
eg : uses NUMTOYMINTERVAL in a SUM analytic function to calculate, for each employee, the total salary of employees hired in the past one year from his or her hire date.
select last_name, hire_date, salary, sum(salary) over (order by hire_date range numtoyminterval(1, 'year') preceding) as t_sal from employees order by last_name, hire_date;

ora_dst_affected : useful when you are changing the time zone data file for your database. The function takes as an argument a datetime expression that resolves to a TIMESTAMP WITH TIME ZONE value or a VARRAY object that contains TIMESTAMP WITH TIME ZONE values.

ora_dst_convert : useful when you are changing the time zone data file for your database.

ora_dst_error : useful when you are changing the time zone data file for your database.

round(date) : returns date rounded to the unit specified by the format model fmt.
eg : rounds a date to the first day of the following year.
select round (to_date('27-OCT-00'), 'YEAR') "New Year" from dual;

sessiontimezone : returns the time zone of the current session.
eg : returns the time zone of the current session.
select sessiontimezone from dual;

sys_extract_utc : extracts the UTC (Coordinated Universal Time -- formerly Greenwich Mean Time) from a datetime value with time zone offset or time zone region name.
eg : extracts the UTC from a specified datetime.
select sys_extract_utc(timestamp '2000-03-28 11:30:00.00 -08:00') from dual;

sysdate : returns the current date and time set for the operating system on which the database server resides.
eg : returns the current operating system date and time.
select to_char(sysdate, 'MM-DD-YYYY HH24:MI:SS') 'NOW' from dual;

systimestamp : returns the system date, including fractional seconds and time zone, of the system on which the database resides. The return type is TIMESTAMP WITH TIME ZONE.
eg : returns the system timestamp
select systimestamp from dual;
eg : shows how to explicitly specify fractional seconds.
select to_char(systimestamp, 'SSSSS.FF') from dual;
eg : returns the current timestamp in a specified time zone.
select systimestamp at time zone 'UTC' from dual;

to_char(datetime) : converts a datetime or interval value of DATE, TIMESTAMP, TIMESTAMP WITH TIME ZONE, TIMESTAMP WITH LOCAL TIME ZONE, INTERVAL DAY TO SECOND, or INTERVAL YEAR TO MONTH data type to a value of varchar2 data type in the format specified by the date format fmt.
eg : converts an interval literal into a text literal.
select to_char(interval, '123-2' year(3) to month) from dual;

to_timestamp : converts char of char, varchar2, nchar, or nvarchar2 data type to a value of timestamp data type.
eg : converts a character string to a timestamp.
select to_timestamp(10-Sep-02 14:10:10.123000', 'DD-Mon-RR HH24:MI:SS.FF') from dual;

to_timestamp_tz : converts char of char, varchar2, nchar, or nvarchar2 data type to a value of TIMESTAMP WITH TIME ZONE data type.
eg : converts a character string to a value of TIMESTAMP WITH TIME ZONE.
select to_timestamp_tz('1999-12-01 11:00:00 -8:00', 'YYYY-MM-DD HH:MI:SS TZH:TZM') from dual;

to_dsinterval : converts a character string of CHAR, VARCHAR2, NCHAR, or NVARCHAR2 data type to an INTERVAL DAY TO SECOND type.
eg :
select employee_id, last_name from employees where hire_date + to_dsinterval('100 00:00:00') <= DATE '2002-11-01' order by employee_id;

to_yminterval : converts a character string of char, varchar2, nchar, or nvarchar2 data type to an interval year to month type.
eg : calculates for each employees in the sample hr.employees table a date one year two months after the hiredate.
select hire_date, hire_date + TO_YMINTERVAL('01-02') "14 months" from employees;
eg : makes the same calculation using the ISO format.
select hire_date, hire_date + TO_YMINTERVAL('P1Y2M') from employees;

trunc(date) : returns date with the time portion of the day truncated to the unit specified by the format model fmt.
eg : truncates a date
select trunc(to_date('27-OCT-92', 'DD-MON-YY'), 'YEAR') "New Year" from dual;

tz_offset : returns the time zone offset corresponding to the argument based on the date the statement is executed.
eg : returns the time zone offset of the US/Eastern time zone from UTC.
select TZ_OFFSET('US/Eastern') from dual;

General Comparison Functions : determine the greatest and or least value from a set of values.
greatest : returns the greatest of the list of one or more expressions.
eg : selects the string with the greatest value.
select greatest('HAPPY', 'HARRIOT', 'HAROLD') "Greatest" from dual;
eg :

select greatest(1, '3.925', '2.4') "Greatest" from dual;

least : returns the least of the list of one or more expressions.
eg : selects the string with the least value.
select least('HARRY', 'HARRIOT', 'HAROLD') "Least" from dual;
eg :
select least(1, '2.1' '.000832') "Least" from dual;

Conversion Functions : convert a value from one datatype to another.
asciistr : takes as its argument a string, or an expression that resolves to a string, in any character set and returns an ASCII version of the string in the database character set.
eg : returns the ASCII string equivalent of the text string "ABÄCDE".
select ASCIISTR('ABÄCDE') from dual;

bin_to_num : converts a bit vector to its equivalent number.
eg : converts a binary value to a number.
select bin_tonum(1,0,1,0) from dual;

cast : converts one built-in data type or collection-typed alue into another built-in data type or collection-typed value.
eg :
select cast('22-Oct-1997' as TIMESTAMP WITH LOCAL TIME ZONE) from dual;
select cast(to_date('22-Oct-1997', 'DD-Mon-YYYY') as TIMESTAMP WITH LOCAL TIME ZONE) from dual;
eg :
select product_id, cast(ad_sourcetext as varchar2(30)) text from print_media order by product_id;

chartorowid : converts a value from CHAR, VARCHAR2, NCHAR, or NVARCHAR2 data type to ROWID data type.
eg : converts a character rowid representation to a rowid.
select last_name from employees where rowid = chartorowid('AAAFd1AAFAAAABSAA/');

compose : takes as its argument a string, or an expression that resolves to a string, in any data type, and returns a Unicode string in the same character set as the input.
eg : returns the o-umlaut code point.
select compose('o' || unistr('\0308')) from dual;

convert : converts a character string from one character set to another.
eg : the following example illustrates character set conversion by converting a Latin-1 string to ASCII.
select convert('Ä Ê Í Õ Ø A B C D E ', 'US7ASCII', 'WE8ISO8859P1') from dual;

decompose : valid only for Unicode characters.
eg : decompress the string "Châteaux" into its component code points.
select decompose (' Châteaux') from dual;

hextoraw : converts char containing hexadecimal digits in the achar, varchar2, nchar, or nvarchar2 data type to a raw value.
eg : creates a simple table with a raw column, and inserts a hexadecimal value that has been converted to RAW
create table test (raw_col RAW(10));
insert into test values (HEXTORAW('7D'));
eg : converts hexadecimal digits to a raw value and casts the raw value to VARCHAR2.
select utl_raw.cast_to_varchar2(hextoraw('4041424344')) from dual;

numtodsinterval : converts n to an INTERVAL DAY TO SECOND literal.
eg : uses NUMTODSINTERVAL in a COUNT analytic function to calculate, for each employee, the number of employees hired by the same manager within the past 100 days from his or her hire date.
select manager_id, last_name, hire_date, count(*) over (partition by manager_id order by hire_date range numtodsinterval(100, 'day') preceding) as t_count from employees order by last_name, hire_date;

numtoyminterval : converts number n to an INTERVAL YEAR TO MONTH literal.
eg : uses NUMTOYMINTERVAL in a SUM analytic function to calculate, for each employee, the total salary of employees hired in the past one year from his or her hire date.

select last_name, hire_date, salary, sum(salary) over (order by hire_date range numtoyminterval(1, 'year') preceding) as t_sal from employees order by last_name, hire_date;

rawtohex : converts raw to a character value containing its hexadecimal representation.
eg : returns the hexadecimal equivalent of a RAW column value.
select rawtohex(raw_column) "Graphics" from graphics;

rawtonhex : converts raw to a character value containing its hexadecimal representation.
eg : returns the hexadecimal equivalent of a RAW column value.
select rawtonhex(raw_column), dum(rawtonhex(raw_column)) "DUMP" from graphics;

rowidtochar : converts a rowid value to varchar2 data type. The result of this conversion is always 18 characters long.
eg :
converts a rowid value in the employees table to a charactervalue.
select rowid from employees where rowidtochar(rowid) like '%JAAB%' order by rowid;

rowidtonchar : converts a rowid value to nvarchar2 data type.
eg :
select length(rowidtochar(rowid)) length, rowidtochar(rowid) from employees order by length;

scn_to_timestamp : takes as an argument a number that evaluates to a system change number(SCN), and returns the approximate timestamp associated with that SCN. This function is useful any time you want to know the timestamp associated with an SCN.
eg : uses the ORA_ROWSCN pseudocolumn to determine the system change number of the last update to a row and uses SCN_TO_TIMESTAMP to convert that SCN to a timestamp.
select SCN_TO_TIMESTAMP(ORA_ROWSCN) from employees where employee_id = 188;
eg : to convert a system change number to a timestamp for use in an Oracle Flashback Query.
select SCN_TO_TIMESTAMP(ORA_ROWSCN) from employees where employee_id = 188;

timestamp_to_scn : takes as an argument a timestamp value and returns the approximate system change number (SCN) associated with that timestamp.
eg :
select timestamp_to_scn(order_date) from orders where order_id = 5000;

to_binary_double : returns a double-precision floating-point number.
eg : converts a value of data type NUMBER to a value of data type BINARY_DOUBLE.
select dec_num, to_binary_double(dec_num) from float_point_demo;
eg : compares extracted dump information from the dec_num and bin_double columns.
select dump(dec_num) "Decimal", dump(bin_double) "Double" from float_point_demo;

to_binary_float : returns a single-precision floating-point number.
eg : converts a value of data type number to a value of data type binary_float;
select dec_num, to_binary_float(dec_num) from float_point_demo;

to_blob : converts long raw and raw values to blob values.
eg : returns the blob of a raw column value.
select to_blob(raw_column) blob from raw_table;

to_char(character) : converts nchar, nvarchar2, clob, or nclob data to the database character set. The value returned is always varchar2.
eg : interprets a simple string as character data.
select to_char('01110') from dual;
eg : 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(datetime) : converts a datetime or interval value of date, timestamp, timestamp with time zone, timestamp with local time zone, interval day to second, or interval year to month data type to a value of varchar2 data type in the format specified by the date format fmt. If you omit fmt, then date is converted to a varchar2 value as follows.
eg :

select sessiontimezone, to_char(tsltz_col, 'DD-MON-YYYY HH24:MI:SSxFF') as tsltz from date_tab order by sessiontimezone, tsltz;
eg : converts an interval literal into a text literal.
select to_char(interval '123-2' YEAR(3) to month) from dual;

to_char(number) : converts n to a value of varchar2 data type, using the optinal number format fmt.
eg : uses implicit conversion to combine a string and a number into a number.
select to_char('01110' + 1) from dual;

to_clob : converts nclob values in a clob column or other character string to clob values.
eg : converts nclob data from the sample pm.print_media table to clob and inserts it into a clob column, replacing existing data in that column.
update print_media set ad_finaltext = to_clob(ad_fltextn);

to_date : converts char of char, varchar2, nchar, or nvarchar2 data type to a value of date data type.
eg : converts a character string into a date/
select to_date('January 15, 1989, 11:00 A.M.',
'Month dd, YYYY, HH:MI A.M.',
'NLS_DATE_LANGUAGE = Americal') from dual;

to_dsinterval converts a character string of char, varchar2, nchar, or nvarchar2 data type to an INTERVAL DAY TO SECOND type.
eg : select from the hr.employees table the employees who had worked for the company for at least 100 days on November 1, 2002.
select employee_id, last_name from employees where hire_date + to_dsinterval('100 00:00:00') <= date '2002-11-01' order by employee_id;
eg : uses the ISO format to display the timestamp 100 days and 5 hours after the beginning of the year 2009.
select to_char(timestamp '2009-01-01 00:00:00' + to_dsinterval(P100DT05H'), 'YYYY-MM-DD HH24:MI:SS') "Time Stamp" from dual;

to_lob : converts long or long raw values in the column long_column to lob values. You can apply this function only to a long or long raw column, and only in the select list of a subquery in an insert statement.
eg : shows how to use the to_lob function on your long data in a hypothetical table old_table.
create table new_table(col1, col2, ... lob_col clob);
insert into new_table (select o.col1, o.col2, ... to_lob(o.old_long_col) from old_table o;

to_multi_byte : returns char with all of its single-byte characters converted to their correspoiding multibyte characters.
eg : converting from a single byte A to a multibyte A in UTF8.
select dump(to_multi_byte('A')) from dual;

to_nchar(character) : converts a character string, char, varchar2, clob, or nclob value to the national character set.
eg : converts varchar2 data from the oe.customers table to the national character set.
select to_nchar(cust_last_name) from customers where customer_id = 103;

to_nchar(datetime) : converts a datetime or interval value of date, timestamp, timestamp with time zone, timestamp with local time zone, interval month to year, or interval day to second data type from the database character set to the national character set.
eg : converts the order_date of all orders whose satus is 9 to the national character set.
select to_char(order_date) as order_date from orders where order_status > 9 order by order_date;

to_nchar(number) : converts n to a string in the national character set. The value n can be of type number, binary_float, or binary_double. The function returns a value of the same type as the argument.
eg : converts the customer_id value from the sample table oe.orders to the national character set.
select to_char(customer_id) "NCHAR_Customer_ID" from orders where order_status > 9 order by "NCHAR_Customer_ID";

to_nclob : converts clob values in a LOB column or other character strings to NCLOB values.
eg : inserts some character data into an NCLOB column of the pm.print_media table by first converting the data with the to_nclob function.
insert into print_media(product_id, ad_id, ad_fltextn) values (3502, 31001, to_nclob('Placeholder for new product description'));

to_number : converts expr to a value of number data type.
eg : convert character string data into a number.

update employees set salary = salary + to_number('100.00', '9G999D99') where last_name = 'Perkins';

to_dsinterval
to_single_byte : returns char with all of its multibyte characters converted to their corresponding single-byte characters.
eg : go from a multibyte A in UTF8 to a single byte ASCII A.
select to_single_byte(chr(15711393)) from dual;

to_timestamp : converts char or char, varchar2, nchar, or nvarchar2 data type to a value of timestamp data type.
eg : converts a character string to a timestamp.
select to_timestamp('10-Sep-02 14:10:10.12300', 'DD-Mon-RR HH24:MI:SS.FF') from dual;

+86-15540808900
to_timestamp_tz : converts char of char, varchar2, nchar, or nvarchar2 data type to a value of timestamp with time zone data type.
eg : converts a character string to a value of TIMESTAMP WITH TIME ZONE.
select to_timestamp_tz('1999-12-01 11:00:00 -8:00', 'YYYY-MM-DD HH:MI:SS TZH:TZM') from dual;
eg : cast a null column in a union operation as timestamp with local time zone using the sample tables oe.order_items and oe.orders.
select order_id, line_item_id, cast(null as timestamp with local time zone) order_date from order_items union select order_id, to_number(null), order_date from orders;

to_yminterval : converts a character string of char, varchar2, nchar, or nvarchar2 data type to an interval year to month type.
eg : calculates for each employee in the sample hr.employees table a date one year two months after the hire date.
select hire_date, hire_date + to_yminterval('01-02') "14months" from employees;

treat : changes the declared type of an expression.
eg : retrieves the salary attribute of all people in the persons table, the value being null for instances of people that are not employees.
select name, treat(value(p) as employee_t).salary salary from persons p;

translate ... using
unistr : takes as its argument a text literal or an expression that resolves to character data and returns it in the national character set.
eg : passes both ASCII characters and unicode encoding values to the unistr function, which returns the string in the national character set.
select unistr('abc\00e5\00f1\00f6') from dual;

Large Object Functions : operate on LOBs.
bfilename : returns a bfile locator that is associated with a physical LOB binary file on the server file system.
eg : inserts a row into the sample table pm.print_media. The example uses the BFILENAME function to identify a binary file on the server file system in the directory /demo/schema/product_media. The example shows how the directory database object media_dir was created in the pm schema.
create directory media_dir as '/demo/schema/product_media';
insert into print_media(product_id, ad_id, ad_graphic) values (3000, 31001, bfilename('media_dir', 'modem_comp_ad.gif'));

empty_blob, empty_clob : return an empty LOB locator that can be used to initialize a LOB variable or, in an insert or update statement, to initialize a LOB column or attribute to empty.empty means that the LOB is initialized, but not populated with data.
eg : initializes the ad_photo column of the sample pm.print_media table to empty.
update print_media set ad_photo = empty_blob();

Collection Functions : operate on nested tables and varrys.
cardinality : returns the number of elements in a nested table. The return type is number. If the nested table is empty, or is a null collection, then CARDINALITY returns NULL.
eg : shows the number of elements in the nested table column ad_textdocs_ntab of the sample table pm.print_media.
select product_id, cardinality(ad_textdocs_ntab) cardinality from print_media order by product_id;

collect : an aggregate function that takes as its argument a column of any type and creates a nested table of the input type out of the rows selected. To get accurate results from this function you must use it within a cast function.

eg : creates a nested table from the varray column of phone numbers in the sample table oe.customers.
create type phone_book as table of phone_list_typ;
select cast(collect(phone_number) as phone_book_t) "Income Level L Phone Book" from customers where income_level = 'L: 300000 and above';
eg : creates a nested table from the column of warehouse names in the sample table oe.warehouses. It uses order by to order the warehouse names.
create type warehouse_name_t as table of varchar2(35);
select cast(collect(warehouse_name order by warehouse_name) as warehouse_name_t) "Warehouses" from warehouses;

powermultiset : takes as input a nested table and returns a nested table of nested tables containing all nonempty subsets (called submultisets) of the input nested table.
eg :
create type cust_address_tab_tab_typ as table of cust_address_tab_typ;
select the nested table column cust_address_ntab from the customers_demo table using the POWERMULTISET function.
select cast(powermultiset(cust_address_ntab) as cust_address_tab_tab_typ) frm customers_demo;

powermultiset_by_cardinality : takes as input a nested table and a cardinality and returns a nested table of nested tables containing all nonempty subsets (called submultisets) of the nested table of the specified cardinality.
eg :
First, create a data type that is a nested table of the cust_address_tab_type data type.
create type cust_address_tab_tab_typ as table of cust_address_tab_typ;
Next, duplicate the elements in all the nested table rows to increase the cardinality of the nested table rows to 2.
update customers_demo set cust_address_ntab = cust_address_ntab multiset union cust_address_ntab;
Now, select the nested table column cust_address_ntab from the customers_demo table using the powermultiset_by_cardinality function.
select cast(powermultiset_by_cardinality(cust_address_ntab, 2) as cust_address_tab_tab_typ) from customers_demo;

set : converts a nested table into a set by eliminating duplicates. The function returns a nested table whose elements are distinct from one another.
eg : selects from the customers_demo table the unique elements of the cust_address_ntab nested table column.
select customer_id, set(cust_address_ntab) address from customers_demo order by customer_id;

Hierarchical Function : applies hierarchical path information to a result set.
sys_connect_by_path : is valid only in hierarchical queries. It returns the path of a column value from root to node, with column values separated by char for each row returned by connect by condition.
eg : returns the path of employee names from employee Kochhar to all employees of Kochhar (and their employees).
select lpad(' ', 2*level-1) || sys_connect_by_path(last_name, '/') "Path" from employees start with last_name = 'Kochhar' connect by prior employee_id = manager_id;

Data Mining Functions : operate on models that have been built using the DBMS_DATA_MINING package or the Oracle Data Mining Java API.
cluster_id : for use with clustering models created by the DBMS_DATA_MINING package or with Oracle Data Miner. It returns the cluster identifier of the predicted cluster with the highest probability for the set of predictors specified in the mining_attribute_clause.
eg : lists the clusters into which customers of a given dataset have been grouped.
This example, and the prerequisite data mining operations, including the creation of the km_sh_clus_sample model and the mining_data_apply_v view, can be found in the demo file $ORACLE_HOME/rdbms/demo/dmkmdemo.sql
eg :
select cluster_id(km_sh_clus_sample using *) as clus, count(*) as cnt from mining_data_apply_v group by cluster_id(km_sh_clus_sample using *) order by cnt desc;

cluster_probability : It returns a measure of the degree of confidence of membership of an input row in a cluster associated with the specified model.
eg : determines the ten most representative customers, based on likelihood, in cluster 2.
select * from (select cust_id, cluster_probability(km_sh_clus_sample, 2 using *) prob from miming_data_apply_v order by prob desc) where rownum < 11;

cluster_set : returns a varray of objects containing all possible clusters that a given row belongs to.

eg : lists the most relevant attributes of each cluster to which customer 101362 belongs with > 20% likelihood.

```
with
clus_tab as (
select id, a.attribute_name aname, a.conditional_operator op, nvl(a.attribute_str_value, round(a.attribute_num_value), 4)) val,
a.attribute_support support, a.attribute_confidence confidence from
table(dbms_data_mining.get_model_details_km('km_sh_clus_sample')) t, table(t.rule.antecedent) a where a.attribute_confidencd >
0.55),
clust as (
select id, cast(collect(cattr(aname, op, to_char(val), support, confidence)) as Cattrs) cl_attrs from clus_tab group by id),
custclus as (
select t.cust_id, s.cluster_id, s.probability from (select cust_id, cluster_set(km_sh_clus_sample, null, 0.2 using *) pset from
mining_data_apply_v where cust_id = 101362) t, table(t.pset) s)
select a.probability prob, a.cluster_id cl_id, b.attr, b.op, b.val, b.supp, b.conf from custclus a, (select t.id, c.* from clust
T, table (t.cl_attrs) c) b where a.cluster_id = b.id order by prob desc, cl_id asc, conf desc, attr asc, val asc;
```

feature_id : returns an Oracle NUMBER that is the identifier of the feature with the highest value in the row.

eg : lists the features and corresponding count of customers in a dataset.

```
select feature_id(nmf_sh_sample using *) as feat, count(*) as cnt from nmf_sh_sample_apply_prepared group by
feature_id(nmf_sh_sample using *) order by cnt desc, feat desc;
```

feature_set : returns a varray of objects containing all possible features.

eg : list the top features corresponding to a given customer record (based on match quality), and determines the top attributes for each feature (based on coefficient > 0.25).

```
with
feat_tab as (
select f.feature_id fid,
    a.attribute_name attr,
    to_char(a.attribute_value) val,
    a.coefficient coeff
    from table (dbms_data_mining.get_model_details_nmf('nmf_sh_sample')) f,
    table (f.attribute_set) a
     where a.coefficient > 0.25
),
feat as (
select fid,
    cast(collect(featattr(attr, val, coeff)) as featattrs) f_attrs
     from feat_tab group by fid),
cust_10_features as (
select t.cust_id, s.feature_id, s.value
     from (select cust_id, feature_set(nmf_sh_sample, 10 using *) pset
    from nmf_sh_sample_apply_prepared
    where cust_id = 100002) t, table(t.pset) s
)
select a.value, a.feature_id fid, b.attr, b.val, b.coeff
    from cust_10_features a, (select t.fid, f.* from feat t, table(t.f_attrs) f) b
    where a.feature_id = b.fid
order by a.value desc, a.feature_id asc, coeff desc, attr asc, val asc;
```

feature_value : returns the value of a given feature.

eg : lists the customers that correspond to feature 3, ordered by match quality.

```
select * from (
select cust_id, feature_value(nmf_sh_sample, 3 using *) match_quality from nmf_sh_sample_apply_prepared order by match_quality
desc)
where rownum < 11;
```

prediction : returns the best prediction for the model.

eg : return by gender the average age of customers who are likely to use an affinity card.

```
select cust_gender, count(*) as cnt, round(avg(age)) as avg_age from mining_data_apply_v where prediction(dt_sh_class_sample cost
```

model using cust_marital_status, education, household_size) = 1 group by cust_gender order by cust_gender;

prediction_bounds : returns an object with two number fields lower and upper. For a regression mining function, the bounds apply to value of the prediction. For a classification mining function, the bounds apply to the probability value.
eg : returns the distribution of customers whose ages are predicted to be between 25 and 45 years with 98% confidence.
select count(cust_id) cust_count, cust_marital_status from (select cust_id, cust_marital_status from mining_data_apply_v where prediction_bounds(glmr_sh_regr_sample, 0.98 using *).lower > 24 and prediction_bounds(glmr_sh_regr_sample, 0.98 using *).upper < 46) group by cust_marital_status;

prediction_cost : returns a measure of cost for a given predicton as an Oracle NUMBER.
eg : finds the ten customers living in Italy who are least expesive to convince to use an affinity card.
```
with
cust_italy as (
select cust_id from mining_data_apply_v
  where country_name = 'Italy'
order by prediction_cost(DT_SH_Class_sample, 1 cost model using *) asc, 1
)
select cust_id from cust_italy where rownum < 11;
```

prediction_details : returns an XML string containing model-specific information related to the scoring of the input row.
eg : uses all attributes from the mining_data_apply_v view that are relevant predictors for the DT_SH_Class_sample decision tree model. For customers who work in technical support and are under age 25, it returns the tree node that results from scoring those records with the DT_SH_Class_sample model.
select cust_id, education, prediction_details(DT_SH_Class_sample using *) treenode from mining_data_apply_v where occupation = 'TechSup' and age < 25 order by cust_id;

prediction_probability : returns the probability for a given prediction as an Oracle NUMBER.
eg : returns the 10 customers living in Italy who are most likely to use an affinity card.
```
select cust_id from (
    select cust_id from mining_data_apply_v
    where country_name = 'Italy'
    order by prediction_probality(DT_SH_Clas_sample, 1 using *) desc, cust_id)
where rownum < 11;
```

prediction_set : returns a varray of objects containing all classes in a multiclass classification scenario.
eg : lists for ten customers, the likelihood and cost of using or rejecting an affinity card. This example has a binary target, but such a query is also useful in multiclass classification such as Low, Med, and High.
```
select t.cust_id, s.prediction, d.probability, s.cost from
    (select cust_id, prediction_set(dt_sh_clas_sample cost model using *) pset
    from ming_data_apply_v where cust_id < 100011) T,
    table (T.pset) s
order by cust_id, s.prediction;
```

XML Functions : operate on or return XML documents or fragments. These functions use arguments that are not defined as part of the ANSI/ISO/IEC SQL Standard but are defined as part of the World Wide Web Consortium (W3C) standards. The processing and operations that the functions perform are defined by the relevant W3C standards.

appendchildxml
deletexml
depth
extract(xml)
existsnode
extractvalue
insertchildxml
insertxmlbefore
path
sys_dburigen
sys_xmlagg

sys_xmlgen
updatexml
xmlagg
xmlcdata
xmlcolattval
xmlcomment
xmlconcat
xmlforset
xmlparse
xmlpi
xmlquery
xmlroot
xmlsequence
xmlserialize
xmltable
xmltransform

Encoding and Decoding Functions : let you inspect and decode data in the database.
decode : compares expr to each search value one by one. If expr is equal to a search, then Oracle Database returns the corresponding result. If no match is found, then Oracle returns default. If default is omitted, then Oracle returns null.
eg : decodes the value warehouse_id. If warehouse_id is 1, then the function returns 'Southlake'; if warehouse_id is 2, then it returns 'San Francisco'; and so forth. If warehouse_id is not 1, 2, 3, or 4, then the function returns 'Non domestic'.
select product_id, decode(warehouse_id, 1, 'Southlake', 2, 'San Francisco', 3, 'New Jersey', 4, 'Seattle', 'Non domestic') "Location" from inventories where product_id < 1775 order by product_id, "Location";

dump : returns a varchar2 value containing the data type code, length in bytes, and internal representation of expr.
eg : extract dump information from a string expression and a column.
select dump('abc', 1016) from dual;
select dump(last_name, 8, 3, 2) "OCTAL" from employees where last_name = 'Hunold' order by employee_id;
select dump(last_name, 10, 3, 2) "ASCII" from employees where last_name = 'Hunold' order by employee_id;

ora_hash : computes a hash value for a given expression. useful for operations such as analyzing a subset of data and generating a random sample.
eg : creates a hash value for each combination of customer ID and product ID in the sh.sales table, divides the hash values into a maximum of 100 buckets, and returns the sum of the amount_sold values in the first_bucket(bucket 0). The third argument (5) provides a seed value for the hash function. You can obtain different hash results for the same query by changing the seed value.
select sum(amount_sold) from sales where ora_hash(concat(cust_id, prod_id), 99, 5) = 0;

vsize : returns the number of bytes in the internal representation of expr. If expr is null, then this function returns null.
eg : returns the number of bytes in the last_name column of the employees in department 10.
select last_name, vsize(last_name) "BYTES" from employees where department_id = 10 order by employee_id;

NULL-Related Functions : facilitate null handling.
coalesce : returns the first non-null expr in the expression list. You must specify at least two expressions. If all occurrences of expr evaluate to null, then the function returns null.
eg : use coalesce as a variety of the case expression.
coalesce(expr1, expr2)
is equivalent to :
case when expr1 is not null then expr1 else expr2 end
eg :
coalesce(expr1, expr2, ..., exprn)
when n >= 3, is equivalent to :
case when expr1 is not null then expr1 else coalesce(expr2, ..., exprn) end
eg : uses the sample oe.product_information table to organize a clearance sale of products. It gives a 10% discount to all products with a list price. If there is no list price, then the sale price is the minimum price. If there is no minimum price, then the sale price is "5".
select product_id, list_price, min_price, coalesce(0.9 * list_price, min_price, 5) "Sale" from product_information where supplier_id = 102050 order by product_id;

lnnvl : provides a concise way to evaluate a condition when one or both operands of condition may be null.
eg : returns only employees who actually receive a commission of less than 20%.
select count(*) from employees where commission_pct < .2;
eg : select count(*) from employees where lnnvl(commission_pct >= .2);

nanvl : useful only for floating-point numbers of type binary_float or binary_double. It returns an alternative value n1 if the input value n2 is NaN (not a number). If n2 is not NaN, then Oracle returns n2.
eg :
insert into float_point_demo values (0, 'NaN', 'NaN');
select * from float_point_demo;
select bin_float, NANVL(bin_float, 0) from float_point_demo;

nullif : compares expr1 and expr2. If they are equal, then the function returns null. If they are not equal, then the function returns expr1.
The NULLIF function is logically equivalent to the following case expression :
case when expr1 = expr2 then null else expr1 end
eg : 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 order by last_name, "Old Job ID";

nvl : lets ou replace null (returned as a blank) with a string in the results of a query. If expr1 is null, then NVL returns expr2. If expr1 is not null, then NVL reutrns expr1.
eg : returns a list of employee names and commissions, substituting "Not Applicable" if the employee receives no commission.
select last_name, NVL(to_char(commission_pct), 'Not Applicable') commission from employees where last_name like 'B%' order by last_name;

nvl2 : lets you determine the value returned by a query based on whether a specified expression is null or not null. If expr1 is not null, then nvl2 returns expr2. If expr1 is null, then NVL2 returns expr3.
eg : 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%' order by last_name;

Environment and Identifier Functions : provide information about the instance and session.
sys_context : returns the value of parameter associated with the context namespace at the current instant.
eg : returns the name of the user who logged onto the database.
connect oe/password
select sys_context('USERENV','SESSION_USER') from dual;
select role from session_roles;
select sys_context('SYS_SESSION_ROLES','RESOURCE') from dual;
select sys_context('SYS_SESSION_ROLES','DBA') from dual;

sys_guid : generates and returns a globally unique identifier (RAW value) made up of 16 bytes.
eg : adds a column to the sample table hr.locations, inserts unique identifiers into each row, and returns the 32-character hexadecimal representation of the 16-byte RAW value of the global unique identifier.
alter table locations add (uid_col RAW(16));
update locations set uid_col = SYS_GUID();
select location_id, uid_col from locations order by location_id, uid_col;

sys_typeid : returns the typeid of the most specific type of the operand. This value is used primarily to identify the type-discriminant column underlying a substitutable column.
eg : returns the most specific types of the object instances stored in the persons table.
select name, sys_typeid(value(p)) "Type_id" from persons p;
eg : returns the most specific types of authors stored in the table books.
select b.title, b.author.name, sys_typeid(author) "Type_ID" from books b;

uid : returns an integer that uniquely identifies the session user (the user who logged on).
eg : returns the UID of the current user.
select uid from dual;

user : returns the name of the session user (the user who logged on) with the data type VARCHAR2.
eg : returns the current user and the user's UID.
select user, uid from dual;

userenv : returns information about the current session. This information can be useful for writing an application-specific audit_trail table or for determining the language-specific characters currently used by your session.
eg : returns the LANGUAGE parameter of the current session.
select userenv('LANGUAGE') "Language" FROM dual;

Aggregate Functions : return a single result row based on groups of rows, rather than on single rows.
eg : calculates the average of the maximum salaries of all the departments in the sample schema hr.
select avg(max(salary)) from employees group by department_id;
avg : returns average value of expr.
eg : calculates the average salary of all employees in the hr.employees table.
select avg(salary) "Average" from employees;

collect : an aggregate function that takes as its argument a column of any type and creates a nested table of the input type out of the rows selected. To get accurate results from this function you must use it within a cast function.
eg : creates a nested table from the varray column of phone numbers in the sample table oe.customers. The nested table includes only the phone numbers of customers with an income level of L:300000 and above.
create type phone_book_t as table of phone_list_typ;
select cast(collect(phone_numbers) as phone_book_t) "Income Level L Phone Book" from customers where income_level = 'L:300000 and above';

corr ： returns the coefficient of correlation of a set of number pairs. (返回一对数字的相关系数)
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 :
covar_pop(expr1, expr2) / (stddev_pop(expr1) * stddev_pop(expr2))

corr_* functions are : corr_s and corr_k
The CORR* functions support nonparametric or rank correlation. They let you find correlations between expressions that are ordinal scaled (where ranking of the values is possible). Correlation coefficients take on a value ranging from -1 to 1, where 1 indicates a perfect relationship, -1 a perfect inverse relation (when one variable increase as the other decrease), and a value close to 0 means no relationship.
CORR_* return values
coefficient : coefficient of correlation
one_sided_sig : positive one-tailed significance of the correlation
one_sided_sig_pos : same as one_sided_sig
one_sided_sig_neg : negative one-tailed significance of the correlation
two_sided_sig : two-tailed significance of the correlation
corr_s : calculates the Spearman's rho correlation coefficient.
eg : derives a coefficient of correlation for each of two different comparisons -- salary and commission_pct, and salary and employee_id.
select count(*) count, corr_s(salary, commission_pct) commission, corr_s(salary, employee_id) empid from employees;
corr_k : calculates the Kendall's tau-b correlation coefficient.
eg : determines whether a correlation exists between an employee's salary and commission percent.
select corr_k(salary, commission_pct, 'COEFFICIENT') coefficient, corr_k(salary, commission_pct, 'TWO_SIDED_SIG') two_sided_p_value from employees;

count : returns the number of rows returned by the query.

covar_pop : returns the population covariance of a set of number pairs.
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 :

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

covar_samp : returns the sample covariance（样本协方差） of a set of number pairs.
Oracle database applies the function t o 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)

cume_dist : calcumates the cumulative distribution（累积分布） of a value in a group of values. The range of values returned by cume_dist is > 0 to <= 1.
eg : calcumates the cumulative distribution of ahypthetical employee with a salary of $15500 and commission rate of 5% among the employees in the sample table oe.employees.
select cume_dist(15500, .05) within group (order by salary, commission_pct) "Cume_Dist of 15500" from employees;

dense_rank : computes the rank of a row in an ordered group of rows and returns the rank as a NUMBER.
eg : computes the ranking of a hypothetical employee with the salary $15500 and a commission of 5% in the sample table oe.employees.
select dense_rank(15500, .05) within group (order by salary desc, commission_pct) "Dense Rank" from employees;

first : 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, then the aggregate operates on the set with only one element.
eg : 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.
```
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 order by department_id;
```

group_id : distinguishes duplicate groups resulting from a group by specification. It is useful in filtering out duplicate grouping from the query result. It returns an Oracle NUMBER to uniquely identify duplicate groups. This function is applicable only ina select statement that contains a group by clause.
eg : assigns the value 1 to the duplicate co.country_region grouping from a query on the sample tables sh.countries and sh.sales.
```
select co.country_region, co.country_subregion, sum(s.amount_sold) "Revenue", 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 grouping sets ((co.country_region, co.country_subregion),
        (co.country_region, co.country_subregion))
    order by co.country_region, co.country_subregion, "Revenue", g;
```
To ensure that only rows with GROUP_ID < 1 are returned, add the following HAVING clause to the end of the statement :
```
having group_id() < 1
```

grouping : distinguishes superaggregate（超级聚合） rows from regular grouped rows. GROUP BY extensions such as ROLLUP and CUBE produce superaggregate rows where the set of all values is represented by null. Using the GROUPING function, you can distinguish a null representing the set of all values in a superaggregate row from a null in a regular row.
eg : uses the sample tables hr.departments and hr.employees, if the GROUPING function returns 1 (indicating a superaggregate row rather than a regular row from the table), then the string "All Jobs" appears in the "JOB" column instead of the null that would otherwise appear.
```
select decode(grouping(department_name), 1, "ALL DEPARTMENTS', department_name) as department, decode(grouping(job_id), 1, 'All Jobs', job_id) as job, count(*) "Total Empl", AVG(salary) * 12 "Average Sal" from employees e, departments d where d.department_id = e.department_id group by rollup (department_name, job_id) order by department, job;
```

grouping_id : returns a number corresponding to the GROUPING bit vector associated with a row. GROUPING_ID is applicable only in a select statement that contains a GROUP BY extension, such as ROLLUP or CUBE, and a GROUPING function. In queries with many group by expressions, determining the group by level of a particular row requires many grouping functions, which leads to cumbersome SQL. GROUPING_ID is useful in these cases.
eg : extract grouping IDs from a query of the sample table sh.sales.
```
select channel_id, promo_id, sum(amount_sold) s_sales,
```

```
    grouping(channel_id) gc,
    grouping(promo_id) gp,
    grouping_id(channel_id, promo_id) gcp,
    grouping_id(promo_id, channel_id) gpc
    from sales
    where promo_id > 496
    group by cube(channel_id, promo_id)
    group by channel_id, promo_id, s_sales, gc;
```

last : refer to FIRST for usage.

listagg : for a specified measure, LISTAGG orders data within each group specified in the order by clause and then concatenates the values of the measure column.
As a single-set aggregate function, LISTAGG operates on all rows and returns a single output row.
As a group-set aggregate, the function operates on and returns an output row for each group defined by the GROUP BY clause.
As an analytic function, LISTAGG partitions the query result set into groups based on one or more expression in the query_partition_clause.
eg : lists all of the employees in Department 30 in the hr.employees table, ordered by hire date and last name.

```
select listagg(last_name, ':') within group (order by hire_date, last_name) "Emp_list",
    min(hire_date) "Earliest"
    from employees where department_id = 30;
```

eg : group-set aggregate example, lists for each department ID in the hr.employees table, the employees in that department in order of their hire date.

```
select department_id "Dept.", listagg(last_name, ';') within group (order by hire_date) "Employees"
    from employees group by department_id order by department_id;
```

max : returns maximum value of expr.
eg : determines the highest salary in the hr.employees table.

```
select max(salary) "Maximum" from employees;
```

median : （中位数）an inverse distribution function that assumes a continuous distribution model. It takes a numeric or datetime value and returns the middle value or an interpolated value that would be the middle value once the value are sorted. Nullsa are ignored in the calculation.
eg : the following query returns the median salary for each department in the hr.employees table.

```
select department_id, median(salary) from employees group by department_id order by department_id;
```

min : returns minimum value of expr.
eg : returns ther earliest hire date in the hr.employees table.

```
select min(hire_date) "Earliest" from employees;
```

percentile_count : 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.
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.
As an analytic function, fow 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).
eg :
calculates the percent rank of a hypothetical employee in the sample table hr.employees with a salary of $15500 and a commission of 5%.

```
select percent_rank(15000, .05) within group (order by salary, commission_pct) "Percent-Rank" from employees;
```

percentile_count : 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.
eg : computes the median salary in each department.

```
select department_id,
    percentile_count(0.5) within group (order by salary desc) "Median count",
```

```
    percentile_disc(0.5) within group (order by salary desc) "Median disc"
    from employees
    group by department_id
    order by department_id;
```

percentile_disc : 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.

percent_rank
rank : calculates the rank of a value in a group of values. The return type is NUMBER.
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 argumants 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.
eg : calculates the rank of a hypothetical employee in the sample table hr.employees with a salary of $15500 and a commission of 5%.
select rank(15500, .05) within group (order by salary, commission_pct) "Rank" from employees;
eg : returns the rank for a $15500 salary among the employee salaries.
select rank(15500) within group (order by salary desc) "Rank of 15500" from employees;

regr_(Linear Regression) Functions（线性回归函数）: The linear regression function fit an ordinary-least-squares regression line（普通最小二乘回归线） to a set of number pairs.
eg : regr_* (expr1, expr2) over (analytic_clause)
note : expr1 is interpreted as a value of the dependent variable（应变量） (a y value), and expr2 is interpreted as a value of the independent variable（自变量） (an x value).
regr_slope : returns the slope of the line. It makes the following computation :
covar_pop(expr1, expr2) / var_pop(expr2)

regr_intercept : returns the y-intercept（截距） of the regression line. It makes the following computation :
avg(expr1) - regr_slop(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_r2 : returns the coefficient of determination（可决系数，亦称测定系数、决定系数、可决指数。） (also called R-squared or goodness of fit) for the regression. The return values are :
null if var_pop(expr2) = 0
1 if var_pop(expr2) = 0 and var_pop(expr2) != 0
power(corr(expr1, expr2) if var_pop(expr1) > 0 and var_pop(expr2) != 0

regr_avgx : evaluates the average of the independent variable (expr2) of the regression line. It makes the following computation :
avt(expr2)

regr_avgy : evaluates the average of the dependent variable (expr1) of the regression line. It makes the following computation :
avg(expr1)

regr_sxx : makes the following computation :
regr_count(expr1, expr2) * var_pop(expr2)

regr_syy : makes the following computation :
regr_count(expr1, expr2) * var_pop(expr1)

regr_sxy : makes the following computation :
regr_count(expr1, expr2) * covar_pop(expr1, expr2)
eg : The following example provides a comparison of the various linear regression functions used in their analytic form.
select job_id, employee_id, salary,

```
    regr_slope(sysdate-hire_date, salary) over (partition by job_id) slope,
    regr_intercept(sysdat-hire_date, salary) over (partition by job_id) intcpt,
    regr_r2(sysdate-hire_date, salary) over (partition by job_id) rsqr,
    regr_count(sysdate-hiredate, salary) over (partition by job_id) count,
    regr_avgx(sysdate-hiredate, salary) over (partition by job_id) afgx,
    regr_avgy(sysdate-hiredate, salary) over (partition by job_id) avgy
    from employees
    where department_id in (50,80)
    order by job_id, employee_id;
```

eg : 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.

```
select job_id,
    regr_slope(sysdate - hire_date, salary) slope,
    regr_intercept(sysdate - hire_dater, salary) intercept
    from employees
    where department_id in (50, 80)
    group by job_id
    order by job_id;
```

eg : 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.

```
select job_id, regr_count(sysdate - hire_date, salary) count
    from employees
    where department_id in (30, 50)
    group by job_id
    order by job_id, count;
```

eg : calculates the coefficient of determination the linear regression of time employed (sysdate - hire_date) and salary using the sample talbe hr.employees.

```
select job_id,
    regr_r2(sysdate - hire_date, salary) regr_r2
    from employees
    where department_id in (80, 50)
    group by job_id
    order by job_id, regr_r2;
```

eg : calculates the average value for time employed (sysdate - hire_date) and salary using the sample table hr.employees. Results are grouped by job_id.

```
select job_id,
    regr_avgy(sysdate - hire_date, salary) avgy,
    regr_avfx(sysdate - hire_date, salary) avgx
    from employees
    where department_id in (30, 50)
    group by job_id,
    order by job_id, avgy, avgx;
```

eg : calculates three types of diagnostic statistics for the linear retgression of time employed (sysdate - hire_date) and salary using the sample table hr.employees.

```
select job_id,
    regr_sxy(sysdate - hire_date, salary) regr_sxy,
    regr_sxx(sysdate - hire_date, salary) regr_sxx,
    regr_xyy(sysdate - hire_date, salary) regr_syy
    from employees
    where department_id in (80, 50)
    group by job_id
    order by job_id;
```

stats_binomial_test : an exact probability test used for dichotomous variables, where only two possible values exist. It tests the difference between a sample proportion and a given proportion. The sample size in such tests is usually small.

eg : determines the probability that reality exactly matches the number of men observed under the assumption that 69% of the population is composed of men.

```
select avg(decode(cust_gender, 'M', 1, 0)) real_proportion, stats_binomial_test(cust_gender, 'M', 0.68, 'EXACT_PROB') exact,
```

stats_binomial_test (cust_gender, 'M', 0.68, 'ONE_SIDED_PROB_OR_LESS') prob_or_less from sh.customers;

stats_crosstab : Crosstabulation is a method used to analyze two nominal variables.
STATS_CORSSTAB Return Values :
CHISQ_OBS : observed value of chi-squared.
CHISQ_SIG : significance of observed chi-squared
CHISQ_DF : degree of freedom for chi-sqared
PHI_COEFICIENT : Phi coefficient
CRAMERS_V : Cramer's V statistic
CONT_COEFFICIENT : contigency coefficient
COHENS_K : Cohen's kappa
eg : determines the strength of the association between gender and income level.
select stats_crosstab(cust_gender, cust_income_level, 'CHISQ_OBS') chi_squared, stats_crosstab(cust_gender, cust_income_level, 'CHISQ_SIG') p_value, stats_crosstab(cust_gender, cust_income_level, 'PHI_COEFFICIENT') phi_coefficient from sh.customers;

stats_f_test : tests whether two variances are significantly different. The observed value of f is the ratio of one variance to the other, so values very different from 1 usually indicate significant differences.
STATS_F_TEST Return values :
STATISTIC : The observed value of f
DF_NUM : Degree of freedom for the number
DF_DEN : Degree of freedom for the denominator
ONE_SIDED_SIG : One-tailed significance of f
TWO_SIDED_SIG : Two-tailed significance of f
eg : determines whether the variance in credit limit between men and women is significantly different. The results, a p_value not close to zero, and an f_statistic close to 1, indicate that the difference between credit limits for men and women are not significant.

```
select variance(decode(cust_gender, 'M', cust_credit_limit, null)) var_men,
    variance(decode(cust_gender, 'F', cust_credit_limit, null)) var_women,
    stats_f_test(cust_gender, cust_credit_limit, 'STATISTIC', 'F') f_statistic,
    stats_f_test(cust_gender, cust_credit_limit) two_sided_p_value
    from sh.customers;
```

stats_ks_test : a Kolmogorov-Smirnov function that compares two samples to test whether they are from the same population or from populations that have the same distribution. It does not assume that the population from which the samples were taken is normally distributed.
STATS_KS_TEST Return values :
STATISTIC : Observed value of D
STG : Significance of D
eg : determines whether the distribution of sales between men and women is due to chance.
select stats_ks_test(cust_gender, amount_sold, 'STATISTIC') ks_statistic, stats_ks_test(cust_gender, amount_sold) p_value from sh.customers c, sh.sales s where c.cust_id = s.cust_id;

stats_mode : takes as its argument a set of alues and returns the value that occurs with the greatest frequency. If more thatn one mode exists, then Oracle Database chooses one and returns only that one value.
To obtain multiple modes (if multiple modes exist), you must use a combination of other functions, as shown in the hypothetical query :
select x from (select x, count(x) as cnt1 from t group by x) where cnt1 = (select max(cnt2) from (select count(x) as cnt2 from t group by x));
eg : returns the mode of salary per department in the hr.employees table.
select department_id, stats_mode(salary) from employees group by department_id order by department_id, stats_mode(salary);
eg : If you need to retrieve all of the modes (in cases with multiple modes), you can do so using a combination of other functions
select commission_pct from (select commission_pct, count(commission_pct) as cnt1 from employees group by commission_pct) where cnt1 = (select max(cnt2) from (select count(commission_pct) as cnt2 from employees group by commission_pct)) order by commission_pct;

stats_mw_test : A Mann Whitney test compares two independent samples to test the null hypothesis that two populations have the same distribution function against the alternative hypothesis that the two distribution functions are different.

eg : determines whether the distribution of sales between men and women is due to chance
select stats_mw_test(cust_gender, amount_sold, 'STATISTIC') z_statistic, stats_mw_test(cust_gender, amount_sold, 'ONE_SIDED_SIG', 'F') one sided_p_value from sh.customers c, sh.sales s where c.cust_id = s.cust_id;

stats_one_way_anova : The one-way analysis of variance function (STATS_ONE_WA_ANOVA) tests differences in means (for groups or variables) for statistical significance by comparing two different estimates of variance. One estimate is based on the variances within each group or category. This is known as the mean squares within or mean square error. The other estimate is based on the variances among the means of the groups. The is known as the mean squares between. If the means of the groups are significantly different, then the mean square between will be larger than expected and will not match the mean squares within. If the mean squares of the group are consistent, then the two variance estimates will be about the same.
eg : determines the significance of the differences in mean sales within an income level and differences in mean sales between income levels. The results, p_values close to zero, indicate that, for both men and women, the difference in the amount of googds sold across different income level is significant.

```
select cust_gender, stats_one_way_anova(cust_income_level, amount_sold, 'F_RATIO') f_ratio,
    stats_one_way_anova(cust_income_level, amount_sold, 'STG') p_value
    from sh.customers c, sh.sales s
    where c.cust_id = s.cust_id
    group by cust_gender
    order by cust_gender;
```

stats_t_test_* are :
stats_t_test_one : a one-sample t-test. This function obtains the value of t by dividing the difference between the sample mean and the known mean by the standard error of the mean (rather than the standard error of the difference of the means, as for STATS_T_TEST_PAIRED).
eg : determines the significance of the difference between the average list price and the constant value 60.
select avg(prod_list_price) group_mean, stats_t_test_one(prod_list_price, 60, 'STATISTIC') t_observed,
stats_t_test_one(prod_list_price, 60) two_sided_p_value from sh.products;

stats_t_test_paired : a two-samle, paired t-test (also known as a crossed t-test). This function obtains the value of t by dividing the difference between the sample means by the standard error of the difference of the means (rather than the standard error of the mean, as for STATS_T_TEST_ONE).

stats_t_test_indep : a t-test of two independent group with the same variance (pooled variances)

stats_t_test_indepu : a t-test of two independent groups with unequal variance (unpooled variances)
The t-test measures the significance of a difference of means. You can use it to compare the means of two groups or the means of one group with a constatn. The one-sample and two-sample stats_t_test* functions take three arguments : two expressions and a return value of type VARCHAR2. The functions return one number, determined by the value of the third argument.

stats_wsr_test
stddev
stddev_pop
stddev_samp
sum
var_pop
var_samp
variance
xmlagg
Analytic Functions : compute an aggregate value based on a group of rows.
avg
eg : calculates for each employee in the employee 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 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 order by manager_id, hire_date, salary;
```

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
Object Reference Functions : manipulate REF values, which are references to objects of specified object types.
deref
make_ref
ref
reftohex
value
Model Functions : can be used only in model_clause of the select statement.
cv
iteration_number
presentnnv
presentv
previous

OLAP Functions
cube_table

Data Cartridge Functions
dataobj_to_partition

eg : returns the absolute value of -15 :
select abs(-15) "Absolute" from dual;
eg :returns the arc cosine of .3 :
select acos(.3) "Arc_Cosine" from dual;
eg :returns the month after the hire_date in the sample table employees
select to_char(add_months(hire_date, 1), 'DD-MM-YYYY') "Next month" from employees where last_name = 'Baer';

User-Defined Functions :