Oracle PL/SQL

PL/SQL Functions

Agenda

01 DUAL Table

02 PL/SQL Functions

03 PL/SQL Built-in Functions

04 PL/SQL Single-row Functions

05 PL/SQL: Conversion Functions

06 PL/SQL: Analytic Functions

07 PL/SQL: User-defined Function

08 PL/SQL: Procedures

DUAL Table

- It has one row called ‘X’ and one column called ‘DUMMY.’
- The DUAL table is used to create SELECT statements and execute functions that are not directly related to a specific database table.
- Queries using a DUAL table return one row as a result.
- DUAL can be useful to do calculations and also to evaluate expressions that are not derived from a table.

DUMMY
X

DUAL Table

- **DUAL will be used to earn many of the single-row functions.**
- **SELECT(319/29) + 12 FROM DUAL;**
- **Here, the SELECT statement returns a value that does not exist in the DUAL table.**
- **The value returned is a result of the calculation executed.**

(319/29)+12
23

PL/SQL Functions

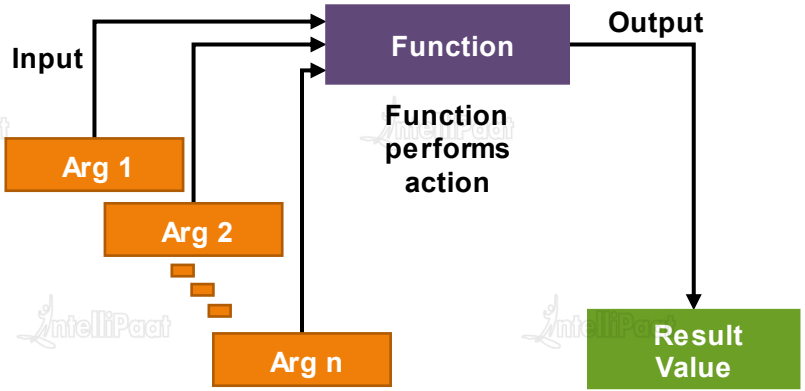
Functions in PL/SQL

- **Functions are small programs that preform an action on a value or column and produce something different as the output.**
- **In SQL, there are many types of functions that are used to transform an input in one form to an output in another form.**
- **These functions are used to manipulate data values.**
- **Functions have both input and output. The input into a function is referred to as an argument.**

Functions in PL/SQL

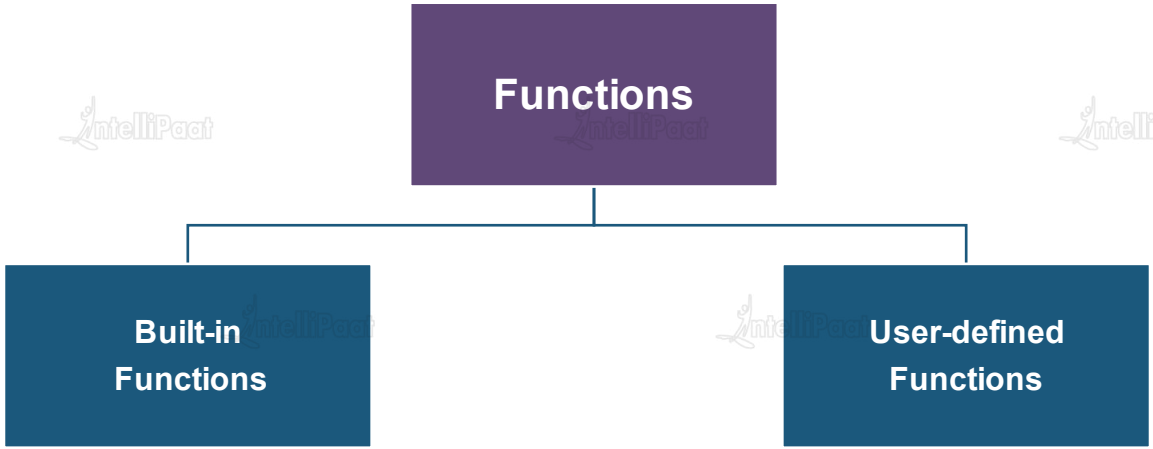
Functions are a very powerful feature of SQL. They are used to transform an input in one form to an output in another form. They can be used to:

- **Perform calculations on data**
- **Modify individual data items**
- **Manipulate input for groups of rows**
- **Format dates and numbers for display**
- **Convert column data types**



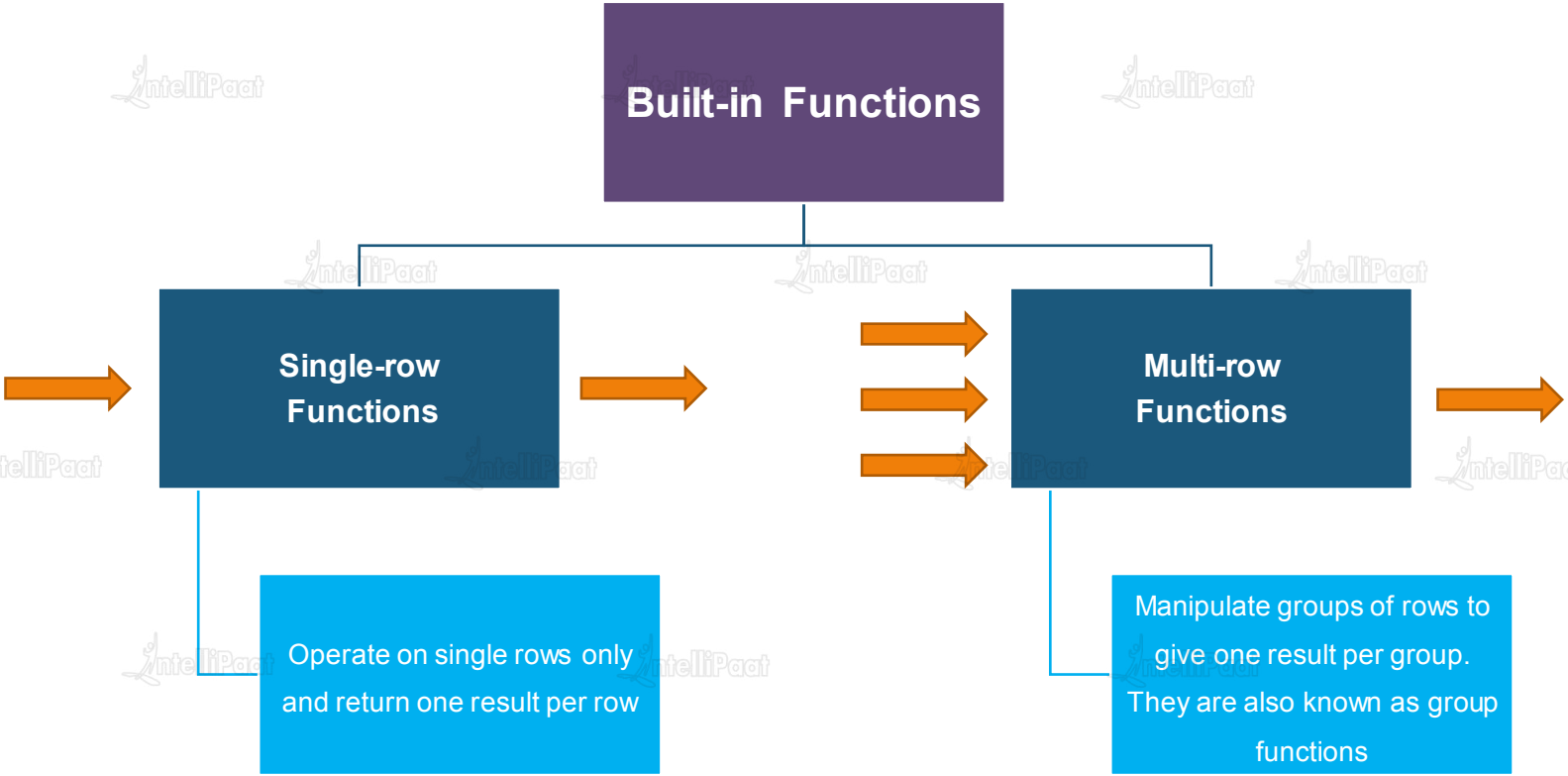
SQL functions sometimes take arguments and always return a value.

Types of PL/SQL Functions



Built-in Functions

Types of Built-in Functions



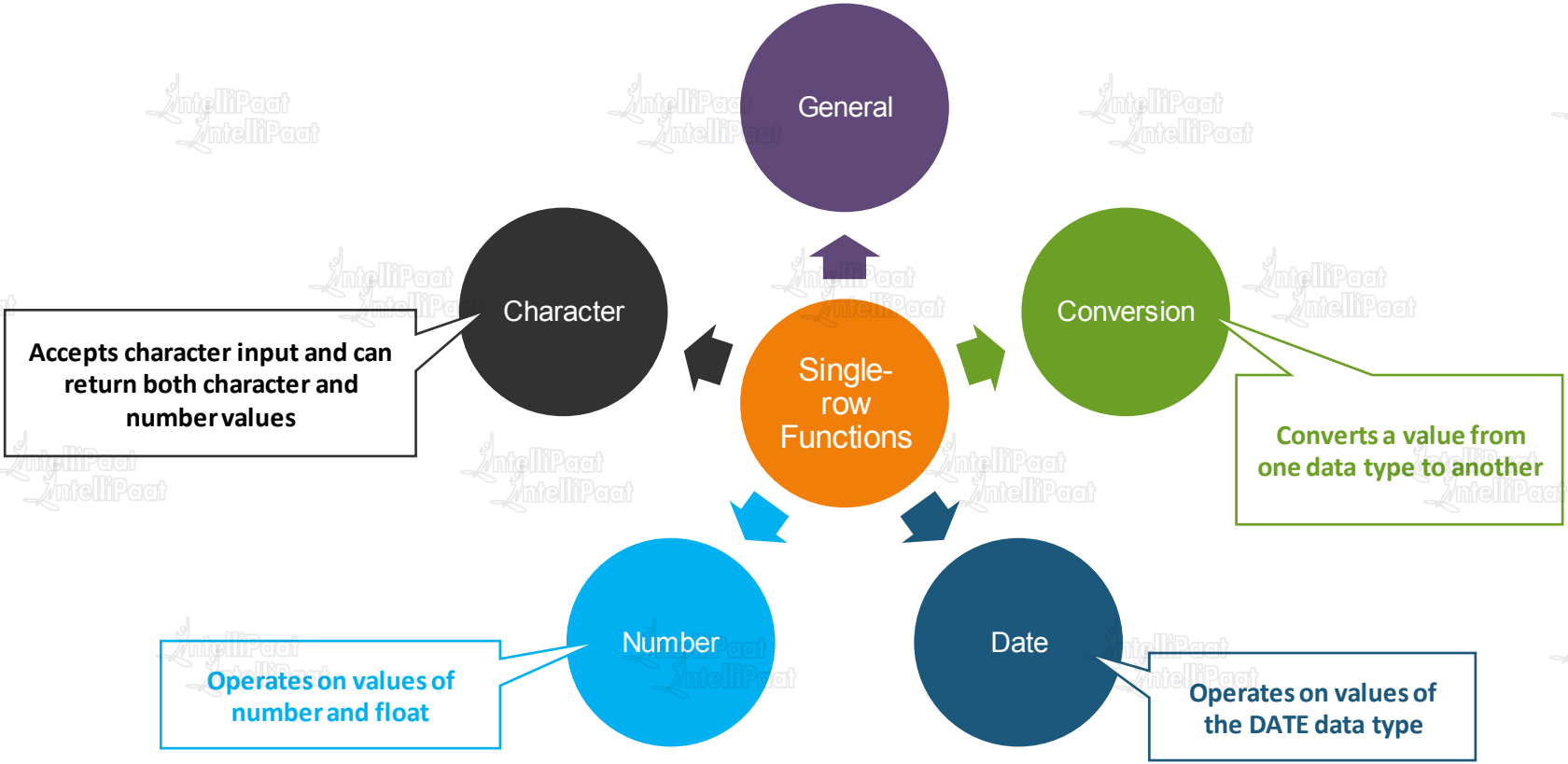
Single-row Functions

Single-row Functions

- Manipulate data items
- Accept arguments and return one value
- Act on each row returned
- Return one result per row
- May modify the data type
- Can be nested
- Accept arguments which can be a column or an expression

```
function_name[(arg1, agr2…)]
```

Single-row Functions



Case Manipulation Functions

- Used to know in which case (upper, lower, or mixed) the data is stored in the database
- These functions convert case for character strings

Use of Function	Result
Select LOWER('SQL Course') from dual;	sql course
Select UPPER('SQL Course') from dual;	SQL COURSE
Select INITCAP ('SQL Course') from dual;	Sql Course

```
SELECT job, salary FROM emp WHERE LOWER (name) = 'smith';
```

```
SELECT 'The job id for' || UPPER (last_name) || 'is' || LOWER(job_id) AS "EMPLOYEE DETAILS" FROM emp;
```

Character Manipulation Functions

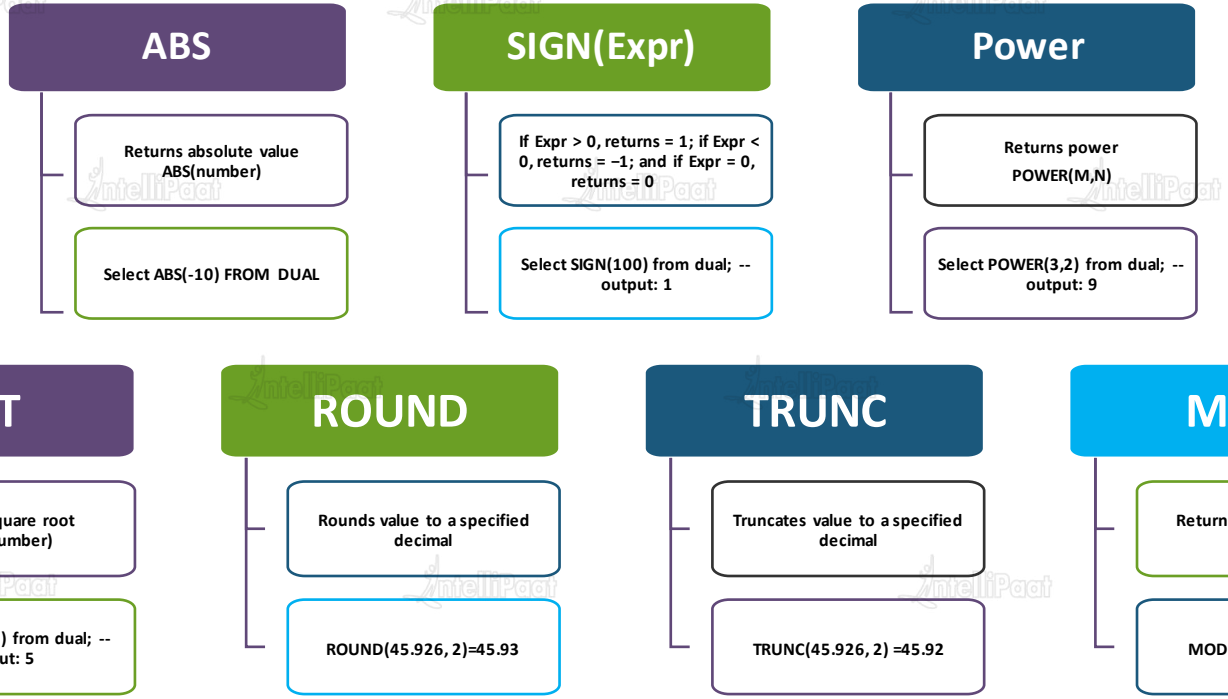
These functions manipulate character strings

Use of Function	Result
Select concat('Hello), 'World') from dual;	HelloWorld
Select SUBSTR('HelloWorld',1,5) from dual;	Hello
Select LENGTH('HelloWorld') from dual;	10
Select INSTR('HelloWorld', 'W') from dual;	6
Select LPAD(salary,10,'*')from dual;	*****24000
Select RPAD(salary, 10, '*')from dual;	24000*****
Select TRIM('H' FROM'HelloWorld') from dual;	HelloWorld

```
Select empno, name, length(name), substr(job,1,3), instr (job,'A') from emp;
```

Number Functions

Numeric functions accept numeric input and return numeric values.



Character Manipulation Functions

```
SELECT ROUND (45. 923,2), ROUND(45.923), ROUND (45.923, -2) FROM DUAL;
```

ROUND (45. 923,2)
45. 92

ROUND (45. 923)
45

ROUND (45. 923,-2)
0

Using the TRUNC Function

```
SELECT TRUNC (45. 923,2), TRUNC(45.923), TRUNC (45.923, -2) FROM DUAL;
```

TRUNC (45. 923,2)
45. 92

TRUNC (45. 923)
45

TRUNC (45. 923,-2)
0

Number Functions

Calculate the remainder salary after it is divided by 5,000 for all employees whose job title is SALESMAN

```
SELECT name, salary, MOD(salary, 5000)
FROM emp
WHERE job = 'SALESMAN';
```

CEIL()

This function is used to round the given number to the highest number.

```
select ceil(-9.4) from dual;
```

FLOOR()

This function is used to round the given number to the lowest number.

```
select floor(-9.4) from dual;
```

Greatest()

This function is used to return the maximum value from a list of numbers.

```
select greatest(5,6,7,8) from dual;
```

Least()

This function is used to return the minimum value from a list of numbers.

```
select least(5,6,7,8) from dual;
```

Working with Dates

- Oracle database stores dates in an internal numeric format: century, year, month, day, hours, minutes, and seconds
- The default date display format is DD-MON-YY
  - Allows you to store 21st century dates in the 20th century by specifying only the last two digits of the year
  - Allows you to store 20th century dates in the 21st century in the same way

```
SQL> select sysdate from dual;

SYSDATE
29-NOV-16
```

```
SQL> SELECT name, hiredate
2 FROM emp
3 WHERE name like 'A%';
NAME    HIREDATE
ADAMS 1983-01-12
ALLEN  1981-02-20
```

Date Functions

- Date functions operate on Oracle dates.
- All date functions return a value with a DATE data type except the MONTHS_BETWEEN function, which returns a numeric data type value.

Function	Description
MONTHS_BETWEEN	Number of months between two dates
ADD_MONTHS	Adds calendar months to a date
NEXT DAY	Next day of a specified date
LAST_DAY	Last day of the month
ROUND	Rounds date
TRUNC	Truncates date
SYSDATE	Returns system date
EXTRACT	Extracts a part of the date

Date Functions

```
SQL> Select ADD_MONTHS(SYSDATE,12) from
dual;

 ADD_MONTH

25-NOV-17

SQL> Select ADD_MONTHSC11-APR-05',33 from
dual;

ADD_MONTH

11-JUL-OS
```

```
SQL> select NEXT_DAY(sysdate,'FRIDAY')  from
dual;
NEXT_DAY(
02-DEC-16
SQL> SELECT LAST_DAY(SYSDATE) FROM
DUAL;
LAST_DAY(
30-NOV-16
SQL> SELECT LAST_DAY('10-FEB-2016') FROM
DUAL;
 LAST_DAY(
29-FEB-16
```

Arithmetic with Dates

- Add or subtract a number to or from a date for a resultant date value
  - Subtract two dates to find the number of days between those dates
  - Add hours to a date by dividing the number of hours by 24

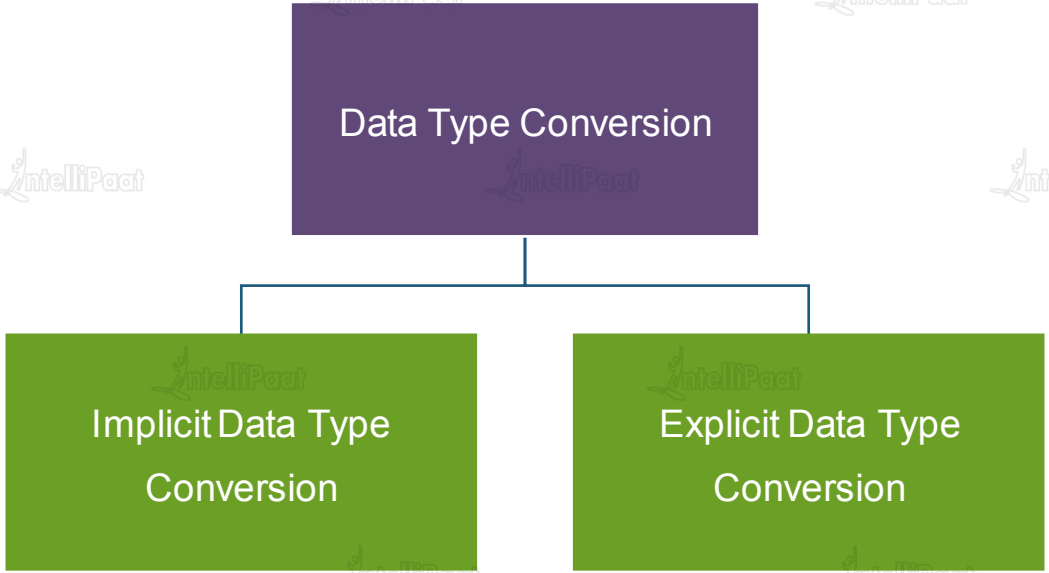
Date + Number (Date)	Date – Number (Date)	Date – Date (Number of Days)	Date + Number/24 Date
•Adds number of days to a date	•Subtracts number of days from a date	•Subtracts one date from another	•Adds number of hours to a date

```
SELECT name, (SYSDATE-hiredate)/7 AS
WEEKS
FROM emp
WHERE deptno = 30;
```

Conversion Functions

Conversion Functions

- Used to convert data from one data type to another data type



Implicit Data Type Conversion

If conversion is performed by Oracle, then it is called implicit data type conversion.

- For assignments, Oracle Server can automatically convert the following:

From	To
VARCHAR2 or CHAR	NUMBER
VARCHAR2 or CHAR	DATE
NUMBER	VARCHAR2
DATE	VARCHAR2

```
SQL) -- Find the total salary of deptno 10
SQL) select sum(salary) from employee where deptno = '10';
SUM(SALARY)
8750
```

- For expression evaluation, Oracle Server can automatically convert the following:

From	To
VARCHAR2 or CHAR	NUMBER
VARCHAR2 or CHAR	DATE

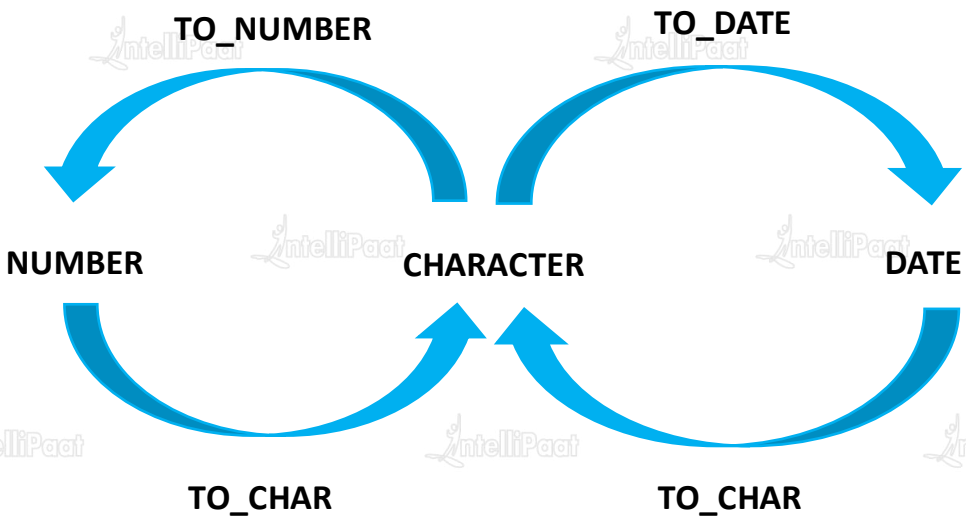
```
SQL> select 1000 + '1000' from dual;
1000+'1000'
2000
```

Explicit Data Type Conversion

If conversion is performed by the User, then it is called explicit data type conversion.

There are three functions:

- TO_CHAR
- TO_NUMBER
- TO_DATE



TO_CHAR Function

TO_CHAR is used to convert a NUMBER/DATE value to CHAR type.

- Format elements are used with this function to convert a number value as a character.

```
TO_CHAR(number; 'format model)
```

9	Represents a number
0	Forces a zero to be displayed
$	Places a floating dollar sign
L	Uses the floating local currency symbol
.	Prints a decimal point
,	Prints a thousand indicator

```
Select empno, name, T0_CHAR(salary19,999') from employee;

Select empno, name, T0_CHAR(salary199,991 from employee;

Select empno, name, TO_CHAR(salary/$9,9991 from employee;

Select empno, name, T0_CHAR(salary,18,8881 from employee;

//invalid No. format
```

TO_CHAR Function with Date Type

TO_CHAR(date, 'format_model')

- Must be enclosed in single quotation marks and is case sensitive
- Can include any valid date format element
- Has an FM element to remove padded blanks or to suppress leading zeros
- Is separated from the date value by a comma

YYYY	Full year in numbers. Example: 2016
YEAR	Year spelled out
MM	2-digit value for a month. Example: 01 for January
MONTH	Full name of the month. Example: January
MON	3-letter abbreviation of a month. Example: Jan
DY	3-letter abbreviation of a day of the week. Example: Mon
DAY	Full name of a day of the week
DD	Numeric day of a month

- Time elements format the time portion of the date

HH24:MI:SSAM | 15:45:32 PM

- Add character strings by enclosing them in double quotation marks

DD "of" MONTH | 12 of OCTOBER

- Number suffixes spell out numbers

Ddspth | fourteenth

Using TO_NUMBER and TO_DATE Functions

- Convert a character string to a number format using the TO_NUMBER function

```
TO_NUMBER(chart[, 'format model'])
```

```
SQL> select to_number('$1,000','L9,999') + 1500 from dual;
TO_NUMBER('$1,000','L9,999')+1500
2500
```

- Convert a character string to a date format using the TO_DATE function

```
TO_DATE(char[, 'format_model'])
```

```
SQL> select to_date('08/01/2016 ',' 'dd/mm/yy')+10 from dual;
 TO_DATE('
18-JAN-16
```

Analytic Functions

Analytic Functions

- Give an aggregate result; they do not group the result set
- Return the aggregate value multiple times with each record
- Are used to compute aggregates
- Allows you to calculate:

Rankings and Percentiles	Moving Window Calculations	Lag/Lead Analysis	First/Last Analysis	Linear Regression Statistics

- Any other non ‘group by’ column or expression can be presented in the select clause

  For example: The columns, empno and salary

Analytic Functions: Syntax

The general syntax of analytic functions is:

```
Function(argl,...,  argn) OVER ( [PARTITION  BY <...>] [ORDER BY <....>][<window_clause>]  )

<window_clause> is like "ROW <?>" or "RANK <?>"
```

- Analytic functions are computed after all joins, such as WHERE clause, GROUP BY, and HAVING, are computed on the query.
- The main ORDER BY clause of the query operates after analytic functions.
- They can only appear in the select list and in the main ORDER BY clause of the query.

```
OVER() without  PARTITION  or

<window  clause>
```

- Over() acts on the entire record set returned by the WHERE clause.

Analytic Functions: Example

```
SELECT empno, depty, COUNT(*) OVER ( ) No_of_Employees

FROM emp WHERE deptno IN (10, 20)

ORDER BY 2, 1;
```

EMPNO	DEPTNO	# of Employees
7782	10	8
7839	10	8
7934	10	8
7369	20	8
7566	20	8
7788	20	8
7876	20	8
7902	20	8

Breaking the Result Set into Groups or Partitions

- PARTITION BY is used to break the result set into groups.
- PARTITION BY can take any non-analytic SQL expression.

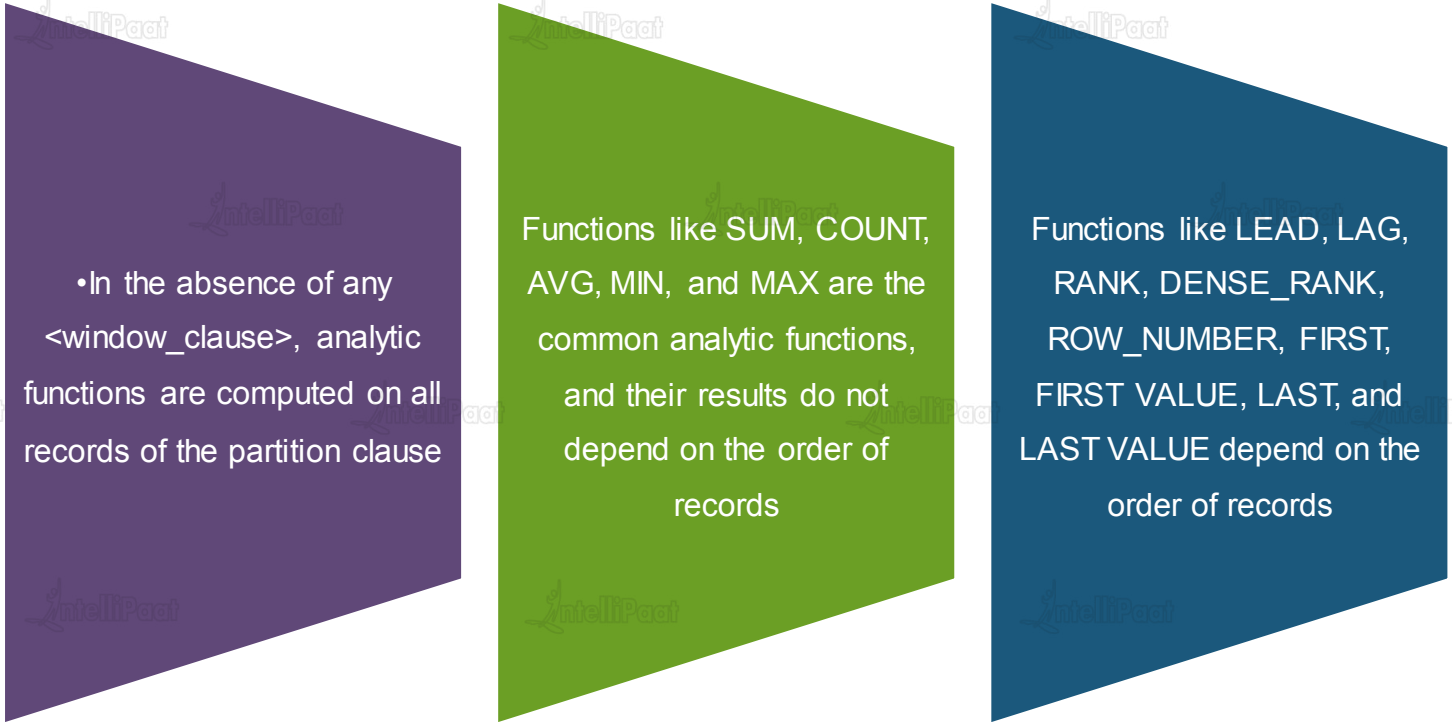
EMPNO	DEPTNO	# of Employees
7782	10	3
7839	10	3
7934	10	3
7369	20	5
7566	20	5
7788	20	5
7876	20	5
7902	20	5

```
SELECT empno, deptno, COUNT(*) OVER (partition by deptno )

No_of_Employees

FROM emp WHERE deptno IN (10, 20) ORDER BY 2, 1;
```

Breaking the Result Set into Groups or Partitions



Specifying the Order of Records in Partition

- By using the 'ORDER BY clause inside the OVER() clause
- Syntax of the ORDER BY clause in the analytic function is:

```
[ORDER BY <sql_expr> [ASC or DESC] NULLS [FIRST or LAST]
```

```
SELECT empno, deptno, COUNT(*) OVER (order by deptno )
No_of_Employees FROM emp WHERE deptno IN (10, 20)
```

EMPNO	DEPTNO	# of Employees
7782	10	3
7934	10	3
7839	10	3
7369	20	8
7902	20	8
7788	20	8
7566	20	8
7876	20	8

```
SELECT empno, deptno, COUNT(*) OVER (order by deptno
desc) No_of_Employees FROM emp WHERE deptno IN (10, 20)
```

EMPNO	DEPTNO	# of Employees
7876	20	5
7369	20	5
7902	20	5
7566	20	5
7788	20	5
7782	10	8
7934	10	8
7839	10	8

ROW NUMBER, RANK, and DENSE_RANK

- All these functions assign integer values to rows depending on their order.
- ROW_NUMBER() gives a running serial number to a partition of records.
- They are very useful in reporting, such as for generating serial numbers for individual partitions.
- To generate separate sets of running serial for employees of departments 10 and 20 based on their HIREDATE:

```
SQL> SELECT empno, deptno, hiredate,
  2  ROW_NUMBER( ) OVER (PARTITION BY deptno ORDER BY hiredate NULLS LAST) SLNo
  3  FROM employee WHERE deptno IN (10, 20) ORDER BY deptno, SLNo;

     EMPNO     DEPTNO HIREDATE       SLNO
---------- ---------- --------- ----------
      7782         10 09-JUN-81          1
      7839         10 17-NOV-81          2
      7934         10 23-JAN-82          3
      7369         20 17-DEC-80          1
      7566         20 02-APR-81          2
      7902         20 03-DEC-81          3
      7788         20 09-FEB-82          4
      7876         20 12-JAN-83          5
```

RANK and DENSE RANK

- These functions compute the rank of a record in comparison with other records in a data set based on the values.
- RANK() ranks items in a group by leaving gaps in the ranking sequence when there are ties.
- DENSE_RANK() ranks items in a group leaving no gaps in the ranking sequence when there are ties.
- In case of a tie between two records at a position *N*, RANK() declares two positions; it skips the position N+1 and gives the position N+2 to the next record. While DENSE_RANK() also declares two positions but does not skip the position N+1.

RANK and DENSE RANK

```
SELECT empno, deptno, salary,
RANK() OVER (PARTITION  BY deptno
ORDER BY salary DESC NULLS LAST) RANK,
DENSE RANK() OVER (PARTITION  BY deptno
ORDER BY salary DESC NULLS LAST)
DENSE_RANK
FROM employee WHERE deptno IN (10, 20)
ORDER BY 2, RANK;
```

EMPNO	DEPTNO	SALARY	RANK	DENSE_RANK
7839	10	5000	1	1
7782	10	2450	2	2
7934	10	1300	3	3
7788	20	3000	1	1
7902	20	3000	1	1
7566	20	2975	3	2
7876	20	1100	4	3
7369	20	800	5	4

LEAD and LAG

- They are useful for comparing values when the relative positions of rows can be known reliably.
- LAG function provides access to a row at a given offset prior to the current position.
- LEAD function provides access to a row at a given offset after the current position.

```
LAG / LEAD (<sql_expr>. <offset>. <default>) OVER (<analytic_clause>)
```

where:

- <sql_expr> is the expression to compute from the leading row
- <offset> is the index of the leading row relative to the current row for LEAD, whereas a previous row for LAG
- <offset> is a positive integer with a default value, 14
- <default> is the value to return if the <offset> points to a row outside the partition range

LEAD and LAG

```
SELECT deptno, empno, salary,
LEAD(salary, 1, 0) OVER (PARTITION  BY deptno ORDER BY salary DESC NULLS LAST)
NEXT_LOWER_SAL,
LAG(salary, 1, 0) OVER (PARTITION  BY deptno ORDER DESC NULLS LAST) PREV_HIGHER_SAL
FROM employee
WHERE deptno IN (10, 20)
ORDER BY deptno, salary DESC;
```

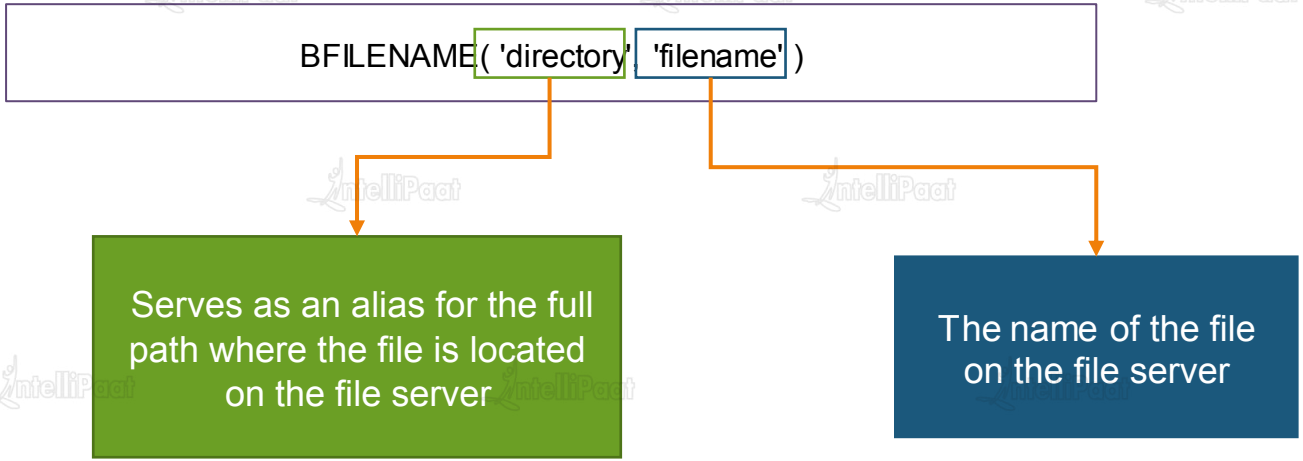
DEPTNO	EMPNO	SALARY	NEXT_LOWER_SAL	PREV_HIGHER_SAL
10	7839	5000	2450	0
10	7782	2450	1300	5000
10	7934	1300	0	2450
20	7788	3000	3000	0
20	7902	3000	2975	3000
20	7566	2975	1100	3000
20	7876	1100	800	2975
20	7369	800	0	1100

Advanced Functions

BFILENAME Function

- Returns a BFILE locator for a physical LOB binary file

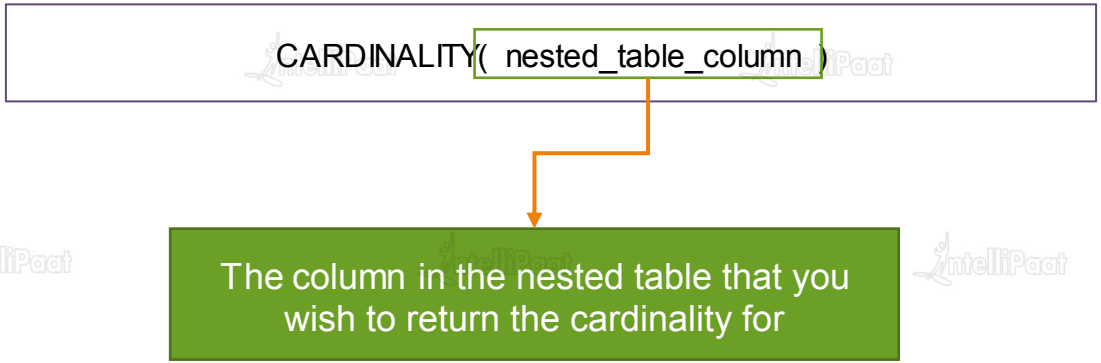
Syntax



CARDINALITY Function

- Returns a number of elements in a nested table
- If the nested table is empty, the CARDINALITY function will return NULL
- If the nested table is a null collection, the CARDINALITY function will return NULL

Syntax



CASE Statement

- The Oracle/PLSQL CASE statement has the functionality of an IF-THEN-ELSE statement. Starting in Oracle 9i, you can use the CASE statement within a SQL statement.

Syntax

```
CASE [ expression ]

  WHEN condition_1 THEN result_1
  WHEN condition_2 THEN result_2
  ...
  WHEN condition_n THEN result_n

  ELSE result

END
```

COALESCE Function

- It returns the first non-null expression in the list. If all expressions evaluate to null, then the COALESCE function will return null.

Syntax

```
COALESCE( expr1, expr2, ... expr_n )
```

DECODE Function

- The Oracle/PLSQL DECODE function has the functionality of an IF-THEN-ELSE statement.

Syntax

```
DECODE( expression , search , result [, search , result]... [, default] )
```

EMPTY_BLOB Function

- It is used to initialize a LOB column to EMPTY in an INSERT or UPDATE statement, or it can be used to initialize a LOB variable.

Syntax

```
EMPTY_BLOB ()
```

EMPTY_CLOB Function

- It is used to initialize a LOB column to EMPTY in an INSERT or UPDATE statement, or it can be used to initialize a LOB variable.

Syntax

```
EMPTY_CLOB ()
```

GROUP_ID Function

- It assigns a number to each group resulting from a GROUP BY clause. The GROUP_ID function is most commonly used to identify duplicated groups in your query results.

Syntax

```
SELECT column1, column2, ... column_n, GROUP_ID()
FROM tables
WHERE conditions
GROUP BY column1, column2, ... column_n;
```

LNNVL Function

- It is used in the WHERE clause of a SQL statement to evaluate a condition when one of the operands contains a NULL value.

Syntax

```
LNNVL( condition )
```

NANVL Function

- It lets you substitute a value for a floating point number such as BINARY_FLOAT or BINARY_DOUBLE, when a NaN (Not a Number) value is encountered.
- Most commonly, it is used to convert NaN values to either NULL or 0.

Syntax

```
NANVL( value, replace_with )
```

NULLIF Function

- It compares expr1 and expr2. If expr1 and expr2 are equal, the NULLIF function returns NULL. Otherwise, it returns expr1.

Syntax

```
NULLIF( expr1, expr2 )
```

NVL Function

- It lets you substitute a value when a null value is encountered.

Syntax

```
NVL( string1, replace_with )
```

NVL2 Function

- This extends the functionality of the NVL function. It lets you substitutes a value when a null value is encountered, and also when a non-null value is encountered.

Syntax

```
NVL2( string1, value_if_not_null,  value_if_null  )
```

SYS_CONTEXT Function

- It is used to retrieve information about the Oracle environment.

Syntax

```
SYS_CONTEXT( namespace, parameter [, length] )
```

UID Function

- It returns the ID number for a user's session (when the user who is currently logged in).

Syntax

```
UID
```

USER Function

- It returns the user_id from the current Oracle session.

Syntax

```
USER
```

USERENV Function

- It is used to retrieve information about the current Oracle session. Although this function still exists in Oracle for backwards compatibility, it is recommended that you use the SYS_CONTEXT function instead.

Syntax

```
USERENV( parameter )
```

User-defined Functions

Create Function

- Helps in extending SQL statements as well as modularizing and abstracting complex business logic
- This function always returns a value and is used in expressions to assign a variable or to directly fetch values from SQL statements by performing insert, update, or delete operations on DML statements

1.IN	1.OUT	1.IN OUT
2.Referring to the procedure or function and allow to overwritten the value of parameter.	2.Can not be referenced by the procedure or function, but the value of the parameter can be overwritten by the procedure or function.	2.Referring to the procedure or function to pass both IN and OUT parameters, modify/update by the function or procedure, and get returned

Create Function: Syntax

```
CREATE [OR REPLACE] FUNCTION  [SCHEMA..] function_name
      [ (parameter [,parameter]) ]
      RETURN  return_datatype
   IS | AS
      [declaration_section
         variable declarations;
         constant  declarations;
      ]
   BEGIN
      [executable_section
         PL/SQL execute/subprogram  body
      ]
   [EXCEPTION]
         [exception_section
         PL/SQL Exception block
         ]
   END [function_name];
   /
```

Function Example

EMP_NO	EMP_NAME	EMP_DEPT	EMP_SALARY
1	Forbs Ross	Web Developer	45k
2	Marks Jems	Program Developer	38k
3	Saulin	Program Developer	34k
4	Zenia Sroll	Web Developer	42k

Create Function

```
SQL>edit fun1
CREATE or REPLACE FUNCTION fun1(no in number)
RETURN varchar2
IS
   name varchar2(20);
BEGIN
   select ename into name from emp1 where eno = no;
   return name;
END;
/
```

PL/SQL Program to Calling Function

```
SQL>edit fun
DECLARE
   no number :=&no;
   name varchar2(20);
BEGIN
   name := fun1(no);
   dbms_output.put_line('Name:'||'    '||name);
   end;
/
```

PL/SQL DROP FUNCTION

- You can drop a PL/SQL function using the DROP FUNCTION statement.

Syntax

```
DROP FUNCTION  function_name;
```

PL/SQL Procedure

PL/SQL Procedure

- Subprogram that returns multiple values
- Allows you to centralize your business logic in the database
- Used by any program that accesses the database

Syntax

```
CREATE [OR REPLACE] PROCEDURE
procedure_name
[(parameter_name [IN | OUT | IN OUT] type [, ...])]
{IS | AS}
BEGIN
  procedure_body
END procedure_name;
```

Procedure Example

EMP_NO	EMP_NAME	EMP_DEPT	EMP_SALARY
1	Forbs Ross	Web Developer	45k
2	Marks Jems	Program Developer	38k
3	Saulin	Program Developer	34k
4	Zenia Sroll	Web Developer	42k

Create Procedure

```
SQL>dit pro1
CREATE or REPLACE PROCEDURE pro1(no in
number,temp  out emp1%rowtype)
IS
BEGIN
   SELECT * INTO  temp FROM emp1 WHERE eno =
no;
END;
/
```

PL/SQL Program to Calling Procedure

```
SQL>edit pro
DECLARE
   temp emp1%rowtype;
   no number :=&no;
BEGIN
   pro1(no,temp);
   dbms_output.put_line(temp.eno||'      '||
                        temp.ename||'    '||
                        temp.edept||'    '||
                        temp.esalary||'  '||);
END;
/
```

PL/SQL DROP PROCEDURE

- You can drop a PL/SQL procedure using the DROP PROCEDURE statement.

Syntax

```
DROP PROCEDURE procedure_name;
```

Quiz

Quiz 1

The || is is an example of what function ? SELECT last_name || ', ' || first_name || ' ' || middle_name FROM employees;

A	Incantination
B	Integration
C	Continuation
D	Concatenation

Quiz 1

The || is is an example of what function ? SELECT last_name || ', ' || first_name || ' ' || middle_name FROM employees;

A	Incantination
B	Integration
C	Continuation
D	Concatenation

Quiz 2

In Oracle SQL, what is the command that returns the first non-null value in a list of values?

A	DECODE
B	COALESCE
C	INSTR
D	ISNOTNULL

Answer 2

In Oracle SQL, what is the command that returns the first non-null value in a list of values?

A	DECODE
B	COALESCE
C	INSTR
D	ISNOTNULL

Quiz 3

Which of the following is an explicit numeric, character, string, or BOOLEAN value not represented by an identifier?

A	Delimiters
B	Literals
C	Comments
D	None of the above

Answer 3

Which of the following is an explicit numeric, character, string, or BOOLEAN value not represented by an identifier?

A	Delimiters
B	Literals
C	Comments
D	None of the above

Quiz 4

Which of the following is used to declare a record?

A	%ROWTYPE
B	%TYPE
C	Both A and B
D	None of the above

Answer 4

Which of the following is used to declare a record?

A	%ROWTYPE
B	%TYPE
C	Both A and B
D	None of the above

Quiz 5

Like all identifiers, the names of constants, variables, and parameters are case sensitive.

A	True
B	False

Answer 5

Like all identifiers, the names of constants, variables, and parameters are case sensitive.

A	True
B	False

India: +91-7847955955

US: 1-800-216-8930 (TOLL FREE)

sales@intellipaat.com

24/7 Chat with Our Course Advisor