NUGGET 3: FUNCTIONS

Persistent University

PERSISTENT

Partners in Innovation

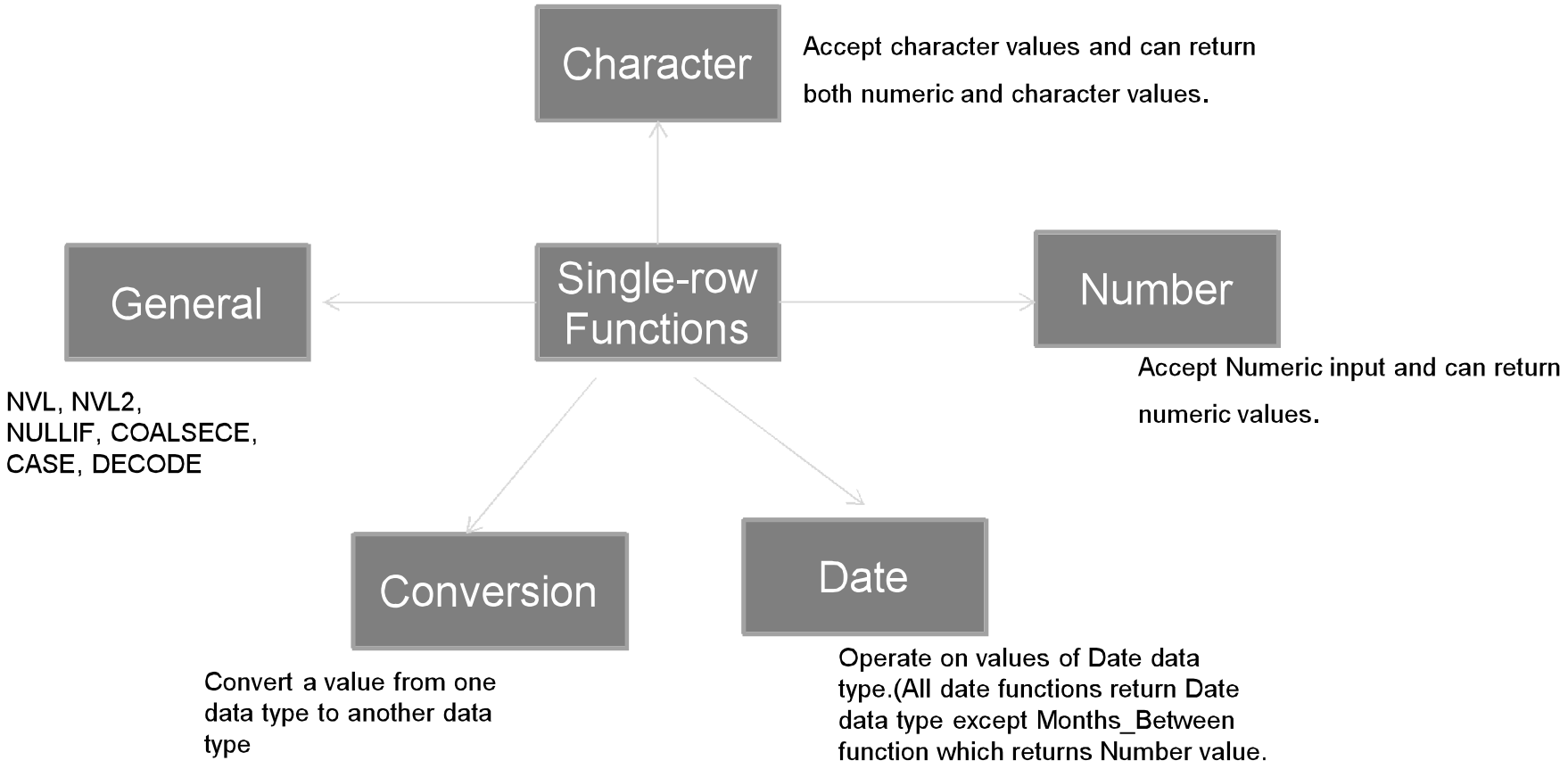
KEY LEARNING POINTS

- Single Row Functions:
  - Conversion functions
  - Conditional Expressions
- Group Functions
  - SUM, AVG, COUNT,MAX, MIN
  - Grouping data using GROUP BY clause
  - Excluding groups using HAVING Clause

FUNCTIONS

- Functions are very powerful feature of SQL and can be used to do the following:
  - Perform calculation on data.
  - Modify individual data items.
  - Manipulate output of group of rows.
  - Format date and numbers for display.
  - Convert column data types.
- SQL functions sometimes takes argument and always returns value.
- There are two distinct types of SQL functions
  - Single Row Functions
  - Multiple Row Functions

SINGLE ROW FUNCTIONS



Note : For Character, Date, Numeric and general Single row functions, Please refer Basic SQL Queries Course.

SAMPLE DATA

PERSISTENT

Table Name : Employee

EmployeeId	FirseName	LastName	Email	PhoneNumber	HireDate	JobId	Salary	CommitionPct	ManagerId	DepartmentId
1	John	Demn	JohnD@yahoo.com	9898780979	1/10/2001	IT_PROF	70000	0.5	NULL	10
2	Ken	Dale	kendaleD@gmail.com	7877787655	4/1/2001	SALES_HEAD	50000	NULL	NULL	10
3	James	Walton	JW@yahoo.com	5787887888	1/1/2001	IT_REP	30000	0.2	1	20
4	robin	sngal	robin@gmail.com	4990988839	5/1/2001	SALES_REP	40000	0.3	2	20
5	ajay	ghosala	ghosala@hotmail.com	9809888898	6/10/2002	SALES_REP	30000	0.4	2	20

CONVERSION FUNCTIONS

- **IMPLICIT Data type conversion:**

  The Oracle server can automatically convert the following

  - From Varchar2/ CHAR To NUMBER
  - From Varchar2/ CHAR To Date
  - From Number to Varchar2
  - From Date to Varchar2

  For Example:

  Test_var varchar2(20)

  Test_var=10

  Test_var number

  Test_var='10'

CONVERSION FUNCTIONS

PERSISTENT

- **EXPLICIT Data type Conversion**: SQL provides three functions to convert a value from one data type to another.
  - To_char
  - To_number
  - To_Date

- **To_char(Date, 'format_model'):** Convert a date value to varchar2 character string with format model given.
  - Format model :
    - must be enclosed in single quotes
    - is separated from date by comma
    - can include any valid data format element.
    - uses fm element to remove padding blanks or leading 0s

CONVERSION FUNCTIONS

- Some of the Elements of Date format model

YEAR	HH
YYYY	HH24
YY	MI
MONTH	SS
MON	AM/PM
MM	DAY
DD	DY
WW	

- Some suffixes in format model
  - TH: Ordinal number (For eg. DDTH for 4th)
  - SP: Spelled Out number (For eg DDSP for FOUR)
  - SPTH or THSP : Spelled out ordinal numbers (For eg. DDSPTH for Fourth)

CONVERSION FUNCTIONS

- **To_char(Number, 'format_model'):** Convert a numeric value to varchar2 character string with format model given.
  - Format model
    - must be enclosed in single quotes
    - is separated from number by comma
    - can include any valid Number format element.
  - Some of the Elements of Number format model
    - 9 : Represents a number (Used to specify the width)
    - 0 : Forces to display 0.
    - $: Places a dollar sign
    - L : Uses the local currency symbol
    - . : Prints decimal point
    - , : Prints a thousand indicator

Note: Oracle server displays a string of # sign in place of whole number if number of digits exceed the number of digits provided in the format model. Oracle server rounds the stored decimal value to the number of decimal spaces provided in the format model.

CONVERSION FUNCTIONS

- To_number(char [,'format_model']): Converts a character string to a number format
- To_date(char[,'format_model']): Converts a character string to a date format

Note: Format elements are same as seen in To_char function.

CONVERSION FUNCTION EXAMPLES

PERSISTENT

- Select sysdate, to_char(sysdate, 'fmDD MONTH YYYY')
  from dual;

SYSDATE	TO_CHAR(SYSDATE,'FMDDMONTHYYYY')
16-JUL-13	16 JULY 2013

- Select sysdate, to_char(sysdate, 'fmDDth " of " MONTH YYYY HH24:MI:SS')
  from dual;

SYSDATE	TO_CHAR(SYSDATE,'FMDDTH"OF"MONTHYYYYH H24:MI:SS')
16-JUL-13	16TH of JULY 2013 15:02:43

- Select to_char(salary, '$99,999.00') salary from employee;

SALARY
$70,000.00
$50,000.00
$30,000.00
$40,000.00
$30,000.00

- Select to_number('10,000.00','99,999.00') "Numeric value"
  from dual;

Numeric value
10000

- Select lastname,hiredate from employee
  where hiredate=to_date('Jun 10, 2002','Mon DD, YYYY');

LASTNAME	HIREDATE
ghosala	10-JUN-02

NESTING FUNCTIONS

- Single Row Functions can be nested to any level.
- Nested functions are evaluated from innermost level to outermost.

F3(F2(F1(col,arg1), arg2), arg3)

- Example:

```
select replace(lpad(substr('I am best', 3),10,'*'),'*','<') from dual;
```

Output of substr('I am best', 3): 'am best' ..Which will go as input to lpad

Output of lpad('am best',10,'*'): '***am best' ..Which will go as input to replace

Output of Replace('***am best','*','<') : '<<<am best'

CONDITIONAL EXPRESSIONS

- Conditional expressions provides the use of IF-Then-Else logic within a SQL statement. There are two types of conditional expressions.
  - CASE Expression
  - DECODE Function

- **CASE Syntax:**

```
CASE Expr       WHEN comparison_expr1 THEN return_expr1
     [WHEN comparison_expr2 THEN return_expr2
     WHEN comparison_exprn THEN return_exprn
     ELSE else_expr]
END
```

  - Oracle searched for the first WHEN ..THEN pair for which expr is equal to comparison_expr and returns the corresponding result_expr. If none of WHEN….THEN pair meet the condition and if ELSE clause exists, it returns else_expr.
  - User can not specify literal NULL for the return_expr or else_expr.
  - Expressions expr and comparison_expr must be of same data type; which can be CHAR, varchar2, number etc

CONDITIONAL EXPRESSIONS

- **DECODE Syntax:**

DECODE(col/expression,

search1, result1

[search2, result2,.....

searchn, resultn]

[,default])

- The DECODE function decodes an expression in IF-THEN-ELSE Logic.
- It decodes expression after comparing it with each search value. If the expr is same as search then result is returned.
- If default value is omitted, a null value is returned where expression does not match any of the search values.

CONDITIONAL EXPRESSION EXAMPLES

- Select employeeid, lastname, jobid, salary,

```
Case Jobid when 'SALES_REP' then '1.05*salary'
           when 'SALES_HEAD' then '1.12*salary'
           Else 'salary'
                END "Calculation for Revised Salary"
From Employee;
```

- select employeeid, lastname, jobid, salary,

```
DECODE(Jobid, 'SALES_REP', '1.05*salary',
              'SALES_HEAD' ,'1.12*salary',
                           Salary) "Calculation for Revised Salary"
From Employee;
```

-- Output of above both the queries is same.

EMPLOYEEID	LASTNAME	JOBID	SALARY	Calculation for Revised Salary
1	Demn	IT_PROF	70000	salary
2	Dale	SALES_HEAD	50000	1.12*salary
3	Walton	IT_REP	30000	salary
4	sngal	SALES_REP	40000	1.05*salary
5	ghosala	SALES_REP	30000	1.05*salary

MULTIPLE ROW FUNCTIONS/ GROUP FUNCTIONS

- Group Functions operate on sets of rows to return one result per group.
- These sets may be whole table or table split into groups based on criteria.

Emp_id	Salary
1	5000
2	4000
3	50000
4	14500
5	23000
6	14000

Sum of Salary in Employee table

Sum(salary)
110500

TYPES OF GROUP FUNCTIONS

- **Avg([Distinct/All] n):** Average value of n; ignoring null values.
- **Max([Distinct/All] expr)**: Maximum value of expression; ignoring null values.
- **Min([Distinct/All] expr)**: Minimum value of expression; ignoring null values.
- **Sum([Distinct/All] n)**:Sum values of n; ignoring null values.
- **Count({*| [Distinct/All] expr})** : Number of rows, where expr evaluates to something other than null. Count(*) counts all rows including null and duplicates.
- **STDDEV([Distinct/All] n)**
- **Variance([Distinct/All] n)**

GROUP FUNCTIONS

Guidelines for using Group Functions:

- DISTINCT makes the function consider only non duplicate values; ALL makes it consider every value including duplicates. The default is ALL.
- Functions with the *expr* argument can accept CHAR, VARCHAR2, NUMBER and Date data type.
- All group functions ignore null values by default.

USING GROUP FUNCTIONS

- **Using AVG and SUM Functions :** Can be used only against NUMERIC Data type.

```
SELECT avg(salary) ,sum(salary)
from Employee;
```

AVG(SALARY)	SUM(SALARY)
44000	220000

```
SELECT avg(salary), round(avg(commitionpct),2) avg_comm
from employee;      -- AVG function ignores null values
```

AVG(SALARY)	AVG_COMM
44000	0.35

- **Using MIN and MAX Functions :** Can be used against any data type.

```
select MIN(salary) ,MAX(salary)
from Employee;
```

MIN(SALARY)	MAX(SALARY)
30000	70000

```
SELECT MIN(hiredate), MAX(hiredate)
from employee;
```

MIN(HIREDATE)	MAX(HIREDATE)
01-JAN-01	10-JUN-02

```
SELECT MIN(firstname), MAX(firstname)
from employee;
```

MIN(FIRSTNAME)	MAX(FIRSTNAME)
James	Robin

USING GROUP FUNCTIONS

- **Using the COUNT Function :** Can be used with any data type.

  It has 3 formats:

  - Count(*): Returns the number of rows in a table including duplicate and rows containing null values in any of the columns.
  - Count(expr): Returns the number of non null values in the column specified by *expr.*
  - Count(distinct expr): Returns the number of unique, non null values in the column specified by *expr.*

Examples:

> SELECT count(*) from employee; Output: 5

> SELECT count(*) from employee where firstname like 'J%'; Output: 2

> SELECT count(commitionpct)from employee; Output: 4

:- ignores null values

> SELECT count(jobid) from employee; Output: 5

:- Considers duplicates as well

> SELECT count(distinct jobid) from employee; Output: 4

GROUPING DATA USING GROUP BY CLAUSE

- **Group By** Clause is used to divide the rows in the table into groups. We can then use group functions to return the summary information per group.

- Syntax :

Select [column1,] ***group_function(Column),…***

From table

[where Condition]

[Group by group_by_expression]

[Order by column];

GROUPING DATA USING GROUP BY CLAUSE

Guidelines:

- When group function is present in the select list, user cannot select individual rows as well; unless the individual column appears in the group by clause.
- Using WHERE clause, rows can be excluded before dividing them into groups.
- Column names must be included in the Group By clause. Column alias is not allowed in Group by clause.
- All columns in the SELECT list; that are not in group function must be in a group by clause.

EXCLUDING GROUP RESULTS

- WHERE clause to restrict the rows that you select. In the same way, HAVING clause is used to restrict the groups.

- **Syntax:**

  Select [column1,] ***group_function(Column),…***

  From table

  [where Condition]

  [Group by group_by_expression]

  [Having group_condition]

  [Order by column];

Oracle server performs the following steps:

- Rows are grouped
- Group function is applied to the group
- Groups that match the criteria in the having clause are displayed.

USING GROUP BY AND HAVING CLAUSE

- SELECT departmentid, sum(salary)
  FROM employee
  GROUP BY departmentid;

DEPARTMENTID	SUM(SALARY)
10	120000
20	100000

- SELECT departmentid, jobid, sum(salary)
  FROM employee
  GROUP BY departmentid, jobid;
  -- Grouping data based on multiple columns

DEPARTMENTID	JOBID	SUM(SALARY)
10	SALES_HEAD	50000
10	IT_PROF	70000
20	IT_REP	30000
20	SALES_REP	70000

- SELECT departmentid,max(salary)
  FROM employee
  GROUP BY departmentid
  HAVING max(salary) >45000;

DEPARTMENTID	MAX(SALARY)
10	70000

SEQUENCE OF ALL THE CLAUSES IN SELECT STATEMENT

- **SELECT** jobid, SUM(salary)
  FROM employee
  WHERE jobid NOT LIKE '%PROF%'
  GROUP BY jobid
  HAVING SUM(salary)>=50000
  ORDER BY SUM(salary)

JOBID	SUM(SALARY)
SALES_HEAD	120000
SALES_REP	100000

ILLEGAL QUERIES USING GROUP BY CLAUSE

- SELECT departmentId, sum(salary)

  FROM Employee;

  Error:- ORA 00937 : Not a single group group function

  Note: All columns/expressions in the SELECT list that is not in aggregate function must be in the group by clause.

- SELECT departmentid, AVG(salary)

  FROM employee

  WHERE AVG(salary) > 10000

  GROUP BY departmentid;

  Error:- ORA 00934 : Group function is not allowed here.

  Note: Group function is not allowed in the Where clause and Where clause can not be used to restrict the groups. There is a Having clause for this purpose.

REFERENCE MATERIAL

- Sites:

http://beginner-sql-tutorial.com/oracle-functions.htm

http://www.oracle-dba-online.com/sql/oracle_sql_functions.htm

http://docs.oracle.com/cd/E11882_01/server.112/e26088/functions002.htm

http://psoug.org/reference/group_by.html

http://vpf-web.harvard.edu/applications/ad_hoc/key_functions_in_oracle_sql.pdf

SESSION 3 SUMMARY

PERSISTENT

With this we have come to an end of our third session where we discussed various Conversion functions, Conditional expressions, Multiple row functions and use of Group by clause and Having clause.

At the end of Nugget 3, we see that you are now able to answer following questions:

- Explain “Conversion functions in detail”
- Explain “Various Group functions with examples”
- Explain “GROUP BY clause in detail”
- Explain “HAVING clause in detail”

THANK YOU!!!!!!!!!!!!!!! 

PERSISTENT
Partners in Innovation

Persistent University