**XJTLU Entrepreneur College (Taicang)**

***School of* *AI and Advanced Computing***

**Lab Manual (Lab 9)**

***CPT103TC:***

***Introduction to Database***

Shaheen Khatoon

Second Part of Semester 1

**Academic Year 2021-2022**

**Preface**

This laboratory manual serves as supplemental material for the laboratory class. Majority of its contents were taken from the materials of Oracle’s DB Design and Programming with SQL training.

The table below shows the delivery plan with reference material.

|  |  |  |
| --- | --- | --- |
| **Labs** | **Topics** | **Reference in DB Programming with SQL** |
| **1** | Introduction to Oracle Application Express  SQL Workshop for uploading and running scripts | Lab set-up  Oracle Application Development Foundation (Self-study)  SQL Scripts |
| **2** | Demonstrate and end to end application building process | Oracle Application Development Foundation (Self-study)  Project OracleFlix-demo |
| **3** | Data modeling using SQL Developer and ER Assistant | SQL Developer and ER Assistant Tutorial |
| **4** | SQL DDL command to create database objects and constraints | Demo CompanyScript.SQl; |
| **5** | Managing constraints and SQL DML | Lab handouts |
| **6** | Basic SQL | Lab handouts |
| 7 | SQL Group functions, subqueries, and set operations | Lab handouts |
| 8 | SQL Joins | Lab handouts |
| **9** | **SQL Single Row functions** | **Lab handouts** |
| 10 | Application development | Lab handouts |

Table of Contents

[**Lab 6: Functions** 2](#_Toc95511211)

[SELECT Statement with Single Row Functions 3](#_Toc95511212)

[Number Functions 7](#_Toc95511213)

[ROUND 7](#_Toc95511214)

[TRUNC 7](#_Toc95511215)

[MOD 7](#_Toc95511216)

[DATE Functions 8](#_Toc95511217)

[Conversion Functions 9](#_Toc95511218)

[Date Conversions to Character Data 11](#_Toc95511219)

[Number Conversion to Character Data 13](#_Toc95511220)

[Character Conversion to Number 14](#_Toc95511221)

[Character Conversion to Date 15](#_Toc95511222)

[Combination Example 16](#_Toc95511223)

[Functions Pertaining to Null Values 16](#_Toc95511224)

[NVL Function 17](#_Toc95511225)

[NVL2 Function 17](#_Toc95511226)

[NULLIF Function 18](#_Toc95511227)

[COALESCE Function 19](#_Toc95511228)

[Conditional Expressions 19](#_Toc95511229)

[CASE Expression 19](#_Toc95511230)

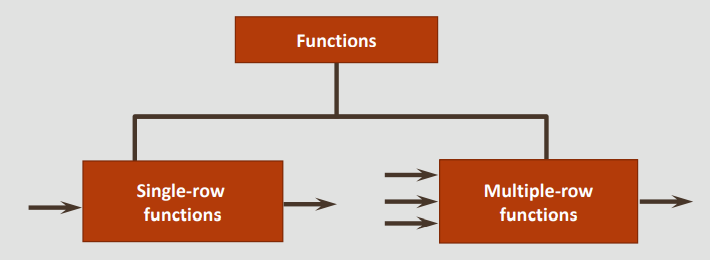
[DECODE Expression 20](#_Toc95511231)

# **Functions**

Oracle has two distinct types of functions:

−Single-Row

−Multiple-Row



Single-row functions operate on single rows only and return one result per row. Single-row functions can be used in the SELECT, WHERE, and ORDER BY clauses

There are different types of single-row functions including character, number, date, and conversion functions

Multiple-row functions can manipulate groups of rows to give one result per group of rows. These functions are also known as group functions discussed earlier. The rows input maybe the whole table or the table split into smaller groups.

Examples of Multiple-Row (Group) functions include:

−MAX: finds the highest value in a group of rows

−MIN: finds the lowest value in a group of rows

−AVG: finds the average value in a group of rows

Single-Row functions are covered in greater depth in the next section. Multiple-Row functions are covered already covered in the previous lab.

## SELECT Statement with Single Row Functions

In SQL, Single-Row functions can be used to:

* Perform calculations such as rounding numbers to a specified decimal place
* Modify individual data items such as converting character values from uppercase to lowercase.

In the following section following objectives are achieved:

* create SELECT Statements using DUAL table
* create SELECT Statements using case manipulation function
* create SELECT Statements using character manipulation functions
* create SELECT Statements using column aliases with functions
* create SELECT Statements using substitution variables
* create SELECT Statements using ROUND, TRUNC, and MOD
* create SELECT Statements using Date functions

| **Example** | **Explanation** |
| --- | --- |
| **DUAL Table:** Dual table is available in oracle**.** The DUAL table is used to create SELECT statements and execute functions not directly related to a specific database table. Queries using the 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 | |
| SELECT (319/29) + 12  FROM DUAL; | DUAL will be used to learn many of the single-row functions.  In this example, the DUAL table is used to execute a SELECT statement that contains a calculation. |
| **Case Manipulation Functions:** Case-manipulation functions are often helpful when you are searching for data and you do not know whether the data you are looking for is in upper or lower case. Case-manipulation functions are used to convert data from the state it is stored in a table to lower, upper or mixed case. These conversions can be used to format the output and can also be used to search for specific strings. | |
| SELECT last\_name  FROM employees  WHERE LOWER(last\_name) = 'abel'; | LOWER(column | expression) converts alpha characters to lower-case. |
| SELECT last\_name  FROM employees  WHERE UPPER(last\_name) = 'ABEL'; | UPPER(column | expression) converts alpha characters to upper-case. |
| SELECT last\_name  FROM employees  WHERE INITCAP(last\_name) = 'Abel'; | INITCAP(column | expression) converts alpha character values to uppercase for the first letter of each word. |
| **Character Manipulation Functions** | |
| SELECT CONCAT('Hello', 'World')  FROM DUAL; | CONCAT: Joins two values together.  The result is ‘HelloWorld’. |
| SELECT CONCAT(first\_name, last\_name)  FROM employees; | The result is ‘EllenAbel’, ‘CurtisDavies’, etc. |
| SELECT SUBSTR('HelloWorld',1,5)  FROM DUAL; | SUBSTR: Extracts a string of a determined length.  extracts a substring of 5 characters from position 1 of 'HelloWorld'. |
| SELECT SUBSTR('HelloWorld', 6)  FROM DUAL; | extracts a substring starting at position 6 of 'HelloWorld' to the end of the string  The result is ‘World’. |
| SELECT SUBSTR(last\_name,1,3)  FROM employees; | extracts a substring of the first 3 characters from employee last names.  The result is ‘Abe’, ‘Dav’, etc. |
| SELECT LENGTH('HelloWorld')  FROM DUAL; | LENGTH: Shows the length of a string as a number value.  The result is 12. |
| SELECT LENGTH(last\_name)  FROM employees; | The result is 4, 6, etc. |
| SELECT INSTR('HelloWorld', 'W')  FROM DUAL; | INSTR: Finds the numeric position of the specified character(s).  The result is 6. |
| SELECT last\_name, INSTR(last\_name, 'a')  FROM employees; | Abel 0  Davies 2  … |
| SELECT LPAD('HelloWorld',15, '-')  FROM DUAL; | LPAD: Pads the left side of a character string, resulting in a right-justified value. LPAD requires 3 arguments: a character string, the total number of characters in the padded string, and the character to pad with.  the string 'HelloWorld' is left padded to 15 characters using the '-' symbol. As the string has a length of 10 characters, 5 '-' symbols are added to the left.  The result is ‘-----HelloWorld’. |
| SELECT LPAD(last\_name, 10,'\*')  FROM employees; | \*\*\*\*\*\*Abel  \*\*\*\*Davies |
| SELECT RPAD('HelloWorld',15, '-')  FROM DUAL; | RPAD: Pads the right-hand side of a character string, resulting in a left-justified value.  The result is ‘HelloWorld-----’. |
| SELECT RPAD(last\_name, 10,'\*')  FROM employees; | Abel\*\*\*\*\*\*  Davies\*\*\*\* |
| SELECT TRIM(LEADING 'a' FROM 'abcba')  FROM DUAL; | TRIM: Removes all specified characters from either the beginning, the end, or both beginning and end of a string.  The result is ‘bcba’. |
| SELECT TRIM(TRAILING 'a' FROM 'abcba')  FROM DUAL; | abcb |
| SELECT TRIM(BOTH 'a' FROM 'abcba')  FROM DUAL; | bcb |
| SELECT REPLACE('JACK and JUE','J','BL')  FROM DUAL; | REPLACE: Replaces a sequence of characters in a string with another set of characters.  The result is ‘BLACK and BLUE’. |
| SELECT REPLACE('JACK and JUE','J')  FROM DUAL; | ACK and UE |
| SELECT REPLACE(last\_name,'a','\*')  FROM employees; | Abel  D\*vies  De H\*\*n |
| **Using Column Aliases With Functions** | |
| SELECT LOWER(last\_name)|| LOWER(SUBSTR(first\_name,1,1))  AS "User Name"  FROM employees; | “User Name” is the alias |
| SELECT LOWER (last\_name)||LOWER(SUBSTR(first\_name,1,1))  FROM f\_staffs; | No alias was used |
| **Substitution Variables** | |
| SELECT first\_name, last\_name, salary, department\_id  FROM employees  WHERE department\_id=:enter\_dept\_id; | When you click Run, a pop-up like the following is displayed by Oracle Application Express:  NOTE: Pop-Up blockers must be disabled, otherwise APEX cannot ask for the variable value, as this is entered via a pop-up. |

## Number Functions

Select and apply the single-row number functions ROUND, TRUNC, and MOD in a SQL query

### ROUND

ROUND ( <*column1*> | <*expression*> , <*decimal\_places*> )

| **Example** | **Result** |
| --- | --- |
| ROUND(45.926) | 46 |
| ROUND(45.926, 0) | 46 |
| ROUND(45.926, 2) | 45.93 |
| ROUND(45.926, -1) | 50 |

### TRUNC

The TRUNC function can be used with both numbers and dates. It is mainly used to terminate the column, expression, or value to a specified number of decimal places.

TRUNC ( <*column1*> | <*expression*> , <*decimal\_places*> )

| **Example** | **Result** |
| --- | --- |
| TRUNC (45.926, 2) | 45.92 |
| TRUNC (45.926, 0) | 45 |
| TRUNC (45.926) | 45 |

### MOD

The MOD function finds the remainder after one value is divided by another value. For example, the MOD of 5 divided by 2 is 1.

MOD can be used to determine whether a value is odd or even. If you divide a value by 2 and there is no remainder, the number must be an even number

For example, if the MOD of x divided by 2 is 0, then x must be an even number

| **Example** | **Explanation** |
| --- | --- |
| SELECT country\_name, MOD(airports,2)  AS "Mod Demo"  FROM wf\_countries; | The "Mod Demo" column will show if number of airports for each country is an odd or even number.  1 means the number is odd, and zero means that it is even |

## DATE Functions

The default display and input format for data is DD-Mon-YYY, for example 02-Feb-2022.

However, the oracle database store date internally wit a numeric format representing the century, year, month, day, hour, minute and second.

| **Example** | **Explanation** |
| --- | --- |
| SELECT SYSDATE  FROM dual; | SYSDATE is a date function that returns the current database server date and time, use the dual table |
| SELECT last\_name, hire\_date + 60  FROM employees; | Adds 60 days to hire\_date. |
| SELECT last\_name, (SYSDATE –  hire\_date)/7  FROM employees; | Displays the number of weeks since the employee was hired. |
| SELECT employee\_id, (end\_date –  start\_date)/365  AS "Tenure in last job"e  FROM job\_history; | Finds the number of days employee held a job, then divides by 365 to display in years. |
| SELECT last\_name, hire\_date  FROM employees  WHERE MONTHS\_BETWEEN  (SYSDATE, hire\_date)>240; | MONTHS\_BETWEEN: takes 2 DATE arguments and returns the number of calendar months between the 2 dates. |
| SELECT ADD\_MONTHS (SYSDATE, 12)  AS "Next Year"  FROM dual; | ADD\_MONTHS: takes 2 arguments, a DATE and a number. Returns a DATE value with the number argument added to the month component of the date. |
| SELECT NEXT\_DAY (SYSDATE, 'Saturday')  AS "Next Saturday"  FROM dual; | NEXT\_DAY: takes 2 arguments, a DATE and a weekday and returns the DATE of the next occurrence of that weekday after the DATE argument. |
| SELECT LAST\_DAY (SYSDATE)  AS "End of the Month"  FROM dual; | LAST\_DAY: takes a DATE argument and returns the DATE of the last day of the month for the DATE argument. |
| SELECT hire\_date,  ROUND (hire\_date, 'Month')  FROM employees  WHERE department\_id=50; | ROUND: returns a DATE rounded to the unit specified by the second argument. |
| SELECT hire\_date,  TRUNC(hire\_date, 'Year')  FROM employees  WHERE department\_id=50; | TRUNC: returns a DATE truncated to the unit specified by the second argument. |

## Conversion Functions

Most programming languages require the programmer to declare the data type of every data object. For the data stored in a database, the SQL programmer defines a data type for every column in the database. Data types will be covered in more detail later in the course, but for now we will be using VARCHAR2, CHAR, NUMBER, and DATE

**VARCHAR2:** Used for character data of variable length, including numbers, dashes, and special characters

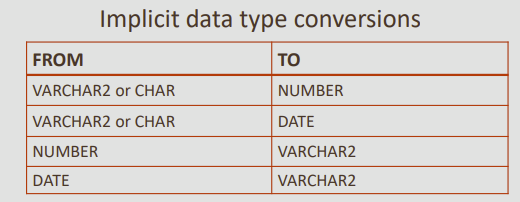
**CHAR:** Used for text and character data of fixed length, including numbers, dashes, and special characters

**NUMBER:** Used to store variable-length numeric data. No dashes, text, or other nonnumeric data are allowed Currency is stored as a number data type

**DATE:** Used for date and time values. Internally, Oracle stores dates as numbers and, by default, DATE information is displayed as DD-Mon-YYYY (for example, 23-Oct-2013)

The Oracle Server can automatically convert VARCHAR2 and CHAR data to NUMBER and DATE data types. It can convert NUMBER and DATE data back to CHARACTER data type . This is known as implicit data conversion

Although this is a convenient feature, it is always best to explicitly make data type conversions to ensure reliability in SQL statements



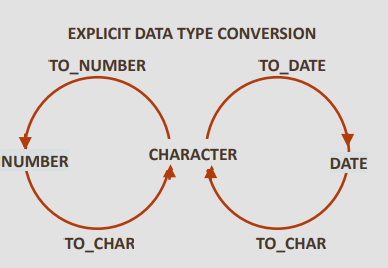
The four data type conversion functions you will learn are:

−To convert date data type to character data type

−To convert number data type to character data type

-To convert character data type to number data type

-To convert character data type to date data types



### Date Conversions to Character Data

It is often desirable to convert a date from its default DD-Mon-YYYY format to another format specified by you

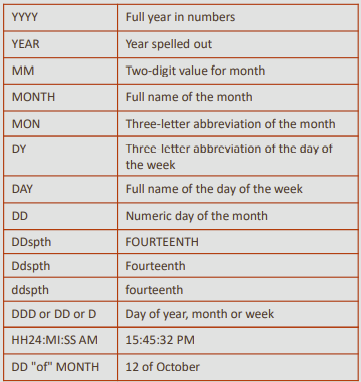
The function to accomplish this task is:

*TO\_CHAR (date column name, 'format model you specify'*

The 'format model' must be enclosed in single quotation marks and is case-sensitive, Separate the date value from the format model with a comma

Any valid date format element can be included. The table below show the different format models that can be used

* Use sp to spell out a number.
* Use th to have the number appear as an ordinal. (1st, 2nd, 3rd, and so on).
* Use an fm element to remove padded blanks or remove leading zeroes from the output.



| **Examples of output using different format models:** | **Output** |
| --- | --- |
| SELECT TO\_CHAR(hire\_date, 'Month dd, YYYY')  FROM employees; | June 07, 1994 |
| SELECT TO\_CHAR(hire\_date, 'fmMonth dd, YYYY')  FROM employees; | June 7, 1994 |
| SELECT TO\_CHAR(hire\_date, 'fmMonth ddth, YYYY')  FROM employees; | adds "th" etc. to the display the day as an ordinal number  June 7th, 1994  January 3rd, 1990 |
| SELECT TO\_CHAR(hire\_date, 'fmDay ddth Mon, YYYY')  FROM employees; | Tuesday 7th Jun, 1994 |
| SELECT TO\_CHAR(hire\_date, 'fmDay ddthsp Mon, YYYY')  FROM employees; | adds "SP to the day format to spell out the day portion of the date  Tuesday, seventh Jun, 1994 |
| SELECT TO\_CHAR(hire\_date, 'fmDay, ddthsp "of" Month, Year')  FROM employees; | displays the year spelled out, and the text "of" between the day and month. Note that double quotes are required around the text literal to be included.  Tuesday, seventh of June, Nineteen Ninety-Four |
| **Examples of output using different format models for time:** |  |
| SELECT TO\_CHAR(SYSDATE, 'hh:mm')  FROM dual; | 02:07 |
| SELECT TO\_CHAR(SYSDATE, 'hh:mm pm')  FROM dual; | 02:07 am |
| SELECT TO\_CHAR(SYSDATE, 'hh:mm:ss pm')  FROM dual; | 02:07:23 am |

### Number Conversion to Character Data

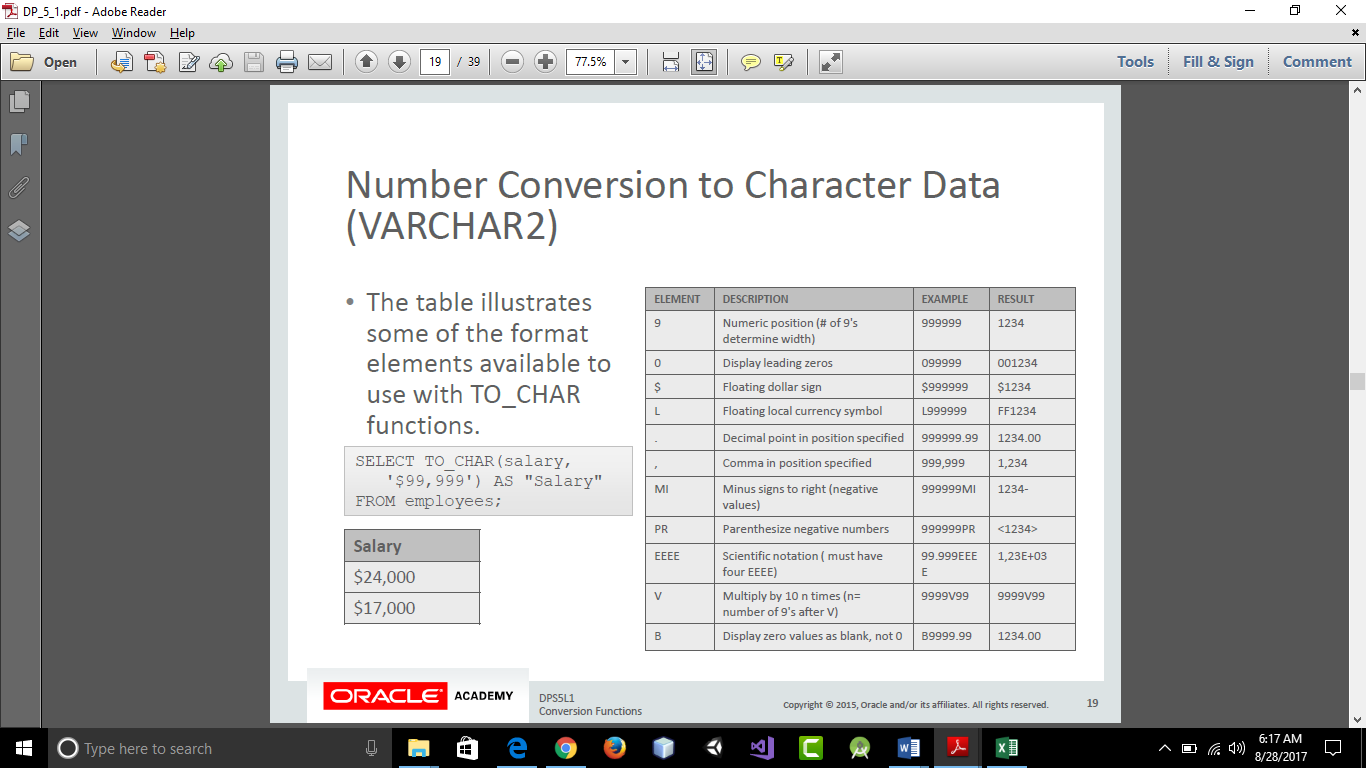
Numbers stored in the database have no formatting

• This means that they have no currency signs/symbols, commas, decimals, or other formatting

• To add formatting, you first need to convert the number to a character format

Syntax: *TO\_CHAR(number, 'format model')*

The table illustrates some of the format elements available to use with TO\_CHAR functions



| **Example** | **Output** |
| --- | --- |
| SELECT TO\_CHAR(salary,  '$99,999') AS "Salary"  FROM employees;  **NOTE:** if the number of "9"s in the format model is less than the number of digits in the number, ##### is displayed, for example:  SELECT TO\_CHAR(24000, '$9,999')  AS "Salary"  FROM employees;  would return ####### as the format model has four "9"s, but the number supplied has five digits. |  |
| Can you identify the format models used to produce the following  output?   1. $3000.00 2. 4,500 3. 9,000.00 4. 0004422 |  |
| 1. SELECT TO\_CHAR(3000, '$9999.99')   FROM dual; | $3000.00 |
| 1. SELECT TO\_CHAR(4500, '99,999')   FROM dual; | 4,500 |
| 1. SELECT TO\_CHAR(9000, '99,999.99')   FROM dual; | 9,000.00 |
| 1. SELECT TO\_CHAR(4422, '0009999')   FROM dual; | 0004422 |

### 

### Character Conversion to Number

It is often desirable to convert a character string to a number. The function for this conversion is:

TO\_NUMBER(character string, 'format model')

The format model is optional, but should be included if the character string being converted contains any characters other than numbers

You cannot reliably perform calculations with character data

| **Example** | **Output** |
| --- | --- |
| SELECT TO\_NUMBER('5,320', '9,999') AS "Number"  FROM dual; | 5320 |
| SELECT last\_name, TO\_NUMBER(bonus, '9999') AS "Bonus"  FROM employees  WHERE department\_id = 80; | Zlotkey 1500  Abel 1700  Taylor 1250  **Note: Try removing one ‘9’ and see the error since bonus has 4 characters.** |

### Character Conversion to Date

To convert a character string to a date format, use:

TO\_DATE('character string', 'format model')

• This conversion takes a non-date value character string such as "November 3, 2001" and converts it to a date value

• The format model tells the server what the character string "looks like":

TO\_DATE('November 3, 2001', 'Month dd, yyyy')

−will return 03-Nov-2001

| **Example** | **Output** |
| --- | --- |
| SELECT TO\_DATE('May10,1989', 'fxMonDD,YYYY') AS "Convert"  FROM DUAL;  the fx (format exact) modifier specifies exact matching for the character argument and the date format model  In the example, note that "May10" has no space between ''May" and "10"  The fx format model matches the character argument as it also has no space between "Mon" and "DD" | 05/10/1989 |

The fx modifier rules are:

* Punctuation and quoted text in the character argument must match the corresponding parts of the format model exactly (except for case)
* The character argument cannot have extra blanks. Without fx, the Oracle Server ignores extra blanks
* Numeric data in the character argument must have the same number of digits as the corresponding element in the format model. Without fx, numbers in the character argument can omit leading zeros

| **Example** | **Output** |
| --- | --- |
| SELECT TO\_DATE('Sep 07, 1965', 'fxMon dd, YYYY') AS "Date"  FROM dual; | 07/Sep/1965 |
| SELECT TO\_DATE('July312004', 'fxMonthDDYYYY') AS "Date"  FROM DUAL; | 31/Jul/2004 |
| SELECT TO\_DATE('June 19, 1990','fxMonth dd, YYYY') AS "Date"  FROM DUAL; | 19/Jun/1990 |

### Combination Example

| **Example** | **Explanation** |
| --- | --- |
| SELECT TO\_CHAR(NEXT\_DAY(ADD\_MONTHS(hire\_date, 6), 'FRIDAY'), 'fmDay, Month DDth, YYYY')  AS "Next Evaluation"  FROM employees  WHERE employee\_id=100; | Friday, December 18th, 1987  Step 1: The hire date is going to have six months added to it.  Step 2: The first Friday following the day returned at Step 1 will be identified. |

### 

## Functions Pertaining to Null Values

Null is the value that is unavailable, unassigned, unknown, or inapplicable

As a result, we cannot test to see if it is the same as another value, because we do not know what value it has

It isn't equal to anything, not even zero! . But just because it really isn't anything doesn't mean that it is not important.

Oracle has four general functions that pertain to the use of null values:

−NVL

−NVL2

−NULLIF

−COALESCE

### NVL Function

* The NVL function converts a null value to a known value of a fixed data type, either date, character, or number.
* The data types of the null value column and the new value must be the same.

*NVL (expression 1 value that may contain a null, expression 2 value to substitute for null)*

| **Example** | **Explanation** |
| --- | --- |
| SELECT country\_name, NVL(internet\_extension, 'None')  AS "Internet extn"  FROM wf\_countries  WHERE location = 'Southern Africa'  ORDER BY internet\_extension DESC; | Null values are replaced with the text 'None'. |
| SELECT last\_name, NVL(commission\_pct, 0)\*250  AS "Commission"  FROM employees  WHERE department\_id IN(80,90); | The commission\_pct column in the employees table contains null values.  The NVL function is used to change the null to zero before arithmetic calculations. |

### NVL2 Function

* The NVL2 function evaluates an expression with three values.
* If the first value is not null, then the NVL2 function returns the second expression.
* If the first value is null, then the third expression is returned.
* The values in expression 1 can have any data type.
* Expression 2 and expression 3 can have any data type except LONG.
* The data type of the returned value is always the same as the data type of expression 2, unless expression 2 is character data, in which case the returned type is VARCHAR2.

NVL2 (expression 1 value that may contain a null, expression 2 value to return if expression 1 is not null, expression 3 value to replace if expression 1 is null)

An easy way to remember NVL2 is to think, "if expression 1 has a value, substitute expression 2; if expression 1 is null, substitute expression 3."

| **Example** | **Explanation** |
| --- | --- |
| SELECT last\_name, salary,  NVL2(commission\_pct, salary + (salary \* commission\_pct), salary) AS income  FROM employees  WHERE department\_id IN(80,90); | The NVL2 function shown uses number data types for expressions 1, 2 and 3.  NVL2 checks if expression 1 (commission\_pct) has a value. If it has a value, expression 2 is returned (salary + (salary \* commission\_pct). If expression 1 is NULL, expression 3 is returned (salary). |

### NULLIF Function

* The NULLIF function compares two expressions.
* If they are equal, the function returns null.
* If they are not equal, the function returns the first expression. Expression 2 and expression 3 can have any data type except LONG.
* The NULLIF function is

NULLIF(expression 1, expression 2)

| **Example** | **Explanation** |
| --- | --- |
| SELECT first\_name, LENGTH(first\_name) AS "Length FN", last\_name,  LENGTH(last\_name) AS "Length LN", NULLIF(LENGTH(first\_name),  LENGTH(last\_name)) AS "Compare Them"  FROM employees; | NULLIF compares the length of employees first and last names.  If the length of both names are the same, NULLIF returns NULL (as in row 2 Curtis Davies), otherwise expression 1 LENGTH of first\_name is returned.  NULLIF functions are often used after doing data migration projects to test if the data in the target system is the same as in the original source systems. So NULLIF is used to look for exceptions, not matches – normally nulls as a result from NULLIF is good, as you would want the data in the source and target systems to be exactly the same |

### COALESCE Function

* The COALESCE function is an extension of the NVL function, except COALESCE can take multiple values.
* The word "coalesce" literally means "to come together" and that is what happens.
* If the first expression is null, the function continues down the line until a not-null expression is found.
* Of course, if the first expression has a value, the function returns the first expression and the function stops.
* The COALESCE function is:

COALESCE (expression 1, expression 2, ...expression n)

| **Example** | **Explanation** |
| --- | --- |
| SELECT last\_name,  COALESCE(commission\_pct, salary, 10)  AS "Comm"  FROM employees  ORDER BY commission\_pct; | • Examine the SELECT statement from the employees table shown. If an employee has a value ( not NULL)for commission\_pct, this is returned, otherwise if salary has a value, return salary . If an employees commission\_pct and salary are NULL, return the number 10 |

# Conditional Expressions

## CASE Expression

* The CASE expression basically does the work of an IF-THEN-ELSE statement.
* Data types of the CASE, WHEN, and ELSE expressions must be the same.

The syntax for a CASE expression is:

CASE expr WHEN comparison\_expr1 THEN return\_expr1

[WHEN comparison\_expr2 THEN return\_expr2

WHEN comparison\_exprn THEN return\_exprn

ELSE else\_expr]

END

| **Example** | **Explanation** |
| --- | --- |
| SELECT last\_name,  CASE department\_id  WHEN 90 THEN 'Management'  WHEN 80 THEN 'Sales'  WHEN 60 THEN 'It'  ELSE 'Other dept.'  END AS "Department"  FROM employees; | The query checks the department\_id.  IF it is 90, then return 'Management'  IF it is 80, then return 'Sales'  IF it is 60, then return 'It'  ELSE return 'Other dept.' |

## DECODE Expression

* The DECODE function evaluates an expression in a similar way to the IF-THEN-ELSE logic.
* DECODE compares an expression to each of the search values.
* The syntax for DECODE is:

DECODE(columnl|expression, search1, result1 [, search2, result2,...,] [, default])

* If the default value is omitted, a null value is returned where a search value does not match any of the values

| **Example** | **Explanation** |
| --- | --- |
| SELECT last\_name,  DECODE(department\_id,  90, 'Management',  80, 'Sales',  60, 'It',  'Other dept.')  AS "Department"  FROM employees; | This query returns exactly the same results as the previous CASE example, but using different syntax. |