5

# Using Conversion Functions and Conditional Expressions



ORACLE

# Objectives

After completing this lesson, you should be able to do the following:

- Describe the various types of conversion functions that are available in SQL
- Use the `TO_CHAR`, `TO_NUMBER`, and `TO_DATE` conversion functions
- Apply conditional expressions in a `SELECT` statement





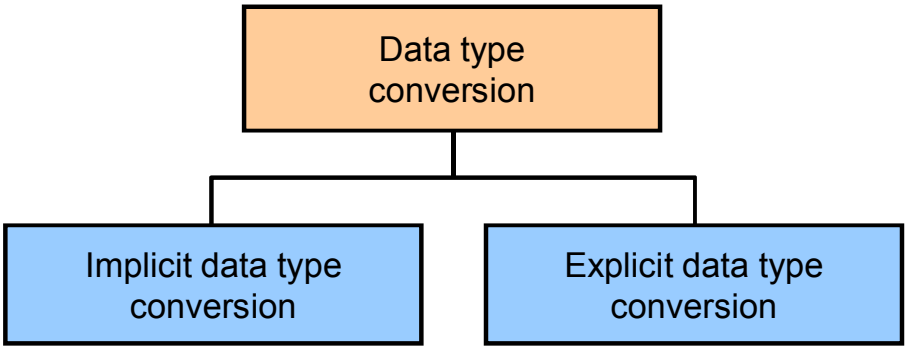


This lesson focuses on functions that convert data from one type to another (for example, conversion from character data to numeric data) and discusses the conditional expressions in SQL `SELECT` statements.

# Lesson Agenda

- Implicit and explicit data type conversion
- `TO_CHAR`, `TO_DATE`, `TO_NUMBER` functions
- General functions:
  - `NVL`
  - `NVL2`
  - `NULLIF`
  - `COALESCE`
- Conditional expressions:
  - `CASE`
  - Searched `CASE`
  - `DECODE`

# Conversion Functions






ORACLE



In addition to Oracle data types, columns of tables in an Oracle Database can be defined by using the American National Standards Institute (ANSI), DB2, and SQL/DS data types. However, the Oracle server internally converts such data types to Oracle data types.

In some cases, the Oracle server receives data of one data type where it expects data of a different data type. When this happens, the Oracle server can automatically convert the data to the expected data type. This data type conversion can be done *implicitly* by the Oracle server or *explicitly* by the user.

Implicit data type conversions work according to the rules explained in the following slides.

Explicit data type conversions are performed by using the conversion functions. Conversion functions convert a value from one data type to another. Generally, the form of the function names follows the convention `data type TO data type`. The first data type is the input data type and the second data type is the output.

**Note:** Although implicit data type conversion is available, it is recommended that you do the explicit data type conversion to ensure the reliability of your SQL statements.

# Implicit Data Type Conversion

In expressions, the Oracle server can automatically convert the following:

| From | To |
|---|---|
| VARCHAR2 or CHAR | NUMBER |
| VARCHAR2 or CHAR | DATE |



Oracle server can automatically perform data type conversion in an expression. For example, the expression `hire_date > '01-JAN-90'` results in the implicit conversion from the string `'01-JAN-90'` to a date. Therefore, a `VARCHAR2` or `CHAR` value can be implicitly converted to a number or date data type in an expression.

**Note:** `CHAR` to `NUMBER` conversions succeed only if the character string represents a valid number.

# Implicit Data Type Conversion

For expression evaluation, the Oracle server can automatically convert the following:

| From | To |
|---|---|
| NUMBER | VARCHAR2 or CHAR |
| DATE | VARCHAR2 or CHAR |

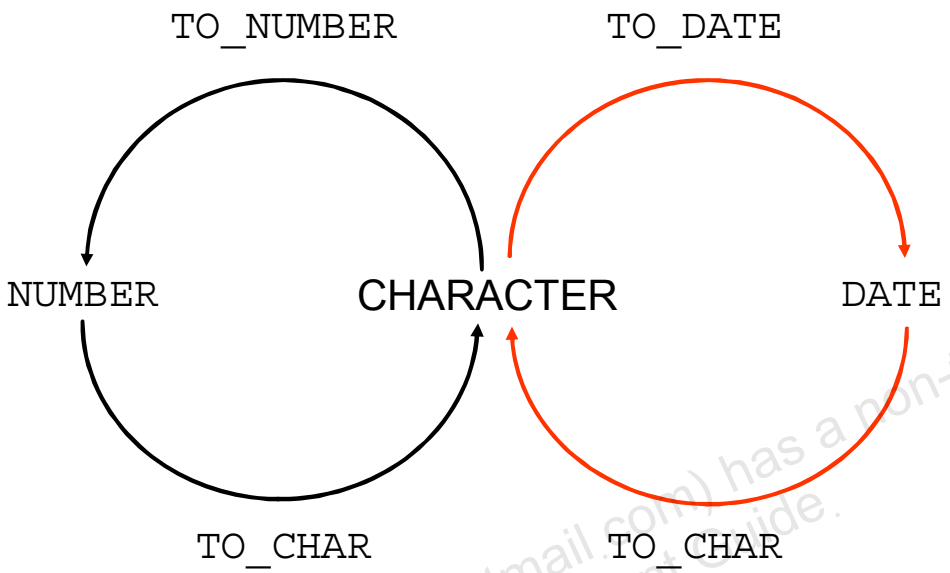






In general, the Oracle server uses the rule for expressions when a data type conversion is needed. For example, the expression `job_id = 2` results in the implicit conversion of the number `2` to the string “2” because `job_id` is a `VARCHAR(2)` column.

# Explicit Data Type Conversion










SQL provides three functions to convert a value from one data type to another:

| Function | Purpose |
|---|---|
| `TO_CHAR(number\|date [, fmt [, nlsparams] ] )` | Converts a number or date value to a `VARCHAR2` character string with the format model *fmt*<br>**Number conversion:** The `nlsparams` parameter specifies the following characters, which are returned by number format elements:<br>• Decimal character<br>• Group separator<br>• Local currency symbol<br>• International currency symbol<br>If `nlsparams` or any other parameter is omitted, this function uses the default parameter values for the session. |

| Function | Purpose |
|---|---|
| TO_NUMBER(*char[,fmt[, nlsparams]]*) | Converts a character string containing digits to a number in the format specified by the optional format model *fmt*.<br>The nlsparams parameter has the same purpose in this function as in the TO_CHAR function for number conversion. |
| TO_DATE(*char[,fmt*[,*nlsparams*]]) | Converts a character string representing a date to a date value according to *fmt* that is specified. If *fmt* is omitted, the format is DD-MON-YY.<br>The nlsparams parameter has the same purpose in this function as in the TO_CHAR function for date conversion. |

**Note:** The list of functions mentioned in this lesson includes only some of the available conversion functions.

For more information, see the "Conversion Functions" section in *Oracle Database SQL Language Reference* for 12*c* database.

# Lesson Agenda

- Implicit and explicit data type conversion
- TO_CHAR, TO_DATE, TO_NUMBER functions
- General functions:
  - NVL
  - NVL2
  - NULLIF
  - COALESCE
- Conditional expressions:
  - CASE
  - Searched CASE
  - DECODE

# Using the TO_CHAR Function with Dates

```
TO_CHAR(date[,'format_model'])
```

The format model:

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







TO_CHAR converts a datetime data type to a value of VARCHAR2 data type in the format specified by the *format_model*. A format model is a character literal that describes the format of datetime stored in a character string. For example, the datetime format model for the string '11-Nov-2000' is 'DD-Mon-YYYY'. You can use the TO_CHAR function to convert a date from its default format to the one that you specify.

**Guidelines**

- The format model must be enclosed within single quotation marks and is case-sensitive.
- The format model can include any valid date format element. But be sure to separate the date value from the format model with a comma.
- The names of days and months in the output are automatically padded with blanks.
- To remove padded blanks or to suppress leading zeros, use the fill mode *fm* element.

```
SELECT employee_id, TO_CHAR(hire_date, 'MM/YY') Month_Hired
FROM   employees
WHERE  last_name = 'Higgins';
```

# Elements of the Date Format Model

| Element | Result |
|---|---|
| YYYY | Full year in numbers |
| YEAR | Year spelled out (in English) |
| MM | Two-digit value for the month |
| MONTH | Full name of the month |
| MON | Three-letter abbreviation of the month |
| DY | Three-letter abbreviation of the day of the week |
| DAY | Full name of the day of the week |
| DD | Numeric day of the month |

# Elements of the Date Format Model

- Time elements format the time portion of the date:

| | |
|---|---|
| `HH24:MI:SS AM` | `15:45:32 PM` |

- Add character strings by enclosing them within double quotation marks:

| | |
|---|---|
| `DD "of" MONTH` | `12 of OCTOBER` |

- Number suffixes spell out numbers:

| | |
|---|---|
| `ddspth` | `fourteenth` |



Use the formats that are listed in the following tables to display time information and literals, and to change numerals to spelled numbers.

| Element | Description |
|---|---|
| AM or PM | Meridian indicator |
| A.M. or P.M. | Meridian indicator with periods |
| HH or HH12 | 12 hour format |
| HH24 | 24 hour format |
| MI | Minute (0–59) |
| SS | Second (0–59) |
| SSSSS | Seconds past midnight (0–86399) |

### Other Formats

| Element | Description |
|---|---|
| / . , | Punctuation is reproduced in the result. |
| “of the” | Quoted string is reproduced in the result. |

### Specifying Suffixes to Influence Number Display

| Element | Description |
|---|---|
| TH | Ordinal number (for example, DDTH for 4TH) |
| SP | Spelled-out number (for example, DDSP for FOUR) |
| SPTH or THSP | Spelled-out ordinal numbers (for example, DDSPTH for FOURTH) |

# Using the TO_CHAR Function with Dates

```
SELECT last_name,
       TO_CHAR(hire_date, 'fmDD Month YYYY')
       AS HIREDATE
FROM   employees;
```

| | LAST_NAME | HIREDATE |
|---|---|---|
| 1 | King | 17 June 2003 |
| 2 | Kochhar | 21 September 2005 |
| 3 | De Haan | 13 January 2001 |
| 4 | Hunold | 3 January 2006 |
| 5 | Ernst | 21 May 2007 |
| 6 | Lorentz | 7 February 2007 |
| 7 | Mourgos | 16 November 2007 |
| 8 | Rajs | 17 October 2003 |

...



The SQL statement in the slide displays the last names and hire dates for all the employees. The hire date appears as 17 June 2003.

**Example**

Modify the example in the slide to display the dates in a format that appears as “Seventeenth of June 2003 12:00:00 AM.”

```
SELECT  last_name,
 TO_CHAR(hire_date,
       'fmDdspth "of" Month YYYY fmHH:MI:SS AM')
 HIREDATE
FROM    employees;
```

Notice that the month follows the format model specified; in other words, the first letter is capitalized and the rest are in lowercase.

# Using the TO_CHAR Function with Numbers

```
TO_CHAR(number[, 'format_model'])
```

These are some of the format elements that you can use with the TO_CHAR function to display a number value as a character:

| Element | Result |
|---|---|
| 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 comma as a thousands indicator |



When working with number values, such as character strings, you should convert those numbers to the character data type using the TO_CHAR function, which translates a value of NUMBER data type to VARCHAR2 data type. This technique is especially useful with concatenation.

### Number Format Elements

If you are converting a number to the character data type, you can use the following format elements:

| Element | Description | Example | Result |
|---|---|---|---|
| 9 | Numeric position (number of 9s determine display width) | 999999 | 1234 |
| 0 | Display leading zeros | 099999 | 001234 |
| $ | Floating dollar sign | $999999 | $1234 |
| L | Floating local currency symbol | L999999 | FF1234 |
| D | Returns the decimal character in the specified position. The default is a period (.). | 9999D99 | 1234.00 |
| . | Decimal point in position specified | 999999.99 | 1234.00 |
| G | Returns the group separator in the specified position. You can specify multiple group separators in a number format model. | 9G999 | 1,234 |
| , | Comma in position specified | 999,999 | 1,234 |
| MI | Minus signs to right (negative values) | 999999MI | 1234- |
| PR | Parenthesize negative numbers | 999999PR | <1234> |
| EEEE | Scientific notation (format must specify four Es) | 99.999EEEE | 1.234E+03 |
| U | Returns in the specified position the “Euro” (or other) dual currency | U9999 | €1234 |
| V | Multiply by 10 *n* times (*n* = number of 9s after V) | 9999V99 | 123400 |
| S | Returns the negative or positive value | S9999 | -1234 or +1234 |
| B | Display zero values as blank, not 0 | B9999.99 | 1234.00 |

# Using the TO_CHAR Function with Numbers

```
SELECT TO_CHAR(salary, '$99,999.00') SALARY
FROM   employees
WHERE  last_name = 'Ernst';
```

| | SALARY |
|---|---|
| 1 | $6,000.00 |





ORACLE



- The Oracle server displays a string of number signs (#) in place of a whole number whose digits exceed the number of digits provided in the format model.
- The Oracle server rounds the stored decimal value to the number of decimal places provided in the format model.

# Using the TO_NUMBER and TO_DATE Functions

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

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

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

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

- These functions have an fx modifier. This modifier specifies the exact match for the character argument and date format model of a TO_DATE function.



You may want to convert a character string to either a number or a date. To accomplish this task, use the TO_NUMBER or TO_DATE functions. The format model that you select is based on the previously demonstrated format elements.

The fx modifier specifies the exact match for the character argument and date format model of a TO_DATE function:

- Punctuation and quoted text in the character argument must exactly match (except for case) the corresponding parts of the format model.
- 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, the numbers in the character argument can omit leading zeros.

**Example**

Display the name and hire date for all employees who started on May 24, 2007. There are two spaces after the month *May* and before the number *24* in the following example. Because the `fx` modifier is used, an exact match is required and the spaces after the word *May* are not recognized:

```
SELECT last_name, hire_date
FROM   employees
WHERE  hire_date = TO_DATE('May  24, 2007', 'fxMonth DD, YYYY');
```

The resulting error output looks like this:

```
ORA-01858: a non-numeric character was found where a numeric was expected
01858. 00000 - "a non-numeric character was found where a numeric was expected"
*Cause:    The input data to be converted using a date format model was
           incorrect. The input data did not contain a number where a number was
           required by the format model.
*Action:   Fix the input data or the date format model to make sure the
           elements match in number and type. Then retry the operation.
```

To see the output, correct the query by deleting the extra space between 'May' and '24'.

```
SELECT last_name, hire_date
FROM   employees
WHERE  hire_date = TO_DATE('May 24, 2007', 'fxMonth DD, YYYY');
```

# Using TO_CHAR and TO_DATE Functions with the RR Date Format

To find employees hired before 1990, use the RR date format, which produces the same results whether the command is run in 1999 or now:

```
SELECT last_name, TO_CHAR(hire_date, 'DD-Mon-YYYY')
FROM  employees
WHERE hire_date < TO_DATE('01-Jan-90','DD-Mon-RR');
```

| | LAST_NAME | TO_CHAR(HIRE_DATE,'DD-MON-YYYY') |
|---|---|---|
| 1 | Popp | 03-Feb-1989 |





ORACLE



To find employees who were hired before 1990, the RR format can be used. Because the current year is greater than 1999, the RR format interprets the year portion of the date from 1950 to 1999.

Alternatively, the following command, results in no rows being selected because the YY format interprets the year portion of the date in the current century (2090).

```
SELECT last_name, TO_CHAR(hire_date, 'DD-Mon-yyyy')
FROM   employees
WHERE  TO_DATE(hire_date, 'DD-Mon-yy') < '01-Jan-90';
```

Notice that no rows are retrieved from the preceding query.

# Lesson Agenda

- Implicit and explicit data type conversion
- TO_CHAR, TO_DATE, TO_NUMBER functions
- General functions:
  - NVL
  - NVL2
  - NULLIF
  - COALESCE
- Conditional expressions:
  - CASE
  - Searched CASE
  - DECODE

# General Functions

The following functions work with any data type and pertain to using nulls:

- `NVL (expr1, expr2)`
- `NVL2 (expr1, expr2, expr3)`
- `NULLIF (expr1, expr2)`
- `COALESCE (expr1, expr2, ..., exprn)`







These functions work with any data type and pertain to the use of null values in the expression list.

| Function | Description |
|---|---|
| `NVL` | Converts a null value to an actual value |
| `NVL2` | If `expr1` is not null, `NVL2` returns `expr2`. If `expr1` is null, `NVL2` returns `expr3`. The argument `expr1` can have any data type. |
| `NULLIF` | Compares two expressions and returns null if they are equal; returns the first expression if they are not equal |
| `COALESCE` | Returns the first non-null expression in the expression list |

**Note:** For more information about the hundreds of functions available, see the “Functions” section in *Oracle Database SQL Language Reference* for 12*c* database.

# NVL Function

Converts a null value to an actual value:

- Data types that can be used are date, character, and number.
- Data types must match:
  - `NVL(commission_pct,0)`
  - `NVL(hire_date,'01-JAN-97')`
  - `NVL(job_id,'No Job Yet')`







To convert a null value to an actual value, use the NVL function.

**Syntax**

`NVL (expr1, expr2)`

In the syntax:

- *expr1* is the source value or expression that may contain a null
- *expr2* is the target value for converting the null

You can use the NVL function with any data type, but the return value is always the same as the data type of *expr1*.

**NVL Conversions for Various Data Types**

| Data Type | Conversion Example |
|---|---|
| NUMBER | NVL(*number_column*,9) |
| DATE | NVL(*date_column*, '01-JAN-95') |
| CHAR or VARCHAR2 | NVL(*character_column*, 'Unavailable') |

# Using the NVL Function

```
SELECT last_name, salary, NVL(commission_pct, 0),
   (salary*12) + (salary*12*NVL(commission_pct, 0)) AN_SAL
FROM employees;
```

| | LAST_NAME | SALARY | NVL(COMMISSION_PCT,0) | AN_SAL |
|---|---|---|---|---|
| 1 | King | 24000 | 0 | 288000 |
| 2 | Kochhar | 17000 | 0 | 204000 |
| 3 | De Haan | 17000 | 0 | 204000 |
| 4 | Hunold | 9000 | 0 | 108000 |
| 5 | Ernst | 6000 | 0 | 72000 |
| 6 | Lorentz | 4200 | 0 | 50400 |
| 7 | Mourgos | 5800 | 0 | 69600 |
| 8 | Rajs | 3500 | 0 | 42000 |
| 9 | Davies | 3100 | 0 | 37200 |
| 10 | Matos | 2600 | 0 | 31200 |

...







To calculate the annual compensation of all employees, you need to multiply the monthly salary by 12 and then add the commission percentage to the result:

```
SELECT last_name, salary, commission_pct,
 (salary*12) + (salary*12*commission_pct) AN_SAL
FROM   employees;
```

Notice that the annual compensation is calculated for only those employees who earn a commission. If any column value in an expression is null, the result is null. To calculate values for all employees, you must convert the null value to a number before applying the arithmetic operator. In the example in the slide, the NVL function is used to convert null values to zero.

# Using the NVL2 Function

```
SELECT last_name,  salary, commission_pct,
       NVL2(commission_pct,
            'SAL+COMM', 'SAL') income
FROM   employees WHERE department_id IN (50, 80);
```

| | LAST_NAME | SALARY | COMMISSION_PCT | INCOME |
|---|---|---|---|---|
| 1 | Mourgos | 5800 | (null) | SAL |
| 2 | Rajs | 3500 | (null) | SAL |
| 3 | Davies | 3100 | (null) | SAL |
| 4 | Matos | 2600 | (null) | SAL |
| 5 | Vargas | 2500 | (null) | SAL |
| 6 | Zlotkey | 10500 | 0.2 | SAL+COMM |
| 7 | Abel | 11000 | 0.3 | SAL+COMM |
| 8 | Taylor | 8600 | 0.2 | SAL+COMM |







The NVL2 function examines the first expression. If the first expression is not null, the NVL2 function returns the second expression. If the first expression is null, the third expression is returned.

**Syntax**

```
NVL2(expr1, expr2, expr3)
```

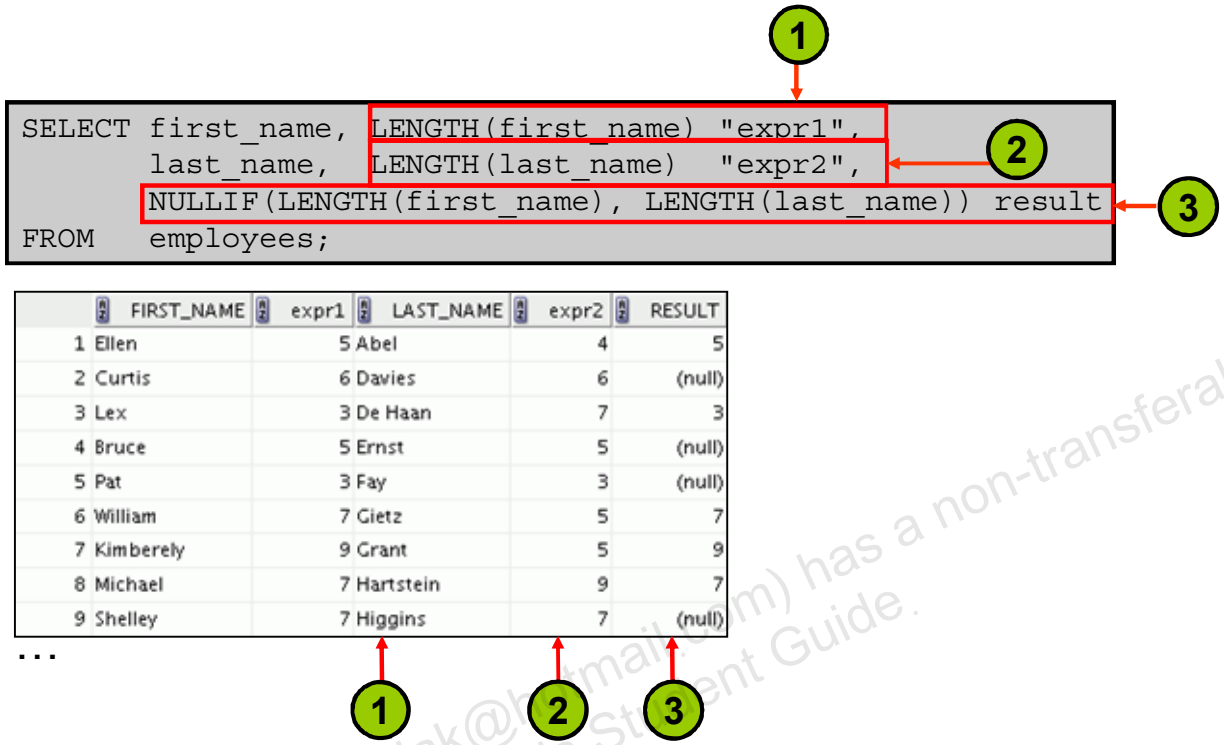
In the syntax:

- *expr1* is the source value or expression that may contain a null
- *expr2* is the value that is returned if *expr1* is not null
- *expr3* is the value that is returned if *expr1* is null

In the example shown in the slide, the COMMISSION_PCT column is examined. If a value is detected, the text literal value of SAL+COMM is returned. If the COMMISSION_PCT column contains a null value, the text literal value of SAL is returned.

**Note:** The argument *expr1* can be any data type, but *expr2* and *expr3* should be the same data type.

# Using the NULLIF Function






The NULLIF function compares two expressions.

**Syntax**

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

In the syntax:

- NULLIF compares *expr1* and *expr2*. If they are equal, the function returns null. If they are not, the function returns *expr1*. However, you cannot specify the literal NULL for *expr1*.

In the example shown in the slide, the length of the first name in the EMPLOYEES table is compared with the length of the last name in the EMPLOYEES table. When the lengths of the names are equal, a null value is displayed. When the lengths of the names are not equal, the length of the first name is displayed.

# Using the COALESCE Function

- The advantage of the COALESCE function over the NVL function is that the COALESCE function can take multiple alternative values.
- If the first expression is not null, the COALESCE function returns that expression; otherwise, it does a COALESCE of the remaining expressions.







The COALESCE function returns the first non-null expression in the list.

**Syntax**

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

In the syntax:

- *expr1* returns this expression if it is not null
- *expr2* returns this expression if the first expression is null and this expression is not null
- *exprn* returns this expression if the preceding expressions are null

Note that all expressions must be of the same data type.

# Using the COALESCE Function

```
SELECT last_name, salary, commission_pct,
COALESCE((salary+(commission_pct*salary)), salary+2000)"New Salary"
FROM   employees;
```

| | LAST_NAME | SALARY | COMMISSION_PCT | New Salary |
|---|---|---|---|---|
| 1 | King | 24000 | (null) | 26000 |
| 2 | Kochhar | 17000 | (null) | 19000 |
| 3 | De Haan | 17000 | (null) | 19000 |
| 4 | Hunold | 9000 | (null) | 11000 |
| 5 | Ernst | 6000 | (null) | 8000 |
| 6 | Lorentz | 4200 | (null) | 6200 |
| 7 | Mourgos | 5800 | (null) | 7800 |
| 8 | Rajs | 3500 | (null) | 5500 |
| 9 | Davies | 3100 | (null) | 5100 |
| 10 | Matos | 2600 | (null) | 4600 |
| 11 | Vargas | 2500 | (null) | 4500 |
| 12 | Zlotkey | 10500 | 0.2 | 12600 |
| 13 | Abel | 11000 | 0.3 | 14300 |
| 14 | Taylor | 8600 | 0.2 | 10320 |
| 15 | Grant | 7000 | 0.15 | 8050 |
| 16 | Whalen | 4400 | (null) | 6400 |
| 17 | Hartstein | 13000 | (null) | 15000 |
| 18 | Fay | 6000 | (null) | 8000 |
| 19 | Higgins | 12008 | (null) | 14008 |
| 20 | Gietz | 8300 | (null) | 10300 |



In the example shown in the slide, for the employees who do not get any commission, your organization wants to give a salary increment of $2,000 and for employees who get commission, the query should compute the new salary that is equal to the existing salary added to the commission amount.

**Note:** Examine the output. For employees who do not get any commission, the New Salary column shows the salary incremented by $2,000 and for employees who get commission, the New Salary column shows the computed commission amount added to the salary.

# Lesson Agenda

- Implicit and explicit data type conversion
- TO_CHAR, TO_DATE, TO_NUMBER functions
- General functions:
  - NVL
  - NVL2
  - NULLIF
  - COALESCE
- Conditional expressions:
  - CASE
  - Searched CASE
  - DECODE

# Conditional Expressions

- Provide the use of the `IF-THEN-ELSE` logic within a SQL statement
- Use the following methods:
  - `CASE` expression
  - Searched `CASE` expression
  - `DECODE` function







The two methods that are used to implement conditional processing (`IF-THEN-ELSE` logic) in a SQL statement are the `CASE` expression and the `DECODE` function.

**Note:** The `CASE` expression complies with the ANSI SQL. The `DECODE` function is specific to Oracle syntax.

# CASE Expression

Facilitates conditional inquiries by doing the work of an IF-THEN-ELSE statement:

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



CASE expressions allow you to use the IF-THEN-ELSE logic in SQL statements without having to invoke procedures.

In a simple CASE expression, the Oracle server searches for the first WHEN ... THEN pair for which expr is equal to comparison_expr and returns return_expr. If none of the WHEN ... THEN pairs meet this condition, and if an ELSE clause exists, the Oracle server returns else_expr. Otherwise, the Oracle server returns a null. You cannot specify the literal NULL for all the return_exprs and the else_expr.

The expressions expr and comparison_expr must be of the same data type, which can be CHAR, VARCHAR2, NCHAR, or NVARCHAR2, NUMBER, BINARY_FLOAT, or BINARY_DOUBLE or must all have a numeric data type. All of the return values (return_expr) must be of the same data type.

# Using the CASE Expression

```
SELECT last_name, job_id, salary,
       CASE job_id WHEN 'IT_PROG'  THEN  1.10*salary
                   WHEN 'ST_CLERK' THEN  1.15*salary
                   WHEN 'SA_REP'   THEN  1.20*salary
       ELSE       salary END     "REVISED_SALARY"
FROM   employees;
```

| | LAST_NAME | JOB_ID | SALARY | REVISED_SALARY |
|---|---|---|---|---|
| 1 | King | AD_PRES | 24000 | 24000 |
| ... | | | | |
| 4 | Hunold | IT_PROG | 9000 | 9900 |
| 5 | Ernst | IT_PROG | 6000 | 6600 |
| 6 | Lorentz | IT_PROG | 4200 | 4620 |
| 7 | Mourgos | ST_MAN | 5800 | 5800 |
| 8 | Rajs | ST_CLERK | 3500 | 4025 |
| 9 | Davies | ST_CLERK | 3100 | 3565 |
| 10 | Matos | ST_CLERK | 2600 | 2990 |
| 11 | Vargas | ST_CLERK | 2500 | 2875 |
| ... | | | | |
| 13 | Abel | SA_REP | 11000 | 13200 |
| 14 | Taylor | SA_REP | 8600 | 10320 |
| 15 | Grant | SA_REP | 7000 | 8400 |



In the SQL statement in the slide, the value of JOB_ID is decoded. If JOB_ID is IT_PROG, the salary increase is 10%; if JOB_ID is ST_CLERK, the salary increase is 15%; if JOB_ID is SA_REP, the salary increase is 20%. For all other job roles, there is no increase in salary.

The same statement can be written with the DECODE function.

# Searched CASE Expression

```
CASE
    WHEN condition1 THEN use_expression1
    WHEN condition2 THEN use_expression2
    WHEN condition3 THEN use_expression3
    ELSE default_use_expression
END
```

```
SELECT last_name,salary,
(CASE WHEN salary<5000 THEN 'Low'
      WHEN salary<10000 THEN 'Medium'
      WHEN salary<20000 THEN 'Good'
      ELSE 'Excellent'
END) qualified_salary
FROM employees;
```







In a searched CASE expression, the search occurs from left to right until an occurrence of the listed condition is found, and then it returns the return expression. If no condition is found to be true, and if an ELSE clause exists, the return expression in the ELSE clause is returned; otherwise, a NULL is returned. The searched CASE evaluates the conditions independently under each of the WHEN options.

The difference between the CASE expression and the searched CASE expression is that in a searched CASE expression, you specify a condition or predicate instead of a *comparison_expression* after the WHEN keyword.

For both simple and searched CASE expressions, all of the *return_exprs* must either have the same data type CHAR, VARCHAR2, NCHAR, or NVARCHAR2, NUMBER, BINARY_FLOAT, or BINARY_DOUBLE or must all have a numeric data type.

The code in the slide is an example of the searched CASE expression.

# DECODE Function

Facilitates conditional inquiries by doing the work of a CASE expression or an IF-THEN-ELSE statement:

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





ORACLE



The DECODE function decodes an expression in a way similar to the IF-THEN-ELSE logic that is used in various languages. The DECODE function decodes *expression* after comparing it to each *search* value. If the expression is the same as *search*, *result* is returned.

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

# Using the DECODE Function

```
SELECT last_name, job_id, salary,
       DECODE(job_id, 'IT_PROG',  1.10*salary,
                      'ST_CLERK', 1.15*salary,
                      'SA_REP',   1.20*salary,
              salary)
       REVISED_SALARY
FROM   employees;
```

| | LAST_NAME | JOB_ID | SALARY | REVISED_SALARY |
|---|---|---|---|---|
| ... | | | | |
| 4 | Hunold | IT_PROG | 9000 | 9900 |
| 5 | Ernst | IT_PROG | 6000 | 6600 |
| 6 | Lorentz | IT_PROG | 4200 | 4620 |
| 7 | Mourgos | ST_MAN | 5800 | 5800 |
| 8 | Rajs | ST_CLERK | 3500 | 4025 |
| 9 | Davies | ST_CLERK | 3100 | 3565 |
| 10 | Matos | ST_CLERK | 2600 | 2990 |
| 11 | Vargas | ST_CLERK | 2500 | 2875 |
| 12 | Zlotkey | SA_MAN | 10500 | 10500 |
| ... | | | | |
| 13 | Abel | SA_REP | 11000 | 13200 |
| 14 | Taylor | SA_REP | 8600 | 10320 |
| 15 | Grant | SA_REP | 7000 | 8400 |



In the SQL statement in the slide, the value of JOB_ID is tested. If JOB_ID is IT_PROG, the salary increase is 10%; if JOB_ID is ST_CLERK, the salary increase is 15%; if JOB_ID is SA_REP, the salary increase is 20%. For all other job roles, there is no increase in salary.

The same statement can be expressed in pseudocode as an IF-THEN-ELSE statement:

```
IF job_id = 'IT_PROG'     THEN  salary = salary*1.10
IF job_id = 'ST_CLERK'    THEN  salary = salary*1.15
IF job_id = 'SA_REP'      THEN  salary = salary*1.20
ELSE salary = salary
```

# Using the DECODE Function

Display the applicable tax rate for each employee in department 80:

```
SELECT last_name, salary,
       DECODE (TRUNC(salary/2000, 0),
                         0, 0.00,
                         1, 0.09,
                         2, 0.20,
                         3, 0.30,
                         4, 0.40,
                         5, 0.42,
                         6, 0.44,
                            0.45) TAX_RATE
FROM   employees
WHERE  department_id = 80;
```



This slide shows another example using the DECODE function. In this example, you determine the tax rate for each employee in department 80 based on the monthly salary. The tax rates are as follows:

| *Monthly Salary Range* | *Tax Rate* |
|---|---|
| $0.00–1,999.99 | 00% |
| $2,000.00–3,999.99 | 09% |
| $4,000.00–5,999.99 | 20% |
| $6,000.00–7,999.99 | 30% |
| $8,000.00–9,999.99 | 40% |
| $10,000.00–11,999.99 | 42% |
| $12,200.00–13,999.99 | 44% |
| $14,000.00 or greater | 45% |

# Quiz

The `TO_NUMBER` function converts either character strings or date values to a number in the format specified by the optional format model.

a. True
b. False



**Answer: b**

# Summary

In this lesson, you should have learned how to:

- Alter date formats for display using functions
- Convert column data types using functions
- Use `NVL` functions
- Use `IF-THEN-ELSE` logic and other conditional expressions in a `SELECT` statement







Remember the following:

- Conversion functions can convert character, date, and numeric values: `TO_CHAR`, `TO_DATE`, `TO_NUMBER`
- There are several functions that pertain to nulls, including `NVL`, `NVL2`, `NULLIF`, and `COALESCE`.
- The `IF-THEN-ELSE` logic can be applied within a SQL statement by using the `CASE` expression, searched `CASE`, or the `DECODE` function.

# Practice 5: Overview

This practice covers the following topics:

- Creating queries that use `TO_CHAR`, `TO_DATE`, and other `DATE` functions
- Creating queries that use conditional expressions such as `CASE`, searched `CASE`, and `DECODE`





ORACLE



This practice provides a variety of exercises using the `TO_CHAR` and `TO_DATE` functions, and conditional expressions such as `CASE`, searched `CASE`, and `DECODE`.

Remember that for nested functions, the results are evaluated from the innermost function to the outermost function.