LAB SHEET 6

Title	: Using Conversion Functions and Conditional Expressions
Objectives	: At the end of the session, students are able to: i. Describe the various types of conversion functions that are available in SQL ii. Use the TO_CHAR, TO_NUMBER, and TO_DATE conversion functions iii. Apply conditional expressions in a SELECT statement
Duration	: 2 Hours

Conversion Functions

In addition to Oracle data types, columns of tables in an Oracle Database can be defined by using the 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.

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 conversion `data type TO data type`. The first data type is the input data type and the second data type is the output.

Implicit Data Type Conversion

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

From	To
`VARCHAR2 or CHAR`	`NUMBER, DATE`
`DATE, NUMBER`	`VARCHAR2 or CHAR`

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.

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

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

Explicit Data Type Conversion

Function	Purpose
`TO_CHAR (number|date, [fmt], [nlsparams])`	Converts a number or date value to a `VARCHAR2` character string with the format model `fmt` **Number conversion:** The `nlsparams` parameter specifies the following characters, which are returned by number format elements:

	• Decimal character • Group separator • Local currency symbol • International currency symbol If `nlsparams` or any other parameter is omitted, this function uses the default parameter values for the session. **Date conversion:** The `nlsparams` parameter specifies the language in which the month and day names and abbreviations are returned. If this parameter is omitted, this function uses the default date languages for the session.
`TO_NUMBER (char, [fmt], [nlsparams])`	Converts a character string containing digits to a number in the format specified by the optional format model `fmt`. 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`. The `nlsparams` parameter has the same purpose in this function as in the `TO_CHAR` function for date conversion.

Using the TO_CHAR Function with Dates

```
TO_CHAR(date, ‘format_model’)
```

The format model:

- Must be enclosed with 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

Try this:

```
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

Sample Format Elements of Valid Date Formats

Element	Description
`SCC or CC`	Century; server prefixes B.C. date with -
`Years in dates YYYY or SYYYY`	Year; server prefixes B.C. date with -
`YYY or YY or Y`	Last three, two, or one digit of the year
`Y,YYY`	Year with comma in this position
`IYYY, IYY, IY, I`	Four-, three-, two-, or one-digit year based on the ISO standard
`SYEAR or YEAR`	Year spelled out; server prefixes B.C. date with -
`B.C. or A.D.`	Indicates B.C. or A.D. year using periods
`BC or AD`	Indicates B.C. or A.D. year
`Q`	Quarter of year
`MM`	Month; two-digit value
`MONTH`	Name of the month padded with blanks to a length of nine characters
`MON`	Name of the month, three-letter abbreviation
`RM`	Roman numeral month
`WW or W`	Week of the year or month
`DDD or DD or D`	Day of the year, month or week
`DAY`	Name of the day padded with blanks to a length of nine characters
`DY`	Name of the day; three-letter abbreviation
`J`	Julian day; the number of days since December 31, 4713 B.C.
`IW`	Weeks in the year from ISO standard (1 to 53)

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 with 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 or HH24`	Hour of day, or hour(1-12), or hour (0-23)
`MI`	Minute (0-59)
`SS`	Second (0-59)
`SSSS`	Seconds past midnight (0-86399)
`/ . ,`	Punctuation is reproduced in the result

"of the"	Quoted string is reproduced in the result
TH	Ordinal number (for example, DDTH for 4TH)
SP	Spelled-out number (for example, DDSP for FOUR)
SPTH	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;
```

The SQL statement displays the last names and hire dates for all employees. The hire date appears as 17 June 1987

Try This:
Modify the example above to display the dates in a format that appears as "Seventeenth of June 1987 12:00:00 AM."

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

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	Description	Example	Result
9	Numeric position (number of 9s determine display width)	999999	1234
0	Display leading zeros	099999	001234
$	Places a floating dollar sign	$999999	$1234
L	Uses the floating local currency symbol	L999999	RM1234
D	Returns the decimal character in the specified position. The default is a period (.)	99D99	99.99
.	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.	9,9999	9G999
,	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

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

- The Oracle server displays a string of number sign (#) 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 that 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 last name and hire date for all employees who started on May 24, 1999. 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, 1999', 'fxMonth DD YYYY');
```

Observe the resulting error.

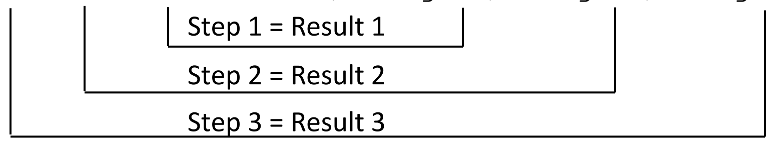
To see the output, correct the query by deleting the extra space between 'May' and '24'.

```
SELECT last_name, hire_date
FROM employees
WHERE hire_date = TO_DATE('May 24, 1999', 'fxMonth DD YYYY');
```

Nesting Functions

Single-row functions can be nested to any level. Nested functions are evaluated from the deepest level to the least deep level.

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



Example:

```
      SELECT last_name,
         UPPER(CONCAT(SUBSTR (LAST_NAME, 1, 8), '_US'))
      FROM employees
      WHERE department_id = 60;
```

The SELECT statement displays the last names of employees in department 60. The evaluation of the SQL statement involves three steps:

1. The inner function retrieves the first eight characters of the last name
   `Result1 = SUBSTR (LAST_NAME, 1,8)`
2. The outer function concatenates the result with `_US`.
   `Result2 = CONCAT(Result1, '_US')`
3. The outermost function converts the results to uppercase.

The entire expression becomes the column heading because no column alias was given.

Example:
Display the date of the next Friday that is six months from the hire date. The resulting date should appear as Friday, August 13th, 1999. Order the results by hire date.

```
      SELECT  TO_CHAR(NEXT_DAY(ADD_MONTHS(hire_date,  6),  'FRIDAY'),  'fmDay,
            Month ddth, YYYY') "Next 6 Month Review"
      FROM employees
      ORDER BY hire_date;
```

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)

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

NVL Functions

Syntax

```
NVL (expr1, expr2)
```

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

Converts a null value to an actual value:

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

Example:

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 values to a number before applying the arithmetic operator as follows:

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

NVL2 Functions

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

Syntax

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

- *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.

Example:

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

In the example, 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.

NULLIF Functions

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*.

Example:

```
SELECT first_name, LENGTH(first_name) "expr1",
       last_name, LENGTH(last_name) "expr2",
       NULLIF(LENGTH(first_name), LENGTH(last_name)) result
FROM employees;
```

The length of the first name in the EMPLOYEES table is compared to 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 names is displayed.

COALESCE Functions

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

Example:

```
SELECT last_name, employee_id
COALESCE(TO_CHAR(commission_pct), TO_CHAR(manager_id), 'No commission
and no manager')
FROM employees;
```

In the example, if the manager_id value is not null, it is displayed. If the manager_id value is null, the commission_pct is displayed. If the manager_id and commission_pct values are null, "No commission and no manager" is displayed. Note that TO_CHAR function is applied so that all expressions are of the same data type.

Conditional Expressions

Provide the use of the IF-THEN-ELSE logic within a SQL statement. Use two methods:

- CASE expression
- DECODE expression

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]
```

You cannot specify the literal NULL for all the return_exprs and the else_expr. The expr and comparison_expr must be of the same data type, which can be CHAR, VARCHAR2, NCHAR, or NVARCHAR2. All of the return values (return_expr) must be of the same data type.

Example:

```
SELECT last_name, job_id, salary,
      CASE job_id WHEN 'IT_PROG' THEN 1.30*salary
                  WHEN 'ST_CLERK' THEN 1.15*salary
                  WHEN 'SA_REP' THEN 1.20*salary
      ELSE  salary END "REVISED_SALARY"
FROM employees;
```

In the statement, the value of JOB_ID is decoded. If JOB_ID is IT_PROG, the salary increase is 30%; 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 following code is an example of the searched CASE expression. 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.

```
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;
```

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

```
SELECT last_name, job_id salary,
      DECODE(job_id, 'IT_PROG', 1.30*salary,
                     'ST_CLERK', 1.15*salary
                     'SA_REP', 1.20*salary,
             Salary) "REVISED_SALARY"

FROM employees;
```

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.

Example:
Determine the tax rate for each employee in department 80 based on the monthly salary. The tax rates are as follows:

Monthly Salary Range (RM)	**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 or greater	42%

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

Exercise

Instructions: Save all your statement as a lab_06_<exercise_no>.txt (saves sql script and its output).

1. Create a report that produces the following for each employee:
   `<employee last name>` earns `<salary> monthly but wants <3 times salary.>`.
   Label the column `Dream Salaries`.

	Dream Salaries
1	Whalen earns $4,400.00 monthly but wants $13,200.00
2	Hartstein earns $13,000.00 monthly but wants $39,000.00
3	Fay earns $6,000.00 monthly but wants $18,000.00

2. Display each employee's last name, hire date, and salary review date, which is the first Monday after six months of service. Label the column `REVIEW`. Format the dates to appear in the format similar to "Monday, the Thirty-First of July, 2000.".

	LAST_NAME	HIRE_DATE	REVIEW
1	Whalen	17-SEP-87	Monday, the Twenty-First of March, 1988
2	Hartstein	17-FEB-96	Monday, the Nineteenth of August, 1996
3	Fay	17-AUG-97	Monday, the Twenty-Third of February, 1998

3. Display the last name, hire date, and day of the week on which the employee started. Label the column DAY. Order the results by the day of the week, starting with Monday.

	LAST_NAME	HIRE_DATE	DAY
1	Grant	24-MAY-99	MONDAY
2	Ernst	21-MAY-91	TUESDAY
3	Taylor	24-MAR-98	TUESDAY

4. Create a query that displays the employees' last names and commission amounts. If an employee does not earn commission, show "No Commission." Label the column `COMM`.

	LAST_NAME	COMM
1	Whalen	No Commission
2	Hartstein	No Commission
3	Zlotkey	.2

5. Using the DECODE function, write a query that displays the grade of all employees based on the value of the column JOB_ID, using the following data:

Job	*Grade*
AD_PRESS	A
ST_MAN	B
IT_PROG	C
None of the above	0

	JOB_ID	GRADE
1	AC_ACCOUNT	0
2	AD_PRESS	A
3	IT_PROG	C

6. Rewrite the statement in **exercise 5** by using the CASE syntax.