3

Using Single-Row Functions to Customize Output

ORACLE

Objectives

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

- Describe various types of functions that are available in SQL
- Use character, number, and date functions in `SELECT` statements
- Describe the use of conversion functions

Objectives

Functions make the basic query block more powerful, and they are used to manipulate data values. This is the first of two lessons that explore functions. It focuses on single-row character, number, and date functions, as well as those functions that convert data from one type to another (for example, conversion from character data to numeric data).

SQL Functions

Input

Output

Function

Function performs action

arg 1

arg 2

arg n

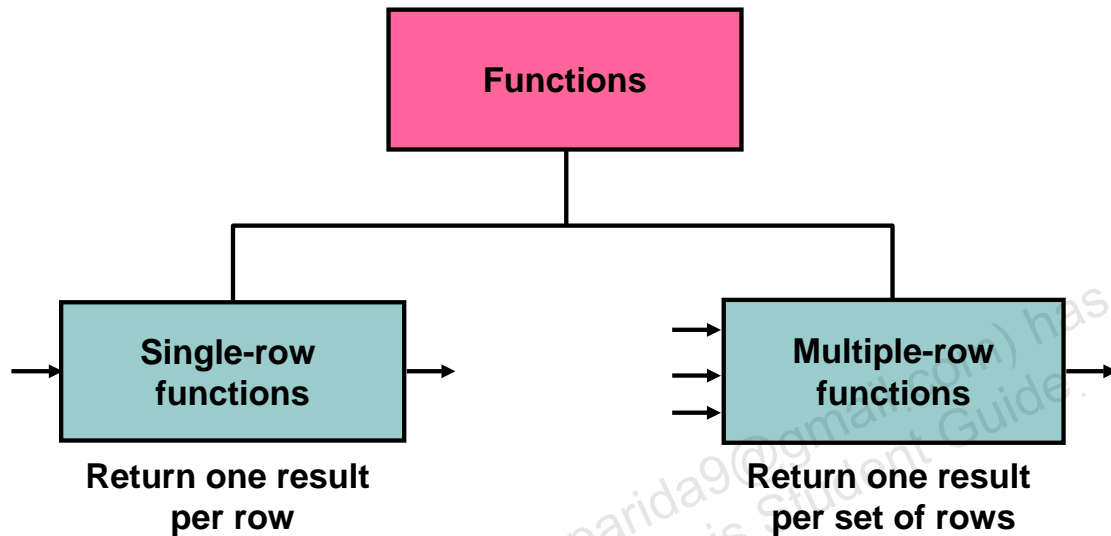**Result value**
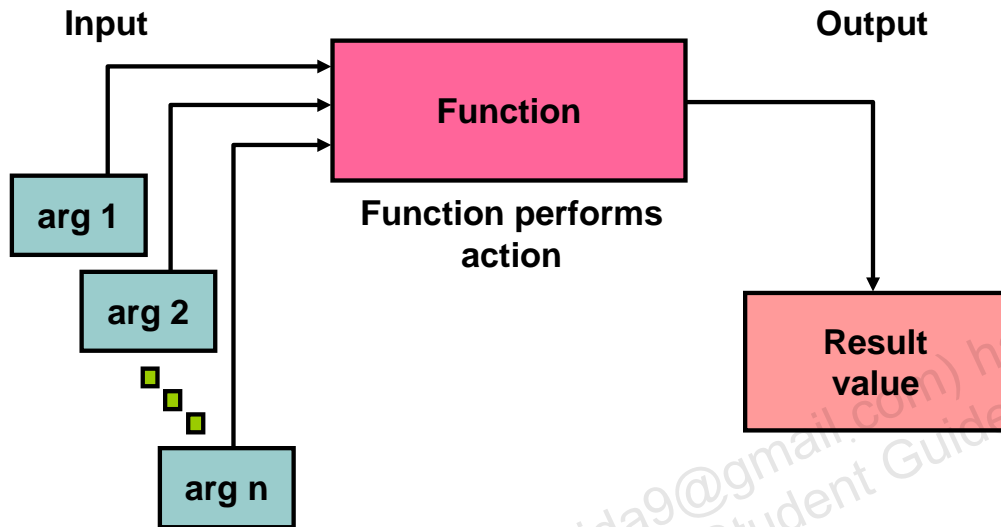
ORACLE

SQL Functions

Functions are a very powerful feature of SQL. They can be used to do the following:

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

SQL functions sometimes take arguments and always return a value.

Note: Most of the functions that are described in this lesson are specific to the Oracle version of SQL.

Two Types of SQL Functions

Functions

Single-row functions

Multiple-row functions

Return one result per row

Return one result per set of rows

SQL Functions (continued)

There are two types of functions:

- Single-row functions
- Multiple-row functions

Single-Row Functions

These functions operate on single rows only and return one result per row. There are different types of single-row functions. This lesson covers the following ones:

- Character
- Number
- Date
- Conversion
- General

Multiple-Row Functions

Functions can manipulate groups of rows to give one result per group of rows. These functions are also known as *group functions* (covered in lesson 4).

Note: For more information and a complete list of available functions and their syntax, see *Oracle SQL Reference*.

Single-Row Functions

Single-row functions:

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

```
function_name [(arg1, arg2,...)]
```

ORACLE

Single-Row Functions

Single-row functions are used to manipulate data items. They accept one or more arguments and return one value for each row that is returned by the query. An argument can be one of the following:

- User-supplied constant
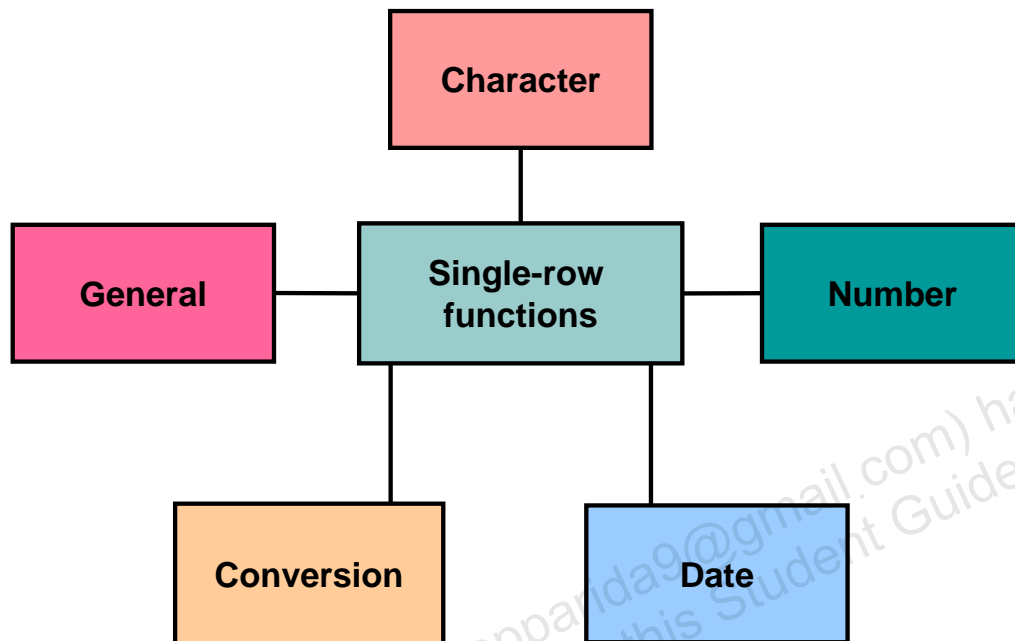- Variable value
- Column name
- Expression

Features of single-row functions include:

- Acting on each row that is returned in the query
- Returning one result per row
- Possibly returning a data value of a different type than the one that is referenced
- Possibly expecting one or more arguments
- Can be used in SELECT, WHERE, and ORDER BY clauses; can be nested

In the syntax:

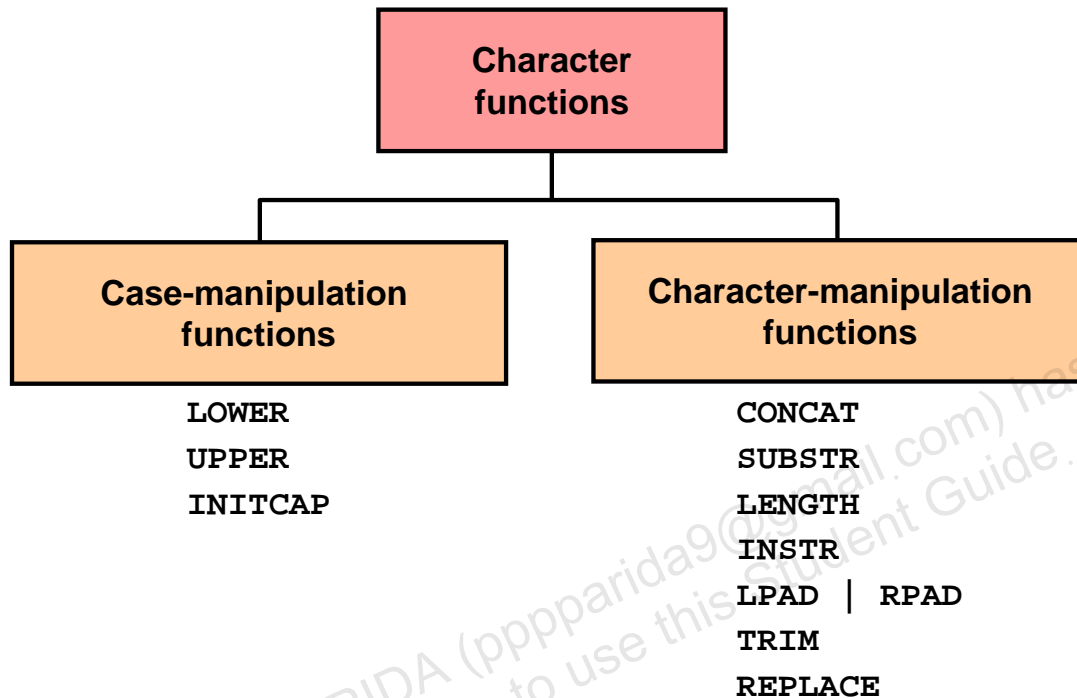|	
function_name	is the name of the function
arg1, arg2	is any argument to be used by the function. This can be represented by a column name or expression.

Single-Row Functions

```
                    ┌──────────────┐
                    │  Character   │
                    └──────┬───────┘
                           │
┌──────────┐      ┌────────┴────────┐      ┌──────────┐
│ General  │──────│   Single-row    │──────│  Number  │
└──────────┘      │   functions     │      └──────────┘
                  └────────┬────────┘
              ┌────────────┴────────────┐
       ┌──────┴──────┐           ┌──────┴──────┐
       │ Conversion  │           │    Date     │
       └─────────────┘           └─────────────┘
```

Single-Row Functions (continued)

This lesson covers the following single-row functions:

- **Character functions:** Accept character input and can return both character and number values
- **Number functions:** Accept numeric input and return numeric values
- **Date functions:** Operate on values of the DATE data type (All date functions return a value of DATE data type except the MONTHS_BETWEEN function, which returns a number.)
- **Conversion functions:** Convert a value from one data type to another
- **General functions:**
  - NVL
  - NVL2
  - NULLIF
  - COALESCE
  - CASE
  - DECODE

Character Functions

Character functions

Case-manipulation functions

Character-manipulation functions

LOWER	CONCAT
UPPER	SUBSTR
INITCAP	LENGTH
	INSTR
	LPAD | RPAD
	TRIM
	REPLACE

ORACLE

Character Functions

Single-row character functions accept character data as input and can return both character and numeric values. Character functions can be divided into the following:

- Case-manipulation functions
- Character-manipulation functions

Function	Purpose
LOWER(*column|expression*)	Converts alpha character values to lowercase
UPPER(*column|expression*)	Converts alpha character values to uppercase
INITCAP(*column|expression*)	Converts alpha character values to uppercase for the first letter of each word; all other letters in lowercase
CONCAT(*column1|expression1, column2|expression2*)	Concatenates the first character value to the second character value; equivalent to concatenation operator (||)
SUBSTR(*column|expression,m[,n]*)	Returns specified characters from character value starting at character position *m, n* characters long (If *m* is negative, the count starts from the end of the character value. If *n* is omitted, all characters to the end of the string are returned.)

Note: The functions discussed in this lesson are only some of the available functions.

Character Functions (continued)

Function	Purpose
LENGTH(*column*│*expression*)	Returns the number of characters in the expression
INSTR(*column*│*expression*, 'string', [,m], [n])	Returns the numeric position of a named string. Optionally, you can provide a position *m* to start searching, and the occurrence *n* of the string. *m* and *n* default to 1, meaning start the search at the beginning of the search and report the first occurrence.
LPAD(*column*│*expression*, n, 'string') RPAD(*column*│*expression*, n, 'string')	Pads the character value right-justified to a total width of *n* character positions Pads the character value left-justified to a total width of *n* character positions
TRIM(*leading*│*trailing*│*both*, *trim_character FROM trim_source*)	Enables you to trim heading or trailing characters (or both) from a character string. If *trim_character* or *trim_source* is a character literal, you must enclose it in single quotation marks. This is a feature that is available in Oracle8*i* and later versions.
REPLACE(*text*, *search_string*, *replacement_string*)	Searches a text expression for a character string and, if found, replaces it with a specified replacement string

Case-Manipulation Functions

These functions convert case for character strings:

Function	Result
LOWER('SQL Course')	sql course
UPPER('SQL Course')	SQL COURSE
INITCAP('SQL Course')	Sql Course

ORACLE

Case-Manipulation Functions

LOWER, UPPER, and INITCAP are the three case-conversion functions.

- **LOWER:** Converts mixed-case or uppercase character strings to lowercase
- **UPPER:** Converts mixed-case or lowercase character strings to uppercase
- **INITCAP:** Converts the first letter of each word to uppercase and remaining letters to lowercase

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

	EMPLOYEE DETAILS
1	The job id for ABEL is sa_rep
2	The job id for DAVIES is st_clerk
3	The job id for DE HAAN is ad_vp

. . .

| 19 | The job id for WHALEN is ad_asst |
| 20 | The job id for ZLOTKEY is sa_man |

Using Case-Manipulation Functions

Display the employee number, name, and department number for employee Higgins:

```
SELECT  employee_id, last_name, department_id
FROM    employees
WHERE   last_name = 'higgins';
no rows selected
```

```
SELECT  employee_id, last_name, department_id
FROM    employees
WHERE   LOWER(last_name) = 'higgins';
```

	EMPLOYEE_ID	LAST_NAME	DEPARTMENT_ID
1	205	Higgins	110

ORACLE

Using Case-Manipulation Functions

The slide example displays the employee number, name, and department number of employee Higgins.

The WHERE clause of the first SQL statement specifies the employee name as higgins. Because all the data in the EMPLOYEES table is stored in proper case, the name higgins does not find a match in the table, and no rows are selected.

The WHERE clause of the second SQL statement specifies that the employee name in the EMPLOYEES table is compared to higgins, converting the LAST_NAME column to lowercase for comparison purposes. Because both names are now lowercase, a match is found and one row is selected. The WHERE clause can be rewritten in the following manner to produce the same result:

```
...WHERE last_name = 'Higgins'
```

The name in the output appears as it was stored in the database. To display the name in uppercase, use the UPPER function in the SELECT statement.

```
SELECT  employee_id, UPPER(last_name), department_id
FROM    employees
WHERE   INITCAP(last_name) = 'Higgins';
```

Character-Manipulation Functions

These functions manipulate character strings:

Function	Result
CONCAT('Hello', 'World')	HelloWorld
SUBSTR('HelloWorld',1,5)	Hello
LENGTH('HelloWorld')	10
INSTR('HelloWorld', 'W')	6
LPAD(salary,10,'*')	*****24000
RPAD(salary, 10, '*')	24000*****
REPLACE ('JACK and JUE','J','BL')	BLACK and BLUE
TRIM('H' FROM 'HelloWorld')	elloWorld

ORACLE

Character-Manipulation Functions

CONCAT, SUBSTR, LENGTH, INSTR, LPAD, RPAD, and TRIM are the character-manipulation functions that are covered in this lesson.

- **CONCAT:** Joins values together (You are limited to using two parameters with CONCAT.)
- **SUBSTR:** Extracts a string of determined length
- **LENGTH:** Shows the length of a string as a numeric value
- **INSTR:** Finds the numeric position of a named character
- **LPAD:** Pads the character value right-justified
- **RPAD:** Pads the character value left-justified
- **TRIM:** Trims heading or trailing characters (or both) from a character string (If *trim_character* or *trim_source* is a character literal, you must enclose it in single quotation marks.)

Note: You can use functions such as UPPER and LOWER with ampersand substitution. For example, use UPPER('&job_title') so that the user does not have to enter the job title in a specific case.

Using the Character-Manipulation Functions

Using the Character-Manipulation Functions

The slide example displays employee first names and last names joined together, the length of the employee last name, and the numeric position of the letter *a* in the employee last name for all employees who have the string REP contained in the job ID starting at the fourth position of the job ID.

Example

Modify the SQL statement in the slide to display the data for those employees whose last names end with the letter *n*.

```
SELECT employee_id, CONCAT(first_name, last_name) NAME,
LENGTH (last_name), INSTR(last_name, 'a') "Contains 'a'?"
FROM    employees
WHERE   SUBSTR(last_name, -1, 1) = 'n';
```

	EMPLOYEE_ID	NAME	LENGTH(LAST_NAME)	Contains 'a'?
1	102	LexDe Haan	7	5
2	200	JenniferWhalen	6	3
3	201	MichaelHartstein	9	2

Number Functions

- ROUND: Rounds value to specified decimal
- TRUNC: Truncates value to specified decimal
- MOD: Returns remainder of division

Function	Result
ROUND(45.926, 2)	45.93
TRUNC(45.926, 2)	45.92
MOD(1600, 300)	100

ORACLE

Number Functions

Number functions accept numeric input and return numeric values. This section describes some of the number functions.

Function	Purpose
ROUND(*column*|*expression*, *n*)	Rounds the column, expression, or value to *n* decimal places or, if *n* is omitted, no decimal places (If *n* is negative, numbers to left of the decimal point are rounded.)
TRUNC(*column*|*expression*, *n*)	Truncates the column, expression, or value to *n* decimal places or, if *n* is omitted, *n* defaults to zero
MOD(*m*,*n*)	Returns the remainder of *m* divided by *n*

Note: This list contains only some of the available number functions.

For more information, see "Number Functions" in *Oracle SQL Reference.*

Using the ROUND Function


```
SELECT ROUND(45.923,2), ROUND(45.923,0),
       ROUND(45.923,-1)
FROM   DUAL;
```

ROUND(45.923,2)	ROUND(45.923,0)	ROUND(45.923,-1)
45.92	46	50
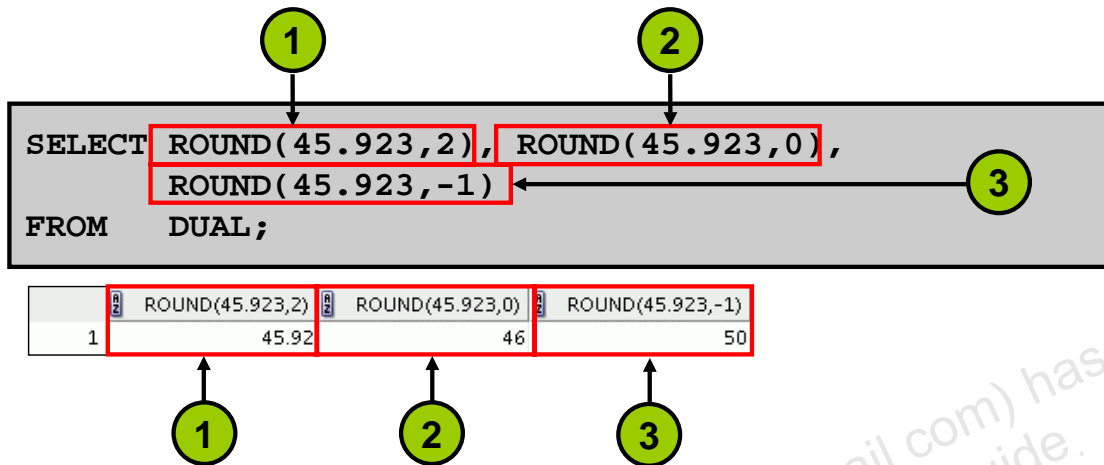
DUAL is a dummy table that you can use to view results from functions and calculations.
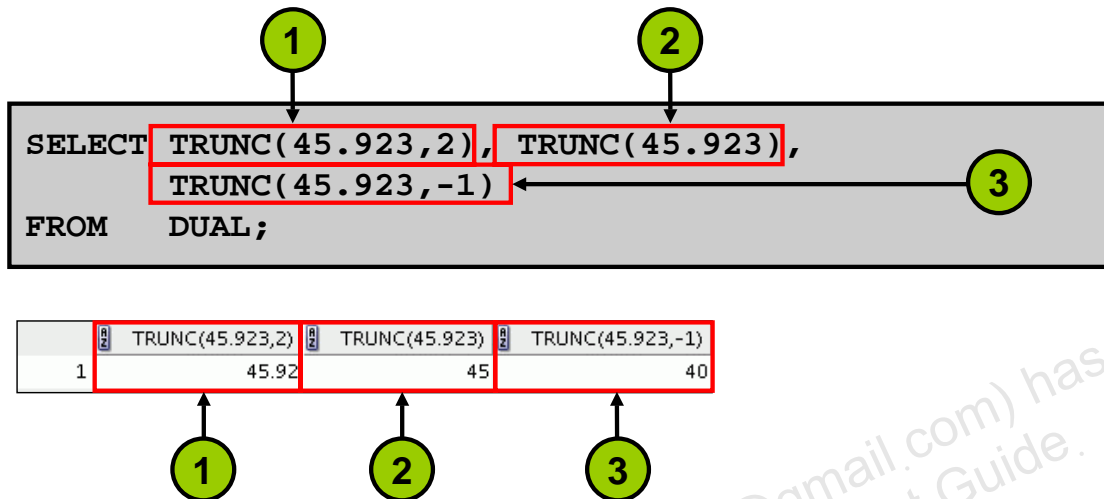
ROUND Function

The ROUND function rounds the column, expression, or value to *n* decimal places. If the second argument is 0 or is missing, the value is rounded to zero decimal places. If the second argument is 2, the value is rounded to two decimal places. Conversely, if the second argument is –2, the value is rounded to two decimal places to the left (rounded to the nearest unit of 10).

The ROUND function can also be used with date functions. You will see examples later in this lesson.

DUAL Table

The DUAL table is owned by the user SYS and can be accessed by all users. It contains one column, DUMMY, and one row with the value X. The DUAL table is useful when you want to return a value once only (for example, the value of a constant, pseudocolumn, or expression that is not derived from a table with user data). The DUAL table is generally used for SELECT clause syntax completeness, because both SELECT and FROM clauses are mandatory, and several calculations do not need to select from actual tables.

Using the TRUNC Function

```
SELECT TRUNC(45.923,2), TRUNC(45.923),
       TRUNC(45.923,-1)
FROM   DUAL;
```

TRUNC(45.923,2)	TRUNC(45.923)	TRUNC(45.923,-1)
45.92	45	40

ORACLE

TRUNC Function

The TRUNC function truncates the column, expression, or value to *n* decimal places.

The TRUNC function works with arguments similar to those of the ROUND function. If the second argument is 0 or is missing, the value is truncated to zero decimal places. If the second argument is 2, the value is truncated to two decimal places. Conversely, if the second argument is –2, the value is truncated to two decimal places to the left. If the second argument is –1, the value is truncated to one decimal place to the left.

Like the ROUND function, the TRUNC function can be used with date functions.

Using the MOD Function

For all employees with job title of Sales Representative, calculate the remainder of the salary after it is divided by 5,000.

```
SELECT last_name, salary, MOD(salary, 5000)
FROM   employees
WHERE  job_id = 'SA_REP';
```

	LAST_NAME	SALARY	MOD(SALARY,5000)
1	Abel	11000	1000
2	Taylor	8600	3600
3	Grant	7000	2000

ORACLE

MOD Function

The MOD function finds the remainder of the first argument divided by the second argument. The slide example calculates the remainder of the salary after dividing it by 5,000 for all employees whose job ID is SA_REP.

Note: The MOD function is often used to determine if a value is odd or even.

Working with Dates

- The Oracle Database stores dates in an internal numeric format: century, year, month, day, hours, minutes, and seconds.

- The default date display format is DD-MON-RR.
    - Enables you to store 21st-century dates in the 20th century by specifying only the last two digits of the year
    - Enables you to store 20th-century dates in the 21st century in the same way

```
SELECT  last_name, hire_date
FROM    employees
WHERE   hire_date < '01-FEB-88';
```

	LAST_NAME	HIRE_DATE
1	Whalen	17-SEP-87
2	King	17-JUN-87

ORACLE

Oracle Date Format

The Oracle Database stores dates in an internal numeric format, representing the century, year, month, day, hours, minutes, and seconds.

The default display and input format for any date is DD-MON-RR. Valid Oracle dates are between January 1, 4712 B.C., and December 31, 9999 A.D.

In the example in the slide, the HIRE_DATE column output is displayed in the default format DD-MON-RR. However, dates are not stored in the database in this format. All the components of the date and time are stored. So, although a HIRE_DATE such as 17-JUN-87 is displayed as day, month, and year, there is also *time* and *century* information associated with the date. The complete data might be June 17, 1987, 5:10:43 p.m.

Oracle Date Format (continued)

This data is stored internally as follows:

CENTURY	YEAR	MONTH	DAY	HOUR	MINUTE	SECOND
19	87	06	17	17	10	43

Centuries and the Year 2000

When a record with a date column is inserted into a table, the *century* information is picked up from the SYSDATE function. However, when the date column is displayed on the screen, the century component is not displayed (by default).

The DATE data type always stores year information as a four-digit number internally: two digits for the century and two digits for the year. For example, the Oracle Database stores the year as 1987 or 2004, and not just as 87 or 04.

Working with Dates

SYSDATE is a function that returns:

* Date
* Time

SYSDATE Function

SYSDATE is a date function that returns the current database server date and time. You can use SYSDATE just as you would use any other column name. For example, you can display the current date by selecting SYSDATE from a table. It is customary to select SYSDATE from a dummy table called DUAL.

Example

Display the current date using the DUAL table.

```
SELECT  SYSDATE
FROM    DUAL;
```

	SYSDATE
1	06-NOV-08

Arithmetic with Dates

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

ORACLE

Arithmetic with Dates

Because the database stores dates as numbers, you can perform calculations using arithmetic operators such as addition and subtraction. You can add and subtract number constants as well as dates.

You can perform the following operations:

Operation	Result	Description
date + number	Date	Adds a number of days to a date
date – number	Date	Subtracts a number of days from a date
date – date	Number of days	Subtracts one date from another
date + number/24	Date	Adds a number of hours to a date

Using Arithmetic Operators with Dates

```
SELECT last_name, (SYSDATE-hire_date)/7 AS WEEKS
FROM    employees
WHERE   department_id = 90;
```

	LAST_NAME	WEEKS
1	King	1116.1485747354497354497354497354497354 5
2	Kochhar	998.00571759259259259259259259259259 3
3	De Haan	825.1485747354497354497354497354497354 5

ORACLE

Using Arithmetic Operators with Dates

The example in the slide displays the last name and the number of weeks employed for all employees in department 90. It subtracts the date on which the employee was hired from the current date (SYSDATE) and divides the result by 7 to calculate the number of weeks that a worker has been employed.

Note: SYSDATE is a SQL function that returns the current date and time. Your results may differ from the example.

If a more current date is subtracted from an older date, the difference is a negative number.

Date Functions

Function	Result
MONTHS_BETWEEN	Number of months between two dates
ADD_MONTHS	Add calendar months to date
NEXT_DAY	Next day of the date specified
LAST_DAY	Last day of the month
ROUND	Round date
TRUNC	Truncate date

ORACLE

Date Functions

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

- **MONTHS_BETWEEN(*date1, date2*):** Finds the number of months between *date1* and
  date2. The result can be positive or negative. If *date1* is later than *date2*, the result is
  positive; if *date1* is earlier than *date2*, the result is negative. The noninteger part of the result
  represents a portion of the month.
- **ADD_MONTHS(*date, n*):** Adds *n* number of calendar months to *date*. The value of *n* must
  be an integer and can be negative.
- **NEXT_DAY(*date, 'char'*):** Finds the date of the next specified day of the week
  (*'char'*) following *date*. The value of *char* may be a number representing a day or a
  character string.
- **LAST_DAY(*date*):** Finds the date of the last day of the month that contains *date*
- **ROUND(*date[,'fmt'*]):** Returns *date* rounded to the unit that is specified by the format
  model *fmt*. If the format model *fmt* is omitted, *date* is rounded to the nearest day.
- **TRUNC(*date[, 'fmt'*]):** Returns *date* with the time portion of the day truncated to the
  unit that is specified by the format model *fmt*. If the format model *fmt* is omitted, *date* is
  truncated to the nearest day.

This list is a subset of the available date functions. The format models are covered later in this lesson.
Examples of format models are month and year.

Using Date Functions

Function	Result
MONTHS_BETWEEN ('01-SEP-95','11-JAN-94')	19.6774194
ADD_MONTHS ('11-JAN-94',6)	'11-JUL-94'
NEXT_DAY ('01-SEP-95','FRIDAY')	'08-SEP-95'
LAST_DAY ('01-FEB-95')	'28-FEB-95'

ORACLE

Date Functions (continued)

For example, display the employee number, hire date, number of months employed, six-month review date, first Friday after hire date, and last day of the hire month for all employees who have been employed for fewer than 120 months.

```
SELECT employee_id, hire_date,
 MONTHS_BETWEEN (SYSDATE, hire_date) TENURE,
 ADD_MONTHS (hire_date, 6) REVIEW,
 NEXT_DAY (hire_date, 'FRIDAY'), LAST_DAY(hire_date)
FROM    employees
WHERE   MONTHS_BETWEEN (SYSDATE, hire_date) < 120;
```

	EMPLOYEE_ID	HIRE_DATE	TENURE	REVIEW	NEXT_DAY(HIRE_DATE,'FRIDAY')	LAST_DAY(HIRE_DATE)
1	107	07-FEB-99	116.96930966248506...	07-AUG-99	12-FEB-99	28-FEB-99
2	124	16-NOV-99	107.67898708183990...	16-MAY-00	19-NOV-99	30-NOV-99
3	149	29-JAN-00	105.25963224313022...	29-JUL-00	04-FEB-00	31-JAN-00
4	178	24-MAY-99	113.42092256571087...	24-NOV-99	28-MAY-99	31-MAY-99

Using Date Functions

Assume `SYSDATE = '25-JUL-03'`:

Function	Result
`ROUND(SYSDATE,'MONTH')`	`01-AUG-03`
`ROUND(SYSDATE ,'YEAR')`	`01-JAN-04`
`TRUNC(SYSDATE ,'MONTH')`	`01-JUL-03`
`TRUNC(SYSDATE ,'YEAR')`	`01-JAN-03`

Date Functions (continued)

The `ROUND` and `TRUNC` functions can be used for number and date values. When used with dates, these functions round or truncate to the specified format model. Therefore, you can round dates to the nearest year or month.

Example

Compare the hire dates for all employees who started in 1997. Display the employee number, hire date, and start month using the `ROUND` and `TRUNC` functions.

```
SELECT employee_id, hire_date,
 ROUND(hire_date, 'MONTH'), TRUNC(hire_date, 'MONTH')
FROM   employees
WHERE  hire_date LIKE '%97';
```

	EMPLOYEE_ID	HIRE_DATE	ROUND(HIRE_DATE,'MONTH')	TRUNC(HIRE_DATE,'MONTH')
1	202	17-AUG-97	01-SEP-97	01-AUG-97
2	142	29-JAN-97	01-FEB-97	01-JAN-97

Practice 3: Overview of Part 1

This practice covers the following topics:

- Writing a query that displays the current date
- Creating queries that require the use of the numeric, character, and date functions
- Performing calculations of years and months of service for an employee

Practice 3: Overview of Part 1

Part 1 of this lesson's practice provides a variety of exercises that use the different functions that are available for the character, number, and date data types.

For Part 1, complete questions 1–6 at the end of this lesson.

Conversion Functions

```
                    ┌─────────────────────┐
                    │     Data type       │
                    │     conversion      │
                    └─────────────────────┘
                    ┌──────────┴──────────┐
          ┌───────────────────┐   ┌───────────────────┐
          │ Implicit data type│   │ Explicit data type│
          │    conversion     │   │    conversion     │
          └───────────────────┘   └───────────────────┘
```
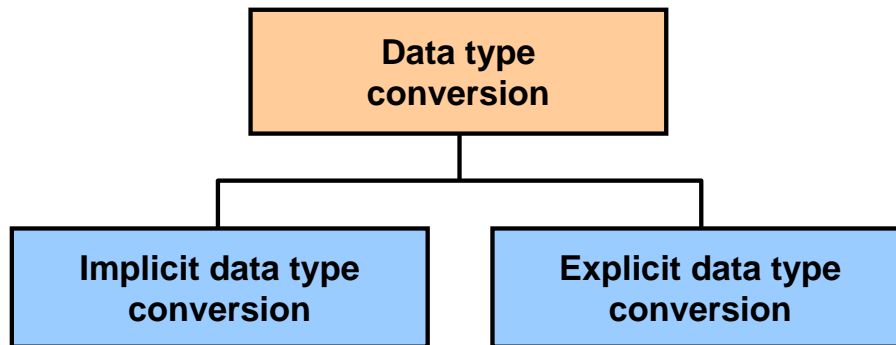
Conversion Functions

In addition to Oracle data types, columns of tables in an Oracle Database can be defined using 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 uses 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 that are explained in the next two slides.

Explicit data type conversions are done 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; the second data type is the output.

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

Implicit Data Type Conversion

For assignments, the Oracle server can automatically convert the following:

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

Implicit Data Type Conversion

The assignment succeeds if the Oracle server can convert the data type of the value used in the assignment to that of the assignment target.

For example, the expression `hire_date > '01-JAN-90'` results in the implicit conversion from the string `'01-JAN-90'` to a date.

Implicit Data Type Conversion

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

From	To
VARCHAR2 or CHAR	NUMBER
VARCHAR2 or CHAR	DATE

ORACLE

Implicit Data Type Conversion (continued)

In general, the Oracle server uses the rule for expressions when a data type conversion is needed in places that are not covered by a rule for assignment conversions.

For example, the expression `salary = '20000'` results in the implicit conversion of the string `'20000'` to the number 20000.

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

Explicit Data Type Conversion

TO_NUMBER   TO_DATE

NUMBER   **CHARACTER**   DATE

TO_CHAR   TO_CHAR
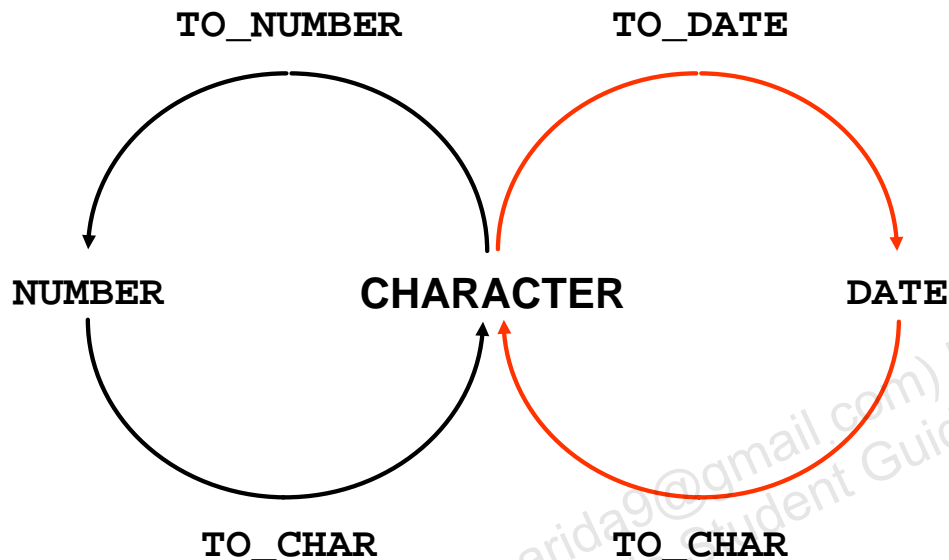
ORACLE

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 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.

Explicit Data Type Conversion
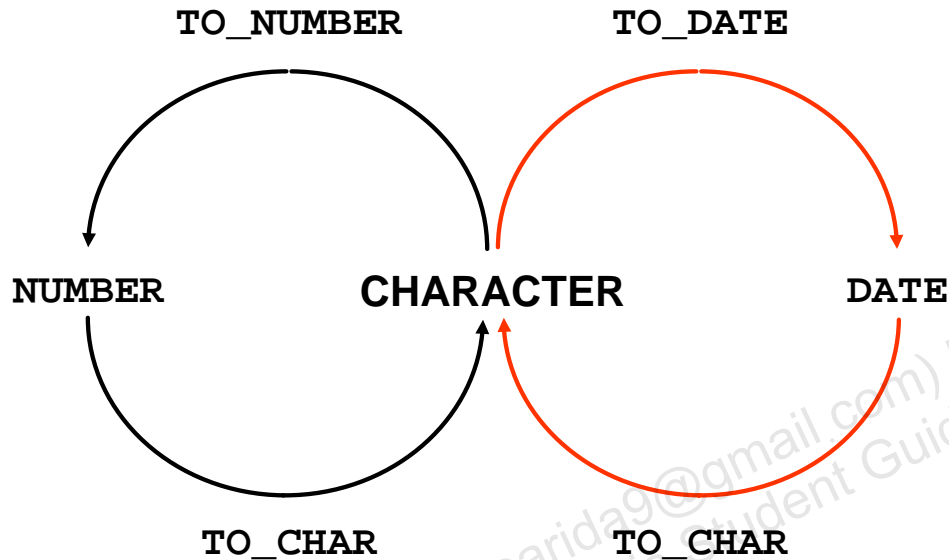
TO_NUMBER   TO_DATE

NUMBER   **CHARACTER**   DATE

TO_CHAR   TO_CHAR

Explicit Data Type Conversion (continued)

Function	Purpose
TO_CHAR(*number*|*date*,[*fmt*], [*nlsparams*])	**Date conversion:** The nlsparams parameter specifies the language in which 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 the *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.

Explicit Data Type Conversion (continued)

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

For more information, see "Conversion Functions" in *Oracle SQL Reference*.

Using the TO_CHAR Function with Dates

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

The format model:

- Must be enclosed by 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

ORACLE

Displaying a Date in a Specific Format

Previously, all Oracle date values were displayed in the DD-MON-YY format. You can use the TO_CHAR function to convert a date from this default format to one that you specify.

Guidelines

- The format model must be enclosed by single quotation marks and is case sensitive.
- The format model can include any valid date format element. Be sure to separate the date value from the format model by 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';
```

	EMPLOYEE_ID	MONTH_HIRED
1	205	06/94

Elements of the Date Format Model

Element	Result
YYYY	Full year in numbers
YEAR	Year spelled out (in English)
MM	Two-digit value for 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

ORACLE

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 digits of 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 -
BC or AD	Indicates B.C. or A.D. year
B.C. or A.D.	Indicates B.C. or A.D. year using periods
Q	Quarter of year
MM	Month: two-digit value
MONTH	Name of month padded with blanks to length of nine characters
MON	Name of month, three-letter abbreviation
RM	Roman numeral month
WW or W	Week of year or month
DDD or DD or D	Day of year, month, or week
DAY	Name of day padded with blanks to a length of nine characters
DY	Name of day; three-letter abbreviation
J	Julian day; the number of days since December 31, 4713 B.C.

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 in double quotation marks:

DD "of" MONTH	12 of OCTOBER

- Number suffixes spell out numbers:

ddspth	fourteenth

ORACLE

Date Format Elements: Time Formats

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)
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	Whalen	17 September 1987
2	Hartstein	17 February 1996
3	Fay	17 August 1997
4	Higgins	7 June 1994
5	Gietz	7 June 1994

...

19	Taylor	24 March 1998
20	Grant	24 May 1999

ORACLE

Using the `TO_CHAR` Function with Dates

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

Example

Modify the slide example 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')
 HIREDATE
FROM     employees;
```

	LAST_NAME	HIREDATE
1	Whalen	Seventeenth of September 1987 12:00:00 AM
2	Hartstein	Seventeenth of February 1996 12:00:00 AM

...

20	Grant	Twenty-Fourth of May 1999 12:00:00 AM

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

ORACLE

Using the TO_CHAR Function with Numbers

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.

Using the TO_CHAR Function with Numbers (continued)

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 in the specified position the decimal character. 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.	9,999	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

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

Guidelines

- The Oracle server displays a string of number signs (#) in place of a whole number whose digits exceed the number of digits that is provided in the format model.
- The Oracle server rounds the stored decimal value to the number of decimal places that is 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 matching for the character argument and date format model of a TO_DATE function.

ORACLE

Using the TO_NUMBER and TO_DATE Functions

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 choose is based on the previously demonstrated format elements.

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

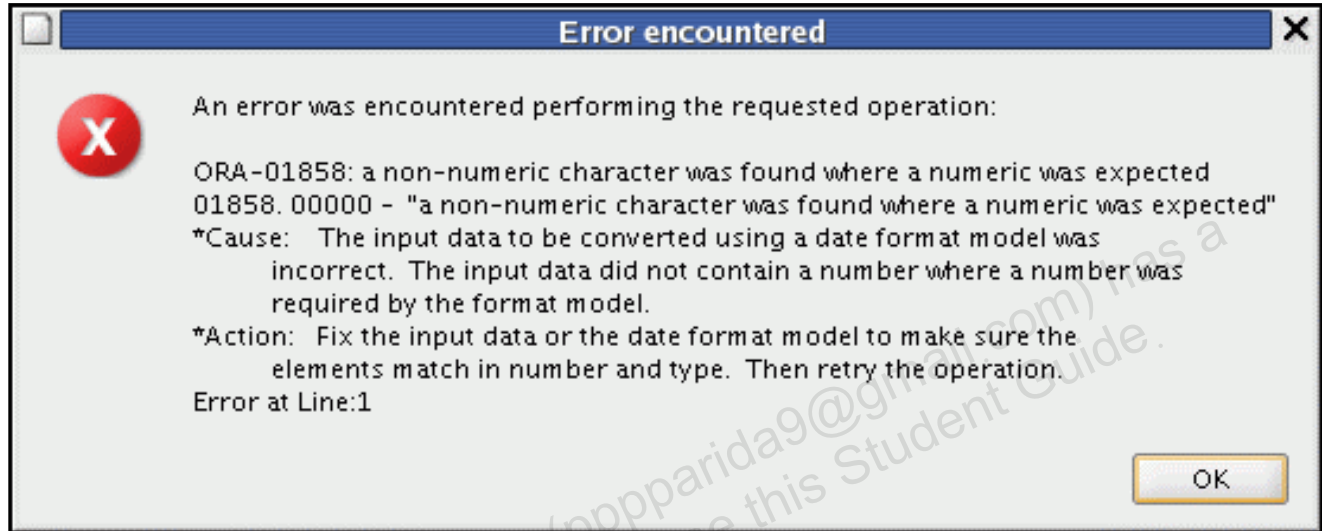- Punctuation and quoted text in the character argument must exactly match (except for case) the corresponding parts of the format model.
- The character argument cannot have extra blanks. Without fx, Oracle ignores extra blanks.
- Numeric data in the character argument must have the same number of digits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

Using the `TO_NUMBER` and `TO_DATE` Functions (continued)

Example

Display the name and hire date for all employees who started on May 24, 1999. There are two spaces after the month *May* and 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');
```

Error encountered ✕

❌ An error was encountered performing the requested operation:

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.
Error at Line:1

[OK]

To fix the above error, change `fx` to `fm` in the query.

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

	LAST_NAME		HIRE_DATE
1	Grant		24-MAY-99

RR Date Format

Current Year	Specified Date	RR Format	YY Format
1995	27-OCT-95	1995	1995
1995	27-OCT-17	2017	1917
2001	27-OCT-17	2017	2017
2001	27-OCT-95	1995	2095

		If the specified two-digit year is:	
		0–49	50–99
If two digits of the current year are:	0–49	The return date is in the current century	The return date is in the century before the current one
	50–99	The return date is in the century after the current one	The return date is in the current century

ORACLE

RR Date Format Element

The RR date format is similar to the YY element, but you can use it to specify different centuries. Use the RR date format element instead of YY so that the century of the return value varies according to the specified two-digit year and the last two digits of the current year. The table in the slide summarizes the behavior of the RR element.

Current Year	Given Date	Interpreted (RR)	Interpreted (YY)
1994	27-OCT-95	1995	1995
1994	27-OCT-17	2017	1917
2001	27-OCT-17	2017	2017

RR Date Format: Example

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	Whalen		17-Sep-1987
2	King		17-Jun-1987
3	Kochhar		21-Sep-1989

RR Date Format: Example

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.

The following command, on the other hand, 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-1990';
```

Nesting Functions

- Single-row functions can be nested to any level.
- Nested functions are evaluated from the deepest level to the least deep level.
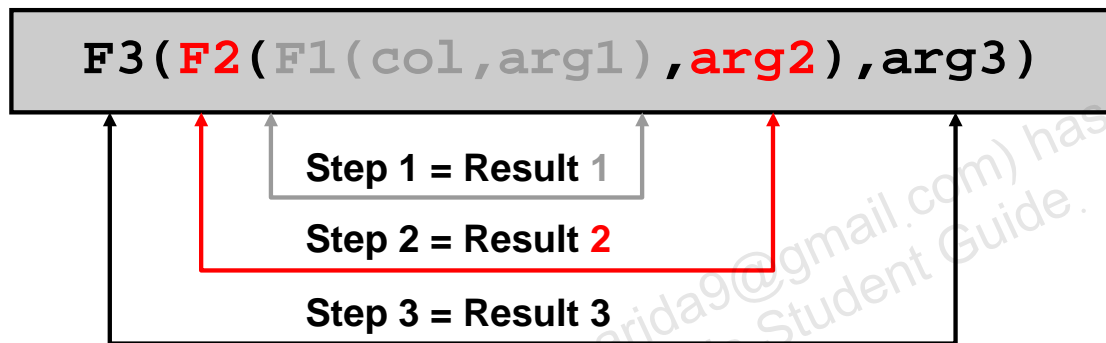
```
F3(F2(F1(col,arg1),arg2),arg3)
```

Step 1 = Result 1

Step 2 = Result 2

Step 3 = Result 3

Nesting Functions

Single-row functions can be nested to any depth. Nested functions are evaluated from the innermost level to the outermost level. Some examples follow to show you the flexibility of these functions.

Nesting Functions

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

	LAST_NAME	UPPER(CONCAT(SUBSTR(LAST_NAME,1,8),'_US'))
1	Hunold	HUNOLD_US
2	Ernst	ERNST_US
3	Lorentz	LORENTZ_US

ORACLE

Nesting Functions (continued)

The slide example 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)`

General Functions

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 "Functions" in *Oracle SQL Reference*.

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')`

NVL Function

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 to convert 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	Whalen	4400	0	52800
2	Hartstein	13000	0	156000
3	Fay	6000	0	72000
4	Higgins	12000	0	144000
5	Gietz	8300	0	99600
6	King	24000	0	288000
7	Kochhar	17000	0	204000
8	De Haan	17000	0	204000

...

ORACLE

Using the NVL Function

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;
```

	LAST_NAME	SALARY	COMMISSION_PCT	AN_SAL
1	Whalen	4400	(null)	(null)

...

17	Zlotkey	10500	0.2	151200
18	Abel	11000	0.3	171600
19	Taylor	8600	0.2	123840
20	Grant	7000	0.15	96600

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);
```

1

2

	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

1

2

ORACLE

Using the NVL2 Function

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

Syntax

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

In the syntax:
- *expr1* is the source value or expression that may contain 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 second expression of SAL+COMM is returned. If the COMMISSION_PCT column holds a null value, the third expression of SAL is returned.

The argument *expr1* can have any data type. The arguments *expr2* and *expr3* can have any data types except LONG. If the data types of *expr2* and *expr3* are different, the Oracle server converts *expr3* to the data type of *expr2* before comparing them unless *expr3* is a null constant. In the latter case, a data type conversion is not necessary. The data type of the return value is always the same as the data type of *expr2*, unless *expr2* is character data, in which case the return value's data type is VARCHAR2.

Using the NULLIF Function


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

	FIRST_NAME		expr1		LAST_NAME		expr2		RESULT
1	Ellen		5		Abel		4		5
2	Curtis		6		Davies		6		(null)
3	Lex		3		De Haan		7		3
4	Bruce		5		Ernst		5		(null)
5	Pat		3		Fay		3		(null)
6	William		7		Gietz		5		7
7	Kimberely		9		Grant		5		9
8	Michael		7		Hartstein		9		7

...

Using the 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. You cannot specify the literal NULL for the first expression.

Syntax

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

In the syntax:
- expr1 is the source value compared to expr2
- expr2 is the source value compared with expr1 (If it is not equal to expr1, expr1 is returned.)

In the example shown in the slide, 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 name is displayed.

Note: The NULLIF function is logically equivalent to the following CASE expression. The CASE expression is discussed on a subsequent page:

```
CASE WHEN expr1 = expr 2 THEN NULL ELSE expr1 END
```

Using the COALESCE Function

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

ORACLE

Using the COALESCE Function

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

All expressions must be of the same data type.

Using the COALESCE Function

```
SELECT last_name,
       COALESCE(manager_id,commission_pct, -1) comm
FROM   employees
ORDER BY commission_pct;
```

	LAST_NAME		COMM
1	Grant		149
2	Taylor		149
3	Zlotkey		100
4	Abel		149
5	King		-1
6	Kochhar		100
7	De Haan		100
8	Hunold		102

...

ORACLE

Using the COALESCE Function (continued)

In the example shown in the slide, if the MANAGER_ID value is not null, it is displayed. If the MANAGER_ID value is null, then the COMMISSION_PCT is displayed. If the MANAGER_ID and COMMISSION_PCT values are null, then the value –1 is displayed.

Conditional Expressions

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

Conditional Expressions

Two methods 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 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 Expression

CASE expressions let you use 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, then the Oracle server returns else_expr. Otherwise, the Oracle server returns null. You cannot specify the literal NULL for all the return_exprs and the else_expr.

All of the expressions (expr, comparison_expr, and return_expr) must be of the same data type, which can be CHAR, VARCHAR2, NCHAR, or NVARCHAR2.

Using the CASE Expression

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

```
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
11	Lorentz	IT_PROG	4200	4620
12	Mourgos	ST_MAN	5800	5800
13	Rajs	ST_CLERK	3500	4025
14	Davies	ST_CLERK	3100	3565
15	Matos	ST_CLERK	2600	2990
16	Vargas	ST_CLERK	2500	2875
17	Zlotkey	SA_MAN	10500	10500
18	Abel	SA_REP	11000	13200
19	Taylor	SA_REP	8600	10320
20	Grant	SA_REP	7000	8400

ORACLE

Using the CASE Expression

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.

The following is an example of a 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, 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:

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

DECODE Function

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
...				
11	Lorentz	IT_PROG	4200	4620
12	Mourgos	ST_MAN	5800	5800
13	Rajs	ST_CLERK	3500	4025
14	Davies	ST_CLERK	3100	3565
15	Matos	ST_CLERK	2600	2990
16	Vargas	ST_CLERK	2500	2875
17	Zlotkey	SA_MAN	10500	10500
18	Abel	SA_REP	11000	13200
19	Taylor	SA_REP	8600	10320
20	Grant	SA_REP	7000	8400

Using the DECODE Function

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;
```

ORACLE

Using the DECODE function (continued)

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%

	LAST_NAME	SALARY	TAX_RATE
1	Zlotkey	10500	0.42
2	Abel	11000	0.42
3	Taylor	8600	0.4

Summary

In this lesson, you should have learned how to:

- Perform calculations on data using functions
- Modify individual data items using functions
- Manipulate output for groups of rows using functions
- Alter date formats for display using functions
- Convert column data types using functions
- Use NVL functions
- Use IF-THEN-ELSE logic

Summary

Single-row functions can be nested to any level. Single-row functions can manipulate the following:
- Character data: LOWER, UPPER, INITCAP, CONCAT, SUBSTR, INSTR, LENGTH
- Number data: ROUND, TRUNC, MOD
- Date data: MONTHS_BETWEEN, ADD_MONTHS, NEXT_DAY, LAST_DAY, ROUND, TRUNC

Remember the following:
- Date values can also use arithmetic operators.
- 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.
- IF-THEN-ELSE logic can be applied within a SQL statement by using the CASE expression or the DECODE function.

SYSDATE and DUAL

SYSDATE is a date function that returns the current date and time. It is customary to select SYSDATE from a dummy table called DUAL.

Practice 3: Overview of Part 2

This practice covers the following topics:

- Creating queries that require the use of numeric, character, and date functions
- Using concatenation with functions
- Writing non-case-sensitive queries to test the usefulness of character functions
- Performing calculations of years and months of service for an employee
- Determining the review date for an employee

ORACLE

Practice 3: Overview of Part 2

Part 2 of this lesson's practice provides a variety of exercises that use the different functions that are available for character, number, and date data types. For Part 2, complete exercises 7–14.

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

Practice 3

Part 1

1. Write a query to display the current date. Label the column `Date`.

	Date
1	06-NOV-08

2. The HR department needs a report to display the employee number, last name, salary, and salary increased by 15.5% (expressed as a whole number) for each employee. Label the column `New Salary`. Place your SQL statement in a text file named `lab_03_02.sql`.

3. Run your query in the `lab_03_02.sql` file.

	EMPLOYEE_ID	LAST_NAME	SALARY	New Salary
1	200	Whalen	4400	5082
2	201	Hartstein	13000	15015
3	202	Fay	6000	6930
4	205	Higgins	12000	13860
5	206	Gietz	8300	9587

. . .

20	178	Grant	7000	8085

4. Modify your `lab_03_02.sql` query to add a column that subtracts the old salary from the new salary. Label the column `Increase`. Save the contents of the file as `lab_03_04.sql`. Run the revised query.

	EMPLOYEE_ID	LAST_NAME	SALARY	New Salary	Increase
1	200	Whalen	4400	5082	682
2	201	Hartstein	13000	15015	2015
3	202	Fay	6000	6930	930
4	205	Higgins	12000	13860	1860
5	206	Gietz	8300	9587	1287
6	100	King	24000	27720	3720
7	101	Kochhar	17000	19635	2635
8	102	De Haan	17000	19635	2635

. . .

20	178	Grant	7000	8085	1085

Practice 3 (continued)

5. Write a query that displays the last name (with the first letter uppercase and all other letters lowercase) and the length of the last name for all employees whose name starts with the letters *J, A,* or *M*. Give each column an appropriate label. Sort the results by the last names of the employees.

	Name		Length
1	Abel		4
2	Matos		5
3	Mourgos		7

Rewrite the query so that the user is prompted to enter a letter that starts the last name. For example, if the user enters H when prompted for a letter, the output should show all employees whose last name starts with the letter *H*.

	Name		Length
1	Hartstein		9
2	Higgins		7
3	Hunold		6

6. The HR department wants to find the duration of employment for each employee. For each employee, display the last name and calculate the number of months between today and the date on which the employee was hired. Label the column MONTHS_WORKED. Order your results by the number of months employed. Round the number of months up to the closest whole number. **Note:** Your results will differ.

	LAST_NAME		MONTHS_WORKED
1	Zlotkey		105
2	Mourgos		108
3	Grant		113
4	Lorentz		117
5	Vargas		124
6	Taylor		127
7	Matos		128
8	Fay		135
9	Davies		141
10	Abel		150

. . .

20	King		257

Practice 3 (continued)

Part 2

7. 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.
4	Higgins earns $12,000.00 monthly but wants $36,000.00.

■ ■ ■

17	Zlotkey earns $10,500.00 monthly but wants $31,500.00.
18	Abel earns $11,000.00 monthly but wants $33,000.00.
19	Taylor earns $8,600.00 monthly but wants $25,800.00.
20	Grant earns $7,000.00 monthly but wants $21,000.00.

If you have time, complete the following exercises:

8. Create a query to display the last name and salary for all employees. Format the salary to be 15 characters long, left-padded with the $ symbol. Label the column `SALARY`.

	LAST_NAME	SALARY
1	Whalen	$$$$$$$$$$$4400
2	Hartstein	$$$$$$$$$$13000
3	Fay	$$$$$$$$$$$6000
4	Higgins	$$$$$$$$$$12000
5	Gietz	$$$$$$$$$$$8300
6	King	$$$$$$$$$$24000
7	Kochhar	$$$$$$$$$$17000

■ ■ ■

16	Vargas	$$$$$$$$$$$2500
17	Zlotkey	$$$$$$$$$$10500
18	Abel	$$$$$$$$$$11000
19	Taylor	$$$$$$$$$$$8600
20	Grant	$$$$$$$$$$$7000

Practice 3 (continued)

9. 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
4	Higgins	07-JUN-94	Monday, the Twelfth of December, 1994
5	Gietz	07-JUN-94	Monday, the Twelfth of December, 1994
6	King	17-JUN-87	Monday, the Twenty-First of December, 1987

. . .

17	Zlotkey	29-JAN-00	Monday, the Thirty-First of July, 2000
18	Abel	11-MAY-96	Monday, the Eighteenth of November, 1996
19	Taylor	24-MAR-98	Monday, the Twenty-Eighth of September, 1998
20	Grant	24-MAY-99	Monday, the Twenty-Ninth of November, 1999

10. Display the last name, hire date, and day of the week on which an 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	Rajs	17-OCT-95	TUESDAY
5	Mourgos	16-NOV-99	TUESDAY
6	Gietz	07-JUN-94	TUESDAY
7	Higgins	07-JUN-94	TUESDAY
8	De Haan	13-JAN-93	WEDNESDAY

. . .

16	Zlotkey	29-JAN-00	SATURDAY
17	Hartstein	17-FEB-96	SATURDAY
18	Lorentz	07-FEB-99	SUNDAY
19	Matos	15-MAR-98	SUNDAY
20	Fay	17-AUG-97	SUNDAY

Practice 3 (continued)

If you want an extra challenge, complete the following exercises:

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

	LAST_NAME	COMM
1	Whalen	No Commission
2	Hartstein	No Commission
3	Fay	No Commission

. . .

14	Davies	No Commission
15	Matos	No Commission
16	Vargas	No Commission
17	Zlotkey	.2
18	Abel	.3
19	Taylor	.2
20	Grant	.15

12. Create a query that displays the first eight characters of the employees' last names and indicates the amounts of their salaries with asterisks. Each asterisk signifies a thousand dollars. Sort the data in descending order of salary. Label the column EMPLOYEES_AND_THEIR_SALARIES.

	EMPLOYEES_AND_THEIR_SALARIES
1	King **************************
2	Kochhar *******************
3	De Haan *******************
4	Hartstei **************
5	Higgins *************
6	Abel ************
7	Zlotkey ***********

. . .

17	Rajs	****
18	Davies	****
19	Matos	***
20	Vargas	***

Practice 3 (continued)

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

Job	*Grade*
AD_PRES	A
ST_MAN	B
IT_PROG	C
SA_REP	D
ST_CLERK	E
None of the above	0

	JOB_ID	GRADE
1	AC_ACCOUNT	0
2	AC_MGR	0
3	AD_ASST	0
4	AD_PRES	A
5	AD_VP	0
6	AD_VP	0
7	IT_PROG	C
8	IT_PROG	C
9	IT_PROG	C
10	MK_MAN	0
11	MK_REP	0
12	SA_MAN	0
13	SA_REP	D
14	SA_REP	D
15	SA_REP	D
16	ST_CLERK	E
17	ST_CLERK	E
18	ST_CLERK	E
19	ST_CLERK	E
20	ST_MAN	B

14. Rewrite the statement in the preceding exercise using the CASE syntax.