

3

Single-Row Functions

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Schedule:	Timing	Topic
	55 minutes	Lecture
	30 minutes	Practice
	85 minutes	Total

Objectives

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

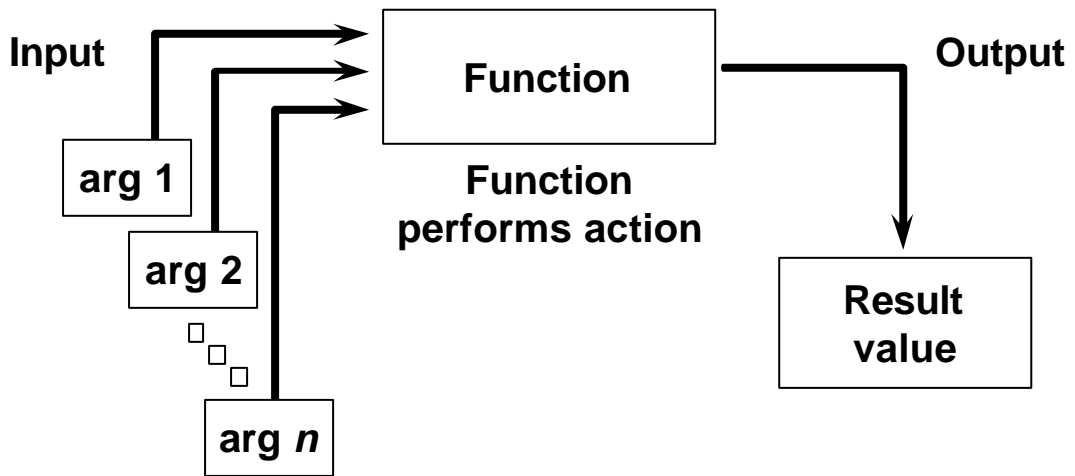
- **Describe various types of functions available in SQL**
- **Use character, number, and date functions in `SELECT` statements**
- **Describe the use of conversion functions**

ORACLE

Lesson Aim

Functions make the basic query block more powerful and 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, character data to numeric data.

SQL Functions



ORACLE

SQL Functions

Functions are a very powerful feature of SQL and 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

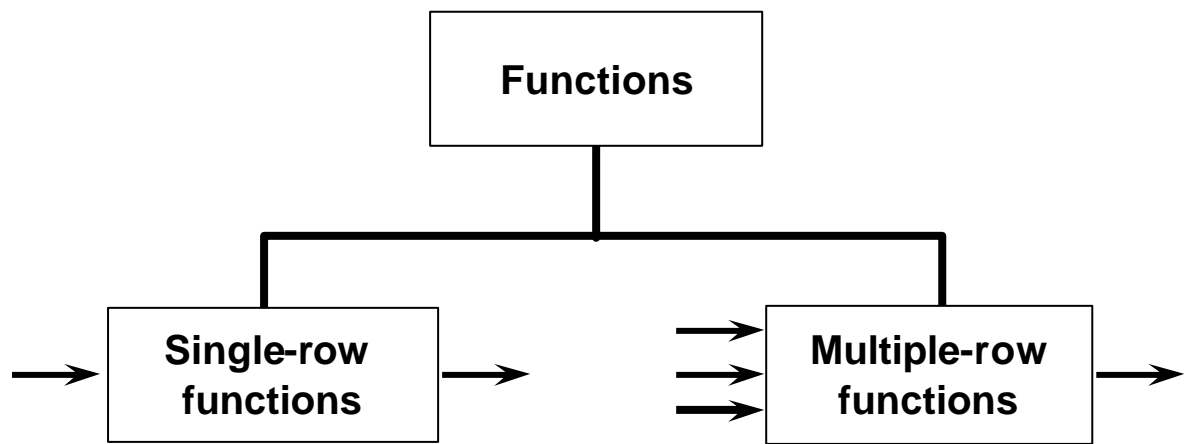
SQL functions sometimes take arguments and always return a value.

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

Instructor Note

This lesson does not discuss all functions in great detail. It presents the most common functions with a brief explanation of them.

Two Types of SQL Functions



ORACLE

SQL Functions (continued)

There are two distinct 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

Multiple -Row Functions

Functions can manipulate groups of rows to give one result per group of rows. These functions are known as group functions. This is covered in a later lesson.

For more information, see *Oracle9i SQL Reference* for the complete list of available functions and their syntax.

Single-Row Functions

Single row functions:

- **Manipulate data items**
- **Accept arguments and return one value**
- **Act on each row returned**
- **Return one result per row**
- **May modify the data type**
- **Can be nested**
- **Accept arguments which 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 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:

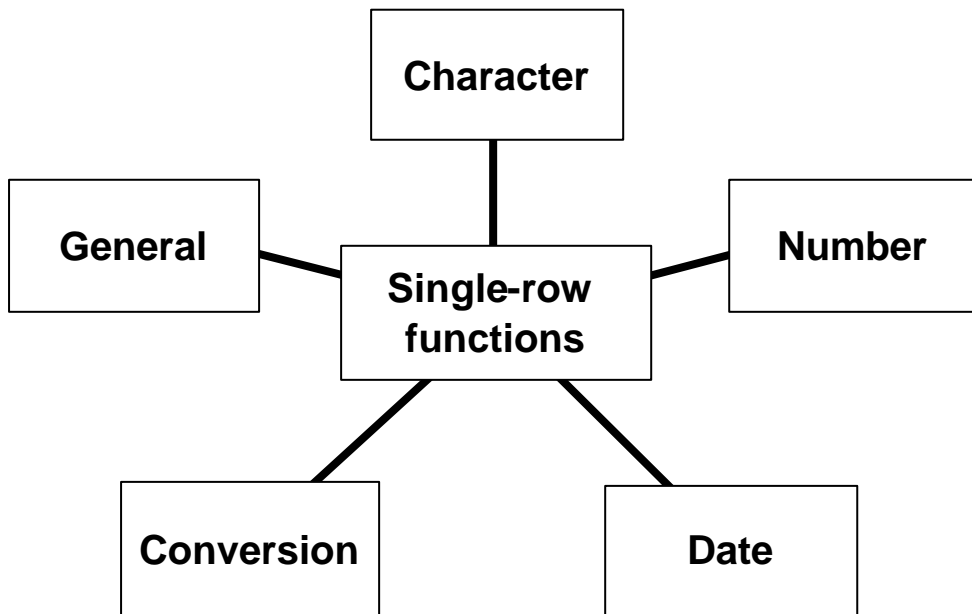
- Acting on each row returned in the query
- Returning one result per row
- Possibly returning a data value of a different type than that referenced
- Possibly expecting one or more arguments
- Can be used in `SELECT`, `WHERE`, and `ORDER BY` clauses; can be nested

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



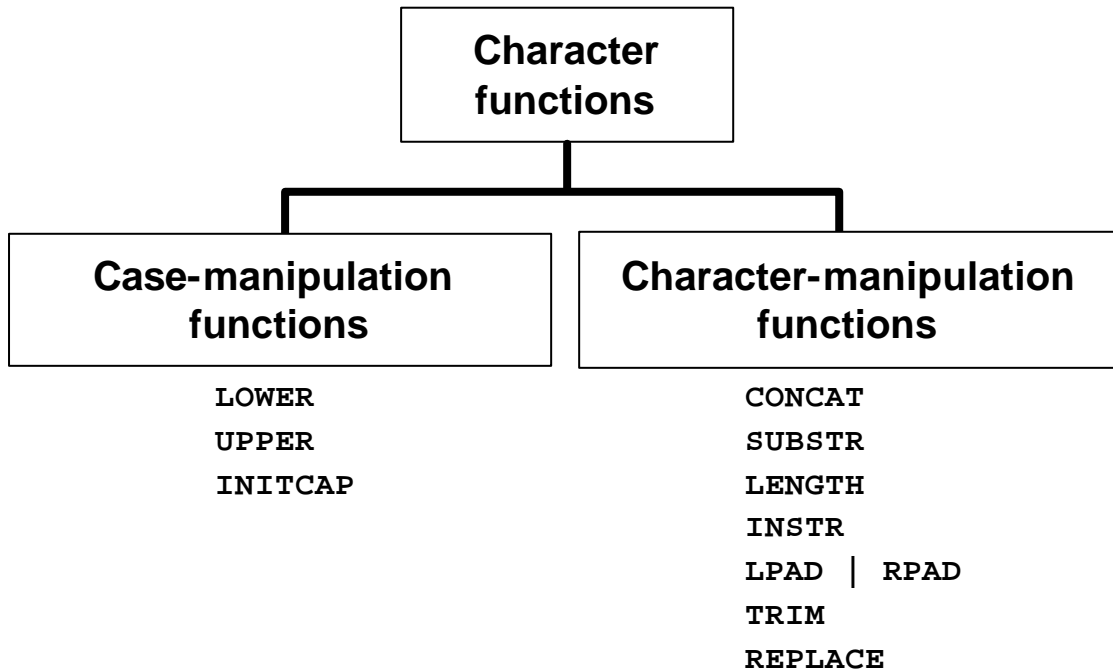
ORACLE

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
 - COALSECE
 - CASE
 - DECODE

Character Functions



ORACLE

3-7

Copyright © Oracle Corporation, 2001. All rights reserved.

Character Functions

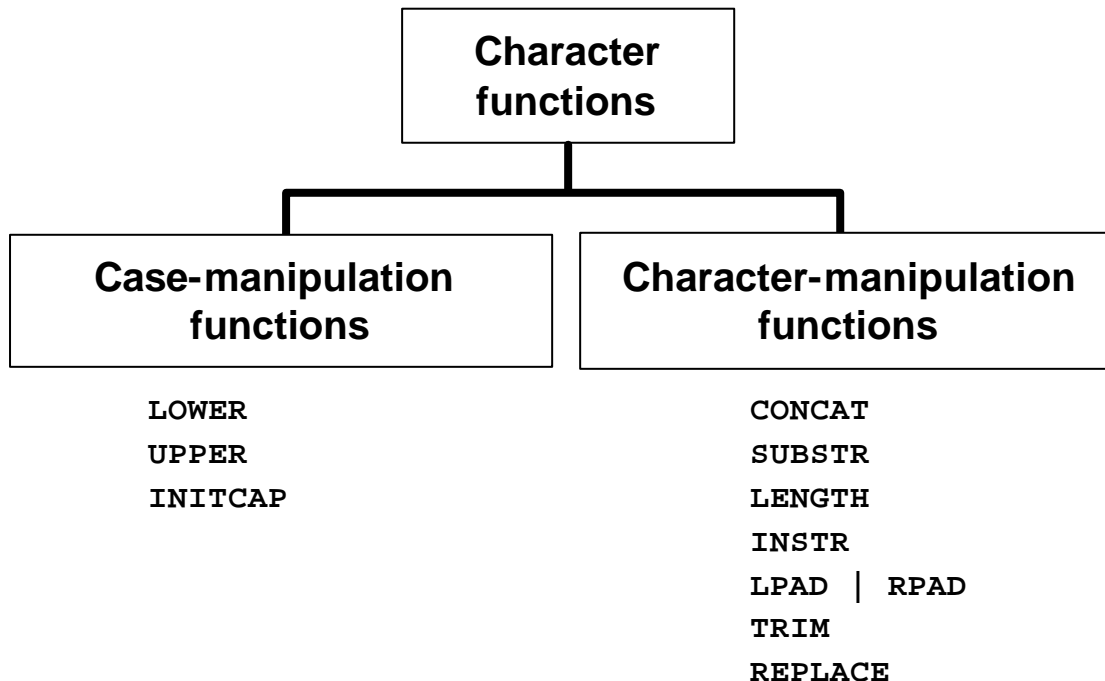
Single-row character functions accept character data as input and can return both character and numeric values. Character functions can be divided into the following:

- Case-manipulation functions
- Character-manipulation functions

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



ORACLE

Character Functions (continued)

Function	Purpose
LENGTH(<i>column</i> / <i>expression</i>)	Returns the number of characters in the expression
INSTR(<i>column</i> / <i>expression</i> , ' <i>string</i> ', [, <i>m</i>], [<i>n</i>])	Returns the numeric position of a named string. Optionally, you can provide a position <i>m</i> to start searching, and the occurrence <i>n</i> of the string. <i>m</i> and <i>n</i> default to 1, meaning start the search at the beginning of the search and report the first occurrence.
LPAD(<i>column</i> <i>expression</i> , <i>n</i> , ' <i>string</i> ') RPAD(<i>column</i> <i>expression</i> , <i>n</i> , ' <i>string</i> ')	Pads the character value right-justified to a total width of <i>n</i> character positions Pads the character value left-justified to a total width of <i>n</i> character positions
TRIM(<i>leading</i> / <i>trailing</i> / <i>both</i> , <i>trim_character</i> FROM <i>trim_source</i>)	Enables you to trim heading or trailing characters (or both) from a character string. If <i>trim_character</i> or <i>trim_source</i> is a character literal, you must enclose it in single quotes. This is a feature available from Oracle8i and later.
REPLACE(<i>text</i> , <i>search_string</i> , <i>replacement_string</i>)	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

3-9

Copyright © Oracle Corporation, 2001. All rights reserved.

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
The job id for KING is ad_pres
The job id for KOCHHAR is ad_vp
The job id for DE HAAN is ad_vp
■ ■ ■
The job id for HIGGINS is ac_mgr
The job id for GIETZ is ac_account

20 rows selected.

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
205	Higgins	110

ORACLE

Case Manipulation Functions (continued)

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. Since both names are lowercase now, 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 capitalized, 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
<code>CONCAT('Hello', 'World')</code>	HelloWorld
<code>SUBSTR('HelloWorld',1,5)</code>	Hello
<code>LENGTH('HelloWorld')</code>	10
<code>INSTR('HelloWorld', 'W')</code>	6
<code>LPAD(salary,10,'*')</code>	*****24000
<code>RPAD(salary, 10, '*')</code>	24000*****
<code>TRIM('H' FROM 'HelloWorld')</code>	elloWorld

ORACLE

Character Manipulation Functions

CONCAT, SUBSTR, LENGTH, INSTR, LPAD, RPAD, and TRIM are the character manipulation functions 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 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 quotes.)

Instructor Note

Be sure to point out RPAD to the students, because this function is needed in a practice exercise. Also, TRIM, which was a new function in Oracle8i, does the job of both the LTRIM and the RTRIM functions.

Using the Character-Manipulation Functions

```
SELECT employee_id, CONCAT(first_name, last_name) NAME,
       job_id, LENGTH (last_name),
       INSTR(last_name, 'a') "Contains 'a'?"
FROM   employees
WHERE  SUBSTR(job_id, 4) = 'REP';
```

EMPLOYEE_ID	NAME	JOB_ID	LENGTH(LAST_NAME)	Contains 'a'?
174	EllenAbel	SA_REP	4	0
176	JonathonTaylor	SA_REP	6	2
178	KimberelyGrant	SA_REP	5	3
202	PatFay	MK_REP	3	2

ORACLE

3-12

Copyright © Oracle Corporation, 2001. All rights reserved.

Character-Manipulation Functions (continued)

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 on the slide to display the data for those employees whose last names end with an *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'?
102	LexDe Haan	7	5
200	JenniferWhalen	6	3
201	MichaelHartstein	9	2

Number Functions

- **ROUND:** Rounds value to specified decimal

`ROUND(45.926, 2)` \longrightarrow 45.93

- **TRUNC:** Truncates value to specified decimal

`TRUNC(45.926, 2)` \longrightarrow 45.92

- **MOD:** Returns remainder of division

`MOD(1600, 300)` \longrightarrow 100

ORACLE

Number Functions

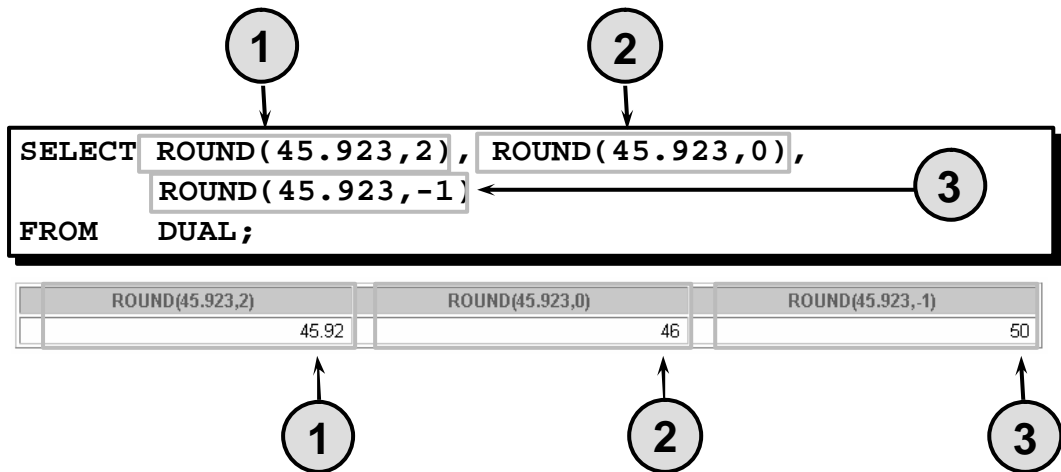
Number functions accept numeric input and return numeric values. This section describes some of the number functions.

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

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

For more information, see *Oracle9i SQL Reference*, “Number Functions.”

Using the ROUND Function



DUAL is a dummy table you can use to view results from functions and calculations.

ORACLE

ROUND Function

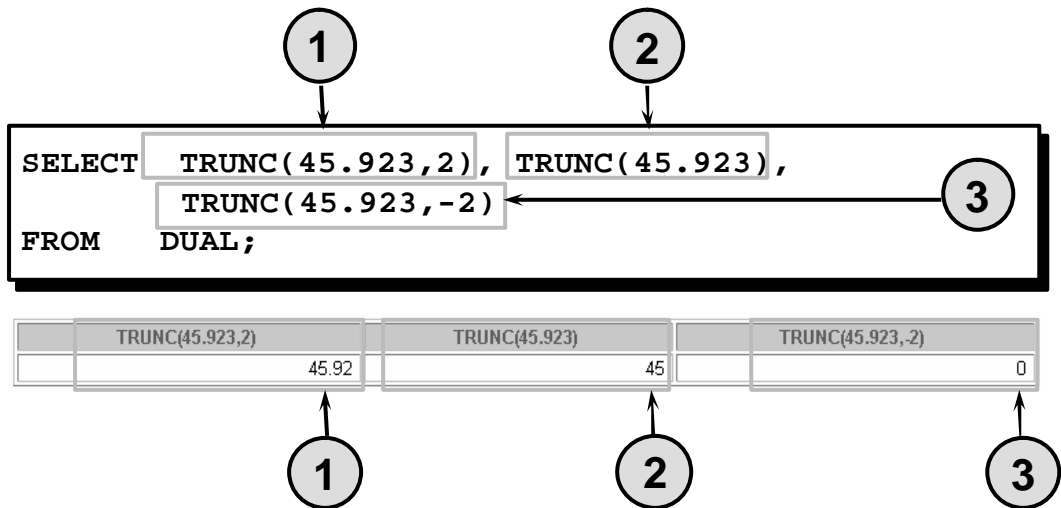
The ROUND function rounds the column, expression, or value to *n* decimal places. If the second argument is 0 or is missing, the value is rounded to zero decimal places. If the second argument is 2, the value is rounded to two decimal places. Conversely, if the second argument is -2, the value is rounded to two decimal places to the left.

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

The 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 instance, 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



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.

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

Using the MOD Function

Calculate the remainder of a salary after it is divided by 5000 for all employees whose job title is sales representative.

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

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

ORACLE

MOD Function

The MOD function finds the remainder of value1 divided by value2. 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.

Instructor Note (for page 3-17)

You can change the default date display setting for a user session by executing the command:

```
ALTER SESSION SET NLS_DATE_FORMAT = 'date format model';
```

The DBA can set the date format for a database to a different format from the default. In either case, changing these settings is usually not a developer's role.

Working with Dates

- Oracle database stores dates in an internal numeric format: century, year, month, day, hours, minutes, seconds.
- The default date display format is DD-MON-RR.
 - Allows you to store 21st century dates in the 20th century by specifying only the last two digits of the year.
 - Allows you to store 20th century dates in the 21st century in the same way.

```
SELECT last_name, hire_date
FROM employees
WHERE last_name like 'G%';
```

LAST_NAME	HIRE_DATE
Gietz	07-JUN-94
Grant	24-MAY-99

ORACLE

Oracle Date Format

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 for the employee Gietz 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 07-JUN-94 is displayed as day, month, and year, there is also *time* and *century* information associated with it. The complete data might be June 7th, 1994 5:10:43 p.m.

This data is stored internally as follows:

CENTURY	YEAR	MONTH	DAY	HOUR	MINUTE	SECOND
19	94	06	07	5	10	43

Centuries and the Year 2000

The Oracle server is year 2000 compliant. 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 1996 or 2001, and not just as 96 or 01.

Working with Dates

SYSDATE is a function that returns:

- **Date**
- **Time**

ORACLE

3-18

Copyright © Oracle Corporation, 2001. All rights reserved.

The 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
28-SEP-01

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

Since 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
King	744.245395
Kochhar	626.102538
De Haan	453.245395

ORACLE

Arithmetic with Dates (continued)

The example on 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	Description
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 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 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

- **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 employed for fewer than 36 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) < 36;
```

EMPLOYEE_ID	HIRE_DATE	TENURE	REVIEW	NEXT_DAY(LAST_DAY(
107	07-FEB-99	31.6982407	07-AUG-99	12-FEB-99	28-FEB-99
124	16-NOV-99	22.4079182	16-MAY-00	19-NOV-99	30-NOV-99
149	29-JAN-00	19.9885633	29-JUL-00	04-FEB-00	31-JAN-00
178	24-MAY-99	28.1498536	24-NOV-99	28-MAY-99	31-MAY-99

Using Date Functions

Assume SYSDATE = '25-JUL-95':

- **ROUND(SYSDATE, 'MONTH') → 01-AUG-95**
- **ROUND(SYSDATE, 'YEAR') → 01-JAN-96**
- **TRUNC(SYSDATE, 'MONTH') → 01-JUL-95**
- **TRUNC(SYSDATE, 'YEAR') → 01-JAN-95**

ORACLE

3-23

Copyright © Oracle Corporation, 2001. All rights reserved.

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(HIR)	TRUNC(HIR)
142	29-JAN-97	01-FEB-97	01-JAN-97
202	17-AUG-97	01-SEP-97	01-AUG-97

Instructor note

If the format model is month, dates 1-15 result in the first day of the current month. Dates 16-31 result in the first day of the next month. If the format model is year, months 1-6 result with January 1st of the current year. Months 7-12 result in January 1st of the next year.

This is a good point to break the lesson in half. Have the students do Practice 3 - Part 1 (1-5) now.

Practice 3, Part One: Overview

This practice covers the following topics:

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

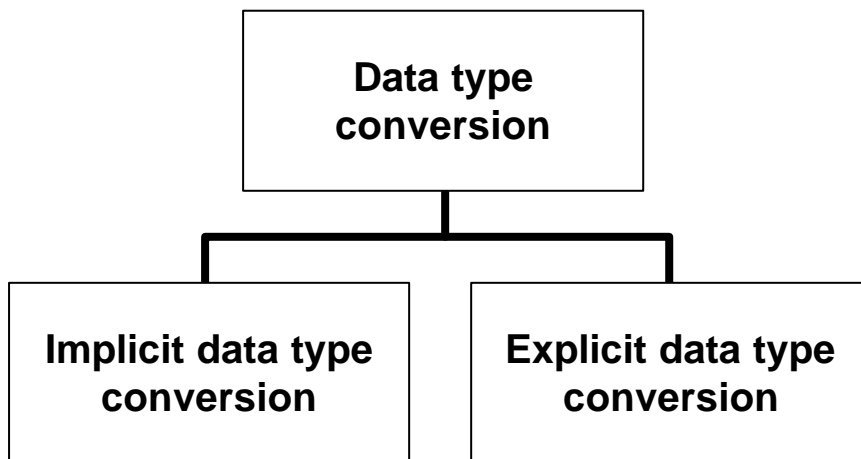
ORACLE

Practice 3, Part One: Overview

This practice is designed to give you a variety of exercises using different functions available for character, number, and date data types.

Complete questions 1-5 at the end of this lesson.

Conversion Functions



ORACLE

Conversion Functions

In addition to Oracle data types, columns of tables in an Oracle9i 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, Oracle server uses data of one data type where it expects data of a different data type. When this happens, Oracle server can automatically convert the data to the expected data type. This data type conversion can be done *implicitly* by Oracle server, or *explicitly* by the user.

Implicit data type conversions work according to the rules 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 last 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

ORACLE

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.

Instructor Note

There are several new data types available in the Oracle9i release pertaining to time. These include: `TIMESTAMP`, `TIMESTAMP WITH TIME ZONE`, `TIMESTAMP WITH LOCAL TIME ZONE`, `INTERVAL YEAR`, `INTERVAL DAY`. These are discussed later in the course.

You can also refer students to the *Oracle9i SQL Reference*, “Basic Elements of Oracle SQL.”

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)

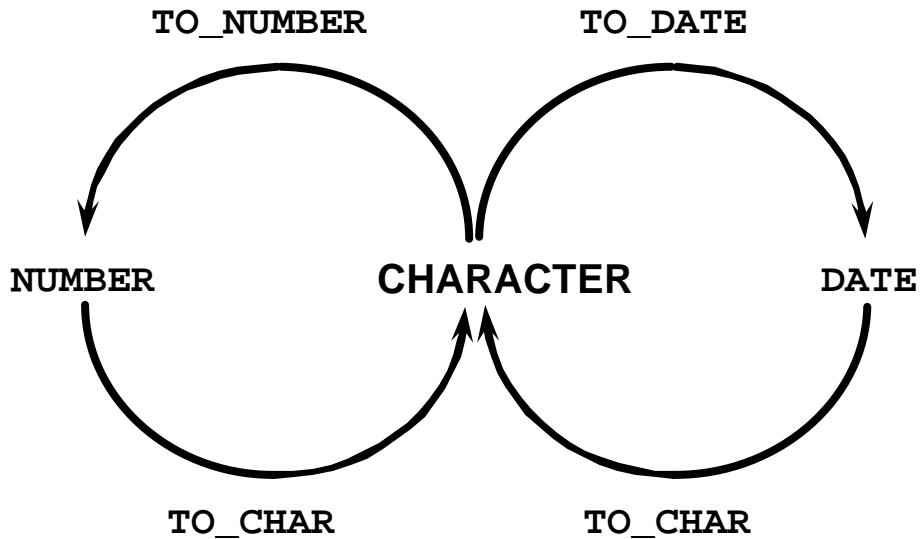
In general, the Oracle server uses the rule for expressions when a data type conversion is needed in places not covered by a rule for assignment conversions.

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

Instructor Note

Implicit data conversion is not solely performed on the data types mentioned. Other implicit data conversions can also be done. For example, VARCHAR2 can be implicitly converted to ROWID.

Explicit Data Type Conversion



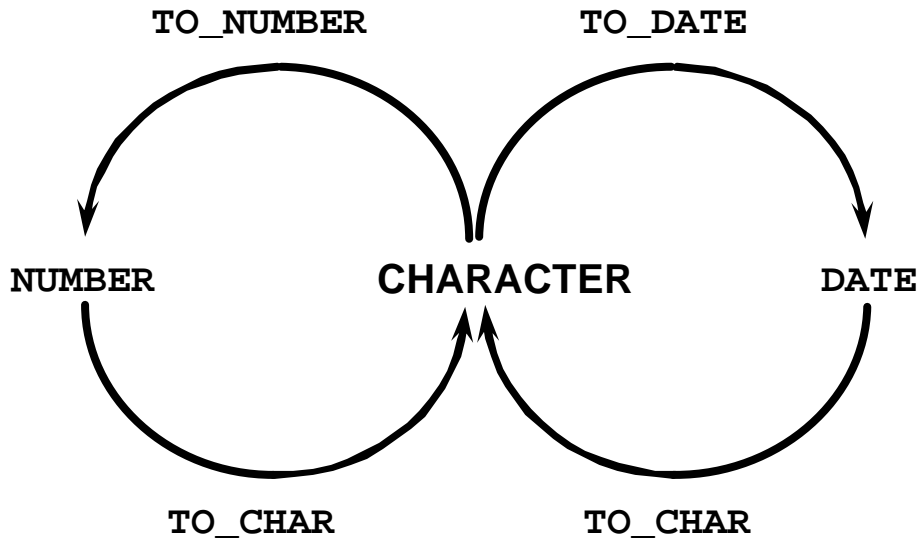
ORACLE

Explicit Data Type Conversion

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

Function	Purpose
<code>TO_CHAR(<i>number</i> <i>date</i>, [<i>fmt</i>], [<i>nlsparams</i>])</code>	<p>Converts a number or date value to a <code>VARCHAR2</code> character string with format model <i>fmt</i>.</p> <p>Number Conversion: The <i>nlsparams</i> parameter specifies the following characters, which are returned by number format elements:</p> <ul style="list-style-type: none">• Decimal character• Group separator• Local currency symbol• International currency symbol <p>If <i>nlsparams</i> or any other parameter is omitted, this function uses the default parameter values for the session.</p>

Explicit Data Type Conversion



ORACLE

Explicit Data Type Conversion (continued)

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

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

For more information, see *Oracle9i SQL Reference*, “Conversion Functions.”

Using the TO_CHAR Function with Dates

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

The format model:

- **Must be enclosed in single quotation marks and 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

3-31

Copyright © Oracle Corporation, 2001. All rights reserved.

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 specified by you.

Guidelines

- The format model must be enclosed in 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.
- You can format the resulting character field with the *iSQL*Plus* COLUMN command covered in a later lesson.

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

EMPLOYEE_ID	MONTH
205	06/94

Elements of the Date Format Model

YYYY	Full year in numbers
YEAR	Year spelled out
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

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	B.C./D. indicator
B.C. or A.D.	B.C./A.D. indicator with 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 31 December 4713 B.C.

Instructor Note

Emphasize the format D, as the students need it for practice 10. The D format returns a value from 1 to 7 representing the day of the week. Depending on the NLS date setting options, the value 1 may represent Sunday or Monday. In the United States, the value 1 represents Sunday.

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 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)

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
King	17 June 1987
Kochhar	21 September 1989
De Haan	13 January 1993
Hunold	3 January 1990
Ernst	21 May 1991
Lorentz	7 February 1999
Mourgos	16 November 1999

...
20 rows selected.

ORACLE

3-36

Copyright © Oracle Corporation, 2001. All rights reserved.

The TO_CHAR Function with Dates

The SQL statement on 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 Seventh of June 1994 12:00:00 AM.

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

LAST_NAME	HIREDATE
King	Seventeenth of June 1987 12:00:00 AM
Kochhar	Twenty-First of September 1989 12:00:00 AM
Higgins	Seventh of June 1994 12:00:00 AM
Gietz	Seventh of June 1994 12:00:00 AM

20 rows selected.
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 you can use with the TO_CHAR function to display a number value as a character:

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 thousand indicator

ORACLE

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.

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
.	Decimal point in position specified	999999.99	1234.00
,	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
V	Multiply by 10 <i>n</i> times (<i>n</i> = number of 9s after V)	9999V99	123400
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
\$6,000.00

ORACLE

Guidelines

- The Oracle server displays a string of hash 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 spaces provided in the format model.

Instructor Note (for page 3-39)

You can demonstrate the code using the `fx` modifier in the file 3_39n. Run the file with the `fx` modifier present, then remove the `fx` modifier and run the statement again.

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

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 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 zeroes.

Example

Display the names and hire dates of all the employees who joined on May 24, 1999. 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');
```

```
WHERE hire_date = TO_DATE('May 24, 1999', 'fxMonth DD, YYYY')
      *
```

ERROR at line 3:

ORA-01858: a non-numeric character was found where a numeric was expected

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

3-41

Copyright © Oracle Corporation, 2001. All rights reserved.

The RR Date Format Element

The RR date format is similar to the YY element, but you can use it to specify different centuries. You can 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 on 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

Instructor Note

RR is available in Oracle7, not Oracle version 6. NLS parameters can be added to the `init.ora` file to set default date formats and language names and abbreviations. For more information, see *Oracle9i SQL Reference*, “ALTER SESSION”.

Demo: `3_hire.sql`

Purpose: To illustrate date format model elements.

Example of RR Date Format

To find employees hired prior to 1990, use the RR 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(HIR
King	17-Jun-1987
Kochhar	21-Sep-1989
Whalen	17-Sep-1987

ORACLE

The RR Date Format Element Example

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

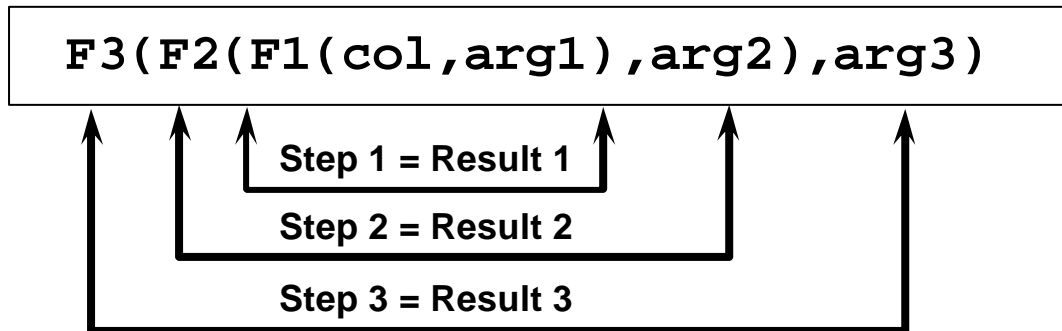
The following command, on the other hand, results in no rows being selected because the YY format interprets the year portion of the date in the current century (2090).

```
SELECT last_name, TO_CHAR(hire_date, 'DD-Mon-yyyy')
FROM   employees
WHERE  TO_DATE(hire_date, 'DD-Mon-yy') < '01-Jan-1990';
```

no rows selected

Nesting Functions

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



ORACLE

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,  
       NVL(TO_CHAR(manager_id), 'No Manager')  
FROM   employees  
WHERE  manager_id IS NULL;
```

LAST_NAME	NVL(TO_CHAR(MANAGER_ID), 'NOMANAGER')
King	No Manager

ORACLE

Nesting Functions (continued)

The slide example displays the head of the company, who has no manager. The evaluation of the SQL statement involves two steps:

1. Evaluate the inner function to convert a number value to a character string.
 - Result1 = TO_CHAR(manager_id)
2. Evaluate the outer function to replace the null value with a text string.
 - NVL(Result1, 'No Manager')

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;
```

Instructor Note

Demo: 3_nest.sql

Purpose: To illustrate nesting of several single row functions

General Functions

These 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*)

ORACLE

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 <i>expr1</i> is not null, NVL2 returns <i>expr2</i> . If <i>expr1</i> is null, NVL2 returns <i>expr3</i> . The argument <i>expr1</i> can have any data type.
NULLIF	Compares two expressions and returns null if they are equal, or the first expression if they are not equal
COALESCE	Returns the first non-null expression in the expression list

Note: For more information on the hundreds of functions available, see *Oracle9i SQL Reference*, “Functions.”

NVL Function

Converts a null 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')`

ORACLE

The 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	<code>NVL(number_column,9)</code>
DATE	<code>NVL(date_column, '01-JAN-95')</code>
CHAR or VARCHAR2	<code>NVL(character_column, 'Unavailable')</code>

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
King	24000	0	288000
Kochhar	17000	0	204000
De Haan	17000	0	204000
Hunold	9000	0	108000
Ernst	6000	0	72000
Lorentz	4200	0	50400
Mourgos	5800	0	69600
Rajs	3500	0	42000

...

20 rows selected.

ORACLE

3-47

Copyright © Oracle Corporation, 2001. All rights reserved.

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 it.

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

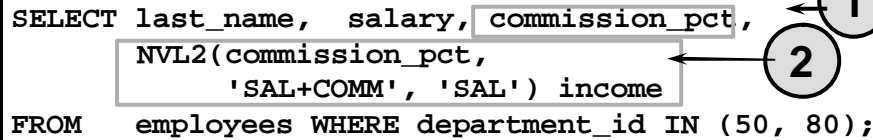
LAST_NAME	SALARY	COMMISSION_PCT	AN_SAL
Vargas	2500		
Zlotkey	10500	.2	151200
Abel	11000	.3	171600
Taylor	8600	.2	123840
Gietz	8300		

20 rows selected.

Notice that the annual compensation is calculated only for 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 on the slide, the NVL function is used to convert null values to zero.

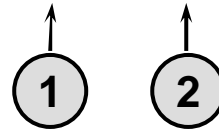
Using the NVL2 Function

```
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
Zlotkey	10500	.2	SAL+COMM
Abel	11000	.3	SAL+COMM
Taylor	8600	.2	SAL+COMM
Mourgos	5800		SAL
Rajs	3500		SAL
Davies	3100		SAL
Matos	2600		SAL
Vargas	2500		SAL

8 rows selected.



ORACLE

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

```
NVL(expr1, expr2, expr3)
```

In the syntax:

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

In the example shown, 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 values, 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 that 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

1

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

2

3

FIRST_NAME	expr1	LAST_NAME	expr2	RESULT
Steven	6	King	4	6
Neena	5	Kochhar	7	5
Lex	3	De Haan	7	3
Alexander	9	Hunold	6	9
Bruce	5	Ernst	5	
Diana	5	Lorentz	7	5
Kevin	5	Mourgos	7	5
Trenna	6	Rajs	4	6
Curtis	6	Davies	6	

...
20 rows selected.



ORACLE

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 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, the job ID in the EMPLOYEES table is compared to the job ID in the JOB_HISTORY table for any employee who is in both tables. The output shows the employee's current job. If the employee is listed more than once, that means the employee has held at least two jobs previously.

Note: The NULLIF function is logically equivalent to the following CASE expression. The CASE expression is discussed in 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, it returns that expression; otherwise, it does a COALESCE of the remaining expressions.

ORACLE

The COALESCE Function

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

Syntax

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

In the syntax:

<i>expr1</i>	returns this expression if it is not null
<i>expr2</i>	returns this expression if the first expression is null and this expression is not null
<i>exprn</i>	returns this expression if the preceding expressions are null

Using the COALESCE Function

```
SELECT    last_name,  
          COALESCE(commission_pct, salary, 10) comm  
FROM      employees  
ORDER BY  commission_pct;
```

LAST_NAME	COMM
Grant	.15
Zlotkey	.2
Taylor	.2
Abel	.3
King	24000
Kochhar	17000
De Haan	17000
Hunold	9000

20 rows selected.

ORACLE

The COALESCE Function

In the example shown, if the COMMISSION_PCT value is not null, it is shown. If the COMMISSION_PCT value is null, then the SALARY is shown. If the COMMISSION_PCT and SALARY values are null, then the value 10 is shown.

Conditional Expressions

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

ORACLE

Conditional Expressions

Two methods used to implement conditional processing (IF-THEN-ELSE logic) within a SQL statement are the CASE expression and the DECODE function.

Note: The CASE expression is new in the Oracle9i Server release. The CASE expression complies with ANSI SQL; DECODE is specific to Oracle syntax.

The 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
```

ORACLE

The CASE Expression

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

In a simple CASE expression, Oracle 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 an ELSE clause exists, then Oracle returns *else_expr*. Otherwise, Oracle 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.

Instructor Note

There is also a searched CASE expression. Oracle searches from left to right until it finds an occurrence of a condition that is true, and then returns *return_expr*. If no condition is found to be true, and an ELSE clause exists, Oracle returns *else_expr*. Otherwise Oracle returns null. For more information, see *Oracle9i SQL Reference*, “Expressions.”

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
...			
Lorentz	IT_PROG	4200	4620
Mourgos	ST_MAN	5800	5800
Rajs	ST_CLERK	3500	4025
...			
Gietz	AC_ACCOUNT	8300	8300

20 rows selected.

ORACLE

Using the CASE Expression

In the preceding SQL statement, 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 DECODE Function

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

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

ORACLE

The DECODE Function

The DECODE function decodes an expression in a way similar to the IF-THEN-ELSE logic 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
...			
Lorentz	IT_PROG	4200	4620
Mourgos	ST_MAN	5800	5800
Rajs	ST_CLERK	3500	4025
...			
Gietz	AC_ACCOUNT	8300	8300

20 rows selected.

ORACLE

Using the DECODE Function

In the preceding SQL statement, 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

3-57

Copyright © Oracle Corporation, 2001. All rights reserved.

Example

This slide shows another example using the DECODE function. In this example, we determine the tax rate for each employee in department 80 based on the monthly salary. The tax rates are as per the values mentioned in the following data.

<i>Monthly Salary Range</i>	<i>Rate</i>
\$0.00 - 1999.99	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
Zlotkey	10500	.42
Abel	11000	.42
Taylor	8600	.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**

ORACLE

Single-Row Functions

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
- 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, Part Two: Overview

This practice covers the following topics:

- **Creating queries that require the use of numeric, character, and date functions**
- **Using concatenation with functions**
- **Writing case-insensitive 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, Part Two: Overview

This practice is designed to give you a variety of exercises using different functions available for character, number, and date data types.

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

Instructor Note

This practice should be done in two parts. Part 1 contains questions 1-5 which cover the material from pages 1-23. Part 2 contains questions 6-14 and matches the material for the remainder of the lesson.

Practice question 6: Be sure to tell the students that their results may differ from the one provided, because `SYSDATE` is used in the exercise.

Instructor hint for practice question 10: The `ORDER BY` clause in the solution sorts on `TO_CHAR(hiredate-1, 'd')`. The format element 'd' returns a '1' for Sunday, '2' for Monday, and so forth. The expression `hiredate-1` effectively "shifts" each hiredate to the previous day, so that an employee hired on a Monday appears to have been hired on Sunday. The `TO_CHAR` function returns a '1' for that employee and the result set is sorted beginning with those employees hired on Monday.

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

Date
28-SEP-01

2. For each employee, display the employee number, last_name, salary, and salary increased by 15% and expressed as a whole number. Label the column New Salary. Place your SQL statement in a text file named lab3_2.sql.
3. Run your query in the file lab3_2.sql.

EMPLOYEE_ID	LAST_NAME	SALARY	New Salary
100	King	24000	27600
101	Kochhar	17000	19550
102	De Haan	17000	19550
103	Hunold	9000	10350
...			
202	Fay	6000	6900
205	Higgins	12000	13800
206	Gietz	8300	9545

20 rows selected.

the new salary. Label the column Increase. Save the contents of the file as lab3_4.sql. Run the revised query.

EMPLOYEE_ID	LAST_NAME	SALARY	New Salary	Increase
100	King	24000	27600	3600
101	Kochhar	17000	19550	2550
102	De Haan	17000	19550	2550
103	Hunold	9000	10350	1350
104	Ernst	6000	6900	900
107	Lorentz	4200	4830	630
124	Mourgos	5800	6670	870
141	Rajs	3500	4025	525
142	Davies	3100	3565	465
143	Matos	2600	2990	390
...				
201	Hartstein	13000	14950	1950
202	Fay	6000	6900	900
205	Higgins	12000	13800	1800
206	Gietz	8300	9545	1245

20 rows selected

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

Name	Length
Abel	4
Matos	5
Mourgos	7

6. For each employee, display the employee's last name, and calculate the number of months between today and the date 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
Zlotkey	20
Mourgos	22
Grant	28
Lorentz	32
Vargas	39
Taylor	42
Matos	42
Fay	49
Davies	56
Abel	65
Hartstein	67
Rajs	71
Higgins	88
Gietz	88
LAST_NAME	MONTHS_WORKED
De Haan	105
Ernst	124
Hunold	141
Kochhar	144
Whalen	168
King	171

20 rows selected.

7. Write a query 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
King earns \$24,000.00 monthly but wants \$72,000.00.
Kochhar earns \$17,000.00 monthly but wants \$51,000.00.
De Haan earns \$17,000.00 monthly but wants \$51,000.00.
Hunold earns \$9,000.00 monthly but wants \$27,000.00.
Ernst earns \$6,000.00 monthly but wants \$18,000.00.
Lorentz earns \$4,200.00 monthly but wants \$12,600.00.
Mourgos earns \$5,800.00 monthly but wants \$17,400.00.
Rajs earns \$3,500.00 monthly but wants \$10,500.00.
Davies earns \$3,100.00 monthly but wants \$9,300.00.
Matos earns \$2,600.00 monthly but wants \$7,800.00.
Vargas earns \$2,500.00 monthly but wants \$7,500.00.
■ ■ ■
Gietz earns \$8,300.00 monthly but wants \$24,900.00.

20 rows selected.

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 \$. Label the column SALARY.

LAST_NAME	SALARY
King	\$\$\$\$\$\$\$\$\$24000
Kochhar	\$\$\$\$\$\$\$\$\$17000
De Haan	\$\$\$\$\$\$\$\$\$17000
Hunold	\$\$\$\$\$\$\$\$\$9000
Ernst	\$\$\$\$\$\$\$\$\$6000
Lorentz	\$\$\$\$\$\$\$\$\$4200
Mourgos	\$\$\$\$\$\$\$\$\$5800
Rajs	\$\$\$\$\$\$\$\$\$3500
■ ■ ■	
Higgins	\$\$\$\$\$\$\$\$\$12000
Gietz	\$\$\$\$\$\$\$\$\$8300

20 rows selected.

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
King	17-JUN-87	Monday, the Twenty-First of December, 1987
Kochhar	21-SEP-89	Monday, the Twenty-Sixth of March, 1990
De Haan	13-JAN-93	Monday, the Nineteenth of July, 1993
Hunold	03-JAN-90	Monday, the Ninth of July, 1990
Ernst	21-MAY-91	Monday, the Twenty-Fifth of November, 1991
Lorentz	07-FEB-99	Monday, the Ninth of August, 1999
Mourgos	16-NOV-99	Monday, the Twenty-Second of May, 2000
Rajs	17-OCT-95	Monday, the Twenty-Second of April, 1996
Davies	29-JAN-97	Monday, the Fourth of August, 1997
■ ■ ■		
Gietz	07-JUN-94	Monday, the Twelfth of December, 1994

20 rows selected.

10. 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
Grant	24-MAY-99	MONDAY
Ernst	21-MAY-91	TUESDAY
Mourgos	16-NOV-99	TUESDAY
Taylor	24-MAR-98	TUESDAY
Rajs	17-OCT-95	TUESDAY
Gietz	07-JUN-94	TUESDAY
Higgins	07-JUN-94	TUESDAY
King	17-JUN-87	WEDNESDAY
De Haan	13-JAN-93	WEDNESDAY
■ ■ ■		
Abel	11-MAY-96	SATURDAY
Lorentz	07-FEB-99	SUNDAY
Fay	17-AUG-97	SUNDAY
Matos	15-MAR-98	SUNDAY

20 rows selected.

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 commission, put "No Commission." Label the column COMM.

LAST_NAME	COMM
King	No Commission
Kochhar	No Commission
De Haan	No Commission
Hunold	No Commission
Ernst	No Commission
Lorentz	No Commission
Mourgos	No Commission
Rajs	No Commission
Davies	No Commission
Matos	No Commission
Vargas	No Commission
Zlotkey	.2
Abel	.3
Taylor	.2
■ ■ ■	
Gietz	No Commission

20 rows selected.

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

EMPLOYEE_AND_THEIR_SALARIES	
King	*****
Kochhar	*****
De Haan	*****
Hartstei	*****
Higgins	*****
Abel	*****
■ ■ ■	
Vargas	**

20 rows selected.

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

<i>Job</i>	<i>Grade</i>
AD_PRES	A
ST_MAN	B
IT_PROG	C
SA_REP	D
ST_CLERK	E
None of the above	0

JOB_ID	G
AD_PRES	A
AD_VP	0
AD_VP	0
IT_PROG	C
IT_PROG	C
IT_PROG	C
ST_MAN	B
ST_CLERK	E
ST_CLERK	E
ST_CLERK	E
■ ■ ■	
AC_MGR	0
AC_ACCOUNT	0

20 rows selected.

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