Single Row Functions in SQL
Introduction to Lab-3

Functions make the basic query block more powerful, and they are used to manipulate data values. This is the first of two labs that explore functions. It focuses on single-row character, number, and date functions.

The main topics of this lab include:

1. SQL Functions
2. Two Types of SQL Functions
3. Single-Row Functions
4. Character Functions
5. Case-Manipulation Functions
6. Using Case-Manipulation Functions
7. Character-Manipulation Functions
8. Using the Character-Manipulation Functions
9. Number Functions
10. Using the ROUND Function
11. Using the TRUNC Function
12. Using the MOD Function
13. Working with Dates
14. Arithmetic with Dates
15. Date Functions
16. Using Date Functions
17. NVL Function
18. Practice SQL Statements
19. SOLUTIONS: Practice SQL Statements

Objectives of this Lab

At the end of this lab, students should be able to:

- Describe various types of functions that are available in SQL
- Use character, number, and date functions in SELECT statements
- Understand the use of NVL function in case of NULL values
1. 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 lab are specific to the Oracle version of SQL.
2. Two Types of SQL Functions

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 a later lab).
3. Single-Row Functions

 funktion_name [(arg1, arg2,...)]

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.

This lab 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**: Convert a value from one data type to another
  - `NVL`
  - `NVL2`
  - `NULLIF`
  - `COALESCE`
  - `CASE`
  - `DECODE`

### 4. 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

#### 5. 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
6. Using Case-Manipulation Functions

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

```sql
SELECT empno, ename, deptno
FROM emp
WHERE ename = 'clark';
no rows selected
```

As the employee names are stored as capital strings, no rows are selected using the above query. However, the correct version of the above query may be written as:

```sql
SELECT empno, ename, deptno
FROM emp
WHERE LOWER(ename) = 'clark';
```

7. Character-Manipulation Functions

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

- **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.)

<table>
<thead>
<tr>
<th>Function</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONCAT('Hello', 'World')</td>
<td>HelloWorld</td>
</tr>
<tr>
<td>SUBSTR('HelloWorld',1,5)</td>
<td>Hello</td>
</tr>
<tr>
<td>LENGTH('HelloWorld')</td>
<td>10</td>
</tr>
<tr>
<td>INSTR('HelloWorld', 'W')</td>
<td>6</td>
</tr>
<tr>
<td>LPAD(salary,10,'*')</td>
<td>*****24000</td>
</tr>
<tr>
<td>RPAD(salary, 10, '*')</td>
<td>24000*****</td>
</tr>
</tbody>
</table>
8. Using the Character-Manipulation Functions

This example displays employee names and employee numbers joined together, the length of the employee name, and the numeric position of the letter \( a \) in the employee name for all employees who have the string `MAN` contained in the job starting at the sixth position of the job.

```sql
SELECT empno, CONCAT(empno, ename) NAME,
       job, LENGTH(ename),
       INSTR(ename, 'a') "Contains 'a'?"
FROM emp
WHERE SUBSTR(job, 6) = 'MAN';
```

Modify the SQL statement in the example to display the data for those employees whose names end with the letter \( n \).

```sql
SELECT empno, CONCAT(empno, ename) NAME,
       LENGTH(ename), INSTR(ename, 'a') "Contains 'a'?"
FROM emp
WHERE SUBSTR(ename, -1, 1) = 'n';
```

9. Number Functions

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

<table>
<thead>
<tr>
<th>Function</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>ROUND(`column</td>
<td>expression, ( n )`</td>
</tr>
<tr>
<td>TRUNC(`column</td>
<td>expression, ( n )`</td>
</tr>
<tr>
<td>MOD(<code>m, n</code>)</td>
<td>Returns the remainder of ( m ) divided by ( n )</td>
</tr>
</tbody>
</table>
10. Using the 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 lab.

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

11. Using the 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.

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

12. Using the MOD Function

The **MOD** function finds the remainder of the first argument divided by the second argument.

This example calculates the remainder of the salary after dividing it by 500 for all employees whose job is **SALESMAN**.

```sql
SELECT ename, sal, MOD(sal, 500)
FROM emp
WHERE job = 'SALESMAN';
```

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

13. Working with Dates

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 this example, 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.

```sql
SELECT ename, hiredate 
FROM emp 
WHERE hiredate < '01-FEB-88';
```

### 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.

```sql
SELECT SYSDATE
FROM DUAL;
```

### 14. 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:

<table>
<thead>
<tr>
<th>Operation</th>
<th>Result</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>date + number</td>
<td>Date</td>
<td>Adds a number of days to a date</td>
</tr>
<tr>
<td>date – number</td>
<td>Date</td>
<td>Subtracts a number of days from a date</td>
</tr>
<tr>
<td>date – date</td>
<td>Number of days</td>
<td>Subtracts one date from another</td>
</tr>
<tr>
<td>date + number/24</td>
<td>Date</td>
<td>Adds a number of hours to a date</td>
</tr>
</tbody>
</table>

**Example:**

```sql
SELECT ename, (SYSDATE-hiredate)/7 AS WEEKS 
FROM emp 
WHERE deptno = 20;
```
15. 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.

16. Using Date Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>MONTHS_BETWEEN</td>
<td>19.6774194</td>
</tr>
<tr>
<td>('01-SEP-95','11-JAN-94')</td>
<td></td>
</tr>
<tr>
<td>ADD_MONTHS ('11-JAN-94',6)</td>
<td>'11-JUL-94'</td>
</tr>
<tr>
<td>NEXT_DAY ('01-SEP-95','FRIDAY')</td>
<td>'08-SEP-95'</td>
</tr>
<tr>
<td>LAST_DAY ('01-FEB-95')</td>
<td>'28-FEB-95'</td>
</tr>
</tbody>
</table>
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 70 months.

```
SELECT empno, hiredate, 
MONTHS_BETWEEN (SYSDATE, hiredate) TENURE, 
ADD_MONTHS (hiredate, 6) REVIEW, 
NEXT_DAY (hiredate, 'FRIDAY'), LAST_DAY(hiredate) 
FROM emp 
WHERE MONTHS_BETWEEN (SYSDATE, hiredate) < 70;
```

Assume SYSDATE = '25-JUL-03':

<table>
<thead>
<tr>
<th>Function</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>ROUND(SYSDATE,'MONTH')</td>
<td>01-AUG-03</td>
</tr>
<tr>
<td>ROUND(SYSDATE,'YEAR')</td>
<td>01-JAN-04</td>
</tr>
<tr>
<td>TRUNC(SYSDATE,'MONTH')</td>
<td>01-JUL-03</td>
</tr>
<tr>
<td>TRUNC(SYSDATE,'YEAR')</td>
<td>01-JAN-03</td>
</tr>
</tbody>
</table>

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 1993. Display the employee number, hire date, and start month using the `ROUND` and `TRUNC` functions.

```
SELECT empno, hiredate, 
ROUND(hiredate, 'MONTH'), TRUNC(hiredate, 'MONTH') 
FROM emp 
WHERE hiredate LIKE '%93';
```

### 17. 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**.

Data types that can be used are date, character, and number. Data types must match:

- `NVL(comm,0)`
- `NVL(hiredate,'01-JAN-97')`
- `NVL(job,'No Job Yet')`

**Example:**

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

```sql
SELECT ename, sal, NVL(comm, 0),
       (sal*12) + (sal*12*NVL(comm, 0)) AN_SAL
FROM emp;
```

### 18. Practice SELECT Statements

Write SELECT statements for the following:

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

2. The HR department needs a report to display the employee number, 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 file **lab_03_02.sql**

4. Modify your query **lab_03_02.sql** 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.

5. Write a query that displays the employee 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 employees’ names.

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

7. 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**.

8. Create a query that displays the first eight characters of the employees’ 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**.

9. The HR department wants to find the length of employment for each employee. For each employee, display the 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.

10. Create a query that displays the employees’ last names and commission amounts. If an employee does not earn commission, show “No Commission.” Label the column COMM.