SQL Functions

Overview

In this lesson, you will learn how to use functions that apply to values in individual rows. Specifically, you will learn how to use functions with character data, numeric data, and dates. You will also learn how to concatenate two columns into a single expression.

Often, it is advantageous to perform "If…Then…Else" operations on data in a database. The SQL standard has provided the CASE statement to perform this type of operation. At times a database might be designed to hold coded data values. When this data is retrieved, it is nice to provide the decoded meaning of this data with the final information.

Objectives

- Use functions in queries.
- Use the UPPER and LOWER functions with character data.
- Use the ROUND and FLOOR functions with numeric data.
- Add a specific number of months or days to a date.
- Calculate the number of days between two dates.
- Use concatenation in a query.
- Use the CASE expression.

1. Single-Row Functions

A single-row function operates on one row at a time and returns one row of output for each row. There are four main types of single-row functions:

- Date/Time functions process dates and times.
- Character functions manipulate strings of characters.
- Numeric functions perform calculations.
- Covert data functions

1.1. Date and Time Functions

In MySQL, there are several useful date and time functions. However, it is important to first briefly look at the main date and time types are available to MySQL. These are shown in the table below:

DATETIME	YYYY-MM-DD HH:MM:SS
DATE	YYYY-MM-DD
TIMESTAMP	YYYYMMDDHHSSMM
TIME	HH:MM:SS
YEAR	YYY

Table 1 MySQL Date and Time Data Types

As you can see from Table 1, the DATE type is stored in a special internal format that includes only year, month and day while the DATETIME data type also stores the hours, minutes, and seconds. If you try to enter a date in a format other than the Year-Month-Day format, it might work but it will not be storing them as you expect.

A description of the functions you will explore in this lesson can be found in Table 2.

Function	Description
NOW() **SYSDATE()** **CURRENT_TIMESTAMP()**	Returns the current local date and time based on the system's clock
CURDATE() **CURRENT_DATE()**	Returns the current local date.
CURTIME() **CURRENT_TIME()**	Returns the current local time.

Table 2 Selected Date and Time Functions

Example 1: Enter the following query and examine how the date is displayed.

```
mysql> SELECT DISTINCT(bdate)
       FROM employee
       WHERE dno = 7;
```

```
+---------------------+
| bdate               |
+---------------------+
| 1958-01-16 00:00:00 |
| 1954-05-22 00:00:00 |
| 1944-06-21 00:00:00 |
| 1966-12-16 00:00:00 |
| 1967-11-11 00:00:00 |
| 1960-03-21 00:00:00 |
| 1950-10-09 00:00:00 |
| 1956-06-19 00:00:00 |
| 1966-06-18 00:00:00 |
| 1977-07-31 00:00:00 |
+---------------------+
10 rows in set (0.00 sec)
```

It is possible to change the format of the date using the DATE_FORMAT() function.

Syntax of this function:
DATE_FORMAT(date, format)
TIME_FORMAT(time, format)

1.1.1. CURRENT DATE and CURRENT TIME

The CURRENT_DATE function returns today's date while the CURRENT_TIME function returns the current time.

Example 2: Enter the following query to display today's date and time. Notice that in MySQL the functions are called using the SELECT statement but no FROM clause is needed.

```
mysql> SELECT CURRENT_DATE(), CURRENT_TIME();

+----------------+----------------+
| CURRENT_DATE() | CURRENT_TIME() |
+----------------+----------------+
| 2018-04-08     | 15:37:39       |
+----------------+----------------+
1 row in set (0.00 sec)
```

1.1.2. MONTH, DAYOFMONTH and YEAR

MySQL provides functions for extracting the month, day or year from any given date.
The syntax of each function is as follows:

- **DAYOFMONTH(date)** returns the day of the month for date, in the range 0 to 31.
- **MONTH(date)** returns the month for date, in the range 0 to 12.
- **YEAR(date)** returns the year for date, in the range 1000 to 9999, or 0 for the "zero" date.

DATEDIFF
The DATEDIFF function subtracts two dates and returns a value in days from one date to the other.

Example 3: The following example calculates the number of days between the1st January 2008 and the 25th December 2008.

```
mysql> SELECT DATEDIFF('2008-12-25','2008-01-01');
+-------------------------------------+
| DATEDIFF('2008-12-25','2008-01-01') |
+-------------------------------------+
|                                 359 |
+-------------------------------------+
1 row in set (0.00 sec)
```

1.1.3. DATE_ADD and DATE_SUB

The DATE_ADD and DATE_SUB functions both perform date arithmetic and allow you to either add or subtract two dates from one another.

Syntax of these functions:
DATE_ADD(date, INTERVAL expr unit)
DATE_SUB(date, INTERVAL expr unit)

Where expr is an expression specifying the interval value to be added or subtracted from the starting date and unit is a keyword indicating the units in which the expression should be interpreted.

Example 4: The following query adds 11 months to the date 1st January 2008 to display a new date of 1st December 2008.

```
mysql> SELECT ADDDATE('2008-01-01', INTERVAL 11 MONTH );
+-----------------------------------------+
| ADDDATE('2008-01-01', INTERVAL 11 MONTH ) |
+-----------------------------------------+
| 2008-12-01                              |
+-----------------------------------------+
1 row in set (0.01 sec)
```

1.1.4. LAST_DAY

The function returns the date of the last day of the month given in a date.

Syntax:
`LAST_DAY(date_value)`

Examples that use the date/time parsing functions

Function	Result
DAYOFMONTH('2011-09-03')	3
MONTH('2011-09-03')	9
YEAR('2011-09-03')	2011
HOUR('11:35:00')	11
MINUTE('11:35:00')	35
SECOND('11:35:00')	0
DAYOFWEEK('2011-09-03')	7
QUARTER('2011-09-03')	3
DAYOFYEAR('2011-09-03')	246
WEEK('2011-09-03')	35
LAST_DAY('2011-09-03')	30
DAYNAME('2011-09-03')	Saturday
MONTHNAME('2011-09-03')	September

Examples that use the date/time formatting functions

Function	Result
DATE_FORMAT('2011-09-03', '%m/%d/%y')	09/03/11
DATE_FORMAT('2011-09-03', '%W, %M %D, %Y')	Saturday, September 3rd, 2011
DATE_FORMAT('2011-09-03', '%e-%b-%y')	3-Sep-11
DATE_FORMAT('2011-09-03 16:45', '%r')	04:45:00 PM
TIME_FORMAT('16:45', '%r')	04:45:00 PM
TIME_FORMAT('16:45', '%l:%i %p')	4:45 PM
DATEDIFF('2011-09-30', '2011-09-03')	27
DATEDIFF('2011-09-30 23:59:59','2011-09-03')	27
DATEDIFF('2011-09-03', '2011-09-30')	-27
TO_DAYS('2011-09-30') - TO_DAYS('2011-09-03')	27
TIME_TO_SEC('10:00') - TIME_TO_SEC('09:59')	60

Example 5: Search for month, day, and year integers.

```
mysql> SELECT essn, dependent_name, bdate
       FROM dependent
       WHERE MONTH(bdate) = 4
       AND DAYOFMONTH(bdate) = 4
       AND YEAR(bdate) = 1997;
+-----------+----------------+------------+
| essn      | dependent_name | bdate      |
+-----------+----------------+------------+
| 444444400 | Johnny         | 1997-04-04 |
+-----------+----------------+------------+
1 row in set (0.00 sec)
```

Example 6: Search for a formatted date.

```
mysql> SELECT fname, lname, DATE_FORMAT(bdate, '%m-%d-%Y')
       FROM employee
       WHERE DATE_FORMAT(bdate, '%m-%d-%Y') = '07-31-1977';
+-------+---------+-------------------------------+
| fname | lname   | DATE_FORMAT(bdate, '%m-%d-%Y') |
+-------+---------+-------------------------------+
| Sam   | Snedden | 07-31-1977                    |
+-------+---------+-------------------------------+
1 row in set (0.01 sec)
```

1.2. Numeric Functions

Numeric functions take one numeric parameter and return one value. A description of the functions you will explore in this lesson can be found in Table 3.

Function	Description
ABS(number)	Returns the absolute value of a number Syntax: `ABS(number)`
ROUND(number[,length])	Returns the number rounded to the precision specified by length. Syntax: `ROUND(number, p) WHERE p = precision`
TRUNCATE(number, length)	Truncates a value to a specified precision (number of digits) Syntax: `TRUNCATE(number, p)` WHERE p = precision
POWER(number, power)	Returns the number raised to the specified power. Syntax: `POWER(number, power)`
RAND([integer])	Return a random floating-point value between 0 and 1. If integer is omitted, the function returns the same number each time it's invoked within the same query. Otherwise, integer provides a seed value for the random number generator.

Table 3 Selected Numeric Functions

Examples that use the numeric functions

Function	Result
ROUND(12.49,0)	12
ROUND(12.50,0)	13
ROUND(12.49,1)	12.5
TRUNCATE(12.51,0)	12
TRUNCATE(12.49,1)	12.4
ABS(-1.25)	1.25
ABS(1.25)	1.25
RAND()	0.5599424546211422
RAND (4)	0.15595286540310166

Example 7: ABS returns the absolute value of the argument.

```
mysql> SELECT ABS(12),  ABS(-12), ABS(0),  ABS(45.34);
+---------+----------+--------+------------+
| ABS(12) | ABS(-12) | ABS(0) | ABS(45.34) |
+---------+----------+--------+------------+
|      12 |       12 |      0 |      45.34 |
+---------+----------+--------+------------+
1 row in set (0.01 sec)
```

Example 8: POWER(a, b), is used to calculate a raised to the b power ; POW is an alias for Power.

```
mysql> SELECT POWER(3, 2), POWER(10, 3), POWER(4.5, 3.2), POWER(10, -3);
+-------------+--------------+--------------------+---------------+
| POWER(3, 2) | POWER(10, 3) | POWER(4.5, 3.2)    | POWER(10, -3) |
+-------------+--------------+--------------------+---------------+
|           9 |         1000 | 123.10623351019093 |         0.001 |
+-------------+--------------+--------------------+---------------+
1 row in set (0.01 sec)
```

1.3. String Functions

String manipulation functions are amongst the most-used functions in programming. Table 4 shows a subset of the most useful string manipulation functions in MySQL. Some of the string functions

Function	Description
CONCAT	Concatenates data from two different character columns and returns a single column. Syntax: `CONCAT(str, strg)`
UPPER/LOWER	Returns a string in all capital or all lowercase letters Syntax: `UPPER(str) , LOWER(str)`
SUBSTR	Returns a substring or part of a given string parameter Syntax: `SUBSTR(strg, p, l)` where p = start position and l = length of characters
LENGTH	Returns the number of characters in a string value Syntax: `LENGTH(str)`

LTRIM(str)	Returns the string with any leading spaces removed. Syntax: `LTRIM(str)`
RTRIM(str)	Returns the string with any trailing spaces removed. Syntax: `RTRIM(str)`
REVERSE(str)	Returns the string with the characters in reverse order. Syntax: `REVERSE(str)`
REPEAT(str,count)	Returns the specified string repeated count times. Syntax: `REPEAT(str,count)`
TRIM	Returns the string without leading or trailing occurrences of the specified remove string. If remove string is omitted, spaces are removed. Syntax: `TRIM([[BOTH|LEADING|TRAILING] [remove] FROM]`
LEFT(str, length)	Returns the specified number of characters from the beginning of the string. Syntax: `LEFT(str, length)`
RIGHT(str, length)	Returns the specified number of characters from the end of the string. Syntax: `RIGHT(str, length)`
LPAD(str,length,pad)	Returns the string padded on the left with the specified pad string until its specified length.
RPAD(str,length,pad)	Returns the string padded on the right with the specified pad string until its specified length.

Table 4 Selected String Functions

Examples of String function

Function	Result
`CONCAT('Last', 'First')`	`'LastFirst'`
`LTRIM('  MySQL  ')`	`'MySQL  '`
`RTRIM('  MySQL  ')`	`'  MySQL'`
`TRIM('  MySQL  ')`	`'MySQL'`
`LOWER('MySQL')`	`'mysql'`
`UPPER('ca')`	`'CA'`
`LEFT('MySQL', 3)`	`'MyS'`
`RIGHT('MySQL', 3)`	`'SQL'`
`SUBSTRING('(559) 555-1212', 7, 8)`	`'555-1212'`
`LENGTH('MySQL')`	`5`
`LENGTH('  MySQL  ')`	`9`

1.3.1. CONCAT

Concatenates data from two different character columns and returns a single column.

Example 9: The following query illustrates the CONCAT function. It lists all employees first and last names concatenated together.

```
mysql> SELECT CONCAT (fname, ' ' , lname) AS NAME
       FROM employee
       WHERE dno = 4;
+------------------+
| NAME             |
+------------------+
| Jennifer Wallace |
| Ahmad Jabbar     |
| Alicia Zelaya    |
+------------------+
3 rows in set (0.00 sec)
```

1.3.2. SUBSTR

Example 10: The following example lists the first three characters of all the employees' first name.

```
mysql> SELECT ssn, SUBSTR(fname,1,3)
       FROM employee
       WHERE dno = 6;
+-----------+-------------------+
| ssn       | SUBSTR(fname,1,3) |
+-----------+-------------------+
| 111111100 | Jar               |
| 111111101 | Jon               |
| 111111102 | Jus               |
| 111111103 | Bra               |
| 333333300 | Kim               |
| 333333301 | Jef               |
| 555555500 | Joh               |
| 555555501 | Nan               |
+-----------+-------------------+
8 rows in set (0.01 sec)
```

1.3.3. UPPER/LOWER

Example 11: The following query lists all employee last names in all capital letters and all first names in all lowercase letters.

```
mysql> SELECT CONCAT(UPPER(lname), LOWER(fname)) AS NAME
       FROM employee;
+-----------------+
| JAMESjared      |
| JONESjon        |
| MARKjustin      |
| KNIGHTbrad      |
| SMITHjohn       |
| WALLISevan      |
| ZELLjosh        |
| VILEandy        |
| BRANDtom        |
| VOSjenny        |
| CARTERchris     |
| GRACEkim        |
| CHASEjeff       |
| WONGfranklin    |
| FREEDalex       |
| BAYSbonnie      |
| BESTalec        |
| SNEDDENsam      |
| ENGLISHjoyce    |
| JAMESjohn       |
| BALLnandita     |
| BENDERbob       |
| JARVISjill      |
| KINGkate        |
| LESLIElyle      |
| KINGbillie      |
| KRAMERjon       |
| KINGray         |
| SMALLgerald     |
| HEADarnold      |
| PATAKIhelga     |
| DREWnaveen      |
| REEDYcarl       |
| HALLsammy       |
| BACHERred       |
| NARAYANramesh   |
| BORGjames       |
| WALLACEjennifer |
| JABBARahmad     |
| ZELAYAalicia    |
+-----------------+
40 rows in set (0.01 sec)
```

1.3.4. LENGTH

Example 12: The following example lists all dependents' names and the length of their names, ordered by the descended dependent_name length.

```
mysql> SELECT dependent_name, LENGTH(dependent_name) AS NAMESIZE
       FROM dependent
       ORDER BY NAMESIZE DESC;
+----------------+----------+
| dependent_name | NAMESIZE |
+----------------+----------+
| Elizabeth      |        9 |
| Theodore       |        8 |
| Michael        |        7 |
| Johnny         |        6 |
| Alice          |        5 |
| Alice          |        5 |
| Tommy          |        5 |
| Chris          |        5 |
| Abner          |        5 |
| Joy            |        3 |
| Sam            |        3 |
+----------------+----------+
11 rows in set (0.00 sec)
```

1.4. Cast and Convert functions

Whenever necessary, you can use Cast or Convert function to perform explicit conversion. This allow you to convert, or cast, an expression from one data type to another.

A CAST is an ANSI-standard function and is used more frequently than CONVERT.

Syntax of CAST function:

```
CAST(expression AS cast_type)
```

Syntax of CONVERT function:

```
CONVERT(expression , cast_type)
```

Example 13: `CAST()`

```
mysql> SELECT cast(123.567 as char);
+------------------------+
| cast(123.567 as char) |
+------------------------+
| 123.567               |
+------------------------+
1 row in set (0.01 sec)

mysql> SELECT cast(123.567 as char(2));
+--------------------------+
| cast(123.567 as char(2)) |
+--------------------------+
| 12                       |
+--------------------------+
1 row in set, 1 warning (0.00 sec)
```

Example 14: `CAST()`

```
mysql> SELECT cast('absdsdsr' as char(2));
+-----------------------------+
| cast('absdsdsr' as char(2)) |
+-----------------------------+
| ab                          |
+-----------------------------+
1 row in set, 1 warning (0.00 sec)
```

Example 15: `CONVERT()`

```
mysql> SELECT convert(123.567 , char);
+-------------------------+
| convert(123.567 , char) |
+-------------------------+
| 123.567                 |
+-------------------------+
1 row in set (0.00 sec)

mysql> SELECT convert(123.567 , char(2));
+----------------------------+
| convert(123.567 , char(2)) |
+----------------------------+
| 12                         |
+----------------------------+
1 row in set, 1 warning (0.00 sec)
```

Example 16: For each employee, display the SSN followed by the null value.

```
mysql> SELECT ssn, CAST(NULL AS CHAR)
       FROM employee;
+-----------+--------------------+
| ssn       | CAST(NULL AS CHAR) |
+-----------+--------------------+
| 111111100 | NULL               |
| 222222200 | NULL               |
| 333333300 | NULL               |
| 444444400 | NULL               |
| 555555500 | NULL               |
| 666666600 | NULL               |
| 888665555 | NULL               |
| 111111101 | NULL               |
| 111111102 | NULL               |
| 111111103 | NULL               |
| 222222201 | NULL               |
| 222222202 | NULL               |
| 222222203 | NULL               |
| 222222204 | NULL               |
| 222222205 | NULL               |
| 333333301 | NULL               |
| 123456789 | NULL               |
| 453453453 | NULL               |
| 666884444 | NULL               |
| 444444401 | NULL               |
| 444444402 | NULL               |
| 444444403 | NULL               |
| 555555501 | NULL               |
| 666666601 | NULL               |
| 666666602 | NULL               |
| 666666603 | NULL               |
| 666666607 | NULL               |
| 666666608 | NULL               |
| 666666609 | NULL               |
| 666666604 | NULL               |
| 666666605 | NULL               |
| 666666606 | NULL               |
| 666666610 | NULL               |
| 666666611 | NULL               |
| 666666612 | NULL               |
| 666666613 | NULL               |
| 333445555 | NULL               |
| 987654321 | NULL               |
| 987987987 | NULL               |
| 999887777 | NULL               |
+-----------+--------------------+
40 rows in set (0.00 sec)
```

Example 17: Get the odd SSNs from the EMPLOYEE table.

```
mysql> SELECT ssn
       FROM employee
       WHERE ssn & 1;
+-----------+
| ssn       |
+-----------+
| 888665555 |
| 111111101 |
| 111111103 |
| 222222201 |
| 222222203 |
| 222222205 |
| 333333301 |
| 123456789 |
| 453453453 |
| 444444401 |
| 444444403 |
| 555555501 |
| 666666601 |
| 666666603 |
| 666666607 |
| 666666609 |
| 666666605 |
| 666666611 |
| 666666613 |
| 333445555 |
| 987654321 |
| 987987987 |
| 999887777 |
+-----------+
23 rows in set (0.00 sec)
```

1.5. Format

FORMAT takes a number and formats it. The result is a string. The second argument is the number of digits to display after the decimal.

Example 18:

```
mysql> SELECT Format(1234.5678, 3)

+----------------------+
| Format(1234.5678, 3) |
+----------------------+
| 1,234.568            |
+----------------------+
1 row in set (0.01 sec)
```

Example 19:

```
mysql> SELECT Format(1234.5678, 0);

+----------------------+
| Format(1234.5678, 0) |
+----------------------+
| 1,235                |
+----------------------+
1 row in set (0.00 sec)
```

Example 20:

```
mysql> SELECT Format(1234.5678, 8);
+----------------------+
| Format(1234.5678, 8) |
+----------------------+
| 1,234.56780000       |
+----------------------+
1 row in set (0.00 sec)
```

1.6. The User Variable and the SET Statement

Often, you will want to save values that are returned from queries. You may want to do this so that you can easily use a value in a later query. You may also simply want to save a result for later display. In both cases, user variables solve the problem. They allow you to store a result and use it later.

Syntax of SET statement:

<user variable> = @ <variable name>

Example 21: Create the user variable SSNNO and initialize it with the value 111111100.

```
mysql> SET @SSNNO = 444444400;
```

Get the last name and the first name of all employees with a SSN greater than the value of the SSNNO user variable that has just been created.

```
mysql> SELECT ssn, fname, lname
       FROM employee
       WHERE ssn > @SSNNO;
+-----------+----------+---------+
| ssn       | fname    | lname   |
+-----------+----------+---------+
| 444444401 | Bonnie   | Bays    |
| 444444402 | Alec     | Best    |
| 444444403 | Sam      | Snedden |
| 453453453 | Joyce    | English |
| 555555500 | John     | James   |
| 555555501 | Nandita  | Ball    |
| 666666600 | Bob      | Bender  |
| 666666601 | Jill     | Jarvis  |
| 666666602 | Kate     | King    |
| 666666603 | Lyle     | Leslie  |
| 666666604 | Billie   | King    |
| 666666605 | Jon      | Kramer  |
| 666666606 | Ray      | King    |
| 666666607 | Gerald   | Small   |
| 666666608 | Arnold   | Head    |
| 666666609 | Helga    | Pataki  |
| 666666610 | Naveen   | Drew    |
| 666666611 | Carl     | Reedy   |
| 666666612 | Sammy    | Hall    |
| 666666613 | Red      | Bacher  |
| 666884444 | Ramesh   | Narayan |
| 888665555 | James    | Borg    |
| 987654321 | Jennifer | Wallace |
| 987987987 | Ahmad    | Jabbar  |
| 999887777 | Alicia   | Zelaya  |
+-----------+----------+---------+
25 rows in set (0.01 sec)
```

```
mysql> SELECT @SSNNO;
+-----------+
| @SSNNO    |
+-----------+
| 444444400 |
+-----------+
1 row in set (0.00 sec)
```

Example 22:

```
mysql> SET @FNAME = (SELECT fname FROM employee WHERE SSN = 444444400);
mysql> SELECT @FNAME;
+--------+
| @FNAME |
+--------+
| Alex   |
+--------+
1 row in set (0.00 sec)
```

Here are some guidelines on using user variables:

- User variables are unique to a connection: variables that you create cannot be seen by anyone else, and two different connections can have two different variables with the same name.
- The variable names can be alphanumeric strings and can also include the period (.), underscore (_), and dollar ($) characters.
- Variable names are case insensitive in MySQL version 5 onward.
- Any variable that isn't initialized has the value NULL; you can also manually set a variable to be NULL.
- Variables are destroyed when a connection closes.
- You should avoid trying to both assign a value to a variable and use the variable as part of a SELECT query. Two reasons for this are that the new value may not be available for use immediately in the same statement, and a variable's type is set when it's first assigned in a query; trying to use it later as a different type in the same SQL statement can lead to unexpected results.

1.7. The Case Expression

A special function is the *case function*. This function serves as a kind of IF-THEN-ELSE statement. It can be compared with the SWITCH statement in Java.

The simple CASE function tests the expression in the CASE clauseagainst the expression in the WHEN clauss. Then, the function returns the result expression for the first test that's true.

Syntax of simple CASE function:

```
CASE input_expression
      WHEN when_expression_1 THEN result_expression_1
       [WHEN when_expression_2 THEN result_expression_2]...
      ELSE else_result_expression]
END
```

Syntax of searched CASE function:

```
CASE
      WHEN conditional_expression_1
      THEN result_expression_1
       [WHEN conditional_expression_2
      THEN result_expression_2]...
       [ELSE else_result_expression]
END
```

Example 23: Get the SSN, the gender, and the first name of each employee that was born before 1960. The gender must be printed as 'Female' or 'Male'.

```
mysql> SELECT SSN,
       CASE sex
       WHEN 'F' THEN 'female'
       ELSE 'Male' END AS gender, fname
       FROM  employee
       WHERE year(bdate)< 1960;
```

SSN	gender	fname
123456789	Male	John
222222200	Male	Evan
222222201	Male	Josh
222222202	Male	Andy
333445555	Male	Franklin
444444400	Male	Alex
444444401	female	Bonnie
666666606	Male	Ray
666884444	Male	Ramesh
888665555	Male	James
987654321	female	Jennifer
987987987	Male	Ahmad
999887777	female	Alicia

13 rows in set (0.01 sec)

This construct is equal to the following IF-THEN-ELSE construct:

```
IF sex= 'F' THEN
      RETURN 'Female'
ELSE
      RETURN 'Male'
ENDIF
```

Example 24: For each employee, display the SSN, their birthyear, and age group.

```
mysql> SELECT SSN, bdate,
       CASE
            WHEN year(bdate) < 1960 THEN 'Managers'
            WHEN year(bdate) < 1970 THEN 'Seniors'
            ELSE 'not Classified' END AS AGE_GROUP
            FROM employee
            ORDER BY bdate;
```

SSN	bdate	AGE_GROUP
888665555	1927-11-10 00:00:00	Managers
987654321	1931-06-20 00:00:00	Managers
222222202	1944-06-21 00:00:00	Managers
333445555	1945-12-08 00:00:00	Managers
666666606	1949-08-16 00:00:00	Managers
444444400	1950-10-09 00:00:00	Managers
666884444	1952-09-15 00:00:00	Managers
222222201	1954-05-22 00:00:00	Managers
123456789	1955-01-09 00:00:00	Managers
444444401	1956-06-19 00:00:00	Managers
222222200	1958-01-16 00:00:00	Managers
999887777	1958-07-19 00:00:00	Managers
987987987	1959-03-29 00:00:00	Managers
666666604	1960-01-01 00:00:00	Seniors
222222205	1960-03-21 00:00:00	Seniors
666666607	1962-02-15 00:00:00	Seniors
453453453	1962-07-31 00:00:00	Seniors
666666603	1963-06-09 00:00:00	Seniors
666666605	1964-08-22 00:00:00	Seniors
111111102	1966-01-12 00:00:00	Seniors
666666601	1966-01-14 00:00:00	Seniors
666666602	1966-04-16 00:00:00	Seniors
444444402	1966-06-18 00:00:00	Seniors
111111100	1966-10-10 00:00:00	Seniors
222222203	1966-12-16 00:00:00	Seniors
666666608	1967-05-19 00:00:00	Seniors
222222204	1967-11-11 00:00:00	Seniors
111111101	1967-11-14 00:00:00	Seniors
111111103	1968-02-13 00:00:00	Seniors
666666600	1968-04-17 00:00:00	Seniors
666666609	1969-03-11 00:00:00	Seniors
555555501	1969-04-16 00:00:00	Seniors
333333301	1970-01-07 00:00:00	not Classified
666666612	1970-01-11 00:00:00	not Classified
666666610	1970-05-23 00:00:00	not Classified
333333300	1970-10-23 00:00:00	not Classified
555555500	1975-06-30 00:00:00	not Classified
666666611	1977-06-21 00:00:00	not Classified
444444403	1977-07-31 00:00:00	not Classified
666666613	1980-05-21 00:00:00	not Classified