16.5.12 Putting Text in the Query Output

The previous example can be refined by marking the commissions as percentages with the percent sign (%). This enables you to put such items as symbols and comments in the output, as in the following example :

```
SELECT salesman_name, comm * 100, '%'
FROM salesman ;
```

```
+----------------+----------+---+
| SALESMAN_NAME  | COMM*100 | % |
+----------------+----------+---+
| Ajay           | 13.00    | % |
| Amit           | 11.00    | % |
| Shally         | 07.00    | % |
| Isha           | 15.00    | % |
+----------------+----------+---+
```

A sample output produced by above query is shown here.

See, the same comment or symbol gets printed with every row of the output, not simply once for the table.

You could insert text in your query also, making it more presentable. For example,

```
SELECT salesman_name, 'gets the commission', comm*100, '%'
FROM salesman ;
```

The sample output produced by above query is shown below :

```
+----------------+--------------------+----------+---+
| SALESMAN_NAME  | gets the commission | COMM*100 | % |
+----------------+--------------------+----------+---+
| Ajay           | gets the commission | 13.00    | % |
| Amit           | gets the commission | 11.00    | % |
| Shally         | gets the commission | 07.00    | % |
| Isha           | gets the commission | 15.00    | % |
+----------------+--------------------+----------+---+
```

EXAMPLE 16.4. *Create a query that produces display in following format :*

```
<studentname> obtained <aggregate> marks and has <aggregate/5> %.
```

Consider table ***Student*** *of example 16.3 for this.*

Solution.

```
mysql> SELECT name , "obtained", aggregate, "marks and has", aggregate/5, "%"
    -> FROM student ;
```

```
+----------+----------+-----------+---------------+-------------+----+
| name     | obtained | aggregate | marks and has | aggregate/5 | %  |
+----------+----------+-----------+---------------+-------------+----+
| Abu Bakar| obtained | 456       | marks and has | 91.2000     | %  |
| Aanya    | obtained | 340       | marks and has | 68.0000     | %  |
| Gurvinder| obtained | 480       | marks and has | 96.0000     | %  |
| Ali      | obtained | 260       | marks and has | 52.0000     | %  |
| Michelle | obtained | 321       | marks and has | 64.2000     | %  |
| Zubin    | obtained | 412       | marks and has | 82.4000     | %  |
| Simran   | obtained | 378       | marks and has | 75.6000     | %  |
| Fatimah  | obtained | 400       | marks and has | 80.0000     | %  |
| Anup     | obtained | 302       | marks and has | 60.4000     | %  |
| Mita     | obtained | 150       | marks and has | 30.0000     | %  |
+----------+----------+-----------+---------------+-------------+----+
10 rows in set (0.00 sec)
```

Scanned with CamScanner

16.5.13 Selecting Specific Rows – WHERE clause

In real life, tables can contain unlimited rows. There is no need to view all the rows when only certain rows are needed. SQL enables you to define criteria to determine which rows are selected for output. The WHERE clause in SELECT statement specifies the criteria for selection of rows to be returned. The SELECT statement with WHERE clause takes the following general form :

```
SELECT <column name> [, <column name>, ... ]
FROM <table name>
WHERE <condition> ;
```

when a WHERE clause is present, the database program goes through the entire table one row at a time and examines each row to determine if the given condition is true. If it is true for a row, that row is displayed in the output. For example, to display the *name* and *aggregate* for students having their aggregate marks more than 350, the command would be

```
mysql> SELECT name, aggregate
    -> FROM student
    -> WHERE aggregate > 350 ;
```

The above query will produce adjacent output :

Only the records having aggregate > 350 have appeared in the output.

```
+-----------+-----------+
| name      | aggregate |
+-----------+-----------+
| Abu Bakar | 456       |
| Gurvinder | 480       |
| Zubin     | 412       |
| Simran    | 378       |
| Fatimah   | 400       |
+-----------+-----------+
5 rows in set (0.03 sec)
```

EXAMPLE 16.5. *Write a query to display name, age and marks (aggregate) of students whose age is greater than or equal to 16 from table student.*

Solution.

```
mysql> SELECT name,age, aggregate FROM student
    -> WHERE age >= 16 ;
+-----------+-----------+-----------+
| name      | age       | aggregate |
+-----------+-----------+-----------+
| Aanya     | 16        | 340       |
| Ali       | 16        | 260       |
| Mita      | 16        | 150       |
+-----------+-----------+-----------+
3 rows in set (0.02 sec)
```

16.5.14 Relational Operators

To compare two values, a relational operator is used. The result of the comparison is true or false. The SQL recognizes following relational operators :

=, >, <, >=, <=, <> (not equal to)

In CHARACTER data type comparisons, < means earlier in the alphabet and > means later in the alphabet. For example, *e* < *f* and *g* > *f*. Apostrophes are necessary around all CHAR, DATE and TIME data. For example, to list all the members not from 'DELHI'

```
SELECT * FROM suppliers
WHERE city < > 'DELHI' ;
```

Scanned with CamScanner

16.5.15 Logical Operators

The logical operators OR (||), AND (&&) and NOT (!) are used to connect search conditions in the WHERE clause. For example,

1. To list the employees' details having grades 'E2' or 'E3' from table *employee* (not the *EMPL* table), logical operator OR will be used as :

```
SELECT ecode, ename, grade, gross
FROM employee
WHERE (grade = 'E2' OR grade = 'E3') ;
```

Symbol || may also be used as **OR** operator *i.e.*, above query may also be written as :

```
SELECT ecode, ename, grade, gross
FROM employee
WHERE (grade = 'E2'||grade = 'E3') ;
```

2. To list all the employees' details having grades as 'E4' but with gross < 9000, logical operator AND will be used as :

```
SELECT ecode, ename, grade, gross
FROM employee
WHERE (grade = 'E4' AND gross <9000) ;
```

Symbol **&&** may also be used as **AND** operator.

3. To list all the employees' details whose grades are other than 'G1', logical operator NOT will be used as :

```
SELECT ecode, ename, grade, gross
FROM employee
WHERE (NOT grade = 'G1') ;
```

Symbol **!** may also be used as **NOT** operator.

when all the logical operators are used together, the order of precedence is NOT (!), AND (&&), and OR (||). However, parentheses can be used to override the default precedence.

EXAMPLE 16.6. *Write a query to display all the details from pet table for species cat /dog having gender(sex) as male ('m')*

Solution.

```
mysql> SELECT * FROM pet
    -> WHERE (species = 'cat' || species = 'dog') && sex = 'm' ;
+--------+--------+---------+--------+------------+------------+
| name   | owner  | species | sex    | birth      | death      |
+--------+--------+---------+--------+------------+------------+
| Claws  | Gwen   | cat     | m      | 1994-03-17 | NULL       |
| Fang   | Benny  | dog     | m      | 1990-08-27 | NULL       |
| Bowser | Diane  | dog     | m      | 1979-08-31 | 1995-07-29 |
+--------+--------+---------+--------+------------+------------+
3 rows in set (0.00 sec)
```

16.5.16 Condition Based on a Range

The BETWEEN operator defines a range of values that the column values must fall into make the condition true. The range includes both lower value and the upper value. For example, to list the items whose *QOH* falls between 30 to 50 (both inclusive), the command would be :

```
SELECT icode, descp, QOH
FROM items
WHERE QOH BETWEEN 30 AND 50 ;
```

Scanned with CamScanner

If the *Items* Table is as shown below :

Table 16.4 *Items*

Icode	Descp	Price	QOH	ROL	ROQ
I01	Milk	15.00	20	10	20
I02	Cake	5.00	60	20	50
I03	Bread	9.00	40	10	40
I04	Biscuit	10.00	50	40	60
I05	Namkeen	15.00	100	50	70
I06	Cream Roll	7.00	10	20	30

Then the above query will produce the following output :

```
+-----------+-----------+-----------+
| Icode     | Descp     | QOH       |
+-----------+-----------+-----------+
| I03       | Bread     | 40        |
| I04       | Biscuit   | 50        |
+-----------+-----------+-----------+
```

Only the items having QOH between 30 to 50 have been listed

The operator NOT BETWEEN is reverse of BETWEEN operator, that is, the rows not satisfying the BETWEEN condition are retrieved. For example,

```
SELECT icode, descp
FROM items
WHERE ROL NOT BETWEEN 100 AND 1000 ;
```

This query will list the items whose ROL is below 100 or above 1000.

EXAMPLE 16.7. *Write a query to display name and aggregate marks of those students who don't have their aggregate marks in the range of 380 – 425.*

Solution.

```
mysql> SELECT name, aggregate
    -> FROM student
    -> WHERE aggregate NOT BETWEEN 380 AND 425 ;
```

```
+-----------+-----------+
| name      | aggregate |
+-----------+-----------+
| Abu Bakar | 456       |
| Aanya     | 340       |
| Gurvinder | 480       |
| Ali       | 260       |
| Michelle  | 321       |
| Simran    | 378       |
| Anup      | 302       |
| Mita      | 150       |
+-----------+-----------+
8 rows in set (0.00 sec)
```

16.5.17 Condition Based on a List

To specify a list of values, IN operator is used. The IN operator selects values that match any value in a given list of values.

For example, to display a list of members from 'DELHI', 'MUMBAI', 'CHENNAI' or 'BANGALORE' cities, you may give

```
SELECT * FROM members
WHERE city IN ('DELHI', 'MUMBAI', 'CHENNAI', 'BANGALORE') ;
```

The NOT IN operator finds rows that do not match in the list. So if you write

```
SELECT * FROM members
WHERE city NOT IN ('DELHI', 'MUMBAI', 'CHENNAI') ;
```

it will list members not from the cities mentioned in the list.

Scanned with CamScanner

EXAMPLE 16.8. *Write a query to display all details of pets of species bird, snake or hamster from table pet.*

Solution.

```
mysql> SELECT * FROM pet
    -> WHERE species IN ('bird', 'snake', 'hamster');
+---------+--------+---------+--------+------------+-----------+
| name    | owner  | species | sex    | birth      | death     |
+---------+--------+---------+--------+------------+-----------+
| Chirpy  | Gwen   | bird    | f      | 1998-09-11 | NULL      |
| Whistler| Gwen   | bird    | NULL   | 1997-12-09 | NULL      |
| Slim    | Benny  | snake   | m      | 1996-04-29 | NULL      |
| Puffball| Diane  | hamster | f      | 1999-03-30 | NULL      |
+---------+--------+---------+--------+------------+-----------+
4 rows in set (0.00 sec)
```

16.5.18 Condition Based on Pattern Matches

SQL also includes a string-matching operator, LIKE, for comparisons on character strings using patterns. Patterns are described using two special wildcard characters :

- percent (%). The % character matches any substring.
- underscore (_). The _ character matches any character.

Patterns are case-sensitive, that is, upper-case characters do not match lower-case characters, or vice-versa. To illustrate pattern matching, consider the following examples :

- 'San%' matches any string beginning with 'San'
- '%idge%' matches any string containing 'idge' as a substring, for example, 'Ridge', 'Bridges', 'Cartridge', 'Ridgeway' etc.
- '- - - -' matches any string of exactly 4 characters.
- '- - -%' matches any string of at least 3 characters.

The LIKE keyword is used to select rows containing columns that match a wildcard pattern.

Examples :

1. To list members which are in areas with pin codes starting with 13, the command is :

```
SELECT firstname, lastname, city
FROM members
WHERE pin LIKE '13%' ;
```

2. To list names of pets who have names ending with 'y', the command would be :

```
SELECT name
FROM emp
WHERE name LIKE '%y' ;
```

Consider the Table 16.2, the above query will produce the adjacent output :

```
+-----------+
| name      |
+-----------+
| Fluffy    |
| Buffy     |
| Chirpy    |
+-----------+
```

3. To list members which are not in areas with pin codes starting with 13, the command is :

```
SELECT firstname, lastname, city
FROM members
WHERE pin NOT LIKE '13%' ;
```

The keyword NOT LIKE is used to select rows that do not match the specified pattern of characters.

Scanned with CamScanner

In order for patterns to include the special pattern characters (that is, %, _), SQL allows the specific of an escape character. The escape character is used immediately before a special pattern character to indicate that the special pattern character is to be treated as a normal character. We define the escape character for a LIKE comparison using the ESCAPE keyword.

To illustrate, consider the following patterns which use a backslash (\) as the escape character.

- LIKE 'wx\%yz%' ESCAPE '\' matches all strings beginning with 'wx%yz'.
- LIKE 'wx\\yz%' ESCAPE '\' matches all string beginning with 'wx\yz'.

The ESCAPE clause can define any character as an escape character. The above two examples use backslash ('\') as the escape character.

EXAMPLE 16.9 *Write query to display the names of pets beginning with 'F'. Use table Pet.*

Solution.

```
mysql> SELECT * FROM pet
    -> WHERE name LIKE "F%";
+--------+--------+---------+--------+------------+------------+
| name   | owner  | species | sex    | birth      | death      |
+--------+--------+---------+--------+------------+------------+
| Fluffy | Harold | cat     | f      | 1993-02-04 | NULL       |
| Fang   | Benny  | dog     | m      | 1990-08-27 | NULL       |
+--------+--------+---------+--------+------------+------------+
2 rows in set (0.00 sec)
```

EXAMPLE 16.10 *Write query to display the names of pets having exactly four letter names. Use table Pet.*

Solution.

```
mysql> SELECT * FROM pet
    -> WHERE name LIKE "____" ;
```

contains 4 underscore symbols

```
+--------+--------+---------+--------+------------+------------+
| name   | owner  | species | sex    | birth      | death      |
+--------+--------+---------+--------+------------+------------+
| Fang   | Benny  | dog     | m      | 1990-08-27 | NULL       |
| Slim   | Benny  | snake   | m      | 1996-04-29 | NULL       |
+--------+--------+---------+--------+------------+------------+
2 rows in set (0.00 sec)
```

16.5.19 Searching for NULL

The NULL value in a column can be searched for in a table using IS NULL in the WHERE clause. (Relational operators like =, < > etc. can't be used with NULL.) For example, to list details of all employees whose departments contain NULL (*i.e.*, novalue), you use the command :

```
SELECT empno, empname, job
FROM emp
WHERE DeptNo IS NULL ;
```

Non-NULL values in a table can be listed using IS NOT NULL.

EXAMPLE 16.11 *Write query to display the names of pets who are no more. (That is the death column stores a non-null value for them). Use table Pet.*

Solution.

```
mysql> SELECT name FROM pet
    -> WHERE death IS NOT NULL;
+-----------+
| name      |
+-----------+
| Bowser    |
+-----------+
1 row in set (0.02 sec)
```

Scanned with CamScanner

16.5.20 Operator Precedence

When an expression has multiple operators, then the evaluation of expression takes place in the order of operator precedence. The operators with higher precedence are evaluated first.

Operator precedences of MySQL are shown in the following list, from highest precedence to the lowest. Operators that are shown together on a line have the same precedence.

```
!
- (unary minus), ~ (unary bit inversion)
^
*, /, DIV, %, MOD
-, +
<, >
&
|
==, >=, >, <=, <, <>, !=, IS, LIKE, REGEXP, IN BETWEEN, CASE, WHEN, THEN, ELSE
NOT
&&, AND
XOR
||, OR
:=
```

16.5.21 Sorting Results – ORDER BY clause

Whenever a SELECT query is executed, the resulting rows emerge in a predecided order. You can sort the results or a query in a specific order using ORDER BY clause. The ORDER BY clause allows sorting of query results by one or more columns. The sorting can be done either in *ascending* or *descending* order, the default order is *ascending*. The data in the table is not sorted ; only the results that appear on the screen are sorted. The ORDER BY clause is used as :

```
SELECT <column name> [, <column name> , ... ]
FROM <table name>
[WHERE <predicate> ]
[ORDER BY <column name> ] ;
```

For example, to display the list of employees in the alphabetical order of their names, you use the command :

```
SELECT * FROM employee
ORDER BY ename ;
```

To display the list of students having aggregate more than 400 in the alphabetical order of their names, you may give the command :

```
SELECT name, aggregate FROM student
WHERE aggregate > 400
ORDER BY name ;
```

Considering the same *student* table, the above query would produce the output shown here.

```
+-----------+-----------+
| name      | aggregate |
+-----------+-----------+
| Abu Bakar | 456       |
| Gurvinder | 480       |
| Zubin     | 412       |
+-----------+-----------+
3 rows in set (0.03 sec)
```

Scanned with CamScanner

To display the list of employees in the descending order of employee code, you use the command :

```
SELECT * FROM employee
ORDER BY ecode DESC ;
```

To specify the sort order, we may specify DESC for descending order or ASC for ascending order. Furthermore, ordering can be performed on multiple attributes. Suppose that we wish to list the entire *employee* relation in descending order of *grade*. If several employees have the same grade, we order them in ascending order by their names.

We express this in *SQL* as follows :

```
SELECT * FROM employee
ORDER BY grade DESC, ename ASC ;
```

See, the multiple fields are separated by commas. In order to fulfill an ORDER BY request, *SQL* must perform a sort. Since sorting a large number of tuples may be costly, it is desirable to sort only when necessary.

Check Point 16.1

1. Maximum how many characters can be stored in a (*i*) text literal (*ii*) numeric literal ?
2. What is a datatype ? Name some datatypes available in MySQL.
3. What are fixed length fields ? What are variable length fields ?
4. Compare Char and Varchar datatypes.
5. What is null value in MySQL database ? Can you use nulls in arithmetic expressions ?
6. Which keyword eliminates the redundant data from a query result ?
7. Which keyword retains duplicate output rows in a query result ?
8. How would you display system date as the result of a query ?
9. How would you calculate 13 * 15 in SQL ?
10. Which function is used to substitute NULL values in a query result ?
11. Which operator concatenates two strings in a query result ?
12. Which comparison operator is used for comparing ?
    (*i*) patterns (*ii*) character values
    (*iii*) null values (*iv*) ranges
    (*v*) list of values.

EXAMPLE 16.12 *Write a query to display name, age, aggregate of students whose aggregate is between 300 and 400. Order the query in descending order of name.*

Solution.

```
mysql> SELECT name, age, aggregate
    -> FROM student
    -> WHERE aggregate BETWEEN 300 AND 400
    -> ORDER BY name DESC;
```

```
+-----------+-------+------------+
| name      | age   | aggregate  |
+-----------+-------+------------+
| Simran    | 15    | 378        |
| Michelle  | 15    | 321        |
| Fatimah   | 14    | 400        |
| Anup      | 15    | 302        |
| Aanya     | 16    | 340        |
+-----------+-------+------------+
5 rows in set (0.01 sec)
```

Scanned with CamScanner

16.6 MYSQL FUNCTIONS

A **function** is a special type of predefined command set that performs some operation and returns a single value. Functions operate on zero, one, two or more values that are provided to them. The values that are provided to functions are called **parameters** or **arguments**.

The MySQL functions have been categorised into various categories, such as *String functions, Mathematical functions, Date and Time functions* and so on. Although MySQL offers a big bouquet of functions, yet in this discussion we shall remain limited to some common functions.

16.6.1 String Functions

The string functions of MySQL can manipulate the text string in many ways. Some commonly used string functions are being discussed below.

	Function	Description	Examples
1.	CHAR( )	Returns the character for each integer passed	1. SELECT CHAR (70, 65, 67, 69) ; 2. SELECT CHAR (65, 67.3, '69.3) ;
2.	CONCAT( )	Returns concatenated string	SELECT CONCAT(name, aggregate) AS "Name Marks" FROM student WHERE age = 14 OR age = 16 ;
3.	LOWER( ) / LCASE( )	Returns the argument in lowercase	SELECT LOWER('MR. OBAMA') AS "LowerName1", LOWER('Ms. Gandhi') AS "LowerName2" ;
4.	SUBSTRING( ), SUBSTR()	Returns the substring as specified	1. SELECT SUBSTR('ABCDEFG', 3, 4) "Subs" ; 2. SELECT SUBSTR ('ABCDEFG', -5, 4) "Subs"
5.	UPPER()/UCASE( )	Converts to uppercase	SELECT UPPER('Large') "Uppercase" ; or SELECT UCASE('Large') "Uppercase" ;
6.	TRIM( )	Removes leading and trailing spaces	SELECT TRIM(' Bar One ') ;
7.	LENGTH( )	Returns the length of a string in bytes	SELECT LENGTH('CANDIDE') "Length in characters" ;

16.6.2 Numeric Functions

The number functions are those functions that accept numeric values and after performing the required operation, return numeric values. Some useful numeric functions are being discussed below :

	Function	Description	Example
1.	MOD()	Returns the remainder of one expression by diving by another expression.	SELECT MOD(11, 4) "Modulus" ;
2.	POWER()/POW()	Returns the value of one expression raised to the power of another expression	SELECT POWER(3, 2) "Raised" ;
3.	ROUND()	Returns numeric expression rounded to an integer. Can be used to round an expression to a number of decimal points	SELECT ROUND(15.193, 1) "Round" ;
4.	SIGN()	This function returns sign of a given number	SELECT SIGN(-15) "Sign" ;
5.	SQRT()	Returns the non-negative square root of numeric expression.	SELECT SQRT(26) "Square root" ;
6.	TRUNCATE()	Returns numeric exp1 truncated to exp2 decimal places. If exp2 is 0, then the result will have no decimal point.	SELECT TRUNCATE(15.79, 1) "Truncate" ;

Scanned with CamScanner

16.6.3 Date and Time Functions

Date functions operate on values of the DATE datatype.

	Function	Description	Example
1.	CURDATE( ) / CURRENT_DATE( ) / CURRENT_DATE	Returns the current date	SELECT CURDATE( ) ;
2.	DATE( )	Extracts the date part of a date or date-time expression	SELECT DATE('2020-12-31 01:02:03') ;
3.	MONTH( )	Returns the month from the date passed	SELECT MONTH('2020-02-03') ;
4.	YEAR( )	Returns the year	SELECT YEAR('2020-02-03') ;
5.	NOW( )	Returns the time at which the function executes	SELECT NOW( ) ;
6.	SYSDATE( )	Returns the current date and time	SELECT NOW(), SLEEP(2), NOW();

EXAMPLE 16.13 *Write a query to create a string from the ASCII values 70, 65, 67, 69.*

Solution.

```
mysql> SELECT CHAR (70, 65, 67, 69 ) ;
+----------------------------+
| char (70, 65, 67, 69 )     |
+----------------------------+
| FACE                       |
+----------------------------+
1 row in set (0.00 sec)
```

EXAMPLE 16.14 *Concatenate name and aggregate for students having age as 14 or 16.*

(Consider table Student of example 16.3)

Solution.

```
mysql> SELECT CONCAT(name, aggregate) AS "Name Marks" FROM student
    -> WHERE age = 14 OR age = 16 ;
+--------------+
| Name Marks   |
+--------------+
| Aanya340     |
| Gurvinder480 |
| Ali260       |
| Fatimah400   |
| Mita150      |
+--------------+
5 rows in set (0.00 sec)
```

EXAMPLE 16.15 *Display names 'MR. OBAMA' and 'MS. Gandhi' into lowercase.*

Solution.

```
mysql> SELECT LOWER('MR. OBAMA') AS "LowerName1" ,
    -> LOWER('Ms. Gandhi') AS "LowerName2" ;
```

Or

```
mysql> SELECT LCASE('MR. OBAMA') AS "LowerName1" ,
    -> LCASE ('Ms. Gandhi') AS "LowerName2" ;
```

```
+-------------+-------------+
| LowerName1  | LowerName2  |
+-------------+-------------+
| mr. obama   | ms. gandhi  |
+-------------+-------------+
1 row in set (0.00 sec)
```

Scanned with CamScanner

EXAMPLE 16.16 *Display 4 characters extracted from 3rd left character onwards from string 'ABCDEFG'.*

Solution.
```
mysql> SELECT SUBSTR('ABCDEFG', 3, 4) "Subs" ;
+------------+
| Subs       |
+------------+
| CDEF       |
+------------+
```

EXAMPLE 16.17 *Display first three characters extracted from jobs of employees 8888 and 8900.*

Solution.
```
mysql> SELECT SUBSTR(job, 1, 3)
    -> FROM empl
    -> WHERE empno = 8888 or empno = 8900 ;
+--------------------+
| SUBSTR(job, 1, 3)  |
+--------------------+
| ANA                |
| CLE                |
+--------------------+
```

EXAMPLE 16.18 *Write a query to remove leading and trailing spaces from string ' Bar One '.*

Solution.
```
mysql> SELECT TRIM('  Bar One  ') ;
+-----------------------+
| TRIM('  Bar One  ')   |
+-----------------------+
| Bar One               |
+-----------------------+
1 row in set (0.00 sec)
```

EXAMPLE 16.19 *How many characters are there in string 'CANDIDE'. ?*

Solution.
```
mysql> SELECT LENGTH('CANDIDE') "Length in characters" ;
+-----------------------+
| Length in characters  |
+-----------------------+
| 7                     |
+-----------------------+
```

EXAMPLE 16.20 *Write a query to extract a sub string from string* ***'Quadratically'*** *which should be 6 characters long and start from 5th character of the given string.*

Solution.
```
mysql> SELECT SUBSTRING('Quadratically',5,6);
+-----------------------------------+
| SUBSTRING('Quadratically',5,6)    |
+-----------------------------------+
| ratica                            |
+-----------------------------------+
1 row in set (0.02 sec)
```

EXAMPLE 16.21 *Find out the remainder of 11 divided by 4 and display 3^2.*

Solution.
```
mysql> SELECT MOD(11, 4) "Modulus", POWER(3, 2) "Raised" ;
+----------+----------+
| Modulus  | Raised   |
+----------+----------+
| 3        |    9     |
+----------+ +--------+
```

Scanned with CamScanner

EXAMPLE 16.22 *Truncate value 15.79 to 1 decimal place.*

Solution.

```
mysql> SELECT TRUNCATE(15.79, 1) "Truncate" ;
+-----------+
| Truncate  |
+-----------+
| 15.7      |
+-----------+
```

EXAMPLE 16.23 *Write a query to display current date on your system.*

Solution.

```
mysql> SELECT CURDATE() ;
+-----------+
| CURDATE() |
+-----------+
| 2009-06-03|
+-----------+
1 row in set (0.00 sec)
```

EXAMPLE 16.24 *Write a query to display date after 10 days of current date on your system.*

Solution.

```
mysql> SELECT CURDATE() + 10 ;
+-----------------+
| CURDATE() + 10  |
+-----------------+
| 20090613        |
+-----------------+
1 row in set (0.00 sec)
```

when you add a number to a date, then returned value is not formatted as date.

EXAMPLE 16.25 *Write a query to extract date from a given datetime value '2020-12-31 01:02:03'.*

Solution.

```
mysql> SELECT DATE('2020-12-31 01:02:03') ;
+-----------------------------+
| DATE('2020-12-31 01:02:03') |
+-----------------------------+
| 2020-12-31                  |
+-----------------------------+
1 row in set (0.03 sec)
```

EXAMPLE 16.26 *Write a query to extract month part from date 3rd Feb 2020.*

SOLUTION.

```
mysql> SELECT MONTH('2020-02-03') ;
+----------------------+
| MONTH('2020-02-03')  |
+----------------------+
| 2                    |
+----------------------+
1 row in set (0.03 sec)
```

Scanned with CamScanner

EXAMPLE 16.27 *Write a query to display current date and time.*

Solution.

```
mysql> SELECT NOW( ) ;
+---------------------+
| NOW()               |
+---------------------+
| 2009-06-03 15:27:20 |
+---------------------+
1 row in set (0.00 sec)
```

EXAMPLE 16.28 *Write a query to illustrate the difference between now( ) and sysdate( ).*

Solution.

```
mysql> SELECT NOW(), SLEEP(2), NOW();
+---------------------+----------+---------------------+
| NOW()               | SLEEP(2) | NOW()               |
+---------------------+----------+---------------------+
| 2009-06-03 15:32:34 | 0        | 2009-06-03 15:32:34 |
+---------------------+----------+---------------------+
1 row in set (2.01 sec)
```

Notice that in first query the now() gives the same value even after 2 seconds (sleep(2)), but sysdate() values have a difference of 2 seconds because of sleep(2)

```
mysql> SELECT SYSDATE(), SLEEP(2), SYSDATE();
+---------------------+----------+---------------------+
| SYSDATE()           | SLEEP(2) | SYSDATE()           |
+---------------------+----------+---------------------+
| 2009-06-03 15:33:15 | 0        | 2009-06-03 15:33:17 |
+---------------------+----------+---------------------+
1 row in set (2.01 sec)
```

Compare the above *two* results. The *now*( ) function will return the same result even if you use it multiple times in same statement, even after creating delays (such as through *sleep*( ) that creates delay of specified seconds) because the *now*( ) function returns the begin-time of statement, whereas *sysdate*( ) function returns the execution time of its own *i.e.*, at what time it started to execute. After reading these lines, compare the above results once again. Now you will clearly understand the difference between the functionality of the two functions *now*( ) and *sysdate*( ).

16.7 AGGREGATE FUNCTIONS

Till now you have learnt to work with functions that operate on individual rows in a table *e.g.*, if you use **Round( )** function then it will round off values from each row of the table.

MySQL also supports and provides **group functions** or **aggregate functions**. As you can make out that the group functions or aggregate functions work upon groups of rows, rather than on single rows. That is why, these functions are sometimes also called **multiple row functions**.

Many group functions accept the following options :

DISTINCT This option causes a group function to consider only distinct values of the argument expression.

ALL This option causes a group function to consider all values including all duplicates.

Scanned with CamScanner

The usage of these options will become clear with the coverage of examples in this section. All the examples that we'll be using here, shall be based upon following table **empl**.

Table 16.5 Database table empl

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
8369	SMITH	CLERK	8902	1990-12-18	800.00	NULL	20
8499	ANYA	SALESMAN	8698	1991-02-20	1600.00	300.00	30
8521	SETH	SALESMAN	8698	1991-02-22	1250.00	500.00	30
8566	MAHADEVAN	MANAGER	8839	1991-04-02	2985.00	NULL	20
8654	MOMIN	SALESMAN	8698	1991-09-28	1250.00	1400.00	30
8698	BINA	MANAGER	8839	1991-05-01	2850.00	NULL	30
8839	AMIR	PRESIDENT	NULL	1991-11-18	5000.00	NULL	10
8844	KULDEEP	SALESMAN	8698	1991-09-08	1500.00	0.00	30
8882	SHIAVNSH	MANAGER	8839	1991-06-09	2450.00	NULL	10
8886	ANOOP	CLERK	8888	1993-01-12	1100.00	NULL	20
8888	SCOTT	ANALYST	8566	1992-12-09	3000.00	NULL	20
8900	JATIN	CLERK	8698	1991-12-03	950.00	NULL	30
8902	FAKIR	ANALYST	8566	1991-12-03	3000.00	NULL	20
8934	MITA	CLERK	8882	1992-01-23	1300.00	NULL	10

1. AVG

This function computes the average of given data.

SYNTAX

```
AVG([DISTINCT | ALL] n)
```

- Returns average value of parameter(s) *n*.

Argument type : Numeric *Return value* : Numeric

EXAMPLE 16.29. *Calculate average salary of all employees listed in table empl.*

Solution.

```
mysql> SELECT AVG(sal) "Average"
       FROM empl ;
+-------------+
| Average     |
+-------------+
| 2073.928571 |
+-------------+
1 row in set (0.01 sec)
```

2. COUNT

This function counts the number of rows in a given column or expression.

SYNTAX

```
COUNT({ * [DISTINCT | ALL] expr})
```

- Returns the number of rows in the query.
- If you specify argument *expr*, this function returns rows where *expr* is not null. You can count either all rows, or only distinct values of *expr*.
- If you specify the asterisk (*), this function returns all rows, including duplicates and nulls.

Argument type : Numeric *Return value* : Numeric

Scanned with CamScanner

EXAMPLE 16.30. *Count number of records in table empl.*

Solution.

```
mysql> SELECT COUNT(*) "Total"
       FROM empl ;
```

```
+----------+
| Total    |
+----------+
|   14     |
+----------+
1 row in set (0.00 sec)
```

EXAMPLE 16.31. *Count number of jobs in table empl.*

Solution.

```
mysql> SELECT COUNT(job) "Job Count"
       FROM empl ;
```

```
+-------------+
| Job Count   |
+-------------+
|     14      |
+-------------+
1 row in set (0.01 sec)
```

EXAMPLE 16.32. *How many distinct jobs are listed in table empl ?*

Solution.

```
mysql> SELECT COUNT(DISTINCT job) "Distinct Jobs"
       FROM empl ;
```

```
+---------------+
| Distinct Jobs |
+---------------+
|       15      |
+---------------+
1 row in set (0.04 sec)
```

3. MAX

This function returns the maximum value from a given column or expression.

SYNTAX

```
MAX([DISTINCT|ALL] expr)
```

- Returns maximum value of argument *expr*.

Argument type : Numeric *Return value* : Numeric

EXAMPLE 16.33. *Display maximum salary from table empl.*

Solution.

```
mysql>  SELECT MAX(sal) "Maximum Salary"
        FROM empl ;
```

```
+----------------+
| Maximum Salary |
+----------------+
|     5000.00    |
+----------------+
1 row in set (0.01 sec)
```

4. MIN

This function returns the minimum value from a given column or expression.

SYNTAX

```
MIN([DISTINCT | ALL] expr)
```

- Returns minimum value of *expr*.

Argument type: Numeric *Return value* : Numeric

EXAMPLE 16.34. *Display the Joining date of seniormost employee.*

Solution.

```
mysql> SELECT MIN(hiredate) "Minimum Hire Date"
       FROM empl ;
```

```
+-------------------+
| Minimum Hire Date |
+-------------------+
|    1990-12-18     |
+-------------------+
1 row in set (0.06 sec)
```

Scanned with CamScanner

5. SUM

This function returns the sum of values in given column or expression.

SYNTAX

```
SUM([DISTINCT | ALL] n)
```

▲ Returns sum of values of *n*.

Argument type : Numeric *Return value* : Numeric

EXAMPLE 16.35. *Display total salary of all employees listed in table empl.*

Solution.

```
mysql> SELECT SUM(sal) "Total Salary"
       FROM empl ;
```

```
+----------------+
| Total Salary   |
+----------------+
|    29035.00    |
+----------------+
1 row in set (0.01 sec)
```

Some more examples of group functions are being given below :

Examples :

1. To calculate the total gross for employees of grade 'E2', the command is :

```
SELECT SUM(gross) FROM employee
WHERE grade = 'E2' ;
```

NOTE

These functions are called *aggregate functions* because they operate on aggregates of tuples. The result of an aggregate function is a single value.

2. To display the average gross of employees with grades 'E1' or 'E2', the command used is :

```
SELECT AVG(gross) FROM employee
WHERE (grade = 'E1' OR grade = 'E2') ;
```

3. To count the number of employees in *employee* table, the *SQL* command is :

```
SELECT COUNT(*)
FROM employee ;
```

4. To count the number of cities, the different members belong to, you use the following command :

```
SELECT COUNT(DISTINCT city)  FROM members ;
```

Here the DISTINCT keyword ensures that multiple entries of the same *city* are ignored. The * is the only argument that includes NULLs when it is used only with COUNT, functions other than COUNT disregard NULLs in any case.

If you want to count the entries including repeats, the keyword ALL is used. The following command will COUNT the number of non NULL *city* fields in the *members* table :

```
SELECT COUNT(ALL city)
FROM members ;
```

In general, GROUP functions

- Return a single value for a set of rows.
- Can be applied to any numeric values, and some Text types and DATE values.

Scanned with CamScanner