3 & 4

SQL

BY KEITH DVORJAK

SQL:

2 books in 1

Advanced and Elite Level SQL From The Ground Up

The information in the following pages is broadly considered to be a truthful and accurate account of facts and as such any inattention, use or misuse of the information in question by the reader will render any resulting actions solely under their purview. There are no scenarios in which the publisher or the original author of this work can be in any fashion deemed liable for any hardship or damages that may befall them after undertaking information described herein.

Additionally, the information in the following pages is intended only for informational purposes and should thus be thought of as universal. As befitting its nature, it is presented without assurance regarding its prolonged validity or interim quality.

SQL: Advanced Level SQL From The Ground Up

Table of Contents

SQL: Elite Level SQL From The Ground Up

Table of contents

Conclusion

SQL:

Advanced Level SQL From The Ground Up

Introduction

Thank you for downloading SQL: Advanced Level SQL From The Ground Up! This book is the third entry of the DIY SQL series. It is preceded by the books SQL: Beginner Level SQL From The Ground Up and SQL: Intermediate Level SQL From The Ground Up, and assumes that the user is familiar with the contents of that book, including beginner SQL scripting, syntax, and terminology. The books should be taken in chronological order for optimal results. This book will cover intermediate SQL manipulation techniques, with code examples to match the concepts explained.

Thank you again for downloading this book! You have many choices available to you for furthering your SQL scripting knowledge. Thank you for selecting the DIY SQL series as your tool of choice!

Chapter 1 Sets

We are aware of how to work with the rows of a table. So far, we have been using queries that work on one row at a time. Now, I will introduce the concept of Set Operations which will allow you to combine the results of multiple queries in one result set. In SQL, there are three set operators:

1. Union
2. Intersect
3. Except

UNION and UNION ALL

UNION clause is required to combine the results of two or more select statements in a single table. However, it is important that the results of both the queries have same number of columns with compatible data types or else it will not be possible to achieve union results. What you need to know here is that when you use the UNION clause there is no surety about the order in which the rows will appear in the result set. Hence, if you require rows to appear in a specific order then it is important to use ORDER BY clause. For faster results you can use UNION ALL.

We already have a table by the name ENGINEERING_STUDENTS. The content for it is as follows:

```
+ - - - - - - - -+ - - - - - - - - - - - - - + - - - - - - - - - - - - - - - - - +
| ENGG_ID | ENGG_NAME        | STUDENT_STRENGTH |
+ - - - - - - - -+ - - - - - - - - - - - - - + - - - - - - - - - - - - - - - - - +
|             1 | Electronics          |                              150 |
|             2 | Software             |                              250 |
|        4 | Mechanical         |                  150 |
|        5 | Biomedical         |                   72 |
|        6 | Instrumentation    |                   80 |
|        7 | Chemical           |                   75 |
|        8 | Civil              |                   60 |
|        9 | Electronics & Com  |                  250 |
|       10 | Electrical         |                   60 |
|       11 | Genetic            |                  150 |
|       12 | Systems            |                  150 |
|       13 | Aerospace          |                  150 |
+ - - - - - - - -+ - - - - - - - - - - - - - + - - - - - - - - - - - - - - - - +
```

In order to explain the concept of SETS, I will create another table ENGINEERING_STUDENTS2016.

```
CREATE TABLE ENGINEERING_STUDENTS2016(ENGG_ID
smallint NOT NULL AUTO_INCREMENT,ENGG_NAME
varchar(35) NOT NULL, STUDENT_STRENGTH
INT(5),PRIMARY KEY(ENGG_ID));
```

The content of the table is as follows:

```
+ - - - - - - - - -+ - - - - - - - - - - - - - + - - - - - - - - - - - - - - - - -+
| ENGG_ID | ENGG_NAME        | STUDENT_STRENGTH |
+ - - - - - - - - -+ - - - - - - - - - - - - - + - - - - - - - - - - - - - - - - -+
|        1 | Software         |                250 |
|        2 | Genetic          |                 75 |
|        3 | Mechanical       |                150 |
|        4 | Instrumentation  |                150 |
|        5 | Chemical         |                 55 |
|        6 | Biomolecular     |                 60 |
|        7 | Process          |                 60 |
|        8 | Corrosion        |                 60 |
+ - - - - - - - - -+ - - - - - - - - - - - - - + - - - - - - - - - - - - - - - - -+
```

Now, let’s use the UNION clause.

```
SELECT * FROM ENGINEERING_STUDENTS

UNION
```

```
SELECT * FROM ENGINEERING_STUDENTS2016;
```

The number of rows in ENGINEERING_STUDENTS IS 12 and the number of rows in ENGINEERING_STUDENTS2016 is 8. Therefore, the total number of rows in the row sets achieved by the union of two tables should 20.

The difference between UNION and UNION ALL is that when you use UNION ALL in your query the result set would display duplicate rows, which is not the case with usage of UNION.

In order to demonstrate I will add one row identical to ENGINEERING_STUDENTS in ENGINEERING_STUDENTS2016;

```
INSERT into ENGINEERING_STUDENTS2016(ENGG_NAME, STUDENT_STRENGTH) VALUES('Electronics & Com','250' );
```

Now, let’s again try the same UNION clause.

```
SELECT * FROM ENGINEERING_STUDENTS

UNION

SELECT * FROM ENGINEERING_STUDENTS2016;
```

```
+ - - - - - - - -+ - - - - - - - - - - - - - + - - - - - - - - - - - - - - - - -+
| ENGG_ID | ENGG_NAME         | STUDENT_STRENGTH    |
+ - - - - - - - -+ - - - - - - - - - - - - - + - - - - - - - - - - - - - - - - -+
|               1 | Electronics           |                                150 |
|               2 | Software              |                                250 |
|               4 | Mechanical            |                                150 |
|               5 | Biomedical            |                                 72 |
|               6 | Instrumentation       |                                 80 |
|               7 | Chemical              |                                 75 |
|               8 | Civil                 |                                 60 |
|               9 | Electronics & Com     |                                250 |
|              10 | Electrical            |                                 60 |
|              11 | Genetic               |                                150 |
|        12 | Systems           |              150 |
|        13 | Aerospace         |              150 |
|         1 | Software          |              250 |
|         2 | Genetic           |               75 |
|         3 | Mechanical        |              150 |
|         4 | Instrumentation   |              150 |
|         5 | Chemical          |               55 |
|         6 | Biomolecular      |               60 |
|         7 | Process           |               60 |
|         8 | Corrosion         |               60 |
+ - - - - - - - -+ - - - - - - - - - - - - - + - - - - - - - - - - - - - - - - - -+
```

20 rows in set (0.00 sec)

What you need to notice here is that in spite of adding another row to ENGINEERING_STUDENTS2016 the number of rows displayed is still 20.

Now, let’s see what happens if we apply UNION ALL to both the tables.

```
SELECT * FROM ENGINEERING_STUDENTS
UNION ALL
SELECT * FROM ENGINEERING_STUDENTS2016;
+ - - - - - - - -+ - - - - - - - - - - - - - +- - - - - - - - - - - - - - - - - -+
| ENGG_ID | ENGG_NAME         | STUDENT_STRENGTH |
+ - - - - - - - -+ - - - - - - - - - - - - - +- - - - - - - - - - - - - - - - - -+
|         1 | Electronics       |              150 |
|         2 | Software          |              250 |
|         4 | Mechanical        |              150 |
|         5 | Biomedical        |               72 |
|         6 | Instrumentation   |               80 |
|         7 | Chemical          |               75 |
|         8 | Civil             |               60 |
|         9 | Electronics & Com |              250 |
|        10 | Electrical        |               60 |
|        11 | Genetic           |              150 |
|        12 | Systems           |              150 |
|        13 | Aerospace         |              150 |
|          1 | Software              |                        250 |
|          2 | Genetic               |                         75 |
|          3 | Mechanical            |                        150 |
|          4 | Instrumentation       |                        150 |
|          5 | Chemical              |                         55 |
|          6 | Biomolecular          |                         60 |
|          7 | Process               |                         60 |
|          8 | Corrosion             |                         60 |
|          9 | Electronics & Com     |                        250 |
+ - - - - - - - -+ - - - - - - - - - - - - +- - - - - - - - - - - - - - - - - -+
21 rows in set (0.00 sec)
```

Now, the number of rows displayed is 21. This is because when you use UNION ALL it displays all the results including the duplicate rows. Duplicate rows were eliminated from the results in the previous examples.

INTERSECT

When you use INTERSECT clause in your query you will get a result set that displays only those records that are common between two tables. INTERSECT clause does not work with MySQL. However, whenever there is a need, you can create a query using IN clause or EXISTS clause depending on how complex your requirement is. The job of the INTERSECT clause is to check records in two or more tables and if the records exists in all the datasets only then it will be displayed in the result set. Thus , the records that are displayed exists in all the datasets on which intersection is imposed.

The syntax for INTERSECT clause is:

```
SELECT column_names

FROM table_1
```

```
WHERE conditions_if_applicable
INTERSECT
SELECT column_names
FROM table_1
WHERE conditions_if_applicable
```

So,

```
SELECT ENGG_ID  FROM ENGINEERING_STUDENTS
INTERSECT
SELECT ENGG_ID FROM ENGINEERING_STUDENTS2016;
```

The above mentioned statement will not work for MYSQL. In order to implement this in MYSQL you will have to use IN clause.

```
SELECT ENGINEERING_STUDENTS.ENGG_ID FROM
ENGINEERING_STUDENTS WHERE
ENGINEERING_STUDENTS.ENGG_ID IN (SELECT
ENGINEERING_STUDENTS2016.ENGG_ID FROM
ENGINEERING_STUDENTS2016);
```

The result set is as follows:

```
+ - - - - - - - - - +
| ENGG_ID  |
+ - - - - - - - - - +
|              1 |
|              2 |
|              4 |
|              5 |
|             6 |
|             7 |
|             8 |
|         9 |
+---------+
```

Observe here that ENGG_ID 3 is missing as 3 exists for ENGINEERING_STUDENTS2016 but not for ENGINEERING_STUDENTS.

EXCEPT

If you use EXCEPT clause between two tables, the result set will display records that exist in first dataset but not in the second one. The syntax for using EXCEPT clause in MySQL is as follows:

```
SELECT column_names

FROM table_1

WHERE conditions_if_applicable

EXCEPT

SELECT column_names

FROM table_1

WHERE conditions_if_applicable
```

Again MYSQL does not support EXCEPT clause. So, you can use NOT IN clause as replacement for EXCEPT clause.

So,

```
SELECT ENGG_ID  FROM ENGINEERING_STUDENTS

EXCEPT

SELECT ENGG_ID FROM ENGINEERING_STUDENTS2016;
```

Will be something like this in MYSQL:

```
SELECT ENGINEERING_STUDENTS.ENGG_ID FROM
```

```
ENGINEERING_STUDENTS WHERE
ENGINEERING_STUDENTS.ENGG_ID NOT IN (SELECT
ENGINEERING_STUDENTS2016.ENGG_ID FROM
ENGINEERING_STUDENTS2016);
+ - - - - - - - - -+
| ENGG_ID |
+ - - - - - - - - -+
|            10 |
|           11 |
|           12 |
|           13 |
+ - - - - - - - - +
4 rows in set (0.05 sec)
```

Chapter 2 Working with Data

We have worked a lot with data while learning SQL. Here In this chapter we will go one step further and discuss how to generate, convert and manipulate string, numeric and temporal data.

Working with String Data

By now you must have become very familiar with String data. The String data type in SQL can be of following three types and we have already worked with all of these:

1. Char
2. Varchar
3. Text

Let’s have a quick recap. CHAR is used to hold strings of fixed length and in case of MySQL it allows you to hold values up to 255 characters in length. The capability of CHAR differs for different data bases. VARCHAR on the other hand can hold strings of much longer length. A VARCHAR in MYSQL can hold up to 65,535 characters in column. When you want strings to hold very large strings of varying length you would be undoubtedly opting for TEXT which can hold up to 4 GB of data. TEXT can further be categorized as TINYTEXT, TEXT, MEDIUMTEXT and LONG TEXT.

With this information in mind let’s get started with String Generation and Manipulation.

Let’s create a table TABLE_OF_STRING as follows:

```
CREATE TABLE TABLE_OF_STRINGS
(
STRING_CHAR CHAR(20),
STRING_VARCHAR VARCHAR(20),
STRING_TEXT TEXT
);
```

Check the table description in the command prompt:

```
desc TABLE_OF_STRINGS;
+ - - - - - - - - - - - - - - - -+ - - - - - - - - - -+ - - - - + - - - + - - - - - - - +  - - - +
| Field                    | Type          | Null | Key | Default | Extra |
+ - - - - - - - - - - - - - - - -+ - - - - - - - - - -+ - - - - + - - - + - - - - - - - +  - - - +
| STRING_CHAR          | char(20)      | YES  |        | NULL     |         |
| STRING_VARCHAR  | varchar(20) | YES  |        | NULL     |         |
| STRING_TEXT          | text             | YES  |        | NULL     |         |
+ - - - - - - - - - - - - - - - -+ - - - - - - - - - -+ - - - - + - - - + - - - - - - - +  - - - +
3 rows in set (0.05 sec)
```

So, you have three fields in this table:

1. STRING_CHAR that takes up to 20 characters
2. STRING_VARCHAR which again has a length of 20 characters
3. STRING_TEXT which takes text values

With these three fields we will study the behaviour of these three types of strings.

We begin with generation of data for this table. Data is inserted using the INSERT statement.

```
INSERT into TABLE_OF_STRINGS(STRING_CHAR,
STRING_VARCHAR, STRING_TEXT) VALUES('i am char', 'i
am varchar','i am text' );
```

While inserting the data all values are quoted in single quote.

If you try to insert a value that exceeds the length, then either the server will throw an exception or truncate the string without giving you any indication about it. MySQL comes in the second category.

Try the following insert statement:

```
INSERT into TABLE_OF_STRINGS(STRING_CHAR,
STRING_VARCHAR, STRING_TEXT)
VALUES('0123456789012345678900', '0123456789','0123456789'
);
```

Now look at the contents of the table:

```
SELECT * FROM TABLE_OF_STRINGS;
+ - - - - - - - - - - - - - - - - - - - -+ - - - -  -- -- - - - - - - - - - + - - - - - - - - - - - +
| STRING_CHAR                   | STRING_VARCHAR   | STRING_TEXT |
+ - - - - - - - - - - - - - - - - - - - -+ - - - -  -- -- - - - - - - - - - + - - - - - - - - - - - +
| i am char                          | i am varchar             | i am text          |
| 01234567890123456789 | 0123456789            | 0123456789     |
+ - - - - - - - - - - - - - - - - - - - -+ - - - -  -- -- - - - - - - - - - + - - - - - - - - - - - +
```

The length of STRING_CHAR is 20. We try to insert the value ‘0123456789012345678900’ which is 22 character lengths. The value that gets stored is ‘01234567890123456789’

Same way the length of STRING_VARCHAR is 20 . Try to update the value of this field with a value that has character length of more than 20.

```
UPDATE TABLE_OF_STRINGS SET STRING_VARCHAR
='0123456789012345678901234567890'
WHERESTRING_CHAR='i am char';
```

```
SELECT * FROM TABLE_OF_STRINGS;
+ -  - - - - - - - - - - - - - - - - - - +- - -  -  - - - - - -  -  - - -  -  -  + - - -  -  -  - - - - -
- -  - +
| STRING_CHAR                  | STRING_VARCHAR          |
STRING_TEXT          |
+ -  - - - - - - - - - - - - - - - - - + - - -  - - - - - -  - - - -  -  + - - -  -  - - - - -
- -  - +
| i am char                          | 01234567890123456789 | i am text                 |
| 01234567890123456789 | 0123456789                        | 0123456789           |
+ -  - - - - - - - - - - - - - - - - - + - - -  - - - - - -  - - - -  -  + - - -  -  - - - - -
- -  - +
2 rows in set (0.05 sec)
```

This happens because by default the sql_mode of my server is not set to strict mode. However , mode for the server can be changed. I will do the same for mine so that if there is any invalid value the server will throw exception.

Let’s first check the SQL mode for our server. To do this you need to give the following instruction at command prompt:

```
SELECT @@session.sql_mode;
+ - - - - - - - - - - - - - - - - - - - +
| @@session.sql_mode        |
+ - - - - - - - - - - -  - - - -- - - - - +
|                                        |
+ - - - - - - - - - - -  - - - -- - - - - +
1 row in set (0.00 sec)
```

To change the value of sql_mode to strict mode I will give the following command:

```
SET SESSION sql_mode='STRICT_TRANS_TABLES';

Query OK, 0 rows affected (0.02 sec)
```

```
\\ SELECT @@session.sql_mode;
+ - - - - - - - -  - - - - - - -  +
| @@session.sql_mode   |
+ - - - - - - - -  - - - - - - -  +
| STRICT_TRANS_TABLES |
+ - - - - - - - -  - - - - - - -  +
1 row in set (0.00 sec)
```

Now that we have set the mode to strict let’s see what happens if we try to feed in a value of longer length.

```
UPDATE TABLE_OF_STRINGS SET STRING_CHAR
='012345678901234567890123456789O

' WHERE STRING_TEXT='i am text';

ERROR 1406 (22001): Data too long for column 'STRING_CHAR'
at row 1
```

So, now you are not allowed to update the value as it is not as per the defined character length.

Now, let’s reduce the length a bit and see if we still face problem again:

```
UPDATE TABLE_OF_STRINGS SET STRING_CHAR
='01234567890' WHERE STRING_TEXT='i am text';

ERROR 1406 (22001): Data too long for column 'STRING_CHAR'
at row 1
```

If you want to check the warnings you will get the following result set:

```
+ - - - - - + - - - - - + - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
+
| Level   | Code   | Message                                                          |
+ - - - - - + - - - - - + - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
+
| Error   | 1406   | Data too long for column 'STRING_CHAR' at row 1 |
+-----+-----+--------------------------------------
+
1 row in set (0.00 sec)
```

Inserting Single quotes along with String values

If you try to insert a string value that has an apostrophe, you will have to be very careful. When the server encounters an apostrophe it can consider it as an end of string.

Let’s try out an example before we proceed further. Look at the query given below. We want to store string ‘I’m a char’ as one of the values for the column name STRING_CHAR.

```
UPDATE TABLE_OF_STRINGS SET STRING_CHAR ='I’ m a
char' WHERE STRING_TEXT='i am text';
```

Let’s execute the query:

```
update TABLE_OF_STRINGS SET STRING_CHAR ='I' m a char'
WHERE STRING_TEXT=

'i am text';

    '>
```

The server takes ‘I’ as one string and moves cursor to next line. In order to make the server accept the apostrophe as a regular character you must add an escape to the string. Here is how you work with apostrophes:

```
UPDATE TABLE_OF_STRINGS SET STRING_CHAR ='I’’ m a
char' WHERE STRING_TEXT='i am text';

UPDATE TABLE_OF_STRINGS SET STRING_CHAR ='I'' m a
char' WHERE STRING_TEXT

='i am text';

Query OK, 1 row affected (0.11 sec)
```

Rows matched: 1 Changed: 1 Warnings: 0

Now, let’s have a look at the contents of TABLE_OF_STRINGS

```
SELECT * FROM TABLE_OF_STRINGS;
+ - - - - - - - - - - - - - - - - - - + - - - - - - - - - - - - - - - - - - + - - - - - - - - - - -
+
| STRING_CHAR                    | STRING_VARCHAR           |
STRING_TEXT |
+ - - - - - - - - - - - - - - - - - - + - - - - - - - - - - - - - - - - - - + - - - - - - - - - - -
+
| I' m a char                          | 01234567890123456789 | i am text        |
| 01234567890123456789 | 0123456789                        | 0123456789  |
+ - - - - - - - - - - - - - - - - - - + - - - - - - - - - - - - - - - - - - + - - - - - - - - - - -
+
2 rows in set (0.00 sec)
```

Char() Function

If you want to insert special characters that are not part of English language then you can access the ASCII character set with the help of MySQL’s in built function char(). There are 255 special characters defined.

Try out the following command on the command prompt:

```
SELECT char(156, 151, 150);

+ - - - - - - - -  - -  - - - - +
| char(156, 151, 150) |
+ - - - - - - - -  - -  - - - - +
| £ùû                            |
+ - - - - - - - -  - -  - - - - +
1 row in set (0.05 sec)
```

You can use concat() function to add a special character in a string.

```
SELECT CONCAT('tr',char(130),'s bien');
+- - - - -  - - - - - - - - - - - - - - - - - - -+
| CONCAT('tr',char(130),'s bien') |
+- - - - -  - - - - - - - - - - - - - - - - - -+
| trés bien                                  |
+- - - - -  - - - - - - - - - - - - - - - - - - -+
1 row in set (0.00 sec)
```

Length() function

The inbuilt length() function returns a number which is the length of the string.

```
SELECT LENGTH(STRING_CHAR)
CHARLEN,LENGTH(STRING_VARCHAR) VARCHARLEN,L
TH(STRING_TEXT) STRINGLEN FROM TABLE_OF_STRINGS;
+ - - - - - - - - -+ - - - - - -- - - - + - - - - - - - - -- -+
| CHARLEN | VARCHARLEN | STRINGLEN |
+ - - - - - - - - -+ - - - - - -- - - - + - - - - - - - - -- -+
|            11 |                 20 |                  9 |
|            20 |                 10 |                10 |
+ - - - - - - - - -+ - - - - - -- - - - + - - - - - - - - -- - +
```

Postion() function

If you want to know the position of a character in a string you can use the position() function.

```
SELECT POSITION('m' IN STRING_CHAR) FROM
TABLE_OF_STRINGS;
+ - - - - - - - - - - - - - - - - - - - - - - - - - - +
| POSITION('m' IN STRING_CHAR) |
+ - - - - - - - - - - - - - - - - - - - - - - - - - - +
|                                            4 |
|                                            0 |
+ - - - - - - - - - - - - - - - - - - - - - - - - - - +


2 rows in set (0.04 sec)
```

Locate() function

Another function is locate(), which can be used instead of position(). The only difference is that you have to mention the position in string from where you want the server to look for that character.

```
SELECT LOCATE('m',STRING_CHAR,1) FROM TABLE_OF_STRINGS;
+ - - - - - - - - - - - - - - - - - - - - - - - +
| LOCATE('m',STRING_CHAR,1) |
+ - - - - - - - - - - - - - - - - - - - - - - - +
|                                          4 |
|                                         0 |
+ - - - - - - - - - - - - - - - - - - - - - - +
```

2 rows in set (0.05 sec)

Strcmp() function

strcmp() function is used to compare two strings. It takes two strings as arguments and returns one of the three values from the following:

1 : when second string is smaller than the first one.

0 : when both string are same.

-1 : when first string is smaller than the second one.

Have a look at the following example:

```
SELECT STRCMP('12345667','21') VALUE1,STRCMP('monkey','man')
VALUE2,STRCM
P('cat','cat') VALUE3;
+ - - - - - -+ - - - - - -  -+ - - - - - - -+
| VALUE1 | VALUE2 | VALUE3 |
+ - - - - - - + - - - - - - -+ - - -  - - - +
|        -1 |            1 |          0 |
+ - - - - - -+ - - - - - -  -+ - - - - - - -+
```

1 row in set (0.00 sec)

replace() function

You can use the replace() function to substitute a part of the string with another string.

```
SELECT REPLACE('hello all','all','world');
+ -  - - - - - - - - - - - - - - - - - - - - - - -+
| REPLACE('hello all','all','world') |
+ - - - - - - - - - - - - - - - - - - - - - - - - -+
| hello world                                    |
+ - - - - - - - - - - - - - - - - - - - - - - - - - +
1 row in set (0.00 sec)1 row in set (0.00 sec)
```

In the above statement, the first string argument in the replace() function is the string in which you want to replace a value. The second string is the part of the string that needs replacement and the third string is the new value.

Working with Numeric data

In this section we will once again have a look at how to work with numeric data.

Arithmetic operations with numeric data

```
SELECT (2+89);
+ - - - - -  -+
| (2+89) |
+ - - - - - +
|    91   |
+ - - - - -+
1 row in set (0.06 sec)
SELECT(98-65);
+ - - - - - - +
| (98-65) |
+ - - - - - - +
|        33 |
+ - - - - - -+
1 row in set (0.05 sec)
SELECT(78*34);
+ - - - - - - - - - - +
|      (78*34) |
+ - - - - - - - - - -+
|          2652 |
+ - - - - - - - - - -+
1 row in set (0.00 sec)
SELECT(67/78);
+ - - - - - - - +
| (67/78) |
+ - - - - - - - +
|  0.8590 |
+ - - - - - - -+
1 row in set (0.00 sec)
```

Advanced Mathematical Functions

You can also find the value of the following functions:

All trigonometric functions such as cos(x), sin(x) etc can be used for calculations. Besides these the other functions that are available are:

Exp(x) returns the value of the base of natural logarithm number e ,

Ln(x) for finding the value of log,

Sqrt(x) for finding the value of a square root

So, let’s try a few examples. The square root of 2 will be

```
SELECT SQRT(2);
+ - - - - - - - - - - - - - - - +
| SQRT(2)                |
+ - - - - - - - - - - - - - - - +
| 1.4142135623731 |
+ - - - - - - - - - - - - - - - +
1 row in set (0.00 sec)
```

```
SELECT EXP(3);
+ - - - - - - - - - - - - - - - - - +
| EXP(3)                            |
+ - - - - - - - - - - - - - - - - - +
| 20.085536923188       |
+ - - - - - - - - - - - - - - - - - +
1 row in set (0.00 sec)
```

You can use modular operator Mod() to find out the remainder after one number is divided by the other.

```
SELECT MOD(9.4,3);
+ - - - - - - - - - - +
| MOD(9.4,3)   |
+ - - - - - - - - - - +
|                0.4 |
+ - - - - - - - - - - +
1 row in set (0.00 sec)
```

Use POW() function to find the value of a number raise to some power. So, if you want to find what is 2 raised to the power 8 then:

```
SELECT POW(2,8);
+ - - - - - - - - - - +
| POW(2,8)      |
+ - - - - - - - -  - -+
|              256  |
+ - - - - - - - - - - +
1 row in set (0.00 sec)
```

You can use functions such as ceil(), floor(), round() and truncate() for limiting the precision of floating point number.

Ceil() function will provide the smallest integer value that is not less than the number that you specify in the argument.

```
SELECT CEIL(9.1);
+ - - - - - - - +
| CEIL(9.1) |
+ - - - - - - - +
|           10 |
+ - - - - - - - +
1 row in set (0.05 sec)
SELECT CEIL(9.9);
+ - - - - - - - +
| CEIL(9.9) |
+ - - - - - - - +
|           10 |
+ - - - - - - - +
1 row in set (0.00 sec)
```

The Floor() function will return the largest integer value which is not greater than the number that you specify in the argument.

```
SELECT FLOOR(8.9);
+ - - - - - - - - - +
| FLOOR(8.9) |
+ - - - - - - - - - +
|                  8 |
+ - - - - - - - - - +
1 row in set (0.05 sec)
SELECT FLOOR (5.1);
+ - - -  - -  - - - - +
| FLOOR (5.1) |
+ - - - - - - - - - +
|                 5 |
+ - - - - - - - - - +
1 row in set (0.00 sec)
```

The round() function is used to return a number rounded to the specified number of decimal digits.

```
SELECT ROUND(9,2);
+ - - - - - - - - - - +
| ROUND(9,2) |
+ - - - - - - - - - - +
|             9.00 |
+ - - - - - - - - - -+
1 row in set (0.06 sec)
SELECT ROUND(134.8686484,3);
+ - - - - - - - - - - - - - - - - - - +
| ROUND(134.8686484,3) |
+ - - - - - - - - - - - - - - - - - - - +
|                              134.869 |
+ - - - - - - - - - - - - - - - - - - - +
1 row in set (0.00 sec)
```

The truncate() function returns the number truncated to the specified number of places.

```
SELECT TRUNCATE(7,2);
+ -  - - - - - - - - - - - +
| TRUNCATE(7,2) |
+ - - - - - - - - - - -  - -+
|                    7.00 |
+ - - - - - - - - - - - - - +
1 row in set (0.02 sec)
SELECT TRUNCATE(5.467863347, 3);
+ - - - -  - - - -  -  -  -  - - - - - - - - +
| TRUNCATE(5.467863347, 3) |
+ - - - -  - - - - - - - - - - - - - - - -  -+
|                                        5.467 |
+ - - - - - - - - - - - - - - - - - - - - - - +
1 row in set (0.00 sec)
```

MySQL and Temporal Data

Temporal data is about built-in time aspects. In case of most of the database servers, the default setting is that of the server on which it resides. In case of MySQL, there are two different types of time zone settings: (1) global time zone (2) session time zone.

```
SELECT @@global.time_zone;
+ - - - - - - - - - - - - - - - - - +
| @@global.time_zone |
+ - - - - - - - - - - - - - - - - - +
| SYSTEM                         |
+ - - - - - - - - - - - - - - - - - +
1 row in set (0.08 sec)
```

The value ‘SYSTEM’ in the result set indicates that the server is making use of the time zone set on the server. Sitting in any part of the world you start a session across the network to a MySQL server located in any other location all that you need to do is change the time zone setting for your session.

```
SET time_zone='+00:00';
```

```
Query OK, 0 rows affected (0.02 sec)
```

Temporal data can be created by copying data from existing date, datetime or time column, by calling an inbuilt function that returns a date, time or datetime or by representing temporal data in a string and then letting the server evaluate it.

```
SELECT @@session.time_zone;
+ - - - - - - - - - - - - - - - - - - +
| @@session.time_zone |
+ - - - - - - - - - - - - - - - - - - +
| +00:00                    |
+ - - - - - - - - - - - - - - - - - - +
1 row in set (0.00 sec)
```

How to work with string representations of temporal data

Date formats are defined as following in MySQL:

1. YYYY stands for year and valid values can be anywhere between 1000 to 9999.
2. MM stands for month and valid value can be anywhere between 01 to 12.
3. DD stands for day and can be anywhere between 01 ton 31.
4. HH stands for hour and the valid value can be anywhere between 00 to 23.
5. HHH stands for hours elapsed and the value can be anywhere between -838 to 838.
6. MI stands for minute and can have any value between 00

to 59.

7. SS stands for second and can have any value between 00 to 59.

In order to create a string value that a server can take as a valid date, time or datetime you will have to provide value values as shown below:

1. The format for date is YYYY-MM-DD.
2. The format for datetime is YYYY-MM-DD HH:MI:SS
3. The format for timestamp is YYYY-MM-DD HH:MI:SS
4. The format for time is HH:MI:SS

So, time stamp for 1:00 PM, 3 rd October 2017 will be as follows:

‘2017-10-03 13:00:00’.

Cast() function

If you want to use a datetime in a format other than the default format then you will have to inform the server to convert the string value that you have provided to a valid datetime format. In this case we can use the cast() function.

Suppose, we represent 1:00 PM, 3 rd October 2017 as 20171003130000.

```
SELECT CAST('20171003130000' AS DATETIME);
+ - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -+
| CAST('20171003130000' AS DATETIME)  |
+ - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -+
| 2017-10-03 13:00:00                              |
+ - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -+
1 row in set (0.00 sec)
```

Same logic is applicable to date and time field.

```
SELECT CAST('2017-10-03' AS DATE);
+ - - - - - - - - - -- - - - - - - - - - - - +
| CAST('2017-10-03' AS DATE) |
+ - - - - -- - -- - - - - - - - - - - - - - - -+
| 2017-10-03                       |
+ - - - - - - - - - - - - - - - - - - - - - - +
1 row in set (0.02 sec)
SELECT CAST('130000' AS TIME);
+ - - - - - - - - - - - - - - - - - - - - - - - -+
| CAST('130000' AS TIME)           |
+ - - - - - - - - - - - - - - - - - - - - - - - - +
| 13:00:00                                 |
+ - - - - - - - - - - - - - - - - - - - - - - - - +
1 row in set (0.00 sec)
```

Date_add() function

Whenever there is a need to add an interval of time to a date you can make use of the date_add() function.

The intervals of time that can be added are:

1. Second: Number of seconds
2. Minute: Number of minutes
3. Hour: Number of hours
4. Day: Number of Days
5. Month: Number of Months
6. Year: Number of years
7. Minute_Second: Minutes and seconds separated by semicolon(:)
8. Hour_Second: Hours, Minutes and seconds separated by semicolon(:)
9. Year_Month: Years and months separated by hyphen (-)

So, now let’s try out few examples:

The current date is:

```
SELECT current_date();
+ - - - - - - - - - - - - +
| current_date() |
+ - - - - - - - - - - - - +
| 2017-11-01       |
+ - - - - - - - - - - - - +
1 row in set (0.06 sec)
```

Now, let’s add 7 years to the current date.

```
SELECT DATE_ADD(CURRENT_DATE(), INTERVAL 7 YEAR);
+ - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - +
| DATE_ADD(CURRENT_DATE(), INTERVAL 7 YEAR) |
+ - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - +
| 2024-11-01                                                              |
+ - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - +
1 row in set (0.03 sec)
SELECT DATE_ADD("1978-06-15", INTERVAL '9-11' year_month);
+ - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -+
| DATE_ADD("1978-06-15", INTERVAL '9-11' year_month) |
+ - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -+
| 1988-05-15                                                                          |
+ - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -+
1 row in set (0.00 sec)
```

Last_day() function

last_day() is another MySQL function that returns a date. It takes a date as an argument and returns the last date for that month.

```
select last_day('2017-11-10');
+- - - - - - - - - - - - - - - - - +
| last_day('2017-11-10') |
+ - - - - - - - - - - - - - - - - - -+
| 2017-11-30                 |
+ - - - - - - - - - - - - - - - - - +
1 row in set (0.00 sec)
```

current_timestamp() function

You can check the current timestamp using current_timestamp() function.

```
select current_timestamp();
+ - - - - - - - - - - - - - - - - - - - - - +
| current_timestamp()           |
+ - - - - - - - - - - - - - - - - - - - - - +
| 2017-11-03 11:45:53           |
+ - - - - - - - - - - - - - - - - - - - - - +
1 row in set (0.00 sec)
```

convert_tz() function

You can convert the present time zone to another time zone using convert_tz() function.

The following statement converts the present time stamp 2017-11-03 12:05:11 from +00:00 time zone to +10:00 time zone.

```
SELECT CONVERT_TZ(CURRENT_TIMESTAMP(),'+00:00','+10:00');
+---------------------------------------------------+
| CONVERT_TZ(CURRENT_TIMESTAMP(),'+00:00','+10:00') |
+---------------------------------------------------+
| 2017-11-03 22:05:13                               |
+---------------------------------------------------+
1 row in set (0.00 sec)
```

dayname() function

If you want to know the name of a day for a date, you can use the dayname() function.

```
SELECT DAYNAME('2017-11-03');
+ - - - - - - - - - - - - - - - - - - - +
| DAYNAME('2017-11-03') |
+ - - - - - - - - - - - - - - - - - - - +
| Friday                |
+ - - - - - - - - - - - - - - - - - - - +
1 row in set (0.08 sec)
```

extract() function

You can use extract() function to retrieve the element of your choice from a date.

```
SELECT EXTRACT(YEAR FROM '2017-11-03 22:05:13');
+ - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - +
| EXTRACT(YEAR FROM '2017-11-03 22:05:13') |
+ - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -  +
|                                                    2017 |
+ - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - +
1 row in set (0.03 sec)

SELECT EXTRACT(MONTH FROM '2017-11-03 22:05:13');
+ - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -+
| EXTRACT(MONTH FROM '2017-11-03 22:05:13')   |
+ - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -+
|                                                            11 |
+ - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -+
1 row in set (0.02 sec)
SELECT EXTRACT(DAY FROM '2017-11-03 22:05:13');
+ - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - +
EXTRACT(DAY FROM '2017-11-03 22:05:13') |
+ - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - +
|                                                      3 |
+ - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - +
1 row in set (0.00 sec)

SELECT EXTRACT(HOUR FROM '2017-11-03 22:05:13');
+ - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -+
| EXTRACT(HOUR FROM '2017-11-03 22:05:13')  |
+ - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -+
|                                                        22 |
+ - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -+
1 row in set (0.00 sec)

SELECT EXTRACT(MINUTE FROM '2017-11-03 22:05:13');
+ - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -+
| EXTRACT(MINUTE FROM '2017-11-03 22:05:13') |
+ - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -+
|                                                              5 |
+ - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -+
1 row in set (0.00 sec)
SELECT EXTRACT(SECOND FROM '2017-11-03 22:05:13');
```

datediff() function

To know the number of days between two dates you can use the datediff() function as shown below:

```
SELECT DATEDIFF('2017-12-02','2017-11-02');
+ - - - - - - - - - - - - - - - - - - - - - - - - - - - -+
| DATEDIFF('2017-12-02','2017-11-02') |
+ - - - - - - - - - - - - - - - - - - - - - - - - - - - -+
|                                            30 |
+ - - - - - - - - - - - - - - - - - - - - - - - - - - - -+
1 row in set (0.00 sec)
SELECT DATEDIFF('17-11-03 22:05:13','2017-10-13 02:05:13');
+ - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - +
| DATEDIFF('17-11-03 22:05:13','2017-10-13 02:05:13') |
+ - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - +
|                                                                    21 |
+ - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - +
1 row in set (0.00 sec)
```

Chapter 3: Grouping and Aggregates

You are already familiar with the importance of grouping data. Let’s have a look at the GROUP BY clause once again. Once again this is how the ENGINEERING_STUDENTS table looks like.

```
SELECT * FROM ENGINEERING_STUDENTS;
+ - - - - - - - -+ - - - - - - - - - - - - - -+ - - - - - - - - - - - - - - - - - +
| ENGG_ID | ENGG_NAME         | STUDENT_STRENGTH |
+ - - - - - - - -+ - - - - - - - - - - - - - -+ - - - - - - - - - - - - - - - - - +
|              1 | Electronics            |                              150 |
|              2 | Software               |                              250 |
|              4 | Mechanical             |                              150 |
|              5 | Biomedical             |                               72 |
|              6 | Instrumentation    |                            80 |
|              7 | Chemical               |                               75 |
|              8 | Civil                  |                                 60 |
|              9 | Electronics & Com |                            250 |
|            10 | Electrical             |                               60 |
|            11 | Genetic                |                              150 |
|            12 | Systems                |                              150 |
|            13 | Aerospace              |                            150 |
+ - - - - - - - -+ - - - - - - - - - - - - - -+ - - - - - - - - - - - - - - - - - +
12 rows in set (0.06 sec)
```

Groupings

Now suppose, we want to see what the class strength in general is. In this case we will retrieve data from the ENGINEERING_STUDENTS table and GROUP BY STUDENT_STRENGTH. The result set that is generated will display one row for every distinct value of student_strength.

```
SELECT STUDENT_STRENGTH FROM ENGINEERING_STUDENTS
GROUP BY STUDENT_STRENGTH;
+ - - - - - - - - - - - - - - - - - +
| STUDENT_STRENGTH |
+ - - - - - - - - - - - - - - - - - +
|                               60 |
|                               72 |
|                               75 |
|                               80 |
|                             150 |
|                             250 |
+ - - - - - - - - - - - - - - - - - +
6 rows in set (0.25 sec)
```

Now if we want to know how many fields in total have same STUDENT_STRENGTH, we can use the count() function.

```
SELECT STUDENT_STRENGTH, COUNT(*) NO_OF_DEPT FROM
ENGINEERING_STUDENTS GR
OUP BY STUDENT_STRENGTH;
+ - - - - - - - - - - - - - - - - -+ - - - - - - - - - - - +
| STUDENT_STRENGTH | NO_OF_DEPT |
+ - - - - - - - - - - - - - - - - -+ - - - - - - - - - - - +
|                               60 |                     2 |
|                               72 |                     1 |
|                               75 |                     1 |
|                               80 |                     1 |
|                             150 |                     5 |
|                             250 |                     2 |
+ - - - - - - - - - - - - - - - - -+ - - - - - - - - - - - +
```

The count() counts number of rows for each distinct value of the field on which the GROUP BY clause is applied. This function will count the number of rows for every group. By putting an asterisk in the parenthesis, we are asking to count everything in the group.

The following statement uses count() function to filter out results.

```
SELECT STUDENT_STRENGTH, COUNT(*) NO_OF_DEPT FROM
```

```
ENGINEERING_STUDENTS GR
OUP BY STUDENT_STRENGTH HAVING COUNT(*)>4;
+ - - - - - - - - - - - - - - - - - + - - - - - - - - - - - +
| STUDENT_STRENGTH | NO_OF_DEPT |
+ - - - - - - - - - - - - - - - - - + - - - - - - - - - - - +
|                          150 |                  5 |
+ - - - - - - - - - - - - - - - - - + - - - - - - - - - - - +
1 row in set (0.06 sec)
```

Aggregate Functions

Aggregate functions can be performed on all the rows of a group. Following are the aggregate functions that can be used with all servers:

1. To get the maximum value within a set use Max() function
2. To get minimum value within a set use Min() function
3. To get average value across a set use Avg() function
4. To get sum of value across a set use Sum() function
5. To get the number of values in a set use count() function

Let’s try out all these functions:

```
SELECT MAX(STUDENT_STRENGTH) FROM
ENGINEERING_STUDENTS;
+ - - - - - - - - - - - - - - - - - - - - - - +
| MAX(STUDENT_STRENGTH) |
+ - - - - - - - - - - - - - - - - - - - - - - +
|                                   250 |
+ - - - - - - - - - - - - - - - - - - - - - - +
1 row in set (0.05 sec)
SELECT MIN(STUDENT_STRENGTH) FROM
ENGINEERING_STUDENTS;
+ - - - - - - - - - - - - - - - - - - - - - -+
| MIN(STUDENT_STRENGTH) |
+ - - - - - - - - - - - - - - - - - - - - - -+
|                                    60 |
+ - - - - - - - - - - - - - - - - - - - - - -+
1 row in set (0.00 sec)
SELECT AVG(STUDENT_STRENGTH) FROM
ENGINEERING_STUDENTS;
+ - - - - - - - - - - - - - - - - - - - - - -+
| AVG(STUDENT_STRENGTH) |
+ - - - - - - - - - - - - - - - - - - - - - -+
|                              133.0833 |
+ - - - - - - - - - - - - - - - - - - - - - -+
1 row in set (0.05 sec)
SELECT SUM(STUDENT_STRENGTH) FROM
ENGINEERING_STUDENTS;
+ - - - - - - - - - - - - - - - - - - - - - - +
| SUM(STUDENT_STRENGTH) |
+ - - - - - - - - - - - - - - - - - - - - - - +
|                                     1597 |
+ - - - - - - - - - - - - - - - - - - - - - - +
1 row in set (0.00 sec)
```

We have already seen how the count() function works.

```
SELECT STUDENT_STRENGTH, COUNT(*) NO_OF_DEPT FROM
ENGINEERING_STUDENTS GR
OUP BY STUDENT_STRENGTH;
+ - - - - - - - - - - - - - - - -+ - - - - - - - - - - - +
| STUDENT_STRENGTH | NO_OF_DEPT |
+ - - - - - - - - - - - - - - - -+ - - - - - - - - - - - +
|                              60 |                    2 |
|                              72 |                    1 |
|                              75 |                    1 |
|                             80 |                     1 |
|                           150 |                     5 |
|                           250 |                     2 |
+ - - - - - - - - - - - - - - - -+ - - - - - - - - - - - +
```

Now let’s replace COUNT(*) BY
COUNT(STUDENT_STRENGTH) and see what happens.

```
SELECT  COUNT(STUDENT_STRENGTH) FROM
ENGINEERING_STUDENTS;
```

```
+ - - - - - - - - - - - - - - - - - - - - - - - - +
| COUNT(STUDENT_STRENGTH) |
+ - - - - - - - - - - - - - - - - - - - - - - - - +
|                                                   12 |
+ - - - - - - - - - - - - - - - - - - - - - - - - +
```

The table has 12 rows, and 12 values for column STUDENT_STRENGTH are available.

Now let’s see how many distinct values this column has.

```
SELECT  COUNT(DISTINCT STUDENT_STRENGTH) FROM
ENGINEERING_STUDENTS;
```

```
+ - - - - - - - - - - - - - - - -  - - - - - - - - - - - - - - +
| COUNT(DISTINCT STUDENT_STRENGTH) |
+ - - - - - - - - - - - - - - - -  - - - - - - - - - - - - - - +
|                                                                        6 |
+ - - - - - - - - - - - - - - - -  - - - - - - - - - - - - - - +
1 row in set (0.08 sec)
```

Chapter 4 Using Subqueries

Subqueries are one of the most interesting features of SQL that actually allows developers to work with lot of flexibility. When you use one SQL statement nested within another SQL statement it is called a sub query. A subquery is always enclosed within parentheses. The SQL server executes the subquery prior to the statement that contains it.

Now let’s have a look at the following two tables:

```
SELECT * FROM ENGINEERING_STUDENTS;
+ - - - - - - - -+ - - - - - - - - - - - + - - - - - - - - - - - - - - - - - +
| ENGG_ID | ENGG_NAME      | STUDENT_STRENGTH |
+ - - - - - - - -+ - - - - - - - - - - - + - - - - - - - - - - - - - - - - - +
|            1 | Electronics        |                         150 |
|            2 | Software           |                         250 |
|            4 | Mechanical         |                         150 |
|            5 | Biomedical         |                          72 |
|            6 | Instrumentation   |                         80 |
|            7 | Chemical           |                          75 |
|            8 | Civil              |                          60 |
|            9 | Electronics & Com |                        250 |
|           10 | Electrical         |                          60 |
|           11 | Genetic            |                        150 |
|           12 | Systems            |                        150 |
|           13 | Aerospace          |                        150 |
+ - - - - - - - -+ - - - - - - - - - - - + - - - - - - - - - - - - - - - - - +
12 rows in set (0.00 sec)
SELECT * FROM DEPT_DATA;
+ - - - - - - -+ - - - - - - - - - - - - - + - - - - - - - - - - -+ - - - - - - - - +
| Dept_ID | HOD                      | NO_OF_Prof     | ENGG_ID |
+ - - - - - - -+ - - - - - - - - - - - - - + - - - - - - - - - - - -+ - - - - - - - - +
|        100 | Miley Andrews        | 7                       |                1 |
|        101 | Alex Dawson          | 6                       |                2 |
|        103 | Anne Joseph          | 5                       |                4 |
|       104 | Sophia Williams       | 8                       |                5 |
|       105 | Olive Brown           | 4                       |                6 |
|       106 | Joshua Taylor         | 6                       |               7 |
|      107 | Ethan Thomas          | 5                       |               8 |
|      108 | Michael Anderson | 8                         |              9 |
|      109 | Martin Jones          | 5                        |             10 |
+ - - - - - - -+ - - - - - - - - - - - - - + - - - - - - - - - - - -+ - - - - - - - - +
9 rows in set (0.16 sec)
```

Now, let's say that we want to find out the student strength for the department that has minimum number of professors.

So for this to happen, we will first find out the department having minimum number of professors is:

```
SELECT MIN(NO_OF_PROF) FROM DEPT_DATA;
+ - - - - - - - - - - - - - - - - +
| MIN(NO_OF_PROF) |
+ - - - - - - - - - - - - - - - - +
| 4                               |
+ - - - - - - - - - - - - - - - - +
```

1 row in set (0.24 sec)

We, now nest this query in another query to get the ENGG_ID for this department.

```
SELECT ENGG_ID FROM DEPT_DATA WHERE NO_OF_PROF=
(SELECT MIN(NO_OF_PROF) FROM DEPT_DATA);
+ - - - - - - - - - +
| ENGG_ID  |
+ - - - - - - - - - +
|              6 |
+ - - - - - - - - - +
```

The above query already has a subquery. Now we will put this entire statement in another statement to finally get the result that we want.

```
SELECT * FROM ENGINEERING_STUDENTS WHERE ENGG_ID=
(SELECT ENGG_ID FROM DEP
T_DATA WHERE NO_OF_PROF=(SELECT MIN(NO_OF_PROF) FROM
DEPT_DATA));
+ - - - - - - - -+ - - - - - - - - - - - - + - - - - - - - - - - - - - - - - -+
| ENGG_ID | ENGG_NAME       | STUDENT_STRENGTH |
+ - - - - - - - -+ - - - - - - - - - - - - + - - - - - - - - - - - - - - - - -+
|              6 | Instrumentation |                                80 |
+ - - - - - - - -+ - - - - - - - - - - - - + - - - - - - - - - - - - - - - - -+
1 row in set (0.06 sec)
```

Let’s see how this query worked:

```
SELECT * FROM ENGINEERING_STUDENTS WHERE
ENGG_ID=(SELECT ENGG_ID FROM DEP

T_DATA WHERE NO_OF_PROF=(SELECT
MIN(NO_OF_PROF) FROM DEPT_DATA));
```

If you look at the DEPT_DATA table you will see that minimum number of professor are 4

```
SELECT * FROM ENGINEERING_STUDENTS WHERE
ENGG_ID=(SELECT ENGG_ID FROM DEP

T_DATA WHERE NO_OF_PROF= 4);
```

The ENGG_ID for this department is 6. So, the statement is further simplified as:

```
SELECT * FROM ENGINEERING_STUDENTS WHERE
ENGG_ID= 6;
```

Now, if you look at ENGINEERING_STUDENTS table you will see that the Engineering department corresponding to ENGGID 6 is ‘Instrumentation’. For this we get the following result set from ENGINEERING_STUDENTS table:

```
+ - - - - - - - -+ - - - - - - - - - - - - + - - - - - - - - - - - - - - - - -+
| ENGG_ID | ENGG_NAME       | STUDENT_STRENGTH |
+ - - - - - - - -+ - - - - - - - - - - - - + - - - - - - - - - - - - - - - - -+
|              6 | Instrumentation |                              80 |
+ - - - - - - - -+ - - - - - - - - - - - - + - - - - - - - - - - - - - - - - -+
1 row in set (0.06 sec)
```

Now, for the same table try finding out the Engineering students details for the department that has maximum number of professors.

```
SELECT * FROM ENGINEERING_STUDENTS WHERE
ENGG_ID=(SELECT ENGG_ID FROM DEP

T_DATA WHERE NO_OF_PROF=(SELECT
MAX(NO_OF_PROF) FROM DEPT_DATA));
```

When you enter this query, you will encounter the following error:

```
ERROR 1242 (21000): Subquery returns more than 1 row
```

When we tried to find out information about department which has minimum number of professors we were able to do so because there was only one department that had least number of professors hence the value returned could be used as an expression however that is not the case with the maximum number of professors. If you have a look at the subquery SELECT ENGG_ID FROM DEPT_DATA WHERE NO_OF_PROF=(SELECT MAX(NO_OF_PROF) FROM DEPT_DATA), it would return two values:

```
SELECT ENGG_ID FROM DEPT_DATA WHERE NO_OF_PROF=
(SELECT MAX(NO_OF_PROF)
OM DEPT_DATA);
+ - - - - - - - - +
| ENGG_ID  |
+ - - - - - - - -+
|             5 |
|             9 |
+ - - - - - - - -+
```

2 rows in set (0.00 sec)

When we tried to find information about the minimum number of professors only one value was retrieved which could be equated to the expression but now we have two values and same operation on the result is not possible.

So, in this case we substitute WHERE clause in the outer main query by IN as shown below:

```
SELECT * FROM ENGINEERING_STUDENTS WHERE ENGG_ID IN
(SELECT ENGG_ID FROM
DEPT_DATA WHERE NO_OF_PROF=(SELECT MAX(NO_OF_PROF)
FROM DEPT_DATA));
+ - - - - - - - -+ - - - - - - - - - - - - - + - - - - - - - - - - - - - - - - - +
| ENGG_ID | ENGG_NAME        | STUDENT_STRENGTH |
+ - - - - - - - -+ - - - - - - - - - - - - - + - - - - - - - - - - - - - - - - - +
|             5 | Biomedical            |                              72 |
|             9 | Electronics & Com |                             250 |
+ - - - - - - - -+ - - - - - - - - - - - - - + - - - - - - - - - - - - - - - - - +
```

2 rows in set (0.60 sec)

Let’s see you this works:

```
SELECT MAX(NO_OF_PROF) FROM DEPT_DATA;
+ - - - - - - - - - - - - - - - - +
| MAX(NO_OF_PROF) |
+ - - - - - - - - - - - - - - - - +
| 8                            |
+ - - - - - - - - - - - - - - - - +
```

1 row in set (0.01 sec)

So, the statement can be simplified as:

```
SELECT * FROM ENGINEERING_STUDENTS WHERE
ENGG_ID IN (SELECT ENGG_ID FROM

DEPT_DATA WHERE NO_OF_PROF=(8));
```

This is further simplified to:

```
SELECT * FROM ENGINEERING_STUDENTS WHERE ENGG_ID IN
(5,9);
```

Hence , the following results:

```
+ - - - - - - - -+ - - - - - - - - - - - - - + - - - - - - - - - - - - - - - - - +
| ENGG_ID | ENGG_NAME        | STUDENT_STRENGTH |
+ - - - - - - - -+ - - - - - - - - - - - - - + - - - - - - - - - - - - - - - - - +
|             5 | Biomedical             |                              72 |
|             9 | Electronics & Com |                            250 |
```

Correlated vs Non correlated Subqueries

S ubqueries can be used in insert and select statements. Subqueries should return a scalar value if it makes use of WHERE clause or a value from the column if it is using IN or NOT IN clause. With this we now come to difference between correlated and no correlated sub queries. A subquery which depends upon the outer query and cannot execute on its own where as in case of non-correlated Subqueries both inner and outer queries are independent of each other.

```
SELECT * FROM DEPT_DATA d WHERE ENGG_ID IN (SELECT
ENGG_ID FROM ENGINEERI
NG_STUDENTS e WHERE e.ENGG_ID=d.ENGG_ID );
+ - - - - - - -+ - - - - - - - - - - - - - - -+ - - - - - - - - - - -+ - - - - - - -  +
| Dept_ID | HOD                         | NO_OF_Prof | ENGG_ID |
+ - - - - - - -+ - - - - - - - - - - - - - - -+ - - - - - - - - - - -+ - - - - - - -  +
|         100 | Miley Andrews       | 7                    |             1 |
|         101 | Alex Dawson          | 6                   |             2 |
|         103 | Anne Joseph          | 5                   |            4 |
|         104 | Sophia Williams     | 8                   |             5 |
|        105 | Olive Brown           | 4                   |             6 |
|        106 | Joshua Taylor        | 6                   |            7 |
|       107 | Ethan Thomas        | 5                   |            8 |
|      108 | Michael Anderson | 8                  |           9 |
|      109 | Martin Jones         | 5                  |          10 |
+ - - - - - - -+ - - - - - - - - - - - - - -+ - - - - - - - - - -+ - - - - - - -  +
```

rows in set (0.16 sec)

When the inner sub query references the outer main query, we call it correlated Subqueries. The inner subquery references the column name of the table that is in outer query.

In case of Non correlated subquery the inner subquery is independent of outer main query.

```
SELECT * FROM DEPT_DATA d WHERE ENGG_ID IN (SELECT
ENGG_ID FROM ENGINEERI
NG_STUDENTS );
+ - - - - - - -+ - - - - - - - - - - - - - + - - - - - - - - - - + - - - - - - - - +
| Dept_ID | HOD                       | NO_OF_Prof | ENGG_ID |
+ - - - - - - -+ - - - - - - - - - - - - - + - - - - - - - - - - + - - - - - - - - +
|        100 | Miley Andrews      | 7                  |             1 |
|       101 | Alex Dawson         | 6                  |             2 |
|      103 | Anne Joseph          | 5                  |             4 |
|      104 | Sophia Williams     | 8                  |             5 |
|      105 | Olive Brown          | 4                  |             6 |
|      106 | Joshua Taylor        | 6                  |            7 |
|     107 | Ethan Thomas        | 5                  |            8 |
|     108 | Michael Anderson | 8                  |            9 |
|     109 | Martin Jones         | 5                  |          10 |
+ - - - - - - -+ - - - - - - - - - - - - - + - - - - - - - - - - + - - - - - - - - +
```

9 rows in set (0.26 sec)

This is an example of a non-correlated subquery.

Before we end this chapter here are few things to keep in mind:

1. Whatever you want to achieve with the help of a subquery can also be accomplished with the help of JOINS.
2. In case of correlated subquery, outer query will get processed before the inner subquery.

Conclusion

This concludes the third book in the series! Thank you for purchasing SQL: Advanced Level SQL From The Ground Up. More fun and exciting exercises can be found in the next edition of the series. Keep an eye out for SQL: Elite Level SQL From The Ground Up. If you enjoyed this book, a positive review is always appreciated!

SQL

Elite Level SQL From The Ground Up

Introduction

Thank you for downloading SQL: Elite Level SQL From The Ground Up! This book is the fourth entry of the DIY SQL series. It is preceded by the books

- SQL: Beginner Level SQL From The Ground Up

-SQL: Intermediate Level SQL From The Ground Up

-SQL: Advanced Level SQL From The Ground Up

…and assumes that the user is familiar with the contents of that book, including SQL scripting, syntax, and terminology. The books should be taken in chronological order for optimal results. This book will cover high level SQL manipulation techniques, with code examples to match the concepts explained.

Thank you again for downloading this book! You have many choices available to you for furthering your SQL scripting knowledge. Thank you for selecting the DIY SQL series as your tool of choice!

Chapter 5. Joins

Inner Join

Let’s refresh our knowledge about Joins first. Have a look at the contents of DEPT_DATA table. The table data is shown below:

```
SELECT * from DEPT_DATA;
+ - - - - - - - + - - - - - - - - - - - - - - + - - - - - - - - - - - + - - - - - - - - - +
| Dept_ID | HOD                  | NO_OF_Prof | ENGG_ID |
+ - - - - - - - + - - - - - - - - - - - - - - + - - - - - - - - - - - + - - - - - - - - - +
|       100 | Miley Andrews      | 7               |           1 |
|       101 | Alex Dawson        | 6               |           2 |
|       103 | Anne Joseph        | 5               |           4 |
|       104 | Sophia Williams    | 8               |           5 |
|       105 | Olive Brown        | 4               |           6 |
|       106 | Joshua Taylor      | 6               |           7 |
|       107 | Ethan Thomas       | 5               |          8 |
|       108 | Michael Anderson | 8               |          9 |
|       109 | Martin Jones       | 5               |         10 |
+ - - - - - - - + - - - - - - - - - - - - - - + - - - - - - - - - - - + - - - - - - - - - +
9 rows in set (0.06 sec)
```

The table has a primary key DEPT_ID and a foreign key ENGG_ID which is related to the ENGINEERING_STUDENTS table.

Now let’s have a look at the contents of ENGINEERING_STUDENTS table:

```
SELECT * from ENGINEERING_STUDENTS;
+ - - - - - - - +- - -  -- -  - - - - - - - - + - - - - - - - - - - - - - - - - - +
| ENGG_ID | ENGG_NAME        | STUDENT_STRENGTH |
+ - - - - - - - + - - -  -- -  - - - - - - - - + - - - - - - - - - - - - - - - - - +
|            1 | Electronics            |                        150 |
|            2 | Software               |                        250 |
|            4 | Mechanical             |                        150 |
|            5 | Biomedical             |                          72 |
|            6 | Instrumentation      |                          80 |
|            7 | Chemical                 |                          75 |
|            8 | Civil                       |                          60 |
|            9 | Electronics & Com   |                        250 |
|          10 | Electrical               |                          60 |
|          11 | Genetic                  |                        150 |
|          12 | Systems                 |                        150 |
|          13 | Aerospace              |                        150 |
+ - - - - - - -+- - - - - -  - - - - - - - - + - - - - - - - - - - - - - - - - - +
12 rows in set (0.00 sec)
```

As you can see there are a total of 9 rows in DEPT_DATA table and 12 rows in ENGINEERING_STUDENTS table. There are obviously some values of ENGG_ID that are present in ENGINEERING_STUDENTS but not in DEPT_DATA. Out of 12 Engineering branches, 9 branches have well defined departments. So, when we join the two tables based on ENGG_ID and no other filtering condition we can expect all nine rows of DEPT_DATA to be part of the result set.

```
SELECT a.ENGG_ID, a.ENGG_NAME,b.DEPT_ID, b.HOD from
ENGINEERING_STUDENTS
a INNER JOIN DEPT_DATA b on a.ENGG_ID=b. ENGG_ID;
+ - - - - - - -+ - - - - - - - - - - - - - -+ - - - - - - -+ - - - - - - - - - - - - - +
| ENGG_ID | ENGG_NAME      | DEPT_ID | HOD                    |
+ - - - - - - -+ - - - - - - - - - - - - - -+ - - - - - - -+ - - - - - - - - - - - - - +
|            1 | Electronics           |        100 | Miley Andrews    |
|            2 | Software               |        101 | Alex Dawson       |
|           4 | Mechanical          |        103 | Anne Joseph        |
|          5 | Biomedical            |        104 | Sophia Williams    |
|         6 | Instrumentation   |        105 | Olive Brown          |
|        7 | Chemical              |       106 | Joshua Taylor        |
|        8 | Civil                      |       107 | Ethan Thomas        |
|       9 | Electronics & Com |       108 | Michael Anderson |
|     10 | Electrical               |       109 | Martin Jones         |
+ - - - - + - - - - - - - - - - - - - -+ - - - - - - -+ - - - - - - - - - - - - - +
9 rows in set (0.00 sec)
```

However, there would be times when you want the result data to display all the rows of both the tables so that you can see which rows are related and which are not. In such cases we can make use of outer join.

Outer join

Have a look at the query given below:

```
SELECT a.ENGG_ID,a.ENGG_NAME, b.HOD,b.DEPT_ID FROM
ENGINEERING_STUDENTS a
LEFT OUTER JOIN DEPT_DATA b ON a.ENGG_ID = b.ENGG_ID;
+ - - - - - - - + - - - - - - - - - - - - - - - + - - - - - - - - - - - - - + - - - - - - - +
| ENGG_ID | ENGG_NAME          | HOD                    | DEPT_ID |
+ - - - - - - - + - - - - - - - - - - - - - - - + - - - - - - - - - - - - - + - - - - - - - +
|           1 | Electronics             | Miley Andrews    |       100 |
|           2 | Software                 | Alex Dawson       |       101 |
|           4 | Mechanical             | Anne Joseph        |       103 |
|           5 | Biomedical             | Sophia Williams    |       104 |
|           6 | Instrumentation     | Olive Brown          |       105 |
|          7 | Chemical                 | Joshua Taylor       |       106 |
|          8 | Civil                        | Ethan Thomas       |       107 |
|         9 | Electronics & Com   | Michael Anderson |      108 |
|        10 | Electrical                | Martin Jones         |      109 |
|       11 | Genetic                    | NULL                     |    NULL |
|       12 | Systems                   | NULL                     |    NULL |
|     13 | Aerospace                | NULL                     |    NULL |
+ - - - - - - - + - - - - - - - - - - - - - - - + - - - - - - - - - - - - - + - - - - - - - +
12 rows in set (0.00 sec)
```

The above result set has 12 rows of data. As you can see, for ENGG_ID 11, 12 and 13 there is no data available in the DEPT_DATA table. Hence, for these three values of ENGG_ID, the corresponding data from DEPT_DATA is shown as NULL.

All twelve rows are displayed when we use Left outer join. Now, let’s see what happens if we use Right outer join.

```
SELECT a.ENGG_ID,a.ENGG_NAME, b.HOD,b.DEPT_ID FROM
ENGINEERING_STUDENTS a
RIGHT OUTER JOIN DEPT_DATA b ON a.ENGG_ID = b.ENGG_ID;
+ - - - - - - - + - - - - - - - - - - - - - + - - - - - - - - - - - - -  + - - - - - - - +
| ENGG_ID | ENGG_NAME        | HOD                    | DEPT_ID |
+ - - - - - - - + - - - - - - - - - - - - - + - - - - - - - - - - - - -  + - - - - - - - +
|           1 | Electronics           | Miley Andrews      |        100 |
|           2 | Software              | Alex Dawson         |        101 |
|           4 | Mechanical           | Anne Joseph         |        103 |
|           5 | Biomedical           | Sophia Williams     |        104 |
|           6 | Instrumentation    | Olive Brown          |        105 |
|           7 | Chemical              | Joshua Taylor        |        106 |
|           8 | Civil                     | Ethan Thomas       |        107 |
|           9 | Electronics & Com | Michael Anderson |        108 |
|         10 | Electrical              | Martin Jones         |        109 |
+ - - - - - - - + - - - - - - - - - - - - - + - - - - - - - - - - - - -  + - - - - - - - +
9 rows in set (0.00 sec)
```

As you can see only 9 rows of data is displayed if we use Right outer join.

Explanation

When you use a left outer join, the table on the left side of the table determines how many rows the result set will have. The table on the right provides result if a match is found or else NULL is displayed in the result set. The same way in right outer join the table on the right determines the number of rows in the result set.

Three way outer join

Sometimes, you may want to outer join one table with two more tables. Have a look at the example given below:

```
SELECT * FROM TEACHER_DATA;
+ - - - - - - - - - + - - - - - - - - - - - - + - - - - - - - - - - + - - - - - - - - - - - -+ - - - -
- - + - - - - - - - -+
| teacher_ID  | teacher_FN       | teacher_LN | teacher_phone | gender |
dept_ID |
+ - - - - - - - - - + - - - - - - - - - - - - + - - - - - - - - - - + - - - - - - - - - - - -+ - - - -
- - + - - - - - - - -+
|                 1 | Deandre           | Becker         |    2025550148 | M          |
101 |
|                 2 | DeonGoodman | Swanson      |    2025550111 | M          |
104 |
|                 3 | Augustin           | Norman       |    2025550190 | M          |
100 |
|                 4 | Alberto             | Hammond    |    2025550162 | M          |
109 |
|                5 | Lana                  | Flowers        |    2025550153 | F
|     108 |
|                6 | Gabriell             | Cobb            |    2025550126 | F            |
107 |
|                7 | Alizye               | Blake            |                 NULL | F
|     103 |
|                8 | Jayce                 | Conner         |                NULL | M
|     100 |
|               9 | Jayce                 | Conner         |                NULL | M
|     100 |
|             10 | Freddie             | Sparks          |                NULL | M
|     109 |
|           11 | Geoffry              | Abbott         |                NULL | M
|     108 |
|           12 | Coby                  | Morton         |                NULL | M
|     107 |
|           13 | Jevon                | Park             |                NULL | M
|     106 |
|            14 | Aidan                   | Marsh            |                NULL | M
|     100 |
|           15 | Madison              | Quinn            |                NULL | M
|     105 |
|           16 | Jade                     | Moody          |              NULL | F
|     106 |
|          17 | Ebony                   | Malone         |              NULL | F
|     104 |
|          18 | Katelyn               | Webster        |              NULL | F
|     105 |
|         19 | Dorothy               | Patton            |              NULL | F
|     101 |
+ - - - - - - - - - + - - - - - - - - - - - + - - - - - - - - - - + - - - - - - - - - - - -+ - - - -
- - + - - - - - - - -+
19 rows in set (0.11 sec)
```

The table has 19 rows of data. Now, we join TEACHER_DATA with ENGINEERING_STUDENTS and DEPT_DATA so that the result set shows which teachers come under which HOD and Department.

```
SELECT a.ENGG_ID,a.ENGG_NAME, b.HOD,b.DEPT_ID,
c.TEACHER_FN FROM ENGINEER
ING_STUDENTS a LEFT OUTER JOIN DEPT_DATA b ON a.ENGG_ID
= b.ENGG_ID LEFT OUTER J
OIN TEACHER_DATA c ON b.DEPT_ID=c.DEPT_ID;
+ - - - - - - - + - - - - - - - - - - - - - + - - - - - - - - - - - - - -+ - - - - - - + - - - - - - -
- - - - - -+
| ENGG_ID | ENGG_NAME       | HOD                      | DEPT_ID|
TEACHER_FN     |
+ - - - - - - - + - - - - - - - - - - - - - + - - - - - - - - - - - - - -+ - - - - - - + - - - - - - -
- - - - - -+
|            1 | Electronics            | Miley Andrews      |     100 |
Augustin             |
|            1 | Electronics            | Miley Andrews      |     100 |
Jayce                    |
|           1 | Electronics          | Miley Andrews     |     100 |
Jayce                 |
|           1 | Electronics          | Miley Andrews     |     100 |
Aidan                 |
|           2 | Software             | Alex Dawson       |     101 |
Deandre               |
|           2 | Software             | Alex Dawson       |     101 |
Dorothy               |
|           4 | Mechanical           | Anne Joseph       |     103 |
Alizye                |
|           5 | Biomedical           | Sophia Williams   |     104 | DeonGoodman
|
|           5 | Biomedical           | Sophia Williams   |     104 |
Ebony                 |
|           6 | Instrumentation      | Olive Brown       |     105 | Madison
|
|           6 | Instrumentation      | Olive Brown       |     105 | Katelyn
|
|           7 | Chemical             | Joshua Taylor     |     106 |
Jevon                 |
|           7 | Chemical             | Joshua Taylor     |     106 |
Jade                  |
|           8 | Civil                | Ethan Thomas      |     107 |
Gabriell              |
|           8 | Civil                | Ethan Thomas      |     107 |
Coby                  |
|           9 | Electronics & Com | Michael Anderson |     108 | Lana
|
|           9 | Electronics & Com | Michael Anderson |     108 | Geoffry
|
|          10 | Electrical           | Martin Jones      |     109 | Alberto
|
|          10 | Electrical           | Martin Jones      |     109 | Freddie
|
|          11 | Genetic              | NULL              |    NULL |
NULL                  |
|          12 | Systems              | NULL              |    NULL |
NULL                  |
|        13 | Aerospace              | NULL                         |    NULL |
NULL                   |
+ - - - - - - - + - - - - - - - - - - - - - + - - - - - - - - - - - - - -+ - - - - - - + - - - - - - -
- - - - - - -+
```

22 rows in set (0.00 sec)

Chapter 6 Conditions

Conditions play a major role when one situation can have different outputs and for every output there is a need to take different actions. Conditional logic plays a very important role in adding flexibility in programming.

Searched Case Expression

We have learnt basics of conditional logic before. In this chapter we will try to explore the topic a little more. We will now see how searched cased expression works. The syntax for this is as follows:

```
CASE
   WHEN CASE1 THEN EXPRESSION1
   WHEN CASE2 THEN EXPRESSION2
   WHEN CASE3 THEN EXPRESSION3
   [ELSE DEFAULT_EXPRESSION]
END
```

In the above syntax, expressions CASE1, CASE2…etc stand for various conditions and the terms - EXPRESSION1, EXPRESSION2…etc stand for the action that we want to take for the respective case. If any case is true the respective expression will be carried out or else default expression will be executed.

```
mysql> SELECT e.ENGG_NAME,STUDENT_STRENGTH,
    ->   CASE
    ->   WHEN e.STUDENT_STRENGTH > 150 THEN 'too many students'
    ->   WHEN e.STUDENT_STRENGTH = 150 THEN 'right student
```

```
strength'
    ->   ELSE 'student strength too less'
    ->   END STUDENT_STATUS
    -> FROM ENGINEERING_STUDENTS e;
+ - - - - - - - - - - - - - - -+ - - - - - - - - - - - - - - - + - - - - - - - - - - - - - - - - - - -
-+
| ENGG_NAME        | STUDENT_STRENGTH |
STUDENT_STATUS           |
+ - - - - - - - - - - - - - - -+ - - - - - - - - - - - - - - - + - - - - - - - - - - - - - - - - - - -
-+
| Electronics             |                              150 | right student strength      |
| Software                |                              250 | too many students           |
| Mechanical              |                              150 | right student strength      |
| Biomedical              |                               72 | student strength too less |
| Instrumentation     |                              80 | student strength too less |
| Chemical                |                               75 | student strength too less |
| Civil                       |                               60 | student strength too less |
| Electronics & Com |                              250 | too many students           |
| Electrical                |                               60 | student strength too less |
| Genetic                  |                              150 | right student strength      |
| Systems                  |                              150 | right student strength      |
| Aerospace              |                              150 | right student strength      |
+ - - - - - - - - - - - - - - -+ - - - - - - - - - - - - - - - + - - - - - - - - - - - - - - - - - - -
-+
12 rows in set (0.00 sec)
```

The above query just provides a simple expression, we are just asking to print a statement for each case but you can also execute a subquery for each case. So, let’s suppose that for the student strength greater than 150, I just want to know the name of the HODs. When, the student strength is 150, I want to know the name of the HOD and number of professors in that department. For student strength less than 150 you just want the quote 'student strength too less' to be printed. The code and result set for this would be as follows:

```
mysql> SELECT e.ENGG_NAME,STUDENT_STRENGTH,
    ->   CASE
    ->   WHEN e.STUDENT_STRENGTH > 150 THEN (SELECT d.HOD
FROM DEPT_DATA d WHERE
d.ENGG_ID=e.ENGG_ID)
    ->   WHEN e.STUDENT_STRENGTH = 150 THEN (SELECT
CONCAT(d.HOD,' AND NUMBER OF
PROFESSORS IN DEPT ARE: ',d.NO_OF_Prof)FROM DEPT_DATA d
WHERE d.ENGG_ID=e.ENGG_
ID)
    ->   ELSE 'student strength too less'
    ->   END STUDENT_STATUS
    -> FROM ENGINEERING_STUDENTS e;
+ - - - - - - - - - - - - - - + - - - - - - - - - - - - - - - - -+ - - - - - - - - - - - - - - - - - -
- - - - - - - - - - - - - - - - - - - - - - -


- - - - - - - - - - - - +
| ENGG_NAME       | STUDENT_STRENGTH | STUDENT_STATUS
              |
+ - - - - - - - - - - - - - - + - - - - - - - - - - - - - - - - -+ - - - - - - - - - - - - - - - - - -
- - - - - - - - - - - - - - - - - - - - - - -
- - - - - - - - - - - - +
| Electronics             |                            150 | Miley Andrews AND
NUMBER OF PROFESSORS
IN DEPT ARE: 7 |
| Software                |                            250 | Alex Dawson
                          |
| Mechanical              |                            150 | Anne Joseph AND NUMBER
OF PROFESSORS IN
DEPT ARE: 5    |
| Biomedical              |                             72 | student strength too less
                          |
| Instrumentation    |                             80 | student strength too less
                          |
| Chemical                |                             75 | student strength too less
                          |
| Civil                   |                             60 | student strength too less
```

```
                    |
| Electronics & Com |                         250 | Michael Anderson
                    |
| Electrical         |                         60 | student strength too less
                    |
| Genetic            |                        150 | NULL
                     |
| Systems            |                        150 | NULL
                     |
| Aerospace          |                       150 | NULL
                     |
+ - - - - - - - - - - - - - - + - - - - - - - -  - - - - - - - -+ - - - - - - - - - - - - - - - - - -
- - - - - - - - - - - - - - - - - - - - - - -

- - - - - - - - - - - - - +
```

12 rows in set (0.09 sec)

Now, suppose we want to know how many professors are working in each department.

We first look at details of the professors in the teacher_data table:

```
SELECT * from TEACHER_DATA;
+ - - - - - - - - - -+ - - - - - - - - - - - - -+ - - - - - - - - - - - + - - - - - - - - - - - - + - - - - - +
- - - - - - - -+
| teacher_ID | teacher_FN  | teacher_LN | teacher_phone | gender | dept_ID |
+ - - - - - - - - - -+ - - - - - - - - - - - - -+ - - - - - - - - - - - + - - - - - - - - - - - - + - - - - - +
- - - - - - - -+
|              1 | Deandre         | Becker        |    2025550148 | M      |
101 |
|              2 | DeonGoodman | Swanson      |    2025550111 | M      |       104
|
|              3 | Augustin         | Norman       |    2025550190 | M      |
100 |
|             4 | Alberto             | Hammond    |    2025550162 | M      |
109 |
|             5 | Lana                 | Flowers       |    2025550153 | F        |
108 |
|           6 | Gabriell         | Cobb        |  2025550126 | F      |
107 |
|           7 | Alizye           | Blake       |        NULL | F      |
103 |
|           8 | Jayce            | Conner      |        NULL | M      |
100 |
|           9 | Jayce            | Conner      |        NULL | M      |
100 |
|          10 | Freddie          | Sparks      |        NULL | M      |
109 |
|          11 | Geoffry          | Abbott      |        NULL | M      |
108 |
|          12 | Coby             | Morton      |        NULL | M      |
107 |
|          13 | Jevon            | Park        |        NULL | M      |
106 |
|          14 | Aidan            | Marsh       |        NULL | M      |
100 |
|          15 | Madison          | Quinn       |        NULL | M      |
105 |
|          16 | Jade             | Moody       |        NULL | F      |
106 |
|          17 | Ebony            | Malone      |        NULL | F      |
104 |
|          18 | Katelyn          | Webster     |        NULL | F      |
105 |
|          19 | Dorothy          | Patton      |        NULL | F      |
101 |
+---------+------------+----------+-----------+----+
-------+
19 rows in set (0.00 sec)
```

Example : using sum() function

This table has 19 rows. It is very short, but has been created only for demo purposes. It may seem to be easy for you to count how many professors are present in each department but in real time scenario this data can be much much more than you think and manual calculations may not be possible. However, the number of professors can be calculated easily with SQL queries. I have already demonstrated how to use the function count(). What I am going to reveal now is how to use the “CASE” search to count and display the result set in multiple columns. In the following example, in every case the server counts number of professors in one department and displays results in individual columns.

```
mysql> SELECT
   -> SUM(CASE WHEN DEPT_ID = 100 THEN 1 ELSE 0 END) P100,
   -> SUM(CASE WHEN DEPT_ID = 101 THEN 1 ELSE 0 END) P101,
   -> SUM(CASE WHEN DEPT_ID = 103 THEN 1 ELSE 0 END) P103,
   -> SUM(CASE WHEN DEPT_ID = 104 THEN 1 ELSE 0 END) P104,
   -> SUM(CASE WHEN DEPT_ID = 105 THEN 1 ELSE 0 END) P105,
   -> SUM(CASE WHEN DEPT_ID = 106 THEN 1 ELSE 0 END) P106,
   -> SUM(CASE WHEN DEPT_ID = 107 THEN 1 ELSE 0 END) P107,
   -> SUM(CASE WHEN DEPT_ID = 108 THEN 1 ELSE 0 END) P108,
   -> SUM(CASE WHEN DEPT_ID = 109 THEN 1 ELSE 0 END) P109
   -> FROM TEACHER_DATA;
+ - - - - + - - - - -+ - - - - + - - - - + - - - - + - - - - -+ - - - - + - - - - + - - - - -+
| P100 | P101 | P103 | P104 | P105 | P106 | P107 | P108 | P109 |
+ - - - - + - - - - -+ - - - - + - - - - + - - - - + - - - - -+ - - - - + - - - - + - - - - -+
|      4 |       2 |      1 |      2 |      2 |       2 |      2 |       2 |      2 |
+ - - - - + - - - - -+ - - - - + - - - - + - - - - + - - - - -+ - - - - + - - - - + - - - - -+
1 row in set (0.01 sec)
```

So, the above result set displays in separate columns how many teachers work for each department.

Chapter 7 : Transactions

So far I have just been focussing on how independent SQL queries work. However, here I will provide information on transaction. In real time scenario we will always get to see multiple SQL statements working together in a group. Also, you will always see that multiple users access the database at the same time. So, we will now try to understand what all we must ensure so that the multiple SQL statements can be executed simultaneously. Following are some important factors that are a must for multiuser databases.

Locking

Locking is a very important mechanism that a server must implement while handling multiple users. So, database can lock some portion when there is any work going on it and if there is any other user wishing to access(read or modify) that same portion at point then that user will have to wait till the time the ongoing work is completed. A database can lock the server in two ways.

In the first method the user will have to request and receive a write lock or read lock to modify or read data respectively. Multiple users can be allowed to read data simultaneously however only one write lock is provided for each table at a time. No read lock is allowed as long as the write lock is on. The issue with this approach is that in this case there are several read and write requests at the same time then that would result in long waiting periods. The second method is where users will have to request and receive a write lock from the server in order to make any modifications in the existing data. However, there is no lock required to query data. The database uses the process of versioning. In this method problem can arise if long queries are made to execute while data is under modification. MySQL can implement both the methods depending on the storage engine.

The server can also implement lock granularities to lock a resource. There are three levels or granularities out of which the server may apply any one lock.

The first type is the Table locks that allow multiple users to modify the same table simultaneously. The second type is page locks which allows users to modify on the same page of table simultaneously where page is a memory segment anywhere between 2 -16 KB in size. The third type row locks ensure that the only the row that is being accessed by an application is locked. MySQL implements page or row locks and the type of granularity implemented again depend on the type of server engine used.

Starting a Transaction

A transaction can be started in two ways. In the first method there is no need to explicitly start a transaction. When the transaction ends the server automatically begins a new transaction. In the second method unless you explicitly start the transaction the transaction does not begin. MySQL server follows the latter approach. Unless you specifically begin a transaction you will be in auto-commit mode. The MySQL transaction starts with the ‘BEGIN WORK’ statement and ends with COMMIT or ROLLBACK statement. All statements in between the beginning and the end are all part of the transaction. When the transaction completes successfully the COMMIT command is issued and changes made will be reflected in all the tables that were involved. In case there is any sort of failure encounter in the middle of the transaction the ROLLBACK command will be issued instead of the

COMMIT command and all the changes made will revert back to what the state was before the transaction started. A session is said to be in AUTOCOMMIT if the AUTOCOMMIT is set to 1. In this case each SQL statement is a complete transaction in itself and is committed by default. In MySQL AUTOCOMMIT is set to 1 so it considers every SQL statement as its own transaction. However, if you give the following command:

```
SET AUTOCOMMIT =0
```

then all statements before the COMMIT statement are considered to be part of one transaction. So, if there is no failure before the COMMIT statement then all statements will be executed and changes will be reflected in the database or else the transaction will roll back.

Properties of Transaction

Transactions have four properties also known by the acronym ACID.

A stands for Atomicity which means that all the statements within the transaction will either be completed successfully or else the transaction will abort wherever it encounters a failure and revert or ROLL back to its former state.

C stands for Consistency which means that the database is able to properly change the state after the transaction is successfully committed.

I stands for isolation which means that all statements execute independent of each other.

D stands for durability which means that in case of s system failure the effect of the committed transaction will persist.

It is possible to use transactions directly in MySQL however in order to ensure safe and guaranteed transaction you need to create tables in a special way. The most popular way to do this is by using InnoDB. For this it is required that your version of MySQL supports for InnoDB. If you MySQL version does not support InnoDB then you will have to download MySQL – Max Binary Distribution for your operating system. If your version of MySQL supports InnoDB then you just need to add TYPE =InnoDB definition to the table creation statement. If Your MySQL version supports GEMINI or BDB then these table types can also be used.

Chapter 8: Indexes and Constraints

Whenever the data is inserted in a table in the database, the server simply places the data in the next available location within the file for that table. The data is not stored in any particular numeric or alphabetical order as you are likely to think. So, if you plan to retrieve data using SELECT statement then the server will carry out a table scan which means that it which check each row to check content and if a match is found the row will be added to the result set.

This type of a search may seem to be simple with the tables mentioned in the book because we have been dealing with tables having upto 20 rows of data. However, databases of various institutions like universities, stock exchange, banks, hospitals etc would have millions of rows of data and retrieving result in this manner can cause delay. In such cases indexes can help you out.

While reading a book, you would refer to the index for important information. So, basically index is used to find a particular piece of information. In books, the index is available at the end and lists the topics in an alphabetical order. In the same manner, the index in case of database keeps information sorted and organized. An Index is also a table however it not contain all the data of the entity instead it will contain only the columns that would be required to locate information.

Now, let’s try to create an index.

Creating an Index

Now, let’s again have a look at the ENGGINEERING_STUDENTS table.

```
mysql> select * from ENGINEERING_STUDENTS;
+ - - - - - - - + - - - - - - - - - - - - - + - - - - - - - - - - - - - - - - - -+
| ENGG_ID | ENGG_NAME        | STUDENT_STRENGTH |
+ - - - - - - - + - - - - - - - - - - - - - + - - - - - - - - - - - - - - - - - -+
|            1 | Electronics            |                              150 |
|            2 | Software                |                              250 |
|            4 | Mechanical           |                             150 |
|            5 | Biomedical            |                                72 |
|            6 | Instrumentation   |                            80 |
|           7 | Chemical               |                                75 |
|           8 | Civil                       |                                 60 |
|           9 | Electronics & Com |                             250 |
|         10 | Electrical              |                                60 |
|        11 | Genetic                  |                             150 |
|        12 | Systems                 |                             150 |
|        13 | Aerospace             |                            150 |
+ - - - - - - - + - - - - - - - - - - - - - + - - - - - - - - - - - - - - - - - -+
12 rows in set (0.09 sec)
```

Now we create a index for this table by giving the following command:

```
mysql> ALTER TABLE ENGINEERING_STUDENTS ADD INDEX ENGG_STU_INDEX(ENGG_NAME);

Query OK, 12 rows affected (0.28 sec)

Records: 12  Duplicates: 0  Warnings: 0
```

The above statement creates an Index by the name ENGG_STU_INDEX.

So, the optimizer can decide what is more beneficial, if there are just a few rows in the table then there is no need to go for the index. If there are more than one index for a table then the optimizer must take a decision about when index can be better for executing the query.

Now let’s create another index.

```
mysql> ALTER TABLE ENGINEERING_STUDENTS ADD
INDEX ENGG_STU_INDEX2(ENGG_ID);

Query OK, 12 rows affected (0.22 sec)

Records: 12  Duplicates: 0  Warnings: 0
```

Now, you can check how many indexes exist for a table.

You can view all the index that exist for a table:

```
mysql> show index from ENGINEERING_STUDENTS;
+ - - - - - - - - - - - - - - - - - - - + - - - - - - - - - -+ - - - - - - - - - - - -+- - - - - - -
- - - - -+ - - - - - - - - - -
-  - + - - - - - - - -+ - - - - - - - - -+ - - -- - - - - - -+  - - - - + - - - -+ - - - - - - - - -
+ - - - - - - - - -+

| Table                    | Non_unique | Key_name       | Seq_in_index |
Column_na
me | Collation | Cardinality | Sub_part | Packed | Null | Index_type | Comment
|

+ - - - - - - - - - - - - - - - - - - - + - - - - - - - - - -+ - - - - - - - - - - - -+- - - - - - -
- - - - -+ - - - - - - - - - -
-  - + - - - - - - - -+ - - - - - - - - -+ - - -- - - - - - -+  - - - - + - - - -+ - - - - - - - - -
+ - - - - - - - - -+

| ENGINEERING_STUDENTS |                  0 | PRIMARY
|                1 | ENGG_ID
     | A             |                2 |          NULL | NULL   |     | BTREE     |
|
| ENGINEERING_STUDENTS |                1 | ENGG_STU_INDEX
|              1 | ENGG_NAME
    | A            |              2 |          NULL | NULL   |     | BTREE     |
|

| ENGINEERING_STUDENTS |               1 | ENGG_STU_INDEX2
|              1 | ENGG_ID
   | A            |               2 |          NULL | NULL   |     | BTREE     |
|

+ - - - - - - - - - - - - - - - - - - + - - - - - - - - - -+ - - - - - - - - - - - -+- - - - - - -
- - - - -+ - - - - - - - - - - -
-  - + - - - - - - - -+ - - - - - - - - -+ - --- - - - - -+  - - - - + - - - -+ - - - - - - - - -
+ - - - - - - - - -+
3 rows in set (0.00 sec)
```

Now, if you look at the result set obtained above, there are three indexes one on the ENGG_ID with the name PRIMARY. Second, on the ENGG_NAME with the name ENGG_STU_INDEX and third on ENGG_ID again with the name ENGG_STU_INDEX2.

We created two indexes but our database shows three indexes in the result set. This is because at time when we created ENGINEERING_STUDENT table we included one constraint stating that ENGG_ID will be the primary key for the table. At the time the table was created the server created an index on the primary key column.

Dropping an Index

Dropping an index is as simple as creating one.

```
mysql> ALTER TABLE ENGINEERING_STUDENTS DROP INDEX
ENGG_STU_INDEX2;
Query OK, 12 rows affected (0.19 sec)
Records: 12  Duplicates: 0  Warnings: 0
```

So, now if you look for the indexes for ENGINEERING_STUDENTS , you will find ENGG_STU_INDEX2 missing.

```
mysql> show index from ENGINEERING_STUDENTS;
+ - - - - - - - - - - - - - - - - - - - + - - - - - - - - - -+ - - - - - - - - - - - -+- - - - - - -
- - - - -+ - - - - - - - - - -
-  - + - - - - - - - -+ - - - - - - - - -+ - - -- - - - - - -+  - - - - + - - - -+ - - - - - - - - -
+ - - - - - - - - -+
| Table                                | Non_unique | Key_name       | Seq_in_index |
Column_nam
e | Collation | Cardinality | Sub_part | Packed | Null | Index_type | Comment |
+ - - - - - - - - - - - - - - - - - - - + - - - - - - - - - -+ - - - - - - - - - - - -+- - - - - - -
- - - - -+ - - - - - - - - - -
-  - + - - - - - - - -+ - - - - - - - - -+ - - -- - - - - - -+  - - - - + - - - -+ - - - - - - - - -
+ - - - - - - - - -+
| ENGINEERING_STUDENTS |                  0 | PRIMARY
|                      1 | ENGG_ID
   | A               |                2 |           NULL | NULL   |       | BTREE
|                 |

| ENGINEERING_STUDENTS |                  1 | ENGG_STU_INDEX
|                1 | ENGG_NAME
   | A               |                2 |            NULL | NULL   |       | BTREE
|                |

+ - - - - - - - - - - - - - - - - - - - + - - - - - - - - - -+ - - - - - - - - - - - -+- - - - - - -
- - - - -+ - - - - - - - - - -
-  - + - - - - - - - -+ - - - - - - - - -+ - - -- - - - - - -+  - - - - + - - - -+ - - - - - - - - -
+ - - - - - - - - -+
2 rows in set (0.00 sec)
```

Unique Index

A unique index as the name suggests does not allow duplicate values to be entered in the indexed column. So, whenever you try to make changes in the indexed column the database, the server will check the unique index to see if the index already contains that value or not. If the same vaule already exists then you will receive and error if you try to enter same value.

```
mysql> SELECT * from TEACHER_DATA;
+ - - - - - - - - - - -+ - - - - - - - - - - - - - - - + - - - - - - - - - - - -+ - - - - - - - - - - - - - -+ - - - -
- - + - - - - - - - - - - +
| teacher_ID | teacher_FN          | teacher_LN | teacher_phone | gender |
dept_ID |
+ - - - - - - - - - - -+ - - - - - - - - - - - - - - - + - - - - - - - - - - - -+ - - - - - - - - - - - - - -+ - - - -
- - + - - - - - - - - - - +
|                 1 | Deandre              | Becker         |    2025550148 | M
|          101 |
|                 2 | DeonGoodman    | Swanson      |    2025550111 | M
|          104 |
|                 3 | Augustin             | Norman       |    2025550190 | M
|         100 |
|                 4 | Alberto              | Hammond    |    2025550162 | M
|         109 |
|                 5 | Lana                   | Flowers        |    2025550153 | F
|        108 |
|                6 | Gabriell              | Cobb            |    2025550126 | F
|        107 |
|               7 | Alizye                  | Blake            |                  NULL |
F            |        103 |
|               8 | Jayce                   | Conner         |                NULL | M
|     100 |
|               9 | Jayce                   | Conner         |                 NULL | M
|     100 |
|             10 | Freddie                | Sparks          |                NULL | M
|     109 |
|             11 | Geoffry                | Abbott         |                NULL | M
|     108 |
|            12 | Coby                    | Morton         |                NULL | M
|     107 |
|            13 | Jevon                   | Park             |                 NULL |
M           |     106 |
|           14 | Aidan                    | Marsh          |                NULL |
M            |     100 |
|        15 | Madison              | Quinn         |               NULL | M
|    105 |
|        16 | Jade                 | Moody         |               NULL | F
|    106 |
|        17 | Ebony               | Malone        |               NULL | F
|    104 |
|       18 | Katelyn              | Webster       |               NULL | F
|    105 |
|       19 | Dorothy              | Patton         |              NULL | F
|    101 |
+ - - - - - - - - -+ - - - - - - - - - - - - + - - - - - - - - - -+ - - - - - - - - - - - -+ - - - -
- - + - - - - - - - - +
```

For this table I have created a unique index by the name TEACHER_INDEX for column TEACHER_PHONE.

```
mysql> ALTER TABLE TEACHER_DATA ADD UNIQUE
TEACHER_INDEX(TEACHER_PHONE);

Query OK, 19 rows affected (0.23 sec)

Records: 19  Duplicates: 0  Warnings: 0
```

Now if you try to enter a phone number value that already exists, you will get an error

```
mysql> INSERT into TEACHER_DATA(teacher_FN,
teacher_LN, teacher_phone,gender,dep

t_ID) VALUES('Gabriella', 'Watson','2025550126','F','100' );

ERROR 1582 (23000): Duplicate entry '2025550126' for key
'TEACHER_INDEX'
```

There is no point in creating a unique index for primary key as the server ensures that only unique values are added to primary column **s**

Multicolumn Indexes

Multicolumn indexes allow you to create index for more than one column. For example, generally the first name, last name and the middle name is stored in separate columns however, while searching for a any information based on any of these three values then using a index for all three columns will be more beneficial.

Constraints

As the name suggests, constraint stands for restriction. Constraints are classified as: Primary constraint, foreign key constraint, unique constraint and check constraint. Primary key constraint is used for defining the primary key for a table; this constraint helps in ensuring that one is able to identify data easily with the help of unique columns. Foreign key constraints are applied to columns that are also present in another table as primary key. Foreign key helps in relating tables and also restrict allowable values in another table. Unique constraints help in ensuring that some columns of a table have unique values and primary key constraint is also a type of unique constraint. Check constraints are used to restrict the allowable inputs for a column. In addition to all this there are two more types of constraints: (1) NOT NULL which does not allow the column to have a NULL value and (2) DEFAULT constraint that allows the column to contain a value (can be NULL also) if not value is supplied.

If you remember how we created some tables before, you would get a better understanding of constraints. For example, the create statement for TEACHER_DATA is as shown below:

```
CREATE TABLE TEACHER_DATA(

teacher_ID bigint NOT NULL AUTO_INCREMENT,

teacher_FN varchar(35) NOT NULL,

teacher_LN varchar(35) NOT NULL,
teacher_phone int(10),

gender ENUM('M','F'),

dept_ID bigint,

PRIMARY KEY (teacher_ID),

FOREIGN KEY (dept_ID) REFERENCES dept_Data(dept_ID));
```

You can also create the table without constraint and implement them later using ALTER statement.

```
ALTER TABLE TEACHER_DATA

ADD CONSTRAINT PRIMARY KEY (teacher_ID);
```

OR

```
ALTER TABLE TEACHER_DATA

ADD CONSTRAINT FOREIGN KEY (dept_ID) REFERENCES dept_Data(dept_ID);
```

For primary key, foreign key and unique key constraints MySQL creates a new index.

Chapter 9: Views

A view is a SQL statement that is stored with a name. In very simple words it is a way of querying data and does not clutter your disk space. It is generally structured in such a manner that most users would not even come to know that they are seeing a view and not the actual table. This can be very beneficial in many ways. Views also help in restricting the access to data. Views are great for generating reports. A view is created by assigning a name to a SELECT query. It is stored so that others can use is as and when required. It would give and impression as if the users are interacting with the table directly.

Creating a view

The syntax for creating a view is as follows:

```
CREATE VIEW VIEW_NAME AS

SELECT COL1, COL2…

FROM TABLE_NAME

WHERE [CONDITION];
```

The SELECT statement can have multiple table names.

Suppose we need to find out who is the HOD for each department. For that we execute the following query:

```
SELECT a.HOD, b.ENGG_NAME FROM DEPT_DATA a,
ENGINEERING_STUDENTS b WHERE
a.ENGG_ID = b.ENGG_ID;
```

The result set for the above mentioned query will be as follows:

```
+ - - - - - - - - - - - - - - -+ - - - - -  - - - - - - - - +
| HOD                    | ENGG_NAME        |
+ - - - - - - - - - - - - - - -+ - - - - -  - - - - - - - - +
| Miley Andrews      | Electronics             |
| Alex Dawson          | Software                 |
| Anne Joseph          | Mechanical             |
| Sophia Williams     | Biomedical              |
| Olive Brown           | Instrumentation     |
| Joshua Taylor        | Chemical                 |
| Ethan Thomas       | Civil                          |
| Michael Anderson | Electronics & Com |
| Martin Jones         | Electrical                |
+ - - - - - - - - - - - - - - -+ - - - - -  - - - - - - - - +
9 rows in set (0.06 sec)
```

In order to have HOD names sorted in alphabetical order we can go for ORDER BY clause.

```
mysql> select a.HOD, b.ENGG_NAME FROM DEPT_DATA a,
ENGINEERING_STUDENTS b WHERE
a.ENGG_ID = b.ENGG_ID ORDER BY HOD;
+ - - - - - - - - - - - - - - -+ - - - - -  - - - - - - - - +
| HOD                    | ENGG_NAME        |
+ - - - - - - - - - - - - - - -+ - - - - -  - - - - - - - - +
| Alex Dawson           | Software                 |
| Anne Joseph           | Mechanical             |
| Ethan Thomas        | Civil                          |
| Joshua Taylor         | Chemical                 |
| Martin Jones          | Electrical                |
| Michael Anderson | Electronics & Com  |
| Miley Andrews      | Electronics             |
| Olive Brown           | Instrumentation     |
| Sophia Williams      | Biomedical            |
+ - - - - - - - - - - - - - - -+ - - - - -  - - - - - - - - +
rows in set (0.03 sec)
```

Now let’s convert this query into a view. The SQL query for the same will be as follows:

```
mysql> CREATE VIEW DEPT_HOD AS select a.HOD,
b.ENGG_NAME FROM DEPT_DATA a,ENGINE

ERING_STUDENTS b WHERE a.ENGG_ID = b.ENGG_ID
ORDER BY HOD;

Query OK, 0 rows affected (0.06 sec)
```

Now, let’s see the content of this view again

```
mysql> SELECT * FROM DEPT_HOD;
+ - - - - - - - - - - - - - - -+ - - - - -  - - - - - - - - +
| HOD                        | ENGG_NAME        |
+ - - - - - - - - - - - - - - -+ - - - - -  - - - - - - - - +
| Alex Dawson           | Software               |
| Anne Joseph           | Mechanical           |
| Ethan Thomas         | Civil                       |
| Joshua Taylor         | Chemical               |
| Martin Jones          | Electrical              |
| Michael Anderson | Electronics & Com  |
| Miley Andrews       | Electronics             |
| Olive Brown           | Instrumentation     |
| Sophia Williams      | Biomedical             |
+ - - - - - - - - - - - - - - -+ - - - - -  - - - - - - - - +
9 rows in set (0.05 sec)
```

Updating views

When the users see data via views, they may at times need to modify the information. MySQL allows you to modify data provided certain conditions are met. Let’s have a look at what these conditions actually are.

1. Aggregate functions such as max(),min() etc are not used.
2. The SELECT clause does not employ GROUP BY or HAVING clause.
3. There are no Subqueries present in the SELECT or WHERE clause

4. The SELECT clause does not employ UNION, UNION ALL or DISTINCT.
5. The FROM clause makes use of at least one table or view that can be updated.
6. The FROM clause uses only INNER JION if there are more than one table or view involved.

Suppose the HOD for the ‘Software’ department has changed and the user seeing the view wants to make the changes. The user may not be aware of the technical logic behind view and tables he may just want to make the necessary changes in what he sees. He can therefore make changes to the view if the view clears all the criteria mentioned above.

```
mysql> UPDATE DEPT_HOD SET HOD = 'Melina
Jones' WHERE HOD = 'Alex Dawson' and EN
GG_NAME='Software' ;
Query OK, 1 row affected (0.41 sec)
```

```
Rows matched: 1  Changed: 1  Warnings: 0
+ - - - - - - - - - - - - - - -+ - - - - -  - - - - - - - - +
| HOD                       | ENGG_NAME        |
+ - - - - - - - - - - - - - - -+ - - - - -  - - - - - - - - +
| Anne Joseph          | Mechanical           |
| Ethan Thomas        | Civil                       |
| Joshua Taylor        | Chemical               |
| Martin Jones          | Electrical              |
| Melina Jones          | Software               |
| Michael Anderson  | Electronics & Com |
| Miley Andrews       | Electronics             |
| Olive Brown           | Instrumentation     |
| Sophia Williams      | Biomedical             |
+ - - - - - - - - - - - - - - -+ - - - - -  - - - - - - - - +
9 rows in set (0.00 sec)
```

Since the HOD names are alphabetically sorted the updated value now appears at row number 5.

Now, let’s check if the information has been updated in DEPT_DATA table as well:

```
SELECT * from DEPT_DATA;
+ - - - - - - -+ - - - - - - - - - - - - - + - - - - - - - -  - -+ - - - - - - - -+
| Dept_ID | HOD                    | NO_OF_Prof | ENGG_ID |
+ - - - - - - -+ - - - - - - - - - - - - - + - - - - - - - -  - -+ - - - - - - - -+
|        100 | Miley Andrews      | 7                  |            1 |
|        101 | Melina Jones       | 6                  |            2 |
|        103 | Anne Joseph        | 5                  |            4 |
|       104 | Sophia Williams     | 8                  |            5 |
|       105 | Olive Brown         | 4                  |            6 |
|       106 | Joshua Taylor       | 6                  |            7 |
|       107 | Ethan Thomas        | 5                  |            8 |
|       108 | Michael Anderson | 8                     |            9 |
|       109 | Martin Jones        | 5                  |           10 |
+ - - - - - - -+ - - - - - - - - - - - - - + - - - - - - - -  - -+ - - - - - - - -+
9 rows in set (0.00 sec)
```

Hence, if you make changes to the view, the changes will also be reflected in the table of the database.

Rows can be inserted in a view provided all NOT NULL fields of the table are part of the view. You can also delete a row from the view and if you do so the row will also be deleted from the base table.

Chapter 10: Metadata

Metadata stands for data about data. Database server stores information about all the tables that it has in a database. Let’s have a look at what all information is stored and where it is kept. The information stored is referred to as data dictionary or system catalog and consists of:

Table name

Information regarding table storage

Column names

Column data types

Default values for columns

Details of constraints on columns

Index names, type and columns

Detail of storage engine

Index storage information and column sort order

Associated tables and columns for foreign keys

All the information mentioned above is stored so that system is able to retrieve and execute SQL queries easily. This also helps the server keep a check that there is no unauthorised modification in the database.

Information_schema

All the information in the information_schema database are views which can be queried. Suppose if you want to know what are the columns in the table ENGINEERING_STUDENTS.

So, I log into my Mysql account as shown below:

```
C:\Users\MYPC>mysql -u schooladmin -p

Enter password: *********

Welcome to the MySQL monitor.  Commands end with ; or \g.

Your MySQL connection id is 5

Server version: 6.0.0-alpha-community-nt-debug MySQL
Community Server (GPL)
```

I don’t change my database. Instead I want to check right now what information is stored in information_schema.

```
SELECT * FROM INFORMATION_SCHEMA;
Information related to following fields will be displayed:
+ - - - - - - - - - - - - - - - - - + - - - - - - - - - - - - -+ - - - - - - -- - - - - - -+ - - - -
- - - -- - - - -- + - - - - - - - - - - - - - -
+ - - - - - - - - - - - - - - - - - - +
TABLE_CATALOG       | TABLE_SCHEMA | TABLE_NAME  |
TABLE_TYPE              | ENGINE
| VERSION                     | ROW_FORMAT   | TABLE_ROWS |
AVG_ROW_LENGTH | DATA_LENGTH        | MAX_DATA_LENGTH |
INDEX_LENGTH   | DATA_FREE    | AUTO_INCREMENT   |
CREATE_TIME                | UPDATE_TIME            | CHECK_TIME       |
TABLE_COLLATION| CHECKSUM       | CREATE_OPTIONS |
TABLE_COMMENT     |
+ - - - - - - - - - - - - - - - - - + - - - - - - - - - - - - -+ - - - - - - -- - - - - - -+ - - - -
- - - -- - - - -- + - - - - - - - - - - - - - -
+ - - - - - - - - - - - - - - - - - - +
```

To see what all tables exist in the school database of MySQL server, I execute the following query:

```
SELECT TABLE_NAME FROM INFORMATION_SCHEMA.tables
WHERE TABLE_SCHEMA = 'sc
```

```
hool';
+  - - - - - - - - - - - - - - - - - - - - - +
| table_name                        |
+  - - - - - - - - - - - - - - - - - - - - - +
| dept_data                         |
| dept_hod                          |
| dept_view                         |
| engineering_students         |
| engineering_students2016 |
| family_data                       |
| school_level                      |
| student_data                    |
| student_view                     |
| table_of_strings               |
| teacher_data                    |
+  - - - - - - - - - - - - - - - - - - - - - +
11 rows in set (0.01 sec)
```

Same way you can view all the information regarding views in a database:

```
mysql> select * from information_schema.VIEWS where table_schema =
'school';
+ - - - - - - - - - - - - - -+ - - - - - - - - - - - - -+ - - - - - - - - - - - - +- - - - - - - - - - -
- - -+
| TABLE_CATALOG | TABLE_SCHEMA | TABLE_NAME   |
VIEW_DEFINITION
| CHECK_OPTION    | IS_UPDATABLE  | DEFINER           |
SECURITY_TYPE |
+ - - - - - - - - - - - - - -+ - - - - - - - - - - - - -+ - - - - - - - - - - - - +- - - - - - - - - - -
- - -+
```

We create a view DEPT_HOD in the last chapter. Let’s see what information we have for it:

```
mysql> select * from information_schema.VIEWS where table_schema =
'school'and t
able_name='DEPT_HOD';
TABLE_CATALOG: NULL
```

```
TABLE_SCHEMA : school
TABLE_NAME : dept_hod
VIEW_DEFINITION: /* ALGORITHM=UNDEFINED */ select `
a`.`HOD` AS `HOD`,`b`.`ENGG_NAME` AS `ENGG_NAME` from
`school`.`dept_data` `a` j
oin `school`.`engineering_students` `b` where (`a`.`ENGG_ID` =
`b`.`ENGG_ID`) or
der by `a`.`HOD`
CHECK_OPTION : NONE
IS_UPDATABLE  : YES
DEFINER: schooladmin@localhost
SECURITY_TYPE : DEFINER
```

The other ways of accessing Metadat is by using SSHOW DATABASES or SHOW TABLES command.

```
mysql> show databases;
+- - - - - - - - - - - - - - - - - -+
| Database                    |
+- - - - - - - - - - - - - - - - - -+
| information_schema |
| school                      |
+- - - - - - - - - - - - - - - - - -+
2 rows in set (0.06 sec)
mysql> SHOW TABLES;
+ - - - - - - - - - - - - - - - - - -  - - - -+
| Tables_in_school                 |
+ - - - - - - - - - - - - - - - - - -  - - - -+
| dept_data                          |
| dept_hod                           |
| dept_view                          |
| engineering_students         |
| engineering_students2016 |
| family_data                        |
| school_level                       |
| student_data                      |
| student_view                      |
| table_of_strings                 |
| teacher_data                    |
+ - - - - - - - - - - - - - - - - -  - - - -+
11 rows in set (0.01 sec)
```

Conclusion

This concludes the final book in the series! Thank you for purchasing SQL: Elite Level SQL From The Ground Up. If you would like to further advance your coding knowledge, I suggest that you look for my JavaScript From The Ground Up series. JavaScript is the industry standard for building websites all over the world, and one of the most in demand programming languages in the software industry. Whether you want to work for a company or start your own, JavaScript is the tool you need to further your goals. Take a look!

Finally, if you enjoyed this book, a positive review is always appreciated!