
03
SQL
WRITTEN BY KEITH DVORJAK

SQL

Advanced Level SQL From The Ground Up

© Copyright 2017 by Keith Dvorjak - All rights reserved.

The transmission, duplication, or reproduction of any of the following work including specific information will be considered an illegal act irrespective of if it is done electronically or in print. This extends to creating a secondary or tertiary copy of the work or a recorded copy and is only allowed with express written consent from the Publisher. All additional right reserved.

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.

Table of Contents

- Aggregate Functions

Chapter 4 Using Subqueries

- Concept
- Correlated vs Non correlated Subqueries

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('012345678901234567890', '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 ‘012345678901234567890’ 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
='012345678901234567890123456789' 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 ='0123456789012345678901234567890

' 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, 3rd 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, 3rd 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

Subqueries 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!