Bucknell UNIVERSITY

CSCI 308, Spring 2023
Course Project
Nishant Shrestha
Report: SQL (MySQL)
Theme: Financial Derivatives

1. Hello World

Here is hello world in MySQL

```
SELECT "Hello World" AS Output;
```

```
 mysql> SELECT "Hello World" AS Output;
+-------------+
| Output      |
+-------------+
| Hello World |
+-------------+
1 row in set (0.00 sec)
```

Source: I knew how to do this

2. Background :

SQL (Structured Query Language) is declarative programming language used to manipulate relational databases. It was developed in 1970 by IBM to use on their DBMS named System R. The language has a wide range of functionality to working with relational databases including deleting, updating, and inserting data and querying data as well. [1]

Multiple Database Management Systems including MySQL, Oracle, and Microsoft SQL use SQL as the core language. My report uses MySQL. MySQl is an open-source software developed by a Swedish company, MySQL AB. The software was released in 1995.[2]

SQL is a widely-used language in finance. In the options market (A form of financial derivative), it is used to manipulate time-series data and generate financial models that are used to price options[3]. Generally, jobs such as data analysts, data engineering seek employees adept in SQL.

1. https://www.businessnewsdaily.com/5804-what-is-sql.html# (What is SQL, BND)
2. https://en.wikipedia.org/wiki/MySQL (MySQL, Wiki)
3. https://www.linkedin.com/learning/sql-for-finance-professionals/sql-in-the-finance-field#:~:text=%2D%20SQL%20can%20be%20a%20very,the%20heart%20of%20financial%20analysis. (SQL in the finance field)

3. BNF Grammar for WHILE and RETURN in SQL

```
WHILE  "(" <condition> ")" <statement>

RETURN (<expression>)?
```

Nothing to output

Source: http://teiid.github.io/teiid-documents/9.0.x/content/reference/BNF_for_SQL_Grammar.html#select

4. Free or fixed format?

SQL (MySQL) is a free format and you can write the whole code in a single line (1) or space it like the code (2). A “;” is used to indicate an end of a code statement and “()“ are also used when creating tables.

Both of the code blocks below perform the same functionality:

USE csci308; # database to be used

# created two tables options and bonds, and used select to check if they work.
(1)

```
CREATE TABLE options ( put VARCHAR(50),price INT);
```

(2)

```
CREATE TABLE bonds (

  coupon VARCHAR(50),

  date INT

);

SELECT * FROM options;
SELECT * FROM bonds;
```

Both ways of creating a table works :

Output:

```
mysql> SELECT * FROM options;
Empty set (0.00 sec)

mysql> SELECT * FROM bonds;
Empty set (0.00 sec)
```

Source: Tested the code out myself.

5. Non-Standard Math Operations (work needed)

^ (Bitwise exclusive OR)

```
SET @Option1 = 10;
SET @Option2 = 6;
SELECT @Option1 ^ @Option2;
```

```
mysql> SELECT @Option1 ^ @Option2;
+------------------+
| @Option1 ^ @Option2 |
+------------------+
|             12 |
+------------------+
```

Source: https://www.dataquest.io/blog/sql-operators/ (sql operators)

6. Arrays in sql (work needed)

SQL supports multidimensional arrays, with not specific limitations to the dimensions. These represent the mapping type.

The following code shows the creation of 1-D and 2-D arrays inside a table in SQL.

1-D array

```
SET @array = JSON_ARRAY(1, 2, 3, 4, 5);
SELECT JSON_EXTRACT(@array, '$[0]') AS Option1, JSON_EXTRACT(@array, '$[1]') AS Option2,
JSON_EXTRACT(@array, '$[2]') AS Optiont3, JSON_EXTRACT(@array, '$[3]') AS Option4, JSON_EXTRACT(@array,
'$[4]') AS Option5;
```

2-D array

```
SET @array = JSON_ARRAY(JSON_ARRAY(1, 2, 3), JSON_ARRAY(4, 5, 6), JSON_ARRAY(7, 8, 9));
SELECT JSON_EXTRACT(JSON_EXTRACT(@array, '$[0]'), '$[0]') AS Option11,
JSON_EXTRACT(JSON_EXTRACT(@array, '$[0]'), '$[1]') AS Option12, JSON_EXTRACT(JSON_EXTRACT(@array,
'$[0]'), '$[2]') AS Option13,
    JSON_EXTRACT(JSON_EXTRACT(@array, '$[1]'), '$[0]') AS Option21,
JSON_EXTRACT(JSON_EXTRACT(@array, '$[1]'), '$[1]') AS Option22, JSON_EXTRACT(JSON_EXTRACT(@array,
'$[1]'), '$[2]') AS Option23,
    JSON_EXTRACT(JSON_EXTRACT(@array, '$[2]'), '$[0]') AS Option31,
JSON_EXTRACT(JSON_EXTRACT(@array, '$[2]'), '$[1]') AS Option32, JSON_EXTRACT(JSON_EXTRACT(@array,
'$[2]'), '$[2]') AS Optiont33;
```

```
Option1  Option2  Option3  Option4  Option5
-------- -------- -------- -------- --------
       1        2        3        4        5


Option11  Option12  Option13  Option21  Option22  Option23  Option31  Option32  Option33
--------  --------  --------  --------  --------  --------  --------  --------  --------
       1         2         3         4         5         6         7         8         9
```

Source: https://www.educba.com/array-in-sql/ (Array in SQL)

7. Regular expression support (fundamentals) how examples of concatenate (ab), repeat (a*), and selection (a|b).

SQL (MySQL) has regular expression support

To demonstrate the basic functionality I am using the regexp_like() function which to carry out regular expression functionality search:

Regexp takes in a string and the regular expression and returns a 1 if there's a match and 0 otherwise:

```
SELECT REGEXP_LIKE('Put', '^Pu*t');
SELECT REGEXP_LIKE('Puuuut', '^Pu*t');
SELECT REGEXP_LIKE('put', 'put|call');
SELECT REGEXP_LIKE('axc', 'put|call');
```

Output of the four lines of code above:

```
+----------------------------+
| REGEXP_LIKE('Put', '^Pu*t') |
+----------------------------+
|                          1 |
+-----------------------------+
1 row in set (0.02 sec)

+--------------------------------+
| REGEXP_LIKE('Puuuut', '^Pu*t') |
+--------------------------------+
|                              1 |
+--------------------------------+
1 row in set (0.00 sec)

+--------------------------------+
| REGEXP_LIKE('put', 'put|call') |
+--------------------------------+
|                              1 |
+--------------------------------+
1 row in set (0.01 sec)

+--------------------------------+
| REGEXP_LIKE('axc', 'put|call') |
+--------------------------------+
|                              0 |
+--------------------------------+
1 row in set (0.00 sec)
```

Source: https://dev.mysql.com/doc/refman/8.0/en/regexp.html (Regular Expression manual of MySQL)

8. Garbage collection in SQL (MySQL)

Garbage collection is automatic in MySQL. The garbage collector it uses is 'purge thread'. There is no direct command to trigger the garbage collector. Finally, it uses a mark and sweep algorithm to carry out the removal of data.

Source: https://dev.mysql.com/doc/refman/5.7/en/optimizing-innodb-configuration-variables.html (go to the next page for more information on InnoDB and the garbage collector)

9. Short-circuit AND and OR (work needed)

SQL has short-circuit AND and OR

```
CREATE FUNCTION option_price(option INT)
RETURNS BOOLEAN
BEGIN
  SET@ price = option + 10;
  SELECT @price;
  DECLARE @flag boolean;
  SET @flag = true;
  RETURN @flag;
END;
```

AND demonstration:

```
1=0 AND option_price(5);
```

OR demonstration:

```
1=1 OR option_price(5)
```

The AND operator just prints out 0 indicating short-circuit AND. The OR operator just prints out 1 indicating short-circuit OR.

Source: https://www.mysqltutorial.org/mysql-or/ (change page to sql-or)

10. Borrowed Object oriented feature (Functions)

```
CREATE FUNCTION option_price(option INT)
RETURNS INT
BEGIN
  SET @price = option + 10;
  RETURN @price;
END;

SELECT option_price(5);
```

```
mysql> SELECT option_price(5);
+----------------+
| option_price(5)|
+----------------+
|              15|
+----------------+
1 row in set (0.00 sec)
```

Source: https://dev.mysql.com/doc/refman/8.0/en/create-procedure.html (Create Procedure and function)

11. Borrowed Object oriented feature (Stored Procedures)

Work like functions or procedures in SQL. These are used to encapsulate SQL statements and logic and can be repeatedly used by other SQL statements

```
CREATE PROCEDURE print_options()
BEGIN
   SELECT 'Options are Call and Put' AS message;
END
```

Run : CALL print_options();

This prints : 'Options are Call and Put'

Source: https://www.w3schools.com/sql/sql_stored_procedures.asp (w3schools Introduction)
https://www.sqlshack.com/learn-mysql-the-basics-of-mysql-stored-procedures/ (code example)

12. Borrowed Object oriented feature (Triggers)

Triggers emulate event handling in SQL. The following code creates a simple table with one variable and the trigger automatically changes the input to an uppercase word.

```
CREATE TABLE derivatives (
  type VARCHAR(50)
);

CREATE TRIGGER uppercase_type
BEFORE INSERT ON derivatives
FOR EACH ROW
SET NEW.type = UPPER(NEW.name);          # UPPER is an in-built function
```

Run: the lines below to see an output:

```
INSERT INTO derivatives (type) VALUES ('option');
INSERT INTO derivatives (type) VALUES ('cAll'');
INSERT INTO derivatives (type) VALUES ('puT');

SELECT * FROM derivatives
```

Output:

```
+----+-------------+
| id | name        |
+----+-------------+
| 1  | OPTION      |
| 2  | CALL        |
| 3  | PUT         |
+----+-------------+
```

Source: https://dev.mysql.com/doc/refman/8.0/en/trigger-syntax.html (Triggers MYSQL reference manual)

13. Control Statements (Borrowed features from Imperative Languages)

SQL is a declarative language, as such control statements aren't native to SQL. However, SQL supports control statements like IF, WHILE, FOR, CASE and a few others

The following code showcases the use of FOR in SQL using a stored procedure.

```
DELIMITER //
CREATE PROCEDURE print_option_five_times()
BEGIN
    DECLARE i INT DEFAULT 1;
    FOR i IN 1..5 DO
        SELECT 'Option';
    END FOR;
END //
DELIMITER ;

CALL print_option_five_times();
```

output:

```
+--------+
| Option |
+--------+
| Option |
| Option |
| Option |
| Option |
| Option |
+--------+
```

Source:
https://www.techtarget.com/searchapparchitecture/tip/The-basics-of-working-with-declarative-programming-languages
(features of a declarative programming language, including SQL)

https://stackoverflow.com/questions/5125096/for-loop-example-in-mysql (code reference)

14. Pass by Value or Reference:

MySQL appears to be pass by value:

```
SET @price = 10;

DELIMITER //
CREATE PROCEDURE increase_price(IN value INT)
BEGIN
   SET value = value + 1;
   SELECT value;
END //
DELIMITER ;

CALL increase_price(@price);
SELECT @price;
```

We see that our variable doesn't change even when the procedure changes it. Thus, this is pass-by-value.

```
+-------+
| value |
+-------+
|    11 |
+-------+
1 row in set (0.00 sec)

+-----------+
| @price |
+-----------+
|        10 |
+-----------+
1 row in set (0.00 sec)
```

Source: Completed the question myself.

15. Default parameters?

MySQL does support default parameters:

```
CREATE FUNCTION add_portfolio_value(call INT, put INT DEFAULT 10)
RETURNS INT
BEGIN
   DECLARE value INT;
   SET value = call + put;
   RETURN result;
END;

SELECT add_portfolio_value(5);
```

```
+---------------------+
| add_portfolio_value(5)  |
+---------------------+
|                 15 |
+---------------------+
```

Source: https://dev.mysql.com/doc/refman/8.0/en/create-procedure.html (Create functions in MySQL)

16. Implicitly or explicitly typed/declared

In MySQL, variables are explicitly declared but implicitly typed when they are declared as standalone variables:

```
SET @price = 20;
SET @price = “options are hard to price”; #showing explicit declaration but implicit typing

SELECT @price;
```

However, when used inside statements, variables can be implicitly declared and typed:

```
SELECT COUNT(*) INTO @coupon FROM bonds;  #bonds is a table in my database
```

Output:

```
mysql> INSERT INTO bonds VALUES ("50%",09);   # inserting an observation to the table bonds
Query OK, 1 row affected (0.01 sec)
```

#The following code implicitly declared and typed the variable @coupon. The program itself identified the type of @coupon during execution and it also wasn’t declared explicitly.

```
 mysql> SELECT COUNT(*) INTO @coupon FROM bonds;
Query OK, 1 row affected (0.01 sec)

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

Source: https://dev.mysql.com/doc/refman/8.0/en/create-procedure.html (Create functions in MySQL)

17. String Data type in MySQL?

MySQL contains the following String data types: CHAR, VARCHAR, BINARY, VARBINARY, BLOB, TEXT, ENUM, and SET. While CHAR, VARCHAR, ENUM, SET, and TEXT are pre-defined data types, BINARY, VARBINARY, and BLOB are nonstandard MySQL extensions.

There are no terminating characters in MySQl for strings. Instead, the length of the string is specified when declaring the variable. For example, VARCHAR(5).

Math operators for strings: Mathematical strings can't be used with strings. However, there are a few functions that can be used to manipulate strings including CONCAT(), LENGTH(), SUBSTRING(), REPLACE().

```
SELECT LENGTH("derivatives"); #returns the length of the string
SELECT CONCAT('Hello', ' ', 'FINANCE'); # concatenates two strings
SELECT SUBSTRING('Finance is awesome', 6, 2); # gets a substring from the string
SELECT REPLACE('FINANCE is banks', 'banks', 'stocks'); #replace 'banks' with 'stocks'.
```

```
mysql> SELECT LENGTH('derivatives');
+-----------------------+
| LENGTH('derivatives') |
+-----------------------+
|                    11 |
+-----------------------+
1 row in set (0.00 sec)


mysql> SELECT CONCAT('Hello', ' ', 'FINANCE');
+---------------------------------+
| CONCAT('Hello', ' ', 'FINANCE') |
+---------------------------------+
| Hello FINANCE                   |
+---------------------------------+
1 row in set (0.00 sec)

mysql> SELECT SUBSTRING('Finance is awesome', 6, 2);
+---------------------------------------+
| SUBSTRING('Finance is awesome', 6, 2) |
+---------------------------------------+
| ce                                    |
+---------------------------------------+
1 row in set (0.00 sec)

mysql> SELECT REPLACE('FINANCE is banks', 'banks', 'stocks');
+------------------------------------------------+
| REPLACE('FINANCE is banks', 'banks', 'stocks') |
+------------------------------------------------+
| FINANCE is stocks                              |
+------------------------------------------------+
1 row in set (0.00 sec)
```

Source:https://dev.mysql.com/doc/refman/8.0/en/string-type-syntax.html#:~:text=The%20string%20data%20types%20are.TABLE%20or%20ALTER%20TABLE%20statement. (MySQL Reference Manual : String Data types)

18. Logical operations

Table 12.5 Logical Operators (Copied directly from the manual)

Name	Description
AND, &&	Logical AND
NOT, !	Negates value
OR, ||	Logical OR
XOR	Logical XOR

Output:
```
mysql> SELECT 1 OR 0;
+--------+
| 1 OR 0 |
+--------+
|      1 |
+--------+
1 row in set (0.00 sec)

mysql> SELECT 1 AND 0;
+---------+
| 1 AND 0 |
+---------+
|       0 |
+---------+
1 row in set (0.01 sec)

mysql> SELECT NOT 0;
+-------+
| NOT 0 |
+-------+
|     1 |
+-------+
1 row in set (0.00 sec)

mysql> SELECT 1 XOR 0;
+---------+
| 1 XOR 0 |
+---------+
|       1 |
+---------+
1 row in set (0.00 sec)
```

Source: https://dev.mysql.com/doc/refman/8.0/en/logical-operators.html (Logical operators - MySQL reference manual)

19. Static vs Dynamic Typing:

MySQL uses statically typed SQL (There are SQL implementations with dynamic typing discussed in the referenced article).

The following code demonstrates static typing:

```
CREATE TABLE contracts (
  price INT,
  type VARCHAR(255)
);

INSERT INTO contracts (price, type) VALUES (50, 'forward');
INSERT INTO contracts (price, type) VALUES ("ji", 60);
```

```
mysql> INSERT INTO contracts (price, type) VALUES (50, 'forward');
Query OK, 1 row affected (0.01 sec)

# the line above works perfectly

mysql> INSERT INTO contracts (price, type) VALUES ("ji", 60);
ERROR 1366 (HY000): Incorrect integer value: 'ji' for column 'price' at row 1
```

# This line does not work since price is statically assigned an Int type, so passing a string will make the program crash. Notice the second input works correctly since VARCHAR can store numeric data, including 60.

Source: https://rockset.com/blog/dynamic-typing-in-sql/ (dynamic typing in SQL)

20. Primitive types (10 of the most common ones.)

String Data types :
- CHAR : length is specified at declaration and stores a fixed length of characters.
- VARCHAR : length is specified at declaration but can store a variable length of characters.

Numeric types:
- INT: store signed or unsigned integers; signed range - -2,147,483,648 to 2,147,483,647, unsigned range is 0 to 4,294,967,295; 4 bytes.
- DECIMALI: used to store exact numeric values with specified precision and scale; range is specified at declaration; size can be varied.
- FLOAT: store floating point numbers; range is -3.4028235E38 to -1.17549435E-38, 0, and 1.17549435E-38 to 3.4028235E38; 4 bytes.
- DOUBLE: has more range than Float; range is -1.7976931348623157E308 to -2.2250738585072014E-308, 0, and 2.2250738585072014E-308 to 1.7976931348623157E308; 8 bytes.

BOOLEAN: 1 is TRUE and 0 is FALSE; 1 byte.

Date time types:
- DATE: used to store date; range is '1000-01-01' to '9999-12-31; 3 bytes.
- TIME: used to store precise time upto the second range is '-838:59:59' to '838:59:59'.; 3 bytes.
- DATETIME: stores both date and time values; range is '1000-01-01 00:00:00' to '9999-12-31 23:59:59'; 8 bytes.

```
CREATE TABLE financial_derivative (
  int INT,
  float FLOAT,
  double DOUBLE,
  decimal DECIMAL(10,3),
  boolean BOOLEAN,
  char CHAR(10),
  varchar VARCHAR(60),
  date DATE,
  time TIME,
  datetime DATETIME
);

INSERT INTO financial_derivative VALUES (
  4,
  3.14159,
  3.14159265359,
  1234.56,
  TRUE,
  'calls',
  'options',
  '2023-04-27',
  '08:30:00',
  '2023-04-27 08:30:00'
);
```

Output:

```
mysql> SELECT * FROM financial_derivative;
+--------+----------+---------------+------------+------------+---------+------------+------------+----------+---------------------+
| int | float | double     | decimal | boolean | char | varchar | date     | time  | datetime         |
+--------+----------+---------------+------------+------------+---------+------------+------------+----------+---------------------+
|      4 |  3.14159 | 3.14159265359 |    1234.56 |          1 | calls   | options      | 2023-04-27 | 08:30:00 | 2023-04-27
08:30:00 |
+--------+----------+---------------+------------+------------+---------+------------+------------+----------+---------------------+
1 row in set (0.01 sec)
```

Source: https://dev.mysql.com/doc/refman/8.0/en/data-types.html(MySQL reference manual: Data types)

21. Enum(Constructed data type)

MySQL has Enum as a constructed data type.It is a string object that only allows permitted values to be added to columns. The ENUM values should always be enclosed in quotation marks.

```
CREATE TABLE stocks (
    symbol VARCHAR(40),
    index ENUM('dow', 's&m')
);
INSERT INTO stocks (symbol, index) VALUES ('goog','s&m'), ('aapl','dow');

SELECT symbol, index FROM stocks WHERE index = 's&m';
```

```
+---------------------+
|                15 |
+---------------------+
```

Source: https://dev.mysql.com/doc/refman/8.0/en/enum.html (Enum data type: MySQL reference manual)

22. SET(Constructed data type)

MySQL has SET as a constructed data type.It is a string object that allows zero or more values and these values must be from a list of permitted values . The ENUM values should always be enclosed in quotation marks. Duplicate values in the SET give off a warning or can set off an error if the necessary setting are enabled. Additionally, a set such as SET('call','put') can have the following values:

' '
'call'
'put',
'call, 'put'

This data type represents the Union data type.

```
CREATE TABLE derivatives_type (
  type SET('forwards', 'futures', 'options')
);

INSERT INTO derivatives_type (type) VALUES ('futures,options'), ('futures,futures'), ('forwards,futures');
```

```
mysql> SELECT * FROM derivatives_type;
+-----------------+
| type            |
+-----------------+
| futures,options  |
| futures          |
| forwards,futures |
+-----------------+
```

We can see that the duplicate value wasn't inserted in the table.

Source: https://dev.mysql.com/doc/refman/8.0/en/enum.html (Enum data type: MySQL reference manual)

23. JSON (Constructed data type)

This data type allows you to store JSON documents as columns in tables. This data type has a list of functions, including JSON_TYPE(), JSON_ARRAY(), and JSON_OBJECT(), to manipulate JSON lists, JSON objects, JSON strings and so on. As a result, you can encapsulate different data types.

```
SELECT JSON_TYPE('["a", "b", 1]');
SELECT JSON_TYPE('"hello"');
SELECT JSON_TYPE('hello'); –explain why this doesn't work


SELECT JSON_ARRAY('a', 1, NOW());
SELECT JSON_OBJECT('option1', 1, 'option2', 'forward');
```

```
mysql> SELECT JSON_TYPE('["put", "call", 1]');
+----------------------------+
| JSON_TYPE('["put", "call", 1]') |
+----------------------------+
| ARRAY                      |
+----------------------------+
1 row in set (0.01 sec)

mysql> SELECT JSON_TYPE('"options"');
+----------------------+
| JSON_TYPE('"options"') |
+----------------------+
| STRING               |
+----------------------+
1 row in set (0.00 sec)

mysql> SELECT JSON_TYPE('options');
ERROR 3141 (22032): Invalid JSON text in argument 1 to function json_type: "Invalid value." at position 0.
```

- This line of code doesn’t represent any form of data type native to sql.

```
mysql> SELECT JSON_ARRAY('put', 1, NOW());
+----------------------------------------+
| JSON_ARRAY('a', 1, NOW())              |
+----------------------------------------+
| ["a", 1, "2023-05-01 12:36:28.000000"] |
+----------------------------------------+
1 row in set (0.00 sec)

mysql> SELECT JSON_OBJECT('option1', 1, 'option2', 'forward');
+-------------------------------------------------+
| JSON_OBJECT('option1', 1, 'option2', 'forward') |
+-------------------------------------------------+
| {"option1": 1, "option2": "forward"}            |
+-------------------------------------------------+
1 row in set (0.00 sec)
```

Source: https://dev.mysql.com/doc/refman/8.0/en/json.html (JSON data type: MySQL reference manual)

24. SPATIAL (Constructed data type)

This data type allows you to represent spatial values. The MySQL spatial extension contains this data type and this data type is a subset of the SQL with Geometry types. This data type also contains special functions to analyze and create geometry value.

SPATIAL data types like GEOMETRY, POINT, and so on contain single geometry values, while MULTIPOINT, MULTIPOLYGON can hole multiple values.

Example showing point data type:

```
CREATE TABLE option_point (
  locality POINT NOT NULL
);

INSERT INTO option_point ( locality) VALUES ( POINT(10, 25));
```

```
SELECT * FROM option_point;
```

```
mysql> SELECT * FROM option_point;
+------------------------------------------------------+
| locality                                             |
+------------------------------------------------------+
| 0x000000000101000000000000000000002440000000000000003940 |
+------------------------------------------------------+
1 row in set (0.00 sec)
```

Source: https://dev.mysql.com/doc/refman/8.0/en/spatial-types.html (SPATIAL data type: MySQL reference manual)

25. Implicit coercion

String to Integer : Yes in some cases.

```
SELECT '123' + 1; -- Output: 124
```

Integer to String : Yes in some cases.

```
SELECT CONCAT('The value of my integer is: ', 123);
```

Decimal to Int : Yes

```
CREATE TABLE port1 (
  call1 INT
);

INSERT INTO port1 (call1) VALUES (123.45);

SELECT * FROM port1;
```

Int to Decimal : Yes

```
CREATE TABLE port2 (
 decimal_call DECIMAL(10,2)
);

INSERT INTO port2 (decimal_call) VALUES (123);
SELECT * FROM port2;
```

Boolean to String: Yes

```
SELECT CONCAT('The value is ', TRUE);
SELECT CONCAT('The value is ', FALSE);
```

String to Boolean: Yes

```
CREATE TABLE port3 (
  bool_call BOOLEAN
);
INSERT INTO port3 (bool_call) VALUES (TRUE);
SELECT * FROM port3;
```

String to Date :

```
SELECT STR_TO_DATE('2023-04-28', '%Y-%m-%d');
```

Date to string:

```
SELECT CONCAT('The date is ', '2023-04-28');
```

Decimal to String:

```
SELECT CONCAT('The price is $', 9.99);
```

String to Decimal:

```
SELECT 10 + '8.99';
```

Boolean to Int:

```
 SELECT TRUE + 5;
```

Int to Boolean:

```
SELECT 1 = TRUE;
```

from\to	Int	Decimal	String	Date	Boolean
Int	-	Yes	Yes	No	Yes
Decimal	Yes	-	Yes	No	No
String	Yes	Yes	-	Yes	Yes
Date	No	No	Yes	-	No
Boolean	Yes	No	Yes	No	-

Note: Decimal behaves the same way as its subtypes like Float. Likewise, with Date and Datetime, and time, and string types including VARCHAR, CHAR, and others.

Output:

```
mysql> SELECT '123' + 1; -- Output: 124
+-----------+
| '123' + 1 |
+-----------+
|       124 |
+-----------+
1 row in set (0.00 sec)

mysql> SELECT CONCAT('The value of my integer is: ', 123);
+---------------------------------------------+
| CONCAT('The value of my integer is: ', 123) |
+---------------------------------------------+
| The value of my integer is: 123             |
+---------------------------------------------+
1 row in set (0.00 sec)

mysql> SELECT * FROM port1;
+-------+
| call1 |
+-------+
|   123 |
+-------+
1 row in set (0.00 sec)

mysql> SELECT * FROM port2;
+--------------+
| decimal_call |
+--------------+
|       123.00 |
+--------------+
1 row in set (0.00 sec)
```

```
mysql> SELECT CONCAT('The value is ', TRUE);
+-------------------------------+
| CONCAT('The value is ', TRUE) |
+-------------------------------+
| The value is 1                |
+-------------------------------+
1 row in set (0.00 sec)

mysql> SELECT CONCAT('The value is ', FALSE);
+--------------------------------+
| CONCAT('The value is ', FALSE) |
+--------------------------------+
| The value is 0                 |
+--------------------------------+
1 row in set (0.00 sec)

mysql> SELECT * FROM port3;
+-----------+
| bool_call |
+-----------+
|         1 |
+-----------+
1 row in set (0.00 sec)

mysql> SELECT STR_TO_DATE('2023-04-28', '%Y-%m-%d');
+---------------------------------------+
| STR_TO_DATE('2023-04-28', '%Y-%m-%d') |
+---------------------------------------+
| 2023-04-28                            |
+---------------------------------------+
1 row in set (0.00 sec)

mysql> SELECT CONCAT('The date is ', '2023-04-28');
+--------------------------------------+
| CONCAT('The date is ', '2023-04-28') |
+--------------------------------------+
| The date is 2023-04-28               |
+--------------------------------------+
1 row in set (0.00 sec)

mysql> SELECT CONCAT('The price is $', 9.99);
+--------------------------------+
| CONCAT('The price is $', 9.99) |
+--------------------------------+
| The price is $9.99             |
+--------------------------------+
1 row in set (0.00 sec)


mysql> SELECT 10 + '8.99';
+--------------------+
| 10 + '8.99'        |
+--------------------+
| 18.990000000000002 |
+--------------------+
1 row in set (0.00 sec)

mysql> SELECT TRUE + 5;
+----------+
| TRUE + 5 |
+----------+
|        6 |
+----------+
1 row in set (0.00 sec)

mysql>  SELECT 1 = TRUE;
+----------+
| 1 = TRUE |
+----------+
|        1 |
+----------+
1 row in set (0.01 sec)
```

Source: https://dev.mysql.com/doc/refman/8.0/en/data-types.html (MySQL reference Manual Data types)