**Advanced SQL**

AUTHOR: COMPILED BY WERE VINCENT

DATE: 17/FEBRUARY/2022

**Advanced SQL**

**Introduction**

The realm of database management and querying has evolved significantly, enabling organizations to extract meaningful insights and efficiently manage their data resources. In the context of relational databases, Structured Query Language (SQL) serves as the backbone, offering a wide array of functionalities to interact with data. One crucial aspect of SQL is the advanced level of proficiency that can be achieved through comprehensive training, allowing individuals to harness its capabilities for sophisticated tasks.

This learning guide introduces the Advanced SQL course, building upon the foundation of SQL Fundamentals. This 75-hour self-directed practical training covers topics such as functions, procedures, triggers, views, and more, enabling students to master complex SQL operations and amplify their database management prowess.

**Modules**

***Module 1: MySQL aggregate functions***

1. *Exercise 1.1: AVG*

The MySQL AVG() function is used to calculate the average value of a numeric column in a table. It takes a single argument, which is typically the column name or an expression involving column(s) on which you want to calculate the average. The result of the AVG() function is the arithmetic mean of all the values in the specified column e.g.

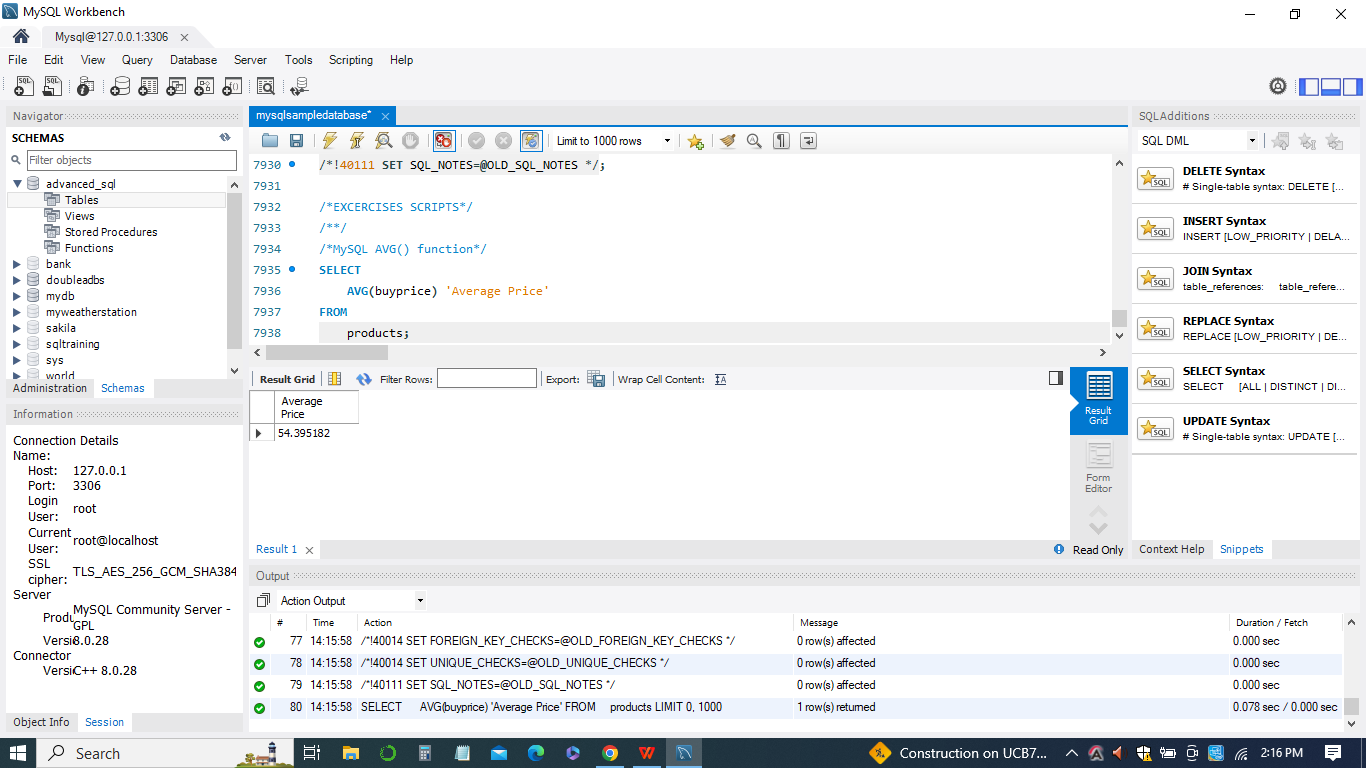
*SELECT*

*AVG(buyprice) 'Average Price'*

*FROM*

*products;*

The AVG(buyprice) part calculates the average value of the 'buyprice' column in the 'products' table.The AS 'Average Price' renames the calculated average value column to 'Average Price' in the result set.



1. *Exercise 1.2: COUNT*

The MySQL COUNT() function is used to count the number of rows in a table that satisfy a certain condition or have non-null values in a specified column. It is commonly used to determine the total number of records that meet a specific criterion. The function can also be used without any argument to count all rows in the table.

In your example:

The CREATE TABLE statement creates a table named count\_demos with columns id and val.

The INSERT INTO statement inserts multiple rows into the count\_demos table with different values for the val column, including some NULL values*.*

*CREATE TABLE count\_demos (*

*id INT AUTO\_INCREMENT PRIMARY KEY,*

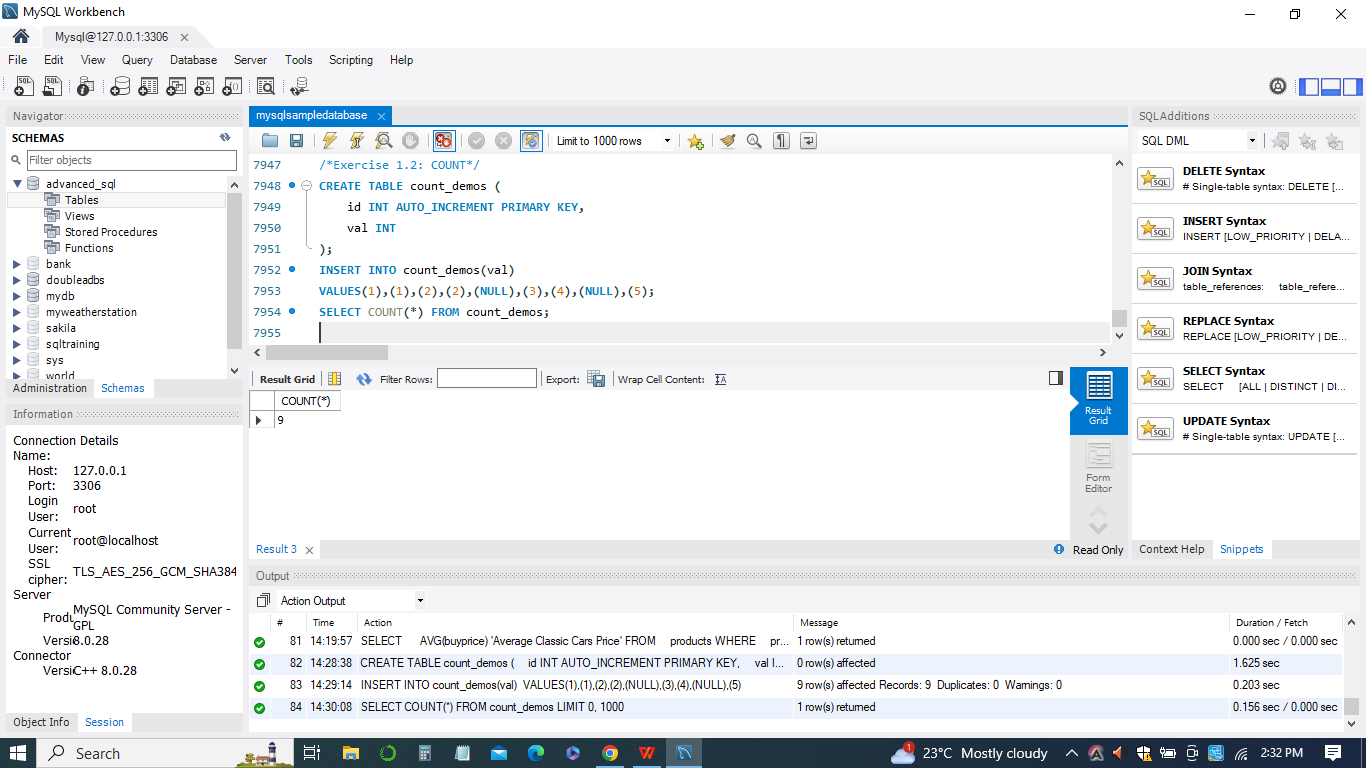
*val INT*

*);*

*INSERT INTO count\_demos(val)*

*VALUES(1),(1),(2),(2),(NULL),(3),(4),(NULL),(5);*

*SELECT COUNT(\*) FROM count\_demos;*

**

1. *Exercise 1.3: SUM*

The MySQL SUM() function is used to calculate the sum of all values in a specified column of a table. It adds up the numeric values in the specified column and returns the total sum.In the example:

*CREATE TABLE sum\_demo (*

*n INT*

*);*

*-INSERT INTO sum\_demo(n)*

*VALUES(1),(1),(2),(NULL),(3);*

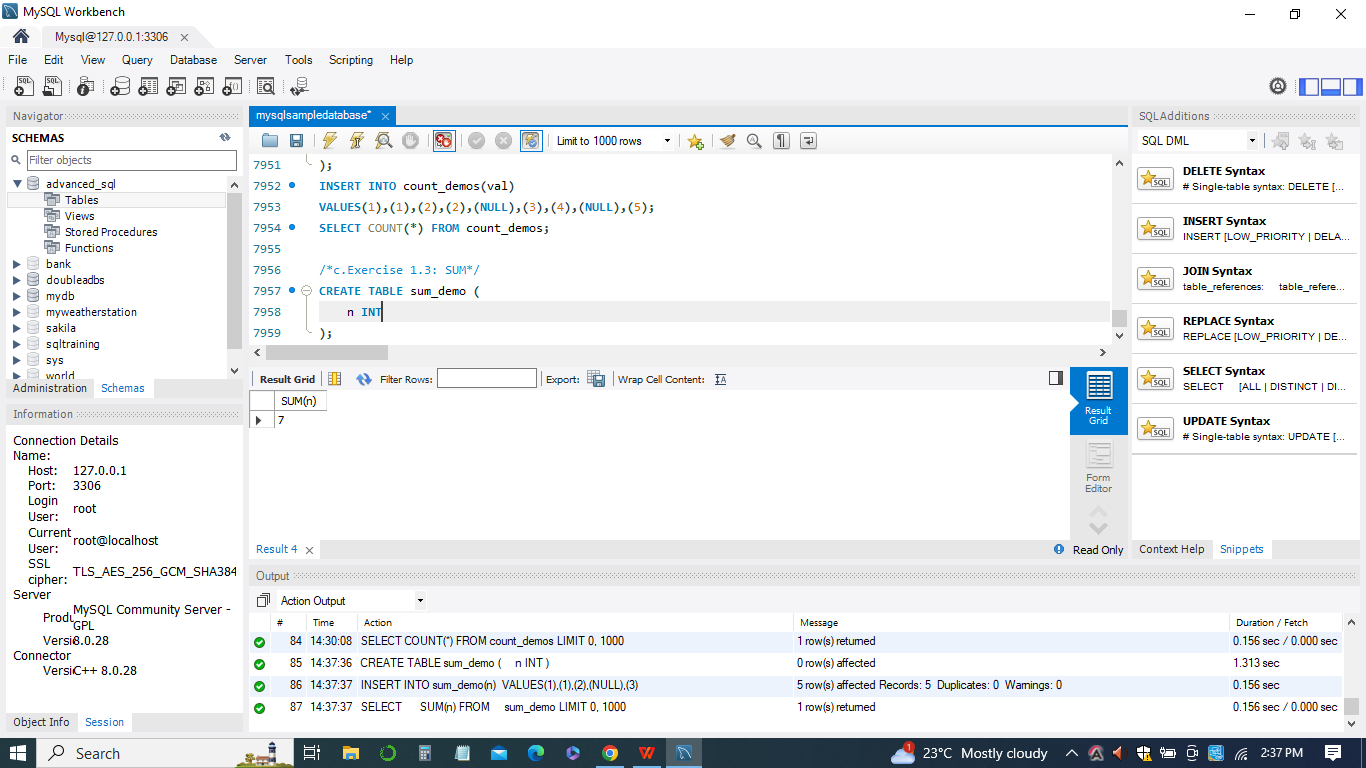
*SELECT*

*SUM(n)*

*FROM*

*sum\_demo;*

The CREATE TABLE statement creates a table named sum\_demo with a single column named n.The INSERT INTO statement inserts multiple rows into the sum\_demo table with different values for the n column, including some NULL values.

**

1. *Exercise 1.4: MIN*

The MySQL MIN() function is used to retrieve the minimum value from a specified column in a table. It finds the smallest value in the column and returns that value.

In the example query:

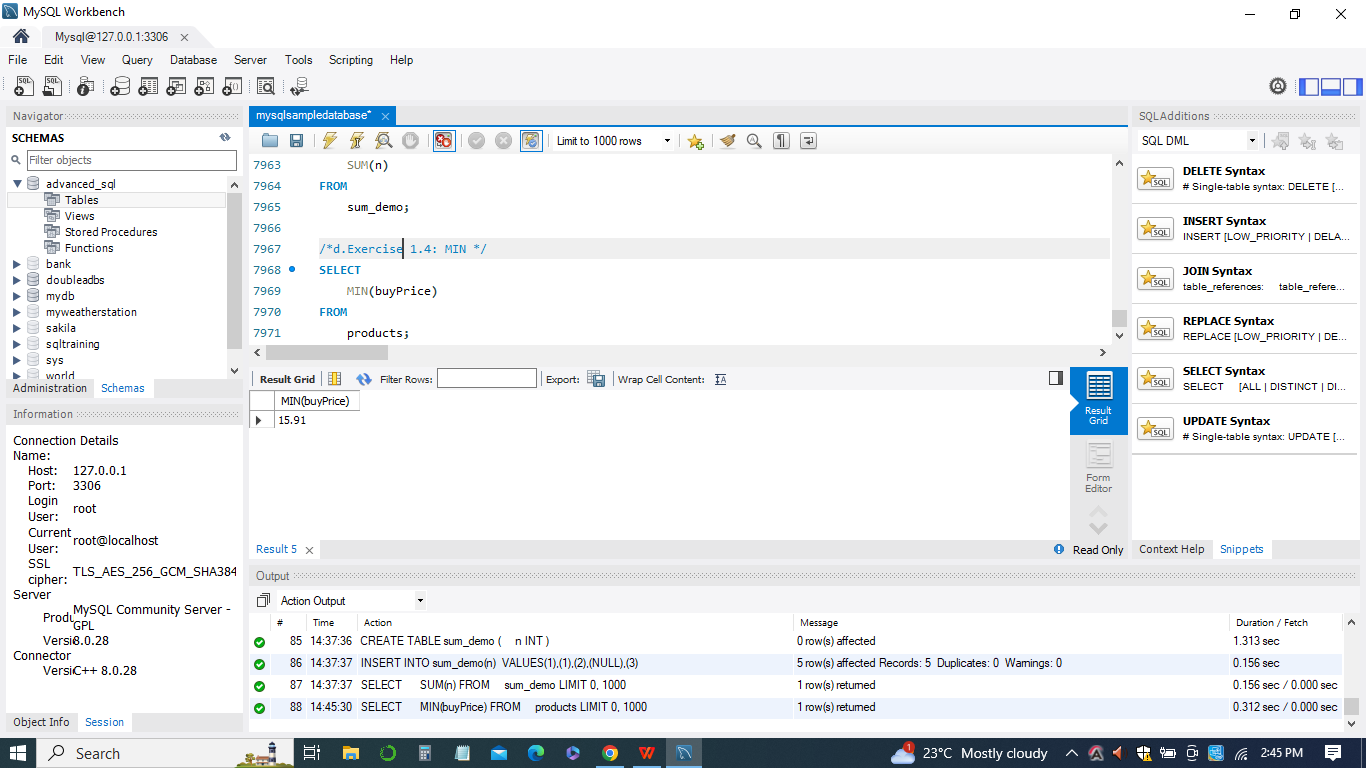
*SELECT*

*MIN(buyPrice)*

*FROM*

*products;*

The query calculates the minimum value from the 'buyPrice' column in the 'products' table. The result will be the smallest value found in that column.

**

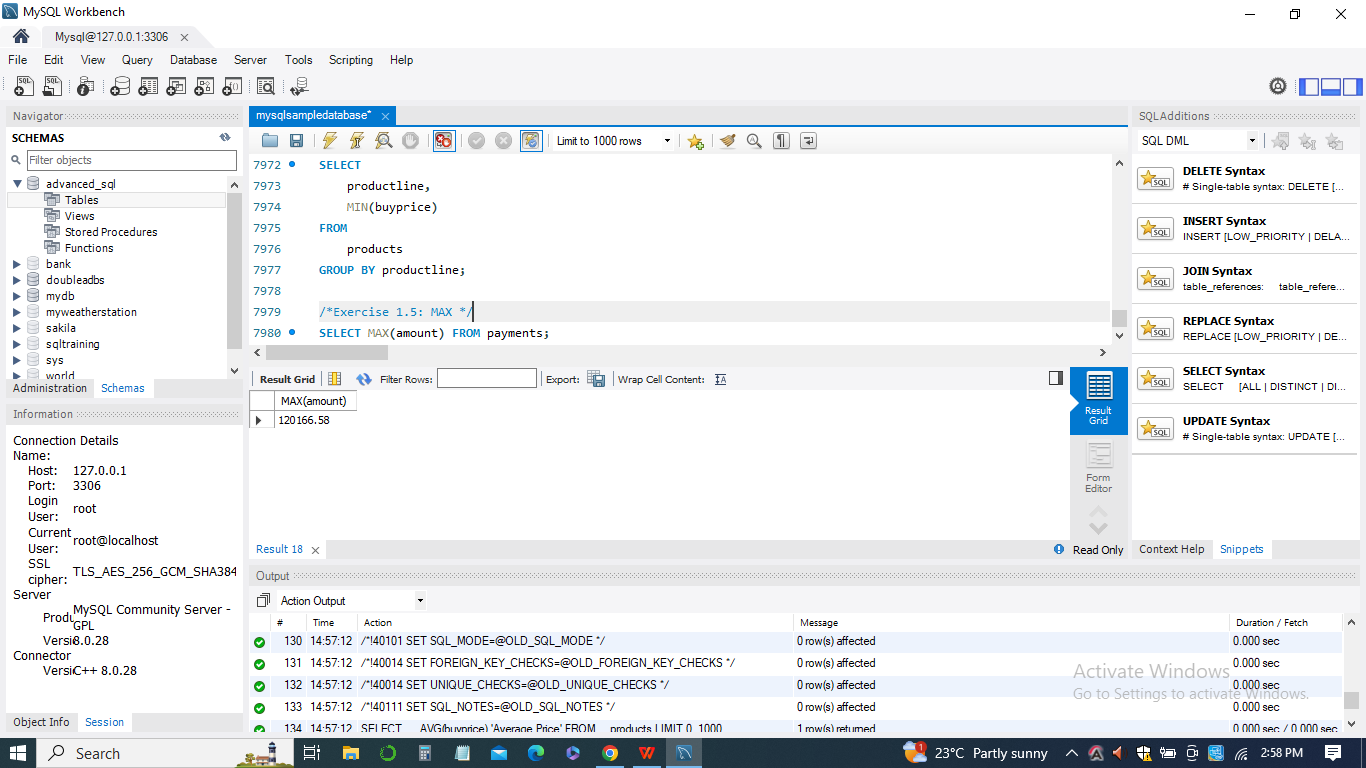
1. *Exercise 1.5: MAX*

The MySQL MAX() function is used to retrieve the maximum value from a specified column in a table. It finds the largest value in the column and returns that value. In the example query:

*SELECT MAX(amount)*

*FROM payments;*

The query calculates the maximum value from the 'amount' column in the 'payments' table.

**

1. *Exercise 1.6: GROUP\_CONCAT*

The MySQL GROUP\_CONCAT() function is used to concatenate multiple values from a column into a single string, where the values are separated by a specified delimiter. It is often used with the GROUP BY clause to aggregate values from multiple rows into a single result.

The CREATE TABLE statement creates a table named t with a single column named v.The INSERT INTO statement inserts multiple rows into the t table with different values for the v column.The query:

*CREATE TABLE t (*

*v CHAR*

*);*

*INSERT INTO t(v) VALUES('A'),('B'),('C'),('B');*

*SELECT*

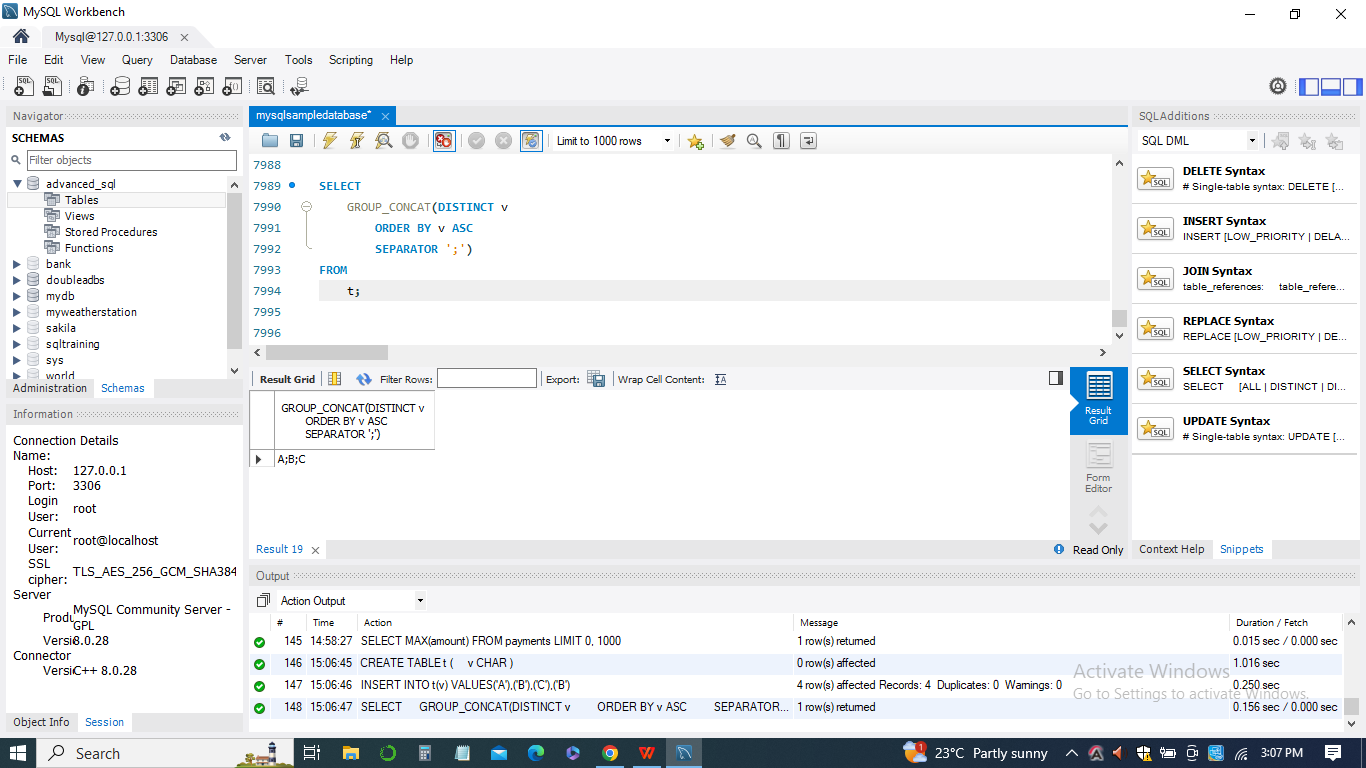
*GROUP\_CONCAT(DISTINCT v*

*ORDER BY v ASC*

*SEPARATOR ';')*

*FROM*

*t;*

**

***Module 2: MySQL string functions***

1. *Exercise 2.1: CONCAT*

The MySQL CONCAT() function is used to concatenate multiple strings together. It takes one or more string values as arguments and combines them into a single string. In in the below examples:

*SELECT CONCAT('MySQL','CONCAT');*

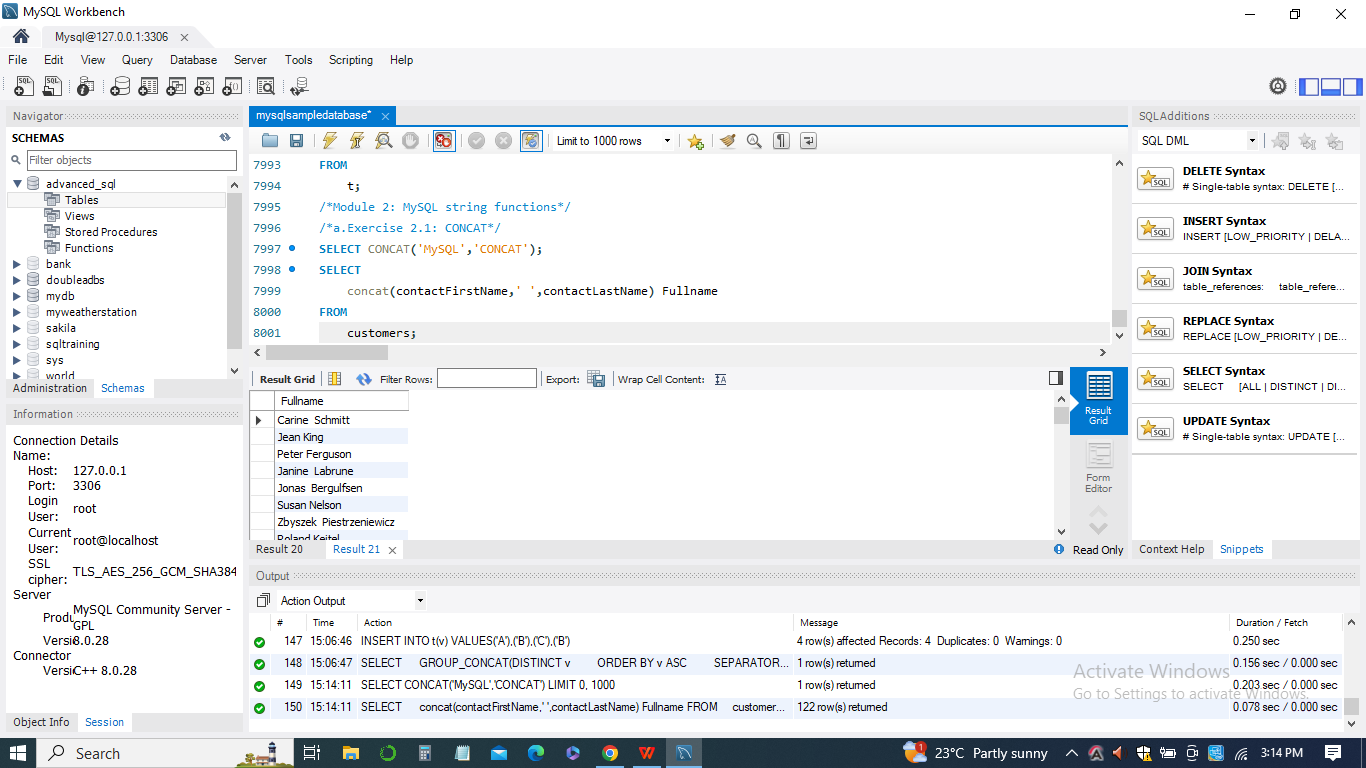
*SELECT*

*concat(contactFirstName,' ',contactLastName) Fullname*

*FROM*

*customers;*

*This query uses the CONCAT() function to concatenate the values of the 'contactFirstName' and 'contactLastName' columns with a space in between. The result will be a new column named 'Fullname' that contains the concatenated full names of the customers from the 'customers' table.*



1. *Exercise 2.2: LENGTH and CHAR\_LENGTH*

In MySQL, there are two functions related to string length: CHAR\_LENGTH() and LENGTH(). These functions provide information about the length of strings, but they have different behaviors based on the character encoding used.

*CHAR\_LENGTH() function:*

The CHAR\_LENGTH() function is used to return the number of characters in a given string. It counts the number of characters based on the specified character set.

*LENGTH() function:*

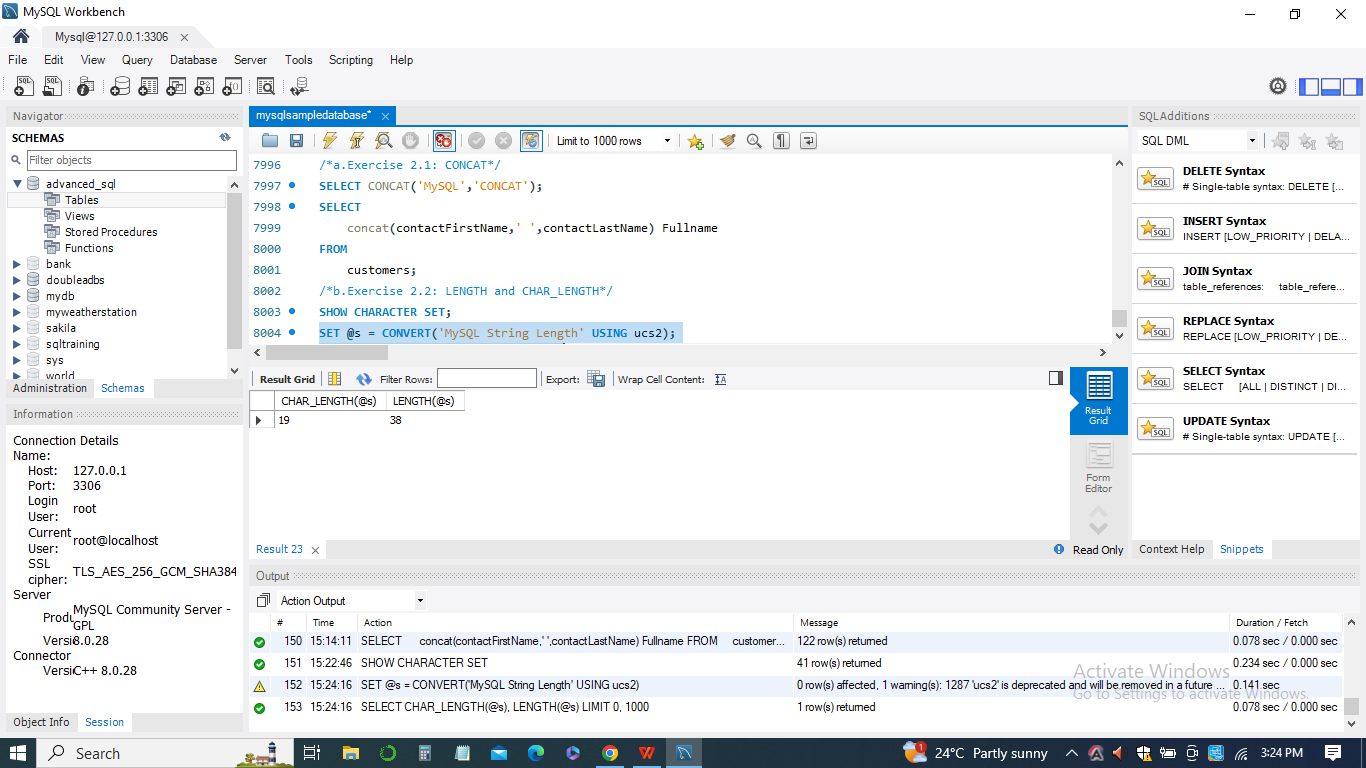
The LENGTH() function returns the number of bytes in a string. It considers each character as a single byte, regardless of the actual character's length in multi-byte character sets.

In the below examples:

*SHOW CHARACTER SET;*

*SET @s = CONVERT('MySQL String Length' USING ucs2);*

*SELECT CHAR\_LENGTH(@s), LENGTH(@s);*



1. *Exercise 2.3: REPLACE*

The MySQL REPLACE() function is used to search for a substring within a given string and replace all occurrences of that substring with another specified string. It's particularly useful when you need to modify or correct data in a text column. In the below examples:

*UPDATE products*

*SET*

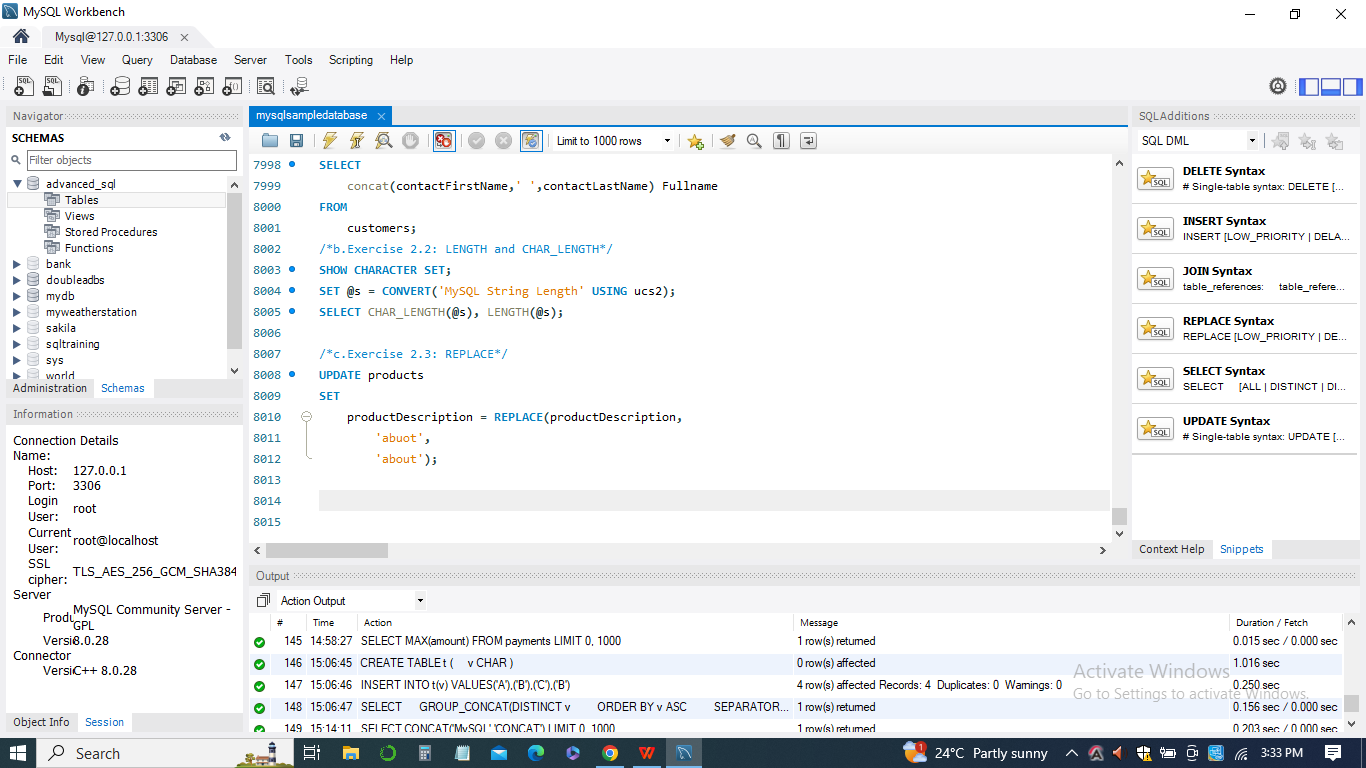
*productDescription = REPLACE(productDescription,*

*'abuot',*

*'about');*

This query performs an update on the 'products' table. It searches for the substring 'abuot' within the 'productDescription' column and replaces all occurrences with the correct spelling 'about'. For instance, if you have a row in the 'products' table with 'productDescription' containing 'This is abuot the product', after running the update query, the value will be updated to 'This is about the product'.

The REPLACE() function is very useful when you need to perform batch updates to correct or modify data in text columns across multiple rows. It can save you time and effort by automatically making changes to large amounts of data in a consistent way.



1. *Exercise 2.4: SUBSTRING*

The MySQL SUBSTRING() function is used to extract a portion of a given string based on the specified starting position and length. It allows you to extract substrings from a string column or a literal string. In the examples:

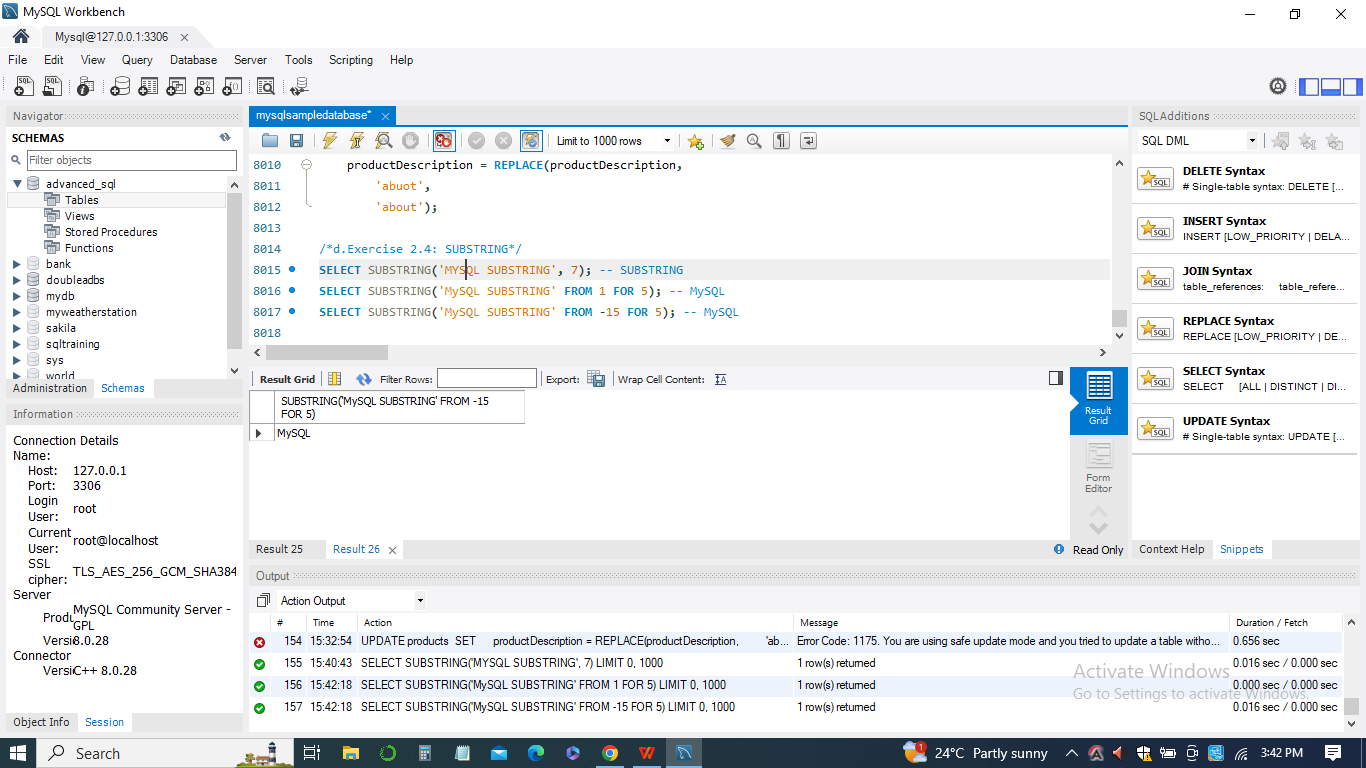
*SELECT SUBSTRING('MYSQL SUBSTRING', 7);*

*SELECT SUBSTRING('MySQL SUBSTRING' FROM 1 FOR 5); -- MySQL*

*SELECT SUBSTRING('MySQL SUBSTRING' FROM -15 FOR 5); -- MySQL*

This query is similar to the previous one, but it uses a negative starting position to count from the end of the string. The negative starting position counts from the end of the string. In this case, it starts 15 characters from the end and extracts the same substring 'MySQL' with a length of 5 characters.

The SUBSTRING() function is useful when you want to extract specific parts of a string, such as words, phrases, or sections, based on their position and length within the string.



1. *Exercise 2.5: LEFT*

The MySQL LEFT() function is used to extract a specified number of characters from the left (start) of a given string. It allows you to retrieve a substring from the beginning of a string up to a certain length.

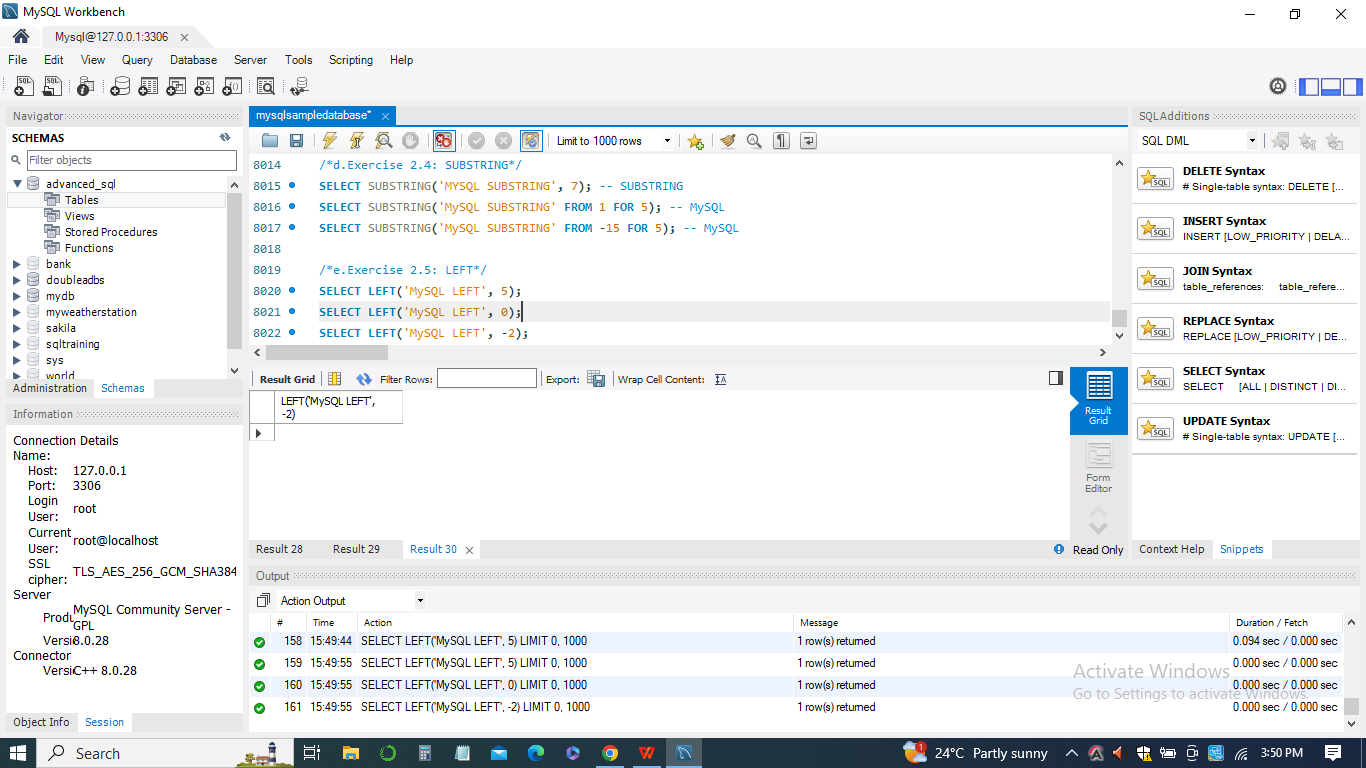
*SELECT LEFT('MySQL LEFT', 5);*

*SELECT LEFT('MySQL LEFT', 0);*

*SELECT LEFT('MySQL LEFT', -2);*

This query uses a negative length (-2) in the LEFT() function to attempt to extract characters from the left of the string 'MySQL LEFT'. MySQL interprets a negative length as 0, resulting in an empty string

The LEFT() function is useful when you want to retrieve a portion of a string starting from the left side of the string, up to a specified length. It's often used for tasks like truncating strings or extracting prefixes from text.



1. *Exercise 2.6: FIND\_IN\_SET*

The MySQL FIND\_IN\_SET() function is used to search for a specified string within a comma-separated list of values. It returns the position (index) of the searched string in the list, or 0 if the string is not found.

*SELECT FIND\_IN\_SET('y','x,y,z'); -- 2*

*SELECT FIND\_IN\_SET('a','x,y,z');*

*CREATE TABLE IF NOT EXISTS divisions (*

*id INT AUTO\_INCREMENT PRIMARY KEY,*

*name VARCHAR(25) NOT NULL,*

*belts VARCHAR(200) NOT NULL*

*);*

*INSERT INTO divisions(name,belts)*

*VALUES ('O-1','white,yellow,orange'),*

*('O-2','purple,green,blue'),*

*('O-3','brown,red,black'),*

*('O-4','white,yellow,orange'),*

*('O-5','purple,green,blue'),*

*('O-6','brown,red'),*

*('O-7','black'),*

*('O-8','white,yellow,orange'),*

*('O-9','purple,green,blue'),*

*('O-10','brown,red');*

*SELECT*

*name,*

*belts*

*FROM*

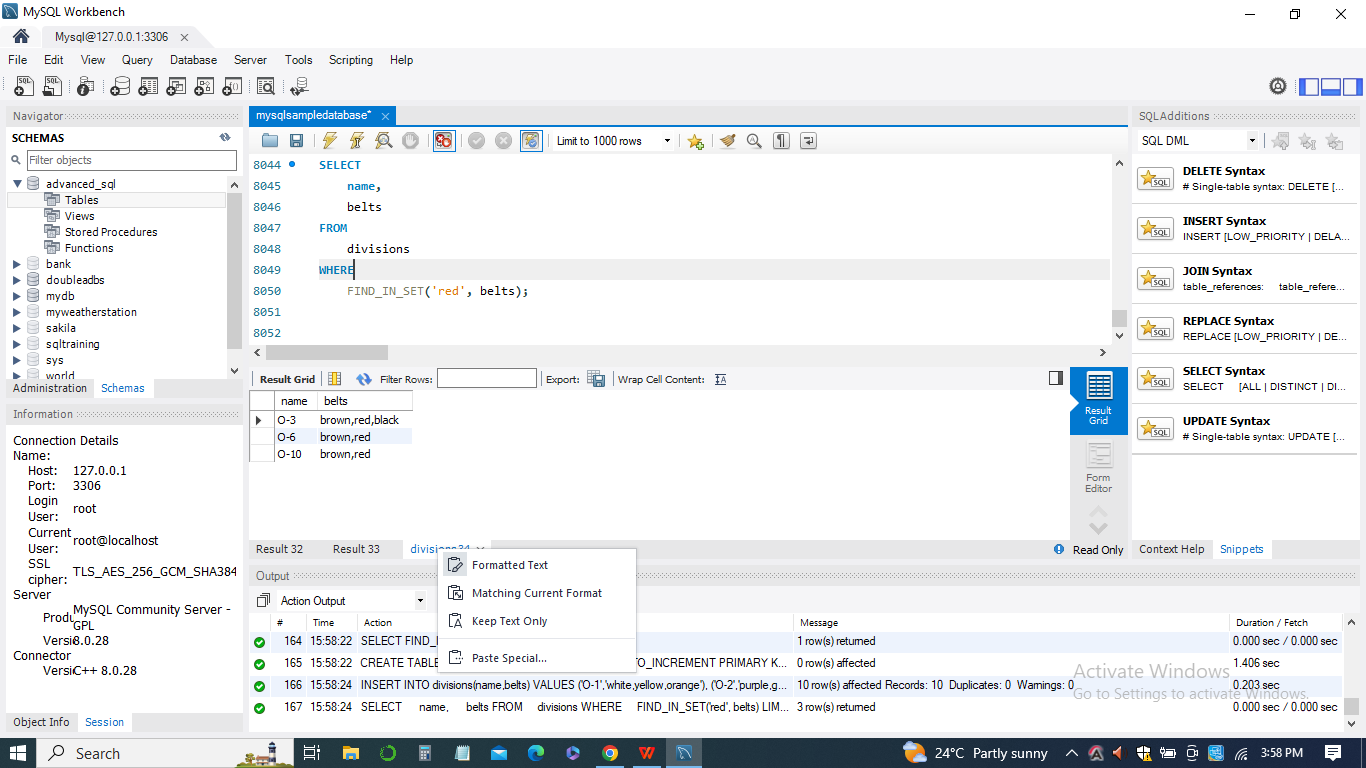
*divisions*

*WHERE*

*FIND\_IN\_SET('red', belts);*

This query selects the 'name' and 'belts' columns from the 'divisions' table for rows where the 'belts' column contains the string 'red'. It uses the FIND\_IN\_SET() function to perform the search within the comma-separated list of belts.

Assuming the 'belts' column contains values like 'white,yellow,orange', 'purple,green,blue', etc., the result will include rows where 'red' is found in the 'belts' list. The FIND\_IN\_SET() function is particularly useful when you have data stored as comma-separated values, such as tags, categories, or options, and you need to query based on specific values within those lists.



1. *Exercise 2.7: TRIM*

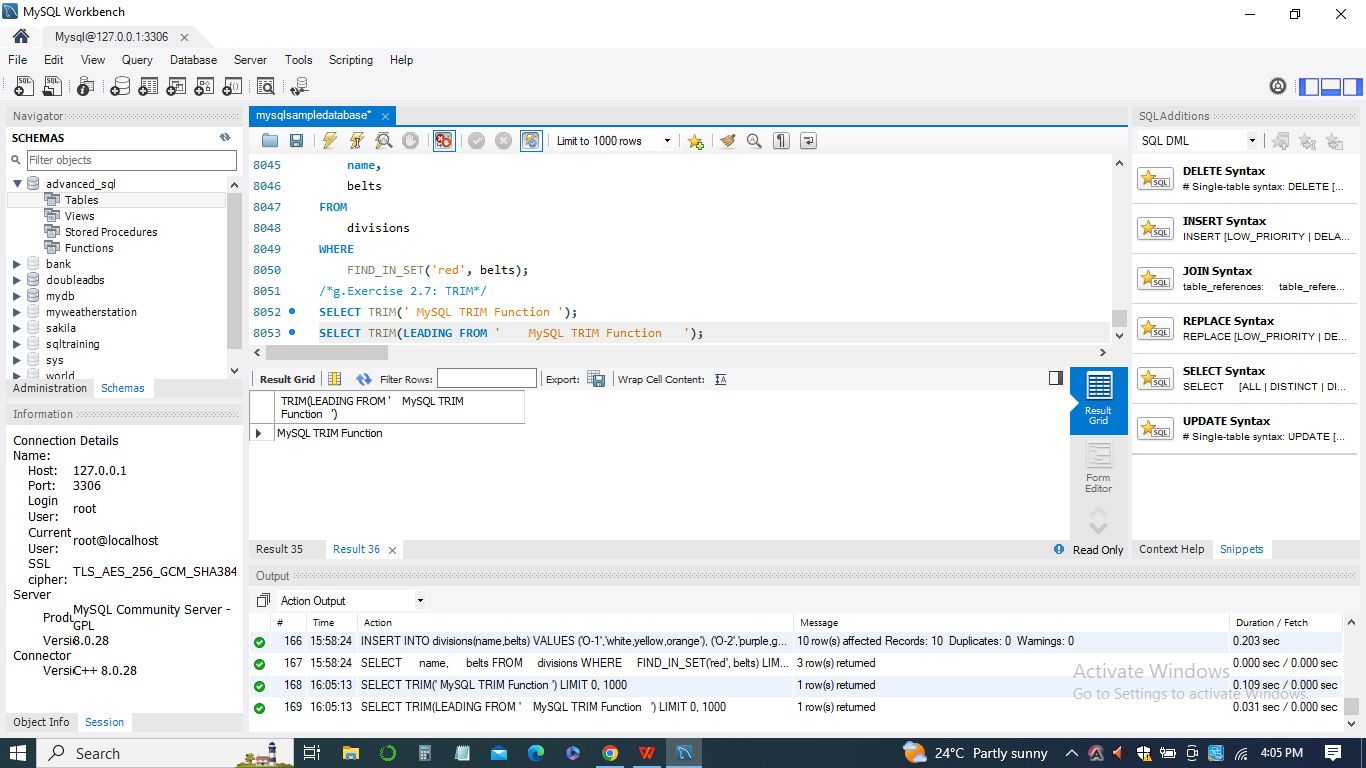
The MySQL TRIM() function is used to remove specified characters or spaces from the beginning, end, or both sides of a string. It helps to clean up and format text data by removing unwanted leading and trailing characters.

*SELECT TRIM(' MySQL TRIM Function ');*

*SELECT TRIM(LEADING FROM ' MySQL TRIM Function ');*

The result is the same as the first query. However, in this case, only leading spaces are removed.

The TRIM() function is useful when you want to remove unnecessary spaces or specific characters from the beginning, end, or both ends of a string. It's often used to clean up data before performing comparisons or storing it in a database.



1. *Exercise 2.8: INSTR*

The MySQL INSTR() function is used to find the position (index) of a substring within a given string. It returns the position of the first occurrence of the substring in the string, or 0 if the substring is not found.

*SELECT INSTR('MySQL INSTR', 'MySQL');*

*SELECT INSTR('MySQL INSTR', 'mysql');*

*EXPLAIN SELECT*

*productname*

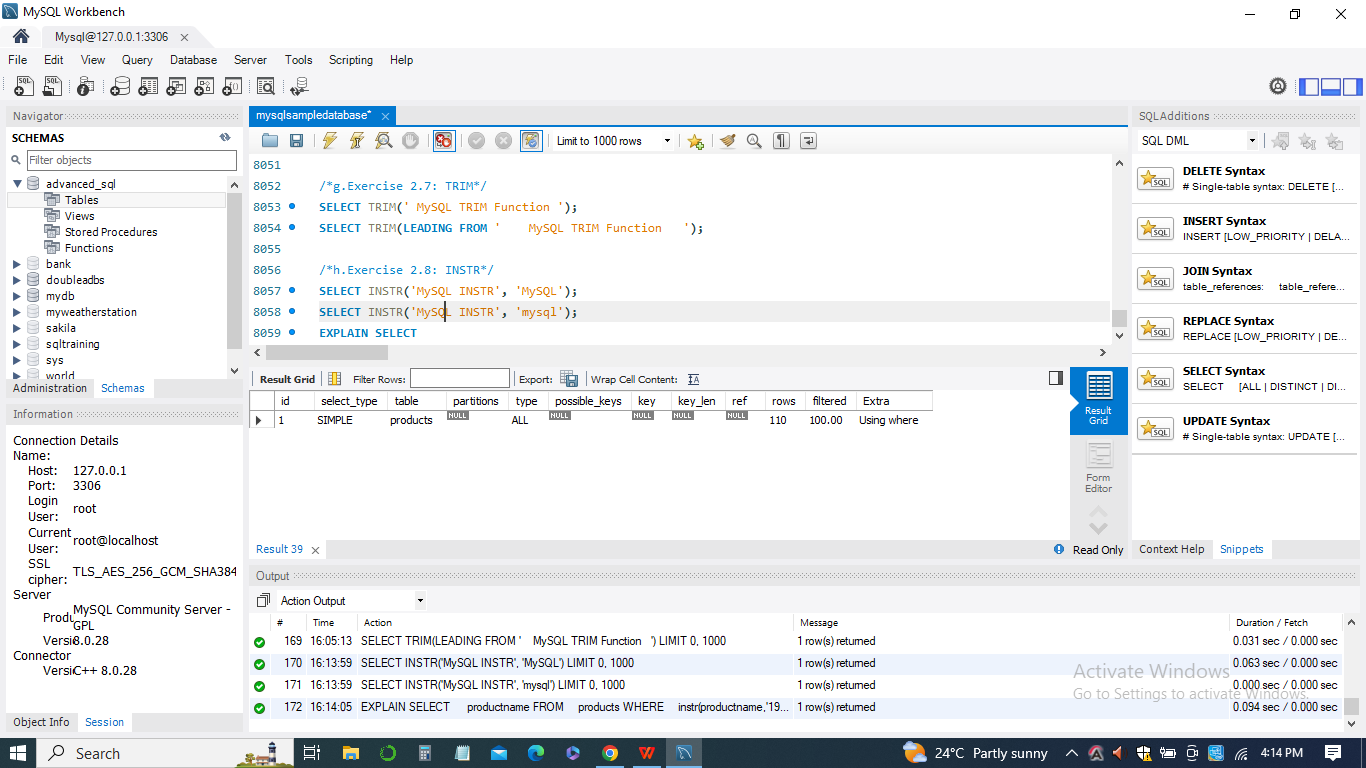
*FROM*

*products*

*WHERE*

*instr(productname,'1900');*

This query uses the INSTR() function within a SELECT statement to search for the substring '1900' within the 'productname' column of the 'products' table. The EXPLAIN keyword is used to analyze how the query will be executed. The INSTR() function can be helpful when you need to locate specific substrings within text data or perform searches within larger strings, such as descriptions or titles.



***Module 3: MySQL date , time and comparison functions***

1. *Exercise 3.1: DATEDIF*

The MySQL DATEDIFF() function is used to calculate the difference in days between two date values. It returns the number of days between the specified dates, considering the start date and end date.

*SELECT DATEDIFF('2011-08-17', '2011-08-17'); -- 0 day*

*SELECT*

*orderNumber,*

*DATEDIFF(requiredDate, shippedDate) daysLeft*

*FROM*

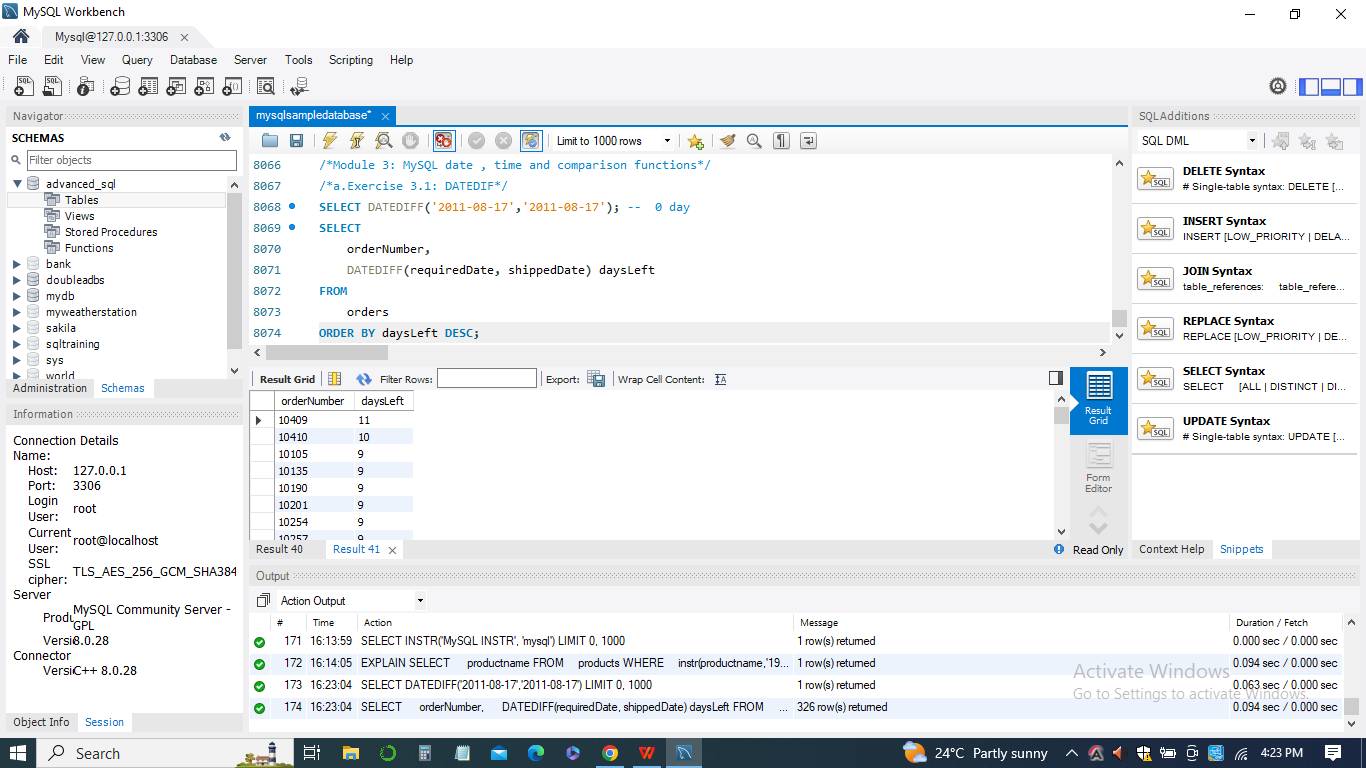
*orders*

*ORDER BY daysLeft DESC;*

This query calculates the difference in days between the 'requiredDate' and 'shippedDate' columns of the 'orders' table for each order. The calculated difference is assigned the alias 'daysLeft'. The results are ordered in descending order based on the calculated difference.

The result will include a list of order numbers along with the number of days left between the required date and the shipped date for each order. Orders with the longest delay will appear first in the result set.

The DATEDIFF() function is commonly used when you need to calculate the time duration between two date values, such as measuring the time elapsed between order placement and order shipment.



1. *Exercise 3.2: DATE\_FORMAT*

The MySQL DATE\_FORMAT() function is used to format a date value into a specific string representation according to a specified format string. It allows you to customize how dates are displayed in the result set.

*SELECT*

*orderNumber,*

*DATE\_FORMAT(orderdate, '%Y-%m-%d') orderDate,*

*DATE\_FORMAT(requireddate, '%a %D %b %Y') requireddate,*

*DATE\_FORMAT(shippedDate, '%W %D %M %Y') shippedDate*

*FROM*

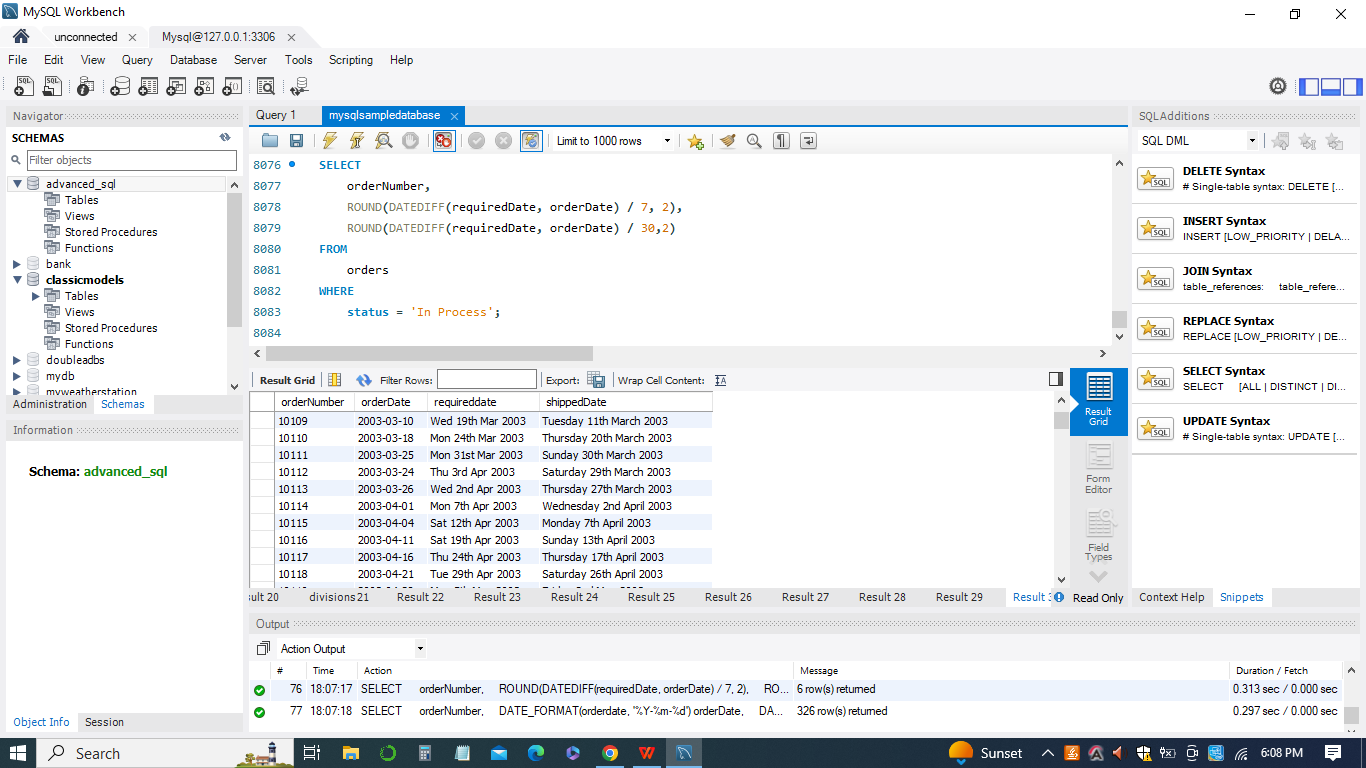
*orders;*

This query retrieves data from the 'orders' table and uses the DATE\_FORMAT() function to format the date values in different ways. DATE\_FORMAT(orderdate, '%Y-%m-%d'): This formats the 'orderdate' column using the format '%Y-%m-%d', which represents the year, month, and day in the 'YYYY-MM-DD' format.

DATE\_FORMAT(requireddate, '%a %D %b %Y'): This formats the 'requireddate' column using the format '%a %D %b %Y', which represents the abbreviated weekday name, day of the month with ordinal suffix, abbreviated month name, and year.

DATE\_FORMAT(shippedDate, '%W %D %M %Y'): This formats the 'shippedDate' column using the format '%W %D %M %Y', which represents the full weekday name, day of the month with ordinal suffix, full month name, and year.

The result set will display the formatted dates according to the specified formats for each order, providing a clear and human-readable representation of the date values. This function is useful when you want to present date information to users in a specific format.



1. *Exercise 3.3: STR\_TO\_DATE*

The MySQL STR\_TO\_DATE() function is used to convert a string representation of a date or time into a MySQL date or datetime value. It requires two arguments: the input string and a format string that defines how the input string is structured.

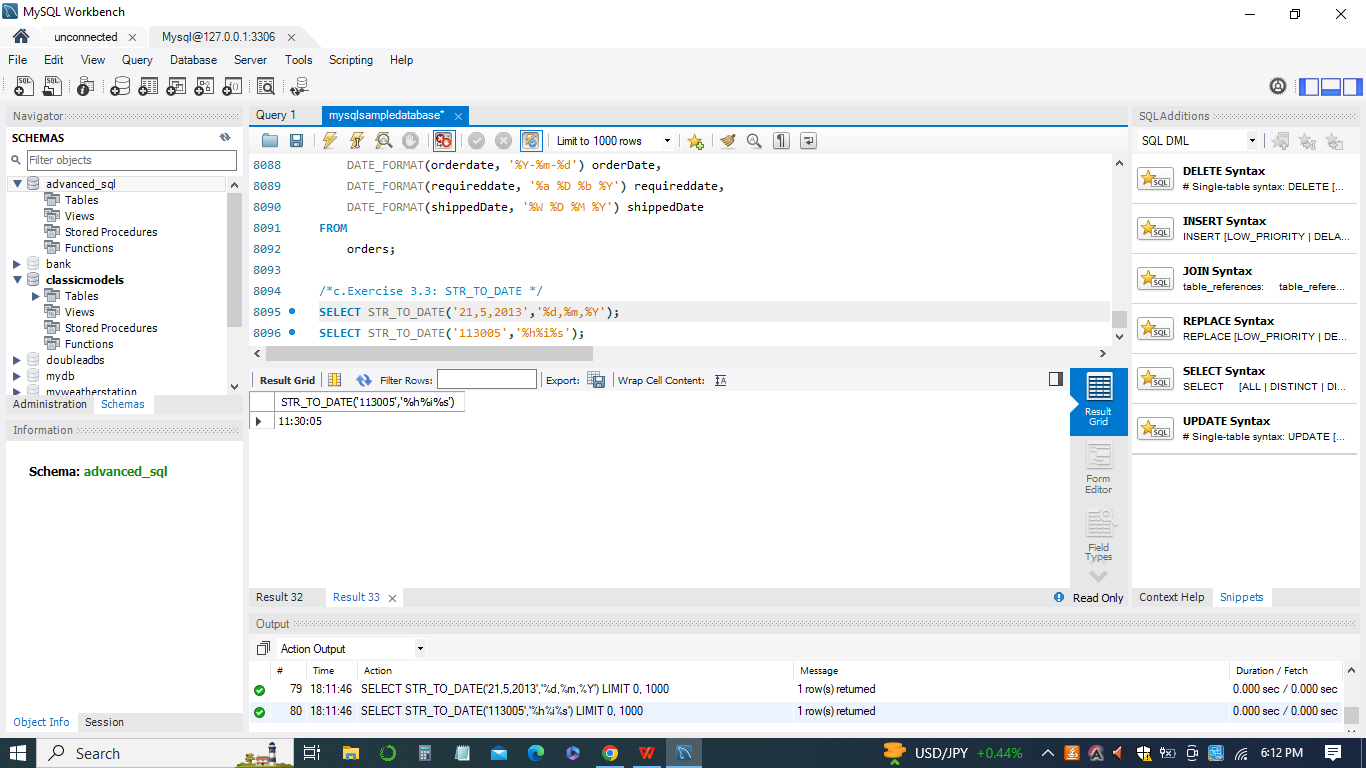
*SELECT STR\_TO\_DATE('21,5,2013','%d,%m,%Y');*

*SELECT STR\_TO\_DATE('113005','%h%i%s');*

This query uses the STR\_TO\_DATE() function to convert the input string '113005' into a MySQL time value. The format string '%h%i%s' indicates that the input string is in the format hour, minute, second.

The result is the time '11:30:05', which corresponds to 11 hours, 30 minutes, and 5 seconds.

The STR\_TO\_DATE() function is useful when you need to convert date and time information stored as strings into MySQL's native date and time data types for proper storage and manipulation.



1. *Exercise 3.4: NOW*

The MySQL NOW() function is used to retrieve the current date and time as a MySQL datetime value. It's commonly used to record the timestamp when a record is created or modified.

*-- mysql now minus 1 hour*

*SELECT (NOW() - INTERVAL 1 HOUR) 'NOW - 1 hour',*

*NOW(),*

*-- mysql now plus 1 hour*

*NOW() + INTERVAL 1 HOUR 'NOW + 1 hour';*

*CREATE TABLE tmp(*

*id INT PRIMARY KEY AUTO\_INCREMENT,*

*title VARCHAR(255) NOT NULL,*

*created\_on DATETIME NOT NULL DEFAULT NOW() -- or CURRENT\_TIMESTAMP*

*);*

*INSERT INTO tmp(title)*

*VALUES('Test NOW() function');*

*SELECT \* FROM tmp;*

This query retrieves all records from the 'tmp' table, including the 'id', 'title', and 'created\_on' columns. The 'created\_on' column will display the timestamp when each record was inserted. The NOW() function is commonly used in scenarios where you need to track the creation or modification time of records in your database.

1. *Exercise 3.5: GREATEST and LEAST*

The MySQL GREATEST() and LEAST() functions are used to find the maximum and minimum values among a list of expressions or values, respectively. They are often used to determine the greatest or smallest value from a set of inputs.

*SELECT GREATEST(10, 20, 30), -- 30*

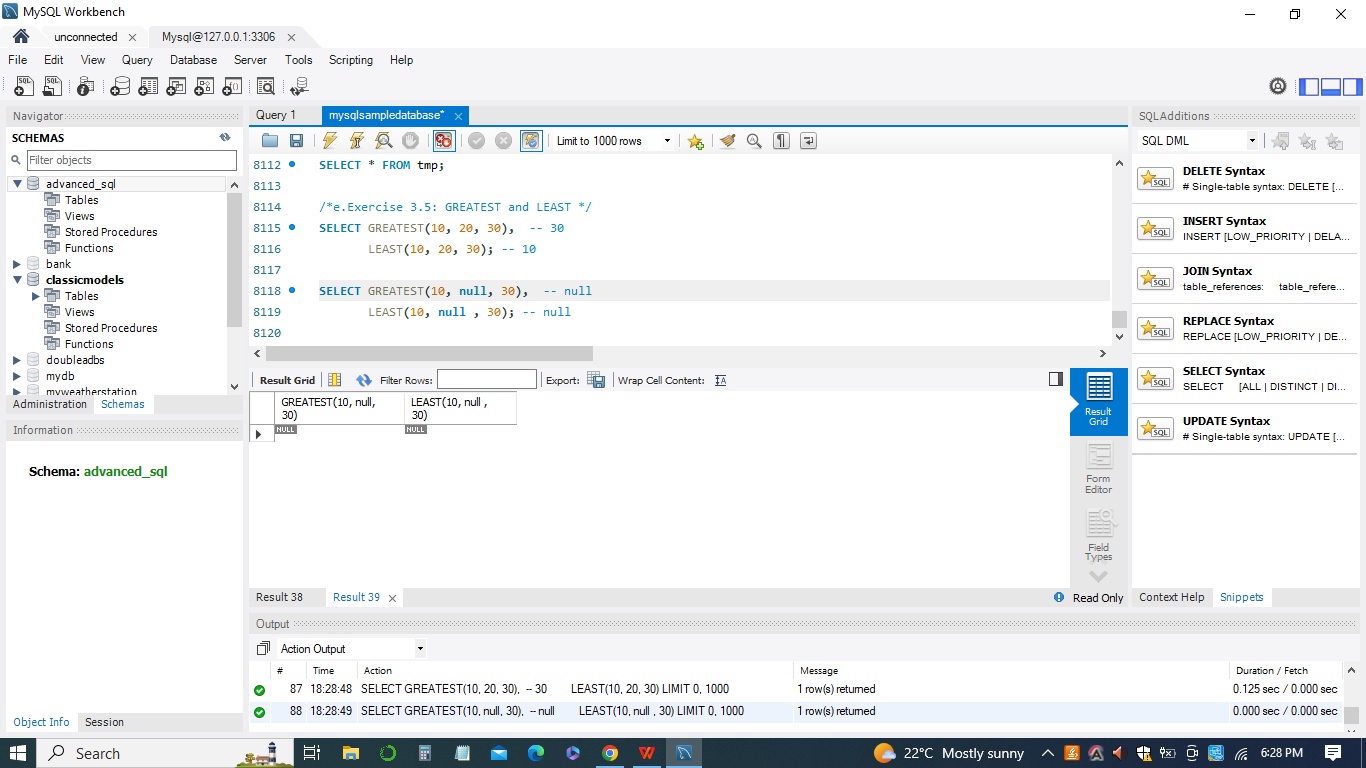
*LEAST(10, 20, 30); -- 10*

*SELECT GREATEST(10, null, 30), -- null*

*LEAST(10, null , 30); -- null*

This query demonstrates how the GREATEST() and LEAST() functions handle NULL values. If any of the input values is NULL, the GREATEST() function returns NULL because it's impossible to determine the greatest value when one of the inputs is unknown. The LEAST() function also returns NULL when any of the input values is NULL.

These functions are useful for comparing values within SQL queries and determining the maximum and minimum values among a set of expressions or columns. Just keep in mind that when NULL values are involved, the result may also be NULL.



1. *Exercise 3.6: ISNULL*

The MySQL ISNULL() function is used to test whether an expression or value is NULL. It returns 1 if the expression is NULL, and 0 if the expression is not NULL.

*CREATE TABLE special\_isnull (*

*start\_date DATE NOT NULL*

*);*

*INSERT INTO special\_isnull(start\_date)*

*VALUES('2000-01-01'),*

*('0000-00-00');*

*SELECT*

*\**

*FROM*

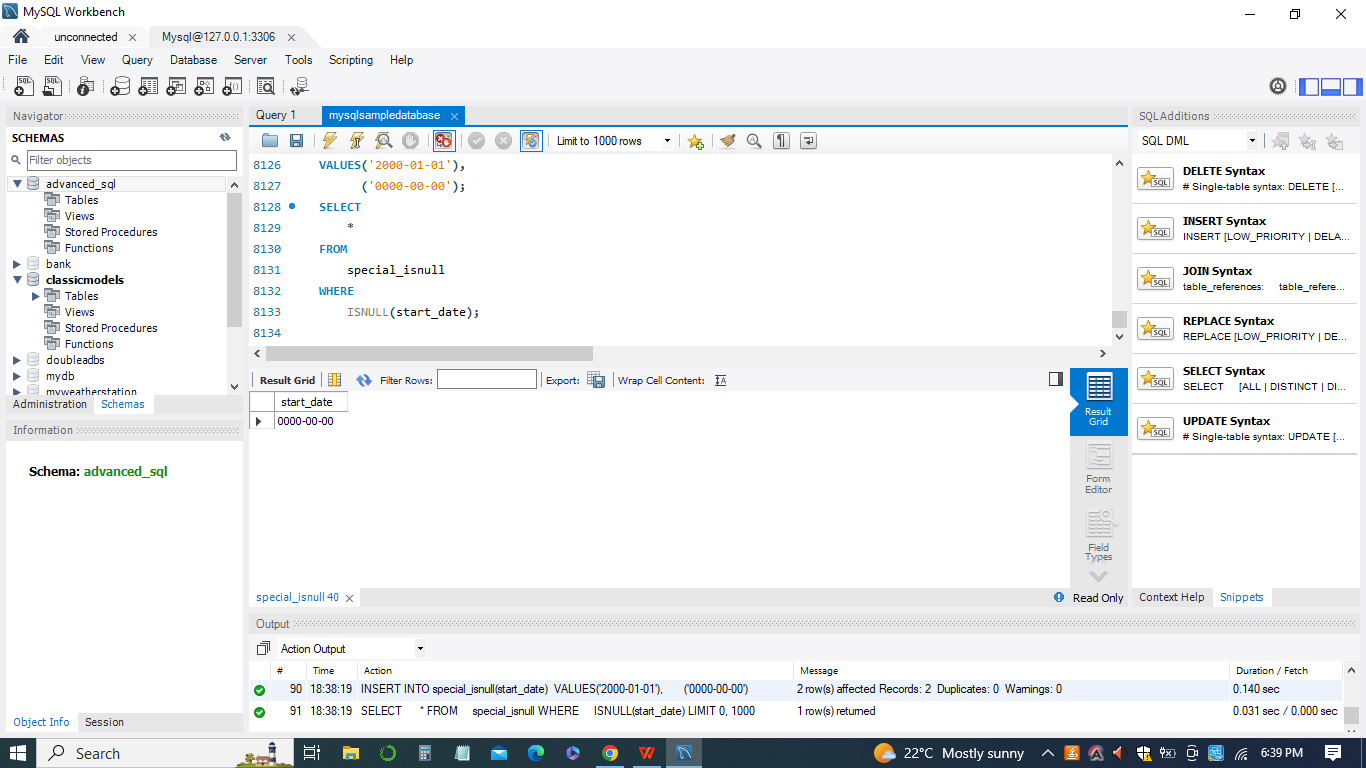
*special\_isnull*

*WHERE*

*ISNULL(start\_date);*

This query retrieves all records from the 'special\_isnull' table where the 'start\_date' column is NULL. However, in this case, it's important to note that the 'start\_date' column is defined as NOT NULL, so it cannot contain NULL values. Therefore, even though the query is checking for NULL values, it will not return any rows because the 'start\_date' column cannot hold NULL values according to its definition.

The ISNULL() function is generally used to check for NULL values in columns that allow NULL, not in columns defined as NOT NULL. In your provided example, due to the NOT NULL constraint on the 'start\_date' column, the result will be an empty result set.



***Module 4: Introduction to database views, MySQL views and create views***

1. *Exercise 4.1: Creating Views in MySQL.*

The MySQL CREATE VIEW statement is used to create a virtual table that is based on the result of a SELECT query. Views allow you to abstract complex queries into a simplified and reusable form, making it easier to query and analyze data.

*CREATE VIEW salePerOrder AS*

*SELECT*

*orderNumber,*

*SUM(quantityOrdered \* priceEach) total*

*FROM*

*orderDetails*

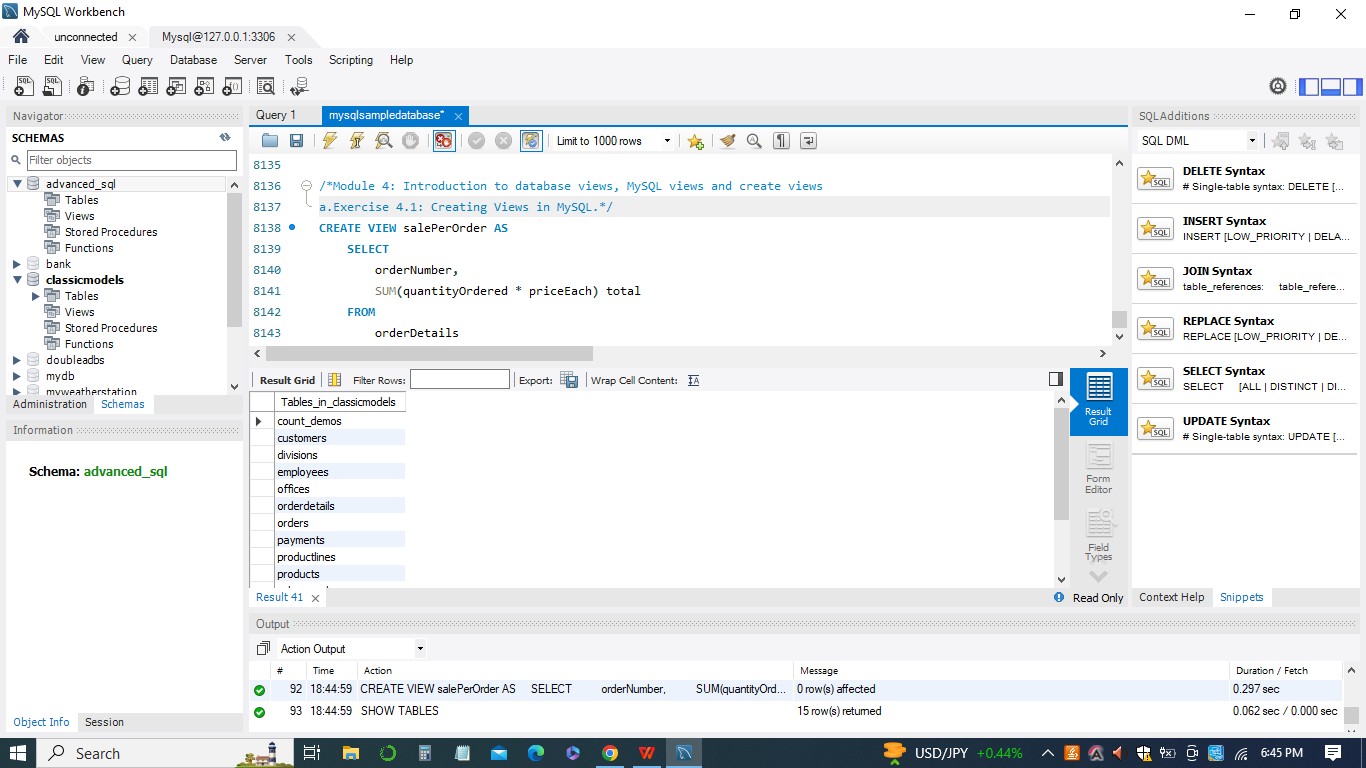
*GROUP by orderNumber*

*ORDER BY total DESC;*

*SHOW TABLES;*

This query creates a view named 'salePerOrder'. The view retrieves data from the 'orderDetails' table and calculates the total sales amount for each order. It uses the SUM() function to calculate the sum of the product of 'quantityOrdered' and 'priceEach' columns for each order. The result is grouped by the 'orderNumber' column and ordered in descending order by the total sales amount.

This command will show a list of full tables and views in the database, along with their types. The 'salePerOrder' view you created using the CREATE VIEW statement acts as a virtual table that stores the results of the query. You can query this view to retrieve the total sales amount for each order without needing to repeat the complex query logic each time. It provides a convenient way to access summarized data.



***Module 5: Creating updatable Views, Ensuring Views Consistency and Managing***

***Views***

1. *Exercise 5.1: How to create an updatable view and update data in the underlying tablethrough the view*

MySQL Updatable Views allow you to perform certain UPDATE operations on the view itself, which will update the underlying data in the base table. However, there are certain conditions that need to be met for a view to be updatable.

*CREATE VIEW officeInfo*

*AS*

*SELECT officeCode, phone, city*

*FROM offices;*

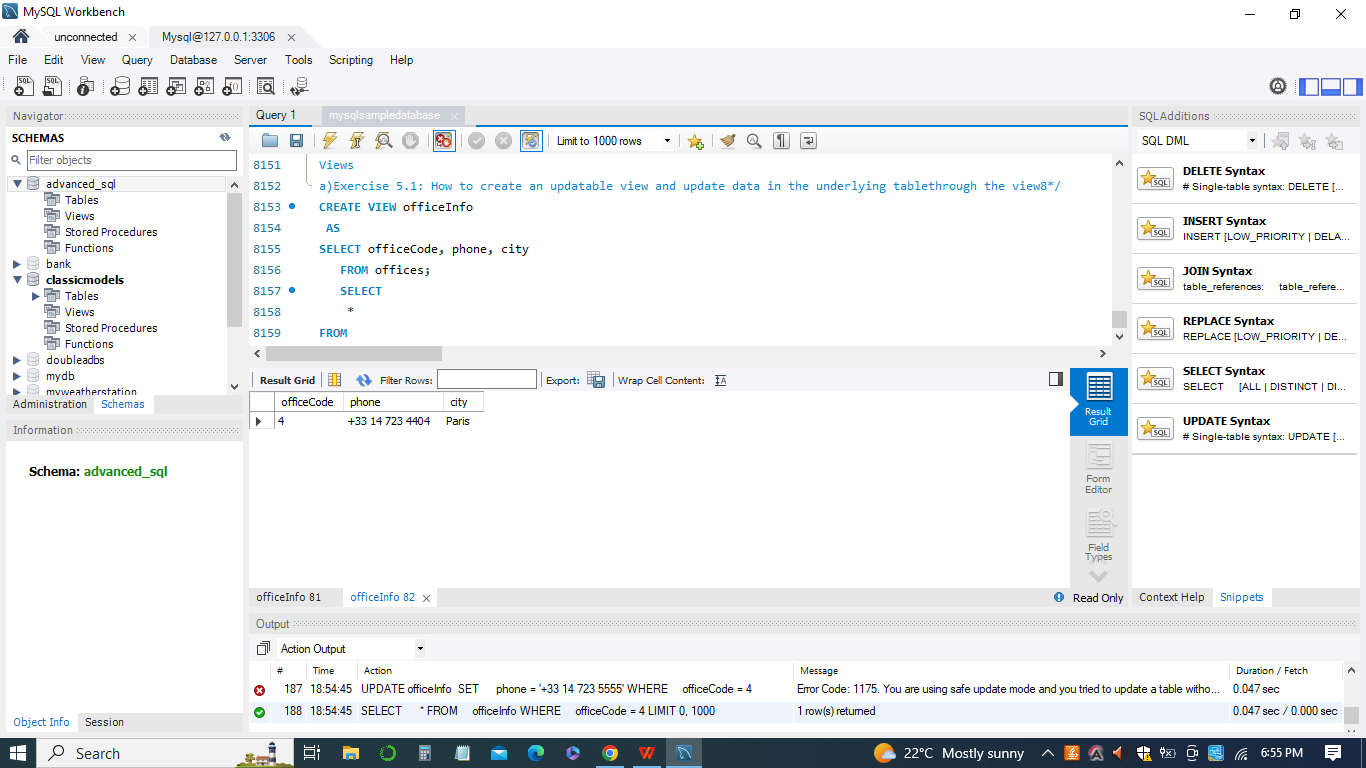
*UPDATE officeInfo*

*SET*

*phone = '+33 14 723 5555'*

*WHERE*

*officeCode = 4;*



1. *Exercise 5.2: Ensuring Views Consistency Using WITH CHECK OPTION Clause*

The WITH CHECK OPTION clause in MySQL views is used to enforce that any data modification made through the view adheres to the conditions specified in the view's WHERE clause. In other words, it ensures that only rows satisfying the view's condition can be modified through the view.

*CREATE OR REPLACE VIEW vps AS*

*SELECT*

*employeeNumber,*

*lastname,*

*firstname,*

*jobtitle,*

*extension,*

*email,*

*officeCode,*

*reportsTo*

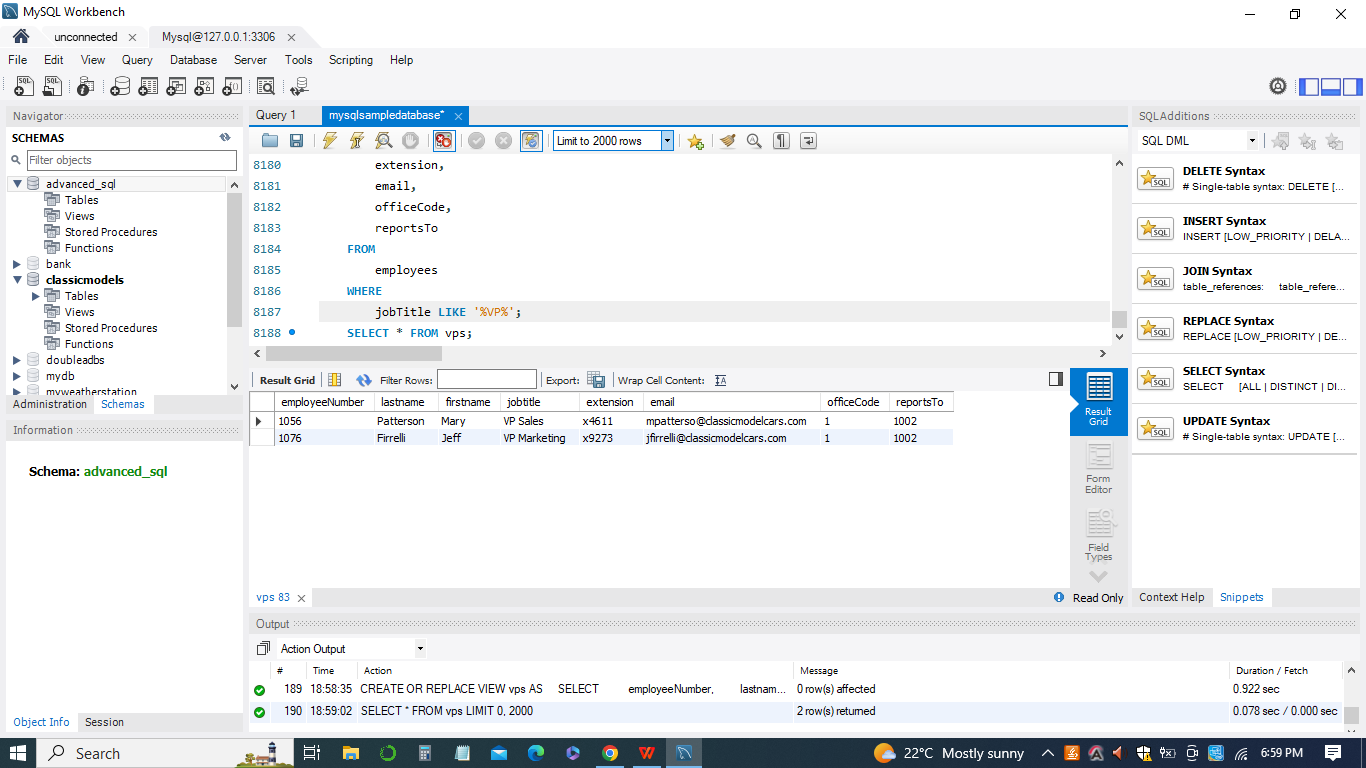
*FROM*

*employees*

*WHERE*

*jobTitle LIKE '%VP%';*

*SELECT \* FROM vps;*



1. *Exercise 5.3: How to manage views in MySQL including displaying, modifying and removing views*

In MySQL, views can be created, altered, and managed to provide a more convenient and efficient way to query data. Your example illustrates the process of creating a view, altering its structure, and then querying it:

*CREATE VIEW organization AS*

*SELECT*

*CONCAT(E.lastname, E.firstname) AS Employee,*

*CONCAT(M.lastname, M.firstname) AS Manager*

*FROM*

*employees AS E*

*INNER JOIN*

*employees AS M ON M.employeeNumber = E.ReportsTo*

*ORDER BY Manager;*

*SHOW CREATE VIEW organization;*

*ALTER VIEW organization*

*AS*

*SELECT CONCAT(E.lastname,E.firstname) AS Employee,*

*E.email AS employeeEmail,*

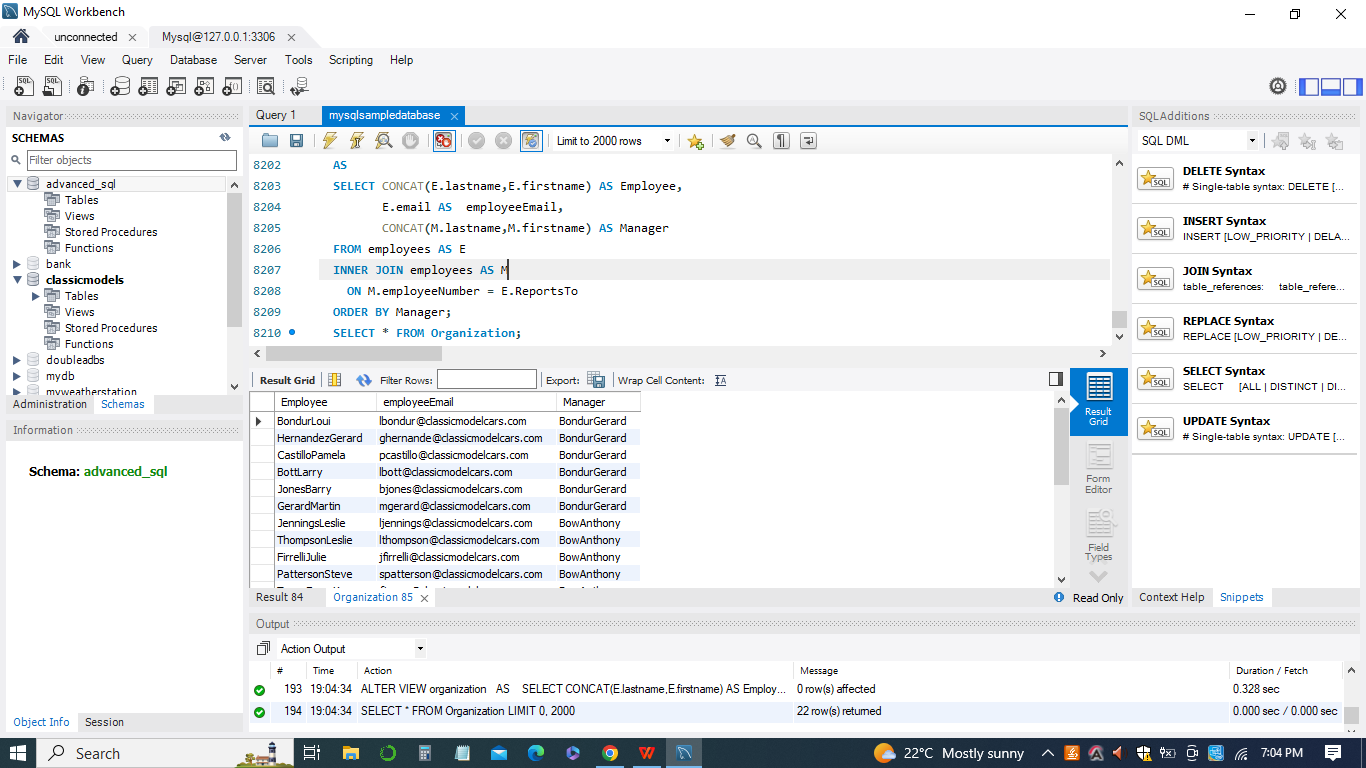
*CONCAT(M.lastname,M.firstname) AS Manager*

*FROM employees AS E*

*INNER JOIN employees AS M*

*ON M.employeeNumber = E.ReportsTo*

*ORDER BY Manager;*



***Module 6: MySQL Stored Procedures Introduction***

1. *Exercise 6.1: How to develop the first MySQL stored procedure using CREATE PROCEDURE statement*

In MySQL, a stored procedure is a named group of SQL statements that can be executed as a single unit. Stored procedures provide a way to encapsulate business logic or specific tasks and can be called with specific parameters.

*SELECT \* FROM products;*

*DELIMITER //*

*CREATE PROCEDURE GetAllProducts()*

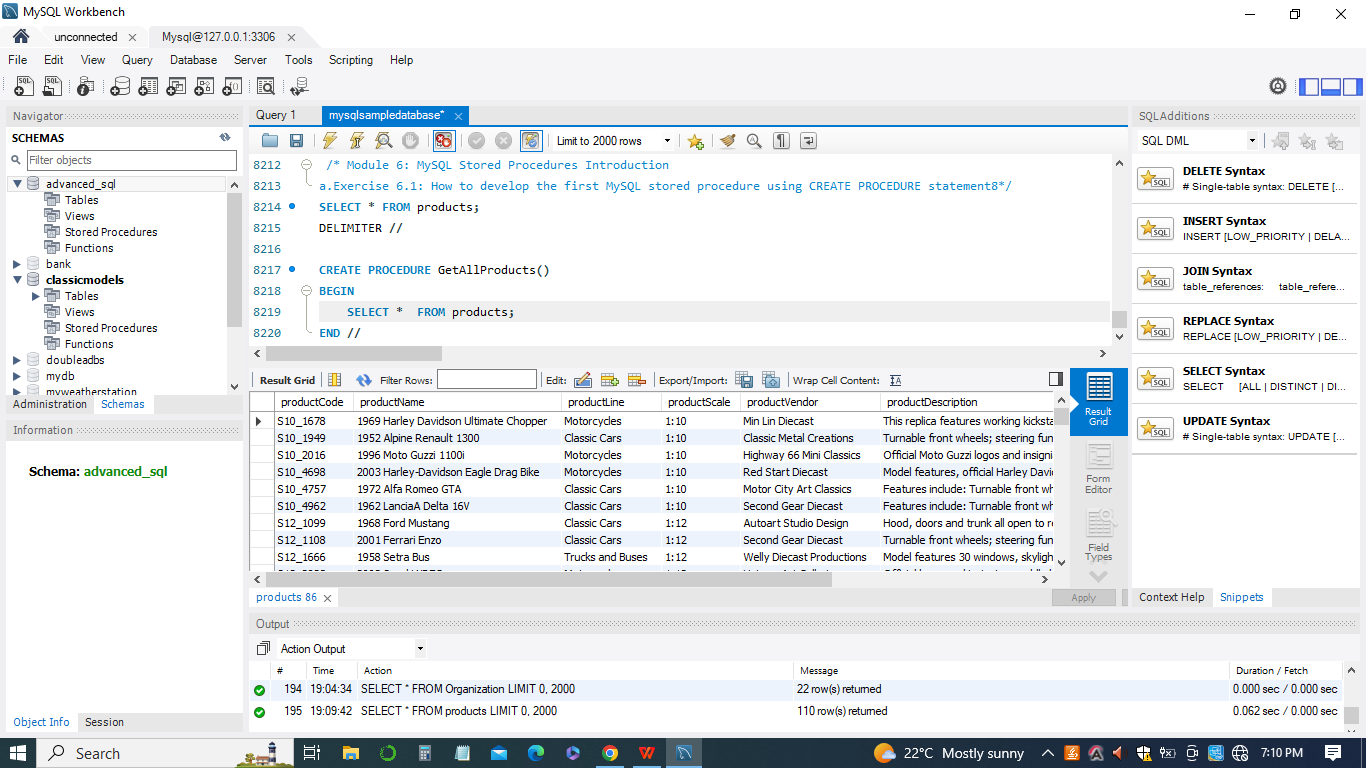
*BEGIN*

*SELECT \* FROM products;*

*END //*

*DELIMITER ;*

The DELIMITER command is used to change the delimiter from the default semicolon (;) to a custom one (in this case, //). This is necessary because you're defining a stored procedure that contains multiple SQL statements, and the semicolon is used to separate individual statements. By changing the delimiter, MySQL knows that the stored procedure definition will continue until the new delimiter is encountered. Calling the stored procedure will execute the SQL statement inside it, which in this case is the SELECT query to retrieve all products from the 'products' table.



1. *Exercise 6.2: How to declare, and use variables in the stored procedure. In addition, you will learn about the scopes of variables.*

In MySQL stored procedures, variables can be declared and used to store temporary values. These variables allow you to perform calculations, store results, and control the flow of your procedure. Your example demonstrates the use of variables in a stored procedure:

*DECLARE productCount INT DEFAULT 0;*

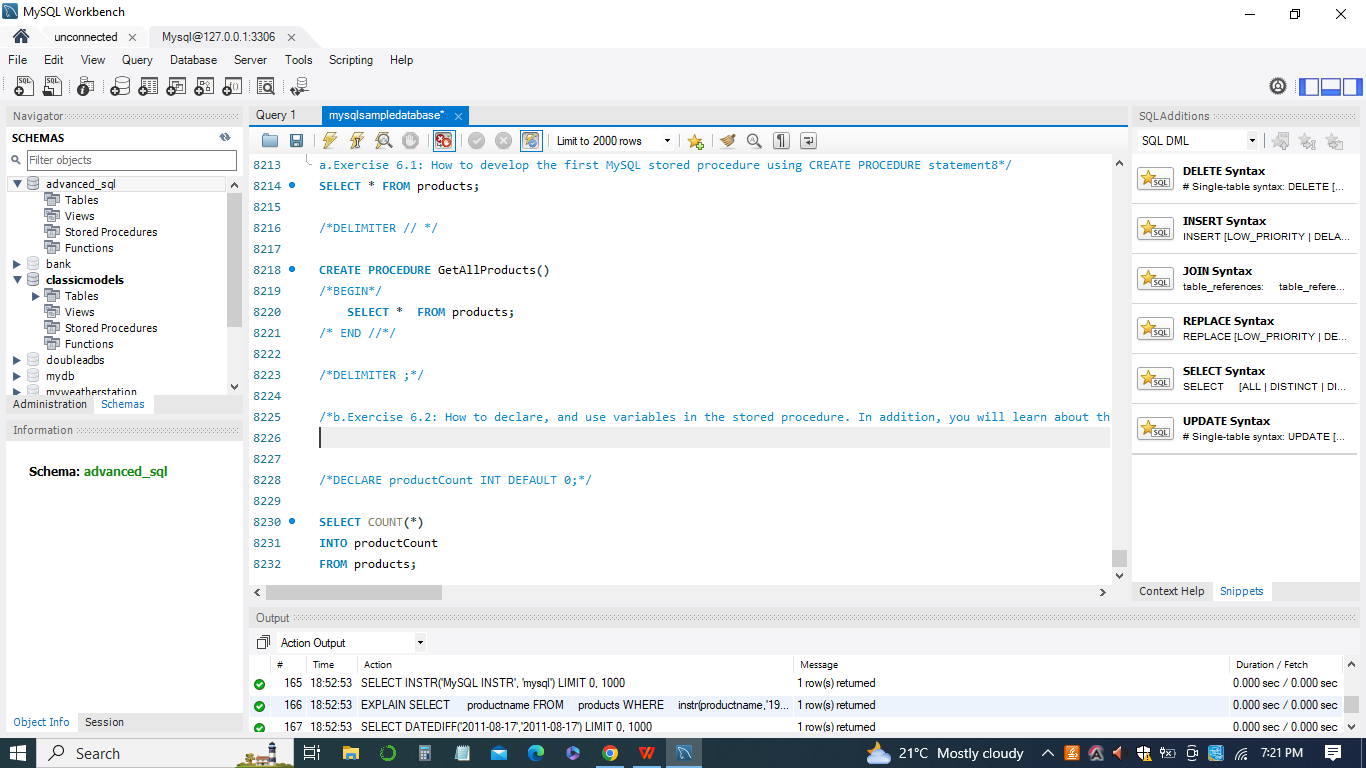
*SELECT COUNT(\*)*

*INTO productCount*

*FROM products;*

This query calculates the total number of rows in the 'products' table using the COUNT(\*) function. The result is assigned to the productCount variable using the INTO clause.

Variables are an important feature in stored procedures, as they allow you to store and manipulate data within the procedure's scope. They can be used in various ways, such as storing query results, intermediate calculation results, or control values.



1. *Exercise 6.3: how to write MySQL stored procedures that have parameters*

In MySQL stored procedures, you can define parameters that allow you to pass values into the procedure when it's called. Your example demonstrates the creation and usage of a stored procedure with parameters:

*CREATE PROCEDURE GetOfficeByCountry(*

*IN countryName VARCHAR(255)*

*)*

*-- BEGIN*

*SELECT \**

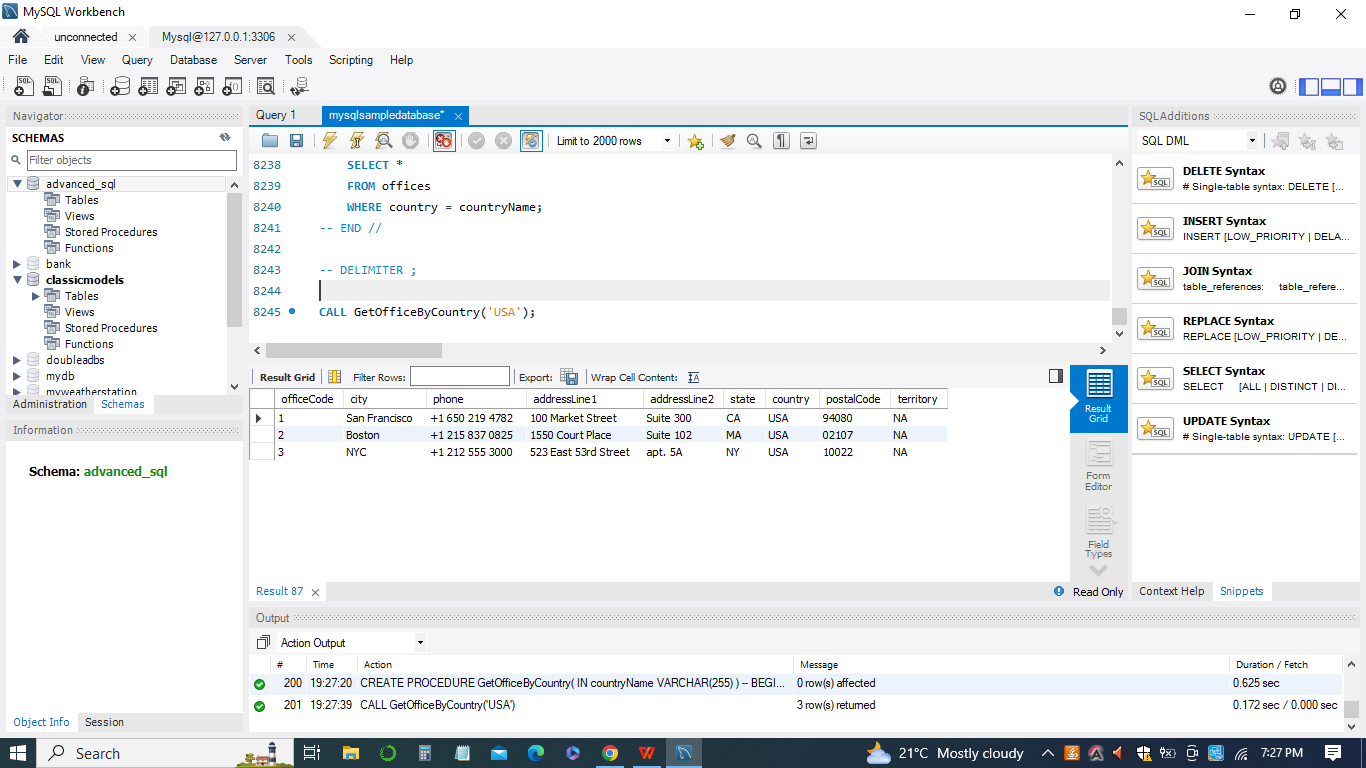
*FROM offices*

*WHERE country = countryName;*

*-- END //*

*-- DELIMITER ;*

*CALL GetOfficeByCountry('USA');*



***Module 7: MySQL Stored Procedure Parameters and Return Multiple Values***

1. *Exercise 7.1: How to Return Multiple Values in stored procedure*

*DELIMITER $$*

*CREATE PROCEDURE get\_order\_by\_cust(*

*IN cust\_no INT,*

*OUT shipped INT,*

*OUT canceled INT,*

*OUT resolved INT,*

*OUT disputed INT)*

*BEGIN*

*-- shipped*

*SELECT*

*count(\*) INTO shipped*

*FROM*

*orders*

*WHERE*

*customerNumber = cust\_no*

*AND status = 'Shipped';*

*-- canceled*

*SELECT*

*count(\*) INTO canceled*

*FROM*

*orders*

*WHERE*

*customerNumber = cust\_no*

*AND status = 'Canceled';*

*-- resolved*

*SELECT*

*count(\*) INTO resolved*

*FROM*

*orders*

*WHERE*

*customerNumber = cust\_no*

*AND status = 'Resolved';*

*-- disputed*

*SELECT*

*count(\*) INTO disputed*

*FROM*

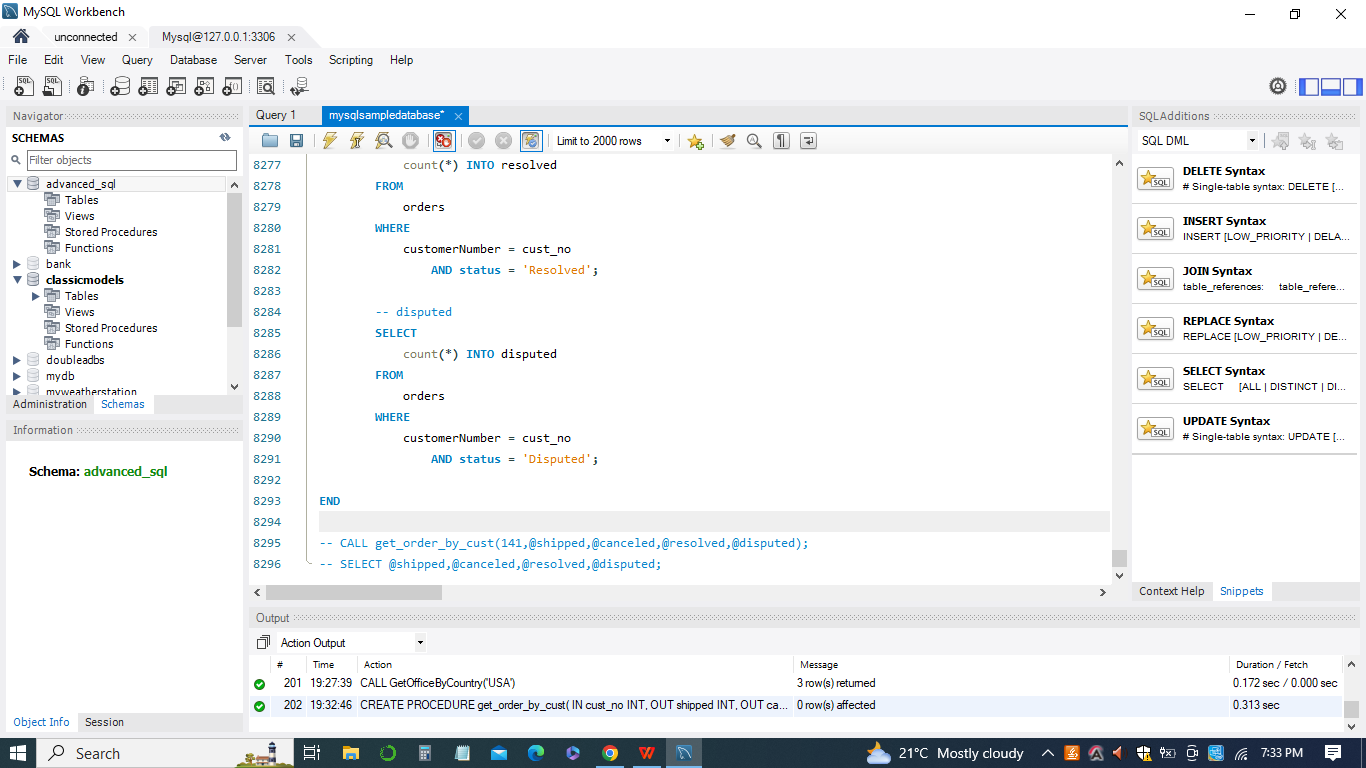
*orders*

*WHERE*

*customerNumber = cust\_no*

*AND status = 'Disputed';*

*END*



***Module 8: MySQL IF and CASE Statements***

1. *Exercise 8.1: how to use MySQL IF statement to execute a block of SQL code based on conditions.*

*DELIMITER $$*

*CREATE PROCEDURE GetCustomerLevel(*

*IN pCustomerNumber INT,*

*OUT pCustomerLevel VARCHAR(20))*

*BEGIN*

*DECLARE credit DECIMAL(10,2) DEFAULT 0;*

*SELECT creditLimit*

*INTO credit*

*FROM customers*

*WHERE customerNumber = pCustomerNumber;*

*IF credit > 50000 THEN*

*SET pCustomerLevel = 'PLATINUM';*

*END IF;*

*END$$*

*DELIMITER ;*

*SELECT*

*customerNumber,*

*creditLimit*

*FROM*

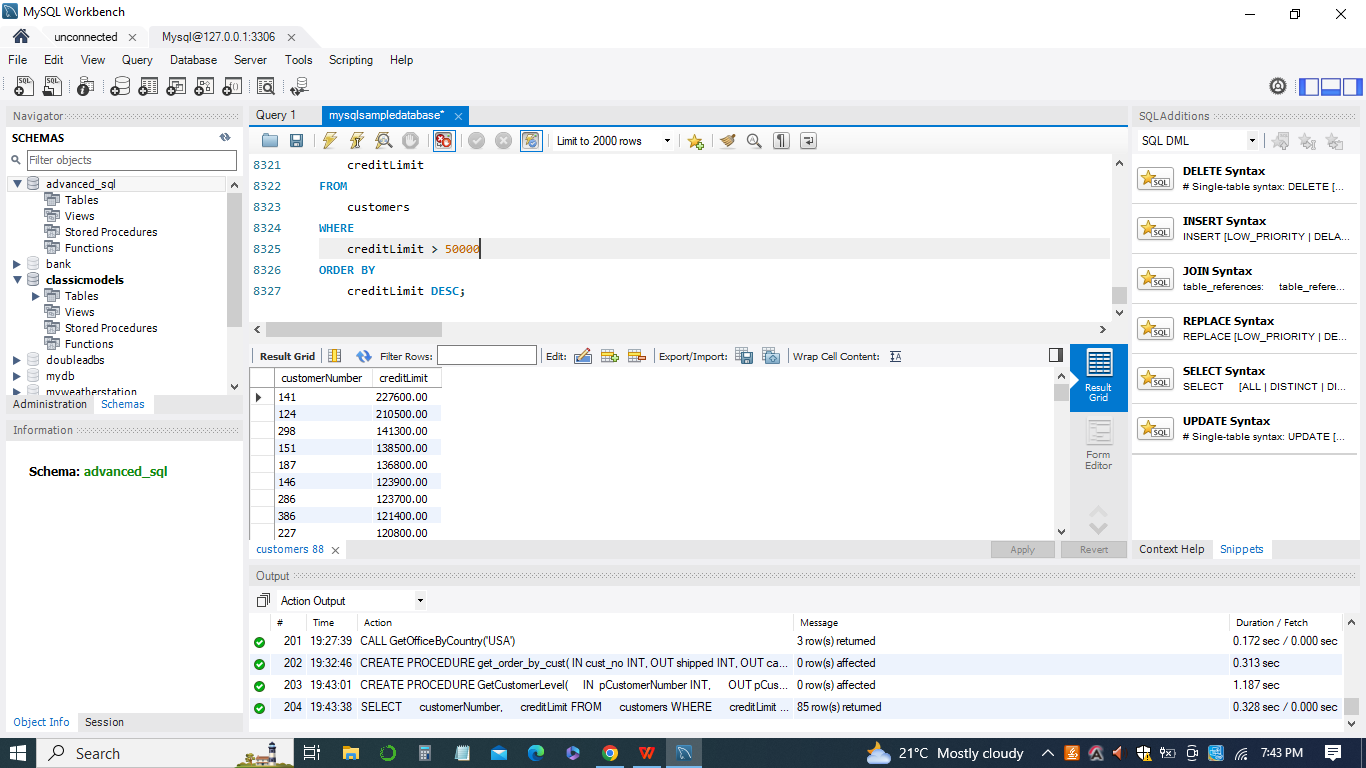
*customers*

*WHERE*

*creditLimit > 50000*

*ORDER BY*

*creditLimit DESC;*



1. *Exercise 8.2: how to use MySQL CASE statements to construct complex conditional statements inside stored programs.*

*DELIMITER $$*

*CREATE PROCEDURE GetCustomerShipping(*

*IN pCustomerNUmber INT,*

*OUT pShipping VARCHAR(50)*

*)*

*BEGIN*

*DECLARE customerCountry VARCHAR(100);*

*SELECT*

*country*

*INTO customerCountry FROM*

*customers*

*WHERE*

*customerNumber = pCustomerNUmber;*

*CASE customerCountry*

*WHEN 'USA' THEN*

*SET pShipping = '2-day Shipping';*

*WHEN 'Canada' THEN*

*SET pShipping = '3-day Shipping';*

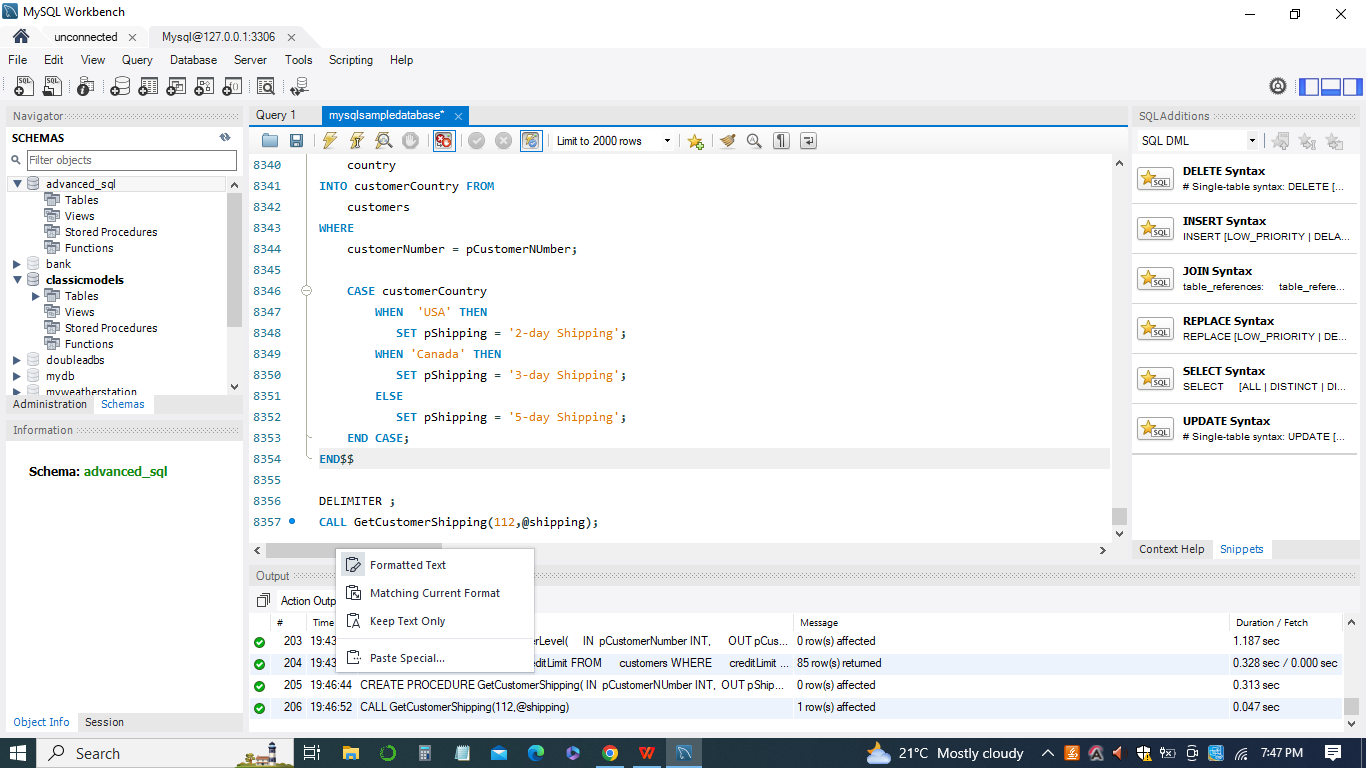
*ELSE*

*SET pShipping = '5-day Shipping';*

*END CASE;*

*END$$*

*DELIMITER ;*



***Module 9: MySQL Loop Statements, Cursor, Listing and Error Handling in Store***

***Procedure***

1. *Exercise 9.1: How to use various loop statements in stored procedure to execute a block of code repeatedly based on a given Boolean condition.*

*DROP PROCEDURE LoopDemo;*

*DELIMITER $$*

*CREATE PROCEDURE LoopDemo()*

*BEGIN*

*DECLARE x INT;*

*DECLARE str VARCHAR(255);*

*SET x = 1;*

*SET str = '';*

*loop\_label: LOOP*

*IF x > 10 THEN*

*LEAVE loop\_label;*

*END IF;*

*SET x = x + 1;*

*IF (x mod 2) THEN*

*ITERATE loop\_label;*

*ELSE*

*SET str = CONCAT(str,x,',');*

*END IF;*

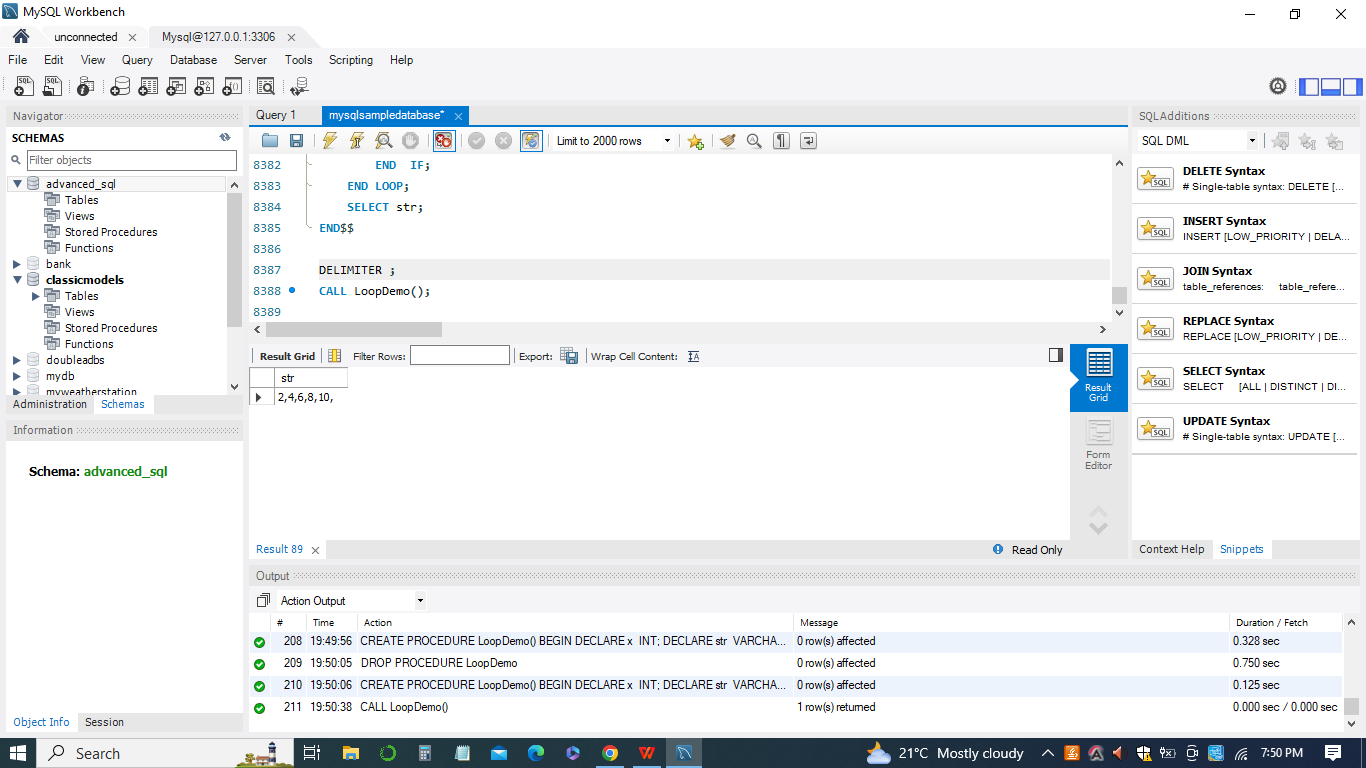
*END LOOP;*

*SELECT str;*

*END$$*

*DELIMITER ;*

*CALL LoopDemo();*



1. *Exercise 9.2: How to use MySQL cursor in stored procedures to loop over a result set and process each row at a time.*

*DELIMITER $$*

*CREATE PROCEDURE createEmailList (*

*INOUT emailList varchar(4000)*

*)*

*BEGIN*

*DECLARE finished INTEGER DEFAULT 0;*

*DECLARE emailAddress varchar(100) DEFAULT "";*

*-- declare cursor for employee email*

*DEClARE curEmail*

*CURSOR FOR*

*SELECT email FROM employees;*

*-- declare NOT FOUND handler*

*DECLARE CONTINUE HANDLER*

*FOR NOT FOUND SET finished = 1;*

*OPEN curEmail;*

*getEmail: LOOP*

*FETCH curEmail INTO emailAddress;*

*IF finished = 1 THEN*

*LEAVE getEmail;*

*END IF;*

*-- build email list*

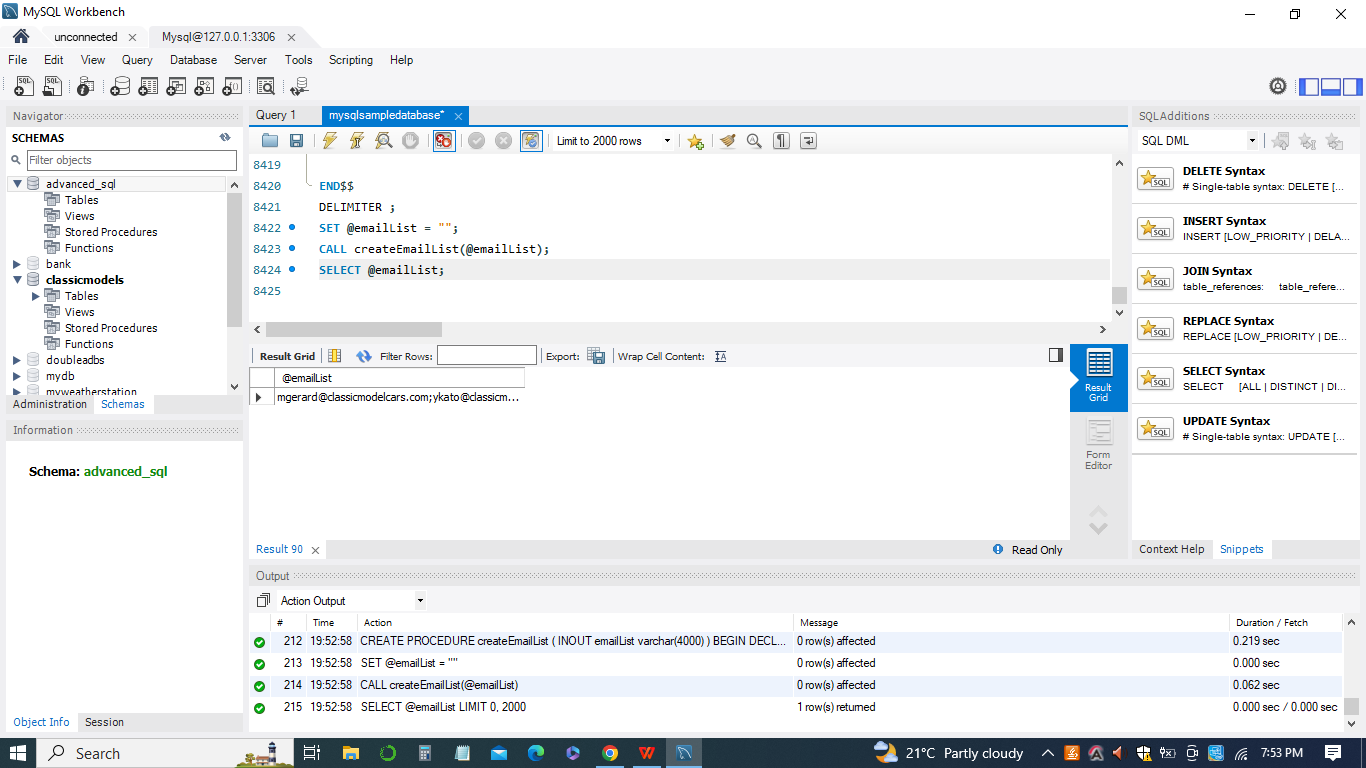
*SET emailList = CONCAT(emailAddress,";",emailList);*

*END LOOP getEmail;*

*CLOSE curEmail;*

*END$$*

*DELIMITER ;*



1. *Exercise 9.3: How to list all stored procedures in a MySQL database.*

*SELECT*

*routine\_name*

*FROM*

*information\_schema.routines*

*WHERE*

*routine\_type = 'PROCEDURE'*

*AND routine\_schema = '<database\_name>';*

*SELECT*

*routine\_name*

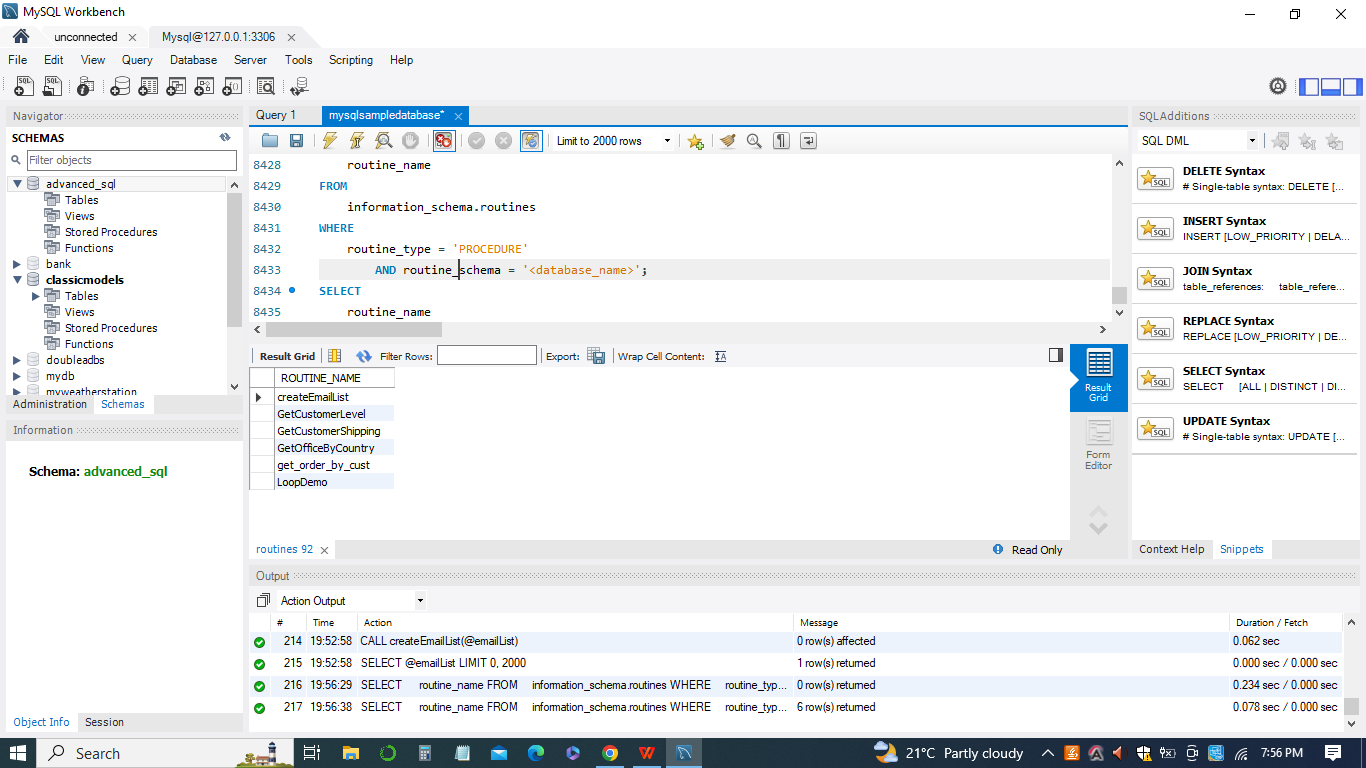
*FROM*

*information\_schema.routines*

*WHERE*

*routine\_type = 'PROCEDURE'*

*AND routine\_schema = 'classicmodels';*



1. *Exercise 9.4: How to use MySQL handler to handle exceptions or errors encountered in stored procedures.*

*DROP PROCEDURE IF EXISTS InsertSupplierProduct;*

*DELIMITER $$*

*CREATE PROCEDURE InsertSupplierProduct(*

*IN inSupplierId INT,*

*IN inProductId INT*

*)*

*BEGIN*

*-- exit if the duplicate key occurs*

*DECLARE EXIT HANDLER FOR 1062 SELECT 'Duplicate keys error encountered' Message;*

*DECLARE EXIT HANDLER FOR SQLEXCEPTION SELECT 'SQLException encountered' Message;*

*DECLARE EXIT HANDLER FOR SQLSTATE '23000' SELECT 'SQLSTATE 23000' ErrorCode;*

*-- insert a new row into the SupplierProducts*

*INSERT INTO SupplierProducts(supplierId,productId)*

*VALUES(inSupplierId,inProductId);*

*-- return the products supplied by the supplier id*

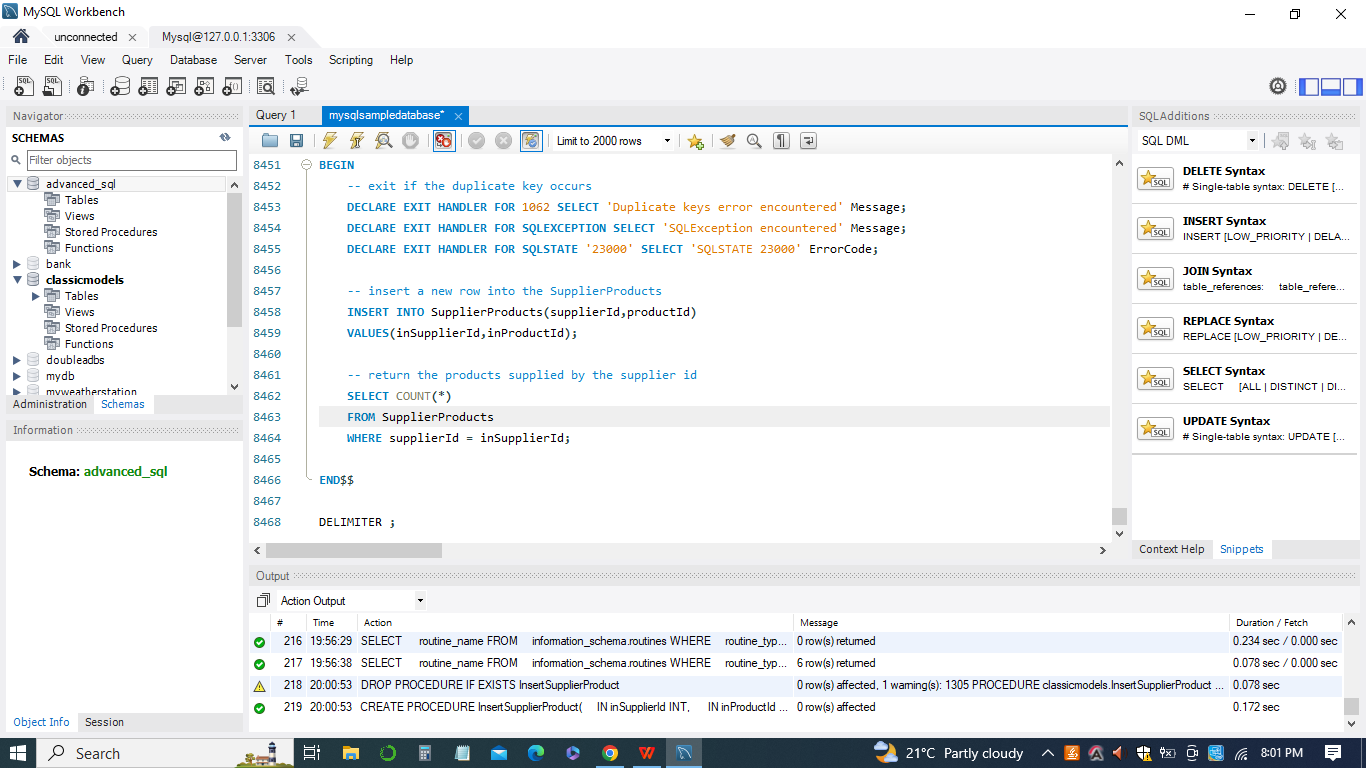
*SELECT COUNT(\*)*

*FROM SupplierProducts*

*WHERE supplierId = inSupplierId;*

*END$$*

*DELIMITER ;*



***Module 10: MySQL Stored Procedures – Raising Error Conditions and Stored Function***

1. *Exercise 10.1: How to use MySQL SIGNAL and RESIGNAL statements to raise error conditions inside stored procedures.*

*DELIMITER $$*

*CREATE PROCEDURE AddOrderItem(*

*in orderNo int,*

*in productCode varchar(45),*

*in qty int,*

*in price double,*

*in lineNo int )*

*BEGIN*

*DECLARE C INT;*

*SELECT COUNT(orderNumber) INTO C*

*FROM orders*

*WHERE orderNumber = orderNo;*

*-- check if orderNumber exists*

*IF(C != 1) THEN*

*SIGNAL SQLSTATE '45000'*

*SET MESSAGE\_TEXT = 'Order No not found in orders table';*

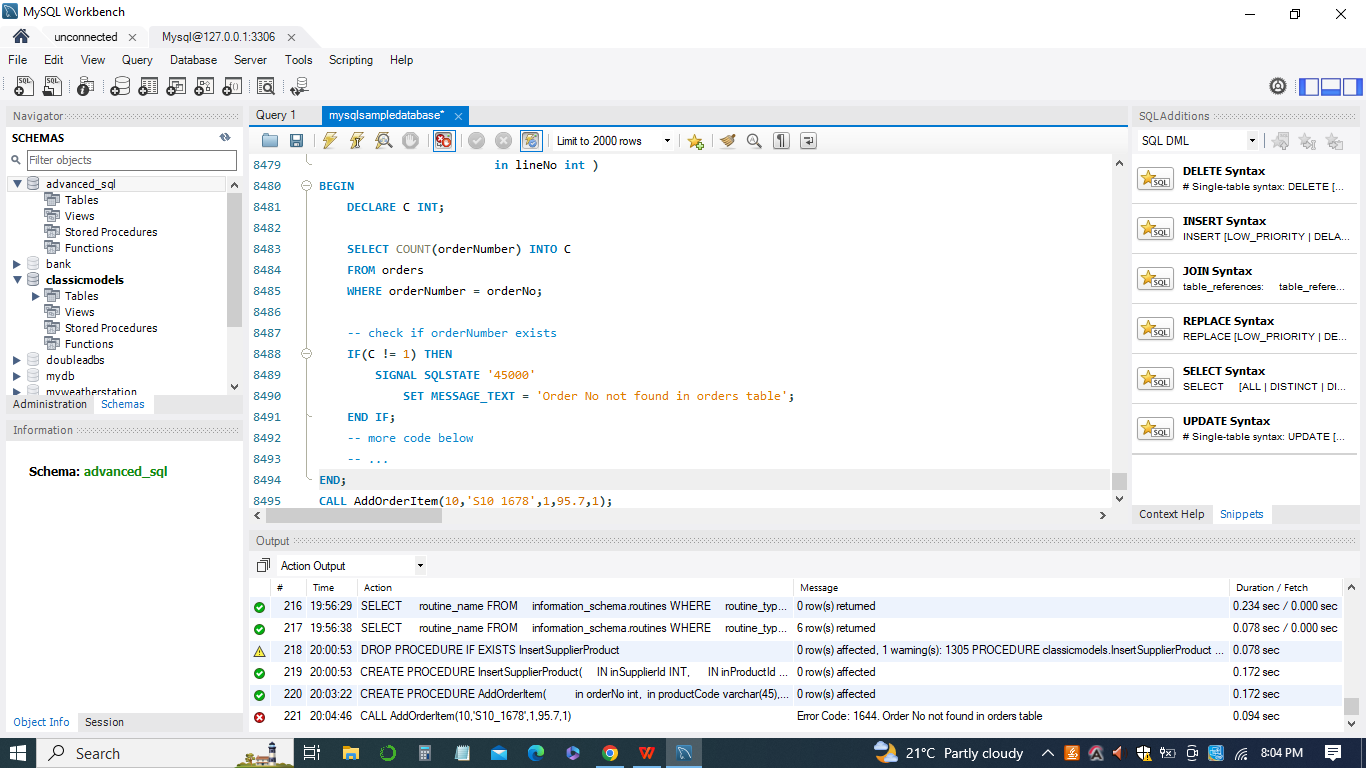
*END IF;*

*-- more code below*

*-- ...*

*END*

*CALL AddOrderItem(10,'S10\_1678',1,95.7,1);*



1. *Exercise 10.2: How to create MySQL stored functions using CREATE FUNCTION statement.*

*DELIMITER $$*

*CREATE FUNCTION CustomerLevel(*

*credit DECIMAL(10,2)*

*)*

*RETURNS VARCHAR(20)*

*DETERMINISTIC*

*BEGIN*

*DECLARE customerLevel VARCHAR(20);*

*IF credit > 50000 THEN*

*SET customerLevel = 'PLATINUM';*

*ELSEIF (credit >= 50000 AND*

*credit <= 10000) THEN*

*SET customerLevel = 'GOLD';*

*ELSEIF credit < 10000 THEN*

*SET customerLevel = 'SILVER';*

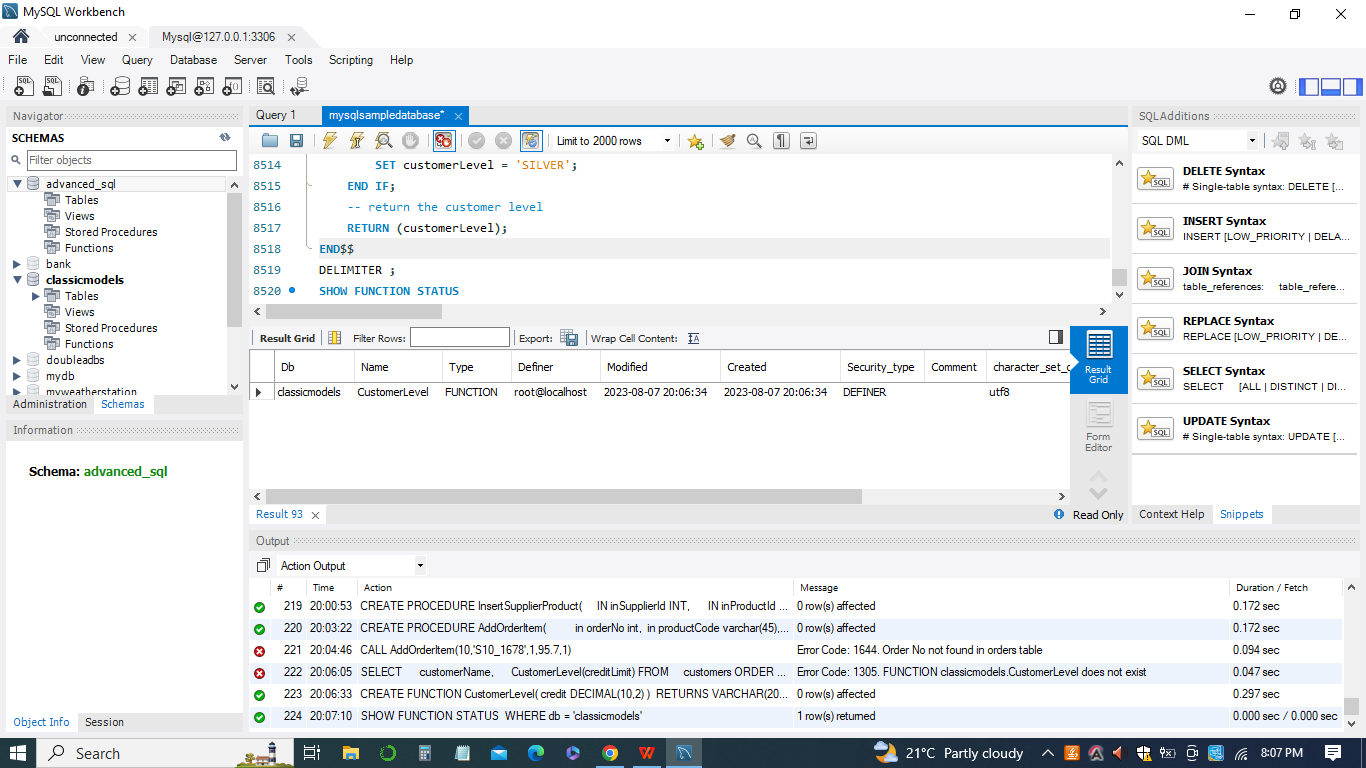
*END IF;*

*-- return the customer level*

*RETURN (customerLevel);*

*END$$*

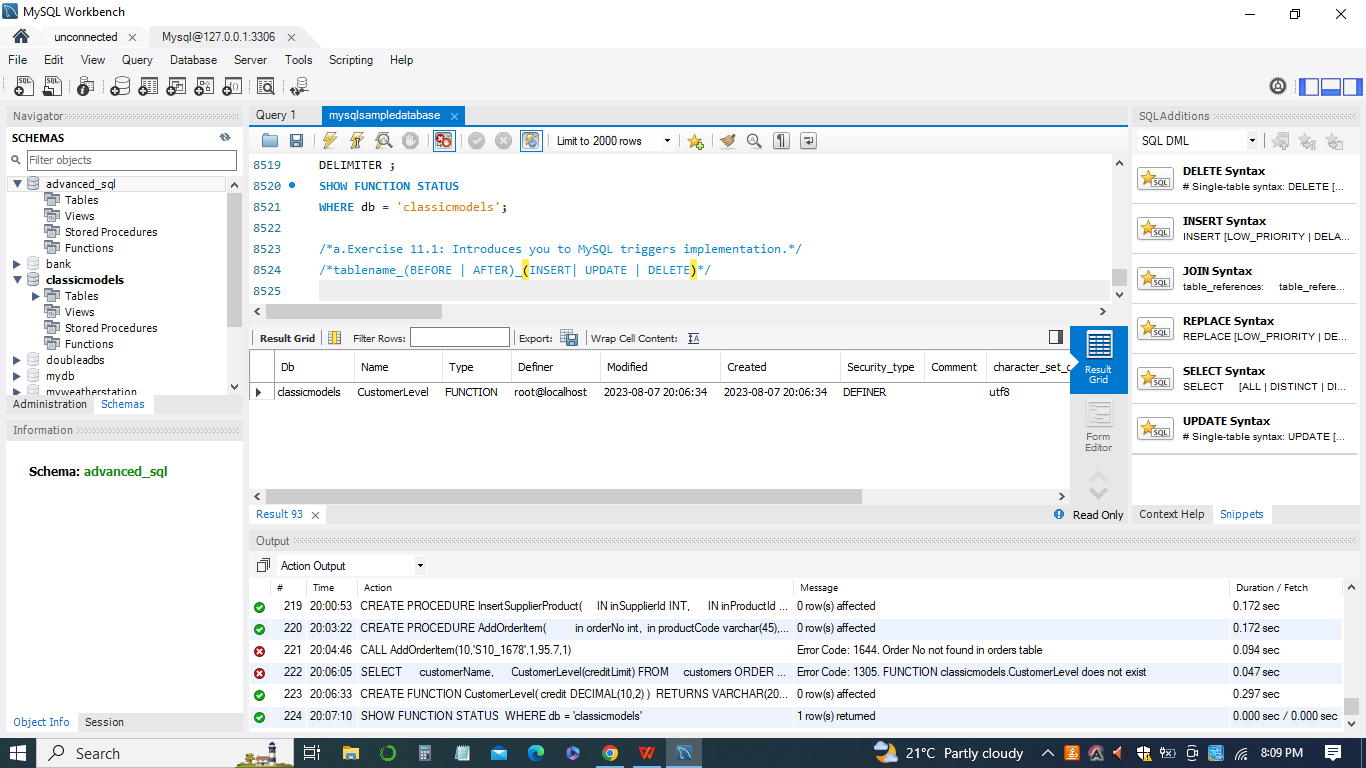
*DELIMITER ;*



***Module 11: MySQL Triggers Implementation, Creating and Managing***

1. *Exercise 11.1: Introduces you to MySQL triggers implementation.*

*tablename\_(BEFORE | AFTER)\_(INSERT| UPDATE | DELETE)*



1. *Exercise 11.2: Shows you how to create a simple trigger in MySQL to audit the changes of a table.*

*CREATE TABLE employees\_audit (*

*id INT AUTO\_INCREMENT PRIMARY KEY,*

*employeeNumber INT NOT NULL,*

*lastname VARCHAR(50) NOT NULL,*

*changedat DATETIME DEFAULT NULL,*

*action VARCHAR(50) DEFAULT NULL*

*);*

*CREATE TRIGGER before\_employee\_update*

*BEFORE UPDATE ON employees*

*FOR EACH ROW*

*INSERT INTO employees\_audit*

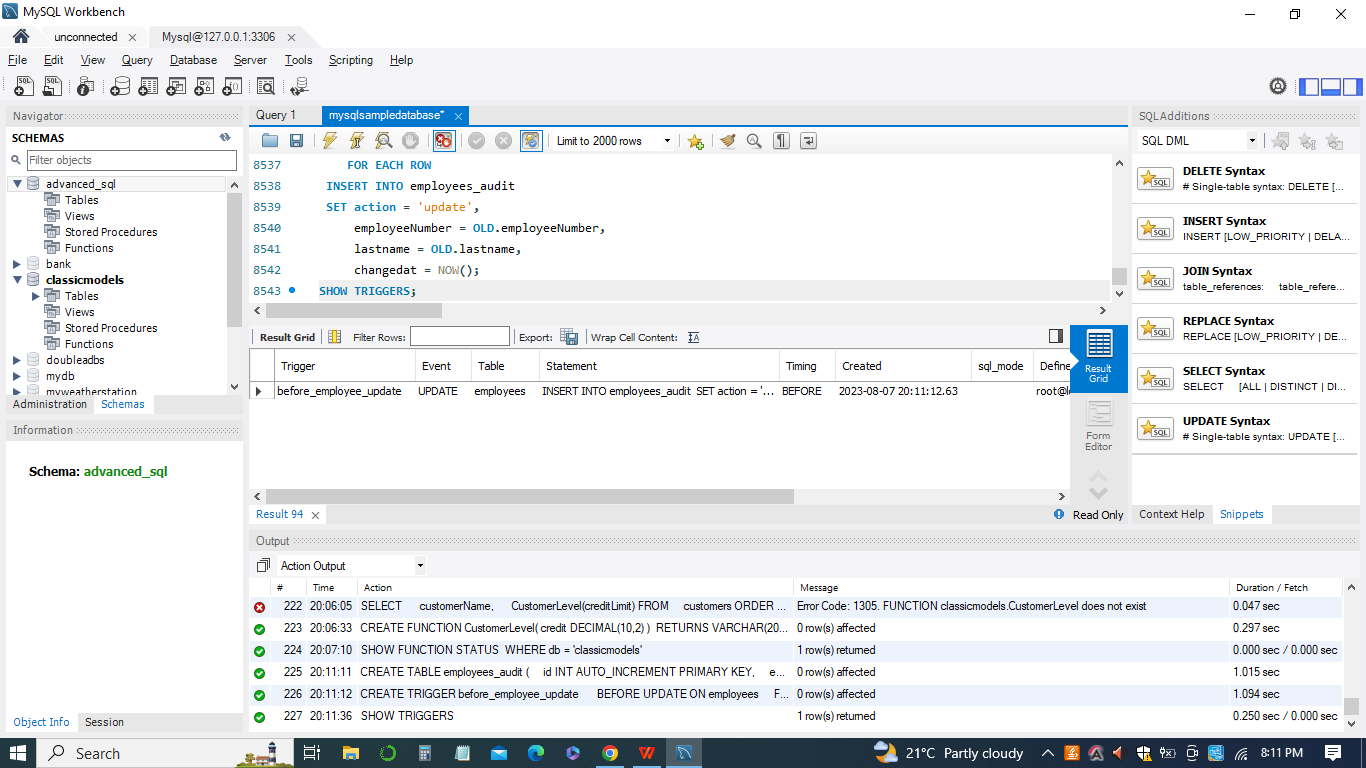
*SET action = 'update',*

*employeeNumber = OLD.employeeNumber,*

*lastname = OLD.lastname,*

*changedat = NOW();*

*SHOW TRIGGERS;*



1. *Exercise 11.3: Shows you how to create multiple triggers for the same trigger event and action time*

*CREATE TABLE PriceLogs (*

*id INT AUTO\_INCREMENT,*

*productCode VARCHAR(15) NOT NULL,*

*price DECIMAL(10,2) NOT NULL,*

*updated\_at TIMESTAMP NOT NULL*

*DEFAULT CURRENT\_TIMESTAMP*

*ON UPDATE CURRENT\_TIMESTAMP,*

*PRIMARY KEY (id),*

*FOREIGN KEY (productCode)*

*REFERENCES products (productCode)*

*ON DELETE CASCADE*

*ON UPDATE CASCADE*

*);*

*DELIMITER $$*

*CREATE TRIGGER before\_products\_update*

*BEFORE UPDATE ON products*

*FOR EACH ROW*

*BEGIN*

*IF OLD.msrp <> NEW.msrp THEN*

*INSERT INTO PriceLOgs(productCode,price)*

*VALUES(old.productCode,old.msrp);*

*END IF;*

*END$$*

*DELIMITER ;*

*SELECT*

*productCode,*

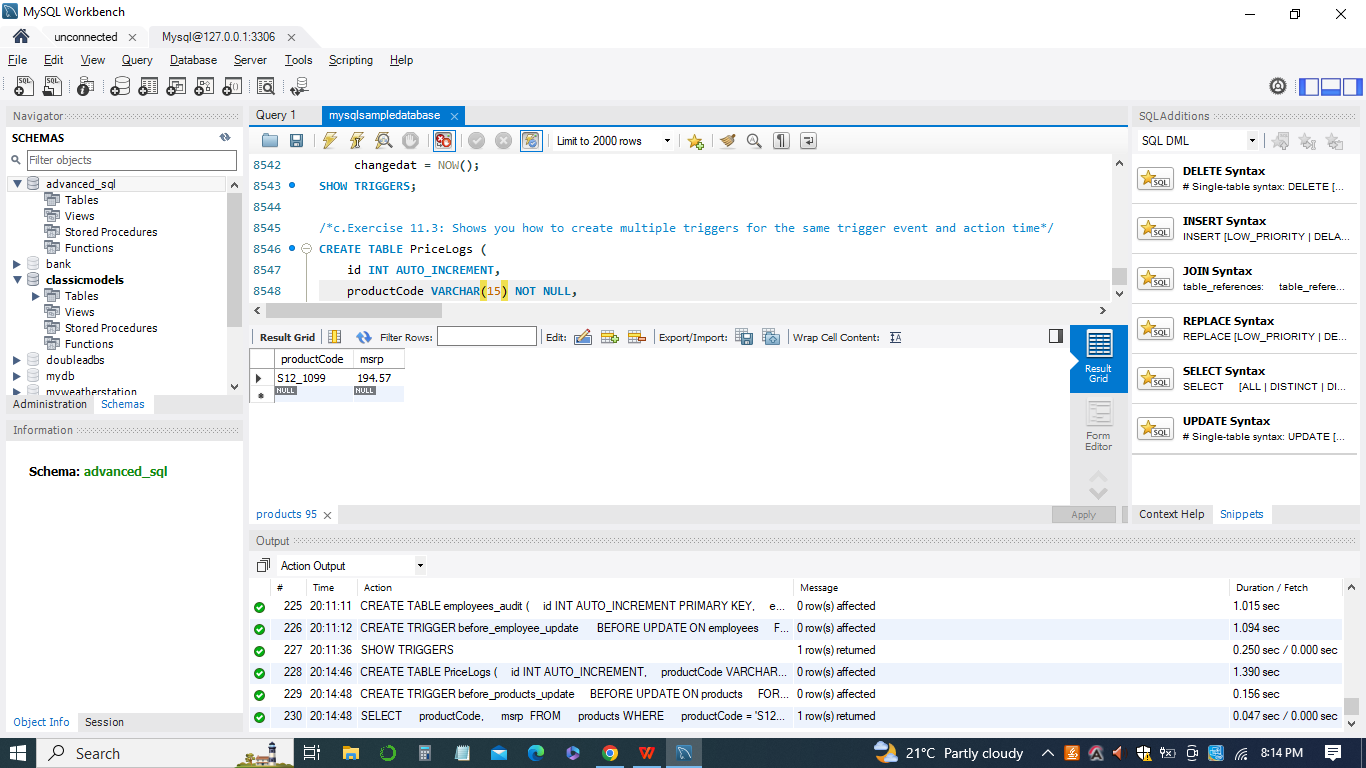
*msrp*

*FROM*

*products*

*WHERE*

*productCode = 'S12\_1099';*



1. *Exercise 11.4: How to manage triggers including displaying, modifying, and removing triggers in MySQL databases*

*SHOW TRIGGERS*

*FROM classicmodels;*



***Module 12: Working with MySQL Scheduled Event and Modifying MySQL Events***

1. *Exercise 12.1 : Working with MySQL Scheduled Event*

*SHOW PROCESSLIST;*

*SET GLOBAL event\_scheduler = ON;*

*SHOW PROCESSLIST;*



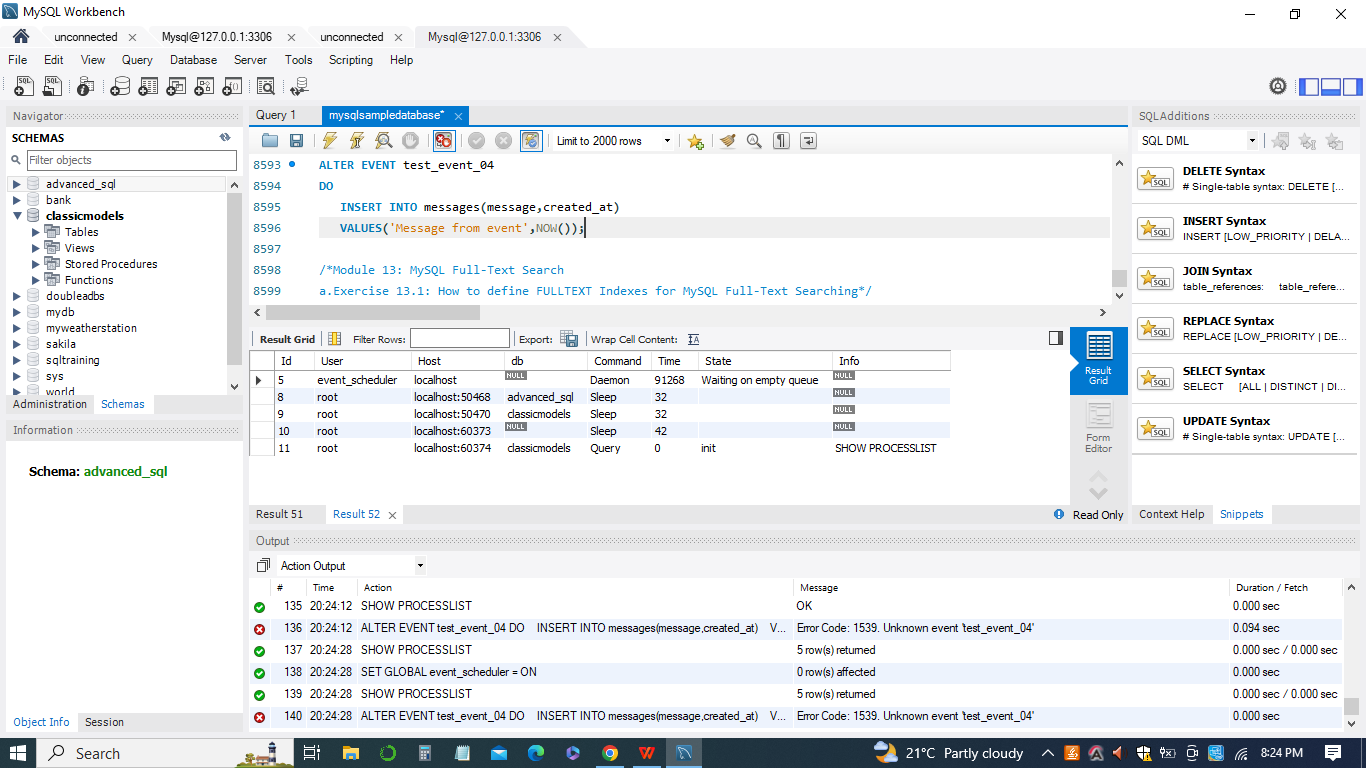
1. *Exercise 12.2 :Modifying MySQL Events*

*ALTER EVENT test\_event\_04*

*DO*

*INSERT INTO messages(message,created\_at)*

*VALUES('Message from event',NOW());*



***Module 13: MySQL Full-Text Search***

1. *Exercise 13.1: How to define FULLTEXT Indexes for MySQL Full-Text Searching*

*CREATE TABLE posts (*

*id INT NOT NULL AUTO\_INCREMENT,*

*title VARCHAR(255) NOT NULL,*

*body TEXT,*

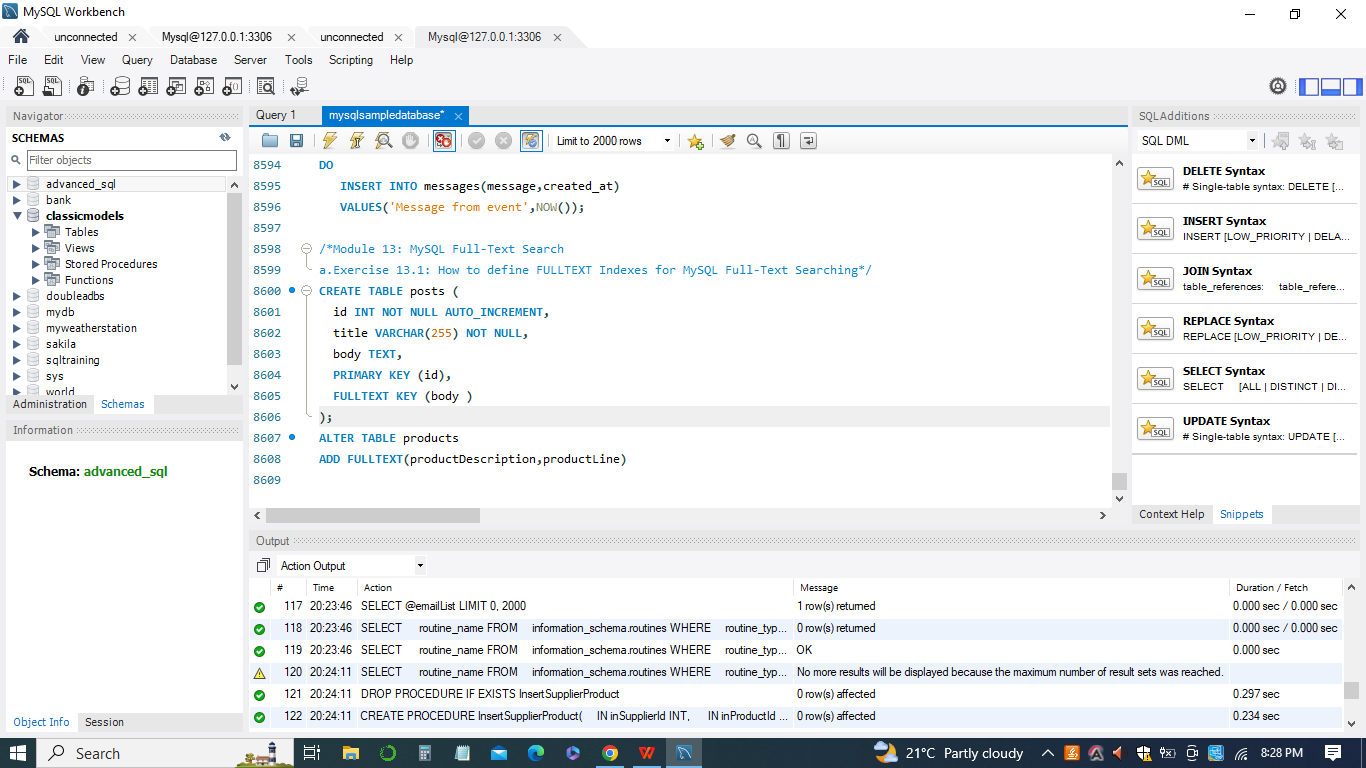
*PRIMARY KEY (id),*

*FULLTEXT KEY (body )*

*);*

*ALTER TABLE offices*

*DROP INDEX address;*



1. *Exercise 13.2: MySQL Natural Language Full-Text Searches*

In MySQL, the FULLTEXT search capability allows you to perform natural language searches on text-based columns. Your example demonstrates how to set up and perform a natural language full-text search:

*ALTER TABLE products*

*ADD FULLTEXT(productline);*

*SELECT*

*productName,*

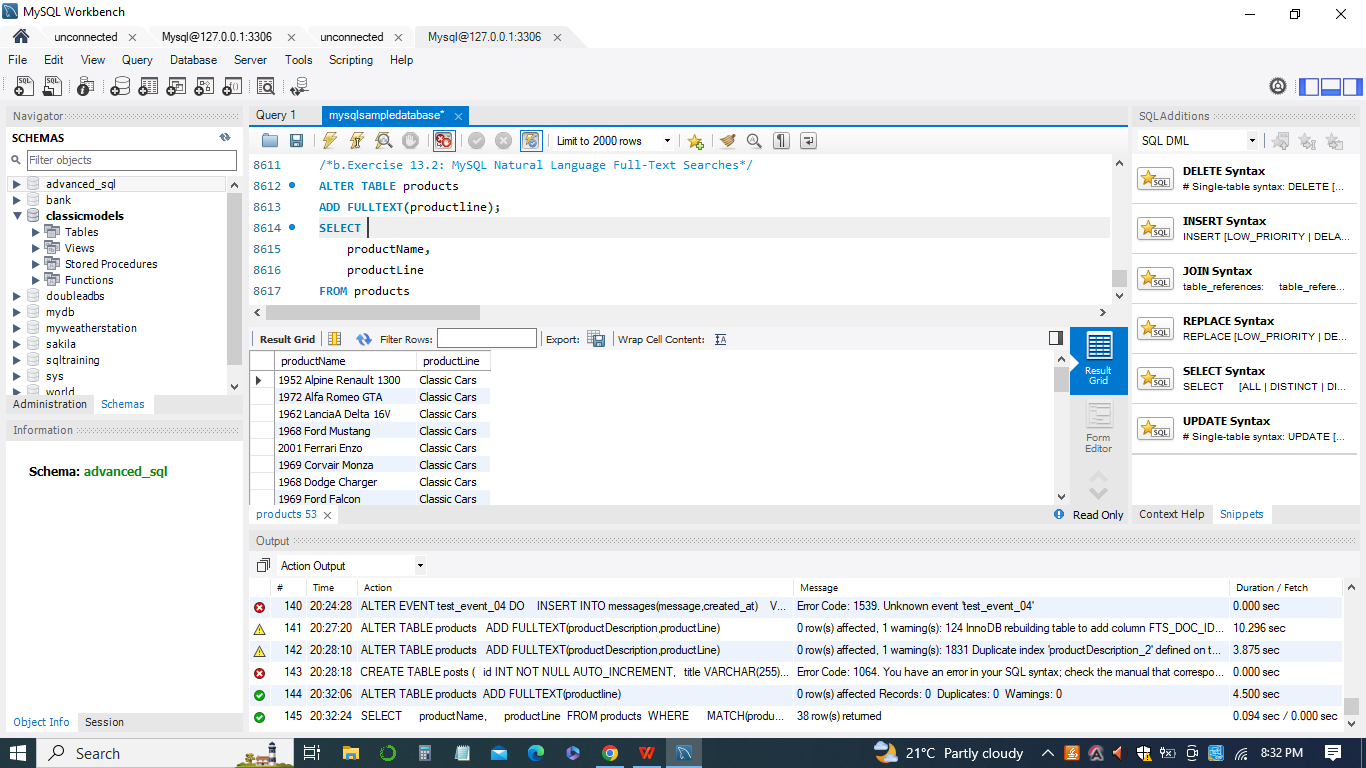
*productLine*

*FROM products*

*WHERE*

*MATCH(productLine)*

*AGAINST('Classic');*



This query searches for products in the 'products' table where the 'productLine' column contains the word 'Classic' using the MATCH ... AGAINST syntax.

SELECT productName, productLine: This part specifies the columns you want to retrieve in the result.

FROM products: This indicates that you're querying the 'products' table.

WHERE MATCH(productLine) AGAINST('Classic'): This is the condition where you use the MATCH ... AGAINST clause to perform the full-text search. It checks if the 'productLine' column matches the word 'Classic'.

The MATCH ... AGAINST clause is used to perform natural language searches. It requires a FULLTEXT index on the columns being searched. The search term 'Classic' is used as an example, and it will match rows where the 'productLine' contains that term.

Natural language full-text searches are useful when you want to search for relevant content based on the actual language used in the data. It's important to note that full-text searches have specific requirements and limitations, and their performance depends on the size of the dataset and the nature of the search terms.

**Conclusion**

In conclusion, the Advanced SQL course offers a comprehensive and structured learning path for individuals seeking to expand their expertise in database management and querying. By delving into functions, stored procedures, triggers, views, and other advanced concepts, students are empowered to handle intricate data operations and enhance their efficiency in working with databases. With a practical hands-on approach, learners can not only gain theoretical knowledge but also gain practical experience that will prove invaluable in real-world scenarios. By completing this course, students can confidently demonstrate their ability to create, manage, and optimize SQL queries, setting the stage for successful and effective data manipulation and management in various professional settings.