NAME: Divyansh Kumar Singh
SAP ID: 590012175
BATCH: B1
DBMS LAB 3

AIM: To implement and use SQL Sub-Query

Lab Objective: To understand the use of SQL Subquery.

Divyansh Kumar Singh

590012175

classmate
Date 26/8/25
Page

LAB-3

26/8

CREATE
DESC
FROM
WHERE
IN
INSERT INTO
SELECT
ANY
SET
DELETE
MAX, MIN
COUNT
AVG
GROUP BY
AS
EXISTS
ROW
FLOOR

⇒ 46 (Approx)

MySQL Subquery

A subquery in MySQL is nested into another SQL query and embedded with SELECT, INSERT, UPDATE, or DELETE statements and the various operators. We can also nest the subquery with another subquery. A subquery is known as the **inner query**, and the query that contains a subquery is known as the **outer query**. The inner query executed first gives the result to the outer query, and then the main/outer query will be performed. MySQL allows us to use subquery anywhere but must be closed within parenthesis. All subquery forms and operations supported by the SQL standard will also be supported in MySQL.

The following are the rules to use subqueries:

o Subqueries should always be in **parentheses.**

o If the main query does not have multiple columns for the subquery, then a subquery can have only one column in the SELECT command.

o We can use various comparison operators with the subquery, such as >, <, =, IN, ANY, SOME, and ALL.
A multiple-row operator is very useful when the subquery returns more than one row.

o We cannot use the **ORDER BY** clause in a subquery, although it can be used inside the main query.

o If we use a subquery in a **set function**, it cannot be immediately enclosed in a set function.

The following are the advantages of using subqueries:

o The subqueries make the queries in a structured form that allows us to isolate each part of a statement.

o The subqueries provide alternative ways to query the data from the table; otherwise, we need to use complex joins and unions.

o The subqueries are more readable than complex join or union statements.

SQL Subquery

The Subquery or Inner query is an SQL query placed inside another SQL query. It is embedded in the HAVING or WHERE clause of the SQL statements.

Following are the important rules which the SQL Subquery must follow:

1. The SQL subqueries can be used with the following statements along with the SQL expression operators:
• SELECT statement,
• UPDATE statement,
• INSERT statement, and
• DELETE statement.

2. The subqueries in SQL are always enclosed in the parenthesis and placed on the right side of the SQL operators.

3. We cannot use the ORDER BY clause in the subquery. But, we can use the GROUP BY clause, which performs the same function as the ORDER BY clause.

4. If the subquery returns more than one record, we must use the multiple value operators before the subquery.
5. We can use the BETWEEN operator within the subquery but not with the subquery.

Subquery with SELECT statement

In SQL, the SELECT statement uses inner or nested queries most frequently. The syntax of the subquery with the SELECT statement is described in the following block:

```
SELECT Column_Name1, Column_Name2, ...., Column_NameN
FROM Table_Name WHERE Column_Name Comparison_Operator
( SELECT Column_Name1, Column_Name2, ...., Column_NameN
FROM Table_Name WHERE condition;
```

Subquery with the INSERT statement

We can also use the subqueries and nested queries with the INSERT statement in Structured Query Language. We can insert the subquery results into the table of the outer query. The syntax of the subquery with the INSERT statement is described in the following block:

```
INSERT INTO Table_Name SELECT * FROM Table_Name WHERE
Column_Name Operator (Subquery);
```

Subquery with the UPDATE statement

The subqueries and nested queries can be used with the UPDATE statement in Structured Query Language to update the existing table's columns. We can easily update one or more columns using a subquery with the UPDATE statement.

Syntax of Subquery with the UPDATE statement

```
UPDATE Table_Name SET Column_Name = New_value WHERE Value
OPERATOR (SELECT COLUMN_NAME FROM TABLE_NAME
WHERE Condition) ;
```

Subquery with the DELETE statement

We can easily delete one or more records from the SQL table using subquery with the DELETE statement in Structured Query Language.

Syntax of Subquery with DELETE statement

DELETE FROM Table_Name WHERE Value OPERATOR (SELECT COLUMN_NAME FROM TABLE_NAME WHERE Condition) ;

MySQL Subquery with Comparison Operator

A comparison operator is an operator used to compare values and returns the result, either true or false. The following comparison operators are used in MySQL <, >, =, <>, <=>, etc. We can use the subquery before or after the comparison operators that return a single value. The returned value can be the arithmetic expression or acolumn function. After that, SQL compares the subquery results with the value on the other side of the comparison operator.

The example below explains it more clearly:
Following is a simple statement that returns the **employee detail whose income is more than 350000** with the help of a subquery:

```
SELECT * FROM employees
WHERE emp_id IN (SELECT emp_id FROM employees
WHERE income > 350000);
```

This query first executes the subquery that returns the **employee id whose income > 350000**. Second, the main query will return the employees all details whose employee id are in the result set returned by the subquery. After executing the statement, we will get the below output, where we can see the employee detail whose income>350000.

Let us see an example of another comparison operator, such as equality (=) to find employee details with **maximum income** using a subquery.

```
SELECT emp_name, city, income FROM employees
WHERE income = (SELECT MAX(income) FROM employees);
```

It will give the output where we can see two employees detail who have maximum income.

MySQL Subquery with IN or NOT-IN Operator

If the subquery produces more than one value, we need to use the IN or NOT IN Operator with the WHERE clause.

Suppose we have a table named **"Student"** and **"Student2"** that contains the following data:

Table: Student

Stud_ID	Name	Email	City
1	Peter	peter@javatpoint.com	Texas
2	Suzi	suzi@javatpoint.com	California
3	Joseph	joseph@javatpoint.com	Alaska
4	Andrew	andrew@javatpoint.com	Los Angeles
5	Brayan	brayan@javatpoint.com	New York

Table: Student2

Stud_ID	Name	Email	City
1	Stephen	stephen@javatpoint.com	Texas
2	Joseph	joseph@javatpoint.com	Los Angeles
3	Peter	peter@javatpoint.com	California
4	David	david@javatpoint.com	New York
5	Maddy	maddy@javatpoint.com	Los Angeles

The following subquery with NOT IN Operator returns the **student detail who does not belong to Los Angeles City** from both tables as follows:

```
SELECT Name, City FROM student
WHERE City NOT IN (
SELECT City FROM student2 WHERE City='Los Angeles');
```

After execution, we can see that the result contains the student details that do not belong to Los Angeles City.

MySQL Subquery in the FROM Clause

If we use a subquery in the FROM clause, MySQL will return the output from a subquery is used as a temporary table. We called this table as a derived table, inline views, or materialized subquery. The following subquery returns the maximum, minimum, and average number of items in the order table:

```
SELECT Max(items), MIN(items), FLOOR(AVG(items))
FROM
(SELECT order_id, COUNT(order_id) AS items FROM orders
GROUP BY order_date) AS Student_order_detail;
```

MySQL Correlated Subqueries

A correlated subquery in MySQL is a subquery that depends on the outer query. It uses the data from the outer query or contains a reference to a parent query that also appears in the outer query. MySQL evaluates it once from each row in the outer query.

```
SELECT emp_name, city, income
FROM employees emp WHERE income > (
SELECT AVG(income) FROM employees WHERE city = emp.city);
```

In the above query, we select an **employee name and city** whose income is higher than the average income of all employees in each city.The subquery executes for every city of the specified table because it will change for every row.
Therefore, the average income will also be changed. Then, the main query filters employee details whose income is higher than the average income from the subquery.

MySQL Subqueries with EXISTS or NOT EXISTS

The EXISTS operator is a Boolean operator that returns a true or false result. It is used with a subquery and checks the existence of data in a subquery. If a subquery returns any record at all, this Operator returns true. Otherwise,

it will return false. The NOT EXISTS Operator used for negation that gives true value when the subquery does not return any row. Otherwise, it returns false. Both EXISTS and NOT EXISTS used with correlated subqueries.
The following example illustrates it more clearly. Suppose we have a table **customer and order.**
The below SQL statements uses EXISTS operator to find the name, occupation, and age of the customer who has placed at least one order.

```
SELECT name, occupation, age FROM customer C
WHERE EXISTS (SELECT * FROM Orders O
WHERE C.cust_id = O.cust_id);
```

This statement uses NOT EXISTS operator that returns the customer details who have not placed an order.

```
SELECT name, occupation, age FROM customer C
WHERE NOT EXISTS (SELECT * FROM Orders O
WHERE C.cust_id = O.cust_id);
```

We can see the below output to understand the above queries result.

MySQL ROW Subqueries

It is a subquery that returns a single row where we can get more than one column values. We can use the following operators for comparing row subqueries =, >, <, >=, <=, <>, !=, <=>. Let us see the following example:

```
SELECT * FROM customer C WHERE ROW(cust_id, occupation) = (
SELECT order_id, order_date FROM Orders O WHERE C.cust_id =
O.cust_id);
```

If given row has cust_id, occupation values equal to the order_id, order_date values of any rows in the first table, the WHERE expression is TRUE, and each query returns those first table rows. Otherwise, the expression is FALSE, and the query produces an empty set.

MySQL Subqueries with ALL, ANY, and SOME

We can use a subquery which is followed by the keyword ALL, ANY, or SOME after a comparison operator. The following are the syntax to use subqueries with ALL, ANY, or SOME:

operand comparison_operator ANY (subquery)
operand comparison_operator ALL (subquery)
operand comparison_operator SOME (subquery)

The ALL keyword compares values with the value returned by a subquery. Therefore, it returns TRUE if the comparison is TRUE for ALL of the values returned by a subquery.
The ANY keyword returns TRUE if the comparison is TRUE for ANY of the values returned by a subquery.
The ANY and SOME keywords are the same because they are the alias of each other. The following example explains it more clearly:

```
SELECT cust_id, name FROM customer WHERE
cust_id > ANY (SELECT cust_id FROM Orders);
```

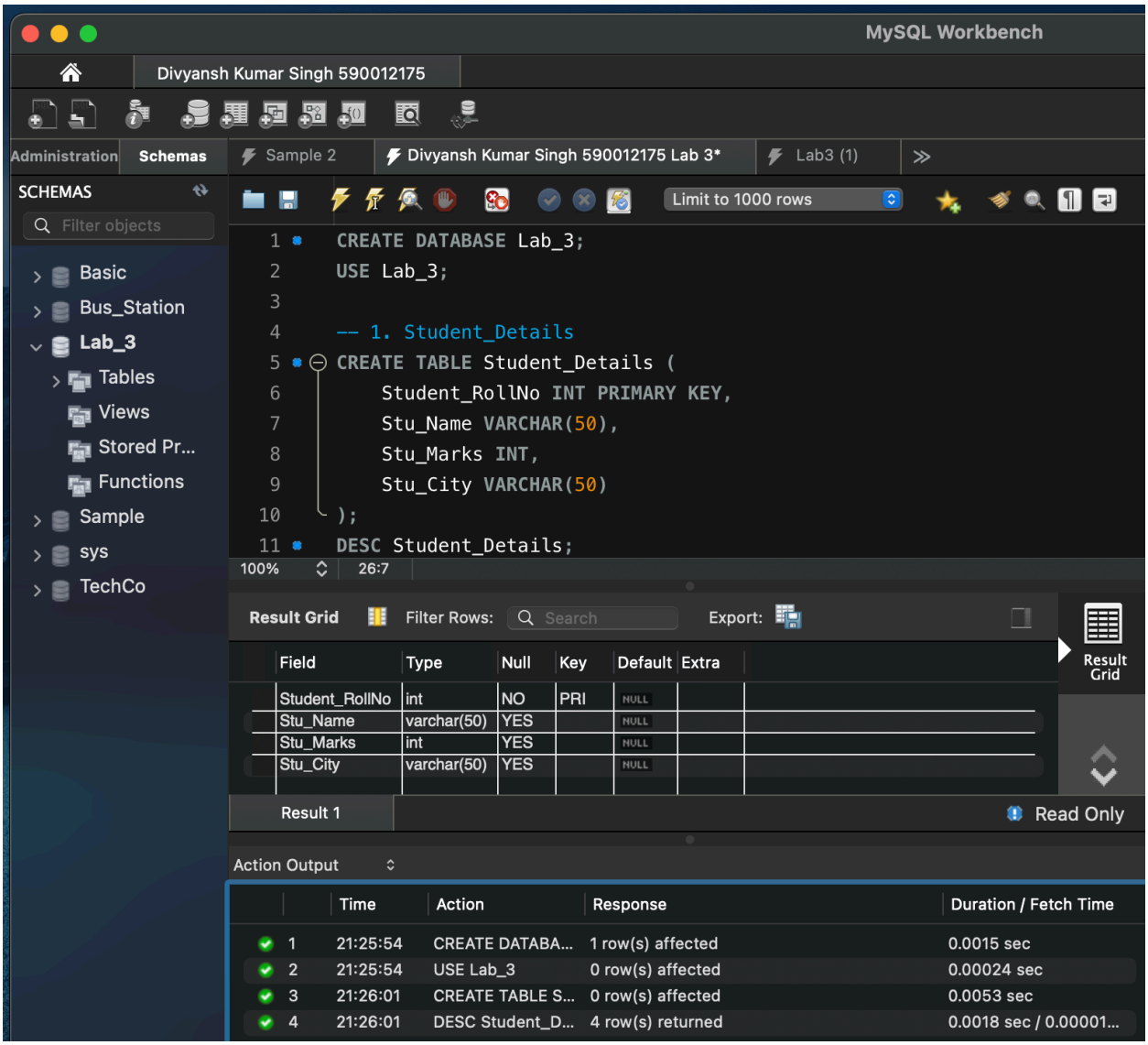
If we use ALL in place of ANY, it will return TRUE when the comparison is TRUE for ALL values in the column returned by a subquery. For example:

```
SELECT cust_id, name FROM customer WHERE
cust_id > ALL (SELECT cust_id FROM Orders);
```

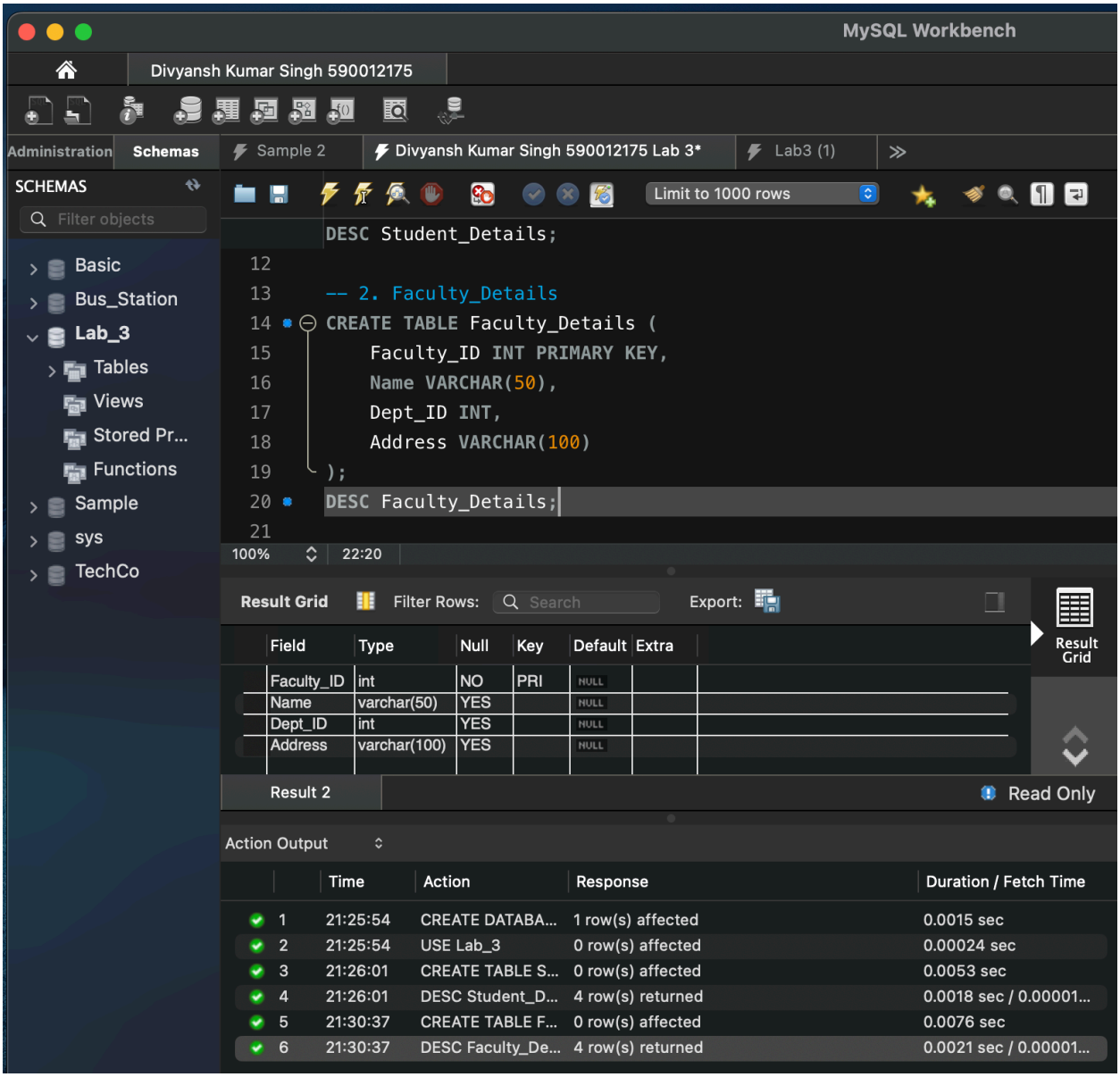
Lab Performance Questions

Tables:

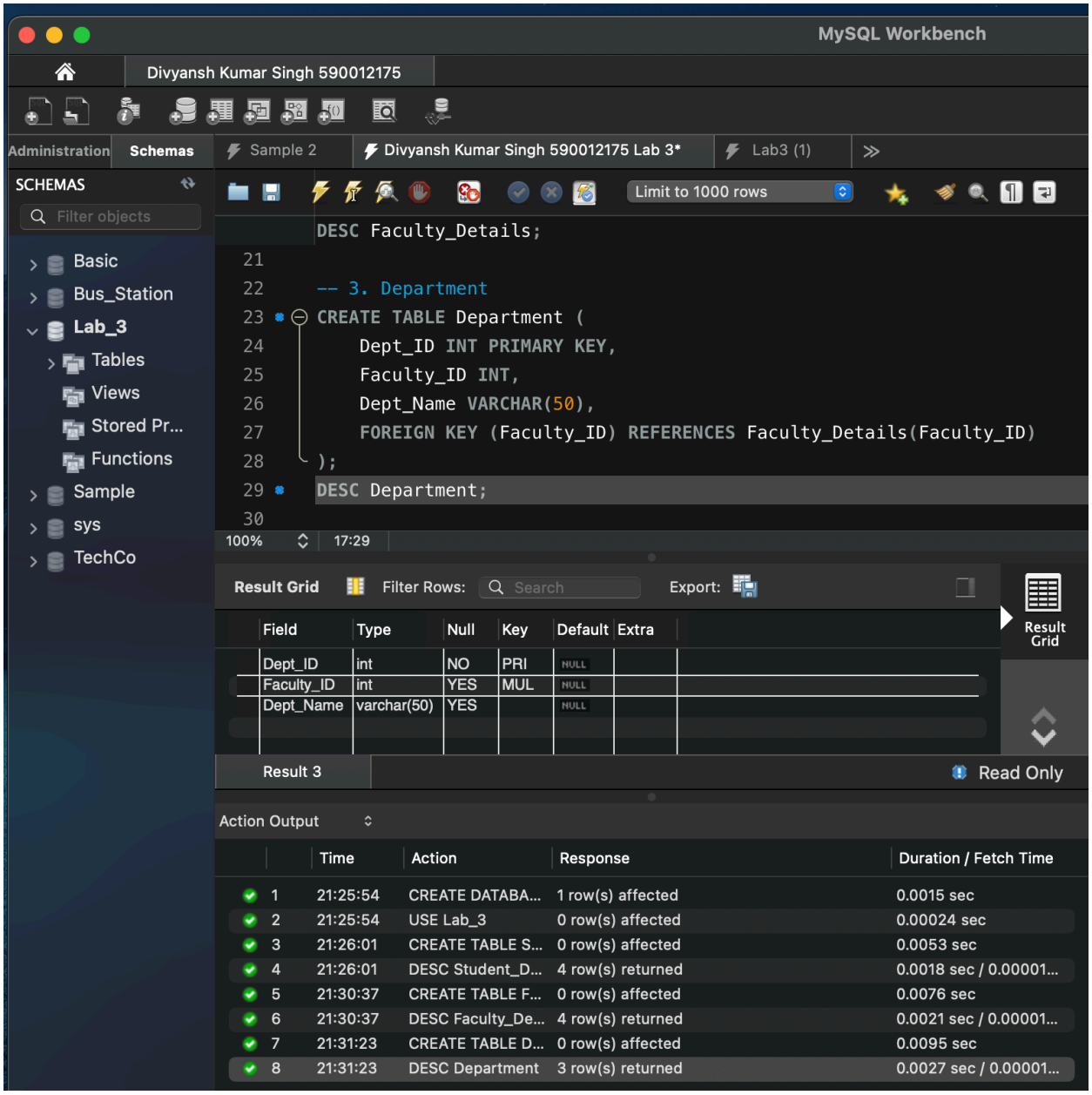
1. Student_Details
- Columns: Student_RollNo, Stu_Name, Stu_Marks, Stu_City

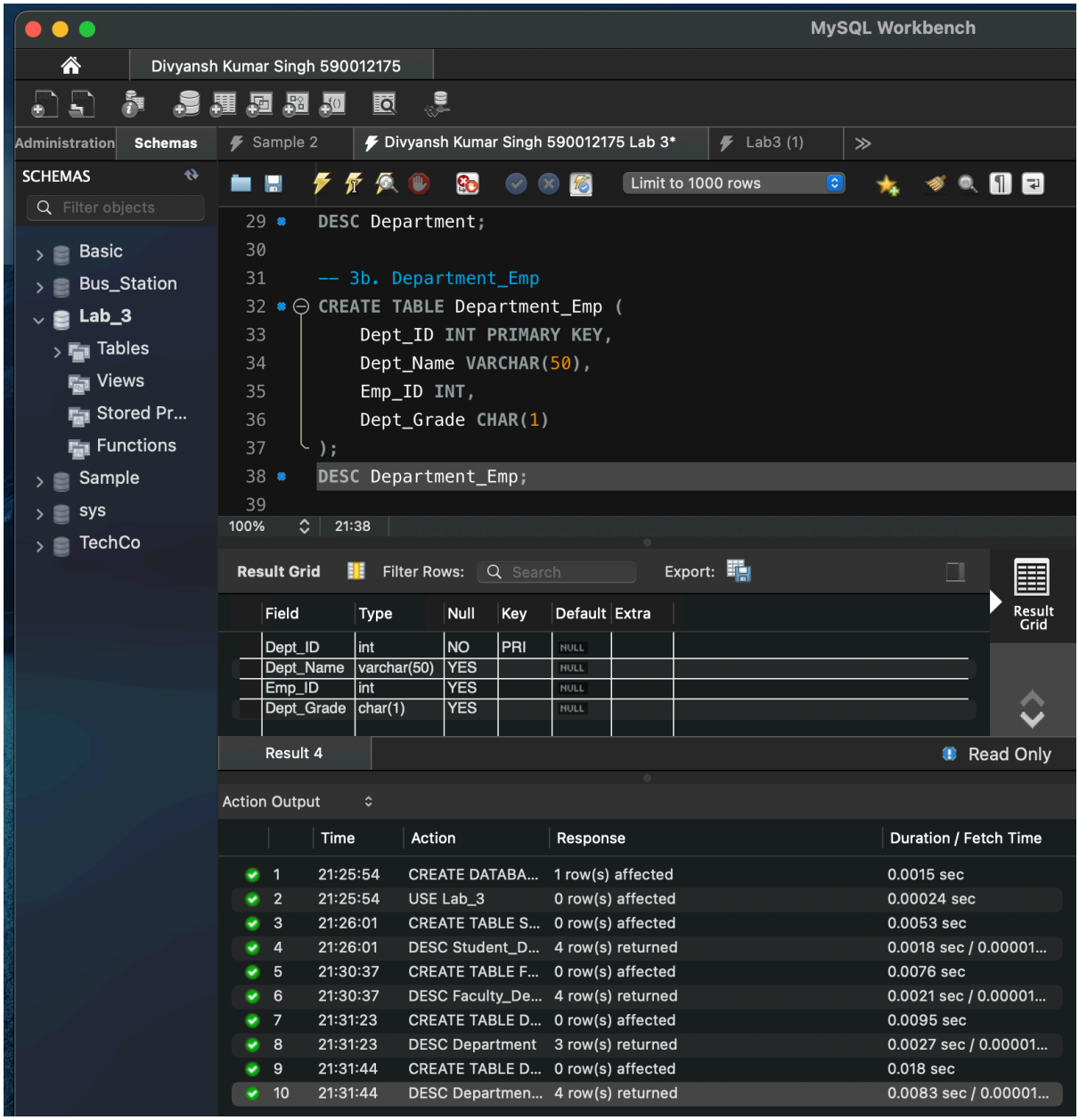


2. Faculty_Details
- Columns: Faculty_ID, Name, Dept_ID, Address



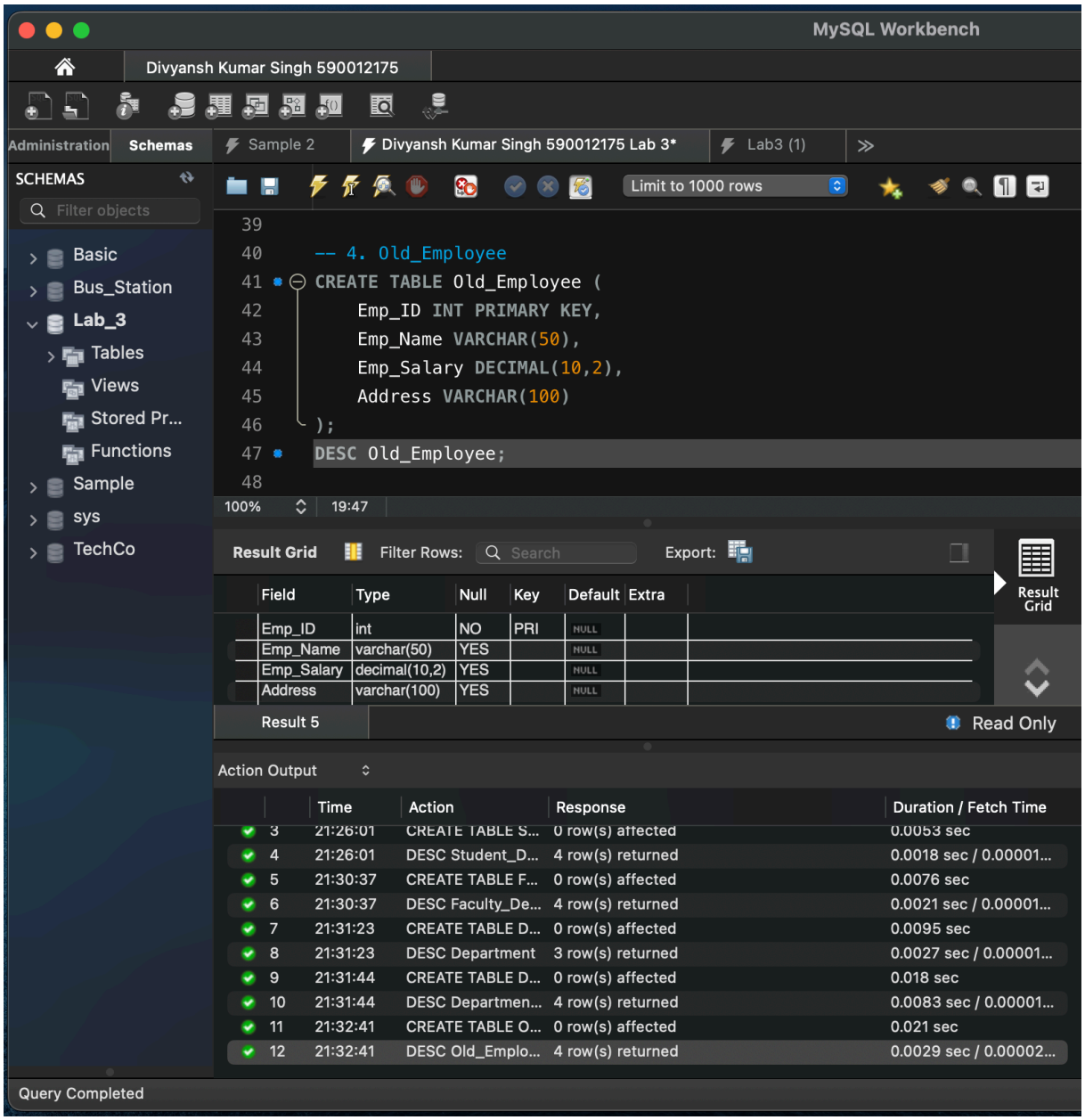
3. Department
- Columns: Dept_ID, Faculty_ID, Dept_Name



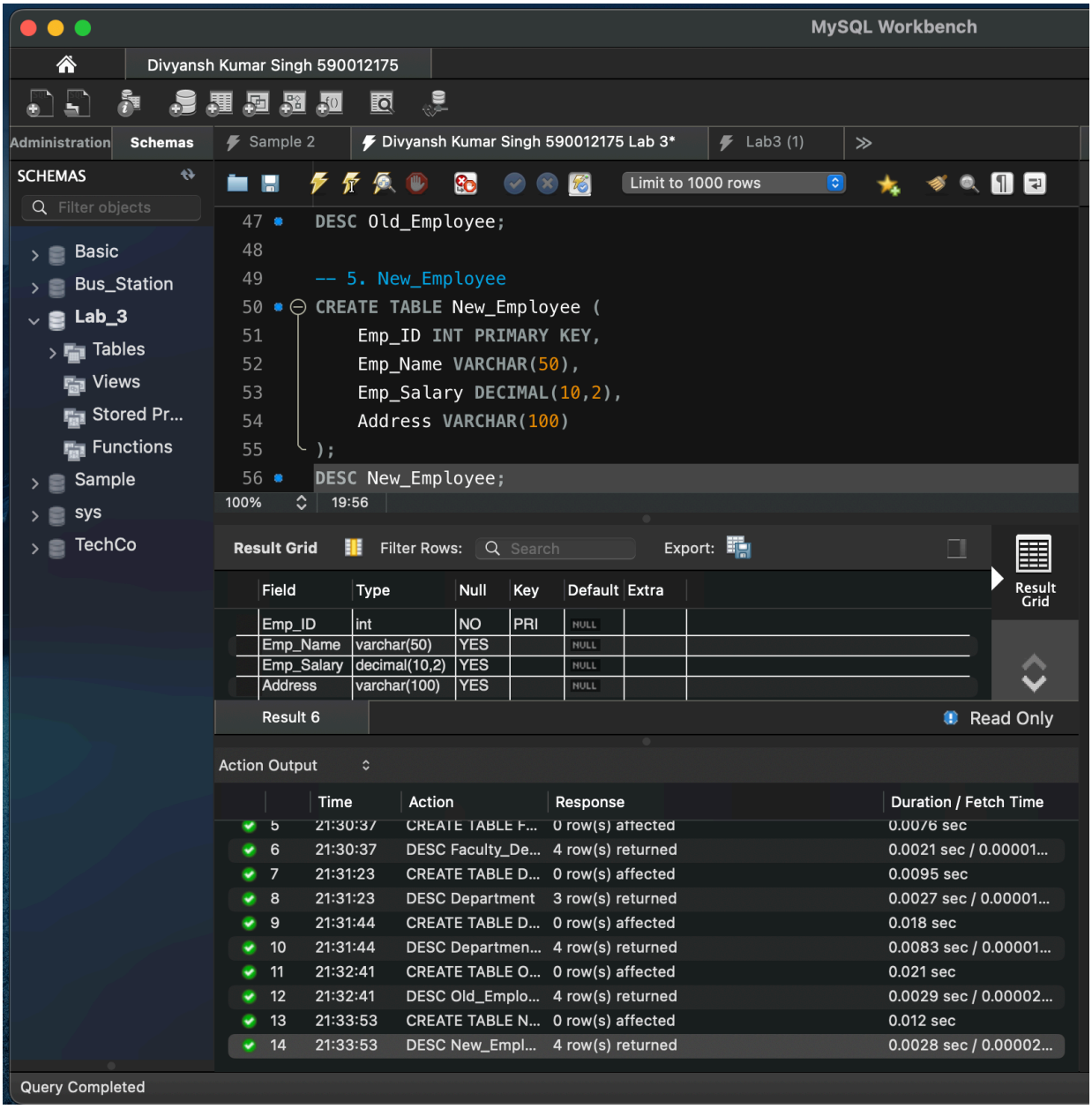

MySQL Workbench
Divyansh Kumar Singh 590012175
Administration
Schemas
Sample 2
Divyansh Kumar Singh 590012175 Lab 3*
Lab3 (1)
SCHEMAS
Filter objects
Basic
Bus_Station
Lab_3
Tables
Views
Stored Pr...
Functions
Sample
sys
TechCo
Limit to 1000 rows
29 DESC Department;
30
31 -- 3b. Department_Emp
32 CREATE TABLE Department_Emp (
33 Dept_ID INT PRIMARY KEY,
34 Dept_Name VARCHAR(50),
35 Emp_ID INT,
36 Dept_Grade CHAR(1)
37);
38 DESC Department_Emp;
39
100%
21:38
Result Grid
Filter Rows:
Search
Export:
Field Type Null Key Default Extra
Dept_ID int NO PRI NULL
Dept_Name varchar(50) YES NULL
Emp_ID int YES NULL
Dept_Grade char(1) YES NULL
Result Grid
Result 4
Read Only
Action Output
Time Action Response Duration / Fetch Time
1 21:25:54 CREATE DATABA... 1 row(s) affected 0.0015 sec
2 21:25:54 USE Lab_3 0 row(s) affected 0.00024 sec
3 21:26:01 CREATE TABLE S... 0 row(s) affected 0.0053 sec
4 21:26:01 DESC Student_D... 4 row(s) returned 0.0018 sec / 0.00001...
5 21:30:37 CREATE TABLE F... 0 row(s) affected 0.0076 sec
6 21:30:37 DESC Faculty_De... 4 row(s) returned 0.0021 sec / 0.00001...
7 21:31:23 CREATE TABLE D... 0 row(s) affected 0.0095 sec
8 21:31:23 DESC Department 3 row(s) returned 0.0027 sec / 0.00001...
9 21:31:44 CREATE TABLE D... 0 row(s) affected 0.018 sec
10 21:31:44 DESC Departmen... 4 row(s) returned 0.0083 sec / 0.00001...

4. Old_Employee

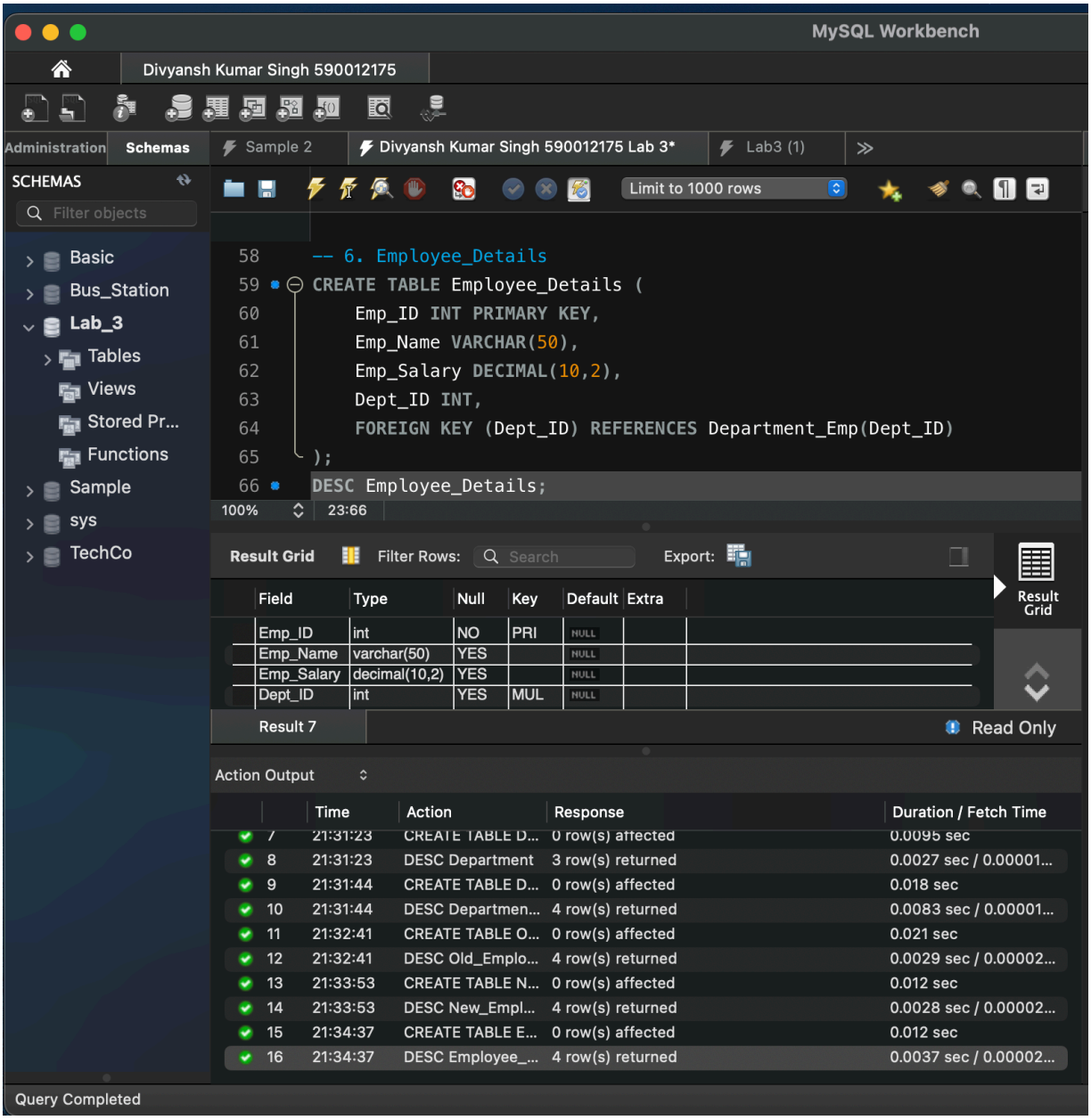
- Columns: Emp_ID, Emp_Name, Emp_Salary, Address



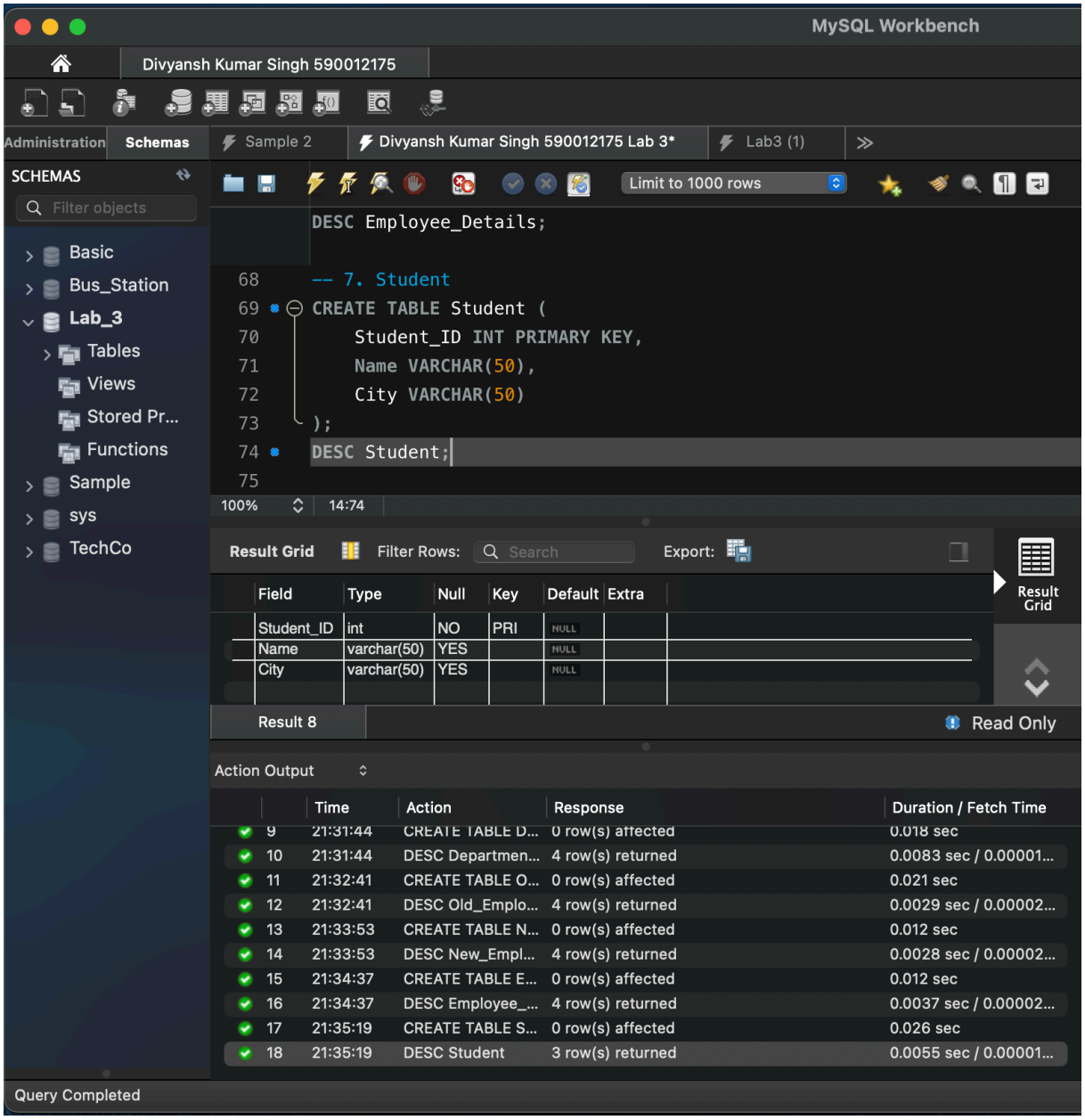
5. New_Employee
- Columns: Emp_ID, Emp_Name, Emp_Salary, Address



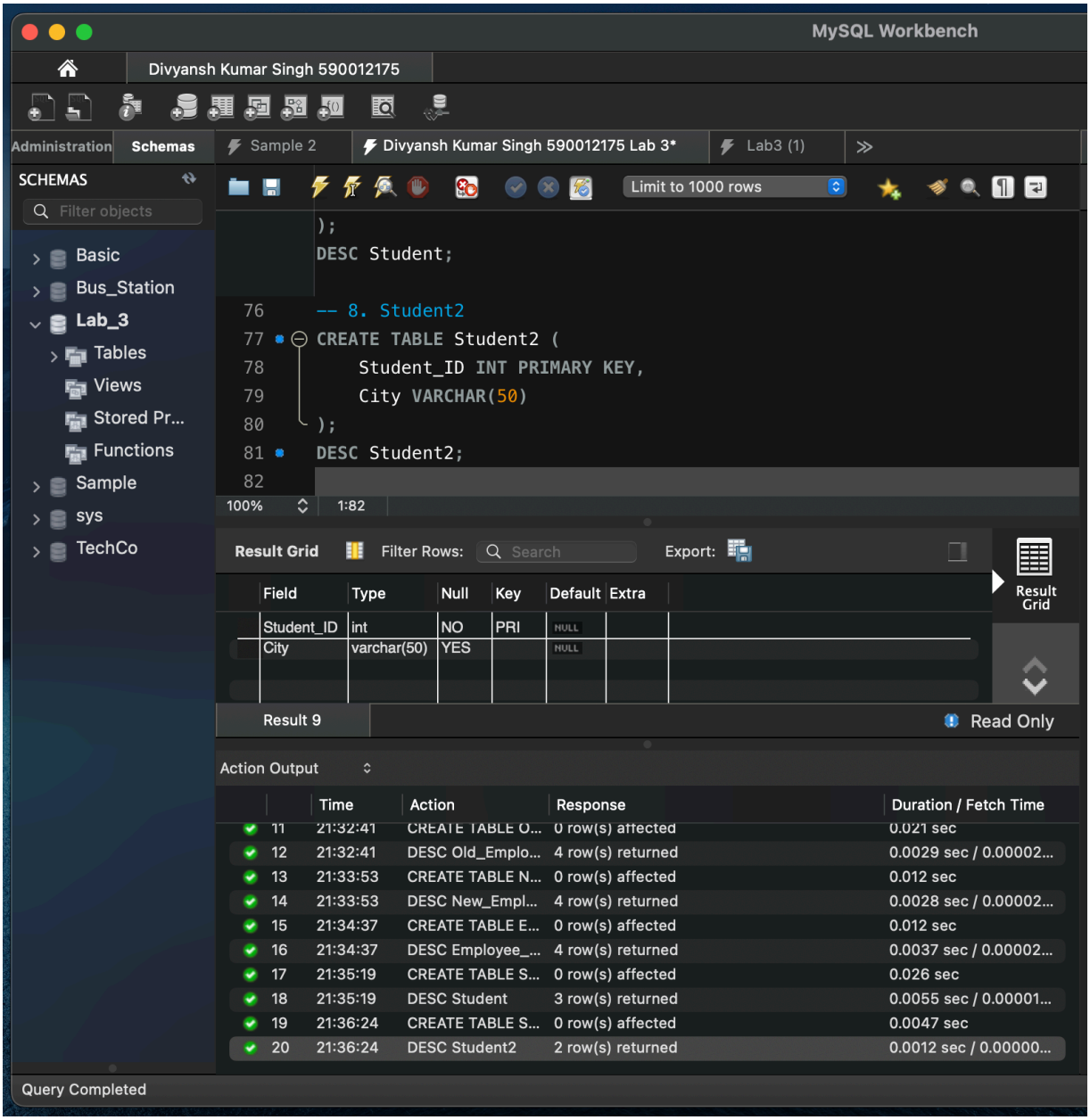
6. Employee_Details
- Columns: Emp_ID, Emp_Name, Emp_Salary, Dept_ID



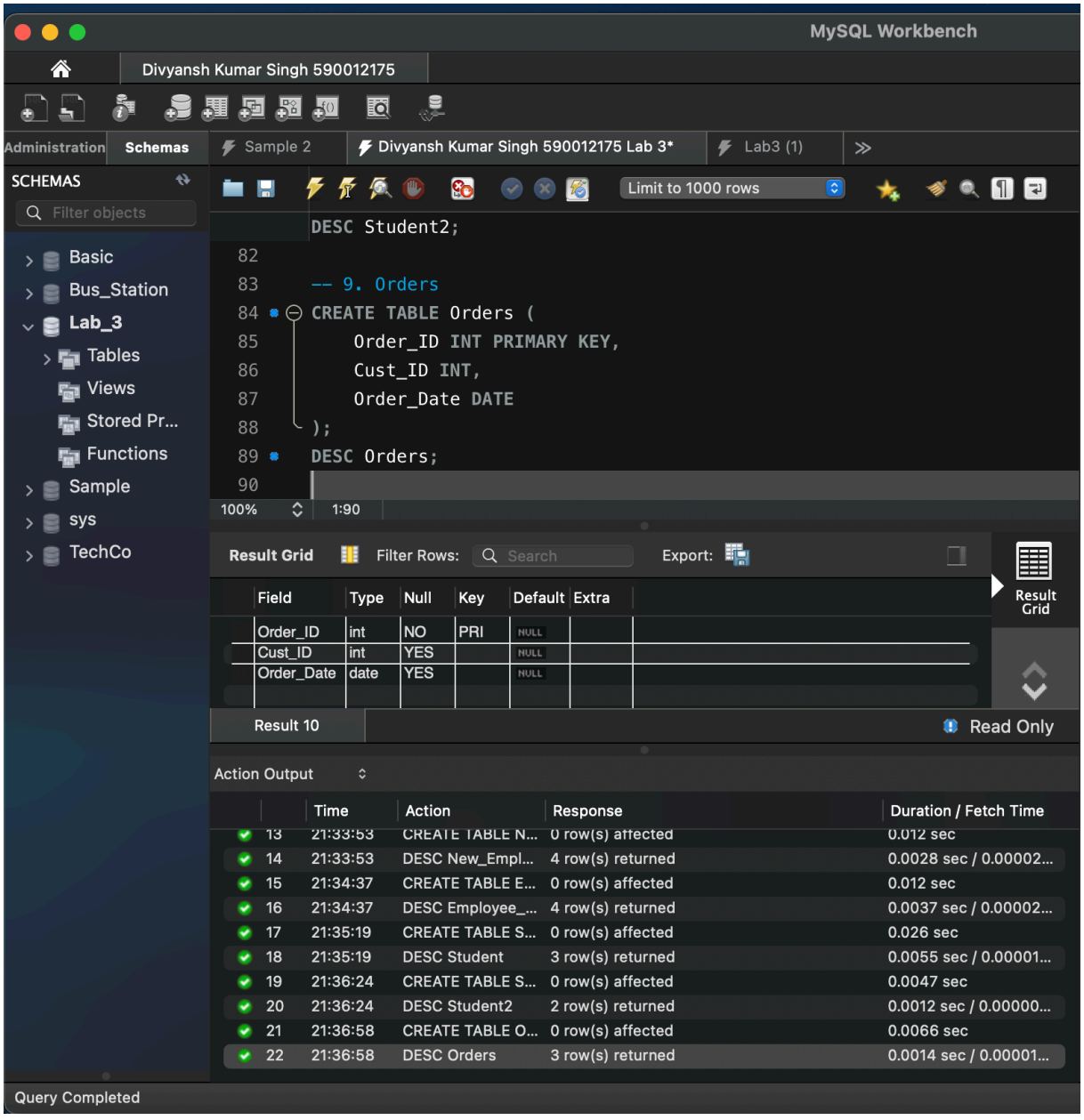
7. Student
- Columns: Student_ID, Name, City



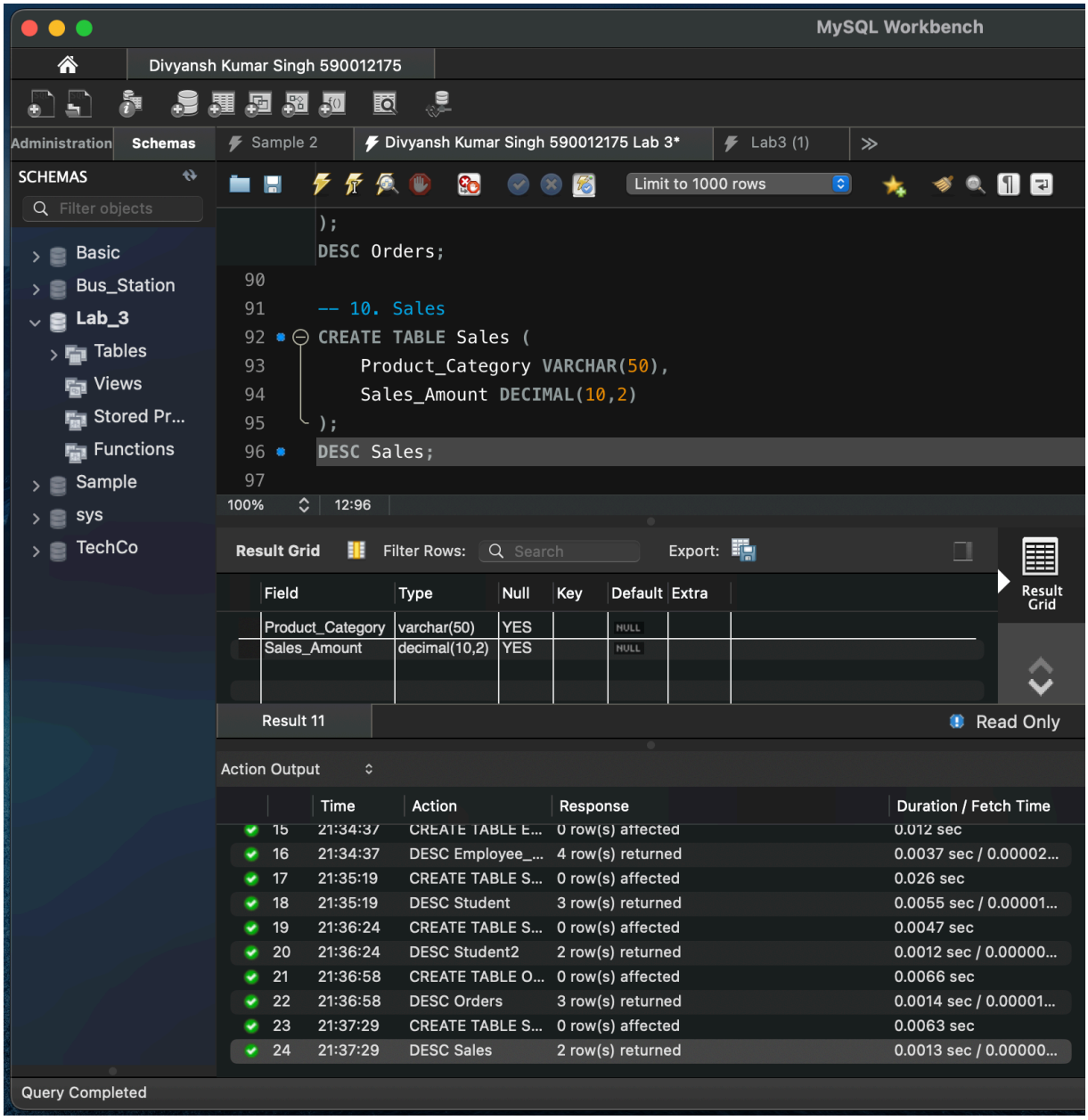
8. Student2
- Columns: Student_ID, City



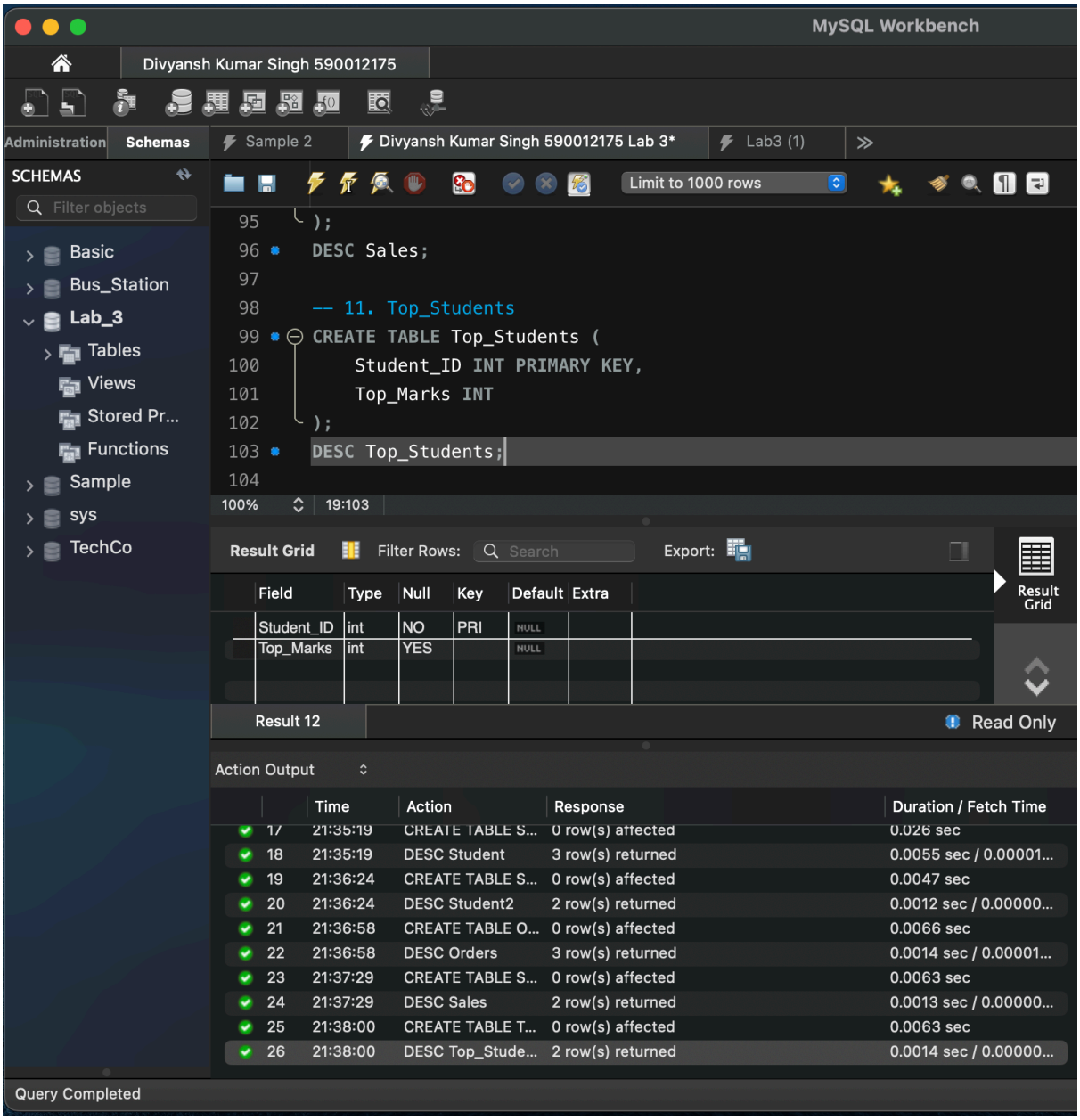
9. Orders
- Columns: Order_ID, Cust_ID, Order_Date



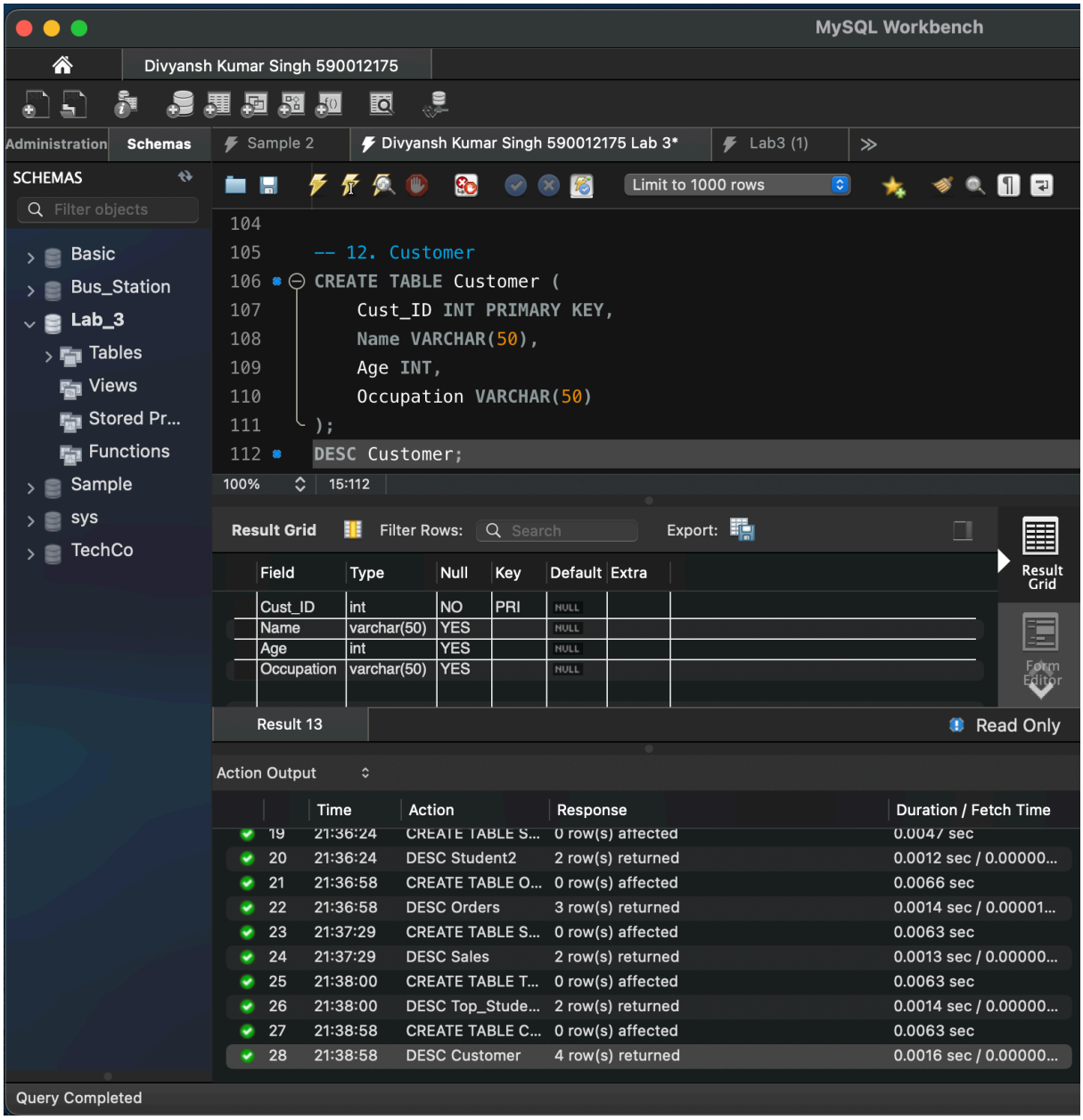
10. Sales
- Columns: Product_Category, Sales_Amount



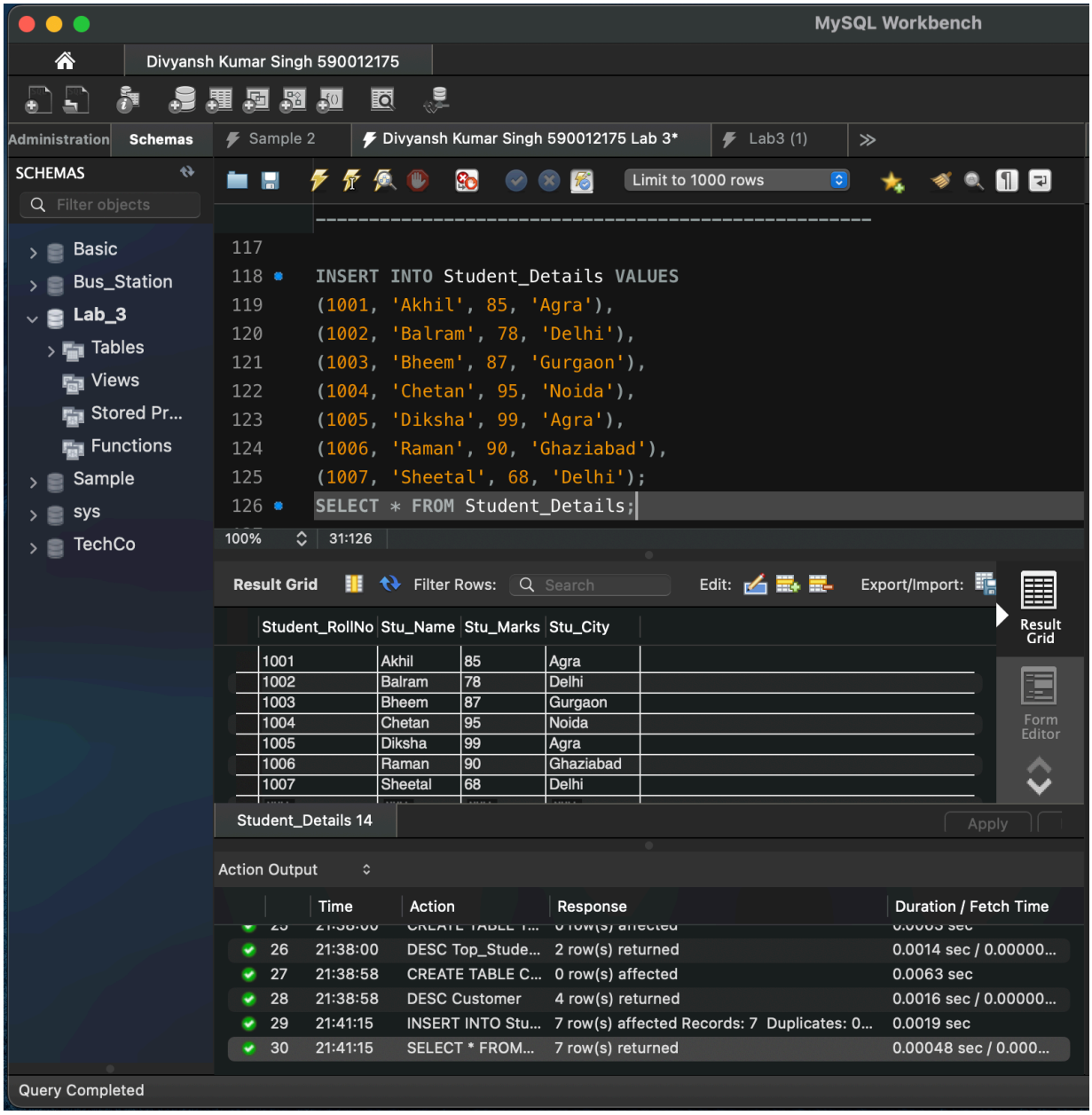
11. Top_Students
- Columns: Student_ID, Top_Marks

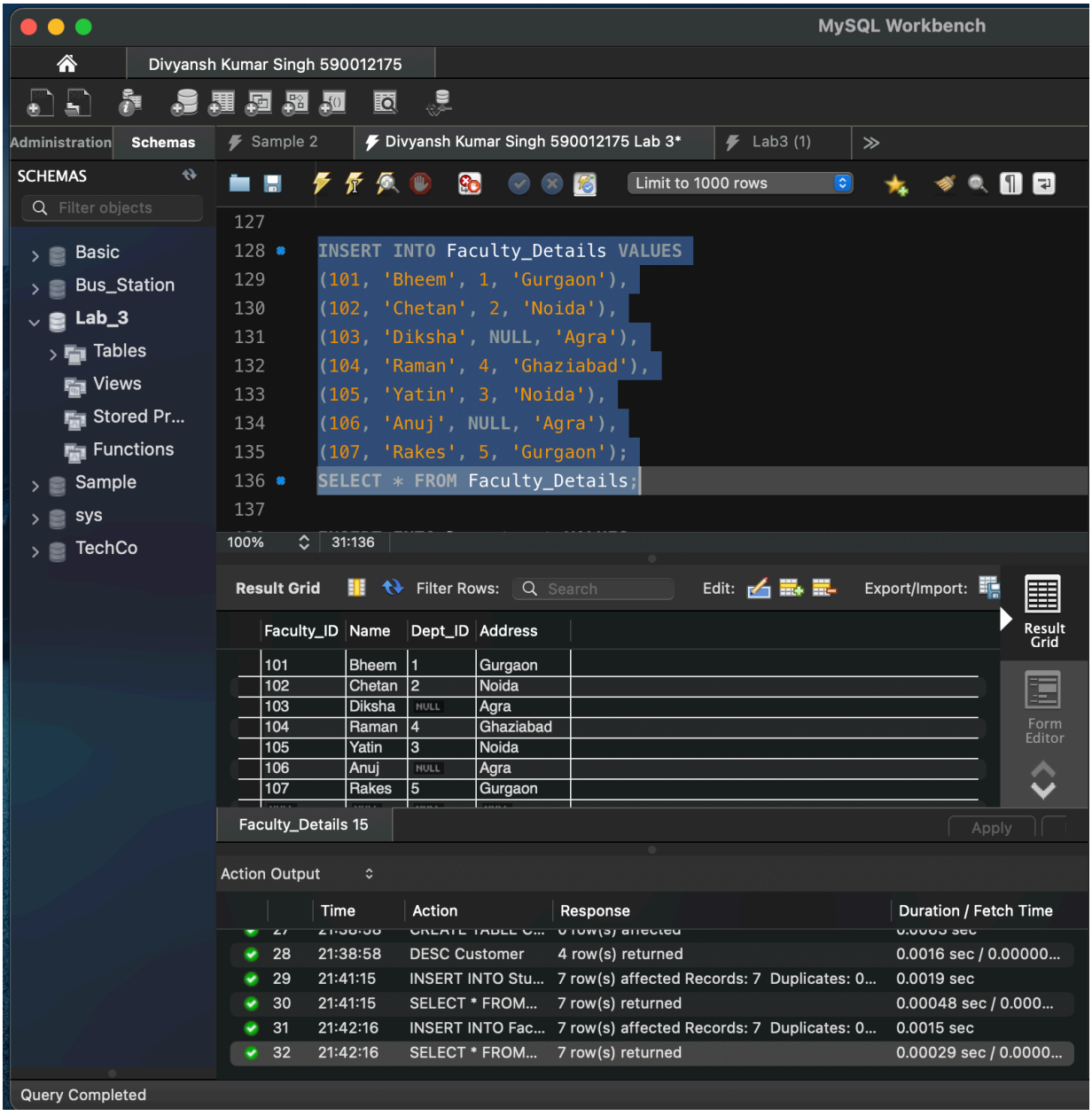


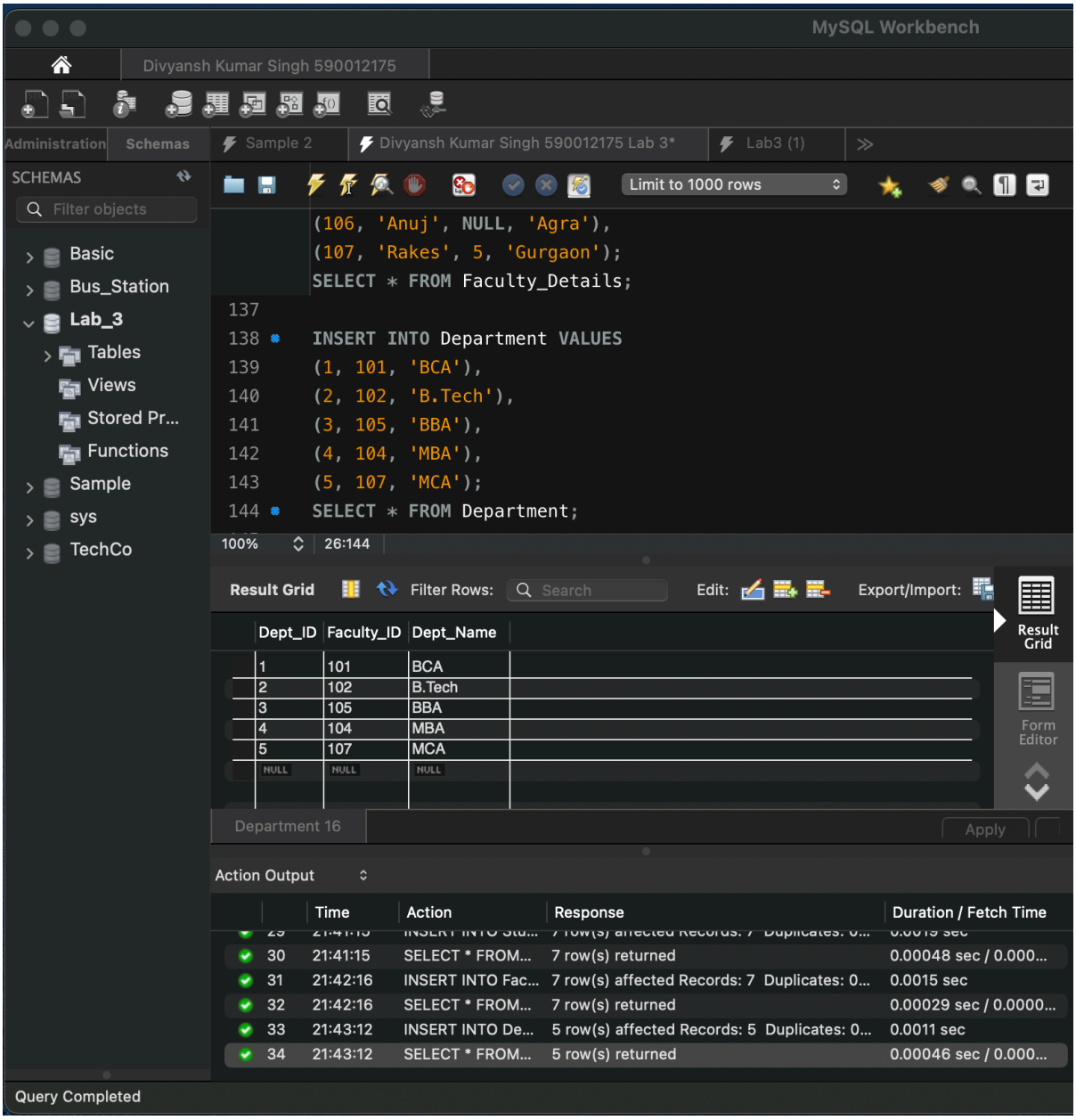
Customer Table because its required for few subquery question.

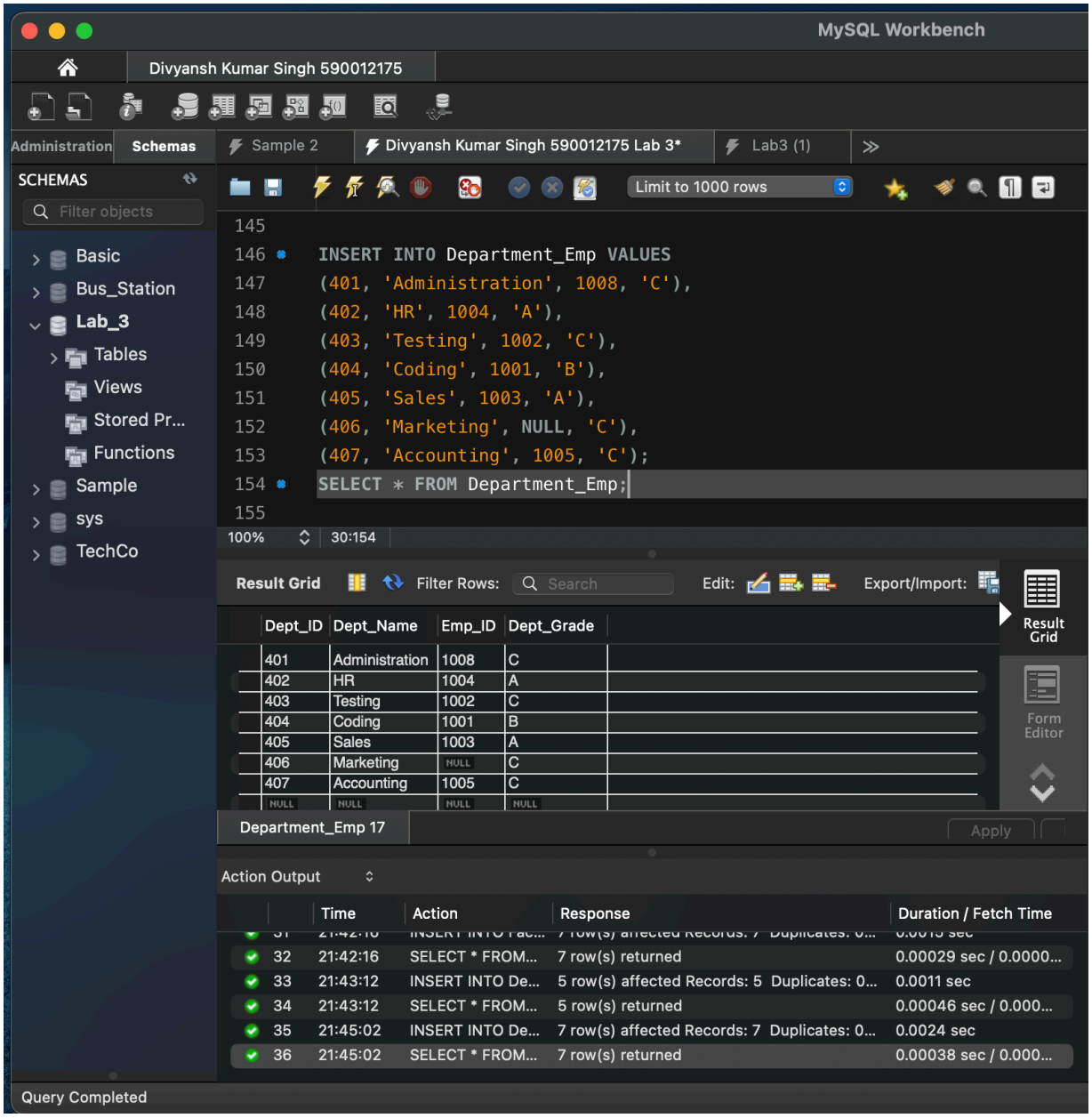


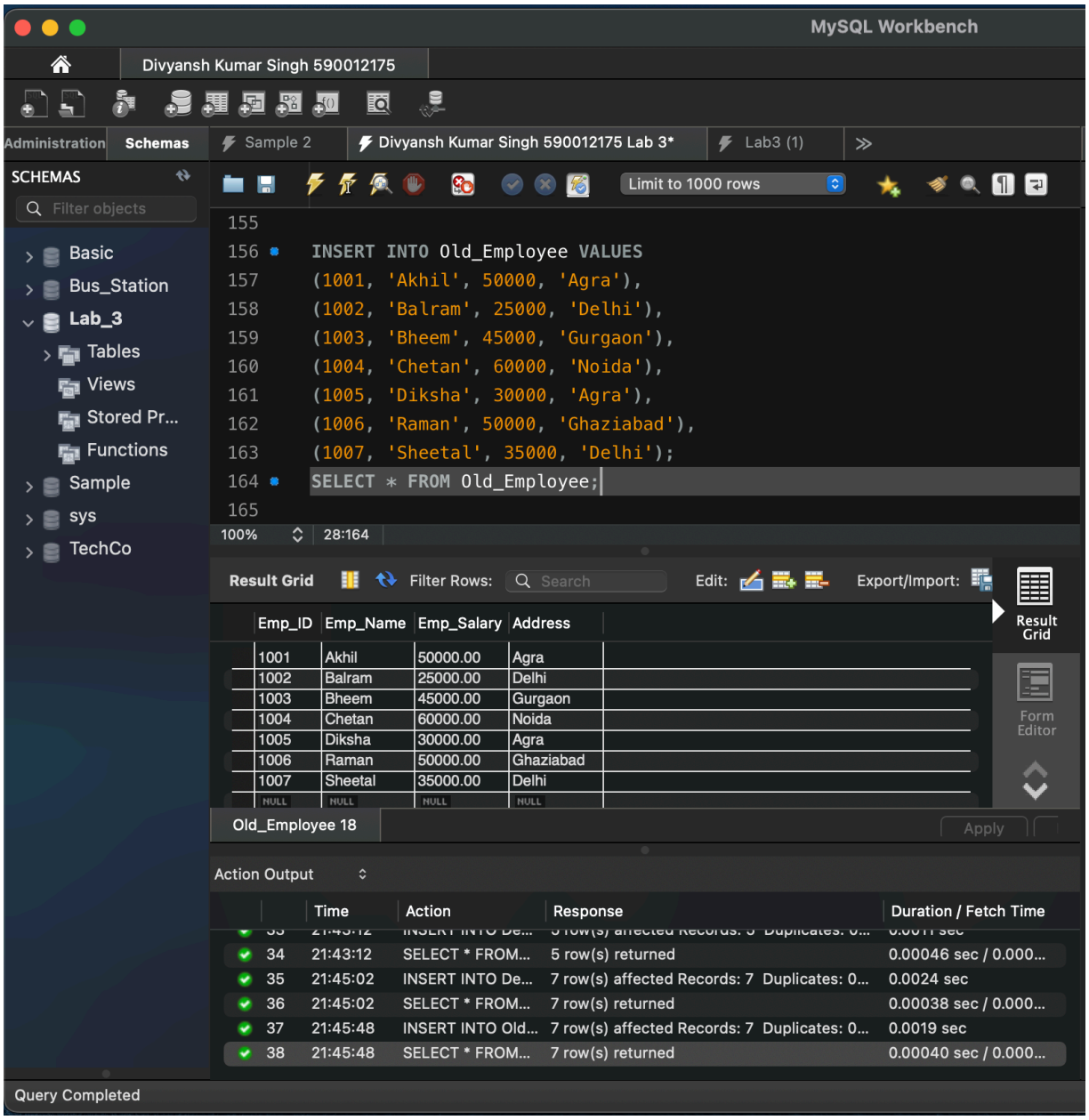
Inserting data into the tables created:

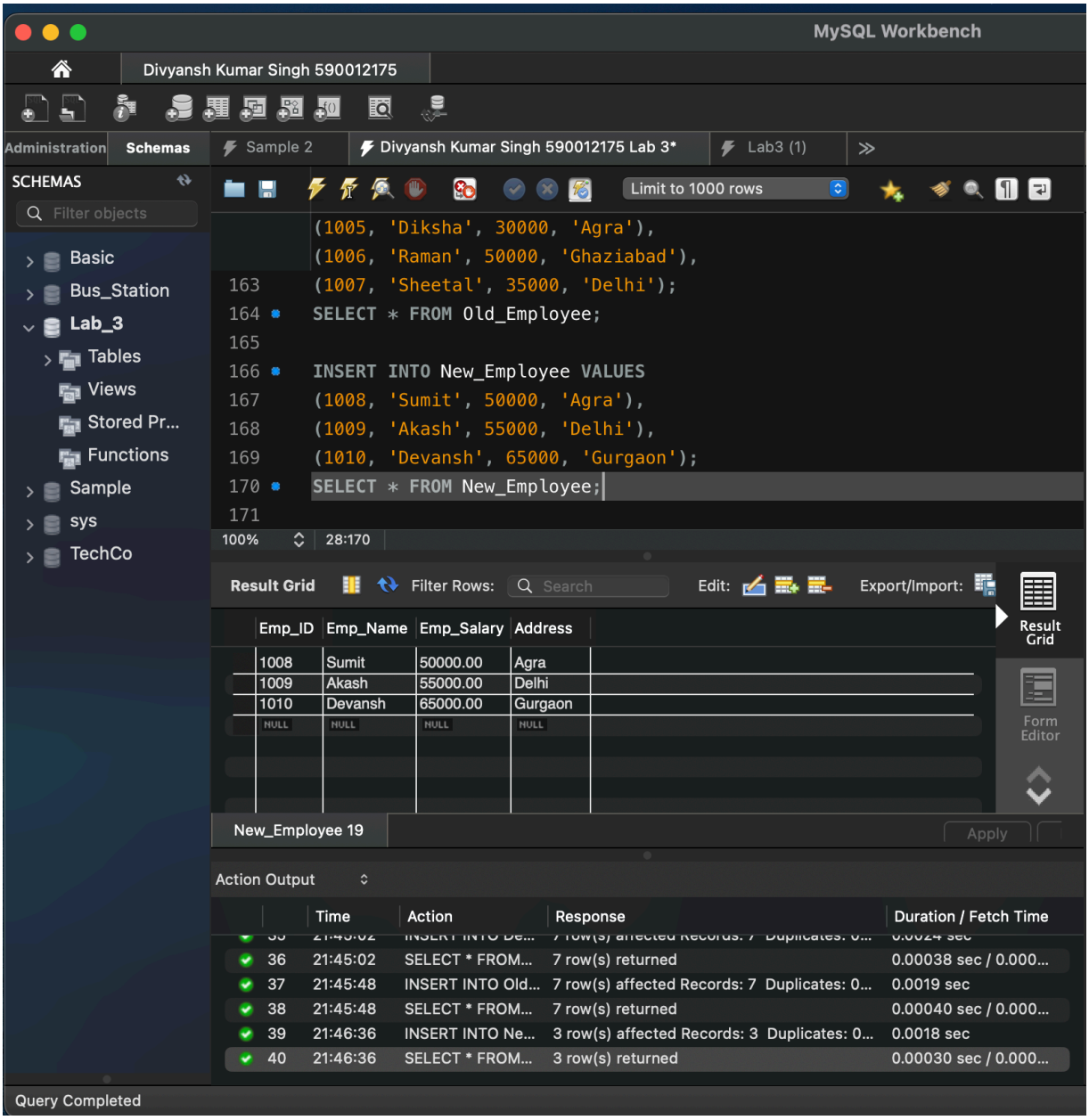


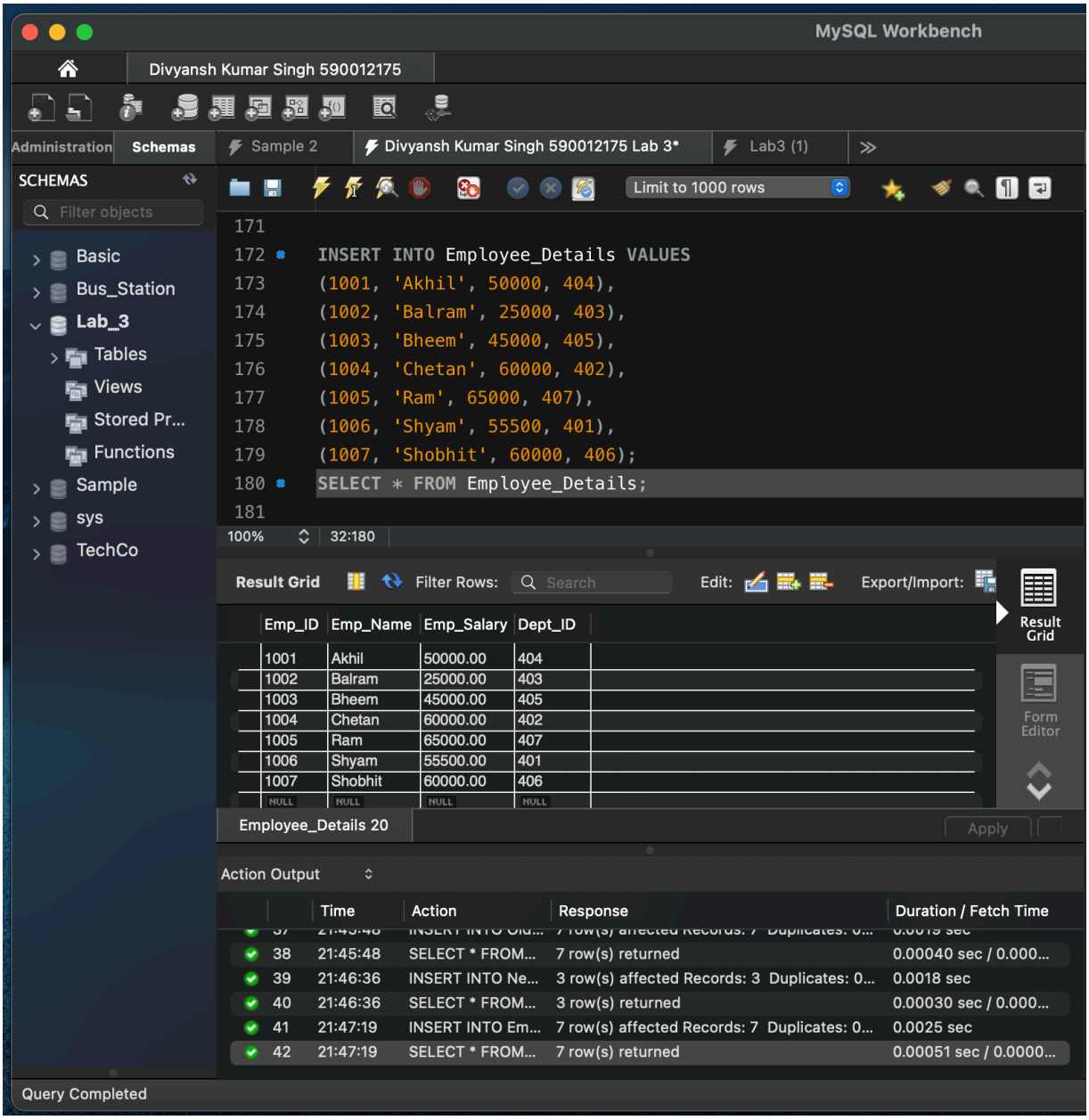

MySQL Workbench
Divyansh Kumar Singh 590012175
Administration
Schemas
Sample 2
Divyansh Kumar Singh 590012175 Lab 3*
Lab3 (1)
SCHEMAS
Filter objects
Basic
Bus_Station
Lab_3
Tables
Views
Stored Pr...
Functions
Sample
sys
TechCo
Limit to 1000 rows
127
128 INSERT INTO Faculty_Details VALUES
129 (101, 'Bheem', 1, 'Gurgaon'),
130 (102, 'Chetan', 2, 'Noida'),
131 (103, 'Diksha', NULL, 'Agra'),
132 (104, 'Raman', 4, 'Ghaziabad'),
133 (105, 'Yatin', 3, 'Noida'),
134 (106, 'Anuj', NULL, 'Agra'),
135 (107, 'Rakes', 5, 'Gurgaon');
136 SELECT * FROM Faculty_Details;
137
100%
31:136
Result Grid
Filter Rows:
Search
Edit:
Export/Import:
Result Grid
Form Editor
Faculty_ID Name Dept_ID Address
101 Bheem 1 Gurgaon
102 Chetan 2 Noida
103 Diksha NULL Agra
104 Raman 4 Ghaziabad
105 Yatin 3 Noida
106 Anuj NULL Agra
107 Rakes 5 Gurgaon
Faculty_Details 15
Apply
Action Output
Time Action Response Duration / Fetch Time
28 21:38:58 DESC Customer 4 row(s) returned 0.0016 sec / 0.00000...
29 21:41:15 INSERT INTO Stu... 7 row(s) affected Records: 7 Duplicates: 0... 0.0019 sec
30 21:41:15 SELECT * FROM... 7 row(s) returned 0.00048 sec / 0.000...
31 21:42:16 INSERT INTO Fac... 7 row(s) affected Records: 7 Duplicates: 0... 0.0015 sec
32 21:42:16 SELECT * FROM... 7 row(s) returned 0.00029 sec / 0.0000...
Query Completed

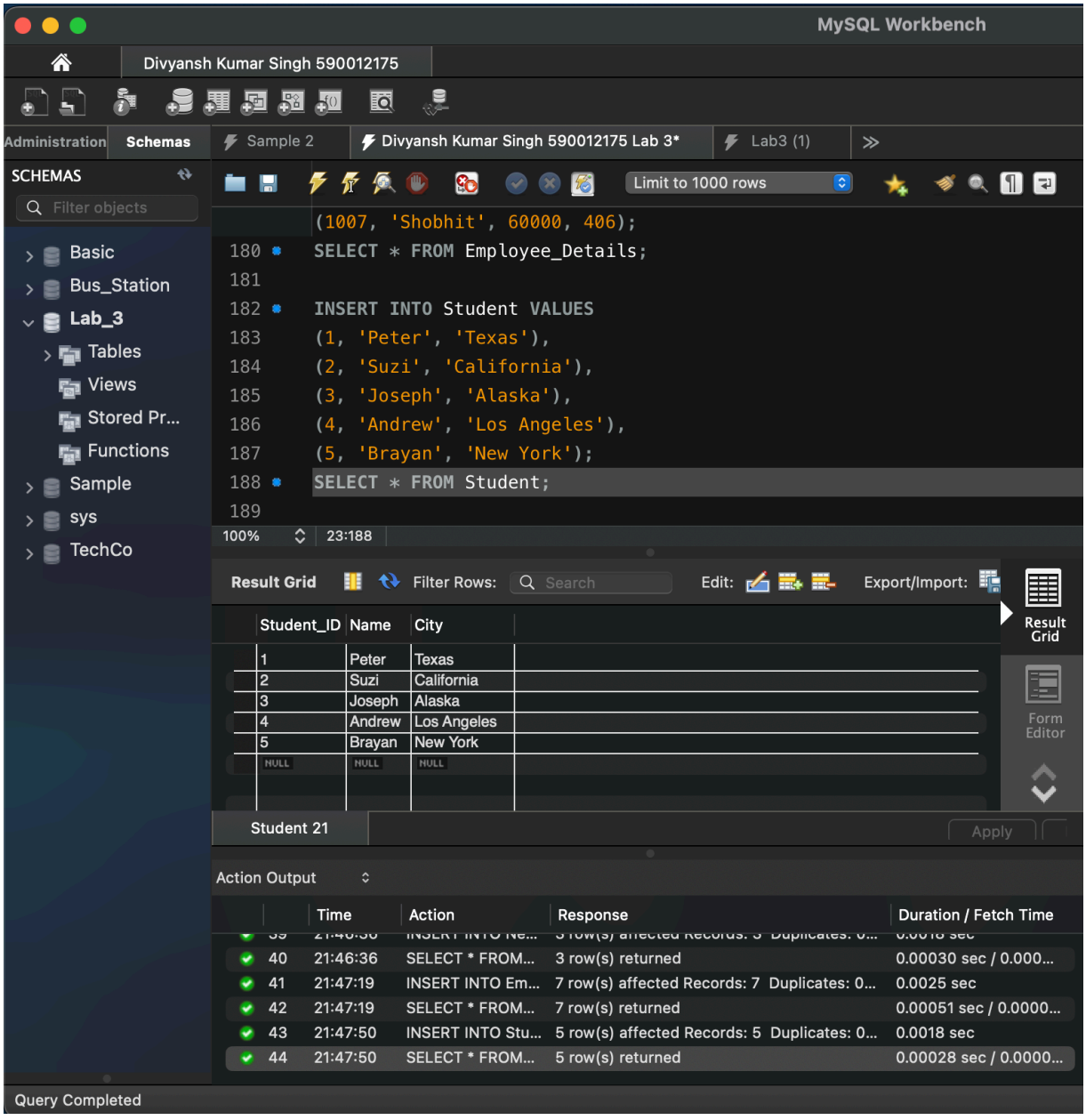

MySQL Workbench
Divyansh Kumar Singh 590012175
Administration
Schemas
Sample 2
Divyansh Kumar Singh 590012175 Lab 3*
Lab3 (1)
SCHEMAS
Filter objects
Basic
Bus_Station
Lab_3
Tables
Views
Stored Pr...
Functions
Sample
sys
TechCo
Limit to 1000 rows
(106, 'Anuj', NULL, 'Agra'),
(107, 'Rakes', 5, 'Gurgaon');
SELECT * FROM Faculty_Details;
INSERT INTO Department VALUES
(1, 101, 'BCA'),
(2, 102, 'B.Tech'),
(3, 105, 'BBA'),
(4, 104, 'MBA'),
(5, 107, 'MCA');
SELECT * FROM Department;
100%
26:144
Result Grid
Filter Rows:
Search
Edit:
Export/Import:
Dept_ID Faculty_ID Dept_Name
1 101 BCA
2 102 B.Tech
3 105 BBA
4 104 MBA
5 107 MCA
Result Grid
Form Editor
Department 16
Apply
Action Output
Time Action Response Duration / Fetch Time
30 21:41:15 SELECT * FROM... 7 row(s) returned 0.00048 sec / 0.000...
31 21:42:16 INSERT INTO Fac... 7 row(s) affected Records: 7 Duplicates: 0... 0.0015 sec
32 21:42:16 SELECT * FROM... 7 row(s) returned 0.00029 sec / 0.0000...
33 21:43:12 INSERT INTO De... 5 row(s) affected Records: 5 Duplicates: 0... 0.0011 sec
34 21:43:12 SELECT * FROM... 5 row(s) returned 0.00046 sec / 0.000...
Query Completed

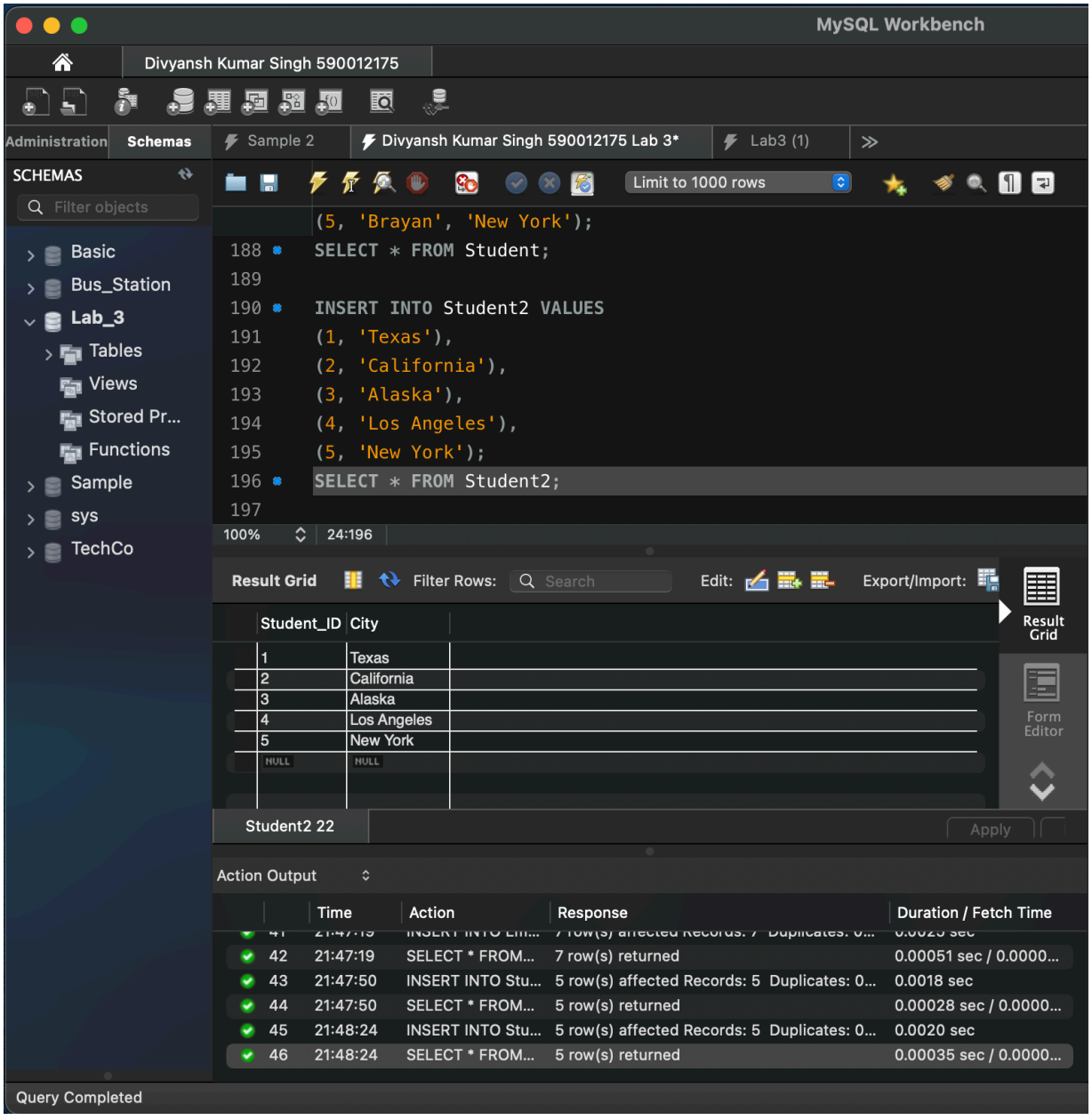

MySQL Workbench
Divyansh Kumar Singh 590012175
Administration
Schemas
Sample 2
Divyansh Kumar Singh 590012175 Lab 3*
Lab3 (1)
SCHEMAS
Filter objects
Basic
Bus_Station
Lab_3
Tables
Views
Stored Pr...
Functions
Sample
sys
TechCo
Limit to 1000 rows
145
146 INSERT INTO Department_Emp VALUES
147 (401, 'Administration', 1008, 'C'),
148 (402, 'HR', 1004, 'A'),
149 (403, 'Testing', 1002, 'C'),
150 (404, 'Coding', 1001, 'B'),
151 (405, 'Sales', 1003, 'A'),
152 (406, 'Marketing', NULL, 'C'),
153 (407, 'Accounting', 1005, 'C');
154 SELECT * FROM Department_Emp;
155
100%
30:154
Result Grid
Filter Rows:
Search
Edit:
Export/Import:
Dept_ID Dept_Name Emp_ID Dept_Grade
401 Administration 1008 C
402 HR 1004 A
403 Testing 1002 C
404 Coding 1001 B
405 Sales 1003 A
406 Marketing NULL C
407 Accounting 1005 C
Result Grid
Form Editor
Department_Emp 17
Apply
Action Output
Time Action Response Duration / Fetch Time
32 21:42:16 SELECT * FROM... 7 row(s) returned 0.00029 sec / 0.0000...
33 21:43:12 INSERT INTO De... 5 row(s) affected Records: 5 Duplicates: 0... 0.0011 sec
34 21:43:12 SELECT * FROM... 5 row(s) returned 0.00046 sec / 0.000...
35 21:45:02 INSERT INTO De... 7 row(s) affected Records: 7 Duplicates: 0... 0.0024 sec
36 21:45:02 SELECT * FROM... 7 row(s) returned 0.00038 sec / 0.000...
Query Completed

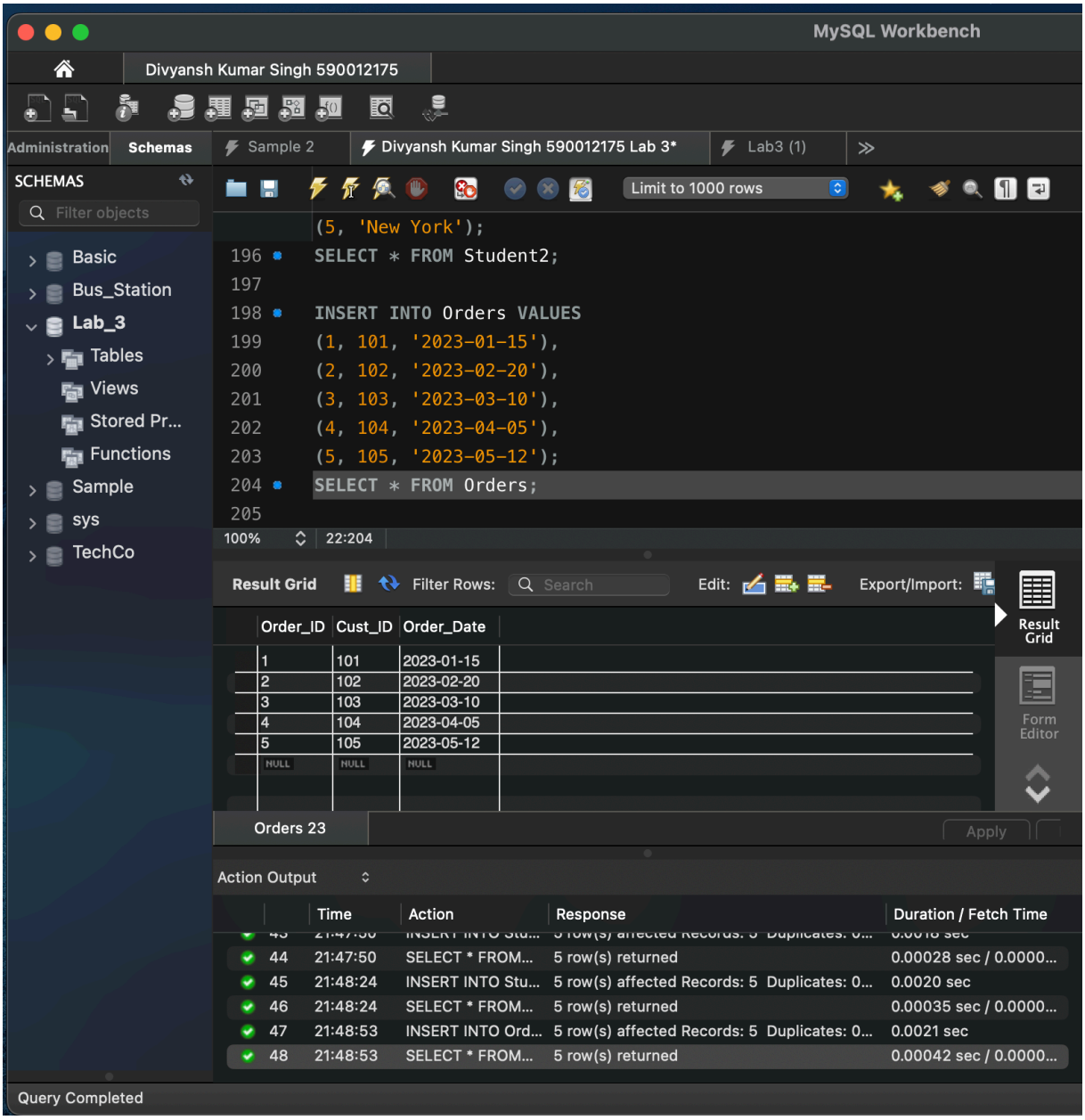

MySQL Workbench
Divyansh Kumar Singh 590012175
Administration
Schemas
Sample 2
Divyansh Kumar Singh 590012175 Lab 3*
Lab3 (1)
SCHEMAS
Filter objects
Basic
Bus_Station
Lab_3
Tables
Views
Stored Pr...
Functions
Sample
sys
TechCo
Limit to 1000 rows
155
156 INSERT INTO Old_Employee VALUES
157 (1001, 'Akhil', 50000, 'Agra'),
158 (1002, 'Balram', 25000, 'Delhi'),
159 (1003, 'Bheem', 45000, 'Gurgaon'),
160 (1004, 'Chetan', 60000, 'Noida'),
161 (1005, 'Diksha', 30000, 'Agra'),
162 (1006, 'Raman', 50000, 'Ghaziabad'),
163 (1007, 'Sheetal', 35000, 'Delhi');
164 SELECT * FROM Old_Employee;
165
100%
28:164
Result Grid
Filter Rows:
Search
Edit:
Export/Import:
Emp_ID Emp_Name Emp_Salary Address
1001 Akhil 50000.00 Agra
1002 Balram 25000.00 Delhi
1003 Bheem 45000.00 Gurgaon
1004 Chetan 60000.00 Noida
1005 Diksha 30000.00 Agra
1006 Raman 50000.00 Ghaziabad
1007 Sheetal 35000.00 Delhi
Result Grid
Form Editor
Old_Employee 18
Apply
Action Output
Time Action Response Duration / Fetch Time
34 21:43:12 SELECT * FROM... 5 row(s) returned 0.00046 sec / 0.000...
35 21:45:02 INSERT INTO De... 7 row(s) affected Records: 7 Duplicates: 0... 0.0024 sec
36 21:45:02 SELECT * FROM... 7 row(s) returned 0.00038 sec / 0.000...
37 21:45:48 INSERT INTO Old... 7 row(s) affected Records: 7 Duplicates: 0... 0.0019 sec
38 21:45:48 SELECT * FROM... 7 row(s) returned 0.00040 sec / 0.000...
Query Completed

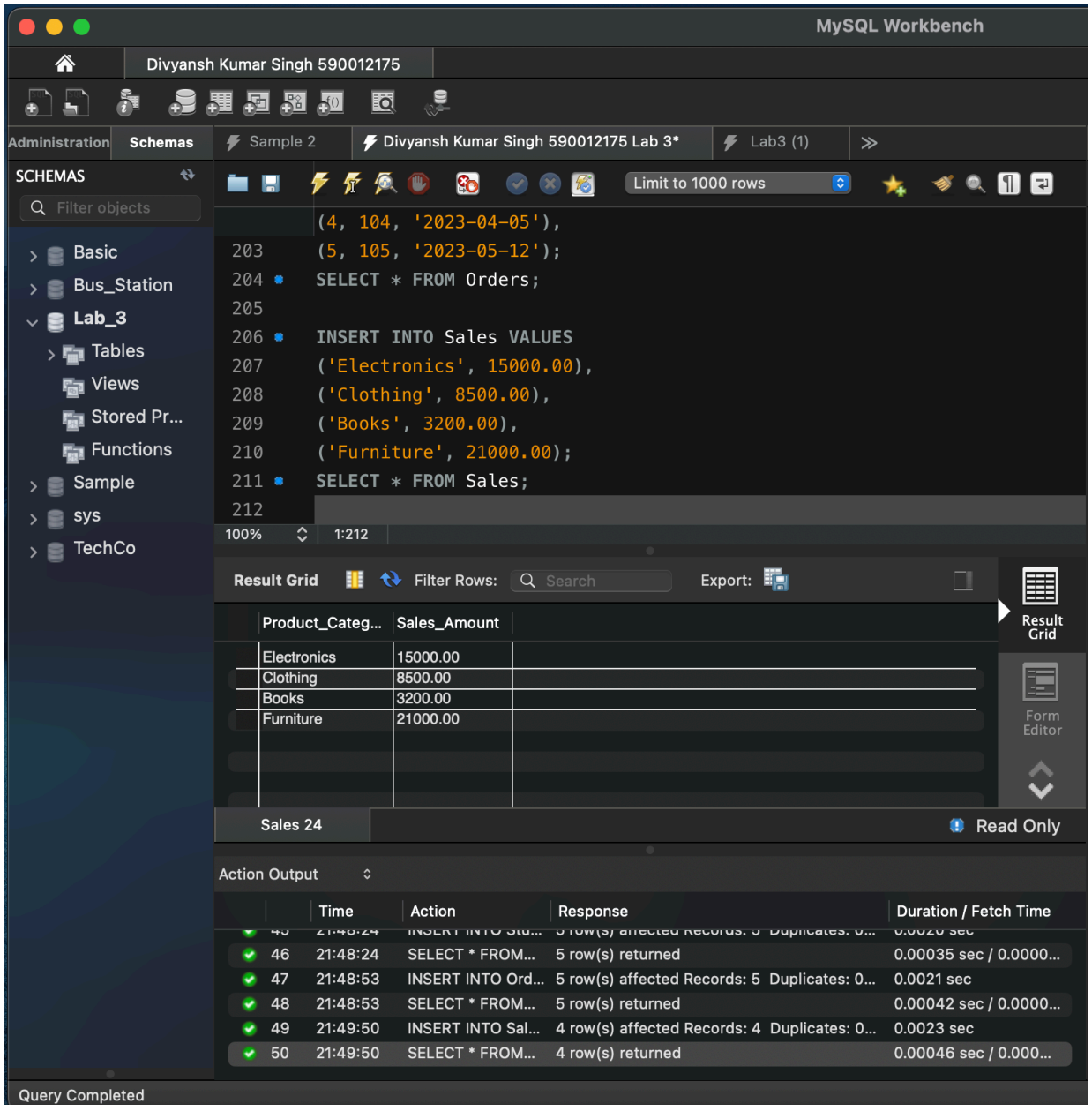

MySQL Workbench
Divyansh Kumar Singh 590012175
Administration
Schemas
Sample 2
Divyansh Kumar Singh 590012175 Lab 3*
Lab3 (1)
SCHEMAS
Filter objects
Basic
Bus_Station
Lab_3
Tables
Views
Stored Pr...
Functions
Sample
sys
TechCo
Limit to 1000 rows
(1005, 'Diksha', 30000, 'Agra'),
(1006, 'Raman', 50000, 'Ghaziabad'),
163 (1007, 'Sheetal', 35000, 'Delhi');
164 SELECT * FROM Old_Employee;
165
166 INSERT INTO New_Employee VALUES
167 (1008, 'Sumit', 50000, 'Agra'),
168 (1009, 'Akash', 55000, 'Delhi'),
169 (1010, 'Devansh', 65000, 'Gurgaon');
170 SELECT * FROM New_Employee;
171
100%
28:170
Result Grid
Filter Rows:
Search
Edit:
Export/Import:
Emp_ID Emp_Name Emp_Salary Address
1008 Sumit 50000.00 Agra
1009 Akash 55000.00 Delhi
1010 Devansh 65000.00 Gurgaon
NULL NULL NULL NULL
Result Grid
Form Editor
New_Employee 19
Apply
Action Output
Time Action Response Duration / Fetch Time
36 21:45:02 SELECT * FROM... 7 row(s) returned 0.00038 sec / 0.000...
37 21:45:48 INSERT INTO Old... 7 row(s) affected Records: 7 Duplicates: 0... 0.0019 sec
38 21:45:48 SELECT * FROM... 7 row(s) returned 0.00040 sec / 0.000...
39 21:46:36 INSERT INTO Ne... 3 row(s) affected Records: 3 Duplicates: 0... 0.0018 sec
40 21:46:36 SELECT * FROM... 3 row(s) returned 0.00030 sec / 0.000...
Query Completed

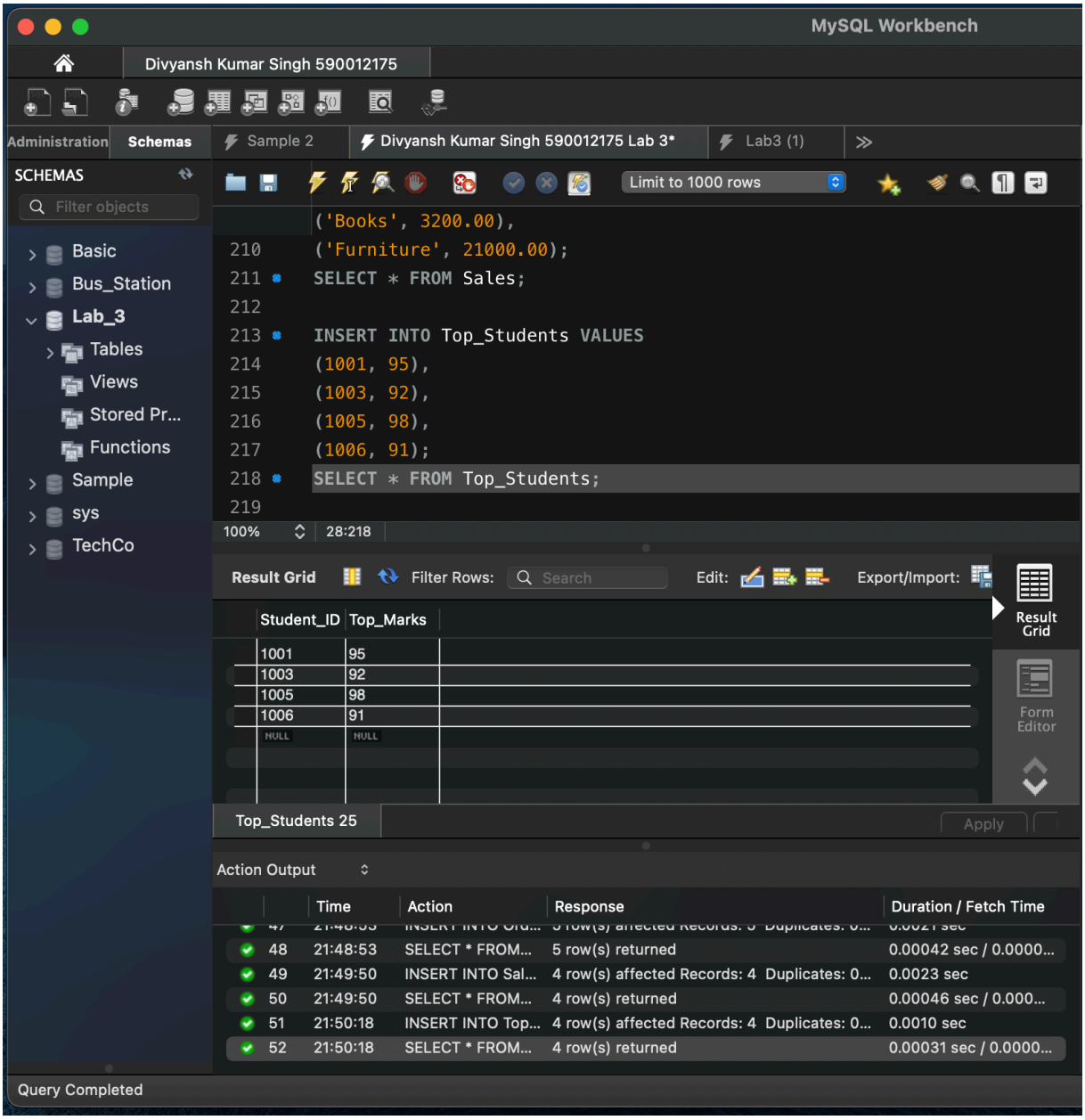

MySQL Workbench
Divyansh Kumar Singh 590012175
Administration
Schemas
Sample 2
Divyansh Kumar Singh 590012175 Lab 3*
Lab3 (1)
SCHEMAS
Filter objects
Basic
Bus_Station
Lab_3
Tables
Views
Stored Pr...
Functions
Sample
sys
TechCo
Limit to 1000 rows
171
172 INSERT INTO Employee_Details VALUES
173 (1001, 'Akhil', 50000, 404),
174 (1002, 'Balram', 25000, 403),
175 (1003, 'Bheem', 45000, 405),
176 (1004, 'Chetan', 60000, 402),
177 (1005, 'Ram', 65000, 407),
178 (1006, 'Shyam', 55500, 401),
179 (1007, 'Shobhit', 60000, 406);
180 SELECT * FROM Employee_Details;
181
100%
32:180
Result Grid
Filter Rows:
Search
Edit:
Export/Import:
Emp_ID Emp_Name Emp_Salary Dept_ID
1001 Akhil 50000.00 404
1002 Balram 25000.00 403
1003 Bheem 45000.00 405
1004 Chetan 60000.00 402
1005 Ram 65000.00 407
1006 Shyam 55500.00 401
1007 Shobhit 60000.00 406
Result Grid
Form Editor
Employee_Details 20
Apply
Action Output
Time Action Response Duration / Fetch Time
38 21:45:48 SELECT * FROM... 7 row(s) returned 0.00040 sec / 0.000...
39 21:46:36 INSERT INTO Ne... 3 row(s) affected Records: 3 Duplicates: 0... 0.0018 sec
40 21:46:36 SELECT * FROM... 3 row(s) returned 0.00030 sec / 0.000...
41 21:47:19 INSERT INTO Em... 7 row(s) affected Records: 7 Duplicates: 0... 0.0025 sec
42 21:47:19 SELECT * FROM... 7 row(s) returned 0.00051 sec / 0.0000...
Query Completed

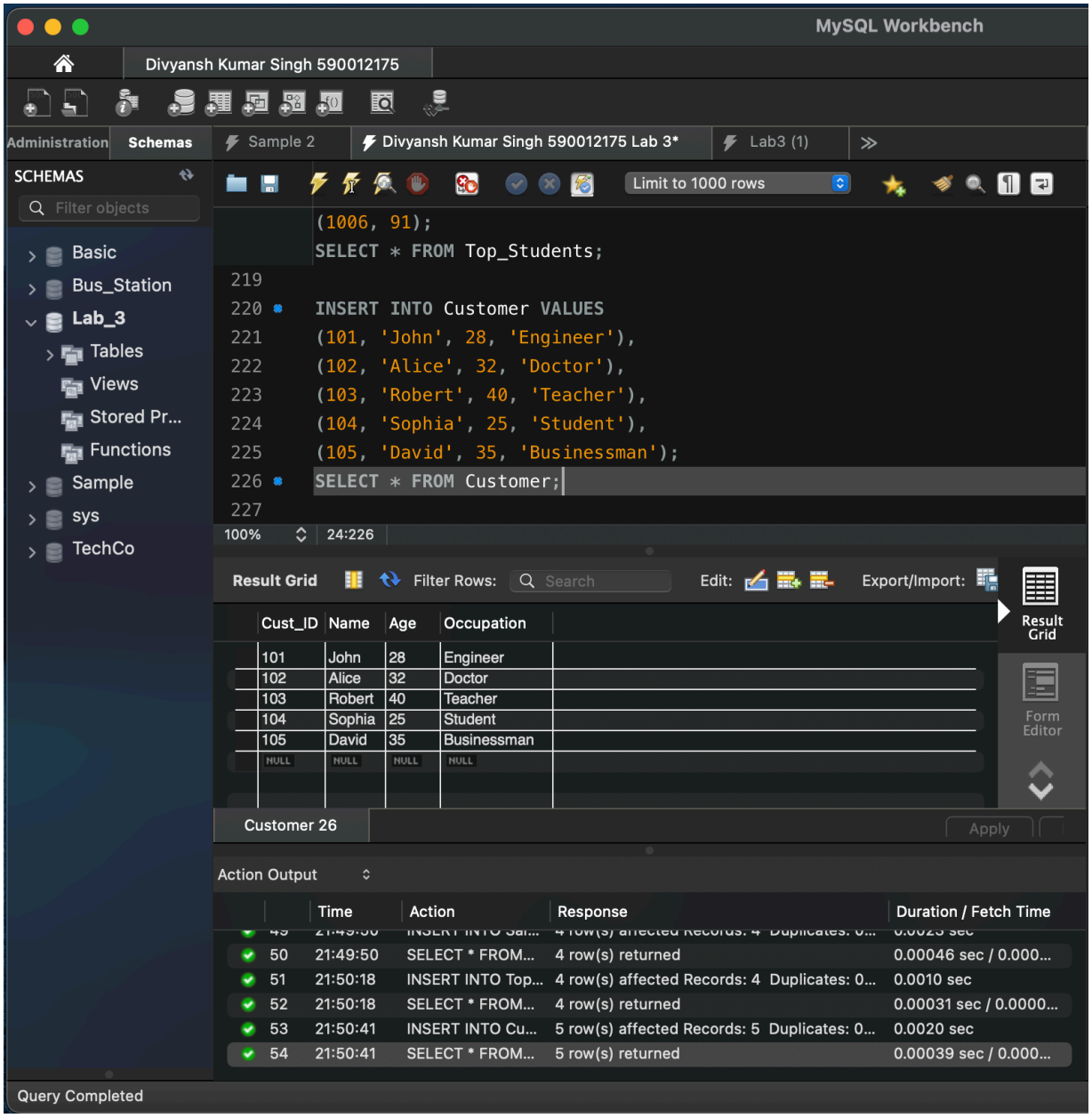

MySQL Workbench
Divyansh Kumar Singh 590012175
Administration
Schemas
Sample 2
Divyansh Kumar Singh 590012175 Lab 3*
Lab3 (1)
SCHEMAS
Filter objects
Basic
Bus_Station
Lab_3
Tables
Views
Stored Pr...
Functions
Sample
sys
TechCo
Limit to 1000 rows
(1007, 'Shobhit', 60000, 406);
SELECT * FROM Employee_Details;
INSERT INTO Student VALUES
(1, 'Peter', 'Texas'),
(2, 'Suzi', 'California'),
(3, 'Joseph', 'Alaska'),
(4, 'Andrew', 'Los Angeles'),
(5, 'Brayan', 'New York');
SELECT * FROM Student;
100%
23:188
Result Grid
Filter Rows:
Search
Edit:
Export/Import:
Student_ID
Name
City
1 Peter Texas
2 Suzi California
3 Joseph Alaska
4 Andrew Los Angeles
5 Brayan New York
Result Grid
Form Editor
Student 21
Apply
Action Output
Time
Action
Response
Duration / Fetch Time
40 21:46:36 SELECT * FROM... 3 row(s) returned 0.00030 sec / 0.000...
41 21:47:19 INSERT INTO Em... 7 row(s) affected Records: 7 Duplicates: 0... 0.0025 sec
42 21:47:19 SELECT * FROM... 7 row(s) returned 0.00051 sec / 0.0000...
43 21:47:50 INSERT INTO Stu... 5 row(s) affected Records: 5 Duplicates: 0... 0.0018 sec
44 21:47:50 SELECT * FROM... 5 row(s) returned 0.00028 sec / 0.0000...
Query Completed


MySQL Workbench
Divyansh Kumar Singh 590012175
Administration
Schemas
Sample 2
Divyansh Kumar Singh 590012175 Lab 3*
Lab3 (1)
SCHEMAS
Filter objects
Basic
Bus_Station
Lab_3
Tables
Views
Stored Pr...
Functions
Sample
sys
TechCo
Limit to 1000 rows
(5, 'Brayan', 'New York');
SELECT * FROM Student;
INSERT INTO Student2 VALUES
(1, 'Texas'),
(2, 'California'),
(3, 'Alaska'),
(4, 'Los Angeles'),
(5, 'New York');
SELECT * FROM Student2;
100%
24:196
Result Grid
Filter Rows:
Search
Edit:
Export/Import:
Student_ID
City
1 Texas
2 California
3 Alaska
4 Los Angeles
5 New York
Result Grid
Form Editor
Student2 22
Apply
Action Output
Time
Action
Response
Duration / Fetch Time
42 21:47:19 SELECT * FROM... 7 row(s) returned 0.00051 sec / 0.0000...
43 21:47:50 INSERT INTO Stu... 5 row(s) affected Records: 5 Duplicates: 0... 0.0018 sec
44 21:47:50 SELECT * FROM... 5 row(s) returned 0.00028 sec / 0.0000...
45 21:48:24 INSERT INTO Stu... 5 row(s) affected Records: 5 Duplicates: 0... 0.0020 sec
46 21:48:24 SELECT * FROM... 5 row(s) returned 0.00035 sec / 0.0000...
Query Completed


MySQL Workbench
Divyansh Kumar Singh 590012175
Administration
Schemas
Sample 2
Divyansh Kumar Singh 590012175 Lab 3*
Lab3 (1)
SCHEMAS
Filter objects
Basic
Bus_Station
Lab_3
Tables
Views
Stored Pr...
Functions
Sample
sys
TechCo
Limit to 1000 rows
(5, 'New York');
196 SELECT * FROM Student2;
197
198 INSERT INTO Orders VALUES
199 (1, 101, '2023-01-15'),
200 (2, 102, '2023-02-20'),
201 (3, 103, '2023-03-10'),
202 (4, 104, '2023-04-05'),
203 (5, 105, '2023-05-12');
204 SELECT * FROM Orders;
205
100%
22:204
Result Grid
Filter Rows:
Search
Edit:
Export/Import:
Order_ID Cust_ID Order_Date
1 101 2023-01-15
2 102 2023-02-20
3 103 2023-03-10
4 104 2023-04-05
5 105 2023-05-12
NULL NULL NULL
Result Grid
Form Editor
Orders 23
Apply
Action Output
Time Action Response Duration / Fetch Time
44 21:47:50 SELECT * FROM... 5 row(s) returned 0.00028 sec / 0.0000...
45 21:48:24 INSERT INTO Stu... 5 row(s) affected Records: 5 Duplicates: 0... 0.0020 sec
46 21:48:24 SELECT * FROM... 5 row(s) returned 0.00035 sec / 0.0000...
47 21:48:53 INSERT INTO Ord... 5 row(s) affected Records: 5 Duplicates: 0... 0.0021 sec
48 21:48:53 SELECT * FROM... 5 row(s) returned 0.00042 sec / 0.0000...
Query Completed


MySQL Workbench
Divyansh Kumar Singh 590012175
Administration
Schemas
Sample 2
Divyansh Kumar Singh 590012175 Lab 3*
Lab3 (1)
SCHEMAS
Filter objects
Basic
Bus_Station
Lab_3
Tables
Views
Stored Pr...
Functions
Sample
sys
TechCo
Limit to 1000 rows
(4, 104, '2023-04-05'),
(5, 105, '2023-05-12');
SELECT * FROM Orders;
INSERT INTO Sales VALUES
('Electronics', 15000.00),
('Clothing', 8500.00),
('Books', 3200.00),
('Furniture', 21000.00);
SELECT * FROM Sales;
100%
1:212
Result Grid
Filter Rows:
Search
Export:
Product_Categ...
Sales_Amount
Electronics
15000.00
Clothing
8500.00
Books
3200.00
Furniture
21000.00
Result Grid
Form Editor
Sales 24
Read Only
Action Output
Time
Action
Response
Duration / Fetch Time
46 21:48:24 SELECT * FROM... 5 row(s) returned 0.00035 sec / 0.0000...
47 21:48:53 INSERT INTO Ord... 5 row(s) affected Records: 5 Duplicates: 0... 0.0021 sec
48 21:48:53 SELECT * FROM... 5 row(s) returned 0.00042 sec / 0.0000...
49 21:49:50 INSERT INTO Sal... 4 row(s) affected Records: 4 Duplicates: 0... 0.0023 sec
50 21:49:50 SELECT * FROM... 4 row(s) returned 0.00046 sec / 0.000...
Query Completed


MySQL Workbench
Divyansh Kumar Singh 590012175
Administration
Schemas
Sample 2
Divyansh Kumar Singh 590012175 Lab 3*
Lab3 (1)
SCHEMAS
Filter objects
Basic
Bus_Station
Lab_3
Tables
Views
Stored Pr...
Functions
Sample
sys
TechCo
Limit to 1000 rows
('Books', 3200.00),
('Furniture', 21000.00);
SELECT * FROM Sales;
INSERT INTO Top_Students VALUES
(1001, 95),
(1003, 92),
(1005, 98),
(1006, 91);
SELECT * FROM Top_Students;
100%
28:218
Result Grid
Filter Rows:
Search
Edit:
Export/Import:
Student_ID
Top_Marks
1001
95
1003
92
1005
98
1006
91
NULL
NULL
Result Grid
Form Editor
Top_Students 25
Apply
Action Output
Time
Action
Response
Duration / Fetch Time
48 21:48:53 SELECT * FROM... 5 row(s) returned 0.00042 sec / 0.0000...
49 21:49:50 INSERT INTO Sal... 4 row(s) affected Records: 4 Duplicates: 0... 0.0023 sec
50 21:49:50 SELECT * FROM... 4 row(s) returned 0.00046 sec / 0.000...
51 21:50:18 INSERT INTO Top... 4 row(s) affected Records: 4 Duplicates: 0... 0.0010 sec
52 21:50:18 SELECT * FROM... 4 row(s) returned 0.00031 sec / 0.0000...
Query Completed

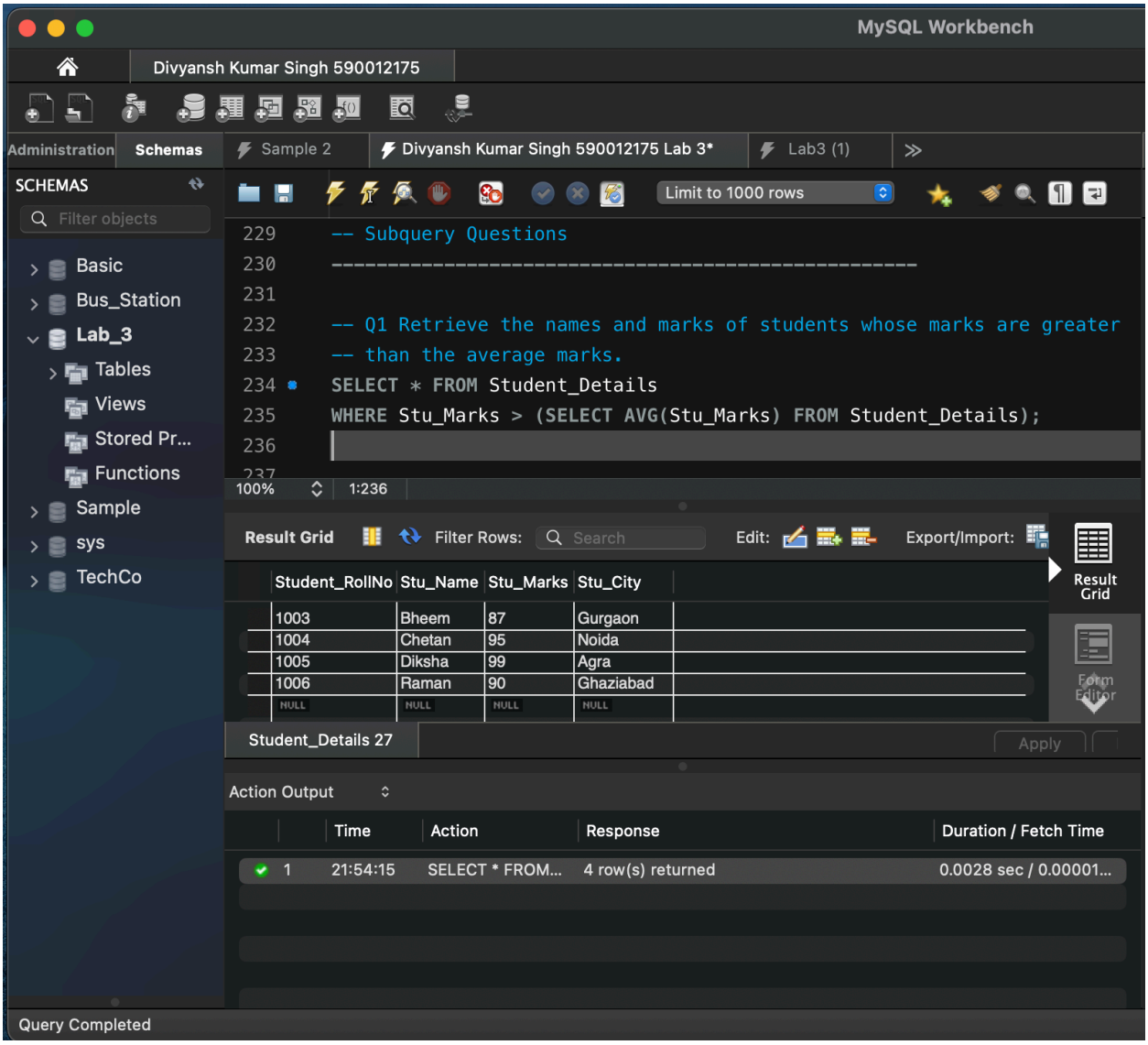

MySQL Workbench
Divyansh Kumar Singh 590012175
Administration
Schemas
Sample 2
Divyansh Kumar Singh 590012175 Lab 3*
Lab3 (1)
SCHEMAS
Filter objects
Basic
Bus_Station
Lab_3
Tables
Views
Stored Pr...
Functions
Sample
sys
TechCo
Limit to 1000 rows
(1006, 91);
SELECT * FROM Top_Students;
INSERT INTO Customer VALUES
(101, 'John', 28, 'Engineer'),
(102, 'Alice', 32, 'Doctor'),
(103, 'Robert', 40, 'Teacher'),
(104, 'Sophia', 25, 'Student'),
(105, 'David', 35, 'Businessman');
SELECT * FROM Customer;
100%
24:226
Result Grid
Filter Rows:
Search
Edit:
Export/Import:
Cust_ID Name Age Occupation
101 John 28 Engineer
102 Alice 32 Doctor
103 Robert 40 Teacher
104 Sophia 25 Student
105 David 35 Businessman
NULL NULL NULL NULL
Result Grid
Form Editor
Customer 26
Apply
Action Output
Time Action Response Duration / Fetch Time
50 21:49:50 SELECT * FROM... 4 row(s) returned 0.00046 sec / 0.000...
51 21:50:18 INSERT INTO Top... 4 row(s) affected Records: 4 Duplicates: 0... 0.0010 sec
52 21:50:18 SELECT * FROM... 4 row(s) returned 0.00031 sec / 0.0000...
53 21:50:41 INSERT INTO Cu... 5 row(s) affected Records: 5 Duplicates: 0... 0.0020 sec
54 21:50:41 SELECT * FROM... 5 row(s) returned 0.00039 sec / 0.000...
Query Completed

SQL Subquery Questions:

1. Basic **SELECT** Subquery

- Retrieve the names and marks of students whose marks are greater than the average marks.

```
SELECT * FROM Student_Details
WHERE Stu_Marks > (SELECT AVG(Stu_Marks) FROM Student_Details);
```

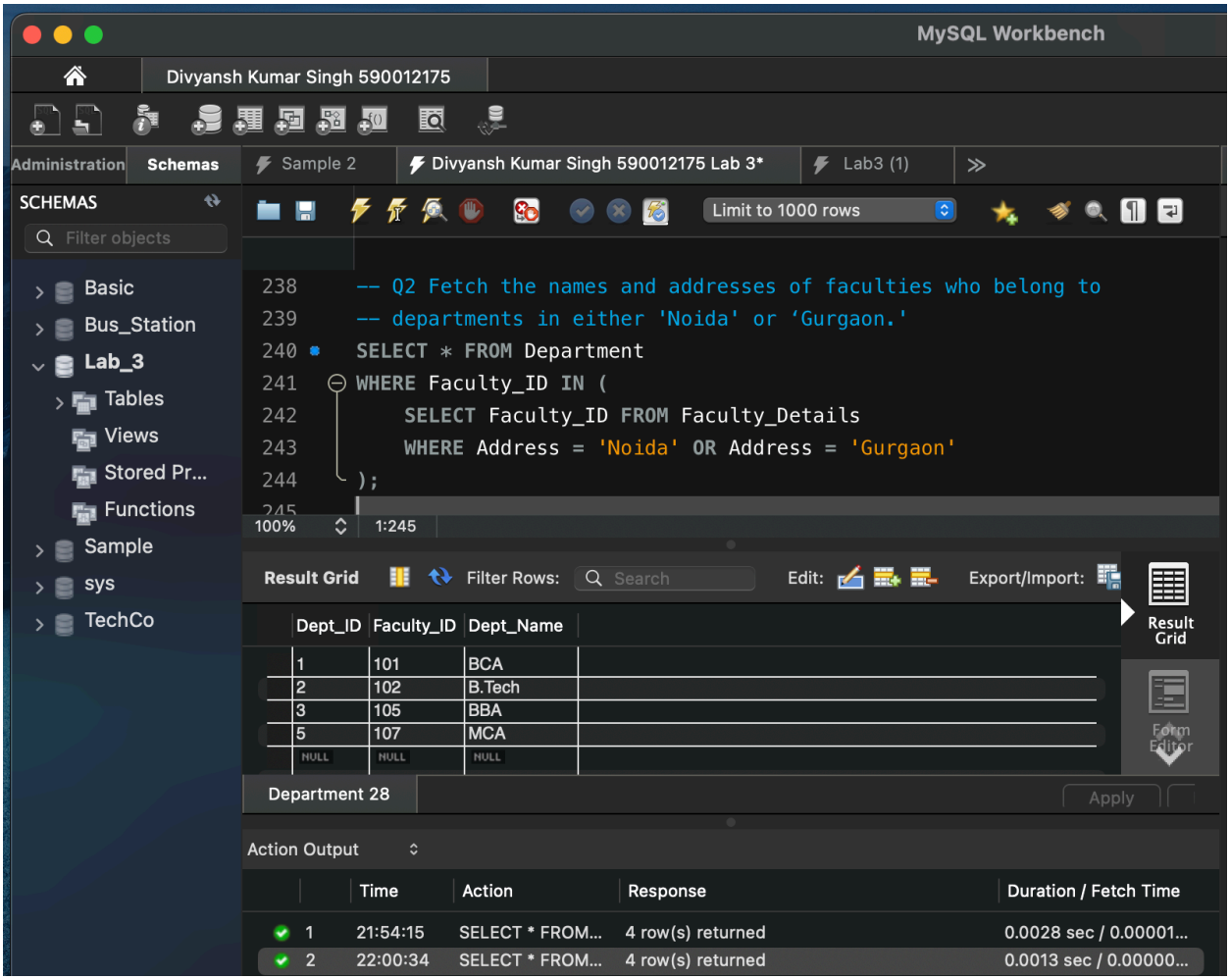


Student_RollNo	Stu_Name	Stu_Marks	Stu_City
1003	Bheem	87	Gurgaon
1004	Chetan	95	Noida
1005	Diksha	99	Agra
1006	Raman	90	Ghaziabad
NULL	NULL	NULL	NULL

2. **IN Operator** Subquery

- Fetch the names and addresses of faculties who belong to departments in either 'Noida' or ‘Gurgaon.’

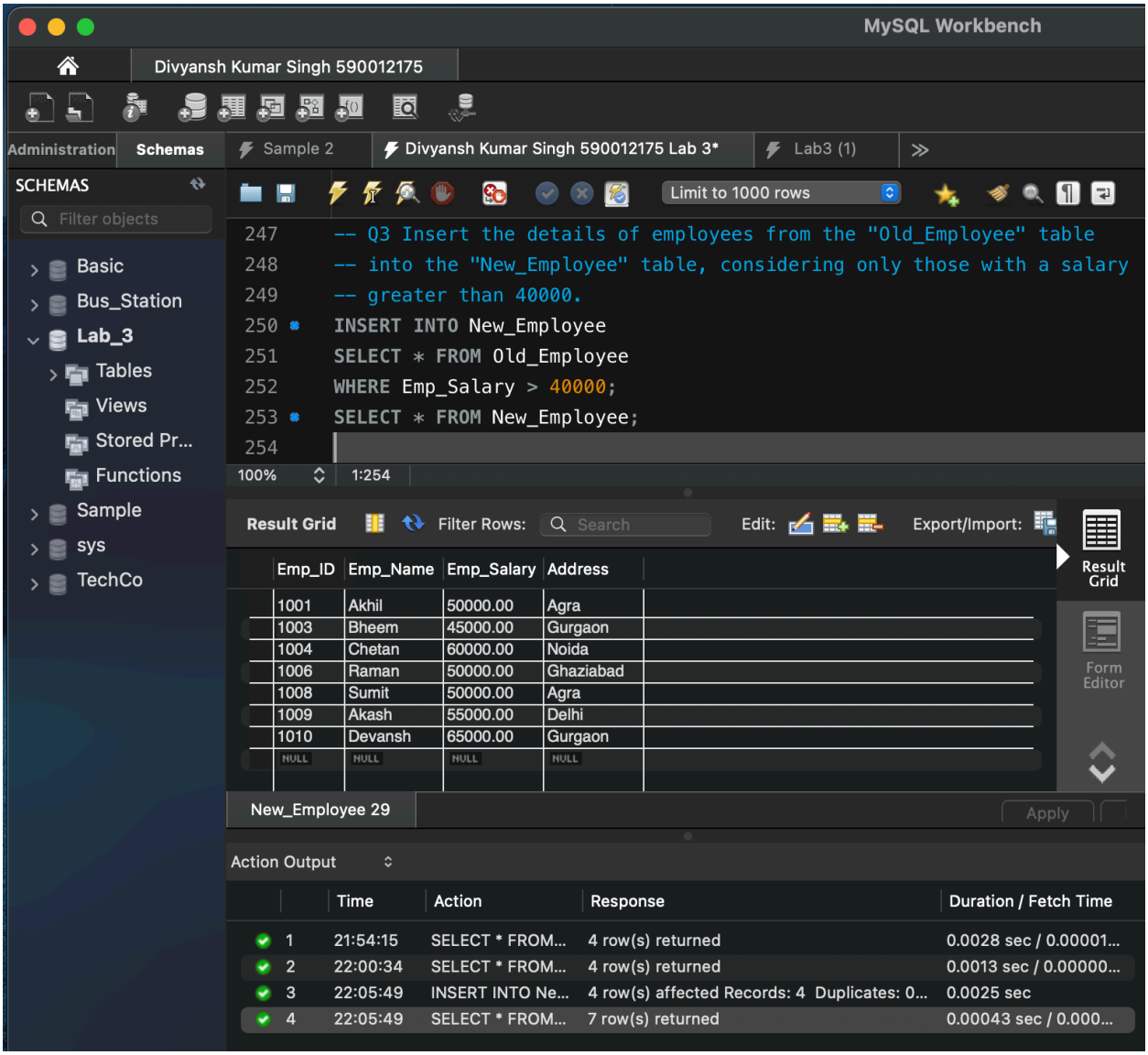
```
> SELECT * FROM Department
  WHERE Faculty_ID IN (
    SELECT Faculty_ID FROM Faculty_Details
    WHERE Address = 'Noida' OR Address = 'Gurgaon'
  );
```



3. **INSERT** with Subquery

- Insert the details of employees from the "Old_Employee" table into the "New_Employee" table, considering only those with a salary greater than 40000.

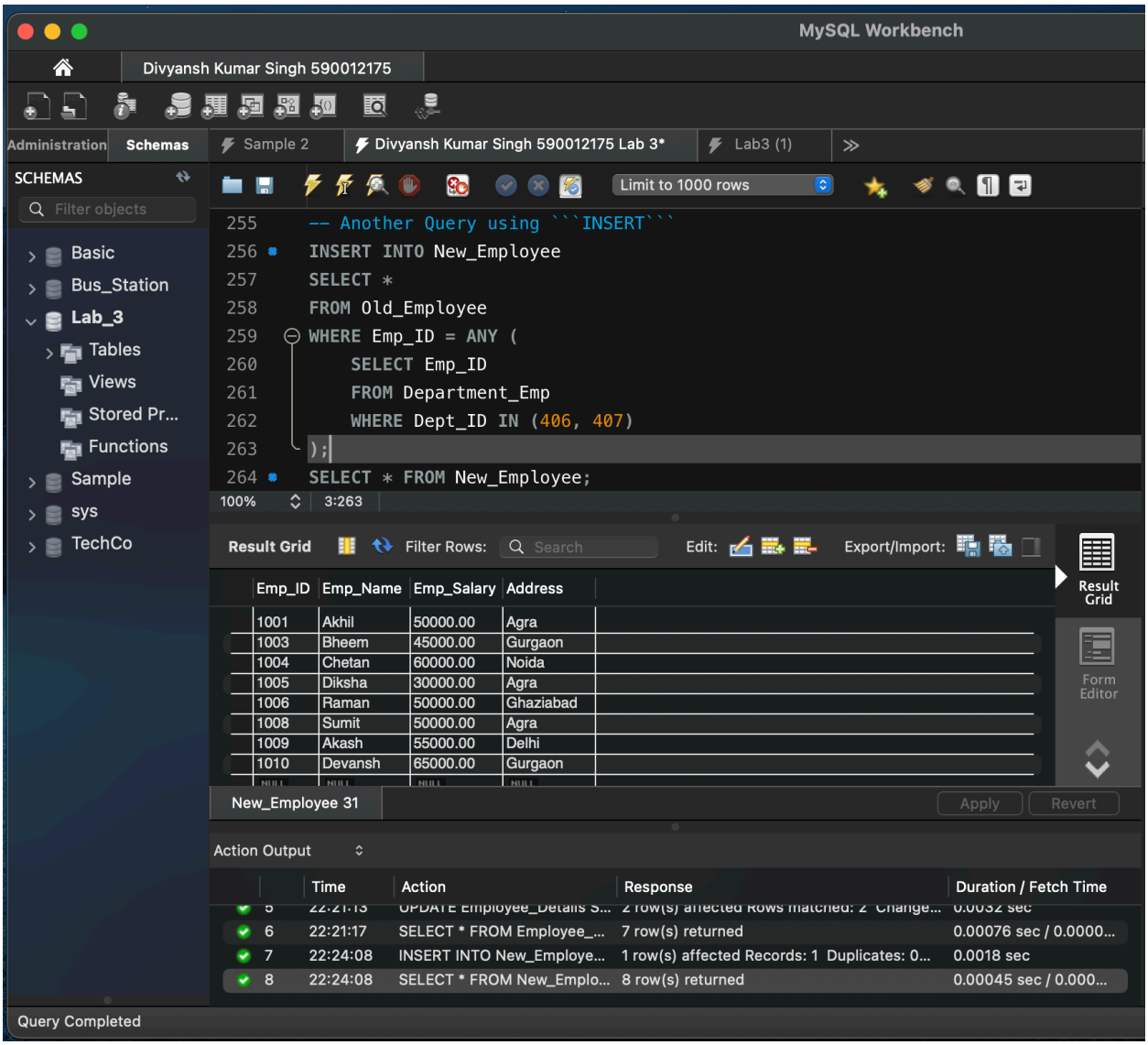
```
> INSERT INTO New_Employee
SELECT * FROM Old_Employee
WHERE Emp_Salary > 40000;

> SELECT * FROM New_Employee;
```



3(b). Another Query using ```INSERT```

```
INSERT INTO New_Employee
SELECT *
FROM Old_Employee
WHERE Emp_ID = ANY (
   SELECT Emp_ID
   FROM Department_Emp
   WHERE Dept_ID IN (406, 407)
);
```

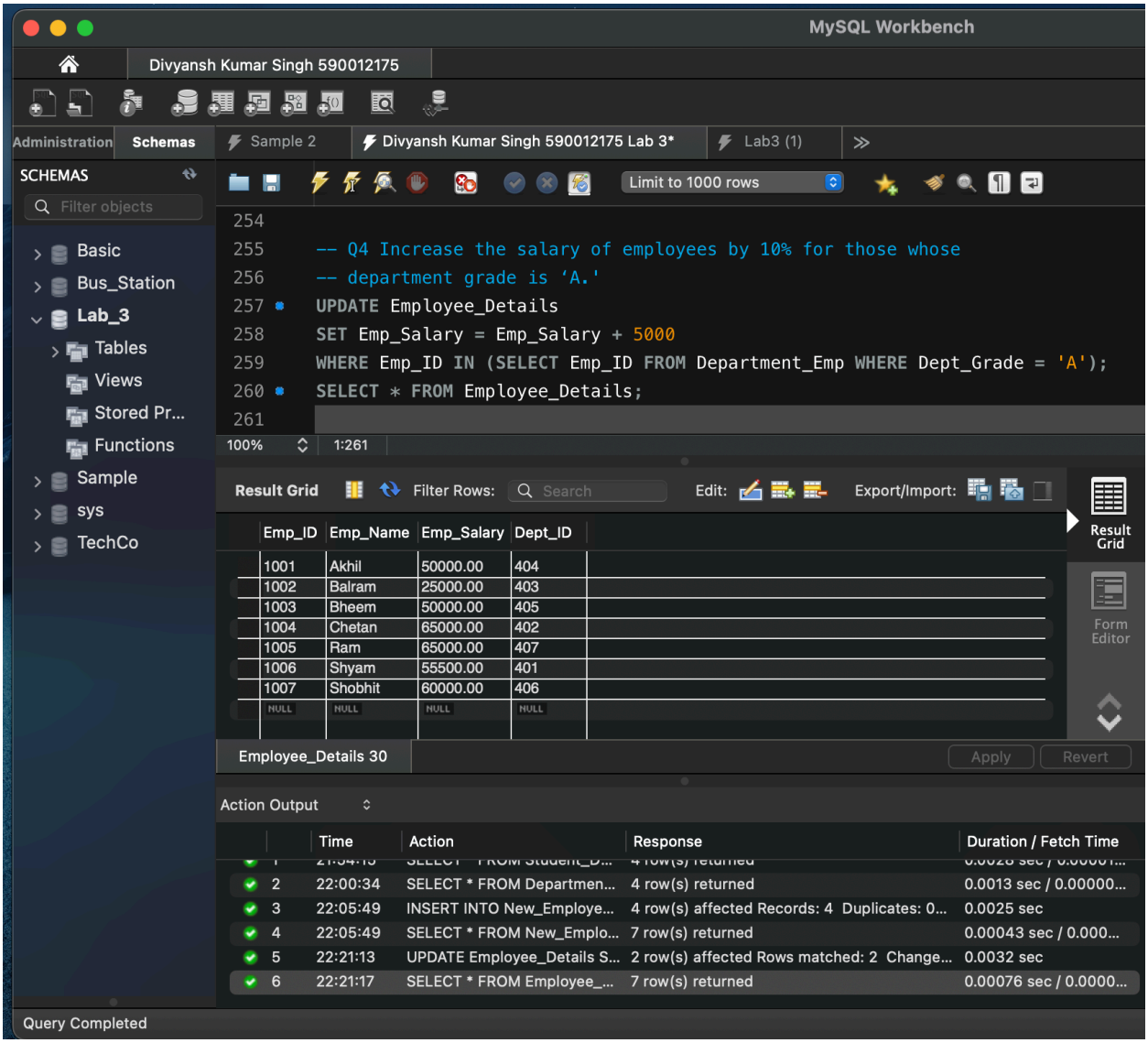
It copies employees from Old_Employee to New_Employee who belong to department 406 or 407.



4. **UPDATE** with Subquery:

- Increase the salary of employees by 10% for those whose department grade is ‘A.'

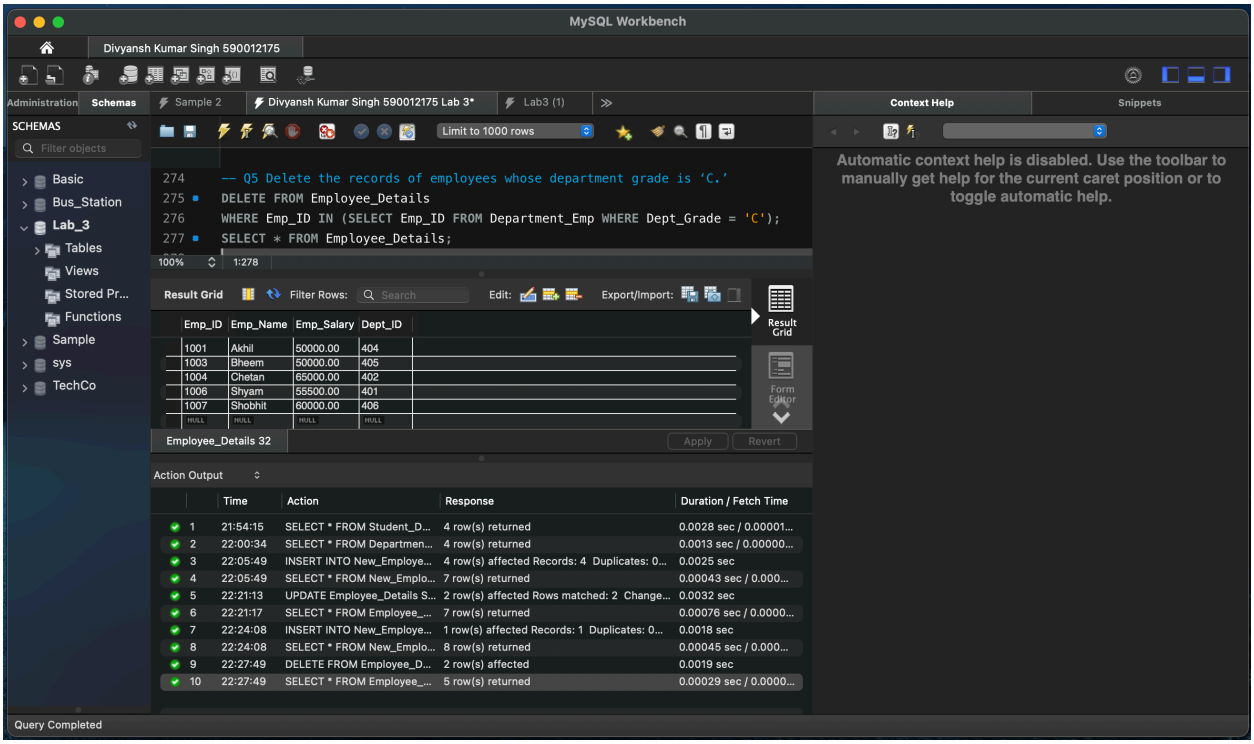
```
> UPDATE Employee_Details
SET Emp_Salary = Emp_Salary + 5000
WHERE Emp_ID IN (SELECT Emp_ID FROM Department_Emp WHERE
Dept_Grade = ‘A');

> SELECT * FROM Employee_Details;
```



5. **DELETE** with Subquery:

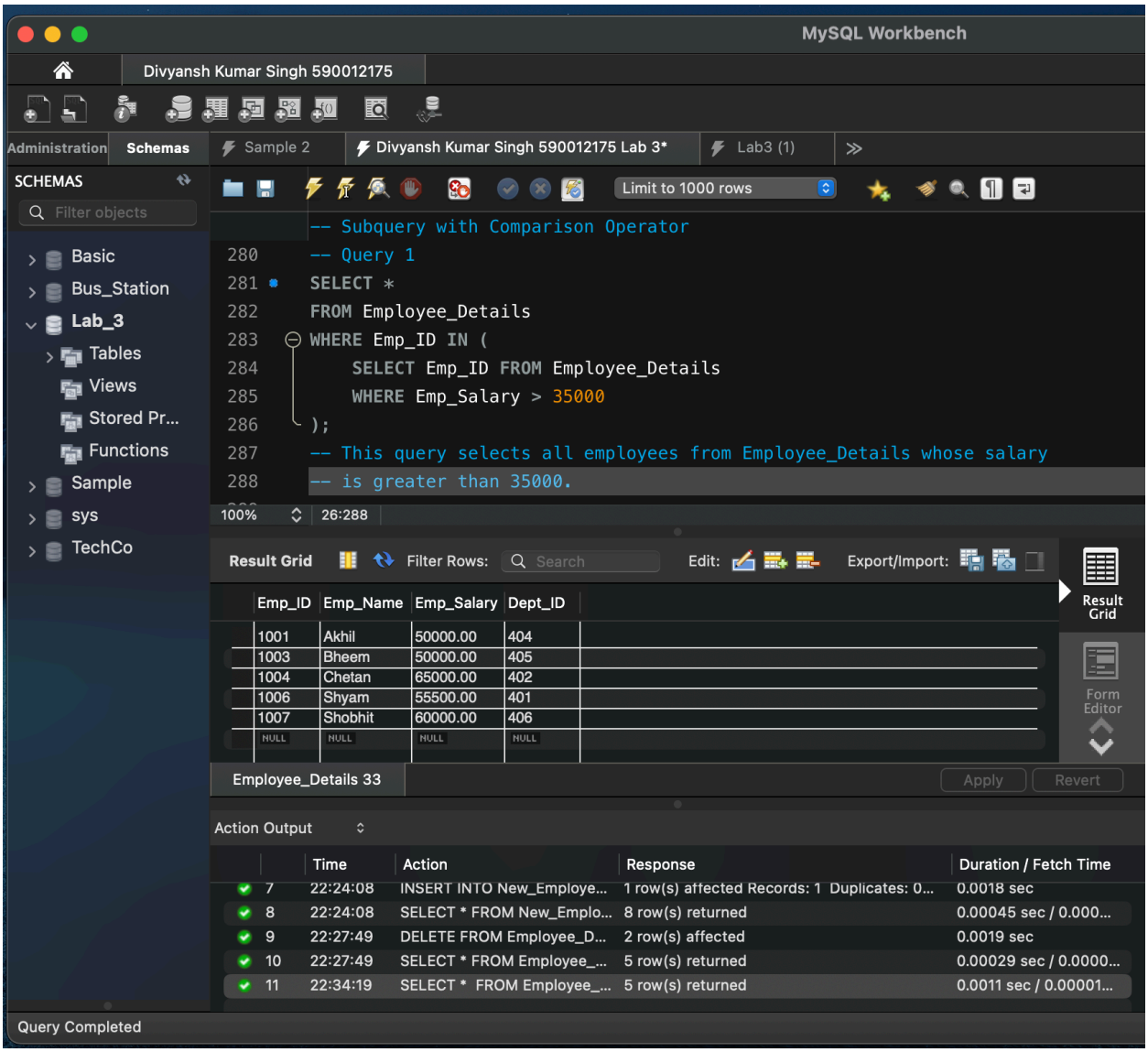
- Delete the records of employees whose department grade is ‘C.’

```
> DELETE FROM Employee_Details
WHERE Emp_ID IN (SELECT Emp_ID FROM Department_Emp WHERE
Dept_Grade = 'C');

> SELECT * FROM Employee_Details;
```

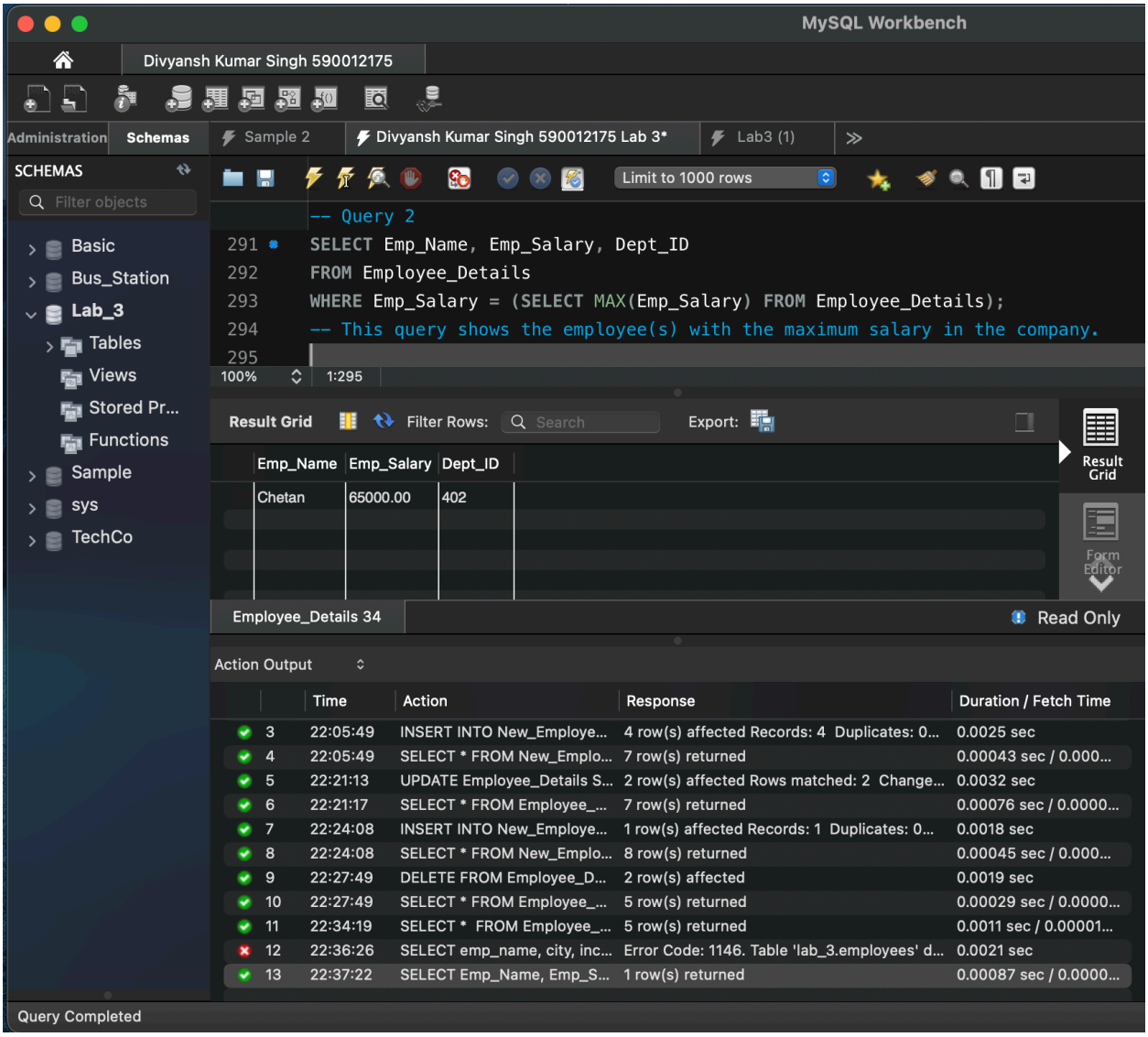


Subquery with Comparison Operator

```
-- Query 1
SELECT *
FROM Employee_Details
WHERE Emp_ID IN (
    SELECT Emp_ID FROM Employee_Details
    WHERE Emp_Salary > 35000
);
-- This query selects all employees from Employee_Details whose salary
-- is greater than 35000.
```



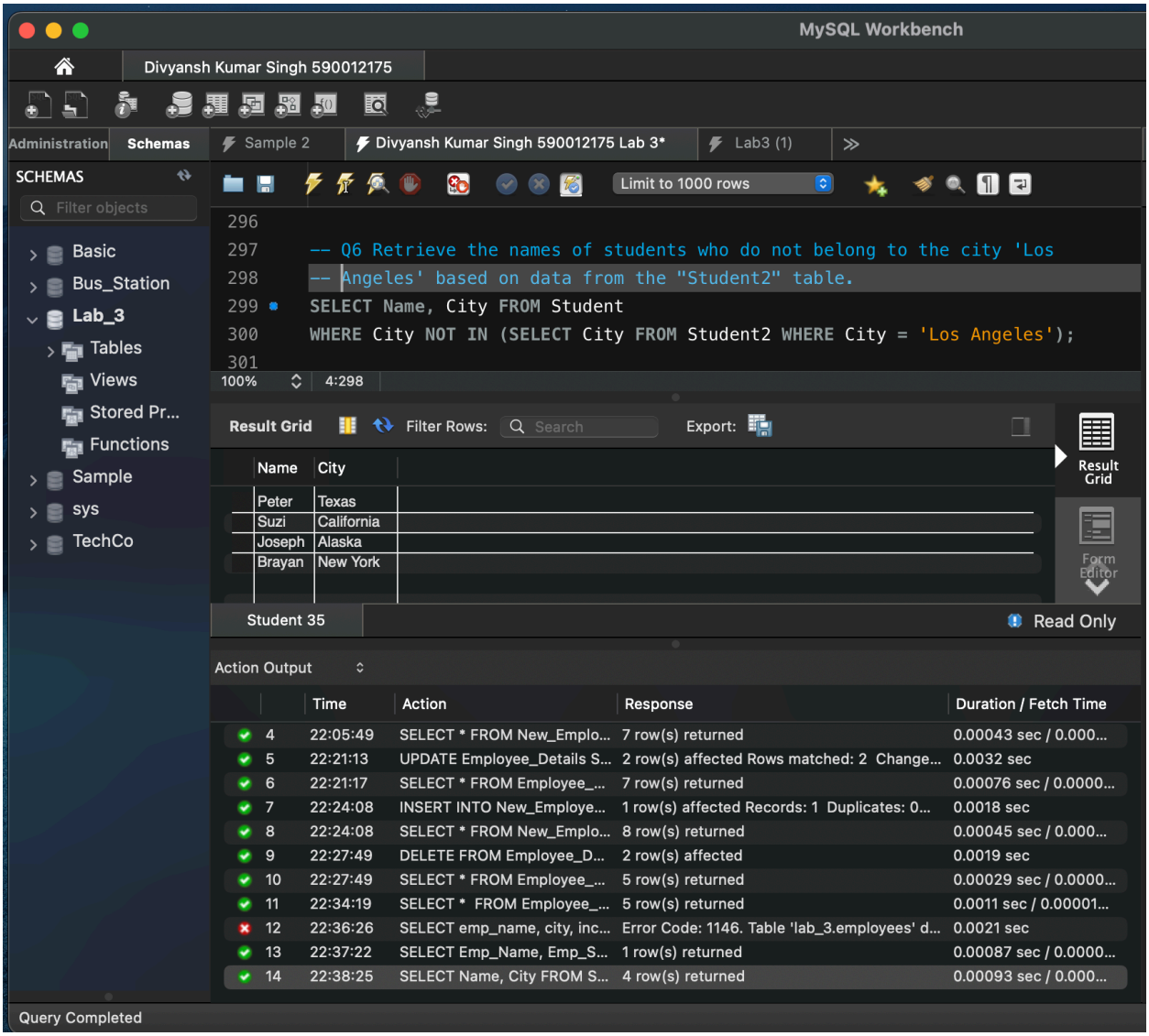
```
-- Query 2
SELECT Emp_Name, Emp_Salary, Dept_ID
FROM Employee_Details
WHERE Emp_Salary = (SELECT MAX(Emp_Salary) FROM Employee_Details);
-- This query shows the employee(s) with the maximum salary in the company.
```



6. Subquery in **WHERE Clause - NOT IN**

- Retrieve the names of students who do not belong to the city 'Los Angeles' based on data from the "Student2" table.

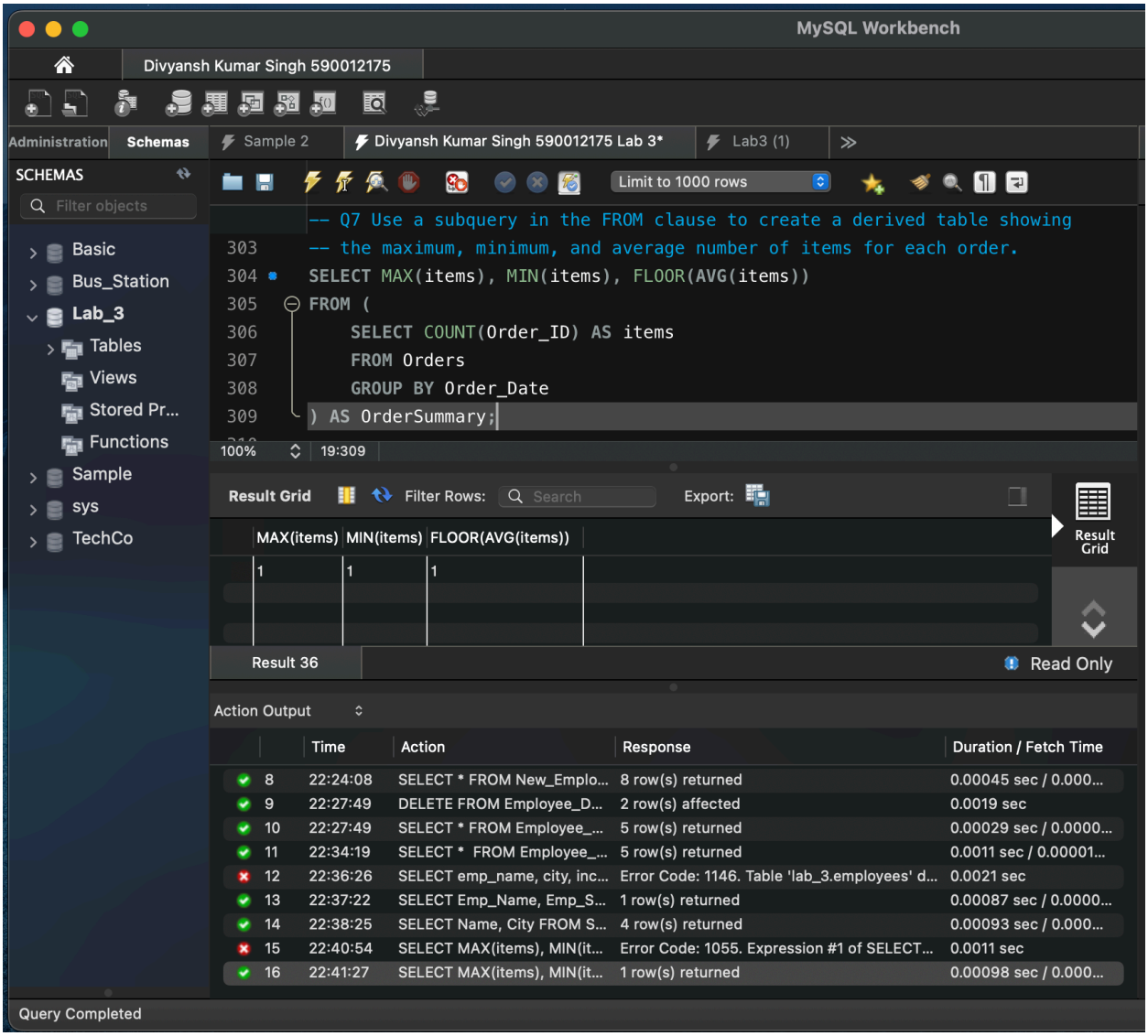
```
SELECT Name, City FROM Student
WHERE City NOT IN (SELECT City FROM Student2 WHERE City = 'Los Angeles');
```



7. Subquery in **FROM** Clause:

- Use a subquery in the FROM clause to create a derived table showing the maximum, minimum, and average number of items for each order.

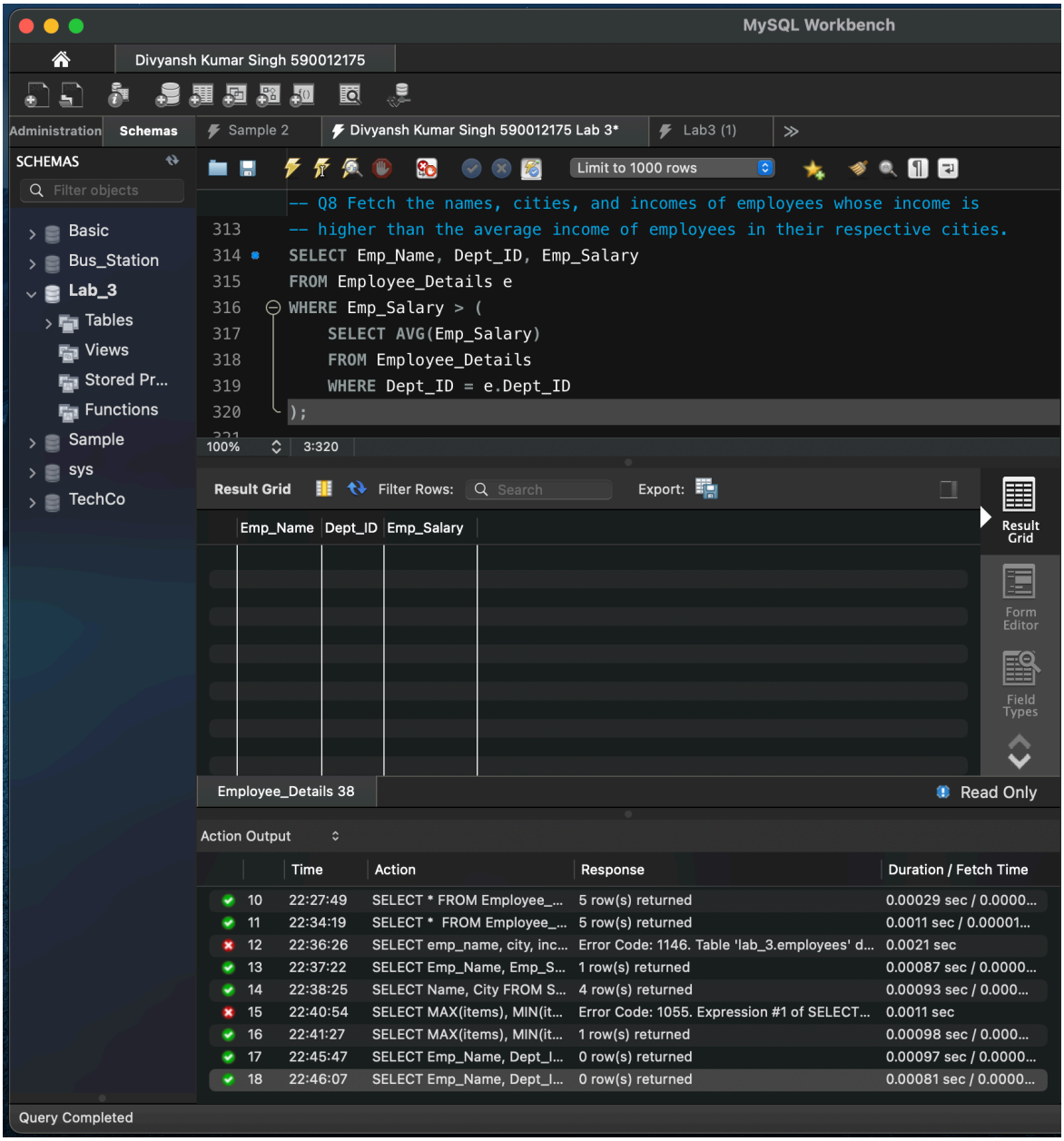
```
SELECT MAX(items), MIN(items), FLOOR(AVG(items))
FROM (
    SELECT COUNT(Order_ID) AS items
    FROM Orders
    GROUP BY Order_Date
) AS OrderSummary;
```



8. Correlated **Subquery**

- Fetch the names, cities, and incomes of employees whose income is higher than the average income of employees in their respective cities.

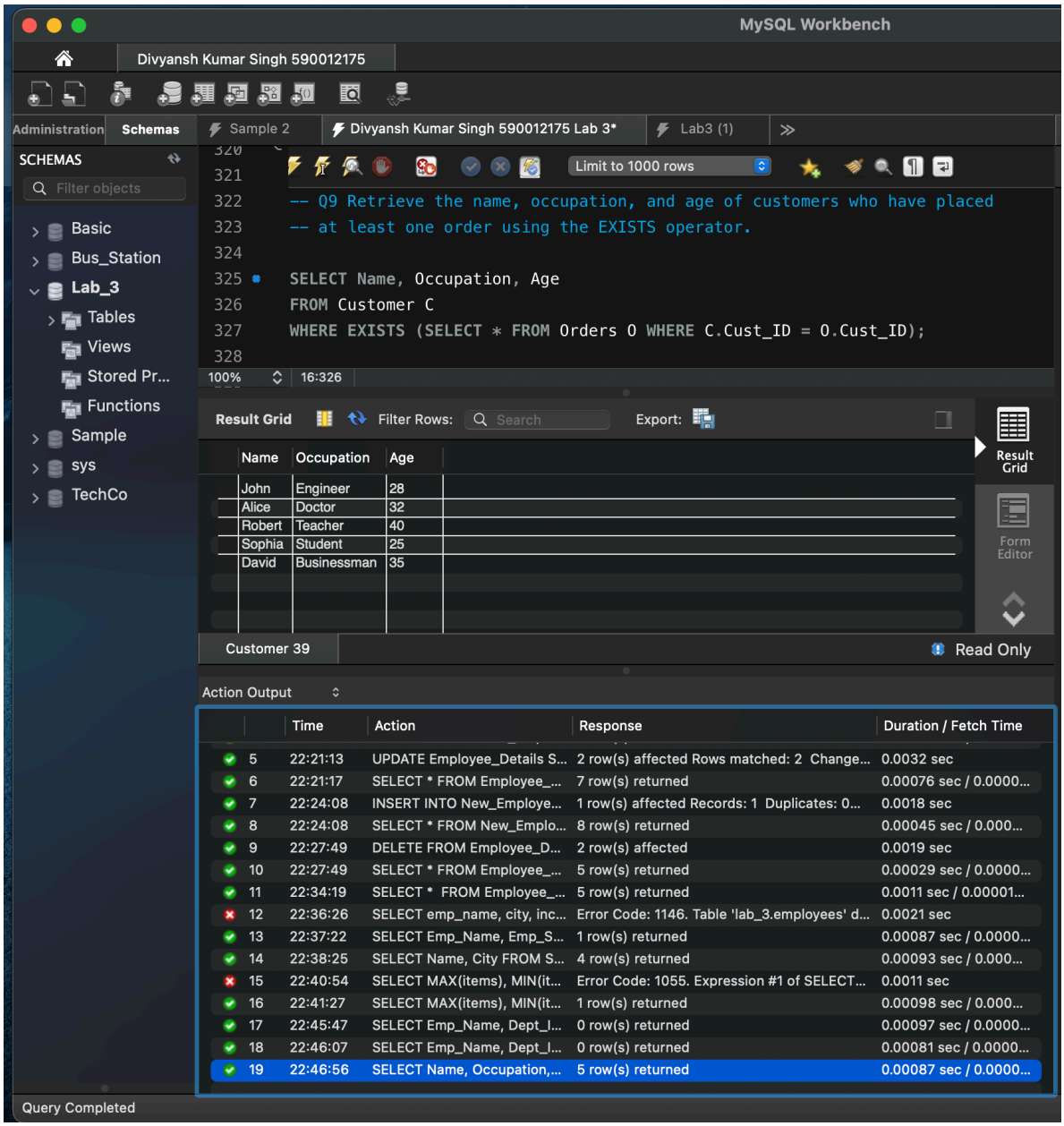
```
SELECT Emp_Name, Dept_ID, Emp_Salary
FROM Employee_Details e
WHERE Emp_Salary > (
    SELECT AVG(Emp_Salary)
    FROM Employee_Details
    WHERE Dept_ID = e.Dept_ID
);
```



9. **EXISTS** Operator

- Retrieve the name, occupation, and age of customers who have placed at least one order using the EXISTS operator.

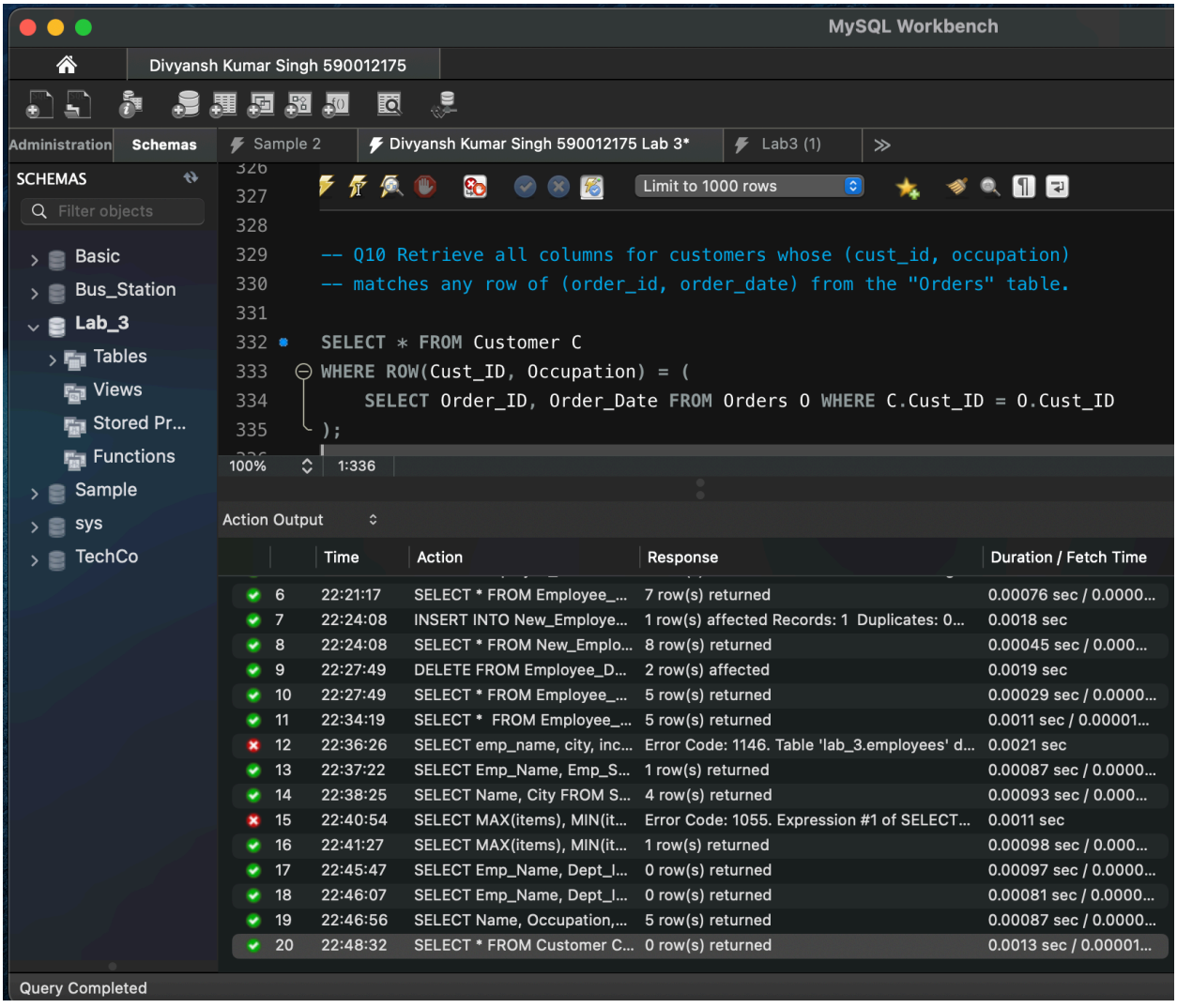
```
SELECT Name, Occupation, Age
FROM Customer C
WHERE EXISTS (SELECT * FROM Orders O WHERE C.Cust_ID = O.Cust_ID);
```



10. **ROW** Subquery

- Retrieve all columns for customers whose (cust_id, occupation) matches any row of (order_id, order_date) from the "Orders" table.

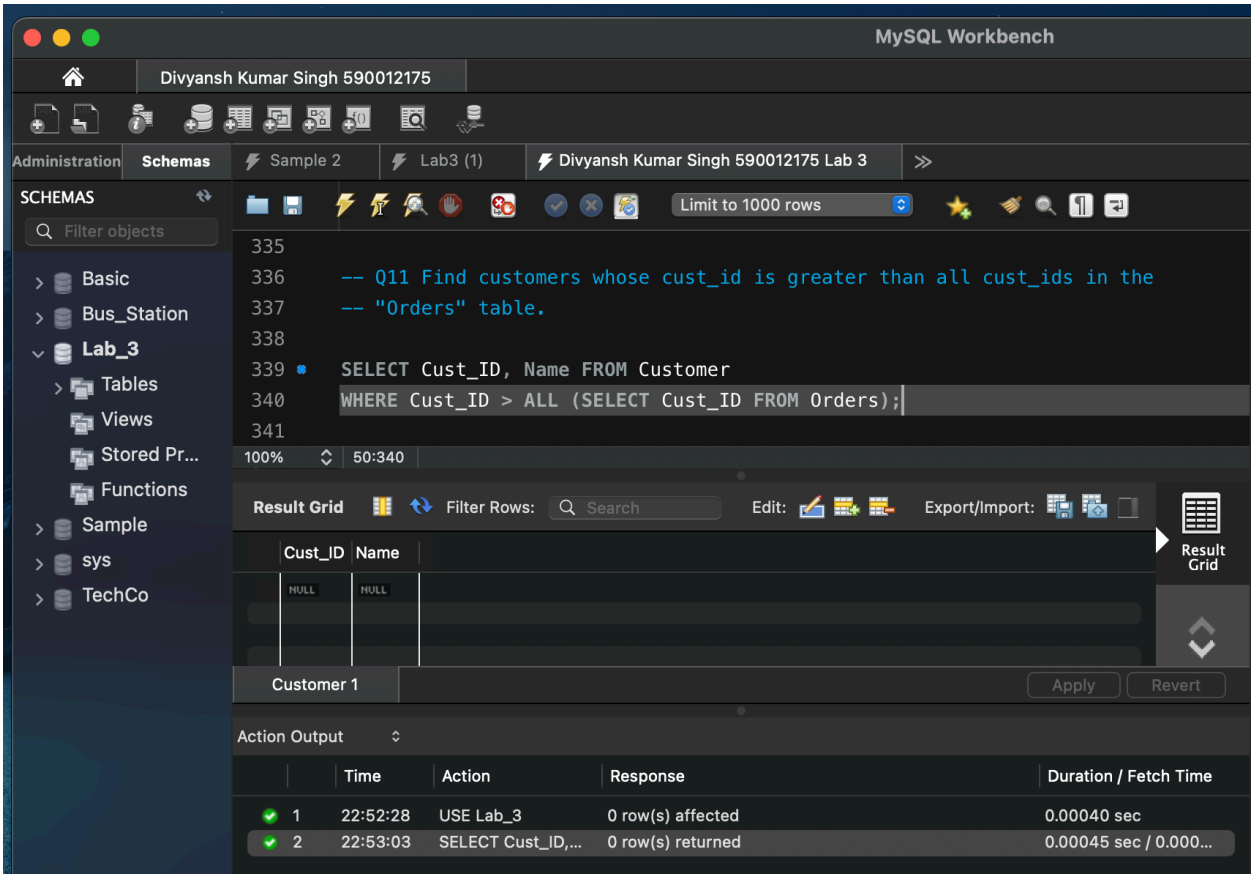
```
SELECT * FROM Customer C
WHERE ROW(Cust_ID, Occupation) = (
    SELECT Order_ID, Order_Date FROM Orders O WHERE C.Cust_ID = O.Cust_ID
);
```



11. Subquery with **ALL Operator**

- Find customers whose cust_id is greater than all cust_ids in the "Orders" table.

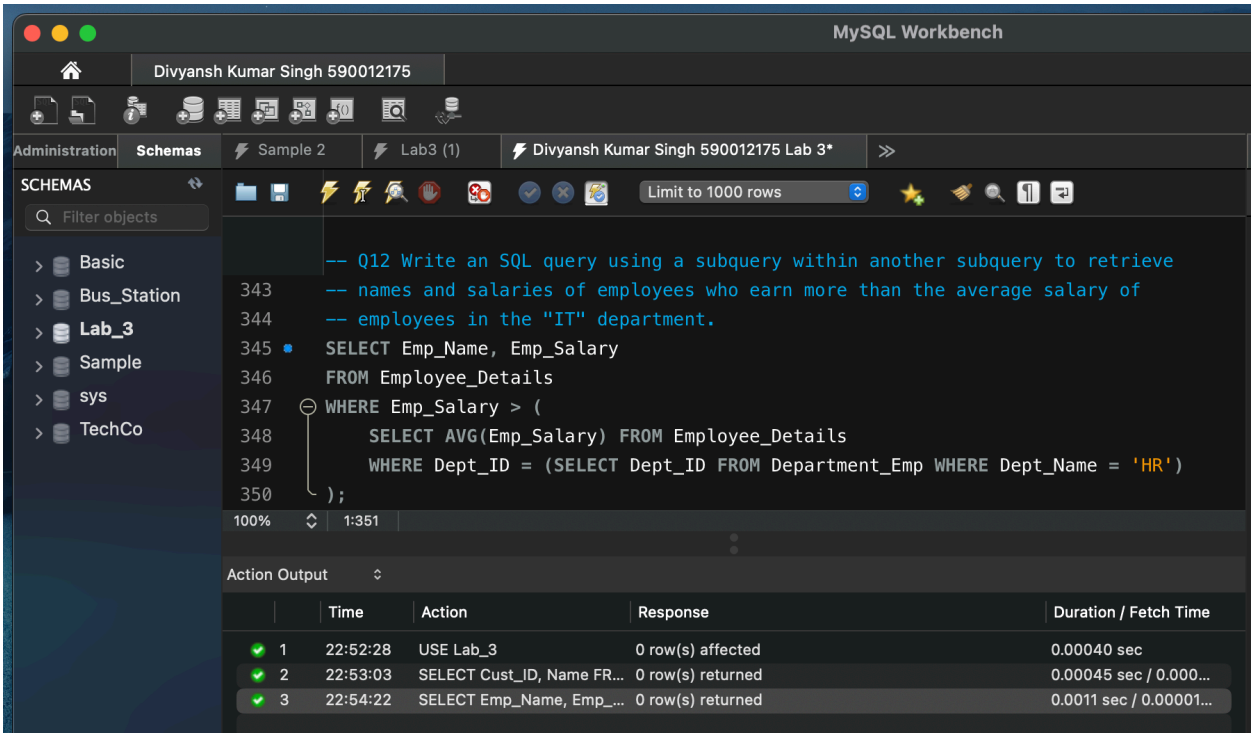
```
SELECT Cust_ID, Name FROM Customer
WHERE Cust_ID > ALL (SELECT Cust_ID FROM Orders);
```



12. **Nested Subqueries**

- Write an SQL query using a subquery within another subquery to retrieve names and salaries of employees who earn more than the average salary of employees in the "IT" department.

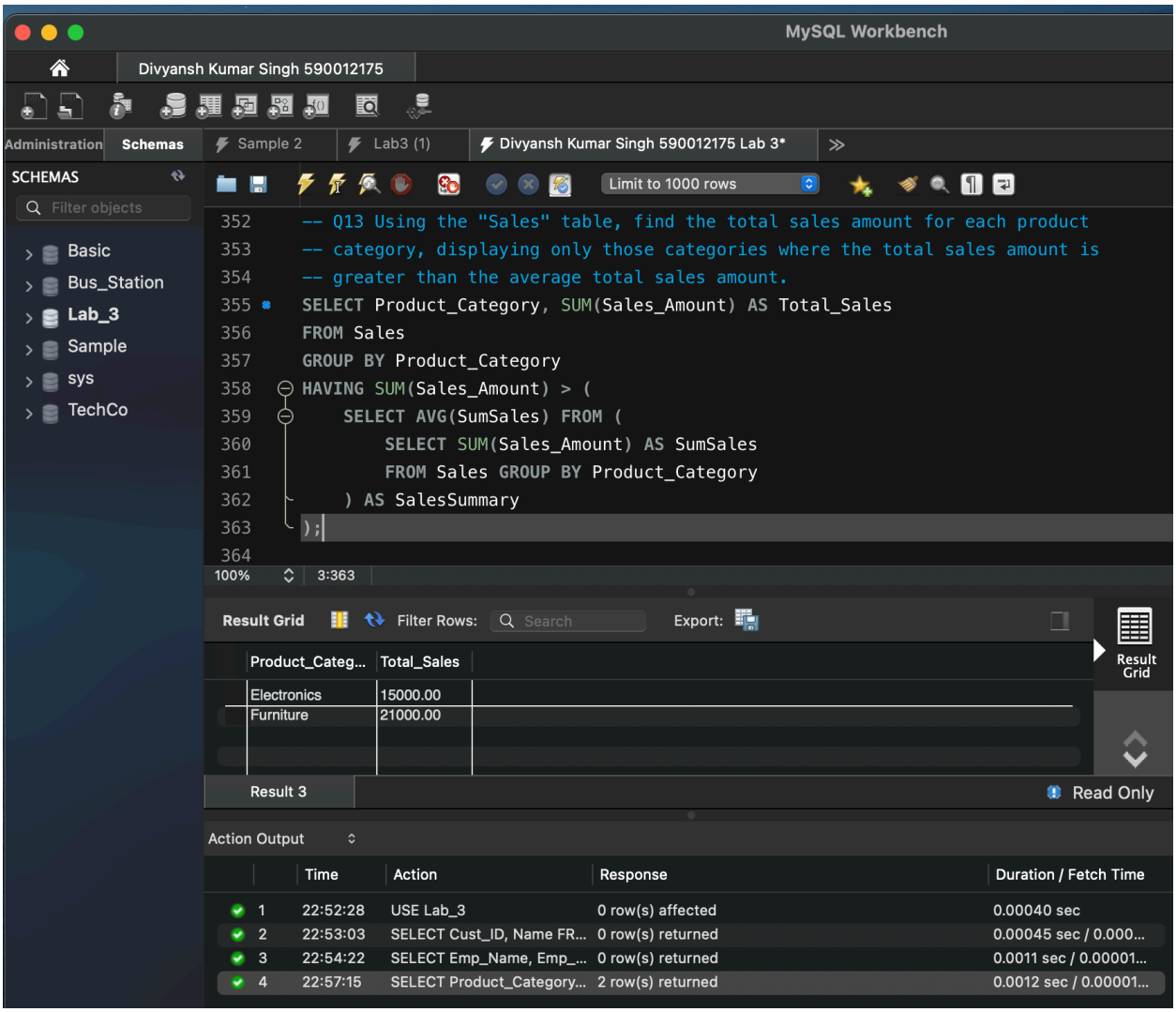
```
SELECT Emp_Name, Emp_Salary
FROM Employee_Details
WHERE Emp_Salary > (
   SELECT AVG(Emp_Salary) FROM Employee_Details
         WHERE Dept_ID = (SELECT Dept_ID FROM Department_Emp
WHERE Dept_Name = 'HR')
);
```



13. Subquery in **HAVING Clause**

- Using the "Sales" table, find the total sales amount for each product category, displaying only those categories where the total sales amount is greater than the average total sales amount.

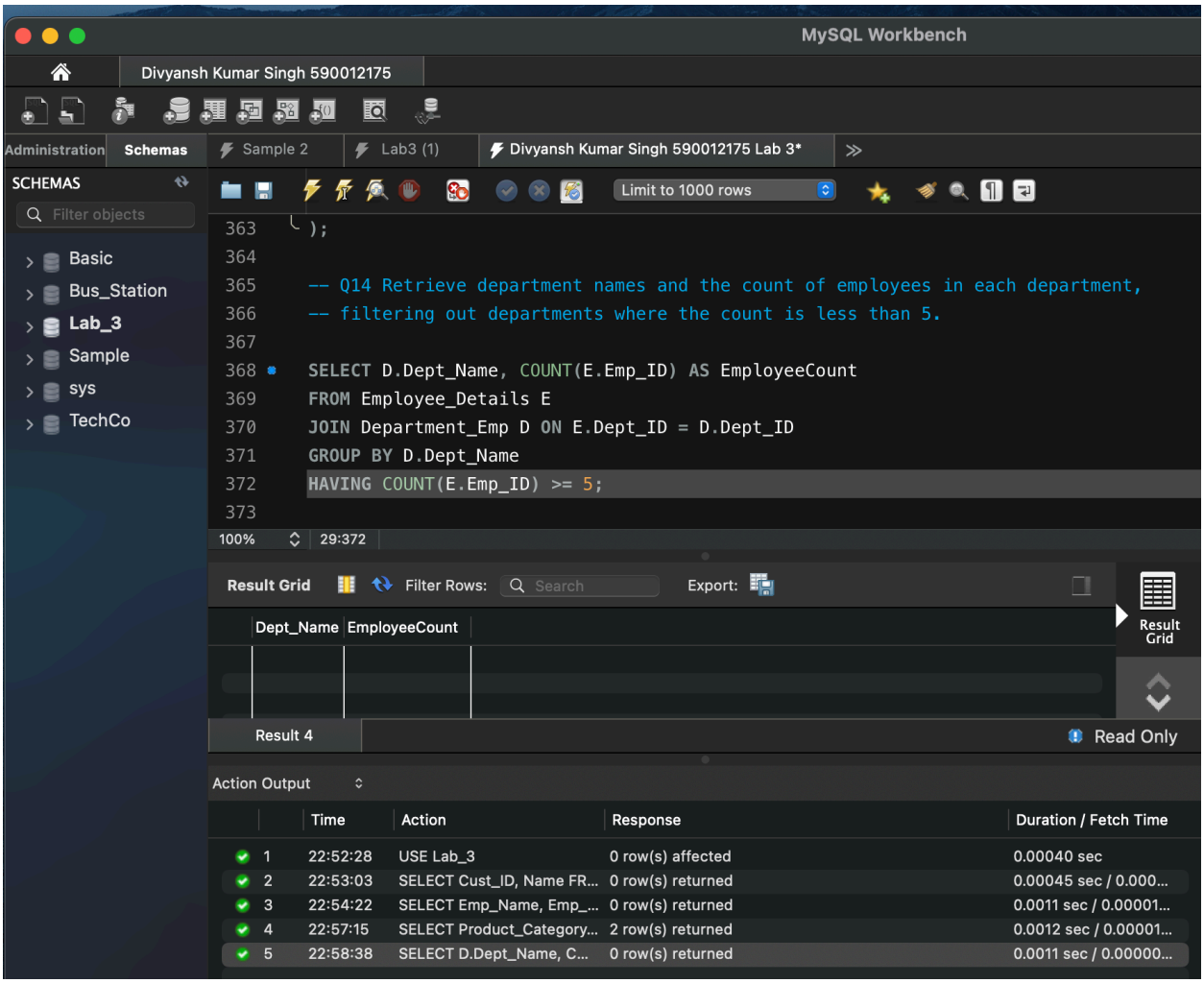
```
SELECT Product_Category, SUM(Sales_Amount) AS Total_Sales
FROM Sales
GROUP BY Product_Category
HAVING SUM(Sales_Amount) > (
   SELECT AVG(SumSales) FROM (
      SELECT SUM(Sales_Amount) AS SumSales
      FROM Sales GROUP BY Product_Category
   ) AS SalesSummary
);
```



14. Subquery with **GROUP BY**

- Retrieve department names and the count of employees in each department, filtering out departments where the count is less than 5.

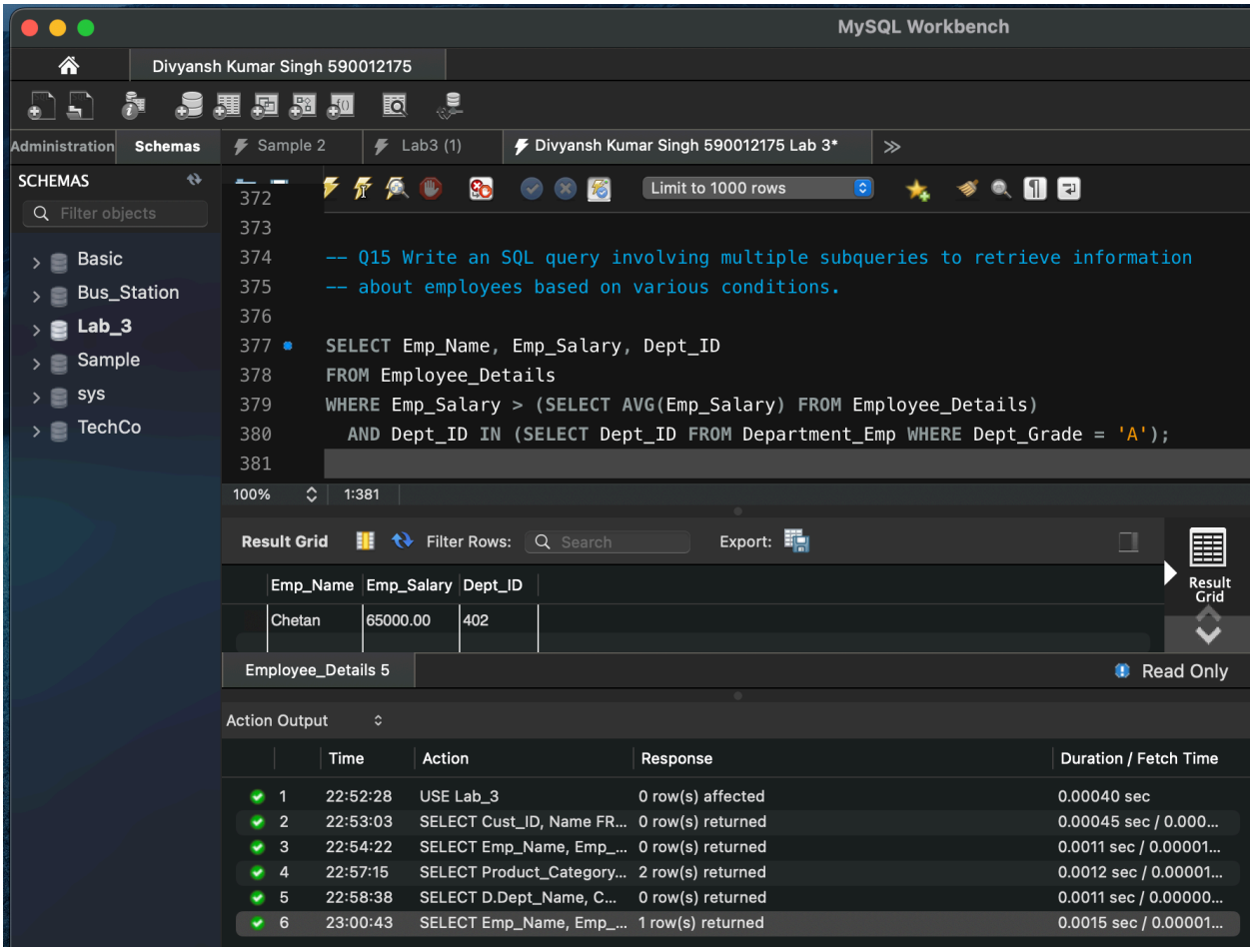
```
SELECT D.Dept_Name, COUNT(E.Emp_ID) AS EmployeeCount
FROM Employee_Details E
JOIN Department_Emp D ON E.Dept_ID = D.Dept_ID
GROUP BY D.Dept_Name
HAVING COUNT(E.Emp_ID) >= 5;
```



15. Multiple **Subqueries in a Single Query**

- Write an SQL query involving multiple subqueries to retrieve information about employees based on various conditions.

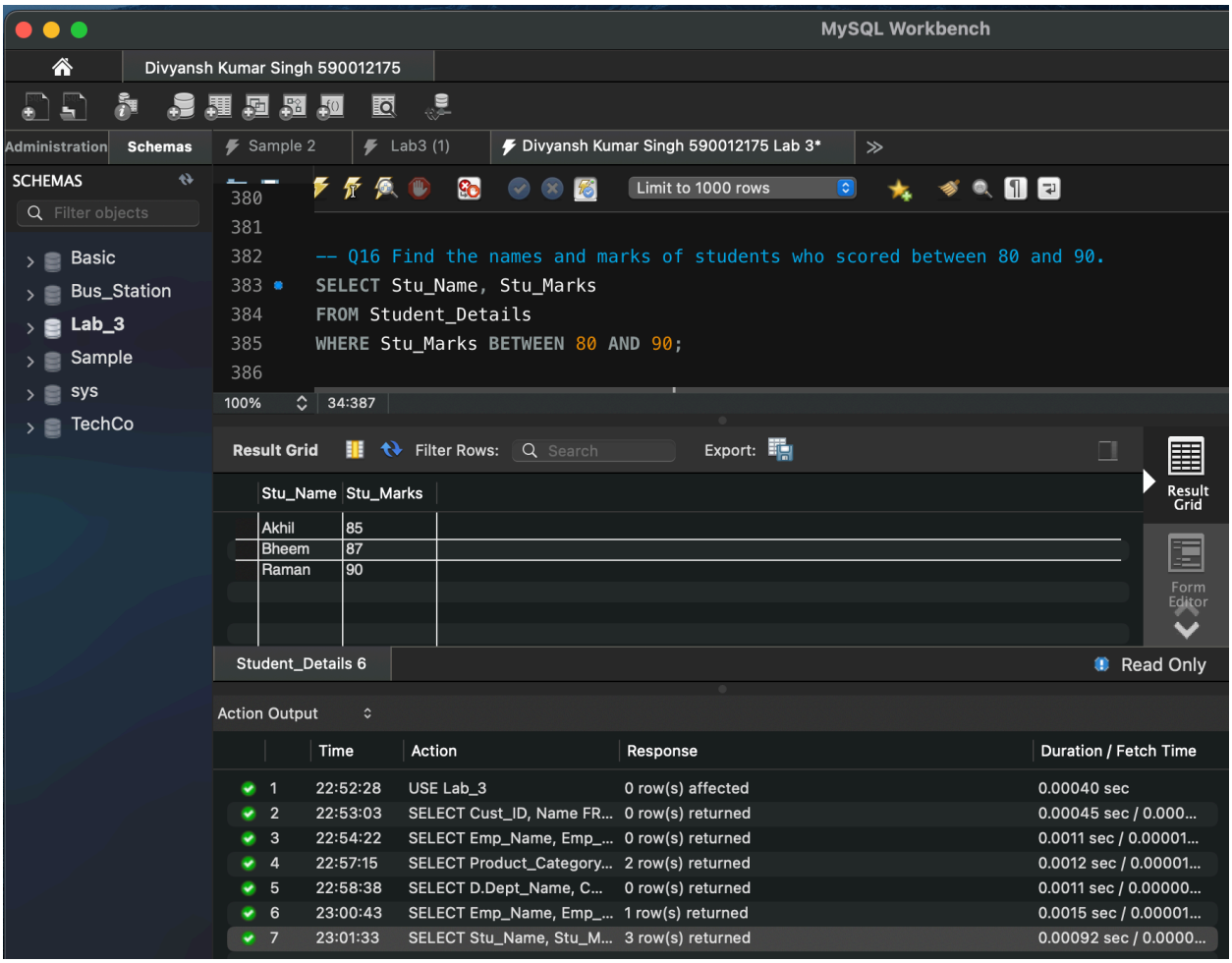
```
SELECT Emp_Name, Emp_Salary, Dept_ID
FROM Employee_Details
WHERE Emp_Salary > (SELECT AVG(Emp_Salary) FROM Employee_Details)
  AND Dept_ID IN (SELECT Dept_ID FROM Department_Emp WHERE Dept_Grade = 'A');
```



16. Subquery with **BETWEEN** Operators

- Find the names and marks of students who scored between 80 and 90.

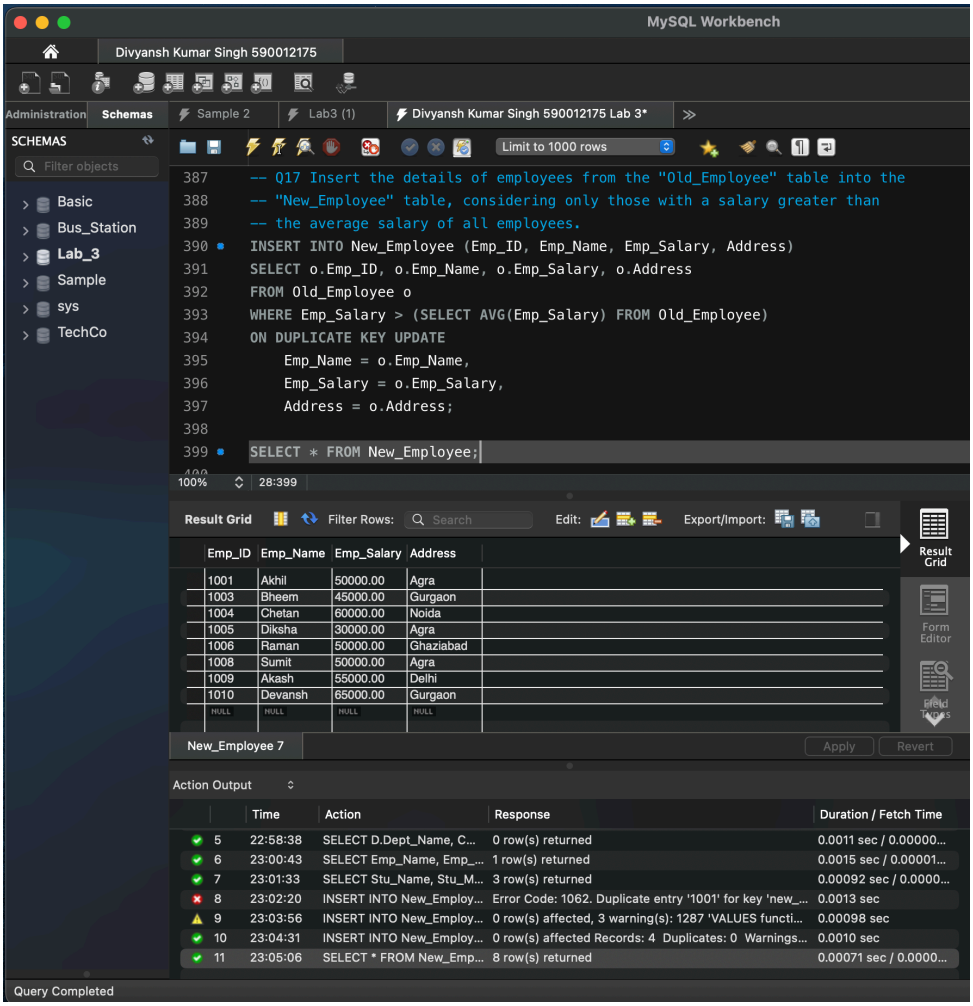
```
SELECT Stu_Name, Stu_Marks
FROM Student_Details
WHERE Stu_Marks BETWEEN 80 AND 90;
```



17. Subquery in **INSERT Statement**

- Insert the details of employees from the "Old_Employee" table into the "New_Employee" table, considering only those with a salary greater than the average salary of all employees.

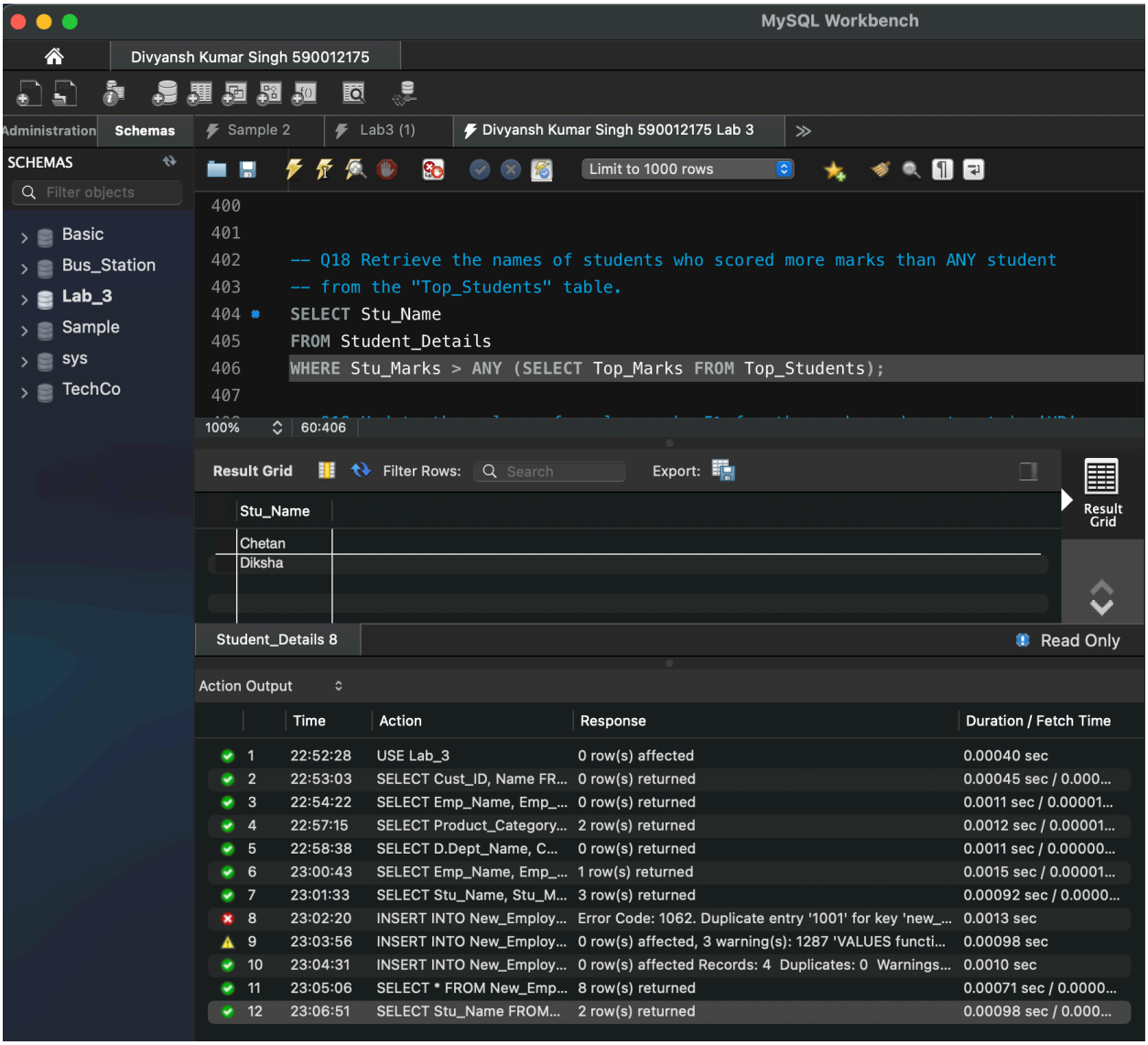
```
> INSERT INTO New_Employee (Emp_ID, Emp_Name, Emp_Salary, Address)
SELECT o.Emp_ID, o.Emp_Name, o.Emp_Salary, o.Address
FROM Old_Employee o
WHERE Emp_Salary > (SELECT AVG(Emp_Salary) FROM Old_Employee)
ON DUPLICATE KEY UPDATE
    Emp_Name = o.Emp_Name,
    Emp_Salary = o.Emp_Salary,
    Address = o.Address;

> SELECT * FROM New_Employee;
```



18. Subquery with **ANY Operator**

- Retrieve the names of students who scored more marks than ANY student from the "Top_Students" table.

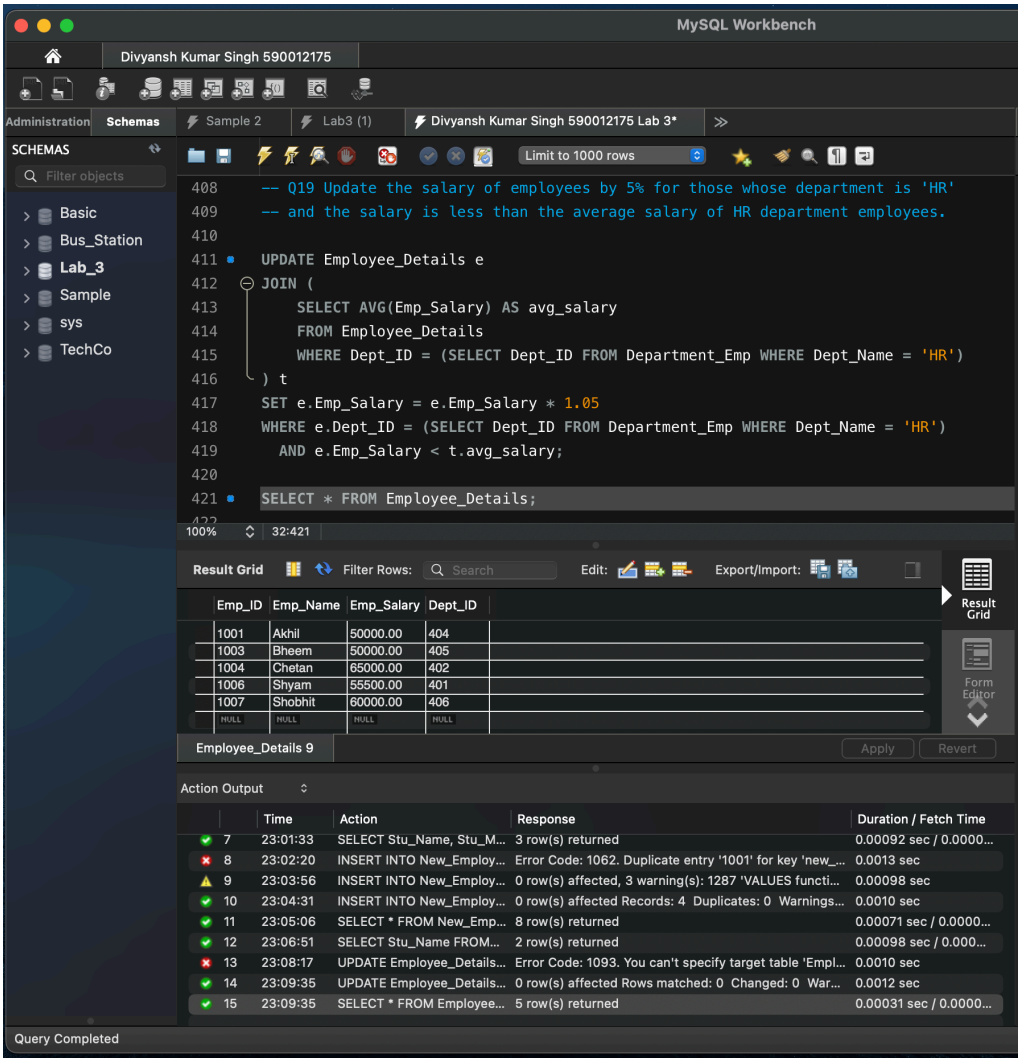
```
SELECT Stu_Name
FROM Student_Details
WHERE Stu_Marks > ANY (SELECT Top_Marks FROM Top_Students);
```



19. Subquery in **UPDATE** Statement

- Update the salary of employees by 5% for those whose department is 'HR' and the salary is less than the average salary of HR department employees.

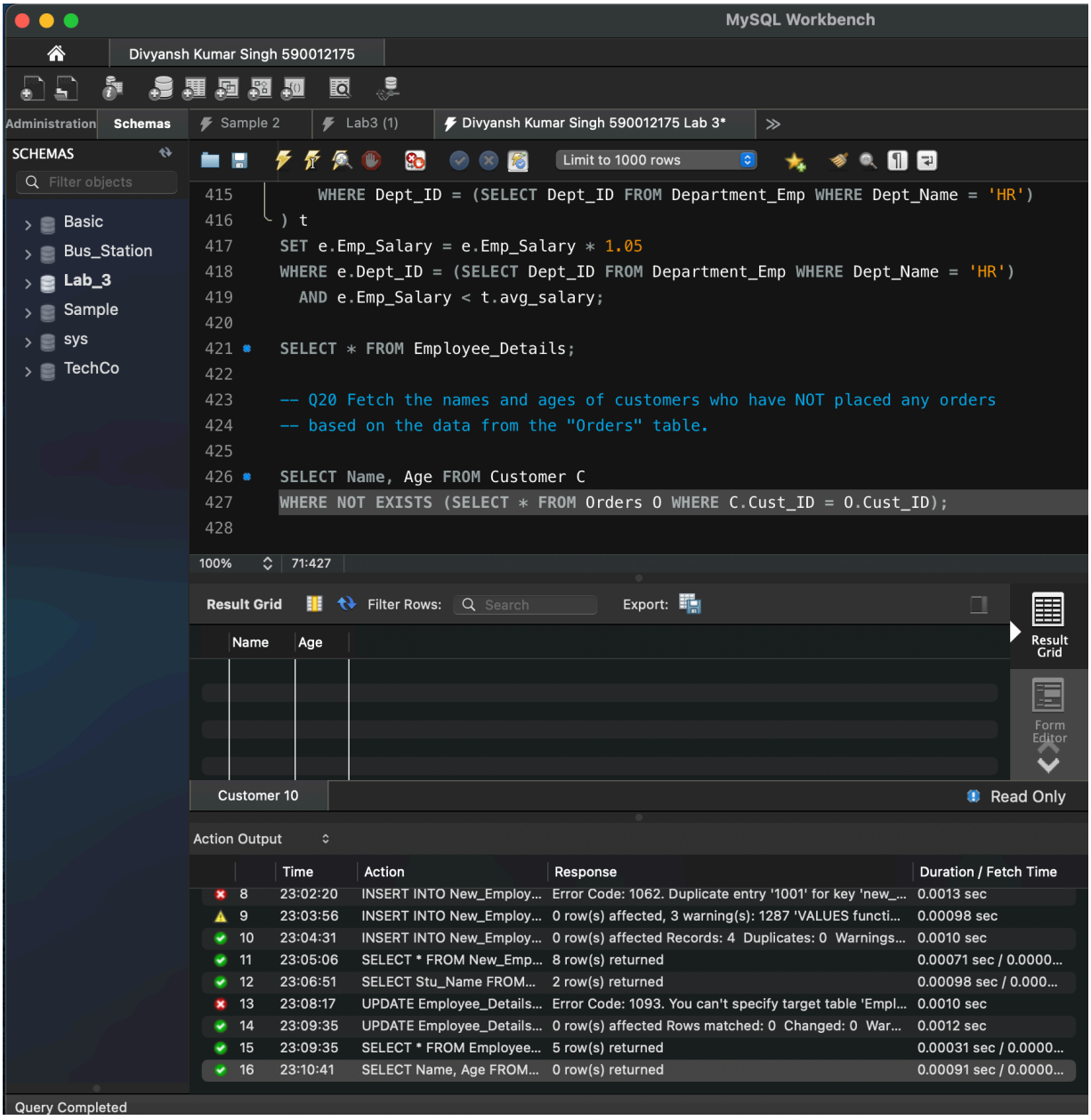
```
UPDATE Employee_Details e
JOIN (
   SELECT AVG(Emp_Salary) AS avg_salary
   FROM Employee_Details
        WHERE Dept_ID = (SELECT Dept_ID FROM Department_Emp
WHERE Dept_Name = 'HR')
) t
SET e.Emp_Salary = e.Emp_Salary * 1.05
WHERE e.Dept_ID = (SELECT Dept_ID FROM Department_Emp WHERE
Dept_Name = 'HR')
  AND e.Emp_Salary < t.avg_salary;

SELECT * FROM Employee_Details;
```



20. Subquery with **NOT EXISTS**

- Fetch the names and ages of customers who have NOT placed any orders based on the data from the "Orders" table.

```
SELECT Name, Age FROM Customer C
WHERE NOT EXISTS (SELECT * FROM Orders O WHERE C.Cust_ID =
O.Cust_ID);
```



CONCLUSION

In this lab, we successfully implemented different types of SQL subqueries using `SELECT`, `INSERT`, `UPDATE`, and `DELETE`.

We explored operators like `IN`, `ANY`, `ALL`, `EXISTS`, and `NOT EXISTS` with practical table data.

The exercise demonstrated the power of subqueries in filtering, aggregating, and updating data efficiently.

We also learned how correlated and nested subqueries provide deeper insights in complex queries.

Overall, this lab enhanced our understanding of SQL subqueries and their applications in real-world database operations.