7/08/2025.

RANKA
DATE / /
PAGE

DBMS Lab

Name : Yashasvani
SAP ID : 590011386

* Assessment :- 0 - 10 Labs (40M)
  - Attend. 5M
  - Performance 5M
* Lab file :- 10 Marks (10M)
* Lab exam :- 50 Marks (50M)
  - Mid 50M
  - Cont 50M

Lab-0
Installation and schema Design

Oracle Apex :- Interface.

Step-1 : Type Oracle Apex in google search engine.

Step-2 : Sign In the account in Oracle Apex.

Step-3 : Request a workspace.

Step-4 : → email
→ unique workspace name (YA_DBMS_25)

Step-5 : Yes / Yes

RANKA
DATE / /
PAGE

Step-6 : → Once you logged in
→ SQL workshop.
→ SQL commands.

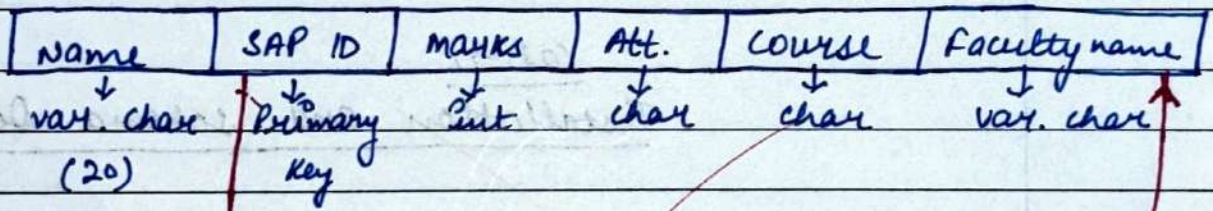
Ques : Design the schema:

1) for (student / Faculty / Employee)

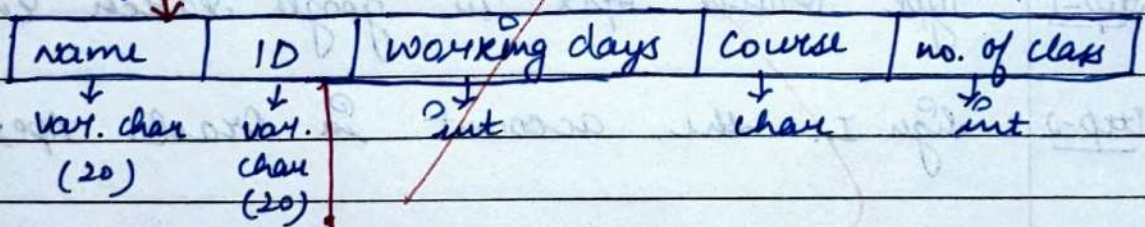
2) Identify the (job description / role / Timetable).

3) How they are (communicating with each other).

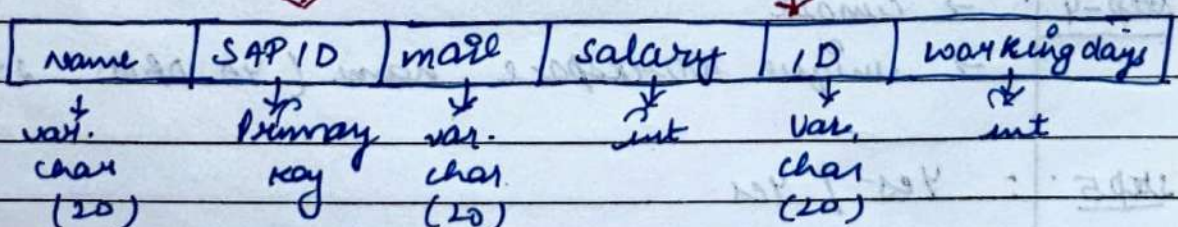
* Student:



* Faculty:



* Employee:



Name: Yashaswani
SAP ID: 590011386
Batch: 2.
Date: 14/08/25.

RANKA
DATE / /
PAGE

14/8/25

Lab-1
Implementation of "DDL" and "DML"

Lab performance questions:

Ques 1: What is the difference between "Truncate" and "Delete"?

	Truncate	Delete
1)	Removes all rows from a table quickly.	1) Removes rows from a table one at a time based on where cond".
2)	Cannot include a WHERE clause.	2) Can include a WHERE clause.
3)	It is a DDL command	3) It is a DML command.
4)	Resets Table storage space back to initial state.	4) Does not reset storage.
5)	Cannot be rolled back in many RDBMS	5) Can be rolled back before commit.
6)	Faster as it does not log row-by-row deletes.	6) Slower as each row deletion is logged.

Ques 2: What is the difference b/w "Truncate" and "DROP"?

	Truncate	DROP.
1)	Remove all rows from a table but keeps the table structure for future use.	1) Removes the entire table, including its structure, data, index, and constant

RANKA
DATE / /
PAGE

2) It is a DDL command.	2) It is also a DDL command.
3) Table can still be used after truncation	3) Table must be recreated if you need it again.

Quess: Done.

Quess4: Done.

Quess5: Done.

Quess6: Done.

Quess7: Done.

Name : Yashavani
SAP : 590011386
Date : 21-08-2025

RANKA
DATE / /
PAGE

Lab-2

To understand the concepts of constraints

21/8

Ques1: Done. * Create.
Ques2: Done. * Insert.
Ques3: Done. * Alter.
Ques4: Done. * Alter, Insert, Add.
Ques5: Done. * Alter, Add constraints.
Ques6: Done. * Create.
Ques7: Done. * Alter, Insert.
Ques8: Done. * Alter, Insert.
Ques9: Done. * Create.
Ques10: Done. * Select, From.
Ques11: Done. * Update.
Ques12: Done. * Delete.
Ques13: Done. * Alter.
Ques14: Done. * Select.

Name : Yashasvani
SAP : 590011386
Date : 28/08/2025.

RANKA
DATE / /
PAGE

28/8/25

55

Lab-3
Use of SQL sub-query

create → 11 queries.
desc → 11 queries.
Insert Into → 29 queries.
Select → 3 queries.
Insert Into → 1 query.

Total → 55 queries.

Select → 41 queries.
where → 25 queries.
Insert into → 2 queries.
update → 2 queries.
set → 2 queries.
Delete → 1 query.
~~Delete →~~
as → 4 queries.
Groupby → 4 queries.
where exists → 1 query.
where not exists → 1 query.
join → 1 query.
commit → 2 queries.
And → 3 queries.

Total → 144 queries.

Name: Yashasvani
SAP: 590011386
Date: 4/09/2025.

RANKA
DATE / /
PAGE

Lab-4

To implement the use of In-built functions.

Create Table → 2
Desc → 2
Insert Into → 3
values → 3
Select → 49
From → 49
Where → 3
Group by → 5
Order by → 4
Rownum → 2
count() → 1
Sum() → 1
avg() → 1
min() → 1
max() → 1
Length() → 1
LengthB() → 1
RPAD() → 2
LPAD() → 1
LTRIM() → 1
RTRIM() → 1
REPLACE() → 1
SUBSTR() → 6
INSTR() → 2
LOWER() → 1
HEX() → 1
ASCII() → 1
TO_char() → 3
TO_number() → 1

count() → 1
Sum() → 1
Avg() → 1
Min() → 2
Max() → 2
Rownum → 2
Listagg → 4+1
Concat_WT() → 1
Length() → 2+2
ELT() → 1
Export_set() → 1
FIELD() → 1
INSTR() → 2+2
TO_CHAR() → 1
_Base64() → 1
RAWTOHEX() → 1
SUBSTR() → 3+2+1
LOWER() → 2
LPAD() → 1
LTRIM() → 1
LENGTHB() → 1
TO_CHAR(TRUNC()) → 1
TO_CHR(TRUNC(MOD)) → 1
TO_CHAR(MOD) → 1
ASCII(SUBSTR()) → 1
RPAD() → 2
REPLACE() → 1
RTRIM() → 1
SOUNDEX() → 1
DBMS_RANDOM.value() → 1

RANKA
DATE / /
PAGE

EXPORT_SET() → 1
FIELD() → 1
FIND_IN_SET() → 1
FORMAT() → 1
MAKE_SET() → 1
GROUP_CONCAT() → 4
LISTAGG → 2
LIKE → 1
~~DIST~~ DISTINCT → 2
WITHIN GROUP (order by) → 1
SOUNDEX() → 1
CONNECT BY → 1
PRIOR → 1
DBMS_RANDOM.value → 1
~~Total queries → 70~~

Group by → 5
ROUND() → 1
POWER() → 2
ABS() → 1
ACOS() → 1+2
SIGN() → 1
SIN() → 1
SQRT() → 1
TAN() → 1
TRUNC() → 2
ASIN() → 1
ATAN2() → 1
ATAN() → 1
CEIL() → 1+1
COS() → 2
EXP() → 1
FLOOR() → 1
GREATEST() → 1
LEAST() → 1
LN() → 1
LOG() → 1+1
MOD() → 1

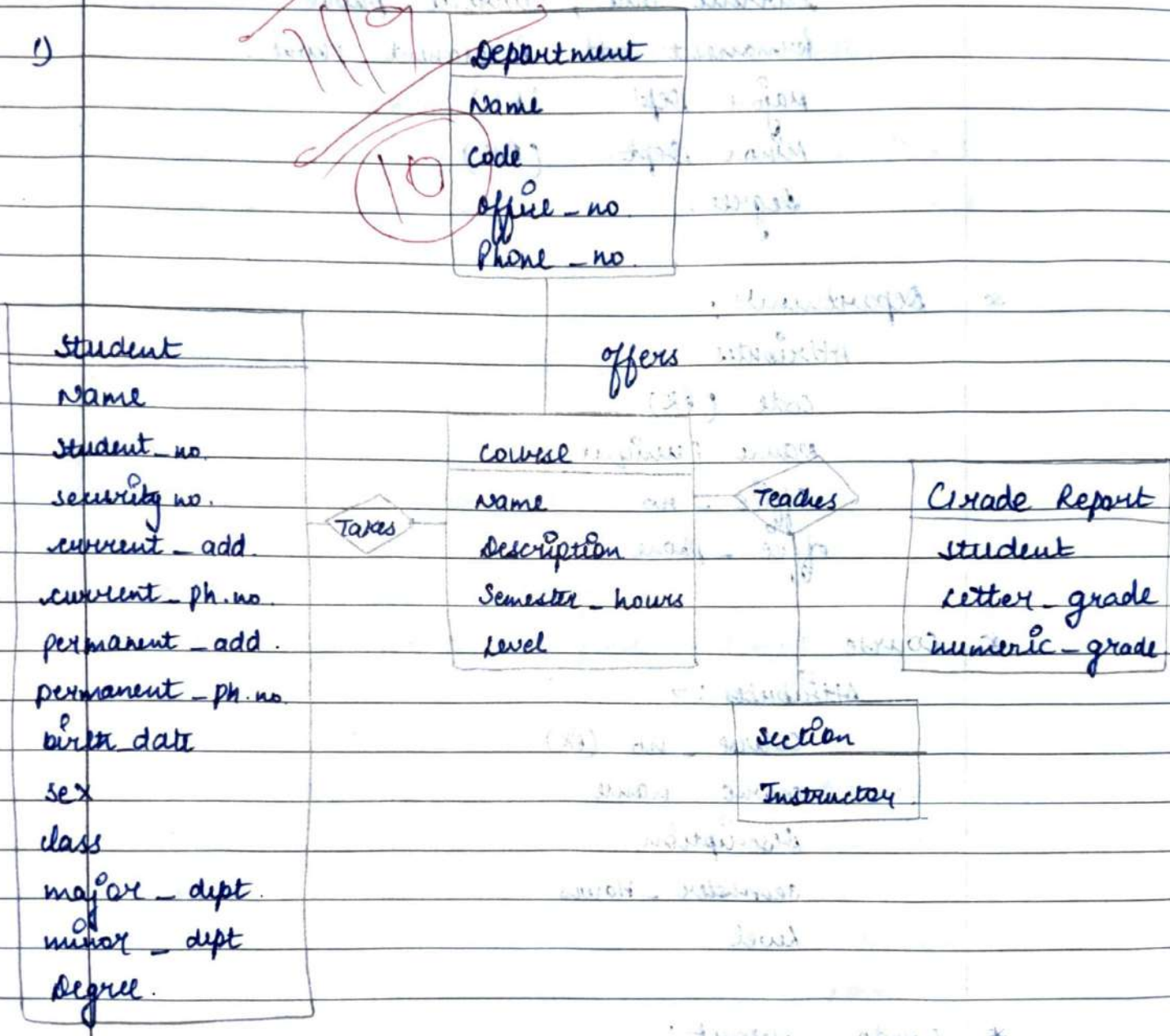
Total queries → 86 queries.

Name: Yashasvani
SAP: 590011386
Date: 11-09-2025.

RANKA
DATE / /
PAGE

Lab-5
Understanding of SQL

1)



* Student :
  Attributes :→
  Student_no. (PK)
  SSN (unique)
  Name
  Birth_date

Sex
Class
current_add., current_phone
permanent_add:, permanent_phone.
Major Dept. (FK)
Minor Dept. (FK)
Degree.

* Department :
Attributes :→
Code (PK)
Name (unique)
Office_no.
office_phone

* Course :
Attributes :→
Course_no. (PK)
course_name
Description.
Semester_Hours
Level

* Grade_report :
Attributes :→
Student_no. (FK)
Letter_grade
Numeric_Grade. (0,1,2,3,4).

RANKA
DATE / /
PAGE

Relationships

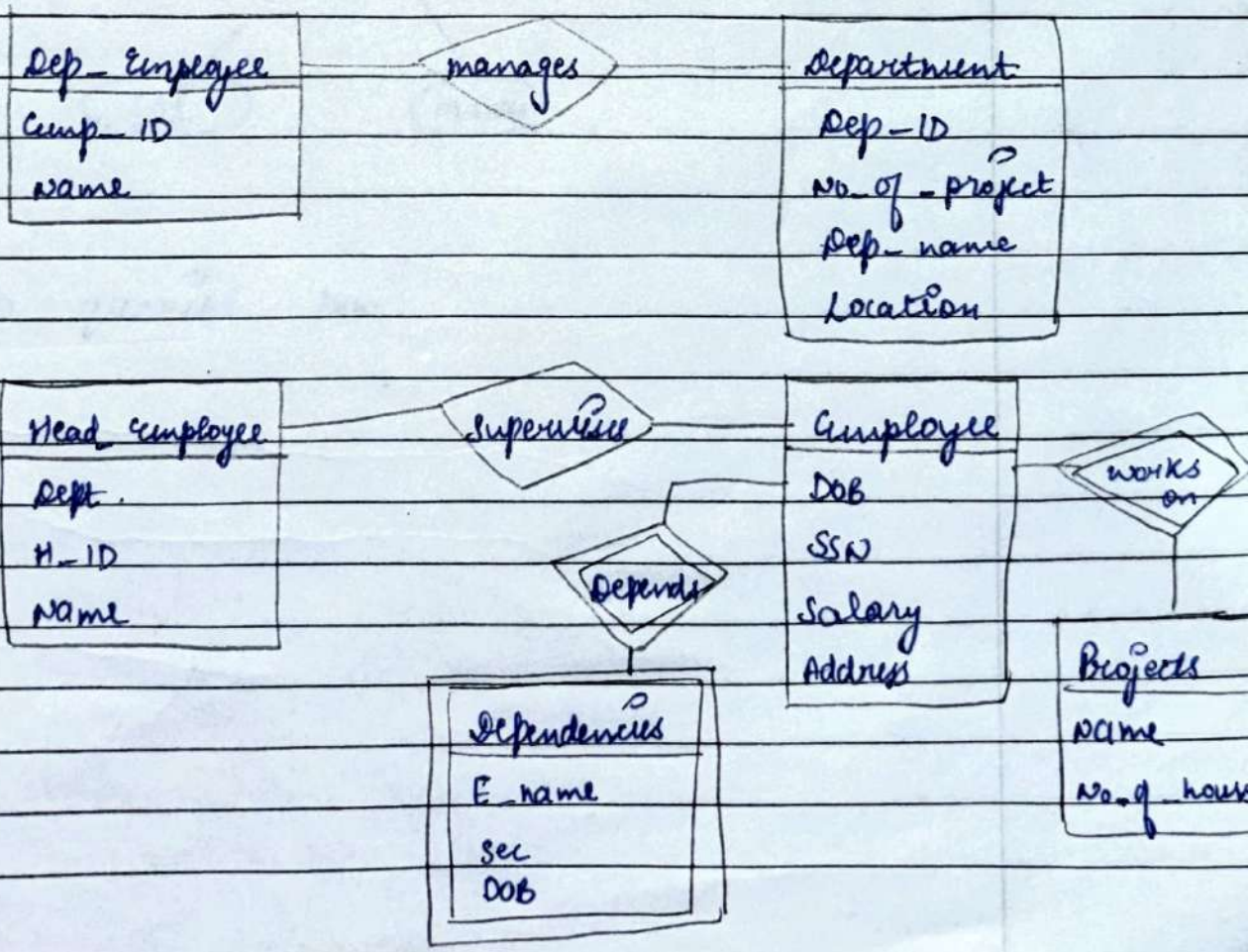
Student – Department.
A student has one major and optional minor department
Student (Major_Dept.) → Department (1:M)
Student (Minor_Dept) → Department (0:M).

Department – Course.
Department → Course (1:M)

Course – Section.
Course → Section (1:M)

student → Grade_report (1:M).
Section → Grade_report (1:M).

2)



Name : Yashaswani

Date : 16/10/25

SAP : 590011386

B-2

16/10

① Student_info ⋈ Student_marks

ID	Name	dept.	Subject	Marks
11	X	P	Maths	20
12	Y	Q	DAA	22
13	Z	R	Discrete	18
14	W	S	OS	14

② Student_info ⋈ Student_info.ID = Student_marks.ID . Student_marks.

Name: Yashasvani

SAP : 590011386

Lab-8

My SQL Views

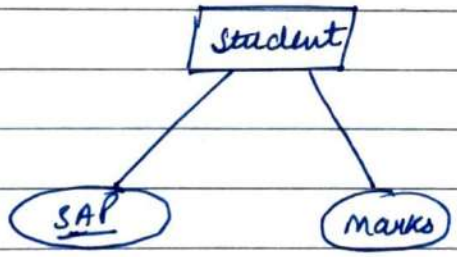
Ques: Create a small ER diagram of student information & write relational calculus.

σ (Student ⋈ Marks)

Student · ID = Marks · ID.

Student :

SAP	Marks
A	20
B	30



30 queries run.

Name: Yashaswani.
SAP: 590011386.

RANKA
DATE / /
PAGE

Lab-9
Indexing

* create Database
* create tabll
* Insert data
* create indexes.
* Show
* Select where
* create index
* Select , from, where
* Select , from , index use
* Show index
* Drop index

11 queries.

Name: Yashaswani
SAP : 590011386

RANKA
DATE / /
PAGE

24/11

Lab-10

PL / SQL commands.

* CONSTANT
* pi
* Boolean flag
* MOD
* SQRT
* length
* Substr
* Greatest
* Least
* POWER.

All done

NAME: Yashasvani

SAP ID: 590011386

DBMS LAB

INDEX

S. No.	Experiment	Date of Submission	Signature
1	Installing Oracle Apex	07/08/2025	W 7/8
2	To implement DDL and DML commands.	14/08/2025	W 14/8
3	To Implement and Understand the Concept Of Constraints	21/08/2025	W 21/8
4	To implement and use SQL Sub-Query	28/08/2025	W 28/8
5	To implement and use Inbuilt Functions	04/09/2025	W 4/9
6	Understanding of SQL	11/09/2025	W 11/9
7	To understand the use of SQL Inbuilt functions.	07/10/2025	W 7/10
8	To understand the use of group by and having clause and execute the SQL commands using JOIN	16/10/2025	W 16/10
9	MySQL Views	30/10/2025	W 30/10
10	Indexing	06/11/2025	W 6/11
11	PL/SQL Commands	~~13~~/11/2025	W ~~27~~/11

EXPERIMENT-3

Lab 3: To implement and use SQL Sub-Query

Objective: To understand the use of SQL Subquery.

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

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

SQL Subquery Questions:

1. Basic **SELECT** Subquery
   - Retrieve the names and marks of students whose marks are greater than the average marks.

2. **IN Operator** Subquery
   - Fetch the names and addresses of faculties who belong to departments in either 'Noida' or '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.

4. **UPDATE** with Subquery:
   - Increase the salary of employees by 10% for those whose department grade is 'A.'

5. **DELETE** with Subquery:
   - Delete the records of employees whose department grade is 'C.'

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.

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.

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.

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

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

11. Subquery with **ALL Operator**
    - Find customers whose cust_id is greater than all cust_ids in the "Orders" table.

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.

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.

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.

15. Multiple **Subqueries in a Single Query**
    - Write an SQL query involving multiple subqueries to retrieve information about employees based on various conditions.

16. Subquery with **BETWEEN** Operators
    - Find the names and marks of students who scored 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.

18. Subquery with **ANY Operator**
    - Retrieve the names of students who scored more marks than ANY student from the "Top_Students" table.

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.

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.

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:

- Subqueries should always be in **parentheses.**
- If the main query does not have multiple columns for the subquery, then a subquery can have only one column in the SELECT command.
- 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.
- We cannot use the **ORDER BY** clause in a subquery, although it can be used inside the main query.
- 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:

- The subqueries make the queries in a structured form that allows us to isolate each part of a statement.
- The subqueries provide alternative ways to query the data from the table; otherwise, we need to use complex joins and unions.
- 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;
```

Examples of Subquery with the SELECT Statement

Example 1: This example uses the **Greater than comparison operator** with the subquery.

Let's take the table Student_Details, which contains Student_RollNo., Stu_Name, Stu_Marks, and Stu_City column.

Student_RollNo.	Stu_Name	Stu_Marks	Stu_City
1001	Akhil	85	Agra
1002	Balram	78	Delhi
1003	Bheem	87	Gurgaon
1004	Chetan	95	Noida
1005	Diksha	99	Agra
1006	Raman	90	Ghaziabad
1007	Sheetal	68	Delhi

The following SQL query returns the record of those students whose marks are greater than the average total marks:

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

Output:

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

Example 2: This example uses the **IN operator** with the subquery.

Let's take the following two tables named **Faculty_Details** and **Department** tables. The **Faculty_Details** table contains the ID, Name, Dept_ID, and address of faculties. The Department table contains the Dept_ID, Faculty_ID, and Dept_Name.

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

Dept_ID	Faculty_ID	Dept_Name
1	101	BCA
2	102	B.Tech
3	105	BBA
4	104	MBA
5	107	MCA

```
SELECT * FROM Department WHERE Faculty_ID IN (
SELECT Faculty_ID FROM Faculty WHERE City = 'Noida' OR City = 'Gurgaon' ) ;
```

Output:

Dept_ID	Faculty_ID	Dept_Name
1	101	BCA
2	102	B.Tech
3	105	BBA
5	107	MCA

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);

Examples of Subquery with the INSERT Statement

Example1: This example inserts the record of one table into another table using subquery with **WHERE clause**.

Let's take Old_Employee and New_Employee tables. The Old_Employee and New_Employee table contain the same number of columns. But, both the tables contain different records.

Table: Old_Employee

Emp_ID	Emp_Name	Emp_Salary	Address
1001	Akhil	50000	Agra
1002	Balram	25000	Delhi
1003	Bheem	45000	Gurgaon
1004	Chetan	60000	Noida
1005	Diksha	30000	Agra
1006	Raman	50000	Ghaziabad
1007	Sheetal	35000	Delhi

Table: New_Employee

Emp_ID	Emp_Name	Emp_Salary	Address
1008	Sumit	50000	Agra
1009	Akash	55000	Delhi
1010	Devansh	65000	Gurgaon

The New_Employee contains the details of new employees. If you want to move the details of those employees whose salary is greater than 40000 from the Old_Employee table to the New_Employee table. Then for this issue, you have to type the following query in SQL:

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

Now, you can check the details of the updated New_Employee table by using the following SELECT query:

SELECT * FROM New_Employee;

Output:

Table: New_Employee

Emp_ID	Emp_Name	Emp_Salary	Address
1008	Sumit	50000	Agra
1009	Akash	55000	Delhi
1010	Devansh	65000	Gurgaon
1001	Akhil	50000	Agra
1003	Bheem	45000	Gurgaon
1004	Chetan	60000	Noida
1006	Raman	50000	Ghaziabad

Example 2: This example describes how to use **ANY operator** with subquery in the INSERT Statement.

Here we have taken the New_Employee, old_Employee, and Department table. The data of the New_Employee table is shown in the following table:

Table: New_Employee

Emp_ID	Emp_Name	Emp_Salary	Dept_ID
1008	Sumit	50000	401

The data of the old_Employee table is shown in the below table:

Table: Old_Employee

Emp_ID	Emp_Name	Emp_Salary	Dept_ID
1001	Akhil	50000	404
1002	Balram	25000	403
1003	Bheem	45000	405
1004	Chetan	60000	402
1005	Ram	65000	407
1006	Shyam	55500	NULL
1007	Shobhit	60000	NULL

The data of Department table is shown in the below table:

Dept_ID	Dept_Name	Emp_ID
401	Administration	1008
402	HR	1004
403	Testing	1002
404	Coding	1001
405	Sales	1003
406	Marketing	NULL
407	Accounting	1005

```
INSERT INTO New_Employee
SELECT * FROM Old_Employee
WHERE Emp_ID = ANY( SELECT Emp_ID FROM Department WHERE Dept_ID = 407 OR Dept_ID = 406 );
```

Now, check the details of the **New_Employee table by using the following SELECT statement:**

Output:

Emp_ID	Emp_Name	Emp_Salary	Dept_ID
1008	Sumit	50000	401
1005	Ram	65000	407

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) ;
```

Example of Subquery with the UPDATE statement

This example updates the record of one table using the IN operator with subquery in the UPDATE statement. Let's take an Employee_Details and Department table. The data of the Employee_Details table is shown in the following table:

Table: Employee_Details

Emp_ID	Emp_Name	Emp_Salary	Dept_ID

1001	Akhil	50000	404
1002	Balram	25000	403
1003	Bheem	45000	405
1004	Chetan	60000	402
1005	Ram	65000	407
1006	Shyam	55500	NULL
1007	Shobhit	60000	NULL

The data of Department table is shown in the below table:

Table: Department

Dept_ID	Dept_Name	Emp_ID	Dept_Grade
401	Administration	1008	B
402	HR	1004	A
403	Testing	1002	A
404	Coding	1001	B
405	Sales	1003	A
406	Marketing	NULL	C
407	Accounting	1005	A

The following updates the salary of those employees whose Department Grade is A:

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

The following query will show the updated data of the Employee_Details table in the output:

```
SELECT * FROM Employee_Details ;
```

Output:

Table: Employee_Details

Emp_ID	Emp_Name	Emp_Salary	Dept_ID
1001	Akhil	50000	404
1002	Balram	30000	403
1003	Bheem	50000	405
1004	Chetan	65000	402
1005	Ram	70000	407
1006	Shyam	55500	NULL
1007	Shobhit	60000	NULL

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 TABL
E_NAME WHERE Condition) ;
```

Example of Subquery with DELETE statement

This example deletes the records from the table using the IN operator with subquery in the DELETE statement. Let's take an Employee_Details and Department table. The data of the Employee_Details table is shown in the following table:

Table: Employee_Details

Emp_ID	Emp_Name	Emp_Salary	Dept_ID
1001	Akhil	50000	404
1002	Balram	25000	403
1003	Bheem	45000	405
1004	Chetan	60000	402
1005	Ram	65000	407
1006	Shyam	55500	NULL
1007	Shobhit	60000	NULL
1008	Ankit	48000	401

The data of Department table is shown in the below table:

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

The following query deletes the record of those employees from the Employee_Details whose Department Grade is C:

```
DELETE FROM Employee_Details WHERE Emp_ID IN ( SELECT Emp_ID FROM Department WHE
RE Dept_Grade = 'C' ) ;
```

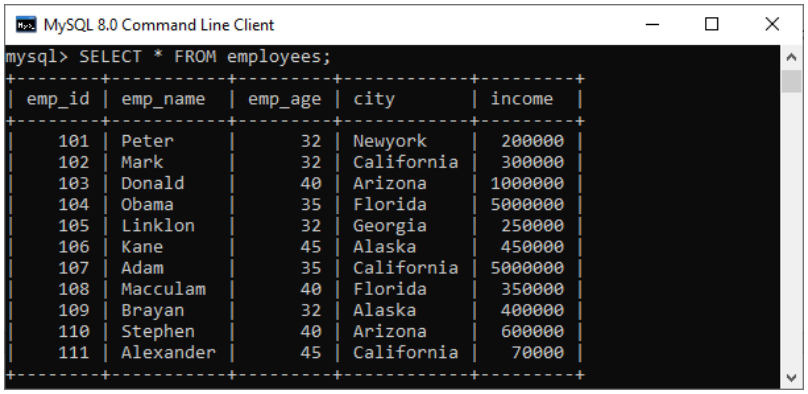
The following query will show the updated data of the Employee_Details table in the output:

```
SELECT * FROM Employee_Details ;
```

Output:

Table: Employee_Details

Emp_ID	Emp_Name	Emp_Salary	Dept_ID
1001	Akhil	50000	404
1003	Bheem	45000	405
1004	Chetan	60000	402
1006	Shyam	55500	NULL
1007	Shobhit	60000	NULL



MySQL Subquery with Comparison Operator

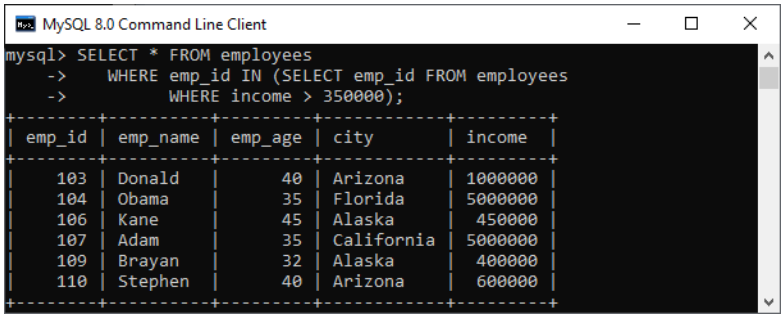
A comparison operator is an operator used to compare values and returns the result, either true or false. The following comparison operators are used in MySQL <, >, =, <>, <=>, etc. We can use the subquery before or after the comparison operators that return a single value. The returned value can be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

column 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);
```

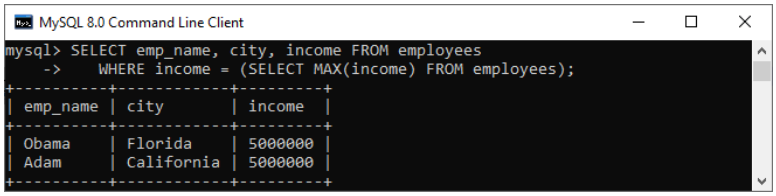
This query first executes the subquery that returns the **employee id whose income > 350000**. Second, the main query will return the employees all details whose employee id are in the result set returned by the subquery.
After executing the statement, we will get the below output, where we can see the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

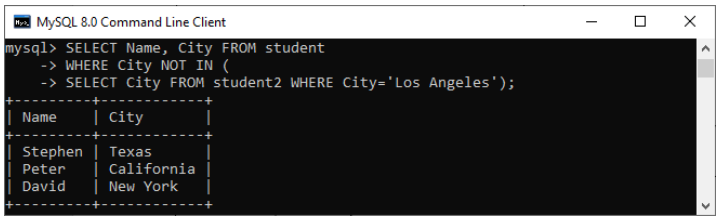
Table: Student2

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.

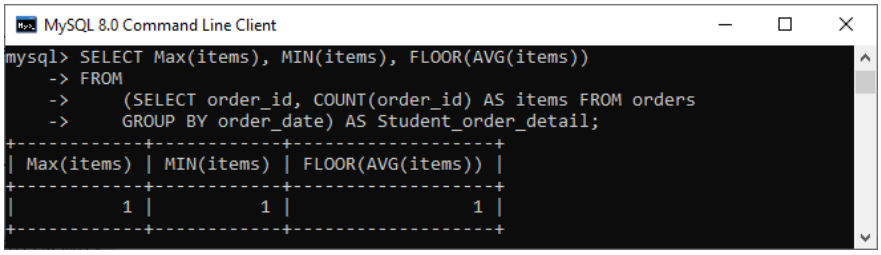


MySQL Subquery in the FROM Clause

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;
```

It will give the output as follows:

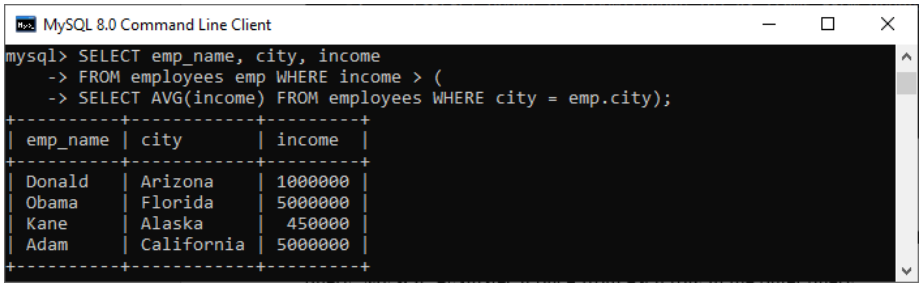


MySQL Correlated Subqueries

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);
```

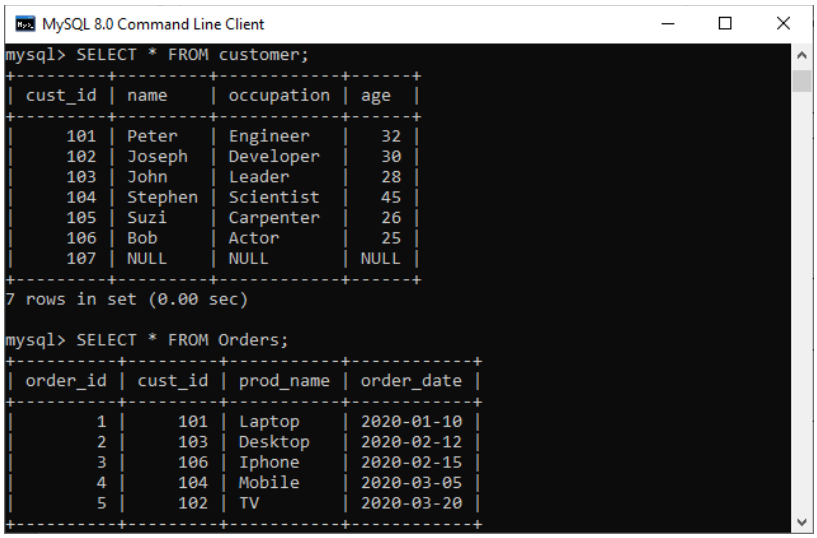
In the above query, we select an **employee name and city** whose income is higher than the average income of all employees in each city.



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** that contains the data as follows:



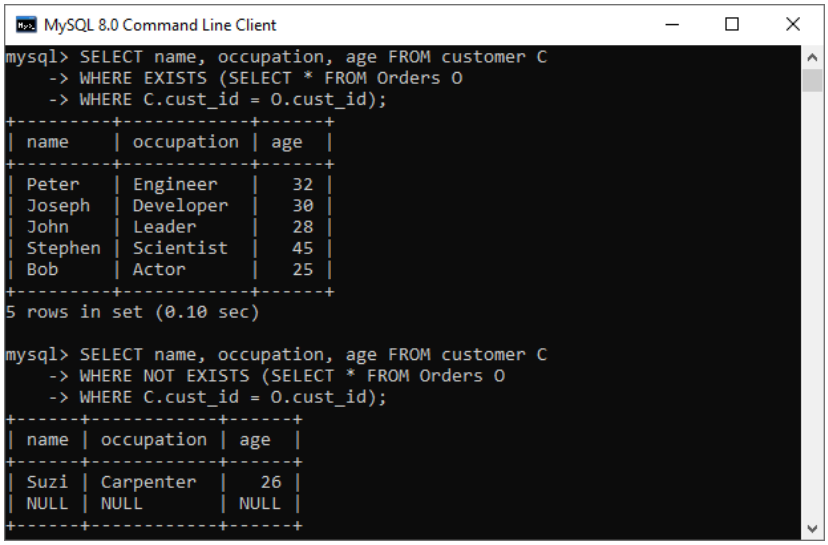
The below SQL statements uses EXISTS operator to find the name, occupation, and age of the customer who has placed at least one order.

```
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.

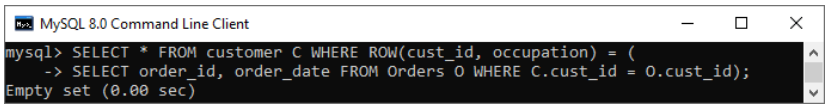


MySQL ROW Subqueries

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, which can be shown in the below image:



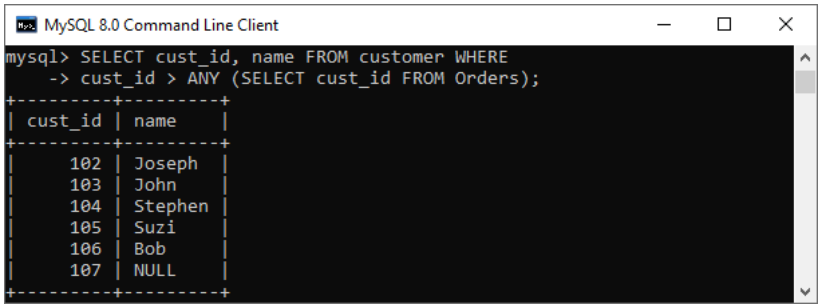
MySQL Subqueries with ALL, ANY, and SOME

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:

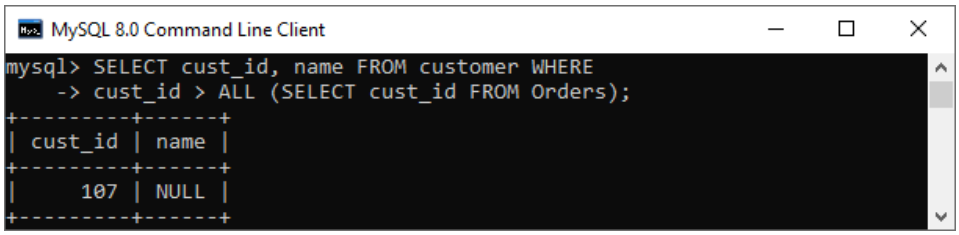
```
SELECT cust_id, name FROM customer WHERE
cust_id > ANY (SELECT cust_id FROM Orders);
```



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);
```

We can see the output as below:



EXPERIMENT-4

Lab 4: To implement and use Inbuilt Functions

Objective: To understand the use of SQL Inbuilt functions.

Inbuilt Functions

1. **Aggregate Functions**
2. **String Functions**
3. **Mathematical Functions**
4. **Date and Time Functions**
5. **Conditional Functions**

1. Aggregate Functions

COUNT(): Counts the number of rows that satisfy the specified condition.
Example: SELECT COUNT(name) FROM employee;

SUM(): Calculates the sum of the specified column values.
Example: SELECT SUM(working_hours) AS "Total working hours" FROM employee;

AVG(): Calculates the average of the specified column values.
Example: SELECT AVG(working_hours) AS "Average working hours" FROM employee;

MIN(): Retrieves the minimum value of the specified column.
Example: SELECT MIN(working_hours) AS Minimum_working_hours FROM employee;

MAX(): Retrieves the maximum value of the specified column.
SELECT MAX(working_hours) AS Maximum_working_hours FROM employee;

LIMIT: Limits the number of rows returned by a query.
Example: SELECT working_date FROM employee LIMIT 1;

ORDER BY with LIMIT: Orders the result set by a specified column and limits the number of rows returned.
Example: SELECT working_hours FROM employee ORDER BY name DESC LIMIT 1;

GROUP_CONCAT(): Concatenates values from multiple rows into a single string within each group.
Example: SELECT emp_id, emp_fname, emp_lname, dept_id, GROUP_CONCAT(designation) AS designation FROM employee2 GROUP BY emp_id, emp_fname, emp_lname, dept_id;

GROUP_CONCAT() with DISTINCT: Concatenates distinct values from multiple rows into a single string within each group.
Example: SELECT emp_fname, dept_id, GROUP_CONCAT(DISTINCT designation) AS designation FROM employee2 GROUP BY emp_id, emp_fname, emp_lname, dept_id;

GROUP_CONCAT() with Separator: Concatenates values using a specified separator within each group.
Example: SELECT emp_fname, GROUP_CONCAT(DISTINCT designation SEPARATOR ';') AS designation FROM employee2 GROUP BY emp_id, emp_fname, emp_lname, dept_id;

GROUP_CONCAT() with CONCAT_WS() and Separator: Concatenates values using a specified separator and a specified separator between values within each group.
Example: SELECT GROUP_CONCAT(CONCAT_WS(',', emp_lname, emp_fname) SEPARATOR ';') AS employeename FROM employee2;

2. MySQL String Function

CONCAT_WS(): Concatenates arguments using a specified separator.
CONCAT(): Concatenates all the arguments into a string.
CHARACTER_LENGTH(): Returns the size of the specified string.
ELT(): Returns the Nth element from a list of strings.
EXPORT_SET(): Returns a string for each bit set.
FIELD(): Returns the index of a string.
FIND_IN_SET(): Returns the value of a string given its position in the argument.
FORMAT(): Formats a number with a specified number of decimal places.
FROM_BASE64(): Encodes a given string to binary format.
HEX(): Returns a specified number or string in hexadecimal format.
INSERT(): Inserts a character into a string at a specified position.
INSTR(): Returns the first occurrence of a substring in a string.
LCASE(): Returns the given string in lowercase.
LEFT(): Returns a specified number of characters from the left side of a string.
LENGTH(): Returns the length of the specified string in bytes.
LIKE(): Checks for pattern matching and returns either 1 or 0.
LOAD_FILE(): Returns the content of a file.
LOCATE(): Returns the position of a given substring in a string.
LOWER(): Returns the given string in lowercase.
LPAD(): Left-pads a string to a specified length.
LTRIM(): Removes leading spaces from a string.
MAKE_SET(): Returns values from a set for the given bit.
MID(): Extracts a substring from a string.
OCTET_LENGTH(): Returns the length of a given string.
OCT(): Returns the octal representation of a number.
ORD(): Returns the code for the leftmost character if it is multi-byte.
POSITION(): Returns the position of a given substring in a string.
QUOTE(): Returns the string passed in single quotes.
REPEAT(): Repeats a string for a specified number of times.
REPLACE(): Replaces all occurrences of a substring within a string.
REVERSE(): Reverses a string.

RIGHT(): Extracts a specified number of characters from the right side of a string.
RPAD(): Pads the specified strings from the right.
RTRIM(): Removes trailing spaces from a specified string.
SOUNDEX(): Returns the soundex string for the specified string.

3. Mathematical Functions

ABS(): Returns the absolute value of the specified number.
ACOS(): Returns the arc cosine of the given number.
SIGN(): Returns the sign of the specified number.
SIN(): Returns the sine value of the given number.
SQRT(): Returns the square root of the given number.
SUM(): Sums the values of given expressions.
TAN(): Returns the tangent of the given number.
TRUNCATE(): Truncates the given number up to a certain number of decimal places.
ASIN(): Returns the arc sine of the given number.
ATAN2(): Returns the arc tangent of the specified numbers n and m.
ATAN(): Returns the arc tangent of the specified number.
AVG(): Calculates the average value from the given expression.
CEIL() (or CEILING()): Returns the smallest value greater than or equal to the specified number.
COS(): Returns the cosine of the given number.
COT(): Returns the cotangent of the given number.
COUNT(): Gets the total count for the specified column of the table.
DEGREES(): Converts the given radian number into a degree.
DIV(): Finds the integer division by dividing the number n by the number m.
EXP(): Finds e raised to the power of the given number.
FLOOR(): Finds the greatest integer less than or equal to the specified number.
GREATEST(): Gets the largest number from the list.
LEAST(): Gets the smallest number from the list.
LN(): Gets the natural logarithm for the specified number.
LOG10(): Gets the base-10 logarithm for the specified number.
LOG(): Returns either the natural logarithm or the specified base logarithm of the given number.
LOG2(): Gets the base-2 logarithm for the specified number.
MAX(): Gets the maximum number from the given column.
MIN(): Gets the minimum number from the given column.
MOD(): Gets the remainder for the specified values.
PI(): Gets the value of pi up to 6 decimal places.
POWER(): Gets the power for the specified values.
POW(): Gets the power for the specified values.
RADIANS(): Converts the given degrees to radians.
RAND(): Generates a random number.
ROUND(): Rounds off the specified number.

Lab Performance Questions

```
CREATE TABLE employee(
   name varchar(45) NOT NULL,
   occupation varchar(35) NOT NULL,
   working_date date,
   working_hours varchar(10)
);

desc employee;

INSERT INTO employee VALUES
('Robin', 'Scientist', '2020-10-04', 12),
('Warner', 'Engineer', '2020-10-04', 10),
('Peter', 'Actor', '2020-10-04', 13),
('Marco', 'Doctor', '2020-10-04', 14),
('Brayden', 'Teacher', '2020-10-04', 12),
('Antonio', 'Business', '2020-10-04', 11);

select * from employee;
```

1. `SELECT COUNT(name) FROM employee;`
2. `SELECT SUM(working_hours) AS "Total working hours" FROM employee;`
3. `SELECT AVG(working_hours) AS "Average working hours" FROM employee;`
4. `SELECT MIN(working_hours) AS Minimum_working_hours FROM employee;`
5. `SELECT MAX(working_hours) AS Maximum_working_hours FROM employee;`
6. `SELECT working_date FROM employee LIMIT 1;`
7. `SELECT working_hours FROM employee ORDER BY name DESC LIMIT 1;`

```
CREATE TABLE employee2(
   emp_id varchar(45),
   emp_fname varchar(10),
   emp_lname varchar(10),
   dept_id varchar(10),
   designation varchar(25)
);

desc employee2;

INSERT INTO employee2 VALUES
(1, 'David', 'Miller', 2, 'Engineer'),
(2, 'Peter', 'Watson', 3, 'Manager'),
(3, 'Mark', 'Boucher', 1, 'Scientist'),
(2, 'Peter', 'Watson', 3, 'BDE'),
(1, 'David', 'Miller', 2, 'Developer'),
(4, 'Adam', 'Warner', 4, 'Receptionist'),
(3, 'Mark', 'Boucher', 1, 'Engineer'),
(4, 'Adam', 'Warner', 4, 'Clerk');

select * from employee2;
```

8. SELECT emp_id, emp_fname, emp_lname, dept_id, GROUP_CONCAT(designation) AS designation FROM employee2 GROUP BY emp_id, emp_fname, emp_lname, dept_id;

9. SELECT emp_fname, dept_id, GROUP_CONCAT(DISTINCT designation) as designation FROM employee2 group by emp_id, emp_fname, emp_lname, dept_id;

10. SELECT emp_fname, GROUP_CONCAT(DISTINCT designation SEPARATOR ';') as designation FROM employee2 group by emp_id, emp_fname, emp_lname, dept_id;

11. SELECT GROUP_CONCAT(CONCAT_WS(',', emp_lname, emp_fname) SEPARATOR ';') as employeename FROM employee2;

MySQL String Function

```
CREATE TABLE sample_table (
   id INT,
   first_name VARCHAR(50),
   last_name VARCHAR(50)
);
 desc sample_table;

INSERT INTO sample_table (id, first_name, last_name)
VALUES
   (1, 'John', 'Doe'),
   (2, 'Jane', 'Smith'),
   (3, 'Alice', 'Johnson');
```

12. SELECT id FROM sample_table;

13. SELECT first_name FROM sample_table;

14. SELECT last_name FROM sample_table;

15. SELECT CONCAT_WS('-', first_name, last_name) AS concat_ws_result FROM sample_table;

16. SELECT CONCAT(first_name, ' ', last_name) AS concat_result FROM sample_table;

17. SELECT CHARACTER_LENGTH(first_name) AS char_length_result FROM sample_table;

18. SELECT ELT(1, first_name, last_name, 'Other') AS elt_result FROM sample_table;

19. SELECT EXPORT_SET(id, 1, ',', '') AS export_set_result FROM sample_table;

20. SELECT FIELD(first_name, 'Alice', 'Jane', 'John') AS field_result FROM sample_table;

21. SELECT FIND_IN_SET(first_name, 'John,Jane,Alice') AS find_in_set_result FROM sample_table;

22. SELECT FORMAT(id, 2) AS format_result FROM sample_table;

23. SELECT FROM_BASE64('SGVsbG8gd29ybGQh') AS from_base64_result FROM sample_table;

24. SELECT HEX(id) AS hex_result FROM sample_table;

25. SELECT INSERT(first_name, 2, 0, 'X') AS insert_result FROM sample_table;

26. SELECT INSTR(first_name, 'hn') AS instr_result FROM sample_table;

27. SELECT LCASE(first_name) AS lcase_result FROM sample_table;

28. SELECT LEFT(first_name, 2) AS left_result FROM sample_table;

29. SELECT LENGTH(first_name) AS length_result FROM sample_table;

30. SELECT first_name LIKE 'J%' AS like_result FROM sample_table;

31. SELECT LOAD_FILE('/path/to/file') AS load_file_result FROM sample_table;

32. SELECT LOCATE('a', first_name) AS locate_result FROM sample_table;

33. SELECT LOWER(first_name) AS lower_result FROM sample_table;

34. SELECT LPAD(first_name, 10, '*') AS lpad_result FROM sample_table;

35. SELECT LTRIM(' ' || first_name) AS ltrim_result FROM sample_table;

36. SELECT MAKE_SET(2, 'Red', 'Green', 'Blue') AS make_set_result FROM sample_table;

37. SELECT MID(first_name, 2, 2) AS mid_result FROM sample_table;

38. SELECT OCTET_LENGTH(first_name) AS octet_length_result FROM sample_table;

39. SELECT OCT(id) AS oct_result FROM sample_table;

40. SELECT ORD(LEFT(first_name, 1)) AS ord_result FROM sample_table;

41. SELECT POSITION('oh' IN first_name) AS position_result FROM sample_table;

42. SELECT QUOTE(first_name) AS quote_result FROM sample_table;

43. SELECT REPEAT(first_name, 2) AS repeat_result FROM sample_table;

44. SELECT REPLACE(first_name, 'Jo', 'X') AS replace_result FROM sample_table;

45. SELECT REVERSE(first_name) AS reverse_result FROM sample_table;

46. SELECT RIGHT(last_name, 2) AS right_result FROM sample_table;

47. SELECT RPAD(first_name, 10, '*') AS rpad_result FROM sample_table;

48. SELECT RTRIM(first_name) AS rtrim_result FROM sample_table;

49. SELECT SOUNDEX(first_name) AS soundex_result FROM sample_table;

Mathematical Functions

```
-- Creating a sample employee table
CREATE TABLE employee (
    employee_id INT PRIMARY KEY,
    salary DECIMAL(10, 2),
    age INT
);

-- Inserting sample data into the employee table
INSERT INTO employee (employee_id, salary, age)
VALUES
    (1, 50000.25, 30),
    (2, 60000.75, 35),
    (3, 75000.50, 28);

-- Using various functions in SQL queries on the employee table
```

50. SELECT employee_id FROM employee;

51. SELECT ABS(salary) AS abs_salary FROM employee;

52. SELECT ACOS(salary / 100000) AS acos_salary FROM employee;

53. SELECT SIGN(salary) AS sign_salary FROM employee;

54. SELECT SIN(salary / 100000) AS sin_salary FROM employee;

55. SELECT SQRT(salary) AS sqrt_salary FROM employee;

56. SELECT salary AS sum_salary FROM employee;

57. SELECT TAN(salary / 100000) AS tan_salary FROM employee;

58. SELECT TRUNCATE(salary, 2) AS truncated_salary FROM employee;

59. SELECT ASIN(salary / 100000) AS asin_salary FROM employee;

60. SELECT ATAN2(salary, age) AS atan2_salary FROM employee;

61. SELECT ATAN(salary / 100000) AS atan_salary FROM employee;

62. SELECT salary AS avg_salary FROM employee;

63. SELECT CEIL(salary) AS ceil_salary FROM employee;

64. SELECT CEILING(salary) AS ceiling_salary FROM employee;

65. SELECT COS(salary / 100000) AS cos_salary FROM employee;

66. SELECT COT(salary / 100000) AS cot_salary FROM employee;

67. SELECT 1 AS count_employee FROM employee;

68. SELECT DEGREES(salary / 100000) AS degrees_salary FROM employee;

69. SELECT salary DIV 2 AS div_salary FROM employee;

70. SELECT EXP(salary / 100000) AS exp_salary FROM employee;

71. SELECT FLOOR(salary) AS floor_salary FROM employee;

72. SELECT GREATEST(salary, 0, -50000) AS greatest_salary FROM employee;

73. SELECT LEAST(salary, 0, -50000) AS least_salary FROM employee;

74. SELECT LN(salary / 100000) AS ln_salary FROM employee;

75. SELECT LOG10(salary / 100000) AS log10_salary FROM employee;

76. SELECT LOG(salary / 100000, 2) AS log_salary FROM employee;

77. SELECT LOG2(salary / 100000) AS log2_salary FROM employee;

78. SELECT salary AS max_salary FROM employee;

79. SELECT salary AS min_salary FROM employee;

80. SELECT MOD(salary, 3) AS mod_salary FROM employee;

81. SELECT PI() AS pi_value FROM employee;

82. SELECT POWER(salary, 2) AS power_salary FROM employee;

83. SELECT POW(salary, 2) AS pow_salary FROM employee;

84. SELECT RADIANS(salary / 100000) AS radians_salary FROM employee;

85. SELECT RAND() AS rand_value FROM employee;

86. SELECT ROUND(salary, 2) AS round_salary FROM employee;

Lab 5: Understanding of SQL

Dr. Keshav Sinha

Question 1

Title: Consider the following set of requirements for a UNIVERSITY database that is used to keep track of students' transcripts.

1. The university keeps track of each student's name, student number, Social Security number, current address and phone number, permanent address and phone number, birth date, sex, class (freshman, sophomore, ..., graduate), major department, minor department (if any), and degree program (B.A., B.S., ..., Ph.D.). Some user applications need to refer to the city, state, and ZIP Code of the student's permanent address and to the student's last name. Both Social Security number and student number have unique values for each student.

a. Each department is described by a name, department code, office number, office phone number, and college. Both name and code have unique values for each department.
b. Each course has a course name, description, course number, number of semester hours, level, and offering department. The value of the course number is unique for each course.
c. Each section has an instructor, semester, year, course, and section number. The section number distinguishes sections of the same course that are taught during the same semester/year; its values are 1, 2, 3, ..., up to the number of sections taught during each semester.
d. A grade report has a student, section, letter grade, and numeric grade (0,1, 2, 3, or 4).

Design an Entity-Relationship diagram for the mail order database and enter the design using a data-modeling tool such as ERWin/free tool. Specify key attributes of each entity type, and structural constraints on each relationship type. Note any unspecified requirements and make appropriate assumptions to make the specification complete.

Question 2

Title. Consider the following set of requirements for a Company database that is used to keep track of employee.

The company is organized into departments. Each department has a unique name, a unique number, and a particular employee who manages the department. We keep track of the start date when that employee began managing the department. A department may have several locations.

a. A department controls a number of projects, each of which has a unique name, a unique number, and a single location.

b. We store each employee's name, Social Security number,2 address, salary, sex (gender), and birth date. An employee is assigned to one department, but may work on several projects, which are not necessarily controlled by the same department. We keep track of the current number of hours per week that an employee works on each project. We also keep track of the direct supervisor of each employee (who is another employee).

c. We want to keep track of the dependents of each employee for insurance purposes. We keep each dependent's first name, sex, birth date, and relationship to the employee.

Design an Entity-Relationship diagram for the company database and enter the design using a data-modeling tool such as ERWin/free tool.

Question 3

Objective: To understand the concept of designing issue related to the database with creating, populating the tables. To understand the concept of data constraints that is enforced on data being stored in the table. Focus on Primary Key and the Foreign Key.

a. **Create the tables for** Company database as per ER diagram of Exp 2.

TABLE 1: EMPLOYEE

[Fname VARCHAR(15) NOT NULL,

Minit CHAR,

Lname VARCHAR(15) NOT NULL,
Ssn CHAR(9) NOT NULL,
Bdate DATE,
Address VARCHAR(30),
Sex CHAR,
Salary DECIMAL(10,2),
Super_ssn CHAR(9),
Dno INT NOT NULL,
PRIMARY KEY (Ssn),
FOREIGN KEY (Super_ssn) REFERENCES EMPLOYEE(Ssn),
FOREIGN KEY (Dno) REFERENCES DEPARTMENT(Dnumber)
]

TABLE 2: DEPARTMENT
[Dname VARCHAR(15) NOT NULL,
Dnumber INT NOT NULL,
Mgr_ssn CHAR(9) NOT NULL,
Mgr_start_date DATE,
PRIMARY KEY (Dnumber),
UNIQUE (Dname),
FOREIGN KEY (Mgr_ssn) REFERENCES EMPLOYEE(Ssn));
]

TABLE 3: DEPT_LOCATIONS
(Dnumber INT NOT NULL,
Dlocation VARCHAR(15) NOT NULL,
PRIMARY KEY (Dnumber, Dlocation),
FOREIGN KEY (Dnumber) REFERENCES DEPARTMENT(Dnumber));

TABLE 4: PROJECT
(Pname VARCHAR(15) NOT NULL,
Pnumber INT NOT NULL,
Plocation VARCHAR(15),
Dnum INT NOT NULL,

PRIMARY KEY (Pnumber),
UNIQUE (Pname),
FOREIGN KEY (Dnum) REFERENCES DEPARTMENT(Dnumber));

TABLE 5: WORKS_ON
(Essn CHAR(9) NOT NULL,
Pno INT NOT NULL,
Hours DECIMAL(3,1) NOT NULL,
PRIMARY KEY (Essn, Pno),
FOREIGN KEY (Essn) REFERENCES EMPLOYEE(Ssn),
FOREIGN KEY (Pno) REFERENCES PROJECT(Pnumber));

TABLE 6: DEPENDENT
(Essn CHAR(9) NOT NULL,
Dependent_name VARCHAR(15) NOT NULL,
Sex CHAR,
Bdate DATE,
Relationship VARCHAR(8),
PRIMARY KEY (Essn, Dependent_name),
FOREIGN KEY (Essn) REFERENCES EMPLOYEE(Ssn));

b. Insert the following data into their respective tables of Company database.

DEPARTMENT

DNAME	DNUMBER	MGRSSN	MGRSTARTDATE
Research	5	333445555	1988-05-22
Administration	4	987654321	1995-01-01
Headquarters	1	888665555	1981-06-19

EMPLOYEE

FNAME	LNAME	SSN	BDATE	ADDRESS	SEX	SALARY	SUPERSSN	DNO
John	Smith	123456789	1965-01-09	731 Fondren, Houston TX	M	30000	333445555	5

EMPLOYEE

FNAME	LNAME	SSN	BDATE	ADDRESS	SEX	SALARY	SUPERSSN	DNO
Franklin	Wong	333445555	1965-12-08	638 Voss, Houston TX	M	40000	888665555	5
Alicia	Zelaya	999887777	1968-01-19	3321 Castle, Spring TX	F	25000	987654321	4
Jennifer	Wallace	987654321	1941-06-20	291 Berry, Bellaire TX	F	43000	888665555	4
Ramesh	Narayan	666884444	1962-09-15	975 Fire Oak, Humble TX	M	38000	333445555	5
Joyce	English	453453453	1972-07-31	5631 Rice, Houston TX	F	25000	333445555	5
Ahmad	Jabbar	987987987	1969-03-29	980 Dallas, Houston TX	M	25000	987654321	4
James	Borg	888665555	1937-11-10	450 Stone, Houston TX	M	55000	null	1

PROJECT

PNAME	PNUMBER	PLOCATION	DNUM
ProductX	1	Bellaire	5
ProductY	2	Sugarland	5
ProductZ	3	Houston	5
Computerization	10	Stafford	4
Reorganization	20	Houston	1
Newbenefits	30	Stafford	4

WORKS_ON

ESSN	PNO	HOURS
123456789	1	32.5
123456789	2	7.5
666884444	3	40.0

WORKS_ON

ESSN	PNO	HOURS
453453453	1	20.0
453453453	2	20.0
333445555	2	10.0
333445555	3	10.0
333445555	10	10.0
333445555	20	10.0
999887777	30	30.0
999887777	10	10.0
987987987	10	35.0
987987987	30	5.0
987654321	30	20.0
987654321	20	15.0
888665555	20	null

DEPENDENT

ESSN	DEPENDENT_NAME	SEX	BDATE	RELATIONSHIP
333445555	Alice	F	1986-04-04	Daughter
333445555	Theodore	M	1983-10-25	Son
333445555	Joy	F	1958-05-03	Spouse
987654321	Abner	M	1942-02-28	Spouse
123456789	Michael	M	1988-01-04	Son
123456789	Alice	F	1988-12-30	Daughter
123456789	Elizabeth	F	1967-05-05	Spouse

DEPT_LOCATIONS

DNUMBER	DLOCATION
1	Houston
4	Stafford
5	Bellaire

DEPT_LOCATIONS

DNUMBER	DLOCATION
5	Houston
5	Sugarland

Question 4

Objective: To understand the concept of data constraints that is enforced on data being stored in the table. Focus on Primary Key and the Foreign Key.

1. Create the tables described below:

Table name: CLIENT_MASTER

Description: used to store client information.

Column name	data type	Size	Constraints
CLIENTNO	Varchar	6	Primary key / first letter must start with 'C'
NAME	Varchar	20	Not Null
ADDRESS 1	Varchar	30	
ADDRESS 2	Varchar	30	
CITY	Varchar	15	
PINCODE	Integer	8	
STATE	Varchar	15	
BALDUE	Decimal	10,2	

Table Name: PRODUCT_MASTER

Description: used to store product information

Column name	data type	Size	Attributes
PRODUCTNO	Varchar	6	Primary Key/ first letter must start with 'P'
DESCRIPTION	Varchar	15	Not Null
PROFITPERCENT	Decimal	4,2	Not Null
UNIT MEASURE	Varchar	10	Not Null
QTYONHAND	Integer	8	Not Null

REORDERL VL	Integer	8	Not Null
SELLPRICE	Decimal	8,2	Not Null
COSTPRICE	Decimal	8,2	Not Null

Table Name: SALESMAN_MASTER

Description: used to store salesman information working for the company.

Column name	**data type**	**Size**	**Attributes**
SALESMANNO	Varchar	6	Primary Key/ first letter must start with 'S'
SALESMANNAME	Varchar	20	Not Null
ADDRESS 1	Varchar	30	Not Null
ADDRESS 2	Varchar	30	
CITY	Varchar	20	
PINCODE	Integer	8	
STATE	Varchar	20	
SALAMT	Real	8,2	Not Null , Cannot be 0
TGTTOGET	Decimal	6,2	Not Null , Cannot be 0
YTDSALES	Double	6,2	Not Null
REMARKS	Varchar	60	

1. **Insert the following data into their respective tables:**

   a) Data for **CLIENT_MASTER** table:

Client no	Name	City	Pincode	State	BalDue
C00001	Ivan bayross	Mumbai	400054	Maharashtra	15000
C00002	Mamta muzumdar	Madras	780001	Tamil nadu	0
C00003	Chhaya bankar	Mumbai	400057	Maharashtra	5000
C00004	Ashwini joshi	Bangalore	560001	Karnataka	0
C00005	Hansel colaco	Mumbai	400060	Maharashtra	2000
C00006	Deepak sharma	Mangalore	560050	Karnataka	0

   b) Data for **PRODUCT_MASTER** table:

Product No	Description	Profit percent	Unit measure	Quantity On hand	Recorder Level	Sell Price	Cost Price

P00001	T-Shirt	5	Piece	200	50	350	250
P0345	Shirts	6	Piece	150	50	500	350
P06734	Cotton jeans	5	Piece	100	20	600	450
P07865	Jeans	5	Piece	100	20	750	500
P07868	Trousers	2	Piece	150	50	850	550
P07885	Pull Overs	2.5	Piece	80	30	700	450
P07965	Denim jeans	4	Piece	100	40	350	250
P07975	Lycra tops	5	Piece	70	30	300	175
P08865	Skirts	5	Piece	75	30	450	300

c) Data for **SALESMAN_MASTER** table:

Salesman No	**Name**	**Address1**	**Address2**	**City**	**Pin Code**	**State**
S00001	Aman	A/14	Worli	Mumbai	400002	Maharashtra
S00002	Omkar	65	Nariman	Mumbai	400001	Maharashtra
S00003	Raj	P-7	Bandra	Mumbai	400032	Maharashtra
S00004	Ashish	A/5	Juhu	Mumbai	400044	Maharashtr(a

2. **Exercise on retrieving records from a table.**
   a. Find out the names of all the clients.
   b. Retrieve the entire contents of the Client_Master table.
   c. Retrieve the list of names, city and the state of all the clients.
   d. List the various products available from the Product_Master table.
   e. List all the clients who are located in Mumbai.
   f. Find the names of salesman who have a salary equal to Rs.3000.
3. **Exercise on updating records in a table**
   a. Change the city of ClientNo ‘C00005’ to ‘Bangalore’.
   b. Change the BalDue of ClientNo ‘C00001’ to Rs.1000.
   c. Change the cost price of ‘Trousers’ to rs.950.00.
   d. Change the city of the salesman to Pune.

4. **Exercise on deleting records in a table**
    a. Delete all salesman from the Salesman_Master whose salaries are equal to Rs.3500.

b. Delete all products from Product_Master where the quantity on hand is equal to 100.

c. Delete from Client_Master where the column state holds the value ‘Tamil Nadu’.

5. **Exercise on altering the table structure**
    a. Add a column called ‘Telephone’ of data type integer to the Client_Master table.
    b. Change the size off SellPrice column in Product _Master to 10, 2.
6. **Exercise on deleting the table structure along with the data**
    a. Destroy the table Client_Master along with its data.

EXPERIMENT-5

Lab 5: To implement and use Inbuilt and Control Flow Functions

Objective: To understand the use of SQL Inbuilt functions.

Inbuilt Functions

1. Date and Time Functions

- CURDATE(): Returns the current date.
- DATE_ADD(): Adds a specified time interval to a date.
- DATE_FORMAT(): Formats a date as specified.
- DATEDIFF(): Calculates the difference between two dates.
- DAY(): Extracts the day from a date.
- DAYNAME(): Returns the name of the day for a given date.
- DAYOFMONTH(): Returns the day of the month for a given date.
- DAYOFWEEK(): Returns the day of the week in numeric format for a given date.
- DAYOFYEAR(): Returns the day of the year for a given date.
- FROM_DAYS(): Converts a day number to a date.
- HOUR(): Extracts the hour from a datetime.
- LAST_DAY(): Returns the last day of the month for a given date.
- NOW(): Returns the current date and time.
- PERIOD_ADD(): Adds a specified number of months to a period.
- PERIOD_DIFF(): Calculates the difference between two periods.
- QUARTER(): Returns the quarter of the year for a given date.
- SECOND(): Extracts the second from a datetime.
- STR_TO_DATE(): Converts a string to a date using a specified format.
- SUBDATE(): Subtracts a specified time interval from a date.
- SUBTIME(): Subtracts a specified time interval from a datetime.
- SYSDATE(): Returns the system date and time.
- TIME(): Extracts the time from a datetime.
- TIME_FORMAT(): Formats a time value.
- TIME_TO_SEC(): Converts a time value to seconds.
- TIMEDIFF(): Calculates the difference between two times or datetimes.
- TIMESTAMP(): Converts an expression to a datetime.
- TO_DAYS(): Converts a date to a day number.
- WEEKDAY(): Returns the index of the day of the week for a given date.
- WEEK(): Returns the week number for a given date.
- WEEKOFYEAR(): Returns the week of the year for a given date.

2. Conditional Functions

- AND Operator: The AND operator is used to combine multiple conditions in a WHERE clause, and all conditions must be true for the row to be included in the result set.
- OR Operator: The OR operator is used to combine multiple conditions in a WHERE clause, and at least one of the conditions must be true for the row to be included in the result set.
- AND and OR Combined: You can use both AND and OR operators in a WHERE clause to create complex conditions, ensuring the proper combination of conditions for filtering rows.
- Boolean: MySQL doesn't have a native Boolean data type, but it uses 0 for FALSE and 1 for TRUE. Boolean logic is often expressed using AND, OR, and NOT operators.

- LIKE Operator: The LIKE operator is used in a WHERE clause to search for a specified pattern in a column. It can include wildcards like `%` (matches any sequence of characters) and `_` (matches any single character).
- IN Operator: The IN operator is used in a WHERE clause to specify multiple values for a column. It checks whether a value matches any value in a list.
- ANY Operator: The ANY operator is used with a subquery and returns true if any of the subquery values meet the specified condition.
- EXISTS Operator: The EXISTS operator is used in a WHERE clause to check if a subquery returns any results. If the subquery returns at least one row, the condition is true.
- NOT Operator: The NOT operator is used to negate a condition in a WHERE clause. It reverses the result of a logical condition.
- Not Equal Operator: The `!=` or `<>` operator is used in a WHERE clause to check if two expressions are not equal.
- IS NULL Operator: The IS NULL operator is used in a WHERE clause to check if a column contains a NULL value.
- IS NOT NULL Operator: The IS NOT NULL operator is used in a WHERE clause to check if a column does not contain a NULL value.
- BETWEEN Operator: The BETWEEN operator is used in a WHERE clause to filter the result set within a range. It is inclusive, including the specified values.

Lab Performance Questions

Date and Time Functions

```
CREATE TABLE student (
   ID INT,
   Name VARCHAR(50),
   DateTime_Birth DATETIME,
   City VARCHAR(50)
);

INSERT INTO student VALUES
(1, 'Mansi Shah', '2010-01-01 18:39:09', 'Pune'),
(2, 'Tejal Wagh', '2010-03-04 05:13:19', 'Nasik'),
(3, 'Sejal Kumari', '2010-05-01 10:31:07', 'Mumbai'),
(4, 'Sonal Jain', '2010-09-09 17:17:07', 'Shimla'),
(5, 'Surili Maheshwari', '2010-07-10 20:45:18', 'Surat');
```

```
1  SELECT ID FROM student;
2  SELECT Name FROM student;
3  SELECT DateTime_Birth FROM student;
4  SELECT CURDATE() AS currentdate;
5  SELECT DATE_ADD(DateTime_Birth, INTERVAL 3 DAY) AS date_added FROM
   student;
6  SELECT DATE_FORMAT(DateTime_Birth, '%Y-%m-%d') AS formatted_date FROM
   student;
7  SELECT DATEDIFF(CURDATE(), DATE(DateTime_Birth)) AS date_diff FROM student;
8  SELECT DAY(DateTime_Birth) AS day_value FROM student;
9  SELECT DAYNAME(DateTime_Birth) AS day_name FROM student;
10 SELECT DAYOFMONTH(DateTime_Birth) AS day_of_month FROM student;
11 SELECT DAYOFWEEK(DateTime_Birth) AS day_of_week FROM student;
12 SELECT DAYOFYEAR(DateTime_Birth) AS day_of_year FROM student;
13 SELECT FROM_DAYS(737846) AS from_days_date FROM student;
14 SELECT HOUR(DateTime_Birth) AS hour_value FROM student;
15 SELECT LAST_DAY(DateTime_Birth) AS last_day_of_month FROM student;
16 SELECT NOW() AS current_datetime FROM student;
```

```
17 SELECT PERIOD_ADD(202201, 3) AS period_added FROM student;
18 SELECT PERIOD_DIFF(202203, 202201) AS period_difference FROM student;
19 SELECT QUARTER(DateTime_Birth) AS quarter_value FROM student;
20 SELECT SECOND(DateTime_Birth) AS second_value FROM student;
21 SELECT STR_TO_DATE('2022-05-20', '%Y-%m-%d') AS string_to_date FROM student;
22 SELECT SUBDATE(DateTime_Birth, INTERVAL 2 DAY) AS date_subtracted FROM
   student;
23 SELECT SUBTIME(DateTime_Birth, '03:15:00') AS time_subtracted FROM student;
24 SELECT SYSDATE() AS system_date FROM student;
25 SELECT TIME(DateTime_Birth) AS time_value FROM student;
26 SELECT TIME_FORMAT(DateTime_Birth, '%H:%i:%s') AS formatted_time FROM
   student;
27 SELECT TIME_TO_SEC(DateTime_Birth) AS time_to_seconds FROM student;
28 SELECT TIMEDIFF(NOW(), DateTime_Birth) AS time_difference FROM student;
29 SELECT TIMESTAMP('2022-04-10') AS timestamp_value FROM student;
30 SELECT TO_DAYS('2022-06-15') AS to_days_value FROM student;
31 SELECT WEEKDAY(DateTime_Birth) AS weekday_index FROM student;
32 SELECT WEEK(DateTime_Birth) AS week_value FROM student;
33 SELECT WEEKOFYEAR(DateTime_Birth) AS week_of_year_value FROM student;
```

Conditional Functions

```
CREATE TABLE sample_table (
   column1 INT,
   column2 VARCHAR(50),
   column3 DATE
);

INSERT INTO sample_table VALUES
(1, 'apple', '2022-01-01'),
(2, 'banana', '2022-02-15'),
(3, 'orange', '2022-03-20'),
(4, 'grape', '2022-04-10'),
(5, 'kiwi', '2022-05-05');

34 SELECT * FROM sample_table WHERE column1 > 2 AND column3 > '2022-03-01';
35 SELECT * FROM sample_table WHERE column1 = 2 OR column2 = 'orange';
36 SELECT * FROM sample_table WHERE (column1 > 2 AND column3 > '2022-03-01') OR
   column2 = 'banana';
37 SELECT * FROM sample_table WHERE column2 LIKE 'a%';
38 SELECT * FROM sample_table WHERE column1 IN (2, 4);
39 SELECT * FROM sample_table WHERE column1 > ANY (SELECT column1 FROM
   sample_table WHERE column3 > '2022-03-01');
40 SELECT * FROM sample_table WHERE EXISTS (SELECT * FROM sample_table
   WHERE column1 = 3);
41 SELECT * FROM sample_table WHERE NOT column1 = 2;
42 SELECT * FROM sample_table WHERE column1 <> 2;
43 SELECT * FROM sample_table WHERE column2 IS NULL;
44 SELECT * FROM sample_table WHERE column2 IS NOT NULL;
45 SELECT * FROM sample_table WHERE column1 BETWEEN 2 AND 4;
```

Control-flow-functions

1. **Case operator**
2. **IF()**
3. **IFNULL()**
4. **NULLIF()**

1 CASE Operator

The CASE statement is utilized to implement a complex conditional construct within a stored program.

1. **Conditional Logic:** Allows the implementation of conditional logic directly within a query. Invaluable for handling various scenarios and producing different results based on specified conditions.

2. **Data Transformation:** Can be used to transform data on-the-fly. Example: Categorizing or labeling data based on specific criteria.

3. **Dynamic Column Selection:** Enables the dynamic selection of different columns based on specific conditions.

4. **Contingency Plans:** Allows the setup of contingency plans in case certain conditions are not met.

5. **Customized Reports:** Crucial in generating customized reports where data formatting or labeling varies based on specific criteria.

6. **Streamlining Data Processing:** Helps in streamlining data processing, potentially avoiding the need for more complex joins or subqueries. Can lead to more efficient queries.

Syntax:

Syntax 1: Simple CASE statement

```
CASE value
   WHEN [compare_value] THEN result
   [WHEN [compare_value] THEN result ...]
   [ELSE result]
END
```

Syntax 2: Searched CASE statement

```
CASE
   WHEN [condition] THEN result
   [WHEN [condition] THEN result ...]
   [ELSE result]
END
```

Explanation

- The first syntax returns the result where `value = compare_value`.
- The second syntax returns the result for the first condition that is true.
- The list of corresponding SQL statements will execute when a search condition evaluates to true.
- The statement list in the `ELSE` part will execute when no search condition matches.

- If there is no matching value found in the `ELSE` part, `NULL` will be returned.
- Each statement list can contain one or more statements, and no empty statement list is allowed.

Conditions:

1. Simple CASE Statement:
   - When the condition is met (CASE 1), it returns the corresponding value.
   - When the condition is not met (CASE 4), it returns the value in the ELSE part.

46.

```
SELECT CASE 1
          WHEN 1 THEN 'this is case one'
          WHEN 2 THEN 'this is case two'
          ELSE 'this is not in the case'
       END as 'how to execute case statement';
```

47.

```
SELECT CASE 4
          WHEN 1 THEN 'this is case one'
          WHEN 2 THEN 'this is case two'
          ELSE 'this is not in the case'
       END as 'how to execute case statement';
```

2. CASE Statement with Matching Condition:
   - When the condition is met (CASE 2), it returns the corresponding value.

48.

```
SELECT CASE 2
          WHEN 1 THEN 'this is case one'
          WHEN 2 THEN 'this is case two'
          ELSE 'this is not in the case'
       END as 'how to execute case statement';
```

3. CASE Statement with Comparison Operators:
   - Using greater than and less than operators to evaluate conditions.

49.

```
  SELECT CASE  WHEN 2>3 THEN 'this is true'
            ELSE 'this is false' END;
```

50.

```
  SELECT CASE  WHEN 2<3 THEN 'this is true'
            ELSE 'this is false' END;
```

4. CASE Statement with No Matching Conditions:
   - If none of the conditions are satisfied, it returns NULL.

51.

```
SELECT CASE BINARY 'A' WHEN 'a' THEN 1
                  WHEN 'b' THEN 2 END;
```

Scenario Based Questions

Question 1: Simple CASE Statement

Consider a table named student_grades with columns student_id and grade. Write a SQL query using the CASE statement to display the grades of students based on the following conditions:

- If the grade is 90 or above, display 'A'.
- If the grade is between 80 and 89, display 'B'.
- If the grade is between 70 and 79, display 'C'.
- If the grade is below 70, display 'F'.

Question 2: CASE Statement with Matching Condition

Assume a table named employee_data with columns employee_id and salary. Write a SQL query using the CASE statement to categorize employees based on their salary:

- If the salary is greater than $100,000, categorize as 'High Income'.
- If the salary is between $50,000 and $100,000, categorize as 'Moderate Income'.
- If the salary is below $50,000, categorize as 'Low Income'.

Question 3: CASE Statement with Comparison Operators

Consider a table named product_inventory with columns product_id and quantity. Write a SQL query using the CASE statement to determine the availability of products:

- If the quantity is greater than 50, display 'In Stock'.
- If the quantity is between 10 and 50, display 'Low Stock'.
- If the quantity is 10 or below, display 'Out of Stock'.

Question 4: CASE Statement with No Matching Conditions

Assume a table named customer_orders with columns order_id and order_status. Write a SQL query using the CASE statement to categorize orders based on their status:

- If the order status is 'Shipped', display 'Order Shipped'.
- If the order status is 'Processing', display 'Order Processing'.
- If the order status is 'Cancelled', display 'Order Cancelled'.
- For any other order status, display 'Unknown Status'.

Question 5: CASE Statement with Multiple Conditions

Create a table named temperature_readings with columns location and temperature. Write a SQL query using the CASE statement to categorize temperature readings based on the following conditions:

- If the temperature is above 30 degrees Celsius, display 'Hot'.
- If the temperature is between 20 and 30 degrees Celsius, display 'Moderate'.
- If the temperature is below 20 degrees Celsius, display 'Cool'.

Solution

52.

```
CREATE TABLE student_grades (
    student_id INT,
    grade INT
);
```

53.

```
INSERT INTO student_grades (student_id, grade) VALUES
(1, 95),
(2, 85),
(3, 75),
(4, 60);
```

54.

```
CREATE TABLE employee_data (
    employee_id INT,
    salary DECIMAL(10, 2)
);
```

55.

```
INSERT INTO employee_data (employee_id, salary) VALUES
(101, 120000),
(102, 75000),
(103, 45000),
(104, 110000);
```

56.

```
CREATE TABLE product_inventory (
    product_id INT,
    quantity INT
);
```

57.

```
INSERT INTO product_inventory (product_id, quantity) VALUES
(1, 75),
(2, 20),
(3, 5),
(4, 100);
```

58.

```
CREATE TABLE customer_orders (
    order_id INT,
    order_status VARCHAR(20)
);
```

59.

```
INSERT INTO customer_orders (order_id, order_status) VALUES
(1, 'Shipped'),
(2, 'Processing'),
(3, 'Cancelled'),
(4, 'Pending');
```

60.

```
CREATE TABLE temperature_readings (
    location VARCHAR(50),
    temperature DECIMAL(5, 2)
);

61.

INSERT INTO temperature_readings (location, temperature) VALUES
('City A', 32.5),
('City B', 25.0),
('City C', 18.5),
('City D', 28.0);


-- Display grades based on conditions

62.

SELECT student_id, grade,
    CASE
        WHEN grade >= 90 THEN 'A'
        WHEN grade BETWEEN 80 AND 89 THEN 'B'
        WHEN grade BETWEEN 70 AND 79 THEN 'C'
        ELSE 'F'
    END AS grade_category
FROM student_grades;


-- Categorize employees based on salary

63.

SELECT employee_id, salary,
    CASE
        WHEN salary > 100000 THEN 'High Income'
        WHEN salary BETWEEN 50000 AND 100000 THEN 'Moderate Income'
        ELSE 'Low Income'
    END AS income_category
FROM employee_data;

-- Determine product availability based on quantity

64.

SELECT product_id, quantity,
    CASE
        WHEN quantity > 50 THEN 'In Stock'
        WHEN quantity BETWEEN 10 AND 50 THEN 'Low Stock'
        ELSE 'Out of Stock'
    END AS availability_status
FROM product_inventory;

-- Categorize orders based on status

65.
```

```
SELECT order_id, order_status,
   CASE
      WHEN order_status = 'Shipped' THEN 'Order Shipped'
      WHEN order_status = 'Processing' THEN 'Order Processing'
      WHEN order_status = 'Cancelled' THEN 'Order Cancelled'
      ELSE 'Unknown Status'
   END AS order_category
FROM customer_orders;

-- Categorize temperature readings based on conditions

66.

SELECT location, temperature,
   CASE
      WHEN temperature > 30 THEN 'Hot'
      WHEN temperature BETWEEN 20 AND 30 THEN 'Moderate'
      ELSE 'Cool'
   END AS temperature_category
FROM temperature_readings;
```

2. IF() Control Flow

The `IF()` function in MySQL is a powerful tool for introducing conditional logic into queries, and it can be used in various scenarios to handle different conditions and produce different results. Here are some common use cases and examples:

Conditional Logic in Queries • The `IF()` function allows you to implement conditional logic directly within a query. For example: SELECT IF(1 > 3, 'true', 'false');
Dynamic Column Selection • You can use `IF()` to dynamically select different columns based on specific conditions.
Data Validation • `IF()` can be used to validate data before inserting or updating a table, ensuring that only valid data is processed.
Contingency Plans • Setting up contingency plans in case certain conditions are not met.
Aggregate Functions with IF() • When used with aggregate functions like `SUM()` or `COUNT()`, `IF()` can selectively include or exclude certain records from the calculation.

Lab Performances

```
67.

CREATE TABLE books (
   book_id VARCHAR(10),
   book_name VARCHAR(50),
   isbn_no VARCHAR(11),
   cate_id VARCHAR(10),
   aut_id VARCHAR(10),
   pub_id VARCHAR(10),
```

```
    dt_of_pub DATE,
    pub_lang VARCHAR(20),
    no_page INT,
    book_price DECIMAL(8, 2)
);
```

68.

```
INSERT INTO books VALUES
('BK001', 'Introduction to Electrodynamics', '0000979001', 'CA001', 'AUT001', 'P003', '2001-05-08', 'English', 201, 85.00),
('BK002', 'Understanding of Steel Construction', '0000979002', 'CA002', 'AUT002', 'P001', '2003-07-15', 'English', 300, 105.50),
('BK003', 'Guide to Networking', '0000979003', 'CA003', 'AUT003', 'P002', '2002-09-10', 'Hindi', 510, 200.00),
('BK004', 'Transfer of Heat and Mass', '0000979004', 'CA002', 'AUT004', 'P004', '2004-02-16', 'English', 600, 250.00),
('BK005', 'Conceptual Physics', '0000979005', 'CA001', 'AUT005', 'P006', '2003-07-16', NULL, 345, 145.00),
('BK006', 'Fundamentals of Heat', '0000979006', 'CA001', 'AUT006', 'P005', '2003-08-10', 'German', 247, 112.00),
('BK007', 'Advanced 3d Graphics', '0000979007', 'CA003', 'AUT007', 'P002', '2004-02-16', 'Hindi', 165, 56.00),
('BK008', 'Human Anatomy', '0000979008', 'CA005', 'AUT008', 'P006', '2001-05-17', 'German', 88, 50.50),
('BK009', 'Mental Health Nursing', '0000979009', 'CA005', 'AUT009', 'P007', '2004-02-10', 'English', 350, 145.00),
('BK010', 'Fundamentals of Thermodynamics', '0000979010', 'CA002', 'AUT010', 'P007', '2002-10-14', 'English', 400, 225.00),
('BK011', 'The Experimental Analysis of Cat', '0000979011', 'CA004', 'AUT011', 'P005', '2007-06-09', 'French', 225, 95.00),
('BK012', 'The Nature of World', '0000979012', 'CA004', 'AUT005', 'P008', '2005-12-20', 'English', 350, 88.00),
('BK013', 'Environment a Sustainable Future', '0000979013', 'CA004', 'AUT012', 'P001', '2003-10-27', 'German', 165, 100.00),
('BK014', 'Concepts in Health', '0000979014', 'CA005', 'AUT013', 'P004', '2001-08-25', NULL, 320, 180.00),
('BK015', 'Anatomy & Physiology', '0000979015', 'CA005', 'AUT014', 'P008', '2000-10-10', 'Hindi', 225, 135.00),
('BK016', 'Networks and Telecommunications', '00009790_16', 'CA003', 'AUT015', 'P003', '2002-01-01', 'French', 95, 45.00);
```

69.

```
select * from books;
```

70.

```
SELECT book_name,
    IF(pub_lang="English", "Engllish Book", "Other Lnaguage")
    AS Language
    FROM books;
```

71.

```
SELECT book_name,isbn_no,
IF((SELECT COUNT(*) FROM books WHERE pub_lang='English')>
(SELECT COUNT(*) FROM books WHERE pub_lang<>'English'),
(CONCAT("Pages: ",no_page)),(CONCAT("Price: ",book_price)))
AS "Page / Price"
FROM books;
```

72.

```
SELECT book_id, book_name,
   IF(pub_lang IS NULL,'N/A',pub_lang) AS "Pub. Language"
   FROM books;
```

73.

```
SELECT book_id, book_name, pub_lang
   FROM books;
```

74.

```
CREATE TABLE purchase (
   invoice_no VARCHAR(10),
   invoice_dt DATE,
   ord_no VARCHAR(20),
   ord_date DATE,
   receive_dt DATE,
   book_id VARCHAR(10),
   book_name VARCHAR(50),
   pub_lang VARCHAR(20),
   cate_id VARCHAR(10),
   receive_qty INT,
   purch_price DECIMAL(8, 2),
   total_cost DECIMAL(10, 2)
);
```

75.

```
INSERT INTO purchase VALUES
('INV0001', '2008-07-15', 'ORD/08-09/0001', '2008-07-06', '2008-07-19', 'BK001', 'Introduction to
Electrodynamics', 'English', 'CA001', 15, 75.00, 1125.00),
('INV0002', '2008-08-25', 'ORD/08-09/0002', '2008-08-09', '2008-08-28', 'BK004', 'Transfer  of
Heat and Mass', 'English', 'CA002', 8, 55.00, 440.00),
('INV0003', '2008-09-20', 'ORD/08-09/0003', '2008-09-15', '2008-09-23', 'BK005', 'Conceptual
Physics', NULL, 'CA001', 20, 20.00, 400.00),
('INV0004', '2007-08-30', 'ORD/07-08/0005', '2007-08-22', '2007-08-30', 'BK004', 'Transfer  of
Heat and Mass', 'English', 'CA002', 15, 35.00, 525.00),
('INV0005', '2007-07-28', 'ORD/07-08/0004', '2007-06-25', '2007-07-30', 'BK001', 'Introduction to
Electrodynamics', 'English', 'CA001', 8, 25.00, 200.00),
('INV0006', '2007-09-24', 'ORD/07-08/0007', '2007-09-20', '2007-09-30', 'BK003', 'Guide to
Networking', 'Hindi', 'CA003', 20, 45.00, 900.00);
```

76.

```
select * from purchase;
```

77.

```
SELECT SUM(IF(pub_lang = 'English',1,0))   AS English,
      SUM(IF(pub_lang <> 'English',1,0)) AS "Non English"
FROM purchase;
```

78.

```
CREATE TABLE publishers (
   pub_id VARCHAR(10),
   pub_name VARCHAR(50),
   pub_city VARCHAR(30),
   country VARCHAR(30),
   country_office VARCHAR(30),
   no_of_branch INT,
   estd DATE
);
```

79.

```
INSERT INTO publishers VALUES
('P001', 'Jex Max Publication', 'New York', 'USA', 'New York', 15, '1969-12-25'),
('P002', 'BPP Publication', 'Mumbai', 'India', 'New Delhi', 10, '1985-10-01'),
('P003', 'New Harrold Publication', 'Adelaide', 'Australia', 'Sydney', 6, '1975-09-05'),
('P004', 'Ultra Press Inc.', 'London', 'UK', 'London', 8, '1948-07-10'),
('P005', 'Mountain Publication', 'Houstan', 'USA', 'Sun Diego', 25, '1975-01-01'),
('P006', 'Summer Night Publication', 'New York', 'USA', 'Atlanta', 10, '1990-12-10'),
('P007', 'Pieterson Grp. of Publishers', 'Cambridge', 'UK', 'London', 6, '1950-07-15'),
('P008', 'Novel Publisher Ltd.', 'New Delhi', 'India', 'Bangalore', 10, '2000-01-01');
```

80.

```
select * from publishers;
```

81.

```
SELECT COUNT(IF(country = 'USA',1,NULL))  USA,
      COUNT(IF(country = 'UK',1,NULL))  UK,
      COUNT(IF(country = 'India',1,NULL))  India,
      COUNT(IF(country = 'Australia',1,NULL))  Australia
FROM publishers;
```

Another way to achieve the similar result you can use the GROUP BY clause and the COUNT function without using the IF function, the display report is quite different.

82.

```
SELECT country, COUNT(country)
   FROM publishers
   GROUP BY country;
```

EXPERIMENT-6

Lab 6: Use of different SQL clauses and join

Objective: To understand the use of group by and having clause and execute the SQL commands using JOIN

Understanding JOINs in MySQL

- A join enables you to retrieve records from two (or more) logically related tables in a single result set.
- JOIN clauses are used to return the rows of two or more queries using two or more tables that share a meaningful relationship based on a common set of values.
- These values are usually the same column name and datatype that appear in both the participating tables being joined. These columns, or possibly a single column from each table, are called the join key or common key.
- Mostly but not all of the time, the join key is the primary key of one table and a foreign key in another table. The join can be performed as long as the data in the columns are matching.
- It can be difficult when the join involves more than two tables. It is a good practice to think of the query as a series of two table joins when the involvement of three or more tables in joins.

MySQL JOIN

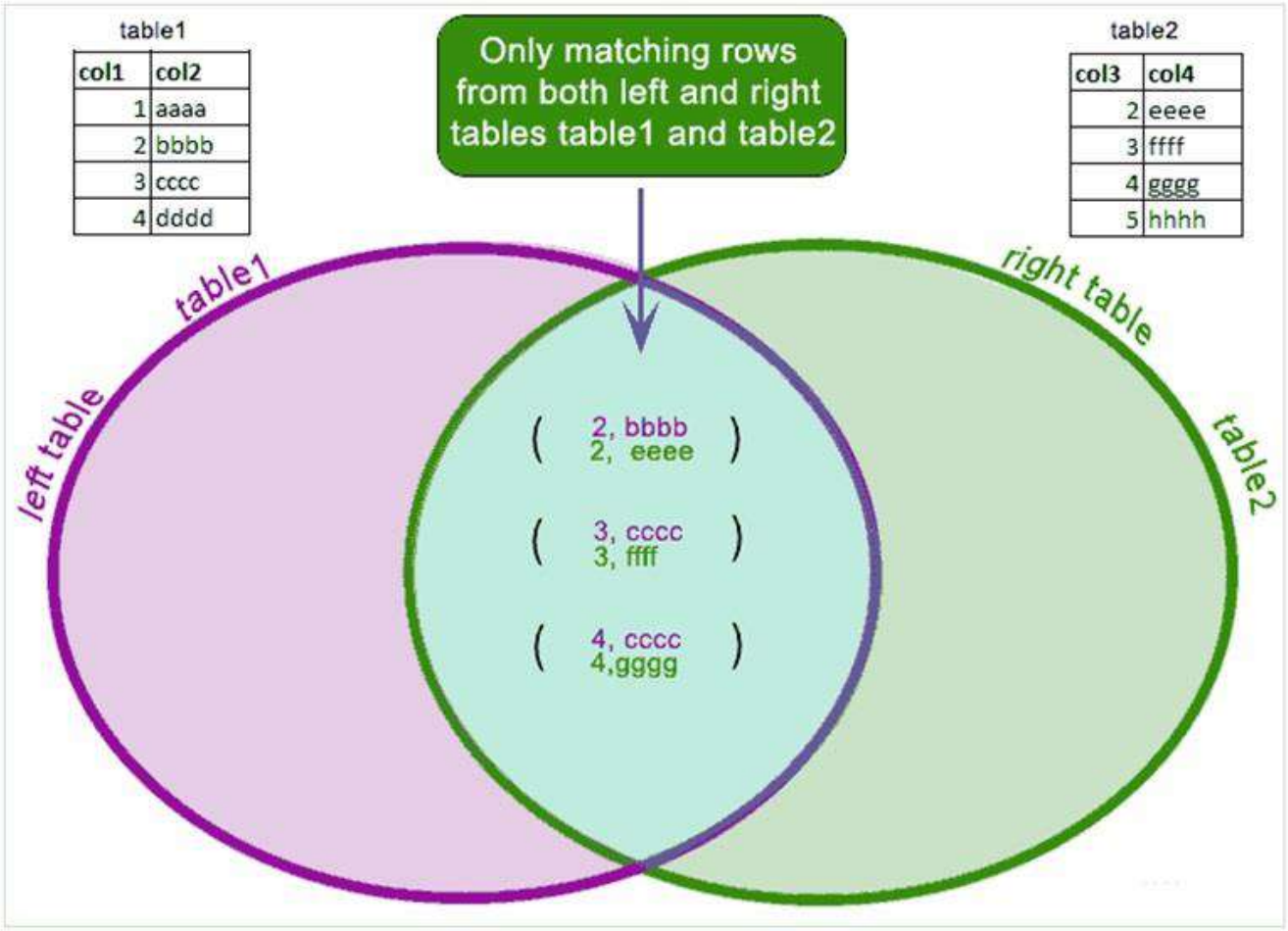
MySQL supports the following JOIN syntaxes for the table_references (A table reference is also known as a join expression.) part of SELECT statements and multiple-table UPDATE and DELETE statements :

Types of MySQL Joins

1. **INNER JOIN**
2. **LEFT JOIN**
3. **RIGHT JOIN**
4. **STRAIGHT JOIN**
5. **CROSS JOIN**
6. **NATURAL JOIN**

1. INNER JOIN

In MySQL the INNER JOIN selects all rows from both participating tables to appear in the result if and only if both tables meet the conditions specified in the ON clause. JOIN, CROSS JOIN, and INNER JOIN are syntactic equivalents. In standard SQL, they are not equivalent. INNER JOIN is used with an ON clause, CROSS JOIN is used otherwise.



INNER JOIN

A	M
1	m
2	n
4	o

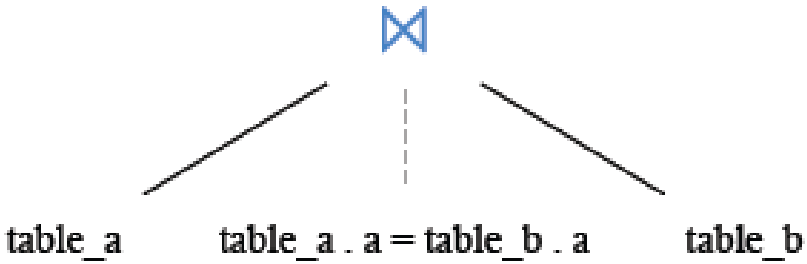
table_A

A	N
2	p
3	q
5	r

table_B

SELECT * FROM table_A
INNER JOIN table_B
ON table_A.A=table_B.A;
Output 1

$$table\_a \bowtie_{table\_a\,.\,a\,=\,table\_b\,.\,a} table\_b$$



book_mast

book_id	book_name	isbn_no	cate_id	aut_id	pub_id	dt_of_pub	pub_lang	no_page	book_price
BK001	Introduction to Electrodynamics	0000979001	CA001	AUT001	P003	2001-05-08	English	201	85.00
BK002	Understanding of Steel Construction	0000979002	CA002	AUT002	P001	2003-07-15	English	300	105.50
BK003	Guide to Networking	0000979003	CA003	AUT003	P002	2002-09-10	Hindi	510	200.00
BK004	Transfer of Heat and Mass	0000979004	CA002	AUT004	P004	2004-02-16	English	600	250.00
BK005	Conceptual Physics	0000979005	CA001	AUT005	P006	2003-07-16	NULL	345	145.00
BK006	Fundamentals of Heat	0000979006	CA001	AUT006	P005	2003-08-10	German	247	112.00
BK007	Advanced 3d Graphics	0000979007	CA003	AUT007	P002	2004-02-16	Hindi	165	56.00
BK008	Human Anatomy	0000979008	CA005	AUT008	P006	2001-05-17	German	88	50.50
BK009	Mental Health Nursing	0000979009	CA005	AUT009	P007	2004-02-10	English	350	145.00
BK010	Fundamentals of Thermodynamics	0000979010	CA002	AUT010	P007	2002-10-14	English	400	225.00
BK011	The Experimental Analysis of Cat	0000979011	CA004	AUT011	P005	2007-06-09	French	225	95.00
BK012	The Nature of World	0000979012	CA004	AUT005	P008	2005-12-20	English	350	88.00
BK013	Environment a Sustainable Future	0000979013	CA004	AUT012	P001	2003-10-27	German	165	100.00
BK014	Concepts in Health	0000979014	CA005	AUT013	P004	2001-08-25	NULL	320	180.00
BK015	Anatomy & Physiology	0000979015	CA005	AUT014	P008	2000-10-10	Hindi	225	135.00
BK016	Networks and Telecommunications	00009790_16	CA003	AUT015	P003	2002-01-01	French	95	45.00

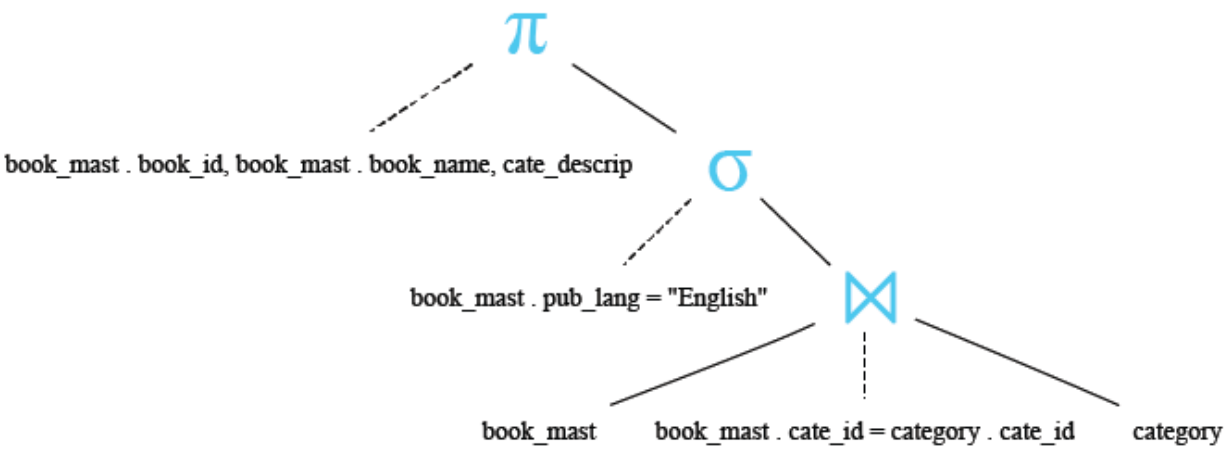
Category

cate_id	cate_descrip
CA001	Science
CA002	Technology
CA003	Computers
CA004	Nature
CA005	Medical

```
SELECT book_mast.book_id,book_mast.book_name,cate_descrip
FROM book_mast
INNER JOIN category
ON book_mast.cate_id=category.cate_id
WHERE book_mast.pub_lang="English";
```

Output 2

$$\pi_{book\_mast\,.\,book\_id\,,\,book\_mast\,.\,book\_name\,,\,cate\_descrip}\left(\sigma_{book\_mast\,.\,pub\_lang\,=\,\text{“English”}}(book\_mast \bowtie_{book\_mast\,.\,cate\_id\,=\,category\,.\,cate\_id} category)\right)$$

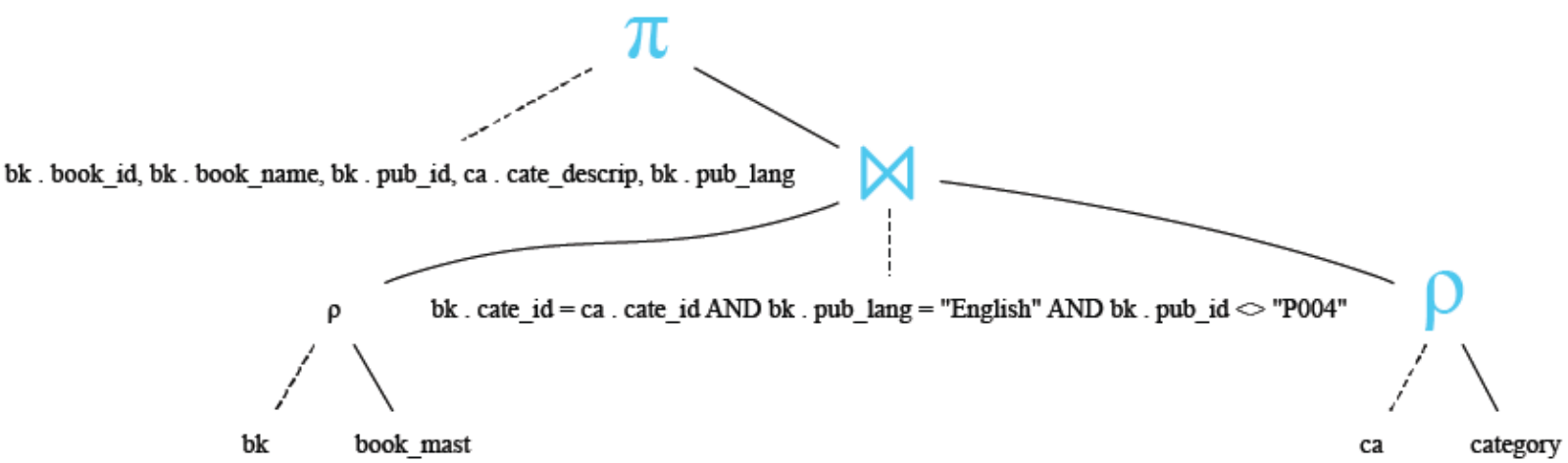


INNER JOIN with alias

```
SELECT bk.book_id,bk.book_name,bk.pub_id,ca.cate_descrip,bk.pub_lang
FROM book_mast AS bk
INNER JOIN category AS ca  ON bk.cate_id=ca.cate_id  AND
bk.pub_lang="English" AND bk.pub_id<>'P004';
```

Output 3

$$\pi_{bk.book\_id,\ bk.book\_name,\ bk.pub\_id,\ ca.cate\_descrip,\ bk.pub\_lang}\ (\rho_{bk}\ book\_mast \bowtie_{bk.cate\_id = ca.cate\_id\ AND\ bk.pub\_lang = \text{“English”}\ AND\ bk.pub\_id <> \text{“P004”}}\ \rho_{ca}\ category)$$



INNER JOIN using three tables

table - doctors

docid	dname
1	A.VARMA
2	D.GOMES

table - specialize

spid	desc	docid
1	special1	1
2	special2	2

table - timeschedule

tid	tday	sit_time	docid
1	MON	17:00:00	1
2	WED	08:00:00	1
3	TUE	16:00:00	2
4	FRI	09:00:00	2

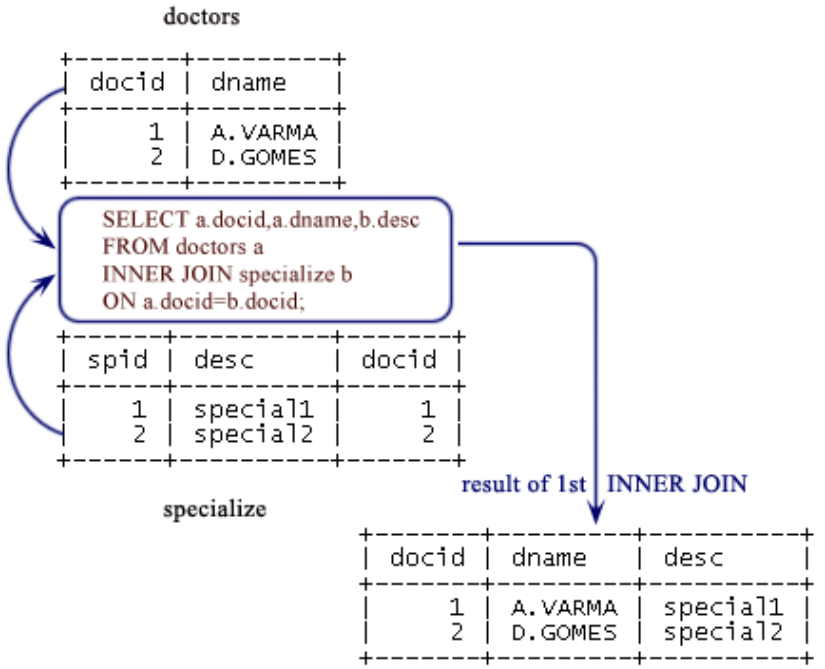
```
SELECT a.docid,a.dname,
      b.desc,c.tday,c.sit_time
  FROM doctors a
    INNER JOIN specialize b
     ON a.docid=b.docid
    INNER JOIN timeschedule c
     ON a.docid=c.docid
       WHERE a.docid=1 AND c.tday='WED';
```

Output 4

Explanation :

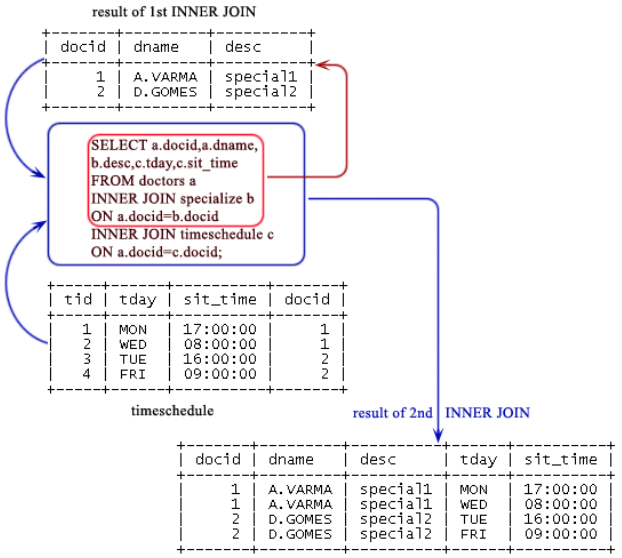
Step-1

SELECT a.docid,a.dname,b.desc
FROM doctors a
INNER JOIN specialize b
ON a.docid=b.docid;



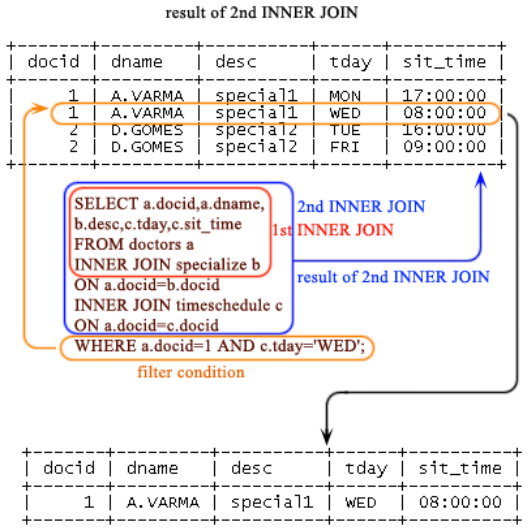
Step-2

```
SELECT a.docid,a.dname,
b.desc,c.tday,c.sit_time
FROM doctors a
INNER JOIN specialize b
ON a.docid=b.docid
INNER JOIN timeschedule c
ON a.docid=c.docid;
```



Step-3

```
SELECT a.docid,a.dname,
b.desc,c.tday,c.sit_time
FROM doctors a
INNER JOIN specialize b
ON a.docid=b.docid
INNER JOIN timeschedule c
ON a.docid=c.docid
WHERE a.docid=1 AND c.tday='WED';
```



2. LEFT JOIN

The MySQL LEFT JOIN joins two tables and fetches rows based on a condition, which is matching in both the tables and the unmatched rows will also be available from the table written before the JOIN clause.

```
SELECT * FROM table_A
LEFT JOIN table_B
ON table_A.A=table_B.A;
```

Output 5

```
SELECT bk1.book_name,bk1.isbn_no,bk1.book_price,bk1.pub_lang
FROM  book_mast bk1
LEFT JOIN book_mast bk2 ON bk1.book_price<bk2.book_price
WHERE bk2.pub_lang='German';
```

Output 6

3. RIGHT JOIN

The RIGHT JOIN is such a join which specifies that all records be fetched from the table on the right side of the join statement, even if the table on the left has no matching record. In this case, the columns from the left table return NULL values.

```
SELECT * FROM table_A
RIGHT JOIN table_B
ON table_A.A=table_B.A;
```

Output 7

table111

id	aval1
1	405
2	401
3	200
4	400

table112

id	bval1	bval2
701	405	16
704	409	14
706	403	13
709	401	12

table113

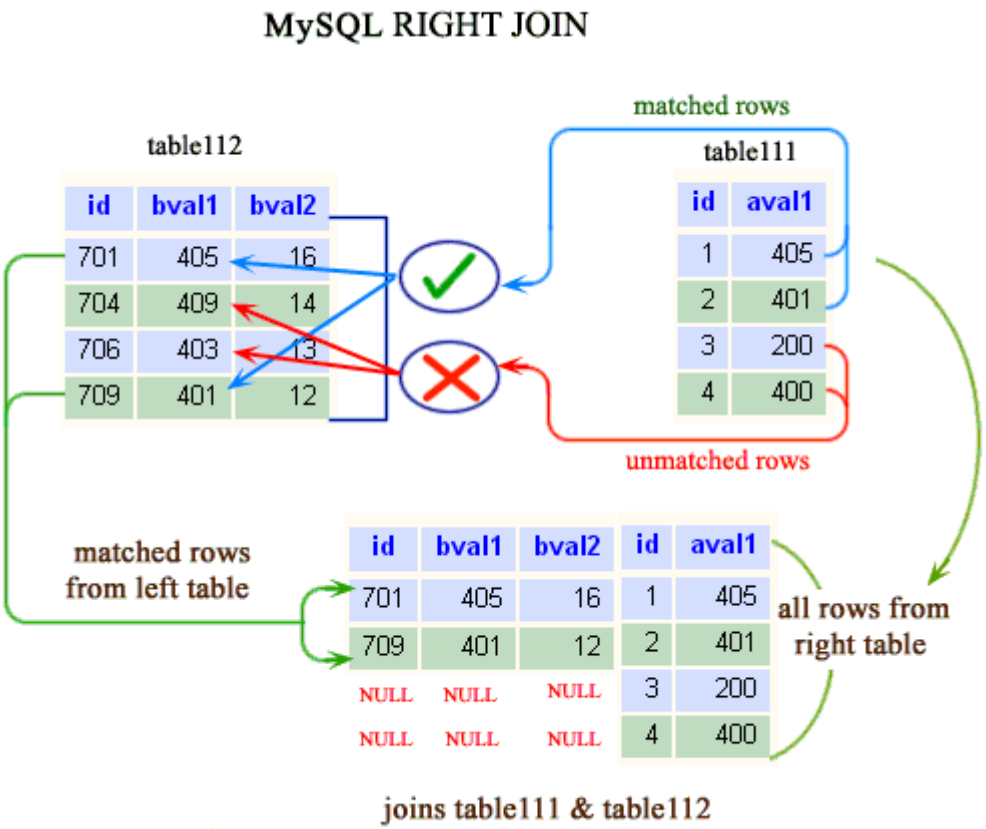
id	cval1
3	17
2	12
5	15
1	16

```
SELECT table112.id,table112.bval1,table112.bval2,
table111.id,table111.aval1
FROM table112
RIGHT JOIN table111
ON table112.bval1=table111.aval1;
```

Output 8

```
SELECT table112.id,table112.bval1,table112.bval2,
table111.id,table111.aval1
FROM table112
RIGHT OUTER JOIN table111
ON table112.bval1=table111.aval1;
```

Output 9




```
SELECT table113.id,table113.cval1,
table111.id,table111.aval1
FROM table113
RIGHT OUTER JOIN table111
USING(id);
```

Output 10

4. STRAIGHT JOIN

A STRAIGHT_JOIN is such a join which scans and combines matching rows (if specified any condition) which are stored in associated tables otherwise it behaves like an INNER JOIN or JOIN of without any condition.

```
SELECT * FROM table_A
STRAIGHT JOIN table_B;
```

Output 11

```
SELECT table112.id,table112.bval1,table112.bval2,
table111.id,table111.aval1
FROM table112
STRAIGHT_JOIN table111;
```

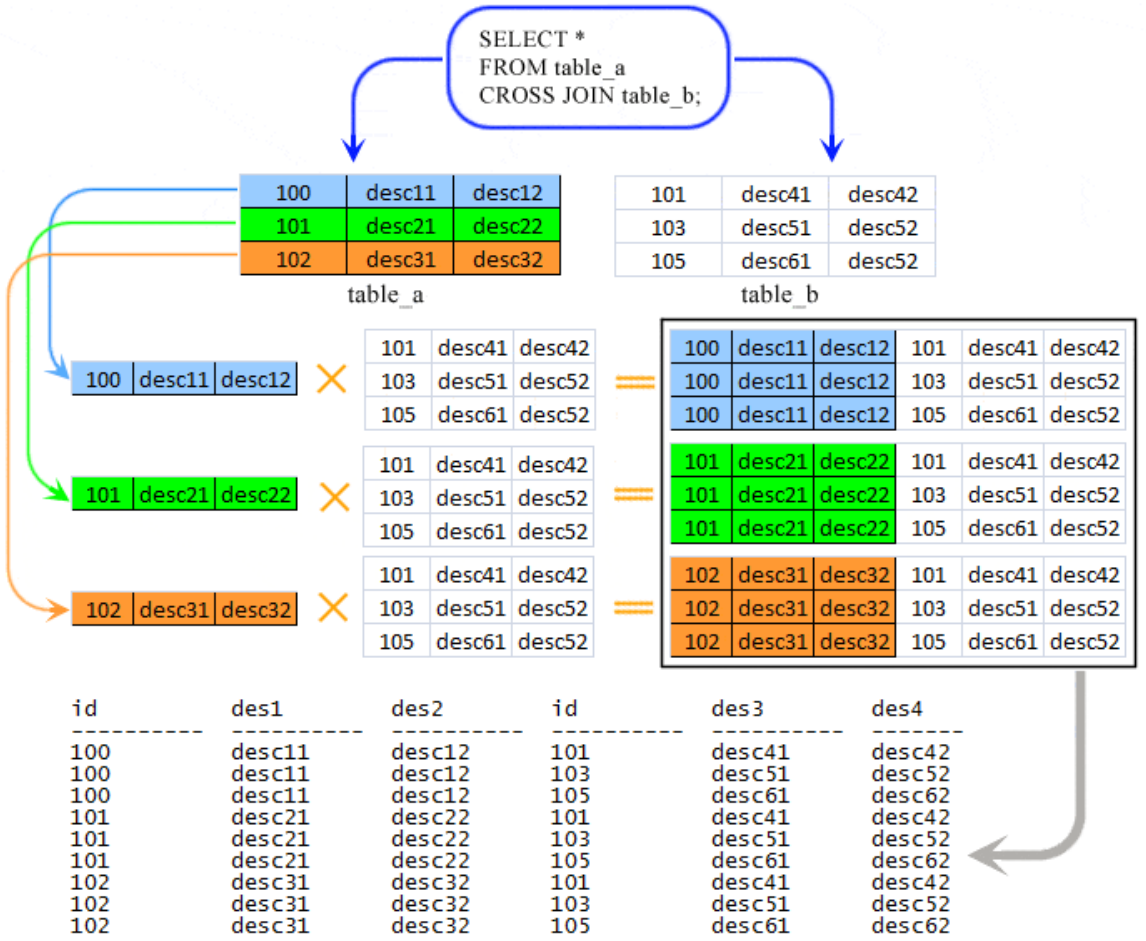
Output 12

```
SELECT table113.id,table113.cval1,
table111.id,table111.aval1
FROM table113
STRAIGHT_JOIN table111
ON table113.id=table111.id;
```

Output 13

5. CROSS JOIN

A CROSS JOIN is such a join which specifies the complete cross product of two tables. For each record in the first table, all the records in the second table are joined, creating a potentially huge result set. This command has the same effect as leaving off the join condition, and its result set is also known as a Cartesian product.



```
SELECT * FROM table_A
CROSS JOIN table_B;
```
Output 14

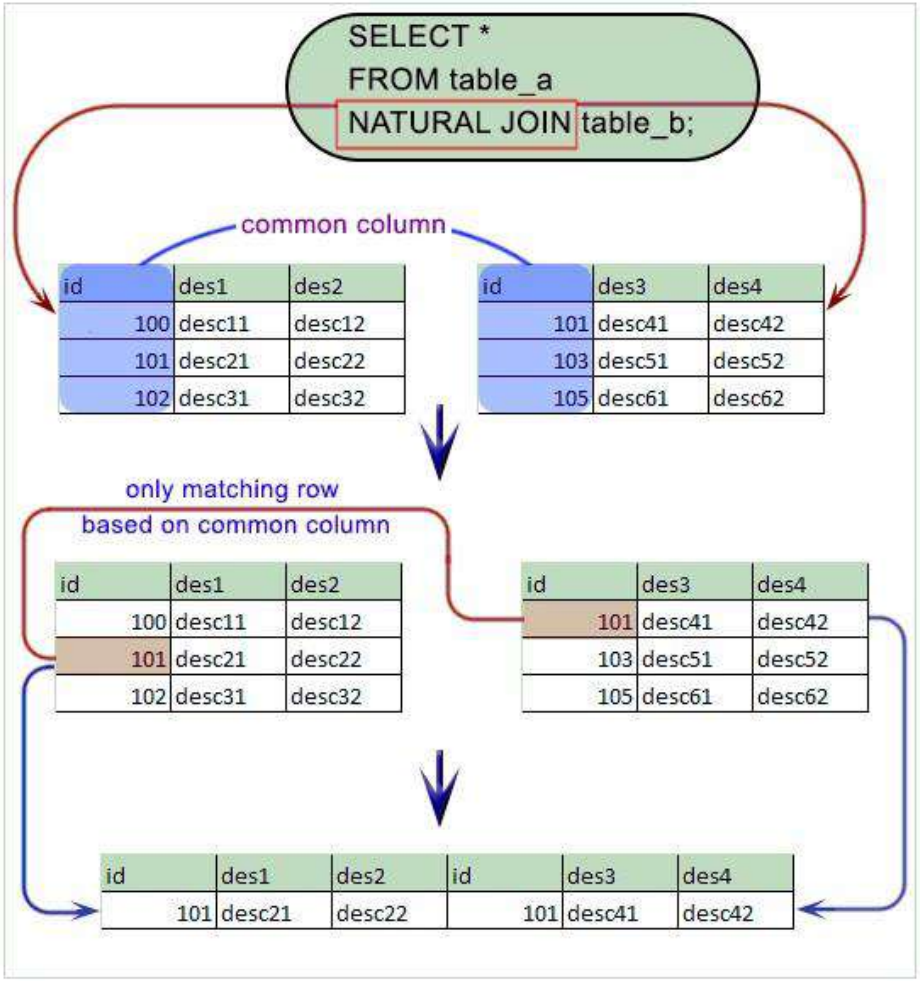
```
SELECT table112.id,table112.bval1,table112.bval2,
table111.id,table111.aval1
FROM table112
CROSS JOIN table111;
```
Output 15

```
SELECT *
FROM table111
LEFT JOIN(table112 CROSS JOIN table113)
ON table111.id=table113.id;
```
Output 16

```
SELECT table111.*,table113.*
FROM table111
CROSS JOIN table113
WHERE table111.id=table113.id;
```
Output 17

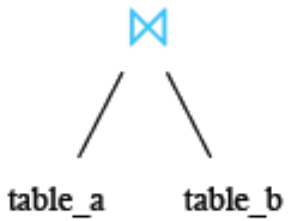
6. NATURAL JOIN

A NATURAL JOIN is such a join that performs the same task as an INNER or LEFT JOIN, in which the ON or USING clause refers to all columns that the tables to be joined have in common.



SELECT * FROM table_A
NATURAL JOIN table_B;
Output 18

$$table\_a \bowtie table\_b$$



NATURAL JOIN

```
SELECT id,aval1,cval1
FROM table111
NATURAL JOIN table113;
```

Output 19

INNER JOIN using ON clause

```
SELECT table1111.id,table111.aval1,table113.cval1
FROM table111
INNER JOIN table113
ON table111.id=table113.id;
```

Output 20

NATURAL JOIN with WHERE clause

```
SELECT id,aval1,cval1
FROM table111
NATURAL JOIN table113
WHERE table111.aval1>200;
```

Output 21

NATURAL JOIN using three tables

```
SELECT id,aval1,cval1
FROM table111
NATURAL JOIN table113
natural join table114
WHERE table111.aval1>200;
```

Output 22

Lab 7: MySQL Views

In the context of relational databases, a view is a virtual table that is based on the result of a SELECT query. A view does not store the data itself but is a saved query that can be treated as a table for querying purposes. It provides a way to simplify complex queries, encapsulate logic, and restrict access to certain columns or rows.

Some key points about views:

1. **Data Abstraction:** Views abstract the underlying complexity of the database schema by allowing users to interact with a simplified representation of the data.

2. **Security:** Views can be used to restrict access to specific columns or rows of a table, providing a layer of security. Users can be granted permission to access the view without exposing the details of the underlying tables.

3. **Simplicity and Reusability:** Views can be used to encapsulate complex SQL queries, making them easier to understand and maintain. Additionally, views can be reused in other queries or views, promoting code reusability.

4. **Dynamic Data Presentation:** Since views are dynamically generated based on the underlying tables or other views, changes in the data of the underlying tables are automatically reflected in the view.

Syntax:

```
CREATE VIEW my_view AS
SELECT column1, column2
FROM my_table
WHERE column3 = 'some_condition';
```

After creating this view, you can query it as if it were a table:

Syntax:

```
SELECT * FROM my_view;
```

It's important to note that the specific syntax for creating views may vary depending on the database management system (e.g., MySQL, PostgreSQL, SQL Server) you are using.

Lab Performance Commands

Run the code and explain every Commands

1. To check the version of the database server:

```
SELECT VERSION();
```

2. To show the privileges of the current user:

```
SHOW PRIVILEGES;
```

3. To show the list of databases:

```
SHOW DATABASES;
```

4. **Create View**

```
CREATE TABLE user (
    userid VARCHAR(50) PRIMARY KEY,
    password VARCHAR(50),
    name VARCHAR(50)
);
```

5.

```
INSERT INTO user (userid, password, name)
VALUES
    ('scott123', '123@sco', 'Scott'),
    ('ferp6734', 'dloeiu@&3', 'Palash'),
    ('diana094', 'ku$j@23', 'Diana');
```

6.

```
SELECT * FROM user;
```

7.

```
CREATE VIEW my_v2 AS SELECT * FROM user;
```

8.

```
SELECT * FROM my_v2;
```

9.

```
CREATE VIEW my_v3 AS SELECT userid AS User_ID, password AS Password, name AS Name  FROM user;
```

10.

```
SELECT * FROM my_v3;
```

11. **CREATE VIEW with WHERE**

```
CREATE TABLE author (
    aut_id VARCHAR(10) PRIMARY KEY,
    aut_name VARCHAR(50),
    country VARCHAR(50),
    home_city VARCHAR(50)
);
```

12.

```
INSERT INTO author (aut_id, aut_name, country, home_city)
VALUES
    ('AUT001', 'William Norton', 'UK', 'Cambridge'),
    ('AUT002', 'William Maugham', 'Canada', 'Toronto'),
    ('AUT003', 'William Anthony', 'UK', 'Leeds'),
    ('AUT004', 'S.B.Swaminathan', 'India', 'Bangalore'),
    ('AUT005', 'Thomas Morgan', 'Germany', 'Arnsberg'),
    ('AUT006', 'Thomas Merton', 'USA', 'New York'),
    ('AUT007', 'Piers Gibson', 'UK', 'London'),
```

```
    ('AUT008', 'Nikolai Dewey', 'USA', 'Atlanta'),
    ('AUT009', 'Marquis de Ellis', 'Brazil', 'Rio De Janeiro'),
    ('AUT010', 'Joseph Milton', 'USA', 'Houston'),
    ('AUT011', 'John Betjeman Hunter', 'Australia', 'Sydney'),
    ('AUT012', 'Evan Hayek', 'Canada', 'Vancouver'),
    ('AUT013', 'E. Howard', 'Australia', 'Adelaide'),
    ('AUT014', 'C. J. Wilde', 'UK', 'London'),
    ('AUT015', 'Butler Andre', 'USA', 'Florida');
```

13.

```
CREATE VIEW view_author
AS SELECT *
FROM author
WHERE country='USA';
```

14.

```
SELECT * FROM view_author;
```

15. **CREATE VIEW with AND and OR**

```
CREATE TABLE publisher (
    pub_id VARCHAR(10) PRIMARY KEY,
    pub_name VARCHAR(50),
    pub_city VARCHAR(50),
    country VARCHAR(50),
    country_office VARCHAR(50),
    no_of_branch INT,
    estd DATE
);
```

16.

```
INSERT INTO publisher (pub_id, pub_name, pub_city, country, country_office,
no_of_branch, estd)
VALUES
    ('P001', 'Jex Max Publication', 'New York', 'USA', 'New York', 15, '1969-12-25'),
```

```
('P002', 'BPP Publication', 'Mumbai', 'India', 'New Delhi', 10, '1985-10-01'),
('P003', 'New Harrold Publication', 'Adelaide', 'Australia', 'Sydney', 6, '1975-09-05'),
('P004', 'Ultra Press Inc.', 'London', 'UK', 'London', 8, '1948-07-10'),
('P005', 'Mountain Publication', 'Houstan', 'USA', 'Sun Diego', 25, '1975-01-01'),
('P006', 'Summer Night Publication', 'New York', 'USA', 'Atlanta', 10, '1990-12-10'),
('P007', 'Pieterson Grp. of Publishers', 'Cambridge', 'UK', 'London', 6, '1950-07-15'),
('P008', 'Novel Publisher Ltd.', 'New Delhi', 'India', 'Bangalore', 10, '2000-01-01');
```

17.

```
CREATE VIEW view_publisher
AS SELECT pub_name,pub_city,country
FROM publisher
WHERE (country='USA' AND pub_city='New York')
OR     (country='India' AND pub_city='Mumbai');
```

18.

```
SELECT * FROM view_publisher;
```

19. **CREATE VIEW with GROUP BY**

```
CREATE TABLE book_mast (
   book_id VARCHAR(10) PRIMARY KEY,
   book_name VARCHAR(100),
   isbn_no VARCHAR(20),
   cate_id VARCHAR(10),
   aut_id VARCHAR(10),
   pub_id VARCHAR(10),
   dt_of_pub DATE,
   pub_lang VARCHAR(50),
   no_page INT,
   book_price DECIMAL(10, 2)
);
```

20.

INSERT INTO book_mast (book_id, book_name, isbn_no, cate_id, aut_id, pub_id, dt_of_pub, pub_lang, no_page, book_price)
VALUES
('BK001', 'Introduction to Electrodynamics', '0000979001', 'CA001', 'AUT001', 'P003', '2001-05-08', 'English', 201, 85.00),
('BK002', 'Understanding of Steel Construction', '0000979002', 'CA002', 'AUT002', 'P001', '2003-07-15', 'English', 300, 105.50),
('BK003', 'Guide to Networking', '0000979003', 'CA003', 'AUT003', 'P002', '2002-09-10', 'Hindi', 510, 200.00),
('BK004', 'Transfer of Heat and Mass', '0000979004', 'CA002', 'AUT004', 'P004', '2004-02-16', 'English', 600, 250.00),
('BK005', 'Conceptual Physics', '0000979005', 'CA001', 'AUT005', 'P006', '2003-07-16', NULL, 345, 145.00),
('BK006', 'Fundamentals of Heat', '0000979006', 'CA001', 'AUT006', 'P005', '2003-08-10', 'German', 247, 112.00),
('BK007', 'Advanced 3d Graphics', '0000979007', 'CA003', 'AUT007', 'P002', '2004-02-16', 'Hindi', 165, 56.00),
('BK008', 'Human Anatomy', '0000979008', 'CA005', 'AUT008', 'P006', '2001-05-17', 'German', 88, 50.50),
('BK009', 'Mental Health Nursing', '0000979009', 'CA005', 'AUT009', 'P007', '2004-02-10', 'English', 350, 145.00),
('BK010', 'Fundamentals of Thermodynamics', '0000979010', 'CA002', 'AUT010', 'P007', '2002-10-14', 'English', 400, 225.00),
('BK011', 'The Experimental Analysis of Cat', '0000979011', 'CA004', 'AUT011', 'P005', '2007-06-09', 'French', 225, 95.00),
('BK012', 'The Nature of World', '0000979012', 'CA004', 'AUT005', 'P008', '2005-12-20', 'English', 350, 88.00),
('BK013', 'Environment a Sustainable Future', '0000979013', 'CA004', 'AUT012', 'P001', '2003-10-27', 'German', 165, 100.00),
('BK014', 'Concepts in Health', '0000979014', 'CA005', 'AUT013', 'P004', '2001-08-25', NULL, 320, 180.00),
('BK015', 'Anatomy & Physiology', '0000979015', 'CA005', 'AUT014', 'P008', '2000-10-10', 'Hindi', 225, 135.00),
('BK016', 'Networks and Telecommunications', '00009790_16', 'CA003', 'AUT015', 'P003', '2002-01-01', 'French', 95, 45.00);

21.

```
CREATE VIEW view_bookmast
AS SELECT pub_lang, count(*)
FROM book_mast
GROUP BY pub_lang;
```

22.

```
SELECT * FROM view_bookmast;
```

23. **CREATE VIEW with ORDER BY**

```
CREATE VIEW view_bookmast1
AS SELECT pub_lang,count(*)
FROM book_mast1
GROUP BY pub_lang ORDER BY pub_lang;
```

24.

```
SELECT * FROM view_bookmast1;
```

25. **CREATE VIEW with BETWEEN and IN**

```
CREATE VIEW view_bookmast2
AS SELECT *
FROM book_mast1
WHERE book_name BETWEEN 'A' AND 'G'
AND no_page IN(165,250,350,400,510);
```

26.

```
SELECT * FROM view_bookmast2;
```

27. **CREATE VIEW with LIKE**

```
CREATE VIEW view_author1
AS SELECT *
FROM author1
```

```
WHERE aut_name
NOT LIKE 'T%' AND aut_name NOT LIKE 'W%';
```

28.

```
SELECT * FROM view_author1;
```

29. **CREATE VIEW using subqueries**

```
CREATE TABLE purchase (
   invoice_no VARCHAR(10) PRIMARY KEY,
   invoice_dt DATE,
   ord_no VARCHAR(20),
   ord_date DATE,
   receive_dt DATE,
   book_id VARCHAR(10),
   book_name VARCHAR(100),
   pub_lang VARCHAR(50),
   cate_id VARCHAR(10),
   receive_qty INT,
   purch_price DECIMAL(10, 2),
   total_cost DECIMAL(10, 2)
);
```

30.

```
INSERT INTO purchase (invoice_no, invoice_dt, ord_no, ord_date, receive_dt, book_id, book_name, pub_lang, cate_id, receive_qty, purch_price, total_cost)
VALUES
   ('INV0001', '2008-07-15', 'ORD/08-09/0001', '2008-07-06', '2008-07-19', 'BK001', 'Introduction to Electrodynamics', 'English', 'CA001', 15, 75.00, 1125.00),
   ('INV0002', '2008-08-25', 'ORD/08-09/0002', '2008-08-09', '2008-08-28', 'BK004', 'Transfer of Heat and Mass', 'English', 'CA002', 8, 55.00, 440.00),
   ('INV0003', '2008-09-20', 'ORD/08-09/0003', '2008-09-15', '2008-09-23', 'BK005', 'Conceptual Physics', NULL, 'CA001', 20, 20.00, 400.00),
   ('INV0004', '2007-08-30', 'ORD/07-08/0005', '2007-08-22', '2007-08-30', 'BK004', 'Transfer of Heat and Mass', 'English', 'CA002', 15, 35.00, 525.00),
```

```
    ('INV0005',  '2007-07-28',  'ORD/07-08/0004',  '2007-06-25',  '2007-07-30',  'BK001',
'Introduction to Electrodynamics', 'English', 'CA001', 8, 25.00, 200.00),
    ('INV0006',  '2007-09-24',  'ORD/07-08/0007',  '2007-09-20',  '2007-09-30',  'BK003',
'Guide to Networking', 'Hindi', 'CA003', 20, 45.00, 900.00);
```

31.

```
CREATE VIEW view_purchase
AS SELECT invoice_no,book_name,cate_id
FROM purchase
WHERE cate_id= (SELECT cate_id FROM book_mast WHERE no_page=201);
```

32.

```
SELECT * FROM view_purchase;
```

33. **CREATE VIEW with JOIN**

```
CREATE TABLE category (
    cate_id VARCHAR(10) PRIMARY KEY,
    cate_descrip VARCHAR(50)
);
```

34.

```
INSERT INTO category (cate_id, cate_descrip)
VALUES
    ('CA001', 'Science'),
    ('CA002', 'Technology'),
    ('CA003', 'Computers'),
    ('CA004', 'Nature'),
    ('CA005', 'Medical');
```

35.

```
CREATE VIEW view_purchase1
```

```
AS SELECT a.cate_id,a.cate_descrip, b.invoice_no,
b.invoice_dt,b.book_name
FROM category a,purchase b
WHERE a.cate_id=b.cate_id;
```

36.

```
SELECT * FROM view_purchase1;
```

37. **CREATE VIEW with UNION**

```
CREATE VIEW view_bookmast3 AS
SELECT *
FROM book_mast
WHERE pub_id='P001' UNION
SELECT *
FROM book_mast
WHERE book_name BETWEEN 'A' AND 'G' UNION
SELECT *
FROM book_mast
WHERE no_page IN(165,250,350,400,510);
```

38.

```
SELECT * FROM view_bookmast3;
```

Lab 8: Indexing

1. In a database, indexing is a technique used to optimize the retrieval of records from a table.

2. Indexes are data structures that store a small portion of the table's data, typically the values of one or more columns, along with pointers to the corresponding rows in the table.

3. These indexes are organized in a way that allows for efficient data searching, sorting, and retrieval.

Why do we use indexing in DBMS?

1. **Faster Data Retrieval:** Indexes help speed up data retrieval operations, such as SELECT queries, by providing a quick path to locate the desired rows in a table.

2. **Types of Indexes:** There are different types of indexes, including:

    2.1. **Primary Index:** This type of index is defined on an ordered data file, typically on a key field such as the primary key of a relation.

    2.2. **Secondary Index:** A secondary index may be generated from a candidate key field with a unique value in every record or a non-key field with duplicate values.

    2.3. **Clustering Index:** A clustering index is defined on an ordered data file, usually on a non-key field.

3. **Structure-Based Indexing**

   3.1. **Dense Index:** In a dense index, there is an index record for every search key value in the database. It provides fast searching but requires more space to store index records.

   3.2. **Sparse Index:** Sparse indexes do not have an index record for every search key. Instead, they contain search key values and pointers to the actual data. If the desired data is not directly found via the index, the system conducts a sequential search until it finds the data.

   3.3. **Multilevel Index:** The index is stored alongside the actual database files on the disk. As the size of the database grows, the indices also increase. Multilevel indexing helps by breaking down the index into smaller indices, allowing the outermost level to be small enough to fit into memory.

   3.4. **B+ Tree:** A B+ tree is a balanced search tree commonly used for indexing. It follows a multilevel index format where leaf nodes denote actual data pointers. B+ trees ensure balanced height and support both random and sequential access.

4. **Maintenance Overhead:** While indexes can improve query performance, they also introduce overhead during data modification operations (INSERT, UPDATE, DELETE), as the indexes need to be updated to reflect changes in the underlying table.

5. **Index Selection:** Deciding which columns to index requires consideration of the types of queries frequently executed against the table. Indexes should be created on columns frequently used in WHERE clauses, JOIN conditions, or ORDER BY clauses.

6. **Indexing Strategies:** DBMSs employ various indexing strategies to organize and access data efficiently. Common strategies include B-tree, hash, and bitmap indexes, each with advantages and limitations.

7. **Monitoring and Optimization:** Index usage and performance should be monitored regularly. Unused indexes may consume resources unnecessarily, and poorly designed indexes can degrade performance. Periodic index optimization, such as index rebuilding or reorganization, may be necessary to maintain optimal performance.

Lab Performance Question

1. Creating a Database and Table

```
CREATE DATABASE IF NOT EXISTS ADBMS_LAB;

use ADBMS_LAB;

drop table employees;

CREATE TABLE employees (
    id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(100),
    department VARCHAR(100),
    salary DECIMAL(10, 2)
);

desc employees;
```

- This script creates a database named "ADBMS_LAB" if it doesn't exist.
- It then selects the newly created database.

- Finally, it creates a table named "employees" with columns for id, name, department, and salary.

2. Inserting Data into the Table

```
INSERT INTO employees (name, department, salary) VALUES
   ('John Doe', 'Engineering', 50000.00),
   ('Jane Smith', 'HR', 45000.00),
   ('Michael Johnson', 'Engineering', 55000.00),
   ('Emily Brown', 'Sales', 48000.00);

SELECT * FROM employees;
```

- Inserts data into the "employees" table, specifying values for name, department, and salary.
- Retrieves all rows from the "employees" table.

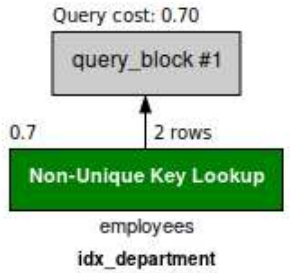
3. Creating Indexes

```
CREATE INDEX idx_department ON employees (department);

CREATE INDEX idx_salary ON employees (salary);

SHOW INDEX FROM employees;
```

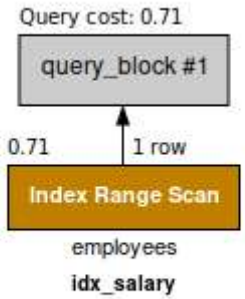
- Creates indexes on the "department" and "salary" columns of the "employees" table, which can improve query performance.

4. Querying with Conditions

```
SELECT * FROM employees WHERE department = 'Engineering';
```



SELECT * FROM employees WHERE salary > 50000.00;

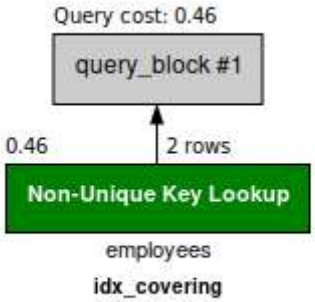


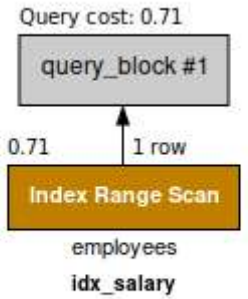
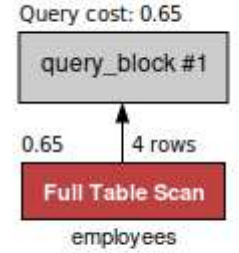
5. Creating Index

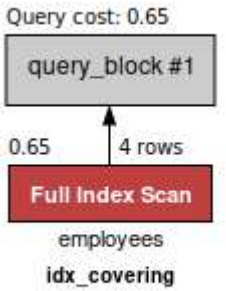
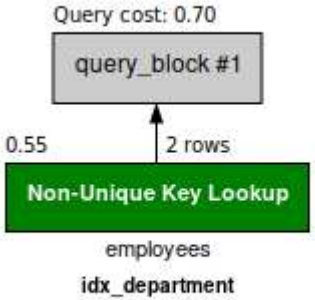
CREATE INDEX idx_covering ON employees (department, salary);

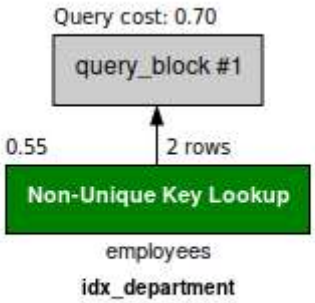
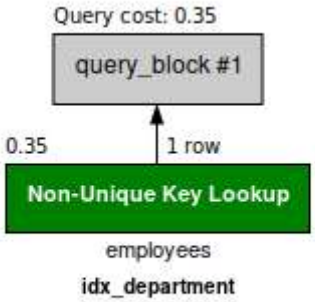
A covering index is an index that contains all the columns required by a query, allowing MySQL to retrieve the data directly from the index without accessing the actual table. This can improve query performance by avoiding the need to fetch data from the table.

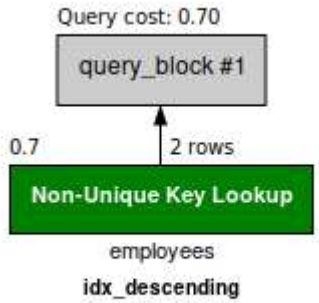
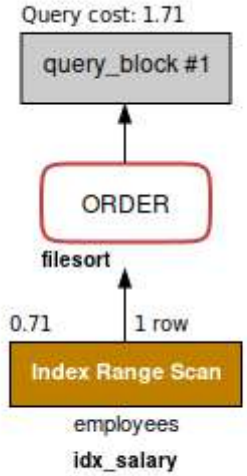
SELECT department, salary FROM employees WHERE department = 'Engineering';



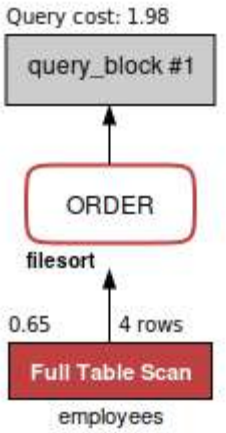
SELECT * FROM employees USE INDEX (idx_salary) WHERE salary > 50000.00; MySQL allows you to provide hints to the query optimizer to specify which index to use for a particular query. This can be useful when you want to ensure that a specific index is used, especially if the optimizer's choice is not optimal.	
SELECT * FROM employees USE INDEX (idx_covering) WHERE salary > 50000.00;	

SELECT department, salary FROM employees USE INDEX (idx_covering) WHERE salary > 50000.00;	
-- Query using index merge SELECT * FROM employees WHERE department = 'Engineering' AND salary > 50000.00; MySQL optimizer can sometimes use multiple indexes and merge their results to fulfill a query. This can be advantageous when individual indexes are selective but together cover the query conditions effectively.	

-- Query utilizing index condition pushdown SELECT * FROM employees WHERE salary > 50000.00 AND department = 'Engineering'; Index condition pushdown is a feature in MySQL that pushes filter conditions into the index scan, reducing the number of rows fetched from the table.	
-- Query with forced index SELECT * FROM employees FORCE INDEX (idx_department) WHERE department = 'Engineering'; You can force MySQL to use a specific index for a query using the FORCE INDEX directive. This can be useful when you know that a particular index is optimal for a query.	

CREATE INDEX idx_descending ON employees (department DESC); To create an index in descending order, you need to explicitly specify the order using the DESC keyword. SELECT department, salary FROM employees USE INDEX (idx_descending) WHERE department = 'Engineering';	
-- Query using an index for descending order scan SELECT * FROM employees WHERE salary > 50000.00 ORDER BY department DESC; When querying data using an index, MySQL automatically utilizes the index for both ascending and descending order scans. However, it's essential to ensure that your query conditions match the index order for optimal performance.	

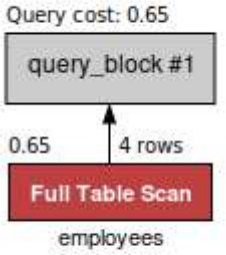
SELECT * FROM employees USE INDEX (idx_descending) WHERE salary > 50000.00 ORDER BY department DESC;



CREATE INDEX idx_mixed_order ON employees (department ASC, salary DESC);

You can also create an index with multiple columns, some in ascending order and others in descending order.

SELECT * FROM employees USE INDEX (idx_mixed_order) WHERE salary > 50000.00;



SHOW INDEX FROM employees;

Table	Non_unique	Key_name	Seq_in_index	Column_name	Collation	Cardinality	Sub_part	Packed	Null	Index_type	Comment	Index_comment	Visible	Expression
employees	0	PRIMARY	1	id	A	4	NULL	NULL		BTREE			YES	NULL
employees	1	idx_department	1	department	A	3	NULL	NULL	YES	BTREE			YES	NULL

employees	1	idx_salary	1	salary	A	4	NULL	NULL	YES	BTREE			YES	NULL
employees	1	idx_covering	1	department	A	3	NULL	NULL	YES	BTREE			YES	NULL
employees	1	idx_covering	2	salary	A	4	NULL	NULL	YES	BTREE			YES	NULL
employees	1	idx_descending	1	department	D	3	NULL	NULL	YES	BTREE			YES	NULL
employees	1	idx_mixed_order	1	department	A	3	NULL	NULL	YES	BTREE			YES	NULL
employees	1	idx_mixed_order	2	salary	D	4	NULL	NULL	YES	BTREE			YES	NULL

6. Drop Index

```
DROP INDEX idx_department ON employees;
DROP INDEX idx_salary ON employees;
DROP INDEX idx_covering ON employees;
DROP INDEX idx_covering ON employees;
DROP INDEX idx_descending ON employees;
DROP INDEX idx_mixed_order ON employees;
SHOW INDEX FROM employees;
```

Run every query and make the report of every Image of Indexing

7. Create the table

```
CREATE TABLE books (
    book_id INT AUTO_INCREMENT PRIMARY KEY,
    title VARCHAR(255),
    author VARCHAR(255),
    publication_year INT
);
```

8. Insert The Data

```
INSERT INTO books (title, author, publication_year) VALUES
('The Adventures of Tom Sawyer', 'Mark Twain', 1876),
('The Picture of Dorian Gray', 'Oscar Wilde', 1890),
('Alice\'s Adventures in Wonderland', 'Lewis Carroll', 1865),
('The War of the Worlds', 'H.G. Wells', 1898),
('Dracula', 'Bram Stoker', 1897),
('Jane Eyre', 'Charlotte Brontë', 1847),
```

('Wuthering Heights', 'Emily Brontë', 1847),
('Frankenstein', 'Mary Shelley', 1818),
('The Count of Monte Cristo', 'Alexandre Dumas', 1844),
('The Adventures of Sherlock Holmes', 'Arthur Conan Doyle', 1892),
('The Strange Case of Dr. Jekyll and Mr. Hyde', 'Robert Louis Stevenson', 1886),
('The Hound of the Baskervilles', 'Arthur Conan Doyle', 1902),
('Les Misérables', 'Victor Hugo', 1862),
('Anna Karenina', 'Leo Tolstoy', 1877),
('War and Peace', 'Leo Tolstoy', 1869),
('Crime and Punishment', 'Fyodor Dostoevsky', 1866),
('Gone with the Wind', 'Margaret Mitchell', 1936),
('The Grapes of Wrath', 'John Steinbeck', 1939),
('The Old Man and the Sea', 'Ernest Hemingway', 1952),
('The Sun Also Rises', 'Ernest Hemingway', 1926),
('Of Mice and Men', 'John Steinbeck', 1937),
('One Hundred Years of Solitude', 'Gabriel García Márquez', 1967),
('Don Quixote', 'Miguel de Cervantes', 1605),
('The Scarlet Letter', 'Nathaniel Hawthorne', 1850),
('The Canterbury Tales', 'Geoffrey Chaucer', 1400),
('Heart of Darkness', 'Joseph Conrad', 1899),
('The Odyssey', 'Homer', 800),
('The Iliad', 'Homer', 800),
('Catch-22', 'Joseph Heller', 1961),
('To the Lighthouse', 'Virginia Woolf', 1927),
('Slaughterhouse-Five', 'Kurt Vonnegut', 1969),
('The Bell Jar', 'Sylvia Plath', 1963),
('The Great Expectations', 'Charles Dickens', 1861),
('The Adventures of Huckleberry Finn', 'Mark Twain', 1884),
('The Chronicles of Narnia', 'C.S. Lewis', 1950),
('Lord of the Flies', 'William Golding', 1954),
('The Outsiders', 'S.E. Hinton', 1967),
('The Road', 'Cormac McCarthy', 2006),
('Beloved', 'Toni Morrison', 1987),
('Pride and Prejudice', 'Jane Austen', 1813),
('The Hobbit', 'J.R.R. Tolkien', 1937),
('The Lord of the Rings', 'J.R.R. Tolkien', 1954),
('Harry Potter and the Philosopher\'s Stone', 'J.K. Rowling', 1997),
('Moby-Dick', 'Herman Melville', 1851),
('Brave New World', 'Aldous Huxley', 1932),

('The Da Vinci Code', 'Dan Brown', 2003),
('Angels & Demons', 'Dan Brown', 2000),
('The Girl with the Dragon Tattoo', 'Stieg Larsson', 2005),
('The Hunger Games', 'Suzanne Collins', 2008),
('The Catcher in the Rye', 'J.D. Salinger', 1951),
('The Martian', 'Andy Weir', 2011),
('A Game of Thrones', 'George R.R. Martin', 1996),
('The Help', 'Kathryn Stockett', 2009),
('The Girl on the Train', 'Paula Hawkins', 2015),
('Gone Girl', 'Gillian Flynn', 2012),
('The Lovely Bones', 'Alice Sebold', 2002),
('Life of Pi', 'Yann Martel', 2001),
('The Alchemist', 'Paulo Coelho', 1988),
('Twilight', 'Stephenie Meyer', 2005),
('The Secret Life of Bees', 'Sue Monk Kidd', 2001),
('The Book Thief', 'Markus Zusak', 2005),
('Water for Elephants', 'Sara Gruen', 2006),
('The Girl with the Pearl Earring', 'Tracy Chevalier', 1999),
('The Help', 'Kathryn Stockett', 2009),
('The Kite Runner', 'Khaled Hosseini', 2003),
('The Glass Castle', 'Jeannette Walls', 2005),
('Eat, Pray, Love', 'Elizabeth Gilbert', 2006),
('The Time Traveler\'s Wife', 'Audrey Niffenegger', 2003),
('The Goldfinch', 'Donna Tartt', 2013),
('Big Little Lies', 'Liane Moriarty', 2014),
('The Nightingale', 'Kristin Hannah', 2015),
('The Underground Railroad', 'Colson Whitehead', 2016),
('The Silent Patient', 'Alex Michaelides', 2019),
('Where the Crawdads Sing', 'Delia Owens', 2018),
('Becoming', 'Michelle Obama', 2018),
('Educated', 'Tara Westover', 2018),
('Circe', 'Madeline Miller', 2018),
('Normal People', 'Sally Rooney', 2018),
('Little Fires Everywhere', 'Celeste Ng', 2017),
('The Testaments', 'Margaret Atwood', 2019),
('The Tattooist of Auschwitz', 'Heather Morris', 2018),
('The Wife Between Us', 'Greer Hendricks', 2018),
('The Silent Patient', 'Alex Michaelides', 2019),
('Where the Crawdads Sing', 'Delia Owens', 2018),

```
('Becoming', 'Michelle Obama', 2018),
('Educated', 'Tara Westover', 2018),
('Circe', 'Madeline Miller', 2018),
('Normal People', 'Sally Rooney', 2018),
('Little Fires Everywhere', 'Celeste Ng', 2017),
('The Testaments', 'Margaret Atwood', 2019),
('The Tattooist of Auschwitz', 'Heather Morris', 2018),
('The Wife Between Us', 'Greer Hendricks', 2018);
```

LAB 10 #PL/SQL Commands

Write a PL/SQL code to accept the value of A, B & C display which is greater.

```
DECLARE
   a NUMBER := 10; -- Input value for A
   b NUMBER := 20; -- Input value for B
   c NUMBER := 15; -- Input value for C
BEGIN
   IF a > b AND a > c THEN
      DBMS_OUTPUT.PUT_LINE('A is greater');
   ELSIF b > a AND b > c THEN
      DBMS_OUTPUT.PUT_LINE('B is greater');
   ELSE
      DBMS_OUTPUT.PUT_LINE('C is greater');
   END IF;
END;
/
```

Using PL/SQL Statements create a simple loop that display message "Welcome to PL/SQL Programming" 20 times.

```
BEGIN
   FOR i IN 1..20 LOOP
      DBMS_OUTPUT.PUT_LINE('Welcome to PL/SQL Programming');
   END LOOP;
END;
/
```

Write a PL/SQL code block to find the factorial of a number.

```
DECLARE
   num NUMBER := 5; -- Input number
   factorial NUMBER := 1;
BEGIN
   FOR i IN 1..num LOOP
      factorial := factorial * i;
   END LOOP;
   DBMS_OUTPUT.PUT_LINE('Factorial of ' || num || ' is ' || factorial);
END;
/
```

Write a PL/SQL program to generate Fibonacci series.

```
-- Create a PL/SQL block to generate Fibonacci series
DECLARE
   -- Variables to hold Fibonacci sequence
   first_number NUMBER := 0;
   second_number NUMBER := 1;
   next_number NUMBER;
   limit_number NUMBER := 100; -- Change this limit as per requirement

   -- Cursor declaration to print Fibonacci series
   CURSOR fibonacci_cursor IS
      SELECT LEVEL AS n, first_number AS Fibonacci_Number
      FROM DUAL
```

```
    CONNECT BY LEVEL <= limit_number;
BEGIN
   -- Print the initial values
   DBMS_OUTPUT.PUT_LINE('Fibonacci Series up to ' || limit_number || ' numbers:');
   DBMS_OUTPUT.PUT_LINE('0'); -- First Fibonacci number
   DBMS_OUTPUT.PUT_LINE('1'); -- Second Fibonacci number

   -- Loop to generate and print Fibonacci series
   FOR i IN 3..limit_number LOOP
      next_number := first_number + second_number;
      first_number := second_number;
      second_number := next_number;
      DBMS_OUTPUT.PUT_LINE(next_number);
   END LOOP;
END;
/
```

Write a PL/SQL code to find the sum of first N numbers.

```
-- Create a PL/SQL block to find the sum of first N numbers
DECLARE
   -- Variable to hold the sum
   total_sum NUMBER := 0;

   -- Variable for the number up to which sum is to be calculated
   N NUMBER := 100; -- Change N to the desired number

BEGIN
   -- Loop to calculate the sum of first N numbers
   FOR i IN 1..N LOOP
      total_sum := total_sum + i;
   END LOOP;

   -- Print the sum
   DBMS_OUTPUT.PUT_LINE('The sum of first ' || N || ' numbers is: ' || total_sum);
END;
/
```

WAP that accept the value of A, B & C display which is greater using Functions and Procedures.

```
DECLARE
   -- Variables to hold the values of A, B, and C
   A NUMBER := 10; -- Replace with your desired value for A
   B NUMBER := 20; -- Replace with your desired value for B
   C NUMBER := 15; -- Replace with your desired value for C

   -- Function to find the maximum of two numbers
   FUNCTION find_maximum(x NUMBER, y NUMBER) RETURN NUMBER IS
   BEGIN
      IF x > y THEN
         RETURN x;
      ELSE
         RETURN y;
      END IF;
   END find_maximum;
```

```
   -- Procedure to compare three numbers and display the greatest
   PROCEDURE compare_and_display_greatest(A IN NUMBER, B IN NUMBER, C IN NUMBER)
IS
      greatest NUMBER;
   BEGIN
      greatest := find_maximum(find_maximum(A, B), C); -- Call the function to find the maximum of
three numbers

      -- Display the result
      IF greatest = A THEN
         DBMS_OUTPUT.PUT_LINE('A is the greatest.');
      ELSIF greatest = B THEN
         DBMS_OUTPUT.PUT_LINE('B is the greatest.');
      ELSE
         DBMS_OUTPUT.PUT_LINE('C is the greatest.');
      END IF;
   END compare_and_display_greatest;
BEGIN
   -- Call the procedure to compare and display the greatest
   compare_and_display_greatest(A, B, C);
END;
/
```

Print the statements "Welcome to PL/SQL Programming" 20 times using Functions and Procedures.

```
DECLARE
   -- Variable to store the message
   message VARCHAR2(100) := 'Welcome to PL/SQL Programming';

   -- Procedure to display message 20 times
   PROCEDURE display_message IS
   BEGIN
      -- Loop to display the message 20 times
      FOR i IN 1..20 LOOP
         DBMS_OUTPUT.PUT_LINE(message);
      END LOOP;
   END display_message;
BEGIN
   -- Call the procedure to display the message
   display_message;
END;
/
```

Factorial of a number using function and procedure.

```
DECLARE
   -- Variable to hold the value for factorial calculation
   n NUMBER := 5; -- Replace with your desired value for factorial calculation

   -- Function to calculate factorial of a number
   FUNCTION factorial(num NUMBER) RETURN NUMBER IS
      result NUMBER := 1;
   BEGIN
      IF num <= 1 THEN
         RETURN result;
```

```
        ELSE
            FOR i IN 2..num LOOP
                result := result * i;
            END LOOP;
            RETURN result;
        END IF;
    END factorial;

    -- Procedure to calculate and display factorial
    PROCEDURE calculate_and_display_factorial(n IN NUMBER) IS
        fact NUMBER;
    BEGIN
        fact := factorial(n); -- Call the function to calculate factorial

        -- Display the result
        DBMS_OUTPUT.PUT_LINE('Factorial of ' || n || ' is: ' || fact);
    END calculate_and_display_factorial;
BEGIN
    -- Call the procedure to calculate and display factorial
    calculate_and_display_factorial(n);
END;
/
```

Fibonacci series of a number using function and procedure.

```
DECLARE
    -- Variable to hold the number of terms in the Fibonacci series
    num_terms NUMBER := 16; -- Replace with your desired number of terms

    -- Function to calculate Fibonacci series
    FUNCTION fibonacci_series(num_terms_in NUMBER) RETURN VARCHAR2 IS
        fib_series VARCHAR2(1000) := '0, 1'; -- Initial values of Fibonacci series
        a NUMBER := 0;
        b NUMBER := 1;
        next_term NUMBER;
    BEGIN
        IF num_terms_in <= 0 THEN
            RETURN 'Invalid input';
        END IF;

        IF num_terms_in = 1 THEN
            RETURN '0';
        END IF;

        IF num_terms_in = 2 THEN
            RETURN '0, 1';
        END IF;

        FOR i IN 3..num_terms_in LOOP
            next_term := a + b;
            fib_series := fib_series || ', ' || next_term;
            a := b;
            b := next_term;
        END LOOP;

        RETURN fib_series;
    END fibonacci_series;
```

```
   -- Procedure to display Fibonacci series
   PROCEDURE display_fibonacci_series(num_terms_in NUMBER) IS
      fib_series VARCHAR2(1000);
   BEGIN
      fib_series := fibonacci_series(num_terms_in); -- Call the function to calculate Fibonacci series

      -- Display the result
      DBMS_OUTPUT.PUT_LINE('Fibonacci series up to ' || num_terms_in || ' terms: ' || fib_series);
   END display_fibonacci_series;
BEGIN
   -- Call the procedure to display Fibonacci series
   display_fibonacci_series(num_terms);
END;
/
```

Find the sum of first N numbers using Function and Procedure.

```
DECLARE
   -- Variable to hold the number of terms
   num_terms NUMBER := 16; -- Replace with your desired number of terms

   -- Function to calculate the sum of first N numbers
   FUNCTION sum_of_first_n_numbers(n NUMBER) RETURN NUMBER IS
      total_sum NUMBER := 0;
   BEGIN
      IF n <= 0 THEN
         RETURN 0; -- Return 0 for non-positive input
      END IF;

      FOR i IN 1..n LOOP
         total_sum := total_sum + i; -- Add each number from 1 to n to total_sum
      END LOOP;

      RETURN total_sum;
   END sum_of_first_n_numbers;

   -- Procedure to display the sum of first N numbers
   PROCEDURE display_sum_of_first_n_numbers(n NUMBER) IS
      sum_result NUMBER;
   BEGIN
      sum_result := sum_of_first_n_numbers(n); -- Call the function to calculate sum
      -- Display the result
      DBMS_OUTPUT.PUT_LINE('Sum of first ' || n || ' numbers: ' || sum_result);
   END display_sum_of_first_n_numbers;
BEGIN
   -- Call the procedure to display the sum
   display_sum_of_first_n_numbers(num_terms);
END;
/
```

Lab Performance Questions

1. Write a PL/SQL code to calculate the area of a circle given its radius.
2. Create a PL/SQL program to check whether a given number is prime or not.
3. Develop a PL/SQL script to reverse a string entered by the user.
4. Design a PL/SQL code to find the largest and smallest number among three given numbers.

5. Implement a PL/SQL program to calculate the compound interest for a given principal amount, rate, and time period.

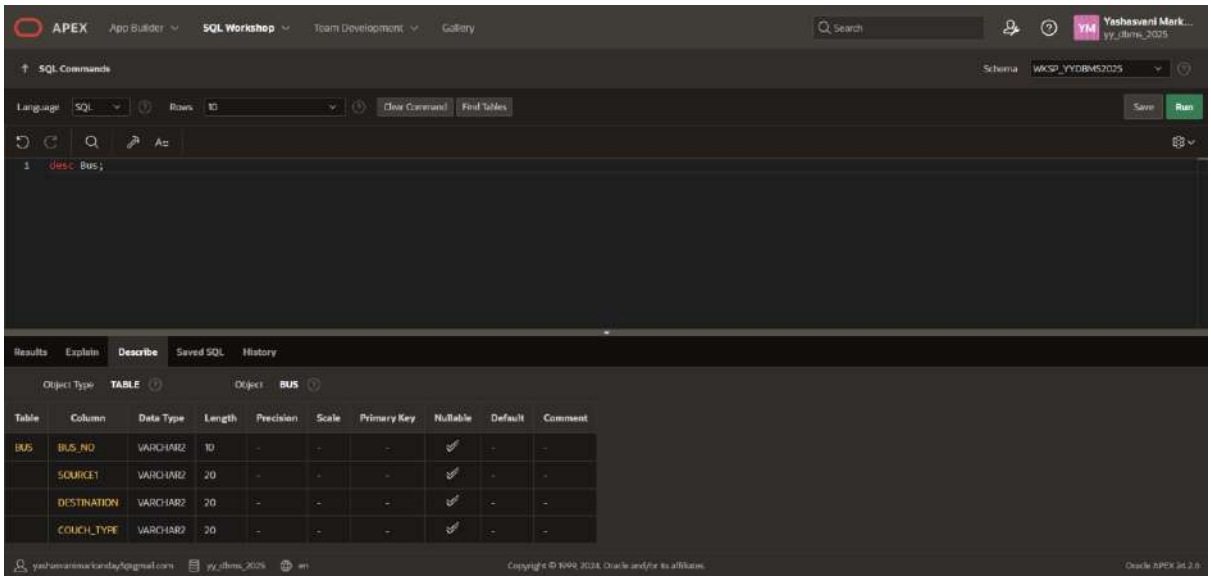
UNIVERSITY OF PETROLEUM AND ENERGY STUDIES

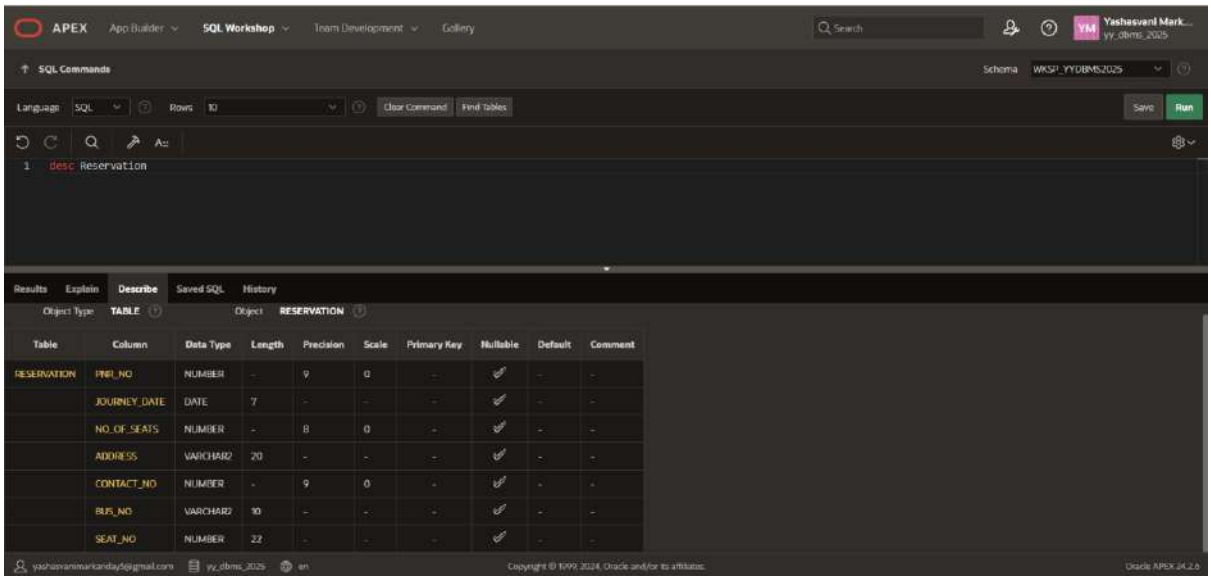
Program Name: B. Tech CS **Semester: III**

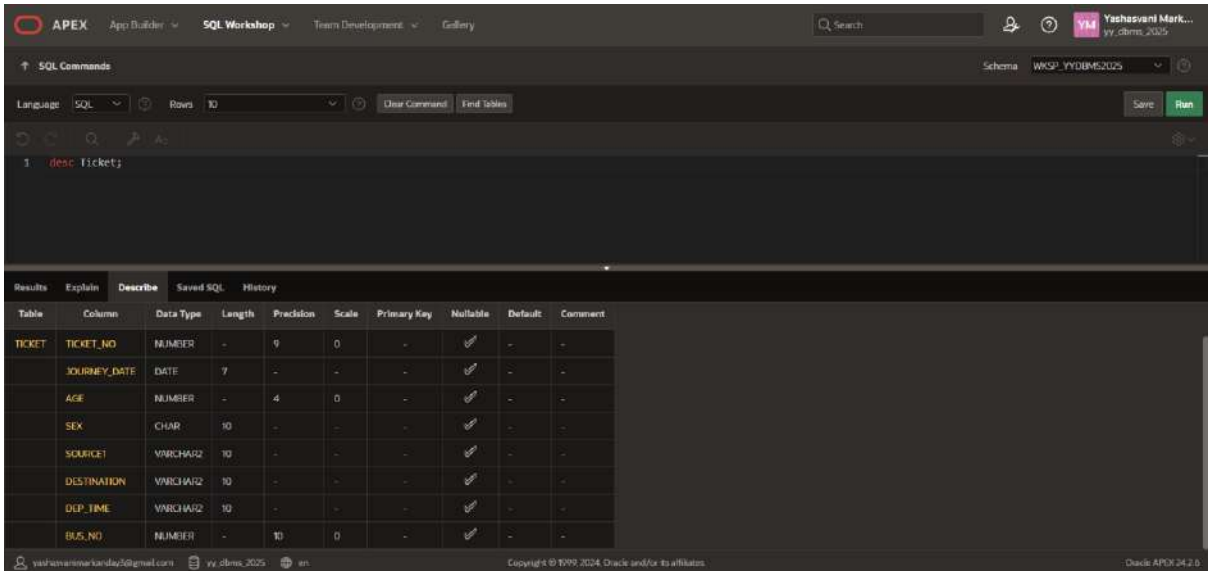
Name: Yashasvani Markanday **SAP ID: 590011386**

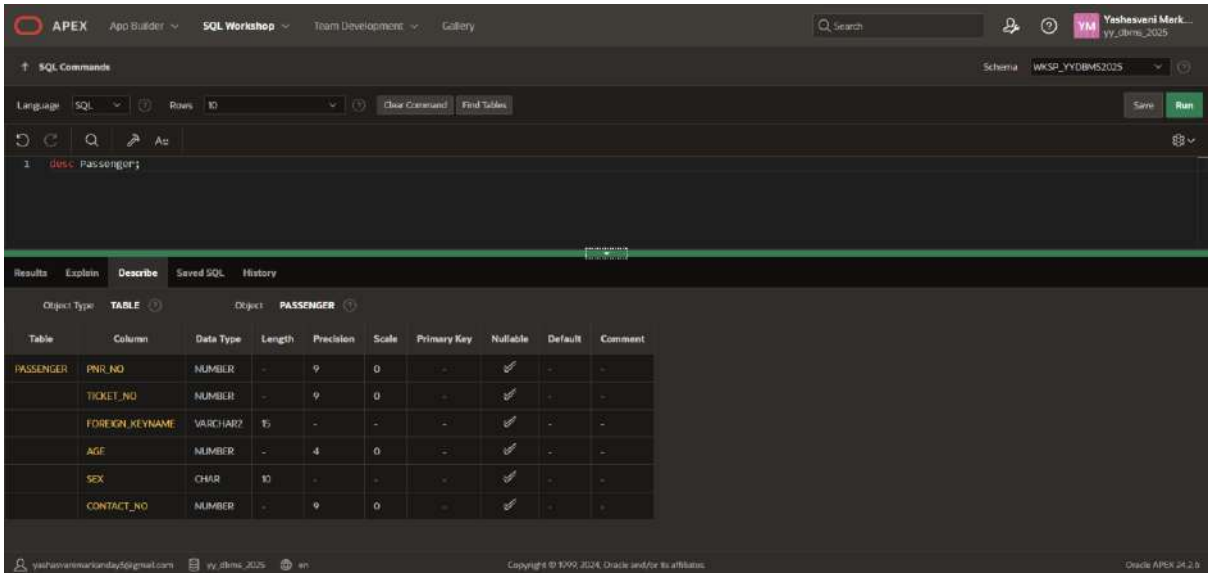
Batch: B2 **Subject: Database and Management System**

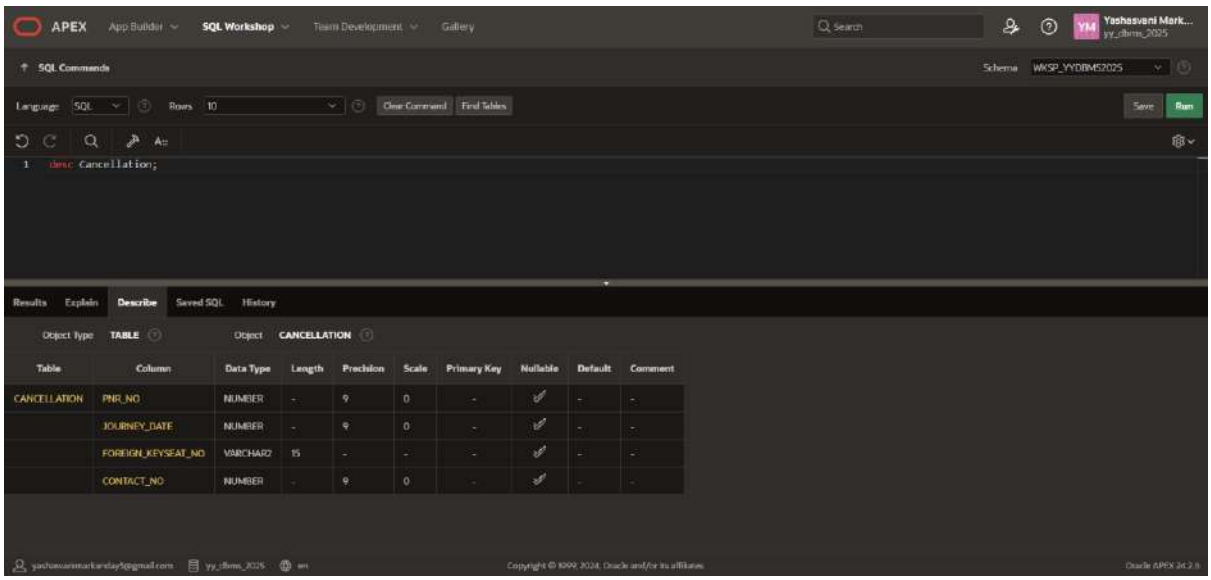
3 Create all the given table.



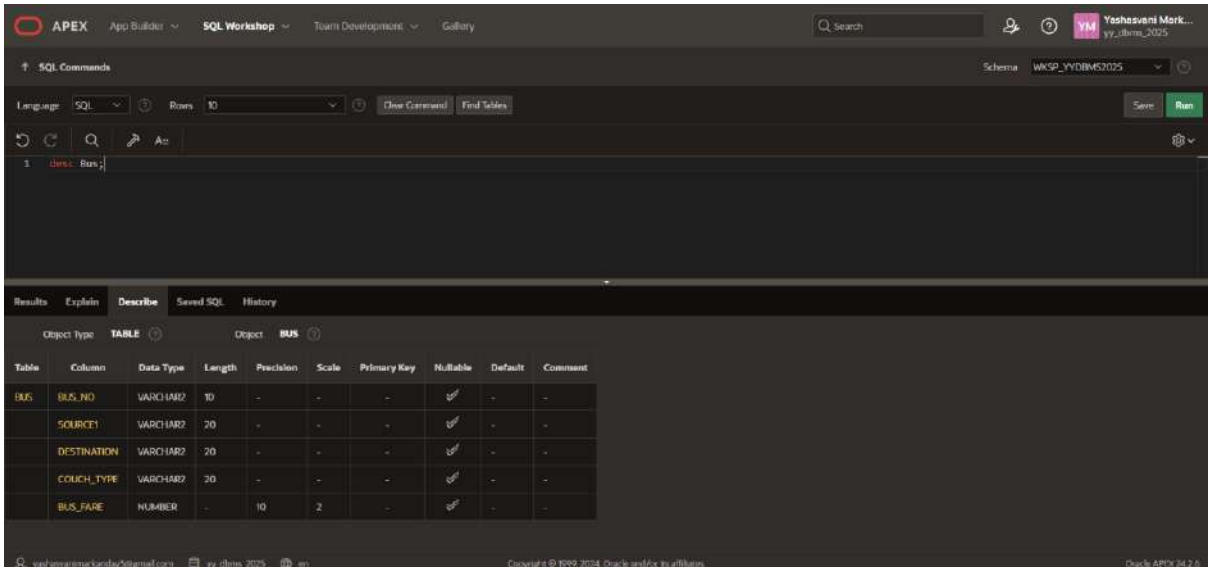



APEX
App Builder
SQL Workshop
Team Development
Gallery
Search
Yashasvani Mark...
yy_dbms_2025
SQL Commands
Schema
WKSP_YYDBMS2025
Language
SQL
Rows
10
Clear Command
Find Tables
Save
Run
1 desc ticket;
Results
Explain
Describe
Saved SQL
History
Table
Column
Data Type
Length
Precision
Scale
Primary Key
Nullable
Default
Comment
TICKET
TICKET_NO
NUMBER
JOURNEY_DATE
DATE
AGE
NUMBER
SEX
CHAR
SOURCE1
VARCHAR2
DESTINATION
VARCHAR2
DEP_TIME
VARCHAR2
BUS_NO
NUMBER
yashasvanimarkanday5@gmail.com
yy_dbms_2025
en
Copyright © 1999, 2024, Oracle and/or its affiliates
Oracle APEX 24.2.6

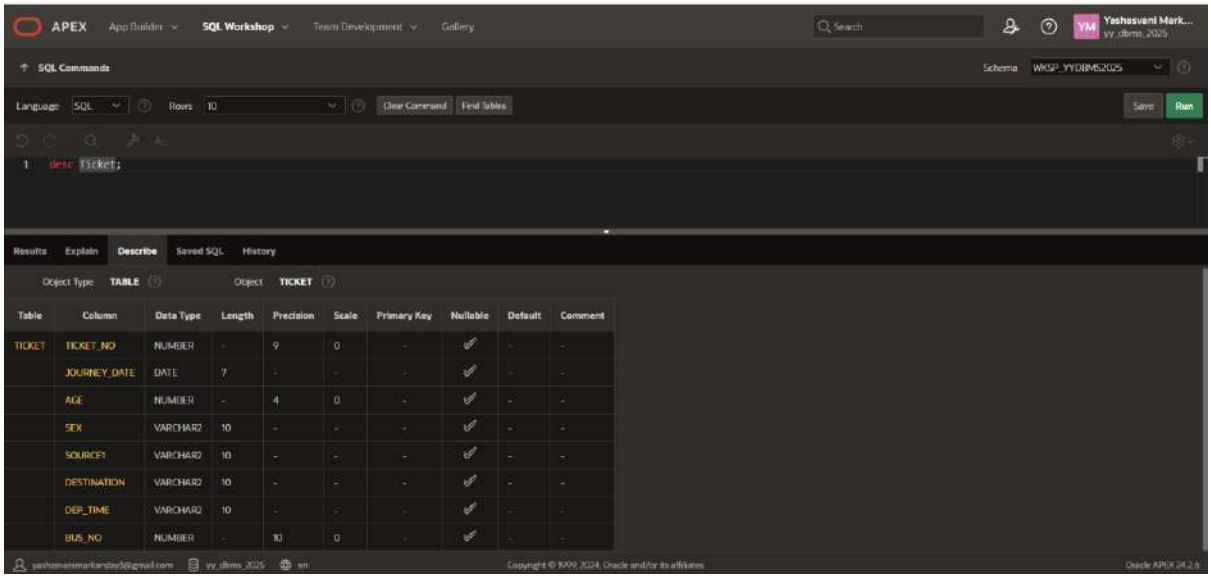

APEX
App Builder
SQL Workshop
Team Development
Gallery
Search
Yashasvani Mark...
yy_dbms_2025
SQL Commands
Schema
WKSP_YYDBMS2025
Language
SQL
Rows
10
Clear Command
Find Tables
Save
Run
1 desc Passenger;
Results
Explain
Describe
Saved SQL
History
Object Type
TABLE
Object
PASSENGER
Table
Column
Data Type
Length
Precision
Scale
Primary Key
Nullable
Default
Comment
PASSENGER
PNR_NO
NUMBER
TICKET_NO
NUMBER
FOREIGN_KEYNAME
VARCHAR2
15
AGE
NUMBER
4
SEX
CHAR
10
CONTACT_NO
NUMBER
yashasvanimarkanday5@gmail.com
yy_dbms_2025
en
Copyright © 1999, 2024, Oracle and/or its affiliates
Oracle APEX 24.2.6


APEX
App Builder
SQL Workshop
Team Development
Gallery
Search
Yashasvani Mark...
yy_dbms_2025
SQL Commands
Schema
WKSP_YYDBMS2025
Language
SQL
Rows
10
Clear Command
Find Tables
Save
Run
1 desc Cancellation;
Results
Explain
Describe
Saved SQL
History
Object Type
TABLE
Object
CANCELLATION
Table
Column
Data Type
Length
Precision
Scale
Primary Key
Nullable
Default
Comment
CANCELLATION
PNR_NO
NUMBER
JOURNEY_DATE
NUMBER
FOREIGN_KEYSEAT_NO
VARCHAR2
15
CONTACT_NO
NUMBER
yashasvanimarkanday5@gmail.com
yy_dbms_2025
en
Copyright © 1999, 2024, Oracle and/or its affiliates
Oracle APEX 24.2.6

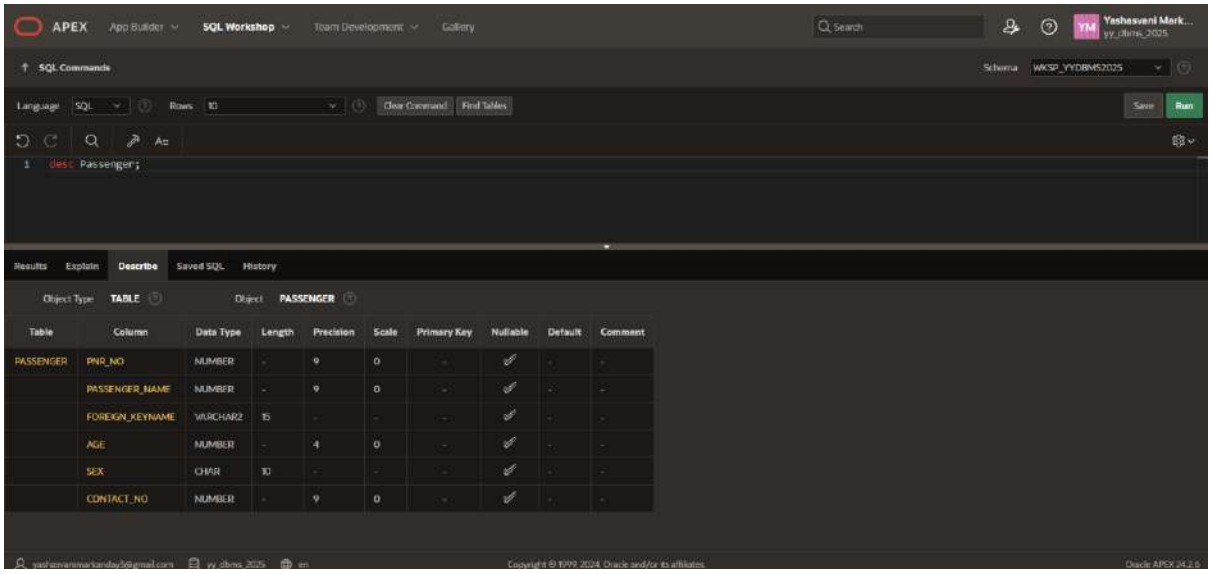
4 Add a new column to the existing relation.



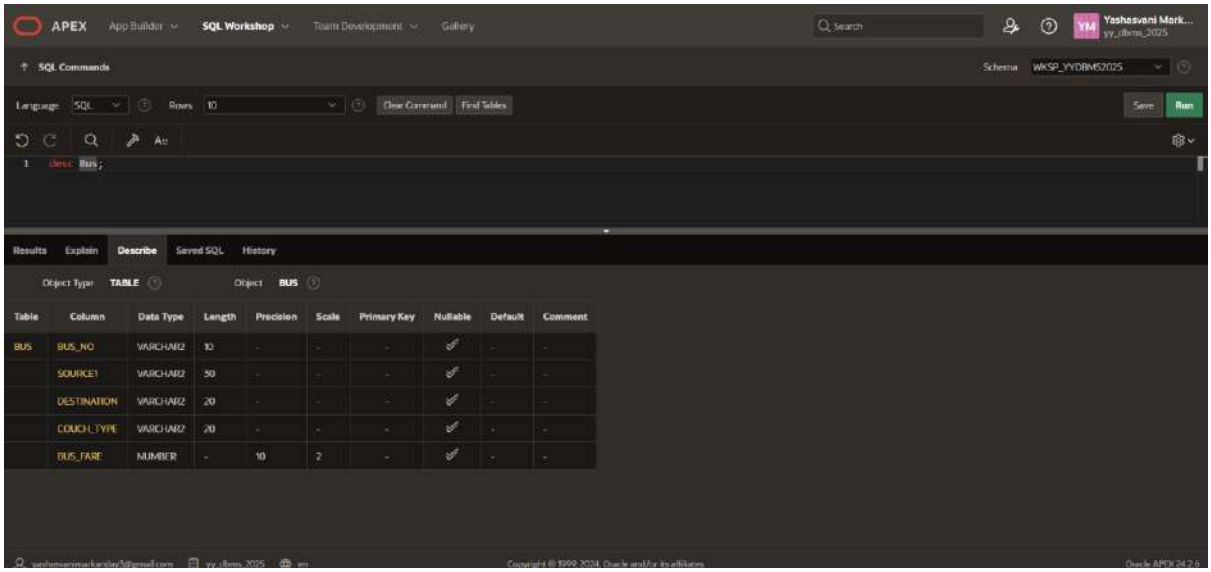
5 Change the datatype of the table from char to varchar2.

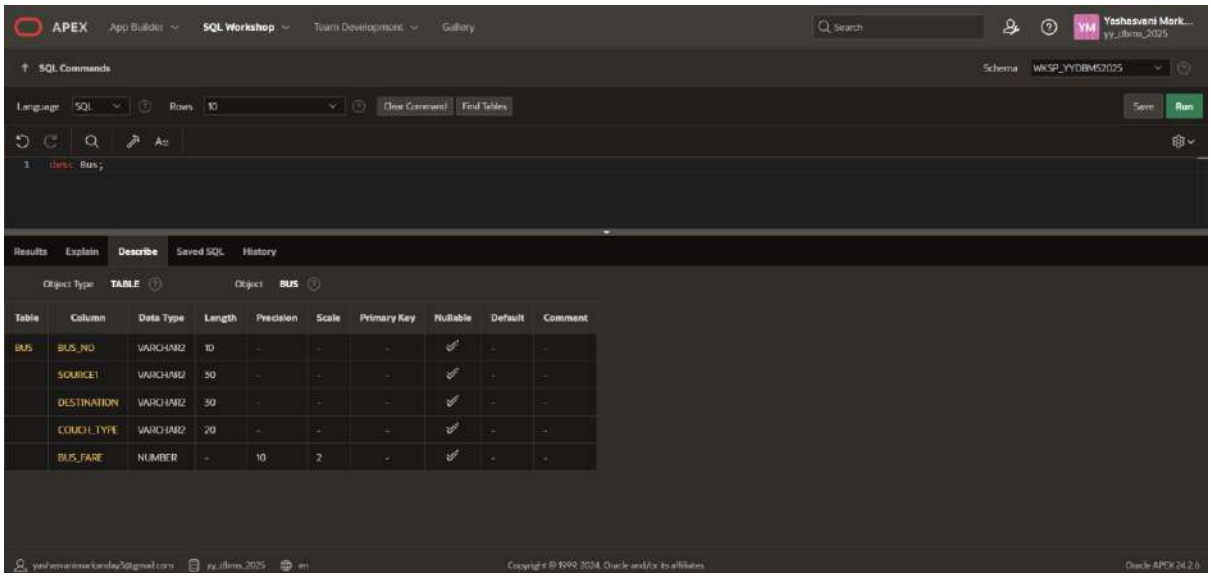


6 Change the name of column/field.



7 Modify the column width of all the table.





Conclusion

Through this lab, I gained practical knowledge of creating, modifying, and managing database tables using SQL commands also introduced the fundamentals of RDBMS concepts, E.F. Codd's rules, and the use of SQL for defining and manipulating data. The exercise reinforced understanding of relationships, constraints, and data types in real-world database and alter column sizes as per requirements. Differences between TRUNCATE, DELETE, and DROP were clarified for better database maintenance strategies. These exercises improved my understanding of schema design and data definition operations.

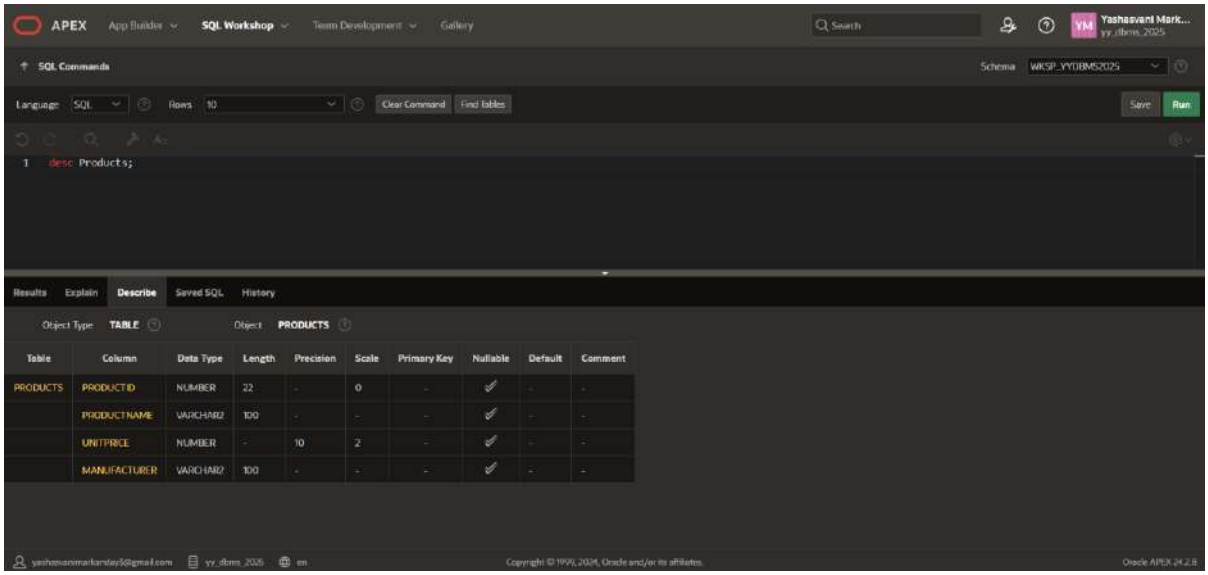
UNIVERSITY OF PETROLEUM AND ENERGY STUDIES

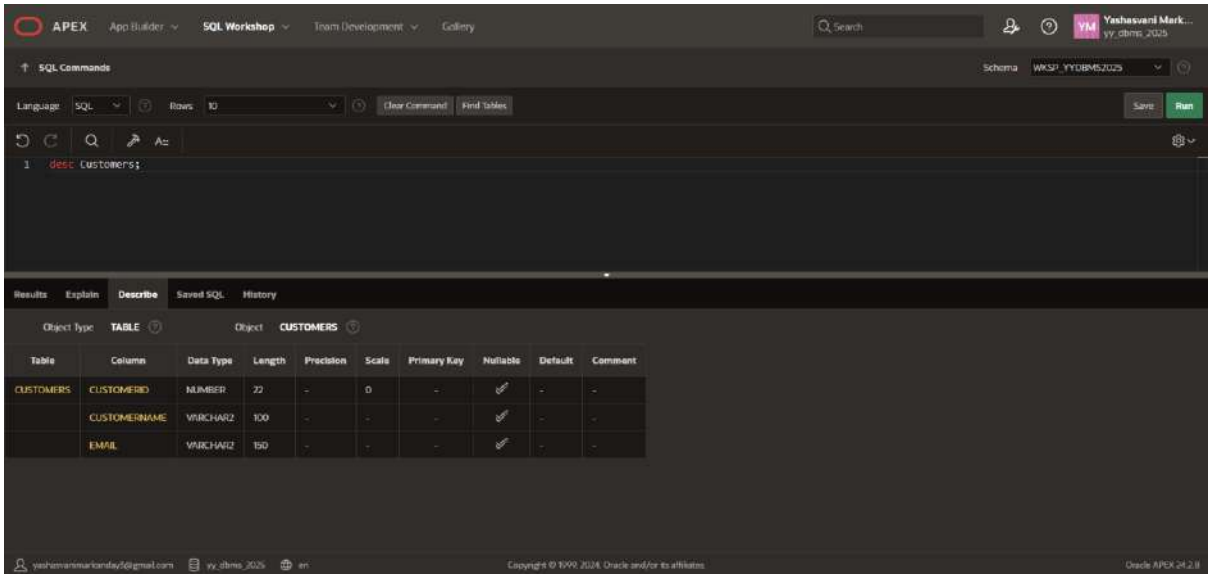
Program Name: B. Tech CS **Semester: III**

Name: Yashasvani Markanday **SAP ID: 590011386**

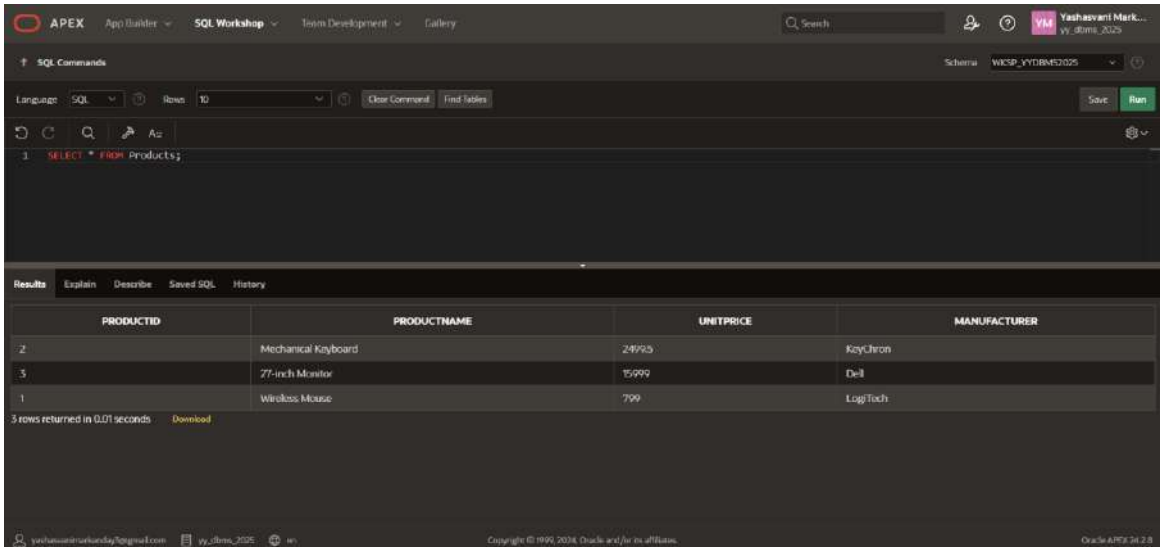
Batch: B2 **Subject: Database and Management System**

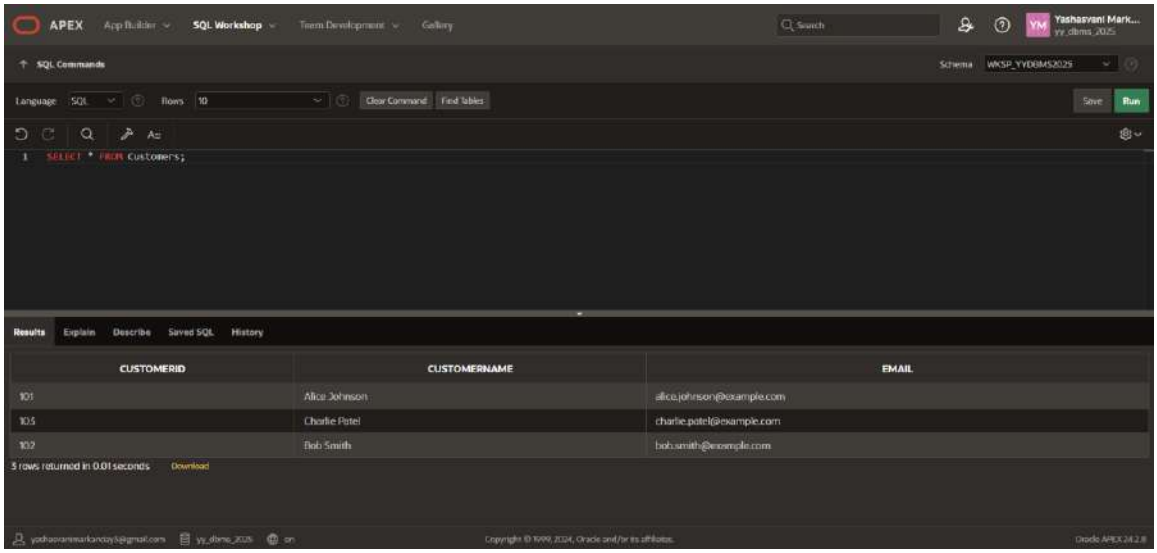
1. Write an SQL script that includes DDL statements to create the following tables: - `Products` with columns: `ProductID` (integer), `ProductName` (varchar), `UnitPrice` (decimal), `Manufacturer` (varchar). - `Customers` with columns: `CustomerID` (integer), `CustomerName` (varchar), `Email` (varchar). Ensure appropriate constraints for data integrity.



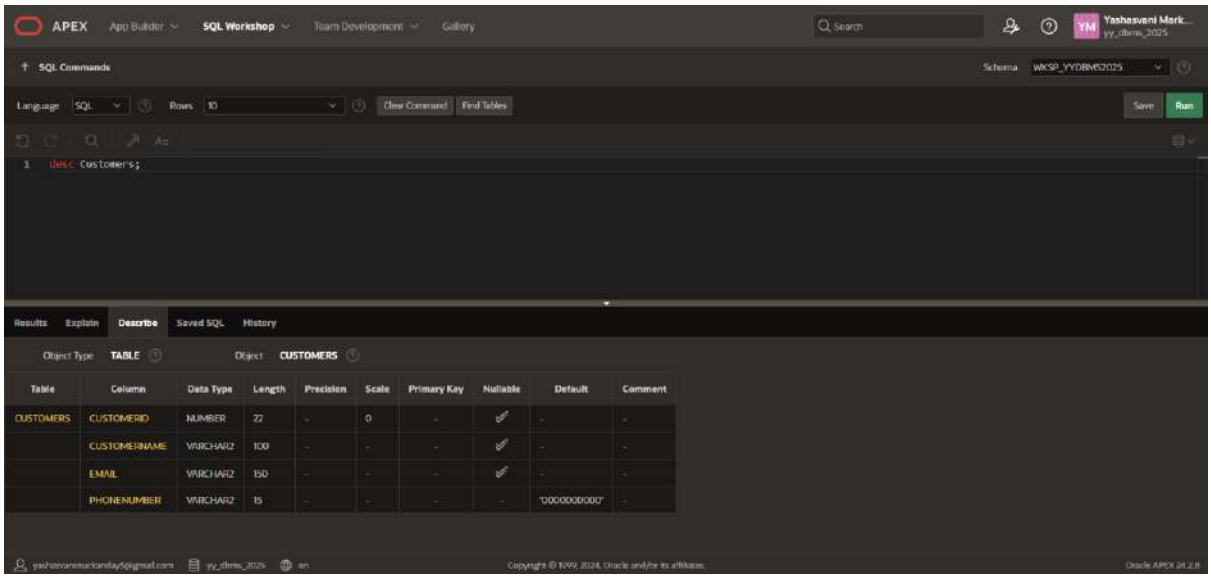


2. Insert at least three sample records into the `Products` table created in Question 1. Ensure each record has a unique product ID. Insert at least three sample records into the `Customers` table created in Question 1. Each record should have a unique customer ID.

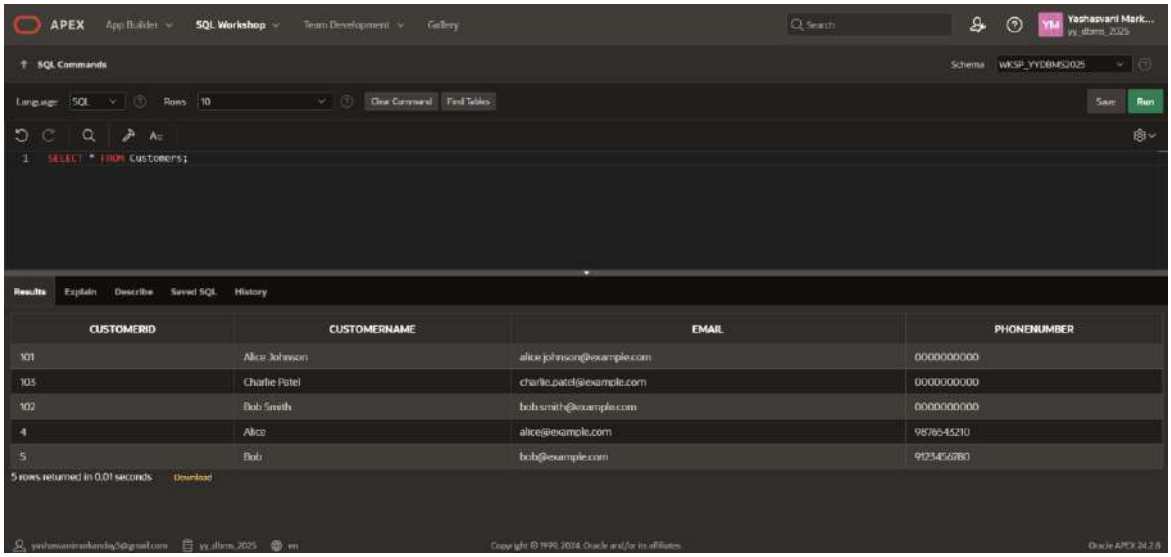




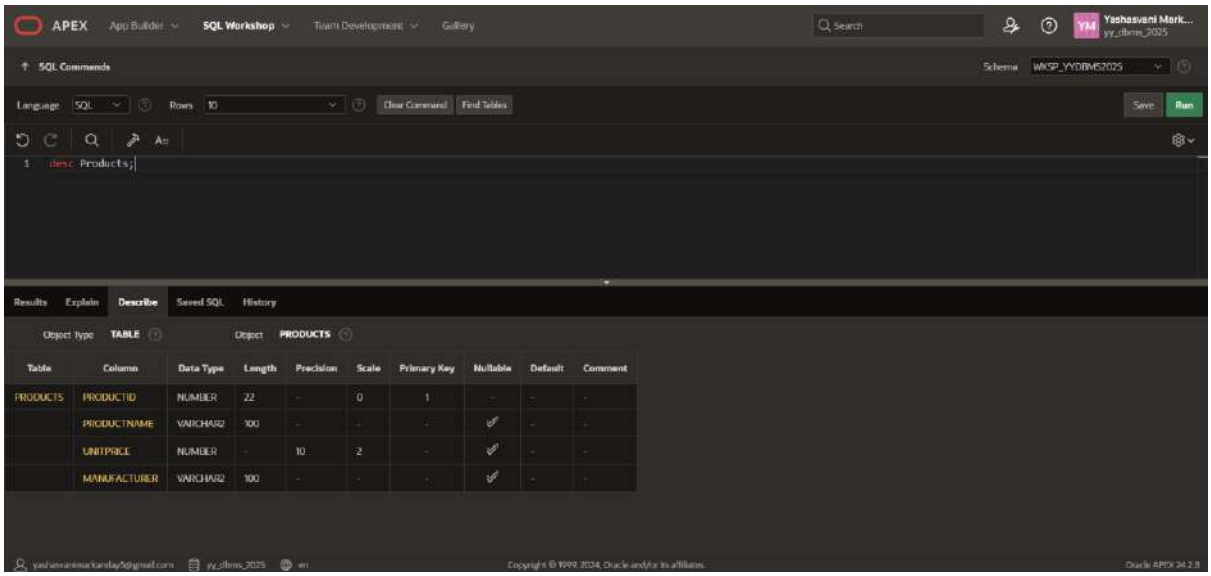
3. Add a column named `PhoneNumber` to the `Customers` table. Apply the NOT NULL constraint to ensure that this information is always provided.

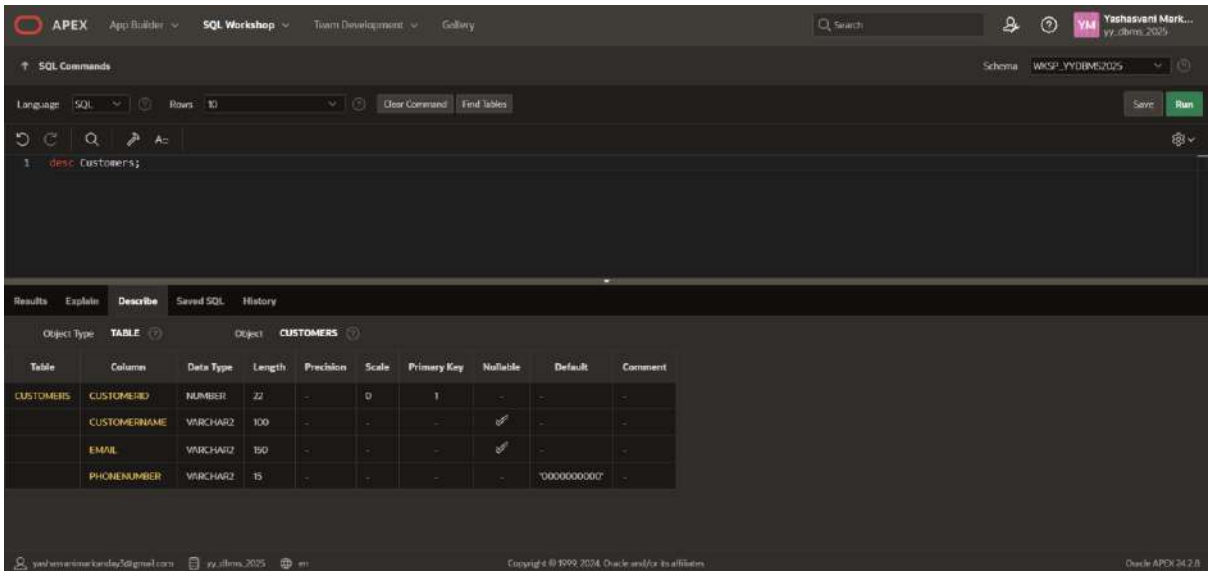


4. Implement a UNIQUE constraint on the `Email` column in the `Customers` table. Provide an example of how this constraint prevents data duplication.

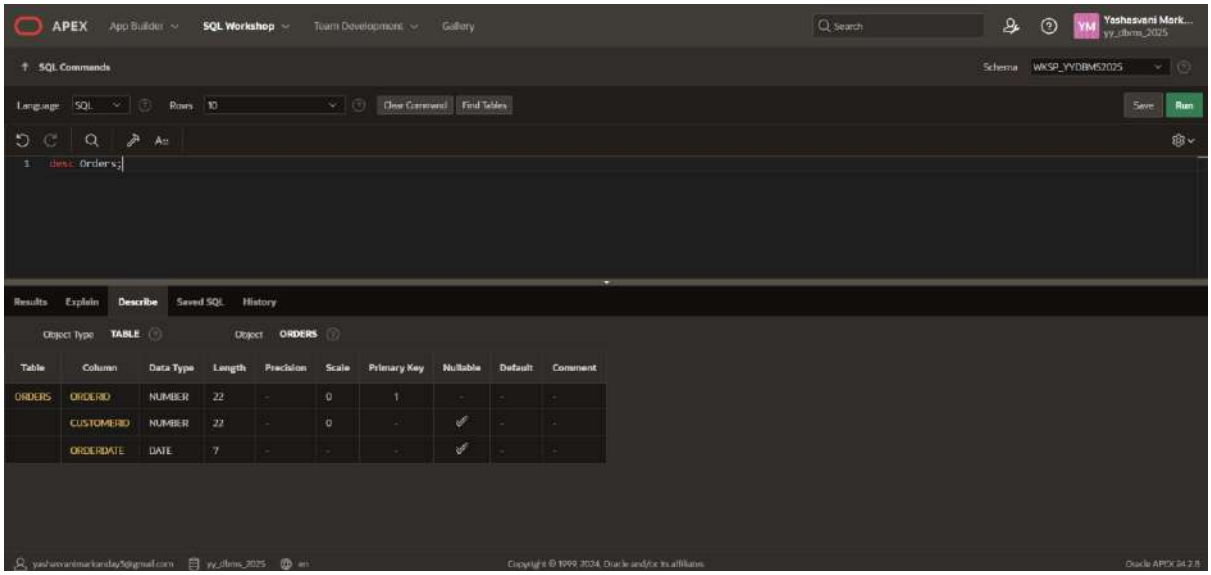


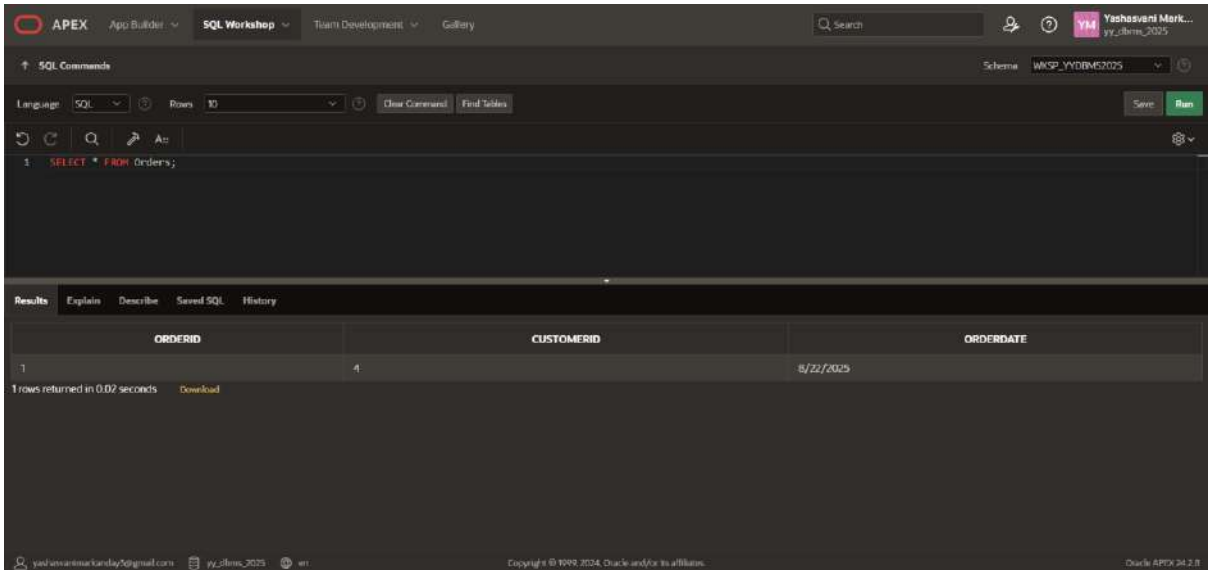
5. Set the `ProductID` column as the primary key for the `Products` table. Also, define `CustomerID` as the primary key for the `Customers` table.

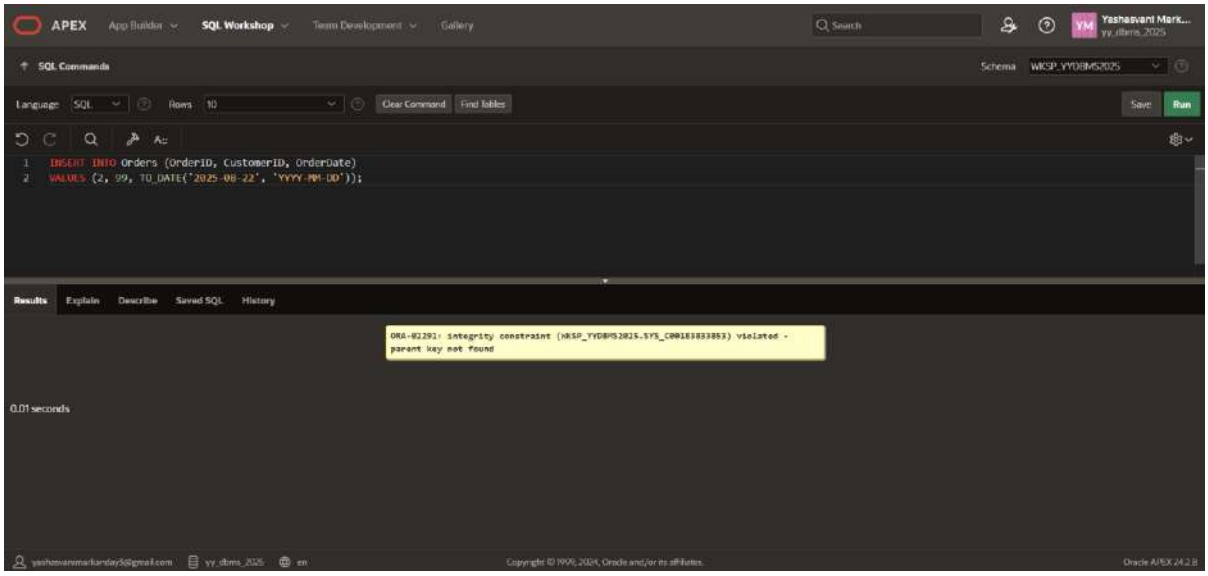




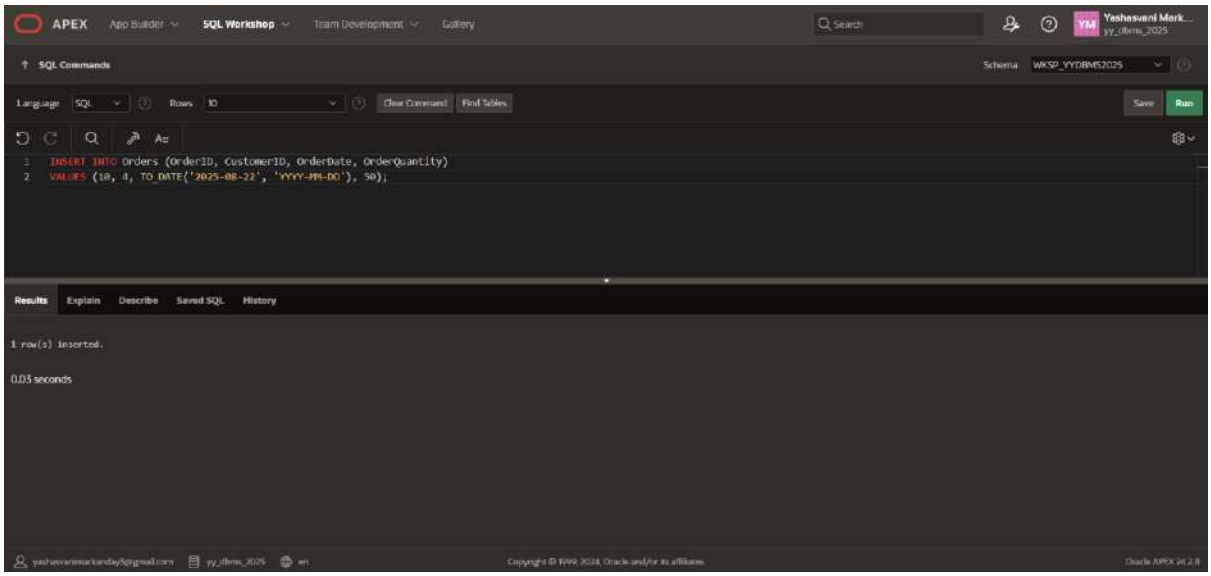
6. Establish a relationship between the `Orders` table (containing columns `OrderID` and `CustomerID`) and the `Customers` table. Ensure referential integrity by using a FOREIGN KEY constraint.

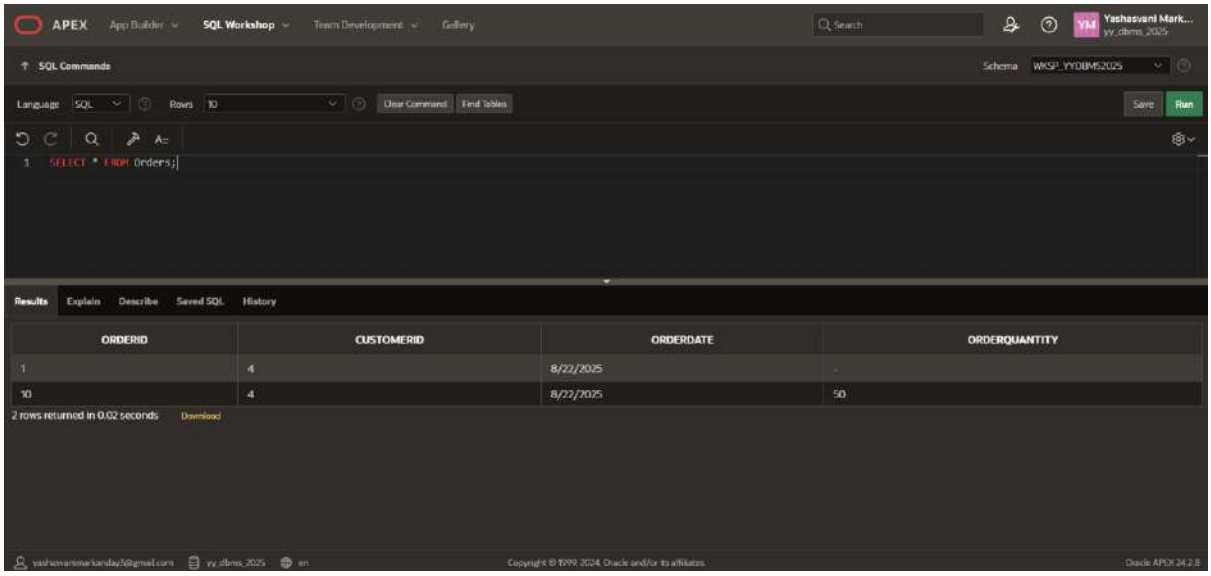


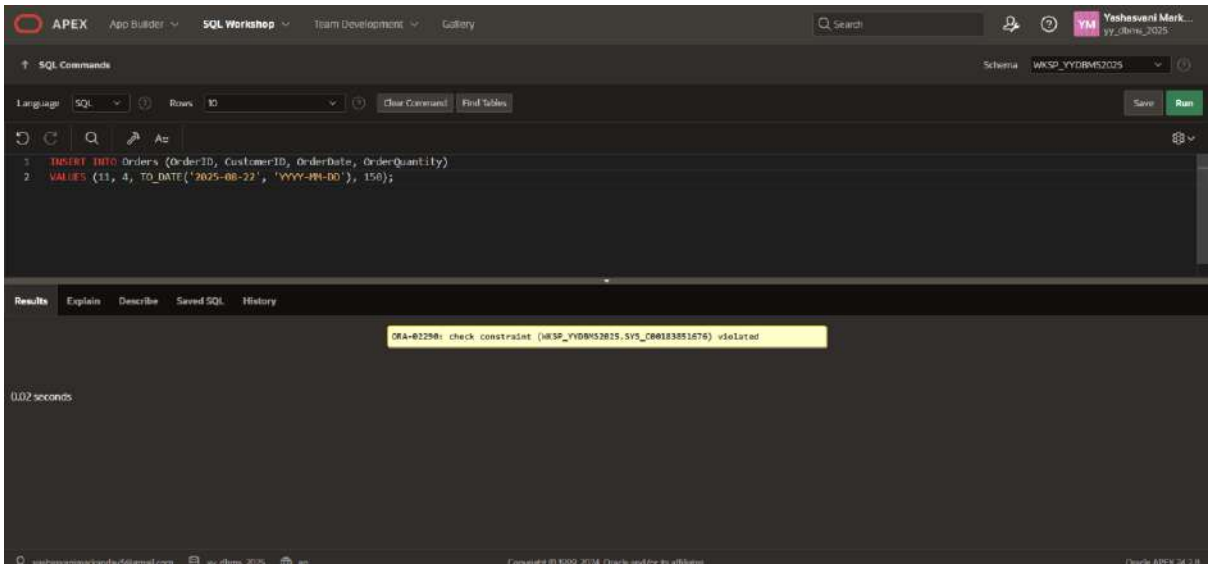




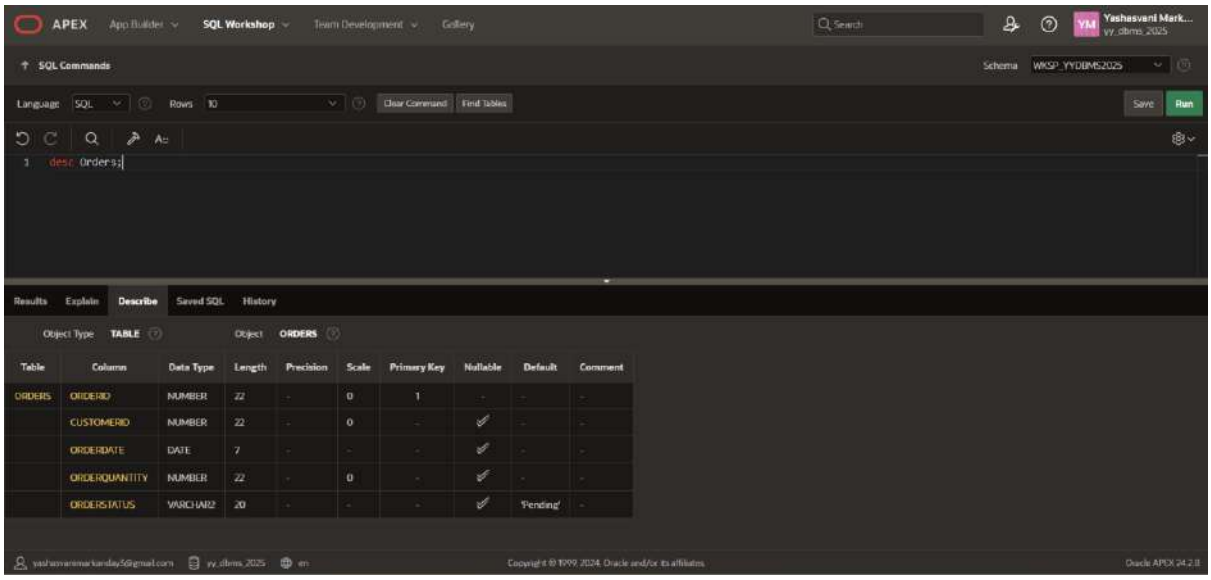
7. Add a column named `OrderQuantity` to the `Orders` table. Apply a CHECK constraint to limit the order quantity to a range between 1 and 100.

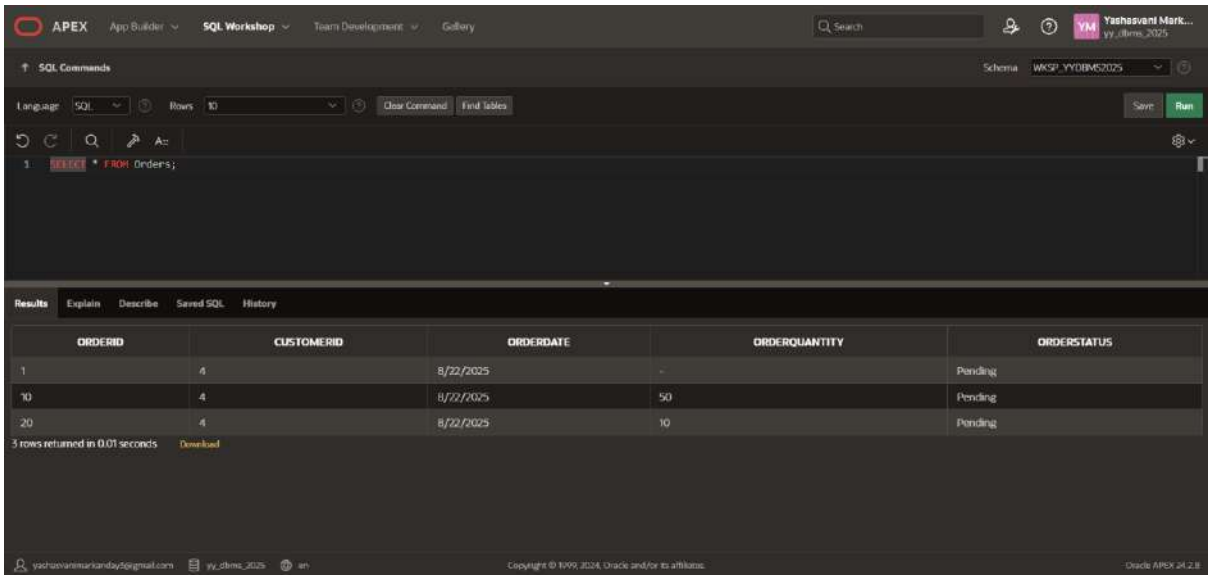


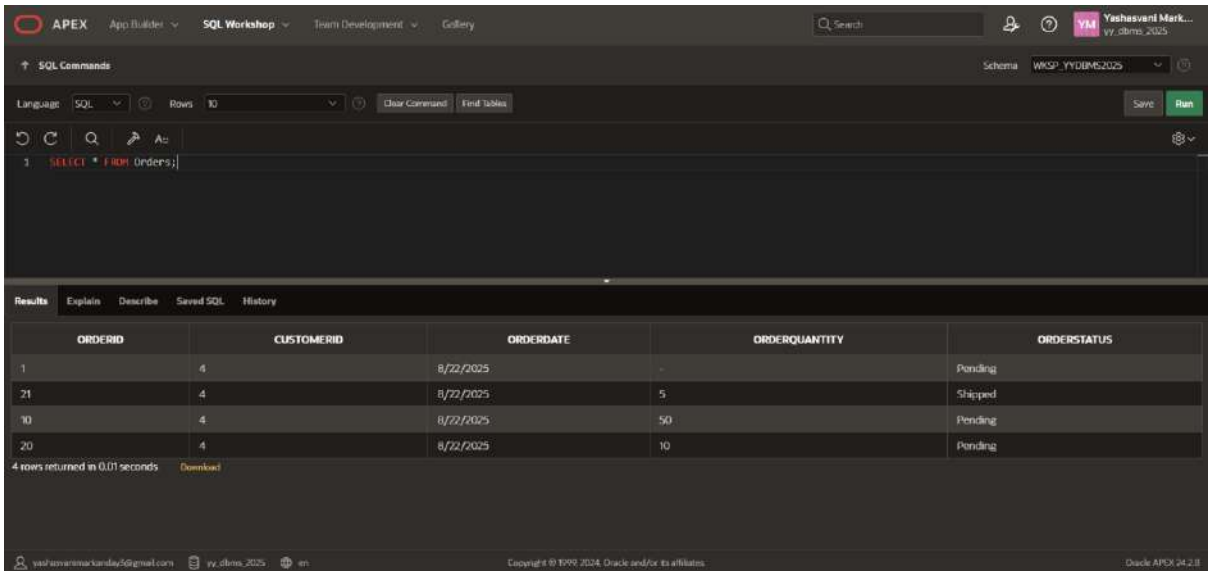

APEX
App Builder
SQL Workshop
Team Development
Gallery
Search
Yashasvani Mark...
yy_dbms_2025
SQL Commands
Schema
WKSP_YYDBMS2025
Language
SQL
Rows
10
Clear Command
Find Tables
Save
Run
1 SELECT * FROM Orders;
Results
Explain
Describe
Saved SQL
History
ORDERID
CUSTOMERID
ORDERDATE
ORDERQUANTITY
1
4
8/22/2025
10
4
8/22/2025
50
2 rows returned in 0.02 seconds
Download
yashasvanimarkanday5@gmail.com
yy_dbms_2025
en
Copyright © 1999, 2024, Oracle and/or its affiliates
Oracle APEX 24.2.8


APEX
App Builder
SQL Workshop
Team Development
Gallery
Search
Yashasvani Mark...
yy_dbms_2025
SQL Commands
Schema
WKSP_YYDBMS2025
Language
SQL
Rows
10
Clear Command
Find Tables
Save
Run
1 INSERT INTO Orders (OrderID, CustomerID, OrderDate, OrderQuantity)
2 VALUES (11, 4, TO_DATE('2025-08-22', 'YYYY-MM-DD'), 150);
Results
Explain
Describe
Saved SQL
History
ORA-02290: check constraint (WKSP_YYDBMS2025.SYS_C0013851676) violated
0.02 seconds

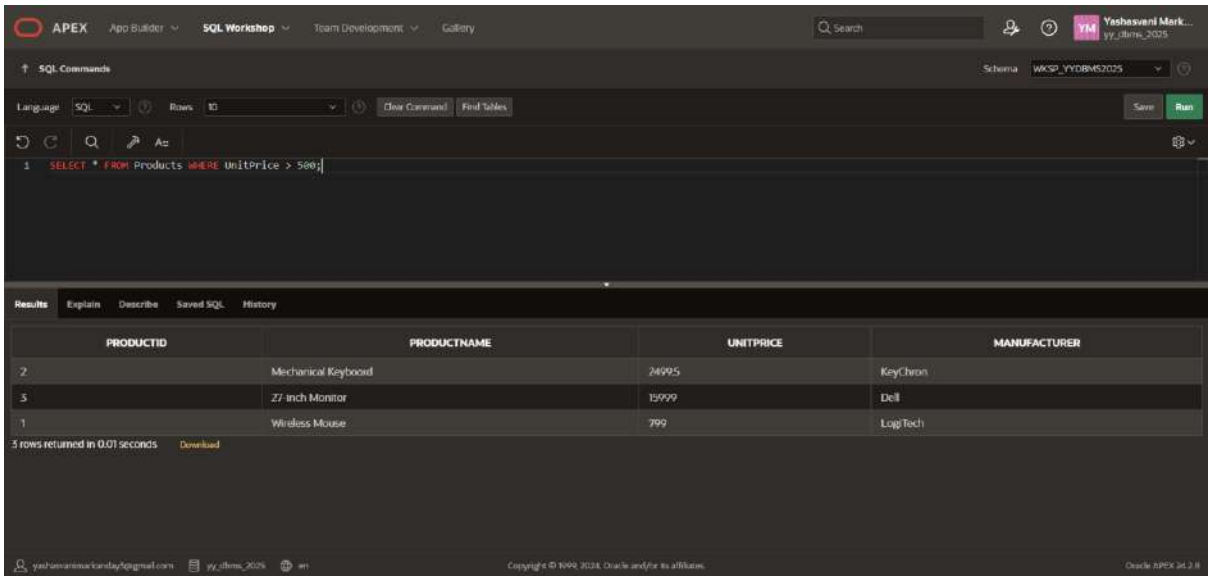
8. Include a column named `OrderStatus` in the `Orders` table. Set a DEFAULT constraint so that if the status is not specified during insertion, it defaults to 'Pending.'

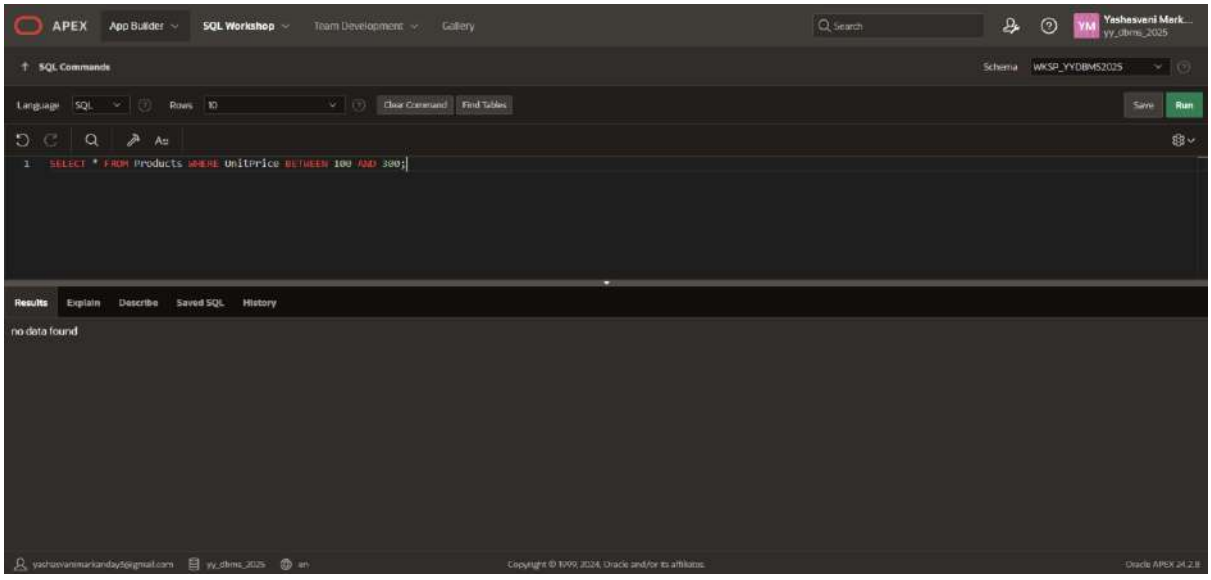




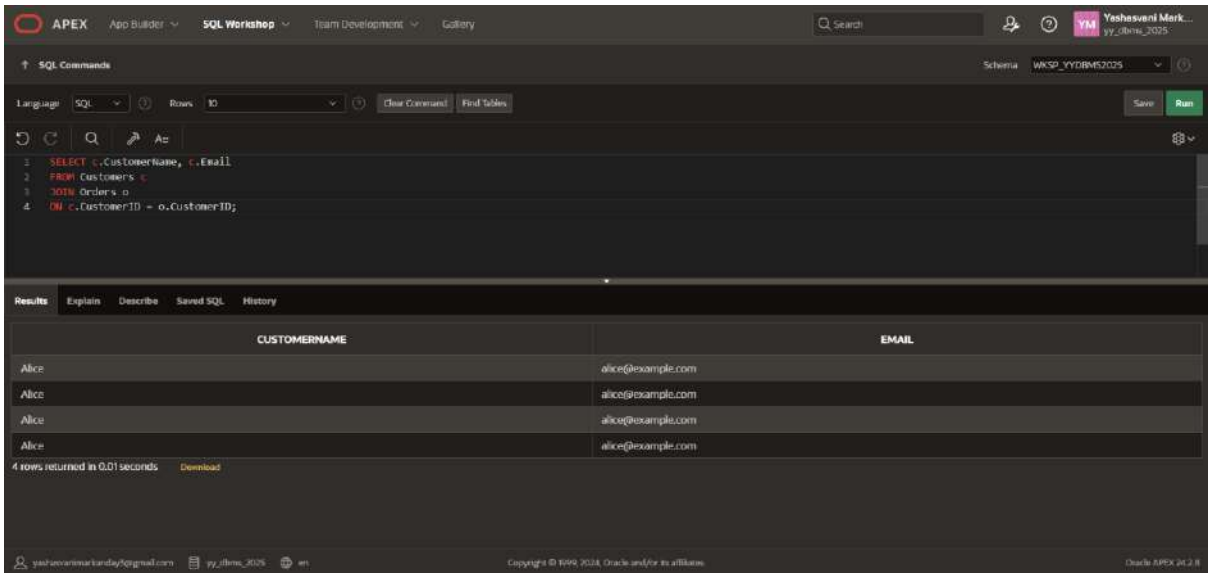


9. Create an index on the `UnitPrice` column in the `Products` table. Explain how this index can improve query performance.

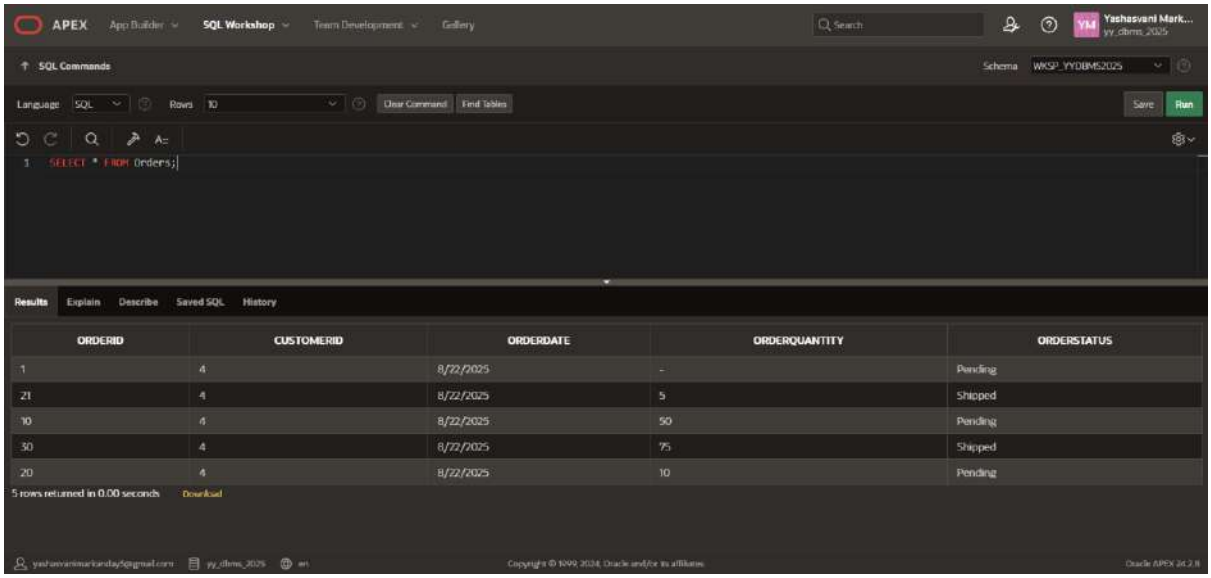




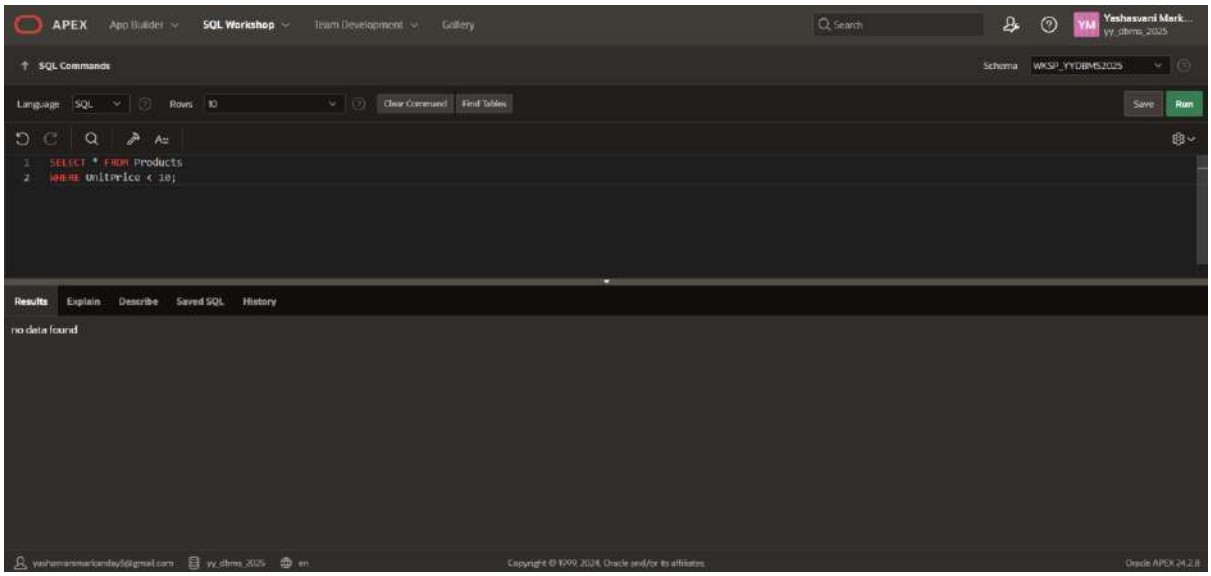
10. Write an SQL query to retrieve the names and email addresses of all customers who placed orders. Join the `Customers` and `Orders` tables to achieve this.



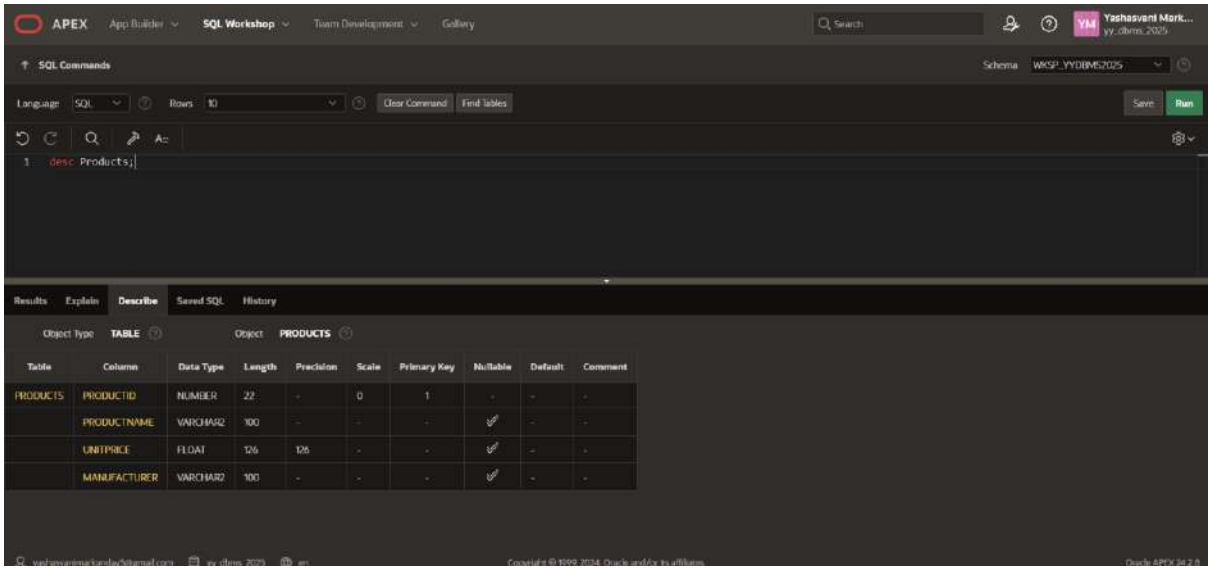
11. Modify the `OrderStatus` of all orders with a quantity greater than 50 to 'Shipped.' Provide the SQL statement for this update.



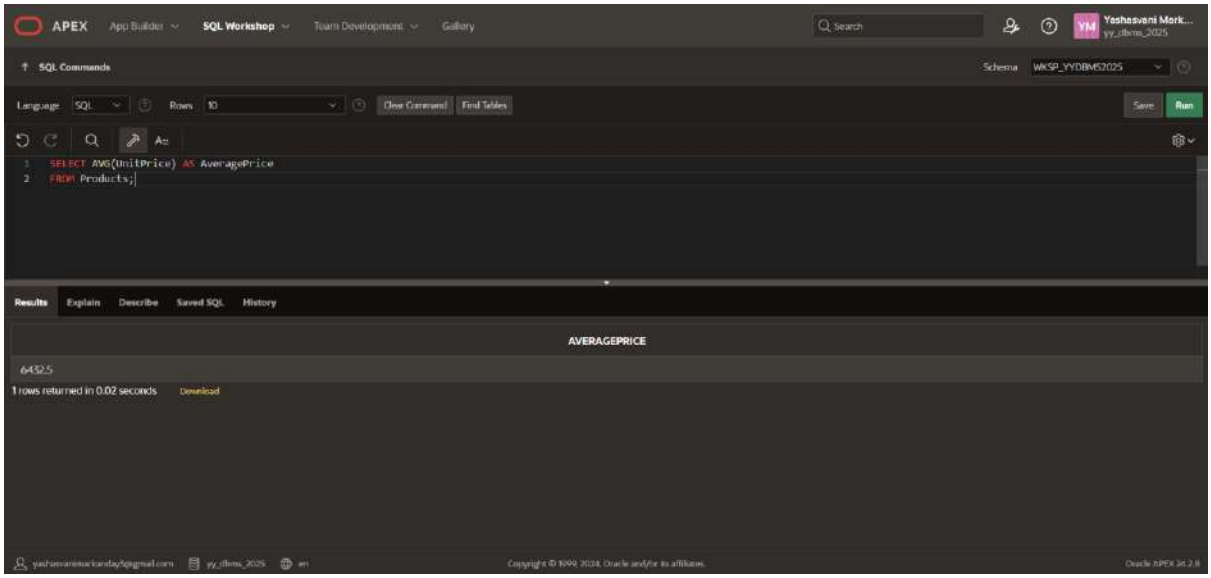
12. Delete all records from the `Products` table where the `UnitPrice` is less than $10. Include the SQL statement for this deletion.



13. The `UnitPrice` column in the `Products` table needs to be changed to a new datatype, `float`. Write an SQL statement using the ALTER TABLE statement to make this change.



14. Write an SQL query to calculate the average `UnitPrice` of all products in the `Products` table.



UNIVERSITY OF PETROLEUM AND ENERGY STUDIES

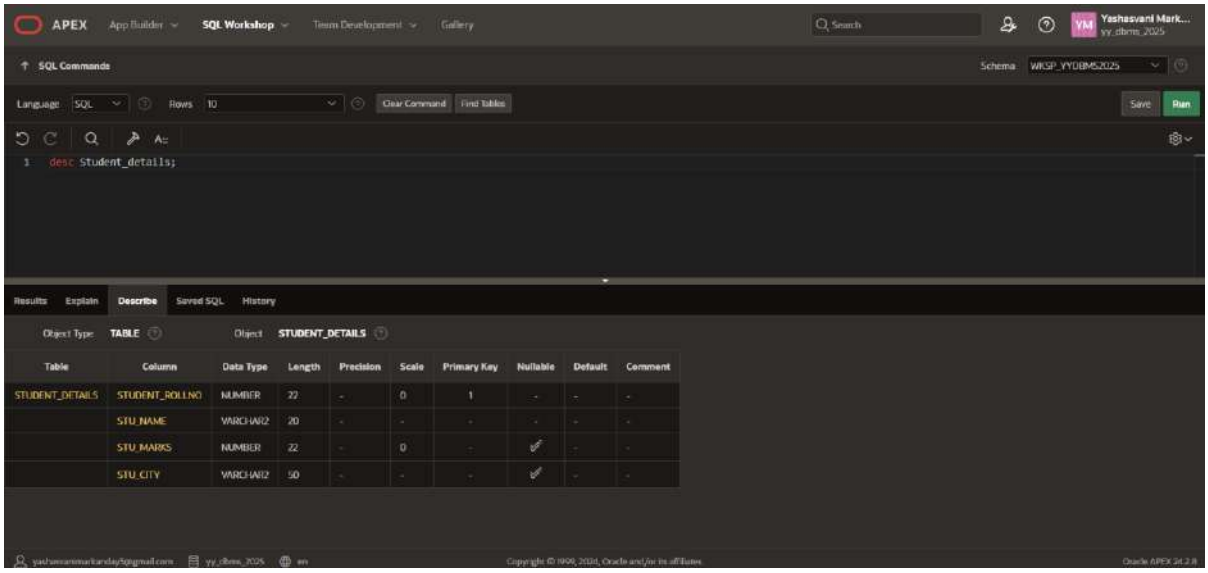
Program Name: B. Tech CS **Semester: III**

Name: Yashasvani Markanday **SAP ID: 590011386**

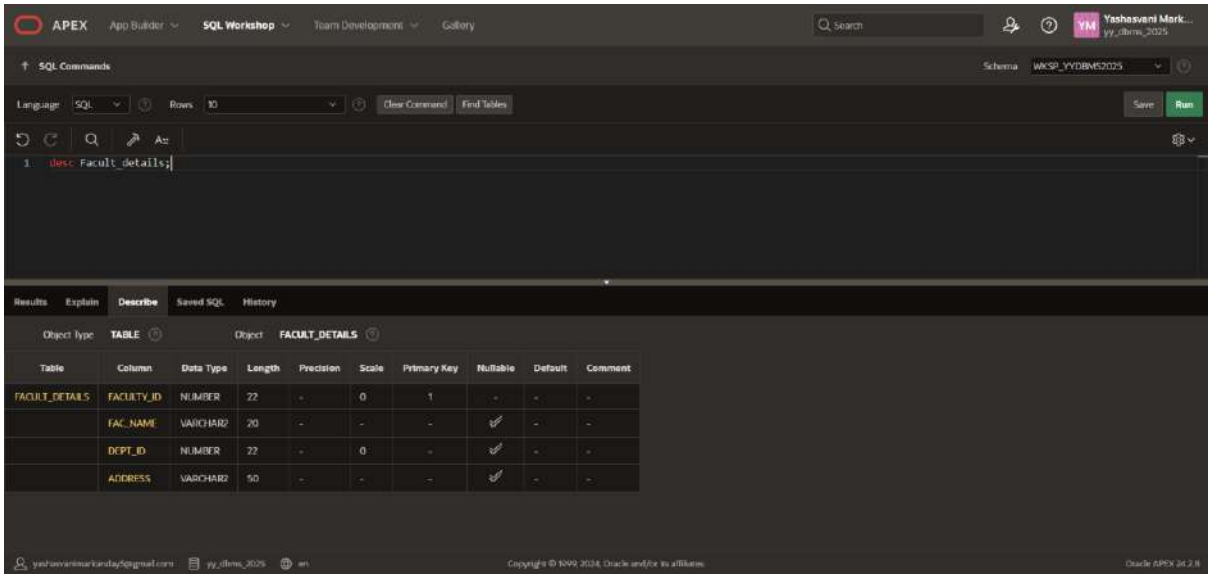
Batch: B2 **Subject: Database and Management System**

Tables:

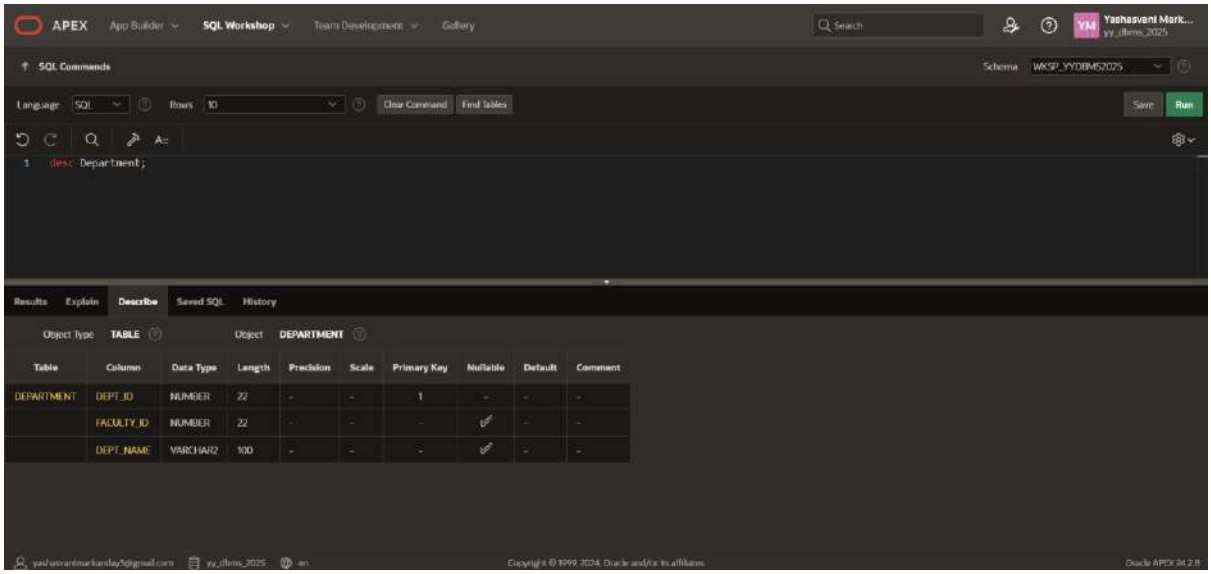
1.



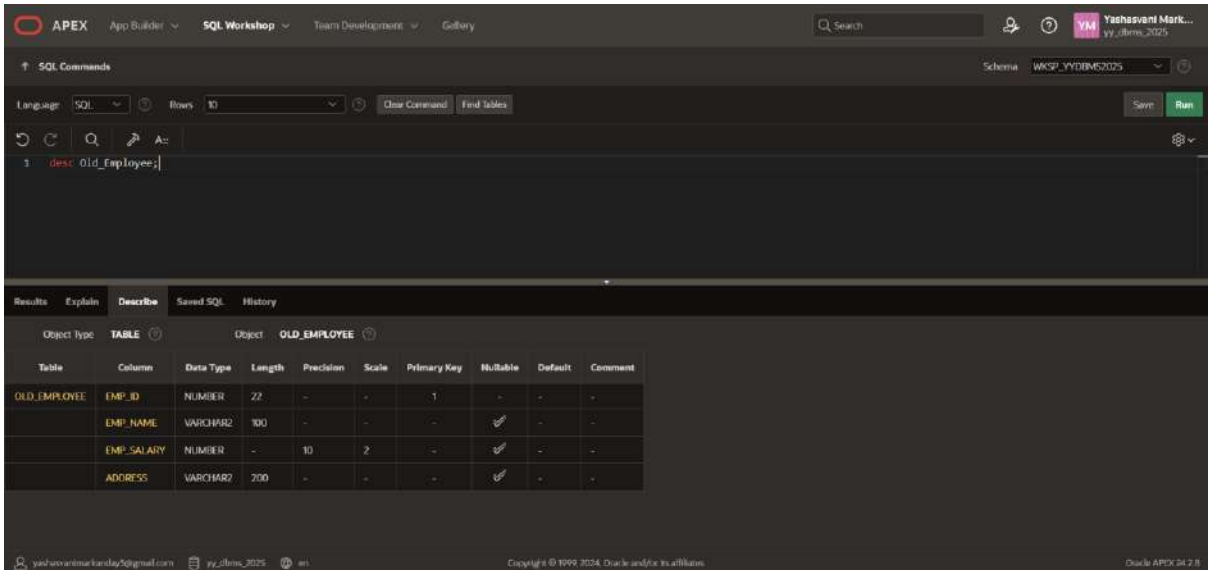
2.



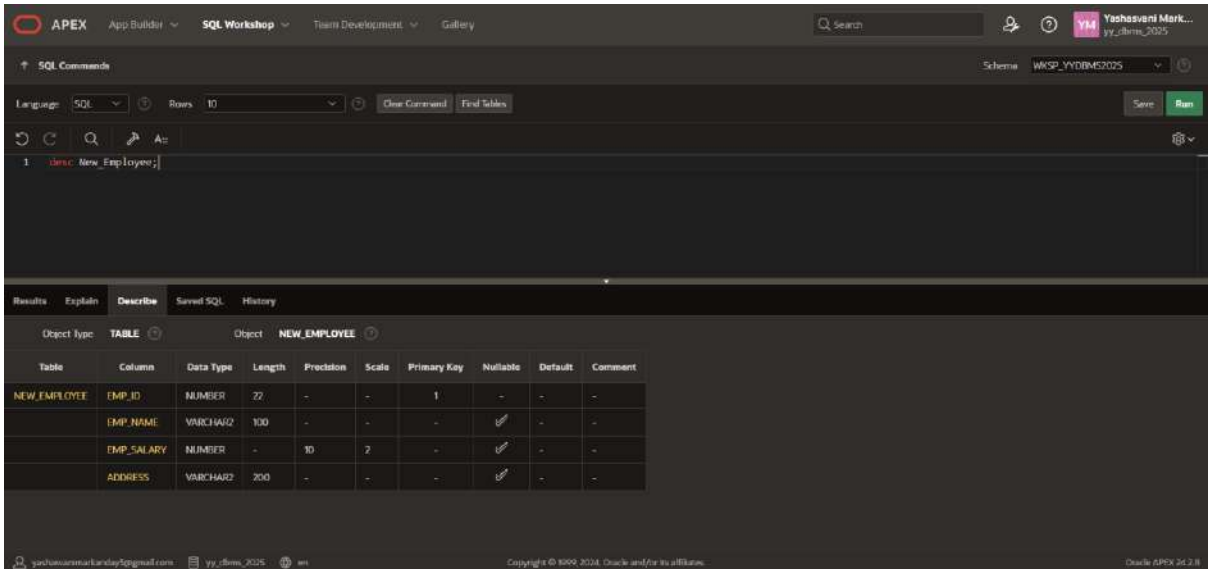
3.


APEX
App Builder
SQL Workshop
Team Development
Gallery
Search
Yashasvani Mark...
yy_dbms_2025
SQL Commands
Schema
WKSP_YYDBMS2025
Language
SQL
Rows
10
Clear Command
Find Tables
Save
Run
1 desc Department;
Results
Explain
Describe
Saved SQL
History
Object Type
TABLE
Object
DEPARTMENT
Table
Column
Data Type
Length
Precision
Scale
Primary Key
Nullable
Default
Comment
DEPARTMENT
DEPT_ID
NUMBER
22
1
FACULTY_ID
NUMBER
22
DEPT_NAME
VARCHAR2
100
Copyright © 1999, 2024, Oracle and/or its affiliates.
Oracle APEX 24.2.8

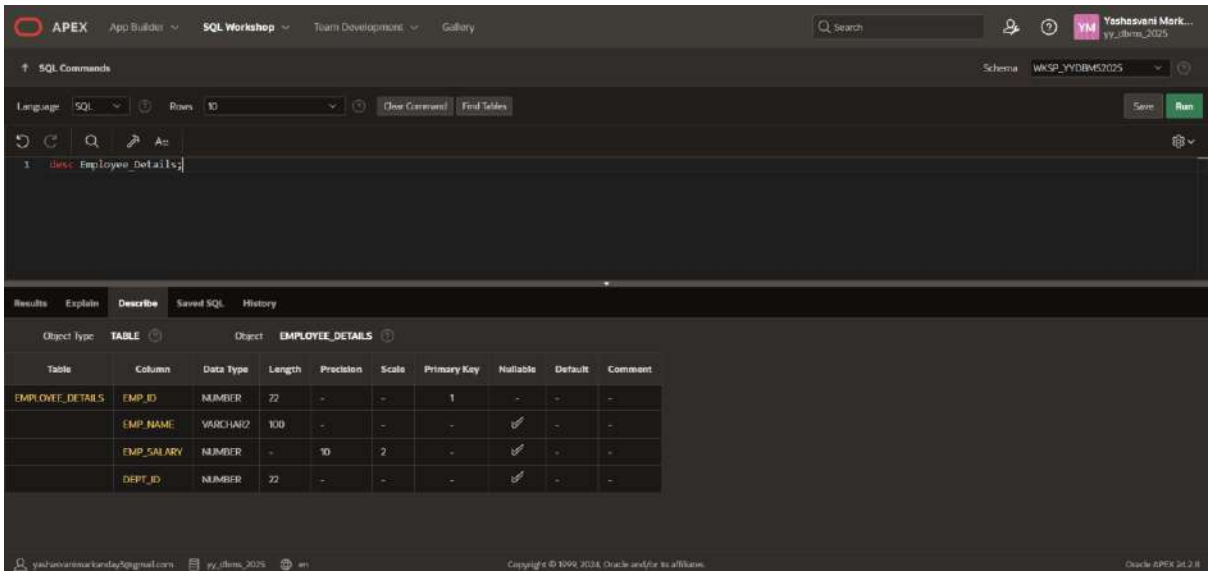
4.


APEX
App Builder
SQL Workshop
Team Development
Gallery
Search
Yashasvani Mark...
yy_dbms_2025
SQL Commands
Schema
WKSP_YYDBMS2025
Language
SQL
Rows
10
Clear Command
Find Tables
Save
Run
1 desc Old_Employee;
Results
Explain
Describe
Saved SQL
History
Object Type
TABLE
Object
OLD_EMPLOYEE
Table
Column
Data Type
Length
Precision
Scale
Primary Key
Nullable
Default
Comment
OLD_EMPLOYEE
EMP_ID
NUMBER
22
1
EMP_NAME
VARCHAR2
100
EMP_SALARY
NUMBER
10
2
ADDRESS
VARCHAR2
200
Copyright © 1999, 2024, Oracle and/or its affiliates.
Oracle APEX 24.2.8

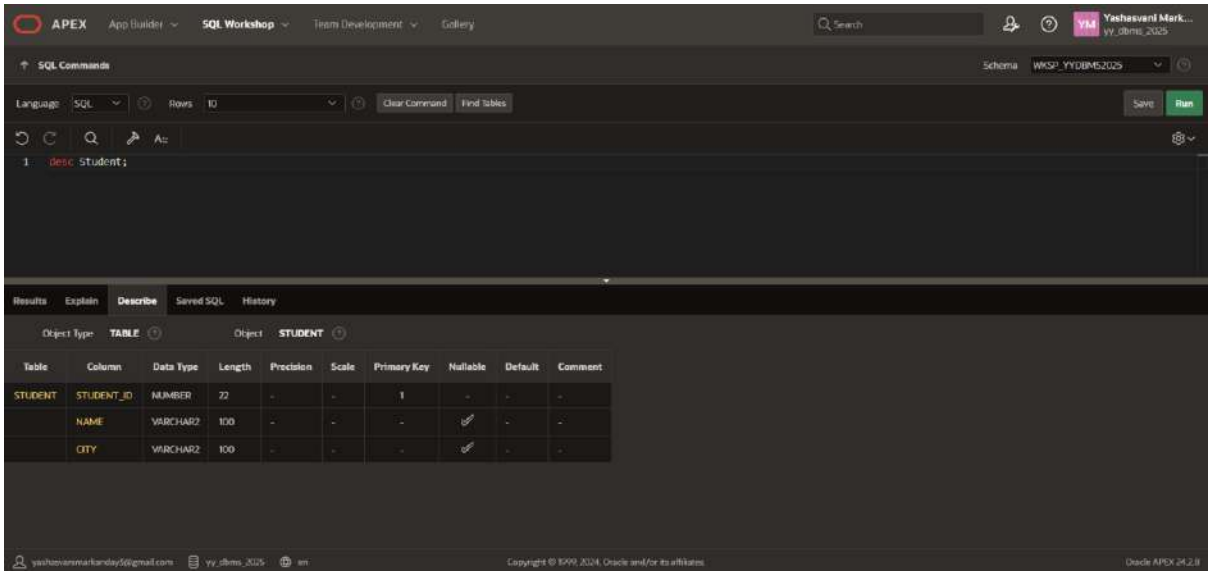
5.


APEX
App Builder
SQL Workshop
Team Development
Gallery
Search
Yashasvani Mark...
yy_dbms_2025
SQL Commands
Schema
WKSP_YYDBMS2025
Language
SQL
Rows
10
Clear Command
Find Tables
Save
Run
1 desc New_Employee;
Results
Explain
Describe
Saved SQL
History
Object Type
TABLE
Object
NEW_EMPLOYEE
Table
Column
Data Type
Length
Precision
Scale
Primary Key
Nullable
Default
Comment
NEW_EMPLOYEE
EMP_ID
NUMBER
22
1
EMP_NAME
VARCHAR2
100
EMP_SALARY
NUMBER
10
2
ADDRESS
VARCHAR2
200

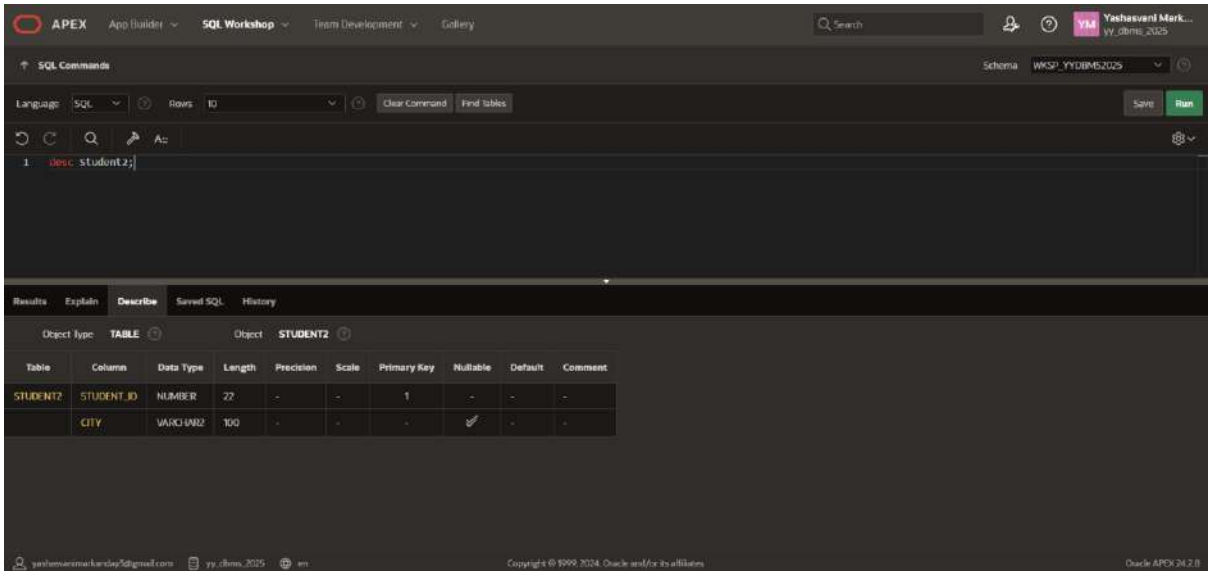
6.


APEX
App Builder
SQL Workshop
Team Development
Gallery
Search
Yashasvani Mark...
yy_dbms_2025
SQL Commands
Schema
WKSP_YYDBMS2025
Language
SQL
Rows
10
Clear Command
Find Tables
Save
Run
1 desc Employee_Details;
Results
Explain
Describe
Saved SQL
History
Object Type
TABLE
Object
EMPLOYEE_DETAILS
Table
Column
Data Type
Length
Precision
Scale
Primary Key
Nullable
Default
Comment
EMPLOYEE_DETAILS
EMP_ID
NUMBER
22
1
EMP_NAME
VARCHAR2
100
EMP_SALARY
NUMBER
10
2
DEPT_ID
NUMBER
22

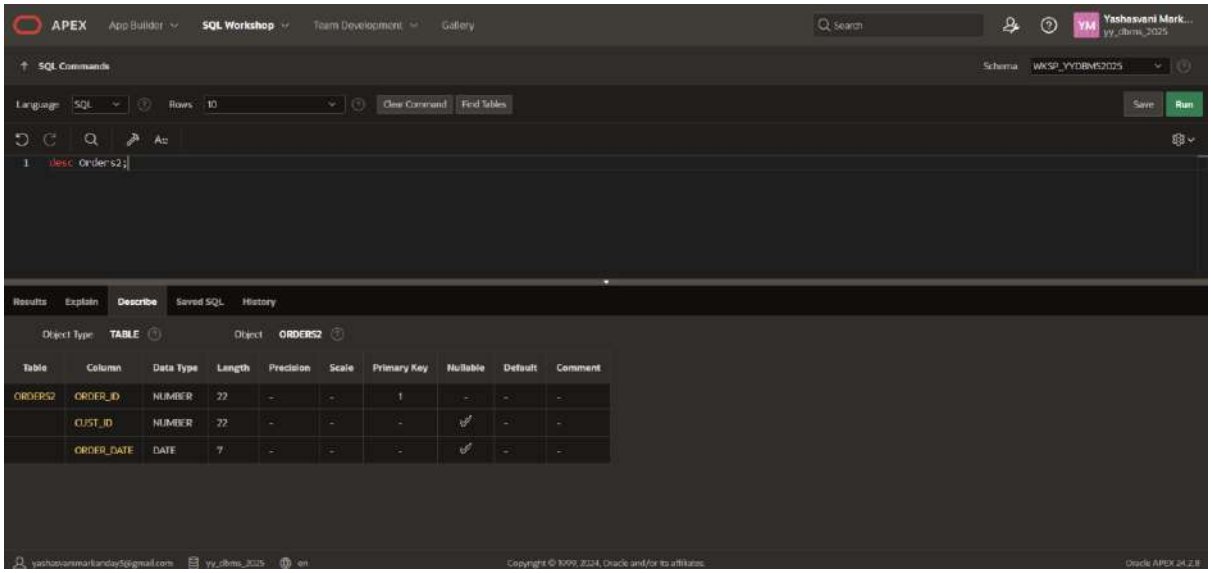
7.


APEX
App Builder
SQL Workshop
Team Development
Gallery
SQL Commands
Schema
WKSP_YYDBMS2025
Language
SQL
Rows
10
Clear Command
Find Tables
Save
Run
1 desc Student;
Results
Explain
Describe
Saved SQL
History
Object Type TABLE
Object STUDENT
Table
Column
Data Type
Length
Precision
Scale
Primary Key
Nullable
Default
Comment
STUDENT
STUDENT_ID
NUMBER
22
1
NAME
VARCHAR2
100
CITY
VARCHAR2
100

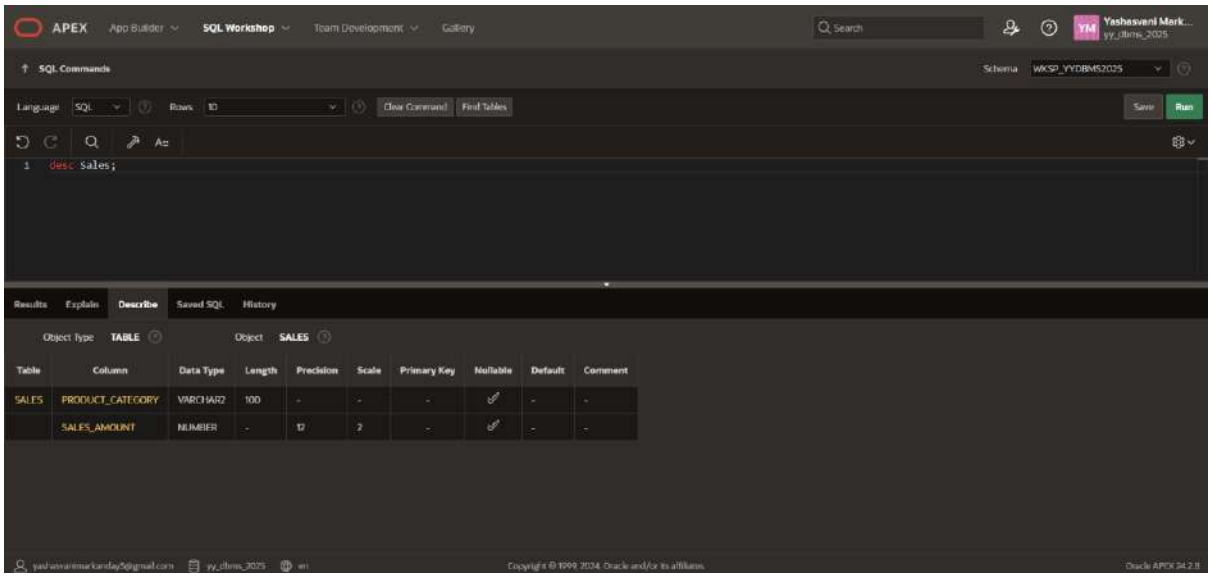
8.


APEX
App Builder
SQL Workshop
Team Development
Gallery
SQL Commands
Schema
WKSP_YYDBMS2025
Language
SQL
Rows
10
Clear Command
Find Tables
Save
Run
1 desc student2;
Results
Explain
Describe
Saved SQL
History
Object Type TABLE
Object STUDENT2
Table
Column
Data Type
Length
Precision
Scale
Primary Key
Nullable
Default
Comment
STUDENT2
STUDENT_ID
NUMBER
22
1
CITY
VARCHAR2
100

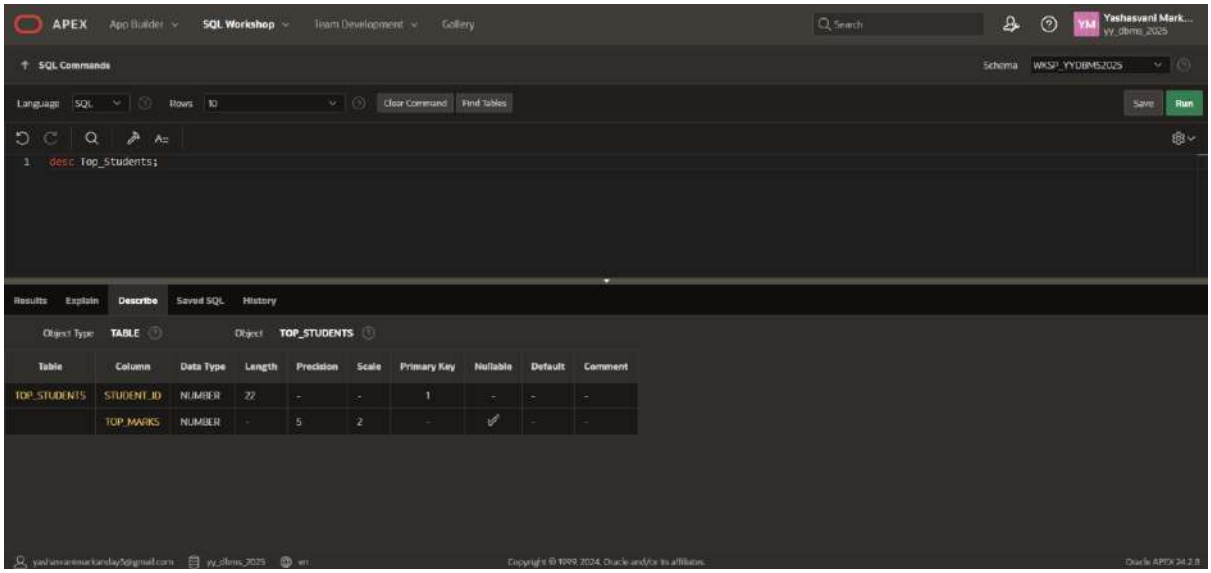
9.


APEX
App Builder
SQL Workshop
Team Development
Gallery
SQL Commands
Schema
WKSP_YYDBMS2025
Language
SQL
Rows
10
Clear Command
Find Tables
Save
Run
1 desc Orders2;
Results
Explain
Describe
Saved SQL
History
Object Type TABLE
Object ORDERS2
Table
Column
Data Type
Length
Precision
Scale
Primary Key
Nullable
Default
Comment
ORDERS2
ORDER_ID
NUMBER
22
1
CUST_ID
NUMBER
22
ORDER_DATE
DATE
7

10.


APEX
App Builder
SQL Workshop
Team Development
Gallery
SQL Commands
Schema
WKSP_YYDBMS2025
Language
SQL
Rows
10
Clear Command
Find Tables
Save
Run
1 desc Sales;
Results
Explain
Describe
Saved SQL
History
Object Type TABLE
Object SALES
Table
Column
Data Type
Length
Precision
Scale
Primary Key
Nullable
Default
Comment
SALES
PRODUCT_CATEGORY
VARCHAR2
100
SALES_AMOUNT
NUMBER
12
2

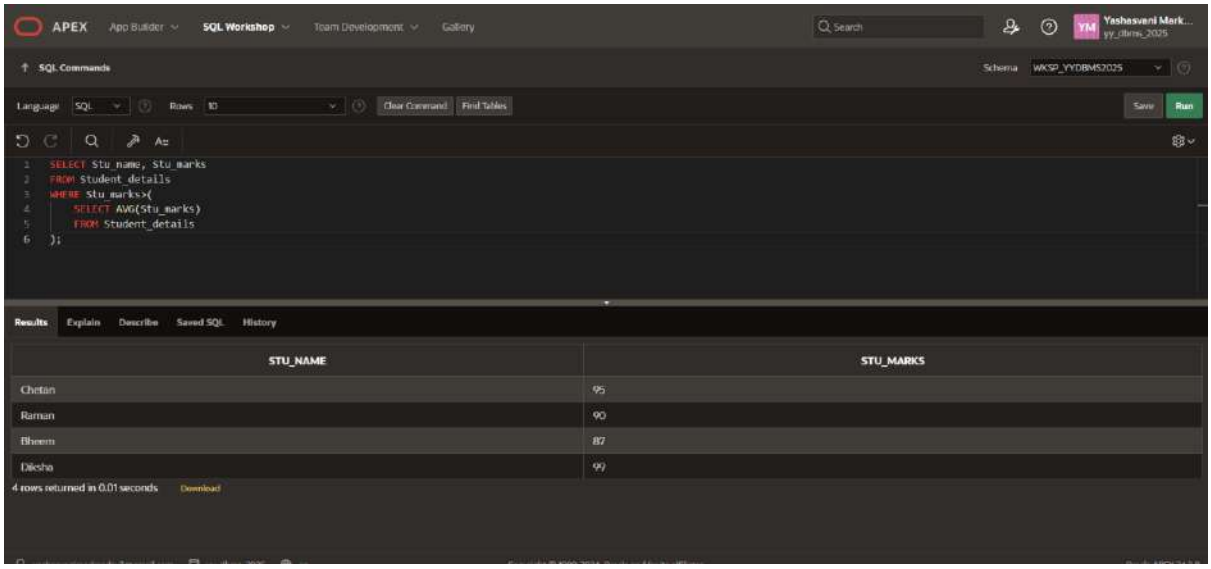
11.



SQL Subquery Questions:

1.Basic **SELECT** Subquery

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

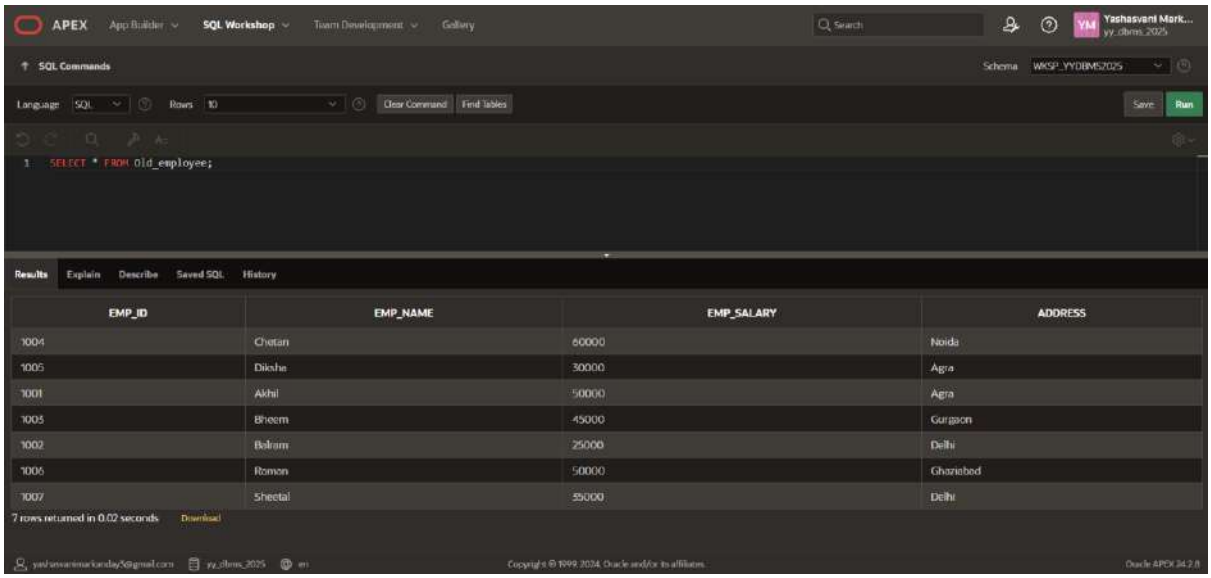


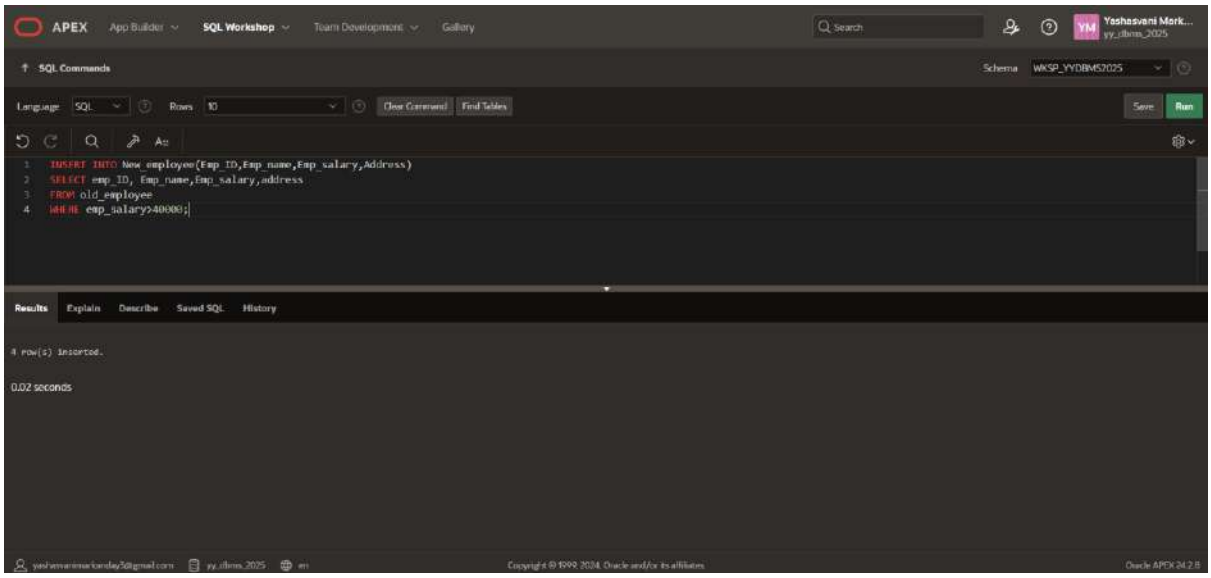
2. IN Operator Subquery

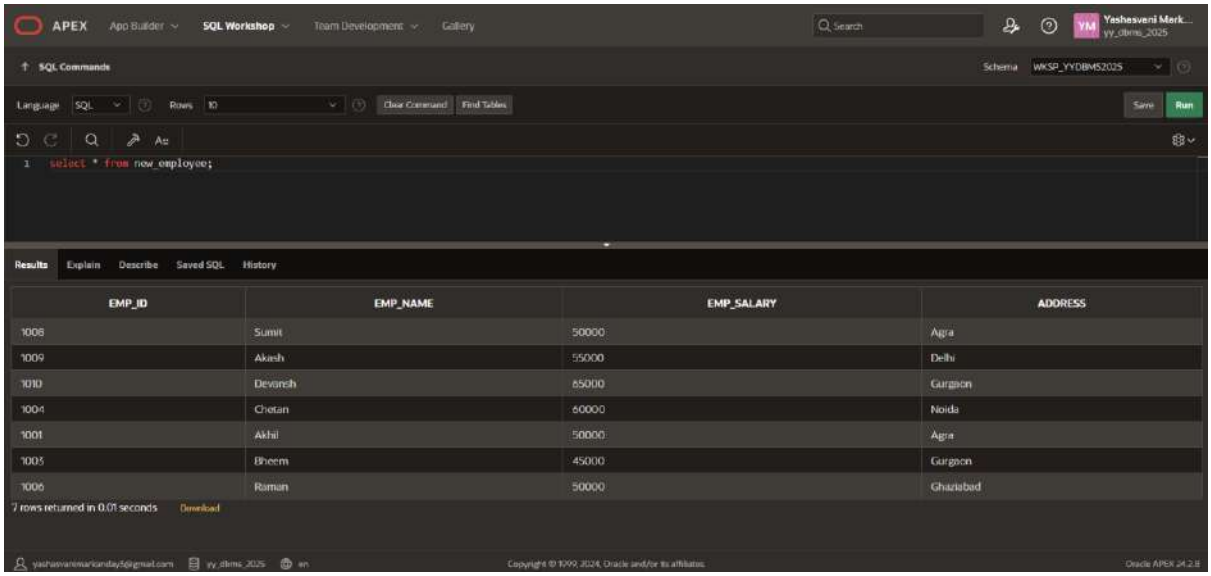
Fetch the names and addresses of faculties who belong to departments in either 'Noida' or ‘Gurgaon.’

3. **INSERT** with Subquery

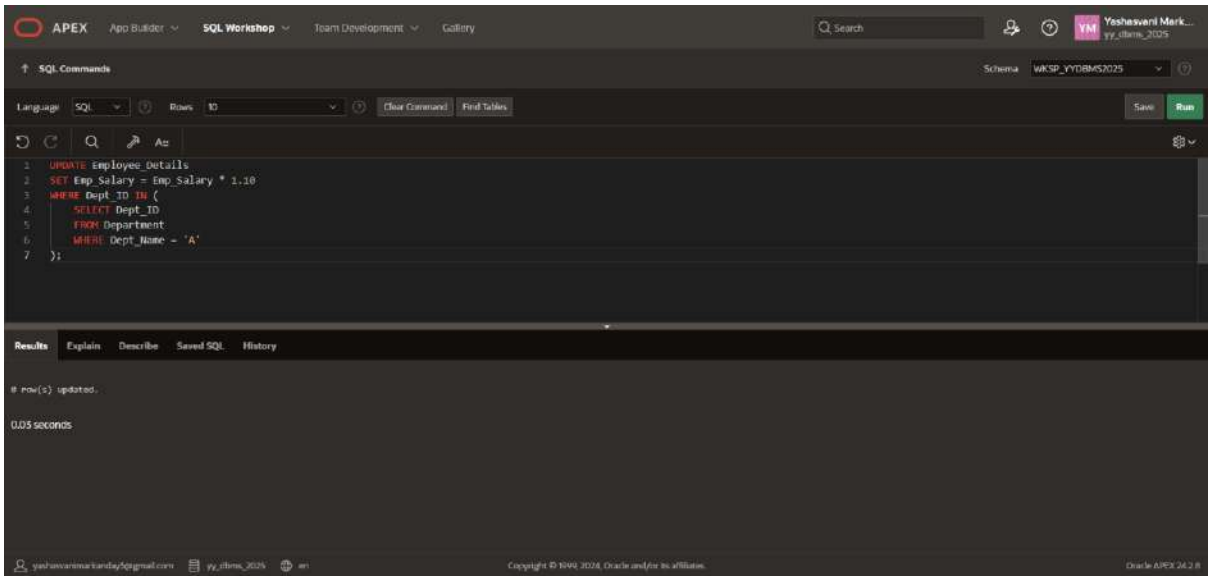
- Insert the details of employees from the "Old_Employee" table into the "New_Employee" table, considering only those with a salary greater than 40000.

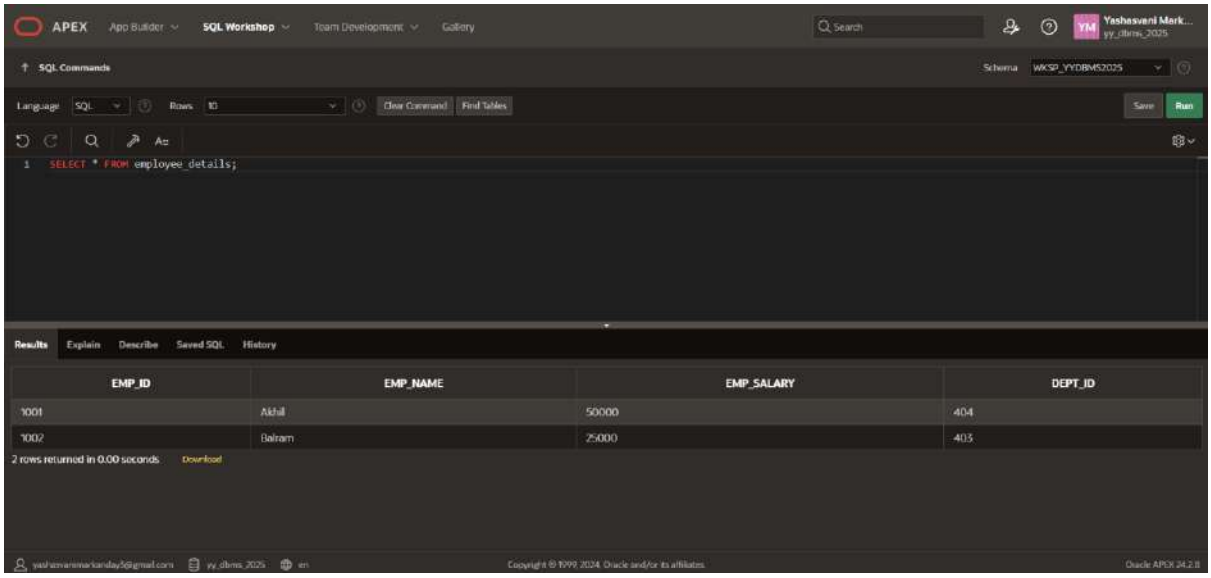




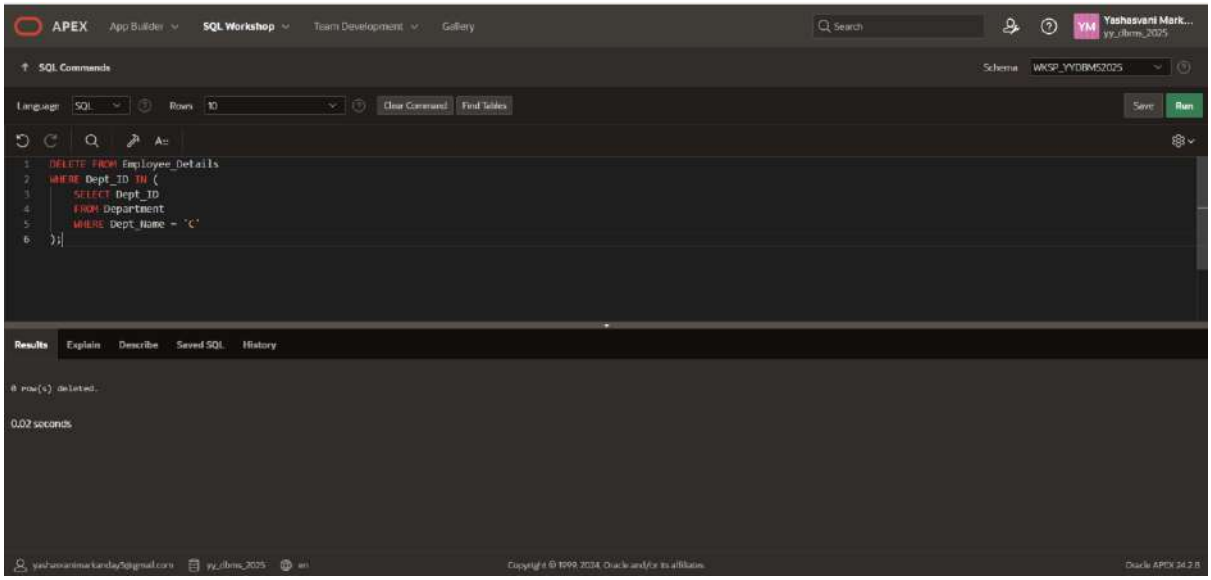


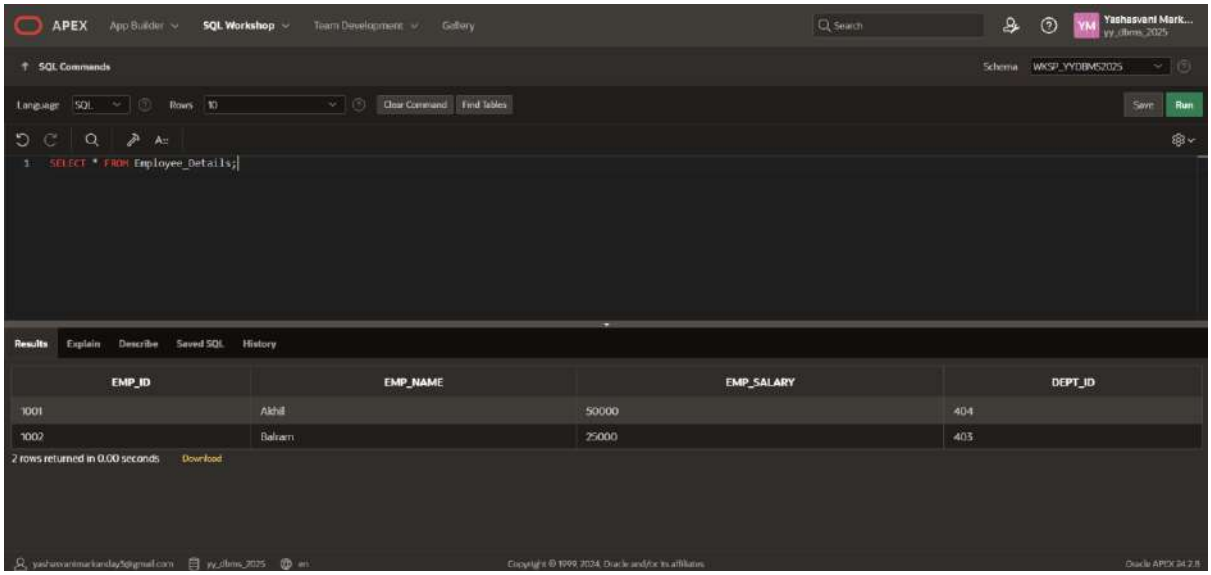
4. UPDATE with Subquery: - Increase the salary of employees by 10% for those whose department grade is 'A.'





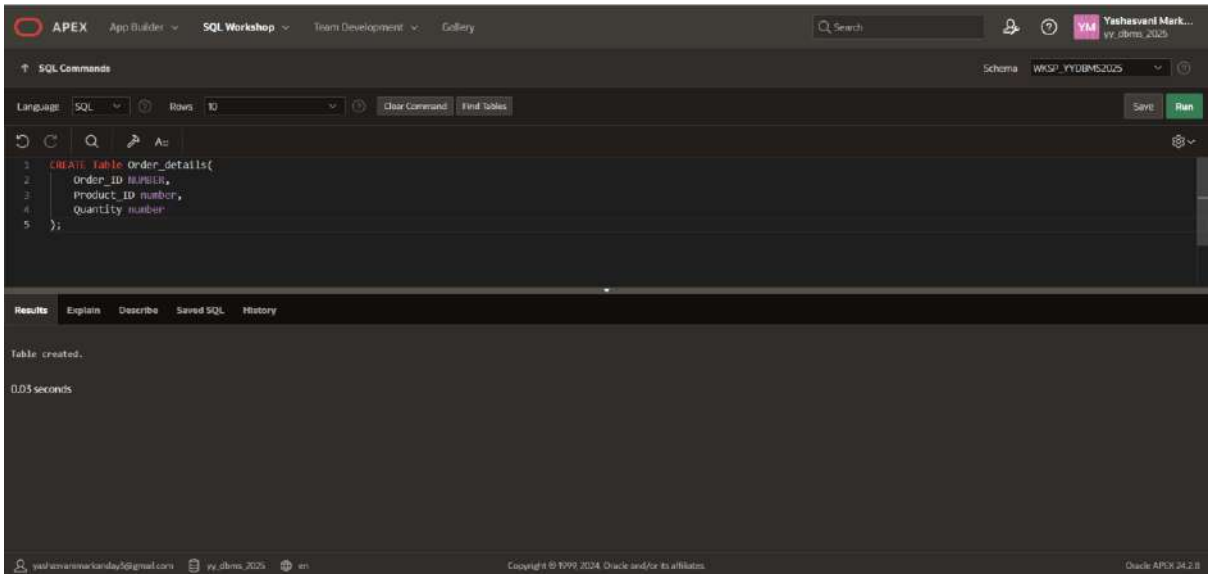
5. DELETE with Subquery: - Delete the records of employees whose department grade is 'C.'

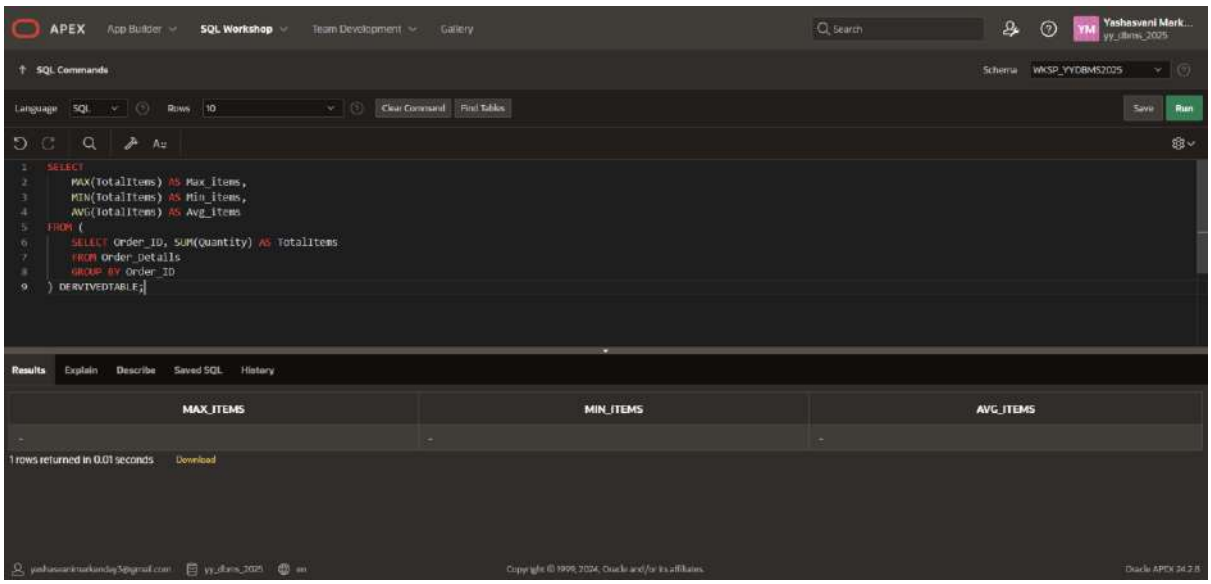




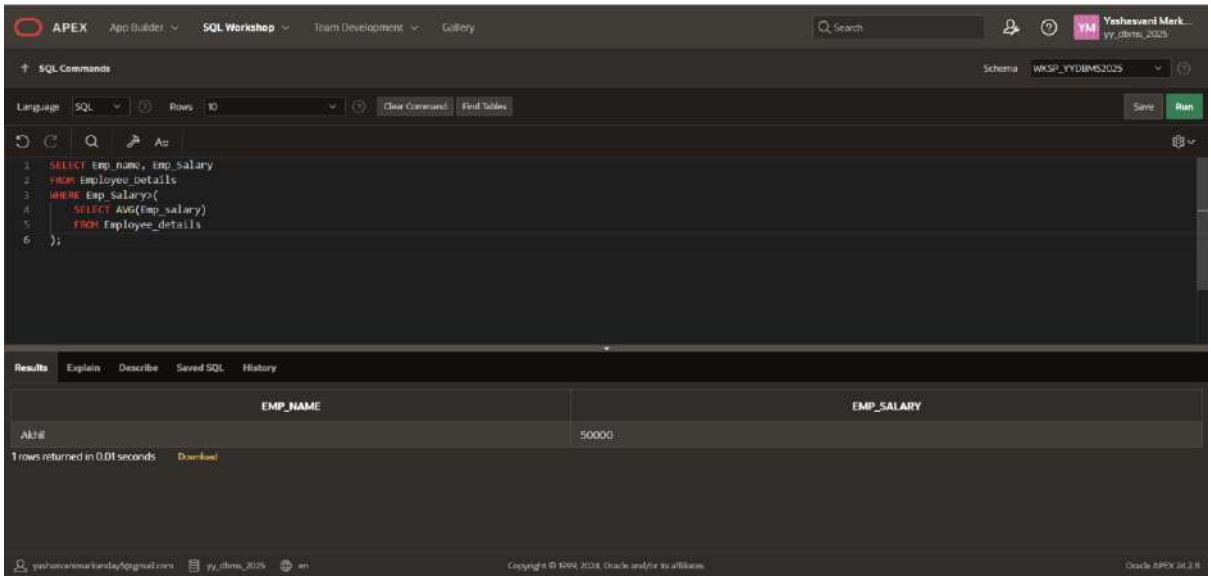
6. Subquery in WHERE Clause - NOT IN - Retrieve the names of students who do not belong to the city 'Los Angeles' based on data from the "Student2" table.

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.

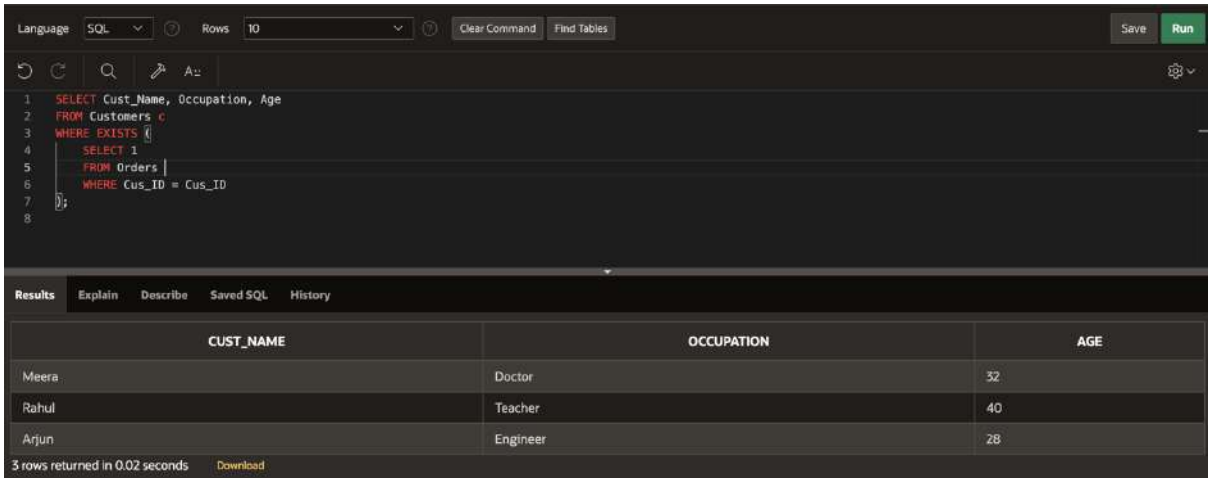




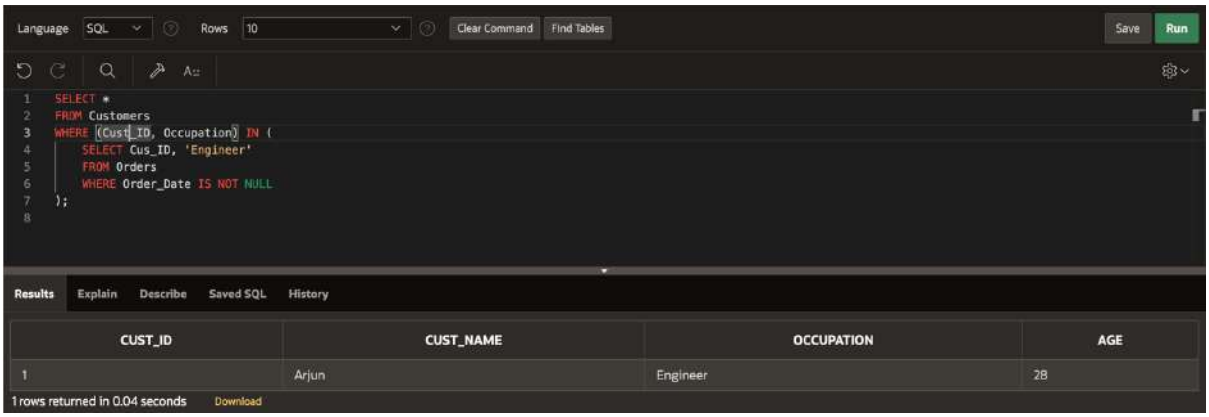
8. Correlated Subquery - Fetch the names, cities, and incomes of employees whose income is higher than the average income of employees in their respective cities.



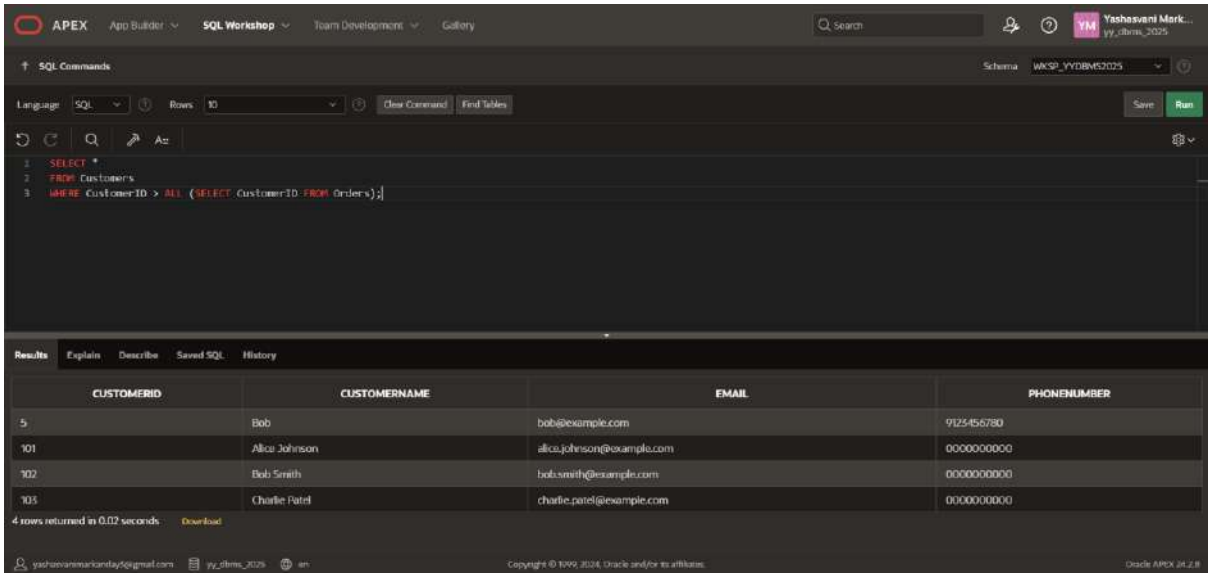
9. EXISTS Operator - Retrieve the name, occupation, and age of customers who have placed at least one order using the EXISTS operator.



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

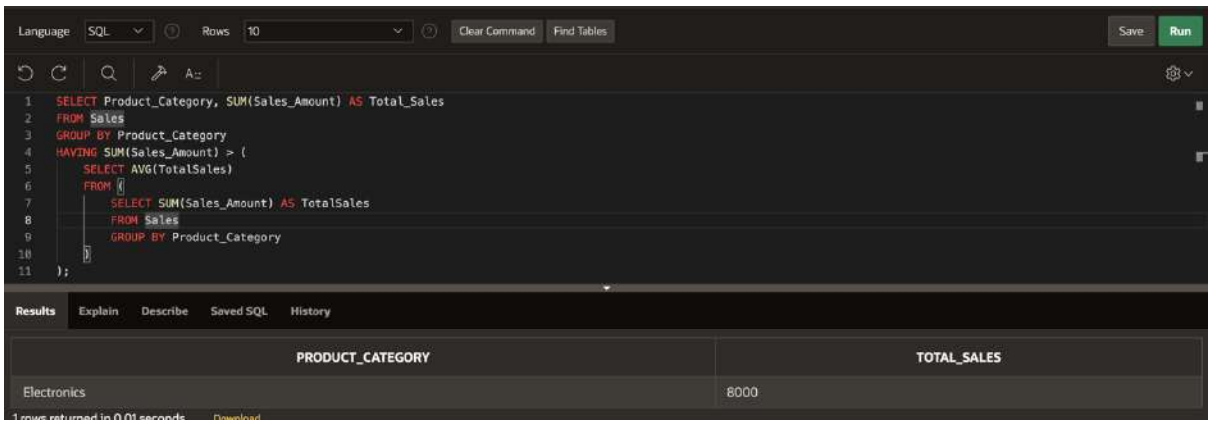


11. Subquery with ALL Operator - Find customers whose cust_id is greater than all cust_ids in the "Orders" table.

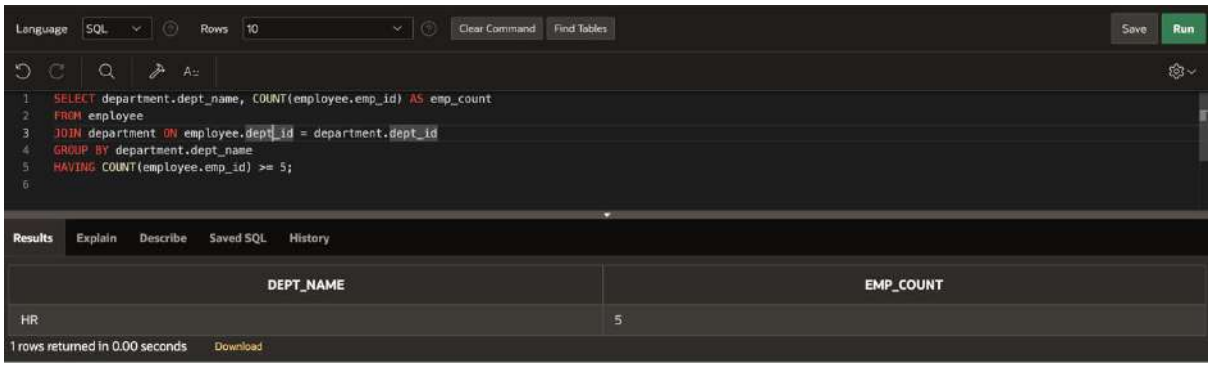


12. Nested Subqueries - Write an SQL query using a subquery within another subquery to retrieve names and salaries of employees who earn more than the average salary of employees in the "IT" department.

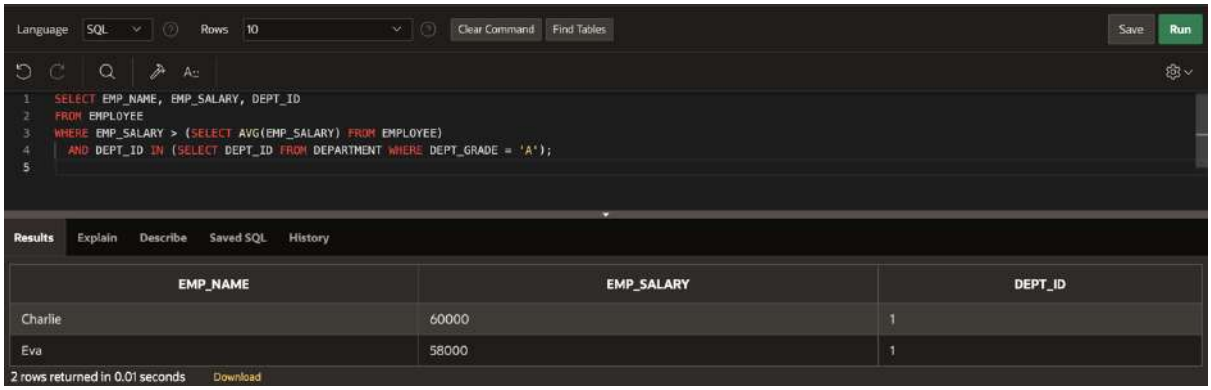
13. Subquery in HAVING Clause - Using the "Sales" table, find the total sales amount for each product category, displaying only those categories where the total sales amount is greater than the average total sales amount.



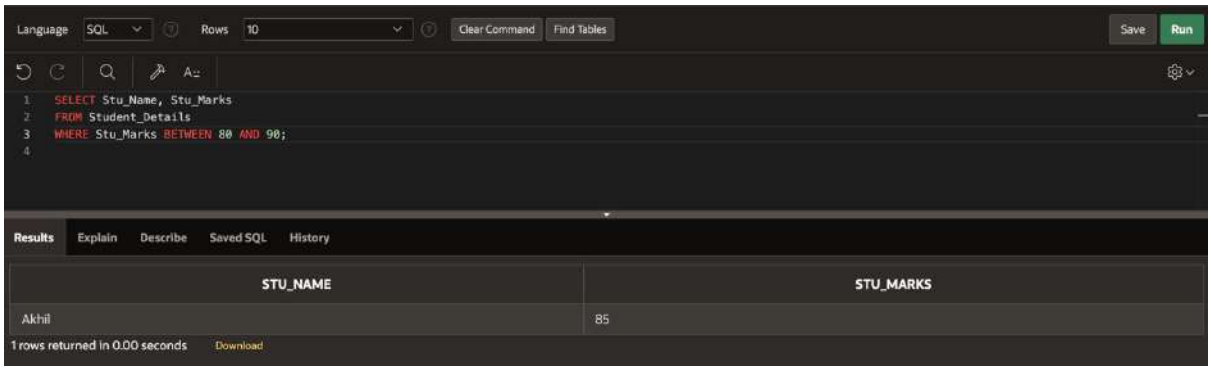
14. Subquery with GROUP BY - Retrieve department names and the count of employees in each department, filtering out departments where the count is less than 5.



15. Multiple Subqueries in a Single Query - Write an SQL query involving multiple subqueries to retrieve information about employees based on various conditions.



16. Subquery with BETWEEN Operators - Find the names and marks of students who scored 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 Emp_ID, Emp_Name, Emp_Salary, Address
FROM Old_Employee
WHERE Emp_Salary > (SELECT AVG(Emp_Salary) FROM Old_Employee);
```

Results | Explain | Describe | Saved SQL | History

2 row(s) inserted.

0.03 seconds

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

Results | Explain | Describe | Saved SQL | History

STU_NAME	STU_MARKS
Rahul	91

1 rows returned in 0.02 seconds Download

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_Detail
SET Emp_Salary = Emp_Salary * 1.05
WHERE Dep_ID = (SELECT Dept_ID FROM Department WHERE Dept_Name = 'HR')
  AND Emp_Salary < (SELECT AVG(Emp_Salary)
                    FROM Employee_Detail
                    WHERE Dep_ID = (SELECT Dept_ID FROM Department WHERE Dept_Name = 'HR'));
```

Results | Explain | Describe | Saved SQL | History

0 row(s) updated.

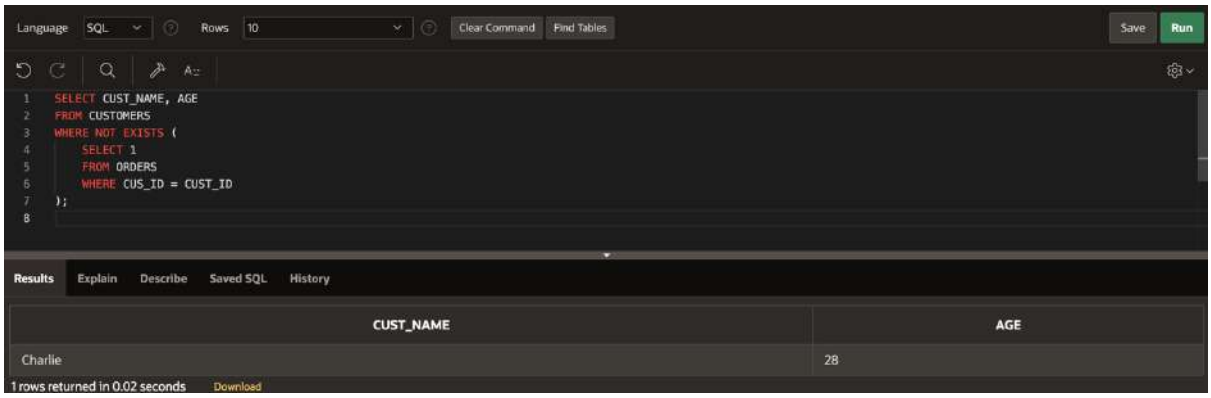
0.02 seconds

```
select * from employee_Detail
```

Results | Explain | Describe | Saved SQL | History

EMP_ID	EMP_NAME	EMP_SALARY	DEP_ID	CITY
202	Anita	45000	102	-
201	Amit	55000	10	Delhi
201	Rahul	66550	101	-
202	Priya	48000	10	Delhi
203	Rohit	60000	20	Mumbai
205	Karan	70000	30	Bangalore
204	Neha	45000	20	Mumbai

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.



UNIVERSITY OF PETROLEUM AND ENERGY STUDIES

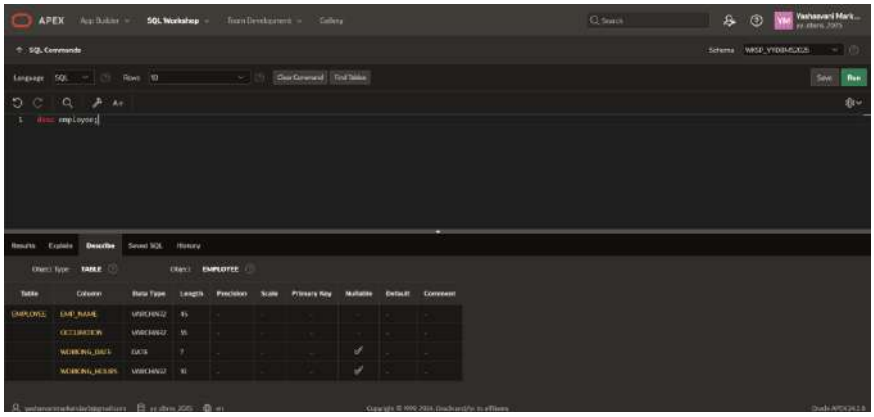
Program Name: B. Tech CS **Semester: III**

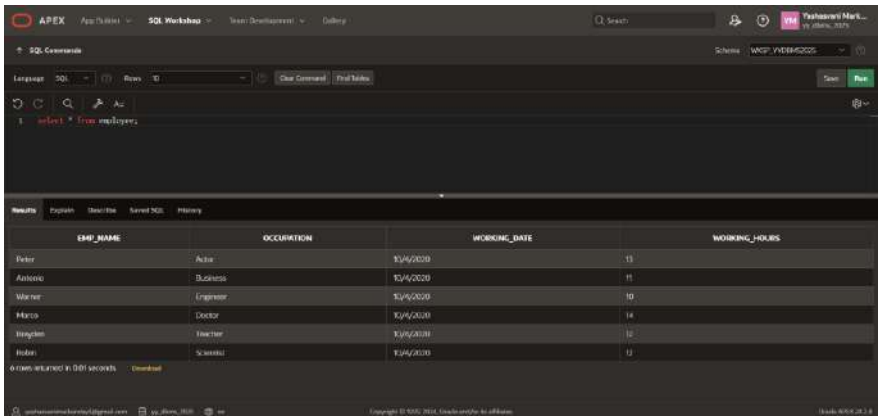
Name: Yashasvani Markanday **SAP ID: 590011386**

Batch: B2 **Subject: Database and Management System**

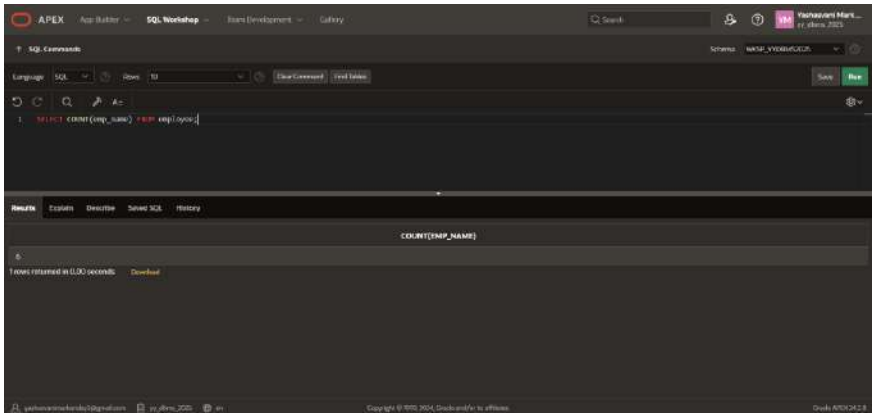
EXPERIMENT-4

Lab 4: To implement and use Inbuilt Functions Objective: To understand the use of SQL Inbuilt functions.

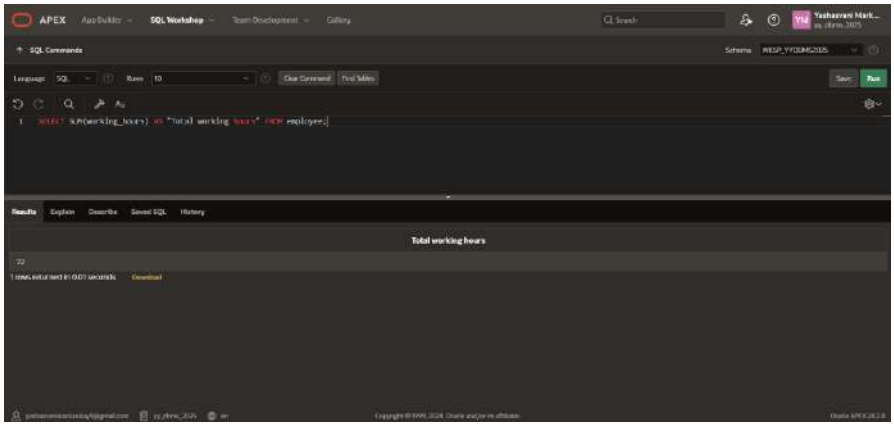




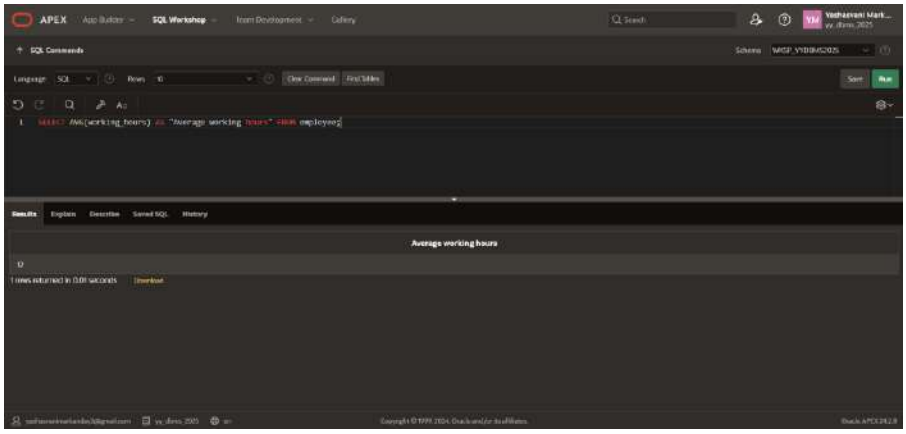
1. SELECT COUNT(name) FROM employee;



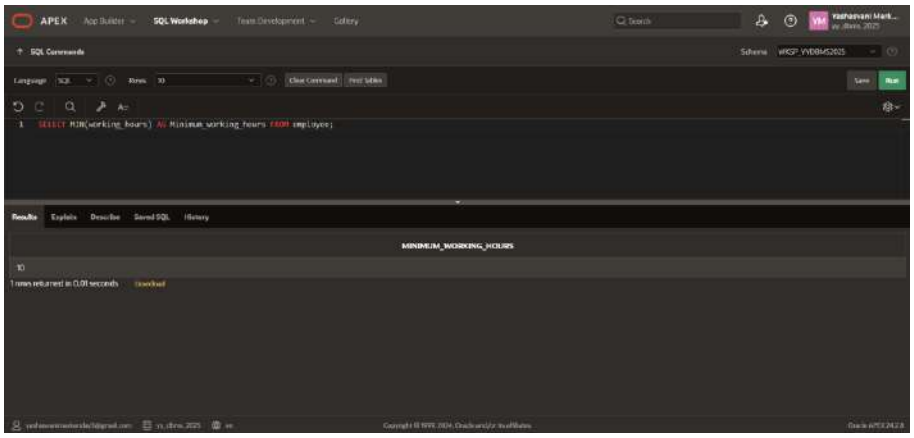
2. SELECT SUM(working_hours) AS "Total working hours" FROM employee;



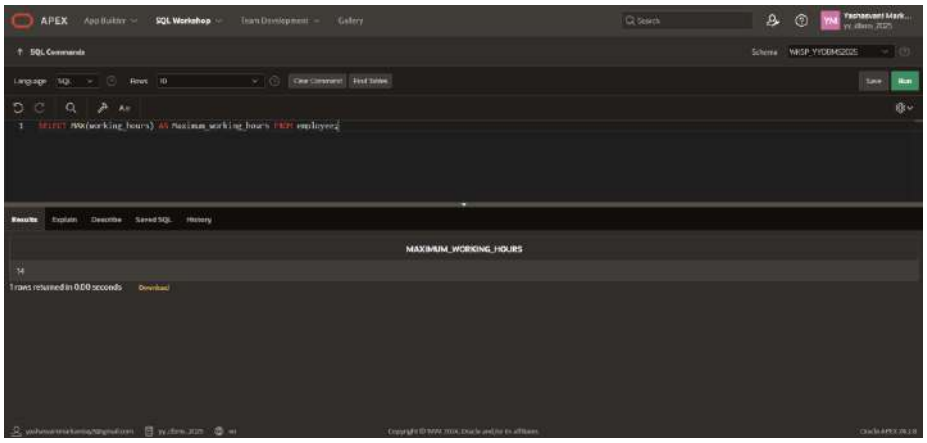
3. SELECT AVG(working_hours) AS "Average working hours" FROM employee;



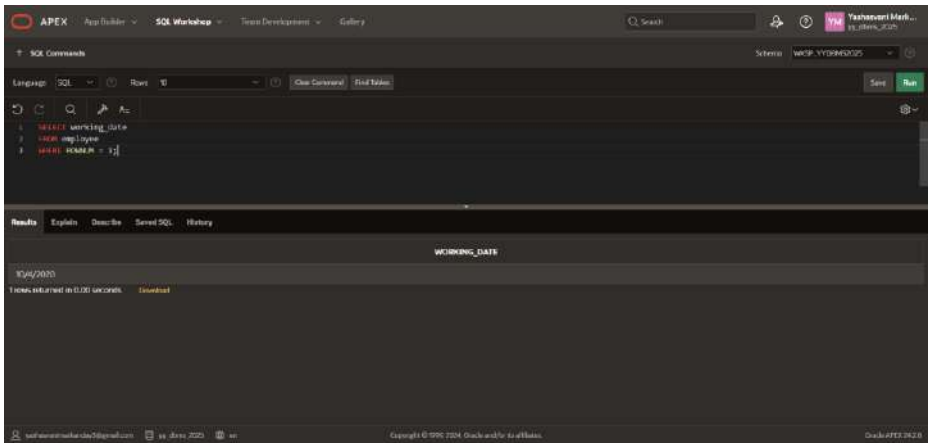
4. SELECT MIN(working_hours) AS Minimum_working_hours FROM employee;



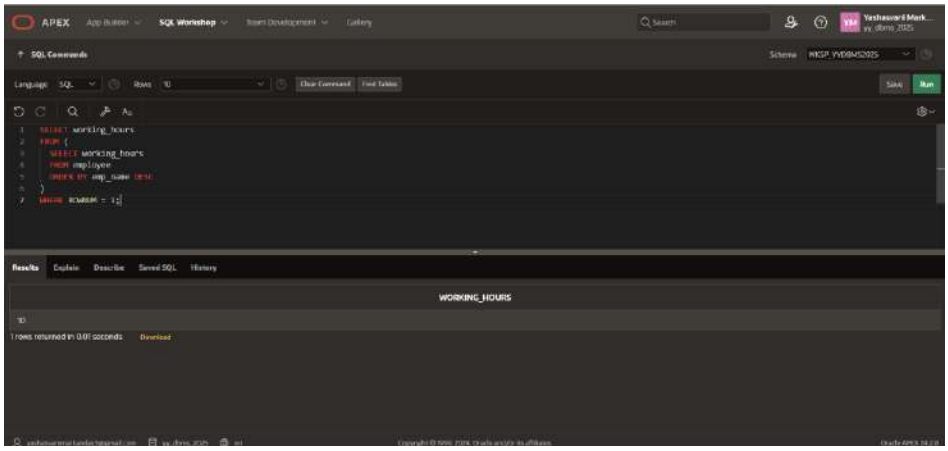
5. SELECT MAX(working_hours) AS Maximum_working_hours FROM employee;

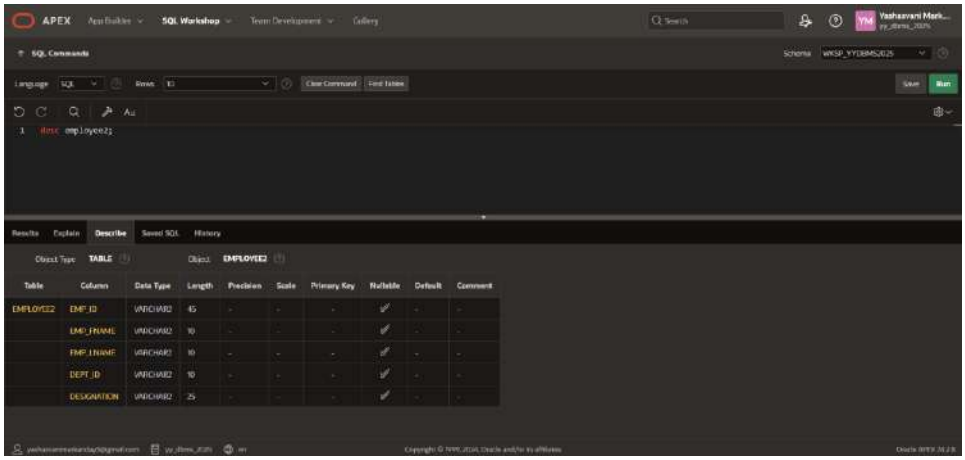


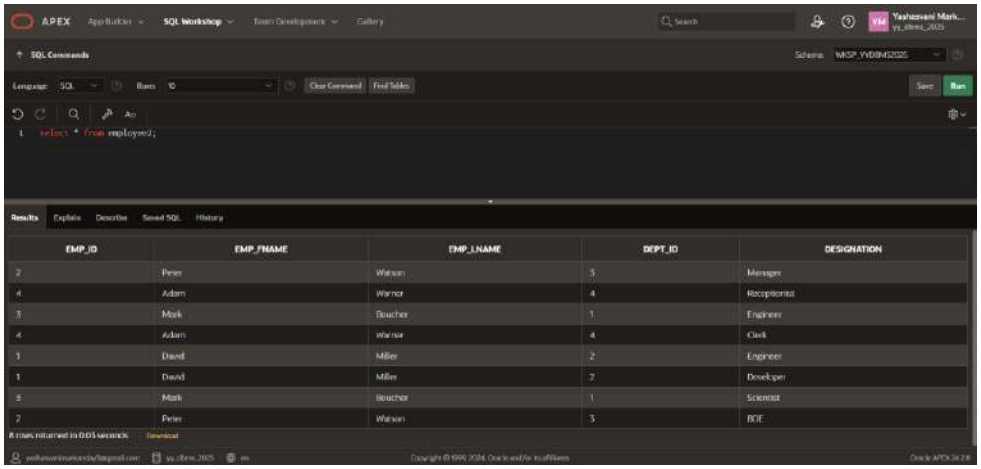
6. SELECT working_date FROM employee LIMIT 1;



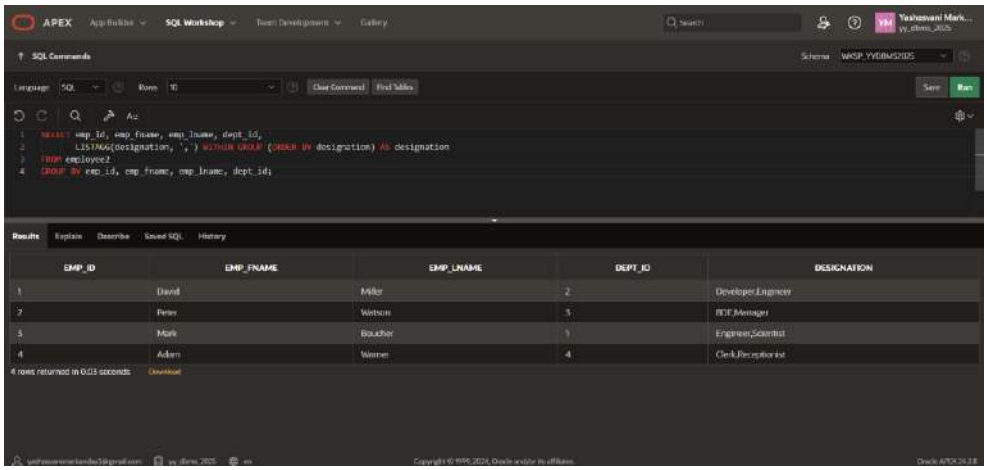
7. SELECT working_hours FROM employee ORDER BY name DESC LIMIT 1;



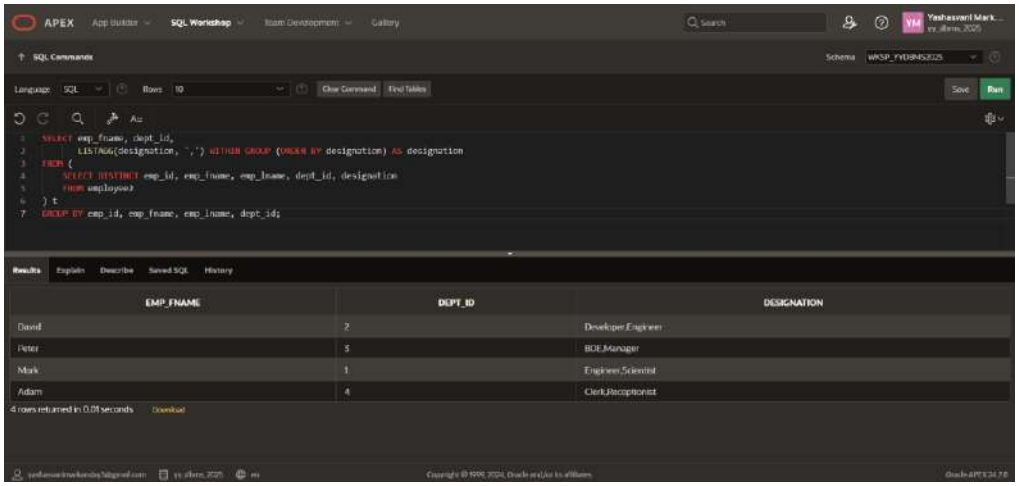




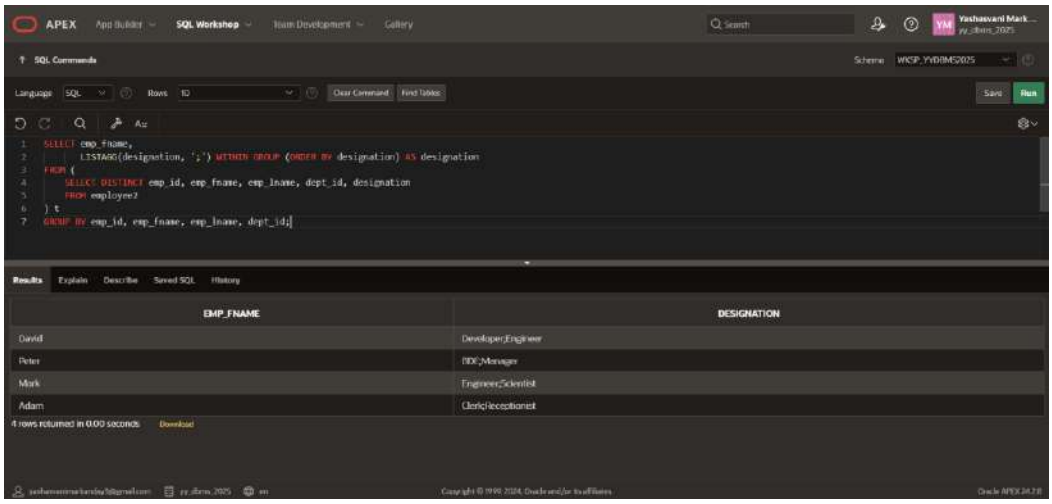
8. SELECT emp_id, emp_fname, emp_lname, dept_id, GROUP_CONCAT(designation) AS designation FROM employee2 GROUP BY emp_id, emp_fname, emp_lname, dept_id;



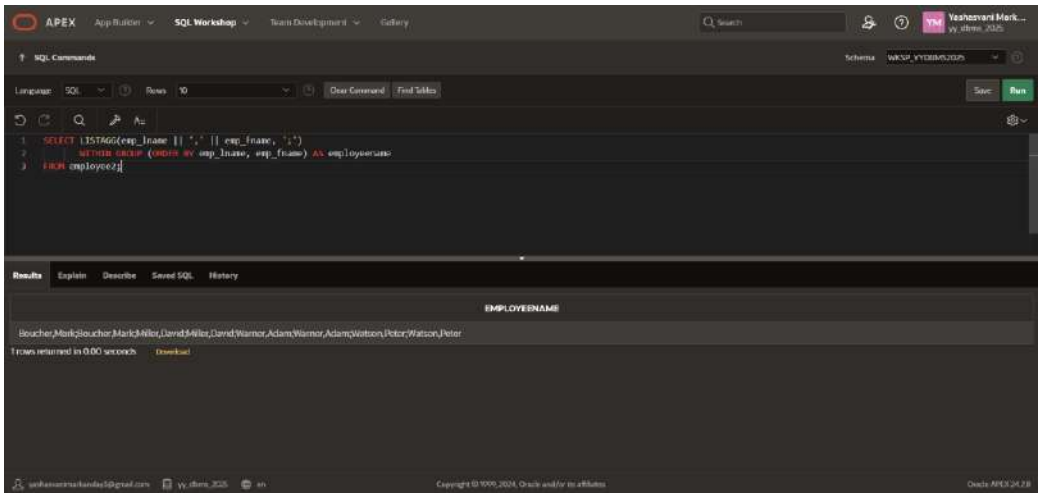
9. SELECT emp_fname, dept_id, GROUP_CONCAT(DISTINCT designation) as designation FROM employee2 group by emp_id, emp_fname, emp_lname, dept_id;



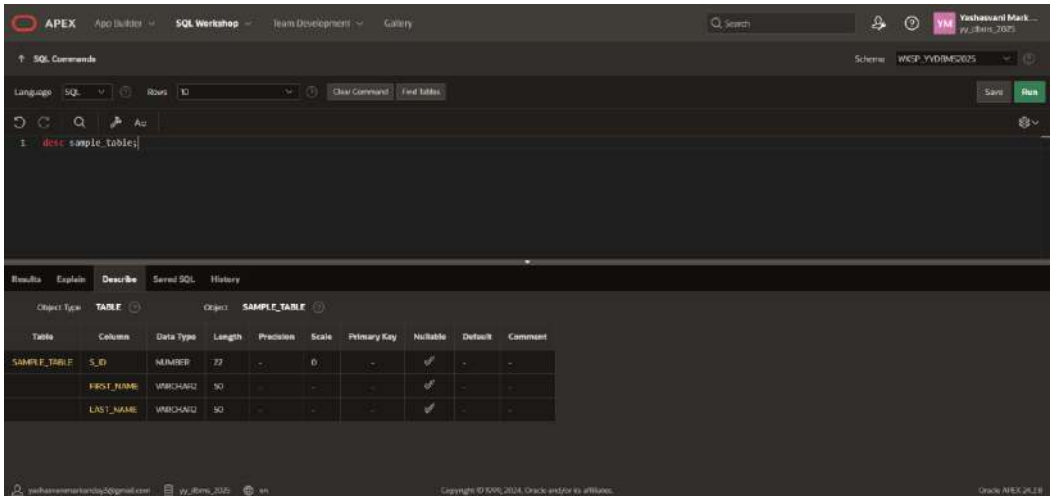
10. SELECT emp_fname, GROUP_CONCAT(DISTINCT designation SEPARATOR ';') as designation FROM employee2 group by emp_id, emp_fname, emp_lname, dept_id;



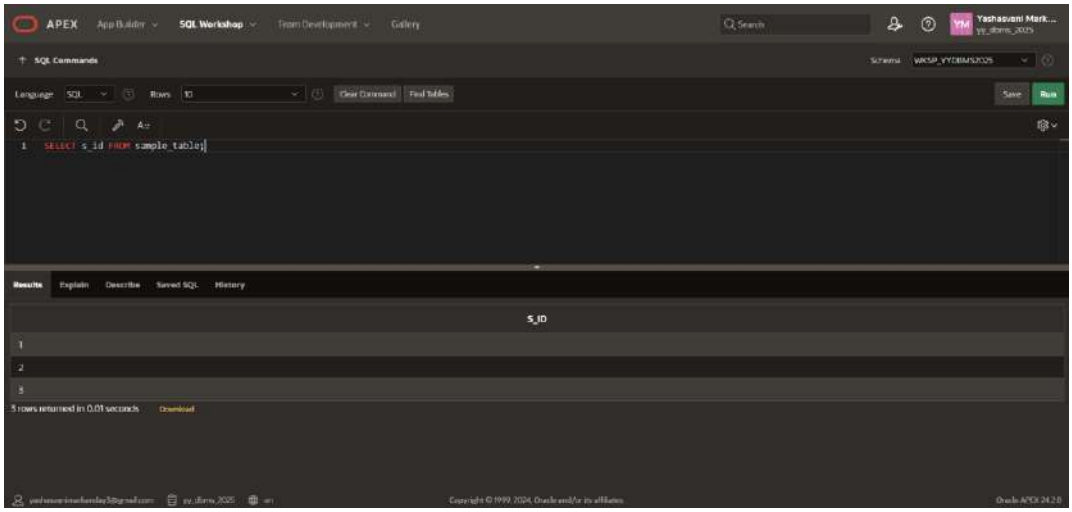
11. SELECT GROUP_CONCAT(CONCAT_WS(',', emp_lname, emp_fname) SEPARATOR ';') as employeename FROM employee2;



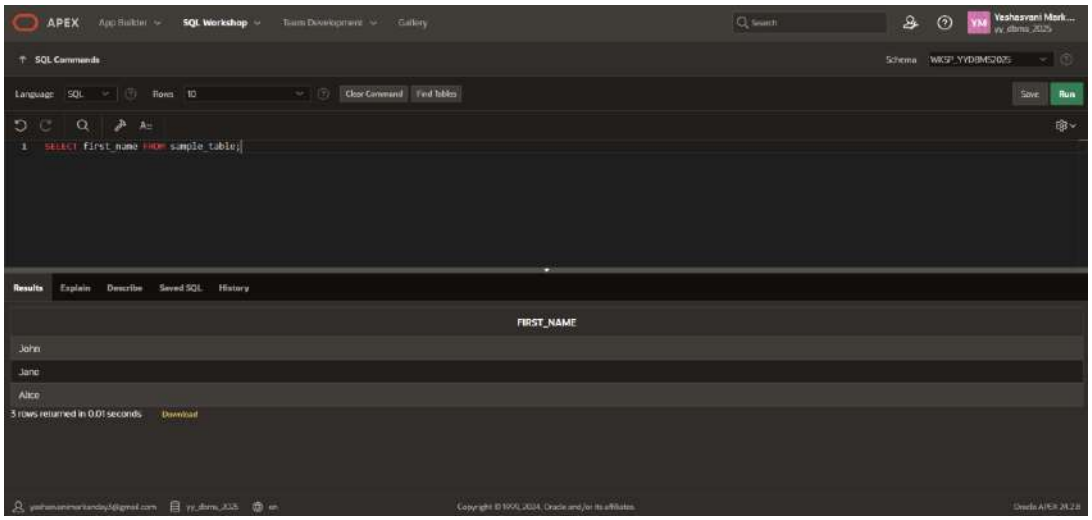
MySQL String Function



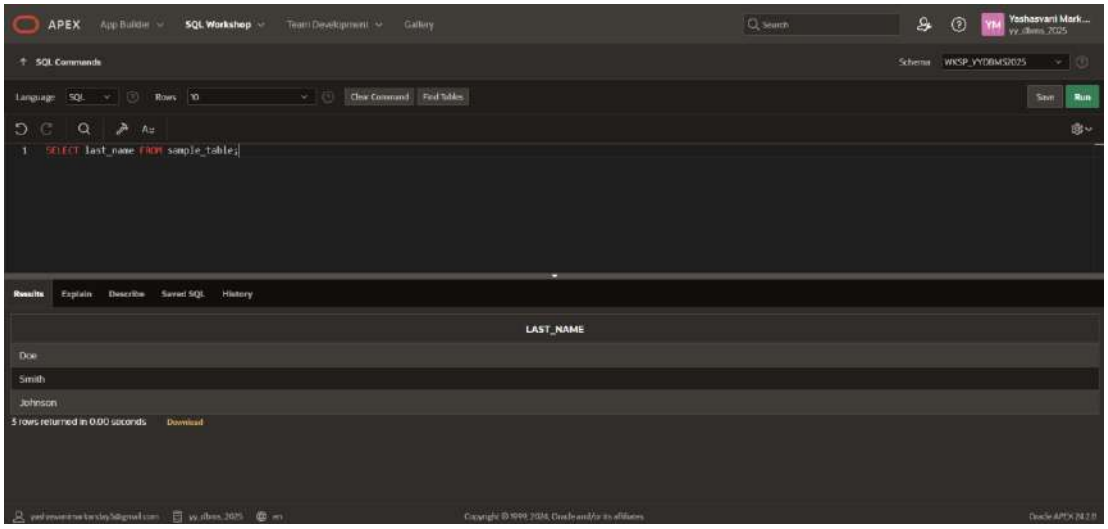
12. SELECT id FROM sample_table;



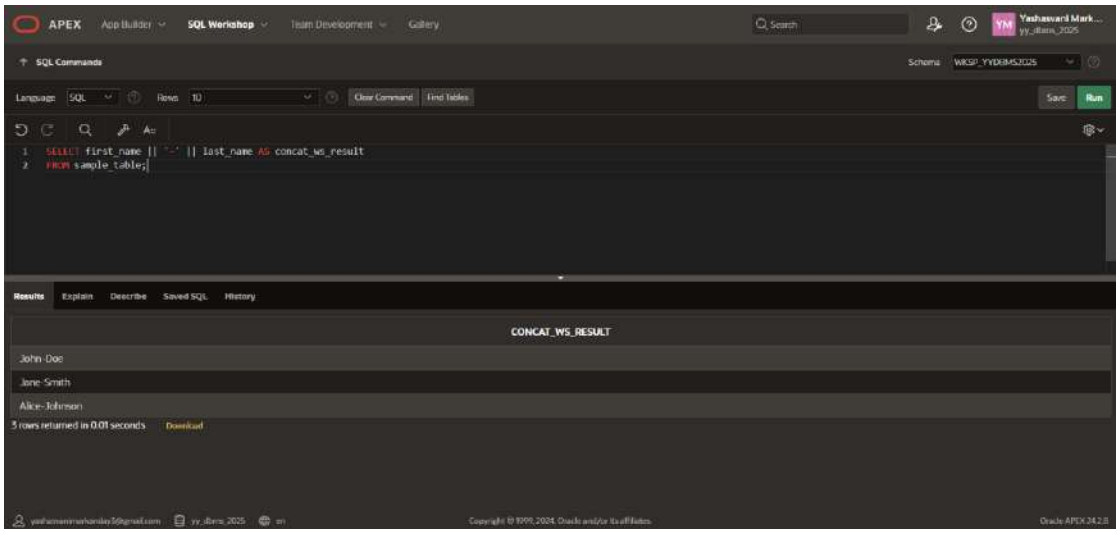
13. SELECT first_name FROM sample_table;



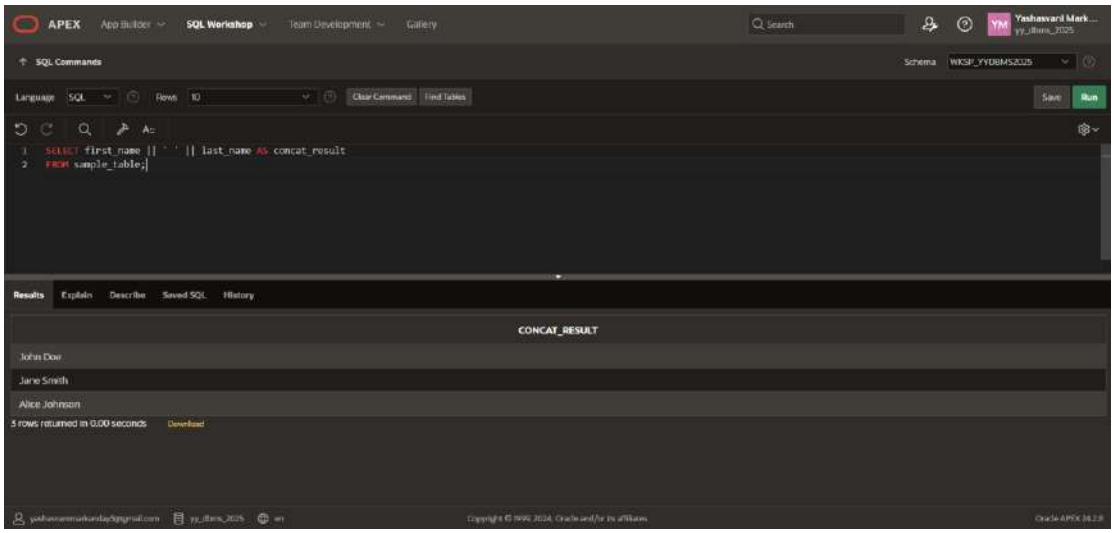
14. SELECT last_name FROM sample_table;



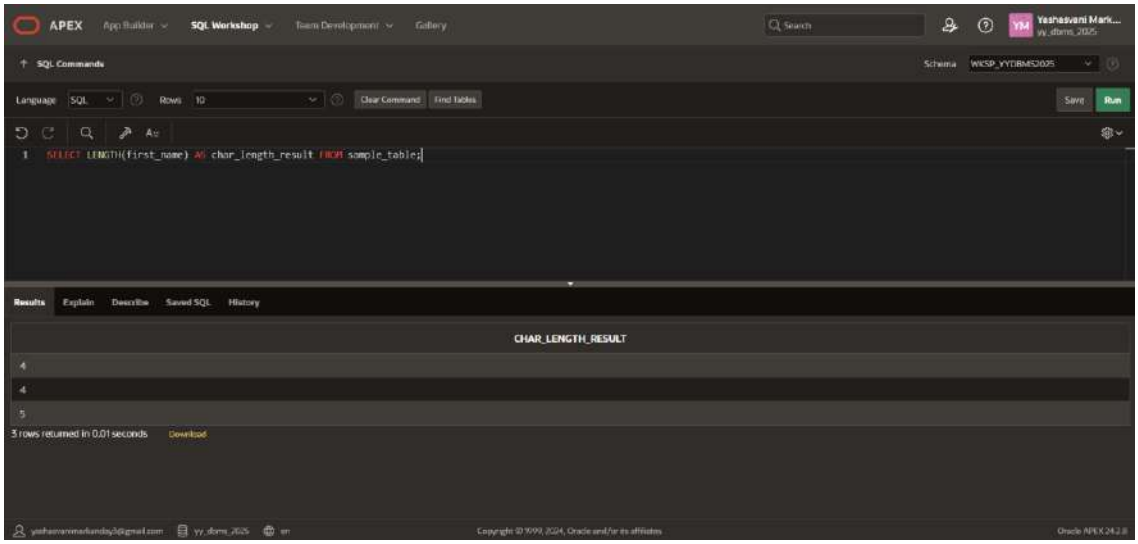
15. SELECT CONCAT_WS('-', first_name, last_name) AS concat_ws_result FROM sample_table;



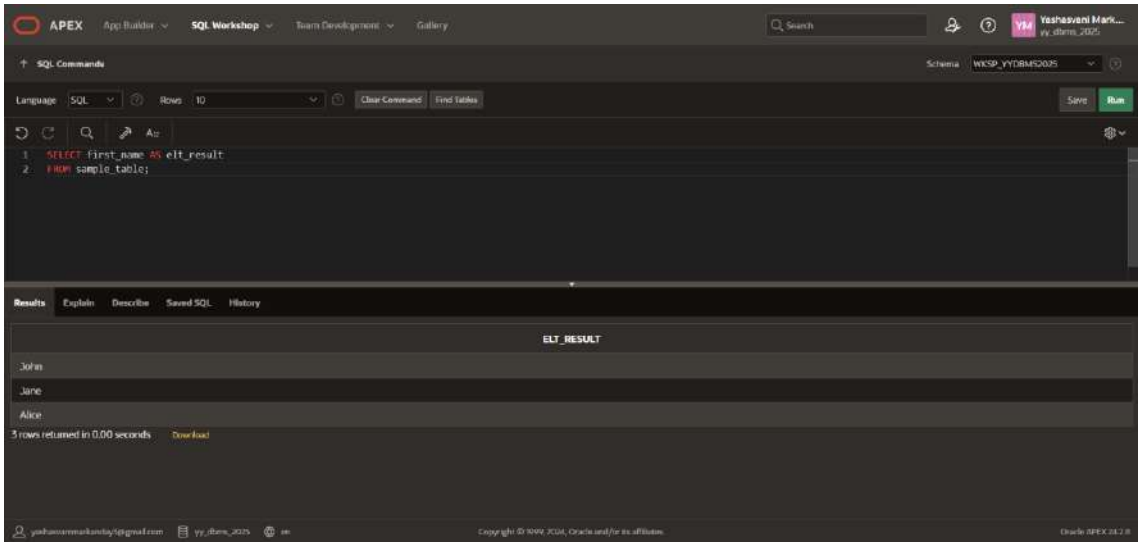
16. SELECT CONCAT(first_name, ' ', last_name) AS concat_result FROM sample_table;



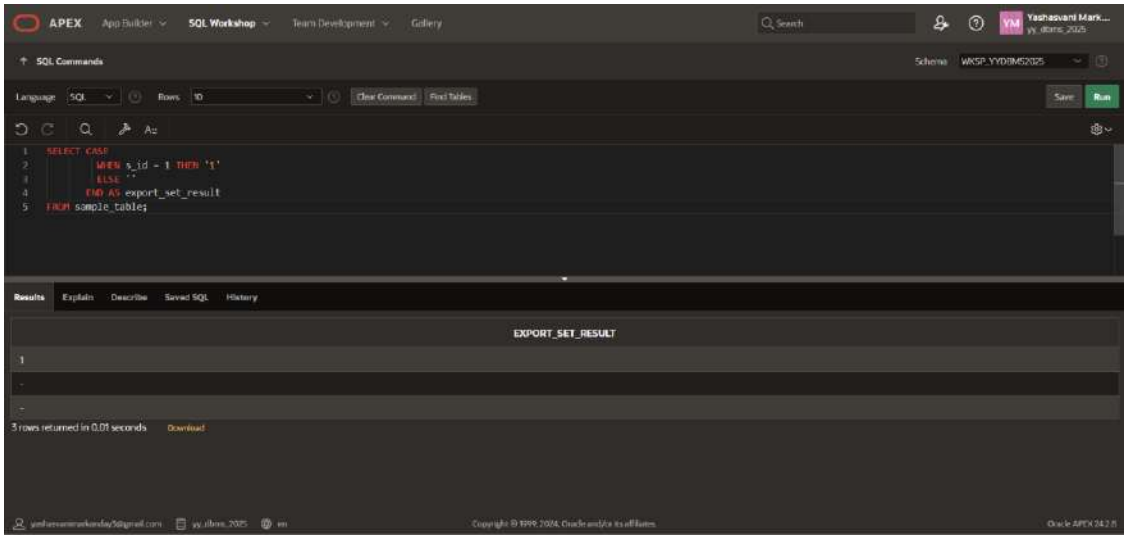
17. SELECT CHARACTER_LENGTH(first_name) AS char_length_result FROM sample_table;



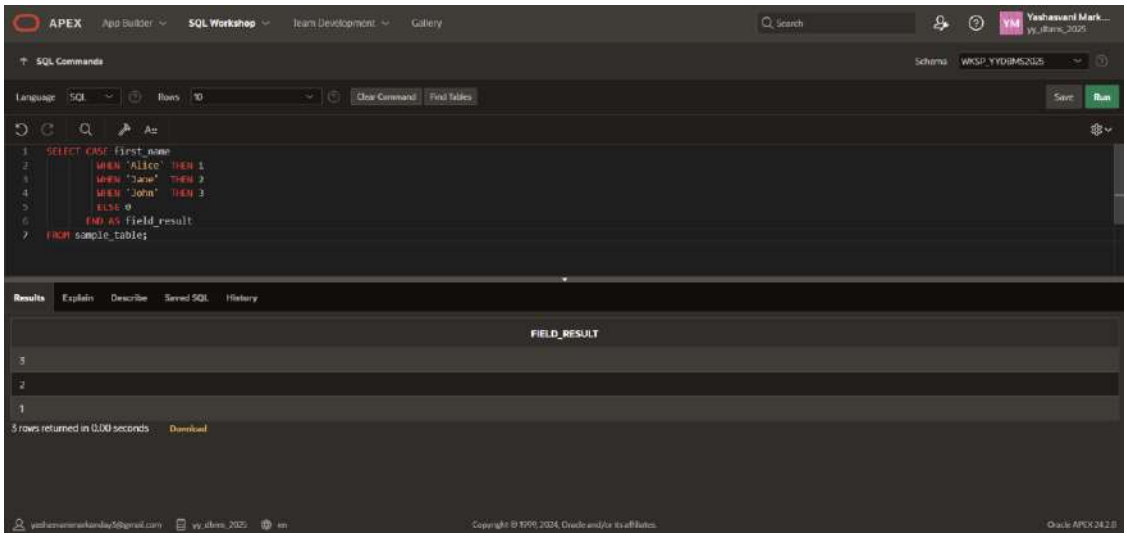
18. SELECT ELT(1, first_name, last_name, 'Other') AS elt_result FROM sample_table;



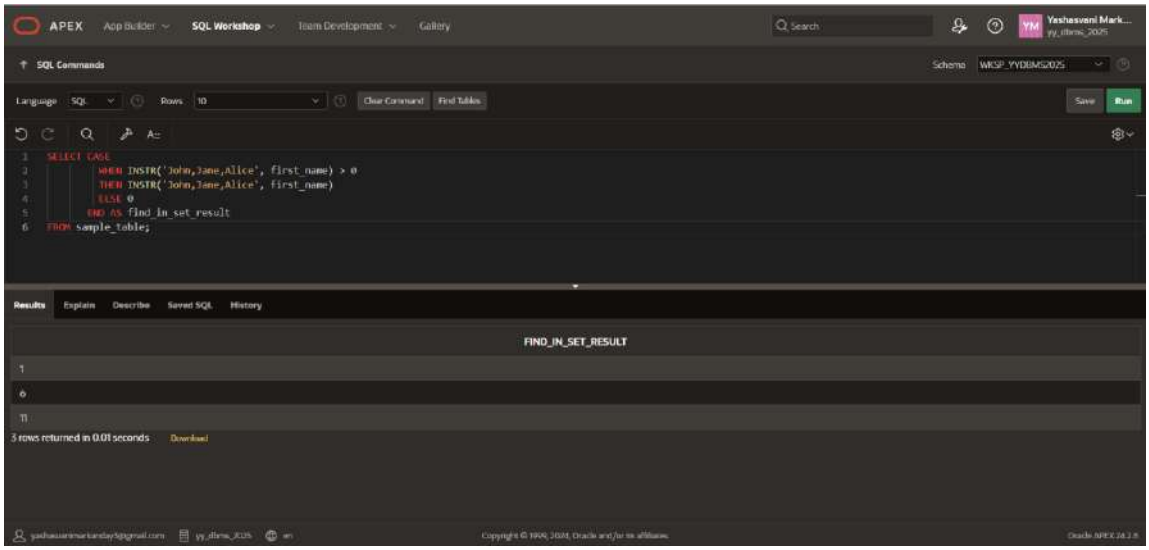
19. SELECT EXPORT_SET(id, 1, ',', '') AS export_set_result FROM sample_table;



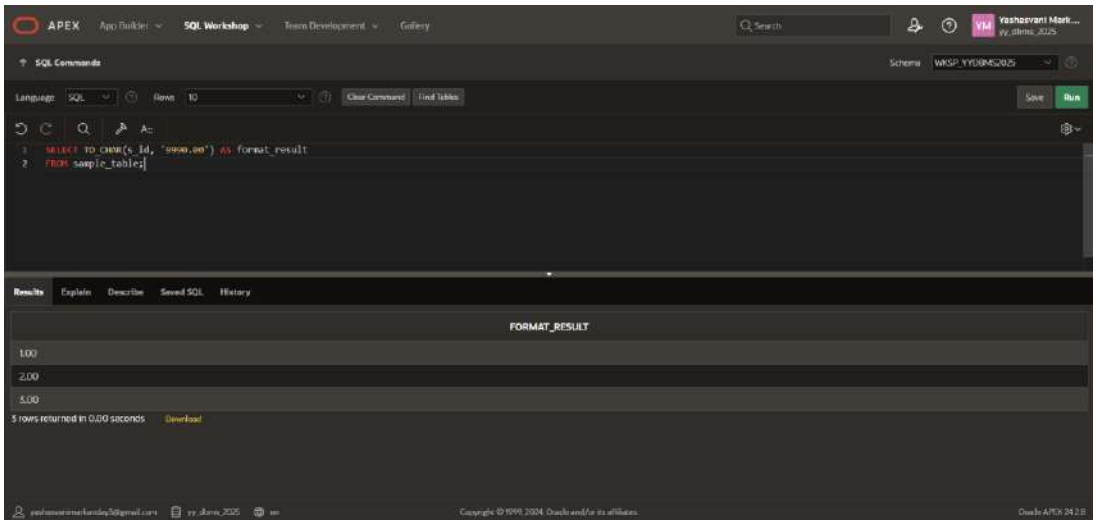
20. SELECT FIELD(first_name, 'Alice', 'Jane', 'John') AS field_result FROM sample_table;



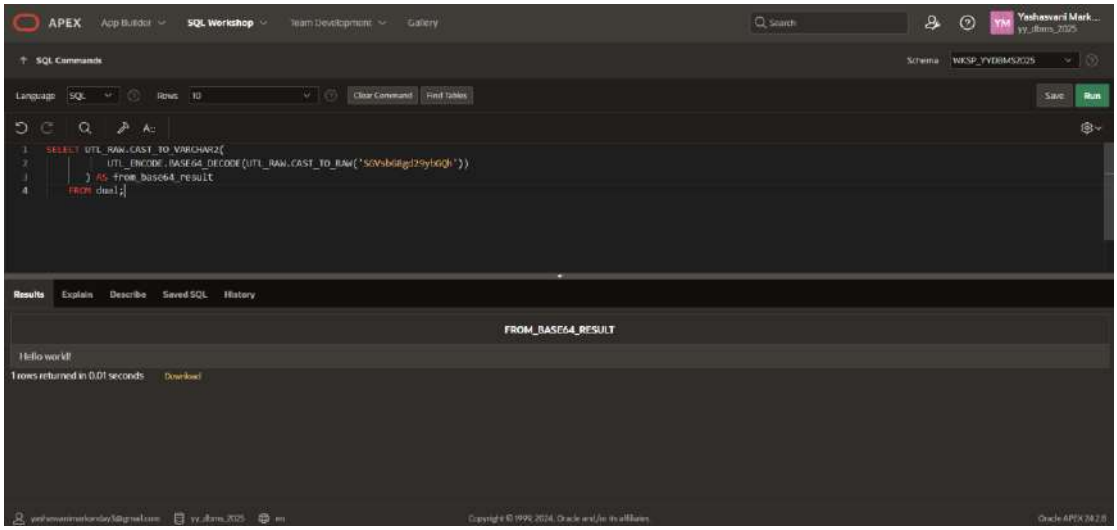
21. SELECT FIND_IN_SET(first_name, 'John,Jane,Alice') AS find_in_set_result FROM sample_table;



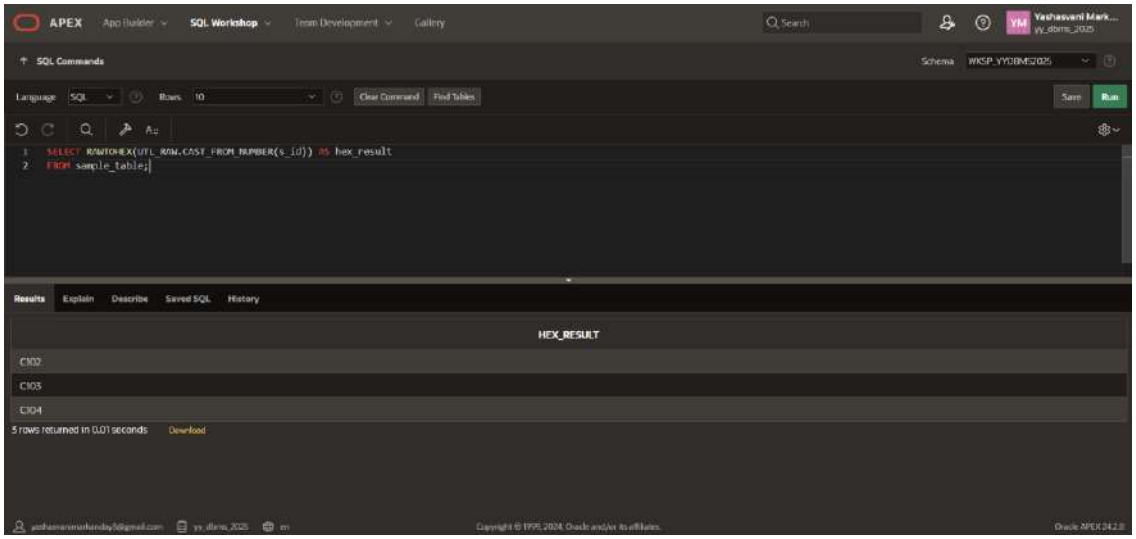
22. SELECT FORMAT(id, 2) AS format_result FROM sample_table;



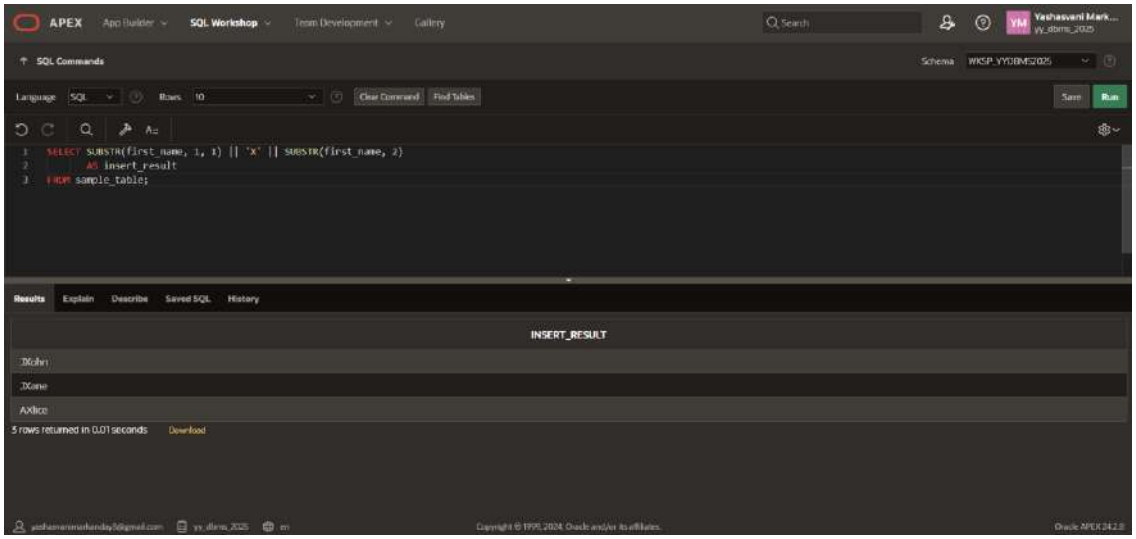
23. SELECT FROM_BASE64('SGVsbG8gd29ybGQh') AS from_base64_result FROM sample_table;



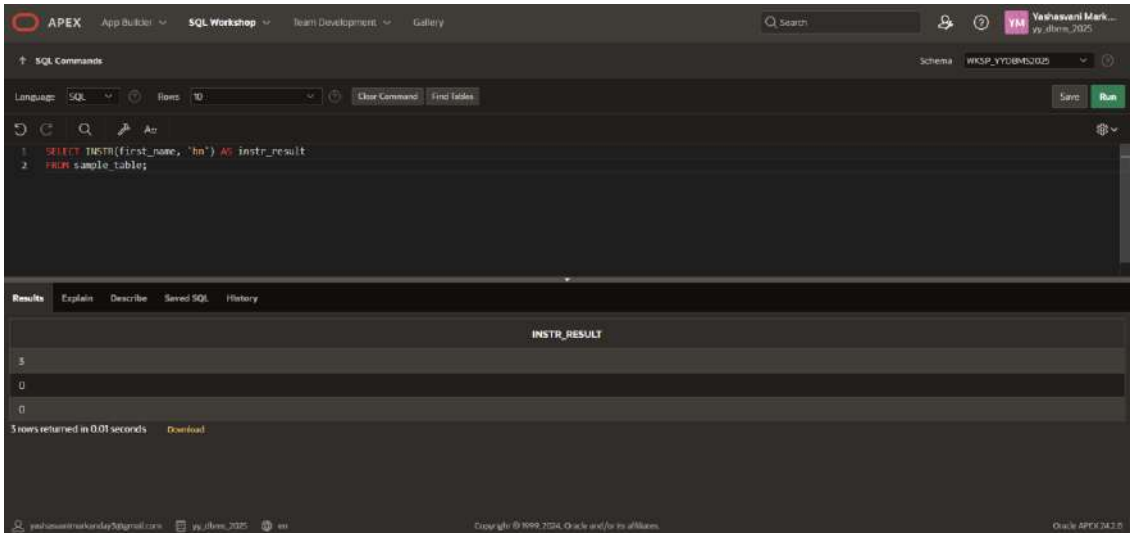
24. SELECT HEX(id) AS hex_result FROM sample_table;



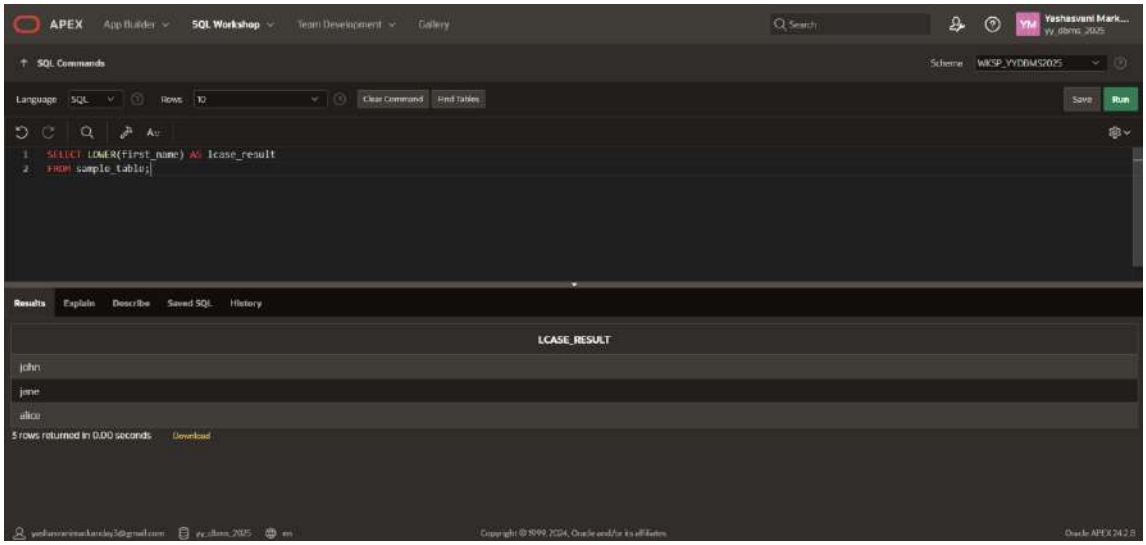
25. SELECT INSERT(first_name, 2, 0, 'X') AS insert_result FROM sample_table;



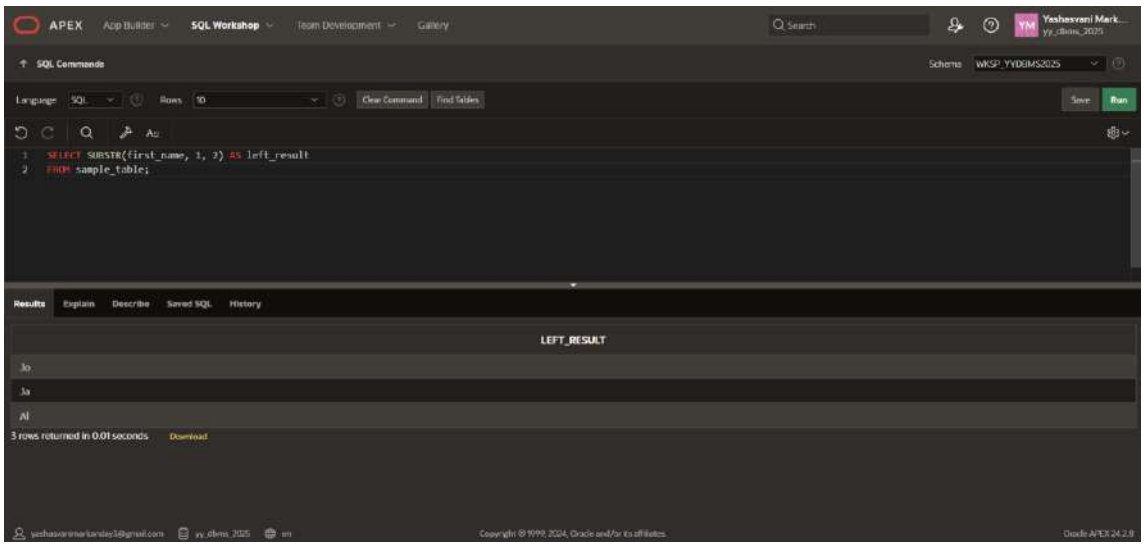
26. SELECT INSTR(first_name, 'hn') AS instr_result FROM sample_table;



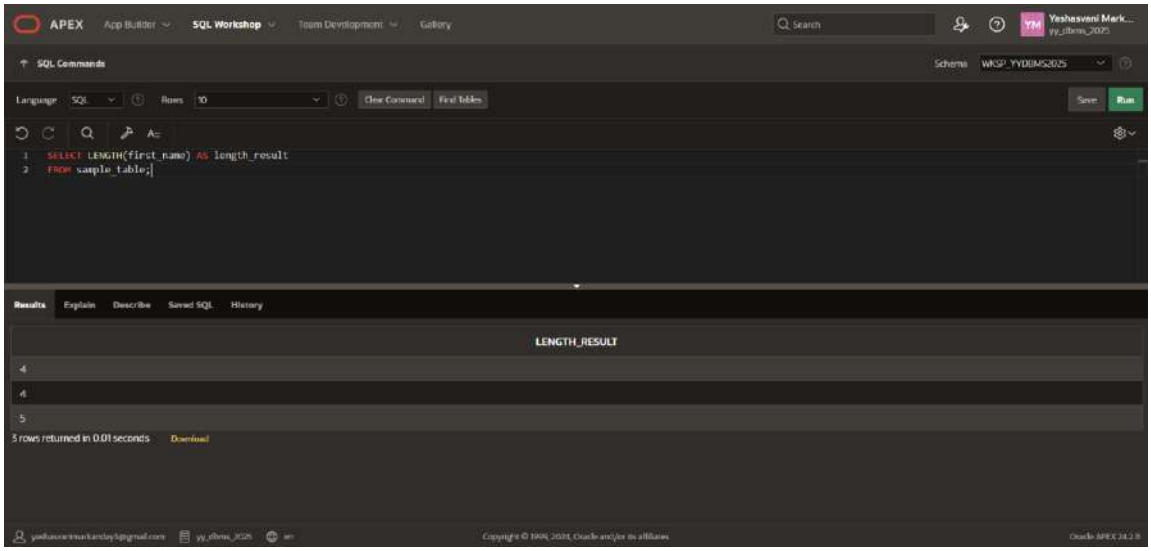
27. SELECT LCASE(first_name) AS lcase_result FROM sample_table;



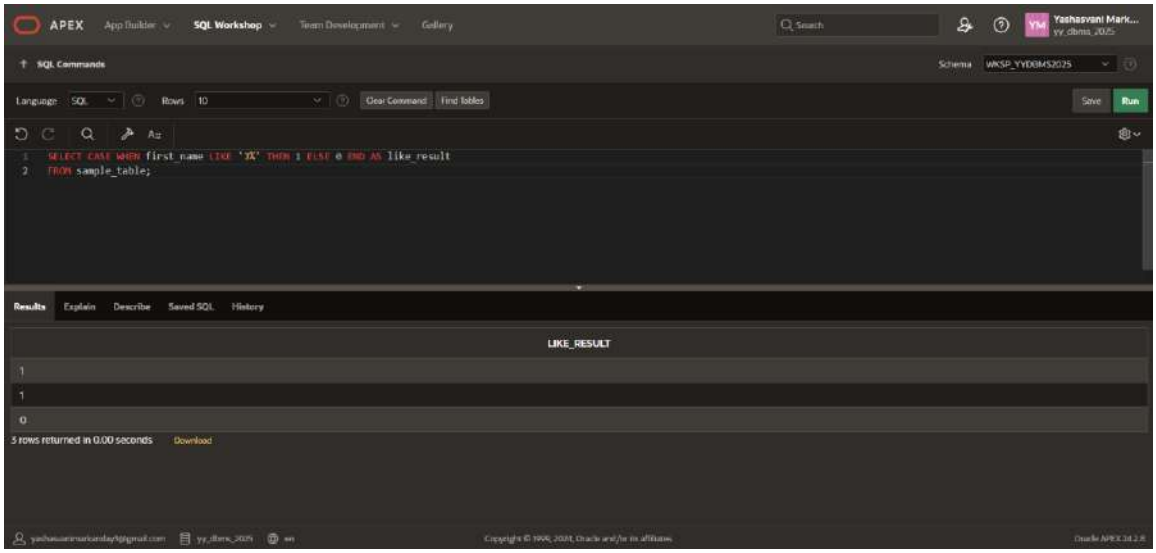
28. SELECT LEFT(first_name, 2) AS left_result FROM sample_table;



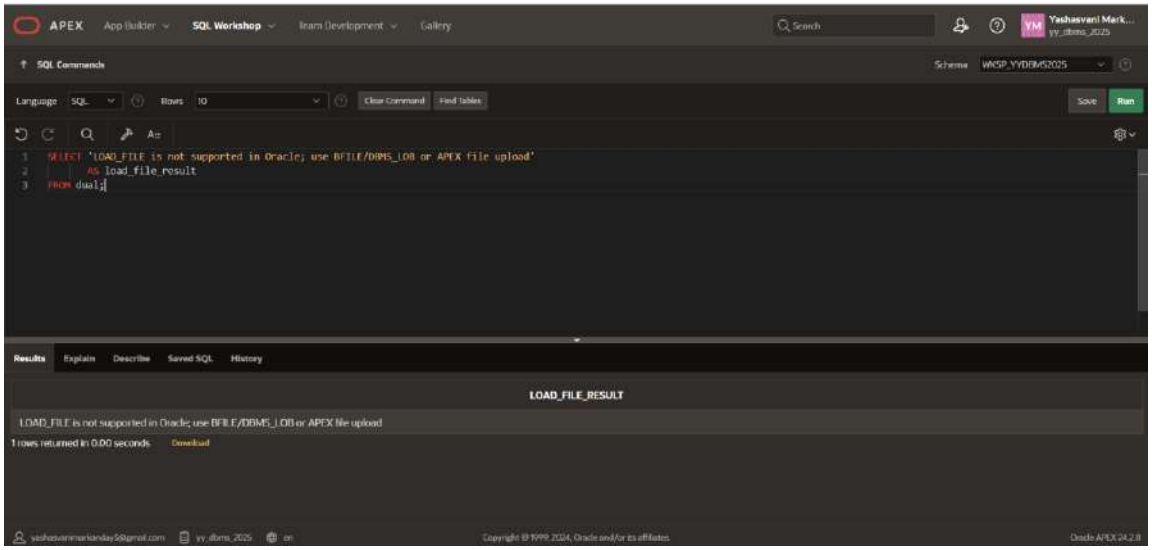
29. SELECT LENGTH(first_name) AS length_result FROM sample_table;



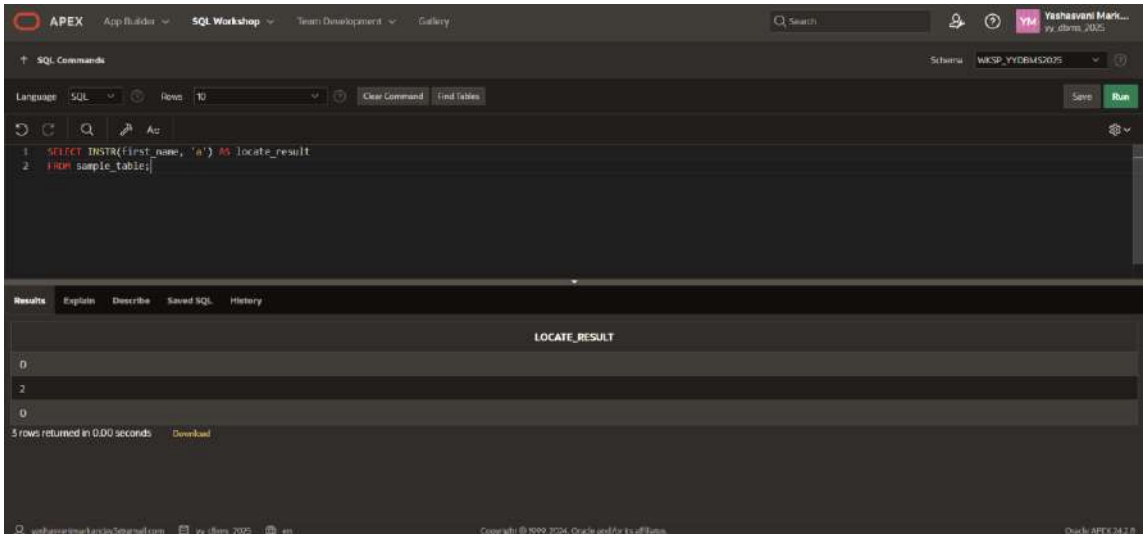
30. SELECT first_name LIKE 'J%' AS like_result FROM sample_table;



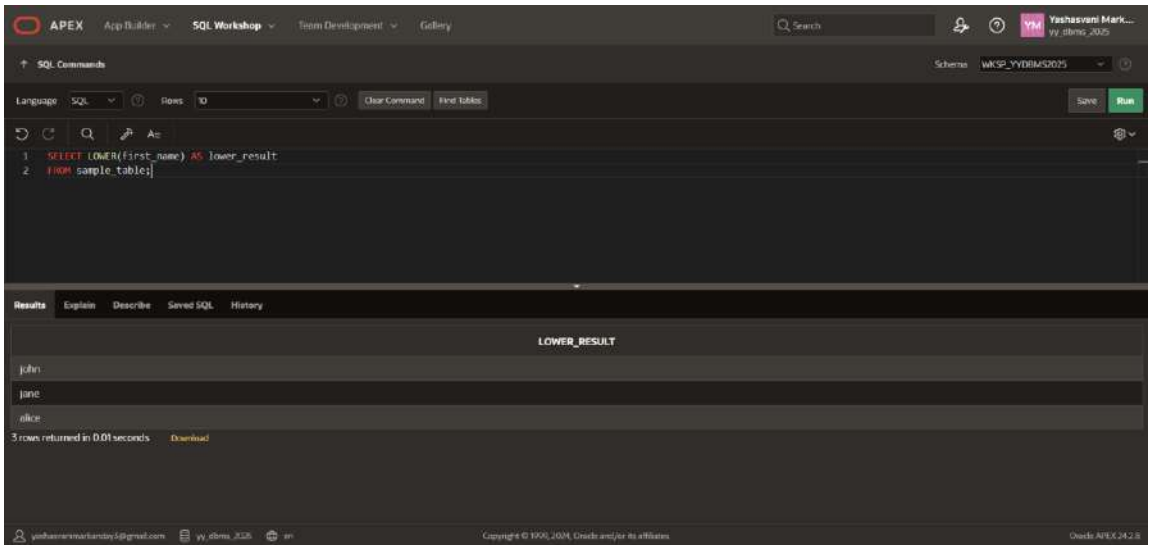
31. SELECT LOAD_FILE('/path/to/file') AS load_file_result FROM sample_table;



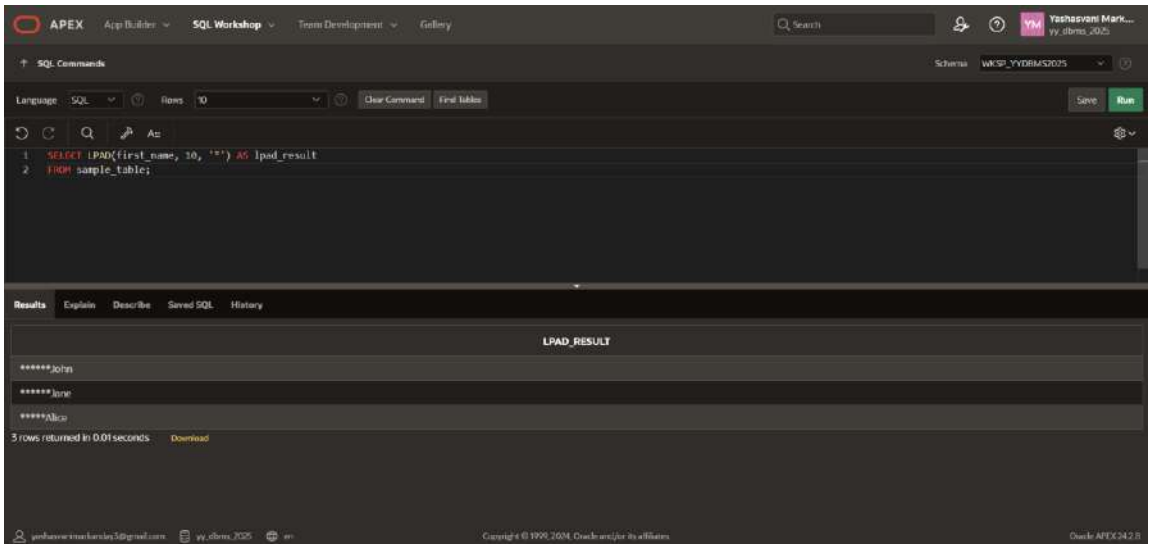
32. SELECT LOCATE('a', first_name) AS locate_result FROM sample_table;



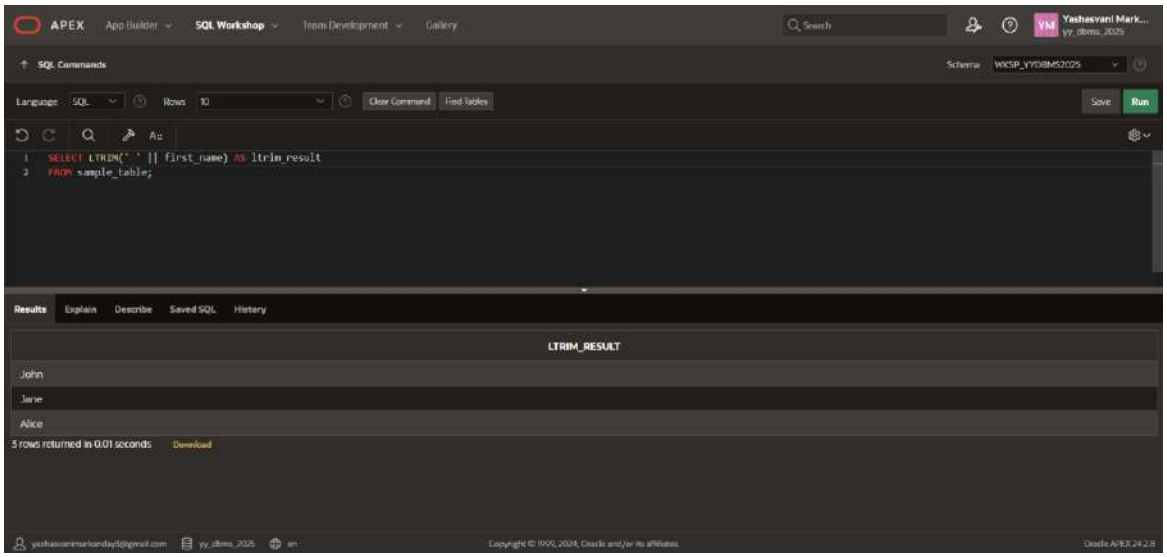
33. SELECT LOWER(first_name) AS lower_result FROM sample_table;



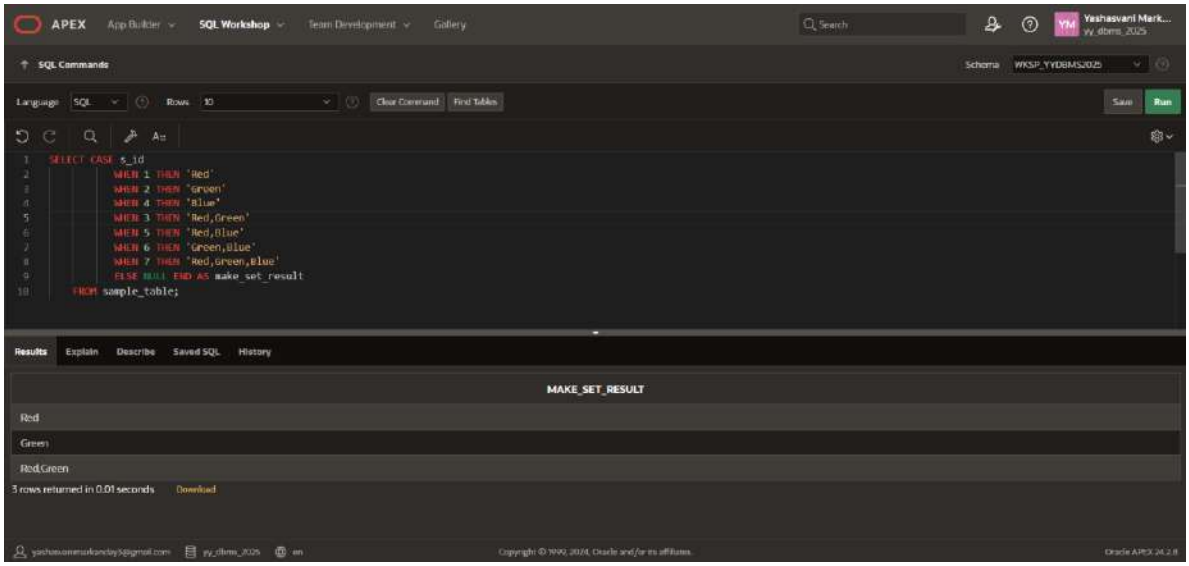
34. SELECT LPAD(first_name, 10, '*') AS lpad_result FROM sample_table;



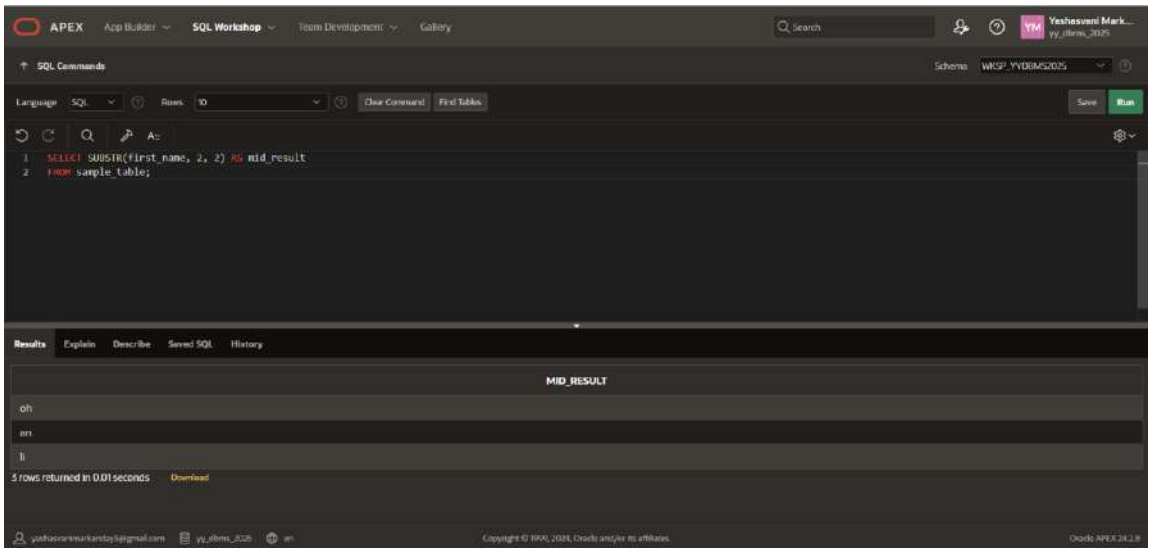
35. SELECT LTRIM(' ' || first_name) AS ltrim_result FROM sample_table;



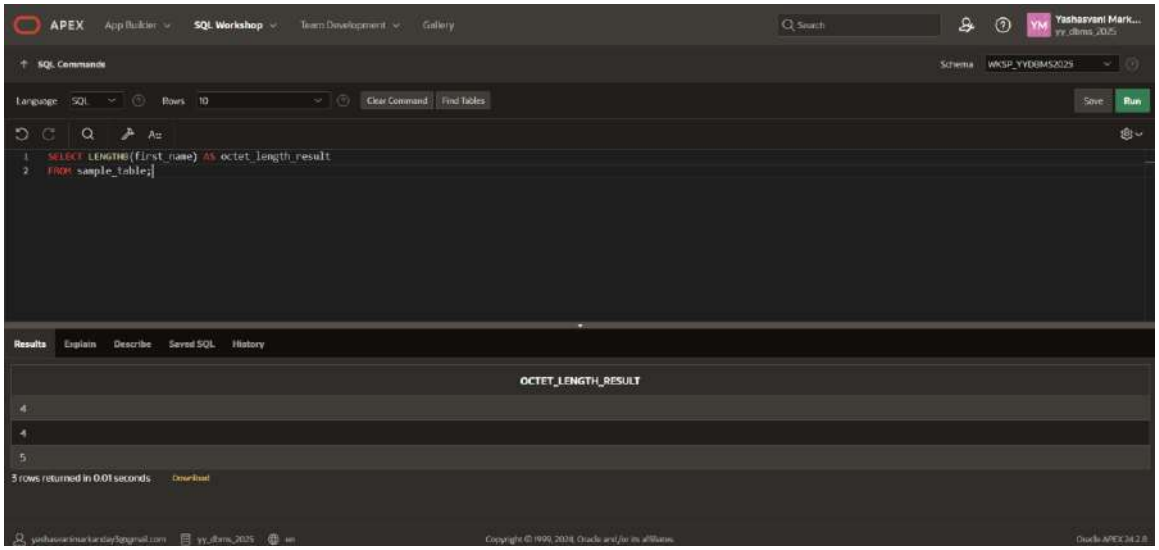
36. SELECT MAKE_SET(2, 'Red', 'Green', 'Blue') AS make_set_result FROM sample_table;



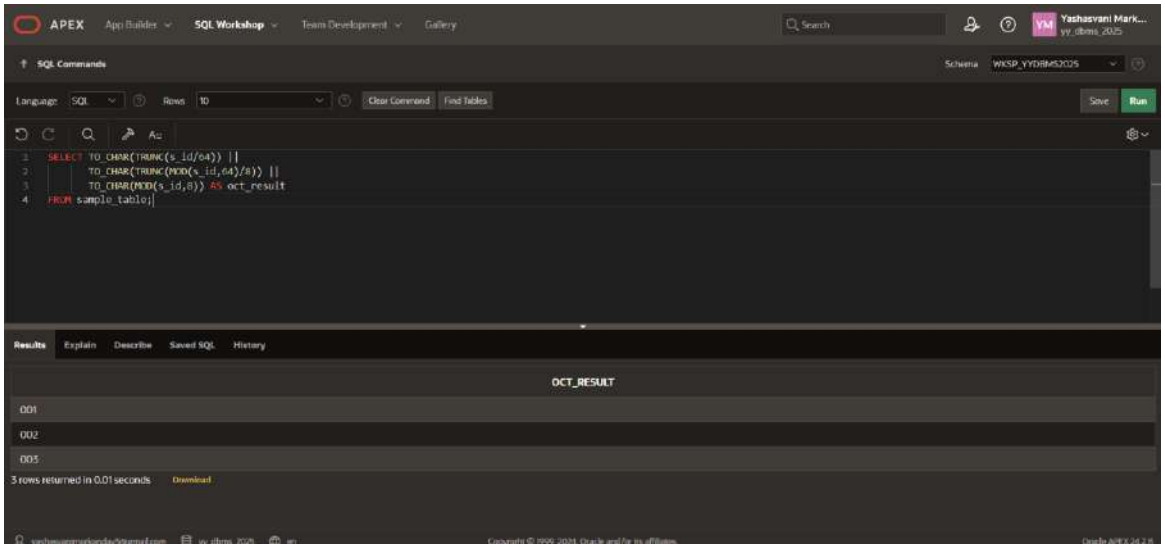
37. SELECT MID(first_name, 2, 2) AS mid_result FROM sample_table;



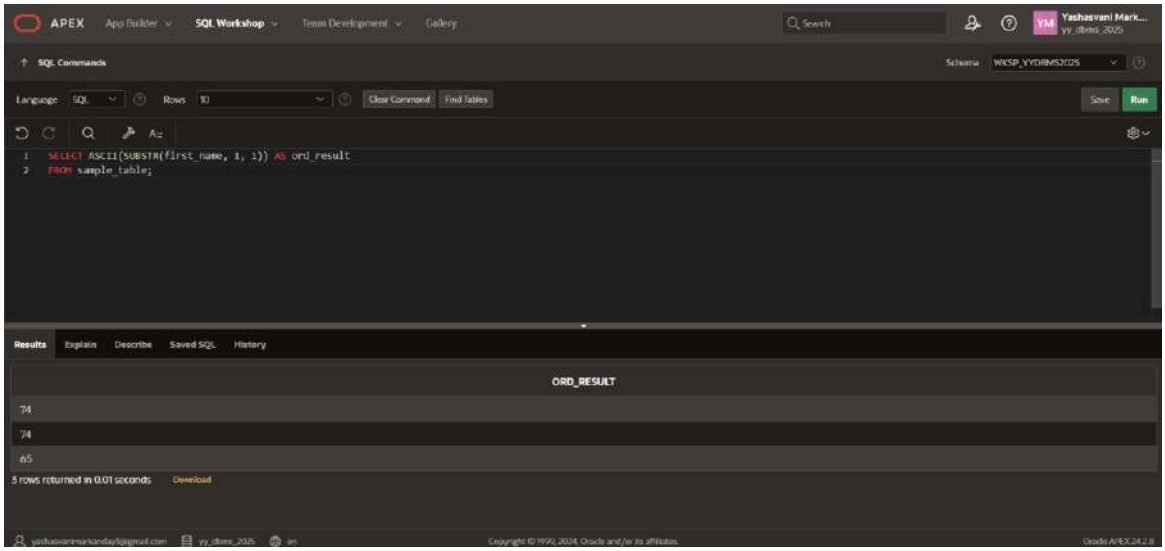
38. SELECT OCTET_LENGTH(first_name) AS octet_length_result FROM sample_table;



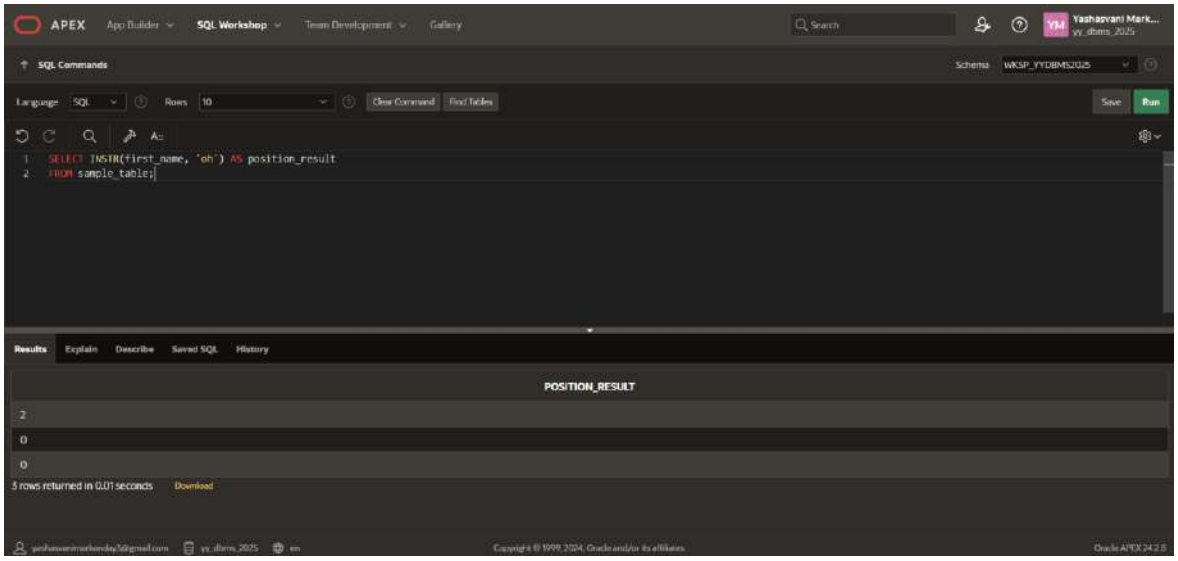
39. SELECT OCT(id) AS oct_result FROM sample_table;



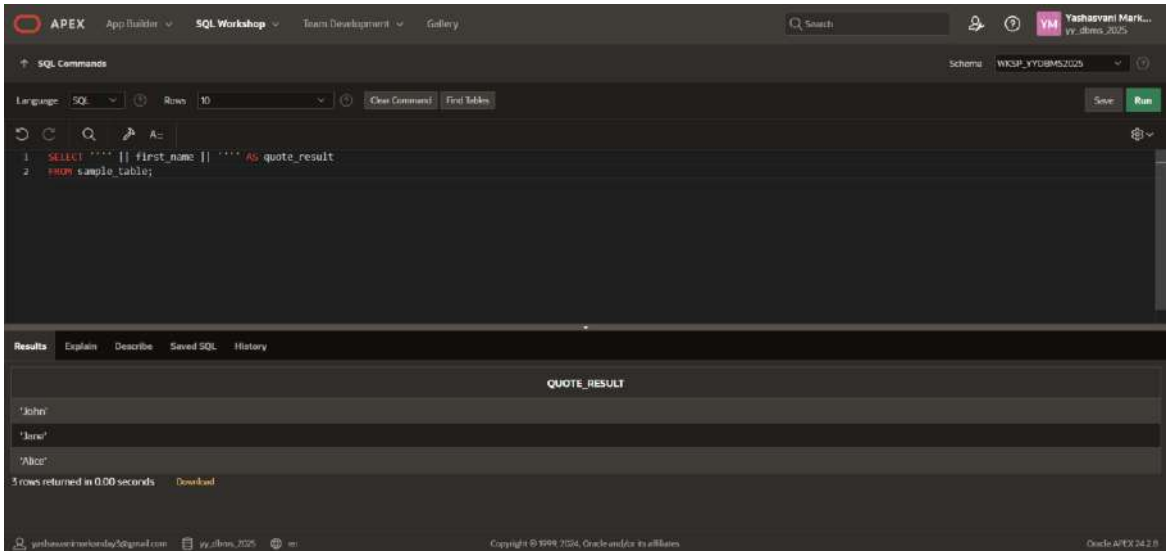
40. SELECT ORD(LEFT(first_name, 1)) AS ord_result FROM sample_table;



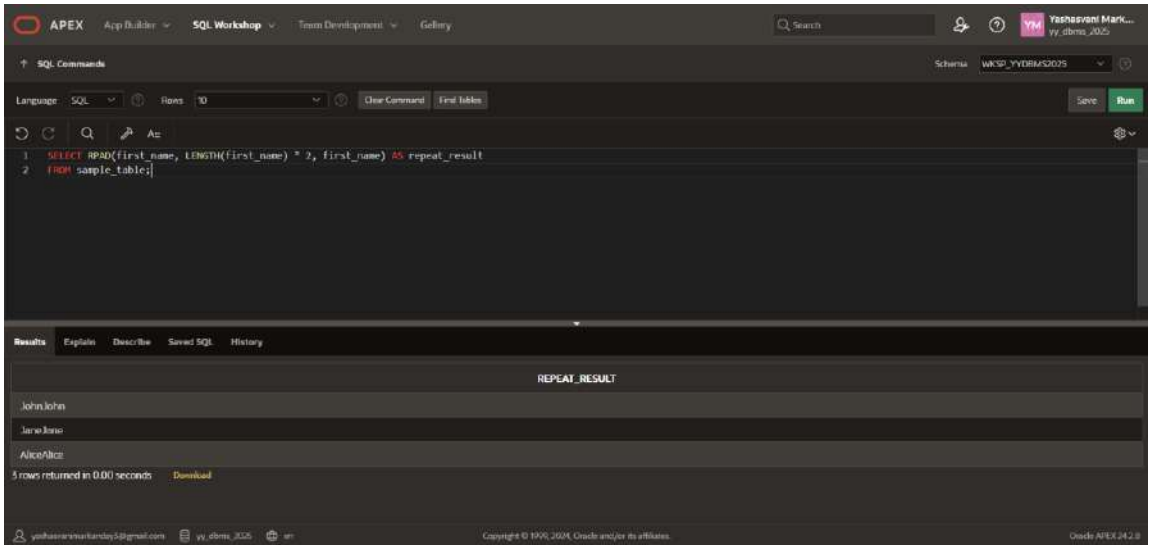
41. SELECT POSITION('oh' IN first_name) AS position_result FROM sample_table;



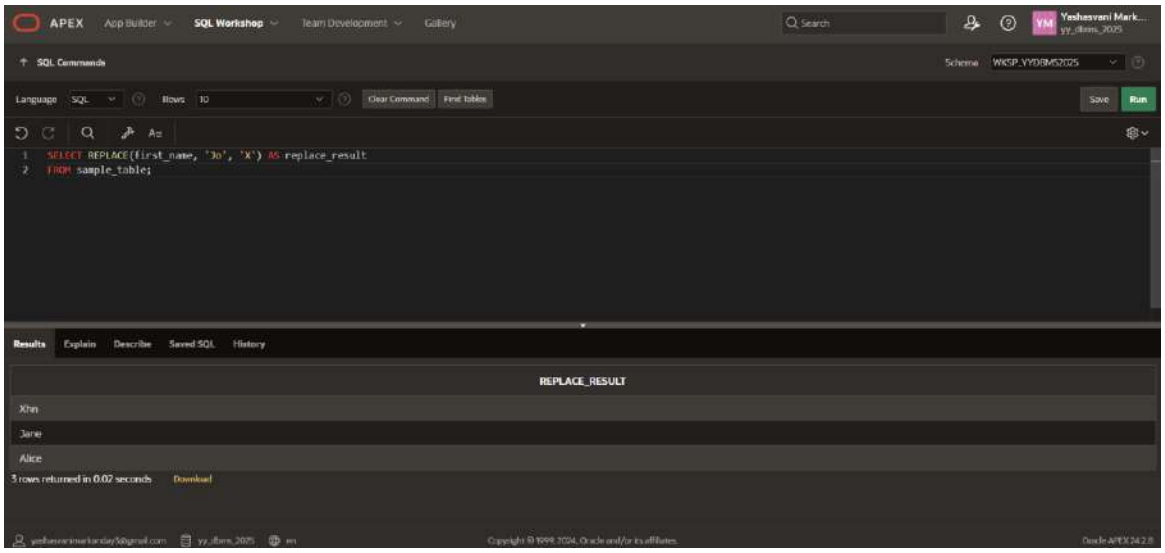
42. SELECT QUOTE(first_name) AS quote_result FROM sample_table;



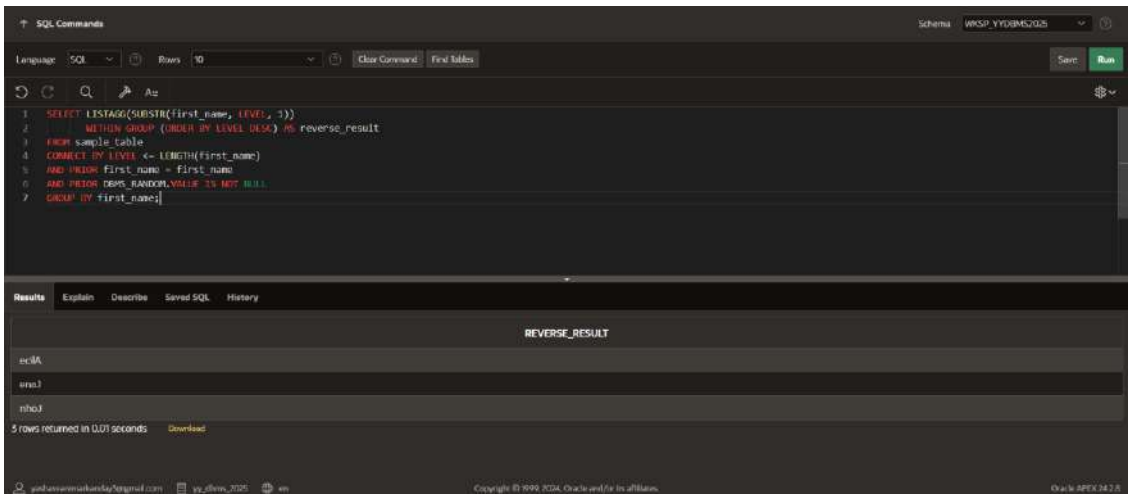
43. SELECT REPEAT(first_name, 2) AS repeat_result FROM sample_table;



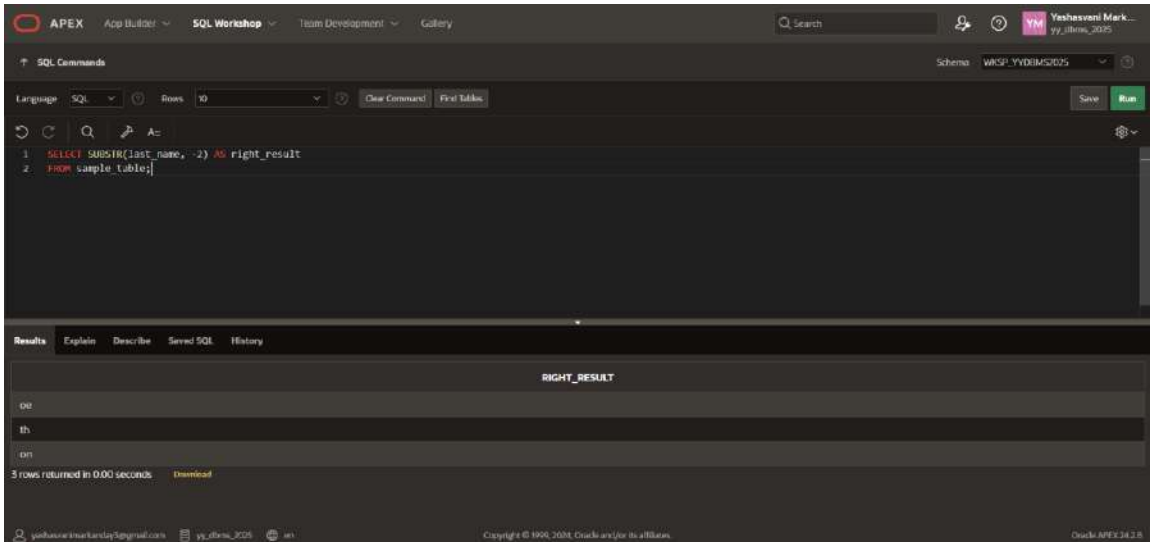
44. SELECT REPLACE(first_name, 'Jo', 'X') AS replace_result FROM sample_table;



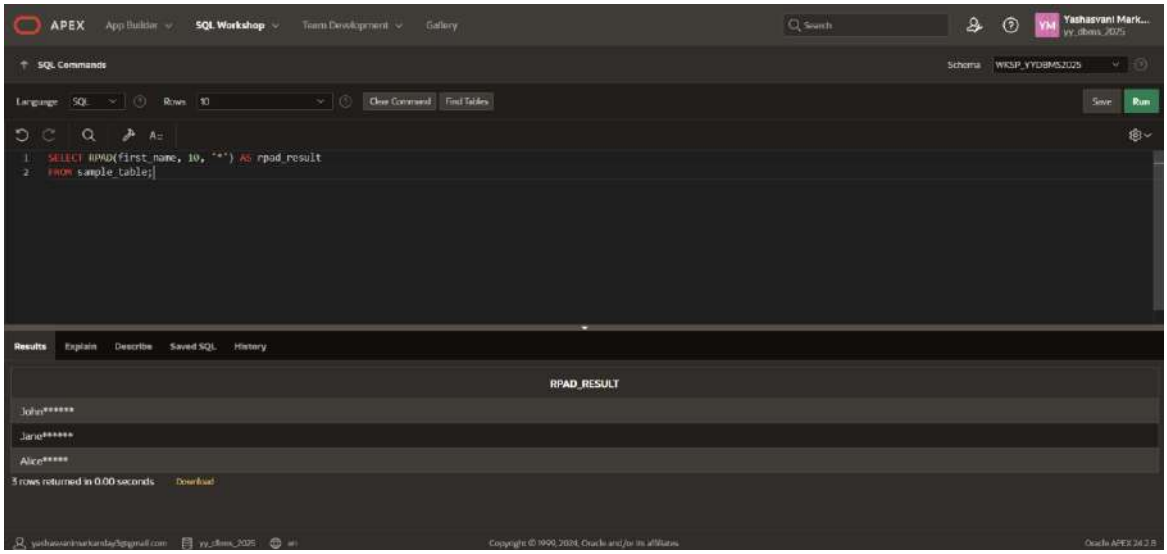
45. SELECT REVERSE(first_name) AS reverse_result FROM sample_table;



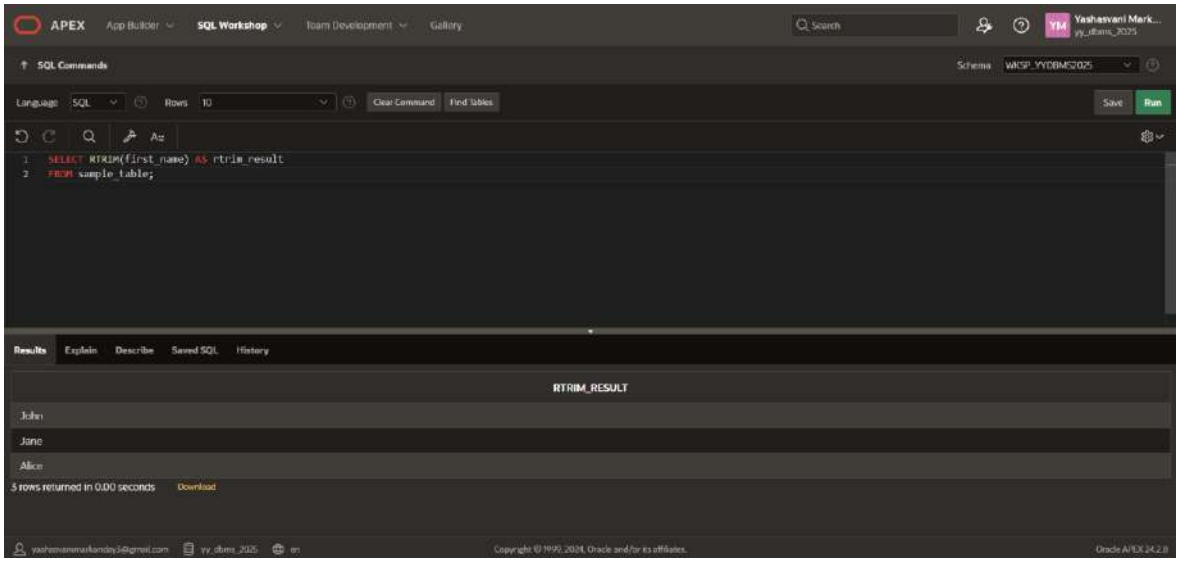
46. SELECT RIGHT(last_name, 2) AS right_result FROM sample_table;



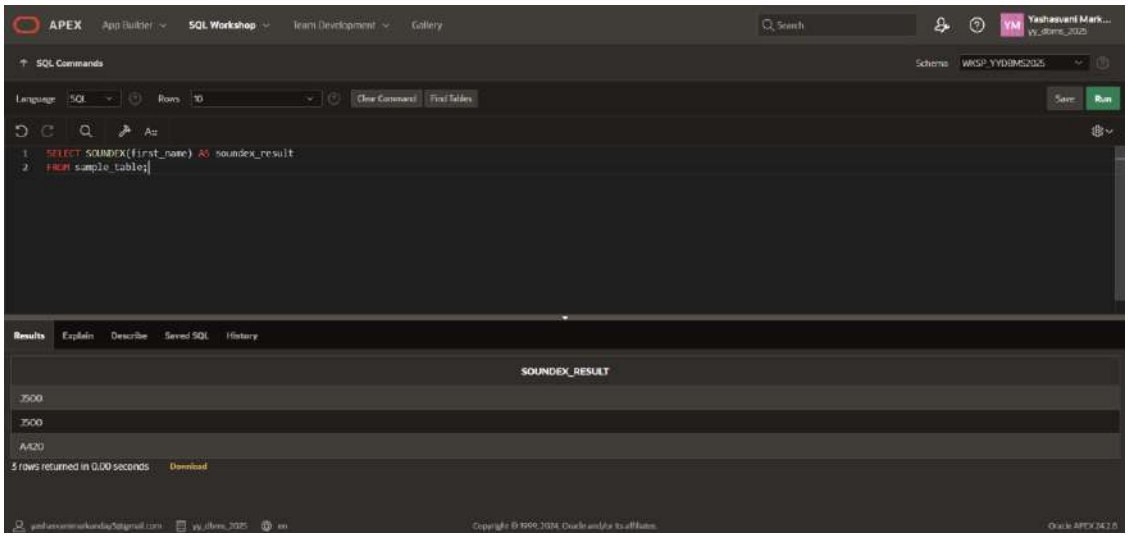
47. SELECT RPAD(first_name, 10, '*') AS rpad_result FROM sample_table;



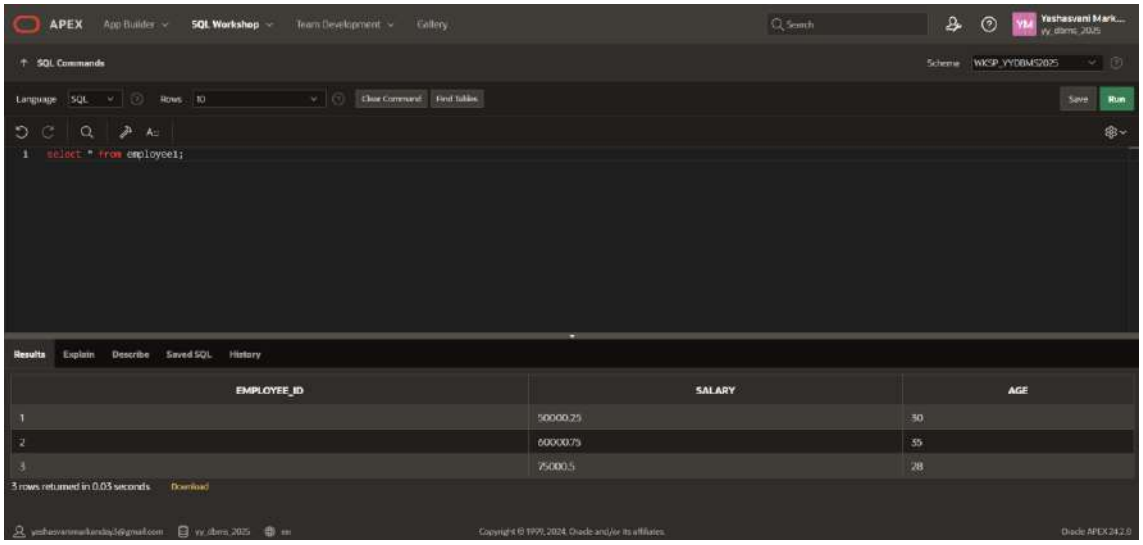
48. SELECT RTRIM(first_name) AS rtrim_result FROM sample_table;



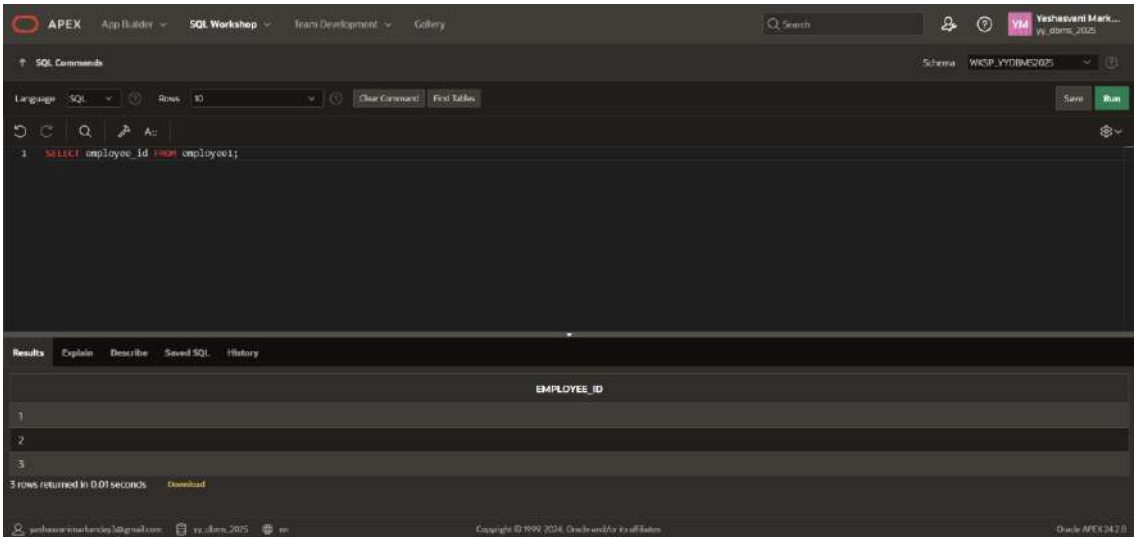
49. SELECT SOUNDEX(first_name) AS soundex_result FROM sample_table;



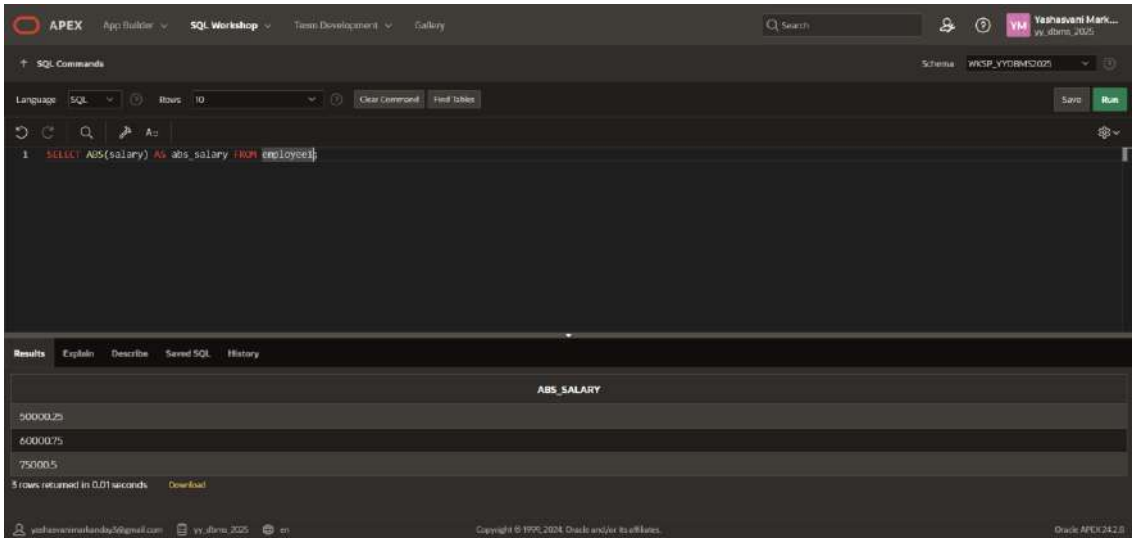
Mathematical Functions



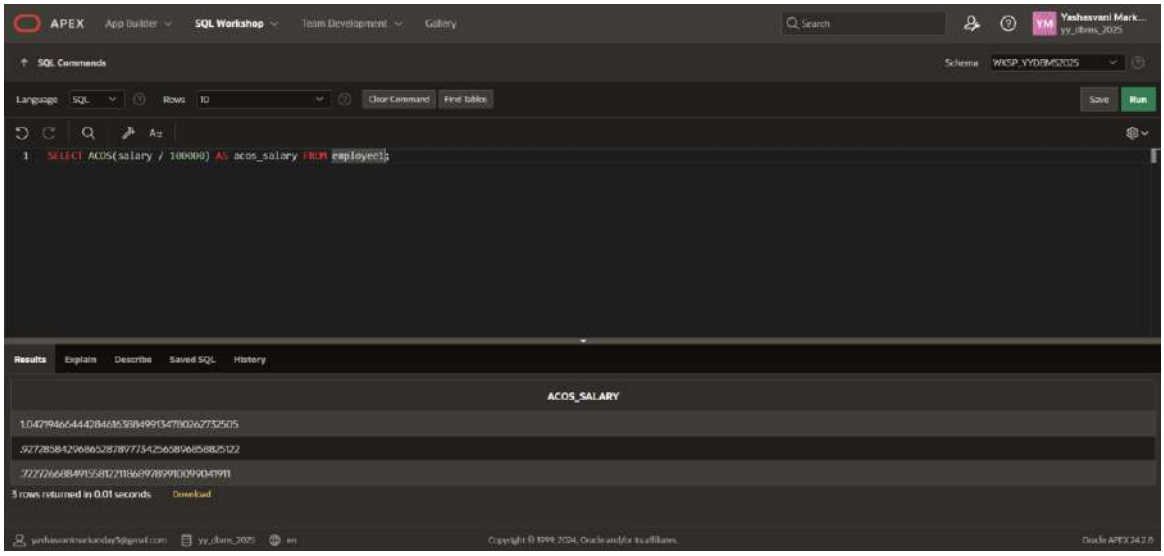
50. SELECT employee_id FROM employee;



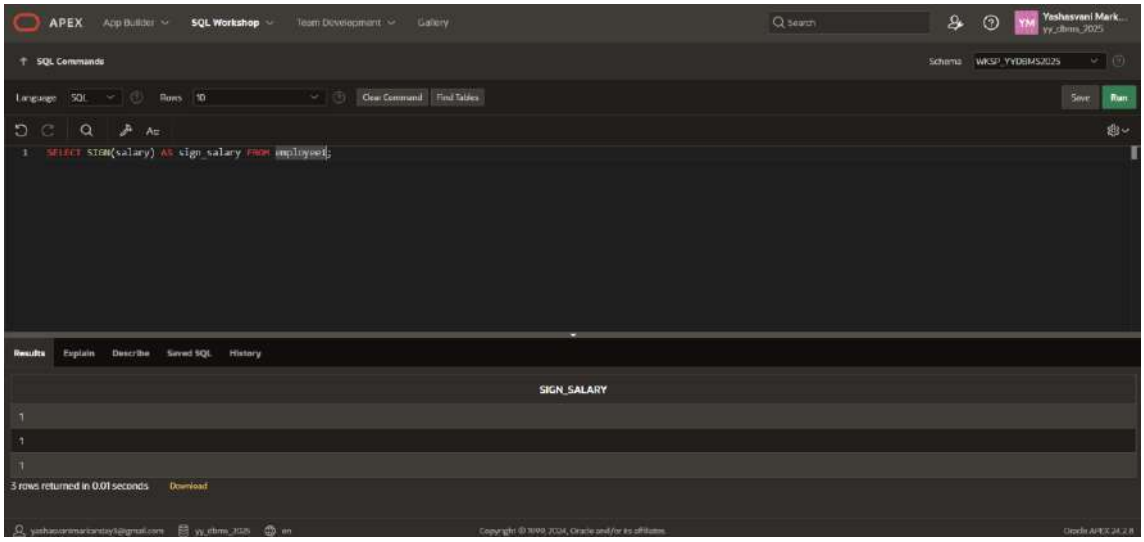
51. SELECT ABS(salary) AS abs_salary FROM employee;



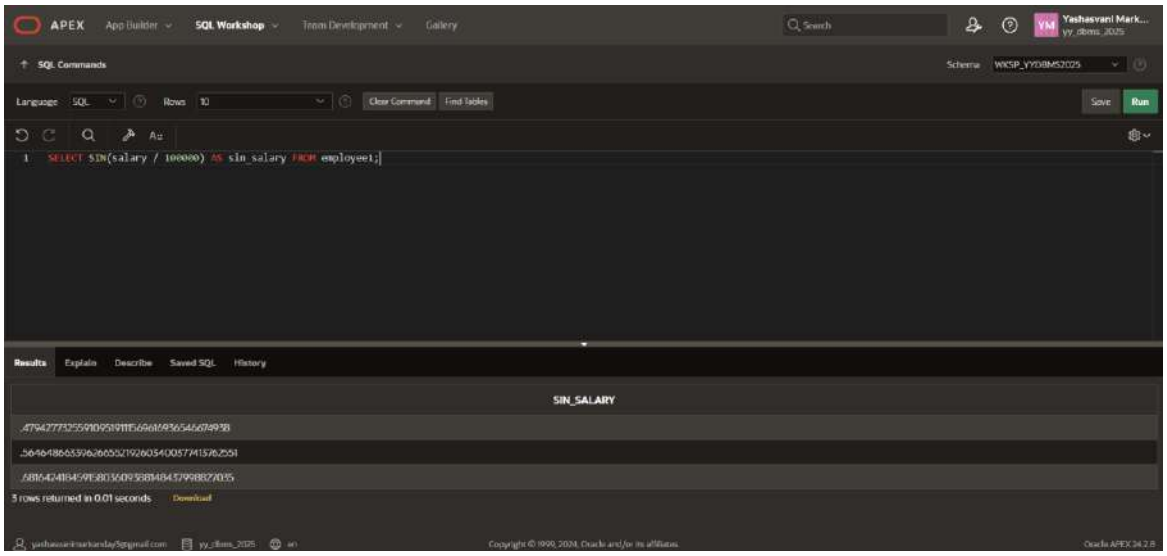
52. SELECT ACOS(salary / 100000) AS acos_salary FROM employee;



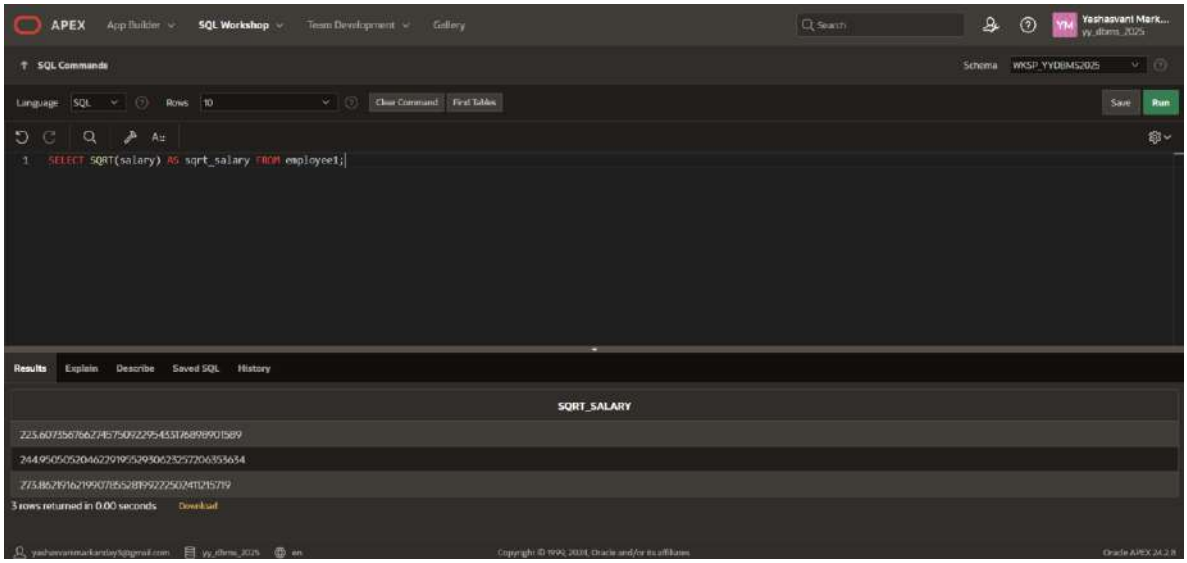
53. SELECT SIGN(salary) AS sign_salary FROM employee;



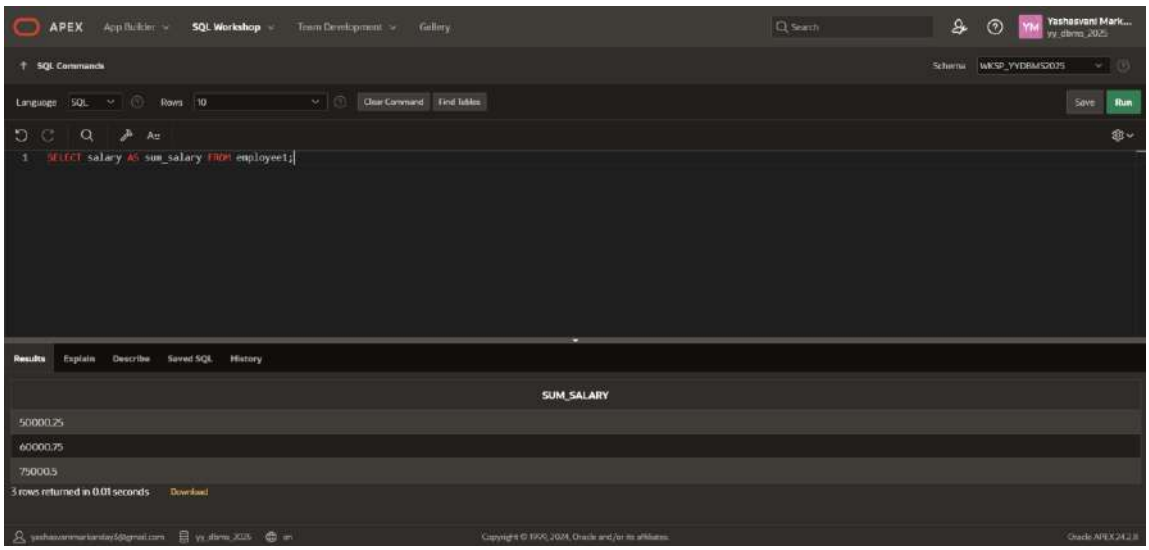
54. SELECT SIN(salary / 100000) AS sin_salary FROM employee;



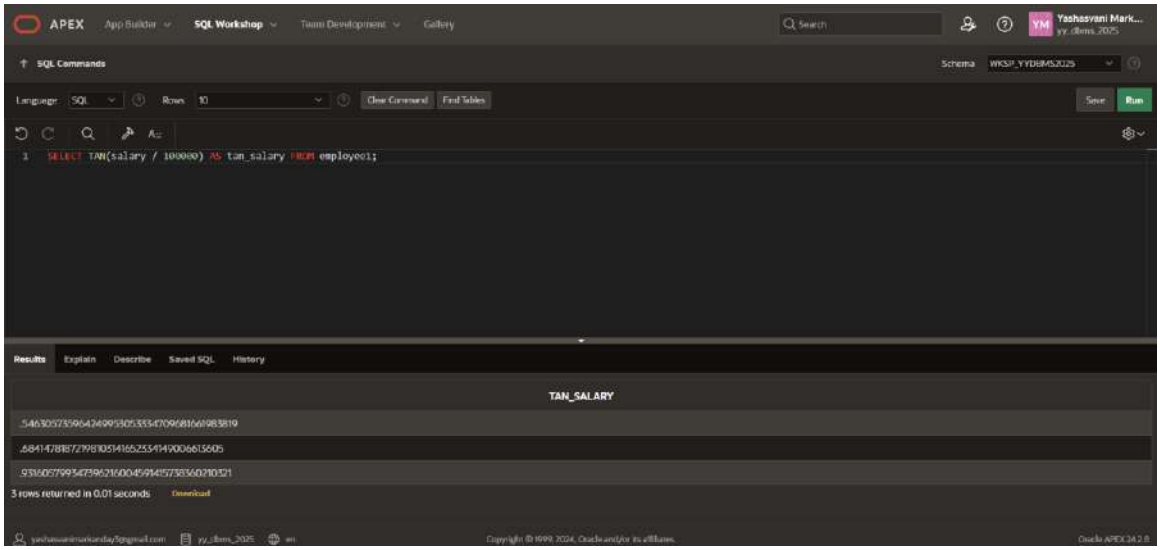
55. SELECT SQRT(salary) AS sqrt_salary FROM employee;



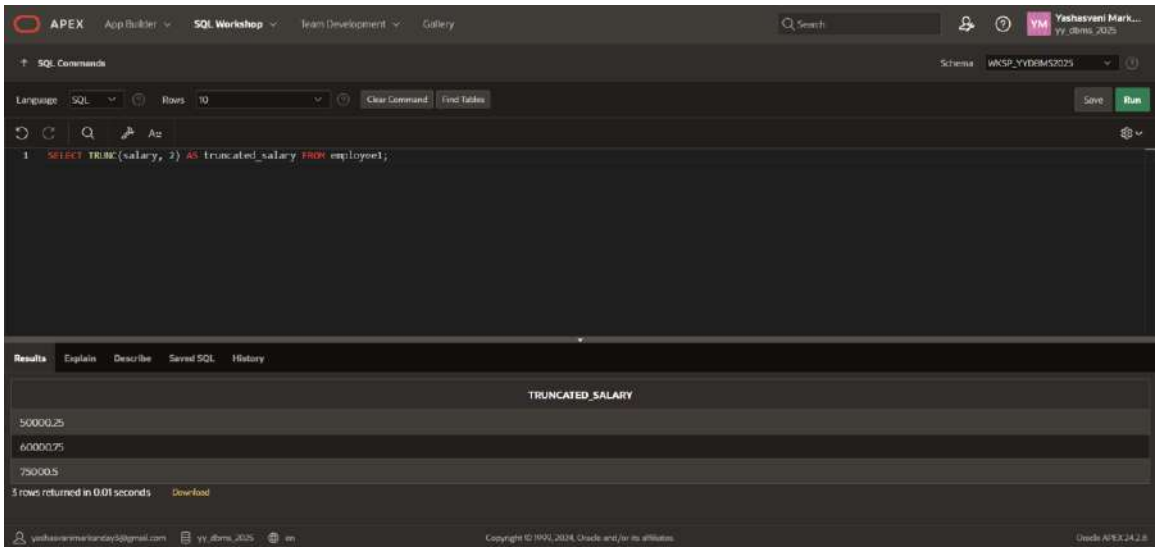
56. SELECT salary AS sum_salary FROM employee;



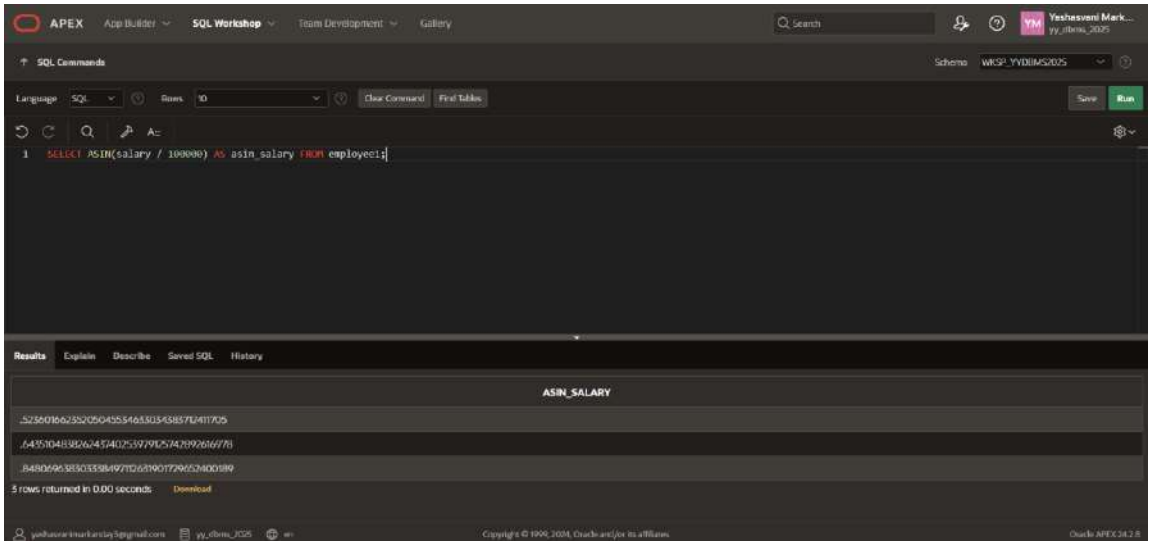
57. SELECT TAN(salary / 100000) AS tan_salary FROM employee;



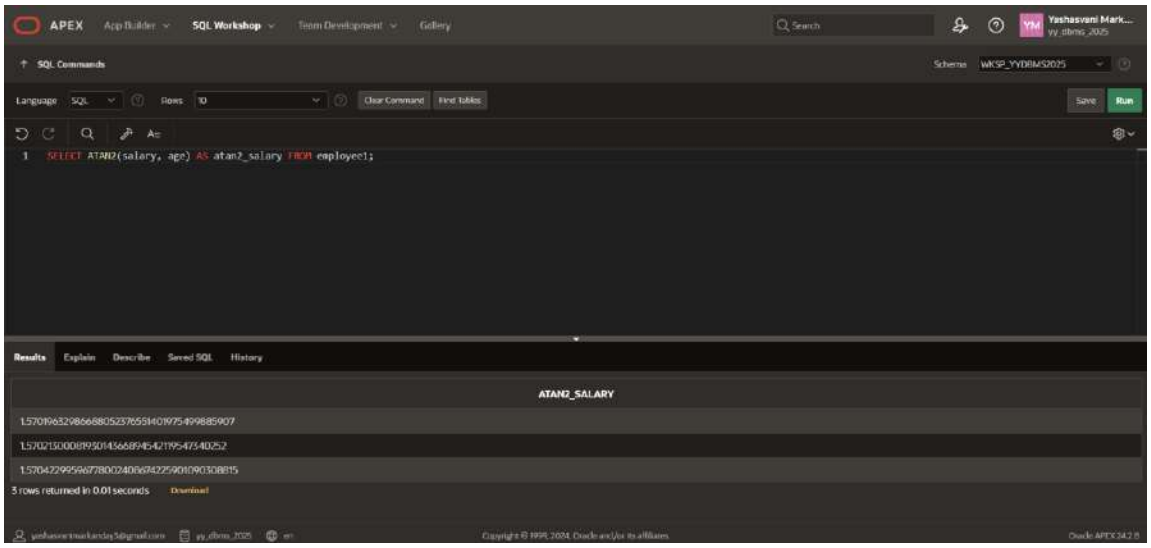
58. SELECT TRUNCATE(salary, 2) AS truncated_salary FROM employee;



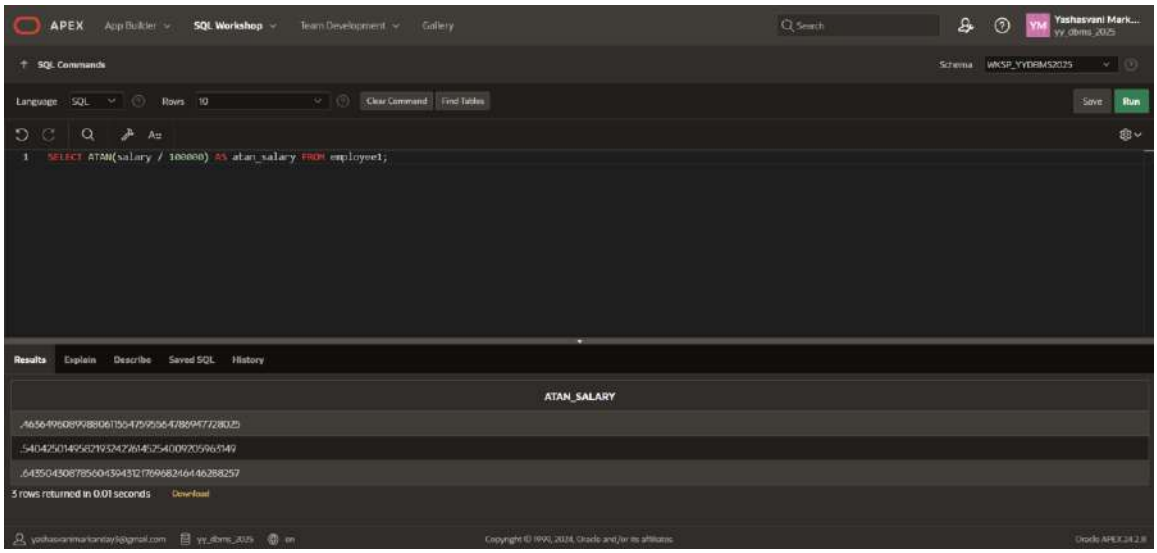
59. SELECT ASIN(salary / 100000) AS asin_salary FROM employee;



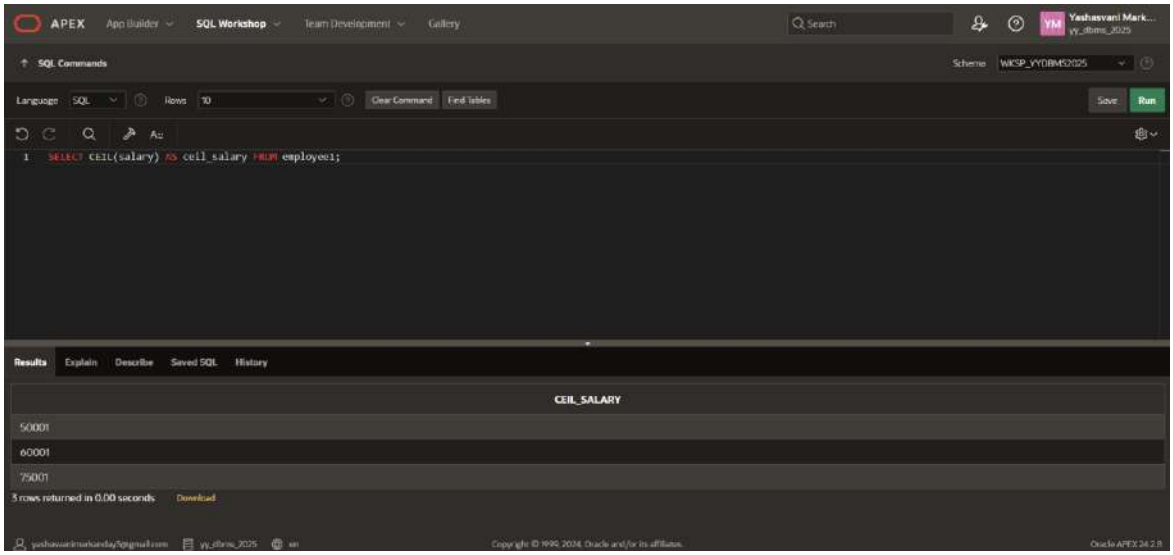
60. SELECT ATAN2(salary, age) AS atan2_salary FROM employee;



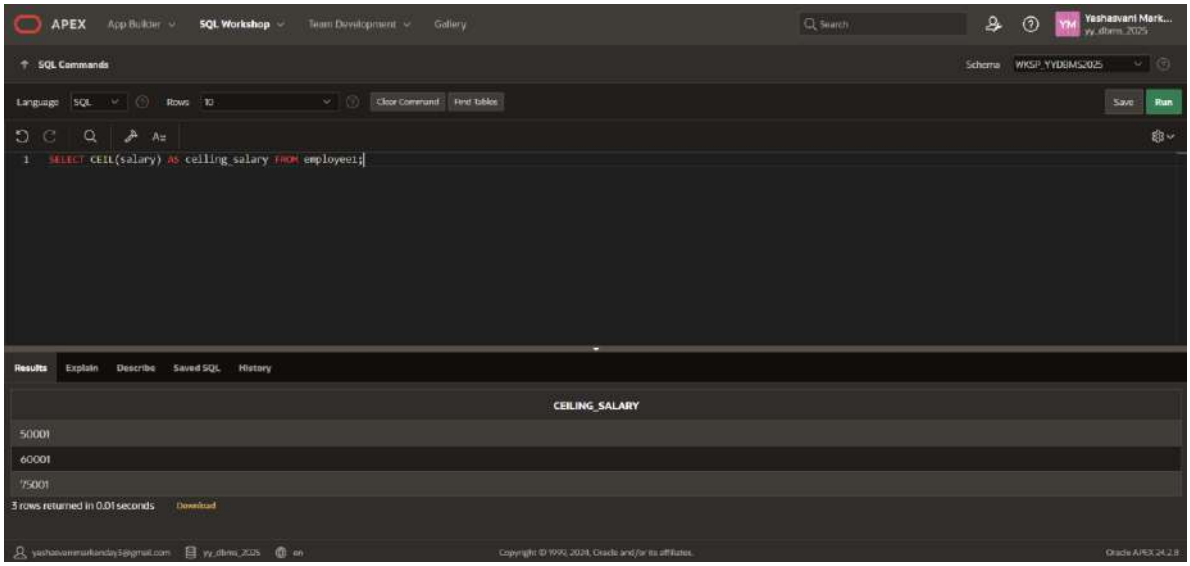
61. SELECT ATAN(salary / 100000) AS atan_salary FROM employee;



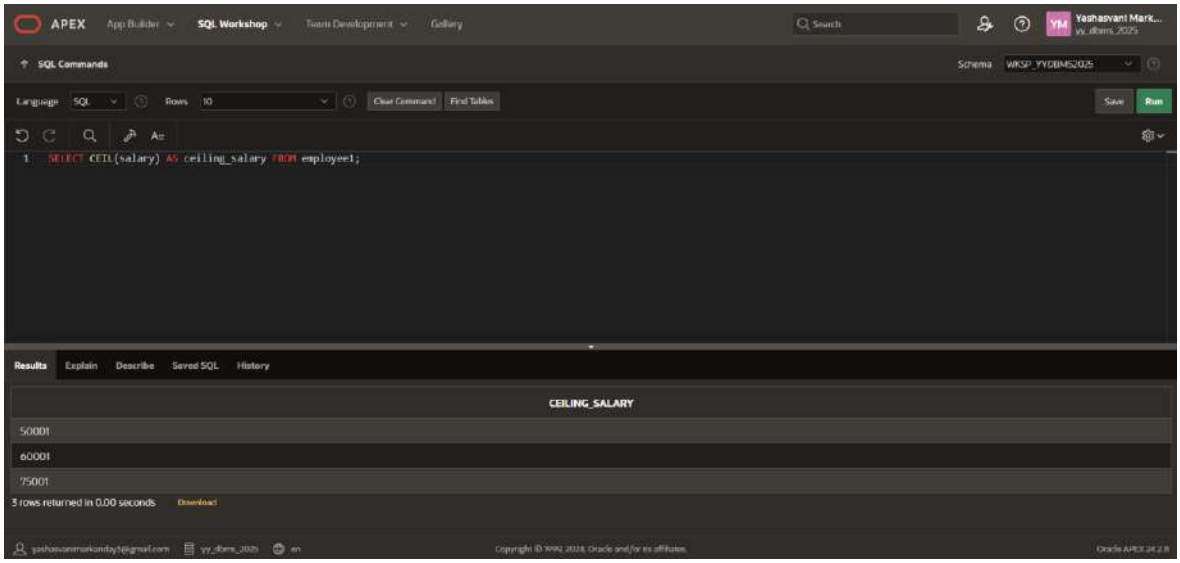
62. SELECT salary AS avg_salary FROM employee;



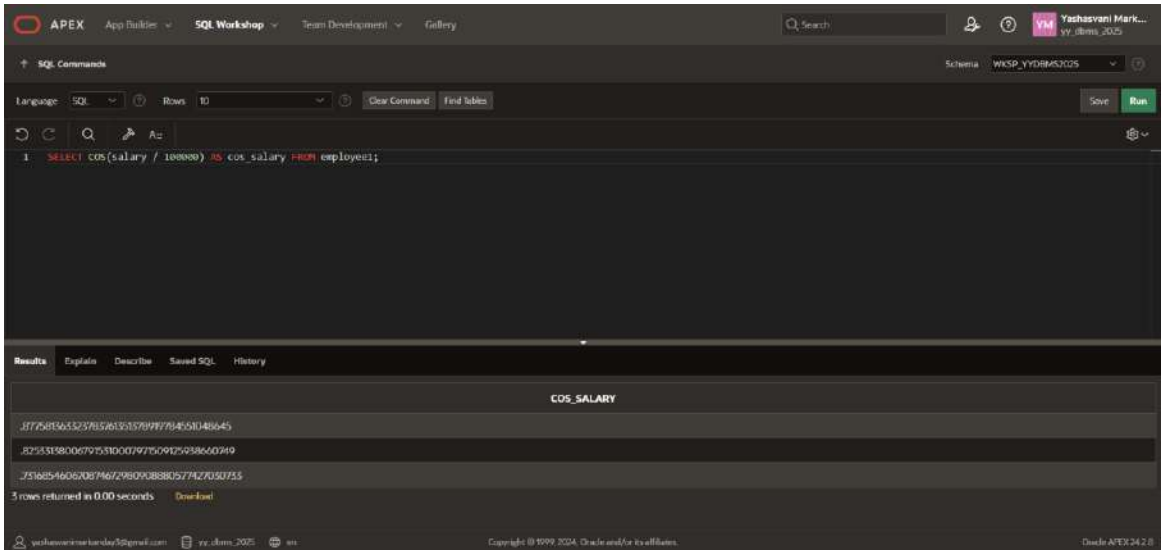
63. SELECT CEIL(salary) AS ceil_salary FROM employee;



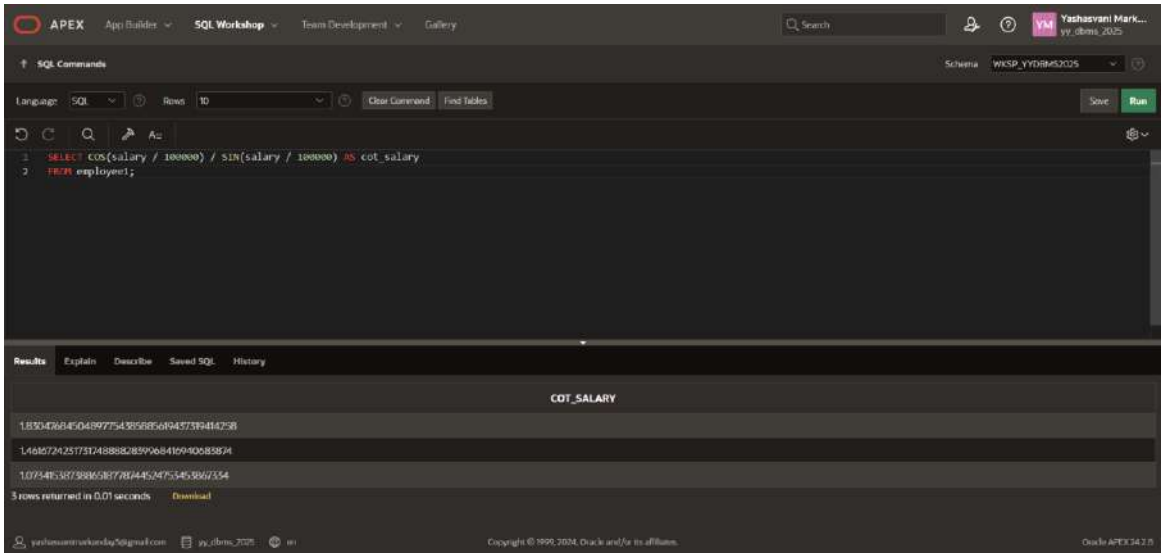
64. SELECT CEILING(salary) AS ceiling_salary FROM employee;



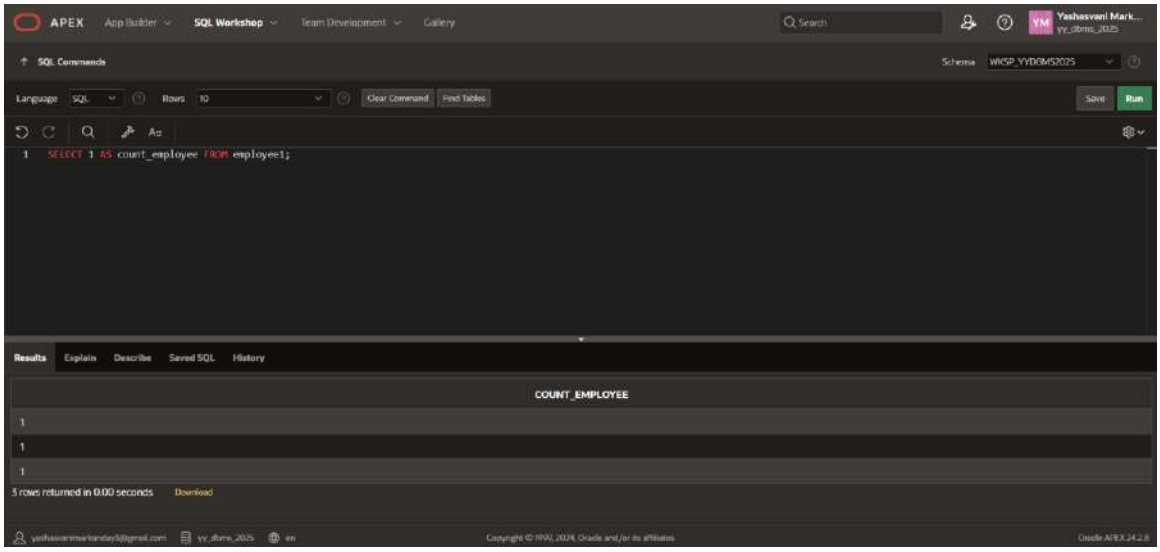
65. SELECT COS(salary / 100000) AS cos_salary FROM employee;



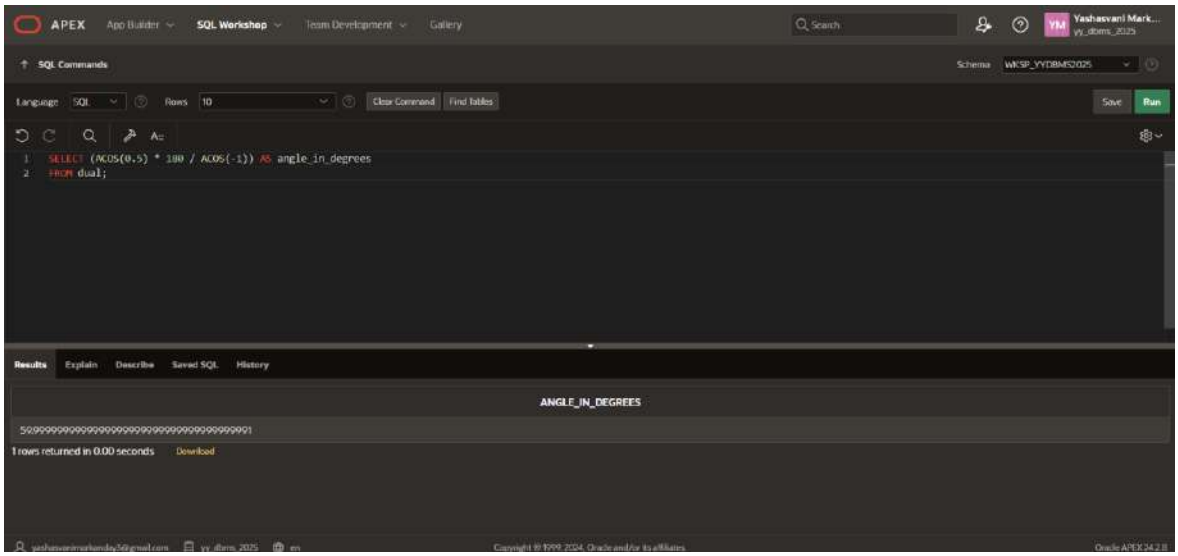
66. SELECT COT(salary / 100000) AS cot_salary FROM employee;



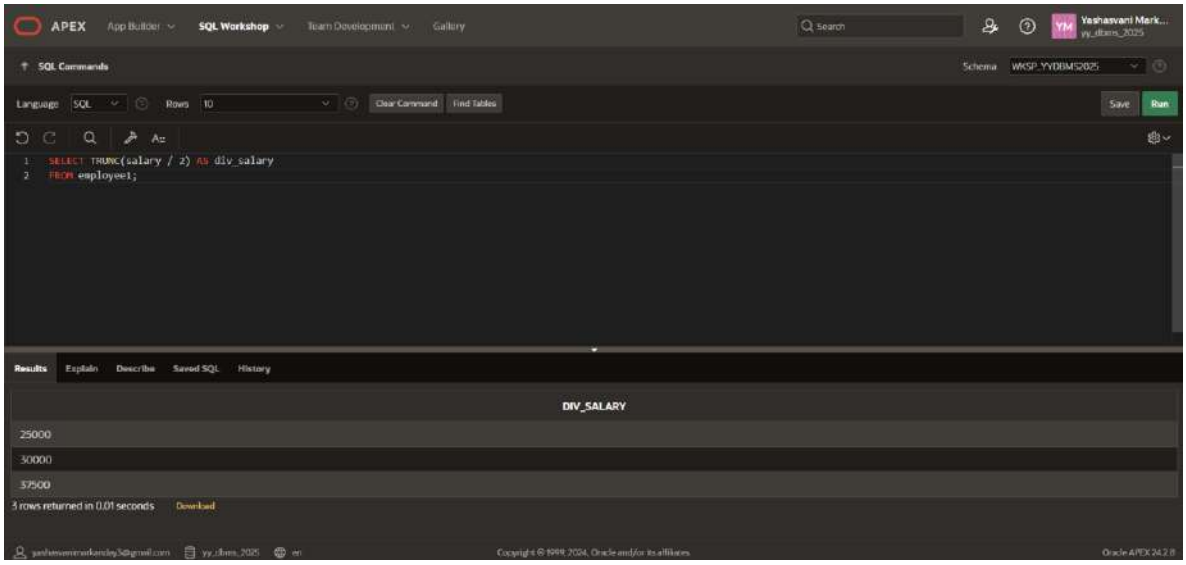
67. SELECT 1 AS count_employee FROM employee;



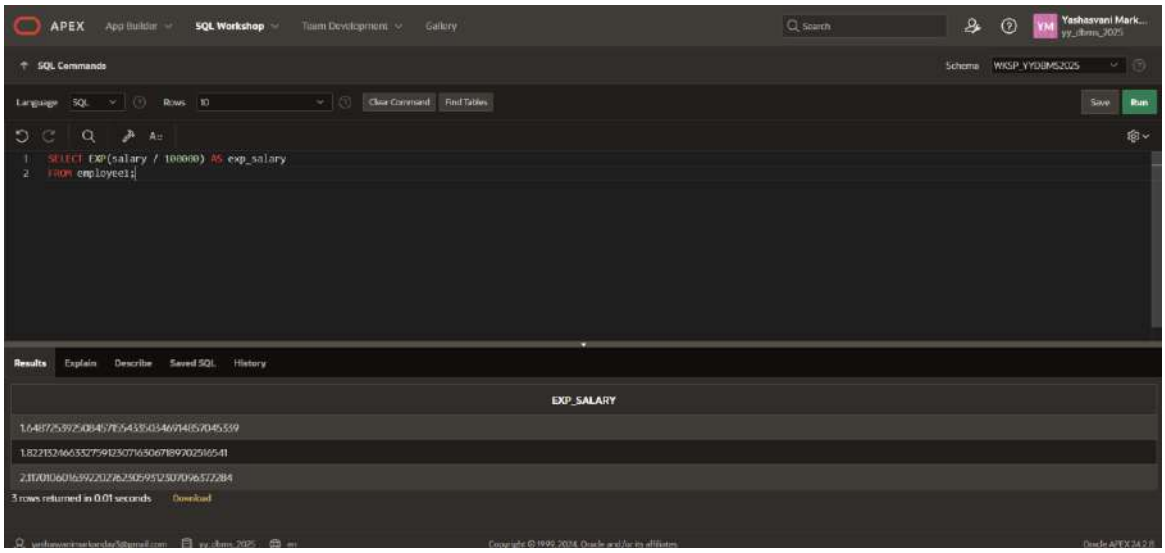
68. SELECT DEGREES(salary / 100000) AS degrees_salary FROM employee;



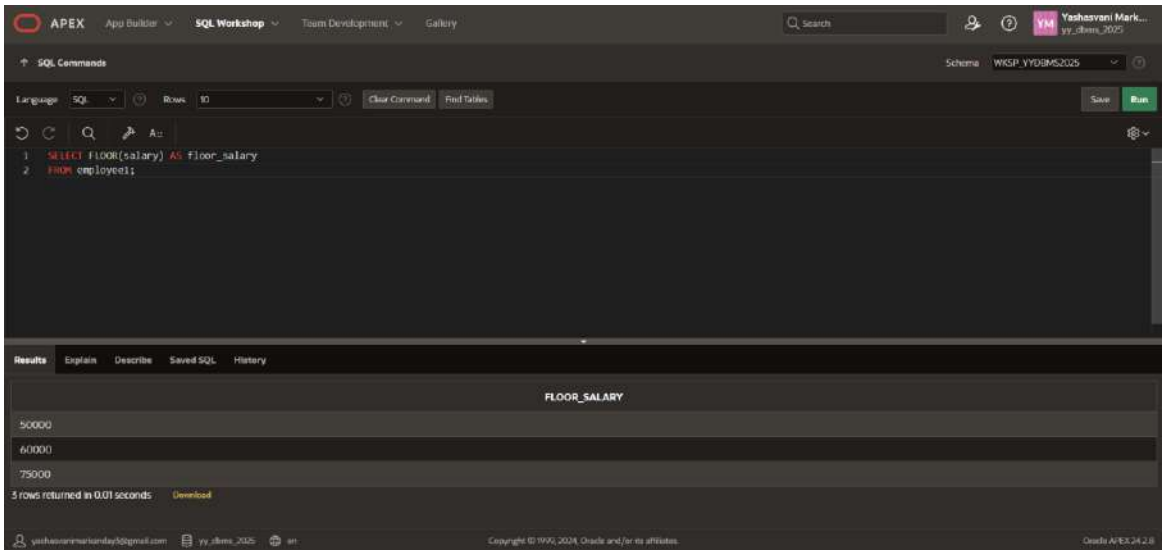
69. SELECT salary DIV 2 AS div_salary FROM employee;



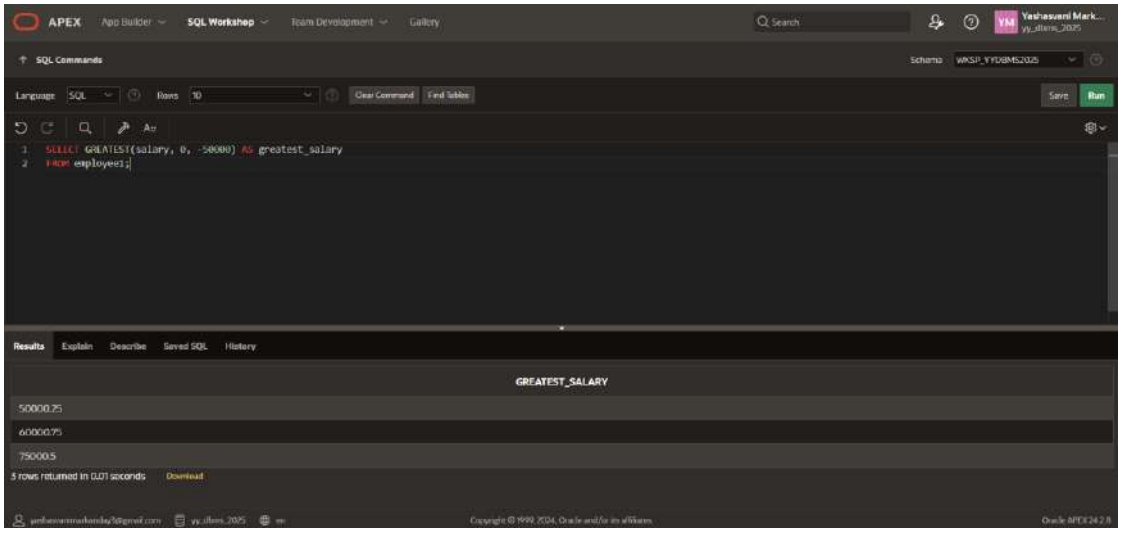
70. SELECT EXP(salary / 100000) AS exp_salary FROM employee;



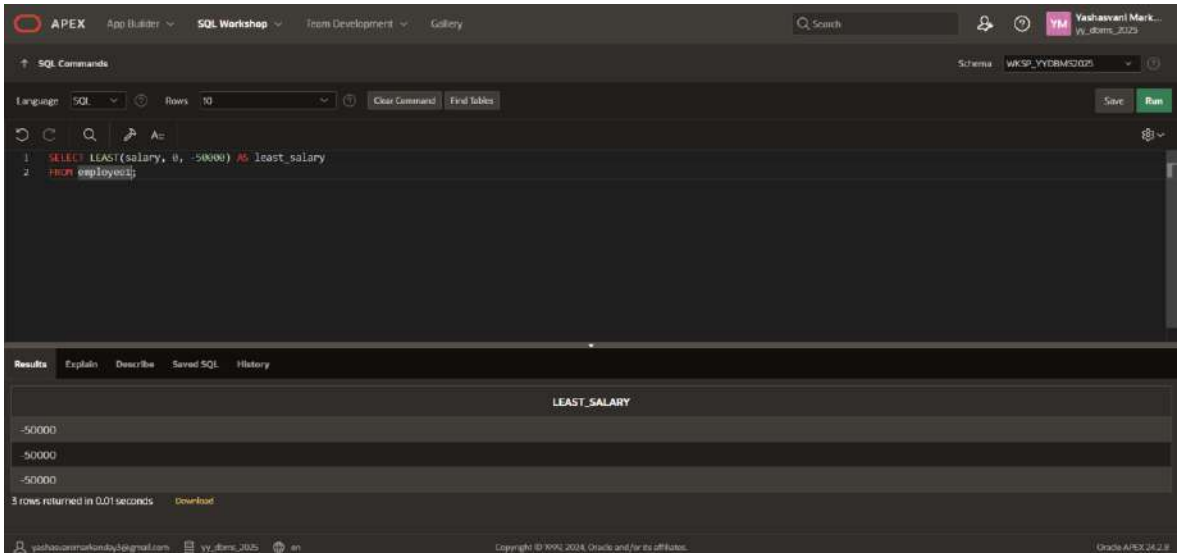
71. SELECT FLOOR(salary) AS floor_salary FROM employee;



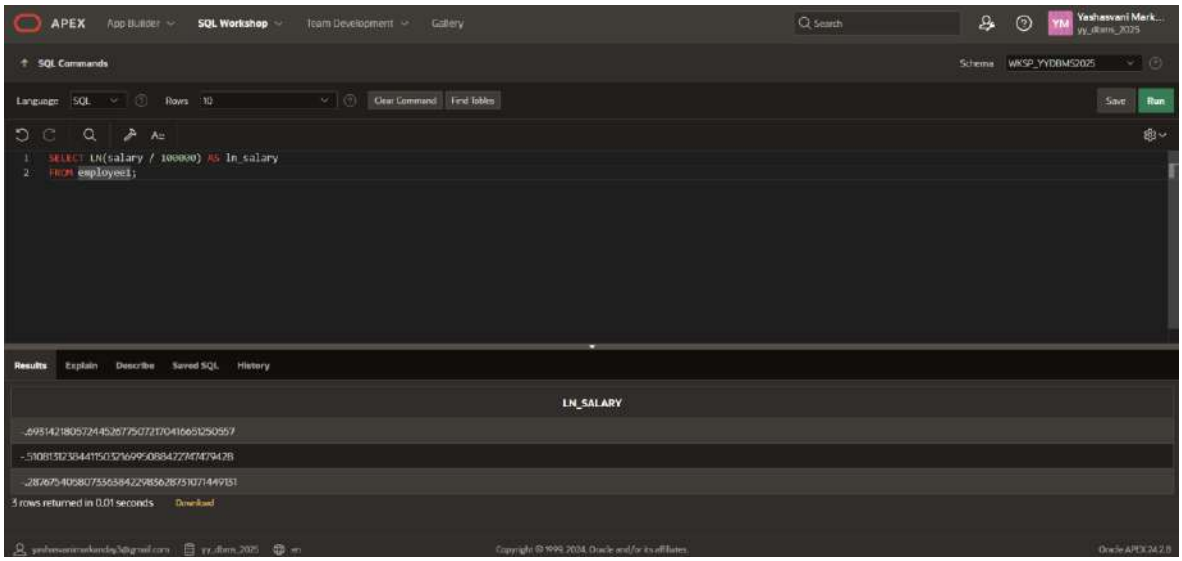
72. SELECT GREATEST(salary, 0, -50000) AS greatest_salary FROM employee;



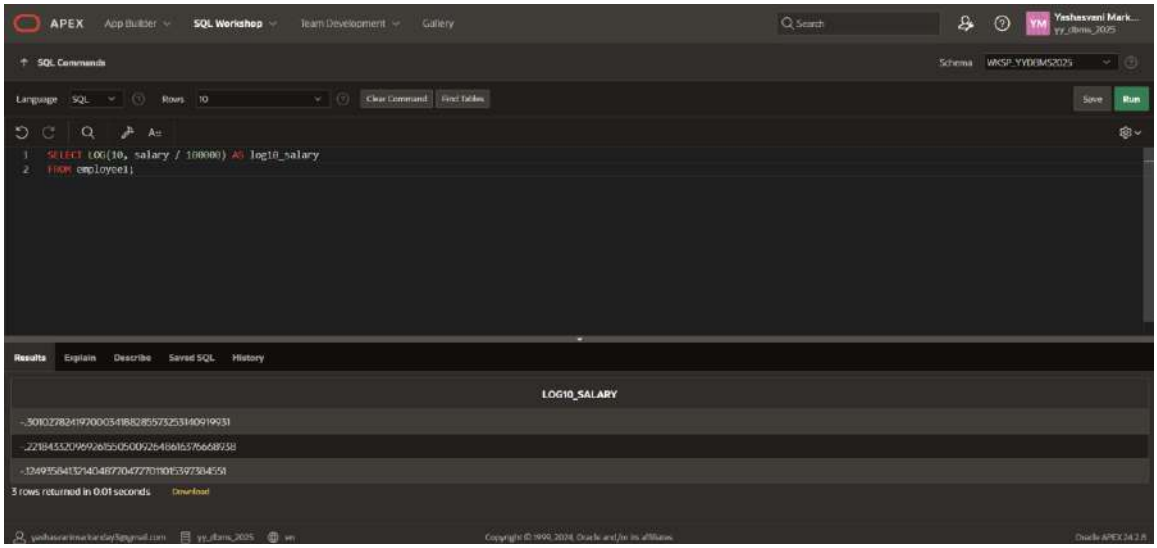
73. SELECT LEAST(salary, 0, -50000) AS least_salary FROM employee;



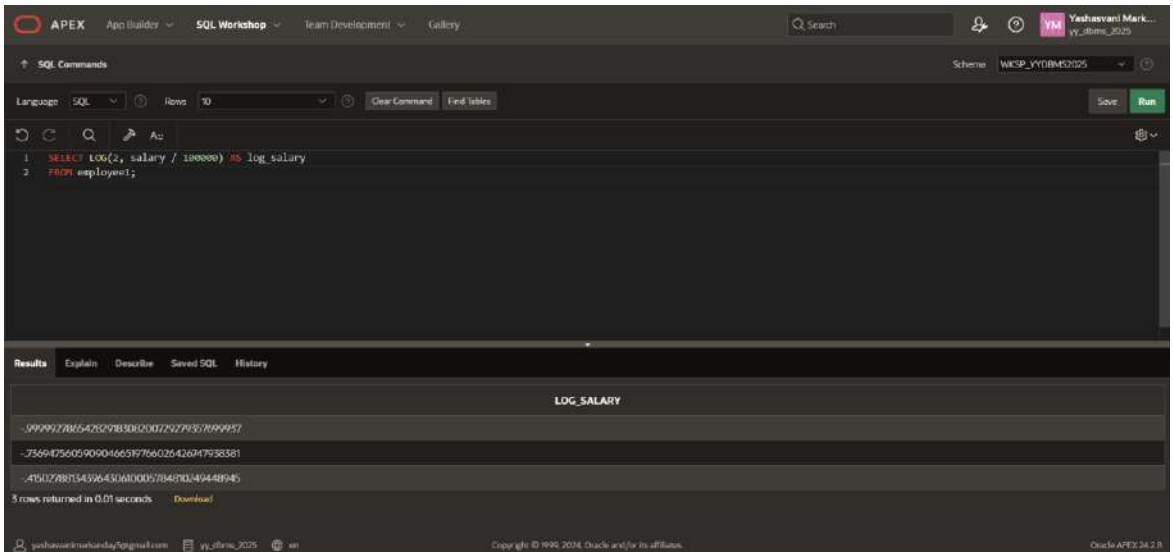
74. SELECT LN(salary / 100000) AS ln_salary FROM employee;



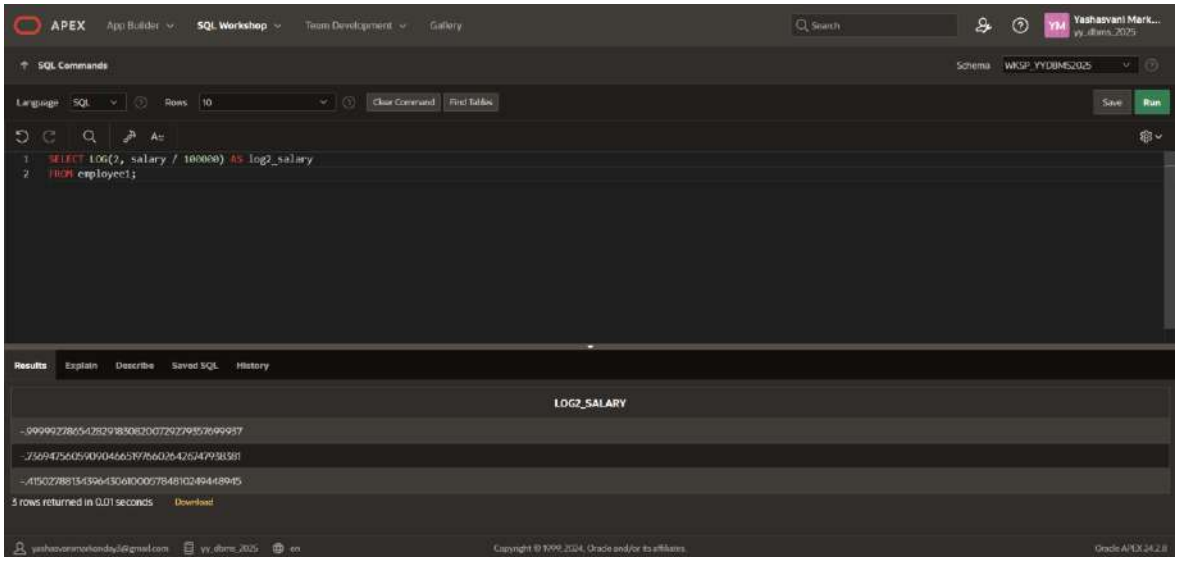
75. SELECT LOG10(salary / 100000) AS log10_salary FROM employee;



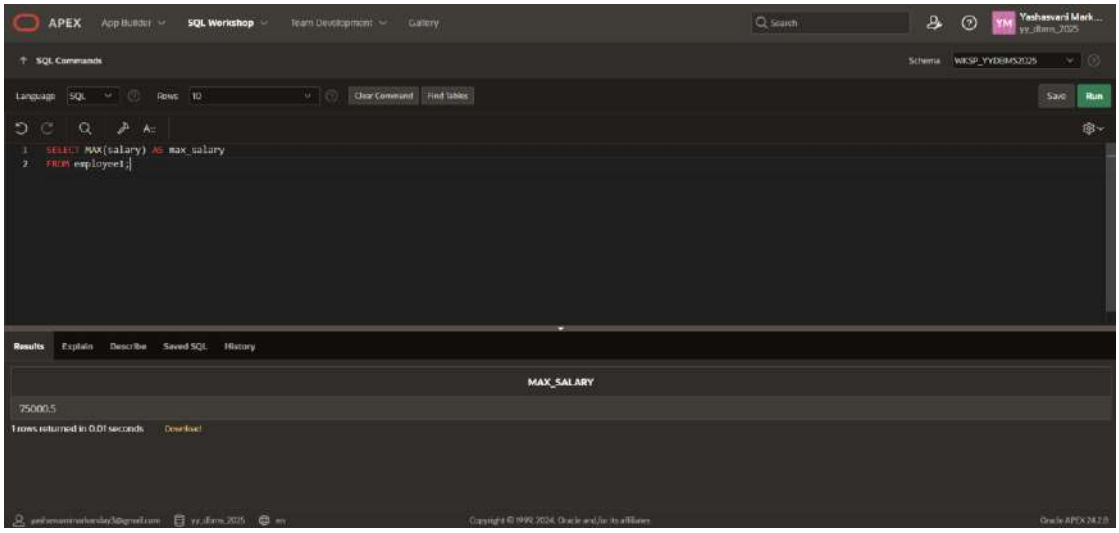
76. SELECT LOG(salary / 100000, 2) AS log_salary FROM employee;



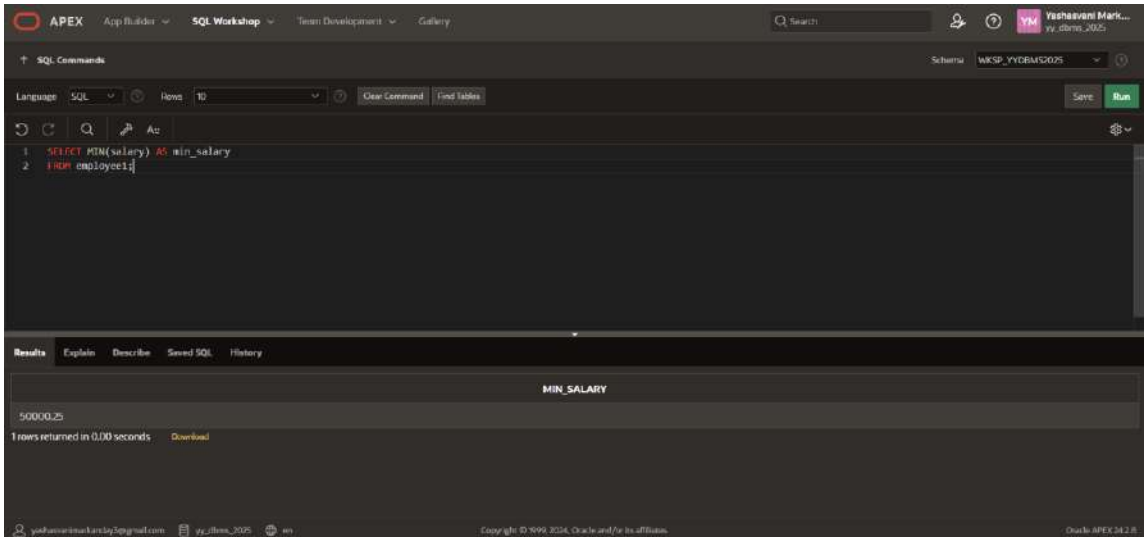
77. SELECT LOG2(salary / 100000) AS log2_salary FROM employee;



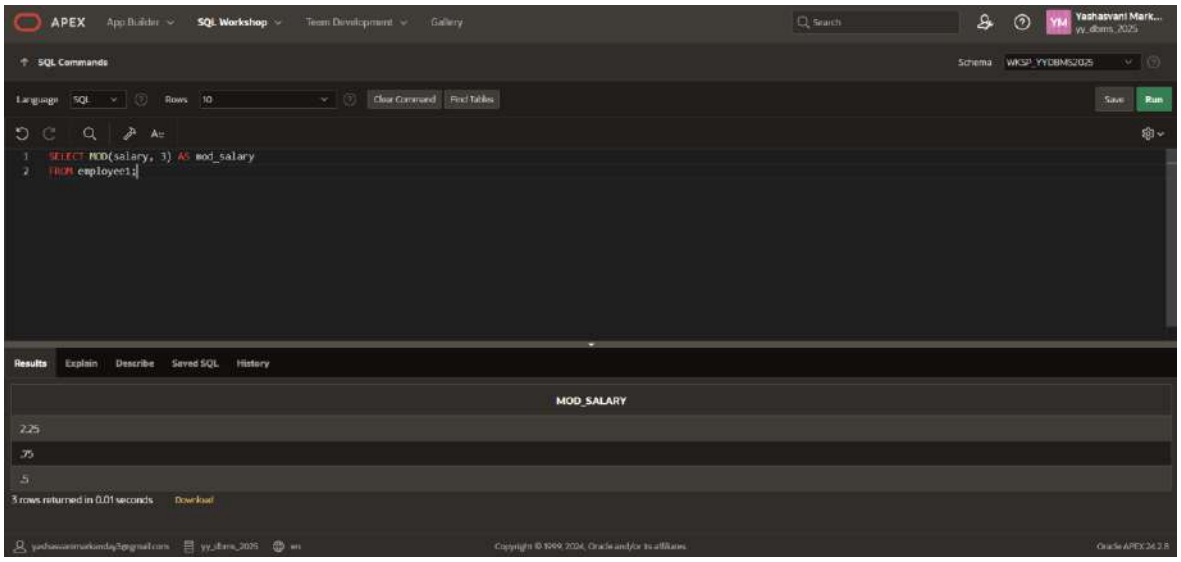
78. SELECT salary AS max_salary FROM employee;



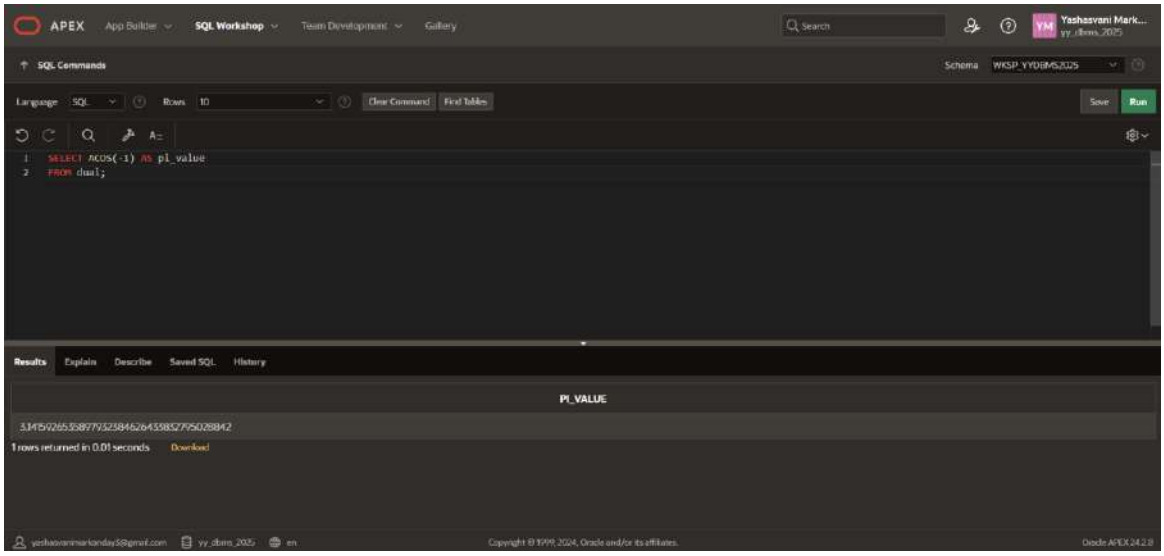
79. SELECT salary AS min_salary FROM employee;



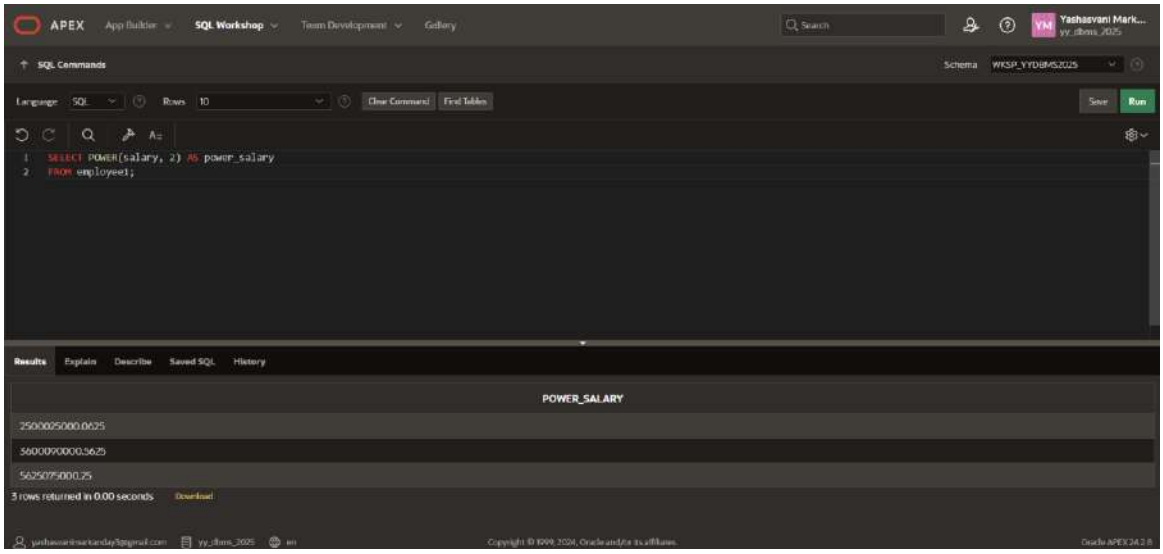
80. SELECT MOD(salary, 3) AS mod_salary FROM employee;



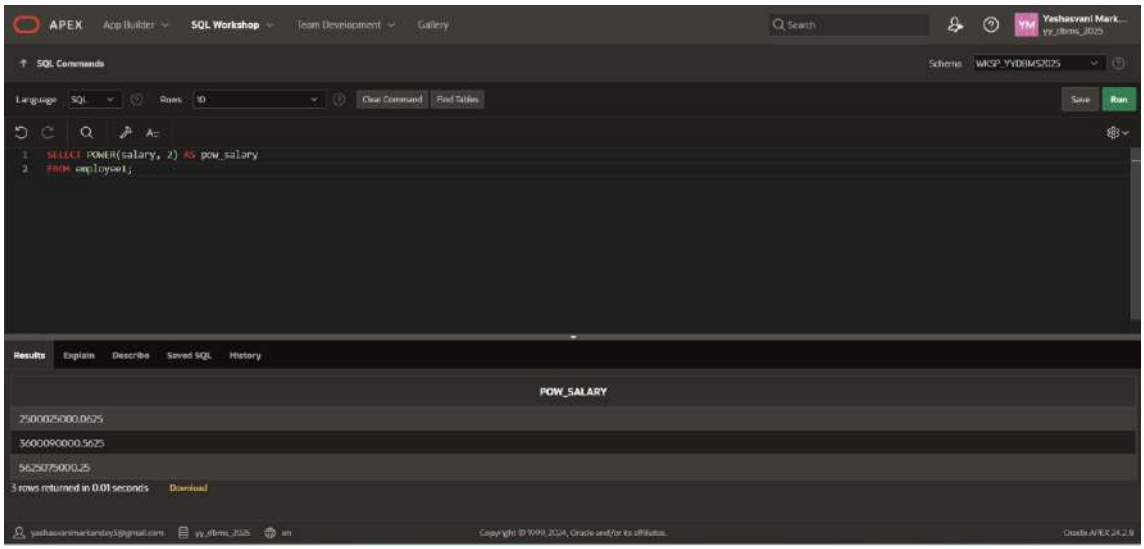
81. SELECT PI() AS pi_value FROM employee;



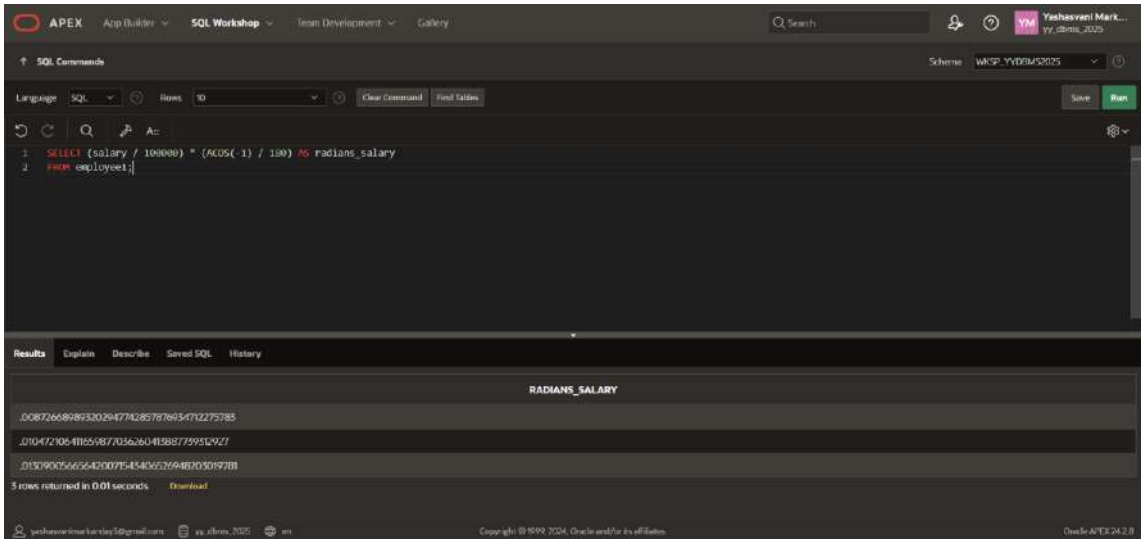
82. SELECT POWER(salary, 2) AS power_salary FROM employee;



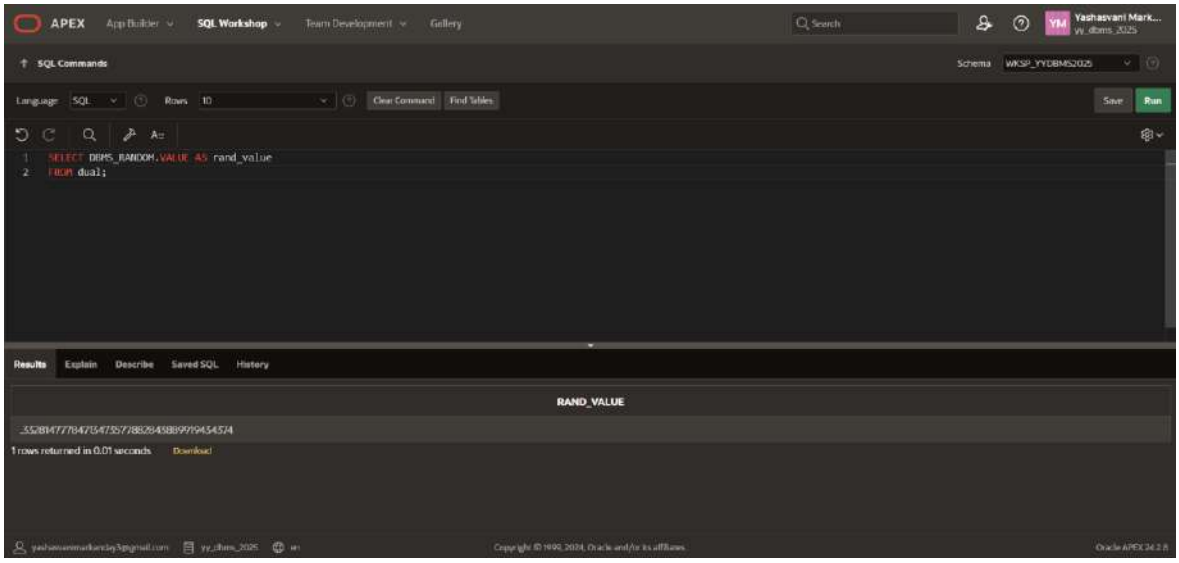
83. SELECT POW(salary, 2) AS pow_salary FROM employee;



84. SELECT RADIANS(salary / 100000) AS radians_salary FROM employee;

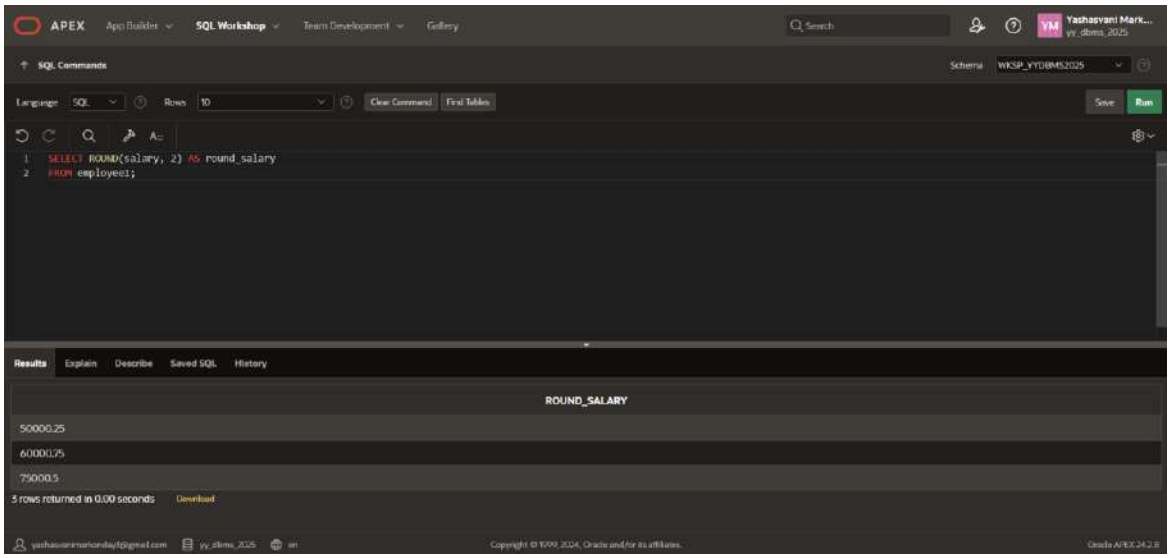


85. SELECT RAND() AS rand_value FROM employee;



++++++++++++++++++

86. SELECT ROUND(salary, 2) AS round_salary FROM employee;



CONCLUSION

In this lab, we explored the use of SQL inbuilt functions for data manipulation and analysis. We practiced aggregate functions like COUNT, SUM, AVG, MIN, and MAX, as well as string functions such as CONCAT, LENGTH, and REPLACE. Mathematical functions including ABS, SQRT, LOG, and POWER were also implemented to understand their role in calculations. Through this experiment, we gained practical knowledge of how inbuilt functions simplify complex queries and enhance database operations.

UNIVERSITY OF PETROLEUM AND ENERGY STUDIES

Program Name: B. Tech CS **Semester: III**

Name: Yashasvani Markanday **SAP ID: 590011386**

Batch: B2 **Subject: DBMS**

LAB - 5

Question 3

a. Create the tables for Company database as per ER diagram of Exp 2.

TABLE 1: EMPLOYEE

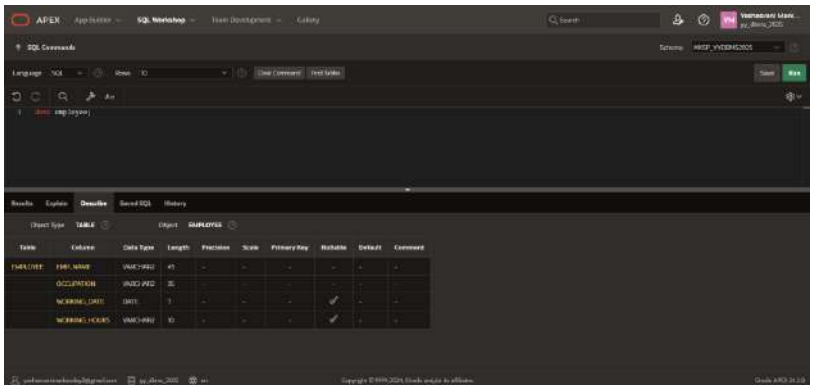


TABLE 2: DEPARTMENT

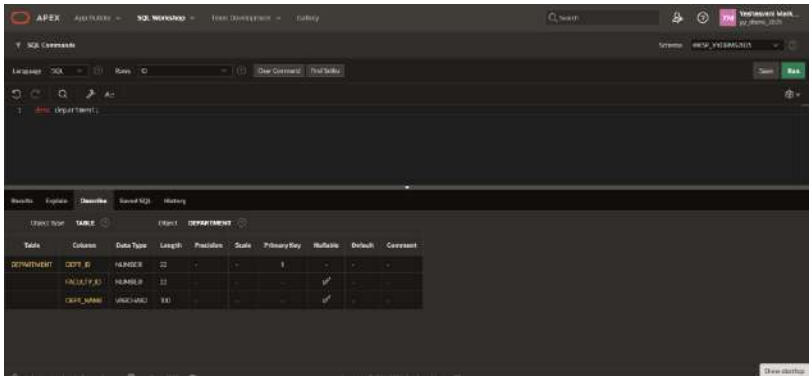


TABLE 3: DEPT_LOCATIONS

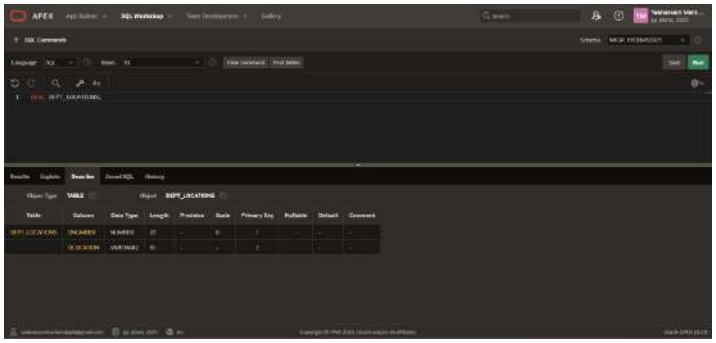


TABLE 4: PROJECT

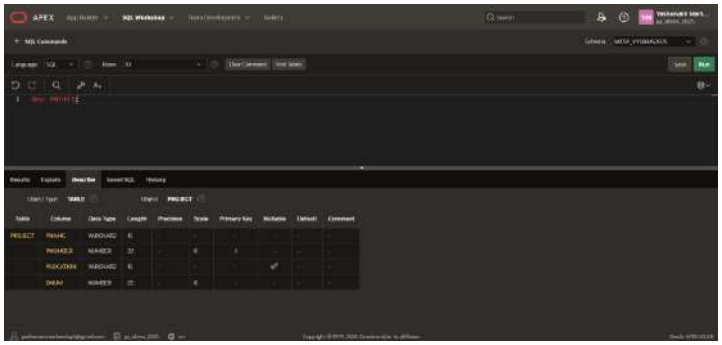


TABLE 5: WORKS_ON

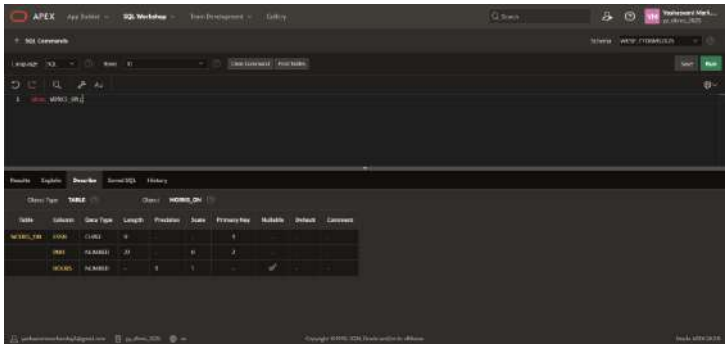
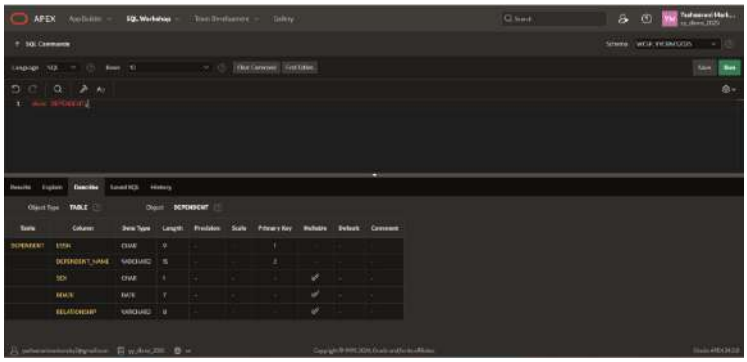
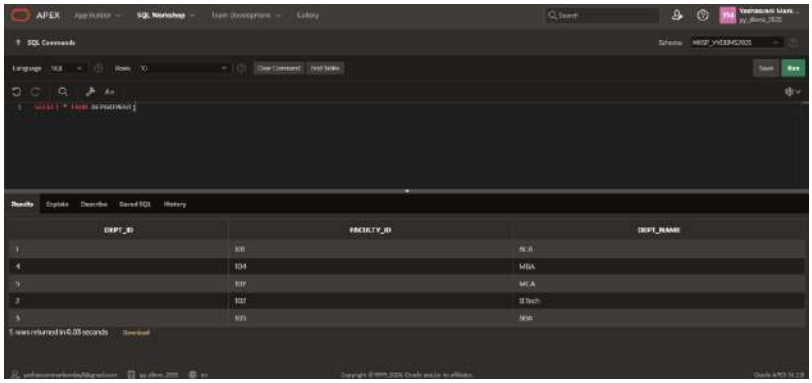


TABLE 6: DEPENDENT

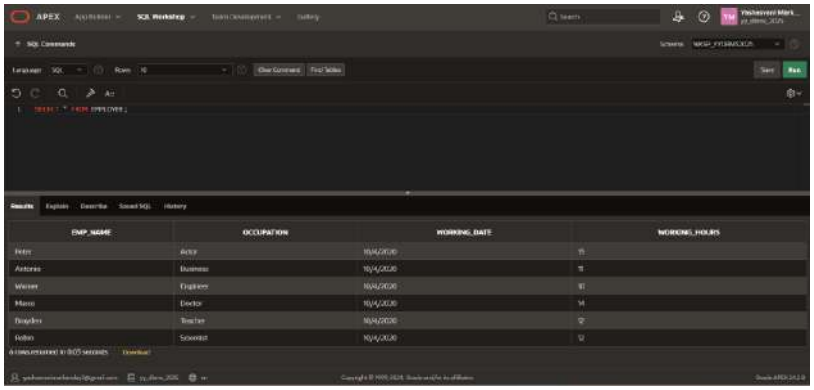


b. Insert the following data into their respective tables of Company database.



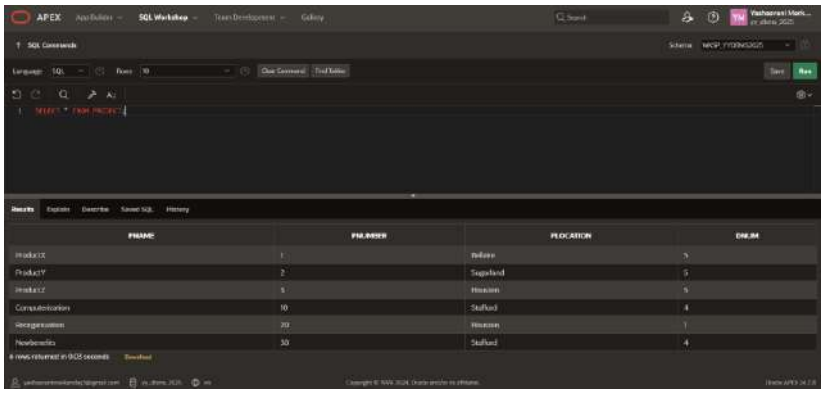
DEPT_ID	FACULTY_ID	DEPT_NAME
1	101	BCA
4	104	MBA
5	105	MCA
2	102	B.Tech
3	103	BBA

5 rows returned in 0.03 seconds



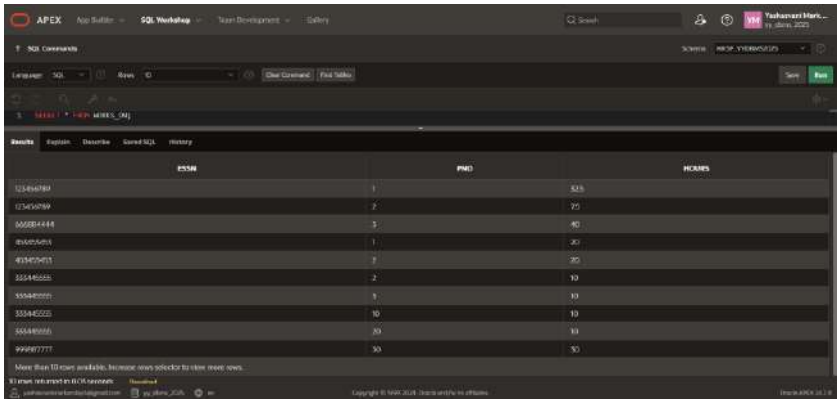
EMP_NAME	OCCUPATION	WORKING_DATE	WORKING_HOURS
Peter	Actor	10/4/2020	11
Antonio	Business	10/4/2020	11
Warren	Engineer	10/4/2020	10
Mason	Doctor	10/4/2020	14
Braydon	Teacher	10/4/2020	12
Rohin	Scientist	10/4/2020	12

6 rows returned in 0.05 seconds



PNAME	PNUMBER	PLOCATION	DNUM
ProductX	1	Bellaire	5
ProductY	2	Sugarland	5
ProductZ	3	Houston	5
Computerization	10	Stafford	4
Reorganization	20	Houston	1
Newbenefits	30	Stafford	4

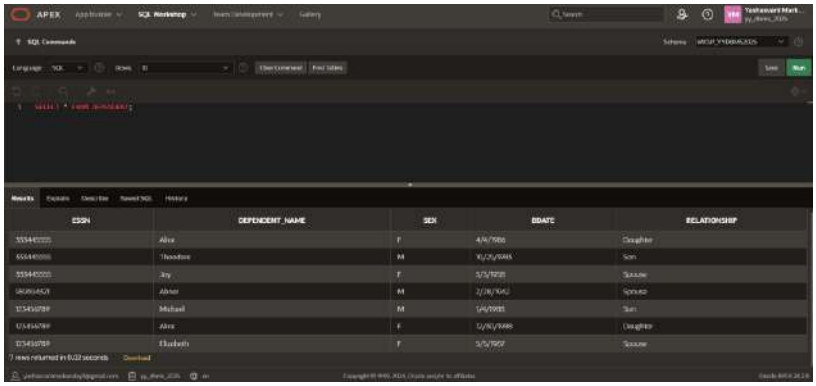
6 rows returned in 0.05 seconds



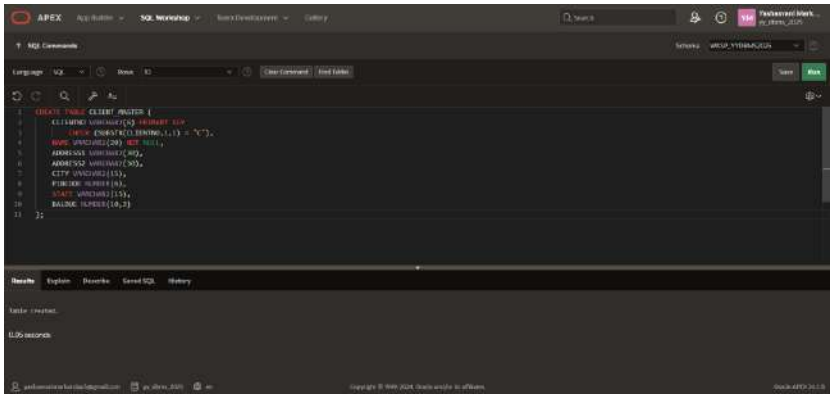
ESSN	PNO	HOURS
123456789	1	32.5
123456789	2	7.5
666884444	3	40
453453453	1	20
453453453	2	20
333445555	2	10
333445555	3	10
333445555	10	10
333445555	20	10
999887777	30	30

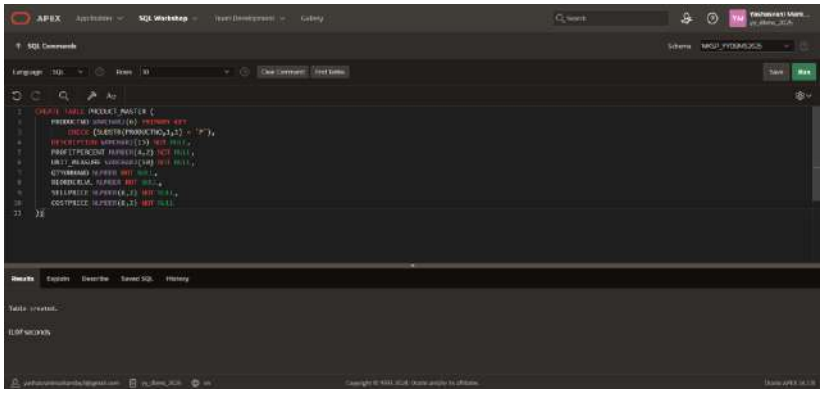
More than 10 rows available. Increase rows selector to view more rows.

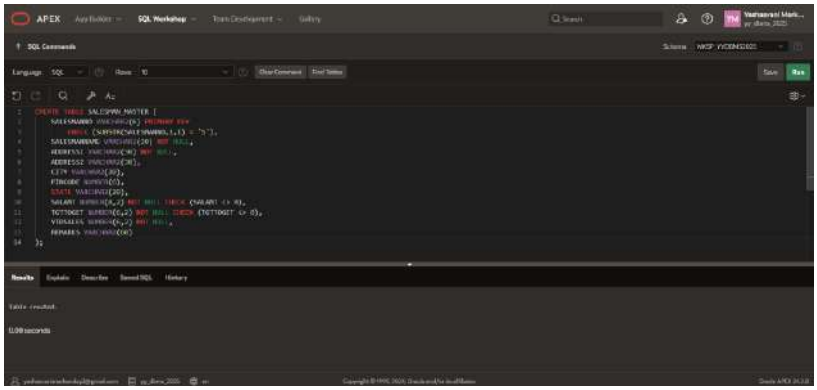
10 rows returned in 0.04 seconds



Question 4

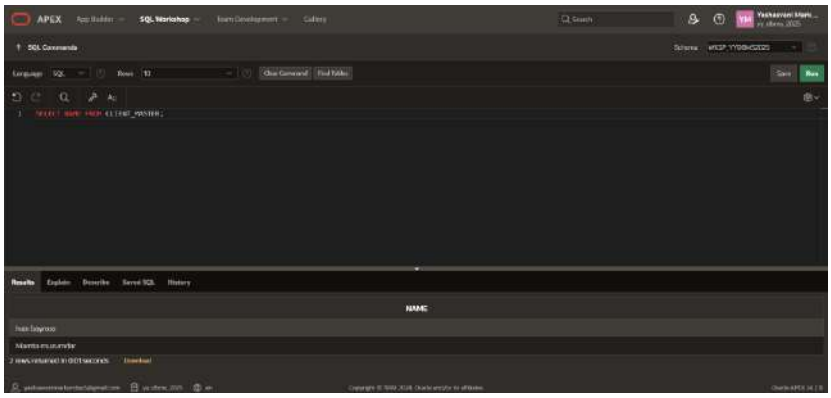




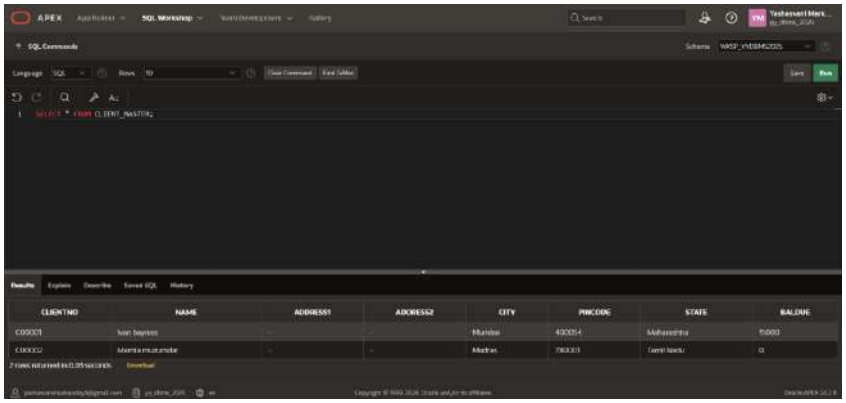


2. Exercise on retrieving records from a table.

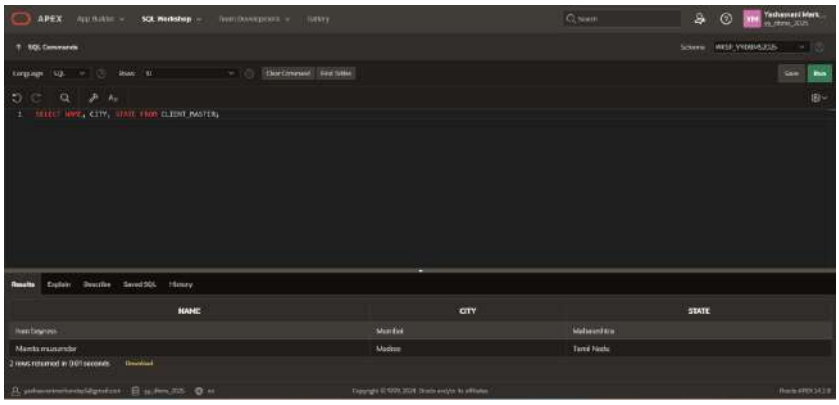
a. Find out the names of all the clients:



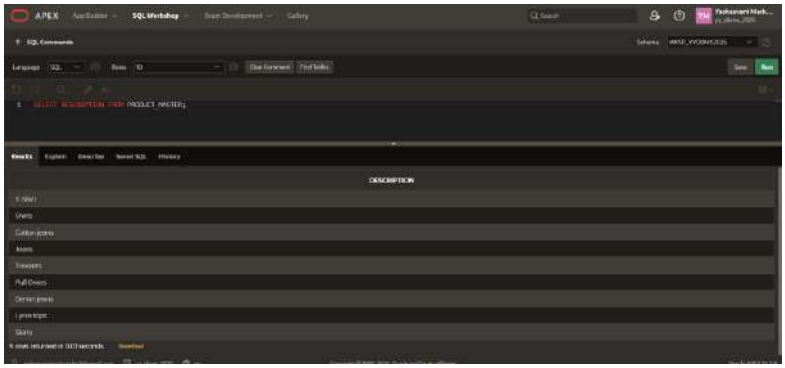
b. Retrieve the entire contents of the Client_Master table



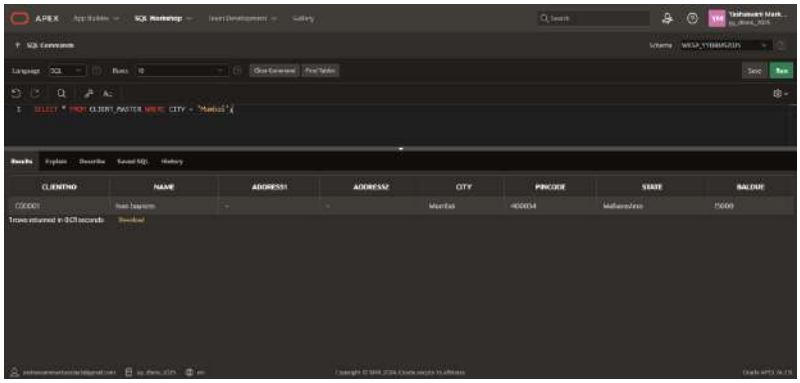
c. Retrieve the list of names, city and the state of all the clients.



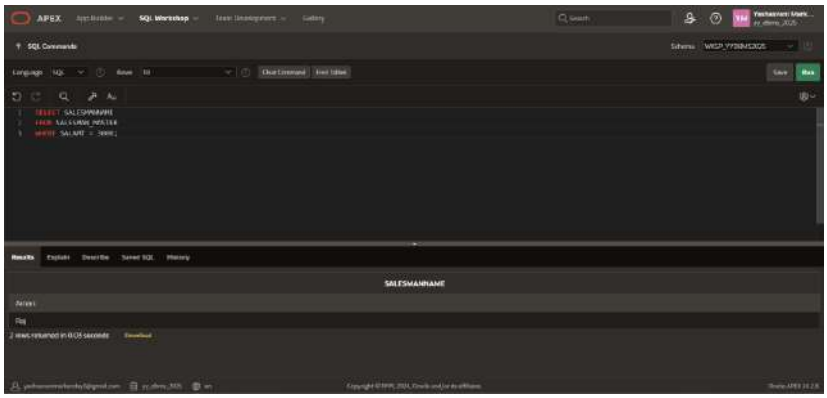
d. List the various products available from the Product_Master table.



e. List all the clients who are located in Mumbai.

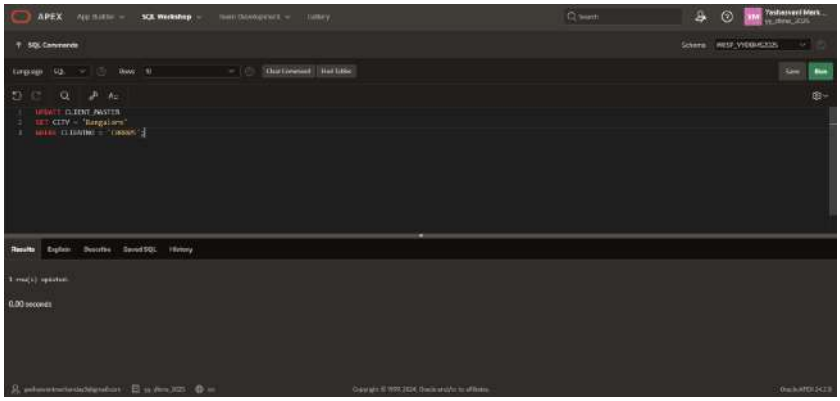


f. Find the names of salesman who have a salary equal to Rs.3000.

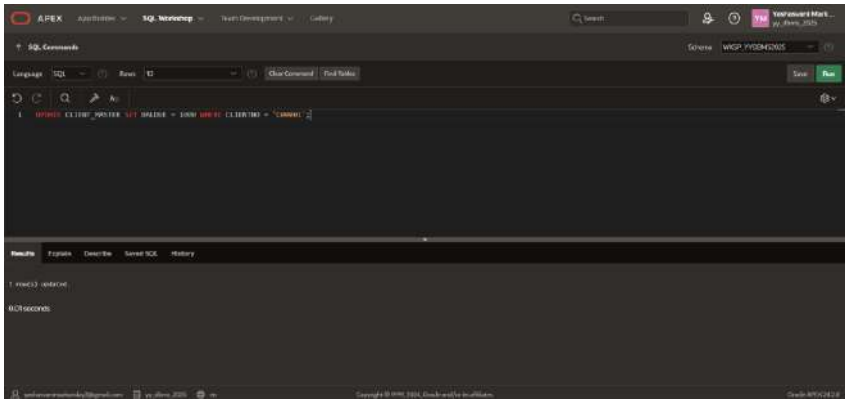


3. Exercise on updating records in a table

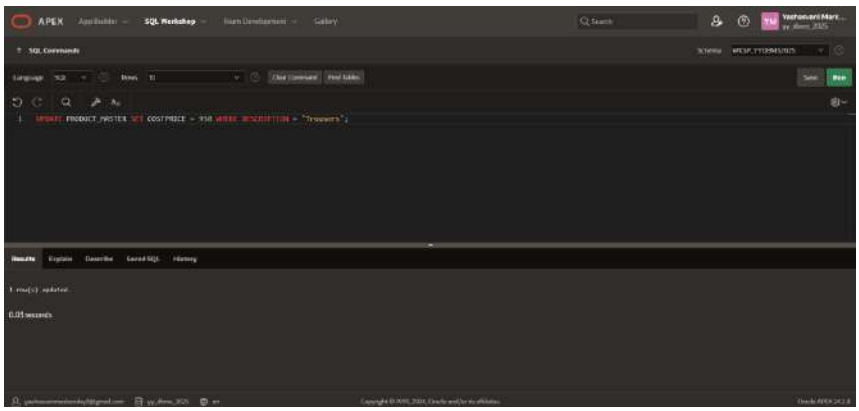
a. Change the city of ClientNo ‘C00005’ to ‘Bangalore’.



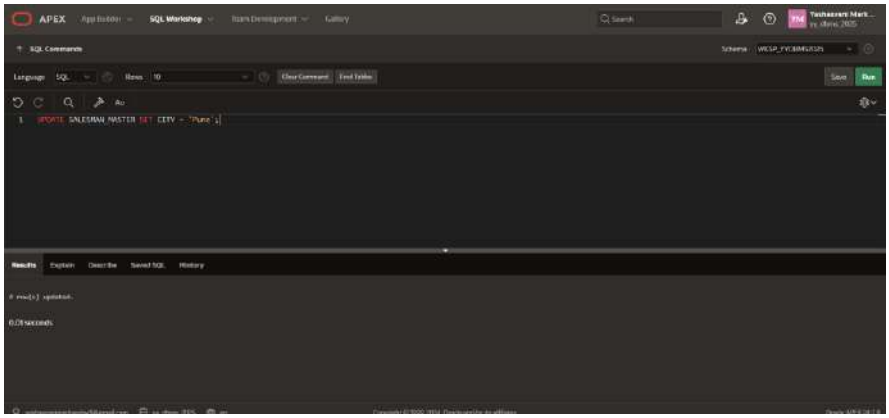
b. Change the BalDue of ClientNo ‘C00001’ to Rs.1000.



c. Change the cost price of ‘Trousers’ to rs.950.00.

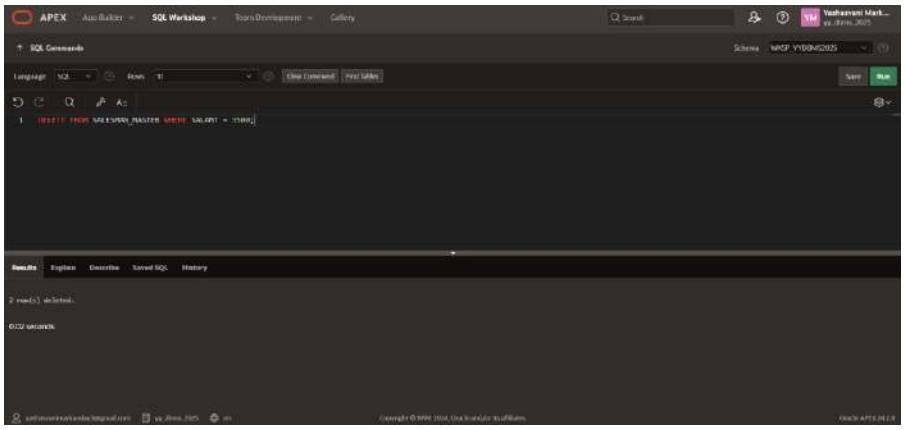


d. Change the city of the salesman to Pune.

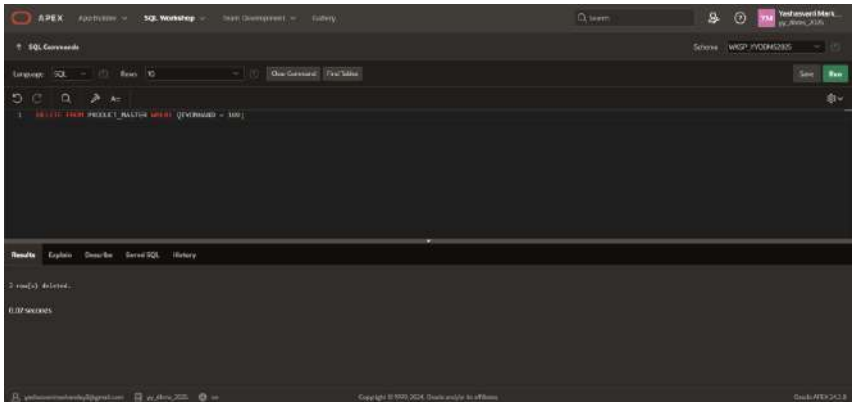


4. Exercise on deleting records in a table

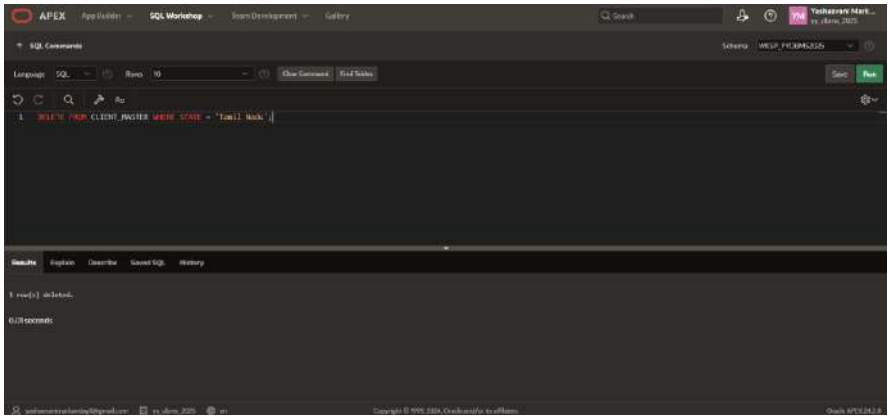
a. Delete all salesman from the Salesman_Master whose salaries are equal to Rs.3500.



b. Delete all products from Product_Master where the quantity on hand is equal to 100.

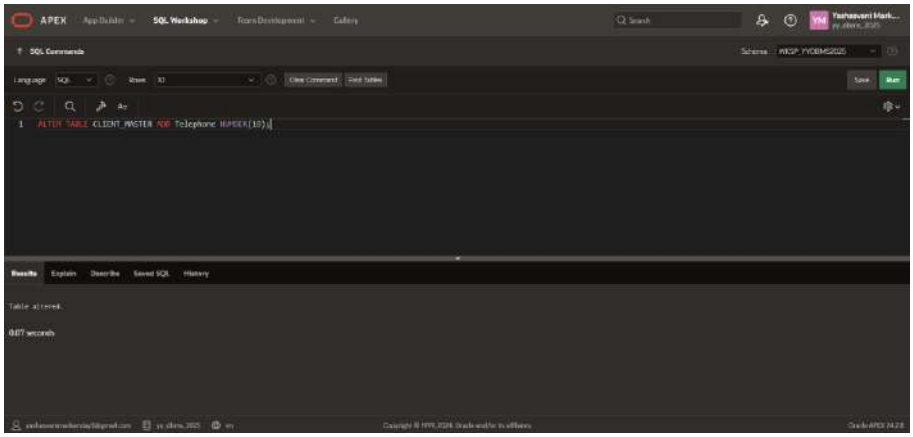


c. Delete from Client_Master where the column state holds the value ‘Tamil Nadu’.

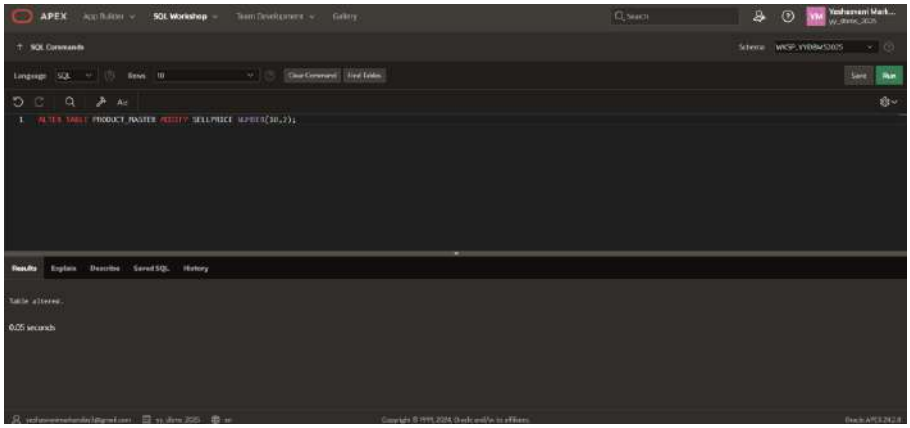


5. Exercise on altering the table structure

a. Add a column called ‘Telephone’ of data type integer to the Client_Master table.

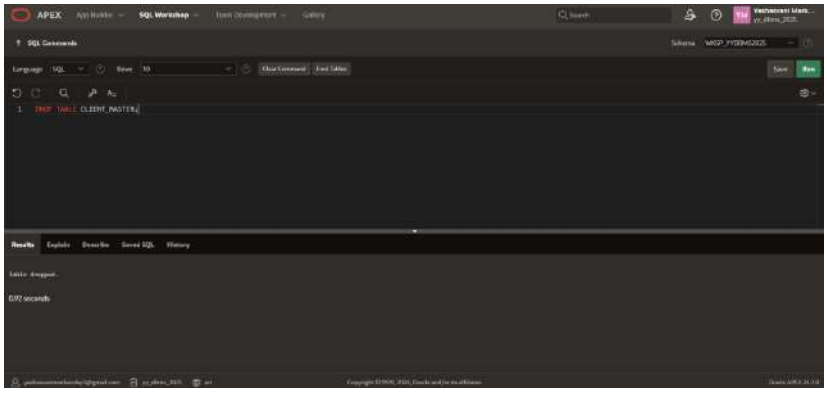


b. Change the size off SellPrice column in Product _Master to 10, 2.



6. Exercise on deleting the table structure along with the data

a. Destroy the table Client_Master along with its data.



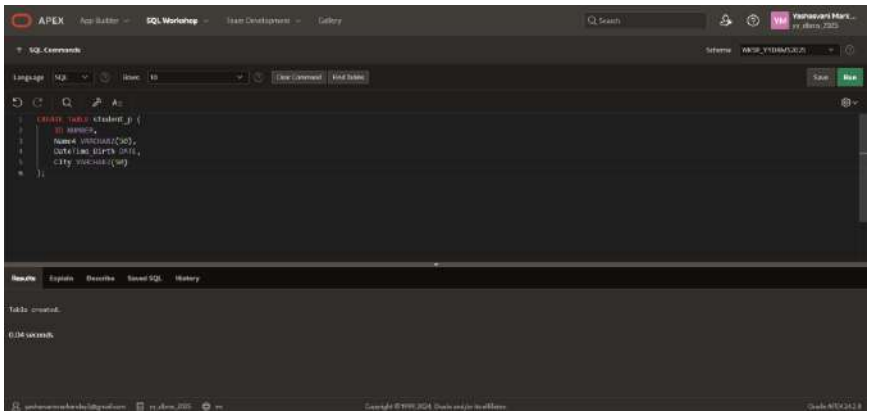
UNIVERSITY OF PETROLEUM AND ENERGY STUDIES

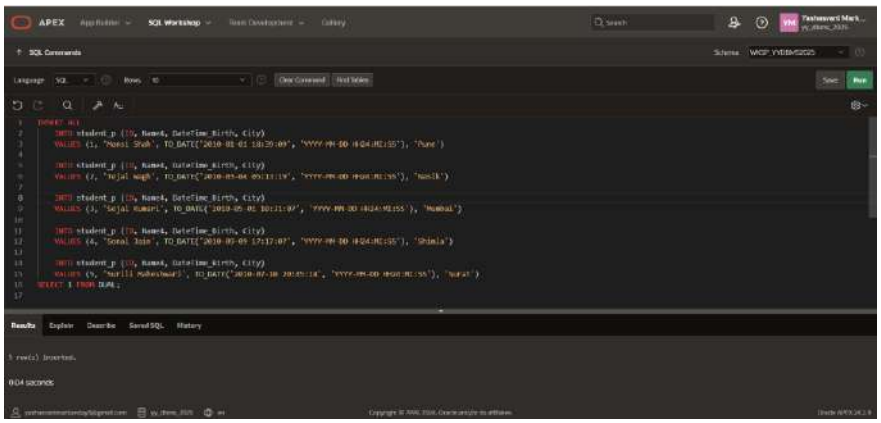
Program Name: B. Tech CS **Semester: III**

Name: Yashasvani Markanday **SAP ID: 590011386**

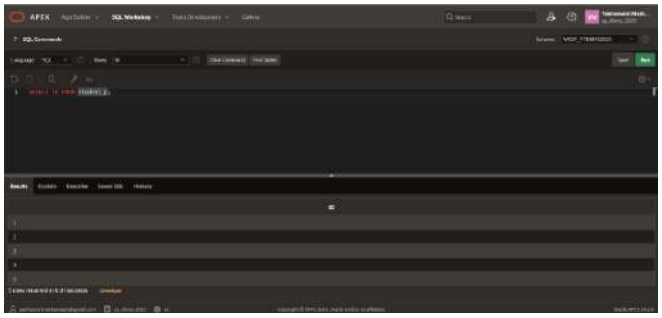
Batch: B2 **Subject: DAA**

Lab-6

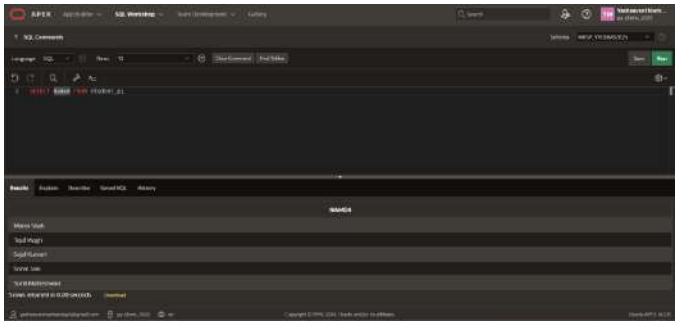




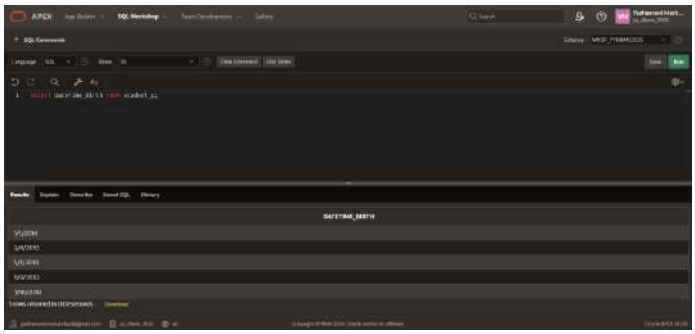
1. SELECT ID FROM student;



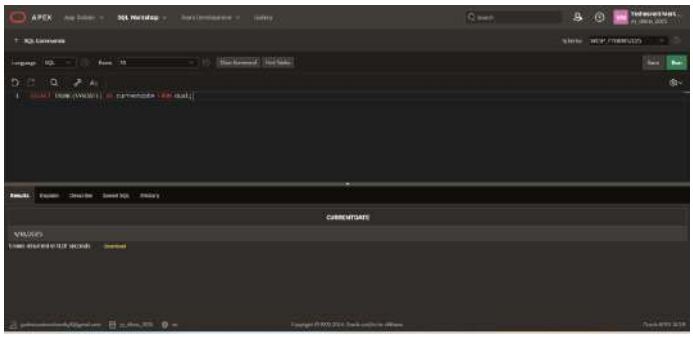
2 SELECT Name FROM student;



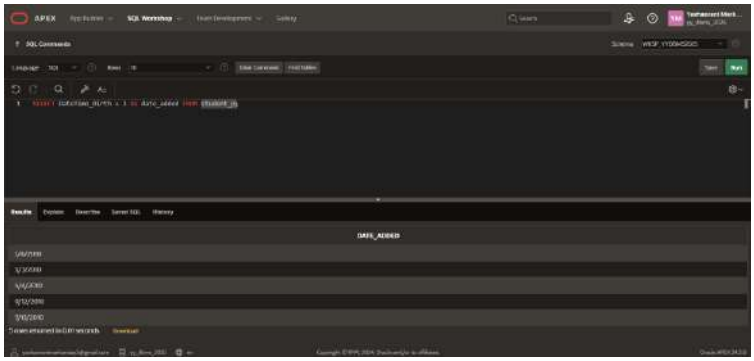
3 SELECT DateTime_Birth FROM student;



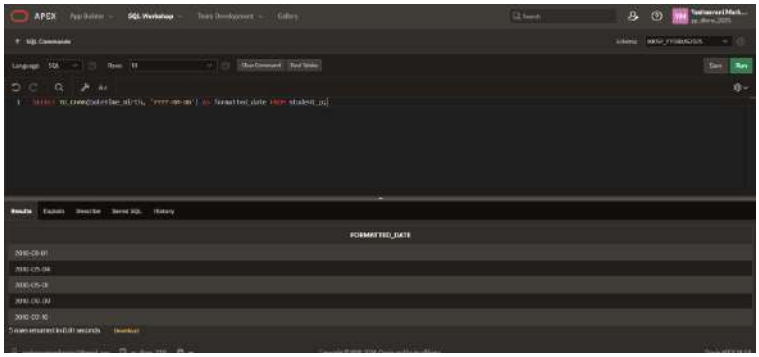
4 SELECT CURDATE() AS currentdate;



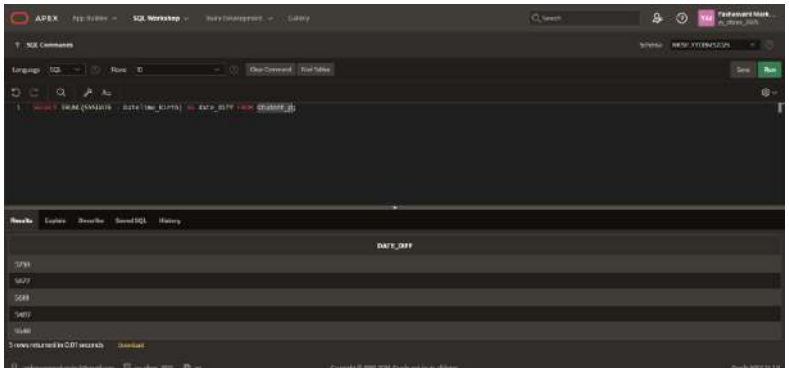
5 SELECT DATE_ADD(DateTime_Birth, INTERVAL 3 DAY) AS date_added FROM student;



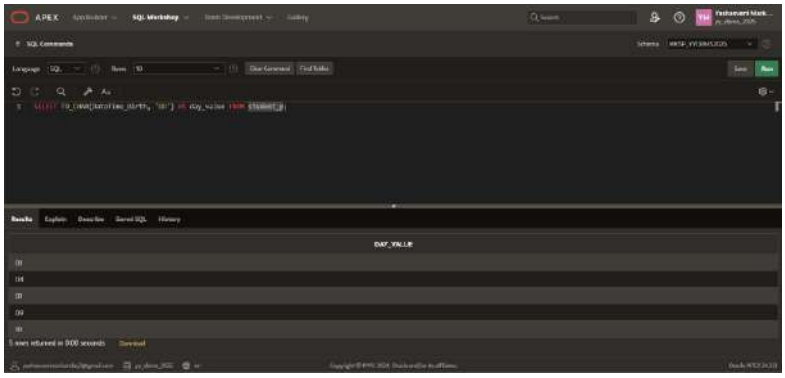
6 SELECT DATE_FORMAT(DateTime_Birth, '%Y-%m-%d') AS formatted_date FROM student;



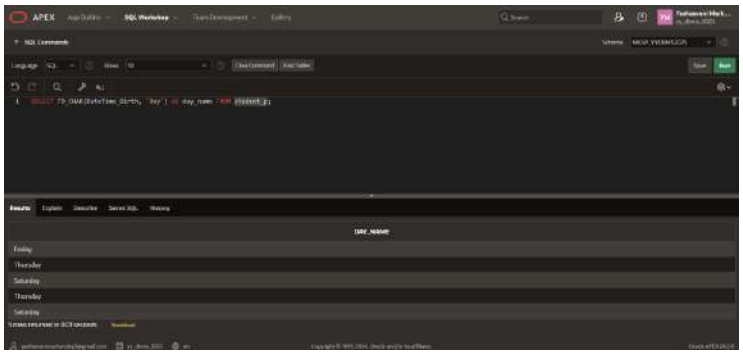
7 SELECT DATEDIFF(CURDATE(), DATE(DateTime_Birth)) AS date_diff FROM student;



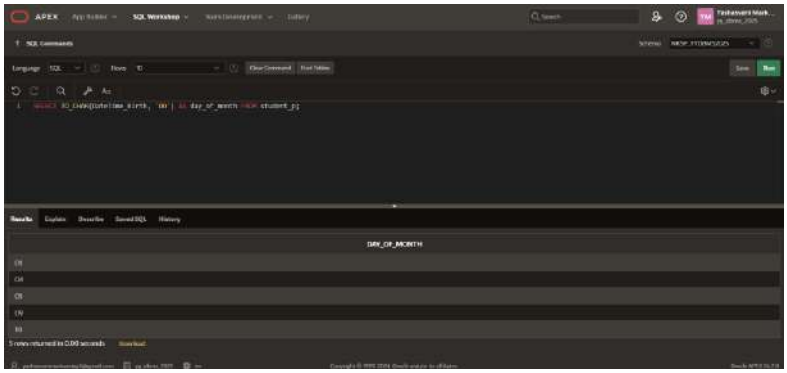
8 SELECT DAY(DateTime_Birth) AS day_value FROM student;



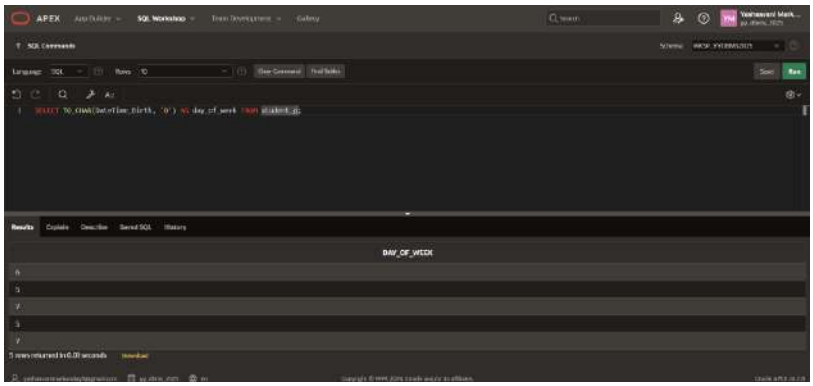
9 SELECT DAYNAME(DateTime_Birth) AS day_name FROM student;



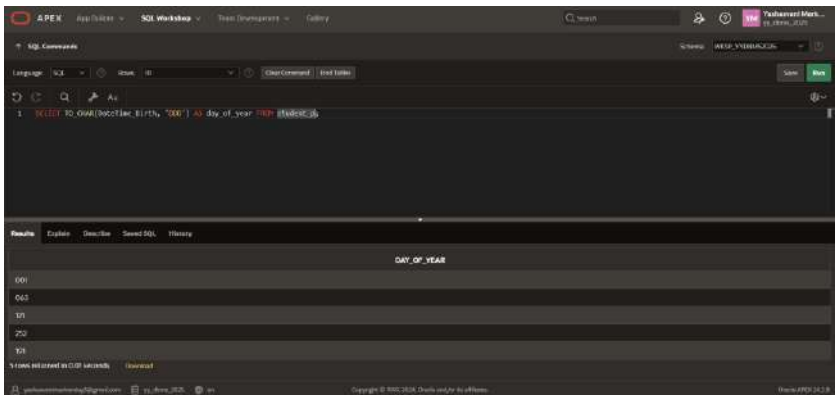
10 SELECT DAYOFMONTH(DateTime_Birth) AS day_of_month FROM student;



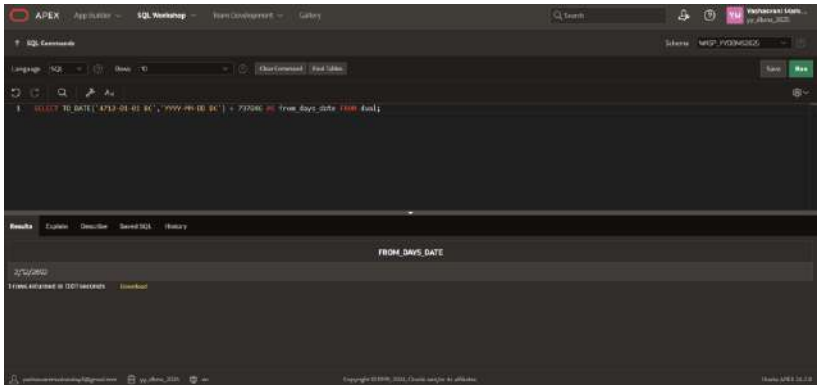
11 SELECT DAYOFWEEK(DateTime_Birth) AS day_of_week FROM student;



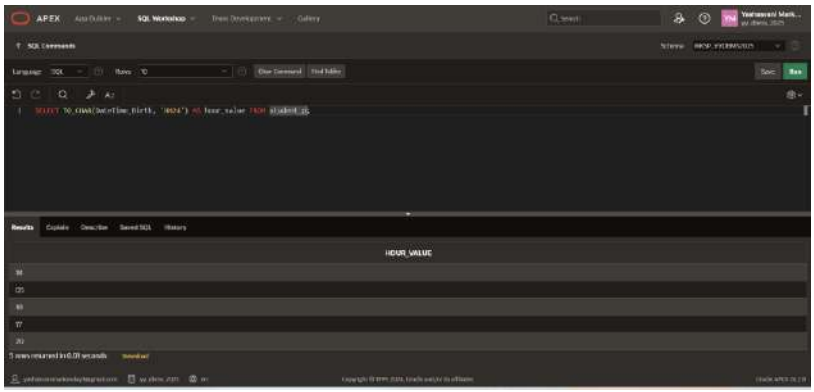
12 SELECT DAYOFYEAR(DateTime_Birth) AS day_of_year FROM student;



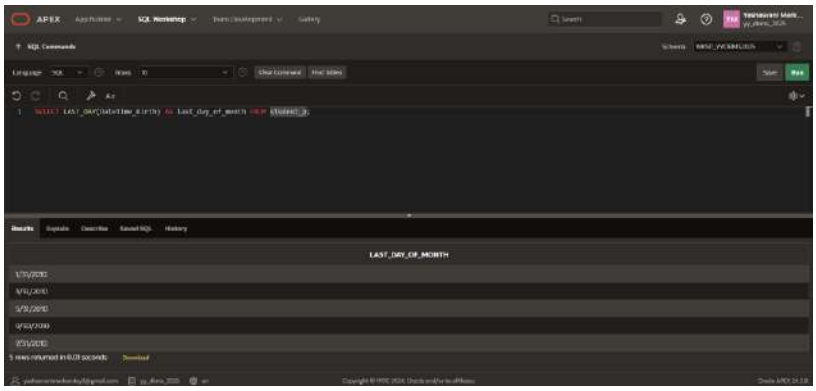
13 SELECT FROM_DAYS(737846) AS from_days_date FROM student;



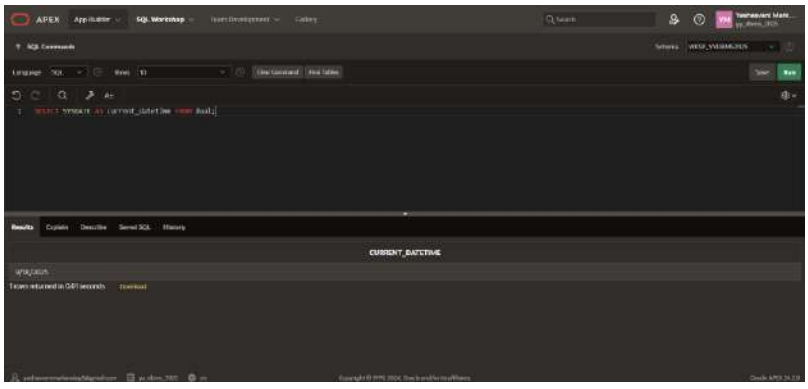
14 SELECT HOUR(DateTime_Birth) AS hour_value FROM student;



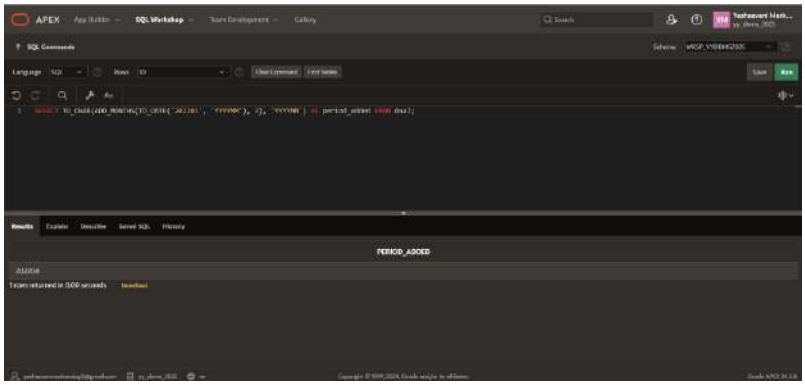
15 SELECT LAST_DAY(DateTime_Birth) AS last_day_of_month FROM student;



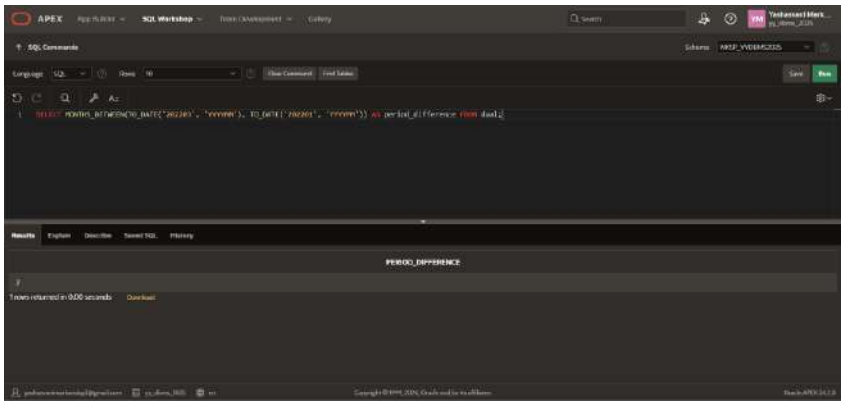
16 SELECT NOW() AS current_datetime FROM student;



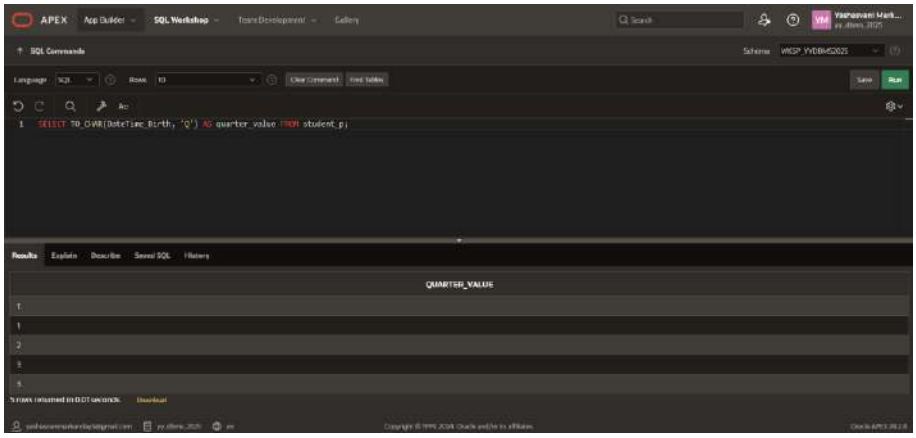
17 SELECT PERIOD_ADD(202201, 3) AS period_added FROM student;



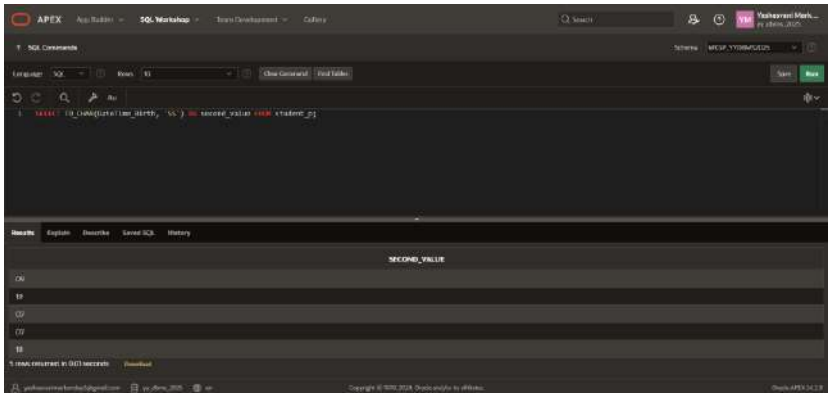
18 SELECT PERIOD_DIFF(202203, 202201) AS period_difference FROM student;



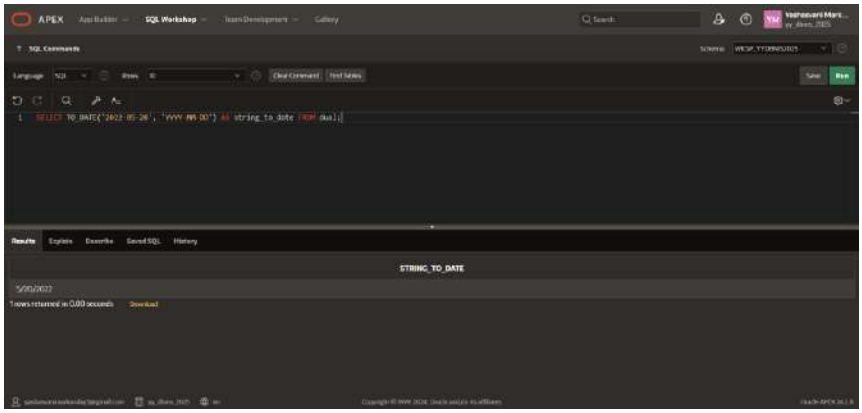
19 SELECT QUARTER(DateTime_Birth) AS quarter_value FROM student;



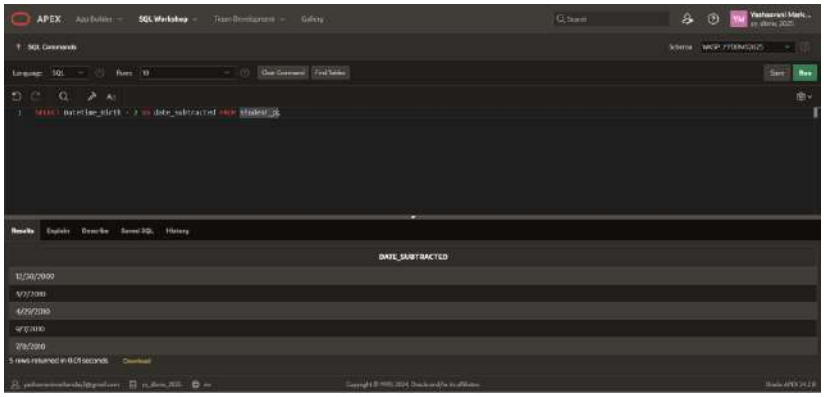
20 SELECT SECOND(DateTime_Birth) AS second_value FROM student;



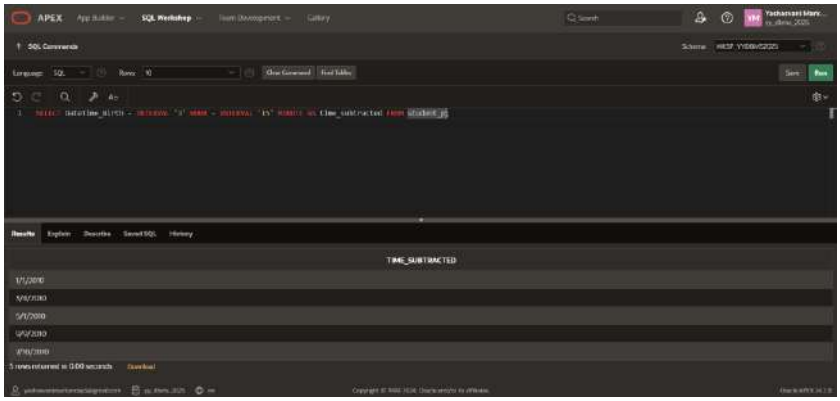
21 SELECT STR_TO_DATE('2022-05-20', '%Y-%m-%d') AS string_to_date FROM student;



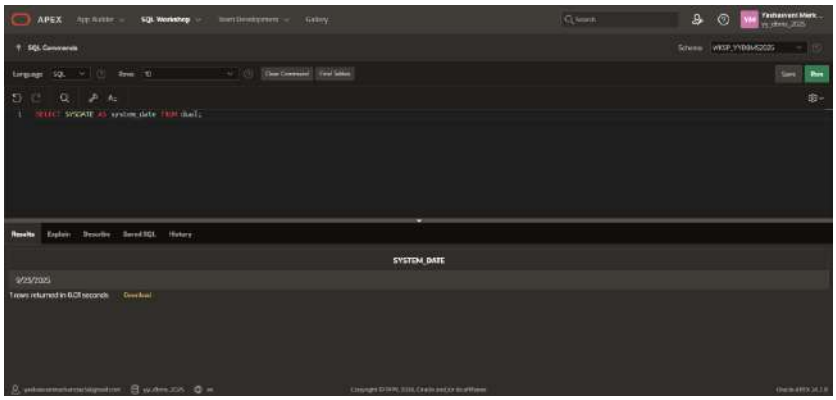
22 SELECT SUBDATE(DateTime_Birth, INTERVAL 2 DAY) AS date_subtracted FROM student;



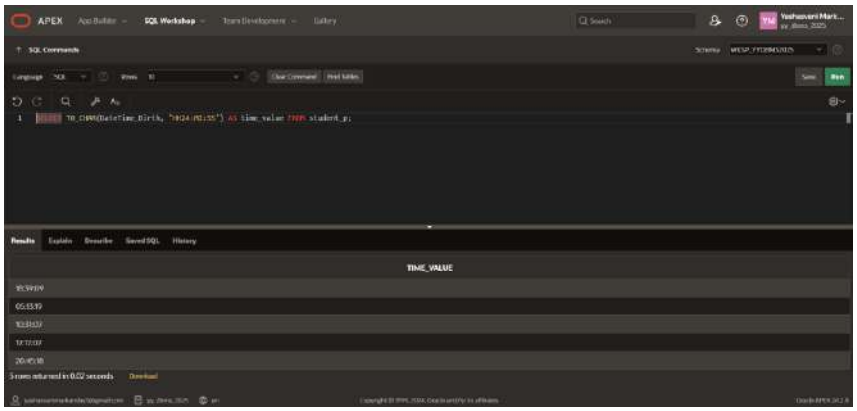
23 SELECT SUBTIME(DateTime_Birth, '03:15:00') AS time_subtracted FROM student;



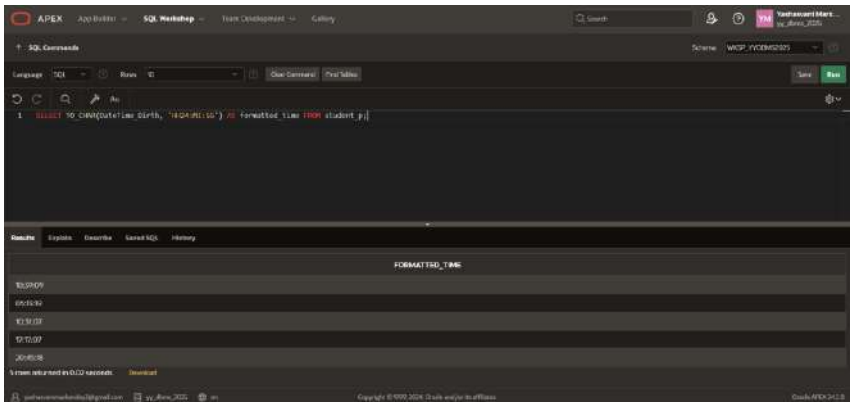
24 SELECT SYSDATE() AS system_date FROM student;



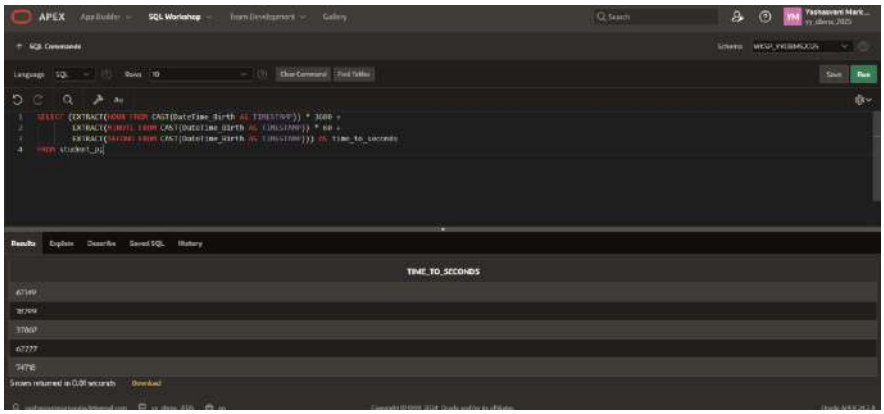
25 SELECT TIME(DateTime_Birth) AS time_value FROM student;



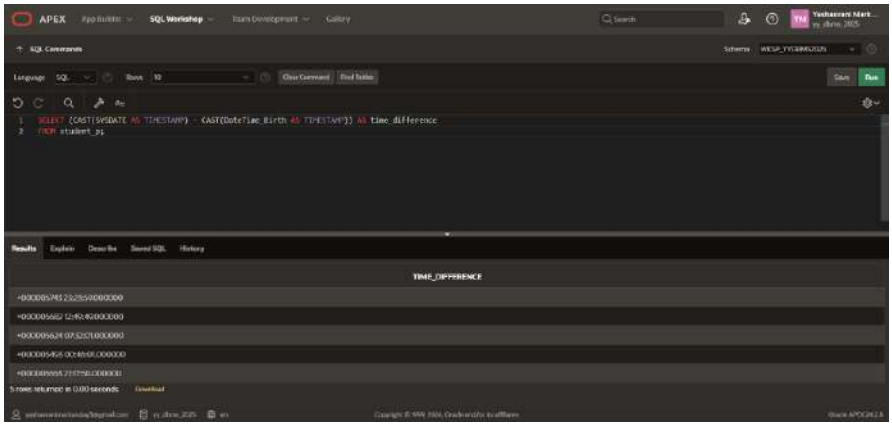
26 SELECT TIME_FORMAT(DateTime_Birth, '%H:%i:%s') AS formatted_time FROM student;



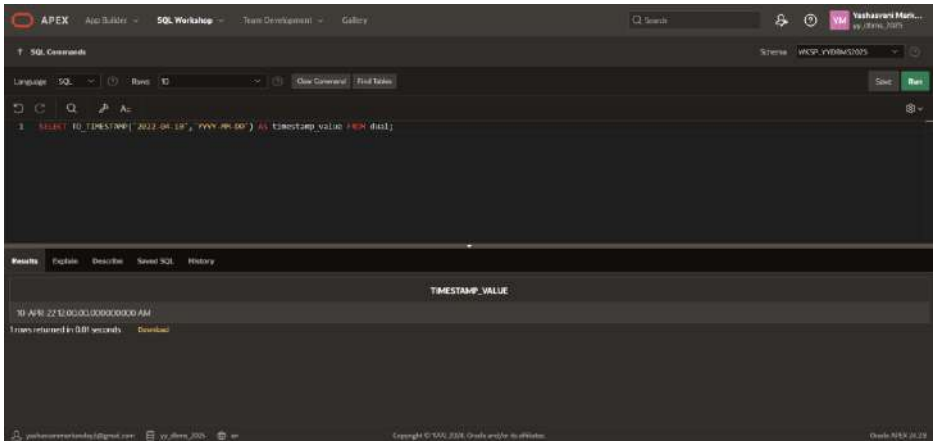
27 SELECT TIME_TO_SEC(DateTime_Birth) AS time_to_seconds FROM student;



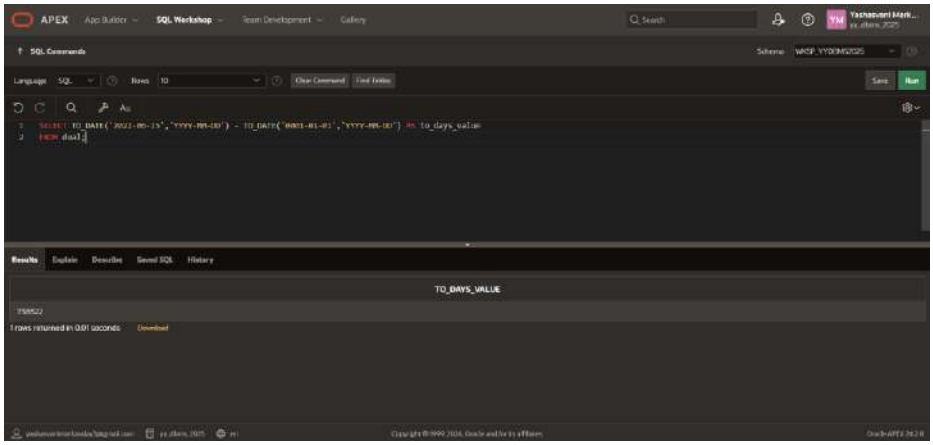
28 SELECT TIMEDIFF(NOW(), DateTime_Birth) AS time_difference FROM student;



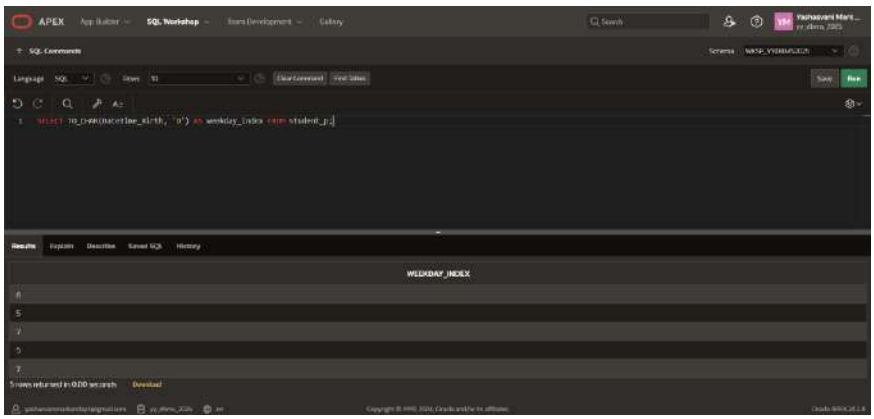
29 SELECT TIMESTAMP('2022-04-10') AS timestamp_value FROM student;



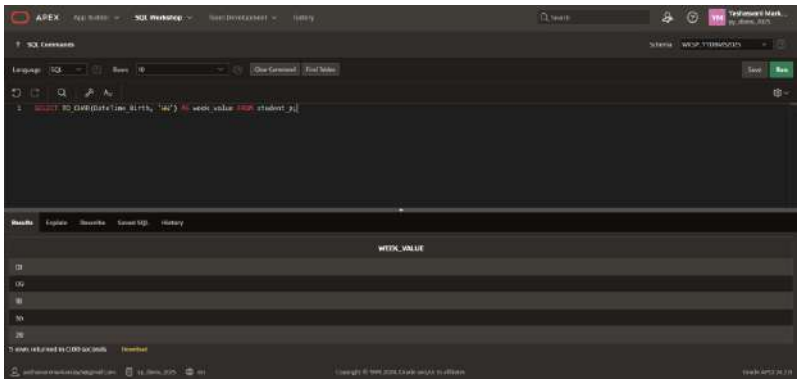
30 SELECT TO_DAYS('2022-06-15') AS to_days_value FROM student;



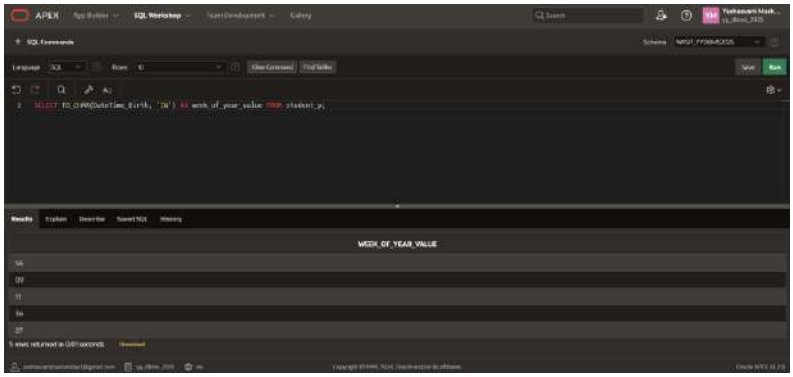
31 SELECT WEEKDAY(DateTime_Birth) AS weekday_index FROM student;



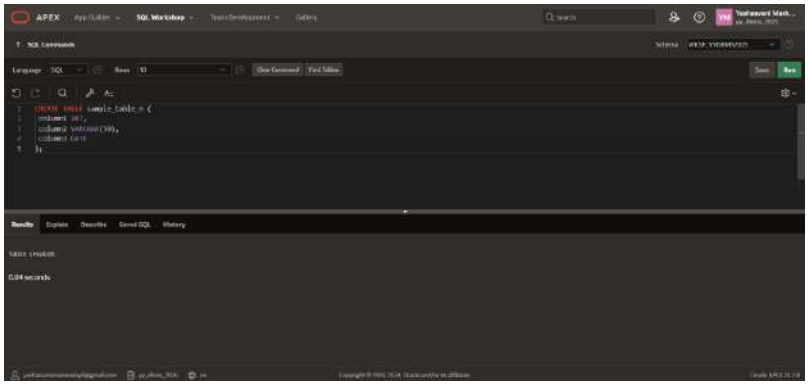
32 SELECT WEEK(DateTime_Birth) AS week_value FROM student;

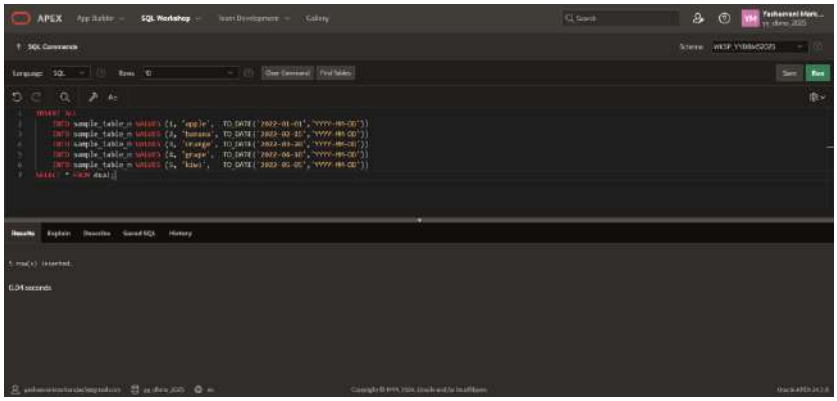


33 SELECT WEEKOFYEAR(DateTime_Birth) AS week_of_year_value FROM student;

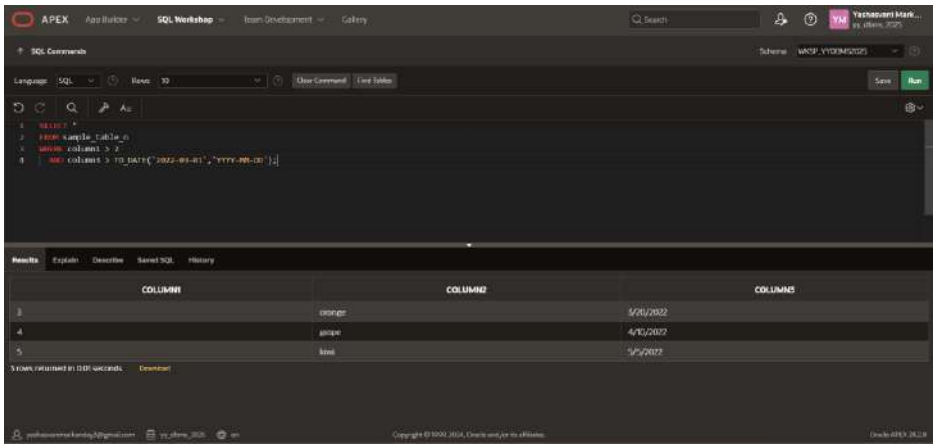


Conditional Functions

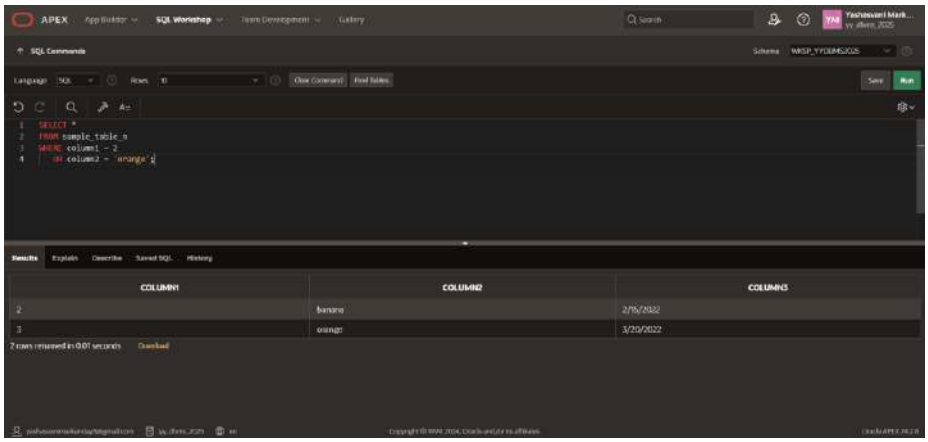




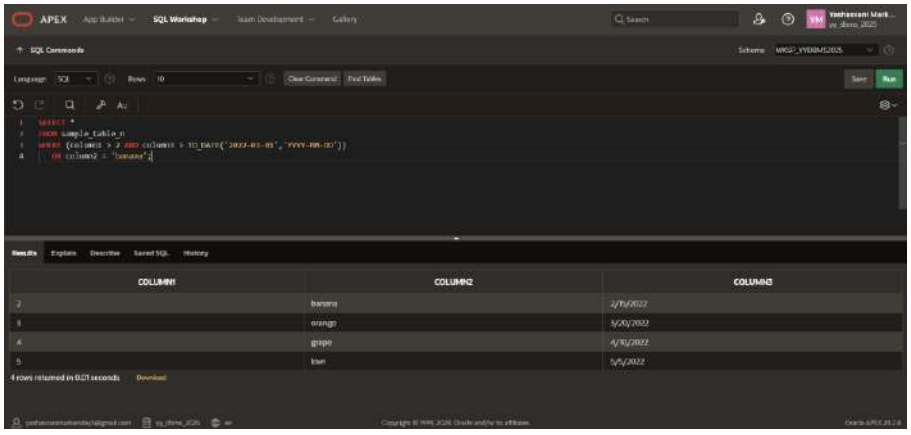
34 SELECT * FROM sample_table WHERE column1 > 2 AND column3 > '2022-03-01';



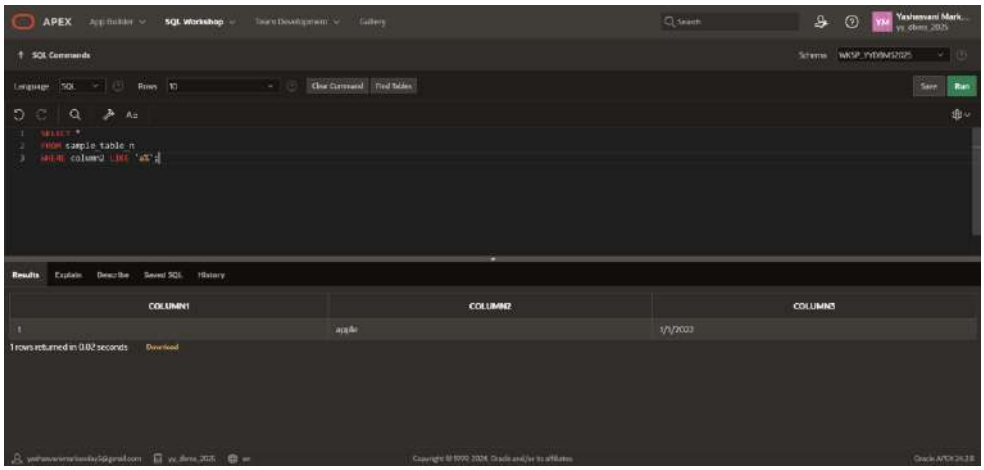
35 SELECT * FROM sample_table WHERE column1 = 2 OR column2 = 'orange';



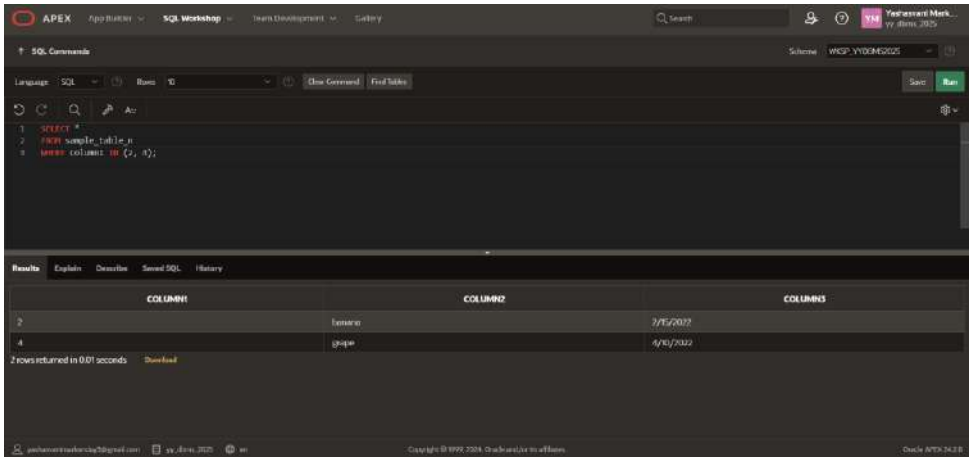
36 SELECT * FROM sample_table WHERE (column1 > 2 AND column3 > '2022-03-01') OR column2 = 'banana';



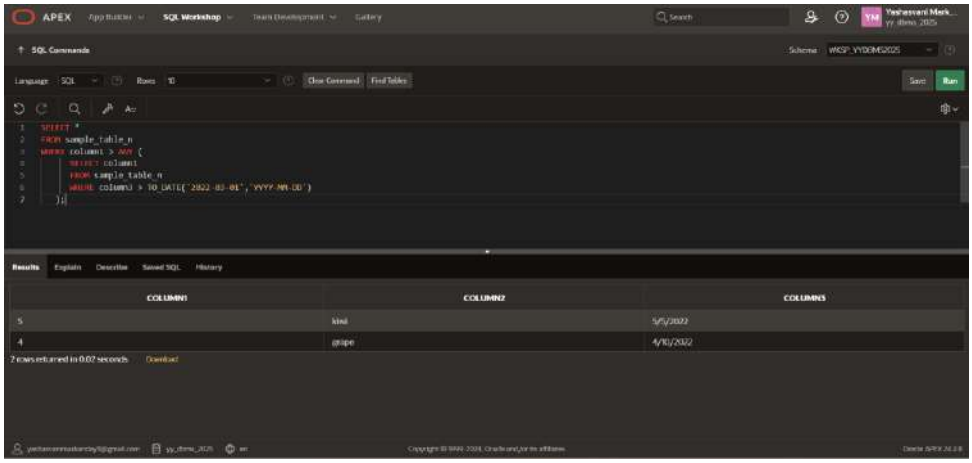
37 SELECT * FROM sample_table WHERE column2 LIKE 'a%';



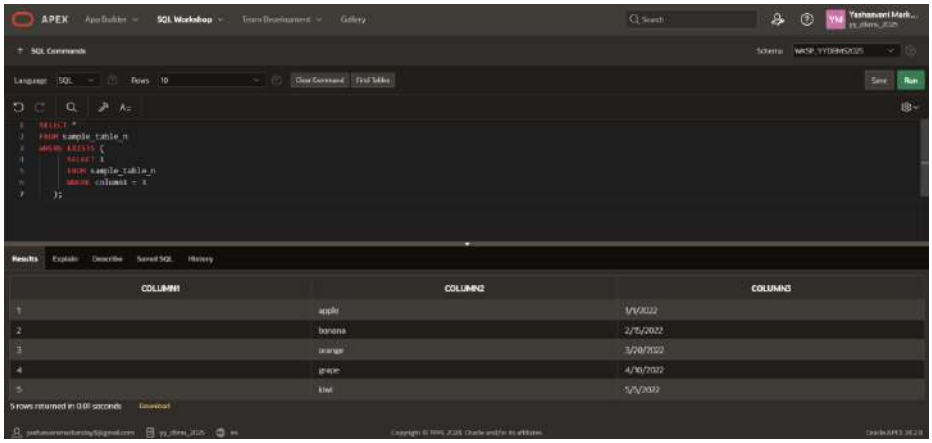
38 SELECT * FROM sample_table WHERE column1 IN (2, 4);



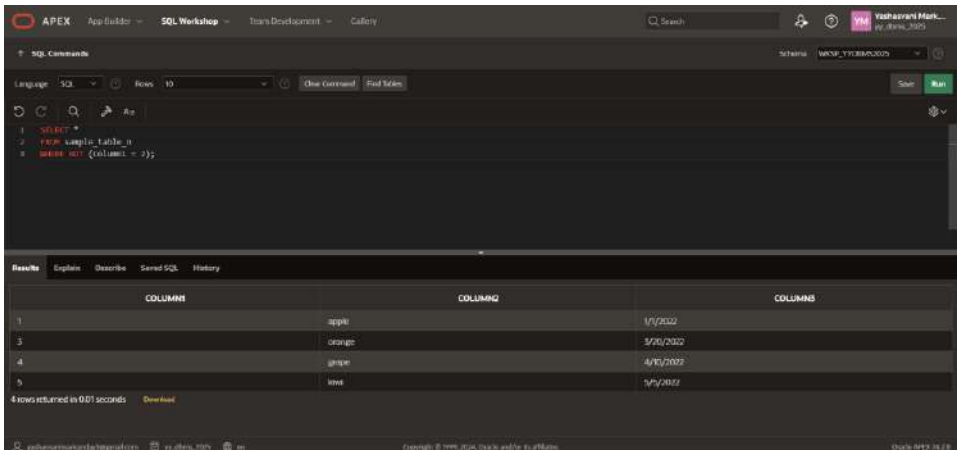
39 SELECT * FROM sample_table WHERE column1 > ANY (SELECT column1 FROM sample_table WHERE column3 > '2022-03-01');



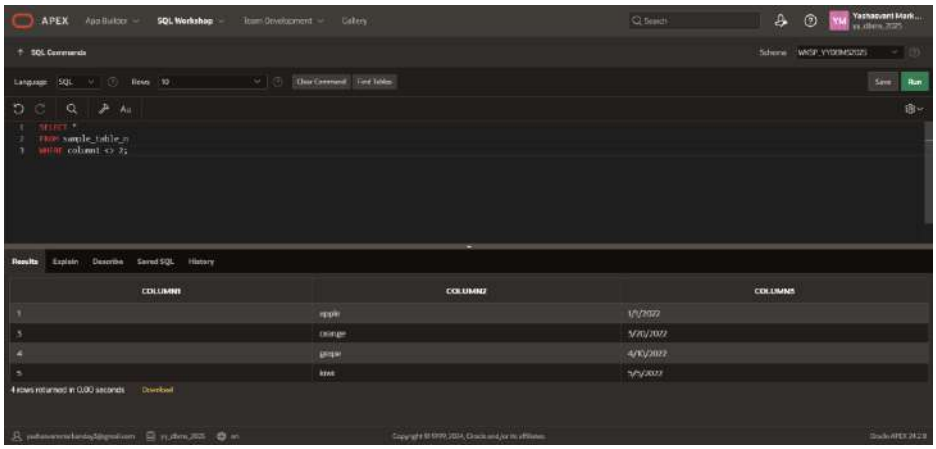
40 SELECT * FROM sample_table WHERE EXISTS (SELECT * FROM sample_table WHERE column1 = 3);



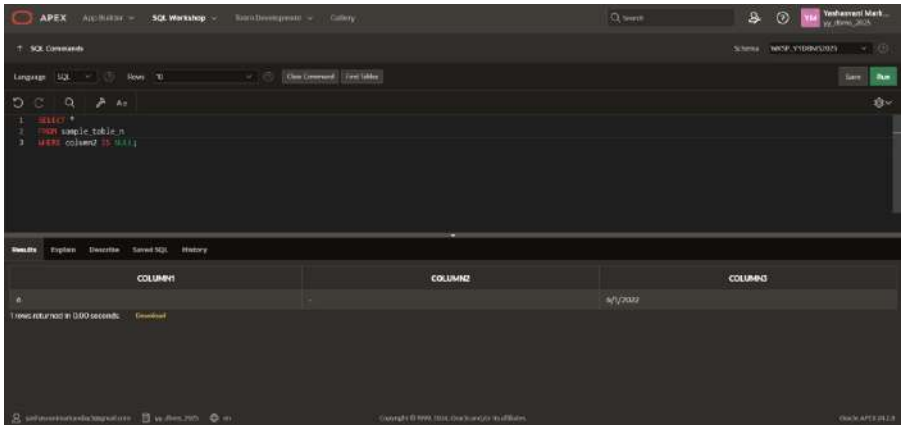
41 SELECT * FROM sample_table WHERE NOT column1 = 2;



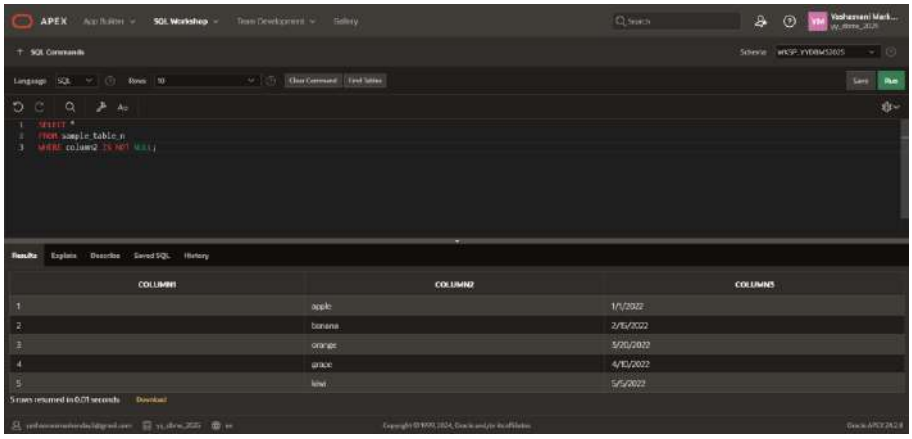
42 SELECT * FROM sample_table WHERE column1 <> 2;



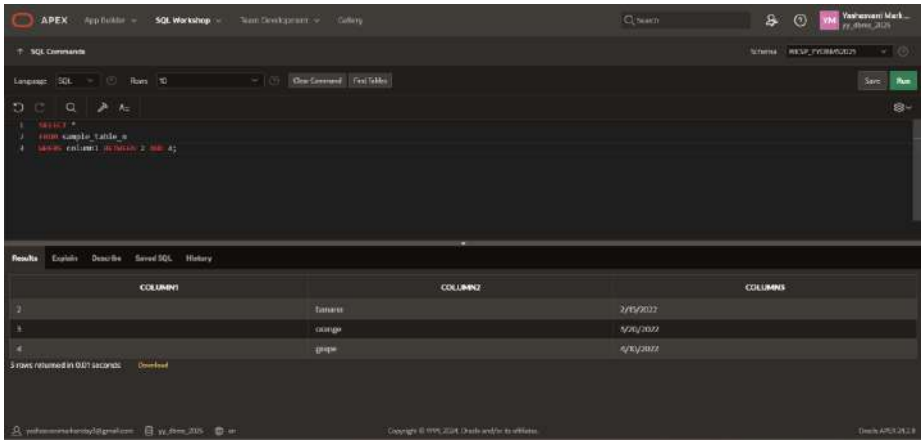
43 SELECT * FROM sample_table WHERE column2 IS NULL;



44 SELECT * FROM sample_table WHERE column2 IS NOT NULL;

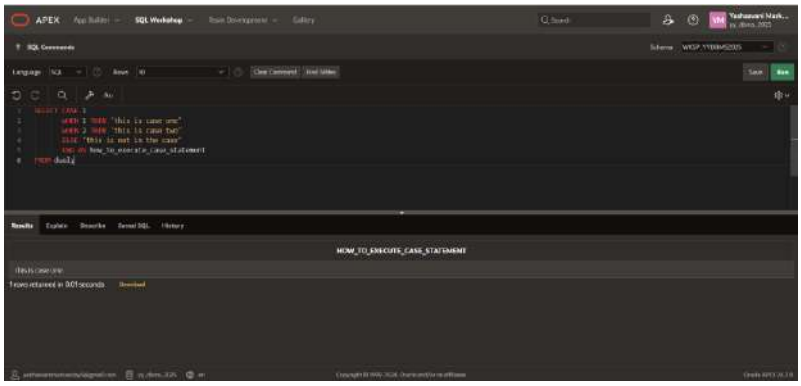


45 SELECT * FROM sample_table WHERE column1 BETWEEN 2 AND 4;

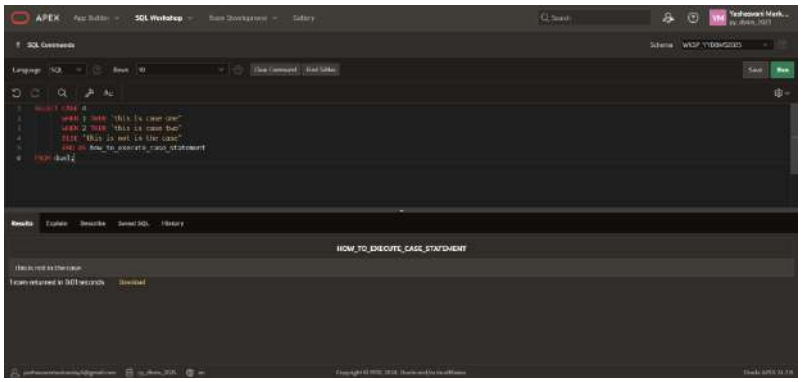


1. Simple CASE Statement: - When the condition is met (CASE 1), it returns the corresponding value. - When the condition is not met (CASE 4), it returns the value in the ELSE part.

46. SELECT CASE 1 WHEN 1 THEN 'this is case one' WHEN 2 THEN 'this is case two' ELSE 'this is not in the case' END as 'how to execute case statement';

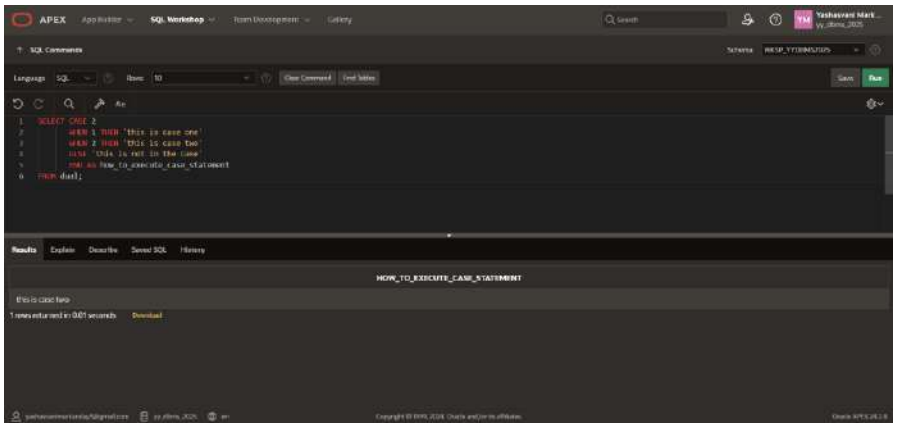


47. SELECT CASE 4 WHEN 1 THEN 'this is case one' WHEN 2 THEN 'this is case two' ELSE 'this is not in the case' END as 'how to execute case statement';



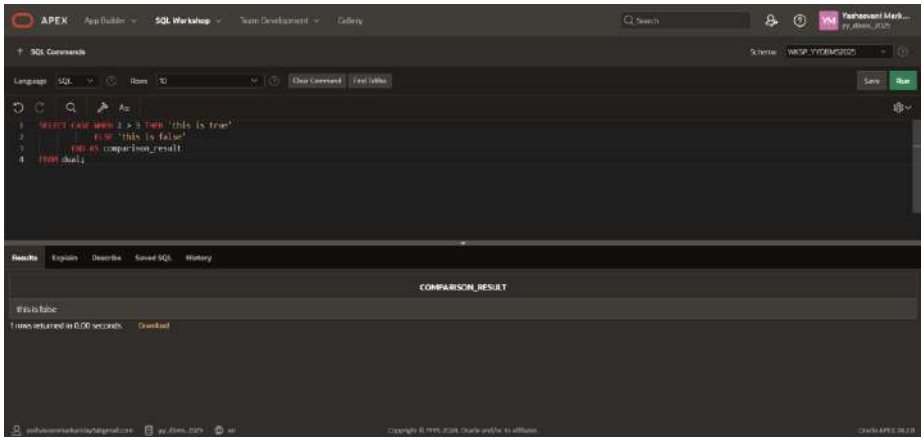
2. CASE Statement with Matching Condition: - When the condition is met (CASE 2), it returns the corresponding value.

48. SELECT CASE 2 WHEN 1 THEN 'this is case one' WHEN 2 THEN 'this is case two' ELSE 'this is not in the case' END as 'how to execute case statement';

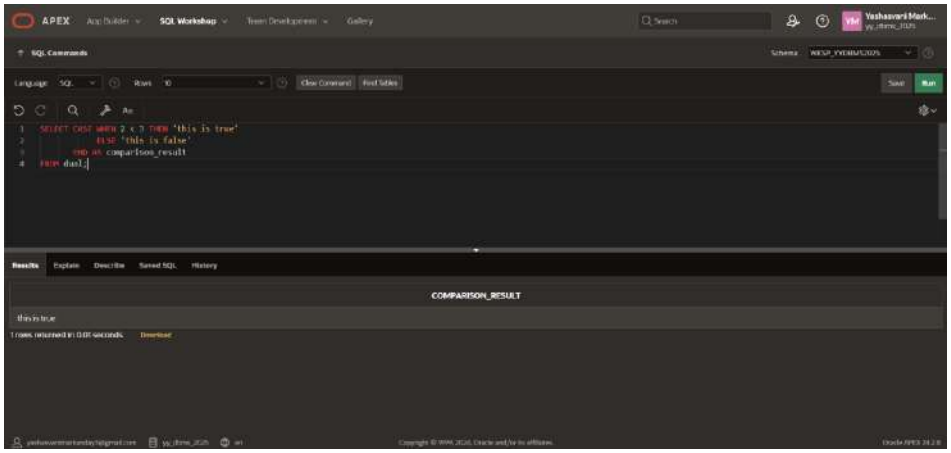


3. CASE Statement with Comparison Operators: - Using greater than and less than operators to evaluate conditions.

49. SELECT CASE WHEN 2>3 THEN 'this is true' ELSE 'this is false' END;

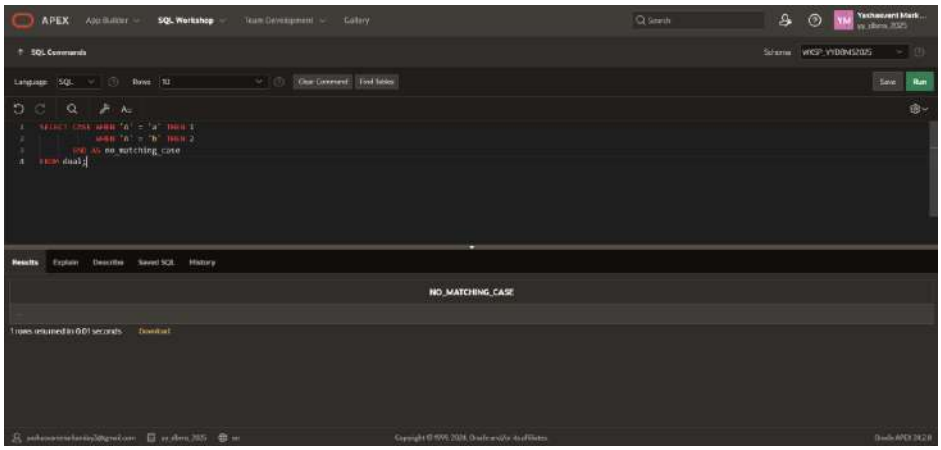


50. SELECT CASE WHEN 2<3 THEN ‘this is true’ ELSE ‘this is false’ END.

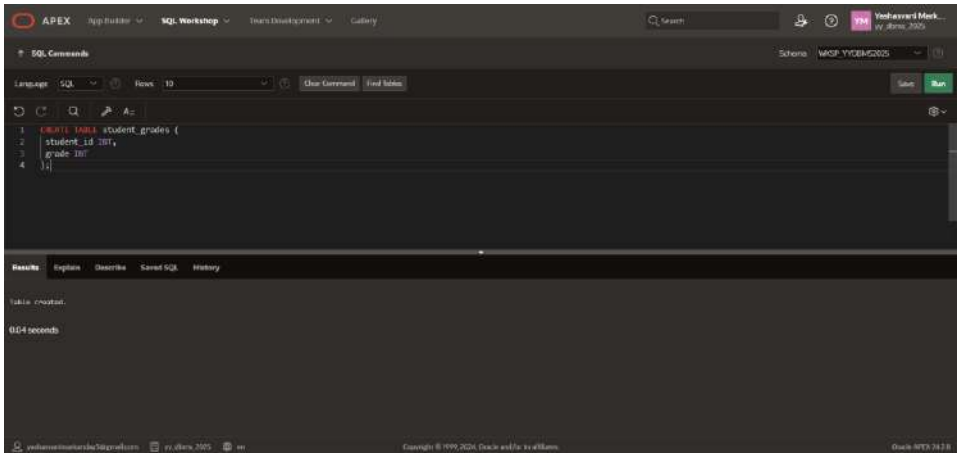


4. CASE Statement with No Matching Conditions: - If none of the conditions are satisfied, it returns NULL.

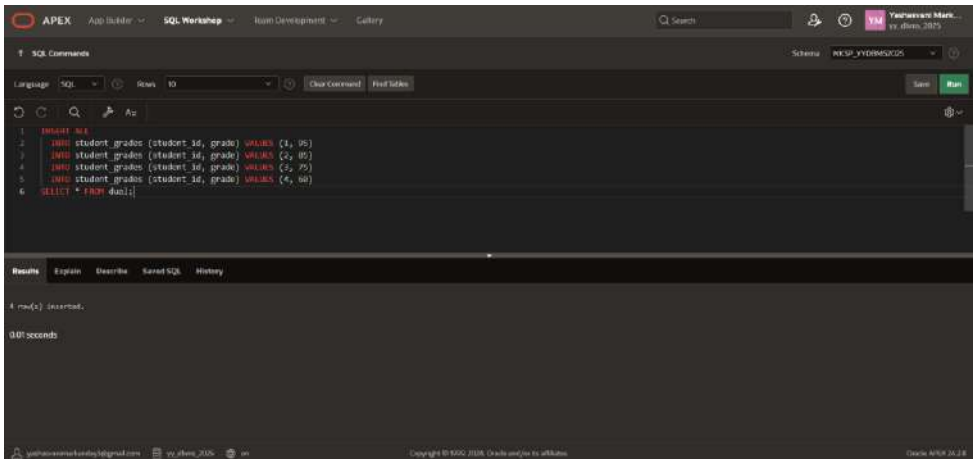
51. SELECT CASE BINARY 'A' WHEN 'a' THEN 1 WHEN 'b' THEN 2 END;



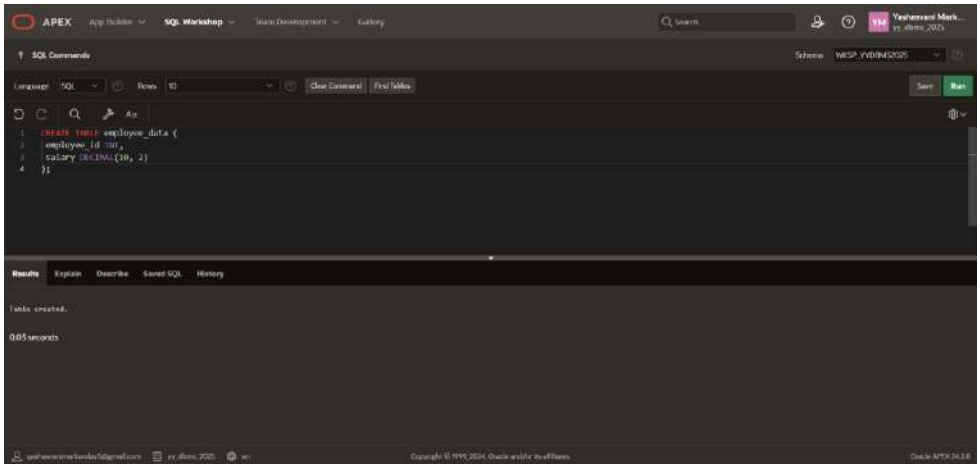
52. CREATE TABLE student_grades (student_id INT, grade INT);



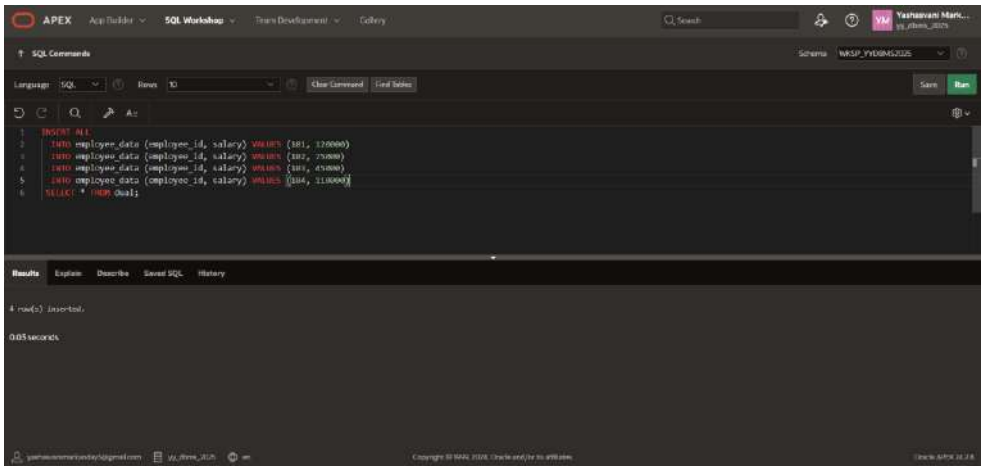
53. INSERT INTO student_grades (student_id, grade) VALUES (1, 95), (2, 85), (3, 75), (4, 60);



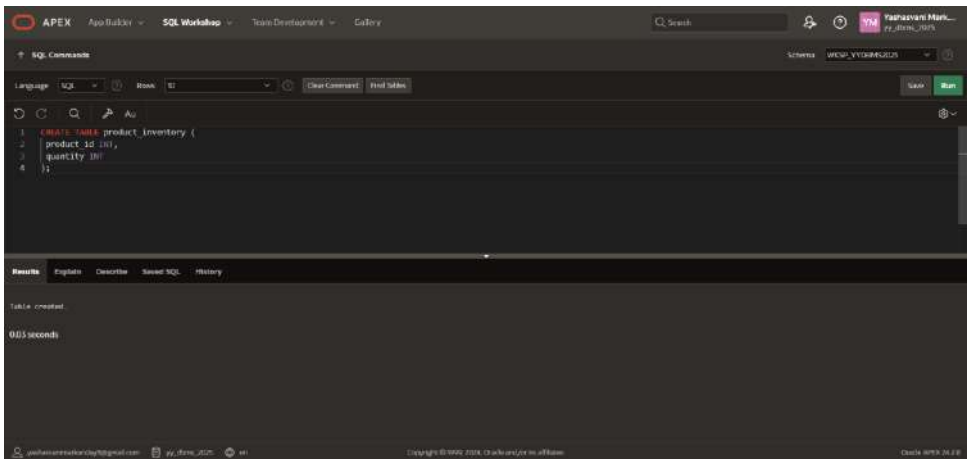
54. CREATE TABLE employee_data (employee_id INT, salary DECIMAL(10, 2));



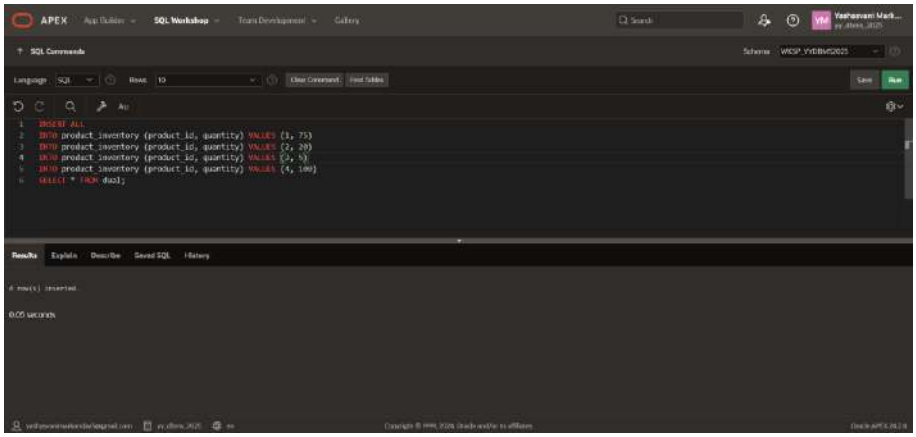
55. INSERT INTO employee_data (employee_id, salary) VALUES (101, 120000), (102, 75000), (103, 45000), (104, 110000);



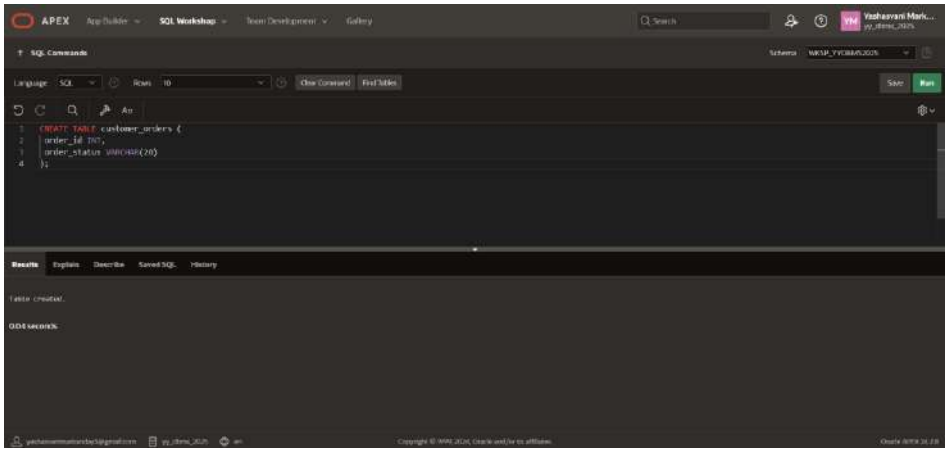
56. CREATE TABLE product_inventory (product_id INT, quantity INT);



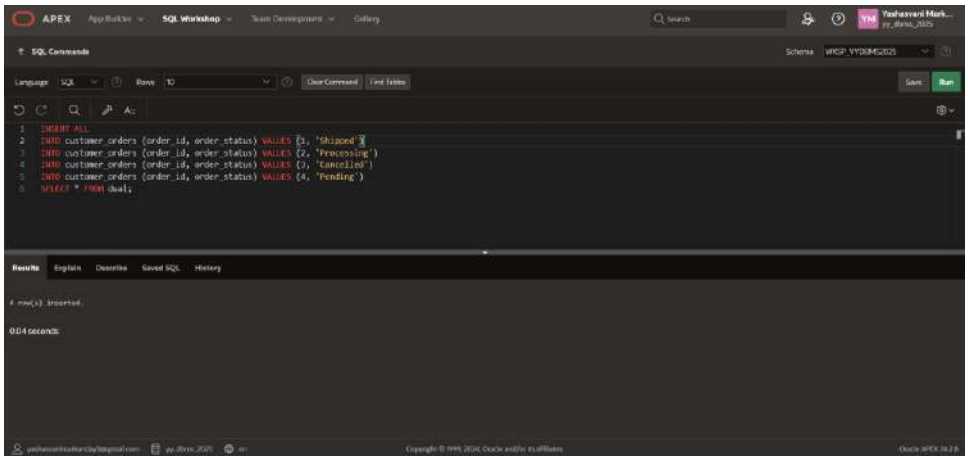
57. INSERT INTO product_inventory (product_id, quantity) VALUES (1, 75), (2, 20), (3, 5), (4, 100);



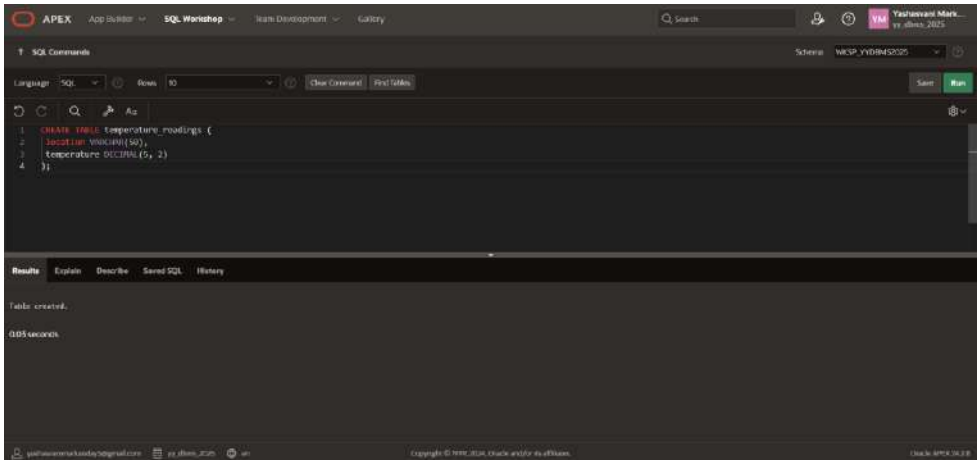
58. CREATE TABLE customer_orders (order_id INT, order_status VARCHAR(20));



59. INSERT INTO customer_orders (order_id, order_status) VALUES (1, 'Shipped'), (2, 'Processing'), (3, 'Cancelled'), (4, 'Pending');

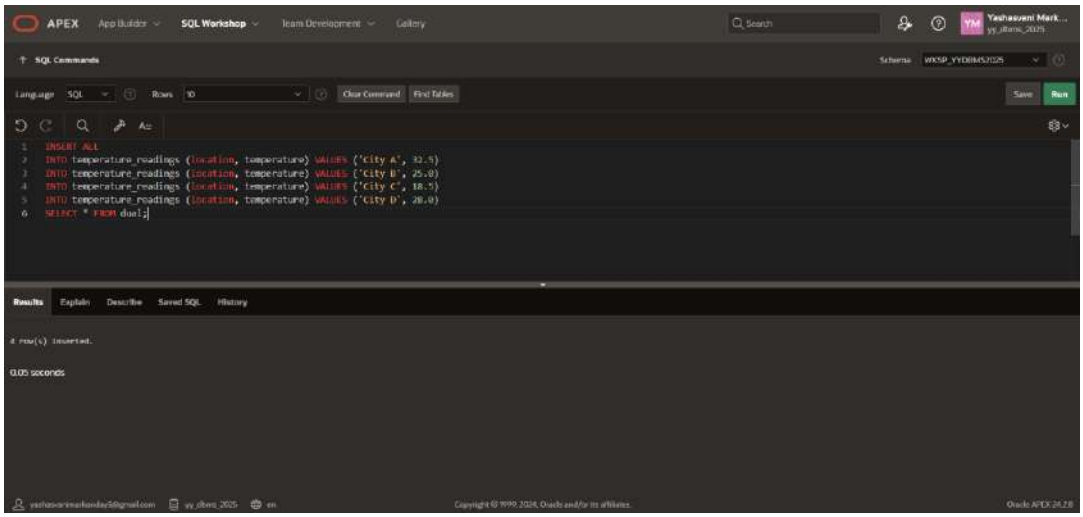


60. CREATE TABLE temperature_readings (location VARCHAR(50), temperature DECIMAL(5, 2));



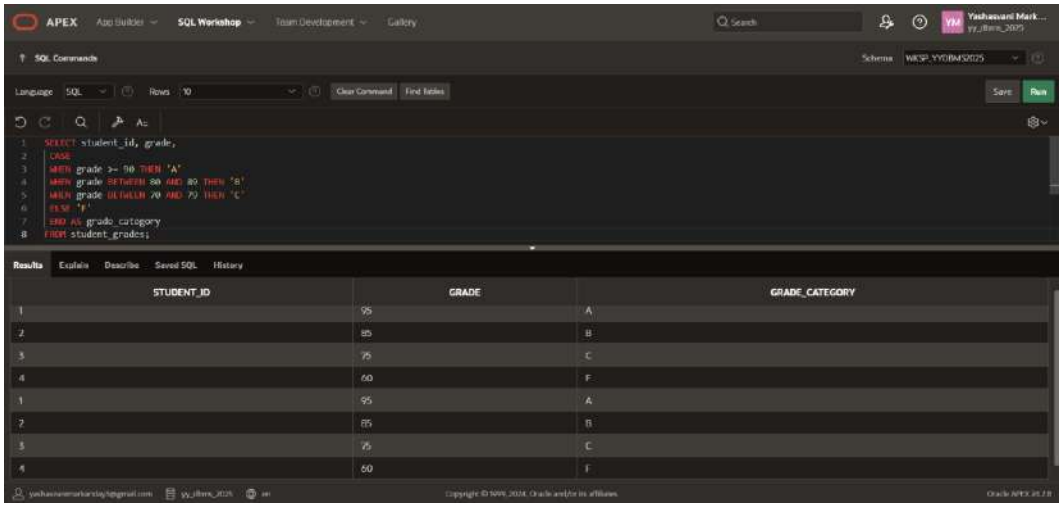
61. INSERT INTO temperature_readings (location, temperature) VALUES ('City A', 32.5), ('City B', 25.0), ('City C', 18.5), ('City D', 28.0);

-- Display grades based on conditions



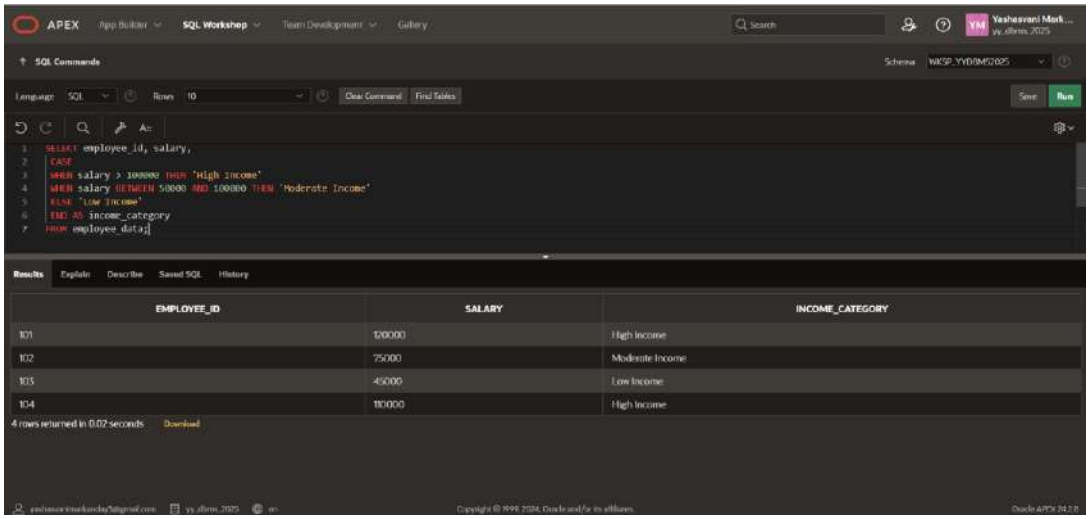
62. SELECT student_id, grade, CASE WHEN grade >= 90 THEN 'A' WHEN grade BETWEEN 80 AND 89 THEN 'B' WHEN grade BETWEEN 70 AND 79 THEN 'C' ELSE 'F' END AS grade_category FROM student_grades;

-- Categorize employees based on salary



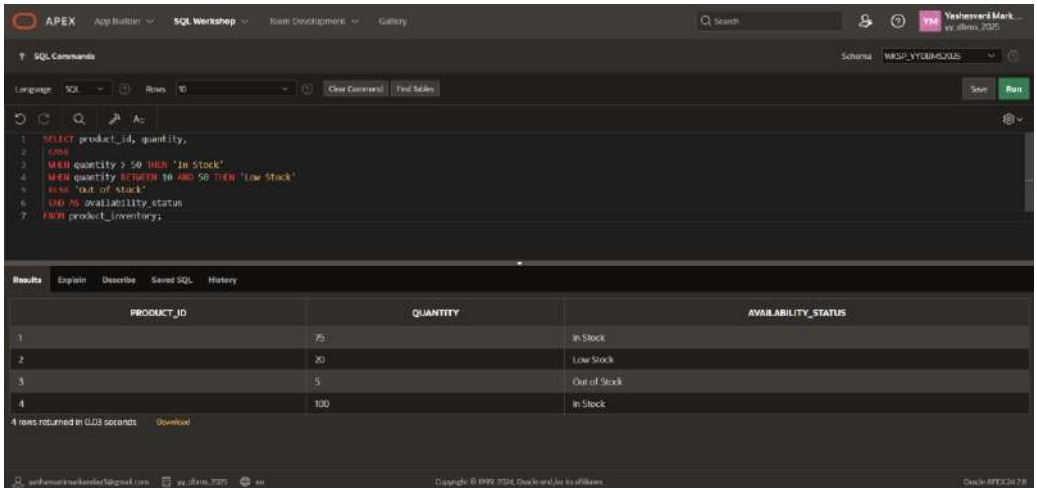
63. SELECT employee_id, salary, CASE WHEN salary > 100000 THEN 'High Income' WHEN salary BETWEEN 50000 AND 100000 THEN 'Moderate Income' ELSE 'Low Income' END AS income_category FROM employee_data;

-- Determine product availability based on quantity



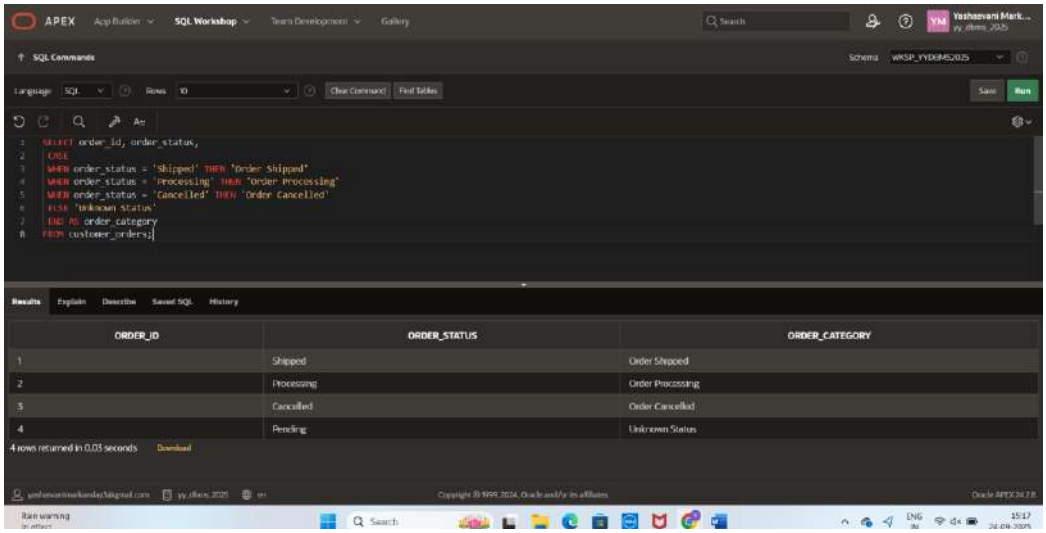
64. SELECT product_id, quantity, CASE WHEN quantity > 50 THEN 'In Stock' WHEN quantity BETWEEN 10 AND 50 THEN 'Low Stock' ELSE 'Out of Stock' END AS availability_status FROM product_inventory;

-- Categorize orders based on status

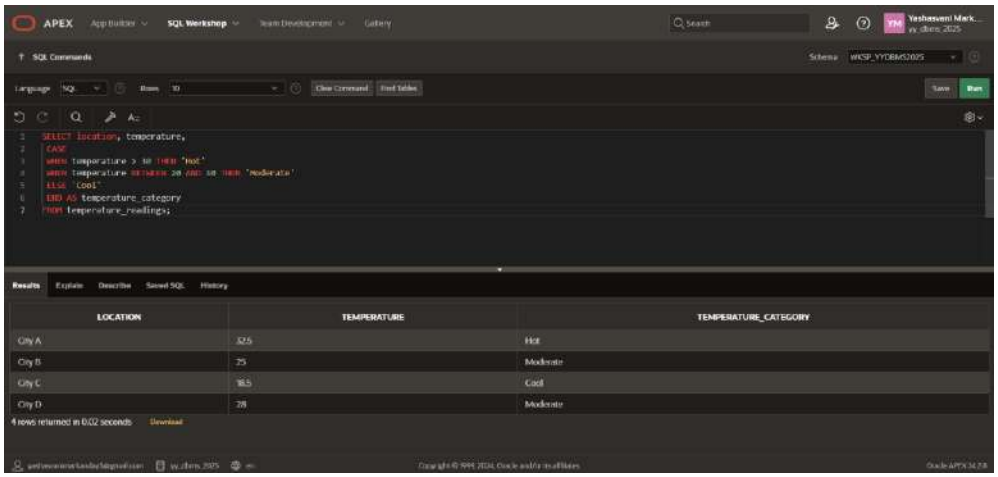


65. SELECT order_id, order_status, CASE WHEN order_status = 'Shipped' THEN 'Order Shipped' WHEN order_status = 'Processing' THEN 'Order Processing' WHEN order_status = 'Cancelled' THEN 'Order Cancelled' ELSE 'Unknown Status' END AS order_category FROM customer_orders;

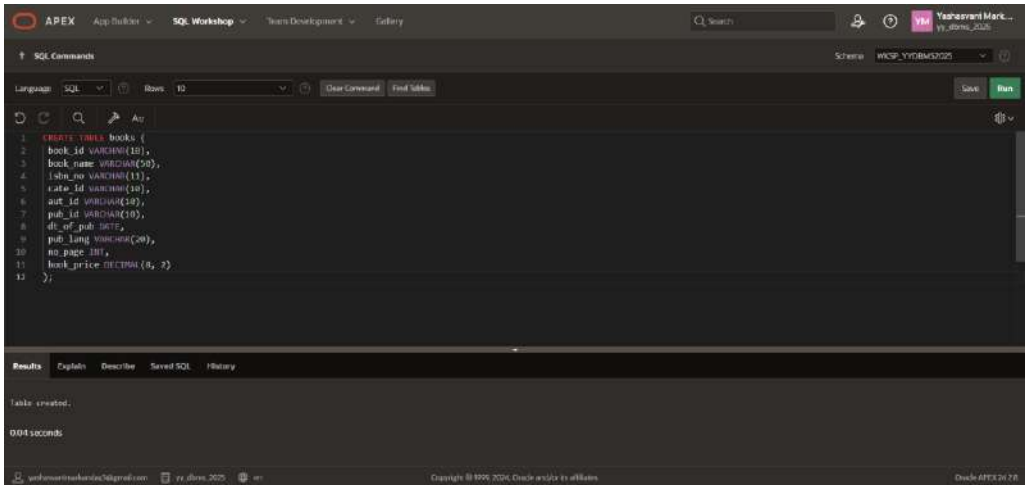
-- Categorize temperature readings based on conditions



66. SELECT location, temperature, CASE WHEN temperature > 30 THEN 'Hot' WHEN temperature BETWEEN 20 AND 30 THEN 'Moderate' ELSE 'Cool' END AS temperature_category FROM temperature_readings;

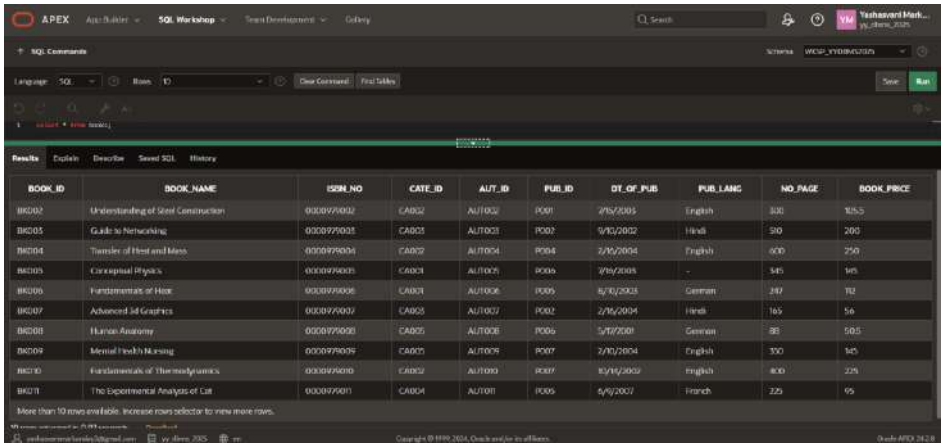


67. CREATE TABLE books (book_id VARCHAR(10), book_name VARCHAR(50), isbn_no VARCHAR(11), cate_id VARCHAR(10), aut_id VARCHAR(10), pub_id VARCHAR(10), dt_of_pub DATE, pub_lang VARCHAR(20), no_page INT, book_price DECIMAL(8, 2));



68. INSERT INTO books VALUES ('BK001', 'Introduction to Electrodynamics', '0000979001', 'CA001', 'AUT001', 'P003', '2001-05- 08', 'English', 201, 85.00), ('BK002', 'Understanding of Steel Construction', '0000979002', 'CA002', 'AUT002', 'P001', '2003- 07-15', 'English', 300, 105.50), ('BK003', 'Guide to Networking', '0000979003', 'CA003', 'AUT003', 'P002', '2002-09-10', 'Hindi', 510, 200.00), ('BK004', 'Transfer of Heat and Mass', '0000979004', 'CA002', 'AUT004', 'P004', '2004-02-16', 'English', 600, 250.00), ('BK005', 'Conceptual Physics', '0000979005', 'CA001', 'AUT005', 'P006', '2003-07-16', NULL, 345, 145.00), ('BK006', 'Fundamentals of Heat', '0000979006', 'CA001', 'AUT006', 'P005', '2003-08-10', 'German', 247, 112.00), ('BK007', 'Advanced 3d Graphics', '0000979007', 'CA003', 'AUT007', 'P002', '2004-02-16', 'Hindi', 165, 56.00), ('BK008', 'Human Anatomy', '0000979008', 'CA005', 'AUT008', 'P006', '2001-05-17', 'German', 88, 50.50), ('BK009', 'Mental Health Nursing', '0000979009', 'CA005', 'AUT009', 'P007', '2004-02-10', 'English', 350, 145.00), ('BK010', 'Fundamentals of Thermodynamics', '0000979010', 'CA002', 'AUT010', 'P007', '2002-10- 14', 'English', 400, 225.00), ('BK011', 'The Experimental Analysis of Cat', '0000979011', 'CA004', 'AUT011', 'P005', '2007-06- 09', 'French', 225, 95.00), ('BK012', 'The Nature of World', '0000979012', 'CA004', 'AUT005', 'P008', '2005-12-20', 'English', 350, 88.00), ('BK013', 'Environment a Sustainable Future', '0000979013', 'CA004', 'AUT012', 'P001', '2003-10- 27', 'German', 165, 100.00), ('BK014', 'Concepts in Health', '0000979014', 'CA005', 'AUT013', 'P004', '2001-08-25', NULL, 320, 180.00), ('BK015', 'Anatomy & Physiology', '0000979015', 'CA005', 'AUT014', 'P008', '2000-10-10', 'Hindi', 225, 135.00), ('BK016', 'Networks and Telecommunications', '00009790_16', 'CA003', 'AUT015', 'P003', '2002- 01-01', 'French', 95, 45.00);

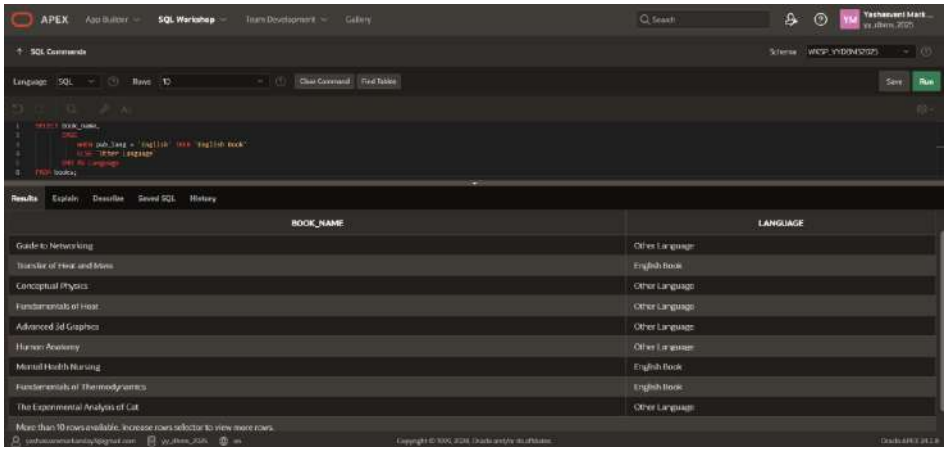
69. select * from books;



BOOK_ID	BOOK_NAME	ISBN_NO	CATE_ID	AUT_ID	PUB_ID	DT_OF_PUB	PUB_LANG	NO_PAGE	BOOK_PRICE
BK002	Understanding of Steel Construction	0000979002	CA002	AUT002	P001	2/15/2005	English	330	305.5
BK005	Guide to Networking	0000979005	CA003	AUT003	P002	9/10/2002	Hindi	510	200
BK004	Transfer of Heat and Mass	0000979004	CA002	AUT004	P004	2/15/2004	English	600	250
BK005	Conceptual Physics	0000979005	CA001	AUT005	P005	2/15/2005	-	345	345
BK006	Fundamentals of Heat	0000979006	CA001	AUT006	P005	8/10/2003	German	287	112
BK007	Advanced 3d Graphics	0000979007	CA003	AUT007	P002	2/15/2004	Hindi	165	56
BK008	Human Anatomy	0000979008	CA005	AUT008	P006	5/12/2001	German	88	505
BK009	Mental Health Nursing	0000979009	CA005	AUT009	P007	2/10/2004	English	350	345
BK010	Fundamentals of Thermodynamics	0000979010	CA002	AUT010	P007	11/15/2002	English	400	225
BK011	The Experimental Analysis of Cat	0000979011	CA004	AUT011	P005	6/6/2007	French	225	95

More than 10 rows available. Increase rows selector to view more rows.

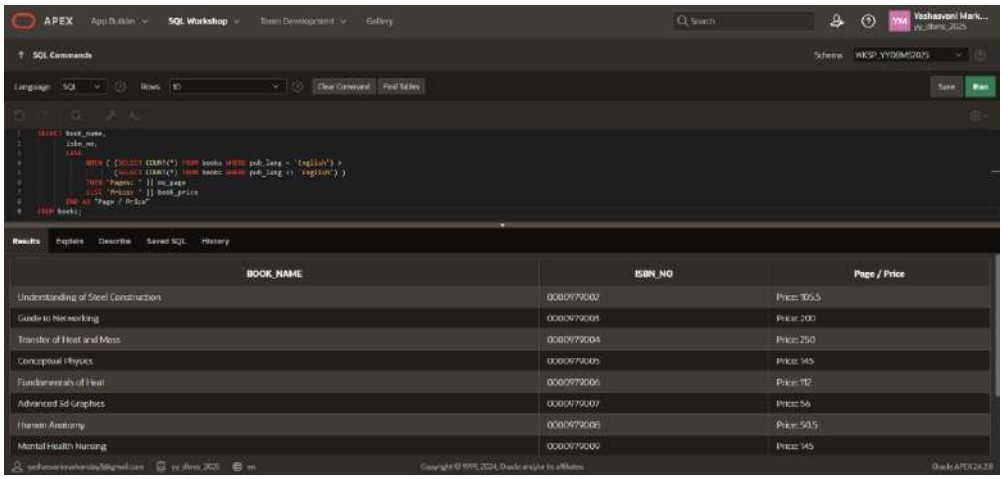
70. SELECT book_name, IF(pub_lang="English", "Engllish Book", "Other Lnaguage") AS Language FROM books;



BOOK_NAME	LANGUAGE
Guide to Networking	Other Language
Transfer of Heat and Mass	English Book
Conceptual Physics	Other Language
Fundamentals of Heat	Other Language
Advanced 3d Graphics	Other Language
Human Anatomy	Other Language
Mental Health Nursing	English Book
Fundamentals of Thermodynamics	English Book
The Experimental Analysis of Cat	Other Language

More than 10 rows available. Increase rows selector to view more rows.

71. SELECT book_name,isbn_no, IF((SELECT COUNT(*) FROM books WHERE pub_lang='English')> (SELECT COUNT(*) FROM books WHERE pub_lang<>'English'), (CONCAT("Pages: ",no_page)),(CONCAT("Price: ",book_price))) AS "Page / Price" FROM books;



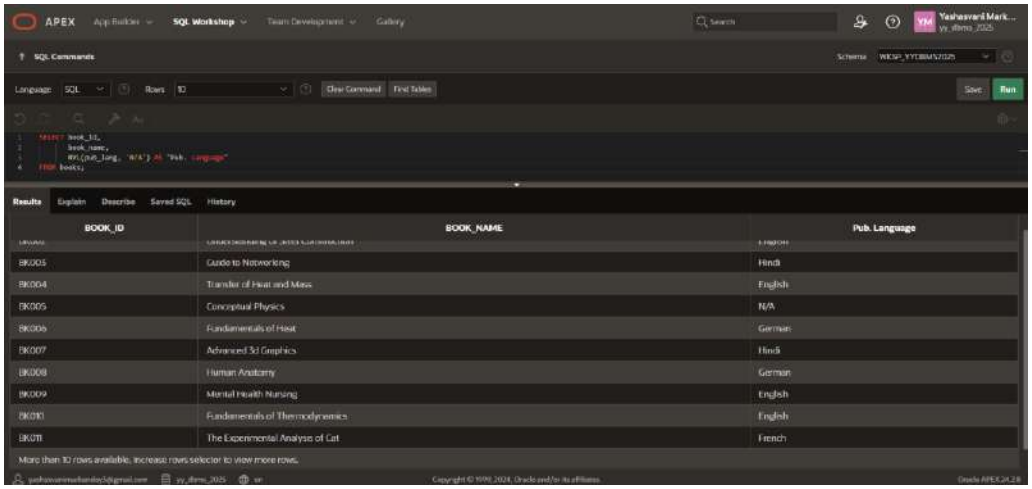
BOOK_NAME	ISBN_NO	Page / Price
Understanding of Steel Construction	0000979002	Price: 305.5
Guide to Networking	0000979005	Price: 200
Transfer of Heat and Mass	0000979004	Price: 250
Conceptual Physics	0000979005	Price: 345
Fundamentals of Heat	0000979006	Price: 112
Advanced 3d Graphics	0000979007	Price: 56
Human Anatomy	0000979008	Price: 505
Mental Health Nursing	0000979009	Price: 345

BOOK_NAME	ISBN_NO	Page / Price
Fundamentals of Heat	0000979006	Price: 112
Advanced 3d Graphics	0000979007	Price: 56
Human Anatomy	0000979008	Price: 50.5
Mental Health Nursing	0000979009	Price: 145
Fundamentals of Thermodynamics	0000979010	Price: 225
The Experimental Analysis of Cat	0000979011	Price: 95

More than 10 rows available. Increase rows selector to view more rows.

10 rows returned in 0.02 seconds Download

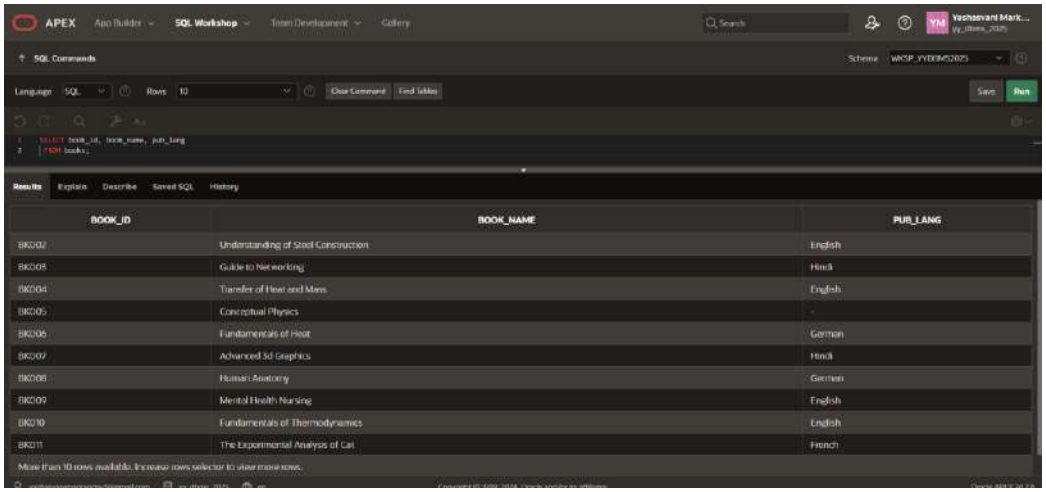
72. SELECT book_id, book_name, IF(pub_lang IS NULL,'N/A',pub_lang) AS "Pub. Language" FROM books;



BOOK_ID	BOOK_NAME	Pub. Language
BK005	Guide to Networking	Hindi
BK004	Transfer of Heat and Mass	English
BK005	Conceptual Physics	N/A
BK006	Fundamentals of Heat	German
BK007	Advanced 3d Graphics	Hindi
BK008	Human Anatomy	German
BK009	Mental Health Nursing	English
BK010	Fundamentals of Thermodynamics	English
BK011	The Experimental Analysis of Cat	French

More than 10 rows available. Increase rows selector to view more rows.

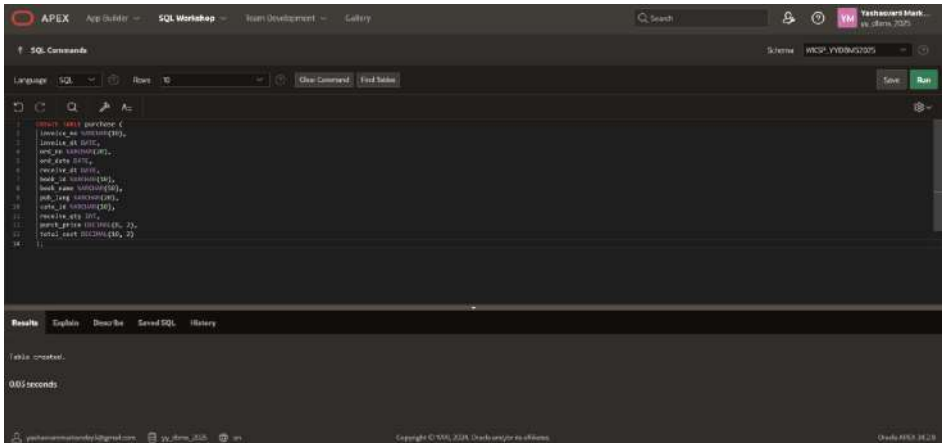
73. SELECT book_id, book_name, pub_lang FROM books;



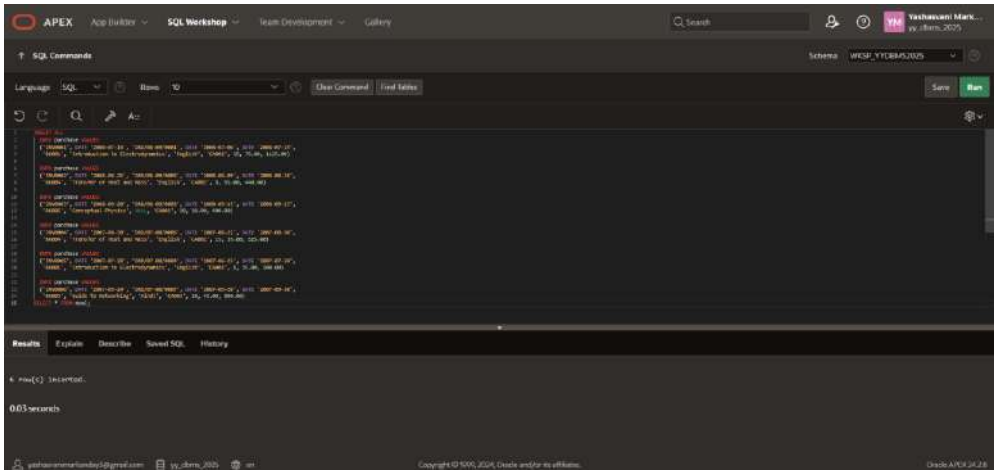
BOOK_ID	BOOK_NAME	PUB_LANG
BK002	Understanding of Steel Construction	English
BK005	Guide to Networking	Hindi
BK004	Transfer of Heat and Mass	English
BK005	Conceptual Physics	-
BK006	Fundamentals of Heat	German
BK007	Advanced 3d Graphics	Hindi
BK008	Human Anatomy	German
BK009	Mental Health Nursing	English
BK010	Fundamentals of Thermodynamics	English
BK011	The Experimental Analysis of Cat	French

More than 10 rows available. Increase rows selector to view more rows.

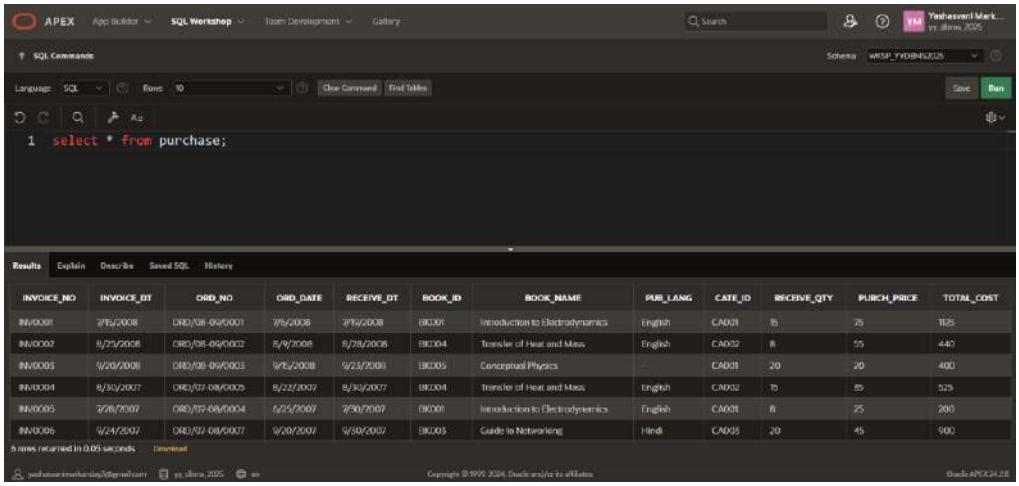
74. CREATE TABLE purchase (invoice_no VARCHAR(10), invoice_dt DATE, ord_no VARCHAR(20), ord_date DATE, receive_dt DATE, book_id VARCHAR(10), book_name VARCHAR(50), pub_lang VARCHAR(20), cate_id VARCHAR(10), receive_qty INT, purch_price DECIMAL(8, 2), total_cost DECIMAL(10, 2));



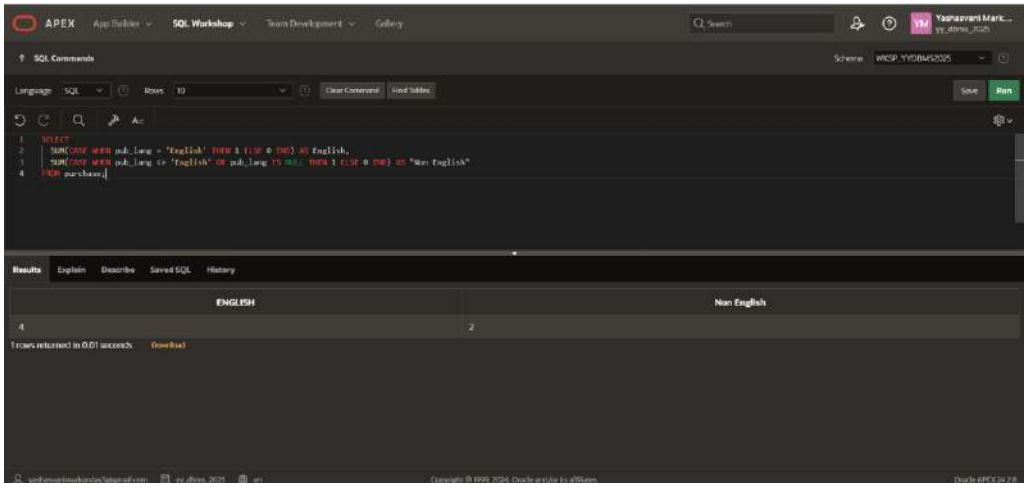
75. INSERT INTO purchase VALUES ('INV0001', '2008-07-15', 'ORD/08-09/0001', '2008-07-06', '2008-07-19', 'BK001', 'Introduction to Electrodynamics', 'English', 'CA001', 15, 75.00, 1125.00), ('INV0002', '2008-08-25', 'ORD/08-09/0002', '2008-08-09', '2008-08-28', 'BK004', 'Transfer of Heat and Mass', 'English', 'CA002', 8, 55.00, 440.00), ('INV0003', '2008-09-20', 'ORD/08-09/0003', '2008-09-15', '2008-09-23', 'BK005', 'Conceptual Physics', NULL, 'CA001', 20, 20.00, 400.00), ('INV0004', '2007-08-30', 'ORD/07-08/0005', '2007-08-22', '2007-08-30', 'BK004', 'Transfer of Heat and Mass', 'English', 'CA002', 15, 35.00, 525.00), ('INV0005', '2007-07-28', 'ORD/07-08/0004', '2007-06-25', '2007-07-30', 'BK001', 'Introduction to Electrodynamics', 'English', 'CA001', 8, 25.00, 200.00), ('INV0006', '2007-09-24', 'ORD/07-08/0007', '2007-09-20', '2007-09-30', 'BK003', 'Guide to Networking', 'Hindi', 'CA003', 20, 45.00, 900.00);



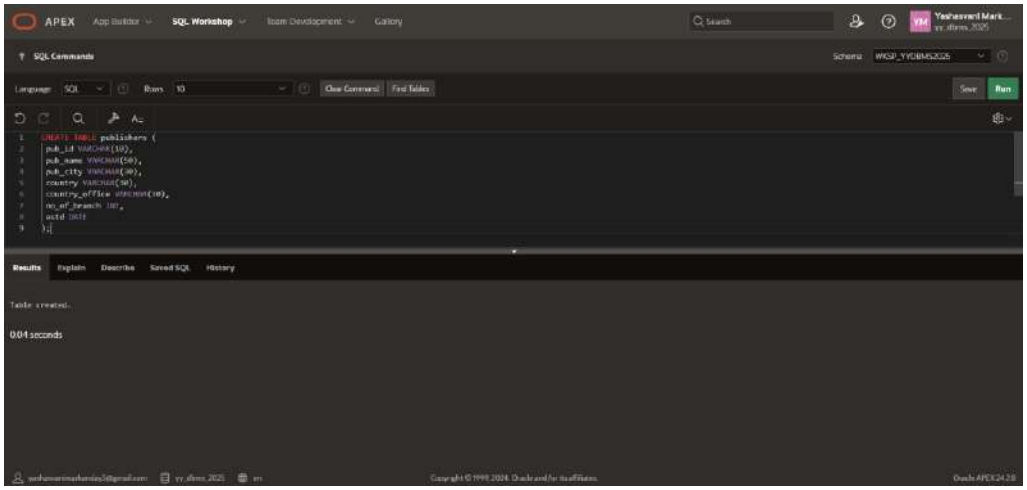
76. select * from purchase;



77. SELECT SUM(IF(pub_lang = 'English',1,0)) AS English, SUM(IF(pub_lang <> 'English',1,0)) AS "Non English" FROM purchase;

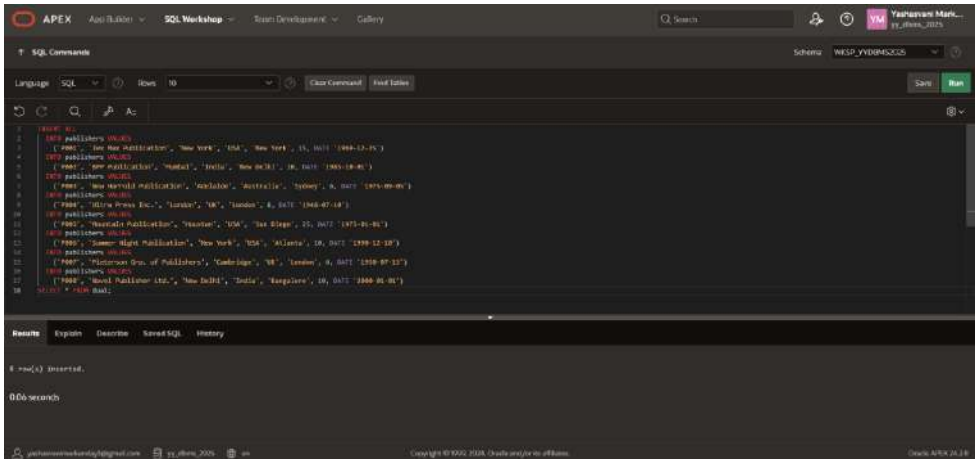


78. CREATE TABLE publishers (pub_id VARCHAR(10), pub_name VARCHAR(50), pub_city VARCHAR(30), country VARCHAR(30), country_office VARCHAR(30), no_of_branch INT, estd DATE);

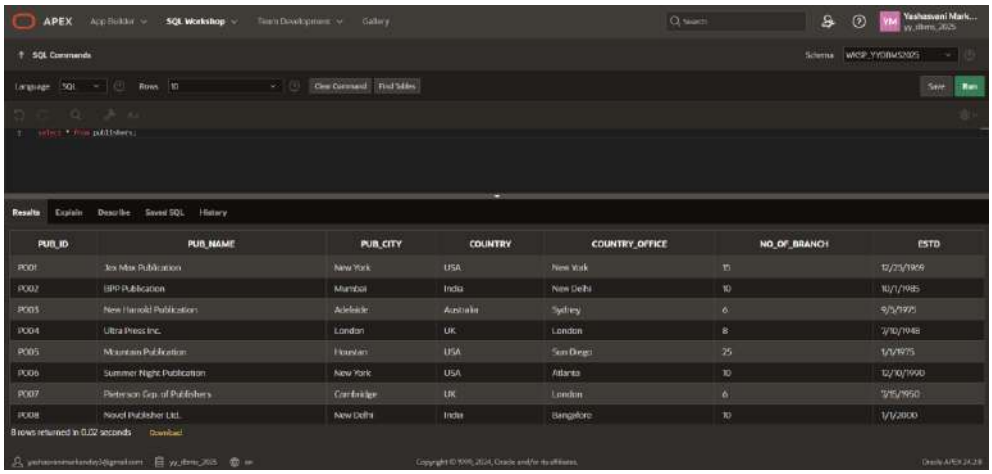


79. INSERT INTO publishers VALUES ('P001', 'Jex Max Publication', 'New York', 'USA', 'New York', 15, '1969-12-25'), ('P002', 'BPP Publication', 'Mumbai', 'India', 'New Delhi', 10, '1985-10-01'), ('P003', 'New Harrold Publication', 'Adelaide', 'Australia', 'Sydney', 6, '1975-09-05'), ('P004', 'Ultra Press Inc.', 'London', 'UK', 'London', 8, '1948-07-10'), ('P005', 'Mountain Publication', 'Houstan', 'USA', 'Sun

Diego', 25, '1975-01-01'), ('P006', 'Summer Night Publication', 'New York', 'USA', 'Atlanta', 10, '1990-12-10'), ('P007', 'Pieterson Grp. of Publishers', 'Cambridge', 'UK', 'London', 6, '1950-07-15'), ('P008', 'Novel Publisher Ltd.', 'New Delhi', 'India', 'Bangalore', 10, '2000-01-01');



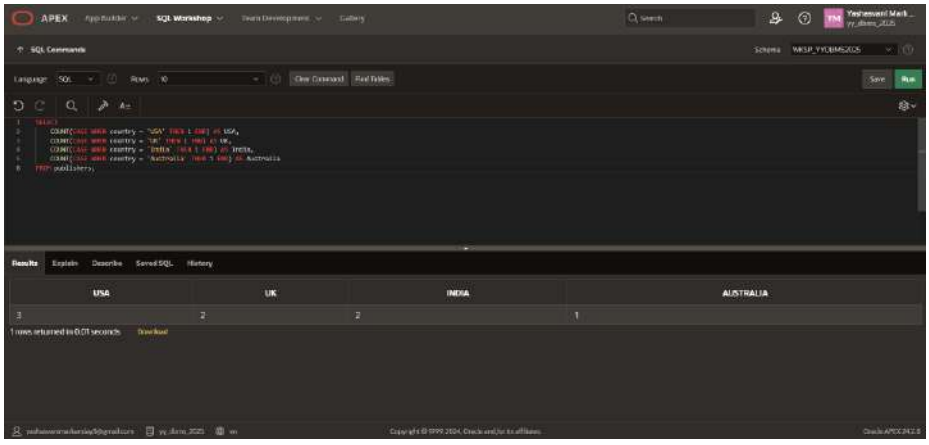
80. select * from publishers;



PUB_ID	PUB_NAME	PUB_CITY	COUNTRY	COUNTRY_OFFICE	NO_OF_BRANCH	ESTD
P001	Jex Max Publication	New York	USA	New York	15	12/25/1969
P002	BPP Publication	Mumbai	India	New Delhi	10	10/1/1985
P003	New Harold Publication	Adelaide	Australia	Sydney	6	9/9/1995
P004	Ultra Press Inc.	London	UK	London	8	7/10/1948
P005	Mountain Publication	Houston	USA	San Diego	25	1/1/1975
P006	Summer Night Publication	New York	USA	Atlanta	10	12/10/1990
P007	Pieterson Grp. of Publishers	Cambridge	UK	London	6	7/15/1950
P008	Novel Publisher Ltd.	New Delhi	India	Bangalore	10	1/1/2000

8 rows returned in 0.02 seconds Download

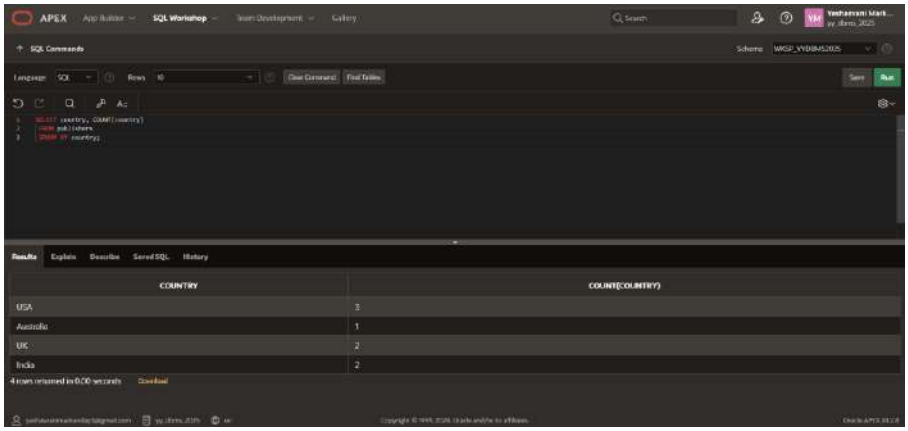
81. SELECT COUNT(IF(country = 'USA',1,NULL)) USA, COUNT(IF(country = 'UK',1,NULL)) UK, COUNT(IF(country = 'India',1,NULL)) India, COUNT(IF(country = 'Australia',1,NULL)) Australia FROM publishers; Another way to achieve the similar result you can use the GROUP BY clause and the COUNT function without using the IF function, the display report is quite different.



USA	UK	INDIA	AUSTRALIA
3	2	2	1

1 rows returned in 0.01 seconds Download

82. SELECT country, COUNT(country) FROM publishers GROUP BY country;


APEX
App Builder
SQL Workshop
Team Development
Gallery
SQL Commands
Schema:
Language:
SQL
Rows
10
Clear Command
Find Tables
Save
Run
SELECT country, COUNT(country)
FROM publishers
GROUP BY country;
Results
Explain
Describe
Saved SQL
History
COUNTRY
COUNT(COUNTRY)
USA
3
Australia
1
UK
2
India
2
4 rows returned in 0.00 seconds
Download

UNIVERSITY OF PETROLEUM AND ENERGY STUDIES

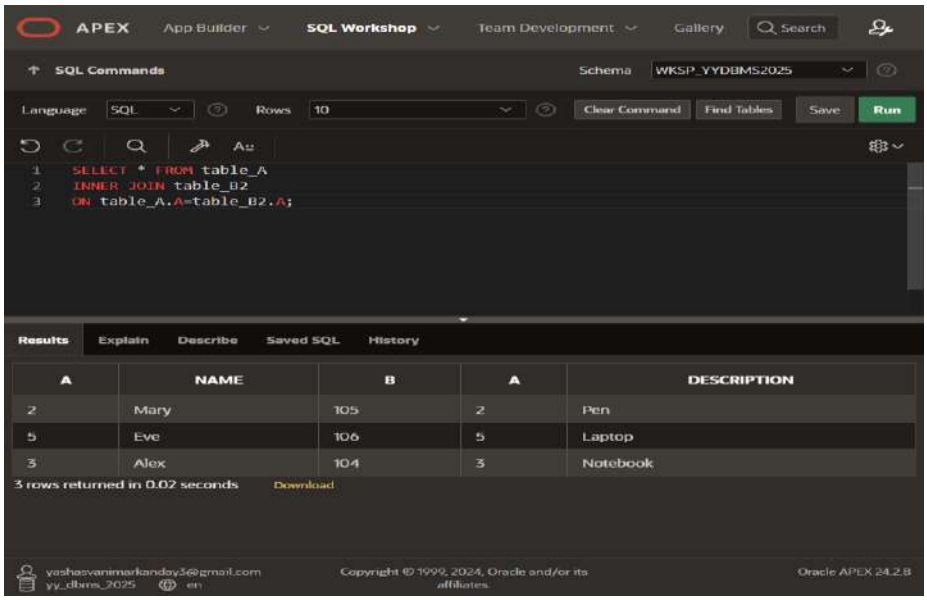
Program Name: B. Tech CS **Semester: III**

Name: Yashasvani Markanday **SAP ID: 590011386**

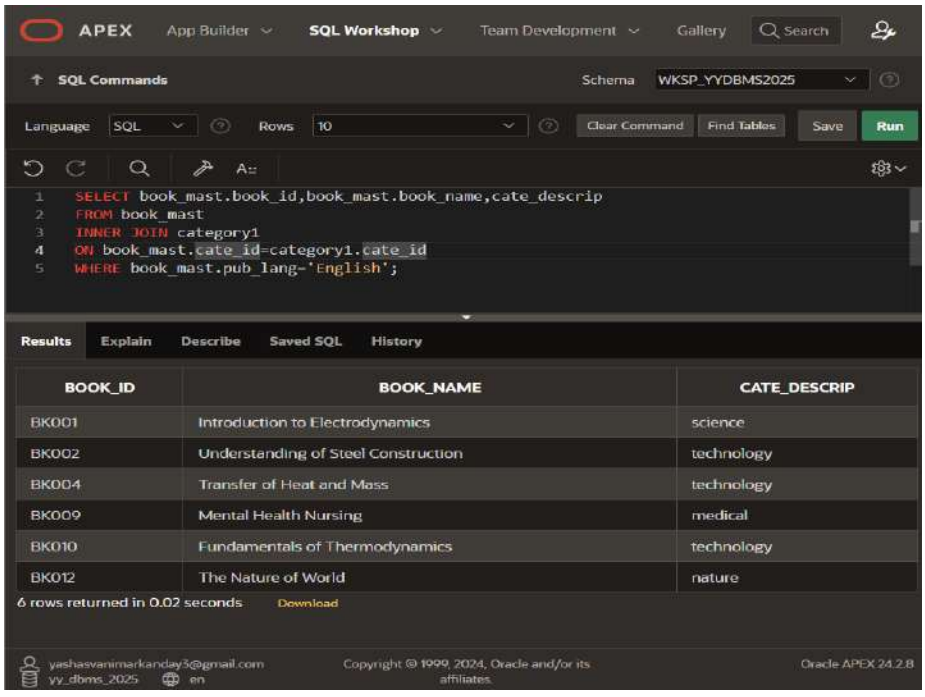
Batch: B2 **Subject: DBMS**

LAB - 7

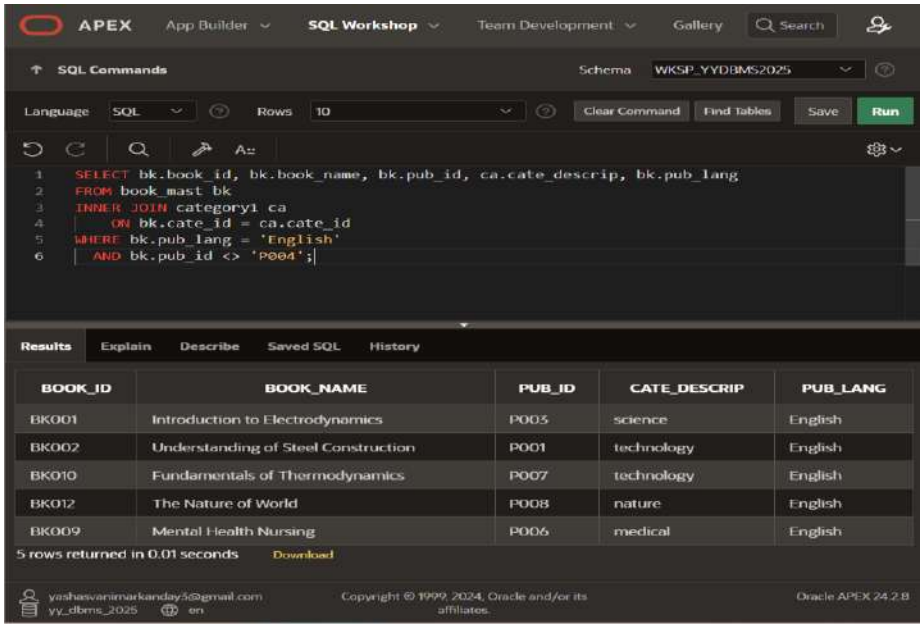
1.)



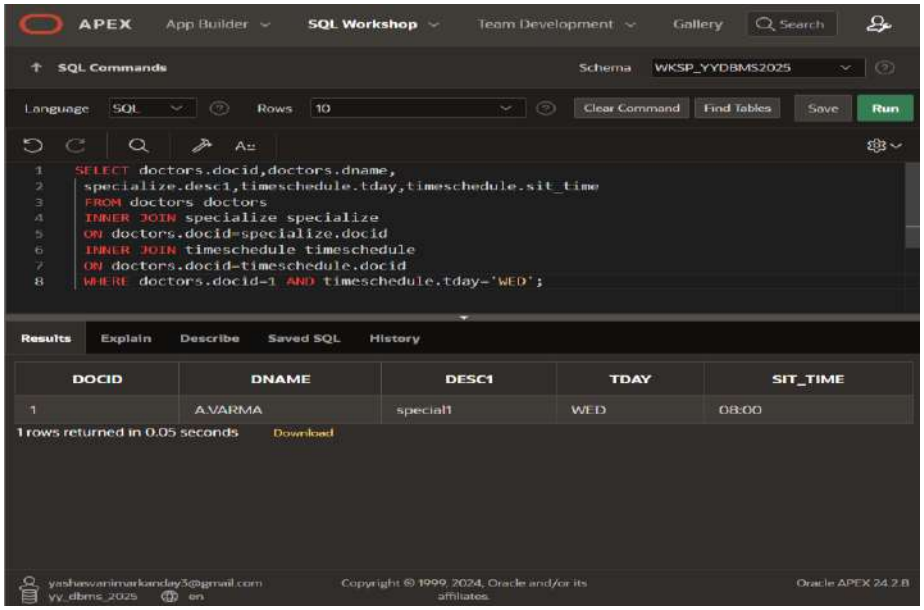
2.)



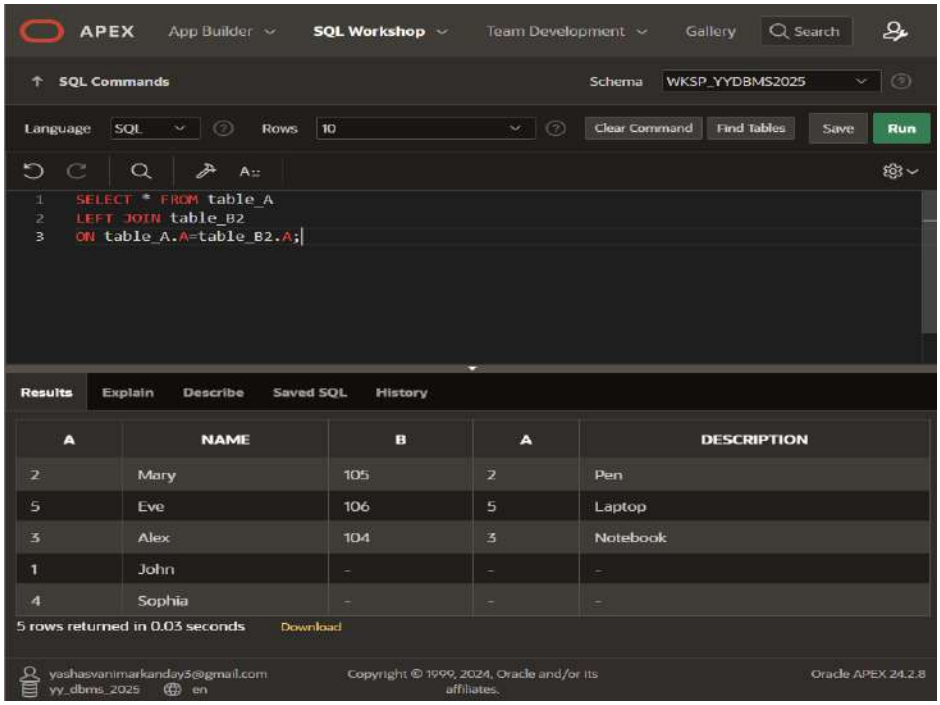
3.)


APEX
App Builder
SQL Workshop
Team Development
Gallery
Search
SQL Commands
Schema
WKSP_YYDBMS2025
Language
SQL
Rows
10
Clear Command
Find Tables
Save
Run
SELECT bk.book_id, bk.book_name, bk.pub_id, ca.cate_descrip, bk.pub_lang
FROM book_mast bk
INNER JOIN category1 ca
ON bk.cate_id = ca.cate_id
WHERE bk.pub_lang = 'English'
AND bk.pub_id <> 'P004';
Results
Explain
Describe
Saved SQL
History
BOOK_ID
BOOK_NAME
PUB_ID
CATE_DESCRIP
PUB_LANG
BK001
Introduction to Electrodynamics
P003
science
English
BK002
Understanding of Steel Construction
P001
technology
English
BK010
Fundamentals of Thermodynamics
P007
technology
English
BK012
The Nature of World
P008
nature
English
BK009
Mental Health Nursing
P006
medical
English
5 rows returned in 0.01 seconds
Download
yashaswanimarkanday3@gmail.com
yy_dbms_2025
en
Copyright © 1999, 2024, Oracle and/or its affiliates.
Oracle APEX 24.2.8

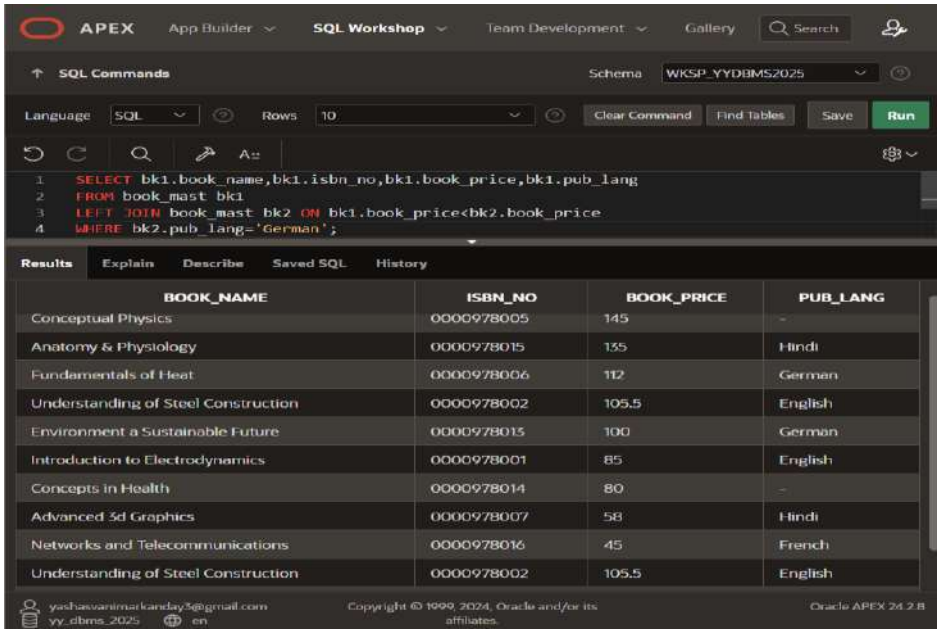
4.)


APEX
App Builder
SQL Workshop
Team Development
Gallery
Search
SQL Commands
Schema
WKSP_YYDBMS2025
Language
SQL
Rows
10
Clear Command
Find Tables
Save
Run
SELECT doctors.docid,doctors.dname,
specialize.desc1,timeschedule.tday,timeschedule.sit_time
FROM doctors doctors
INNER JOIN specialize specialize
ON doctors.docid=specialize.docid
INNER JOIN timeschedule timeschedule
ON doctors.docid=timeschedule.docid
WHERE doctors.docid=1 AND timeschedule.tday='WED';
Results
Explain
Describe
Saved SQL
History
DOCID
DNAME
DESC1
TDAY
SIT_TIME
1
A.VARMA
special1
WED
08:00
1 rows returned in 0.05 seconds
Download
yashaswanimarkanday3@gmail.com
yy_dbms_2025
en
Copyright © 1999, 2024, Oracle and/or its affiliates.
Oracle APEX 24.2.8

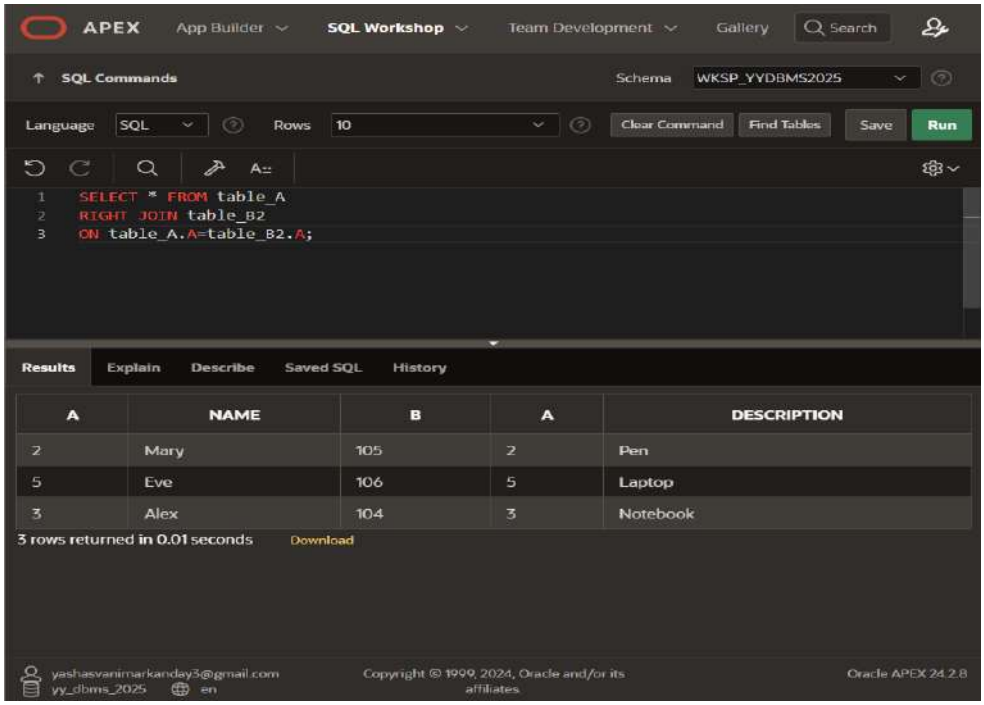
5.)


APEX
App Builder
SQL Workshop
Team Development
Gallery
Search
SQL Commands
Schema
WKSP_YYDBMS2025
Language
SQL
Rows
10
Clear Command
Find Tables
Save
Run
SELECT * FROM table_A
LEFT JOIN table_B2
ON table_A.A=table_B2.A;
Results
Explain
Describe
Saved SQL
History
A
NAME
B
A
DESCRIPTION
2 Mary 105 2 Pen
5 Eve 106 5 Laptop
3 Alex 104 3 Notebook
1 John - - -
4 Sophia - - -
5 rows returned in 0.03 seconds
Download
yashasvanimarkanday5@gmail.com
yy_dbms_2025
en
Copyright © 1999, 2024, Oracle and/or its affiliates.
Oracle APEX 24.2.8

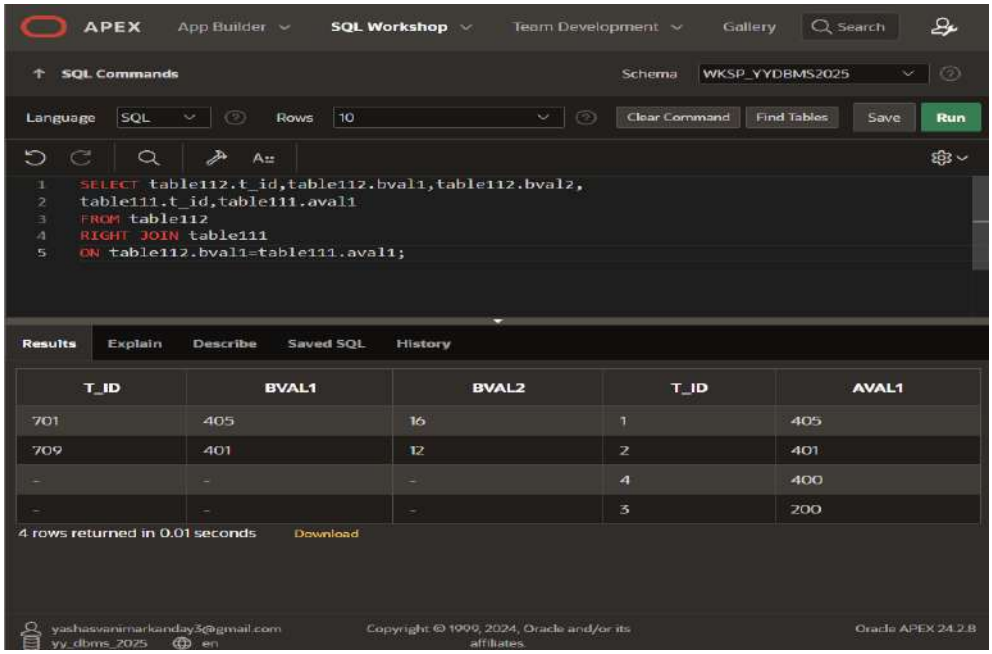
6.)


APEX
App Builder
SQL Workshop
Team Development
Gallery
Search
SQL Commands
Schema
WKSP_YYDBMS2025
Language
SQL
Rows
10
Clear Command
Find Tables
Save
Run
SELECT bk1.book_name,bk1.isbn_no,bk1.book_price,bk1.pub_lang
FROM book_mast bk1
LEFT JOIN book_mast bk2 ON bk1.book_price<bk2.book_price
WHERE bk2.pub_lang='German';
Results
Explain
Describe
Saved SQL
History
BOOK_NAME
ISBN_NO
BOOK_PRICE
PUB_LANG
Conceptual Physics 0000978005 145 -
Anatomy & Physiology 0000978015 135 Hindi
Fundamentals of Heat 0000978006 112 German
Understanding of Steel Construction 0000978002 105.5 English
Environment a Sustainable Future 0000978013 100 German
Introduction to Electrodynamics 0000978001 85 English
Concepts in Health 0000978014 80 -
Advanced 3d Graphics 0000978007 58 Hindi
Networks and Telecommunications 0000978016 45 French
Understanding of Steel Construction 0000978002 105.5 English
yashasvanimarkanday5@gmail.com
yy_dbms_2025
en
Copyright © 1999, 2024, Oracle and/or its affiliates.
Oracle APEX 24.2.8

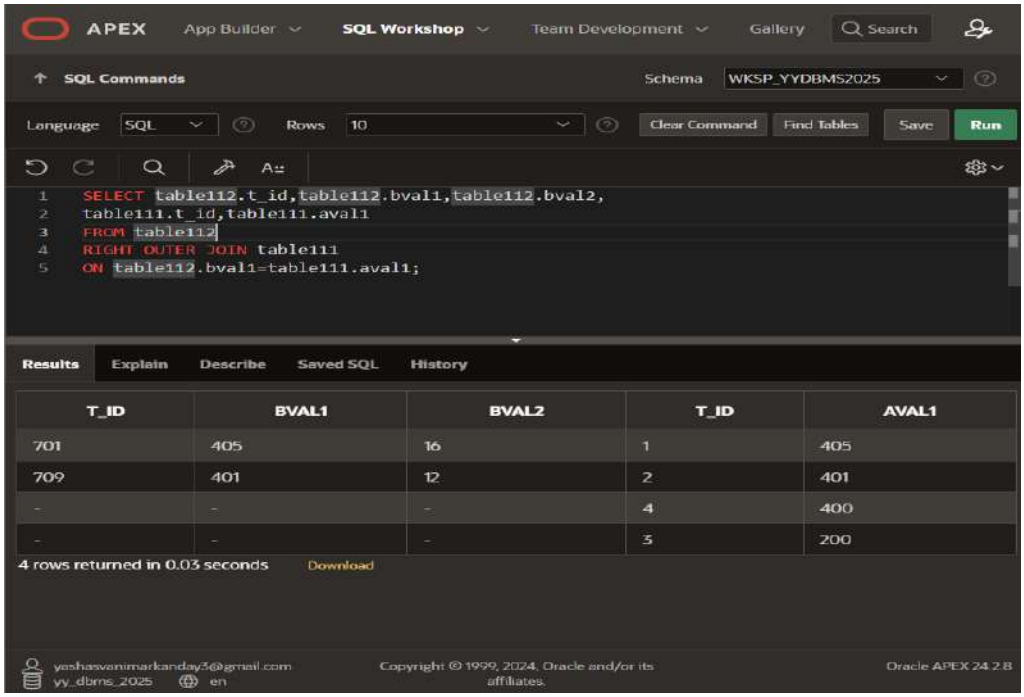
7.)


APEX
App Builder
SQL Workshop
Team Development
Gallery
Search
SQL Commands
Schema
WKSP_YYDBMS2025
Language
SQL
Rows
10
Clear Command
Find Tables
Save
Run
SELECT * FROM table_A
RIGHT JOIN table_B2
ON table_A.A=table_B2.A;
Results
Explain
Describe
Saved SQL
History
A
NAME
B
A
DESCRIPTION
2
Mary
105
2
Pen
5
Eve
106
5
Laptop
3
Alex
104
3
Notebook
3 rows returned in 0.01 seconds
Download
yashasvanimarkanday3@gmail.com
yy_dbms_2025
en
Copyright © 1999, 2024, Oracle and/or its affiliates
Oracle APEX 24.2.8

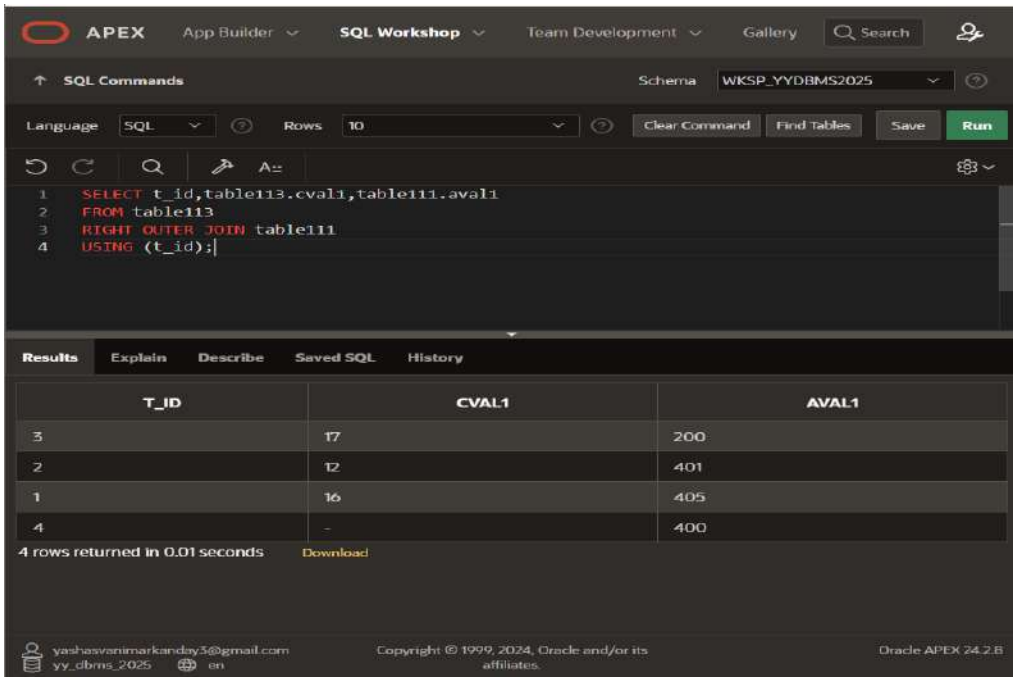
8.)


APEX
App Builder
SQL Workshop
Team Development
Gallery
Search
SQL Commands
Schema
WKSP_YYDBMS2025
Language
SQL
Rows
10
Clear Command
Find Tables
Save
Run
SELECT table112.t_id,table112.bval1,table112.bval2,
table111.t_id,table111.aval1
FROM table112
RIGHT JOIN table111
ON table112.bval1=table111.aval1;
Results
Explain
Describe
Saved SQL
History
T_ID
BVAL1
BVAL2
T_ID
AVAL1
701
405
16
1
405
709
401
12
2
401
-
-
-
4
400
-
-
-
3
200
4 rows returned in 0.01 seconds
Download
yashasvanimarkanday3@gmail.com
yy_dbms_2025
en
Copyright © 1999, 2024, Oracle and/or its affiliates
Oracle APEX 24.2.8

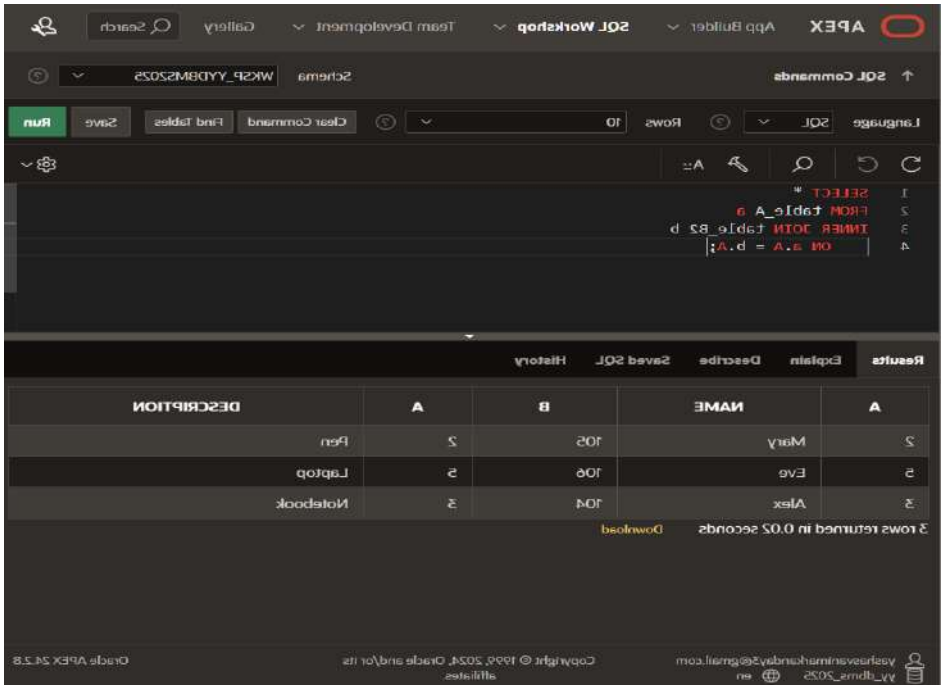
9.)


APEX
App Builder
SQL Workshop
Team Development
Gallery
Search
SQL Commands
Schema
WKSP_YYDBMS2025
Language
SQL
Rows
10
Clear Command
Find Tables
Save
Run
SELECT table112.t_id,table112.bval1,table112.bval2,
table111.t_id,table111.aval1
FROM table112
RIGHT OUTER JOIN table111
ON table112.bval1=table111.aval1;
Results
Explain
Describe
Saved SQL
History
T_ID
BVAL1
BVAL2
T_ID
AVAL1
701
405
16
1
405
709
401
12
2
401
-
-
-
4
400
-
-
-
3
200
4 rows returned in 0.03 seconds
Download
yashasvanimarkanday3@gmail.com
yy_dbms_2025
en
Copyright © 1999, 2024, Oracle and/or its affiliates.
Oracle APEX 24.2.8

10.)

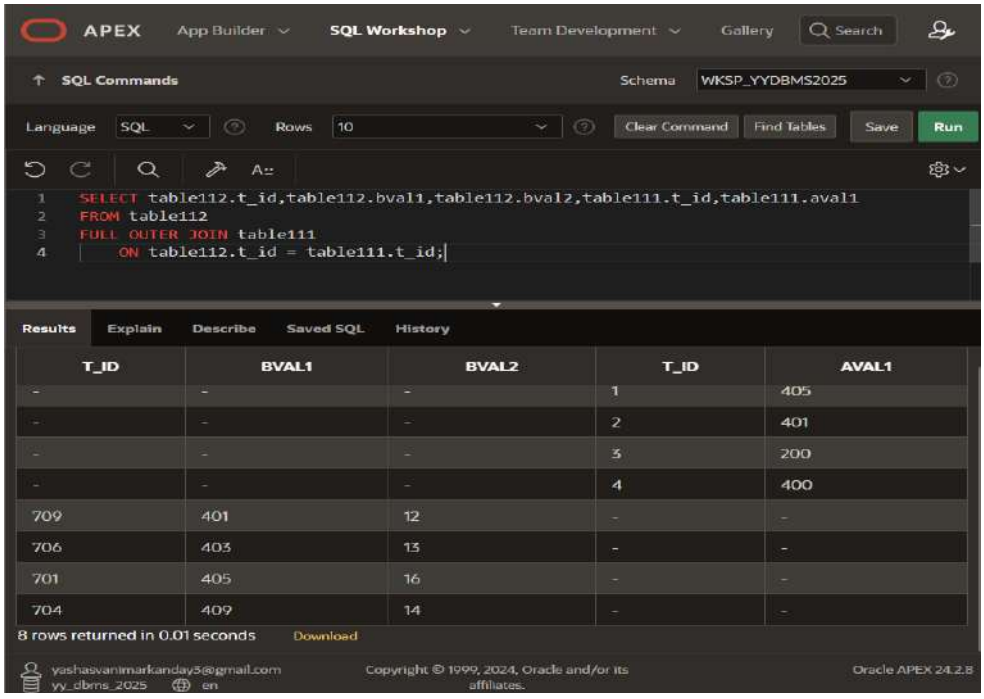

APEX
App Builder
SQL Workshop
Team Development
Gallery
Search
SQL Commands
Schema
WKSP_YYDBMS2025
Language
SQL
Rows
10
Clear Command
Find Tables
Save
Run
SELECT t_id,table113.cval1,table111.aval1
FROM table113
RIGHT OUTER JOIN table111
USING (t_id);
Results
Explain
Describe
Saved SQL
History
T_ID
CVAL1
AVAL1
3
17
200
2
12
401
1
16
405
4
-
400
4 rows returned in 0.01 seconds
Download
yashasvanimarkanday3@gmail.com
yy_dbms_2025
en
Copyright © 1999, 2024, Oracle and/or its affiliates.
Oracle APEX 24.2.8

11.)

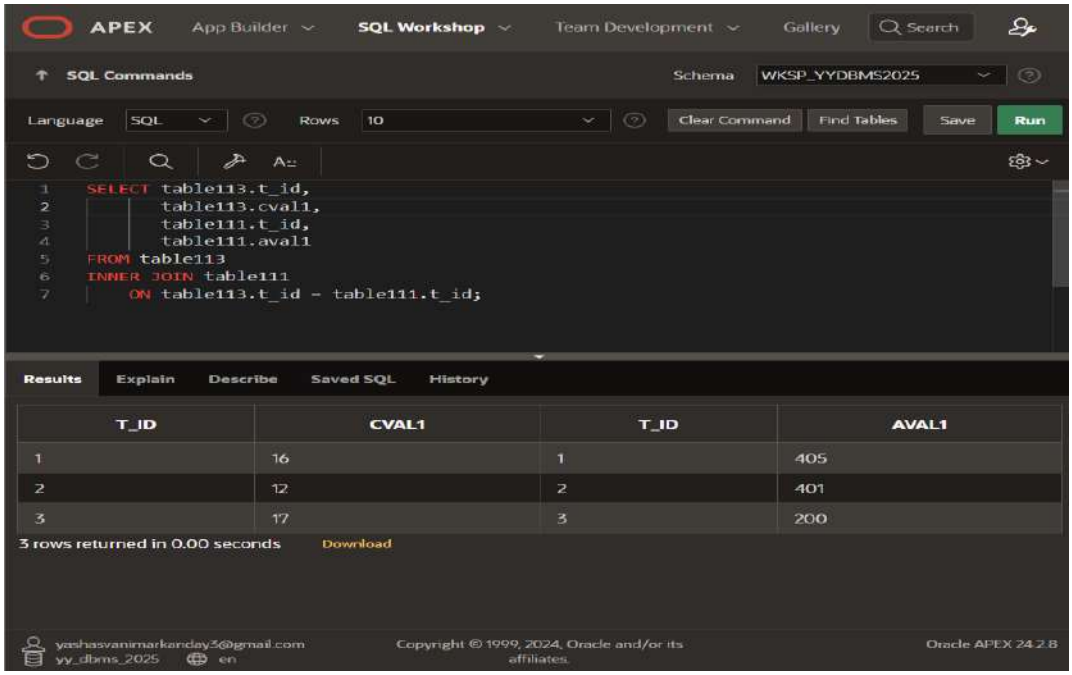


As there is not straight join in oracle so inner join is equivalent to straight join only!

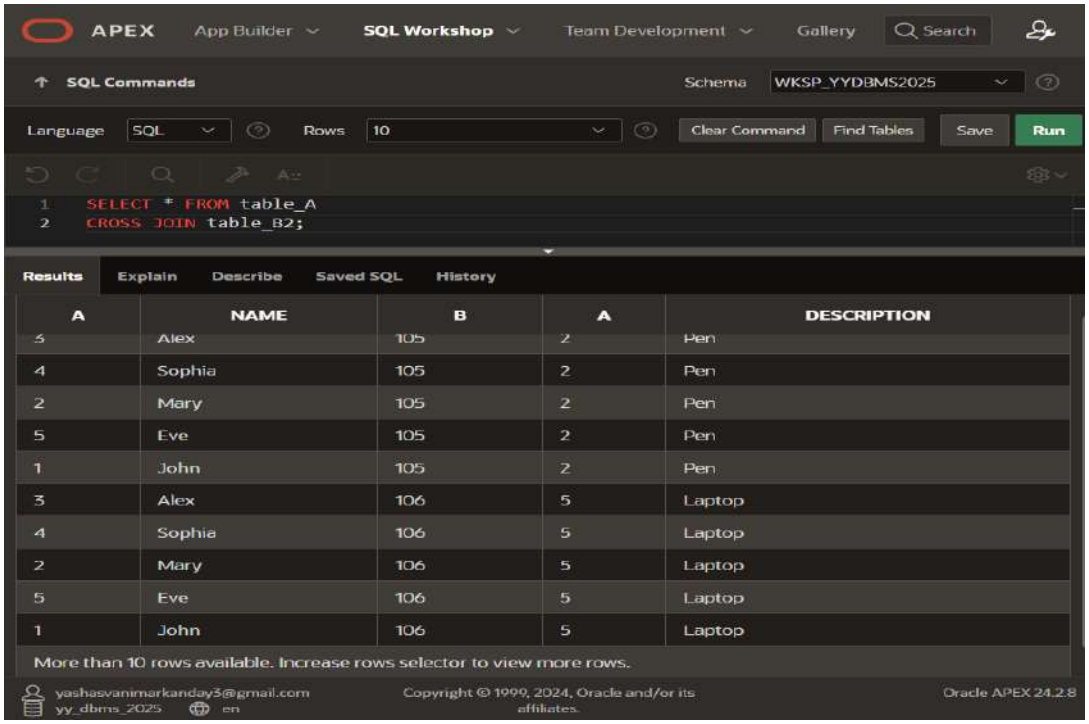
12.)



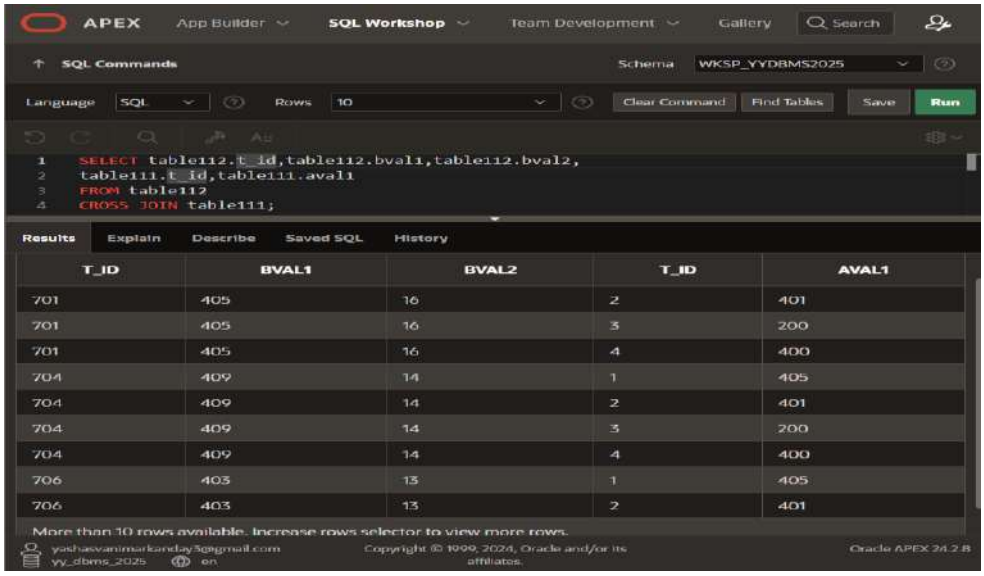
13.)


APEX
App Builder
SQL Workshop
Team Development
Gallery
Search
SQL Commands
Schema
WKSP_YYDBMS2025
Language
SQL
Rows
10
Clear Command
Find Tables
Save
Run
SELECT table113.t_id,
table113.cval1,
table111.t_id,
table111.aval1
FROM table113
INNER JOIN table111
ON table113.t_id = table111.t_id;
Results
Explain
Describe
Saved SQL
History
T_ID
CVAL1
T_ID
AVAL1
1
16
1
405
2
12
2
401
3
17
3
200
3 rows returned in 0.00 seconds
Download
yashasvanimarkanday3@gmail.com
yy_dbms_2025
en
Copyright © 1999, 2024, Oracle and/or its affiliates.
Oracle APEX 24.2.8

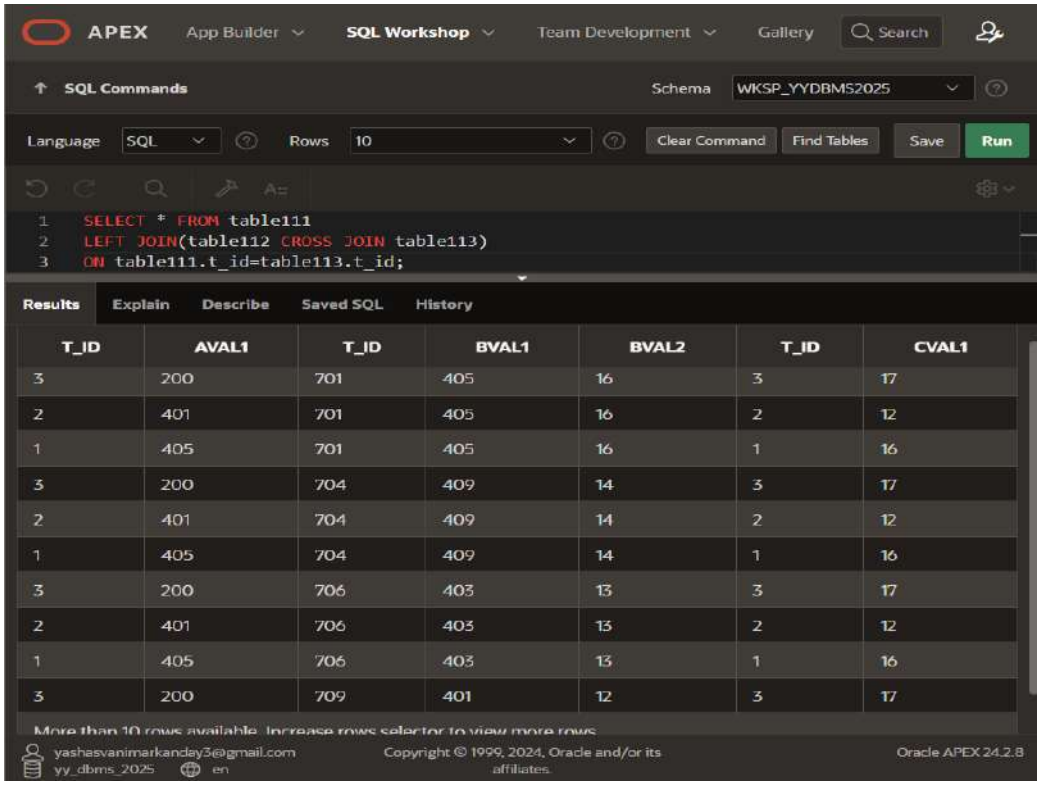
14.)


APEX
App Builder
SQL Workshop
Team Development
Gallery
Search
SQL Commands
Schema
WKSP_YYDBMS2025
Language
SQL
Rows
10
Clear Command
Find Tables
Save
Run
SELECT * FROM table_A
CROSS JOIN table_B2;
Results
Explain
Describe
Saved SQL
History
A
NAME
B
A
DESCRIPTION
3
Alex
105
2
Pen
4
Sophia
105
2
Pen
2
Mary
105
2
Pen
5
Eve
105
2
Pen
1
John
105
2
Pen
3
Alex
106
5
Laptop
4
Sophia
106
5
Laptop
2
Mary
106
5
Laptop
5
Eve
106
5
Laptop
1
John
106
5
Laptop
More than 10 rows available. Increase rows selector to view more rows.
yashasvanimarkanday3@gmail.com
yy_dbms_2025
en
Copyright © 1999, 2024, Oracle and/or its affiliates.
Oracle APEX 24.2.8

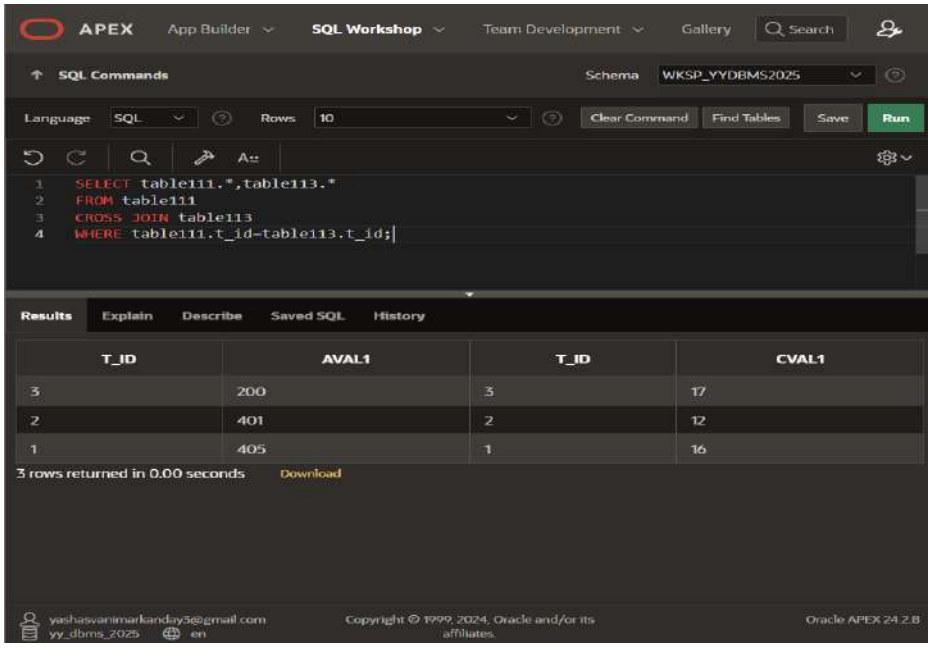
15.)


APEX
App Builder
SQL Workshop
Team Development
Gallery
Search
SQL Commands
Schema
WKSP_YYDBMS2025
Language
SQL
Rows
10
Clear Command
Find Tables
Save
Run
1 SELECT table112.t_id,table112.bval1,table112.bval2,
2 table111.t_id,table111.aval1
3 FROM table112
4 CROSS JOIN table111;
Results
Explain
Describe
Saved SQL
History
T_ID
BVAL1
BVAL2
T_ID
AVAL1
701 405 16 2 401
701 405 16 3 200
701 405 16 4 400
704 409 14 1 405
704 409 14 2 401
704 409 14 3 200
704 409 14 4 400
706 403 13 1 405
706 403 13 2 401
More than 10 rows available. Increase rows selector to view more rows.
yashasvanimarkanday3@gmail.com
yy_dbms_2025
en
Copyright © 1999, 2024, Oracle and/or its affiliates.
Oracle APEX 24.2.8

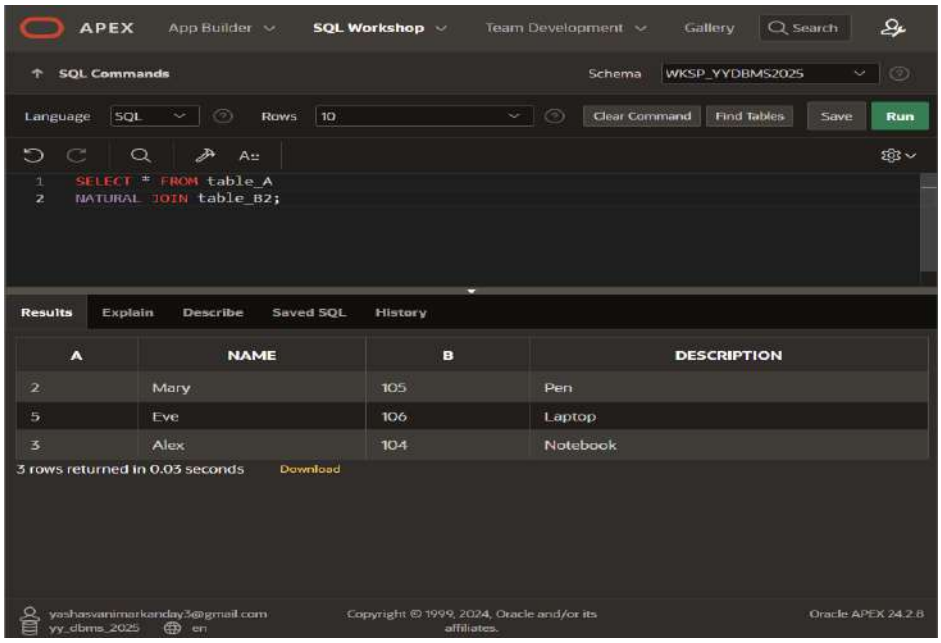
16.)


APEX
App Builder
SQL Workshop
Team Development
Gallery
Search
SQL Commands
Schema
WKSP_YYDBMS2025
Language
SQL
Rows
10
Clear Command
Find Tables
Save
Run
1 SELECT * FROM table111
2 LEFT JOIN(table112 CROSS JOIN table113)
3 ON table111.t_id=table113.t_id;
Results
Explain
Describe
Saved SQL
History
T_ID AVAL1 T_ID BVAL1 BVAL2 T_ID CVAL1
3 200 701 405 16 3 17
2 401 701 405 16 2 12
1 405 701 405 16 1 16
3 200 704 409 14 3 17
2 401 704 409 14 2 12
1 405 704 409 14 1 16
3 200 706 403 13 3 17
2 401 706 403 13 2 12
1 405 706 403 13 1 16
3 200 709 401 12 3 17
More than 10 rows available. Increase rows selector to view more rows.
yashasvanimarkanday3@gmail.com
yy_dbms_2025
en
Copyright © 1999, 2024, Oracle and/or its affiliates.
Oracle APEX 24.2.8

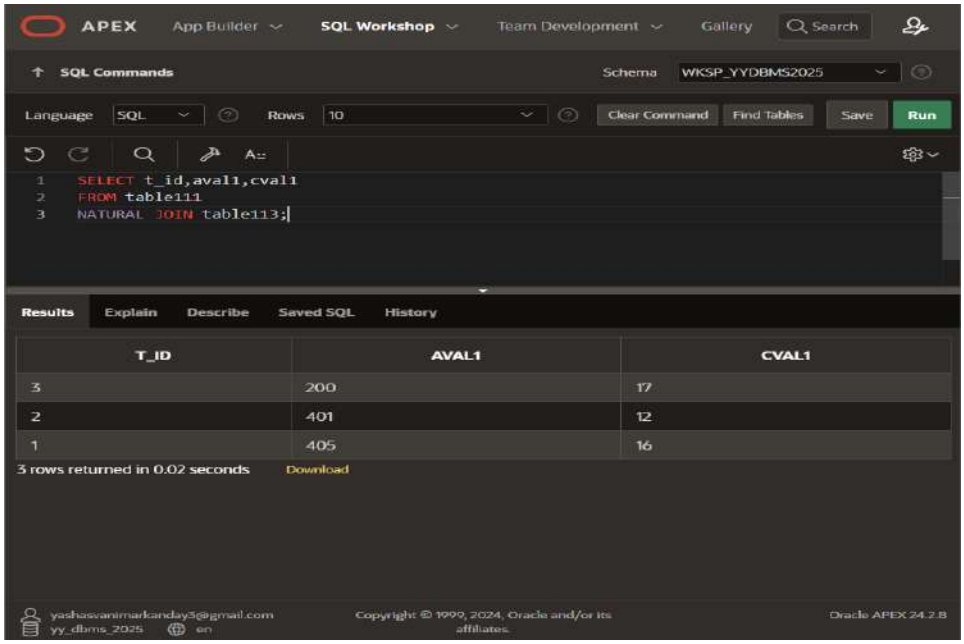
17.)


APEX
App Builder
SQL Workshop
Team Development
Gallery
Search
SQL Commands
Schema
WKSP_YYDBMS2025
Language
SQL
Rows
10
Clear Command
Find Tables
Save
Run
SELECT table111.*,table113.*
FROM table111
CROSS JOIN table113
WHERE table111.t_id=table113.t_id;
Results
Explain
Describe
Saved SQL
History
T_ID
AVAL1
T_ID
CVAL1
3
200
3
17
2
401
2
12
1
405
1
16
3 rows returned in 0.00 seconds
Download
yashaswanimarkanday3@gmail.com
yy_dbms_2025
en
Copyright © 1999, 2024, Oracle and/or its affiliates.
Oracle APEX 24.2.8

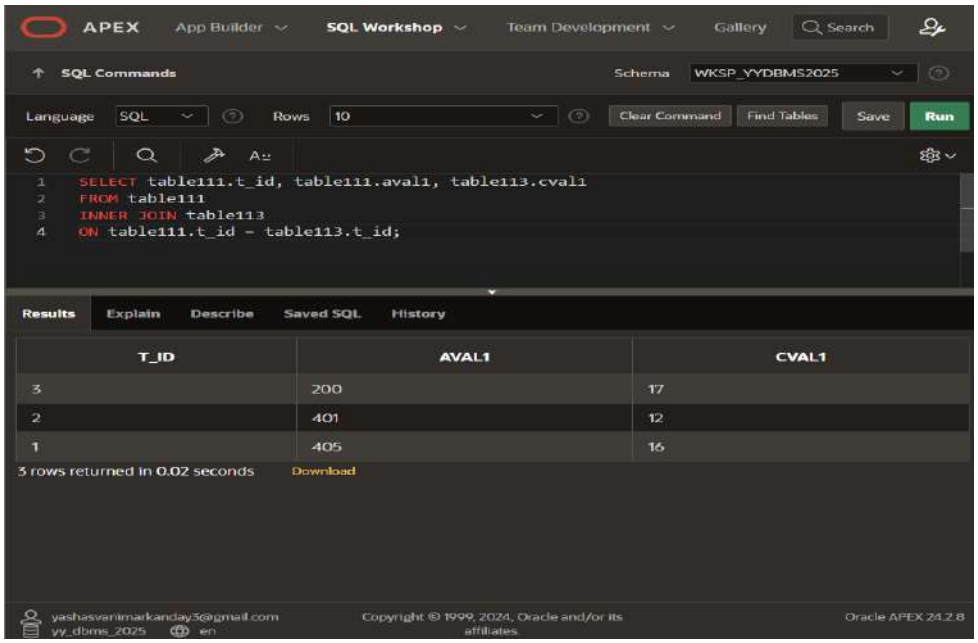
18.)


APEX
App Builder
SQL Workshop
Team Development
Gallery
Search
SQL Commands
Schema
WKSP_YYDBMS2025
Language
SQL
Rows
10
Clear Command
Find Tables
Save
Run
SELECT * FROM table_A
NATURAL JOIN table_B2;
Results
Explain
Describe
Saved SQL
History
A
NAME
B
DESCRIPTION
2
Mary
105
Pen
5
Eve
106
Laptop
3
Alex
104
Notebook
3 rows returned in 0.03 seconds
Download
yashaswanimarkanday3@gmail.com
yy_dbms_2025
en
Copyright © 1999, 2024, Oracle and/or its affiliates.
Oracle APEX 24.2.8

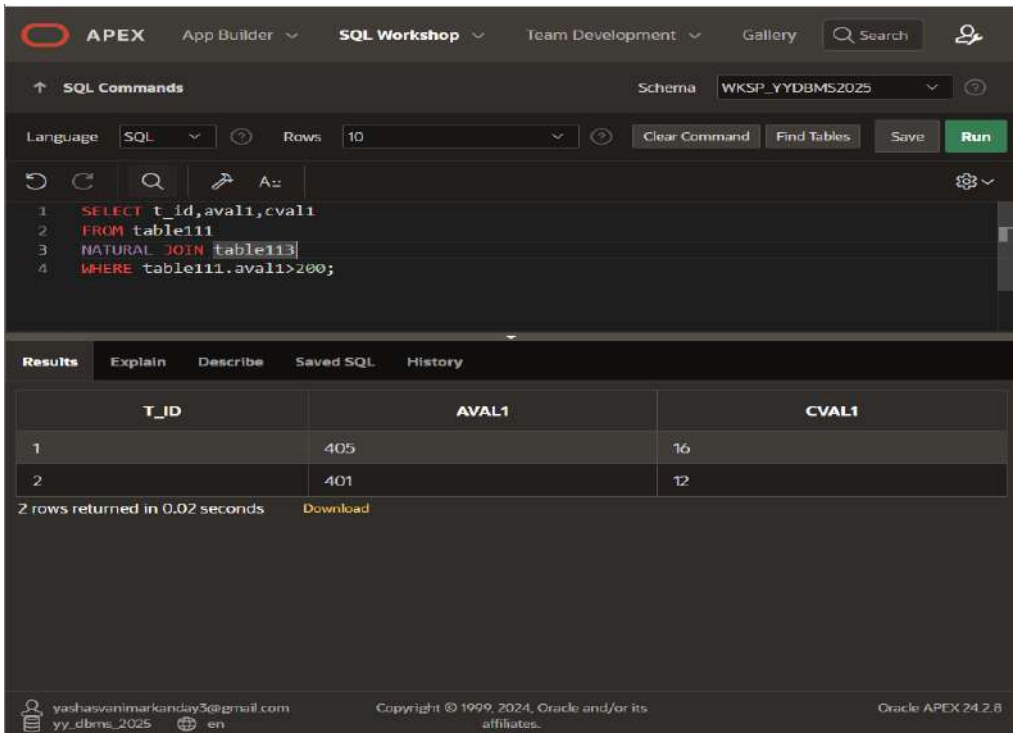
19.)


APEX
App Builder
SQL Workshop
Team Development
Gallery
Search
SQL Commands
Schema
WKSP_YYDBMS2025
Language
SQL
Rows
10
Clear Command
Find Tables
Save
Run
SELECT t_id,aval1,cval1
FROM table111
NATURAL JOIN table113;
Results
Explain
Describe
Saved SQL
History
T_ID
AVAL1
CVAL1
3
200
17
2
401
12
1
405
16
3 rows returned in 0.02 seconds
Download
yashasvanimarkanday3@gmail.com
yy_dbms_2025
en
Copyright © 1999, 2024, Oracle and/or its affiliates.
Oracle APEX 24.2.8

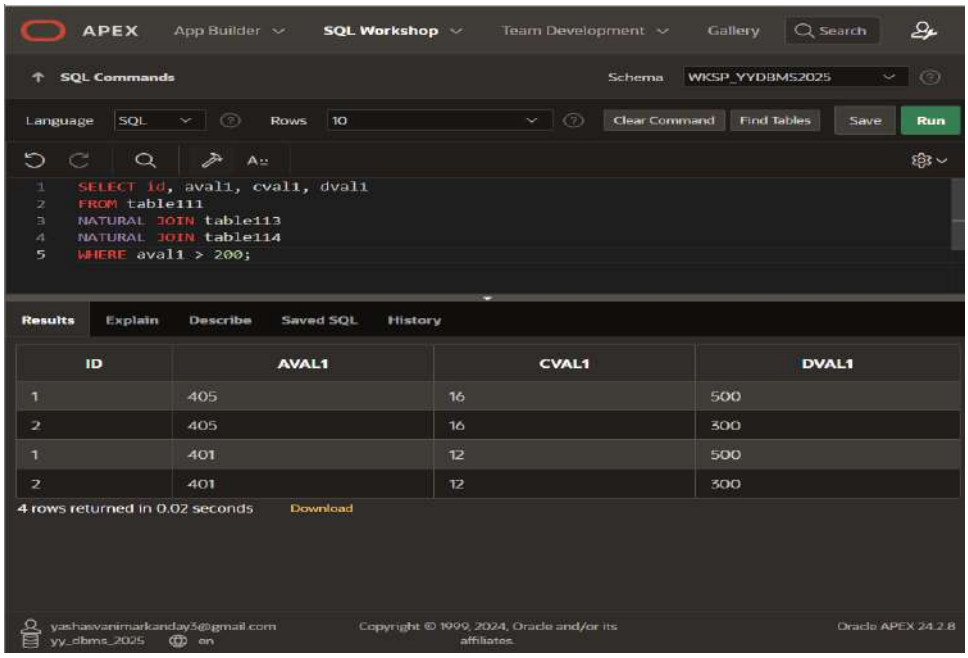
20.)


APEX
App Builder
SQL Workshop
Team Development
Gallery
Search
SQL Commands
Schema
WKSP_YYDBMS2025
Language
SQL
Rows
10
Clear Command
Find Tables
Save
Run
SELECT table111.t_id, table111.aval1, table113.cval1
FROM table111
INNER JOIN table113
ON table111.t_id = table113.t_id;
Results
Explain
Describe
Saved SQL
History
T_ID
AVAL1
CVAL1
3
200
17
2
401
12
1
405
16
3 rows returned in 0.02 seconds
Download
yashasvanimarkanday3@gmail.com
yy_dbms_2025
en
Copyright © 1999, 2024, Oracle and/or its affiliates.
Oracle APEX 24.2.8

21).)


APEX
App Builder
SQL Workshop
Team Development
Gallery
Search
SQL Commands
Schema
WKSP_YYDBMS2025
Language
SQL
Rows
10
Clear Command
Find Tables
Save
Run
1 SELECT t_id,aval1,cval1
2 FROM table111
3 NATURAL JOIN table113
4 WHERE table111.aval1>200;
Results
Explain
Describe
Saved SQL
History
T_ID
AVAL1
CVAL1
1
405
16
2
401
12
2 rows returned in 0.02 seconds
Download
yashasvanimarkanday3@gmail.com
yy_dbms_2025
en
Copyright © 1999, 2024, Oracle and/or its affiliates.
Oracle APEX 24.2.8

22.)


APEX
App Builder
SQL Workshop
Team Development
Gallery
Search
SQL Commands
Schema
WKSP_YYDBMS2025
Language
SQL
Rows
10
Clear Command
Find Tables
Save
Run
1 SELECT id, aval1, cval1, dval1
2 FROM table111
3 NATURAL JOIN table113
4 NATURAL JOIN table114
5 WHERE aval1 > 200;
Results
Explain
Describe
Saved SQL
History
ID
AVAL1
CVAL1
DVAL1
1
405
16
500
2
405
16
300
1
401
12
500
2
401
12
300
4 rows returned in 0.02 seconds
Download
yashasvanimarkanday3@gmail.com
yy_dbms_2025
en
Copyright © 1999, 2024, Oracle and/or its affiliates.
Oracle APEX 24.2.8

UNIVERSITY OF PETROLEUM AND ENERGY STUDIES

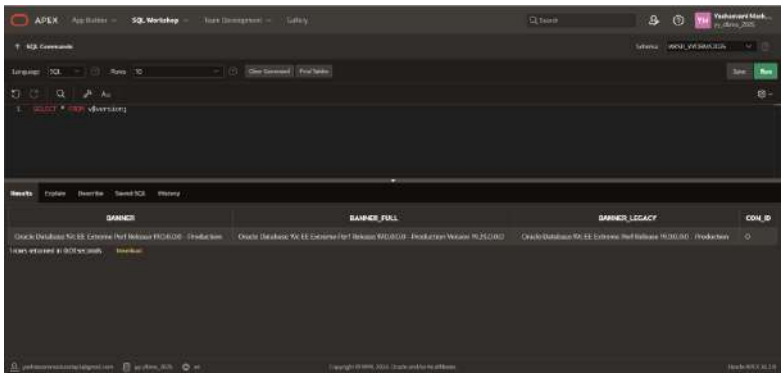
Program Name: B. Tech CS **Semester: III**

Name: Yashasvani Markanday **SAP ID: 590011386**

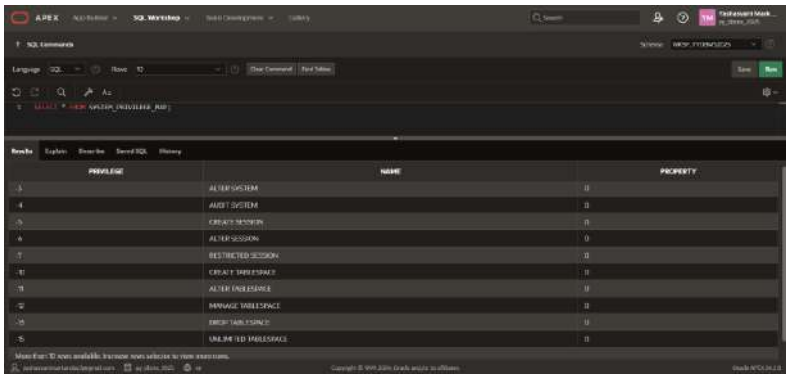
Batch: B2 **DBMS**

LAB-8

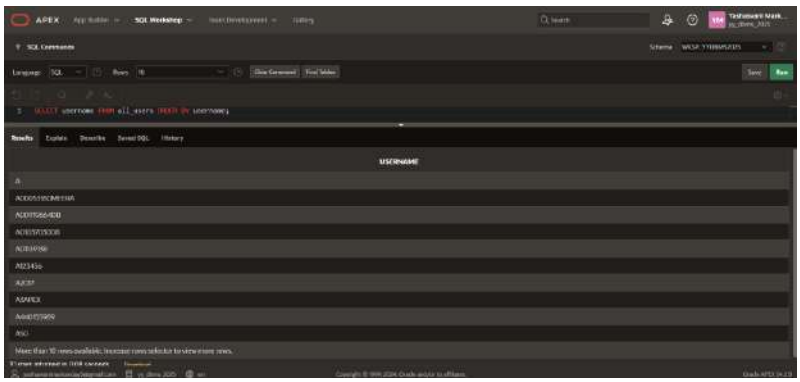
1.



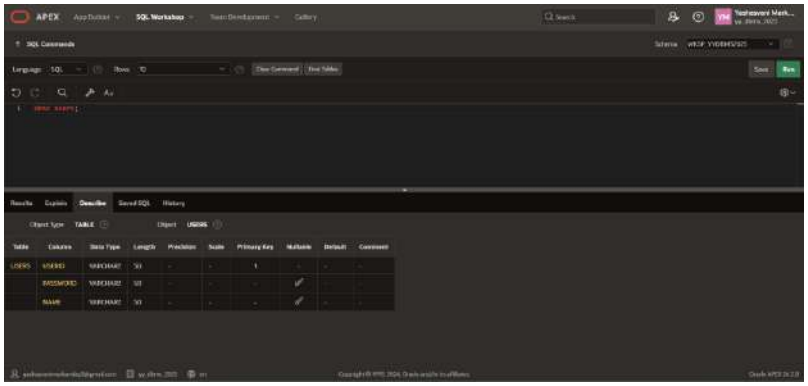
2.



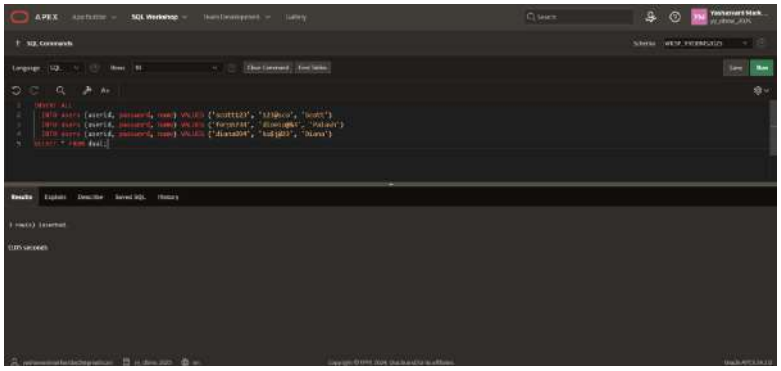
3.



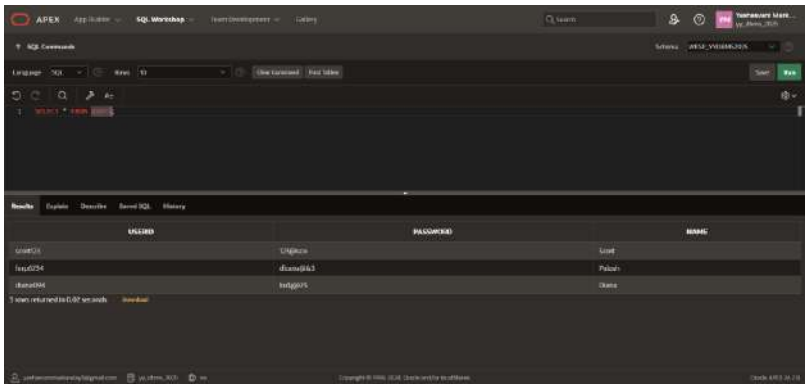
4.



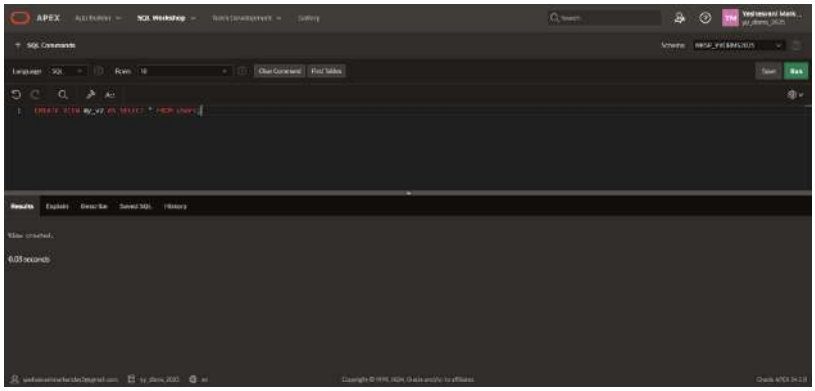
5.



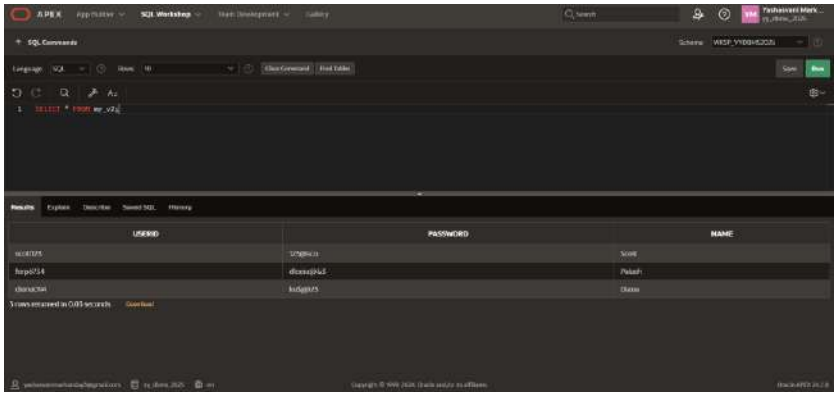
6.



7.

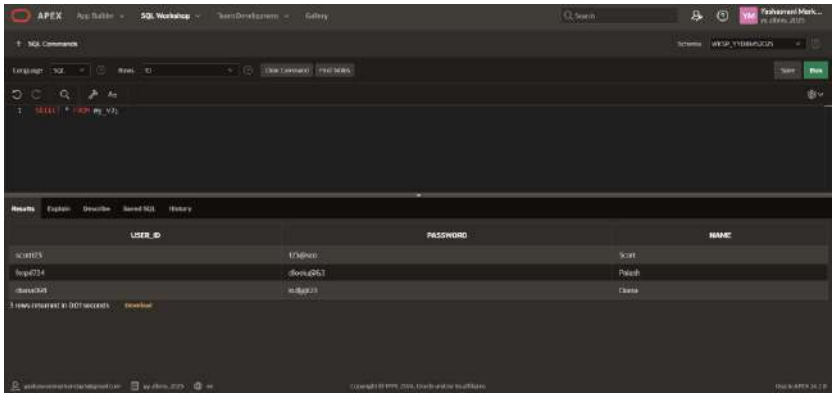


8.

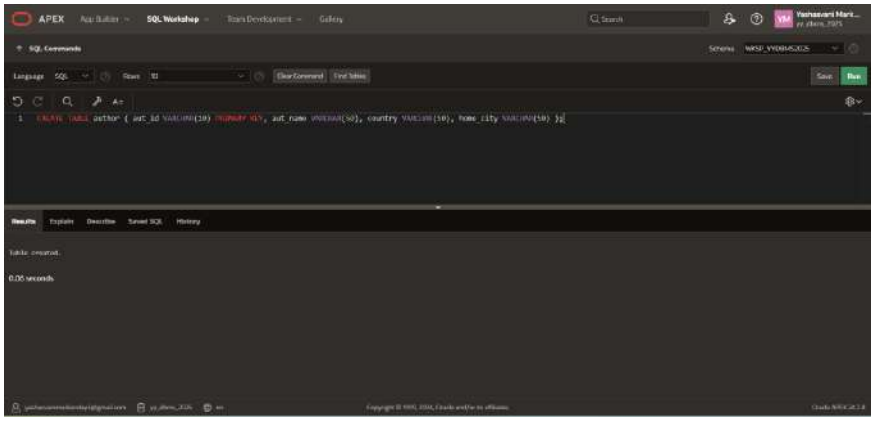


9.

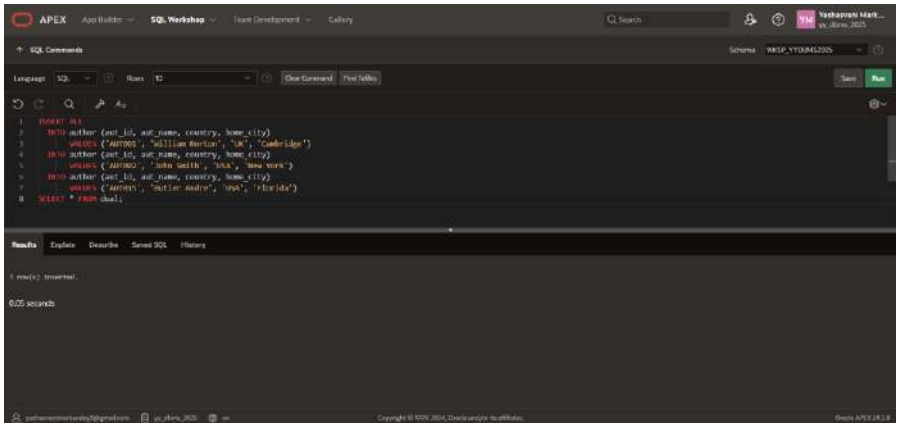
10.



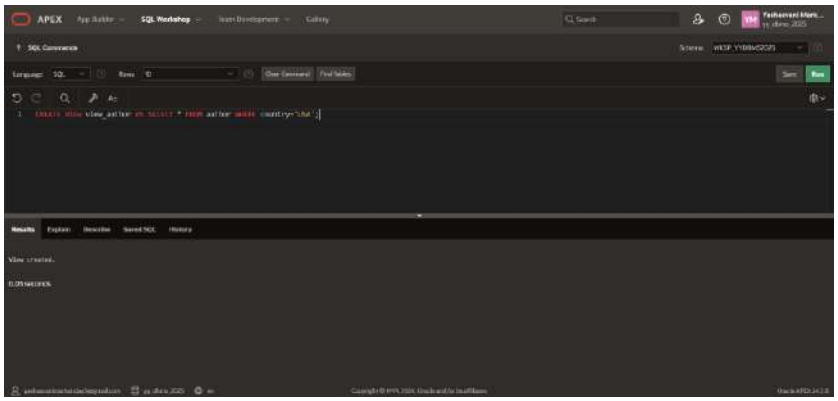
11.



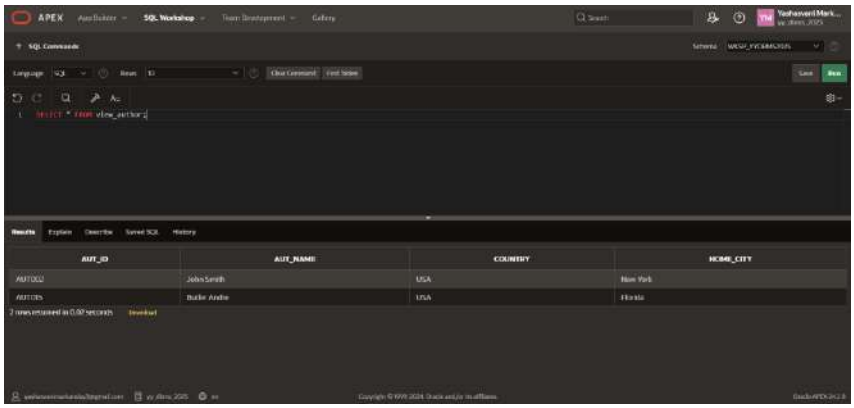
12.


APEX
SQL Workshop
SQL Commands
INSERT ALL
INTO author (aut_id, aut_name, country, home_city)
VALUES ('AUTH03', 'William Morton', 'UK', 'Cambridge')
INTO author (aut_id, aut_name, country, home_city)
VALUES ('AUTH02', 'John Smith', 'USA', 'New York')
INTO author (aut_id, aut_name, country, home_city)
VALUES ('AUTH01', 'Butler Andre', 'USA', 'Florida')
SELECT * FROM dual;
Results
0.05 seconds

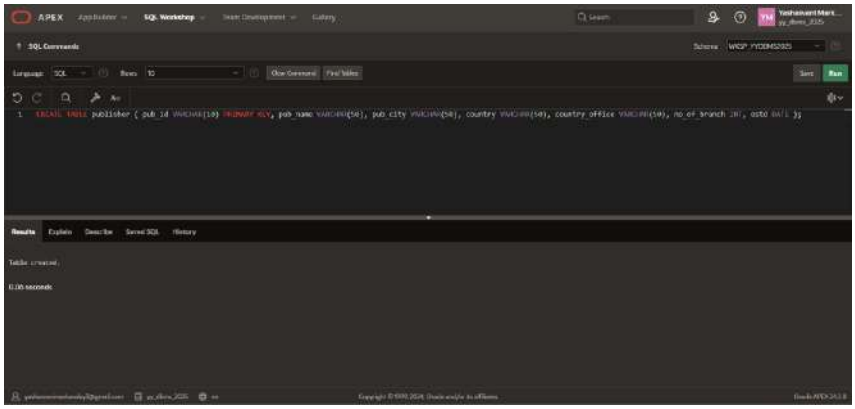
13.


APEX
SQL Commands
CREATE VIEW view_author AS SELECT * FROM author WHERE country='USA';
Results
View created.

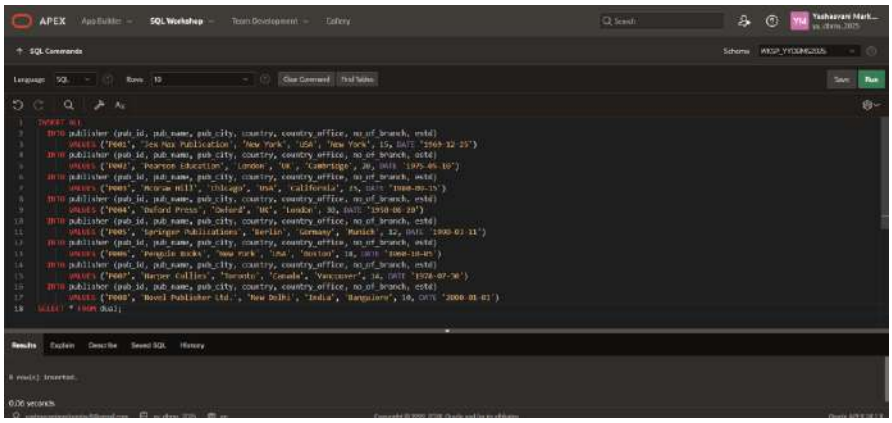
14.


APEX
SQL Commands
SELECT * FROM view_author;
Results
AUT_ID
AUT_NAME
COUNTRY
HOME_CITY
AUTH02
John Smith
USA
New York
AUTH01
Butler Andre
USA
Florida

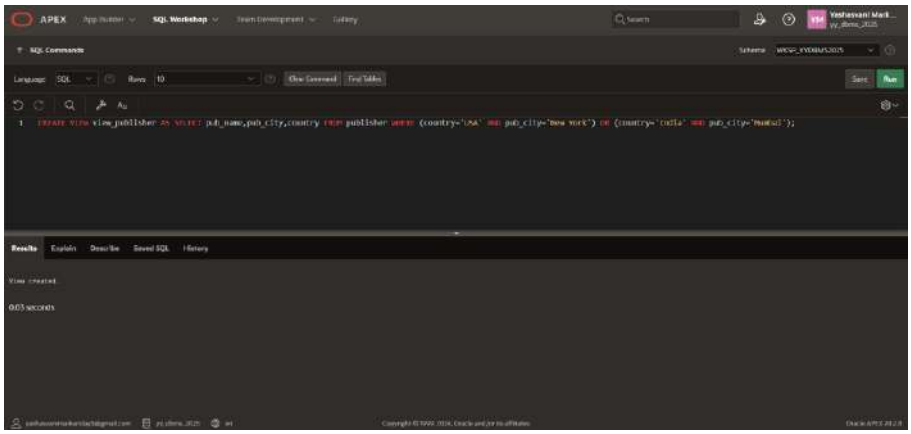
15.



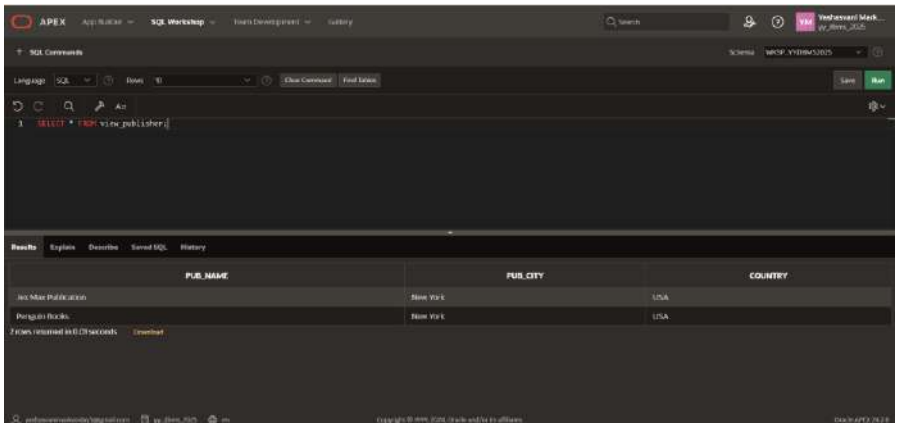
16.



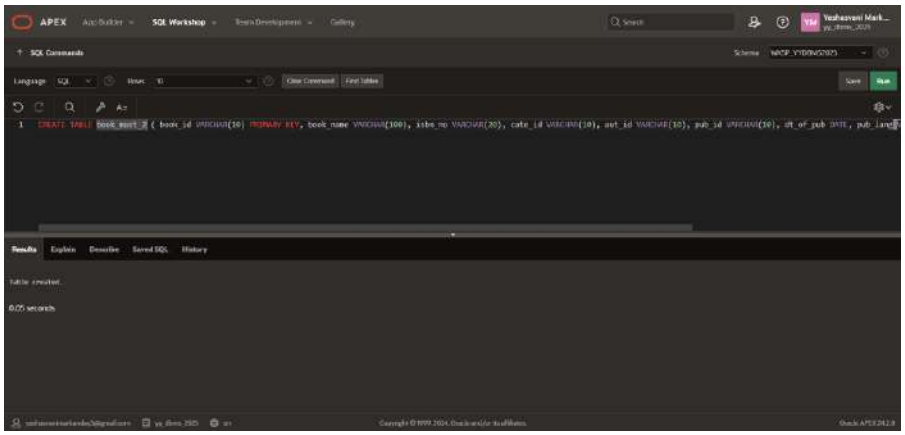
17.



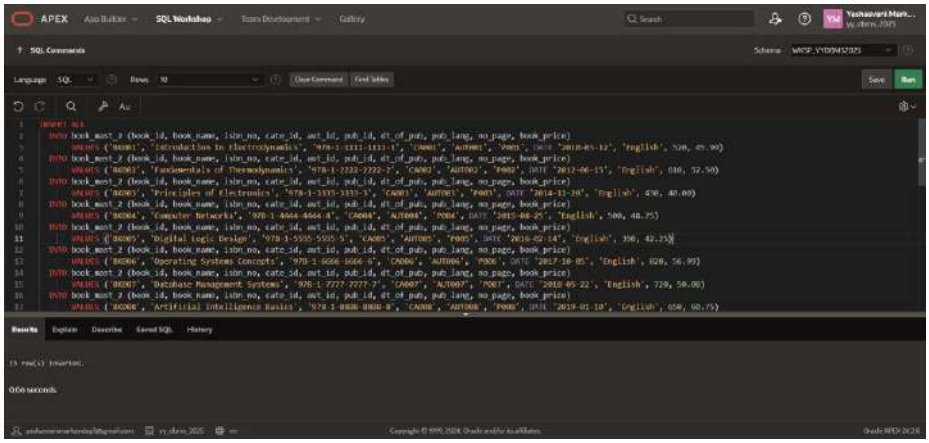
18.


APEX
SQL Workshop
SELECT * FROM view_publisher;
PUB_NAME
PUB_CITY
COUNTRY
Jax Max Publication
New York
USA
Penguin Books
New York
USA

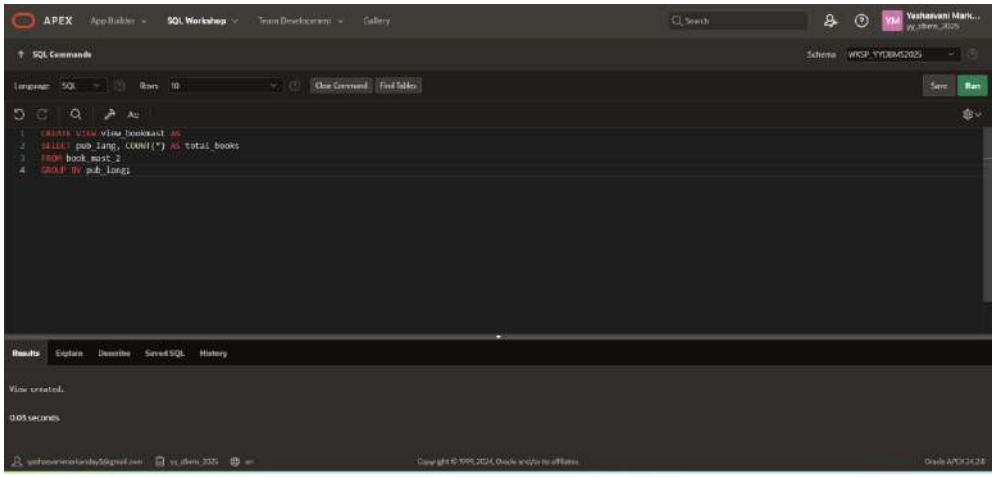
19.


APEX
SQL Workshop
CREATE TABLE book_mast_2 (book_id VARCHAR(10) PRIMARY KEY, book_name VARCHAR(100), isbn_no VARCHAR(20), cate_id VARCHAR(10), aut_id VARCHAR(10), pub_id VARCHAR(10), dt_of_pub DATE, pub_lang
Table created.
0.05 seconds

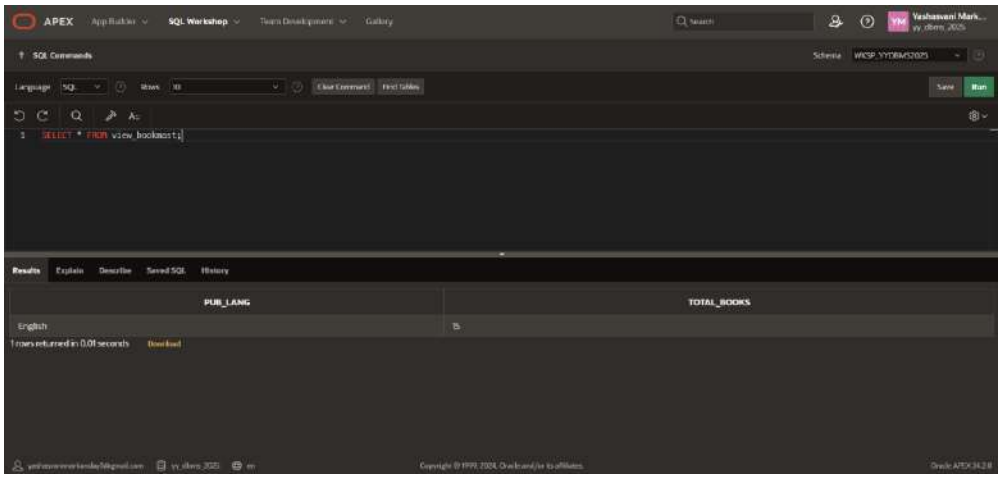
20.


APEX
SQL Workshop
INSERT ALL
INTO book_mast_2 (book_id, book_name, isbn_no, cate_id, aut_id, pub_id, dt_of_pub, pub_lang, no_page, book_price)
VALUES ('BK0007', 'Database Management Systems', '978-1-7777-7777-7', 'CA009', 'AUT007', 'P007', DATE '2018-05-22', 'English', 720, 50.00)
INTO book_mast_2 (book_id, book_name, isbn_no, cate_id, aut_id, pub_id, dt_of_pub, pub_lang, no_page, book_price)
VALUES ('BK008', 'Artificial Intelligence Basics', '978-1-8888-8888-8', 'CA008', 'AUT008', 'P008', DATE '2019-01-10', 'English', 650, 60.75)
SELECT 1 FROM DUAL;
3 row(s) inserted.
0.016 seconds

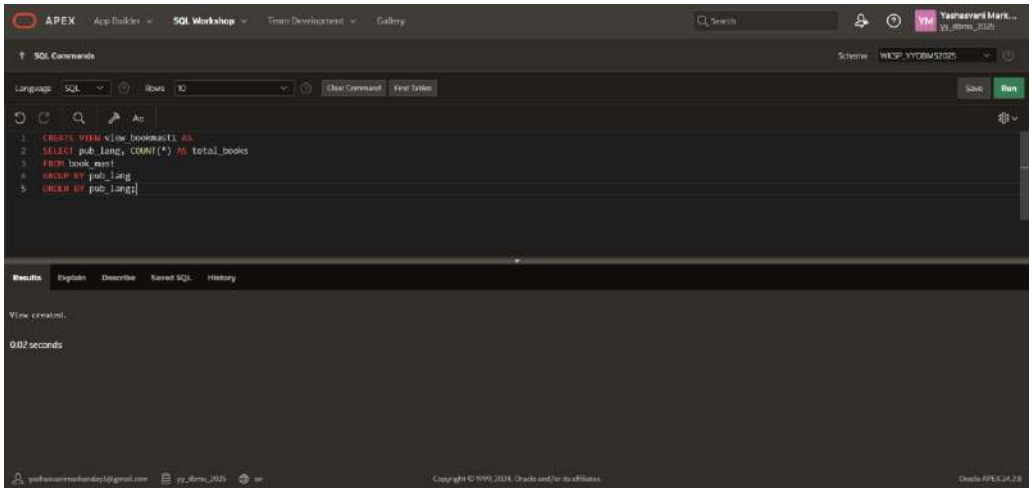
21.


APEX
SQL Workshop
SQL Commands
CREATE VIEW view_bookmast AS
SELECT pub_lang, COUNT(*) AS total_books
FROM book_mast_2
GROUP BY pub_lang;
Results
Explain
Describe
Saved SQL
History
View created.
0.05 seconds

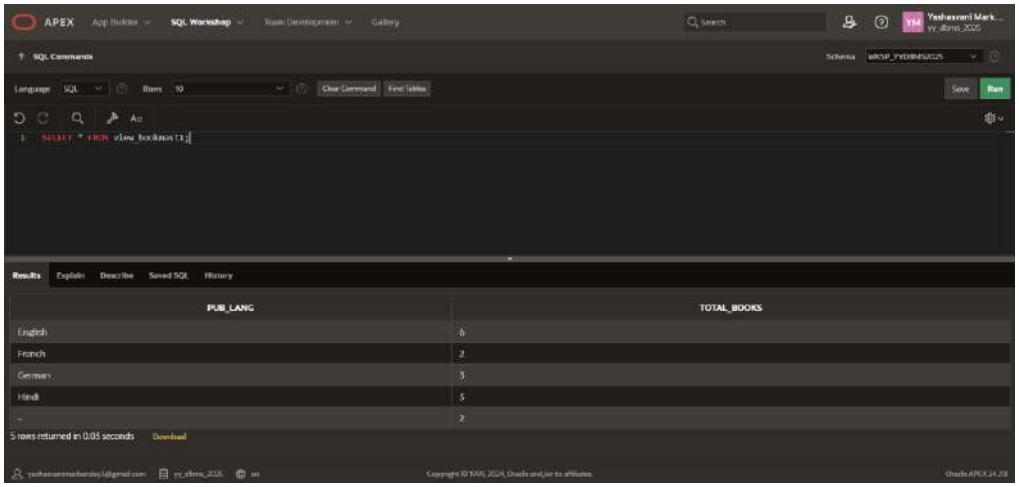
22.


APEX
SQL Workshop
SQL Commands
SELECT * FROM view_bookmast;
Results
Explain
Describe
Saved SQL
History
PUB_LANG
TOTAL_BOOKS
English
5
1 rows returned in 0.01 seconds
Download

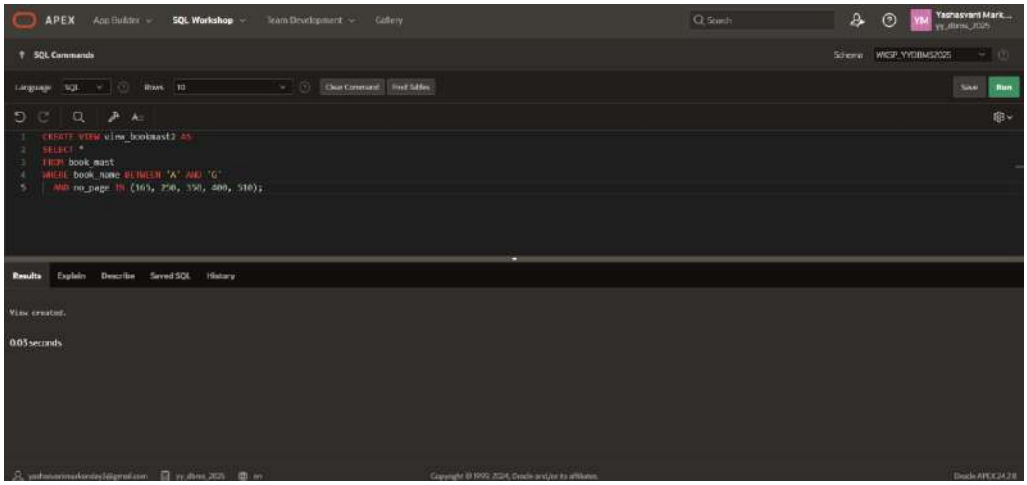
23.


APEX
SQL Workshop
SQL Commands
CREATE VIEW view_bookmast1 AS
SELECT pub_lang, COUNT(*) AS total_books
FROM book_mast
GROUP BY pub_lang
ORDER BY pub_lang;
Results
Explain
Describe
Saved SQL
History
View created.
0.02 seconds

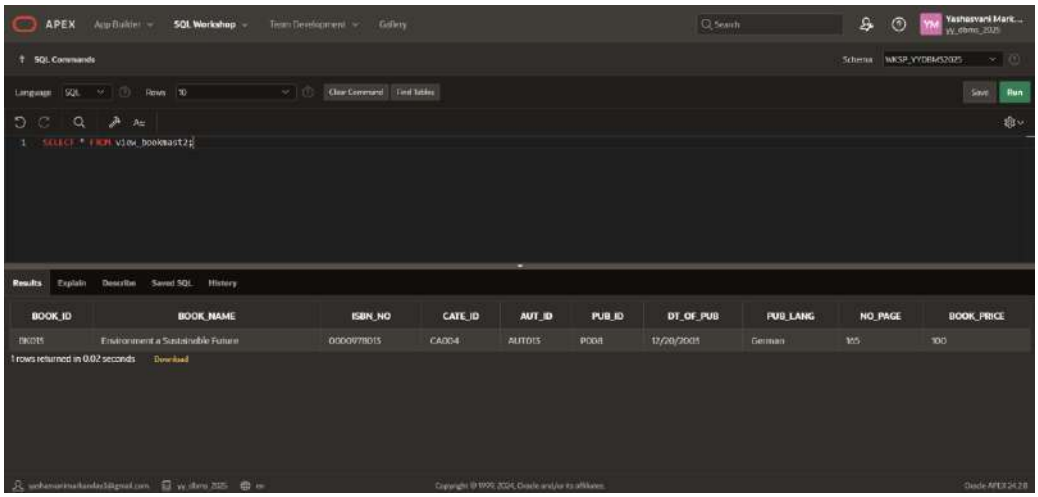
24.


APEX
SQL Workshop
SELECT * FROM view_bookmast1;
Results
PUB_LANG
TOTAL_BOOKS
English
6
French
2
German
3
Hindi
5
-
2
5 rows returned in 0.05 seconds

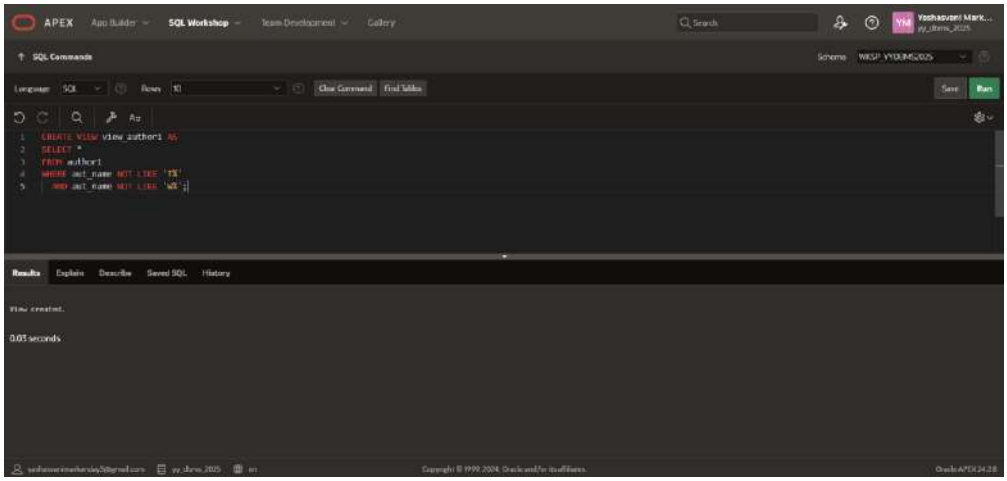
25.


APEX
SQL Workshop
CREATE VIEW view_bookmast2 AS
SELECT *
FROM book_mast
WHERE book_name BETWEEN 'A' AND 'G'
AND no_page IN (165, 250, 350, 400, 510);
Results
View created.
0.03 seconds

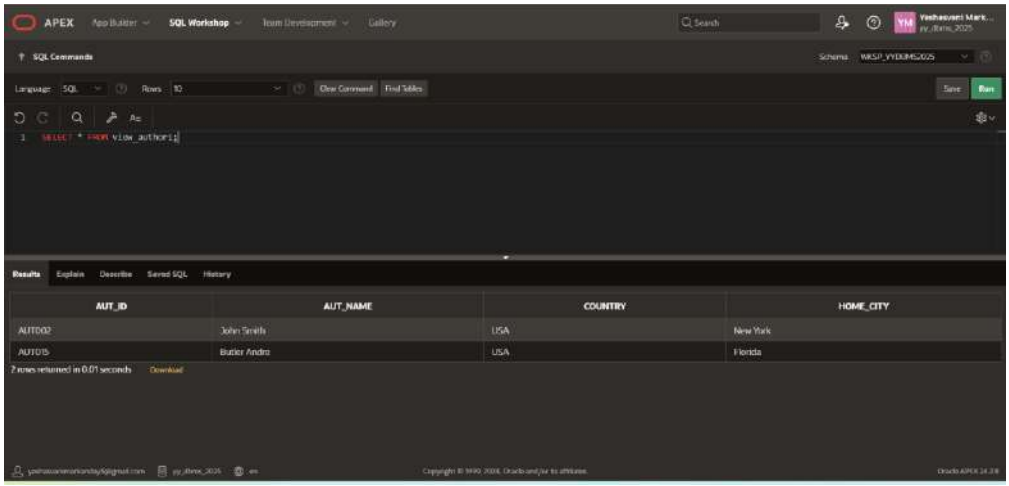
26.


APEX
SQL Workshop
SELECT * FROM view_bookmast2;
Results
BOOK_ID
BOOK_NAME
ISBN_NO
CATE_ID
AUT_ID
PUB_ID
DT_OF_PUB
PUB_LANG
NO_PAGE
BOOK_PRICE
BK015
Environment a Sustainable Future
0000977815
CA004
AUT015
P008
12/20/2005
German
165
100
1 rows returned in 0.02 seconds

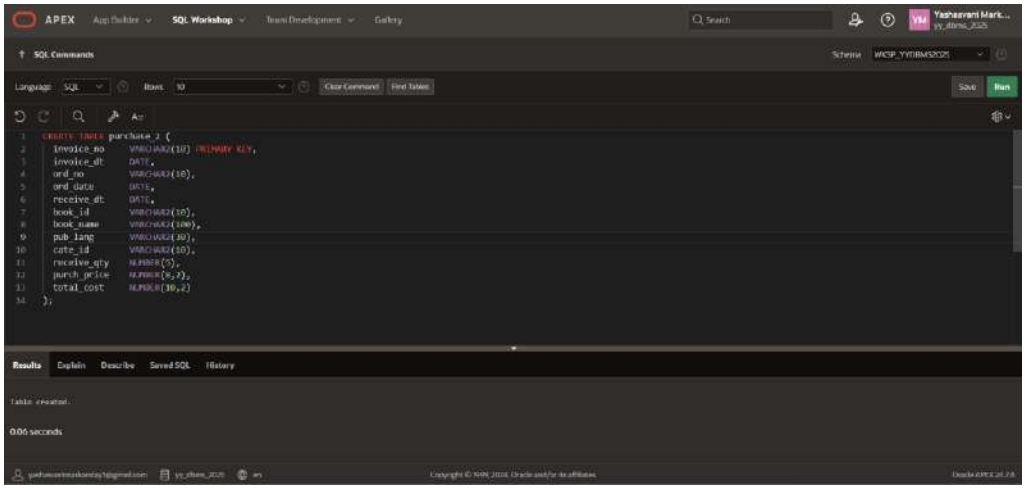
27.


APEX
SQL Workshop
SQL Commands
CREATE VIEW view_author1 AS
SELECT *
FROM author1
WHERE aut_name NOT LIKE 'T%'
AND aut_name NOT LIKE 'W%';
Results
Explain
Describe
Saved SQL
History
View created.
0.03 seconds

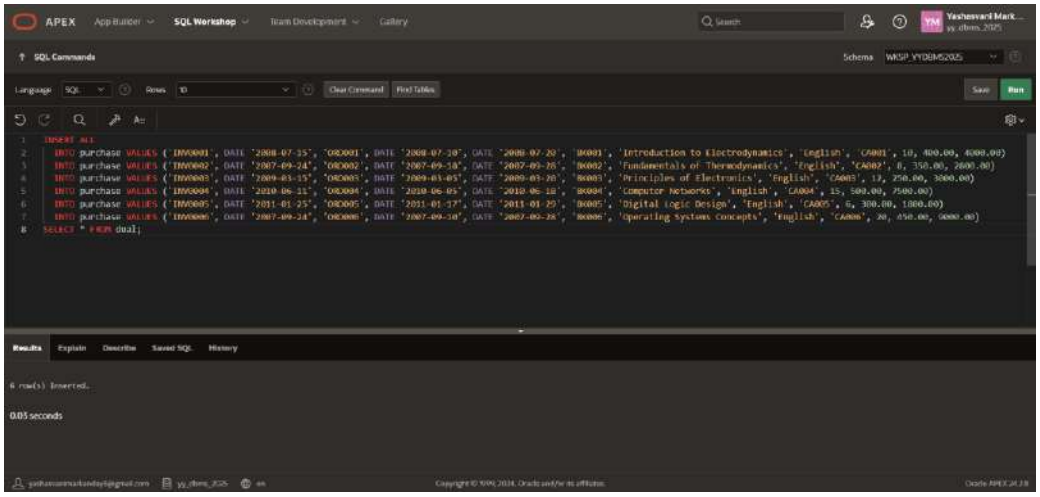
28.


APEX
SQL Workshop
SQL Commands
SELECT * FROM view_author1;
Results
Explain
Describe
Saved SQL
History
AUT_ID
AUT_NAME
COUNTRY
HOME_CITY
AUT002
John Smith
USA
New York
AUT015
Butler Andru
USA
Florida
2 rows returned in 0.01 seconds
Download

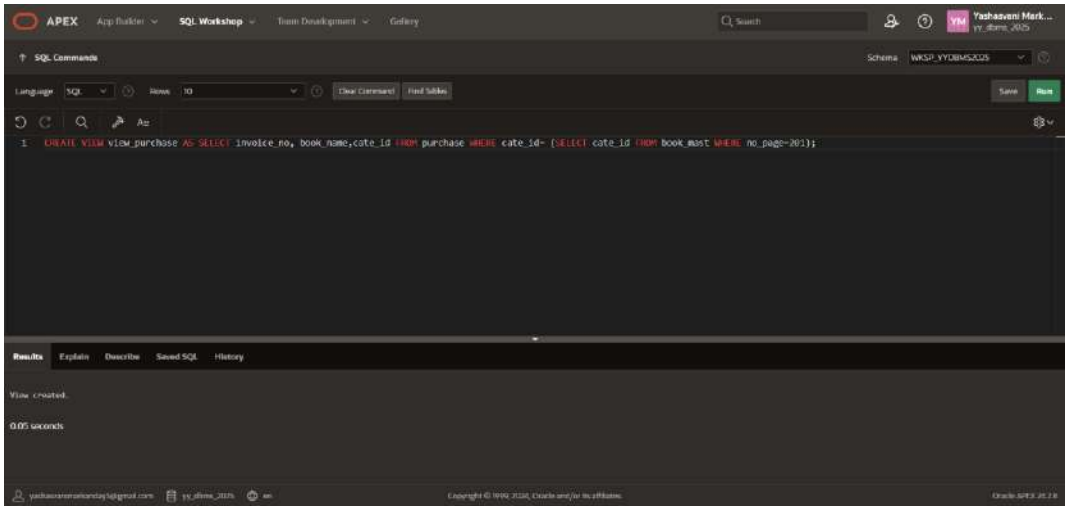
29.


APEX
SQL Workshop
SQL Commands
CREATE TABLE purchase_2 (
invoice_no VARCHAR2(10) PRIMARY KEY,
invoice_dt DATE,
ord_no VARCHAR2(10),
ord_date DATE,
receive_dt DATE,
book_id VARCHAR2(10),
book_name VARCHAR2(100),
pub_lang VARCHAR2(30),
cate_id VARCHAR2(10),
receive_qty NUMBER(5),
purch_price NUMBER(8,2),
total_cost NUMBER(10,2)
);
Results
Explain
Describe
Saved SQL
History
Table created.
0.06 seconds

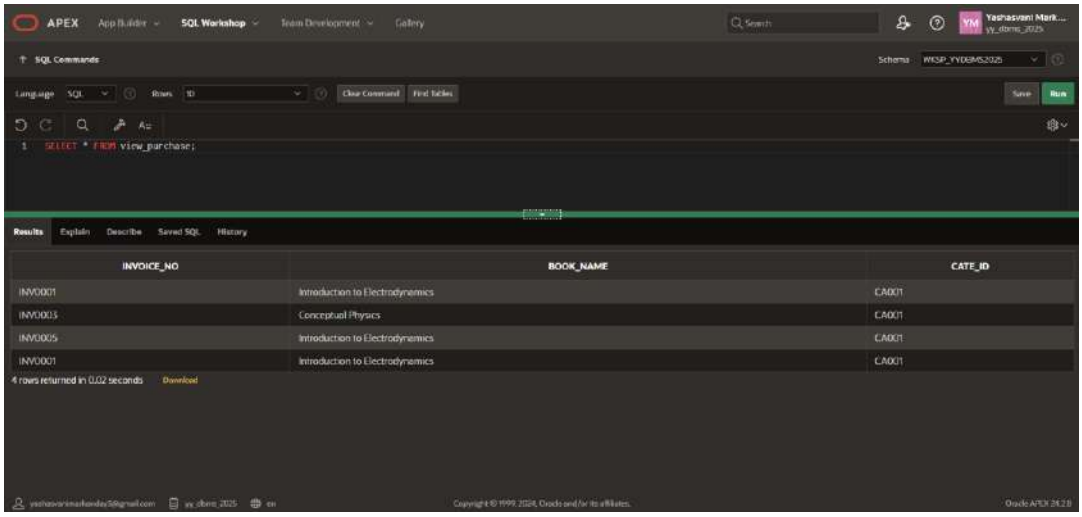
30.


APEX
SQL Workshop
SQL Commands
INSERT ALL
INTO purchase VALUES ('INV0001', DATE '2008-07-15', 'ORD001', DATE '2008-07-10', DATE '2008-07-20', 'BK001', 'Introduction to Electrodynamics', 'English', 'CA001', 10, 400.00, 4000.00)
INTO purchase VALUES ('INV0002', DATE '2007-09-24', 'ORD002', DATE '2007-09-18', DATE '2007-09-28', 'BK002', 'Fundamentals of Thermodynamics', 'English', 'CA002', 8, 350.00, 2800.00)
INTO purchase VALUES ('INV0003', DATE '2009-03-15', 'ORD003', DATE '2009-03-05', DATE '2009-03-20', 'BK003', 'Principles of Electronics', 'English', 'CA003', 12, 250.00, 3000.00)
INTO purchase VALUES ('INV0004', DATE '2010-06-11', 'ORD004', DATE '2010-06-05', DATE '2010-06-18', 'BK004', 'Computer Networks', 'English', 'CA004', 15, 500.00, 7500.00)
INTO purchase VALUES ('INV0005', DATE '2011-01-25', 'ORD005', DATE '2011-01-17', DATE '2011-01-29', 'BK005', 'Digital Logic Design', 'English', 'CA005', 6, 300.00, 1800.00)
INTO purchase VALUES ('INV0006', DATE '2007-09-24', 'ORD006', DATE '2007-09-18', DATE '2007-09-28', 'BK006', 'Operating System Concepts', 'English', 'CA006', 20, 450.00, 9000.00)
SELECT * FROM dual;
Results
Explain
Describe
Saved SQL
History
6 row(s) inserted.
0.05 seconds

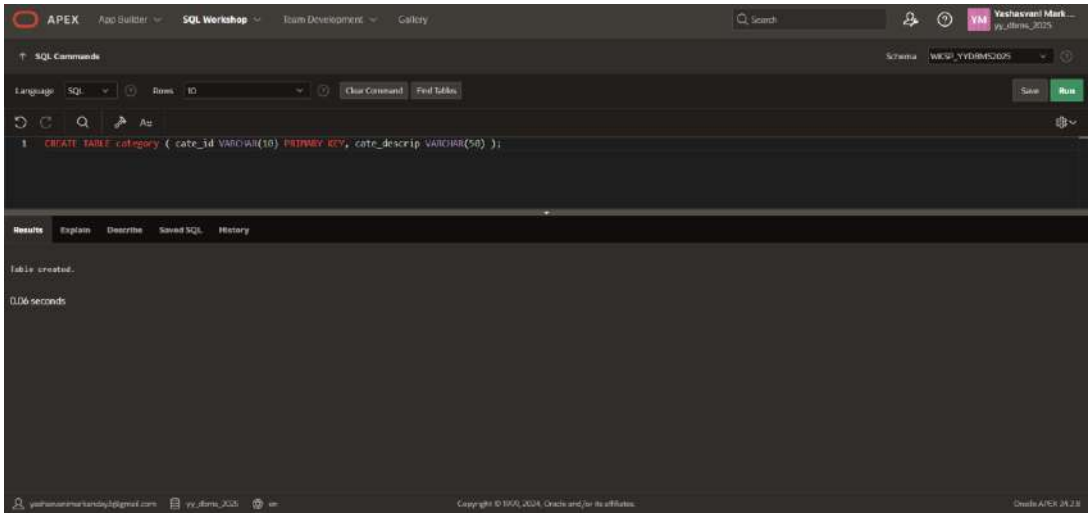
31.


APEX
SQL Workshop
SQL Commands
CREATE VIEW view_purchase AS SELECT invoice_no, book_name,cate_id FROM purchase WHERE cate_id= (SELECT cate_id FROM book_mast WHERE no_page=201);
Results
Explain
Describe
Saved SQL
History
View created.
0.05 seconds

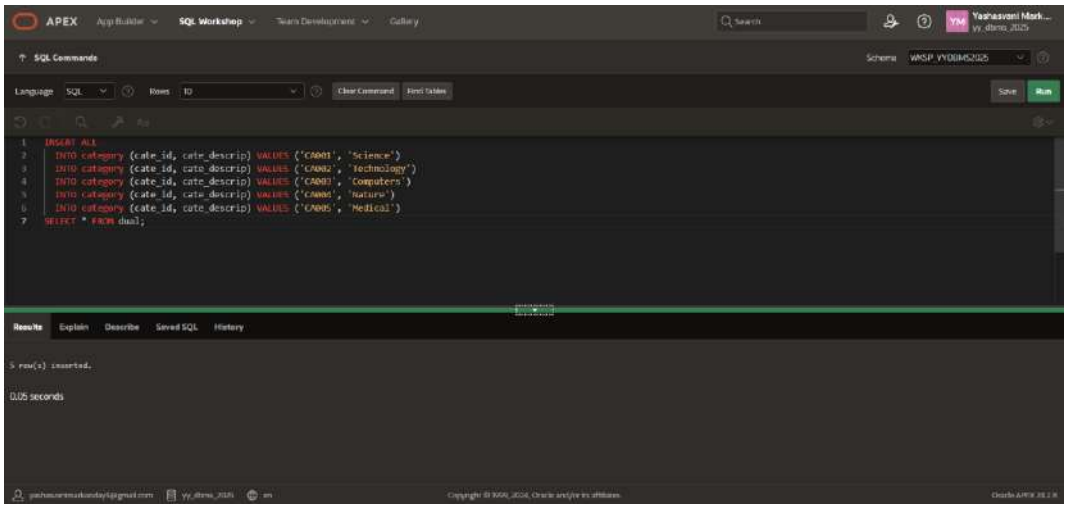
32.


APEX
SQL Workshop
SQL Commands
SELECT * FROM view_purchase;
Results
Explain
Describe
Saved SQL
History
INVOICE_NO
BOOK_NAME
CATE_ID
INV0001
Introduction to Electrodynamics
CA001
INV0003
Conceptual Physics
CA001
INV0005
Introduction to Electrodynamics
CA001
INV0001
Introduction to Electrodynamics
CA001
4 rows returned in 0.02 seconds
Download

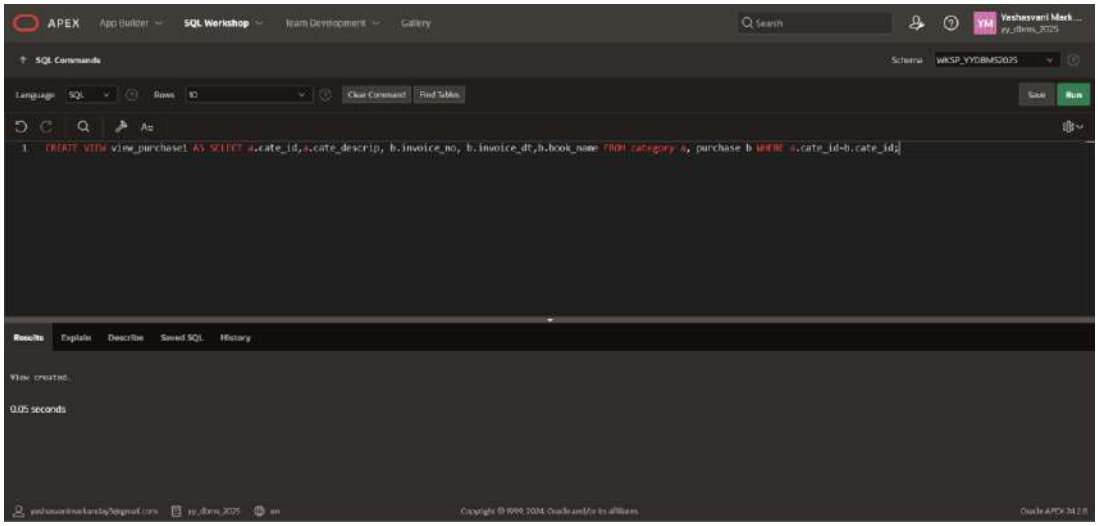
33.


APEX
App Builder
SQL Workshop
Team Development
Gallery
SQL Commands
Schema
Language
SQL
Rows
10
Clear Command
Find Tables
Save
Run
1 CREATE TABLE category (cate_id VARCHAR(10) PRIMARY KEY, cate_descrip VARCHAR(50));
Results
Explain
Describe
Saved SQL
History
Table created.
0.06 seconds

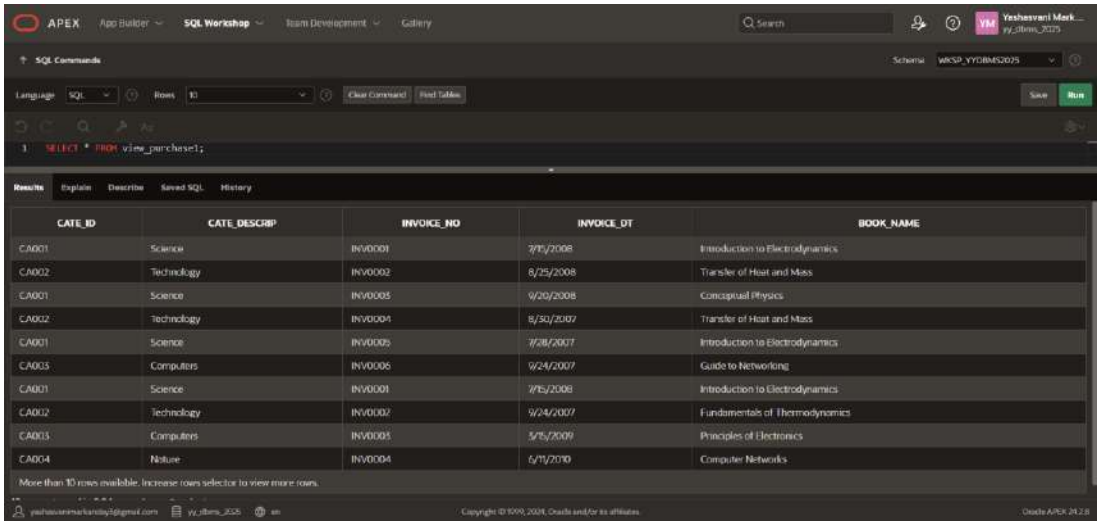
34.


APEX
App Builder
SQL Workshop
Team Development
Gallery
SQL Commands
Schema
WKSP_YYDBMS2025
Language
SQL
Rows
10
Clear Command
Find Tables
Save
Run
1 INSERT ALL
2 INTO category (cate_id, cate_descrip) VALUES ('CA001', 'Science')
3 INTO category (cate_id, cate_descrip) VALUES ('CA002', 'Technology')
4 INTO category (cate_id, cate_descrip) VALUES ('CA003', 'Computers')
5 INTO category (cate_id, cate_descrip) VALUES ('CA004', 'Nature')
6 INTO category (cate_id, cate_descrip) VALUES ('CA005', 'Medical')
7 SELECT * FROM dual;
Results
Explain
Describe
Saved SQL
History
5 row(s) inserted.
0.05 seconds

35.


APEX
App Builder
SQL Workshop
Team Development
Gallery
SQL Commands
Schema
WKSP_YYDBMS2025
Language
SQL
Rows
10
Clear Command
Find Tables
Save
Run
1 CREATE VIEW view_purchase1 AS SELECT a.cate_id,a.cate_descrip, b.invoice_no, b.invoice_dt,b.book_name FROM category a, purchase b WHERE a.cate_id=b.cate_id;
Results
Explain
Describe
Saved SQL
History
View created.
0.05 seconds

36.

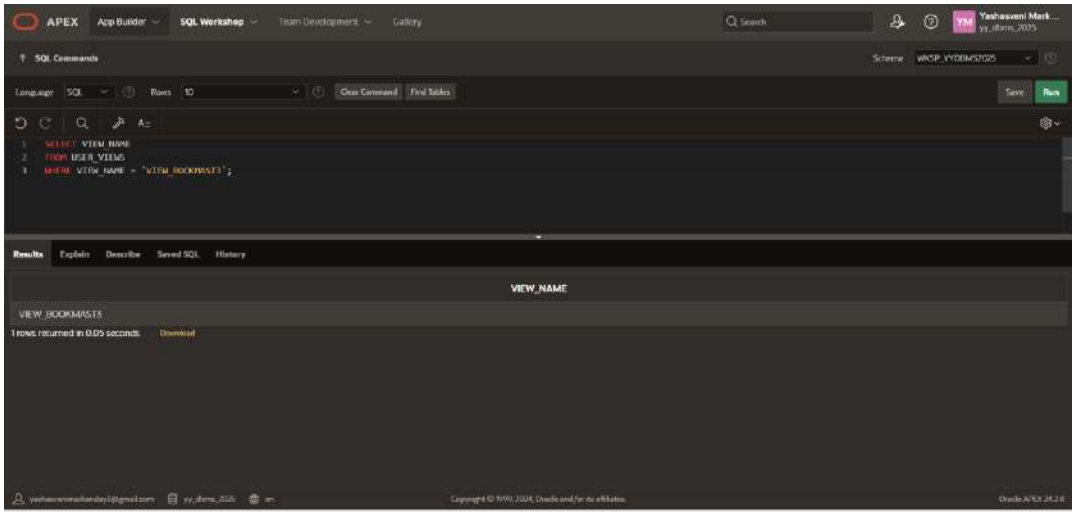


```
SELECT * FROM view_purchase1;
```

CATE_ID	CATE_DESCRIP	INVOICE_NO	INVOICE_DT	BOOK_NAME
CA001	Science	INV0001	7/15/2008	Introduction to Electrodynamics
CA002	Technology	INV0002	8/25/2008	Transfer of Heat and Mass
CA001	Science	INV0003	9/20/2008	Conceptual Physics
CA002	Technology	INV0004	8/30/2007	Transfer of Heat and Mass
CA001	Science	INV0005	7/28/2007	Introduction to Electrodynamics
CA003	Computers	INV0006	9/24/2007	Guide to Networking
CA001	Science	INV0001	7/15/2008	Introduction to Electrodynamics
CA002	Technology	INV0002	9/24/2007	Fundamentals of Thermodynamics
CA003	Computers	INV0003	5/15/2009	Principles of Electronics
CA004	Nature	INV0004	6/11/2010	Computer Networks

More than 10 rows available. Increase rows selector to view more rows.

37.

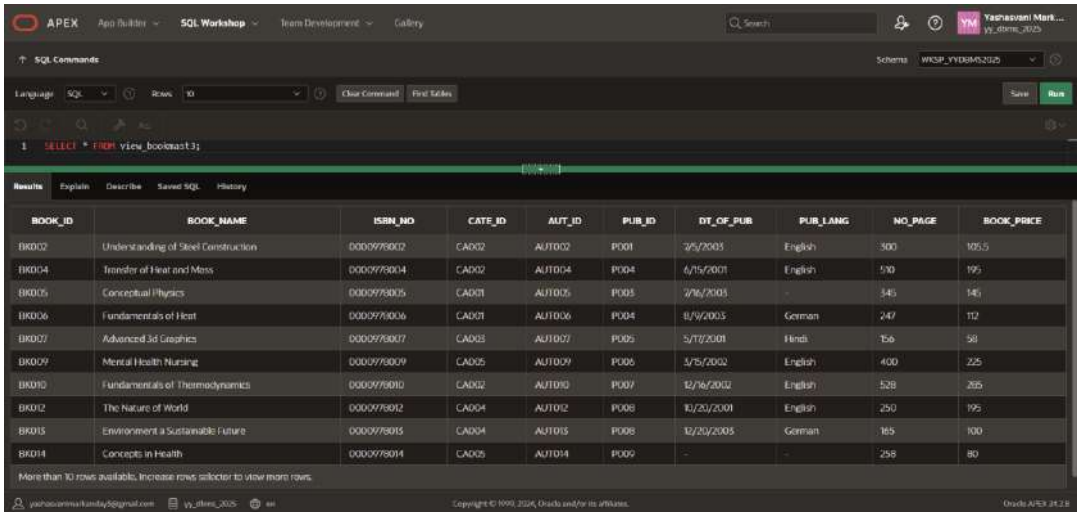


```
SELECT VIEW_NAME
FROM USER_VIEWS
WHERE VIEW_NAME = 'VIEW_BOOKMAST3';
```

VIEW_NAME
VIEW_BOOKMAST3

1 rows returned in 0.05 seconds Download

38.



```
SELECT * FROM view_bookmast3;
```

BOOK_ID	BOOK_NAME	ISBN_NO	CATE_ID	AUT_ID	PUB_ID	DT_OF_PUB	PUB_LANG	NO_PAGE	BOOK_PRICE
BK002	Understanding of Steel Construction	0009978002	CA002	AUT002	P001	3/5/2003	English	300	105.5
BK004	Transfer of Heat and Mass	0009978004	CA002	AUT004	P004	6/15/2001	English	530	195
BK005	Conceptual Physics	0009978005	CA001	AUT005	P003	7/16/2003	-	345	145
BK006	Fundamentals of Heat	0009978006	CA001	AUT006	P004	8/9/2003	German	247	112
BK007	Advanced 3d Graphics	0009978007	CA003	AUT007	P005	5/17/2001	Hindi	156	58
BK009	Mental Health Nursing	0009978009	CA005	AUT009	P006	5/15/2002	English	400	225
BK010	Fundamentals of Thermodynamics	0009978010	CA002	AUT010	P007	12/16/2002	English	528	265
BK012	The Nature of World	0009978012	CA004	AUT012	P008	10/20/2001	English	250	195
BK013	Environment a Sustainable Future	0009978013	CA004	AUT013	P008	12/20/2003	German	165	100
BK014	Concepts in Health	0009978014	CA005	AUT014	P009	-	-	256	80

More than 10 rows available. Increase rows selector to view more rows.

OUTPUTS:

1.

```
-- yashasvani
CREATE DATABASE IF NOT EXISTS ADBMS_LAB;
USE ADBMS_LAB;
DROP TABLE IF EXISTS employees;
CREATE TABLE employees (
    id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(100),
    department VARCHAR(100),
    salary DECIMAL(10, 2)
);
DESC employees;
```

Field	Type	Null	Key	Default	Extra
id	int	NO	PRI	NULL	auto_increment
name	varchar(100)	YES		NULL	
department	varchar(100)	YES		NULL	
salary	decimal(10,2)	YES		NULL	

2.

```
-- yashasvani
INSERT INTO employees (name, department, salary) VALUES
('John Doe', 'Engineering', 50000.00),
('Jane Smith', 'HR', 45000.00),
('Michael Johnson', 'Engineering', 55000.00),
('Emily Brown', 'Sales', 48000.00);

SELECT * FROM employees;
```

id	name	department	salary
1	John Doe	Engineering	50000.00
2	Jane Smith	HR	45000.00
3	Michael Johnson	Engineering	55000.00
4	Emily Brown	Sales	48000.00
NULL	NULL	NULL	NULL

3.

```
-- yashasvani
CREATE INDEX idx_department ON employees (department);
CREATE INDEX idx_salary ON employees (salary);
SHOW INDEX FROM employees;
```

Table	Non_unique	Key_name	Seq_in_index	Column_name	Collation	Cardinality	Sub_part	Packed	Null	Index_type	Comment	Index_comment	Visible	Expression
employees	0	PRIMARY	1	id	A	4	NULL	NULL		BTREE			YES	NULL
employees	1	idx_department	1	department	A	3	NULL	NULL	YES	BTREE			YES	NULL
employees	1	idx_salary	1	salary	A	4	NULL	NULL	YES	BTREE			YES	NULL

4.

```
-- yashasvani
SELECT * FROM employees WHERE department = 'Engineering';
SELECT * FROM employees WHERE salary > 50000.00;
```

id	name	department	salary
3	Michael Johnson	Engineering	55000.00
NULL	NULL	NULL	NULL

5.

```
29      -- yashasvani
30 •    CREATE INDEX idx_covering ON employees (department, salary);
31 •    SELECT department, salary FROM employees WHERE department = 'Engineering';
32 •    SELECT * FROM employees USE INDEX (idx_salary) WHERE salary > 50000.00;
33 •    SELECT * FROM employees USE INDEX (idx_covering) WHERE salary > 50000.00;
34 •    SELECT department, salary FROM employees USE INDEX (idx_covering) WHERE salary > 50000.00;
35 •    SELECT * FROM employees WHERE department = 'Engineering' AND salary > 50000.00;
36 •    SELECT * FROM employees WHERE salary > 50000.00 AND department = 'Engineering';
37 •    SELECT * FROM employees FORCE INDEX (idx_department) WHERE department = 'Engineering';
38 •    CREATE INDEX idx_descending ON employees (department DESC);

38 •    CREATE INDEX idx_descending ON employees (department DESC);
39 •    SELECT department, salary FROM employees USE INDEX (idx_descending) WHERE department = 'Engineering';
40 •    SELECT * FROM employees WHERE salary > 50000.00 ORDER BY department DESC;
41 •    SELECT * FROM employees USE INDEX (idx_descending) WHERE salary > 50000.00 ORDER BY department DESC;
42 •    CREATE INDEX idx_mixed_order ON employees (department ASC, salary DESC);
43 •    SELECT * FROM employees USE INDEX (idx_mixed_order) WHERE salary > 50000.00;
44 •    SHOW INDEX FROM employees;
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

id	name	department	salary
3	Michael Johnson	Engineering	55000.00
NULL	NULL	NULL	NULL

6.

```
46      -- yashasvani
47 •    DROP INDEX idx_department ON employees;
48 •    DROP INDEX idx_salary ON employees;
49 •    DROP INDEX idx_covering ON employees;
50 •    DROP INDEX idx_descending ON employees;
51 •    DROP INDEX idx_mixed_order ON employees;
52 •    SHOW INDEX FROM employees;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

department	salary
Engineering	50000.00
Engineering	55000.00

7.

```
-- yashasvani
CREATE TABLE books_n(
book_id INT AUTO_INCREMENT PRIMARY KEY,
title VARCHAR(255),
author VARCHAR(255),
publication_year int
);
```

28 14:27:41 CREATE TABLE books_n(book_id INT AUTO_INCREMENT PRIMARY KEY, title VARCHAR(255), author VA... 0 row(s) affected

8.

```sql
-- yashasvani
INSERT INTO books (title, author, publication_year) VALUES
('The Adventures of Tom Sawyer', 'Mark Twain', 1876),
('The Picture of Dorian Gray', 'Oscar Wilde', 1890),
('Alice\'s Adventures in Wonderland', 'Lewis Carroll', 1865),
('The War of the Worlds', 'H.G. Wells', 1898),
('Dracula', 'Bram Stoker', 1897),
('Jane Eyre', 'Charlotte Brontë', 1847),
('Wuthering Heights', 'Emily Brontë', 1847),
('Frankenstein', 'Mary Shelley', 1818),
('The Count of Monte Cristo', 'Alexandre Dumas', 1844),
('The Adventures of Sherlock Holmes', 'Arthur Conan Doyle', 1892),
('The Strange Case of Dr. Jekyll and Mr. Hyde', 'Robert Louis Stevenson', 1886),
('The Hound of the Baskervilles', 'Arthur Conan Doyle', 1902),
('Les Misérables', 'Victor Hugo', 1862),
('Anna Karenina', 'Leo Tolstoy', 1877),
('War and Peace', 'Leo Tolstoy', 1869),
('Crime and Punishment', 'Fyodor Dostoevsky', 1866),
('Gone with the Wind', 'Margaret Mitchell', 1936),
('The Grapes of Wrath', 'John Steinbeck', 1939),
('The Old Man and the Sea', 'Ernest Hemingway', 1952),
('The Sun Also Rises', 'Ernest Hemingway', 1926),
('Of Mice and Men', 'John Steinbeck', 1937),
('One Hundred Years of Solitude', 'Gabriel García Márquez', 1967),

('Don Quixote', 'Miguel de Cervantes', 1605),
('The Scarlet Letter', 'Nathaniel Hawthorne', 1850),
('The Canterbury Tales', 'Geoffrey Chaucer', 1400),
('Heart of Darkness', 'Joseph Conrad', 1899),
('The Odyssey', 'Homer', 800),
('The Iliad', 'Homer', 800),
('Catch-22', 'Joseph Heller', 1961),
('To the Lighthouse', 'Virginia Woolf', 1927),
('Slaughterhouse-Five', 'Kurt Vonnegut', 1969),
('The Bell Jar', 'Sylvia Plath', 1963),
('The Great Expectations', 'Charles Dickens', 1861),
('The Adventures of Huckleberry Finn', 'Mark Twain', 1884),
('The Chronicles of Narnia', 'C.S. Lewis', 1950),
('Lord of the Flies', 'William Golding', 1954),
('The Outsiders', 'S.E. Hinton', 1967),
('The Road', 'Cormac McCarthy', 2006),
('Beloved', 'Toni Morrison', 1987),
('Pride and Prejudice', 'Jane Austen', 1813),
('The Hobbit', 'J.R.R. Tolkien', 1937),
('The Lord of the Rings', 'J.R.R. Tolkien', 1954),
('Harry Potter and the Philosopher\'s Stone', 'J.K. Rowling', 1997),
('Moby-Dick', 'Herman Melville', 1851),
('Brave New World', 'Aldous Huxley', 1932),
('The Da Vinci Code', 'Dan Brown', 2003),
```

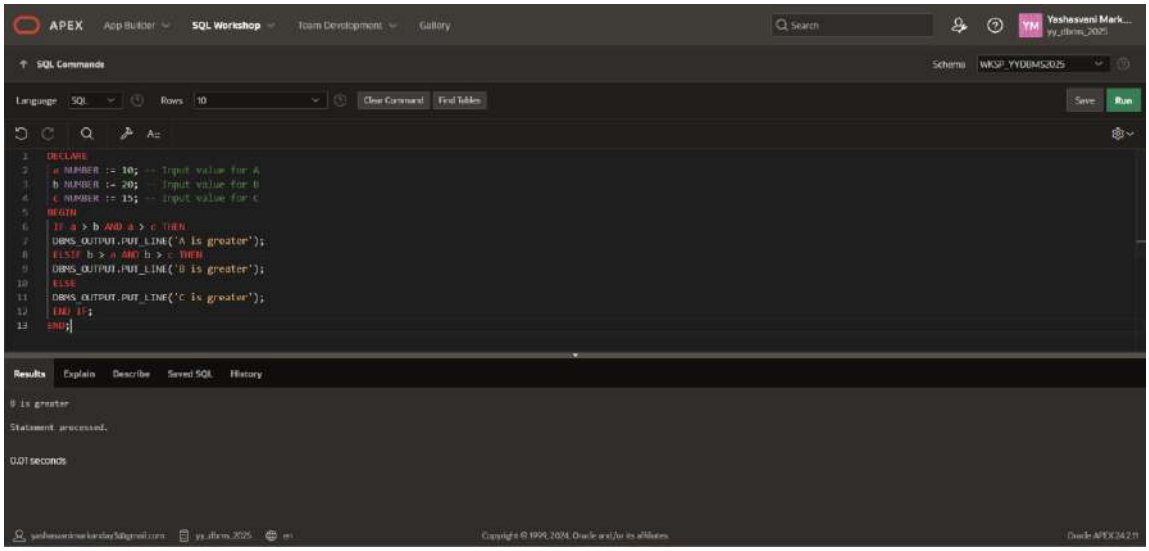
```
('Angels & Demons', 'Dan Brown', 2000),
('The Girl with the Dragon Tattoo', 'Stieg Larsson', 2005),
('The Hunger Games', 'Suzanne Collins', 2008),
('The Catcher in the Rye', 'J.D. Salinger', 1951),
('The Martian', 'Andy Weir', 2011),
('A Game of Thrones', 'George R.R. Martin', 1996),
('The Help', 'Kathryn Stockett', 2009),
('The Girl on the Train', 'Paula Hawkins', 2015),
('Gone Girl', 'Gillian Flynn', 2012),
('The Lovely Bones', 'Alice Sebold', 2002),
('Life of Pi', 'Yann Martel', 2001),
('The Alchemist', 'Paulo Coelho', 1988),
('Twilight', 'Stephenie Meyer', 2005),
('The Secret Life of Bees', 'Sue Monk Kidd', 2001),
('The Book Thief', 'Markus Zusak', 2005),
('Water for Elephants', 'Sara Gruen', 2006),
('The Girl with the Pearl Earring', 'Tracy Chevalier', 1999),
('The Help', 'Kathryn Stockett', 2009),
('The Kite Runner', 'Khaled Hosseini', 2003),
('The Glass Castle', 'Jeannette Walls', 2005),
('Eat, Pray, Love', 'Elizabeth Gilbert', 2006),
('The Time Traveler\'s Wife', 'Audrey Niffenegger', 2003),
('The Goldfinch', 'Donna Tartt', 2013),
('Big Little Lies', 'Liane Moriarty', 2014),

('The Nightingale', 'Kristin Hannah', 2015),
('The Underground Railroad', 'Colson Whitehead', 2016),
('The Silent Patient', 'Alex Michaelides', 2019),
('Where the Crawdads Sing', 'Delia Owens', 2018),
('Becoming', 'Michelle Obama', 2018),
('Educated', 'Tara Westover', 2018),
('Circe', 'Madeline Miller', 2018),
('Normal People', 'Sally Rooney', 2018),
('Little Fires Everywhere', 'Celeste Ng', 2017),
('The Testaments', 'Margaret Atwood', 2019),
('The Tattooist of Auschwitz', 'Heather Morris', 2018),
('The Wife Between Us', 'Greer Hendricks', 2018);
```

29 14:34:45 INSERT INTO books (title, author, publication_year) VALUES ('The Adventures of Tom Sawyer', 'Mark Twain', ... 82 row(s) affected Records: 82 Duplicates: 0 Warnings: 0

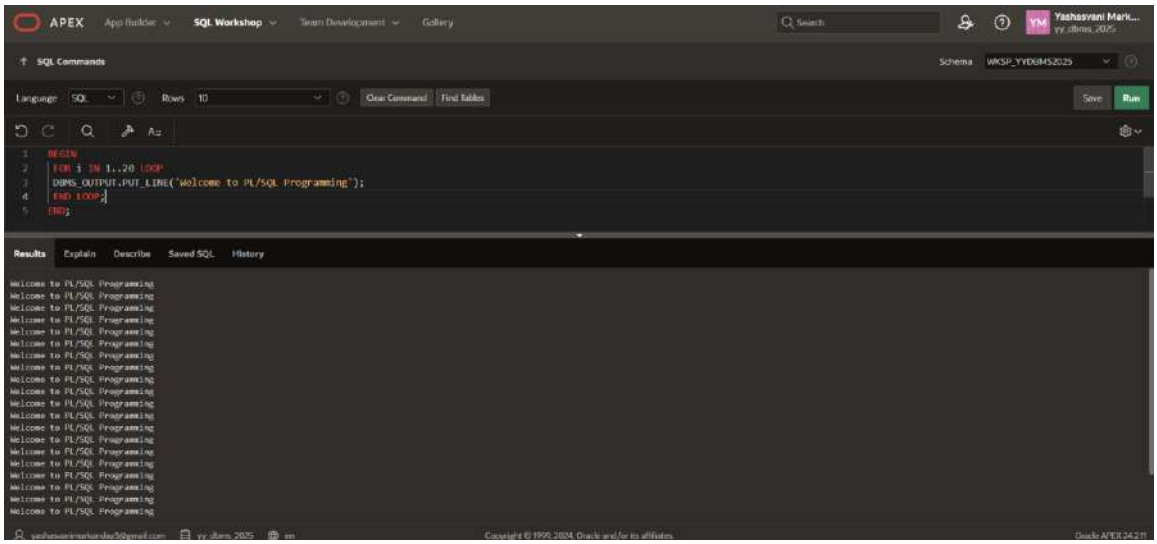
Name: Yashasvani Markanday **SAPID: 590011386**

DBMS LAB – 10 #PL/SQL Commands

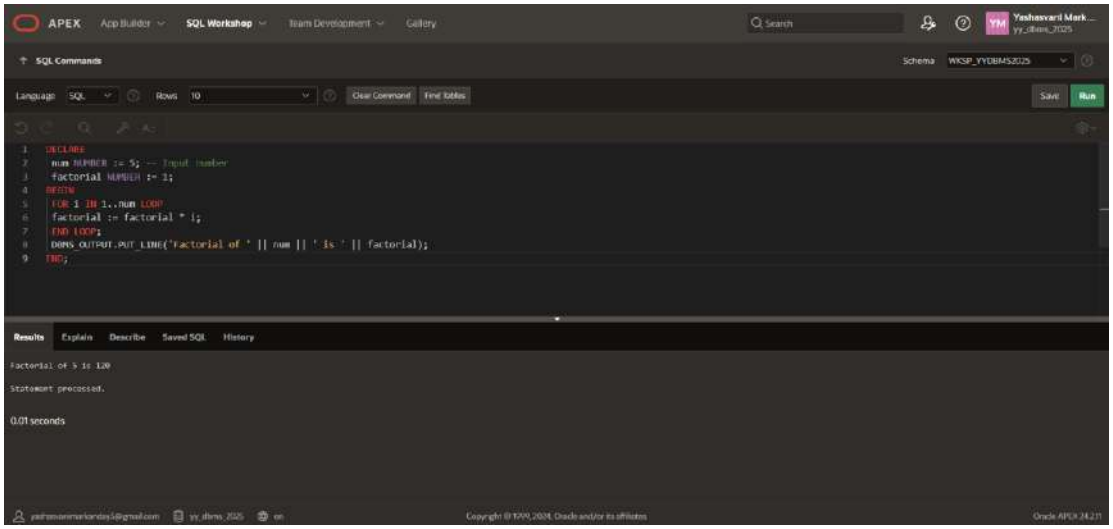
Write a PL/SQL code to accept the value of A, B & C display which is greater



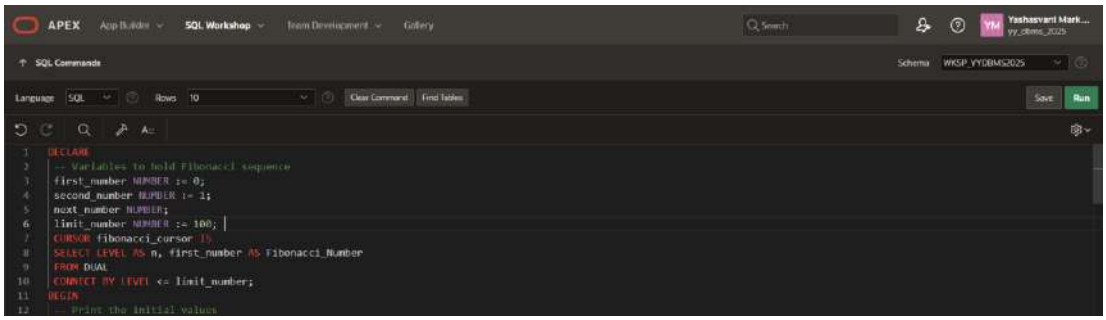
Using PL/SQL Statements create a simple loop that display message “Welcome to PL/SQL Programming” 20 times.

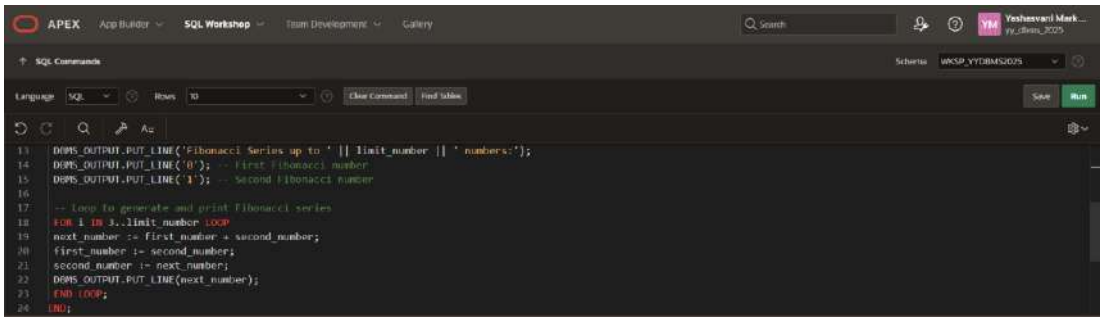


Write a PL/SQL code block to find the factorial of a number.

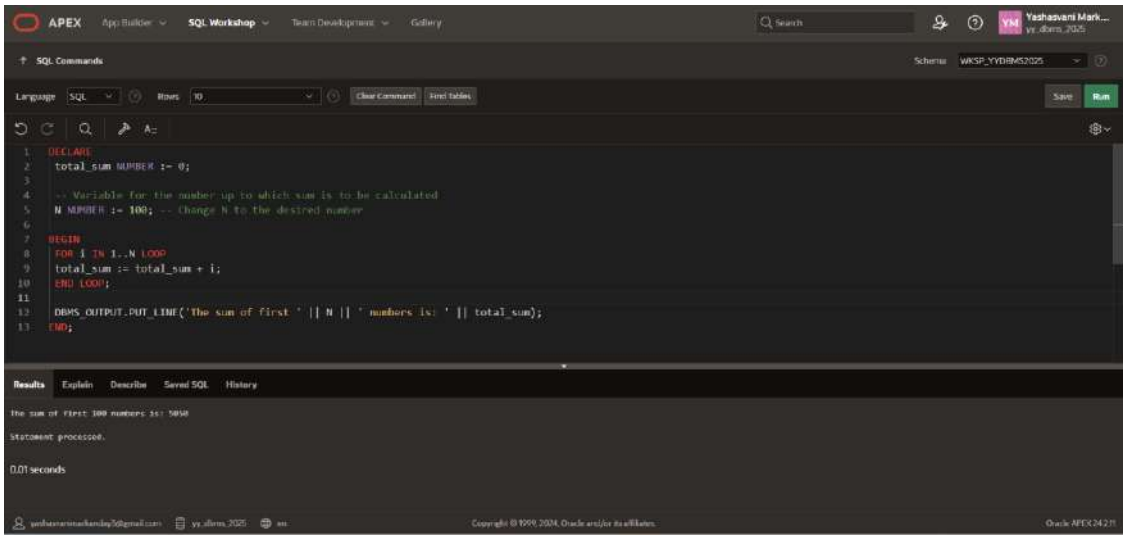


Write a PL/SQL program to generate Fibonacci series.

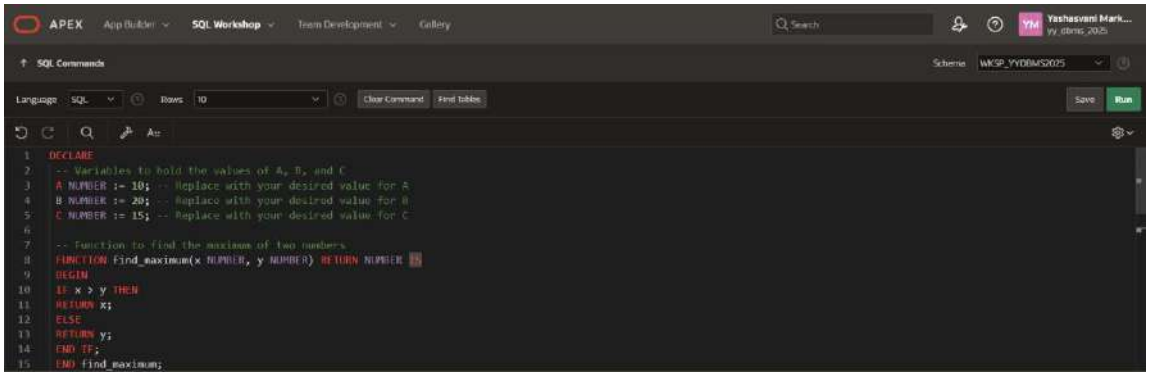




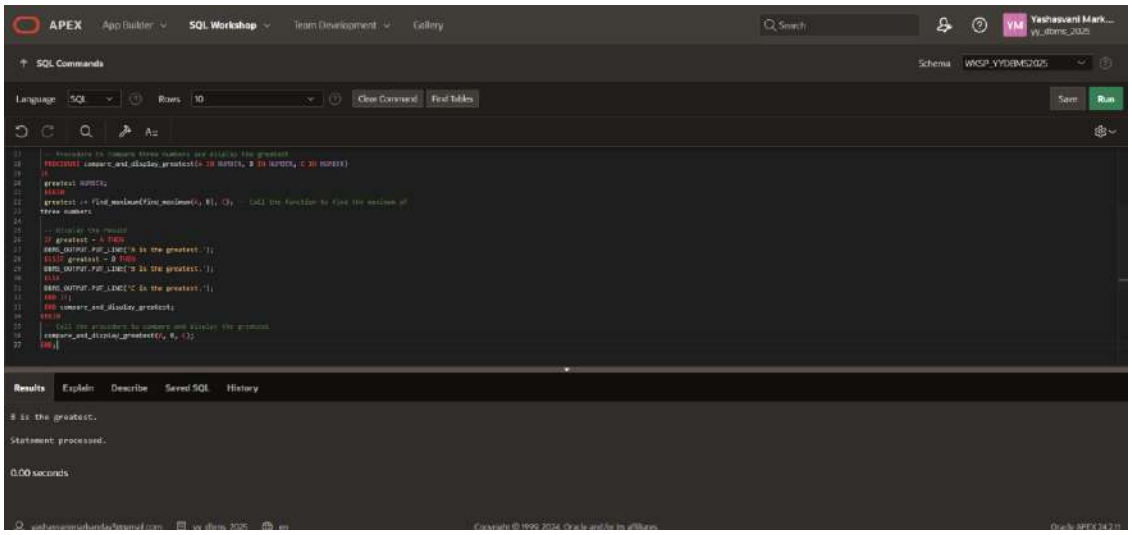
Write a PL/SQL code to find the sum of first N numbers.



WAP that accept the value of A, B & C display which is greater using Functions and Procedures.

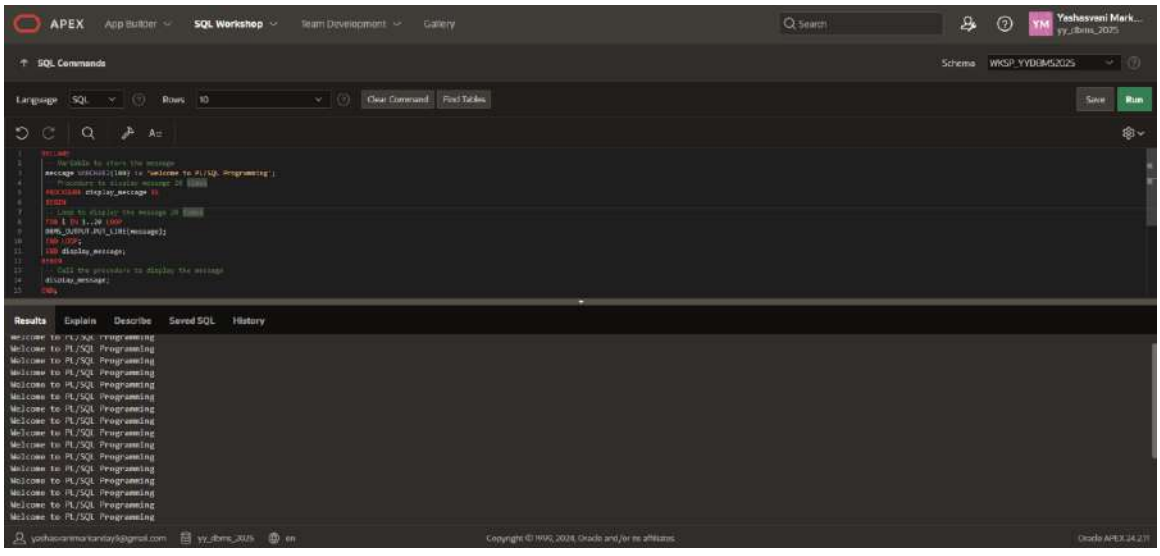


```
DECLARE
  -- Variables to hold the values of A, B, and C
  A NUMBER := 10; -- Replace with your desired value for A
  B NUMBER := 20; -- Replace with your desired value for B
  C NUMBER := 15; -- Replace with your desired value for C

  -- Function to find the maximum of two numbers
  FUNCTION find_maximum(x NUMBER, y NUMBER) RETURN NUMBER IS
  BEGIN
  IF x > y THEN
  RETURN x;
  ELSE
  RETURN y;
  END IF;
  END find_maximum;
```



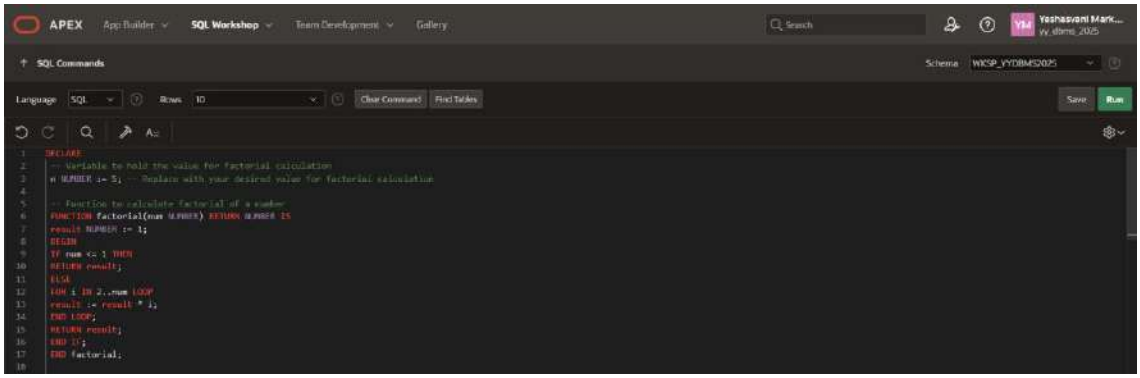
```
Results  Explain  Describe  Saved SQL  History

B is the greatest.

Statement processed.

0.00 seconds
```

Print the statements “Welcome to PL/SQL Programming” 20 times using Functions and Procedures.



```
Results  Explain  Describe  Saved SQL  History

Welcome to PL/SQL Programming
Welcome to PL/SQL Programming
Welcome to PL/SQL Programming
Welcome to PL/SQL Programming
Welcome to PL/SQL Programming
Welcome to PL/SQL Programming
Welcome to PL/SQL Programming
Welcome to PL/SQL Programming
Welcome to PL/SQL Programming
Welcome to PL/SQL Programming
Welcome to PL/SQL Programming
Welcome to PL/SQL Programming
Welcome to PL/SQL Programming
Welcome to PL/SQL Programming
Welcome to PL/SQL Programming
Welcome to PL/SQL Programming
Welcome to PL/SQL Programming
Welcome to PL/SQL Programming
```

Factorial of a number using function and procedure.

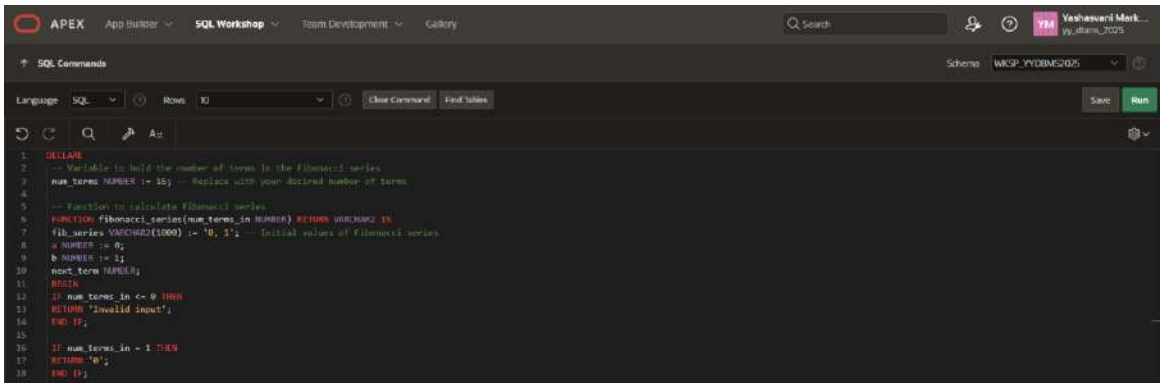


```
DECLARE
  -- Variable to hold the value for factorial calculation
  n NUMBER := 5; -- Replace with your desired value for factorial calculation

  -- Function to calculate factorial of a number
  FUNCTION factorial(num NUMBER) RETURN NUMBER IS
    result NUMBER := 1;
  BEGIN
    IF num <= 1 THEN
      RETURN result;
    ELSE
      FOR i IN 2..num LOOP
        result := result * i;
      END LOOP;
      RETURN result;
    END IF;
  END factorial;
```

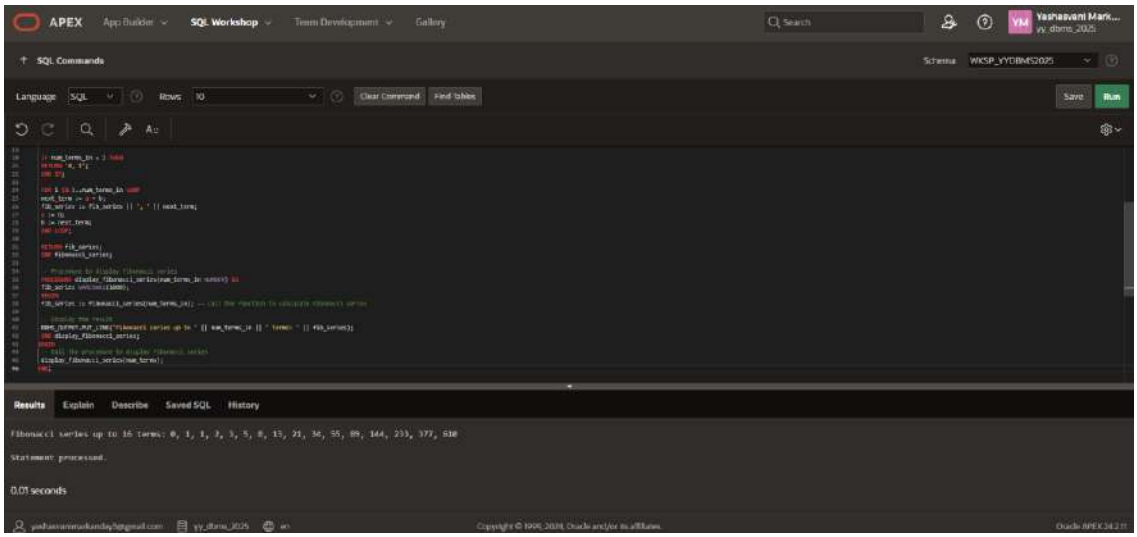
```
  -- Procedure to calculate and display factorial
  PROCEDURE calculate_and_display_factorial(n IN NUMBER) IS
    fact NUMBER;
  BEGIN
    fact := factorial(n); -- Call the function to calculate factorial

    -- Display the result
    DBMS_OUTPUT.PUT_LINE('Factorial of ' || n || ' is: ' || fact);
  END calculate_and_display_factorial;
BEGIN
  -- Call the procedure to calculate and display factorial
  calculate_and_display_factorial(n);
END;
```

```
Results  Explain  Describe  Saved SQL  History

Factorial of 5 is: 120

Statement processed.

0.01 seconds
```

Fibonacci series of a number using function and procedure.

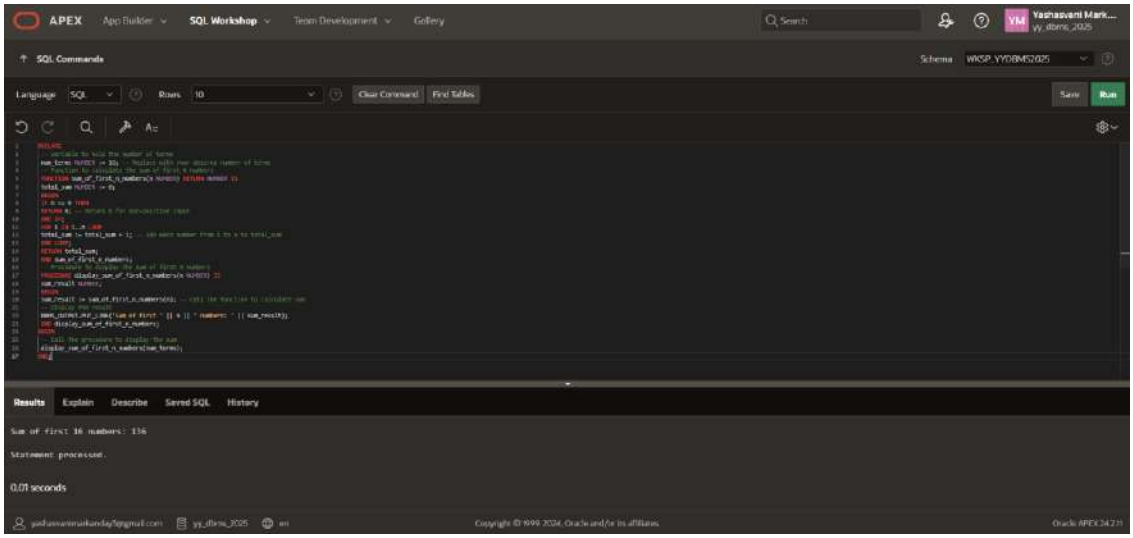


```
DECLARE
  -- Variable to hold the number of terms in the Fibonacci series
  num_terms NUMBER := 16; -- Replace with your desired number of terms

  -- Function to calculate Fibonacci series
  FUNCTION fibonacci_series(num_terms_in NUMBER) RETURN VARCHAR2 IS
    fib_series VARCHAR2(1000) := '0, 1'; -- Initial values of Fibonacci series
    a NUMBER := 0;
    b NUMBER := 1;
    next_term NUMBER;
  BEGIN
    IF num_terms_in <= 0 THEN
      RETURN 'Invalid input';
    END IF;

    IF num_terms_in = 1 THEN
      RETURN '0';
    END IF;
```



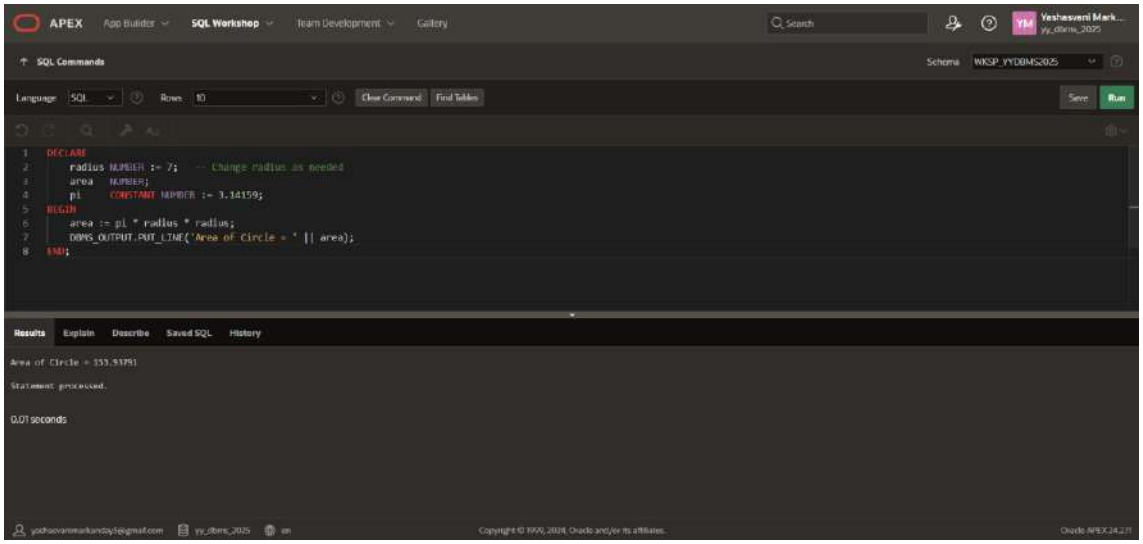
```
Results  Explain  Describe  Saved SQL  History

Fibonacci series up to 16 Terms: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610

Statement processed.

0.01 seconds
```

Find the sum of first N numbers using Function and Procedure.

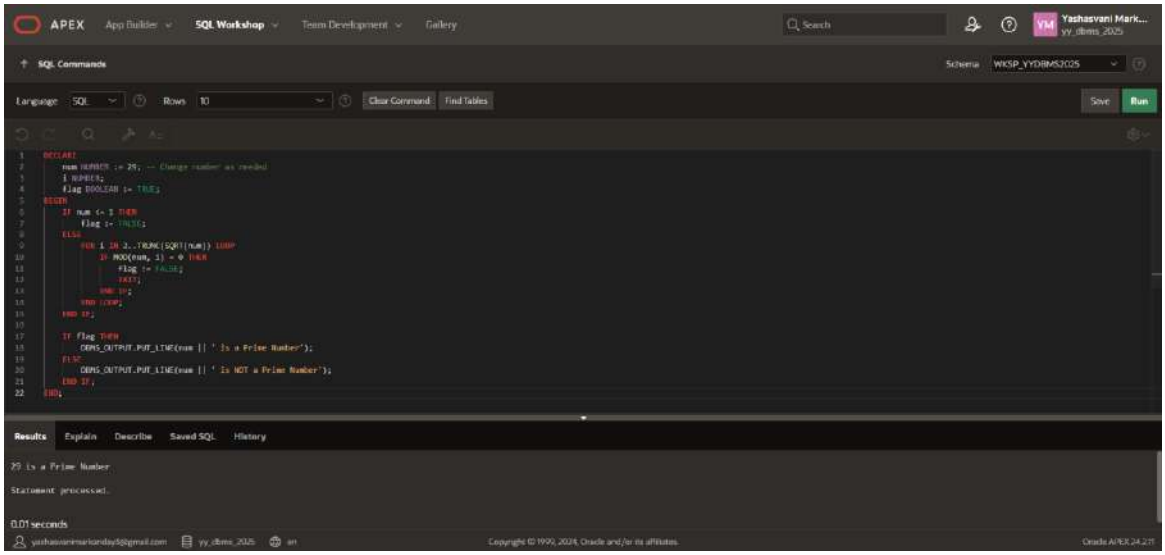


Lab Performance Questions

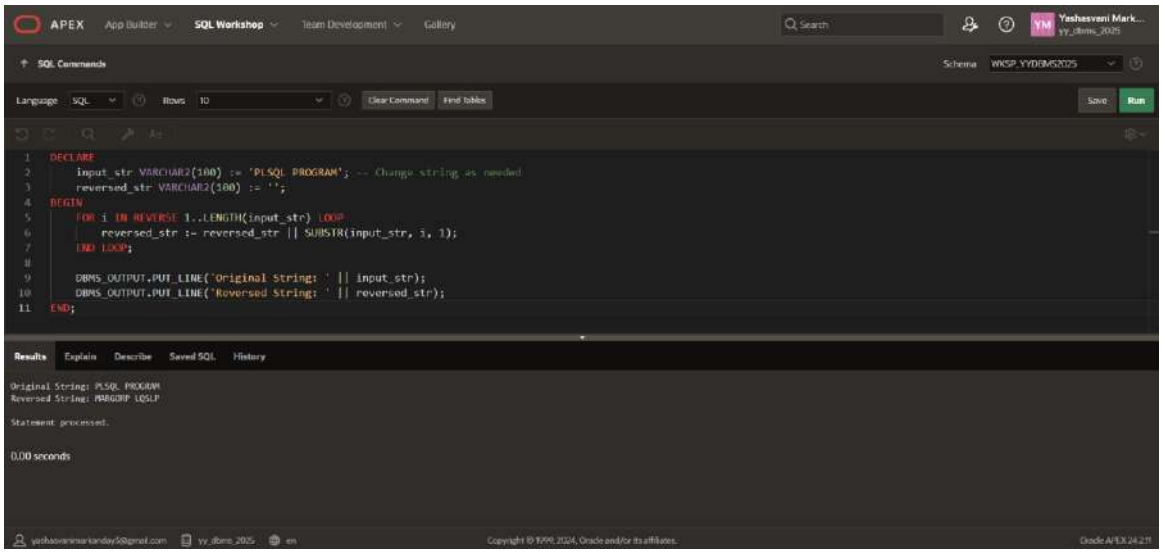
1. Write a PL/SQL code to calculate the area of a circle given its radius.



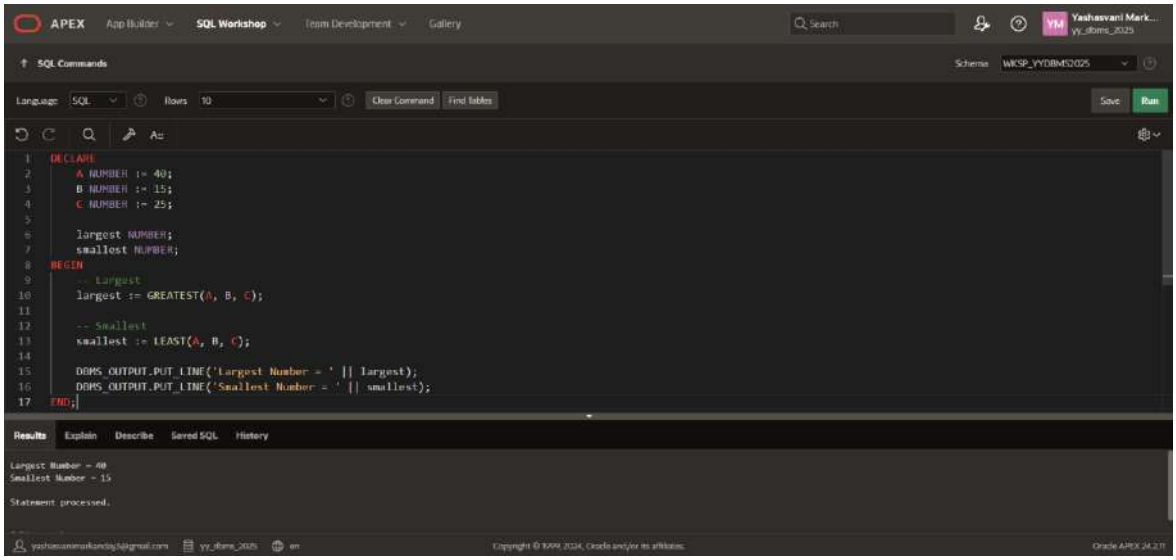
2. Create a PL/SQL program to check whether a given number is prime or not.



3. Develop a PL/SQL script to reverse a string entered by the user.



4. Design a PL/SQL code to find the largest and smallest number among three given numbers.



5. Implement a PL/SQL program to calculate the compound interest for a given principal amount, rate, and time period