EXPERIMENT 4

Student Name: Jay Varshney **UID: 24BDA70206**
Branch: AIT-CSE (FSD) **Section/Group: 24AIT_KRG2**
Semester: 4
Subject Name: Database Management System **Subject Code: 24CSH-298**

Experiment 4 – Data Analysis Using SQL and PL/SQL

Experiment
Experiment 4: Creating tables, inserting data, performing conditional queries, and using PL/SQL blocks to analyze schema violations and student grades. This experiment demonstrates table creation, updates, conditional logic, and ordering in Oracle SQL and PL/SQL.

Aim
The aim of this experiment is to practice working with Oracle SQL tables, using conditional logic to determine status and grades, and displaying results using SELECT queries and PL/SQL blocks.

Objective

- To create and populate tables in Oracle SQL.
- To use CASE statements for conditional evaluation of violation counts and student grades.
- To add and update columns based on conditions.
- To use PL/SQL anonymous blocks for status messages.
- To sort query results based on defined criteria.

Software Requirements

- Database: Oracle XE or Oracle Live SQL

Practical / Experiment Steps

1. Create a table schema_violations with columns id, schema_name, and violation_count.
2. Insert data for various departments into the schema_violations table.
3. Select violation status for each department using a CASE statement.

4. Add a new column approval_status to schema_violations.
5. Update approval_status based on violation count using a CASE statement.
6. Display the updated schema_violations table.
7. Execute a PL/SQL block to print a system status message based on a variable v_count.
8. Create a students table with columns name and marks.
9. Insert student data into the students table.
10. Display student grades using a CASE statement based on marks.
11. Order schema_violations by severity using a CASE statement in ORDER BY.

Procedure of the Experiment

1. Open Oracle XE or Live SQL and connect to the database.
2. Create the schema_violations and students tables.
3. Insert sample data into both tables.
4. Execute SELECT queries with CASE statements to analyze violation and grade data.
5. Alter and update tables using conditional logic.
6. Write and execute a PL/SQL anonymous block for dynamic status messages.
7. Sort and retrieve data based on defined severity.
8. Observe outputs at each step and take screenshots for documentation.

Input / Output Details

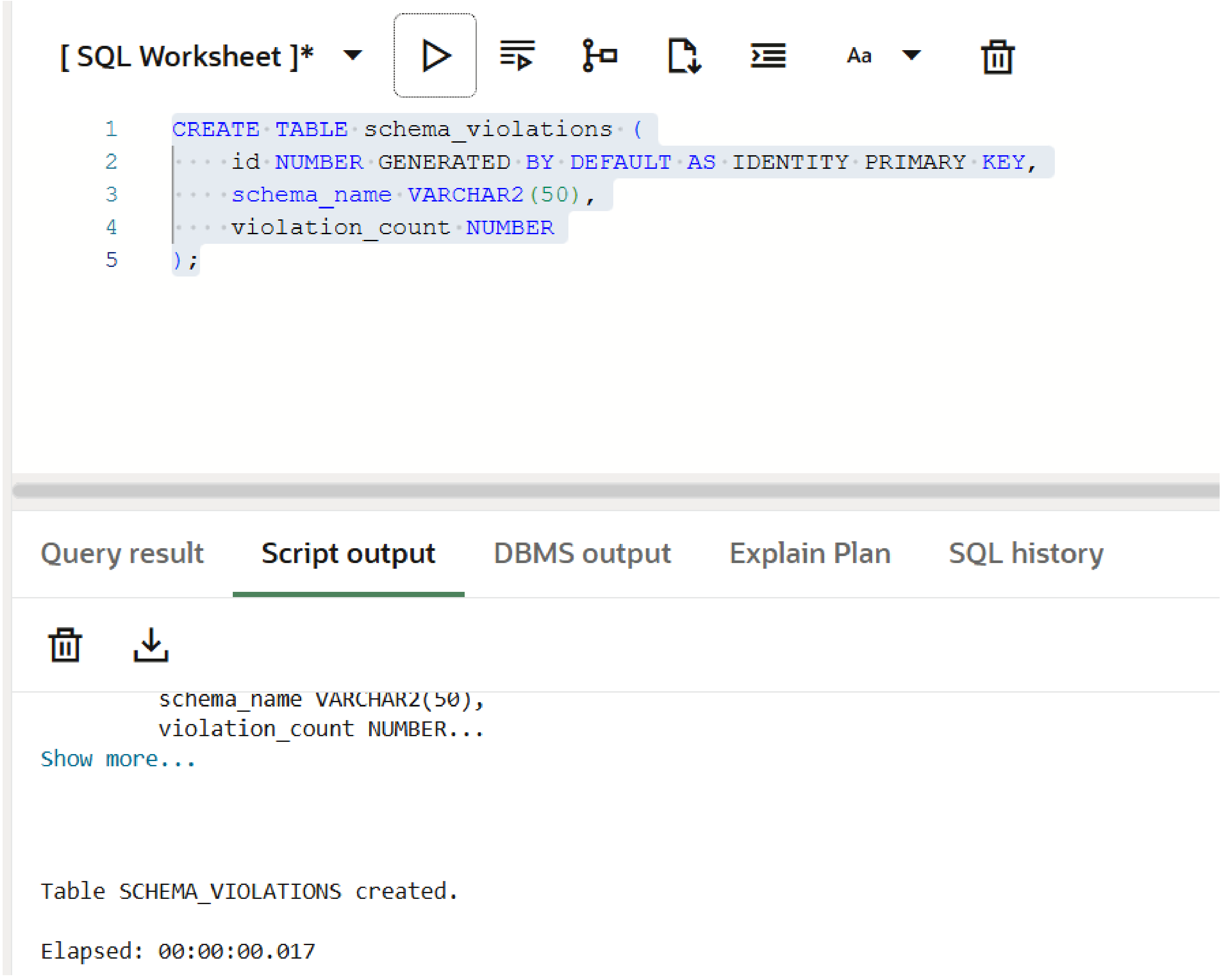
Input

- schema_violations table: id, schema_name, violation_count
- students table: name, marks
- PL/SQL block variable: v_count
- Conditional logic in SELECT and UPDATE statements

Step-wise Output

EXPERIMENT 4

Step 1 – Create schema_violations table

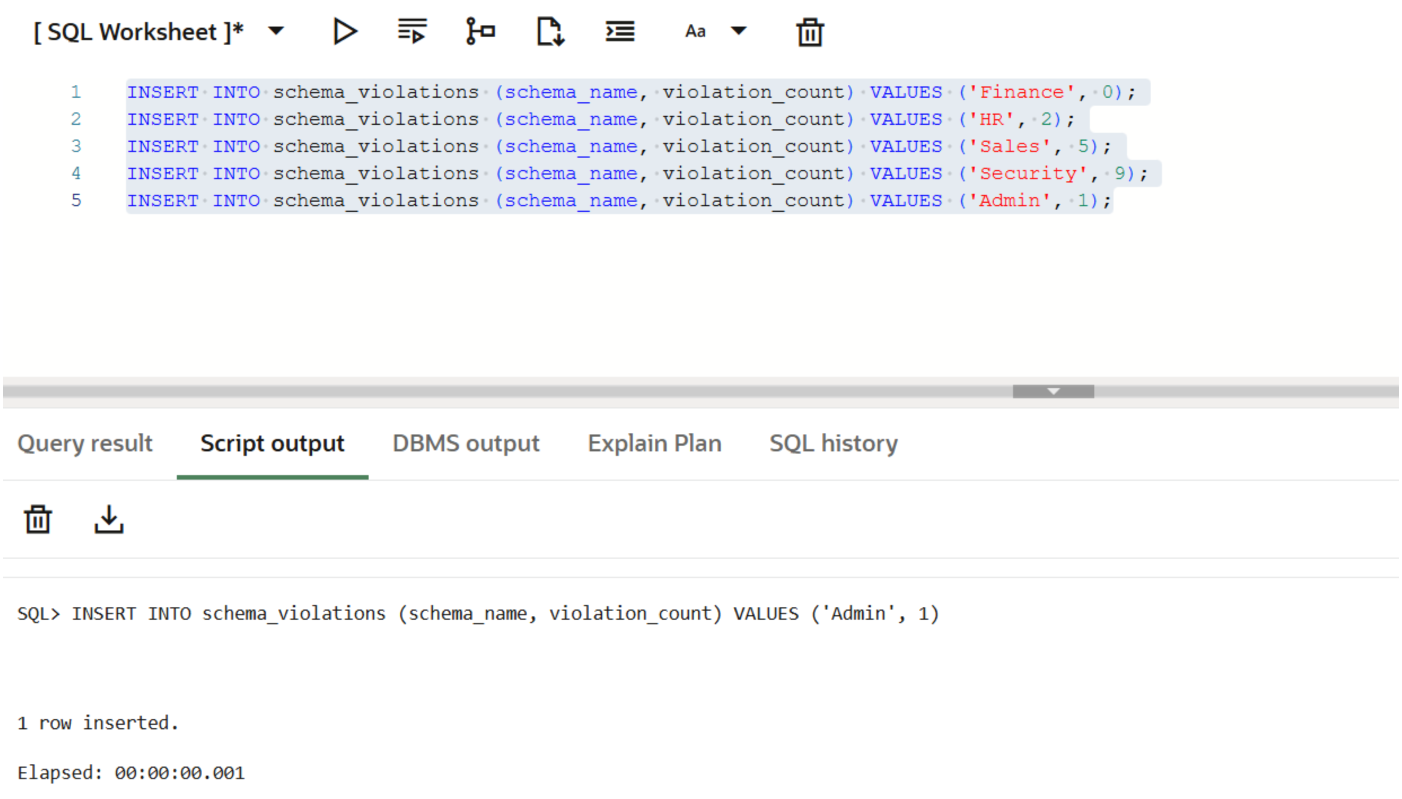


```
CREATE TABLE schema_violations (
    id NUMBER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
    schema_name VARCHAR2(50),
    violation_count NUMBER
);
```

```
        schema_name VARCHAR2(50),
        violation_count NUMBER...
Show more...


Table SCHEMA_VIOLATIONS created.

Elapsed: 00:00:00.017
```

EXPERIMENT 4

Step 2 – Insert data into schema_violations

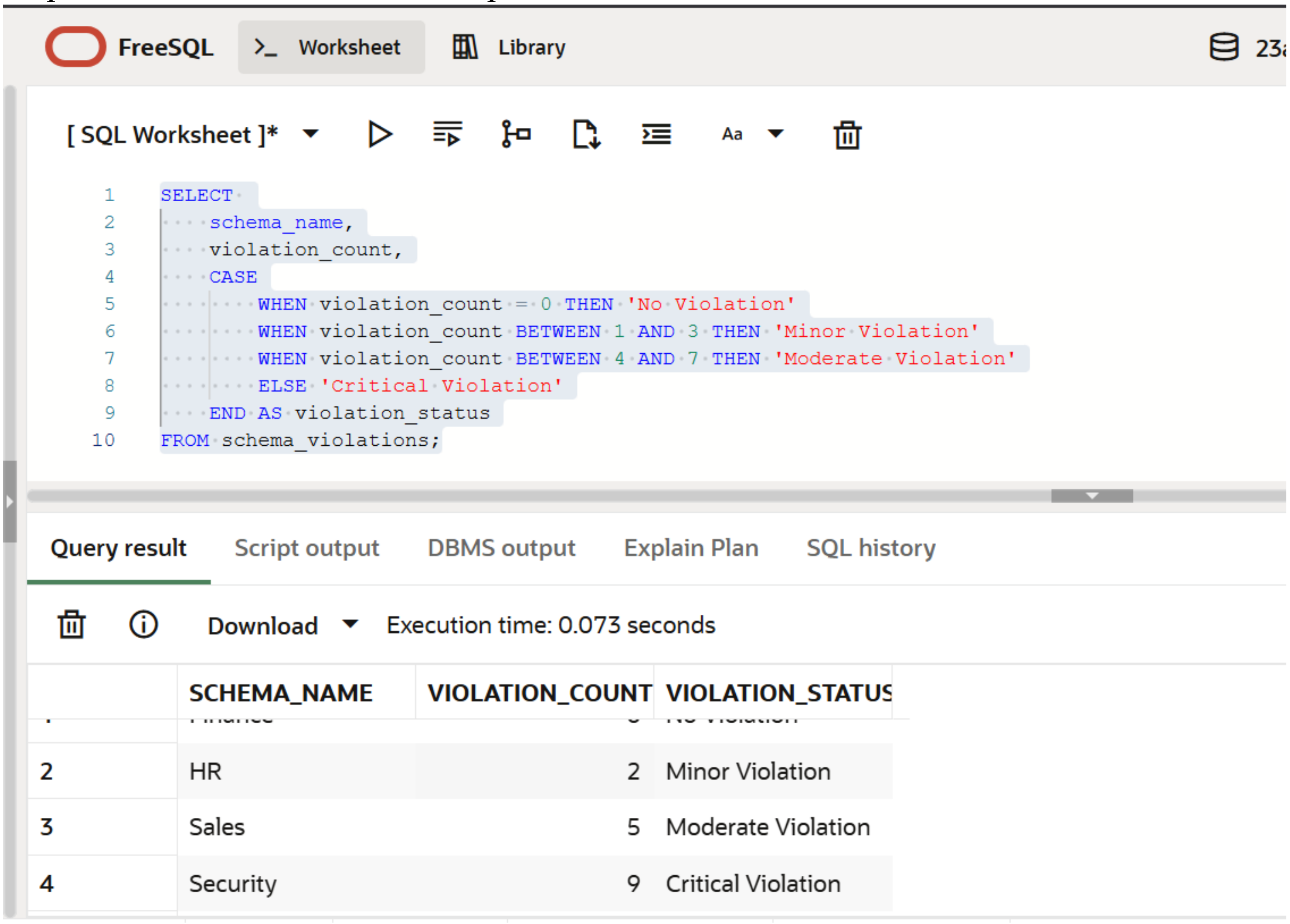


```
INSERT INTO schema_violations (schema_name, violation_count) VALUES ('Finance', 0);
INSERT INTO schema_violations (schema_name, violation_count) VALUES ('HR', 2);
INSERT INTO schema_violations (schema_name, violation_count) VALUES ('Sales', 5);
INSERT INTO schema_violations (schema_name, violation_count) VALUES ('Security', 9);
INSERT INTO schema_violations (schema_name, violation_count) VALUES ('Admin', 1);
```

```
SQL> INSERT INTO schema_violations (schema_name, violation_count) VALUES ('Admin', 1)


1 row inserted.

Elapsed: 00:00:00.001
```

EXPERIMENT 4

Step 3 – Violation status of each department



```sql
SELECT
    schema_name,
    violation_count,
    CASE
        WHEN violation_count = 0 THEN 'No Violation'
        WHEN violation_count BETWEEN 1 AND 3 THEN 'Minor Violation'
        WHEN violation_count BETWEEN 4 AND 7 THEN 'Moderate Violation'
        ELSE 'Critical Violation'
    END AS violation_status
FROM schema_violations;
```

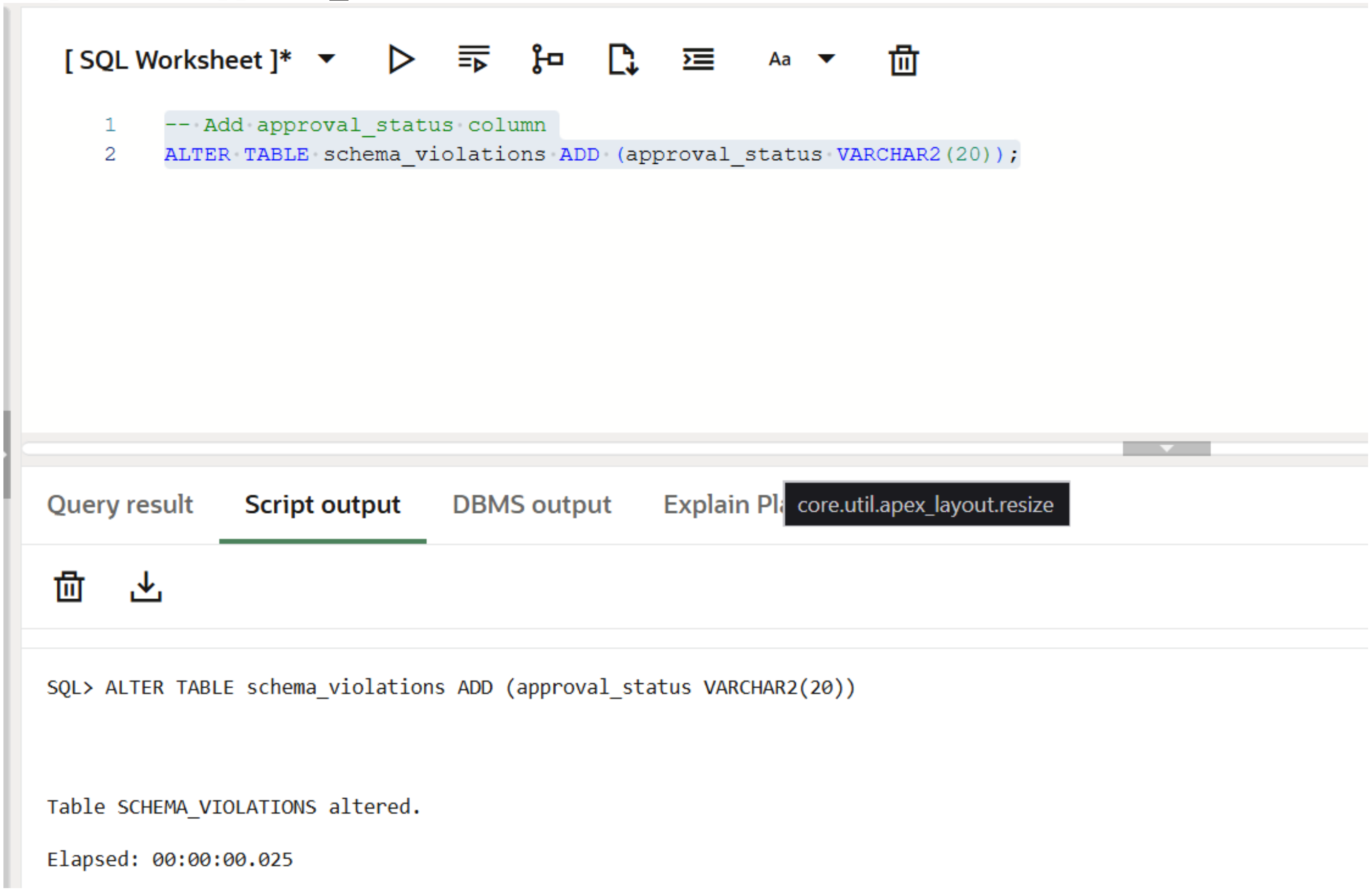
Execution time: 0.073 seconds

	SCHEMA_NAME	VIOLATION_COUNT	VIOLATION_STATUS
1	Finance	0	No Violation
2	HR	2	Minor Violation
3	Sales	5	Moderate Violation
4	Security	9	Critical Violation

schema_name	violation_count	violation_status
Finance	0	No Violation
HR	2	Minor Violation
Sales	5	Moderate Violation
Security	9	Critical Violation
Admin	1	Minor Violation

EXPERIMENT 4

Step 4 – Add approval_status column

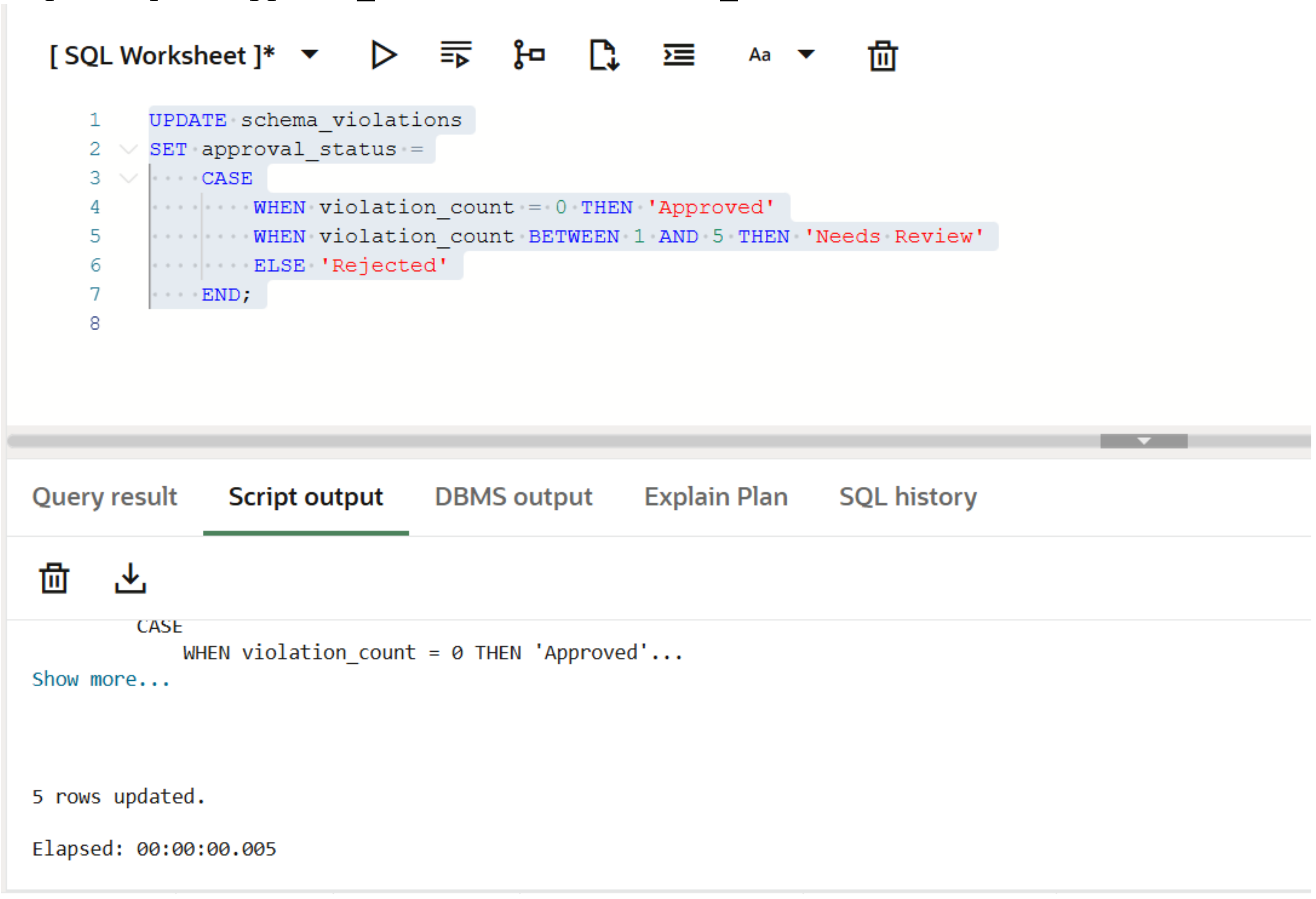


```
-- Add approval_status column
ALTER TABLE schema_violations ADD (approval_status VARCHAR2(20));
```

```
SQL> ALTER TABLE schema_violations ADD (approval_status VARCHAR2(20))


Table SCHEMA_VIOLATIONS altered.

Elapsed: 00:00:00.025
```

EXPERIMENT 4

Step 5 – Update approval_status based on violation_count

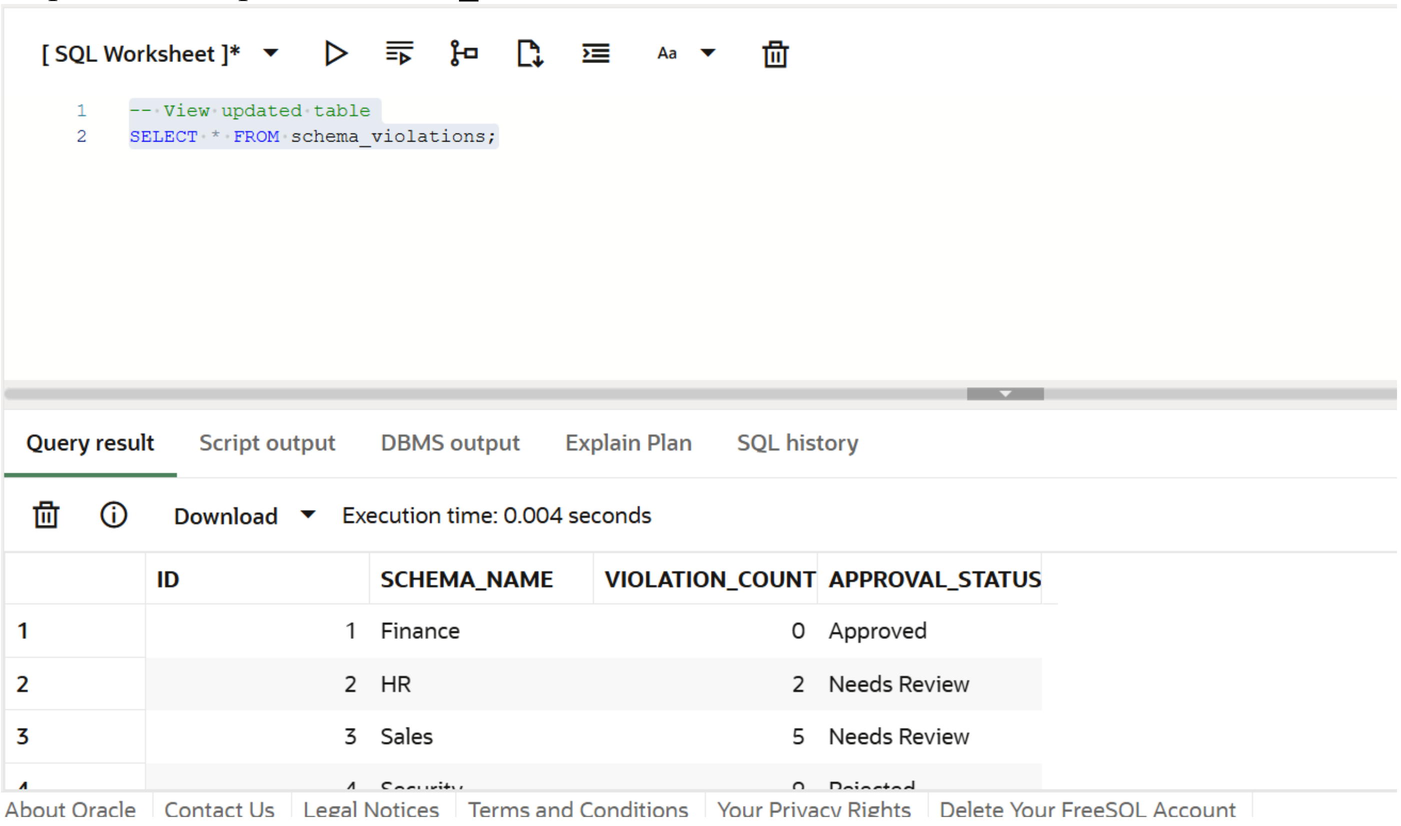


```
UPDATE schema_violations
SET approval_status =
    CASE
        WHEN violation_count = 0 THEN 'Approved'
        WHEN violation_count BETWEEN 1 AND 5 THEN 'Needs Review'
        ELSE 'Rejected'
    END;
```

```
    CASE
        WHEN violation_count = 0 THEN 'Approved'...
Show more...



5 rows updated.

Elapsed: 00:00:00.005
```

EXPERIMENT 4

Step 6 – View updated schema_violations table



```sql
-- View updated table
SELECT * FROM schema_violations;
```

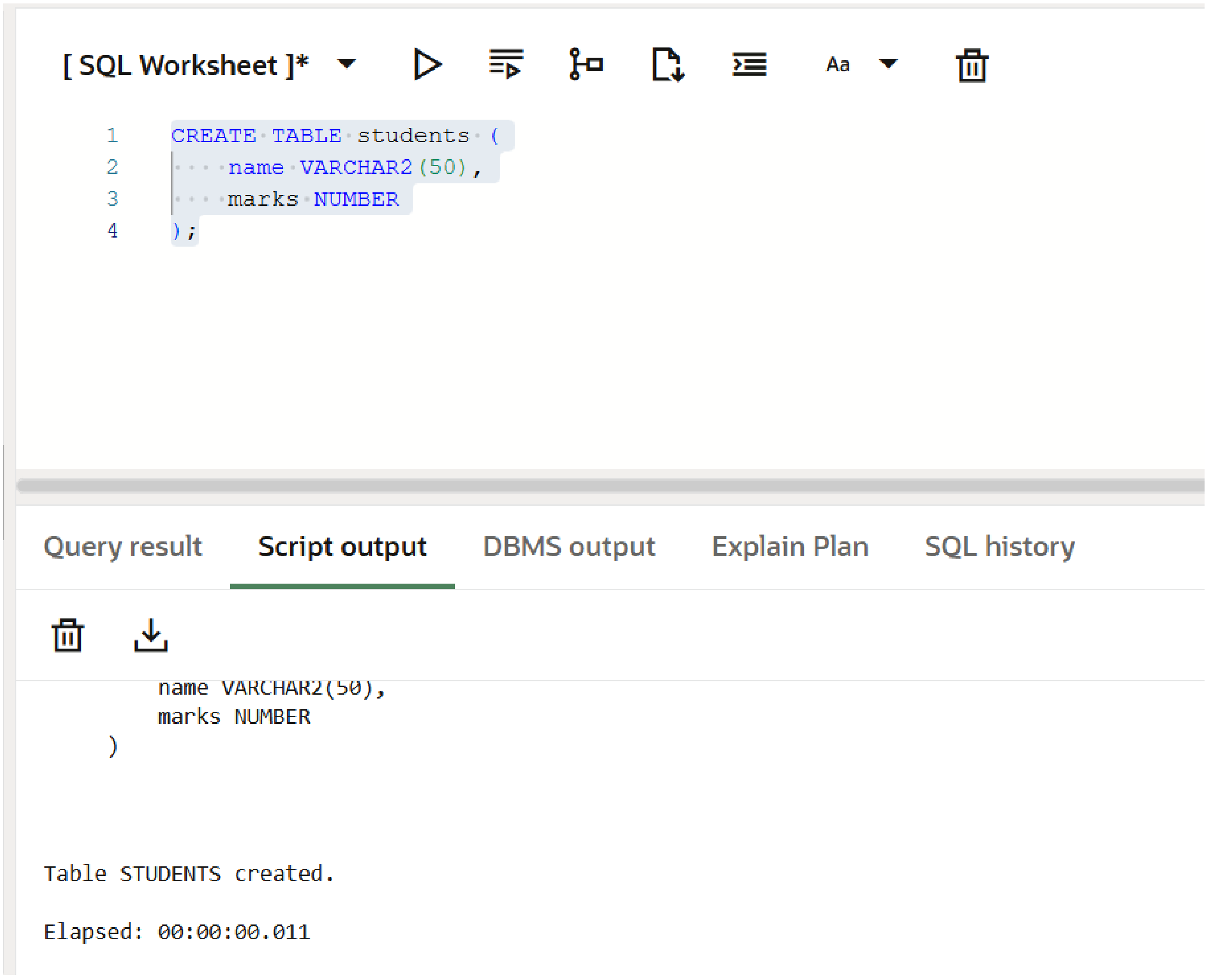
Execution time: 0.004 seconds

	ID	SCHEMA_NAME	VIOLATION_COUNT	APPROVAL_STATUS
1	1	Finance	0	Approved
2	2	HR	2	Needs Review
3	3	Sales	5	Needs Review
4	4	Security	9	Rejected

id	schema_name	violation_count	violation_status	approval_status
1	Finance	0	No Violation	Approved
2	HR	2	Minor Violation	Needs Review
3	Sales	5	Moderate Violation	Needs Review
4	Security	9	Critical Violation	Rejected
5	Admin	1	Minor Violation	Needs Review

Step 7 – PL/SQL anonymous block for status message
Output:

EXPERIMENT 4

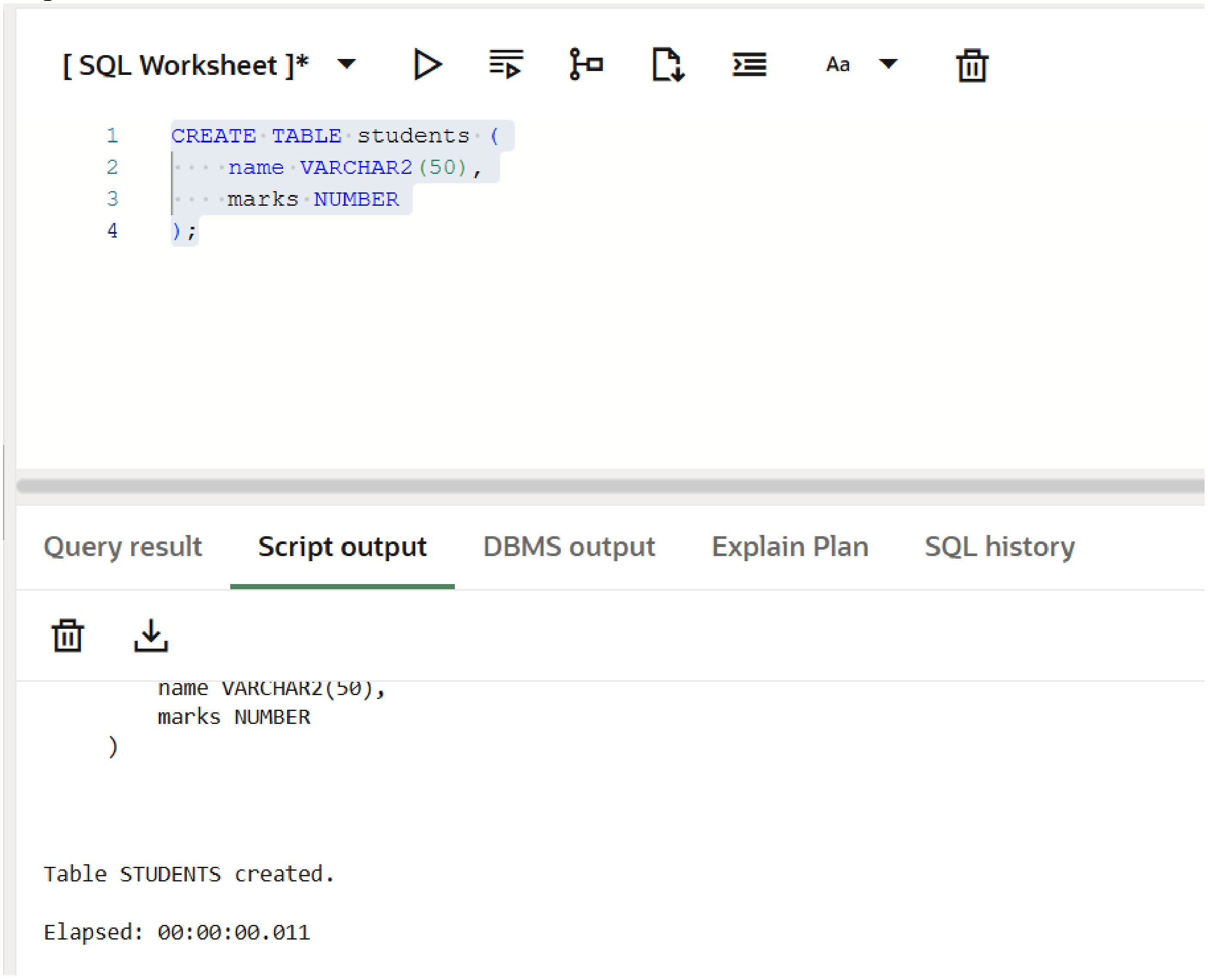


```sql
CREATE TABLE students (
    name VARCHAR2(50),
    marks NUMBER
);
```

```
        name VARCHAR2(50),
        marks NUMBER
    )



Table STUDENTS created.

Elapsed: 00:00:00.011
```

EXPERIMENT 4

Step 8 – Create students table

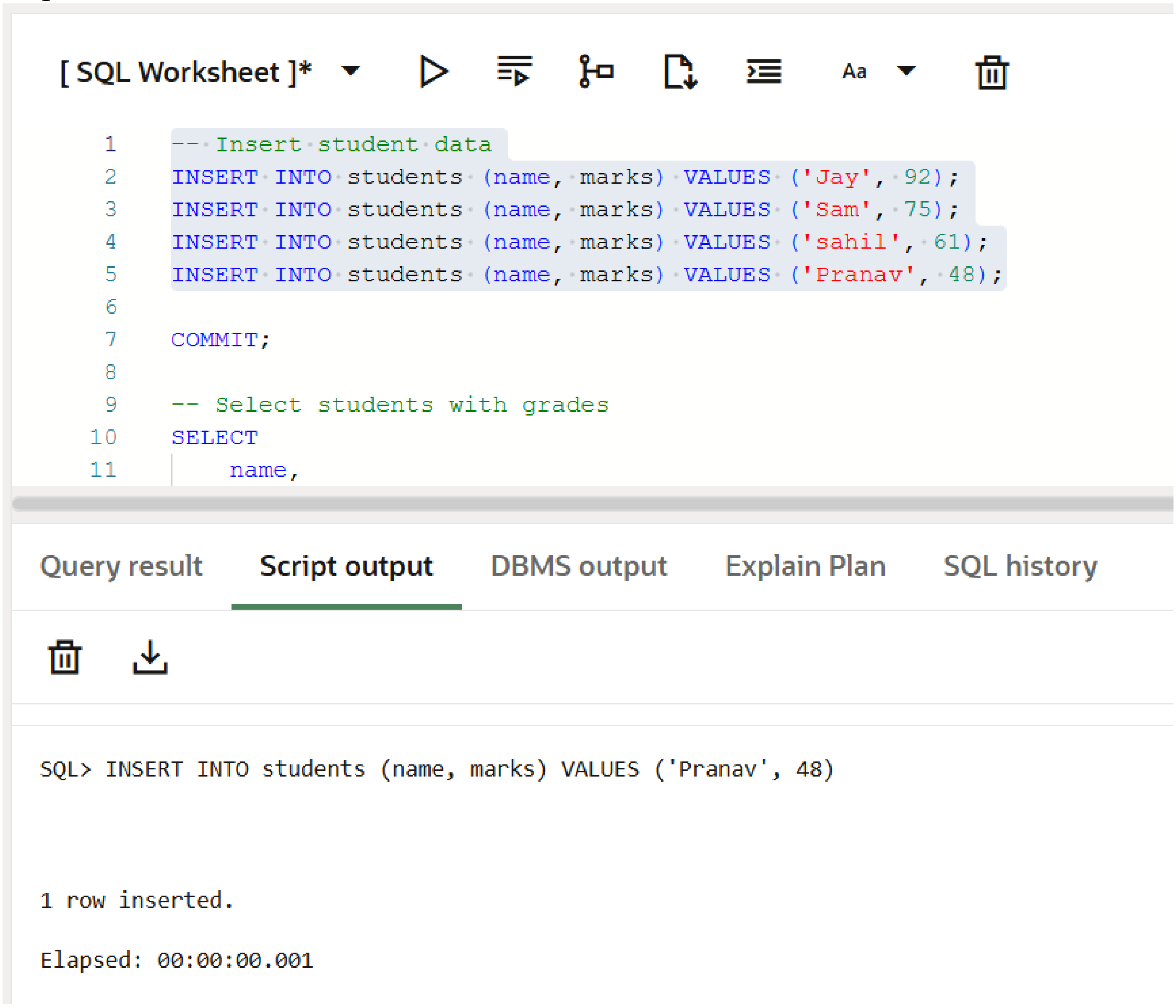


```
CREATE TABLE students (
    name VARCHAR2(50),
    marks NUMBER
);
```

```
        name VARCHAR2(50),
        marks NUMBER
    )



Table STUDENTS created.

Elapsed: 00:00:00.011
```

EXPERIMENT 4

Step 9 – Insert student data

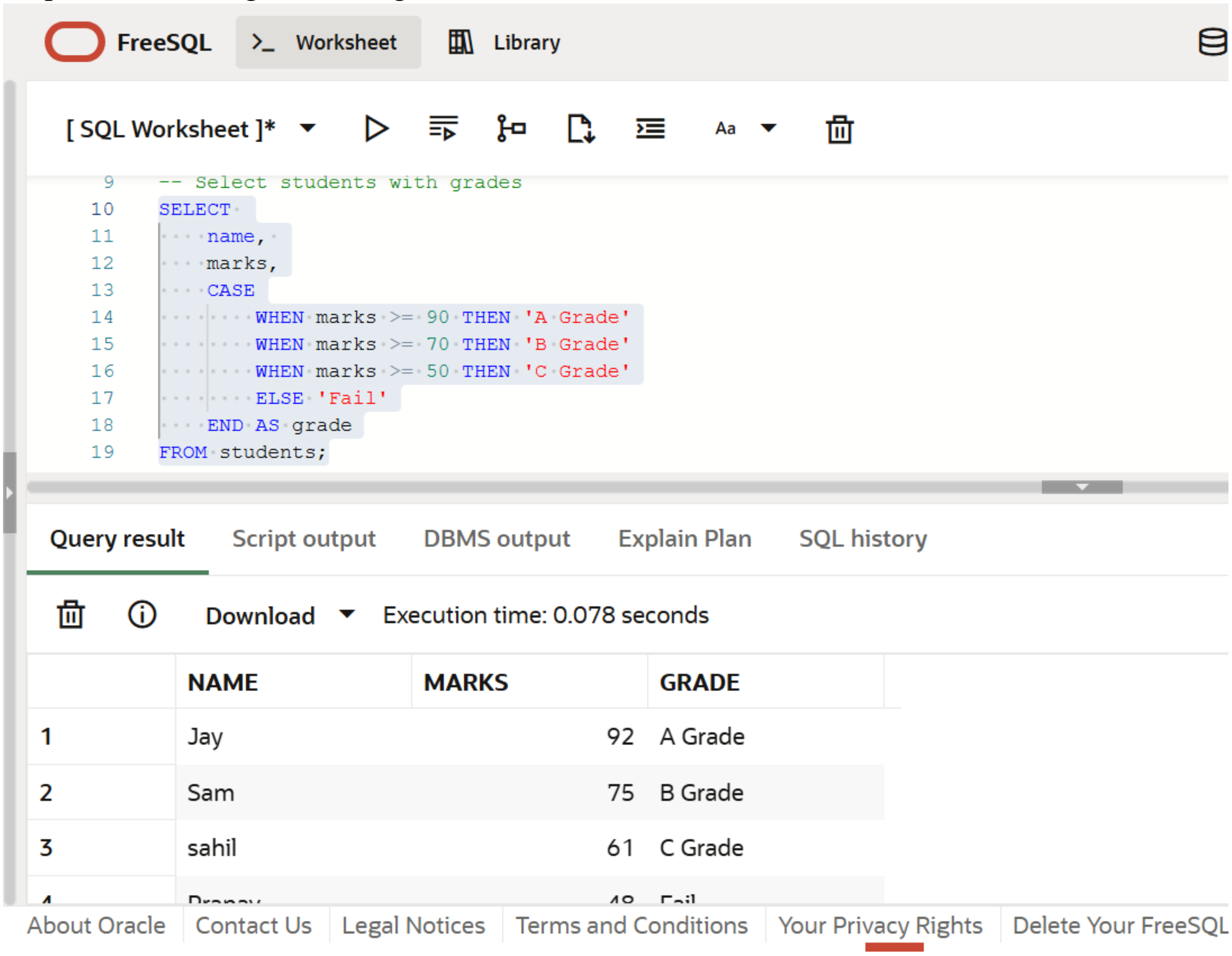


```sql
-- Insert student data
INSERT INTO students (name, marks) VALUES ('Jay', 92);
INSERT INTO students (name, marks) VALUES ('Sam', 75);
INSERT INTO students (name, marks) VALUES ('sahil', 61);
INSERT INTO students (name, marks) VALUES ('Pranav', 48);

COMMIT;

-- Select students with grades
SELECT
    name,
```

```
SQL> INSERT INTO students (name, marks) VALUES ('Pranav', 48)



1 row inserted.

Elapsed: 00:00:00.001
```

EXPERIMENT 4

Step 10 – Student grades using CASE statement



```sql
-- Select students with grades
SELECT
    name,
    marks,
    CASE
        WHEN marks >= 90 THEN 'A Grade'
        WHEN marks >= 70 THEN 'B Grade'
        WHEN marks >= 50 THEN 'C Grade'
        ELSE 'Fail'
    END AS grade
FROM students;
```

Execution time: 0.078 seconds

	NAME	MARKS	GRADE
1	Jay	92	A Grade
2	Sam	75	B Grade
3	sahil	61	C Grade
4	Pranav	48	Fail

name marks grade

Jay 92 A Grade

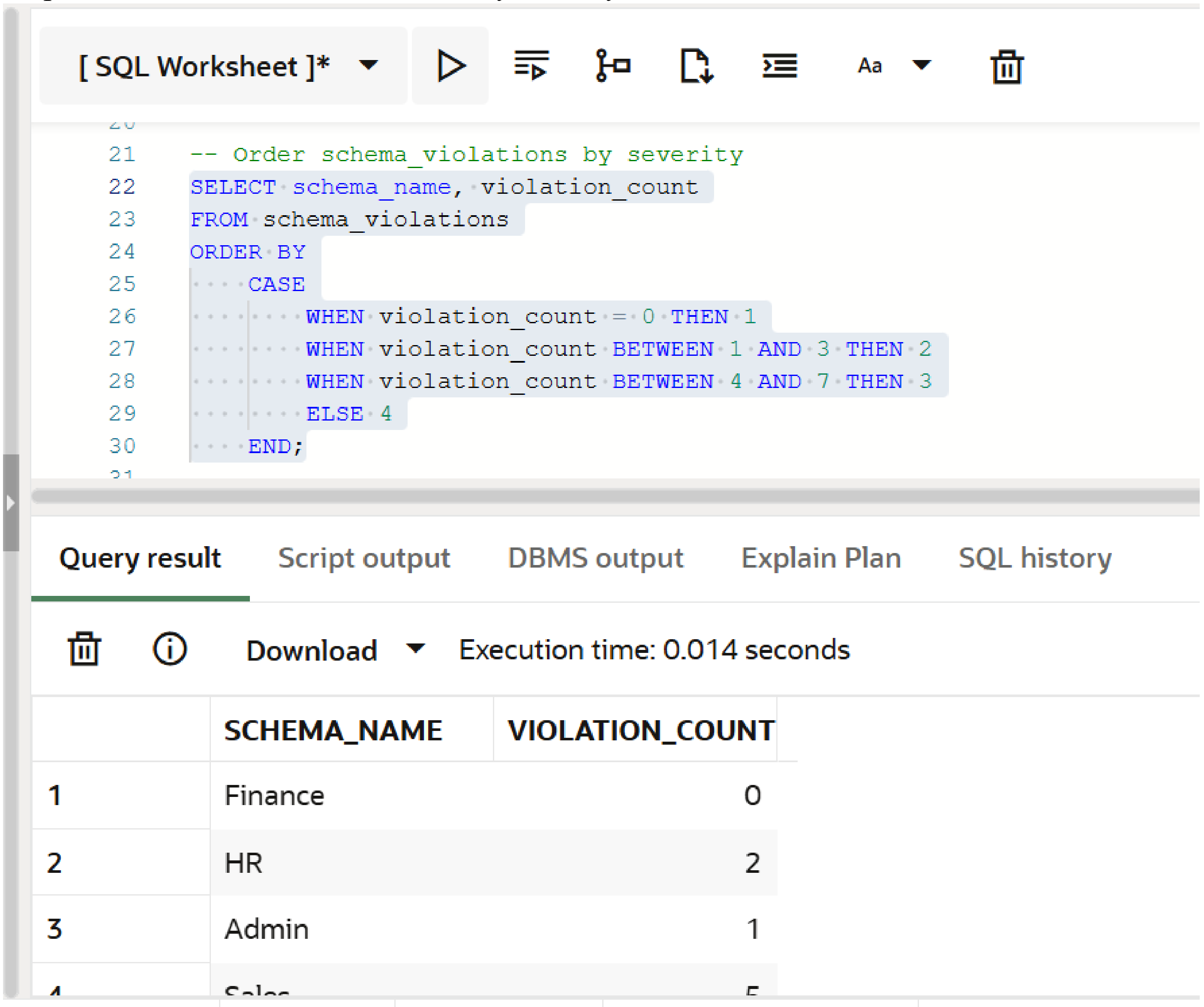
Sam 75 B Grade

Sahil 61 C Grade

Pranav 48 Fail

EXPERIMENT 4

Step 11 – Schema violations ordered by severity



```sql
-- Order schema_violations by severity
SELECT schema_name, violation_count
FROM schema_violations
ORDER BY
    CASE
        WHEN violation_count = 0 THEN 1
        WHEN violation_count BETWEEN 1 AND 3 THEN 2
        WHEN violation_count BETWEEN 4 AND 7 THEN 3
        ELSE 4
    END;
```

	SCHEMA_NAME	VIOLATION_COUNT
1	Finance	0
2	HR	2
3	Admin	1
4	Sales	5

schema_name	violation_count
Finance	0
HR	2
Admin	1
Sales	5
Security	9

EXPERIMENT 4

Learning Outcome

After completing this experiment, the student will be able to:

- Create and populate tables in Oracle SQL.
- Use CASE statements to evaluate conditions in queries.
- Update table data based on conditional logic.
- Write PL/SQL blocks for dynamic status messages.
- Sort query results using CASE statements in ORDER BY.
- Analyze data and assign grades or approval statuses automatically.
- Interpret step-wise outputs for better understanding of database operations.