

Semester	T.E. Semester VI Div C Batch – CMPN
Subject	ADBMS Lab
Subject Professor Incharge	Prof. Divya Nimbalkar
Assisting Teachers	Prof. Divya Nimbalkar

Student Name	Nilesh Patni
Roll Number	24102C2001
Grade and Subject Teacher's Signature	

Experiment Number	03
Experiment Title	Implementation of Role-Based Access Control (RBAC) with Row Level Security
Objectives (Skill Set / Knowledge Tested / Imparted)	<ul style="list-style-type: none"> • To secure the database of the Crowdsourced Traffic Incident Reporting System by implementing Role-Based Access Control (RBAC) and Row-Level Security (RLS) so that different users access data according to their roles.

Aim: To create database roles, assign appropriate privileges, test role-based access by logging in as different users, and implement row-level security for sensitive data in the Crowdsourced Traffic Incident Reporting System.

Theory:

Role-Based Access Control (RBAC): -

Role-Based Access Control is a database security mechanism where permissions are assigned to roles instead of individual users. Users inherit privileges by being assigned a role. This simplifies security management and ensures controlled access to data.

Row-Level Security (RLS): -

Row-Level Security restricts access to specific rows in a table based on user identity or role. It ensures that users can view or modify only the data that belongs to them.

Roles in Crowdsourced Traffic Incident Reporting System: - • Admin: Manages users, incidents, and system data.

- Authority: Verifies incidents and updates incident status.
- User: Reports incidents and views only their own data.

Sensitive tables such as Incident and Feedback require restricted access to maintain data privacy and integrity.

Lab Outcome Mapped: CO

CO Statement: Students will be able to design and implement database security using roles, privileges, and row-level security

IMPLEMENTATION :

Create Tables (Database Schema)

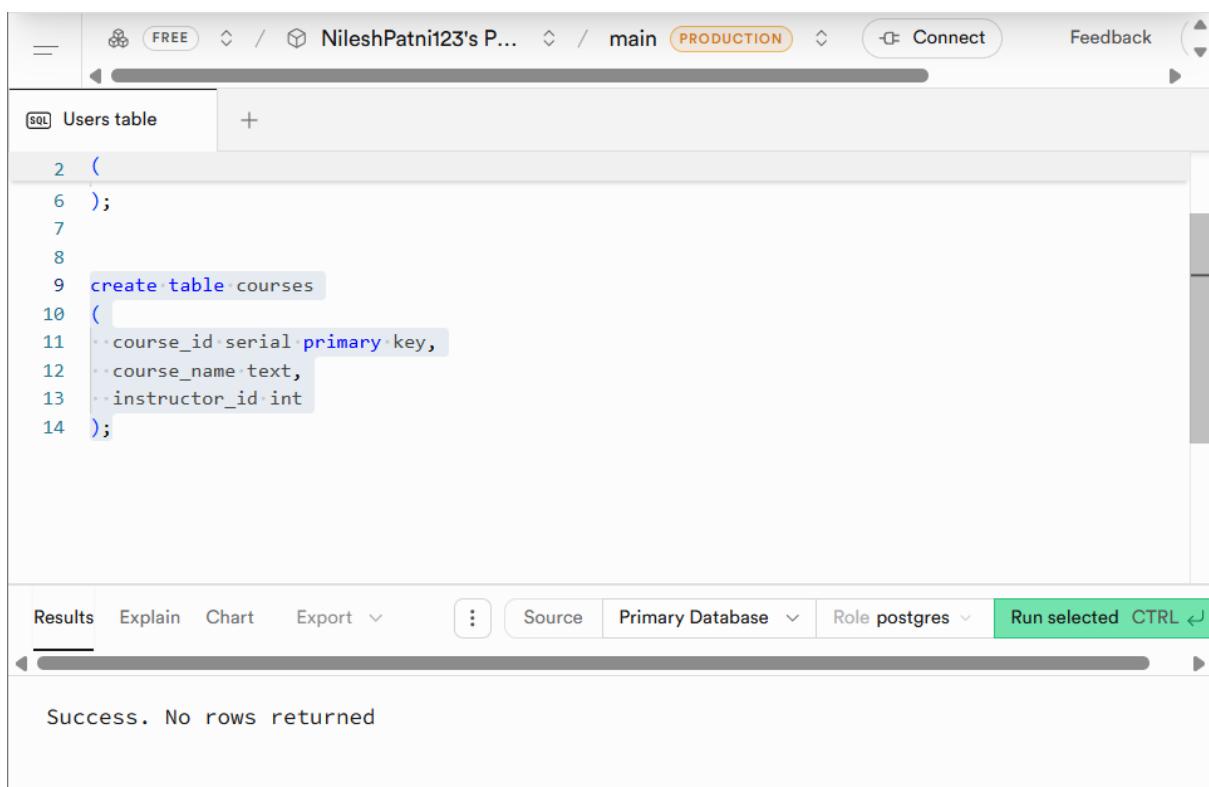
-- Users table :

The screenshot shows a PostgreSQL database interface. The top bar includes a 'FREE' badge, the user 'NileshPatni123's profile, the database 'main' (set to 'PRODUCTION'), a 'Connect' button, and a 'Feedback' link. The main area has a tab for 'Users table'. Below it is a code editor containing the following SQL:

```
1 create table users
2 (
3     user_id serial primary key,
4     username text,
5     role text
6 );
```

Below the code editor are tabs for 'Results', 'Explain', 'Chart', and 'Export'. The 'Run selected' button is highlighted in green. The results pane displays the message: "Success. No rows returned".

-- Courses table :



The screenshot shows a PostgreSQL database interface. In the top navigation bar, the database is identified as 'NileshPatni123's P...' and the schema is 'main'. The status bar indicates 'PRODUCTION'. Below the navigation, there is a toolbar with 'Results', 'Explain', 'Chart', 'Export', and other options. The main area displays the following SQL code:

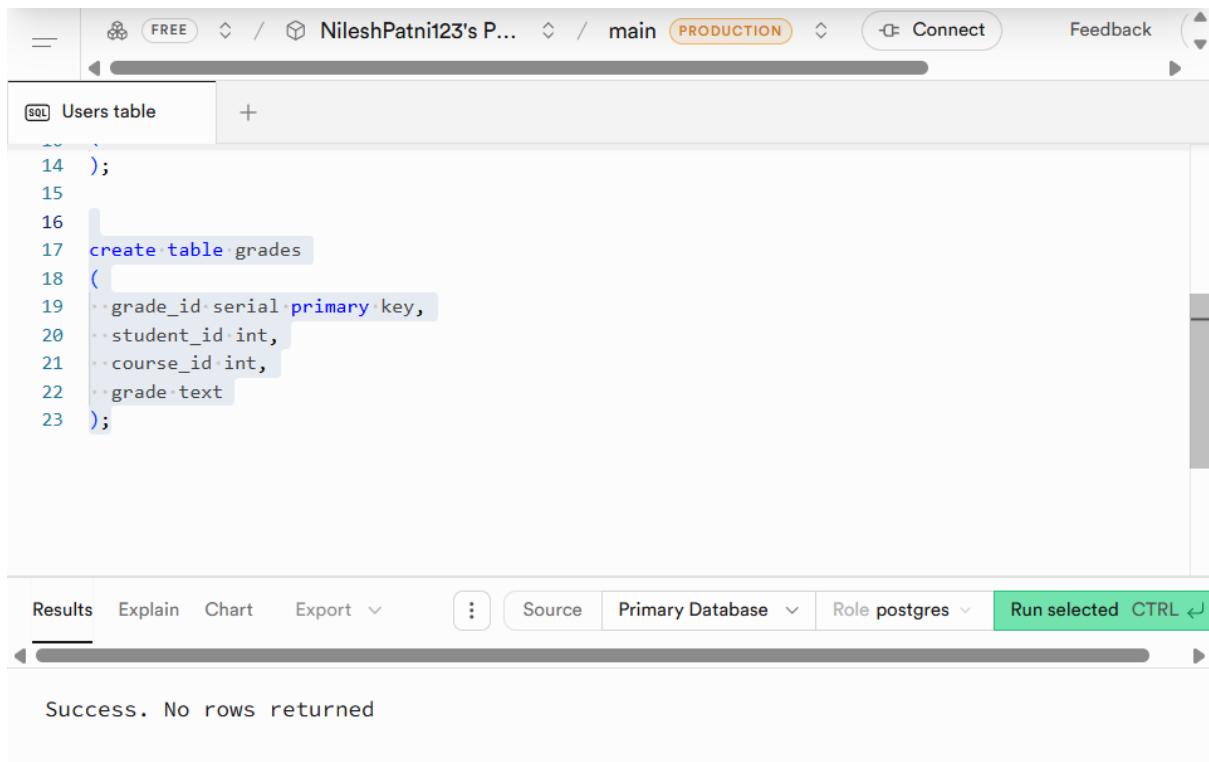
```

2  (
6  );
7
8
9 create table courses
10 (
11   course_id serial primary key,
12   course_name text,
13   instructor_id int
14 );

```

At the bottom of the interface, the status bar shows 'Success. No rows returned'.

-- Grades table (Sensitive) :



The screenshot shows a PostgreSQL database interface. In the top navigation bar, the database is identified as 'NileshPatni123's P...' and the schema is 'main'. The status bar indicates 'PRODUCTION'. Below the navigation, there is a toolbar with 'Results', 'Explain', 'Chart', 'Export', and other options. The main area displays the following SQL code:

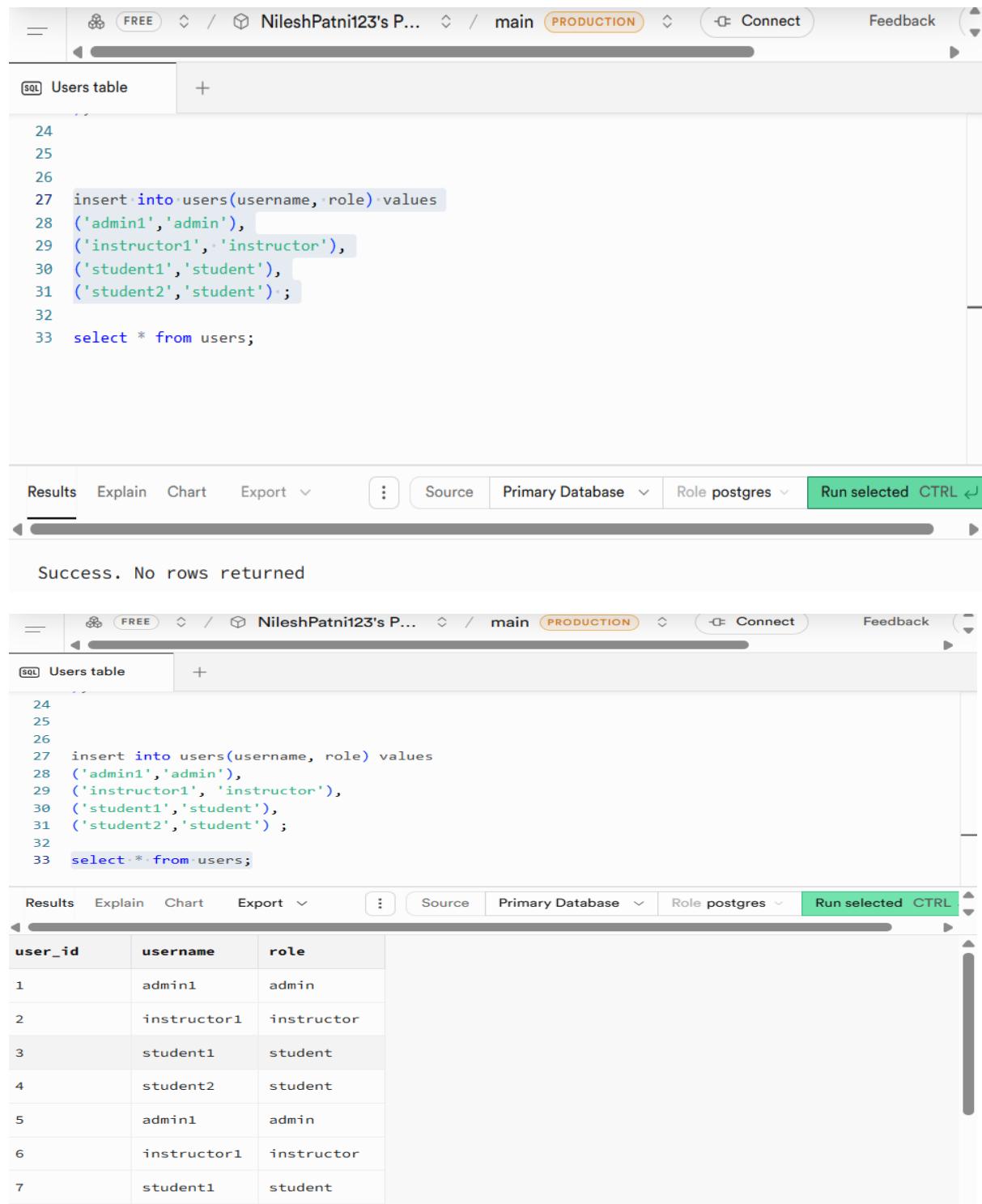
```

14 );
15
16
17 create table grades
18 (
19   grade_id serial primary key,
20   student_id int,
21   course_id int,
22   grade text
23 );

```

At the bottom of the interface, the status bar shows 'Success. No rows returned'.

Insert Sample Data :



The screenshot shows a PostgreSQL database interface with two main sections: a SQL editor at the top and a results viewer at the bottom.

SQL Editor (Top Section):

```

24
25
26
27 insert into users(username, role) values
28 ('admin1','admin'),
29 ('instructor1', 'instructor'),
30 ('student1','student'),
31 ('student2','student') ;
32
33 select * from users;

```

Results Viewer (Bottom Section):

Success. No rows returned

The results viewer shows the output of the last query:

user_id	username	role
1	admin1	admin
2	instructor1	instructor
3	student1	student
4	student2	student
5	admin1	admin
6	instructor1	instructor
7	student1	student

The screenshot shows a PostgreSQL database interface. The top bar includes the VIT logo, the text "NileshPatni123's P...", "main PRODUCTION", "Connect", and "Feedback". The main area is titled "Users table" and contains the following SQL code:

```
33 select * from users;
34
35
36
37 insert into courses(course_name, instructor_id)
38 values ('Machine Learning',2);
```

Below the code, the results pane displays the message: "Success. No rows returned".

The screenshot shows a PostgreSQL database interface. The top bar includes the VIT logo, the text "NileshPatni123's P...", "main PRODUCTION", "Connect", and "Feedback". The main area is titled "Users table" and contains the following SQL code:

```
38 values ('Machine Learning',2);
39
40
41 insert into grades(student_id, course_id, grade) values
42 (3,1,'A'),
43 (4,1,'B');
```

Below the code, the results pane displays the message: "Success. No rows returned".

Create Roles (RBAC) :

```

43 (4,1,'B') ;
44
45
46 create role admin_role ;
47 create role instructor_role ;
48 create role student_role ;

```

Results Explain Chart Export ⋮ Source Primary Database ▼ Role postgres ▼ Run selected CTRL ⌘

Error: Failed to run sql query: ERROR: 42710: role "admin_role" already exists Debug with Assistant

Grant Privileges to Roles :

Admin – Full access

```

46 create role admin_role ;
47 create role instructor_role ;
48 create role student_role ;
49
50
51 grant all privileges on all tables in schema public to admin_role;

```

Results Explain Chart Export ⋮ Source Primary Database ▼ Role postgres ▼ Run selected CTRL ⌘

Success. No rows returned

Instructor – View courses, update grades

```
52
53
54
55 grant select on courses to instructor_role;
56 grant select, update on grades to instructor_role;
```

Results Explain Chart Export ⋮ Source Primary Database ▼ Role postgres ▼ Run selected CTRL ↵

Success. No rows returned

Student – Read-only access

```
58
59
60 grant select on courses to student_role;
61 grant select on grades to student_role;
```

Results Explain Chart Export ⋮ Source Primary Database ▼ Role postgres ▼ Run selected CTRL ↵

Success. No rows returned

Test Role-Based Access Control :

Instructor Test :

```
64 SET ROLE instructor_role;
65 SELECT * FROM grades;
66 UPDATE grades SET grade = 'A+' WHERE student_id = 3;
67
```

Results Explain Chart Export ⋮ Source Primary Database ▼ Role postgres ▼ Run selected CTRL ↵

Error: Failed to run sql query: ERROR: 42501: permission denied to set role "instructor_role"

Debug with Assistant

```
--  
64  
65 SET ROLE instructor_role;  
66 SELECT * FROM grades;  
67 UPDATE grades SET grade = 'A+' WHERE student_id = 3;  
68
```

grade_id	student_id	course_id	grade
1	3	1	A
2	4	1	B

```
--  
64  
65 SET ROLE instructor_role;  
66 SELECT * FROM grades;  
67 UPDATE grades SET grade = 'A+' WHERE student_id = 3;  
68
```

Success. No rows returned

Student Test :

```
--  
72  
73 set role student_role;  
74 select * from courses;
```

Error: Failed to fetch (api.supabase.com)

Debug with Assistant

```
67
70  delete from grades ;
71
72
73  set role student_role;
74  select * from courses;
```

The screenshot shows a PostgreSQL terminal window with the following interface elements at the top: Results, Explain, Chart, Export, Source, Primary Database (set to Postgres), Role postgres, and Run selected (highlighted in green). Below these are scroll bars. The main area displays the results of a query:

course_id	course_name	instructor_id
1	Machine Learning	2

```
76
77 UPDATE grades SET grade='A' WHERE student_id=3;
78 select * from grades;
79
```

The screenshot shows a PostgreSQL terminal window with the same interface elements as the previous one. The main area displays the results of the UPDATE query followed by the SELECT query:

Success. No rows returned

Enable Row-Level Security (RLS) :

```
80
81
82 ALTER TABLE grades ENABLE ROW LEVEL SECURITY;
83
```

The screenshot shows a PostgreSQL terminal window with the same interface elements. The main area displays the results of the ALTER TABLE command:

Success. No rows returned

Student-Specific Grade Access (RLS Policy) :

Students can see only their own grades

```
83
84
85 CREATE POLICY student_grade_policy
86 ON grades
87 FOR SELECT
88 TO student_role
89 USING (student_id = current_setting('app.current_student')::INT);
90
```

Results Explain Chart Export ▾ Source Primary Database ▾ Role postgres ▾ Run selected CTRL ↵

Success. No rows returned

Set current student context:

```
91
92
93 SET app.current_student = '3';
94
```

Results Explain Chart Export ▾ Source Primary Database ▾ Role postgres ▾ Run selected CTRL ↵

Success. No rows returned

Student sees only his/her row :

```
95
96 SET ROLE student_role;
97 SELECT * FROM grades;
98
```

Results Explain Chart Export ▾ Source Primary Database ▾ Role postgres ▾ Run selected CTRL ↵

Success. No rows returned

Instructor RLS Policy (Full Access) :

```

100
101 CREATE POLICY instructor_grade_policy
102 ON grades
103 FOR ALL
104 TO instructor_role
105 USING (true);
106

```

Results Explain Chart Export ▾ Source Primary Database ▾ Role postgres ▾ Run selected CTRL ↵

Success. No rows returned

```

108 insert into grades(student_id, course_id, grade) values
109 (3,1,'A'),
110 (4,1,'B') ;
111 select * from grades;
112
113 UPDATE grades SET grade='A' WHERE student_id=3;
114 select * from grades;

```

Results Explain Chart Export ▾ Source Primary Database ▾ Role postgres ▾ Run selected CTRL ↵

grade_id	student_id	course_id	grade
4	4	1	B
6	4	1	B
3	3	1	A
5	3	1	A

Conclusion :-

In this experiment, Role-Based Access Control (RBAC) and Row-Level Security (RLS) were successfully implemented for the Crowdsourced Traffic Incident Reporting System. Different roles were created and assigned appropriate privileges to control access to sensitive data. Row-level security ensured that users could access only their own incident and feedback records. This experiment demonstrates effective database security implementation and highlights the importance of access control in real-world database applications.

----- END -----