

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.

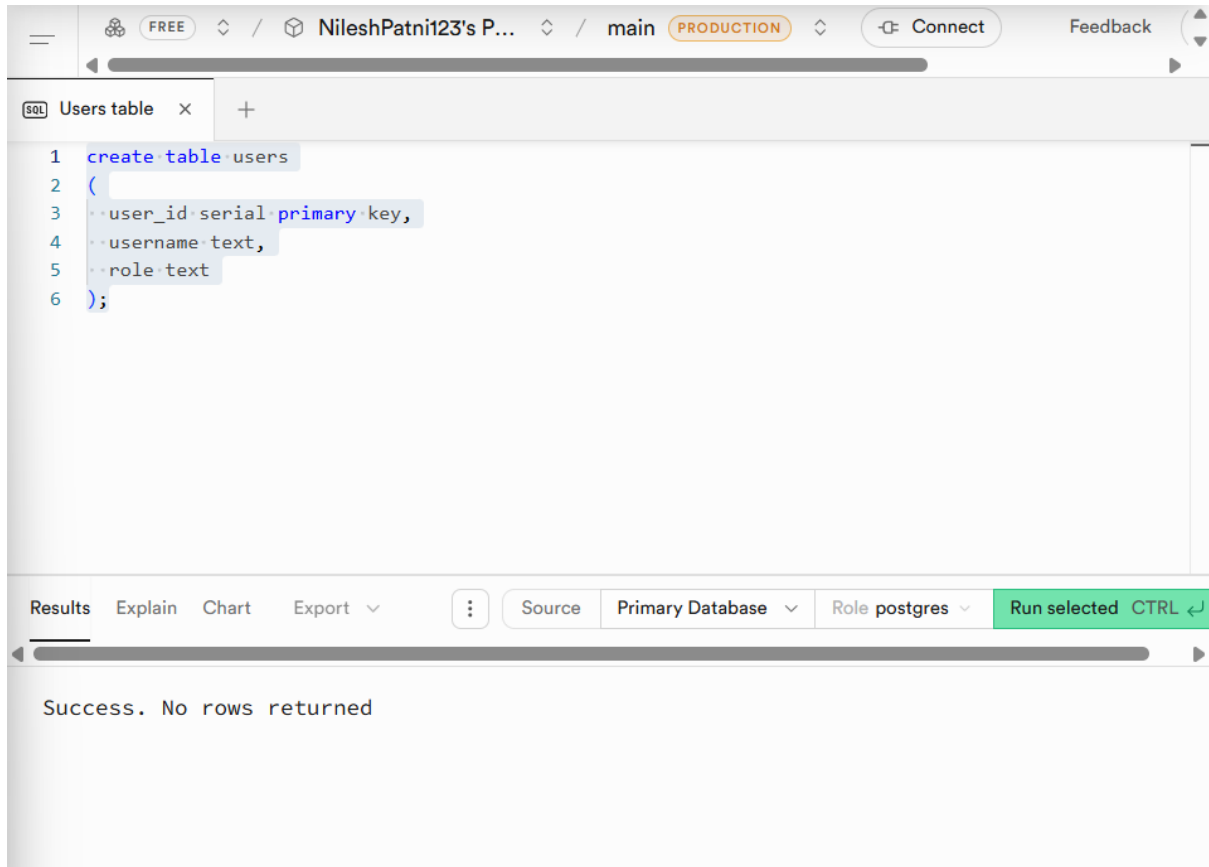
Lab Outcome Mapped: CO

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 database management interface. At the top, there's a navigation bar with a 'FREE' label, a user profile 'NileshPatni123's P...', a database 'main', and a status 'PRODUCTION'. A 'Connect' button and a 'Feedback' link are also present. Below the navigation bar, there's a tab labeled 'SQL Users table'. The main area contains SQL code for creating a table named 'users'. The code is as follows:

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

At the bottom of the interface, there's a toolbar with buttons for 'Results', 'Explain', 'Chart', 'Export', and a 'Run selected' button (highlighted in green). Below the toolbar, the status 'Success. No rows returned' is displayed.

-- Courses table :

The screenshot shows a database management interface with a top bar containing 'FREE', 'NileshPatni123's P...', 'main', 'PRODUCTION', 'Connect', and 'Feedback'. Below the bar, a tab labeled 'SQL Users table' is active. The SQL editor contains the following 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 );

```

Below the editor, a toolbar includes 'Results', 'Explain', 'Chart', 'Export', a menu icon, 'Source', 'Primary Database', 'Role postgres', and a green 'Run selected CTRL ↵' button. The bottom status bar displays 'Success. No rows returned'.

-- Grades table (Sensitive) :

The screenshot shows the same database management interface. The 'SQL Users table' tab is active, and the SQL editor contains the following 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 );

```

The toolbar and status bar are identical to the previous screenshot, showing 'Success. No rows returned'.

Insert Sample Data :

FREE

NileshPatni123's P...

main

PRODUCTION

Connect

Feedback

SQL Users table

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

Explain

Chart

Export

Source

Primary Database

Role postgres

Run selected CTRL

Success. No rows returned

FREE

NileshPatni123's P...

main

PRODUCTION

Connect

Feedback

SQL Users table

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

Explain

Chart

Export

Source

Primary Database

Role postgres

Run selected CTRL

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 database management tool interface. At the top, there's a header with a logo, a username 'NileshPatni123's P...', a database name 'main', and a status 'PRODUCTION'. Below the header, there's a tab labeled 'Users table'. The main area contains a SQL query:

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

Below the query, there's a row of buttons: 'Results', 'Explain', 'Chart', 'Export', and a dropdown menu. To the right of these buttons are 'Source', 'Primary Database', 'Role postgres', and a green button 'Run selected CTRL ↵'. The bottom section shows the execution result: 'Success. No rows returned'.



The screenshot shows the same database management tool interface. The SQL query is now:

```
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');
```

The execution result is still 'Success. No rows returned'.

Create Roles (RBAC) :

The screenshot shows a database client interface with a SQL editor. The query being executed is:

```

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

```

The interface shows a toolbar with options like 'Results', 'Explain', 'Chart', 'Export', 'Source', 'Primary Database', 'Role postgres', and a 'Run selected' button. Below the editor, an error message is displayed:

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

A 'Debug with Assistant' button is visible next to the error message.

Grant Privileges to Roles :

Admin – Full access

The screenshot shows the same database client interface. The query being executed is:

```

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;

```

The interface shows the same toolbar as the previous screenshot. Below the editor, a success message is displayed:

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

```

57
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
65 SET ROLE instructor_role;
66 SELECT * FROM grades;
67 UPDATE grades SET grade = 'A+' WHERE student_id = 3;
68

```

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

```

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

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

```

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

Success. No rows returned

Student Test :

```

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

```

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

Error: Failed to fetch (api.supabase.com) [Debug with Assistant](#)

```

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

```

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

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

```

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

Success. No rows returned

Enable Row-Level Security (RLS) :

```

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

```

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

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 -----