Royal University of Bhutan

འབྲུག་རྒྱལ་འཛིན་གཙུག་ལག་སློབ་སྡེ།།

College of Science and Technology
Rinchending: Bhutan

DBS101
Database Systems Fundamentals
SS(2024)

Practical{4}   Report

Submitted By;

Student Name : Tashi Penjor

Enrollment No.: 02230306

Programme : BESWE

Date : 17/03/2024

Royal University of Bhutan

འབྲུག་རྒྱལ་འཛིན་གཙུག་ལག་སློབ་སྡེ།

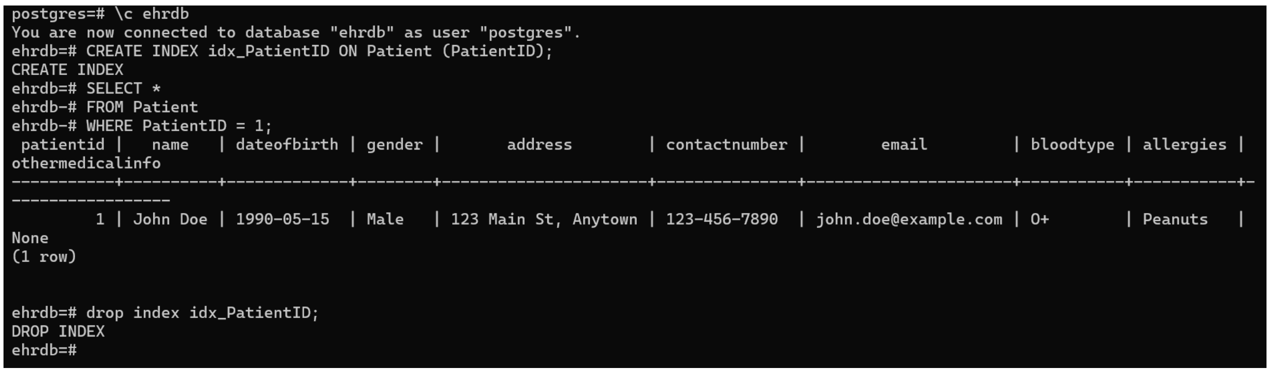
College of Science and Technology

Rinchending: Bhutan

Table of content

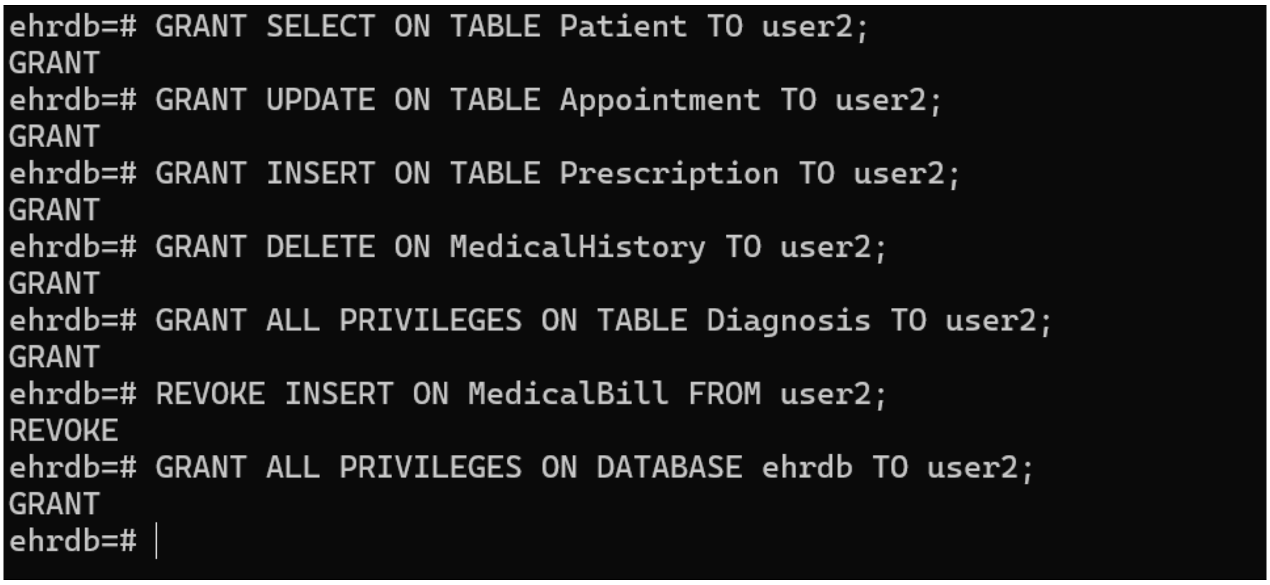
Topic : Guided Session

Task 1 : Created index on patient column with index patientID and selected patient with ID 1 and deleted the index.



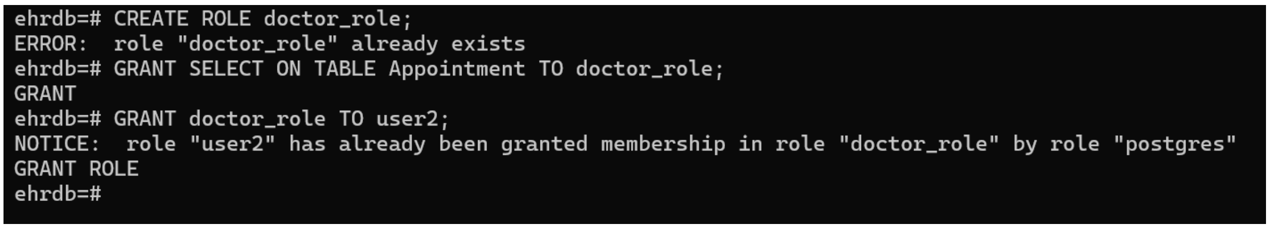
```
postgres=# \c ehrdb
You are now connected to database "ehrdb" as user "postgres".
ehrdb=# CREATE INDEX idx_PatientID ON Patient (PatientID);
CREATE INDEX
ehrdb=# SELECT *
ehrdb-# FROM Patient
ehrdb-# WHERE PatientID = 1;
 patientid |   name   | dateofbirth | gender |       address        | contactnumber |        email         | bloodtype | allergies |
othermedicalinfo
-----------+----------+-------------+--------+----------------------+---------------+----------------------+-----------+-----------+-
-----------------
         1 | John Doe | 1990-05-15  | Male   | 123 Main St, Anytown | 123-456-7890  | john.doe@example.com | O+        | Peanuts   |
None
(1 row)


ehrdb=# drop index idx_PatientID;
DROP INDEX
ehrdb=#
```

Task 2 : created the user2 and granted some privileges to the user 2.



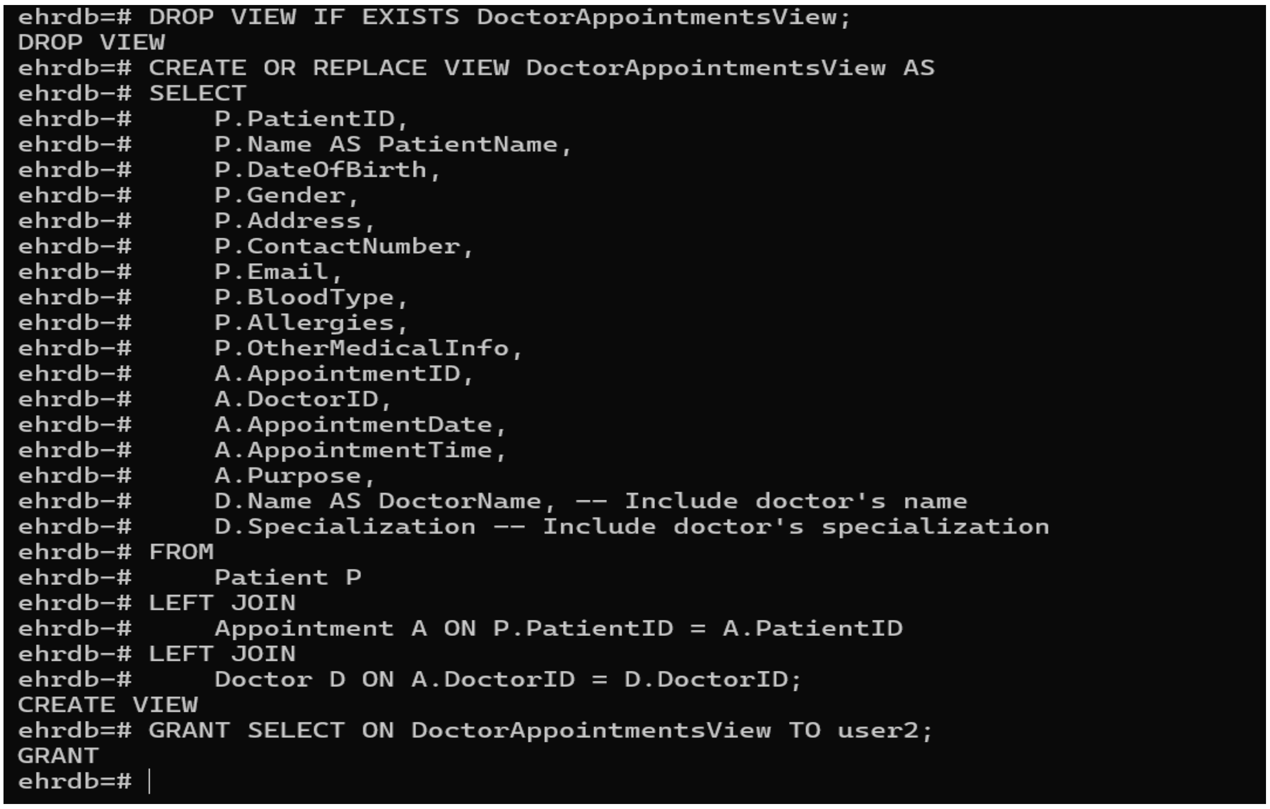
```
ehrdb=# GRANT SELECT ON TABLE Patient TO user2;
GRANT
ehrdb=# GRANT UPDATE ON TABLE Appointment TO user2;
GRANT
ehrdb=# GRANT INSERT ON TABLE Prescription TO user2;
GRANT
ehrdb=# GRANT DELETE ON MedicalHistory TO user2;
GRANT
ehrdb=# GRANT ALL PRIVILEGES ON TABLE Diagnosis TO user2;
GRANT
ehrdb=# REVOKE INSERT ON MedicalBill FROM user2;
REVOKE
ehrdb=# GRANT ALL PRIVILEGES ON DATABASE ehrdb TO user2;
GRANT
ehrdb=# |
```

Task 3 : created the role of doctor but the role already existed so granted few privileges.



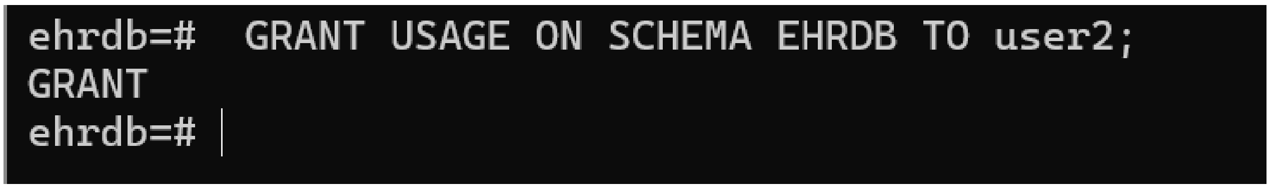
```
ehrdb=# CREATE ROLE doctor_role;
ERROR:  role "doctor_role" already exists
ehrdb=# GRANT SELECT ON TABLE Appointment TO doctor_role;
GRANT
ehrdb=# GRANT doctor_role TO user2;
NOTICE:  role "user2" has already been granted membership in role "doctor_role" by role "postgres"
GRANT ROLE
ehrdb=#
```

TASK 4 : to see if the doctor’s appointment is correctly structured, first I dropped the doctor’s appointment view if it exists and created the new doctor's appointment view using left join. Granted the access to user 2.

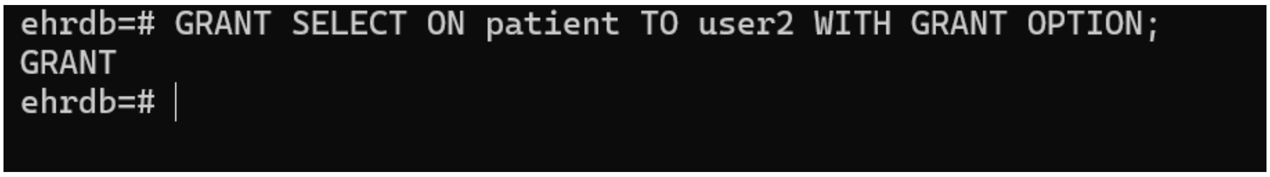


```
ehrdb=# DROP VIEW IF EXISTS DoctorAppointmentsView;
DROP VIEW
ehrdb=# CREATE OR REPLACE VIEW DoctorAppointmentsView AS
ehrdb-# SELECT
ehrdb-#     P.PatientID,
ehrdb-#     P.Name AS PatientName,
ehrdb-#     P.DateOfBirth,
ehrdb-#     P.Gender,
ehrdb-#     P.Address,
ehrdb-#     P.ContactNumber,
ehrdb-#     P.Email,
ehrdb-#     P.BloodType,
ehrdb-#     P.Allergies,
ehrdb-#     P.OtherMedicalInfo,
ehrdb-#     A.AppointmentID,
ehrdb-#     A.DoctorID,
ehrdb-#     A.AppointmentDate,
ehrdb-#     A.AppointmentTime,
ehrdb-#     A.Purpose,
ehrdb-#     D.Name AS DoctorName, -- Include doctor's name
ehrdb-#     D.Specialization -- Include doctor's specialization
ehrdb-# FROM
ehrdb-#     Patient P
ehrdb-# LEFT JOIN
ehrdb-#     Appointment A ON P.PatientID = A.PatientID
ehrdb-# LEFT JOIN
ehrdb-#     Doctor D ON A.DoctorID = D.DoctorID;
CREATE VIEW
ehrdb=# GRANT SELECT ON DoctorAppointmentsView TO user2;
GRANT
ehrdb=# |
```

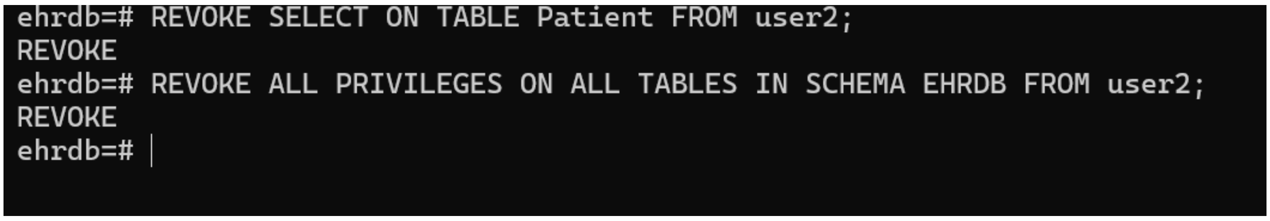
Task 5 : granted authorization schema to user 2


```
ehrdb=#  GRANT USAGE ON SCHEMA EHRDB TO user2;
GRANT
ehrdb=#
```

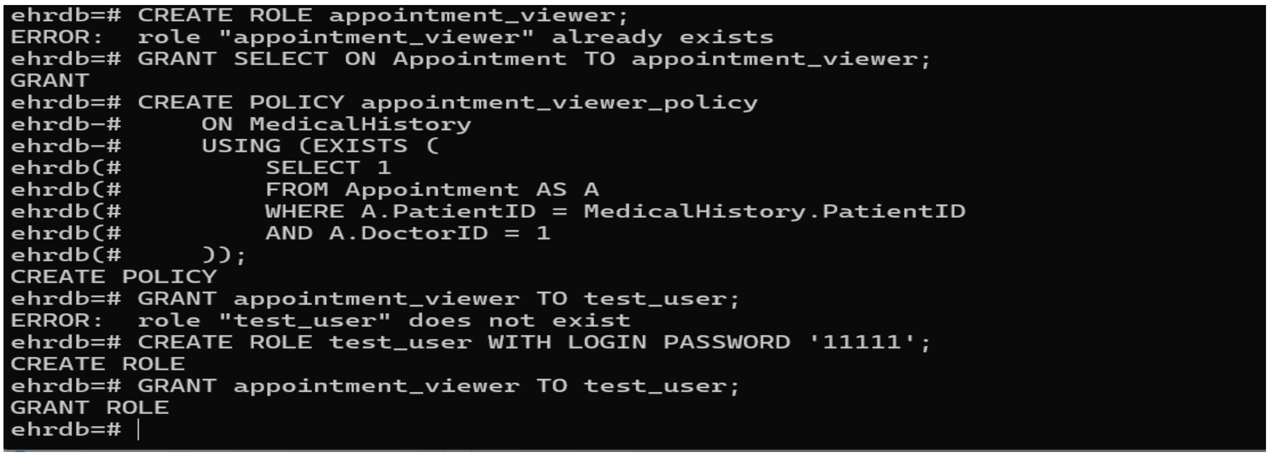
Task 6 : transferring the privilege to user 2 from patient


```
ehrdb=# GRANT SELECT ON patient TO user2 WITH GRANT OPTION;
GRANT
ehrdb=#
```

Task 7 : revoking all the privileges from the user 2 that has been granted to user 2

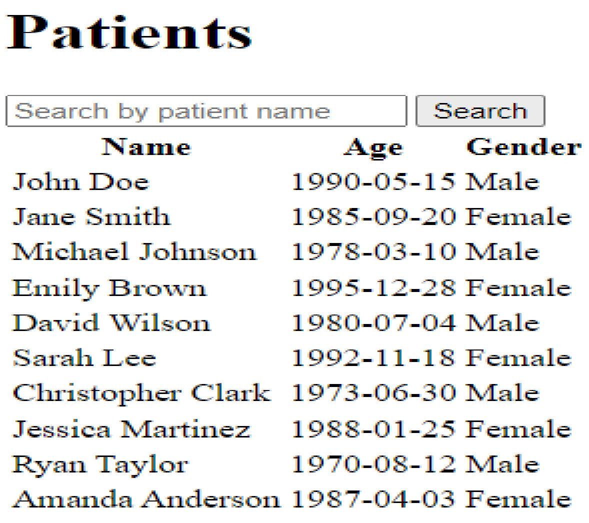

```
ehrdb=# REVOKE SELECT ON TABLE Patient FROM user2;
REVOKE
ehrdb=# REVOKE ALL PRIVILEGES ON ALL TABLES IN SCHEMA EHRDB FROM user2;
REVOKE
ehrdb=#
```

Task 8 : I tried to create the role of appointment viewer but it already exists, so grant a role to view the appointment and policy creator. Since the role of test user doesn't exist. I created the role and granted the test user to view the appointment.



```
ehrdb=# CREATE ROLE appointment_viewer;
ERROR:  role "appointment_viewer" already exists
ehrdb=# GRANT SELECT ON Appointment TO appointment_viewer;
GRANT
ehrdb=# CREATE POLICY appointment_viewer_policy
ehrdb-#     ON MedicalHistory
ehrdb-#     USING (EXISTS (
ehrdb(#         SELECT 1
ehrdb(#         FROM Appointment AS A
ehrdb(#         WHERE A.PatientID = MedicalHistory.PatientID
ehrdb(#         AND A.DoctorID = 1
ehrdb(#     ));
CREATE POLICY
ehrdb=# GRANT appointment_viewer TO test_user;
ERROR:  role "test_user" does not exist
ehrdb=# CREATE ROLE test_user WITH LOGIN PASSWORD '11111';
CREATE ROLE
ehrdb=# GRANT appointment_viewer TO test_user;
GRANT ROLE
ehrdb=# |
```

Task 9 : Accessing the SQL from a programming language. We need to connect the database to the server and publish the database.



Patients

Name	Age	Gender
John Doe	1990-05-15	Male
Jane Smith	1985-09-20	Female
Michael Johnson	1978-03-10	Male
Emily Brown	1995-12-28	Female
David Wilson	1980-07-04	Male
Sarah Lee	1992-11-18	Female
Christopher Clark	1973-06-30	Male
Jessica Martinez	1988-01-25	Female
Ryan Taylor	1970-08-12	Male
Amanda Anderson	1987-04-03	Female

Royal University of Bhutan

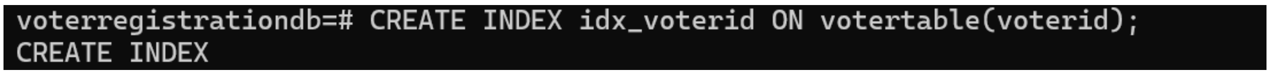
འབྲུག་རྒྱལ་འཛིན་གཙུག་ལག་སློབ་སྡེ།།

College of Science and Technology

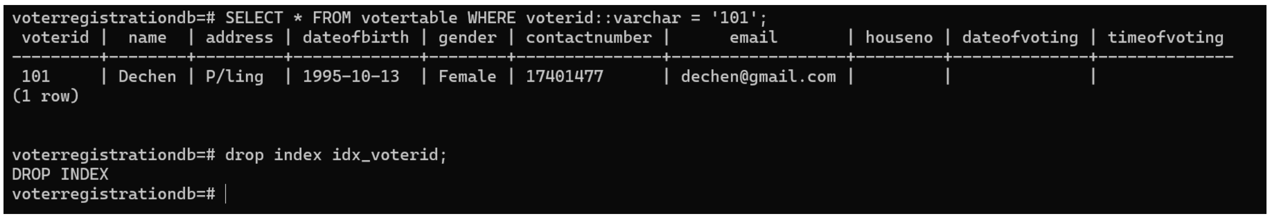
Rinchending: Bhutan

Topic : Practical 4

Task 1 : to find the specific value in the database I created an index on the voter table and selected a person having a voter id 101 and dropped the index.

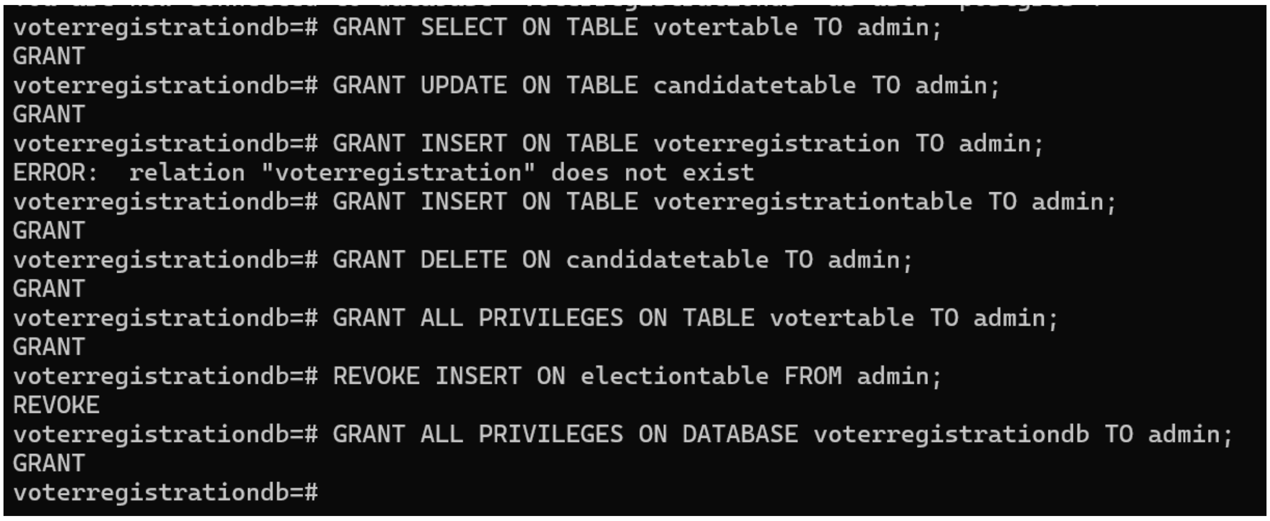


```
voterregistrationdb=# CREATE INDEX idx_voterid ON votertable(voterid);
CREATE INDEX
```



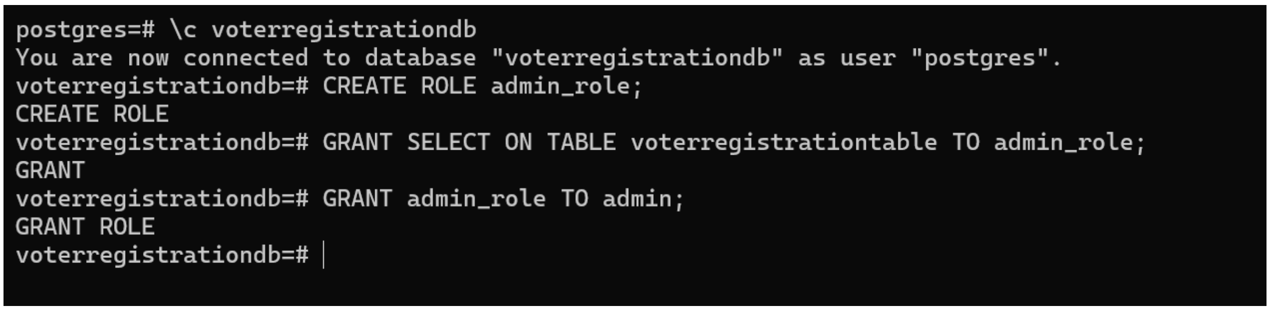
```
voterregistrationdb=# SELECT * FROM votertable WHERE voterid::varchar = '101';
 voterid |  name  | address | dateofbirth | gender | contactnumber |       email       | houseno | dateofvoting | timeofvoting
---------+--------+---------+-------------+--------+---------------+-------------------+---------+--------------+--------------
 101     | Dechen | P/ling  | 1995-10-13  | Female | 17401477      | dechen@gmail.com |         |              |
(1 row)


voterregistrationdb=# drop index idx_voterid;
DROP INDEX
voterregistrationdb=# |
```

Task 2 : I granted some privileges to the admin in the voter registration database



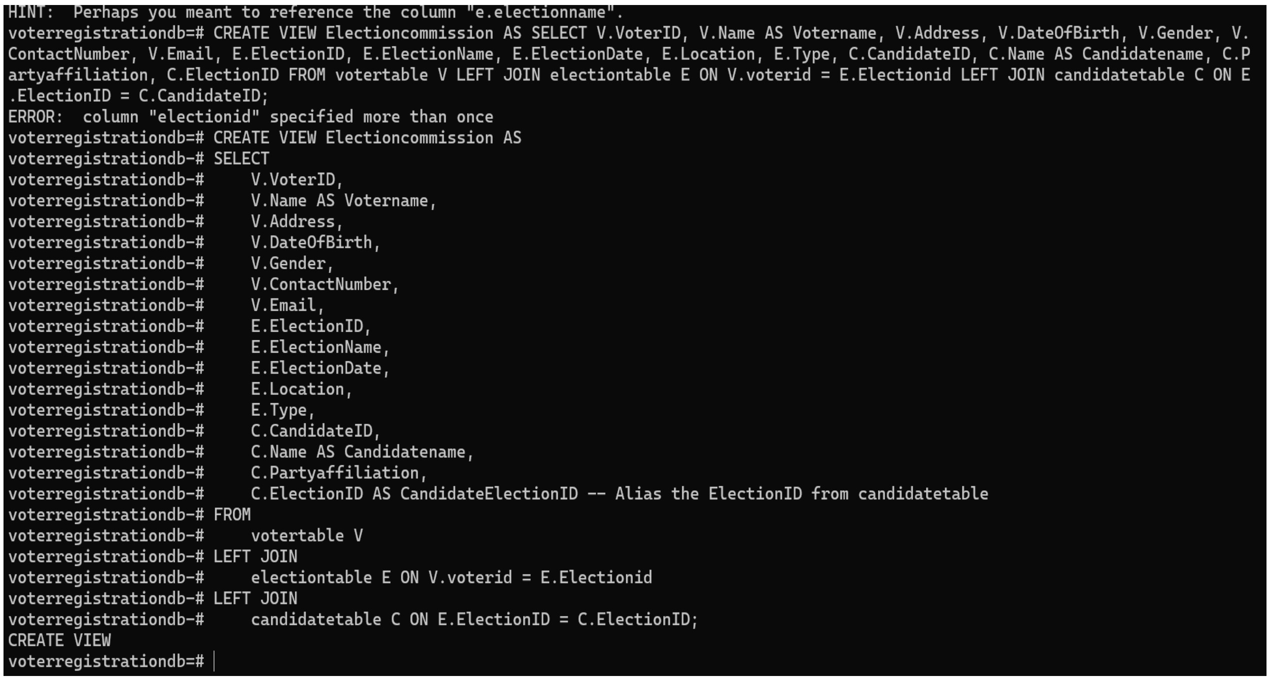
```
voterregistrationdb=# GRANT SELECT ON TABLE votertable TO admin;
GRANT
voterregistrationdb=# GRANT UPDATE ON TABLE candidatetable TO admin;
GRANT
voterregistrationdb=# GRANT INSERT ON TABLE voterregistration TO admin;
ERROR:  relation "voterregistration" does not exist
voterregistrationdb=# GRANT INSERT ON TABLE voterregistrationtable TO admin;
GRANT
voterregistrationdb=# GRANT DELETE ON candidatetable TO admin;
GRANT
voterregistrationdb=# GRANT ALL PRIVILEGES ON TABLE votertable TO admin;
GRANT
voterregistrationdb=# REVOKE INSERT ON electiontable FROM admin;
REVOKE
voterregistrationdb=# GRANT ALL PRIVILEGES ON DATABASE voterregistrationdb TO admin;
GRANT
voterregistrationdb=#
```

Task 3 : created an admin role and granted all privileges to admin role.

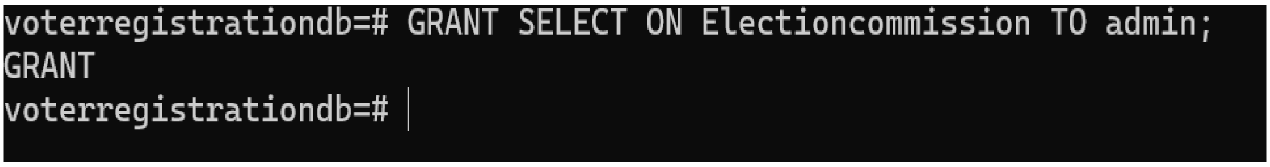


```
postgres=# \c voterregistrationdb
You are now connected to database "voterregistrationdb" as user "postgres".
voterregistrationdb=# CREATE ROLE admin_role;
CREATE ROLE
voterregistrationdb=# GRANT SELECT ON TABLE voterregistrationtable TO admin_role;
GRANT
voterregistrationdb=# GRANT admin_role TO admin;
GRANT ROLE
voterregistrationdb=# |
```

Task 4 : created a authorisation view as election commission by left joing the voter table, election table and candidate table and granting the view to admin.

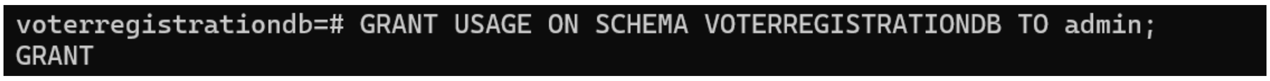


```
HINT:  Perhaps you meant to reference the column "e.electionname".
voterregistrationdb=# CREATE VIEW Electioncommission AS SELECT V.VoterID, V.Name AS Votername, V.Address, V.DateOfBirth, V.Gender, V.
ContactNumber, V.Email, E.ElectionID, E.ElectionName, E.ElectionDate, E.Location, E.Type, C.CandidateID, C.Name AS Candidatename, C.P
artyaffiliation, C.ElectionID FROM votertable V LEFT JOIN electiontable E ON V.voterid = E.Electionid LEFT JOIN candidatetable C ON E
.ElectionID = C.CandidateID;
ERROR:  column "electionid" specified more than once
voterregistrationdb=# CREATE VIEW Electioncommission AS
voterregistrationdb-# SELECT
voterregistrationdb-#     V.VoterID,
voterregistrationdb-#     V.Name AS Votername,
voterregistrationdb-#     V.Address,
voterregistrationdb-#     V.DateOfBirth,
voterregistrationdb-#     V.Gender,
voterregistrationdb-#     V.ContactNumber,
voterregistrationdb-#     V.Email,
voterregistrationdb-#     E.ElectionID,
voterregistrationdb-#     E.ElectionName,
voterregistrationdb-#     E.ElectionDate,
voterregistrationdb-#     E.Location,
voterregistrationdb-#     E.Type,
voterregistrationdb-#     C.CandidateID,
voterregistrationdb-#     C.Name AS Candidatename,
voterregistrationdb-#     C.Partyaffiliation,
voterregistrationdb-#     C.ElectionID AS CandidateElectionID -- Alias the ElectionID from candidatetable
voterregistrationdb-# FROM
voterregistrationdb-#     votertable V
voterregistrationdb-# LEFT JOIN
voterregistrationdb-#     electiontable E ON V.voterid = E.Electionid
voterregistrationdb-# LEFT JOIN
voterregistrationdb-#     candidatetable C ON E.ElectionID = C.ElectionID;
CREATE VIEW
voterregistrationdb=# |
```



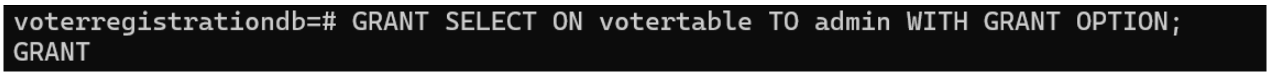
```
voterregistrationdb=# GRANT SELECT ON Electioncommission TO admin;
GRANT
voterregistrationdb=# |
```

Task 5 : created an authorisation on schemas.



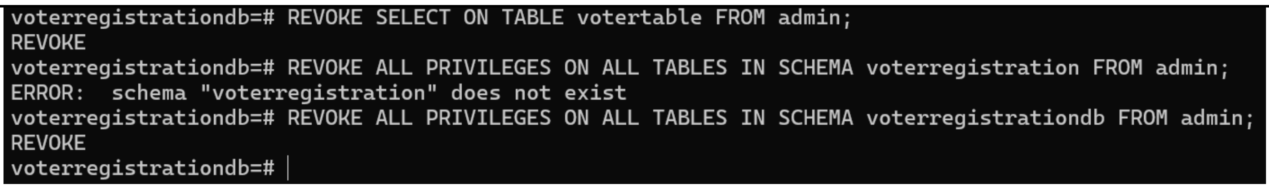
```
voterregistrationdb=# GRANT USAGE ON SCHEMA VOTERREGISTRATIONDB TO admin;
GRANT
```

Task 6 : Transfering of privileges.



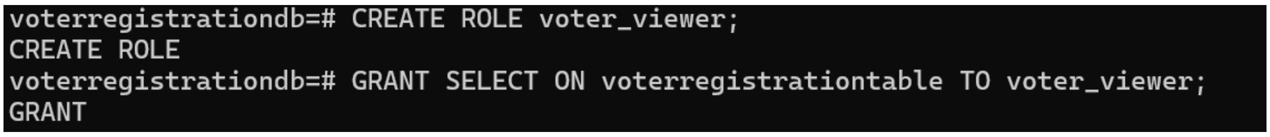
```
voterregistrationdb=# GRANT SELECT ON votertable TO admin WITH GRANT OPTION;
GRANT
```

Task 7 : Revoking of all privileges.

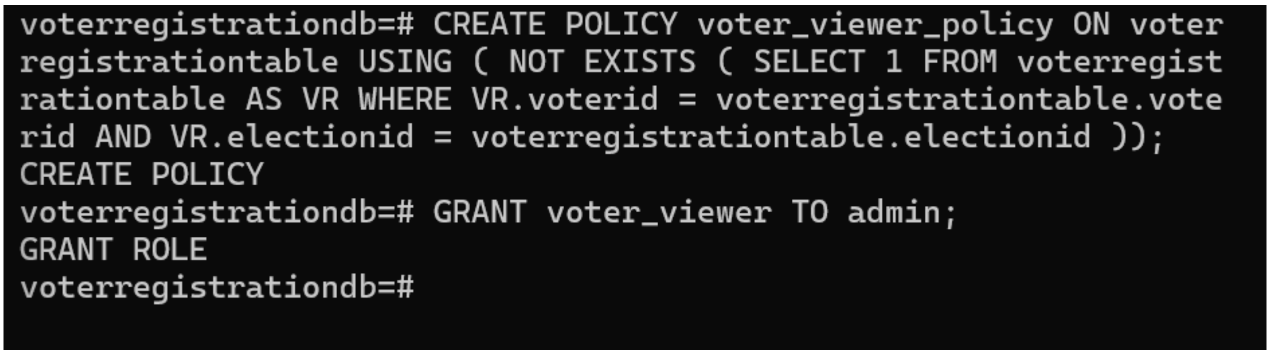


```
voterregistrationdb=# REVOKE SELECT ON TABLE votertable FROM admin;
REVOKE
voterregistrationdb=# REVOKE ALL PRIVILEGES ON ALL TABLES IN SCHEMA voterregistration FROM admin;
ERROR:  schema "voterregistration" does not exist
voterregistrationdb=# REVOKE ALL PRIVILEGES ON ALL TABLES IN SCHEMA voterregistrationdb FROM admin;
REVOKE
voterregistrationdb=# |
```

Task 8 : I have created a role as voter viewer and granted the privilege to access the voter registration table. After that I created the policy to ensure that a voter can only once.

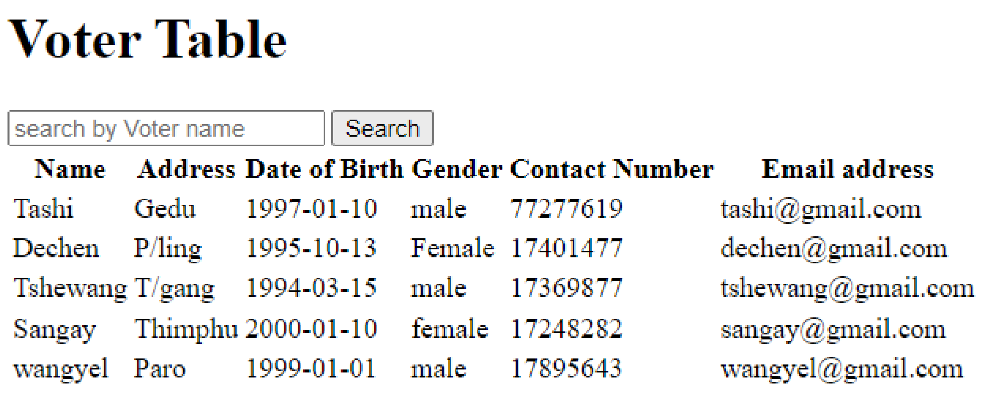


```
voterregistrationdb=# CREATE ROLE voter_viewer;
CREATE ROLE
voterregistrationdb=# GRANT SELECT ON voterregistrationtable TO voter_viewer;
GRANT
```



```
voterregistrationdb=# CREATE POLICY voter_viewer_policy ON voter
registrationtable USING ( NOT EXISTS ( SELECT 1 FROM voterregist
rationtable AS VR WHERE VR.voterid = voterregistrationtable.vote
rid AND VR.electionid = voterregistrationtable.electionid ));
CREATE POLICY
voterregistrationdb=# GRANT voter_viewer TO admin;
GRANT ROLE
voterregistrationdb=#
```

Task 9 : : Accessing the SQL from a programming language. We need to connect the database to the server and publish the database.



Voter Table

Name	Address	Date of Birth	Gender	Contact Number	Email address
Tashi	Gedu	1997-01-10	male	77277619	tashi@gmail.com
Dechen	P/ling	1995-10-13	Female	17401477	dechen@gmail.com
Tshewang	T/gang	1994-03-15	male	17369877	tshewang@gmail.com
Sangay	Thimphu	2000-01-10	female	17248282	sangay@gmail.com
wangyel	Paro	1999-01-01	male	17895643	wangyel@gmail.com

Conclusion:

To sum up, the tasks performed on the voterregistrationdb database highlight the role of database security and authorization in controlling information access to allow various users to view, avail, and alter the data as required. In addition to the use of indexes, this was accomplished using access to the database from a programming language, automating index creation using a trigger, and utilizing public roles and groups. From the preceding tasks, it was learned how database security controls such as Privilege Management are safeguarded by information and data integrity, confidentiality, and accessibility. Indexes assist in optimizing queries. As shown in this example, when creating an index on the Voter table's VoterID column, the index can directly access all the columns produced by the simulation without evaluating the query result. This treatment is critical in databases to change data sources. Privilege and access management is a mission-critical database security requirement. By granting and revoking privileges and authorizations on tables, schema, and database, both users have adequate access to perform their tasks securely without exceeding limitations.