

PostgreSQL Project-3

1. Define Database Structure :

```
Query  Query History
28  (3, 'Zxcv1234!'),
29  (3, 'AsDfGhJkL#');
30
31  SELECT * FROM Users;
```

Data Output Messages Notifications

	user_id [PK] integer	username character varying (100)	email character varying (100)
1	1	alice	alice@example.com
2	2	bob	bob@example.com
3	3	charlie	charlie@example.com
4	4	Ravikumar	Ravi@gmail.com
5	5	sajida	Sajida@gmail.com
6	6	Hari	Hari@gmail.com

```
Query  Query History
24  (1, 'A1b2C3d4!'),
25  (1, 'XyZ9876#'),
26  (2, 'P@ssw0rd!'),
27  (2, 'QwErTy123$'),
28  (3, 'Zxcv1234!'),
29  (3, 'AsDfGhJkL#');
30
31  SELECT * FROM Users;
32
33  SELECT * FROM Passwords;
```

Data Output Messages Notifications

	password_id [PK] integer	user_id integer	password character varying (100)	creation_date timestamp without time zone
1	1	1	A1b2C3d4!	2024-05-31 12:54:03.042816
2	2	1	XyZ9876#	2024-05-31 12:54:03.042816
3	3	2	P@ssw0rd!	2024-05-31 12:54:03.042816
4	4	2	QwErTy123\$	2024-05-31 12:54:03.042816
5	5	3	Zxcv1234!	2024-05-31 12:54:03.042816
6	6	3	AsDfGhJkL#	2024-05-31 12:54:03.042816

2. Implement SQL Triggers:

No limit

Query Query History

47

48 -- Trigger to call the function before inserting or updating passwords

49 **CREATE TRIGGER** password_length_trigger

50 **BEFORE INSERT OR UPDATE ON** Passwords

51 **FOR EACH ROW EXECUTE FUNCTION** check_password_length();

52

53

Data Output

Messages

Notifications

CREATE TRIGGER

Query returned successfully in 86 msec.

3. Utilize SQL Functions

Query Query History

67 **SELECT** user_id, **COUNT**(*) **AS** password_count

68 **FROM** Passwords

69 **GROUP BY** user_id;

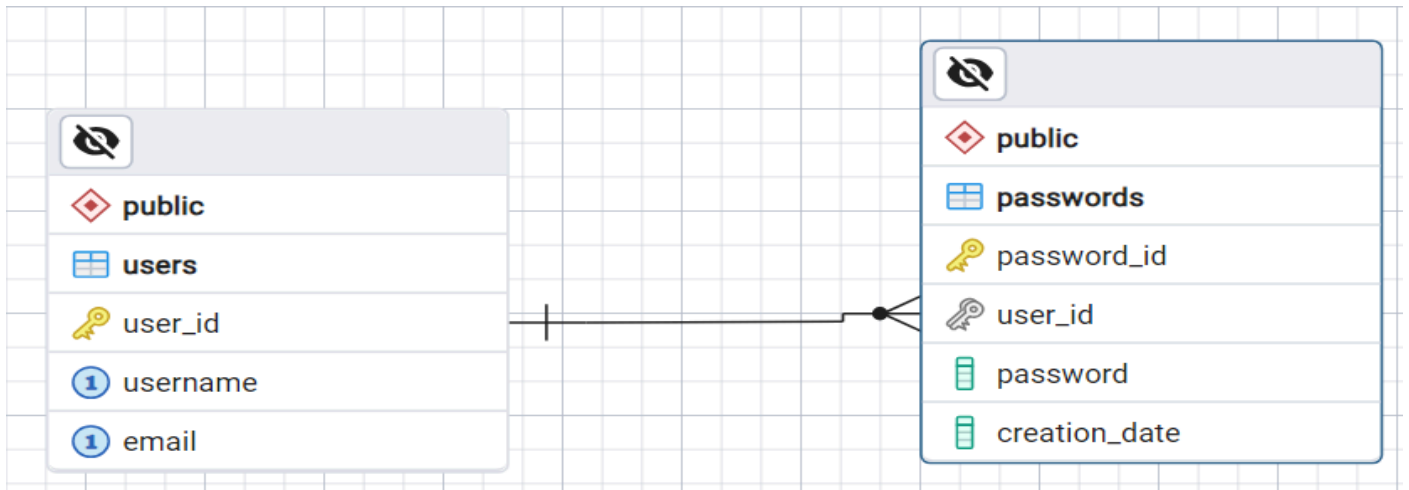
Data Output

Messages

Notifications

	user_id integer	password_count bigint
1	3	2
2	2	2
3	1	2

5. Entity-Relationship Diagram (ERD):



7. Example of Concurrent Access Control:

SQL_Project/postgres@PostgreSQL 16

Query Query History

```

71 -- 7.Example of using transactions to ensure data consistency
72 BEGIN;
73 UPDATE Users SET email = 'newemail@example.com' WHERE user_id = 1;
74 -- Other operations...
75 COMMIT;
  
```

Data Output Messages Notifications

COMMIT

Query returned successfully in 225 msec.

Query Query History

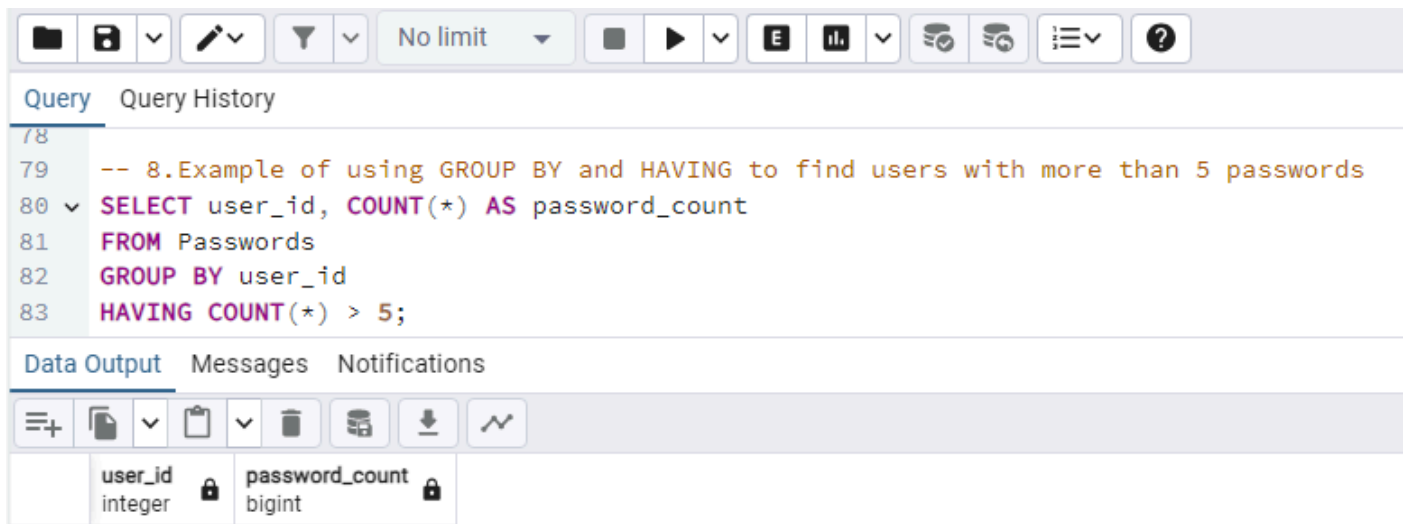
```

71 -- 7.Example of using transactions to ensure data consistency
72 BEGIN;
73 UPDATE Users SET email = 'newemail@example.com' WHERE user_id = 1;
74 -- Other operations...
75 COMMIT;
76
77 SELECT email FROM users WHERE user_id = 1;
  
```

Data Output Messages Notifications

	email character varying (100)
1	newemail@example.com

8. Apply GROUP BY and HAVING Clauses:

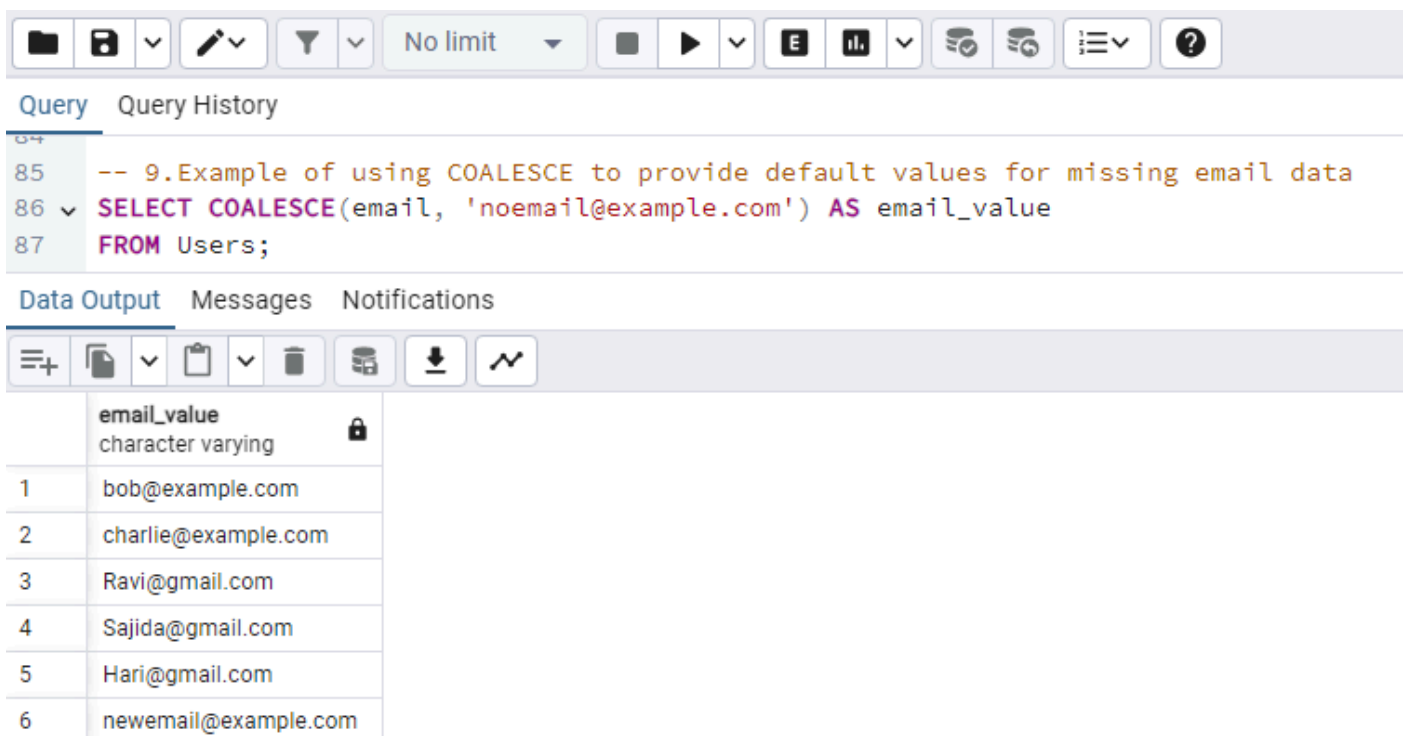


The screenshot shows a SQL IDE interface. At the top is a toolbar with icons for file operations, query execution, and settings. Below the toolbar are tabs for 'Query' and 'Query History'. The 'Query' tab is active, displaying a SQL query. Below the query editor are tabs for 'Data Output', 'Messages', and 'Notifications'. The 'Data Output' tab is active, showing a table with two columns: 'user_id' (integer) and 'password_count' (bigint).

```
78
79 -- 8.Example of using GROUP BY and HAVING to find users with more than 5 passwords
80 v SELECT user_id, COUNT(*) AS password_count
81    FROM Passwords
82   GROUP BY user_id
83  HAVING COUNT(*) > 5;
```

user_id	password_count
integer	bigint

9. Handling Missing Data:



The screenshot shows a SQL IDE interface. At the top is a toolbar with icons for file operations, query execution, and settings. Below the toolbar are tabs for 'Query' and 'Query History'. The 'Query' tab is active, displaying a SQL query. Below the query editor are tabs for 'Data Output', 'Messages', and 'Notifications'. The 'Data Output' tab is active, showing a table with one column: 'email_value' (character varying).

```
84
85 -- 9.Example of using COALESCE to provide default values for missing email data
86 v SELECT COALESCE(email, 'noemail@example.com') AS email_value
87    FROM Users;
```

email_value
character varying
1 bob@example.com
2 charlie@example.com
3 Ravi@gmail.com
4 Sajida@gmail.com
5 Hari@gmail.com
6 newemail@example.com

10. Implementing Stored Procedures:

No limit

E

Query

Query History

-- 10.Example of a stored procedure to fetch passwords for a user within a date range
CREATE OR REPLACE FUNCTION get_user_passwords(user_id INT, start_date DATE, end_date DATE)
RETURNS TABLE (password VARCHAR, creation_date TIMESTAMP) AS \$\$
BEGIN
 RETURN QUERY
 SELECT password, creation_date
 FROM Passwords
 WHERE user_id = get_user_passwords.user_id
 AND creation_date BETWEEN start_date AND end_date;
END;
\$\$ LANGUAGE plpgsql;

Data Output Messages Notifications

CREATE FUNCTION

Query returned successfully in 535 msec.

11: Implement Stored Procedures

```
Query Query History
101 -- 11.Create a stored procedure to generate a new password and insert it into the Passwords table
102 CREATE OR REPLACE FUNCTION generate_and_insert_password(
103     user_id_param INT,
104     password_length INT
105 ) RETURNS VOID AS $$
106 DECLARE
107     new_password VARCHAR := '';
108     characters TEXT := 'ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789!@#$%^&*()_+';
109     i INT := 0;
110 BEGIN
111     -- Generate a random password
112     FOR i IN 1..password_length LOOP
113         new_password := new_password || substr(characters, floor(random() * length(characters) + 1), 1);
114     END LOOP;
115
116     -- Insert the new password into the Passwords table
117     INSERT INTO Passwords (user_id, password) VALUES (user_id_param, new_password);
118 END;
119 $$ LANGUAGE plpgsql;

Data Output Messages Notifications

CREATE FUNCTION

Query returned successfully in 232 msec.
```