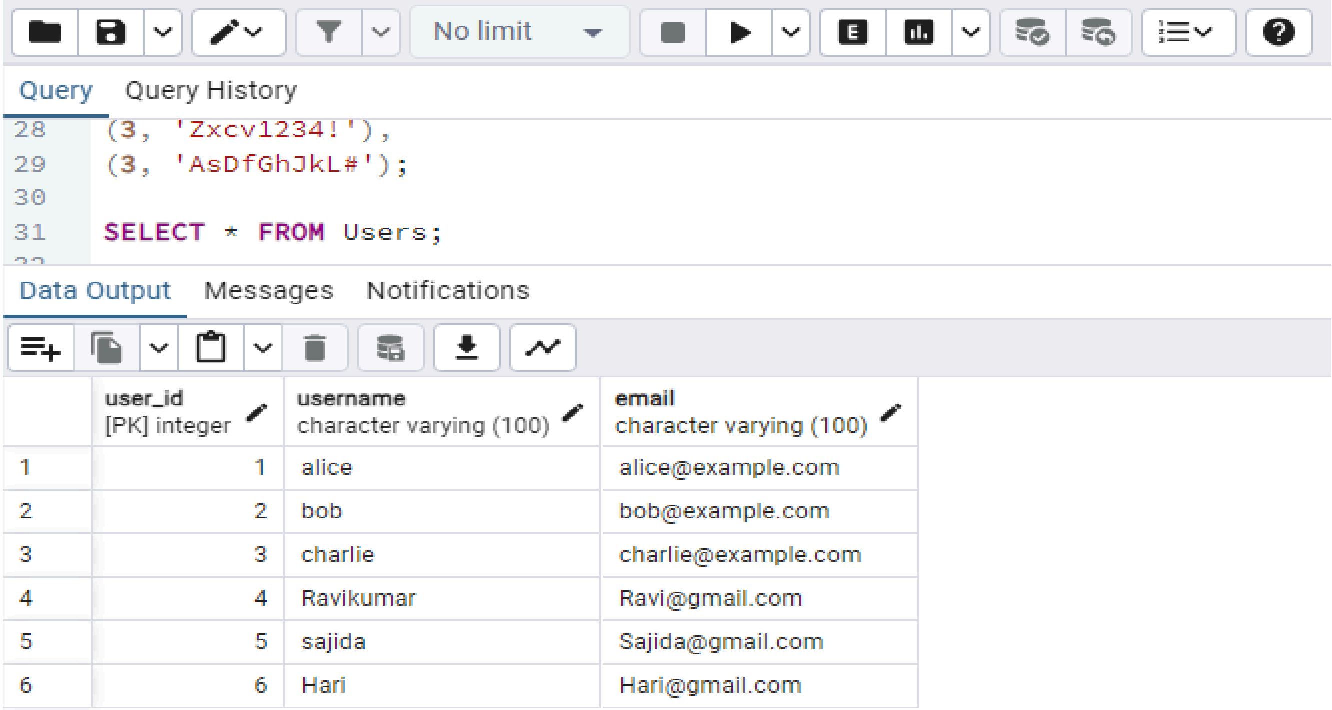
PostgreSQL Project-3

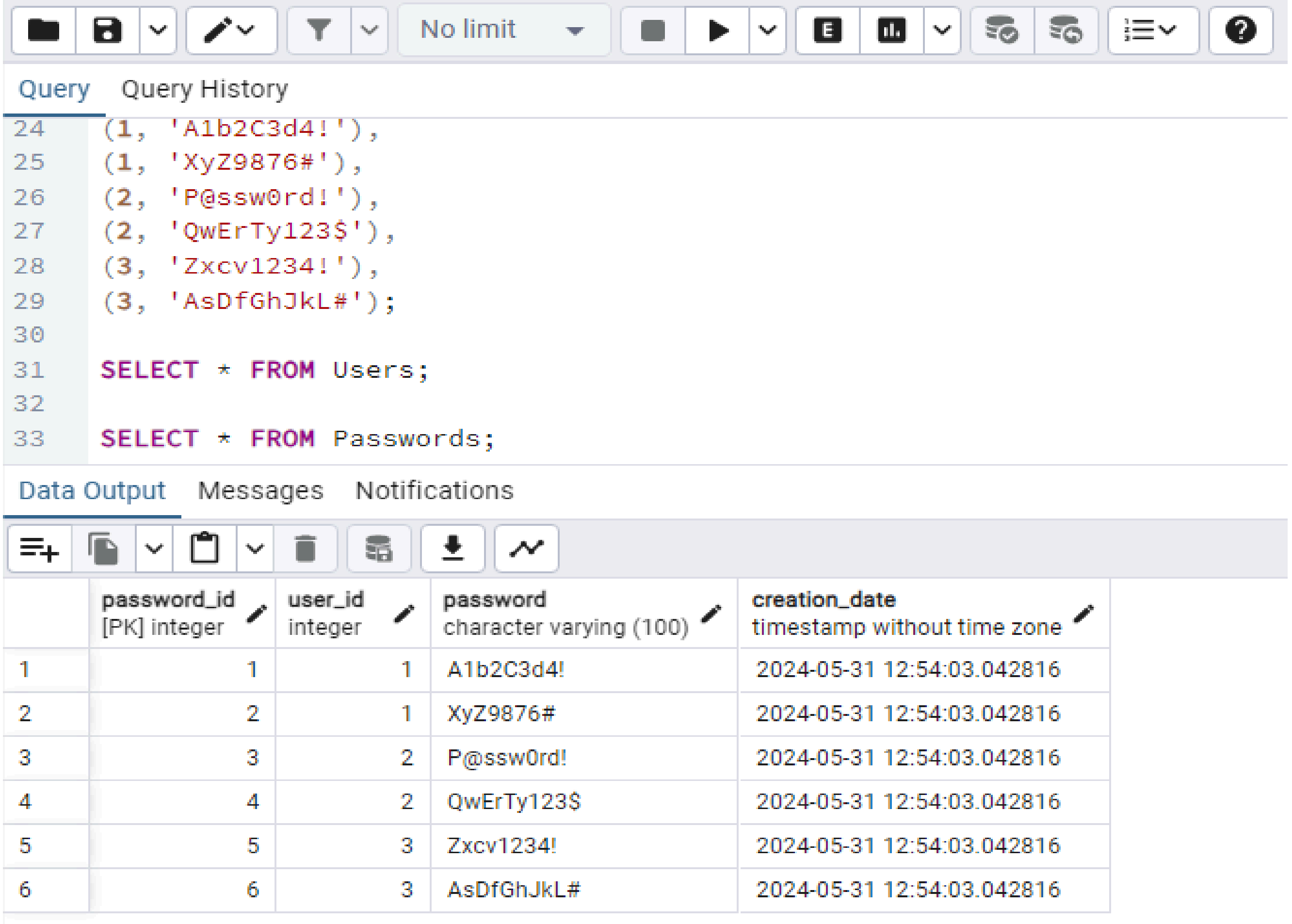
1. Define Database Structure :



```sql
(3, 'Zxcv1234!'),
(3, 'AsDfGhJkL#');

SELECT * FROM Users;
```

	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



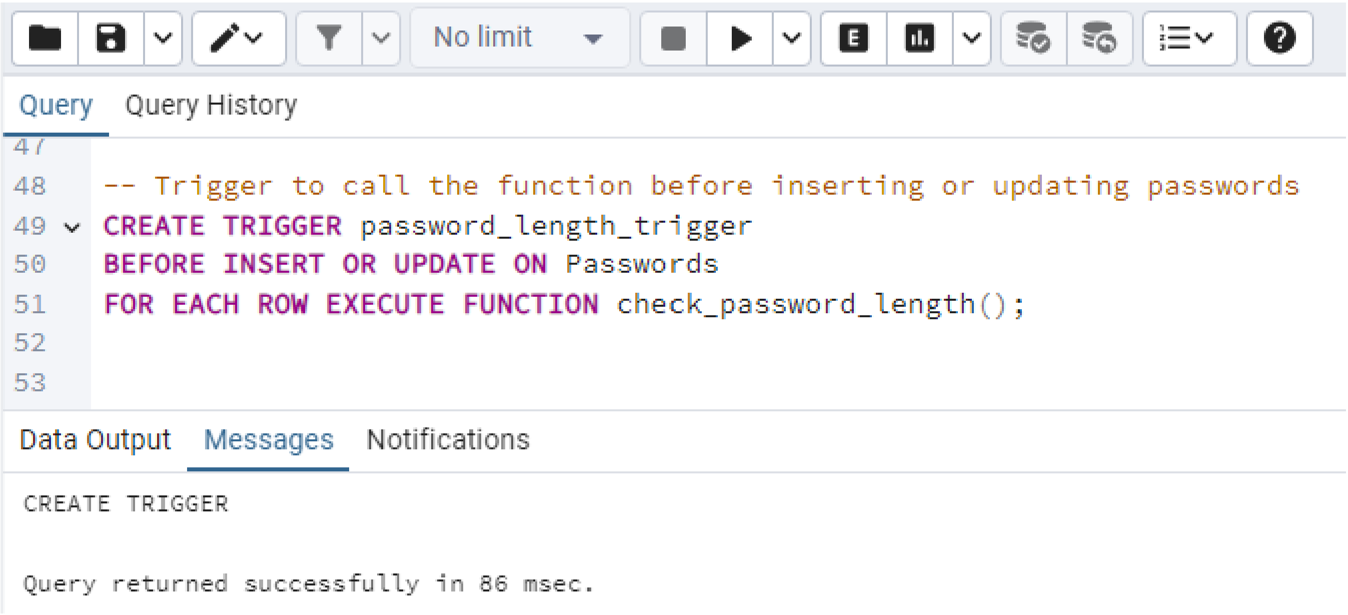
```sql
(1, 'A1b2C3d4!'),
(1, 'XyZ9876#'),
(2, 'P@ssw0rd!'),
(2, 'QwErTy123$'),
(3, 'Zxcv1234!'),
(3, 'AsDfGhJkL#');

SELECT * FROM Users;

SELECT * FROM Passwords;
```

	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:

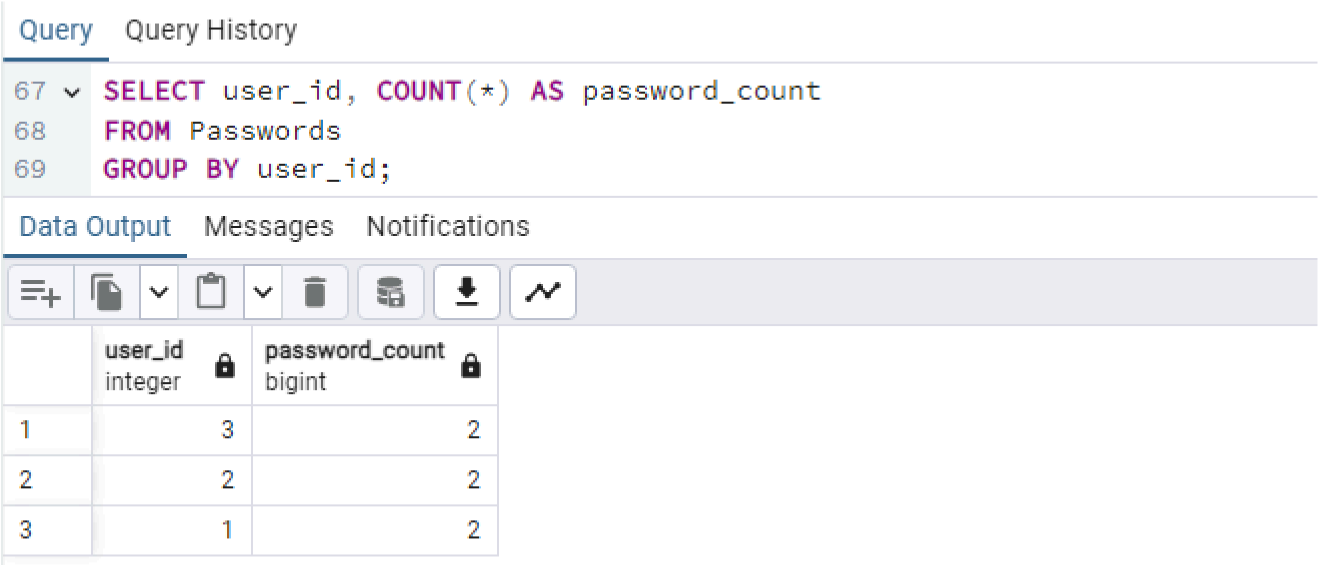


```sql
-- Trigger to call the function before inserting or updating passwords
CREATE TRIGGER password_length_trigger
BEFORE INSERT OR UPDATE ON Passwords
FOR EACH ROW EXECUTE FUNCTION check_password_length();
```

```
CREATE TRIGGER

Query returned successfully in 86 msec.
```

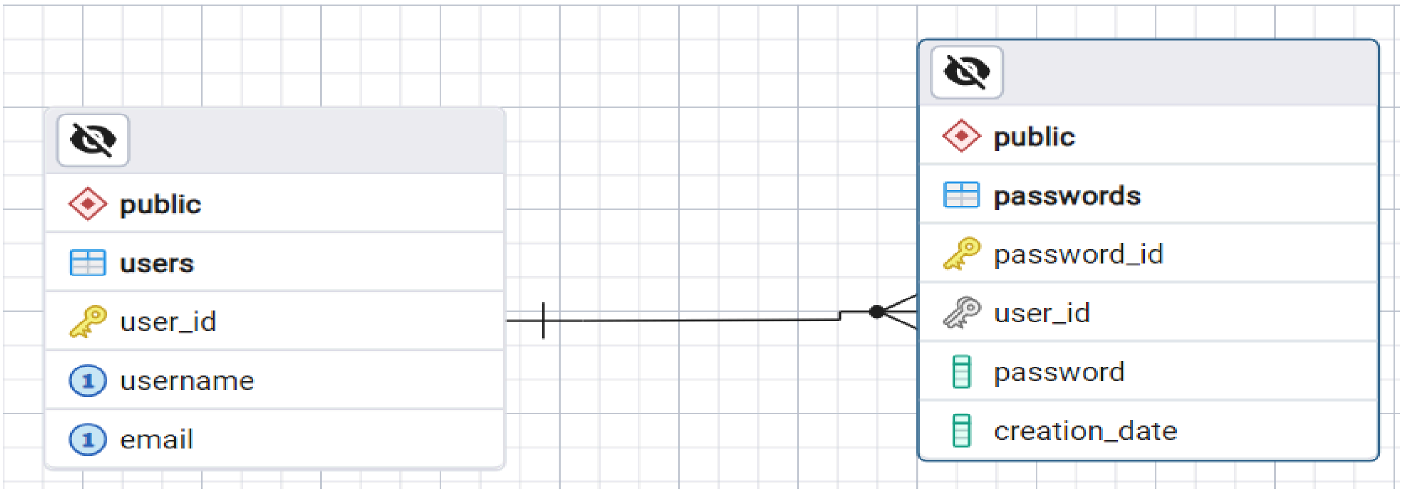
3. Utilize SQL Functions



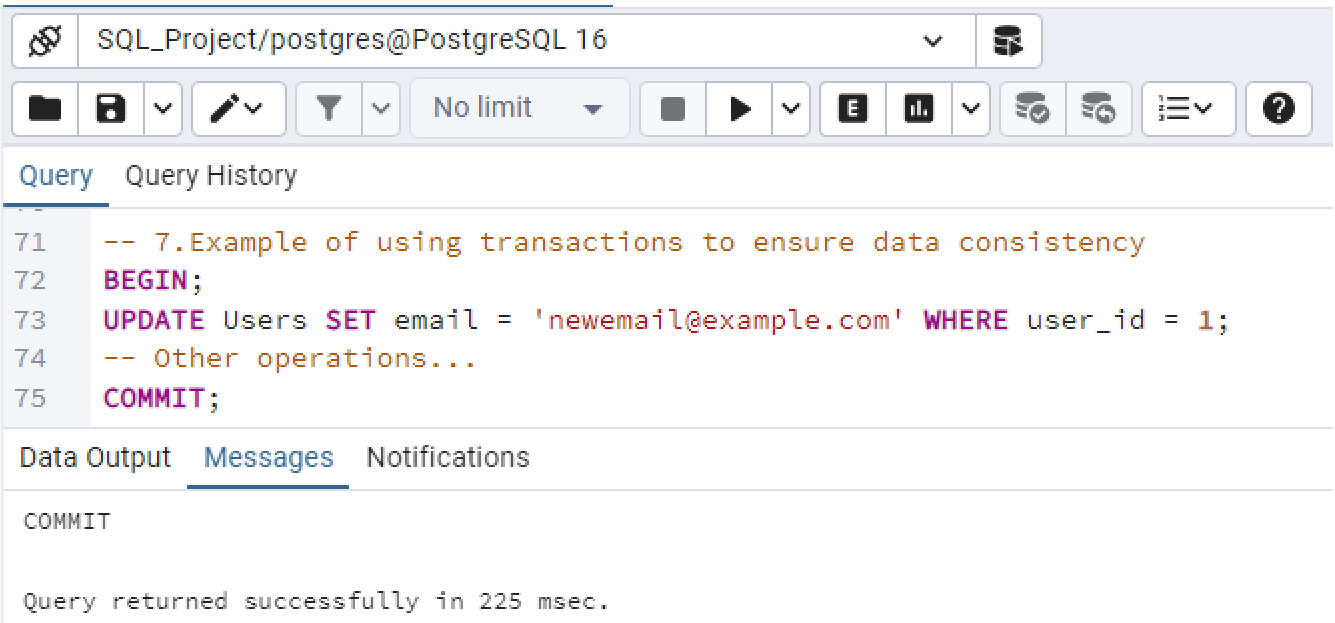
```sql
SELECT user_id, COUNT(*) AS password_count
FROM Passwords
GROUP BY user_id;
```

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

5. Entity-Relationship Diagram (ERD):



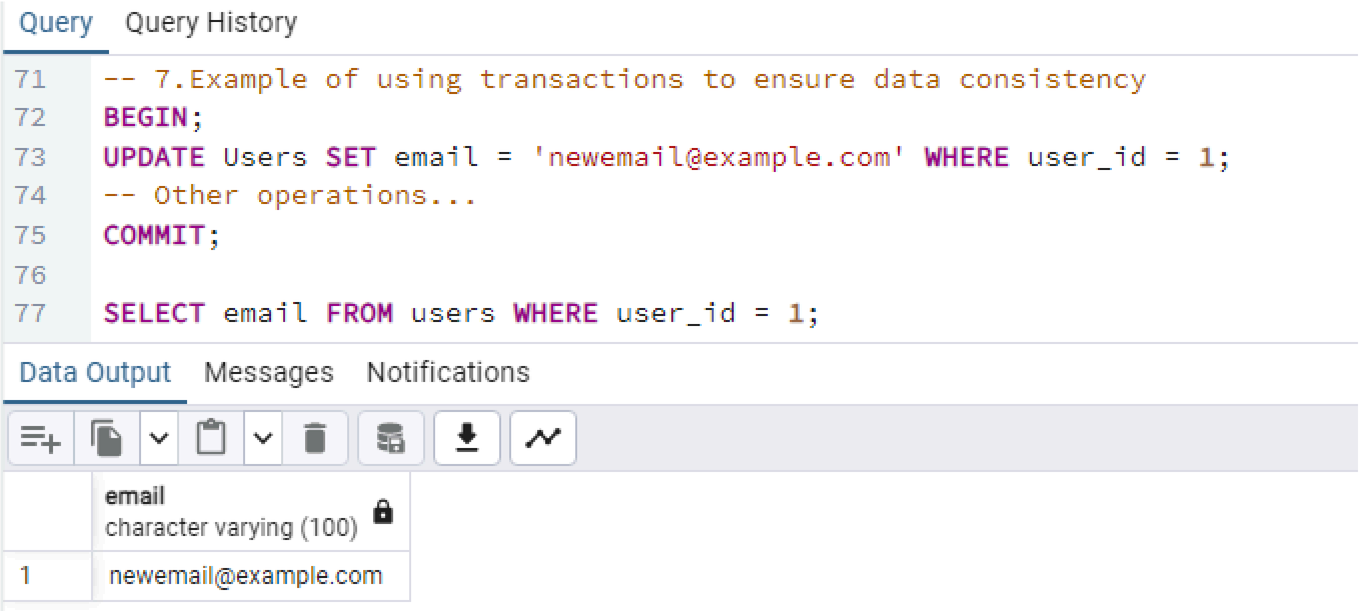
7. Example of Concurrent Access Control:



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

```
COMMIT

Query returned successfully in 225 msec.
```

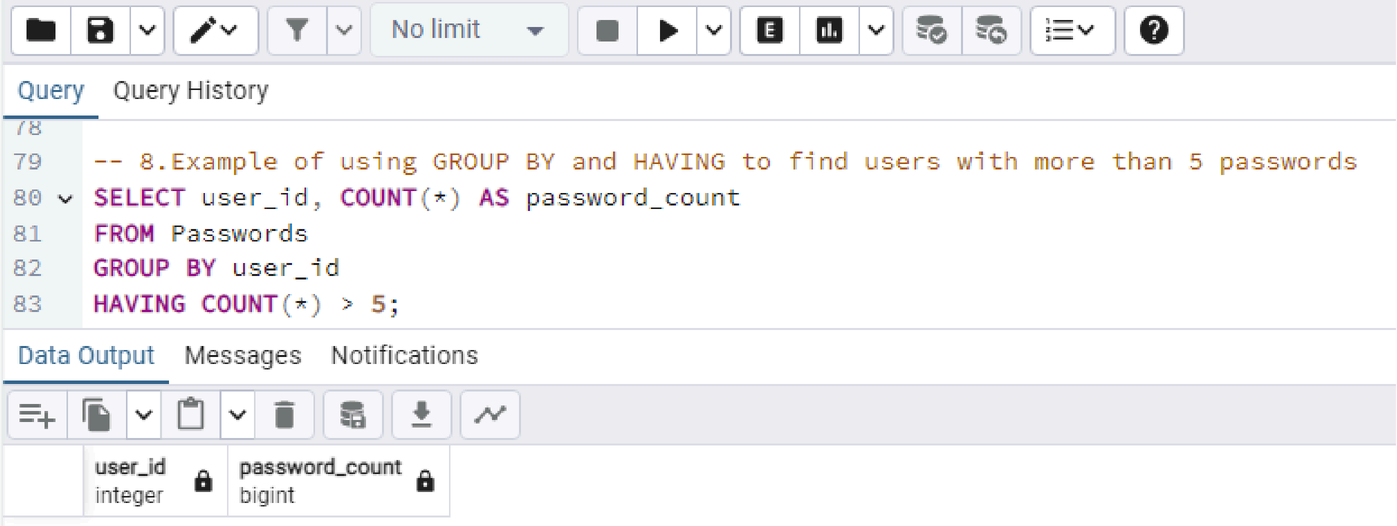


```sql
-- 7.Example of using transactions to ensure data consistency
BEGIN;
UPDATE Users SET email = 'newemail@example.com' WHERE user_id = 1;
-- Other operations...
COMMIT;

SELECT email FROM users WHERE user_id = 1;
```

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

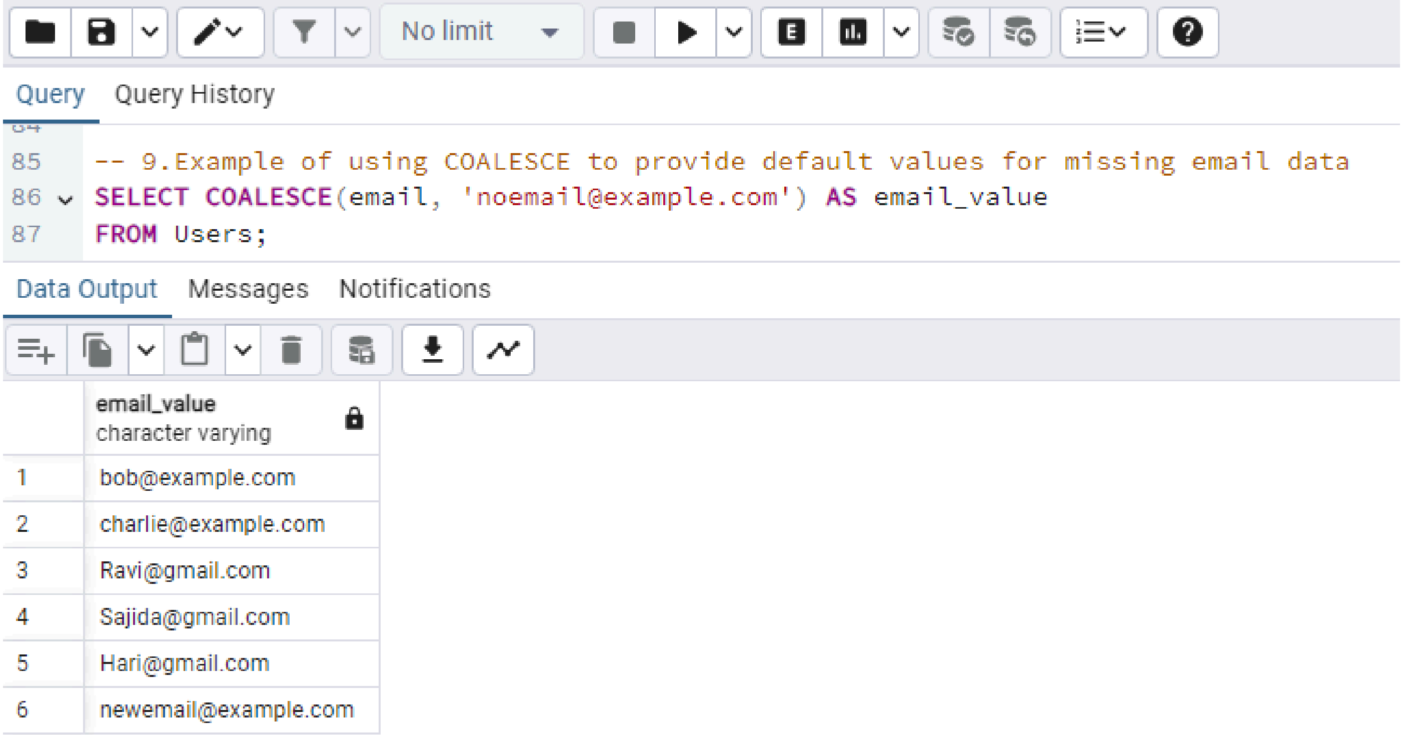
8. Apply GROUP BY and HAVING Clauses:



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

	user_id integer	password_count bigint

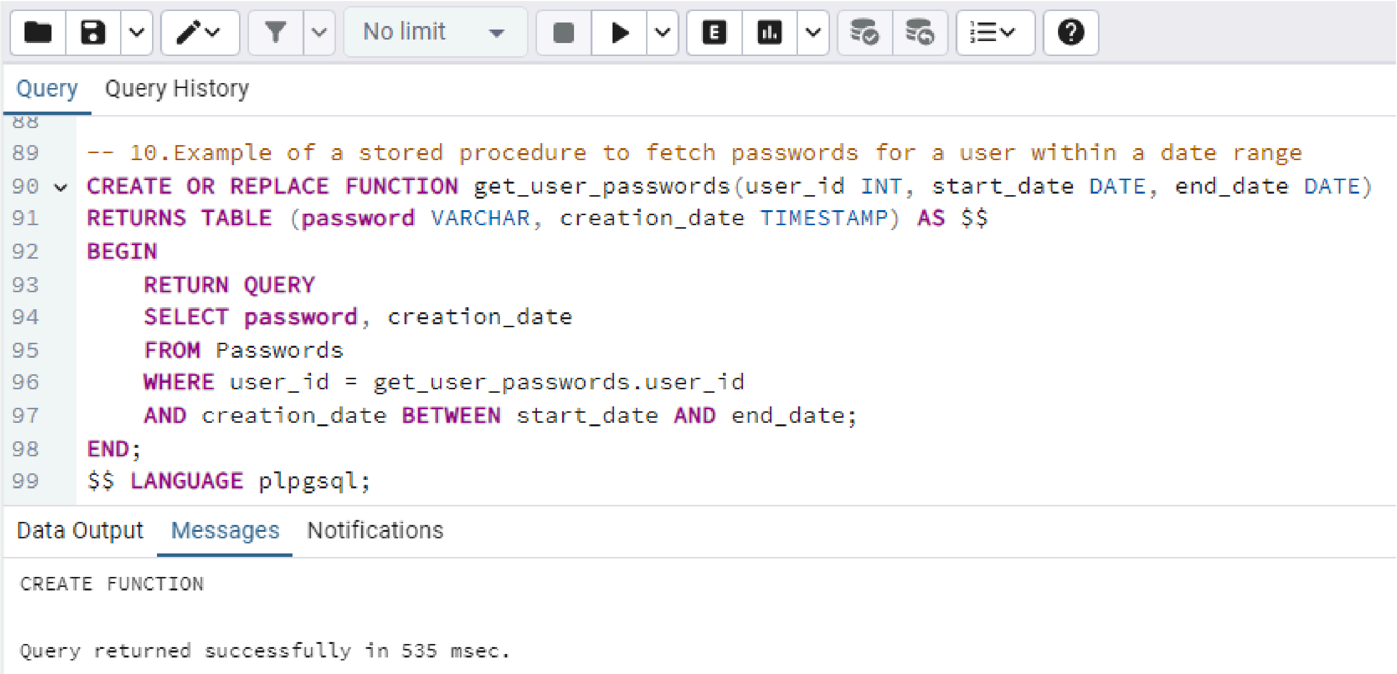
9. Handling Missing Data:



```
-- 9.Example of using COALESCE to provide default values for missing email data
SELECT COALESCE(email, 'noemail@example.com') AS email_value
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:

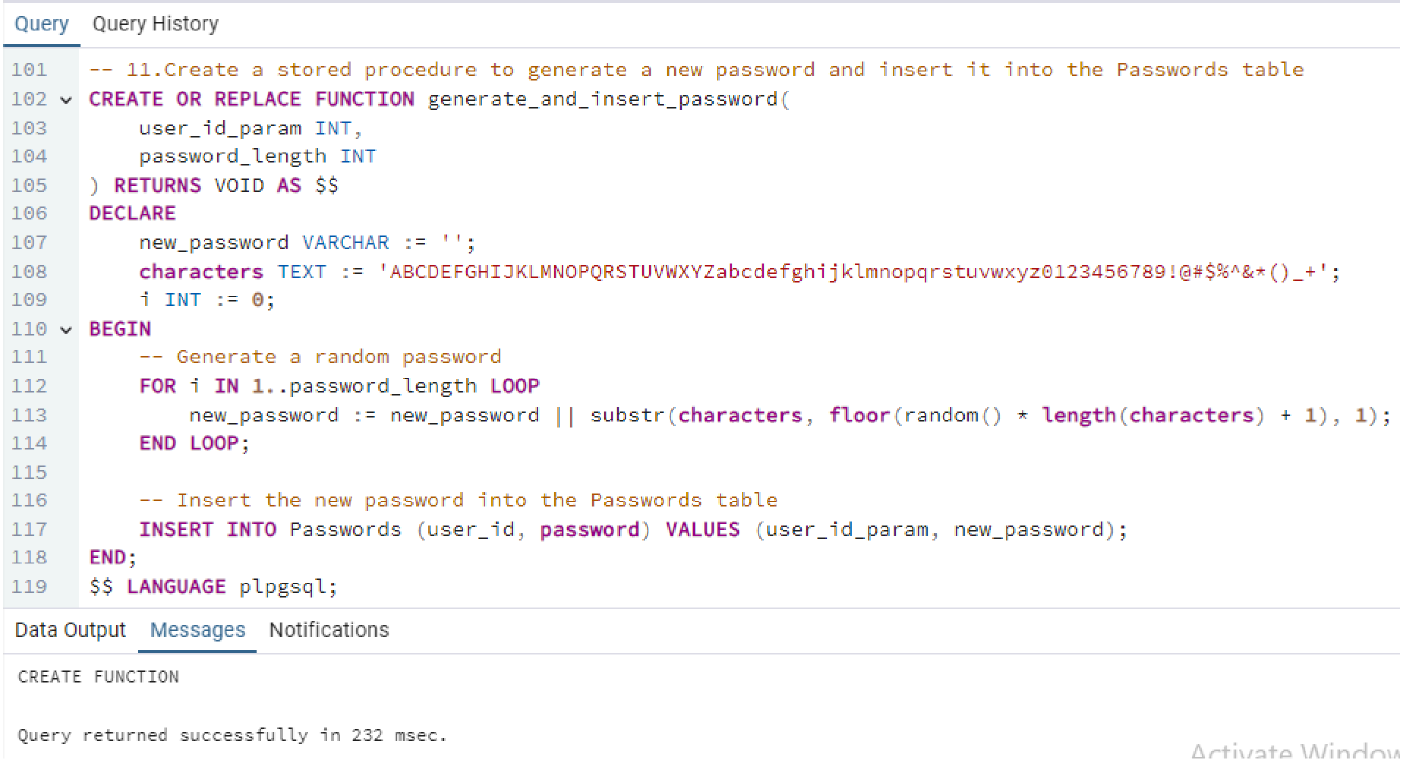


```sql
-- 10.Example of a stored procedure to fetch passwords for a user within a date range
CREATE OR REPLACE FUNCTION get_user_passwords(user_id INT, start_date DATE, end_date DATE)
RETURNS TABLE (password VARCHAR, creation_date TIMESTAMP) AS $$
BEGIN
    RETURN QUERY
    SELECT password, creation_date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

```
CREATE FUNCTION

Query returned successfully in 535 msec.
```

11: Implement Stored Procedures



```sql
-- 11.Create a stored procedure to generate a new password and insert it into the Passwords table
CREATE OR REPLACE FUNCTION generate_and_insert_password(
    user_id_param INT,
    password_length INT
) RETURNS VOID AS $$
DECLARE
    new_password VARCHAR := '';
    characters TEXT := 'ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789!@#$%^&*()_+';
    i INT := 0;
BEGIN
    -- Generate a random password
    FOR i IN 1..password_length LOOP
        new_password := new_password || substr(characters, floor(random() * length(characters) + 1), 1);
    END LOOP;

    -- Insert the new password into the Passwords table
    INSERT INTO Passwords (user_id, password) VALUES (user_id_param, new_password);
END;
$$ LANGUAGE plpgsql;
```

```
CREATE FUNCTION

Query returned successfully in 232 msec.
```