MANİSA
CELAL BAYAR
UNIVERSITY

CSE 3139

DATABASE MANAGEMENT SYSTEMS

FALL 2023

COURSE PROJECT

Report

Sena Kılınç– 210316036
Berk Kocamanoğlu-210316069

CONTENTS

1. Introduction 3

2. E–R Diagram 4

3. Creating Tables and Relations 4

4. Creating Important Queries for The Application 5

QUERY 1 – Find the usernames of the people who liked the tweet of the person who tweeted we want. 5

QUERY 2 – Find username and number of followers. 5

QUERY 3 – Find the content of the most liked tweets. 6

QUERY 4 – Find notifications received by a specific user and their times. 6

QUERY 5 – Find the number of tweets that users liked and replied to. 6

QUERY 6 – Find the latest messages from each user. 6

QUERY 7 – Find the user_name and user_id of people whose username ends with 'u'. 7

QUERY 8 – Find people with the lowest number of tweets who do not follow a selected user. 7

QUERY 9 – Find the full_name of people who liked a selected tweet but did not like the other selected tweet. 8

QUERY 10 – Find the usernames of those who use the dark theme and message each other. 8

5. Additional Part 9

Sample Output 12

Note 13

1. Introduction

This report details the database structure and functionality of a fictional social media platform called Twitter. Twitter is a platform that allows users to create profiles, share posts, follow and message other users. At the same time, the user can save the posts he likes and design his profile from the settings as he wishes.

Users and Profile Management

Users can create a comprehensive profile that includes personal and contact information. Each user has a unique user ID, username, password, name-surname, e-mail address and joining date. Users can add their own biographies and update personal information.

Following Dynamics

Users can follow and become followers of other users. These interactions are managed by cross-references in the 'followers' table. Each user can see how many people they follow and how many followers they have on their profile page.

Messaging Features

Users can communicate with each other via private messages. The messaging system includes the content and timestamp of messages along with the user IDs of the sender and receiver.

Post and Engagement

Users can share text-based posts. Each post contains a user ID and a timestamp along with the post text. Other users can interact with likes and replies. The 'likes' and 'replies' tables hold these interactions.

Personalization and Settings

Users can manage personal settings such as notification preferences, language selections and themes through the 'settings' table.

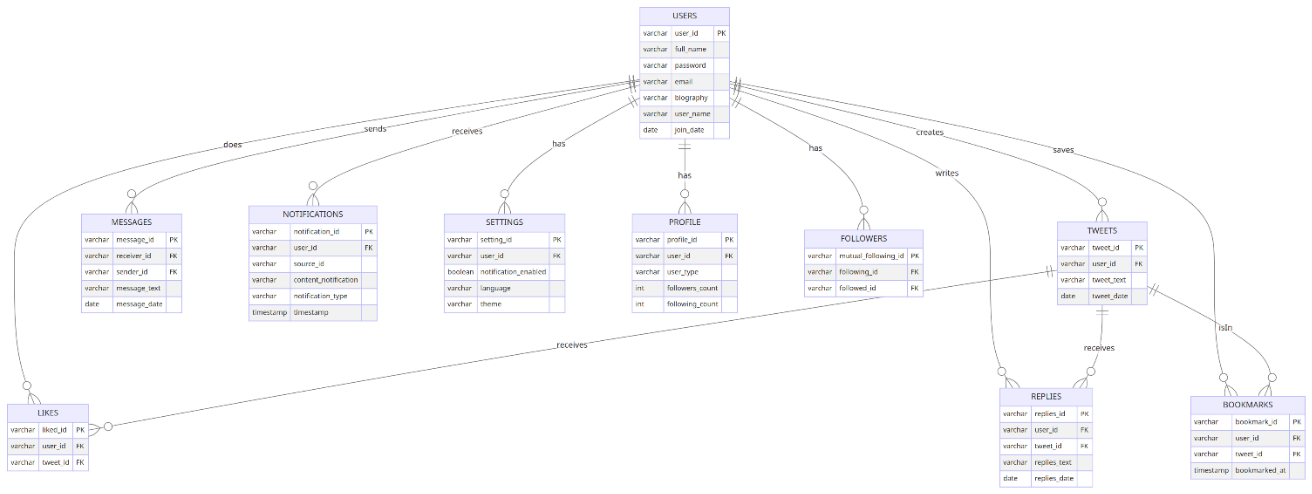
Book Marking and Notifications

Users can bookmark posts that interest them and receive notifications based on in-platform events.

This scenario is designed to support a dynamic social networking environment where social interaction and content sharing are at the forefront. The database design has been carefully designed to enrich the user experience and ensure the efficient operation of the platform.

2. E–R Diagram

Link to diagram: Twitter_E-R diagram



USERS		
varchar	user_id	PK
varchar	full_name	
varchar	password	
varchar	email	
varchar	biography	
varchar	user_name	
date	join_date	

MESSAGES		
varchar	message_id	PK
varchar	receiver_id	FK
varchar	sender_id	FK
varchar	message_text	
date	message_date	

NOTIFICATIONS		
varchar	notification_id	PK
varchar	user_id	FK
varchar	source_id	
varchar	content_notification	
varchar	notification_type	
timestamp	timestamp	

SETTINGS		
varchar	setting_id	PK
varchar	user_id	FK
boolean	notification_enabled	
varchar	language	
varchar	theme	

PROFILE		
varchar	profile_id	PK
varchar	user_id	FK
varchar	user_type	
int	followers_count	
int	following_count	

FOLLOWERS		
varchar	mutual_following_id	PK
varchar	following_id	FK
varchar	followed_id	FK

TWEETS		
varchar	tweet_id	PK
varchar	user_id	FK
varchar	tweet_text	
date	tweet_date	

LIKES		
varchar	liked_id	PK
varchar	user_id	FK
varchar	tweet_id	FK

REPLIES		
varchar	replies_id	PK
varchar	user_id	FK
varchar	tweet_id	FK
varchar	replies_text	
date	replies_date	

BOOKMARKS		
varchar	bookmark_id	PK
varchar	user_id	FK
varchar	tweet_id	FK
timestamp	bookmarked_at	

3. Creating Tables and Relations

USERS: Represents every user in the system. Each user has a unique user_id (PK - Primary Key), full name, password, email address, bio, username and joining date.

TWEETS: Represents tweets created by users. Each tweet has a unique tweet_id (PK) key, user_id (FK - Foreign Key) containing the associated user, tweet text, and tweet date.

MESSAGES: Represents private messaging between users. Each message contains a unique message_id (PK), user_ids (FK) of the recipient and sender user, message text, and message date.

LIKES: Represents a user liking a tweet. Each like contains a unique liked_id (PK), the user_id (FK) of the user who liked it, and the tweet_id (FK) of the liked tweet.

FOLLOWERS: Represents users following each other. Each record contains a unique mutual_following_id (PK) and user_ids (FK) of following and followed users.

NOTIFICATIONS: Represents the notifications users receive. Each notification contains a unique notification_id (PK), the user's user_id (FK), the notification's source, content, type, and timestamp.

REPLIES: Represents users' replies to a tweet. Each reply contains a unique replies_id (PK), the user_id (FK) of the user who wrote the reply, the tweet_id (FK) of the replied tweet, the reply text, and the reply date.

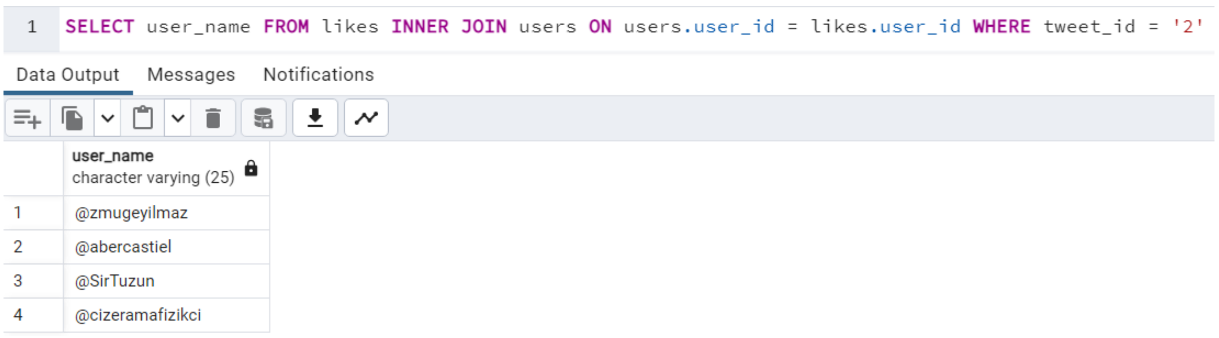
SETTINGS: Represents users' system settings. Each setting contains a unique setting_id (PK), associated user's user_id (FK), notification preference, language selection, and theme.

PROFILE: Represents user profiles. Each profile contains a unique profile_id (PK), the associated user's user_id (FK), user type, and follower counts.

BOOKMARKS: Represents users to save specific tweets. Each bookmark contains a unique bookmark_id (PK), the user_id (FK) of the user, the tweet_id (FK) of the recorded tweet, and the time it was recorded.

4. Creating Important Queries for The Application

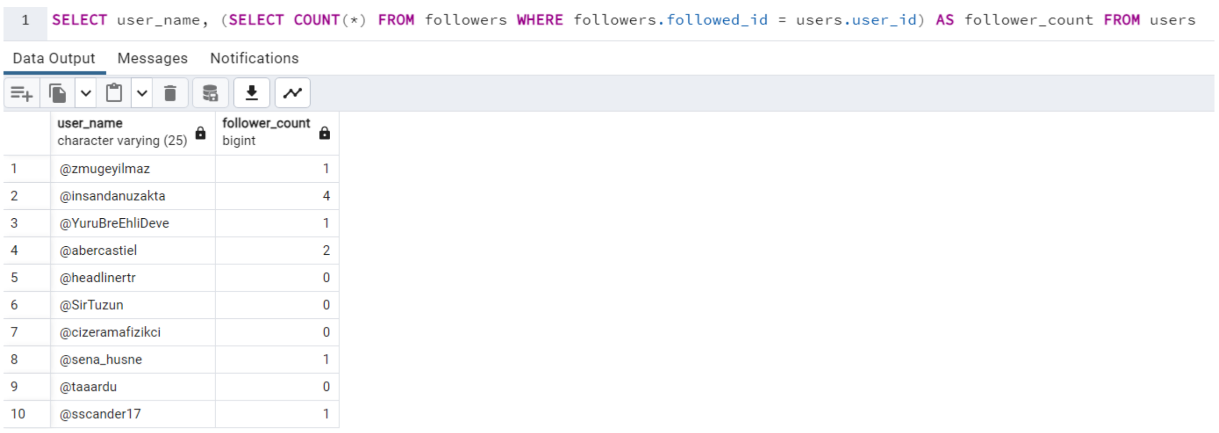
QUERY 1 – Find the usernames of the people who liked the tweet of the person who tweeted we want.
SELECT user_name FROM likes INNER JOIN users ON users.user_id = likes.user_id WHERE tweet_id = '2'



```sql
SELECT user_name FROM likes INNER JOIN users ON users.user_id = likes.user_id WHERE tweet_id = '2'
```

	user_name character varying (25)
1	@zmugeyilmaz
2	@abercastiel
3	@SirTuzun
4	@cizeramafizikci

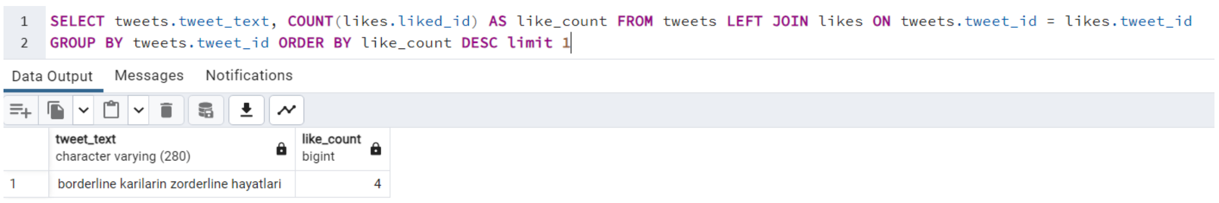
QUERY 2 – Find username and number of followers.
SELECT user_name, (SELECT COUNT(*) FROM followers WHERE followers.followed_id = users.user_id) AS follower_count FROM users



```sql
SELECT user_name, (SELECT COUNT(*) FROM followers WHERE followers.followed_id = users.user_id) AS follower_count FROM users
```

	user_name character varying (25)	follower_count bigint
1	@zmugeyilmaz	1
2	@insandanuzakta	4
3	@YuruBreEhliDeve	1
4	@abercastiel	2
5	@headlinertr	0
6	@SirTuzun	0
7	@cizeramafizikci	0
8	@sena_husne	1
9	@taaardu	0
10	@sscander17	1

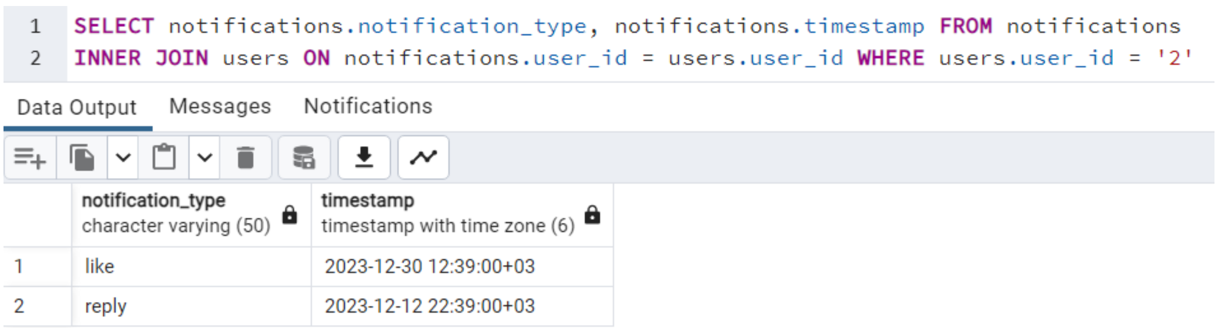
QUERY 3 – Find the content of the most liked tweets.
SELECT tweets.tweet_text, COUNT(likes.liked_id) AS like_count FROM tweets LEFT JOIN likes ON tweets.tweet_id = likes.tweet_id GROUP BY tweets.tweet_id ORDER BY like_count DESC limit 1



```
SELECT tweets.tweet_text, COUNT(likes.liked_id) AS like_count FROM tweets LEFT JOIN likes ON tweets.tweet_id = likes.tweet_id
GROUP BY tweets.tweet_id ORDER BY like_count DESC limit 1
```

	tweet_text character varying (280)	like_count bigint
1	borderline karilarin zorderline hayatlari	4

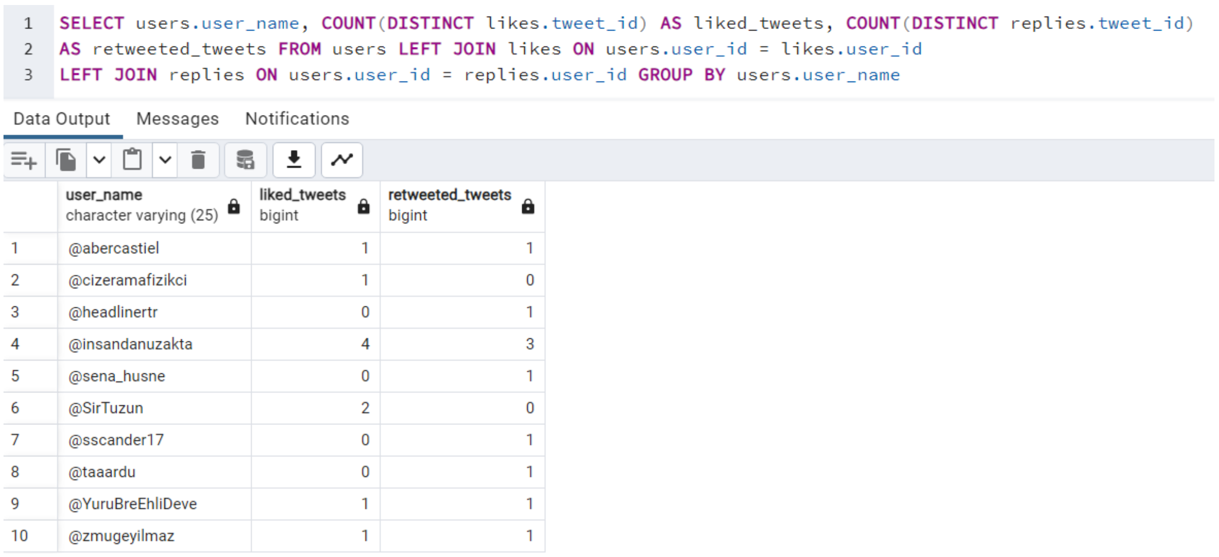
QUERY 4 – Find notifications received by a specific user and their times.
SELECT notifications.notification_type, notifications.timestamp FROM notifications INNER JOIN users ON notifications.user_id = users.user_id WHERE users.user_id = '2'



```
SELECT notifications.notification_type, notifications.timestamp FROM notifications
INNER JOIN users ON notifications.user_id = users.user_id WHERE users.user_id = '2'
```

	notification_type character varying (50)	timestamp timestamp with time zone (6)
1	like	2023-12-30 12:39:00+03
2	reply	2023-12-12 22:39:00+03

QUERY 5 – Find the number of tweets that users liked and replied to.
SELECT users.user_name, COUNT(DISTINCT likes.tweet_id) AS liked_tweets, COUNT(DISTINCT replies.tweet_id) AS retweeted_tweets FROM users LEFT JOIN likes ON users.user_id = likes.user_id LEFT JOIN replies ON users.user_id = replies.user_id GROUP BY users.user_name

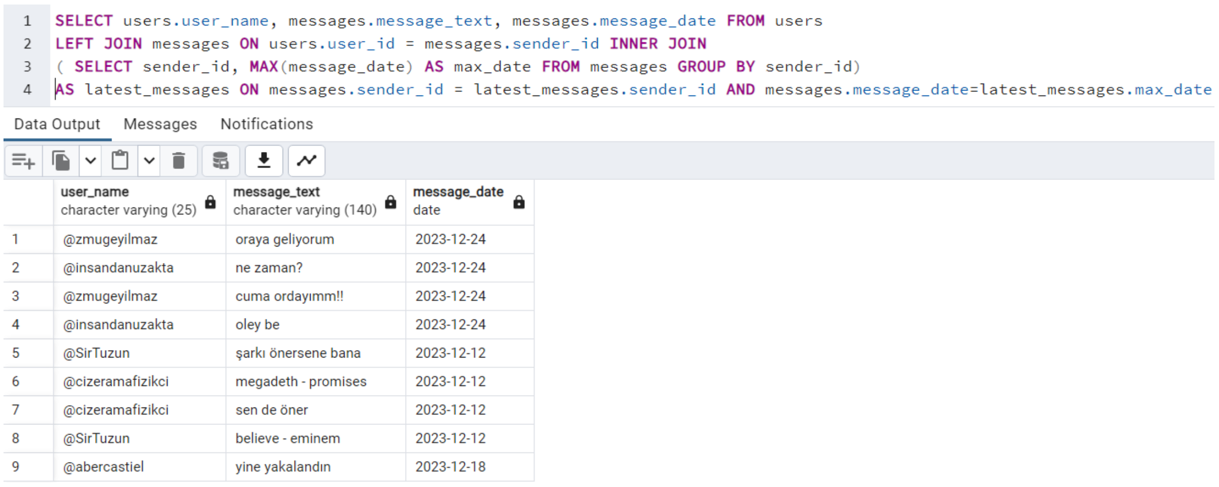


```
SELECT users.user_name, COUNT(DISTINCT likes.tweet_id) AS liked_tweets, COUNT(DISTINCT replies.tweet_id)
AS retweeted_tweets FROM users LEFT JOIN likes ON users.user_id = likes.user_id
LEFT JOIN replies ON users.user_id = replies.user_id GROUP BY users.user_name
```

	user_name character varying (25)	liked_tweets bigint	retweeted_tweets bigint
1	@abercastiel	1	1
2	@cizeramafizikci	1	0
3	@headlinertr	0	1
4	@insandanuzakta	4	3
5	@sena_husne	0	1
6	@SirTuzun	2	0
7	@sscander17	0	1
8	@taaardu	0	1
9	@YuruBreEhliDeve	1	1
10	@zmugeyilmaz	1	1

QUERY 6 – Find the latest messages from each user.
SELECT users.user_name, messages.message_text, messages.message_date FROM users LEFT JOIN messages ON users.user_id = messages.sender_id INNER JOIN (SELECT sender_id, MAX(message_date) AS max_date FROM messages GROUP BY sender_id) AS

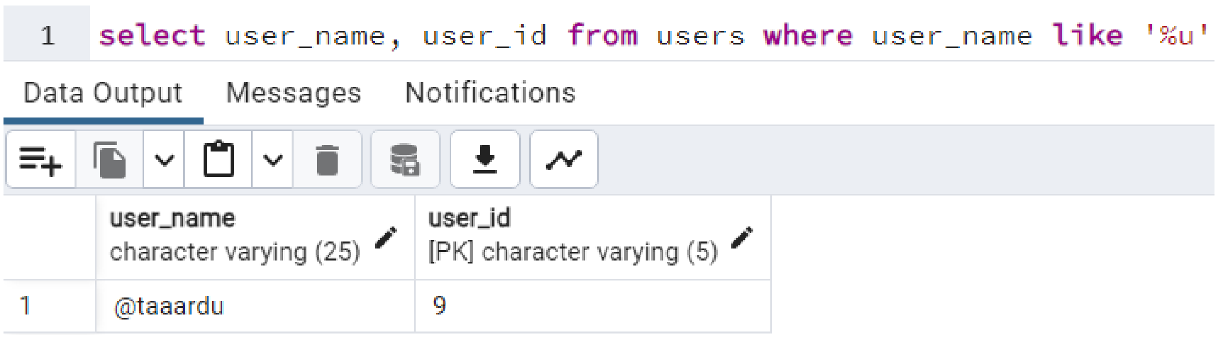
latest_messages ON messages.sender_id = latest_messages.sender_id AND messages.message_date latest_messages.max_date



```sql
SELECT users.user_name, messages.message_text, messages.message_date FROM users
LEFT JOIN messages ON users.user_id = messages.sender_id INNER JOIN
( SELECT sender_id, MAX(message_date) AS max_date FROM messages GROUP BY sender_id)
AS latest_messages ON messages.sender_id = latest_messages.sender_id AND messages.message_date=latest_messages.max_date
```

	user_name character varying (25)	message_text character varying (140)	message_date date
1	@zmugeyilmaz	oraya geliyorum	2023-12-24
2	@insandanuzakta	ne zaman?	2023-12-24
3	@zmugeyilmaz	cuma ordayımm!!	2023-12-24
4	@insandanuzakta	oley be	2023-12-24
5	@SirTuzun	şarkı önersene bana	2023-12-12
6	@cizeramafizikci	megadeth - promises	2023-12-12
7	@cizeramafizikci	sen de öner	2023-12-12
8	@SirTuzun	believe - eminem	2023-12-12
9	@abercastiel	yine yakalandın	2023-12-18

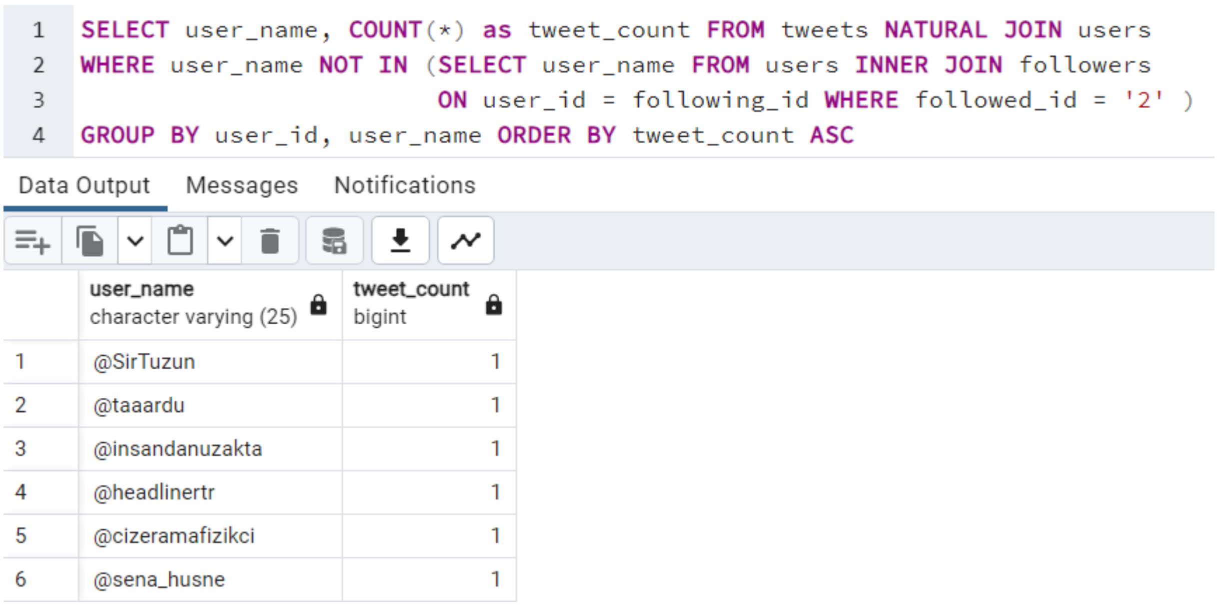
QUERY 7 – Find the user_name and user_id of people whose username ends with ‘u’.
select user_name, user_id from users where user_name like '%u'



```sql
select user_name, user_id from users where user_name like '%u'
```

	user_name character varying (25)	user_id [PK] character varying (5)
1	@taaardu	9

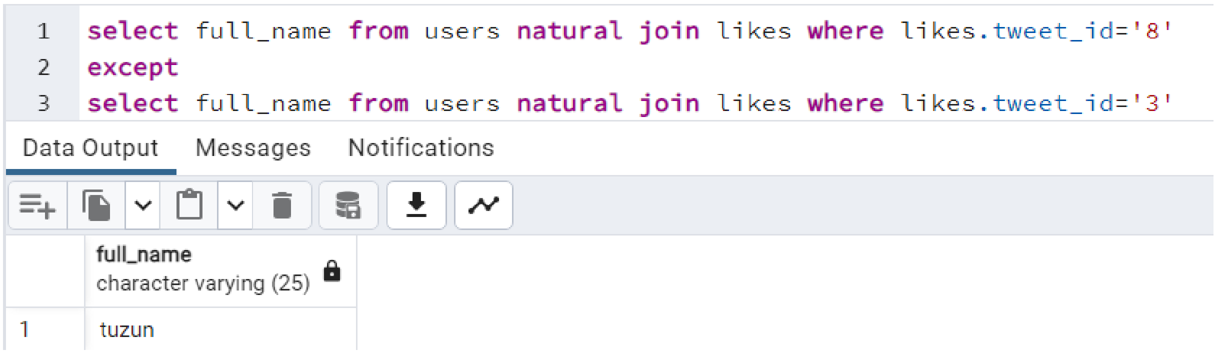
QUERY 8 – Find people with the lowest number of tweets who do not follow a selected user.
SELECT user_name, COUNT(*) as tweet_count FROM tweets NATURAL JOIN users WHERE user_name NOT IN (SELECT user_name FROM users INNER JOIN followers ON user_id = following_id WHERE followed_id = '2') GROUP BY user_id, user_name ORDER BY tweet_count ASC


```
SELECT user_name, COUNT(*) as tweet_count FROM tweets NATURAL JOIN users
WHERE user_name NOT IN (SELECT user_name FROM users INNER JOIN followers
                        ON user_id = following_id WHERE followed_id = '2' )
GROUP BY user_id, user_name ORDER BY tweet_count ASC
```

Data Output Messages Notifications

	user_name character varying (25)	tweet_count bigint
1	@SirTuzun	1
2	@taaardu	1
3	@insandanuzakta	1
4	@headlinertr	1
5	@cizeramafizikci	1
6	@sena_husne	1

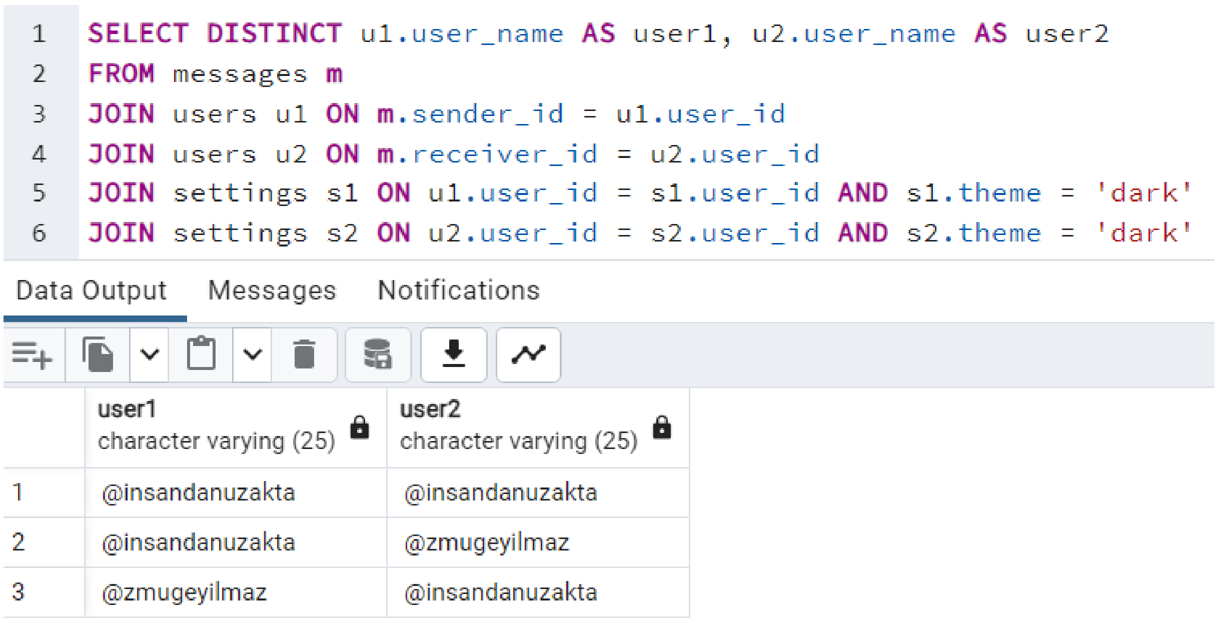
QUERY 9 – Find the full_name of people who liked a selected tweet but did not like the other selected tweet.
select full_name from users natural join likes where likes.tweet_id='8' except select full_name from users natural join likes where likes.tweet_id='3'


```
select full_name from users natural join likes where likes.tweet_id='8'
except
select full_name from users natural join likes where likes.tweet_id='3'
```

Data Output Messages Notifications

	full_name character varying (25)
1	tuzun

QUERY 10 – Find the usernames of those who use the dark theme and message each other.
SELECT DISTINCT u1.user_name AS user1, u2.user_name AS user2 FROM messages m JOIN users u1 ON m.sender_id = u1.user_id JOIN users u2 ON m.receiver_id = u2.user_id JOIN settings s1 ON u1.user_id = s1.user_id AND s1.theme = 'dark' JOIN settings s2 ON u2.user_id = s2.user_id AND s2.theme = 'dark'



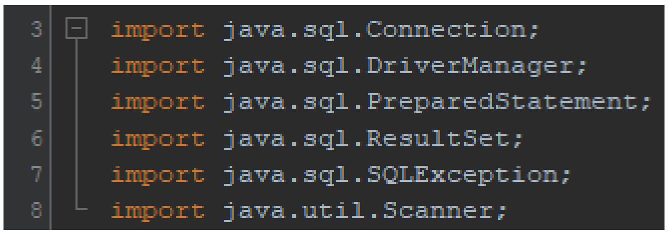
```sql
SELECT DISTINCT u1.user_name AS user1, u2.user_name AS user2
FROM messages m
JOIN users u1 ON m.sender_id = u1.user_id
JOIN users u2 ON m.receiver_id = u2.user_id
JOIN settings s1 ON u1.user_id = s1.user_id AND s1.theme = 'dark'
JOIN settings s2 ON u2.user_id = s2.user_id AND s2.theme = 'dark'
```

Data Output Messages Notifications

	user1 character varying (25)	user2 character varying (25)
1	@insandanuzakta	@insandanuzakta
2	@insandanuzakta	@zmugeyilmaz
3	@zmugeyilmaz	@insandanuzakta

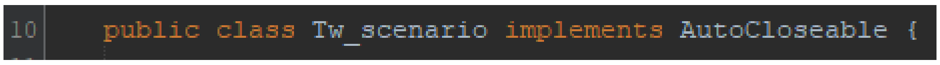
5. Additional Part

In this part, we combined Postgresql and Java. Email and password are entered from the console. This information is pulled from the database, its accuracy is compared, and if it is correct, the user's messages and tweets are displayed on the screen. Just like seeing your profile. If the inputs are incorrect, we receive an error message.



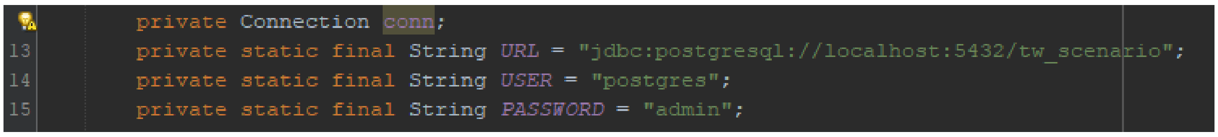
```java
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.Scanner;
```

Imports: Imports Connection, DriverManager, PreparedStatement, ResultSet, SQLException classes from the java.sql package. These are part of the JDBC API and are used for database operations in Java. java.util.Scanner is used to read user input.



```java
public class Tw_scenario implements AutoCloseable {
```

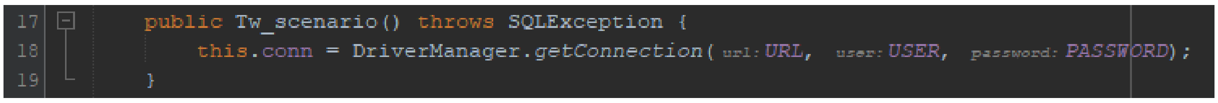
It defines a public class called tw_scenario. This class implements the AutoCloseable interface, which means the class can automatically release resources (in this case, database connection).



```java
    private Connection conn;
    private static final String URL = "jdbc:postgresql://localhost:5432/tw_scenario";
    private static final String USER = "postgres";
    private static final String PASSWORD = "admin";
```

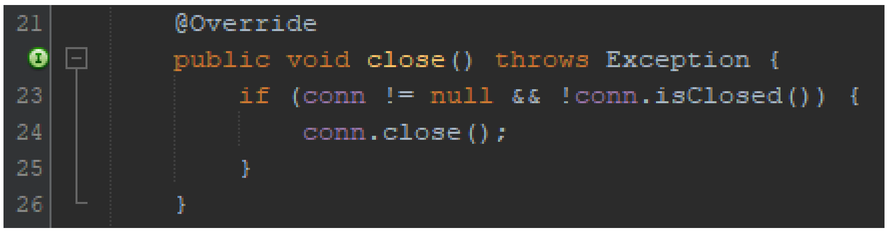
conn: A variable of type Connection that holds the database connection.

URL, USER, PASSWORD: Stores the information required for the database connection. These constants (final) are used by all instances of the class and their values cannot be changed.



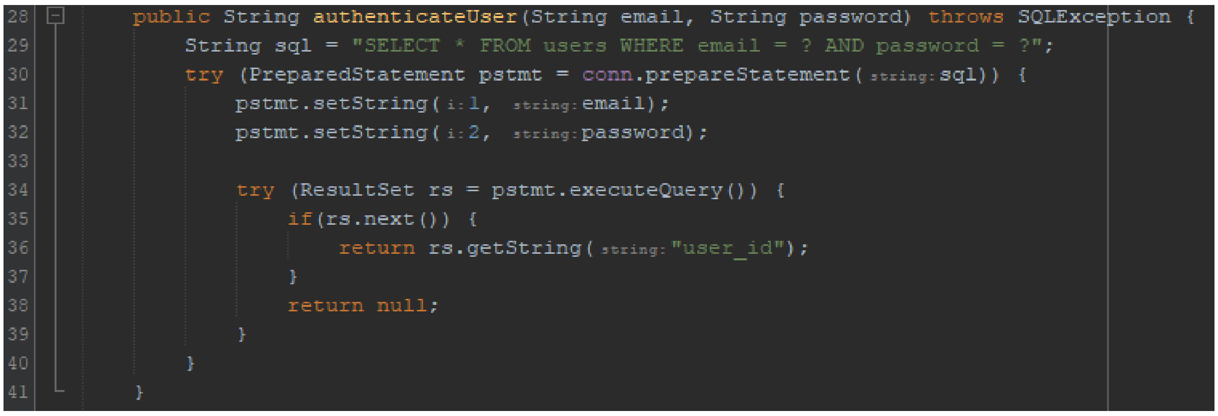
```java
    public Tw_scenario() throws SQLException {
        this.conn = DriverManager.getConnection(URL, USER, PASSWORD);
    }
```

Constructor method of tw_scenario class. This method is called when the class instance is created and initiates the database connection.



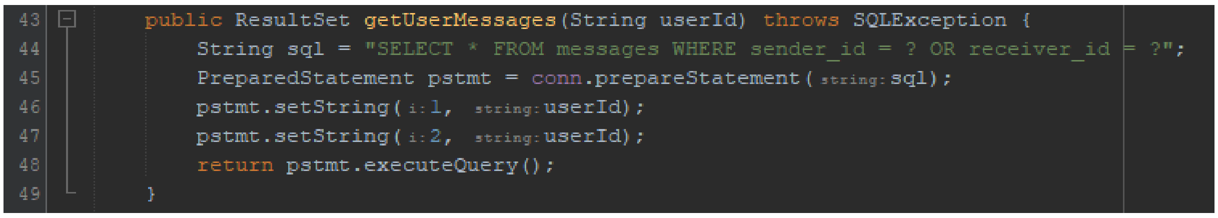
```java
    @Override
    public void close() throws Exception {
        if (conn != null && !conn.isClosed()) {
            conn.close();
        }
    }
```

It overrides the close() method from the AutoCloseable interface. This method ensures that resources (in this case, the database connection) are properly released.



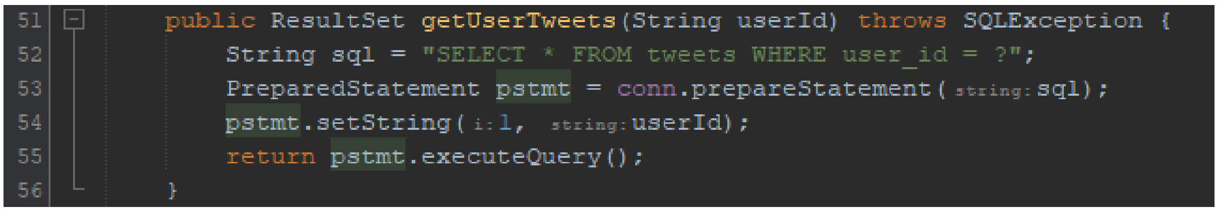
```java
    public String authenticateUser(String email, String password) throws SQLException {
        String sql = "SELECT * FROM users WHERE email = ? AND password = ?";
        try (PreparedStatement pstmt = conn.prepareStatement(sql)) {
            pstmt.setString(1, email);
            pstmt.setString(2, password);

            try (ResultSet rs = pstmt.executeQuery()) {
                if(rs.next()) {
                    return rs.getString("user_id");
                }
                return null;
            }
        }
    }
```

The method used for user authentication. It checks the database to see if there is a user matching the given email and password.



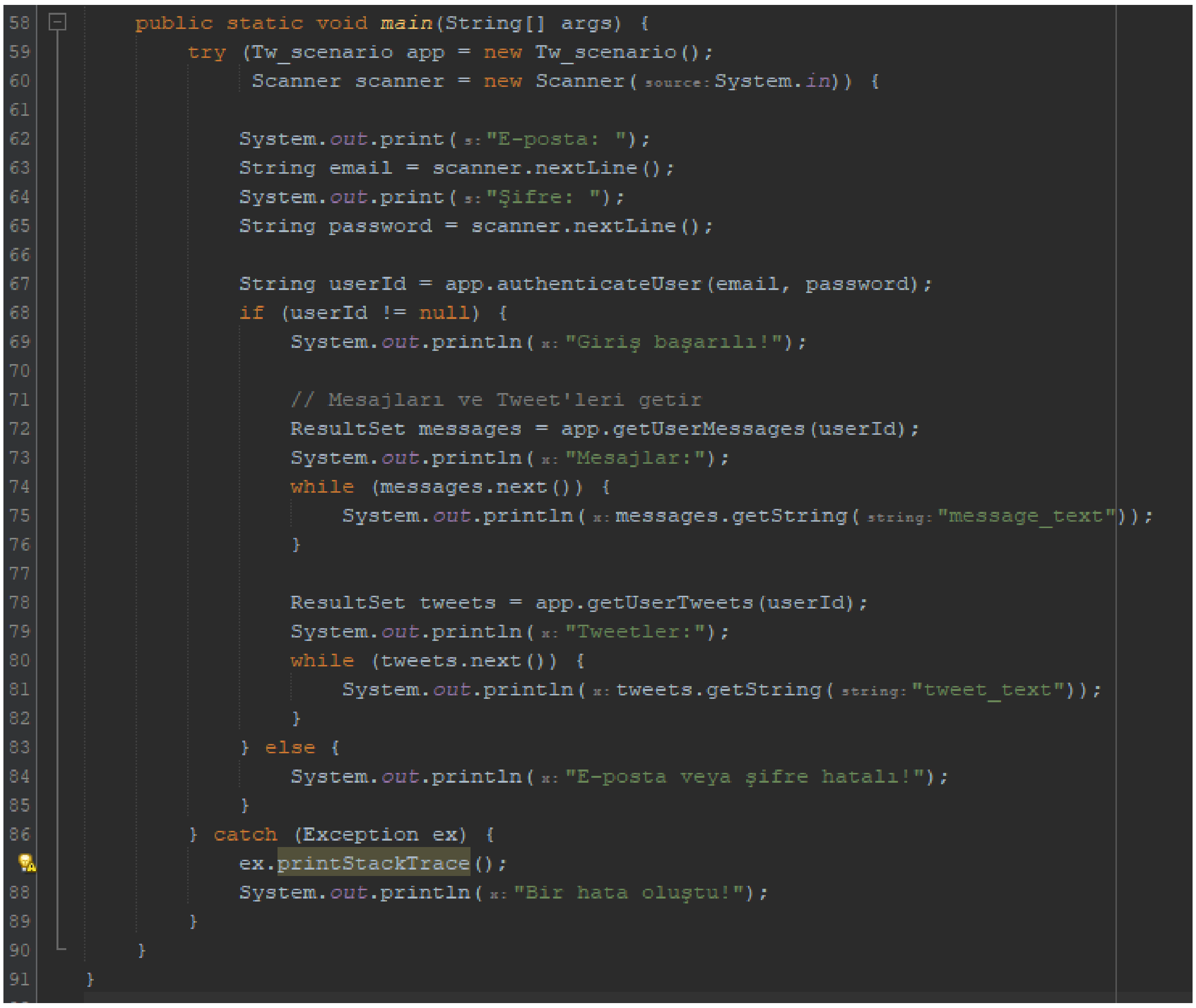
```java
    public ResultSet getUserMessages(String userId) throws SQLException {
        String sql = "SELECT * FROM messages WHERE sender_id = ? OR receiver_id = ?";
        PreparedStatement pstmt = conn.prepareStatement( string: sql);
        pstmt.setString( i: 1,  string: userId);
        pstmt.setString( i: 2,  string: userId);
        return pstmt.executeQuery();
    }
```

This method retrieves messages belonging to the given userId from the database. The sql variable contains the SQL query. Here ? signs represent parameters. Using PreparedStatement reduces the risk of SQL injection and makes the query safer. pstmt.setString(1, userId); and pstmt.setString(2, userId); rows for both ? rows in the SQL query. It also assigns the userId to the parameter. pstmt.executeQuery(); runs the query and returns the results as ResultSet.



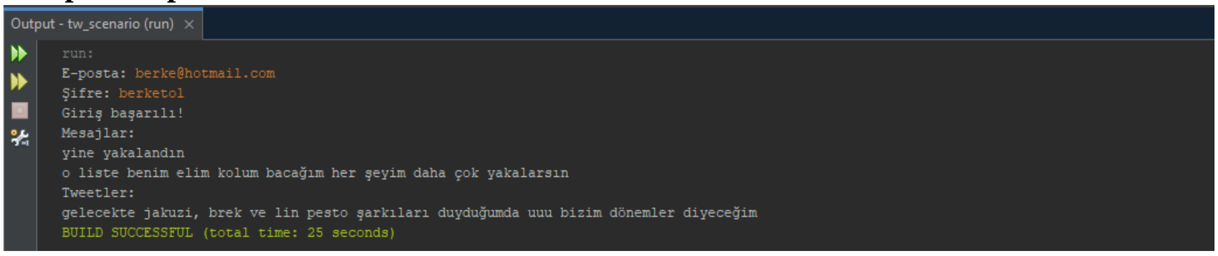
```java
    public ResultSet getUserTweets(String userId) throws SQLException {
        String sql = "SELECT * FROM tweets WHERE user_id = ?";
        PreparedStatement pstmt = conn.prepareStatement( string: sql);
        pstmt.setString( i: 1,  string: userId);
        return pstmt.executeQuery();
    }
```

This method retrieves tweets belonging to the specified userId from the database. Again, the SQL query is prepared using PreparedStatement and the userId parameter is added to the query. pstmt.executeQuery(); runs the query and returns results.


```java
    public static void main(String[] args) {
        try (Tw_scenario app = new Tw_scenario();
             Scanner scanner = new Scanner(source: System.in)) {

            System.out.print(s: "E-posta: ");
            String email = scanner.nextLine();
            System.out.print(s: "Şifre: ");
            String password = scanner.nextLine();

            String userId = app.authenticateUser(email, password);
            if (userId != null) {
                System.out.println(x: "Giriş başarılı!");

                // Mesajları ve Tweet'leri getir
                ResultSet messages = app.getUserMessages(userId);
                System.out.println(x: "Mesajlar:");
                while (messages.next()) {
                    System.out.println(x: messages.getString(string: "message_text"));
                }

                ResultSet tweets = app.getUserTweets(userId);
                System.out.println(x: "Tweetler:");
                while (tweets.next()) {
                    System.out.println(x: tweets.getString(string: "tweet_text"));
                }
            } else {
                System.out.println(x: "E-posta veya şifre hatalı!");
            }
        } catch (Exception ex) {
            ex.printStackTrace();
            System.out.println(x: "Bir hata oluştu!");
        }
    }
}
```

Main method is the main entry point of the application. Creates an instance of the tw_scenario class and receives email and password input from the user using the Scanner class. The user's email and password are sent to the authenticateUser method and a userId is returned as a result of the verification. If userId is not null (i.e. the user is successfully verified), the user's messages and tweets are retrieved using the getUserMessages and getUserTweets methods. As a result, the user's messages and tweets are printed on the screen. This is accomplished by looping over ResultSet objects. At each iteration, the next record is taken from the ResultSet and the relevant data (message_text or tweet_text) is printed to the screen. If user authentication fails, "Email or password is incorrect!" message is displayed. All these operations are performed within a try-with-resources block. This ensures that in the event of a possible SQLException, the error will be caught and the Tw_scenario instance of the program will be properly closed.

Sample Output



```
run:
E-posta: berke@hotmail.com
Şifre: berketol
Giriş başarılı!
Mesajlar:
yine yakalandın
o liste benim elim kolum bacağım her şeyim daha çok yakalarsın
Tweetler:
gelecekte jakuzi, brek ve lin pesto şarkıları duyduğumda uuu bizim dönemler diyeceğim
BUILD SUCCESSFUL (total time: 25 seconds)
```

Note
These methods demonstrate basic JDBC operations for retrieving user data from the database and processing this data. In terms of security and performance, more checks and optimizations may be required in a real application.