COMP 305 - DATABASE MANAGEMENT SYSTEMS

DATABASE OF A SOCIAL MEDIA WEBSITE

Akif Eren Kavak - 042201004

Ezgi Ulusan - 042201033
Sinem Özbey - 042101122

1. Introduction

We created a database for an Ekşisözlük-like website. First we determined the datas that we will need in our site in particular captions. These are: user datas, entry datas, comment datas, message datas, saved entries datas, channel datas, like datas and report datas. We determined the attributes of each of them and determined the relations between them.

We used toad for oracle as database manager and used oracle for creating the database.

Our database codes includes creating the necessary tables with their attributes, creating the relations between that tables, inputting, altering and deleting rows to/from these tables, triggering a transaction table for holding the summary of entried datas.

2. Data Identification

In that part you will introduced with the attributes of the tables and which data types we set for them.

2.1. Users Table

That table holds user's basic datas for entering to the website.

in the most left column you see the names of the attributes, right side of it there are data types of that attributes and nullability condition and primary/foreign key part is following them.

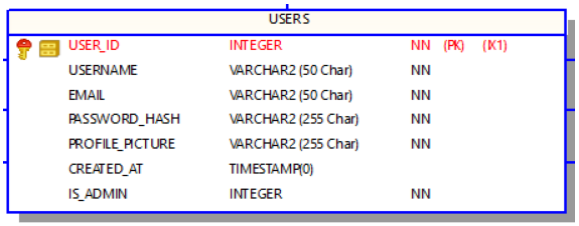

Fig 1. Users Table

2.2. Entries Table

Entries are the captions in the website each of them have a content and user_id of the user who entered it. All of them are categorized under a channel. We used clob for holding the content because it might be a big string and clobs are good choice for it.

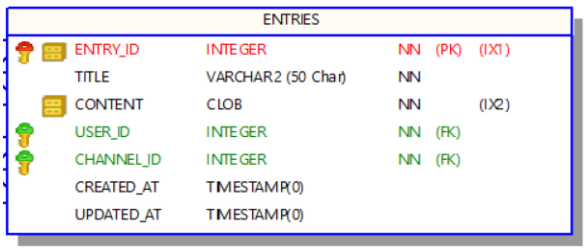

Fig 2. Entries Table

2.3. Channels Table

That is the table where our entries will be categorised, we are holding the ids, names and descriptions of the channels in that table.

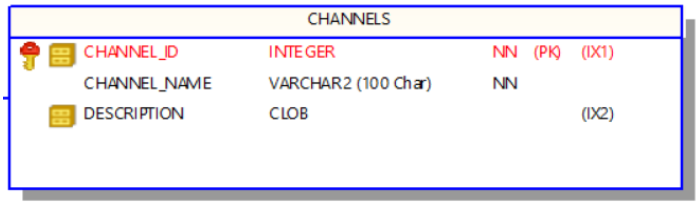

Fig 3. Channels Table

2.4. Comments Table

Comments will be placed under the entries. Each comment will be placed under one entry and we will hold the data of the author's ID, content and when the comment is created.

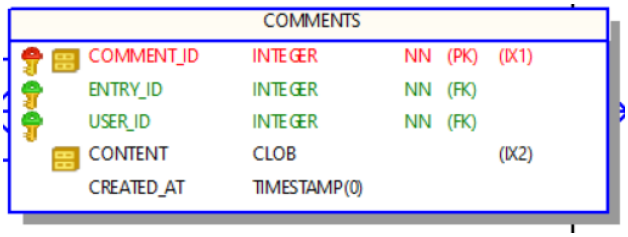

Fig 4. Comments Table

2.5. Likes Table

Users will be allowed to like the entries. We wanted to know which user liked what so that we can create a liked part specifically for each user. And also we can use dynamic queries for calculating each entry's like count.

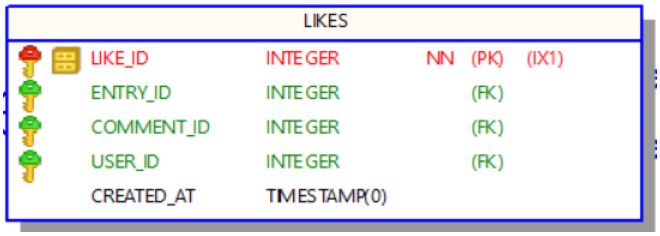

Fig 5. Likes Table

2.6. Saved Entries Table

Each user will be allowed to put a save mark to any entry so that they can achieve to the entries they wanted to see more after.

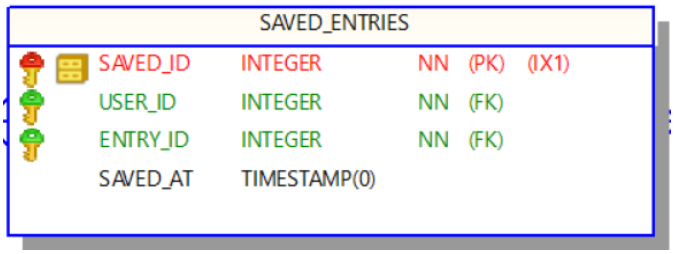

Fig 6. Saved Entries Table

2.7. Reports Table

If an entry violates our policies users will be able to report them and admins will evaluate them for a healthier community.

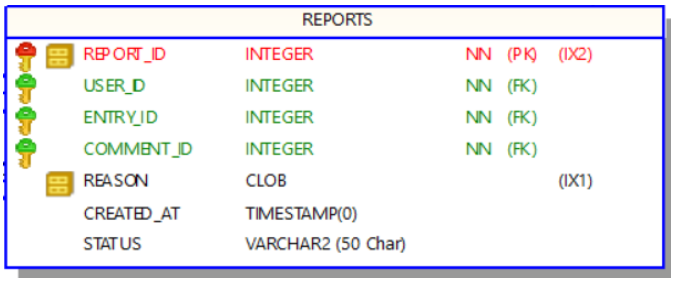

Fig 7. Reports Table

2.8. Messages Table

Users will be able to send messages to any other user. Our database will hold the messages' informations like when did they sent, received, what is the content, are they readed etc.

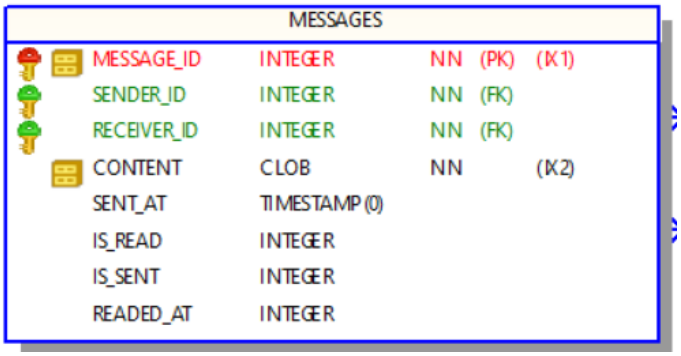

Fig 8. Messages Table

2.9. Database Schema and Relations

Some tables are holding the same attributes and in order to prevent data repetition we used foreign keys and linked the tables.

They can be observed on our ER diagram.

Fig 9. Database Schema and Relations

Here are the foreign keys at the each table and their parent tables:

2.10. Entries Table

- **user_id → Users(user_id)**
  The user_id in the Entries table references the user_id in the Users table.
- **channel_id → Channels(channel_id)**
  The channel_id in the Entries table references the channel_id in the Channels table.

2.11. Comments Table

- **entry_id → Entries(entry_id)**
  The entry_id in the Comments table references the entry_id in the Entries table.
- **user_id → Users(user_id)**
  The user_id in the Comments table references the user_id in the Users table.

2.12. Likes Table

- **entry_id → Entries(entry_id)** (nullable)
  The entry_id in the Likes table references the entry_id in the Entries table.
- **comment_id → Comments(comment_id)** (nullable)
  The comment_id in the Likes table references the comment_id in the Comments table.

- **user_id → Users(user_id)**
  The user_id in the Likes table references the user_id in the Users table.

2.13. Reports Table

- **user_id → Users(user_id)**
  The user_id in the Reports table references the user_id in the Users table.
- **entry_id → Entries(entry_id)** (nullable)
  The entry_id in the Reports table references the entry_id in the Entries table.
- **comment_id → Comments(comment_id)** (nullable)
  The comment_id in the Reports table references the comment_id in the Comments table.

2.14. Tags Table

- **entry_id → Entries(entry_id)**
  The entry_id in the Tags table references the entry_id in the Entries table.

2.15. Saved_Entries Table

- **user_id → Users(user_id)**
  The user_id in the Saved_Entries table references the user_id in the Users table.
- **entry_id → Entries(entry_id)**
  The entry_id in the Saved_Entries table references the entry_id in the Entries table.

2.16.Messages Table

- **sender_id → Users(user_id)**
  The sender_id in the Messages table references the user_id in the Users table.
- **receiver_id → Users(user_id)**
  The receiver_id in the Messages table references the user_id in the Users table.

3. IMPLEMENTATION

3.1. Creating Tables

We used toad for creating the tables automatically. Then we fetched the codes to a txt file so that we can create the tables at one code file when it is necessary.

3.2. Data Management

We created 3 packages for adding, altering and deleting the rows of the tables. And 1 package for deleting the duplicates in the database. Here are the body procedures of the user table for deleting altering, adding and removing duplicates as an example:

3.3. Inserting Row

```
CREATE OR REPLACE PACKAGE BODY insert_pkg AS

    -- USERS
    PROCEDURE insert_into_users(
        p_user_id IN NUMBER,
        p_username IN VARCHAR2,
        p_email IN VARCHAR2,
        p_password_hash IN VARCHAR2,
        p_profile_picture IN VARCHAR2,
        p_is_admin IN NUMBER
    ) IS
    BEGIN
        INSERT INTO C##PROJECT_USER.USERS (
            USER_ID, USERNAME, EMAIL, PASSWORD_HASH, PROFILE_PICTURE, CREATED_AT, IS_ADMIN
        ) VALUES (
            p_user_id,
            p_username,
            p_email,
            p_password_hash,
            p_profile_picture,
            SYSTIMESTAMP,
            p_is_admin
        );
        DBMS_OUTPUT.PUT_LINE('User inserted successfully: ' || p_username);
    END insert_into_users;
```

Fig 10. Inserting Row

3.4. Updating Row

```
CREATE OR REPLACE PACKAGE BODY update_pkg AS

    -- USERS Table
    PROCEDURE update_users(
        p_user_id IN NUMBER,
        p_username IN VARCHAR2,
        p_email IN VARCHAR2,
        p_password_hash IN VARCHAR2,
        p_profile_picture IN VARCHAR2,
        p_is_admin IN NUMBER
    ) IS
    BEGIN
        UPDATE C##PROJECT_USER.USERS
        SET USERNAME = p_username,
            EMAIL = p_email,
            PASSWORD_HASH = p_password_hash,
            PROFILE_PICTURE = p_profile_picture,
            IS_ADMIN = p_is_admin
        WHERE USER_ID = p_user_id;

        DBMS_OUTPUT.PUT_LINE('User updated successfully: ' || p_user_id);
    END update_users;
```

Fig 11. Updating Row

3.5. Deleting Row

```
CREATE OR REPLACE PACKAGE BODY delete_pkg AS

    -- USERS -
    PROCEDURE delete_from_users(p_user_id IN NUMBER) IS
    BEGIN
        DELETE FROM C##PROJECT_USER.USERS WHERE USER_ID = p_user_id;
        DBMS_OUTPUT.PUT_LINE('User deleted successfully: ' || p_user_id);
    END delete_from_users;

    -- CHANNELS -
    PROCEDURE delete_from_channels(p_channel_id IN NUMBER) IS
    BEGIN
        DELETE FROM C##PROJECT_USER.CHANNELS WHERE CHANNEL_ID = p_channel_id;
        DBMS_OUTPUT.PUT_LINE('Channel deleted successfully: ' || p_channel_id);
    END delete_from_channels;

    -- ENTRIES -
    PROCEDURE delete_from_entries(p_entry_id IN NUMBER) IS
    BEGIN
        DELETE FROM C##PROJECT_USER.ENTRIES WHERE ENTRY_ID = p_entry_id;
        DBMS_OUTPUT.PUT_LINE('Entry deleted successfully: ' || p_entry_id);
    END delete_from_entries;
```

Fig 12. Deleting Row

The structure of the code is the same for other tables just the attribute names are differs and all of the codes can be achieved from the github link at the bottom of that report.

3.6. Deleting Duplicates

```
CREATE OR REPLACE PACKAGE BODY delete_duplicates_pkg AS

    -- Delete USERS Duplicates
    PROCEDURE delete_duplicates_from_users IS
    BEGIN
        DELETE FROM C##PROJECT_USER.USERS
        WHERE user_id IN (
            SELECT user_id
            FROM (
                SELECT user_id, ROW_NUMBER() OVER (PARTITION BY username, email ORDER BY user_id) AS row_num
                FROM C##PROJECT_USER.USERS
            )
            WHERE row_num > 1
        );
        DBMS_OUTPUT.PUT_LINE('Duplicate users deleted successfully.');
    END delete_duplicates_from_users;
```

Fig 12. Deleting Duplicates

That procedure deletes the instances with the same attributes except user_id. Here is an example usage of it:

USER_ID	USERNAME	EMAIL	PASSWORD_HASH	PROFILE_PICTURE	CREATED_AT	IS_ADMIN
2001	DuplicateUser	duplicateuser@example.com	hashed_password_here	profile_picture_url_here	13/12/2024 17:03:07	0
2002	DuplicateUser	duplicateuser@example.com	hashed_password_here	profile_picture_url_here	13/12/2024 17:03:07	0

Fig 13. User instances with the same attributes except user_id

```
1  BEGIN
2      delete_duplicates_pkg.delete_duplicates_from_users;
3  END;
4
```

Fig 14. calling of delete_duptlicates procedure

USER_ID	USERNAME	EMAIL	PASSWORD_HASH	PROFILE_PICTURE	CREATED_AT	IS_ADMIN
2001	DuplicateUser	duplicateuser@example.com	hashed_password_here	profile_picture_url_here	13/12/2024 17:03:07	0

Fig 15. Unduplicated version of the table

4.TRANSACTION MANAGEMENT

For keeping the log of data inserting we created a transaction table and wrote a trigger. Below you can see the structure of that table and a code snippet triggers when a new row inserted into comments table.

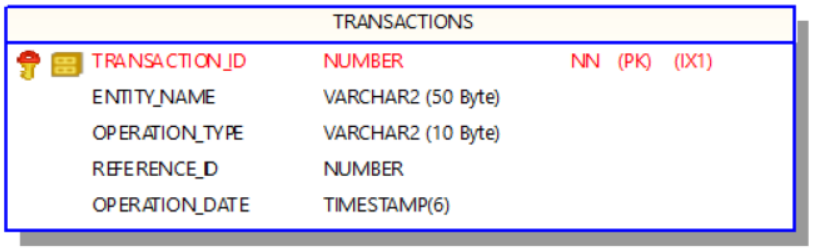

Fig 16. Transactions table

```
CREATE OR REPLACE TRIGGER comments_after_insert_trigger
AFTER INSERT ON C##PROJECT_USER.COMMENTS
FOR EACH ROW
BEGIN
    INSERT INTO C##PROJECT_USER.TRANSACTIONS (
        ENTITY_NAME,
        OPERATION_TYPE,
        REFERENCE_ID,
        OPERATION_DATE
    ) VALUES (
        'COMMENTS', -- Operated table
        'INSERT', -- operation type
        :NEW.COMMENT_ID, -- COMMENT_ID of inserted comment
        SYSTIMESTAMP -- Operation time
    );
END;
/
```

Fig 17. Trigger for inserting comment

Here how the transaction table's products looks like:

TRANSACTION_ID	ENTITY_NAME	OPERATION_TYPE	REFERENCE_ID	OPERATION_DATE
1	USERS	INSERT	9	6/12/2024 11:32:42,904000
4	ENTRIES	INSERT	1	6/12/2024 12:07:55,507000
5	COMMENTS	INSERT	11	6/12/2024 12:08:11,761000
6	COMMENTS	INSERT	12	6/12/2024 12:08:29,272000
7	USERS	INSERT	4	6/12/2024 13:07:42,184000
8	USERS	INSERT	44	6/12/2024 13:15:16,136000
9	USERS	INSERT	4	6/12/2024 13:20:25,620000

Fig 18. Transaction table's output

5. RESULTS

In that part you'll see some examples of how the procedures are called and how do we see the results that we made on the tables by the help of SQL queries.

5.1.Insert Procedure

```
BEGIN
    insert_pkg.insert_into_users(
        p_user_id => 1283,
        p_username => 'Mustafa Kemal Ataturk',
        p_email => 'mustafa.kemal@example.com',
        p_password_hash => 'hashed_password_here',
        p_profile_picture => 'profile_picture_url_here',
        p_is_admin => 1
    );
END;
```

Fig 19. Usage of insert_user procedure

```
Select *
From C##PROJECT_USER.USERS
Where user_id = 1283;
```

Fig 20. SQL query for specified user

USER_ID	USERNAME	EMAIL	PASSWORD_HASH	PROFILE_PICTURE	CREATED_AT	IS_ADMIN
1283	Mustafa Kemal Ataturk	mustafa.kemal@example.com	hashed_password_here	profile_picture_url_here	13/12/2024 16:48:03	1

Fig 21. Output of query

5.2.Update Procedur

```
BEGIN
    update_pkg.update_users(
        p_user_id => 1283,
        p_username => 'Mustafa Kemal Ataturk',
        p_email => 'new_email@example.com',
        p_password_hash => 'hashed_password_here',
        p_profile_picture => 'profile_picture_url_here',
        p_is_admin => 0
    );
END;
```

Fig 22. Usage of update_user procedure

```
Select *
From C##PROJECT_USER.USERS
Where email = 'new_email@example.com';
```

Fig 23. SQL query for specified user

USER_ID	USERNAME	EMAIL	PASSWORD_HASH	PROFILE_PICTURE	CREATED_AT	IS_ADMIN
1283	Mustafa Kemal Ataturk	new_email@example.com	hashed_password_here	profile_picture_url_here	13/12/2024 16:48:03	0

Fig 24. Output of query

5.3.Delete Procedure

```
BEGIN
    delete_pkg.delete_from_users(
        p_user_id => 1283
    );
END;
```

Fig 25. Usage of delete_user procedure

```
Select *
From C##PROJECT_USER.USERS
Where email = 'new_email@example.com';
```

Fig 26. SQL query for specified user

USER_ID	USERNAME	EMAIL	PASSWORD_HASH	PROFILE_PICTURE	CREATED_AT	IS_ADMIN

Fig 27. Output of query

6. CONCLUSION, TROUBLED POINTS

Project proceed very well. We can have a well-functioning database with the following codes very easily. And the database can be used by the help of some libraries like flask, for using at any backend part of website.

There were some points at the project we struggled. When using Toad for oracle you need to be careful about **schemas.** In Toad Schema is basically a user in the database and by default Toad opens the database with Sys schema which is not a good environment for developing the project. Sys schema does not allow you to use triggers and there are lots of side tables in it which are for database's own functioning. So when a project starts first thing first a new schema-user should be created and granted with the necessary permissions.

7.CODES

In this part you can find all the codes we've used for tables, packages and triggers

7.1.Table Codes

```
DROP TABLE C##PROJECT_USER.USERS CASCADE CONSTRAINTS;

CREATE TABLE C##PROJECT_USER.USERS
(
  USER_ID         INTEGER                   NOT NULL,
  USERNAME        VARCHAR2(50 CHAR)         NOT NULL,
  EMAIL           VARCHAR2(50 CHAR)         NOT NULL,
  PASSWORD_HASH   VARCHAR2(255 CHAR)        NOT NULL,
  PROFILE_PICTURE VARCHAR2(255 CHAR)        NOT NULL,
  CREATED_AT      TIMESTAMP(0),
  IS_ADMIN        INTEGER                   NOT NULL
)
TABLESPACE USERS
PCTFREE    10
INITRANS   1
MAXTRANS   255
STORAGE    (
           PCTINCREASE      0
           BUFFER_POOL      DEFAULT
           )
LOGGING
NOCOMPRESS
NOCACHE;


CREATE UNIQUE INDEX C##PROJECT_USER.USERS_PK ON C##PROJECT_USER.USERS
(USER_ID)
LOGGING
TABLESPACE USERS
PCTFREE    10
INITRANS   2
```

```
MAXTRANS   255
STORAGE    (
           PCTINCREASE      0
           BUFFER_POOL      DEFAULT
          );

ALTER TABLE C##PROJECT_USER.USERS ADD (
  CONSTRAINT USERS_PK
  PRIMARY KEY
  (USER_ID)
  USING INDEX C##PROJECT_USER.USERS_PK
  ENABLE VALIDATE);



DROP TABLE C##PROJECT_USER.CHANNELS CASCADE CONSTRAINTS;

CREATE TABLE C##PROJECT_USER.CHANNELS
(
  CHANNEL_ID    INTEGER                     NOT NULL,
  CHANNEL_NAME  VARCHAR2(100 CHAR)          NOT NULL,
  DESCRIPTION   CLOB
)
LOB (DESCRIPTION) STORE AS SECUREFILE (
  TABLESPACE  USERS
  ENABLE      STORAGE IN ROW
  CHUNK       8192
  RETENTION
  NOCACHE
  LOGGING)
TABLESPACE USERS
PCTFREE    10
INITRANS   1
MAXTRANS   255
STORAGE    (
           PCTINCREASE      0
           BUFFER_POOL      DEFAULT
          )
LOGGING
NOCOMPRESS
NOCACHE;


CREATE UNIQUE INDEX C##PROJECT_USER.CHANNELS_PK ON C##PROJECT_USER.CHANNELS
(CHANNEL_ID)
LOGGING
TABLESPACE USERS
PCTFREE    10
INITRANS   2
MAXTRANS   255
STORAGE    (
           PCTINCREASE      0
```

```
        BUFFER_POOL      DEFAULT
        );

ALTER TABLE C##PROJECT_USER.CHANNELS ADD (
  CONSTRAINT CHANNELS_PK
  PRIMARY KEY
  (CHANNEL_ID)
  USING INDEX C##PROJECT_USER.CHANNELS_PK
  ENABLE VALIDATE);

DROP TABLE C##PROJECT_USER.ENTRIES CASCADE CONSTRAINTS;

CREATE TABLE C##PROJECT_USER.ENTRIES
(
  ENTRY_ID    INTEGER                       NOT NULL,
  TITLE       VARCHAR2(50 CHAR)             NOT NULL,
  CONTENT     CLOB                          NOT NULL,
  USER_ID     INTEGER                       NOT NULL,
  CHANNEL_ID  INTEGER                       NOT NULL,
  CREATED_AT  TIMESTAMP(0),
  UPDATED_AT  TIMESTAMP(0)
)
LOB (CONTENT) STORE AS SECUREFILE (
  TABLESPACE  USERS
  ENABLE      STORAGE IN ROW
  CHUNK       8192
  RETENTION
  NOCACHE
  LOGGING)
TABLESPACE USERS
PCTFREE    10
INITRANS   1
MAXTRANS   255
STORAGE    (
        PCTINCREASE      0
        BUFFER_POOL      DEFAULT
        )
LOGGING
NOCOMPRESS
NOCACHE;

CREATE UNIQUE INDEX C##PROJECT_USER.ENTRIES_PK ON C##PROJECT_USER.ENTRIES
(ENTRY_ID)
LOGGING
TABLESPACE USERS
PCTFREE    10
INITRANS   2
MAXTRANS   255
STORAGE    (
        PCTINCREASE      0
        BUFFER_POOL      DEFAULT
        );
```

```
ALTER TABLE C##PROJECT_USER.ENTRIES ADD (
  CONSTRAINT ENTRIES_PK
  PRIMARY KEY
  (ENTRY_ID)
  USING INDEX C##PROJECT_USER.ENTRIES_PK
  ENABLE VALIDATE);

ALTER TABLE C##PROJECT_USER.ENTRIES ADD (
  FOREIGN KEY (CHANNEL_ID)
  REFERENCES C##PROJECT_USER.CHANNELS (CHANNEL_ID)
  ENABLE VALIDATE
,  FOREIGN KEY (USER_ID)
  REFERENCES C##PROJECT_USER.USERS (USER_ID)
  ENABLE VALIDATE);


DROP TABLE C##PROJECT_USER.COMMENTS CASCADE CONSTRAINTS;

CREATE TABLE C##PROJECT_USER.COMMENTS
(
  COMMENT_ID  INTEGER                    NOT NULL,
  ENTRY_ID    INTEGER                    NOT NULL,
  USER_ID     INTEGER                    NOT NULL,
  CONTENT     CLOB,
  CREATED_AT  TIMESTAMP(0)
)
LOB (CONTENT) STORE AS SECUREFILE (
  TABLESPACE  USERS
  ENABLE      STORAGE IN ROW
  CHUNK       8192
  RETENTION
  NOCACHE
  LOGGING)
TABLESPACE USERS
PCTFREE    10
INITRANS   1
MAXTRANS   255
STORAGE    (
          PCTINCREASE      0
          BUFFER_POOL      DEFAULT
         )
LOGGING
NOCOMPRESS
NOCACHE;

CREATE UNIQUE INDEX C##PROJECT_USER.COMMENTS_PK ON C##PROJECT_USER.COMMENTS
(COMMENT_ID)
LOGGING
TABLESPACE USERS
PCTFREE    10
INITRANS   2
```

```
MAXTRANS  255
STORAGE   (
          PCTINCREASE     0
          BUFFER_POOL     DEFAULT
         );

ALTER TABLE C##PROJECT_USER.COMMENTS ADD (
 CONSTRAINT COMMENTS_PK
 PRIMARY KEY
 (COMMENT_ID)
 USING INDEX C##PROJECT_USER.COMMENTS_PK
 ENABLE VALIDATE);


ALTER TABLE C##PROJECT_USER.COMMENTS ADD (
 FOREIGN KEY (ENTRY_ID)
 REFERENCES C##PROJECT_USER.ENTRIES (ENTRY_ID)
 ENABLE VALIDATE
,  FOREIGN KEY (USER_ID)
 REFERENCES C##PROJECT_USER.USERS (USER_ID)
 ENABLE VALIDATE);

DROP TABLE C##PROJECT_USER.LIKES CASCADE CONSTRAINTS;

CREATE TABLE C##PROJECT_USER.LIKES
(
 LIKE_ID    INTEGER,
 ENTRY_ID   INTEGER,
 COMMENT_ID  INTEGER,
 USER_ID    INTEGER,
 CREATED_AT  TIMESTAMP(0)
)
TABLESPACE USERS
PCTFREE   10
INITRANS  1
MAXTRANS  255
STORAGE   (
          PCTINCREASE     0
          BUFFER_POOL     DEFAULT
         )
LOGGING
NOCOMPRESS
NOCACHE;


CREATE UNIQUE INDEX C##PROJECT_USER.LIKES_PK ON C##PROJECT_USER.LIKES
(LIKE_ID)
LOGGING
TABLESPACE USERS
PCTFREE   10
INITRANS  2
MAXTRANS  255
```

```
STORAGE    (
        PCTINCREASE     0
        BUFFER_POOL     DEFAULT
        );

ALTER TABLE C##PROJECT_USER.LIKES ADD (
  CONSTRAINT LIKES_PK
  PRIMARY KEY
  (LIKE_ID)
  USING INDEX C##PROJECT_USER.LIKES_PK
  ENABLE VALIDATE);


ALTER TABLE C##PROJECT_USER.LIKES ADD (
  FOREIGN KEY (ENTRY_ID)
  REFERENCES C##PROJECT_USER.ENTRIES (ENTRY_ID)
  ENABLE VALIDATE
,  FOREIGN KEY (COMMENT_ID)
  REFERENCES C##PROJECT_USER.COMMENTS (COMMENT_ID)
  ENABLE VALIDATE
,  FOREIGN KEY (USER_ID)
  REFERENCES C##PROJECT_USER.USERS (USER_ID)
  ENABLE VALIDATE);


DROP TABLE C##PROJECT_USER.REPORTS CASCADE CONSTRAINTS;

CREATE TABLE C##PROJECT_USER.REPORTS
(
  REPORT_ID   INTEGER                    NOT NULL,
  USER_ID     INTEGER                   NOT NULL,
  ENTRY_ID    INTEGER                    NOT NULL,
  COMMENT_ID  INTEGER                        NOT NULL,
  REASON      CLOB,
  CREATED_AT  TIMESTAMP(0),
  STATUS      VARCHAR2(50 CHAR)
)
LOB (REASON) STORE AS SECUREFILE (
  TABLESPACE  USERS
  ENABLE      STORAGE IN ROW
  CHUNK       8192
  RETENTION
  NOCACHE
  LOGGING)
TABLESPACE USERS
PCTFREE    10
INITRANS   1
MAXTRANS   255
STORAGE    (
        PCTINCREASE     0
        BUFFER_POOL     DEFAULT
        )
```

```
LOGGING
NOCOMPRESS
NOCACHE;


CREATE UNIQUE INDEX C##PROJECT_USER.REPORTS_PK ON C##PROJECT_USER.REPORTS
(REPORT_ID)
LOGGING
TABLESPACE USERS
PCTFREE    10
INITRANS   2
MAXTRANS   255
STORAGE    (
            PCTINCREASE      0
            BUFFER_POOL      DEFAULT
           );

ALTER TABLE C##PROJECT_USER.REPORTS ADD (
  CONSTRAINT REPORTS_PK
  PRIMARY KEY
  (REPORT_ID)
  USING INDEX C##PROJECT_USER.REPORTS_PK
  ENABLE VALIDATE);


ALTER TABLE C##PROJECT_USER.REPORTS ADD (
  FOREIGN KEY (USER_ID)
  REFERENCES C##PROJECT_USER.USERS (USER_ID)
  ENABLE VALIDATE
,  FOREIGN KEY (ENTRY_ID)
  REFERENCES C##PROJECT_USER.ENTRIES (ENTRY_ID)
  ENABLE VALIDATE
,  FOREIGN KEY (COMMENT_ID)
  REFERENCES C##PROJECT_USER.COMMENTS (COMMENT_ID)
  ENABLE VALIDATE);

DROP TABLE C##PROJECT_USER.SAVED_ENTRIES CASCADE CONSTRAINTS;

CREATE TABLE C##PROJECT_USER.SAVED_ENTRIES
(
  SAVED_ID  INTEGER                     NOT NULL,
  USER_ID   INTEGER                    NOT NULL,
  ENTRY_ID  INTEGER                     NOT NULL,
  SAVED_AT  TIMESTAMP(0)
)
TABLESPACE USERS
PCTFREE    10
INITRANS   1
MAXTRANS   255
STORAGE    (
            PCTINCREASE      0
            BUFFER_POOL      DEFAULT
```

```
        )
LOGGING
NOCOMPRESS
NOCACHE;

CREATE UNIQUE INDEX C##PROJECT_USER.SAVED_ENTRIES_PK ON
C##PROJECT_USER.SAVED_ENTRIES
(SAVED_ID)
LOGGING
TABLESPACE USERS
PCTFREE    10
INITRANS   2
MAXTRANS   255
STORAGE    (
            PCTINCREASE      0
            BUFFER_POOL      DEFAULT
           );

ALTER TABLE C##PROJECT_USER.SAVED_ENTRIES ADD (
  CONSTRAINT SAVED_ENTRIES_PK
  PRIMARY KEY
  (SAVED_ID)
  USING INDEX C##PROJECT_USER.SAVED_ENTRIES_PK
  ENABLE VALIDATE);

ALTER TABLE C##PROJECT_USER.SAVED_ENTRIES ADD (
  FOREIGN KEY (USER_ID)
  REFERENCES C##PROJECT_USER.USERS (USER_ID)
  ENABLE VALIDATE
,  FOREIGN KEY (ENTRY_ID)
  REFERENCES C##PROJECT_USER.ENTRIES (ENTRY_ID)
  ENABLE VALIDATE);

DROP TABLE C##PROJECT_USER.MESSAGES CASCADE CONSTRAINTS;

CREATE TABLE C##PROJECT_USER.MESSAGES
(
  MESSAGE_ID   INTEGER                    NOT NULL,
  SENDER_ID    INTEGER                   NOT NULL,
  RECEIVER_ID  INTEGER                    NOT NULL,
  CONTENT      CLOB                      NOT NULL,
  SENT_AT      TIMESTAMP(0),
  IS_READ      INTEGER,
  IS_SENT      INTEGER,
  READED_AT    INTEGER
)
LOB (CONTENT) STORE AS SECUREFILE (
  TABLESPACE  USERS
  ENABLE      STORAGE IN ROW
  CHUNK       8192
  RETENTION
  NOCACHE
```

```
  LOGGING)
TABLESPACE USERS
PCTFREE    10
INITRANS   1
MAXTRANS   255
STORAGE    (
            PCTINCREASE      0
            BUFFER_POOL      DEFAULT
           )
LOGGING
NOCOMPRESS
NOCACHE;


CREATE UNIQUE INDEX C##PROJECT_USER.MESSAGES_PK ON C##PROJECT_USER.MESSAGES
(MESSAGE_ID)
LOGGING
TABLESPACE USERS
PCTFREE    10
INITRANS   2
MAXTRANS   255
STORAGE    (
            PCTINCREASE      0
            BUFFER_POOL      DEFAULT
           );

ALTER TABLE C##PROJECT_USER.MESSAGES ADD (
  CONSTRAINT MESSAGES_PK
  PRIMARY KEY
  (MESSAGE_ID)
  USING INDEX C##PROJECT_USER.MESSAGES_PK
  ENABLE VALIDATE);


ALTER TABLE C##PROJECT_USER.MESSAGES ADD (
  FOREIGN KEY (SENDER_ID)
  REFERENCES C##PROJECT_USER.USERS (USER_ID)
  ENABLE VALIDATE
,  FOREIGN KEY (RECEIVER_ID)
  REFERENCES C##PROJECT_USER.USERS (USER_ID)
  ENABLE VALIDATE);
```

7.2.Package Codes

7.2.1.Insert Package

```
CREATE OR REPLACE PACKAGE insert_pkg AS
   -- USERS
```

```
PROCEDURE insert_into_users(
    p_user_id IN NUMBER,
    p_username IN VARCHAR2,
    p_email IN VARCHAR2,
    p_password_hash IN VARCHAR2,
    p_profile_picture IN VARCHAR2,
    p_is_admin IN NUMBER
);

-- CHANNELS
PROCEDURE insert_into_channels(
    p_channel_id IN NUMBER,
    p_channel_name IN VARCHAR2,
    p_description IN CLOB
);

-- ENTRIES
PROCEDURE insert_into_entries(
    p_entry_id IN NUMBER,
    p_title IN VARCHAR2,
    p_content IN CLOB,
    p_user_id IN NUMBER,
    p_channel_id IN NUMBER
);

-- MESSAGES
PROCEDURE insert_into_messages(
    p_message_id IN NUMBER,
    p_sender_id IN NUMBER,
    p_receiver_id IN NUMBER,
    p_content IN CLOB,
    p_is_read IN NUMBER
);

-- COMMENTS
PROCEDURE insert_into_comments(
    p_comment_id IN NUMBER,
    p_entry_id IN NUMBER,
    p_user_id IN NUMBER,
    p_content IN CLOB
);

-- LIKES
PROCEDURE insert_into_likes(
    p_like_id IN NUMBER,
    p_entry_id IN NUMBER,
    p_comment_id IN NUMBER,
    p_user_id IN NUMBER
);

-- REPORTS
PROCEDURE insert_into_reports(
```

```
        p_report_id IN NUMBER,
        p_user_id IN NUMBER,
        p_entry_id IN NUMBER,
        p_comment_id IN NUMBER,
        p_reason IN CLOB,
        p_status IN VARCHAR2
    );

    -- SAVED_ENTRIES
    PROCEDURE insert_into_saved_entries(
        p_saved_id IN NUMBER,
        p_user_id IN NUMBER,
        p_entry_id IN NUMBER
    );
END insert_pkg;
/


CREATE OR REPLACE PACKAGE BODY insert_pkg AS

    -- USERS
    PROCEDURE insert_into_users(
        p_user_id IN NUMBER,
        p_username IN VARCHAR2,
        p_email IN VARCHAR2,
        p_password_hash IN VARCHAR2,
        p_profile_picture IN VARCHAR2,
        p_is_admin IN NUMBER
    ) IS
    BEGIN
        INSERT INTO C##PROJECT_USER.USERS (
            USER_ID, USERNAME, EMAIL, PASSWORD_HASH, PROFILE_PICTURE, CREATED_AT,
IS_ADMIN
        ) VALUES (
            p_user_id,
            p_username,
            p_email,
            p_password_hash,
            p_profile_picture,
            SYSTIMESTAMP,
            p_is_admin
        );
        DBMS_OUTPUT.PUT_LINE('User inserted successfully: ' || p_username);
    END insert_into_users;

    -- CHANNELS
    PROCEDURE insert_into_channels(
        p_channel_id IN NUMBER,
        p_channel_name IN VARCHAR2,
        p_description IN CLOB
    ) IS
    BEGIN
```

```
    INSERT INTO C##PROJECT_USER.CHANNELS (
        CHANNEL_ID, CHANNEL_NAME, DESCRIPTION
    ) VALUES (
        p_channel_id,
        p_channel_name,
        p_description
    );
    DBMS_OUTPUT.PUT_LINE('Channel inserted successfully: ' || p_channel_name);
END insert_into_channels;

-- ENTRIES
PROCEDURE insert_into_entries(
    p_entry_id IN NUMBER,
    p_title IN VARCHAR2,
    p_content IN CLOB,
    p_user_id IN NUMBER,
    p_channel_id IN NUMBER
) IS
BEGIN
    INSERT INTO C##PROJECT_USER.ENTRIES (
        ENTRY_ID, TITLE, CONTENT, USER_ID, CHANNEL_ID, CREATED_AT, UPDATED_AT
    ) VALUES (
        p_entry_id,
        p_title,
        p_content,
        p_user_id,
        p_channel_id,
        SYSTIMESTAMP,
        SYSTIMESTAMP
    );
    DBMS_OUTPUT.PUT_LINE('Entry inserted successfully: ' || p_title);
END insert_into_entries;

-- MESSAGES
PROCEDURE insert_into_messages(
    p_message_id IN NUMBER,
    p_sender_id IN NUMBER,
    p_receiver_id IN NUMBER,
    p_content IN CLOB,
    p_is_read IN NUMBER
) IS
BEGIN
    INSERT INTO C##PROJECT_USER.MESSAGES (
        MESSAGE_ID, SENDER_ID, RECEIVER_ID, CONTENT, SENT_AT, IS_READ
    ) VALUES (
        p_message_id,
        p_sender_id,
        p_receiver_id,
        p_content,
        SYSTIMESTAMP,
        p_is_read
    );
```

```
    DBMS_OUTPUT.PUT_LINE('Message inserted successfully: ' || p_message_id);
END insert_into_messages;

-- COMMENTS
PROCEDURE insert_into_comments(
    p_comment_id IN NUMBER,
    p_entry_id IN NUMBER,
    p_user_id IN NUMBER,
    p_content IN CLOB
) IS
BEGIN
    INSERT INTO C##PROJECT_USER.COMMENTS (
        COMMENT_ID, ENTRY_ID, USER_ID, CONTENT, CREATED_AT
    ) VALUES (
        p_comment_id,
        p_entry_id,
        p_user_id,
        p_content,
        SYSTIMESTAMP
    );
    DBMS_OUTPUT.PUT_LINE('Comment inserted successfully: ' || p_comment_id);
END insert_into_comments;

-- LIKES
PROCEDURE insert_into_likes(
    p_like_id IN NUMBER,
    p_entry_id IN NUMBER,
    p_comment_id IN NUMBER,
    p_user_id IN NUMBER
) IS
BEGIN
    INSERT INTO C##PROJECT_USER.LIKES (
        LIKE_ID, ENTRY_ID, COMMENT_ID, USER_ID, CREATED_AT
    ) VALUES (
        p_like_id,
        p_entry_id,
        p_comment_id,
        p_user_id,
        SYSTIMESTAMP
    );
    DBMS_OUTPUT.PUT_LINE('Like inserted successfully: ' || p_like_id);
END insert_into_likes;

-- REPORTS
PROCEDURE insert_into_reports(
    p_report_id IN NUMBER,
    p_user_id IN NUMBER,
    p_entry_id IN NUMBER,
    p_comment_id IN NUMBER,
    p_reason IN CLOB,
    p_status IN VARCHAR2
) IS
```

```
  BEGIN
    INSERT INTO C##PROJECT_USER.REPORTS (
      REPORT_ID, USER_ID, ENTRY_ID, COMMENT_ID, REASON, CREATED_AT, STATUS
    ) VALUES (
      p_report_id,
      p_user_id,
      p_entry_id,
      p_comment_id,
      p_reason,
      SYSTIMESTAMP,
      p_status
    );
    DBMS_OUTPUT.PUT_LINE('Report inserted successfully: ' || p_report_id);
  END insert_into_reports;

  -- SAVED_ENTRIES
  PROCEDURE insert_into_saved_entries(
    p_saved_id IN NUMBER,
    p_user_id IN NUMBER,
    p_entry_id IN NUMBER
  ) IS
  BEGIN
    INSERT INTO C##PROJECT_USER.SAVED_ENTRIES (
      SAVED_ID, USER_ID, ENTRY_ID, SAVED_AT
    ) VALUES (
      p_saved_id,
      p_user_id,
      p_entry_id,
      SYSTIMESTAMP
    );
    DBMS_OUTPUT.PUT_LINE('Saved Entry inserted successfully: ' || p_saved_id);
  END insert_into_saved_entries;

END insert_pkg;
/
```

7.2.2.Update Package

```
CREATE OR REPLACE PACKAGE update_pkg AS
  PROCEDURE update_users(
    p_user_id IN NUMBER,
    p_username IN VARCHAR2,
    p_email IN VARCHAR2,
    p_password_hash IN VARCHAR2,
    p_profile_picture IN VARCHAR2,
    p_is_admin IN NUMBER
  );

  PROCEDURE update_channels(
    p_channel_id IN NUMBER,
    p_channel_name IN VARCHAR2,
    p_description IN CLOB
```

```
    );

    PROCEDURE update_entries(
        p_entry_id IN NUMBER,
        p_title IN VARCHAR2,
        p_content IN CLOB
    );

    PROCEDURE update_likes(
    p_like_id IN NUMBER,
    p_entry_id IN NUMBER,
    p_comment_id IN NUMBER,
    p_user_id IN NUMBER
    );


    PROCEDURE update_messages(
        p_message_id IN NUMBER,
        p_content IN CLOB,
        p_is_read IN NUMBER
    );

    PROCEDURE update_comments(
        p_comment_id IN NUMBER,
        p_content IN CLOB
    );

    PROCEDURE update_reports(
        p_report_id IN NUMBER,
        p_status IN VARCHAR2
    );

    PROCEDURE update_saved_entries(
        p_saved_id IN NUMBER,
        p_entry_id IN NUMBER
    );
END update_pkg;
/

CREATE OR REPLACE PACKAGE BODY update_pkg AS

    -- USERS Table
    PROCEDURE update_users(
        p_user_id IN NUMBER,
        p_username IN VARCHAR2,
        p_email IN VARCHAR2,
        p_password_hash IN VARCHAR2,
        p_profile_picture IN VARCHAR2,
        p_is_admin IN NUMBER
    ) IS
    BEGIN
        UPDATE C##PROJECT_USER.USERS
```

```
    SET USERNAME = p_username,
        EMAIL = p_email,
        PASSWORD_HASH = p_password_hash,
        PROFILE_PICTURE = p_profile_picture,
        IS_ADMIN = p_is_admin
    WHERE USER_ID = p_user_id;

    DBMS_OUTPUT.PUT_LINE('User updated successfully: ' || p_user_id);
END update_users;

-- CHANNELS Table
PROCEDURE update_channels(
    p_channel_id IN NUMBER,
    p_channel_name IN VARCHAR2,
    p_description IN CLOB
) IS
BEGIN
    UPDATE C##PROJECT_USER.CHANNELS
    SET CHANNEL_NAME = p_channel_name,
        DESCRIPTION = p_description
    WHERE CHANNEL_ID = p_channel_id;

    DBMS_OUTPUT.PUT_LINE('Channel updated successfully: ' || p_channel_id);
END update_channels;

-- ENTRIES Table
PROCEDURE update_entries(
    p_entry_id IN NUMBER,
    p_title IN VARCHAR2,
    p_content IN CLOB
) IS
BEGIN
    UPDATE C##PROJECT_USER.ENTRIES
    SET TITLE = p_title,
        CONTENT = p_content,
        UPDATED_AT = SYSTIMESTAMP
    WHERE ENTRY_ID = p_entry_id;

    DBMS_OUTPUT.PUT_LINE('Entry updated successfully: ' || p_entry_id);
END update_entries;

-- likes table
PROCEDURE update_likes(
    p_like_id IN NUMBER,
    p_entry_id IN NUMBER,
    p_comment_id IN NUMBER,
    p_user_id IN NUMBER
) IS
BEGIN
    UPDATE C##PROJECT_USER.LIKES
    SET ENTRY_ID = p_entry_id,
        COMMENT_ID = p_comment_id,
```

```
        USER_ID = p_user_id,
        CREATED_AT = SYSTIMESTAMP -- Tarihi güncelliyoruz
    WHERE LIKE_ID = p_like_id;

        IF SQL%ROWCOUNT > 0 THEN
    DBMS_OUTPUT.PUT_LINE('Like updated successfully: ' || p_like_id);
ELSE
    DBMS_OUTPUT.PUT_LINE('Like not found: ' || p_like_id);
END IF;
END update_likes;


-- MESSAGES Table
PROCEDURE update_messages(
    p_message_id IN NUMBER,
    p_content IN CLOB,
    p_is_read IN NUMBER
) IS
BEGIN
    UPDATE C##PROJECT_USER.MESSAGES
    SET CONTENT = p_content,
        IS_READ = p_is_read
    WHERE MESSAGE_ID = p_message_id;

    DBMS_OUTPUT.PUT_LINE('Message updated successfully: ' || p_message_id);
END update_messages;

-- COMMENTS Table
PROCEDURE update_comments(
    p_comment_id IN NUMBER,
    p_content IN CLOB
) IS
BEGIN
    UPDATE C##PROJECT_USER.COMMENTS
    SET CONTENT = p_content
    WHERE COMMENT_ID = p_comment_id;

    DBMS_OUTPUT.PUT_LINE('Comment updated successfully: ' || p_comment_id);
END update_comments;

-- REPORTS Table
PROCEDURE update_reports(
    p_report_id IN NUMBER,
    p_status IN VARCHAR2
) IS
BEGIN
    UPDATE C##PROJECT_USER.REPORTS
    SET STATUS = p_status
    WHERE REPORT_ID = p_report_id;

    DBMS_OUTPUT.PUT_LINE('Report updated successfully: ' || p_report_id);
END update_reports;
```

```
    -- SAVED_ENTRIES Table
    PROCEDURE update_saved_entries(
        p_saved_id IN NUMBER,
        p_entry_id IN NUMBER
    ) IS
    BEGIN
        UPDATE C##PROJECT_USER.SAVED_ENTRIES
        SET ENTRY_ID = p_entry_id
        WHERE SAVED_ID = p_saved_id;

        DBMS_OUTPUT.PUT_LINE('Saved Entry updated successfully: ' || p_saved_id);
    END update_saved_entries;

END update_pkg;
/
```

7.2.3.Delete Package

```
CREATE OR REPLACE PACKAGE delete_pkg AS
    PROCEDURE delete_from_users(p_user_id IN NUMBER);

    PROCEDURE delete_from_channels(p_channel_id IN NUMBER);

    PROCEDURE delete_from_entries(p_entry_id IN NUMBER);

    PROCEDURE delete_from_messages(p_message_id IN NUMBER);

    PROCEDURE delete_from_comments(p_comment_id IN NUMBER);

    PROCEDURE delete_from_reports(p_report_id IN NUMBER);

    PROCEDURE delete_from_saved_entries(p_saved_id IN NUMBER);

    PROCEDURE delete_from_likes(p_like_id IN NUMBER);

END delete_pkg;
/

CREATE OR REPLACE PACKAGE BODY delete_pkg AS

    -- USERS -
    PROCEDURE delete_from_users(p_user_id IN NUMBER) IS
    BEGIN
        DELETE FROM C##PROJECT_USER.USERS WHERE USER_ID = p_user_id;
        DBMS_OUTPUT.PUT_LINE('User deleted successfully: ' || p_user_id);
    END delete_from_users;

    -- CHANNELS -
    PROCEDURE delete_from_channels(p_channel_id IN NUMBER) IS
    BEGIN
        DELETE FROM C##PROJECT_USER.CHANNELS WHERE CHANNEL_ID = p_channel_id;
```

```
   DBMS_OUTPUT.PUT_LINE('Channel deleted successfully: ' || p_channel_id);
END delete_from_channels;

-- ENTRIES -
PROCEDURE delete_from_entries(p_entry_id IN NUMBER) IS
BEGIN
   DELETE FROM C##PROJECT_USER.ENTRIES WHERE ENTRY_ID = p_entry_id;
   DBMS_OUTPUT.PUT_LINE('Entry deleted successfully: ' || p_entry_id);
END delete_from_entries;


--LIKES TABLE
PROCEDURE delete_from_likes(
p_like_id IN NUMBER
) IS
BEGIN
DELETE FROM C##PROJECT_USER.LIKES
WHERE LIKE_ID = p_like_id;

IF SQL%ROWCOUNT > 0 THEN
   DBMS_OUTPUT.PUT_LINE('Like deleted successfully: ' || p_like_id);
ELSE
   DBMS_OUTPUT.PUT_LINE('Like not found: ' || p_like_id);
END IF;
END delete_from_likes;


-- MESSAGES -
PROCEDURE delete_from_messages(p_message_id IN NUMBER) IS
BEGIN
   DELETE FROM C##PROJECT_USER.MESSAGES WHERE MESSAGE_ID = p_message_id;
   DBMS_OUTPUT.PUT_LINE('Message deleted successfully: ' || p_message_id);
END delete_from_messages;

-- COMMENTS -
PROCEDURE delete_from_comments(p_comment_id IN NUMBER) IS
BEGIN
   DELETE FROM C##PROJECT_USER.COMMENTS WHERE COMMENT_ID = p_comment_id;
   DBMS_OUTPUT.PUT_LINE('Comment deleted successfully: ' || p_comment_id);
END delete_from_comments;

-- REPORTS -
PROCEDURE delete_from_reports(p_report_id IN NUMBER) IS
BEGIN
   DELETE FROM C##PROJECT_USER.REPORTS WHERE REPORT_ID = p_report_id;
   DBMS_OUTPUT.PUT_LINE('Report deleted successfully: ' || p_report_id);
END delete_from_reports;

-- SAVED_ENTRIES -
PROCEDURE delete_from_saved_entries(p_saved_id IN NUMBER) IS
BEGIN
   DELETE FROM C##PROJECT_USER.SAVED_ENTRIES WHERE SAVED_ID = p_saved_id;
```

```
        DBMS_OUTPUT.PUT_LINE('Saved Entry deleted successfully: ' || p_saved_id);
    END delete_from_saved_entries;

END delete_pkg;

/
```

7.2.4.Delete Duplicate Package

```
CREATE OR REPLACE PACKAGE delete_duplicates_pkg AS
    -- Procedure Definitions
    PROCEDURE delete_duplicates_from_users;
    PROCEDURE delete_duplicates_from_entries;
    PROCEDURE delete_duplicates_from_channels;
    PROCEDURE delete_duplicates_from_messages;
    PROCEDURE delete_duplicates_from_comments;
    PROCEDURE delete_duplicates_from_reports;
    PROCEDURE delete_duplicates_from_saved_entries;
    PROCEDURE delete_duplicates_from_likes;
END delete_duplicates_pkg;
/

CREATE OR REPLACE PACKAGE BODY delete_duplicates_pkg AS

    -- Delete USERS Duplicates
    PROCEDURE delete_duplicates_from_users IS
    BEGIN
        DELETE FROM C##PROJECT_USER.USERS
        WHERE user_id IN (
            SELECT user_id
            FROM (
                SELECT user_id, ROW_NUMBER() OVER (PARTITION BY username, email ORDER BY user_id)
AS row_num
                FROM C##PROJECT_USER.USERS
            )
            WHERE row_num > 1
        );
        DBMS_OUTPUT.PUT_LINE('Duplicate users deleted successfully.');
    END delete_duplicates_from_users;

    PROCEDURE delete_duplicates_from_entries IS
    BEGIN
        DELETE FROM C##PROJECT_USER.ENTRIES
        WHERE entry_id IN (
            SELECT entry_id
            FROM (
                SELECT entry_id, ROW_NUMBER() OVER (PARTITION BY title, content ORDER BY entry_id)
AS row_num
                FROM C##PROJECT_USER.ENTRIES
            )
            WHERE row_num > 1
        );
        DBMS_OUTPUT.PUT_LINE('Duplicate entries deleted successfully.');
```

```
    END delete_duplicates_from_entries;

    PROCEDURE delete_duplicates_from_channels IS
    BEGIN
        DELETE FROM C##PROJECT_USER.CHANNELS
        WHERE channel_id IN (
            SELECT channel_id
            FROM (
                SELECT channel_id, ROW_NUMBER() OVER (PARTITION BY channel_name ORDER BY
channel_id) AS row_num
                FROM C##PROJECT_USER.CHANNELS
            )
            WHERE row_num > 1
        );
        DBMS_OUTPUT.PUT_LINE('Duplicate channels deleted successfully.');
    END delete_duplicates_from_channels;

    PROCEDURE delete_duplicates_from_messages IS
    BEGIN
        DELETE FROM C##PROJECT_USER.MESSAGES
        WHERE message_id IN (
            SELECT message_id
            FROM (
                SELECT message_id, ROW_NUMBER() OVER (PARTITION BY sender_id, receiver_id, content
ORDER BY message_id) AS row_num
                FROM C##PROJECT_USER.MESSAGES
            )
            WHERE row_num > 1
        );
        DBMS_OUTPUT.PUT_LINE('Duplicate messages deleted successfully.');
    END delete_duplicates_from_messages;

    PROCEDURE delete_duplicates_from_comments IS
    BEGIN
        DELETE FROM C##PROJECT_USER.COMMENTS
        WHERE comment_id IN (
            SELECT comment_id
            FROM (
                SELECT comment_id, ROW_NUMBER() OVER (PARTITION BY entry_id, user_id, content
ORDER BY comment_id) AS row_num
                FROM C##PROJECT_USER.COMMENTS
            )
            WHERE row_num > 1
        );
        DBMS_OUTPUT.PUT_LINE('Duplicate comments deleted successfully.');
    END delete_duplicates_from_comments;

    PROCEDURE delete_duplicates_from_reports IS
    BEGIN
        DELETE FROM C##PROJECT_USER.REPORTS
        WHERE report_id IN (
            SELECT report_id
```

```
        FROM (
            SELECT report_id, ROW_NUMBER() OVER (PARTITION BY user_id, entry_id, comment_id
ORDER BY report_id) AS row_num
            FROM C##PROJECT_USER.REPORTS
        )
        WHERE row_num > 1
    );
    DBMS_OUTPUT.PUT_LINE('Duplicate reports deleted successfully.');
  END delete_duplicates_from_reports;

  PROCEDURE delete_duplicates_from_saved_entries IS
  BEGIN
    DELETE FROM C##PROJECT_USER.SAVED_ENTRIES
    WHERE saved_id IN (
      SELECT saved_id
      FROM (
        SELECT saved_id, ROW_NUMBER() OVER (PARTITION BY user_id, entry_id ORDER BY
saved_id) AS row_num
        FROM C##PROJECT_USER.SAVED_ENTRIES
      )
      WHERE row_num > 1
    );
    DBMS_OUTPUT.PUT_LINE('Duplicate saved entries deleted successfully.');
  END delete_duplicates_from_saved_entries;

  PROCEDURE delete_duplicates_from_likes IS
  BEGIN
    DELETE FROM C##PROJECT_USER.LIKES
    WHERE like_id IN (
      SELECT like_id
      FROM (
        SELECT like_id, ROW_NUMBER() OVER (PARTITION BY entry_id, comment_id, user_id
ORDER BY like_id) AS row_num
        FROM C##PROJECT_USER.LIKES
      )
      WHERE row_num > 1
    );
    DBMS_OUTPUT.PUT_LINE('Duplicate likes deleted successfully.');
  END delete_duplicates_from_likes;

END delete_duplicates_pkg;
/
```

7.3.Transaction Table

```
CREATE TABLE C##PROJECT_USER.TRANSACTIONS (
  TRANSACTION_ID   NUMBER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
  ENTITY_NAME      VARCHAR2(50), -- Name of the operated table
  OPERATION_TYPE   VARCHAR2(10), -- operation type (INSERT, UPDATE, DELETE)
  REFERENCE_ID     NUMBER,       -- ID in the referenced table
  OPERATION_DATE   TIMESTAMP DEFAULT SYSTIMESTAMP -- operation date
);
```

7.4.Trigger Codes

```
CREATE OR REPLACE TRIGGER users_after_insert_trigger
AFTER INSERT ON C##PROJECT_USER.USERS
FOR EACH ROW
BEGIN
   INSERT INTO C##PROJECT_USER.TRANSACTIONS (
      ENTITY_NAME,
      OPERATION_TYPE,
      REFERENCE_ID,
      OPERATION_DATE
   ) VALUES (
      'USERS', -- Operated table
      'INSERT', -- operation type
      :NEW.USER_ID,
      SYSTIMESTAMP -- operation date
   );
END;
/

CREATE OR REPLACE TRIGGER entries_after_insert_trigger
AFTER INSERT ON C##PROJECT_USER.ENTRIES
FOR EACH ROW
BEGIN
   INSERT INTO C##PROJECT_USER.TRANSACTIONS (
      ENTITY_NAME,
      OPERATION_TYPE,
      REFERENCE_ID,
      OPERATION_DATE
   ) VALUES (
      'ENTRIES', -- Operated table
      'INSERT', -- operation type
      :NEW.ENTRY_ID,
      SYSTIMESTAMP -- operation date
   );
END;
/

CREATE OR REPLACE TRIGGER messages_after_insert_trigger
AFTER INSERT ON C##PROJECT_USER.MESSAGES
FOR EACH ROW
BEGIN
   INSERT INTO C##PROJECT_USER.TRANSACTIONS (
      ENTITY_NAME,
      OPERATION_TYPE,
      REFERENCE_ID,
      OPERATION_DATE
   ) VALUES (
      'MESSAGES', -- Operated table
      'INSERT', -- operation type
      :NEW.MESSAGE_ID,
      SYSTIMESTAMP -- operation date
```

```
    );
END;
/

CREATE OR REPLACE TRIGGER comments_after_insert_trigger
AFTER INSERT ON C##PROJECT_USER.COMMENTS
FOR EACH ROW
BEGIN
    INSERT INTO C##PROJECT_USER.TRANSACTIONS (
        ENTITY_NAME,
        OPERATION_TYPE,
        REFERENCE_ID,
        OPERATION_DATE
    ) VALUES (
        'COMMENTS', -- Operated table
        'INSERT', -- operation type
        :NEW.COMMENT_ID, -- COMMENT_ID of inserted comment
        SYSTIMESTAMP -- Operation time
    );
END;
/

CREATE OR REPLACE TRIGGER likes_after_insert_trigger
AFTER INSERT ON C##PROJECT_USER.LIKES
FOR EACH ROW
BEGIN
    INSERT INTO C##PROJECT_USER.TRANSACTIONS (
        ENTITY_NAME,
        OPERATION_TYPE,
        REFERENCE_ID,
        OPERATION_DATE
    ) VALUES (
        'LIKES', -- Operated table
        'INSERT', -- operation type
        :NEW.LIKE_ID,
        SYSTIMESTAMP -- operation date
    );
END;
/

CREATE OR REPLACE TRIGGER reports_after_insert_trigger
AFTER INSERT ON C##PROJECT_USER.REPORTS
FOR EACH ROW
BEGIN
    INSERT INTO C##PROJECT_USER.TRANSACTIONS (
        ENTITY_NAME,
        OPERATION_TYPE,
        REFERENCE_ID,
        OPERATION_DATE
    ) VALUES (
        'REPORTS', -- Operated table
        'INSERT', -- operation type
```

```
        :NEW.REPORT_ID,
        SYSTIMESTAMP -- operation date
    );
END;
/

CREATE OR REPLACE TRIGGER saved_entries_after_insert_trigger
AFTER INSERT ON C##PROJECT_USER.SAVED_ENTRIES
FOR EACH ROW
BEGIN
    INSERT INTO C##PROJECT_USER.TRANSACTIONS (
        ENTITY_NAME,
        OPERATION_TYPE,
        REFERENCE_ID,
        OPERATION_DATE
    ) VALUES (
        'SAVED_ENTRIES', -- Operated table
        'INSERT', -- operation type
        :NEW.SAVED_ID,
        SYSTIMESTAMP -- operation date
    );
END;
/
CREATE OR REPLACE TRIGGER channels_after_insert_trigger
AFTER INSERT ON C##PROJECT_USER.CHANNELS
FOR EACH ROW
BEGIN
    INSERT INTO C##PROJECT_USER.TRANSACTIONS (
        ENTITY_NAME,
        OPERATION_TYPE,
        REFERENCE_ID,
        OPERATION_DATE
    ) VALUES (
        'CHANNELS', -- Operated table
        'INSERT', -- operation type
        :NEW.CHANNEL_ID,
        SYSTIMESTAMP -- operation date
    );
END;
/
```