PROJECT REPORT

DATABASE MANAGEMENT SYSTEM(UCS310)

Submitted By:

NAME OF THE STUDENT:	ROLL NO.
PARUSH	102003404
ISHAN MAZUMDAR	102003410
RISHABH HANDOO	102003393
AAKARSH MIGLANI	102003415

Batch: COE 16

Submitted To: DR. GEETA KASANA

DEPARTMENT OF COMPUTER SCIENCE AND

ENGINEERING THAPAR INSTITUTE OF ENGINEERING

AND TECHNOLOGY, PATIALA-147001, PUNJAB

JAN-JUNE 2022

ACKNOWLEDGEMENT

We would like to express our special thanks of gratitude to our teacher Dr .Geeta Kasana for her able guidance and support in completing the Database Project. She has guided us throughout the Project Development. We are thankful for the patience with which she stood by us till the end of our Project.

INDEX

PROBLEM STATEMENT

Instagram is a social networking service that enables its users to upload and share their photos and videos with other users. Instagram users can choose to share information either publicly or privately. Anything shared publicly can be seen by any other user, whereas privately shared content can only be accessed by the specified set of people. Instagram also enables its users to share through many other social networking platforms, such as Facebook, Twitter, Flickr, and Tumblr.

We need to store data about users, their uploaded photos, posts, their friends, etc .

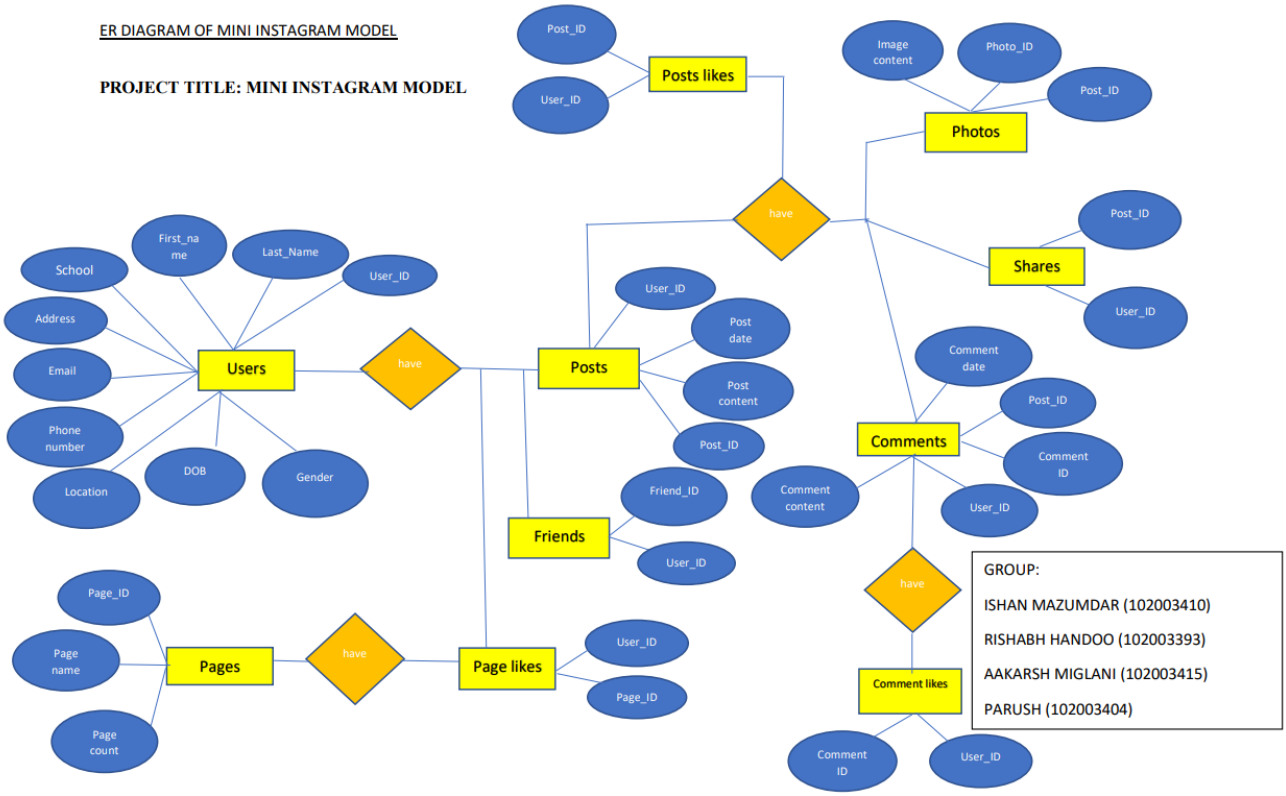
Proposed Model:

The Mini Instagram Model will handle all the necessary and minute details easily and proper database security accordingly to the user. This requires software, which will store data about users, posts, friends, post likes etc. & all activities that take place in instagram.

The objectives are-

- To increase efficiency with reduced cost.
- To reduce the burden of paper work.
- To save time management for recording details of each and every user and their posts, friends, etc.
- To generate required ananlysis easily.

ENTITY RELATIONSHIP DIAGRAM



ER-DIAGRAM TO TABLES

`Users	
PK	User ID
	Lastname
	First name
	Address
	Email
	Phone no
	DOB
	Gender

Posts	
PK	Post ID
	Post content
	Post date
FK	User ID

Friends	
PK	Friend ID
FK	User ID

Page likes	
PK	Page ID
FK	User ID

Pages	
PK	Page ID
	Page name
	Page content

Post likes	
PK	Post ID
FK	User ID

Photos	
PK	Photo ID
FK	Post ID
	Image content

Shares	
FK	Post ID
FK	User ID

Comment likes	
PK	Comment ID
FK	User ID

Comments	
PK	Comment ID
FK	Post ID
FK	User ID
	Comment date
	Comment content

NORMALISATION

Normalization is the process of minimizing redundancy from a relation or set of relations. Redundancy in relation may cause insertion, deletion, and update anomalies. So, it helps to minimize the redundancy in relations. Normal forms are used to eliminate or reduce redundancy in database tables.

1. First Normal Form –

If a relation contains composite or multi-valued attribute, it violates first normal form or a relation is in first normal form if it does not contain any composite or multi-valued attribute. A relation is in first normal form if every attribute in that relation is singled valued attribute.

2. Second Normal Form –

To be in second normal form, a relation must be in first normal form and relation must not contain any partial dependency. A relation is in 2NF if it has No Partial Dependency, i.e., no non-prime attribute (attributes which are not part of any candidate key) is dependent on any proper subset of any candidate key of the table.
Partial Dependency – If the proper subset of candidate key determines non-prime attribute, it is called partial dependency.

3. Third Normal Form –

A relation is in third normal form, if there is no transitive dependency for non-prime attributes as well as it is in second normal form.
A relation is in 3NF if at least one of the following condition holds in every non-trivial function dependency X –> Y

1. X is a super key.
2. Y is a prime attribute (each element of Y is part of some candidate key).

Transitive dependency – If A->B and B->C are two FDs then A->C is called transitive dependency.

FUNCTIONS

1. USERS COUNT:

```
DECLARE
   ans int;
   total int ;
   Function Total_user return int  AS
   BEGIN
      SELECT count(*) INTO total FROM users ;
      return total;
   END ;
BEGIN
   ans:=Total_user();
   DBMS_OUTPUT.PUT_LINE(ans);
  END;
```

```
299  -- --function
300  -- -- total no of  users
301  DECLARE
302      ans int;
303      total int ;
304      Function Total_user return int  AS
305      BEGIN
306          SELECT count(*) INTO total FROM users ;
307          return total;
308      END ;
309  BEGIN
310      ans:=Total_user();
311      DBMS_OUTPUT.PUT_LINE(ans);
312  END;
313
```

```
Statement processed.
10
```

2.TOTAL LIKES OF A POST:

```
declare
   total int;
   ans int;
   function likecount(pst_id posts_likes.post_id%type) return int as
   begin
      select count(*) into total from posts_likes  where  post_id=pst_id;
      return total;
   end;
begin
   ans:=likecount(1);
   DBMS_OUTPUT.PUT_LINE(ans);
end;
```

```
315
316  -- --total likes on a give post
317  declare
318      total int;
319      ans int;
320      function likecount(pst_id posts_likes.post_id%type) return int as
321      begin
322          select count(*) into total from posts_likes  where  post_id=pst_id;
323          return total;
324      end;
325  begin
326      ans:=likecount(1);
327      DBMS_OUTPUT.PUT_LINE(ans);
328  end;
329
```

```
Statement processed.
6
```

3. NUMBER OF FRIENDS OF ANY USER:

```
declare
   total int ;
   ans int;
   us_name users.firstname%type;
   function friends_count(us_id users.user_id%type) return int as
   begin
      select firstname into us_name from users where users.user_id=us_id;
      DBMS_OUTPUT.PUT_LINE(us_name);
      select count(*) into total from friends where friends.user_id=us_id;
      return total;
   end;
begin
   ans:=friends_count(1);
   DBMS_OUTPUT.PUT_LINE(ans);
end;
```

```
331  -- -- no of friends of a given user
332  declare
333      total int ;
334      ans int;
335      us_name users.firstname%type;
336      function friends_count(us_id users.user_id%type) return int as
337      begin
338          select firstname into us_name from users where users.user_id=us_id;
339          DBMS_OUTPUT.PUT_LINE(us_name);
340          select count(*) into total from friends where friends.user_id=us_id;
341          return total;
342      end;
343  begin
344      ans:=friends_count(1);
345      DBMS_OUTPUT.PUT_LINE(ans);
346  end;
347
```

```
Statement processed.
akshit
2
```

CURSORS

1.PAGE NAMES AND THEIR LIKES:

```
declare
   pag_name pages.page_name%type;
   total int ;
   cursor c1 is select page_id from page_likes group by page_id;
   rec c1%rowtype;
begin
   open c1;
   loop
      fetch c1 into rec;
      select count(*) into total from page_likes where
page_likes.page_id=rec.page_id;
      select page_name into pag_name from pages where
pages.page_id=rec.page_id;
      exit when c1%notfound;
      dbms_output.put_line(pag_name || '   ' || total);
   end loop;
   close c1;
end;
```

```
348 -- -- ********************************************************************************************************
349
350 -- --cursor
351 -- -- name of all pages with their like count
352 declare
353     pag_name pages.page_name%type;
354     total int ;
355     cursor c1 is select page_id from page_likes group by page_id;
356     rec c1%rowtype;
357 begin
358     open c1;
359     loop
360         fetch c1 into rec;
361         select count(*) into total from page_likes where page_likes.page_id=rec.page_id;
362         select page_name into pag_name from pages where pages.page_id=rec.page_id;
363         exit when c1%notfound;
364         dbms_output.put_line(pag_name || '    ' || total);
365     end loop;
366     close c1;
367 end;
368
```

```
Statement processed.
andy_the_aircon   3
bill_the_king   3
surely_an_athlete   3
i_killed_cupid   1
jimmy_not_choo   3
```

2.NUMBER OF POSTS OF EACH USER:

```
declare
   cursor c5 is select user_id from posts group by user_id;
   rec c5%rowtype;
   total int;
   us_name users.firstname%type;
begin
   open c5;
   loop
      fetch c5 into rec;
      select firstname into us_name from users where
users.user_id=rec.user_id;
      select count(*) into total from posts where posts.user_id=rec.user_id;
      exit when c5%notfound;
      dbms_output.put_line(us_name || '   ' || total);
   end loop;
   close c5;
end;
```

```
-- -- no of post of each user

declare
    cursor c5 is select user_id from posts group by user_id;
    rec c5%rowtype;
    total int;
    us_name users.firstname%type;
begin
    open c5;
    loop
        fetch c5 into rec;
        select firstname into us_name from users where users.user_id=rec.user_id;
        select count(*) into total from posts where posts.user_id=rec.user_id;
        exit when c5%notfound;
        dbms_output.put_line(us_name || '    ' || total);
    end loop;
    close c5;
end;
```

```
Statement processed.
Alec        2
akshit      2
Drew        2
Rishi       2
Emelia      1
Manue       3
Isiah       2
Genesis     1
```

3.NAME OF PEOPLE WHO HAVE LIKED THE POST:

```
declare
   cursor c1(n number) is select posts_likes.user_id, users.firstname from posts_likes,users where posts_likes.user_id=users.user_id and post_id=n;
begin
   for rec in c1(1) loop
      dbms_output.put_line(rec.firstname);
   end loop;
end;
```

```
-- -- return all the names of people who have liked their post
declare
    cursor c1(n number) is select posts_likes.user_id, users.firstname from posts_likes,users where posts_likes.user_id=users.user_id and post_id=n;
begin
    for rec in c1(1) loop
        dbms_output.put_line(rec.firstname);
    end loop;
end;
```

```
User created.
akshit
Rishi
Isiah
Emelia
Manue
Alec
```

4.PRINTING ALL THE COMMENTS OF THE POST:

```
declare
  cursor c2(n2 number) is select post_id, comment_content from
comments where post_id=n2;
begin
 for rec in c2(2) loop
   dbms_output.put_line('comment: '||rec.comment_content);
 end loop;
end;
```

5.USERS WHO LIKED A COMMENT:

```
declare
   my_com comments.comment_content%type;
   com number:=1;
   cursor c3( p number) is select comments.post_id,
comments.comment_id, users.firstname from comments,users where
(comments.comment_id=com) and (comments.post_id=p) and
(users.user_id=comments.user_id);
begin
      select comments.comment_content into my_com from comments
where comments.comment_id=com;
      dbms_output.put_line('comment: '||my_com);
```

```
    for rec in c3(1) loop
        dbms_output.put_line('who liked: '||rec.firstname);
    end loop;
end;
```

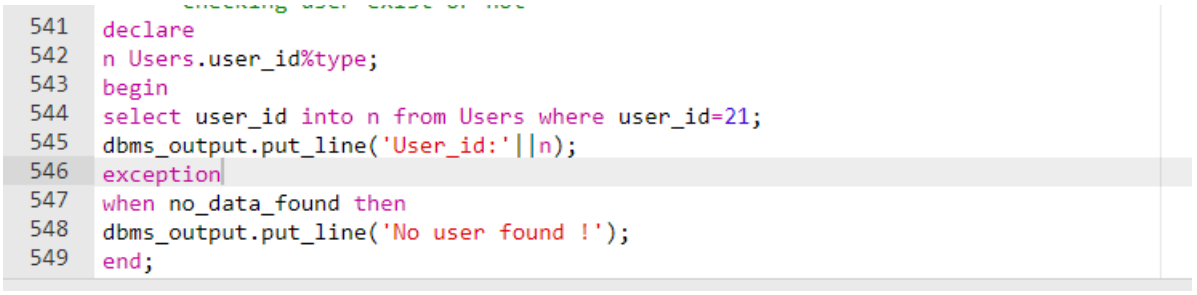
```
-- -- ***************************************************************************************************************************************
-- --  who liked the comment(comment detail)
declare
    my_com comments.comment_content%type;
    com number:=1;
    cursor c3( p number) is select comments.post_id, comments.comment_id, users.firstname from comments,users where (comments.comment_id=com) and (comments.post_id=p) and (users.user_id=comments.user_id);
begin
        select comments.comment_content into my_com from comments where comments.comment_id=com;
        dbms_output.put_line('comment: '||my_com);
    for rec in c3(1) loop
        dbms_output.put_line('who liked: '||rec.firstname);
    end loop;
end;

```

```
Statement processed.
comment: Never seen a selfie of yours that I don't like
who liked: akshit
```

EXCEPTION HANDLING

1.CHECK IF USER EXISTS:

```
declare
n Users.user_id%type;
begin
select user_id into n from Users where user_id=21;
dbms_output.put_line('User_id:'||n);
exception
when no_data_found then
dbms_output.put_line('No user found !');
end;
```



TRIGGERS

1.CAPITALISE FIRST CHARACTER OF FIRST AND LAST NAME:

```
create or replace trigger fna_upper
before insert or update
on Users
for each row
begin
  :new.firstname:=initcap(:new.firstname);
  :new.lastname:=initcap(:new.lastname);
end;
```

```
515 --trigger
516 create or replace trigger fna_upper
517 before insert or update
518 on Users
519 for each row
520 begin
521   :new.firstname:=initcap(:new.firstname);
522   :new.lastname:=initcap(:new.lastname);
523 end;
524 insert into users values (16,'ishan','mazi','chandi','ishanmazi@gamil.com','9852485823',to_date('12-05-2002','dd-mm-yyyy'),'male');
525 select * from users;
```

2.DATAS RELATED TO A POST ARE DELETED:

```
create or replace trigger delete_this after delete of post_id on posts for
each row

begin
   delete from posts_likes where post_id=:old.post_id;
   delete from shares where post_id=:old.post_id;
   delete from comments where post_id=:old.post_id;
end;
delete from posts where post_id=30;
```

```
528  -- -- ************************************************************************************************
529  -- -- if post deleted then data releated to that post will be deleted
530  -- -- not working
531  create or replace trigger delete_this after delete of post_id on posts for each row
532
533  begin
534      delete from posts_likes where post_id=:old.post_id;
535      delete from shares where post_id=:old.post_id;
536      delete from comments where post_id=:old.post_id;
537  end;
538  delete from posts where post_id=30;
```

3. IF ANY USER DELETED THEN ITS USER ID WILL BE STORED IN DELETED USER TABLE WORKING IN A WAY:

create table deleted_users(n number);

create or replace trigger delete_this after delete of user_id on users for each row
declare
  n number:=1
begin
  insert into deleted_users values(n);
end;
delete from users where user_id=n;

```
527  -- -- ************************************************************************************************
528  -- -- is any user deleted then its user id will store in deleted user table
529  -- -- working in a way
530
531  create table deleted_users(n number);
532
533  create or replace trigger delete_this after delete of user_id on users for each row
534  declare
535      n number:=1
536  begin
537      insert into deleted_users values(n);
538  end;
539  delete from users where user_id=n;
540
```

PROCEDURES

1.TO INSERT NEW USER INTO TABLE:

```
CREATE OR REPLACE PROCEDURE new_user(
        new_user_id IN users.user_id%TYPE,
        new_first_name IN users.firstname%TYPE,
        new_last_name IN users.lastname%TYPE,
        new_address IN users.address%TYPE,
        new_email IN users.email%TYPE,
        new_phone_number IN users.phone_no%TYPE,
        new_dob IN users.dob%TYPE,
        new_gender IN users.gender%TYPE )
IS
BEGIN

 INSERT INTO users
 VALUES (new_user_id, new_first_name,new_last_name,
new_address,new_email, new_phone_number,new_dob,new_gender);

 COMMIT;

END;
begin
new_user(15,'rishabh','handoo','chandi','rishabhhandoo@gamil.com','985
2456823',to_date('12-04-2002','dd-mm-yyyy'),'male');
end;
select * from users;
```

```
-- -- procedures
-- -- to insert new user into table
-- store procedure
CREATE OR REPLACE PROCEDURE new_user(
       new_user_id IN users.user_id%TYPE,
       new_first_name IN users.firstname%TYPE,
       new_last_name IN users.lastname%TYPE,
       new_address IN users.address%TYPE,
       new_email IN users.email%TYPE,
       new_phone_number IN users.phone_no%TYPE,
       new_dob IN users.dob%TYPE,
       new_gender IN users.gender%TYPE )
IS
BEGIN

  INSERT INTO users
  VALUES (new_user_id, new_first_name,new_last_name, new_address,new_email, new_phone_number,new_dob,new_gender);

  COMMIT;

END;
begin
new_user(15,'rishabh','handoo','chandi','rishabhhandoo@gamil.com','9852456823',to_date('12-04-2002','dd-mm-yyyy'),'male');
end;
select * from users;
```

2.TO INSERT NEW POSTS:

```
CREATE OR REPLACE PROCEDURE new_post(
       new_post_id IN posts.post_id%TYPE,
    new_post_content IN posts.post_content%TYPE,
       new_user_id IN posts.user_id%TYPE
       )
IS
BEGIN

 INSERT INTO posts
 VALUES (new_post_id,new_post_content,sysdate,new_user_id);

 COMMIT;
END;
begin
  new_post(18,'bts',15);
end;
```

```
-- -- procedure to insert new posts
-- -- no need to add date of new post
CREATE OR REPLACE PROCEDURE new_post(
       new_post_id IN posts.post_id%TYPE,
     new_post_content IN posts.post_content%TYPE,
       new_user_id IN posts.user_id%TYPE
       )
IS
BEGIN

  INSERT INTO posts
  VALUES (new_post_id,new_post_content,sysdate,new_user_id);

  COMMIT;
END;
begin
    new_post(18,'bts',15);
end;
```

3.NUMBER OF SHARES OF A POST:

```
CREATE or REPLACE PROCEDURE no_of_share(p_id in int)
AS
no_shares int;
BEGIN
select count(*) into no_shares from shares where p_id = post_id;

dbms_output.put_line(no_shares);
End;

begin
no_of_share(1);
end;
```

```
479  -- -- ***********************************************************************************************************
480  -- -- no of shares of a specific post
481  CREATE or REPLACE PROCEDURE no_of_share(p_id in int)
482  AS
483  no_shares int;
484  BEGIN
485  select count(*) into no_shares from shares where p_id = post_id;
486
487  dbms_output.put_line(no_shares);
488  End;
489
490  begin
491  no_of_share(1);
492  end;
493
```

```
Statement processed.
3
```

4.MOST LIKED POST:

```
create or replace procedure most_liked
as
likes int;

begin

select max(count()) into likes from posts_likes group by
posts_likes.post_id order by count() fetch first 1 rows only;
dbms_output.put_line(likes);
```

end;

begin
most_liked;
end;

```
496  -- -- procedure for most liked post
497  create or replace procedure most_liked
498  as
499  likes int;
500
501  begin
502
503  select max(count(*)) into likes from posts_likes group by posts_likes.post_id order by count(*) fetch first 1 rows only;
504  dbms_output.put_line(likes);
505
506  end;
507
508
509  begin
510  most_liked;
511  end;
512
```

```
Statement processed.
6
```