FINAL REPORT

Photo Collage Management System

BY
ANUSHKA PODDATURI

Introduction

Photo collages have become a popular way to display multiple pictures in a single frame. They are utilized in a variety of platforms, such as social networking, digital photo albums, and customized gifts. With the increase in the number of pictures people take with their smartphones and cameras, managing photo collages has become a difficult task. To address this problem, we propose a photo collage management system that can help users organize, search, retrieve, and create photo collages.

Without a collage management system, manually collecting and sorting a colossal number of photos has become a tedious and time-consuming process. People may have to spend hours labeling each photo and identifying duplicate copies. Also, it is a difficult task to identify a specific photo from a massive pile of pictures. Handling a considerably large amount of photos, sorting them, managing them based on location or per se photographer, and creating a backup of each photo are some tasks that may not be possible without a proper management system.

With the help of our photo collage management system, Users can organize each individual photo with a unique Photo Id which eventually helps in reducing redundancy. A database can efficiently store and retrieve photos, making it easy to organize. Users can quickly search for specific photos based on various criteria, such as date, location, or keywords. Also, a database can handle large amounts of data making it a scalable solution. As users add more photos a database can easily accommodate them without affecting performance.

Integrating, with other systems such as social media websites or photo editing software is easily possible with a Photo Collage Management System. Also, customization can be done to meet the specific needs of the user for example we can create custom tags, add comments, add locations, or create albums. Moreover, we can get various analytics on photo usage.

The following are some of the key features databases can perform:

- Retrieve all photos by location: This search will produce all the images related to the specified place.
- Retrieve all comments for a specific photo: This search will return all of the comments from all of the images as well as photos based on a particular comment.

- Retrieve all tags for a specific photo: According to the tag the user chooses, as well as the tag id, photographs will be returned by this query.
- Retrieve all photos in a specific album: Photos in album format will be returned in response to this query.
- Retrieve all photos based on the title: This search will produce all images linked to a specific title or set of titles.
- Retrieve all photos based on members: This search will return all images depending on the members included in the images, and vice versa, if desired.

If a UI is built on top of the photo collage management system, the following features could be added:

- Ability to upload photos and create new collages through a web interface
- Ability to search for collages by title or description
- Ability to view a gallery of all photos in a specific collage
- Ability to view the number of times a photo has been viewed
- Ability to edit and delete collages and photos
- Integration with social media platforms to share collages and photos with friends and family
- Personalized recommendations for new collages based on user preferences and browsing history

Professional photographers that need to manage huge photo collections and make collages for their clients might use the proposed photo collage management system. Moreover, designers and painters can use our system to keep track of their portfolios and organize their artwork. A sizable portion of the picture collage management system's clients is social media influencers, marketers, event planners, and students. There are various products available in the market like Canva, Adobe Spark, etc. these products take a few pictures as input from users and enable users to edit the pictures and make a photo collage of their liking, whereas there are few products like google photos or apple photos which help users in storing their data in cloud storage as a backup. But none of the available products help in maintaining the data logically and no product can help in retrieving data based on the photo Id or type of photo. Hence the proposed system can fulfill the market gap in organizing, searching, and retrieving the data logically.

Assumptions/Notes About Data Entities and Relationships

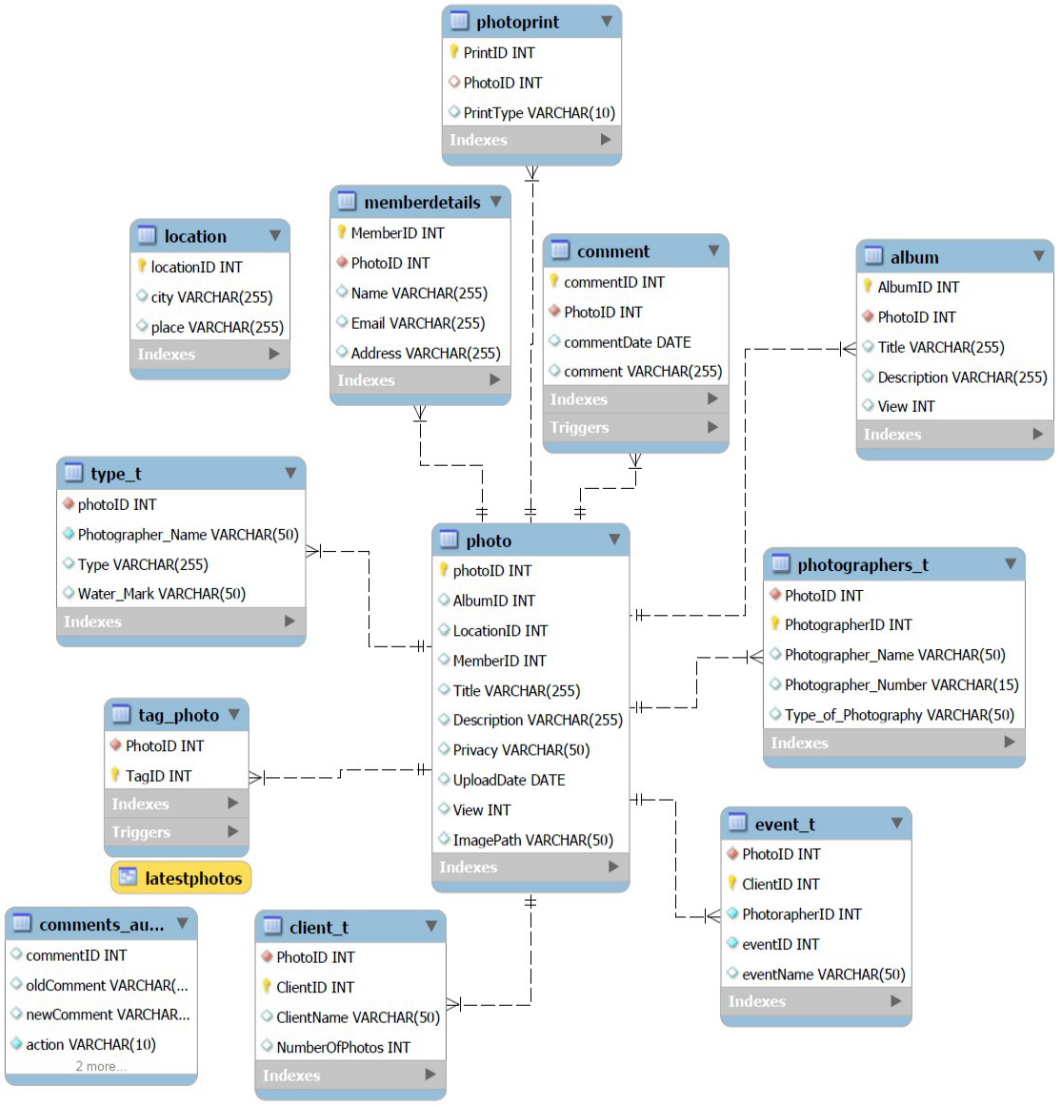
Photo Collage Management System is used to manage all the photos in a System, Where all the data is Related to each other Logically. In this DataBase Various aspects of a photo are being captured, accordingly some assumptions are made regarding the data. Some of the Key Assumptions are listed below.

Each Photo being added to the document is unique so based on this assumption a unique ID is created for each Photo. Similarly Each event is unique so no two events will have same unique ID. The same applies to various other categories like Album, Comments, Tags, Members etc.

Since, photo is the primary source of information to the photo collage management system an assumption is album, comment, location, tag, member details and some other details are in a one to one relationship with the unique photo.

And Finally, Since the cost of photos can change from time to time an assumption is made while creating the cost function and random costs have been included to help in the calculations.

ERR Diagram


photoprint
PrintID INT
PhotoID INT
PrintType VARCHAR(10)
Indexes
memberdetails
MemberID INT
PhotoID INT
Name VARCHAR(255)
Email VARCHAR(255)
Address VARCHAR(255)
Indexes
location
locationID INT
city VARCHAR(255)
place VARCHAR(255)
Indexes
comment
commentID INT
PhotoID INT
commentDate DATE
comment VARCHAR(255)
Indexes
Triggers
album
AlbumID INT
PhotoID INT
Title VARCHAR(255)
Description VARCHAR(255)
View INT
Indexes
type_t
photoID INT
Photographer_Name VARCHAR(50)
Type VARCHAR(255)
Water_Mark VARCHAR(50)
Indexes
photo
photoID INT
AlbumID INT
LocationID INT
MemberID INT
Title VARCHAR(255)
Description VARCHAR(255)
Privacy VARCHAR(50)
UploadDate DATE
View INT
ImagePath VARCHAR(50)
Indexes
photographers_t
PhotoID INT
PhotographerID INT
Photographer_Name VARCHAR(50)
Photographer_Number VARCHAR(15)
Type_of_Photography VARCHAR(50)
Indexes
tag_photo
PhotoID INT
TagID INT
Indexes
Triggers
latestphotos
event_t
PhotoID INT
ClientID INT
PhotorapherID INT
eventID INT
eventName VARCHAR(50)
Indexes
comments_au...
commentID INT
oldComment VARCHAR(...
newComment VARCHAR...
action VARCHAR(10)
2 more...
client_t
PhotoID INT
ClientID INT
ClientName VARCHAR(50)
NumberOfPhotos INT
Indexes

Design of the Database

PHOTO TABLE	PHOTO ID	Foreign Key	ALBUM ID, LOCATION ID, MEMBER ID, TITLE, DESCRIPTION,PRIVACY,UPLOAD DATE, VIEW, IM PATH	54
ALBUM TABLE	ALBUM ID	PHOTO ID	TITLE, DESCRIPTION, VIEW	16
MEMBER DETAILS TA	MEMBER ID	PHOTO ID	NAME, EMAIL, ADDRE	52
COMMENT TABLE	COMMENT ID	PHOTO ID	COMMENT DATE, COMMENT	53
LOCATION TABLE	LOCATION ID	Foreign Key	CITY, PLACE	57
TAG PHOTO TABLE	TAG ID	PHOTO ID	ATTRIBUTES	50
TYPE PHOTO	PHOTO ID	FOREIGN KEY	PHOTOGRAPHER, TYP	50
CLIENT TABLE	CLIENT ID	PHOTO ID	CLIENT NAME, NUMB PHOTOS	50
EVENT TABLE	EVENT ID	PHOTO ID	CLIENT ID, PHOTOGRA ID, EVENTNAME	50

PHOTOGRAPHERS	PHOTOGRAPHERPHOTO ID		PHOTOGRAPHY NAME PHOTO GRAPHY NUM TYPE OF PHOTOGRAP	53
COMMENT AUDIT TA	COMMENT ID	PHOTO ID	COMMENT DATE, COMMENT	19
PHOTOPRINT TABLE	PRINTID	PHOTOID	PRINT TYPE	54

Views & Stored Procedures created on Database

Stored Procedure

One :Adding watermark to Fashion photos

```
CREATE PROCEDURE generate_fashion_watermark()
BEGIN
DECLARE WaterMark VARCHAR (50);
DECLARE TYPE_NEW VARCHAR(50);
SET watermark = "FALSE" ;
SELECT TYPE INTO TYPE_NEW FROM TYPE_T
WHERE type = "Fashion" ;
IF Type_NEW = "Fashion" THEN
SET WaterMark = "TRUE";
END IF ;
Update type_t set Water_Mark = Watermark WHERE TYPE = "FASHION" ;
END //
DELIMITER ;
```

Result : We created this procedure to add a watermark everytime we add a fashion photo in our database. We can always create one more procedure by just changing the “Fashion” to a different type in the future if we want to add a watermark. It makes it easier when a client asks for a customized picture album with having watermark on specific photo types.

Sample : call generate_fashion_watermark();

Result Grid Filter Rows: Search Export:

photoID	Photographer_Na...	Type	Water_Mark
1	Manish Daria	Potrait	NULL
2	Melissa_Marly	Landscape	NULL
3	Janey_Sandra	Candid	NULL
4	Alia_Reddy	Traditional	NULL
5	Susan_Mary	Panoramic	NULL
6	Jacob_Lopez	Street Photography	NULL
7	Emily_Rose	Nature	NULL
8	Oliver_Walker	Black and White	NULL
9	Evelyn_Moon	Macro	NULL
10	James_Thomas	Cityscape	NULL
11	Megan_Lewis	Fashion	TRUE
12	Isaac_Marshall	Abstract	NULL
13	Natalie_Jones	Portrait	NULL
14	Benjamin_Wong	Astrophotography	NULL
15	Sophia_Lin	Travel	NULL
16	Daniel_Kim	Night	NULL
17	Emma_Liu	Documentary	NULL
18	Ethan_Chen	Fine Art	NULL
19	Ava_Wang	Wildlife	NULL
20	David_Lee	Architecture	NULL
21	Grace_Kim	Food	NULL
22	William_Chen	Urban	NULL
23	Chloe_Zhang	Sports	NULL
24	Samuel_Li	Seascape	NULL
25	Abigail_Smith	HDR	NULL
26	Lucas_Kang	Family	NULL

Sample Result : As you can see in the above procedure, as soon as I called the procedure, Only the fashion type changed into true and the others are null.

Two : To get all the records in the Member Details

```
DELIMITER //

CREATE PROCEDURE totalMembers()

BEGIN

DECLARE total INT;

DECLARE i INT DEFAULT 0;

SELECT COUNT(*) INTO total FROM memberDetails;

WHILE i < total DO

SET i = i + 1;

END WHILE;

SELECT CONCAT('Total Members: ', total) AS 'Result';

END //

DELIMITER ;
```

Sample : To get all the records and give the total number of members in the database.

Result : call totalmembers;

Result
Total Members: 48

Three- To get a list of album's that have been viewed more than specific number of times.

```
DELIMITER //
CREATE PROCEDURE GetAlbumsByView(Viewcount INT)
BEGIN
DECLARE views INT;
SELECT count(View) into views FROM Album WHERE View > Viewcount;
Select views as ViewCount;
END //
DELIMITER ;
```

Sample : Everytime we need to see how many times a album is viewed, we created a procedure. It often helps us retrieve and manage our most viewed photos.

Result : Call GetAlbumsByView(100);

ViewCount
4

From the above code, we know that 4 Album's have been viewed more than 100 times.

Four : To update a new address of a member

```
DELIMITER //
CREATE PROCEDURE updateAddresses(member_ID INT, newAddress VARCHAR(255))
BEGIN
DECLARE i INT DEFAULT 0;
DECLARE total INT;
SELECT COUNT(*) INTO total FROM memberDetails;
WHILE i < total DO
SET i = i + 1;
UPDATE memberDetails SET Address = newAddress WHERE MemberID = member_id;
END WHILE;
SELECT CONCAT('Addresses Updated to: ', newAddress) AS 'Result';
END //
DELIMITER ;
```

Result : Everytime we need to update a new address for a member , instead of using alter table everytime, we created a function which makes it easier.

Sample : Before running the code

MemberID	PhotoID	Name	Email	Address
23	1	SQL	SQL@gmail.com	213 Cedar Street
212	2	Robin	Robin@gmail.com	231 Cedar Street
213	3	Barney	Barney@gmail.com	213 M G Road
241	4	Ted	Ted@gmail.com	241 Khairahtabad

Here in the above image, you can see the Address of member 241 is “241 Khairathabad”.

Code : call updateAddresses(241, '222 West Reneer Road');

MemberID	PhotoID	Name	Email	Address
23	1	SQL	SQL@gmail.com	213 Cedar Street
212	2	Robin	Robin@gmail.com	231 Cedar Street
213	3	Barney	Barney@gmail.com	213 M G Road
241	4	Ted	Ted@gmail.com	222 West Reneer Road

After running the code the address changed to “222 West Reneer Road”.

VIEW:

1)

View 1 - latest Photos

```
create view latestPhotos AS select * from photo

Where UploadDate > DATE_SUB(NOW(), INTERVAL 1 YEAR) order by uploadDate desc;
```

Result – The above mentioned view retriews the last one year photos from the photo table

Given below the sample output of the above view:

select * from latestPhotos;

photoID	AlbumID	LocationID	MemberID	Title	Description	Privacy	UploadDate	View	ImagePath
13	64	164	1064	College	Description for College	Public	2023-01-12	4	/path/64/13.jpg
29	73	173	1073	Graduation Day	Description for Graduation Day	Private	2023-01-12	24	/path/73/29.jpg
45	73	173	1073	Graduation Day	Description for Graduation Day	Private	2023-01-12	7	/path/73/45.jpg
14	73	173	1073	Graduation Day	Description for Graduation Day	Private	2022-09-13	1	/path/73/14.jpg
30	73	173	1073	Graduation Day	Description for Graduation Day	Private	2022-09-13	5	/path/73/30.jpg

Stored Triggers

One - Comments Audit Trigger Insert

```
drop trigger if exists comments_audit_trigger_insert;

DELIMITER //

CREATE TRIGGER comments_audit_trigger_insert

BEFORE INSERT on comment

FOR EACH ROW

BEGIN

DECLARE action_type VARCHAR(10);

DECLARE old_Comment VARCHAR(10);
```

```
    SET action_type = 'INSERT';

    SET old_Comment = NULL;

  INSERT INTO comments_audit (commentID, oldComment, newComment, action)

  VALUES (NEW.commentID, old_Comment, new.comment, action_type);

END //
```

Result - The above trigger executes whenever the photographer inserts new data to the comments table,

For Example, Photographer adds comments to a picture saying “Low Exposure”. Then Trigger gets Executed and comment_audit table is updated inside the trigger.

Since this is a new comment, old Comment in the comments audit table is given as null and new comment is getting updated from the photographers comments and to keep a track of all the comment;, Action, Time and Comment ID are also updated in comments_audit table.

Sample Output:

INSERT into comment Values (102,12,'2021-04-01','Blurry picture');

commentID	oldComment	newComment	action	timestamp	photoCopy
92	NULL	Eyes CLosed	INSERT	2023-05-03 10:41:48	NULL
93	NULL	bad lighting	INSERT	2023-05-03 10:41:48	NULL
94	NULL	Good Picture	INSERT	2023-05-03 10:41:48	NULL
95	NULL	Over exposure	INSERT	2023-05-03 10:41:48	NULL
97	NULL	Eyes CLosed	INSERT	2023-05-03 10:41:48	NULL

comments_audit 74 Read Only

Two - Comments Audit Trigger

```
drop trigger if exists comments_audit_trigger;

select * from comment;

DELIMITER //

CREATE TRIGGER comments_audit_trigger

BEFORE UPDATE on comment

FOR EACH ROW

BEGIN
```

```
DECLARE action_type VARCHAR(10);

  SET action_type = 'UPDATE';

 INSERT INTO comments_audit (commentID, oldComment, newComment, action)

 VALUES (OLD.commentID, old.comment, new.comment, action_type);

END //
```

Result - The above trigger executes whenever the photographer updates data in the comments table,

For Example, Photographer adds comments to a picture saying “Low Exposure”. Then Trigger gets Executed and comment_audit table is updated inside the trigger.

Whenever trigger gets executed,old data is saved in comments table as old comment and new comment is updated from the photographers comments. Action, Time and Comment ID are also updated in comments_audit table. A single picture can be edited n number of times so the commentID in comments table is not unique.

Sample Output:

Update comment
set comment = 'Too low Contrast' where commentID = 102;

Update comment
set comment = 'Good Resolved' where commentID = 118;

commentID	oldComment	newComment	action	timestamp	photoCopy
118	Too low Contrast	Too low Contrast	UPDATE	2023-05-02 14:27:56	NULL
118	Too low Contrast	Good Resolved	UPDATE	2023-05-02 14:28:04	NULL
NULL	NULL	NULL	DELETE	2023-05-02 14:28:23	222
118	Good Resolved	Too low Contrast	UPDATE	2023-05-02 16:29:44	NULL
118	Too low Contrast	Good Resolved	UPDATE	2023-05-02 16:29:44	NULL

Three - Tag Delete Trigger

```
drop trigger if exists tag_delete_trigger;

DELIMITER //

CREATE TRIGGER tag_delete_trigger
```

```
after DELETE on tag_photo

FOR EACH ROW

BEGIN

DECLARE action_type VARCHAR(10);

   SET action_type = 'DELETE';

  INSERT INTO comments_audit (action, photoCopy)

  VALUES (action_type, old.tagID);

END //
```

Result: A single picture can have many tags. Here a tag represents persons in the picture. All this tag details are saved in Tag Photo table. The trigger above executes whenever someone tries to delete data from the tag photo table. The above trigger saves the deleted info in the audit table with Action Type and also Tag ID.

Sample Output:
delete from tag_photo where tagID = 919;

action	timestamp	photoCopy
DELETE	2023-05-02 14:28:23	222
DELETE	2023-05-02 16:58:19	919

Stored Function

One - Get Location From Photo

```
DELIMITER //

CREATE FUNCTION getLocationFromPhoto(ID INT)

RETURNS VARCHAR(255)

READS SQL DATA

DETERMINISTIC
```

```
BEGIN

DECLARE location VARCHAR(255);

DECLARE location_ID VARCHAR(255);

        select locationID into location_id from photo where photoID = ID;

  SELECT city into location

  FROM location

  WHERE LocationID = location_id;

  RETURN location;

END //
```

Result: Every photo has a location tagged to it. In the Photo Table only Location ID is given. Location table is the lookup table for Photo. So, the above function is used to retrieve location from the photo table using photoID. GetLocationFromPhoto Function accepts photo Id as an input parameter. This function retrieves location ID from the photo table and then passe the location ID to to location table and returns the location name as output.

Sample result:

Select photoID, locationID, UploadDate, View, getLocationFromPhoto(2) as City from photo where photoID = 2;

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

photoID	locationID	UploadDate	View	City
2	101	2021-10-27	12	Hyderabad

Two - To get the the number of photographers in a given type of photography

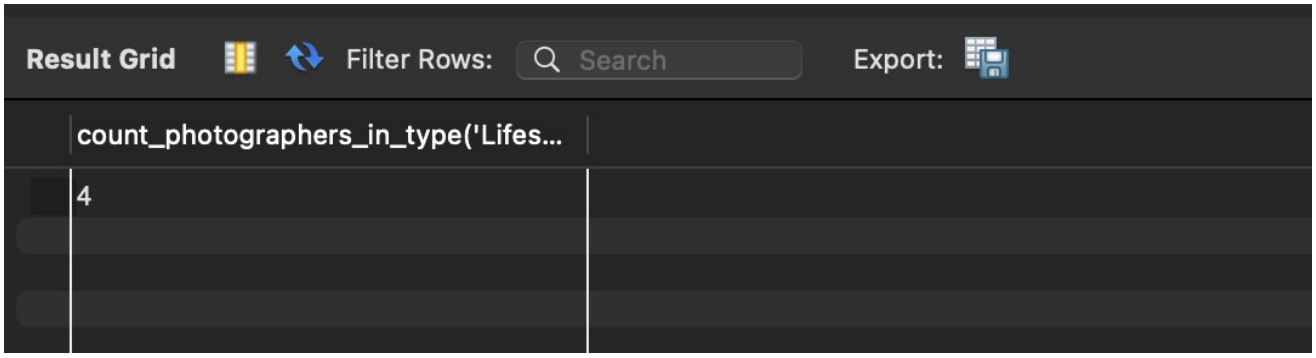
```
DELIMITER //
 CREATE FUNCTION count_photographers_in_type(type_name VARCHAR(50))
 RETURNS INT
 BEGIN
   DECLARE num_photographers INT;
```

```
  SELECT COUNT(*) INTO num_photographers FROM photographers_T WHERE Type_of_photography =
type_name ;
  RETURN num_photographers;
END //

DELIMITER ;
```

Result: Every photo is taken by a different photographer. However, we cannot differentiate which how many different kinds of photographers we have. So, the above function is used to retrieve the total number of photographers in one type. We can call the above function with the type for example :fashion and we get the number. This makes it easier for a business to keep track of the photographers and helps while recruiting to understand the shortage/ excess of one type.

Sample Result : SELECT count_photographers_in_type('Lifestyle');



Three - To calculate the cost of an album based on print and album type

```
DELIMITER //
CREATE FUNCTION calculate_cost(album_title VARCHAR(255), print_type VARCHAR(255), num_photos INT)
RETURNS DECIMAL(10,2)
READS SQL DATA
DETERMINISTIC
BEGIN
DECLARE cost_per_photo DECIMAL(10,2);
DECLARE cost_per_print DECIMAL(10,2);
DECLARE total_cost DECIMAL(10,2);
IF album_title IN ('wedding', 'Reception', 'Birthday') THEN
```

```
SET cost_per_photo = 30;
ELSEIF album_title LIKE “%Trip%”  THEN
SET cost_per_photo = 25;
ELSE
SET cost_per_photo = 15;
END IF;
IF print_type = 'hard' THEN
SET cost_per_print = 20;
ELSE
SET cost_per_print = 10;
END IF;
SET total_cost = (num_photos * cost_per_photo * cost_per_print);
RETURN total_cost;
END//
DELIMITER ;
```

Result: Each photo from each different album has a different cost. The cost for print type is also different. This function calculates the total cost of the album depending on the print type and the album and the number of photos needed.

Sample Result : SELECT calculate_cost('wedding', 'hard', 20);

Total_Cost
10000.00

Four- Get Email Address By ID

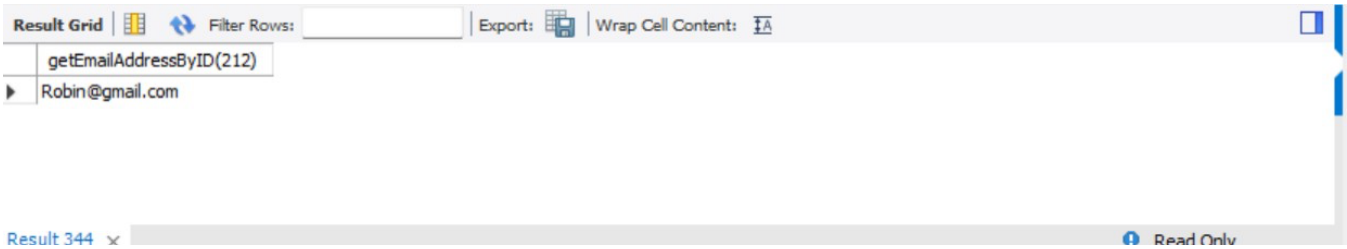
```
drop function getEmailAddressByID;
DELIMITER //
CREATE FUNCTION getEmailAddressByID(member_ID INT)
RETURNS VARCHAR(255)
READS SQL DATA
DETERMINISTIC
BEGIN
DECLARE emailAddress VARCHAR(255);
SELECT Email INTO emailAddress FROM memberDetails WHERE MemberID = member_ID;
RETURN emailAddress;
```

```
END //
```

Result: Member Details Table has all the details of a particular member in the table

The function mentioned above takes memberID as input parameter and returns member email Address as output

SELECT getEmailAddressByID(212);



Five - Get Address By Name

```
drop function getAddressByName;
DELIMITER //
CREATE FUNCTION getAddressByName(nameForAddress VARCHAR(255))
RETURNS VARCHAR(255)
READS SQL DATA
DETERMINISTIC
BEGIN
DECLARE addressByName VARCHAR(255);
SELECT Address INTO addressByName FROM memberDetails WHERE Name = nameForAddress;
RETURN addressByName;
END //
DELIMITER ;
```

Result:

Member Details Table has all the details of a particular member in the table

The function mentioned above takes name as input parameter and returns member Address as output

SELECT getAddressByName('Rachel');

	getAddressByName('Rachel')
▶	2731 MC DR