Final report of

**Database Programming Project**

**CSE305**

Team name: FacemashPlus

Team members:

Qiuyao Liu qiuyao.liu@stonybrook.edu

Tarun Lohani tarun.lohani@stonybrook.edu

Jonathan Lynch jonathan.lynch@stonybrook.edu

**E-R Diagram -**

**C:\Users\Tarun\Downloads\E-R Diagram FM+ 3 (1).png**

**** - Entity  - Primary Key

 - Attribute  - Role

**Rationale for E-R Model**

Our E-R diagram have total 13 entities.

We started to build our diagram with entity "**Users**" as users are prime part of our website and all the functionalities revolve around users. Now there are clearly two group of users, one group who will be using the website and the other one who will be managing the backend and advertisements. Hence we split the Users entity in two child entities namely "**Employees**" and "**FMPlus Users**" which have a IsA relationship with the Users. All the common attributes have been put along with the Users entity and specific attributes are assigned to Employees and FMPlus Users. FMPlus Users have a set valued attribute Preferences. This have been denoted by another entity "**Preferences**" following normalization rule.

Employees have tasks related to Advertising and Managers manage these employees and other. But since both of them are essentially Employees, we didn't split them in separate entities but we are maintaining the difference using the EmpType attribute in Employees entity. Managers will have privileges to add, delete and update most of the tables while Representatives will only deal with the advertisement part. Representatives are responsible to publish the advertisements, hence we created one more entity "**Advertisements**" and it is directly joined with Employees.

FMPlus Users can be involved into four functionalities - Managing own page, Group activities, Messaging and Purchases. Groups also can have their own pages. Hence first we created an entity "**Groups**". Afterwards we created an entity "**Pages**" and joined FMPlus Users and Groups both to Pages. Pages will contain posts, hence we created "**Posts**" as another entity joined with Pages. Posts will contain comments, hence we created "**Comments**" as another entity and joined with Posts. We created "**Group Members**" entity to keep track of the FMPlus Users in a Group. Now to support messaging functionality, we created a new entity "**Messages**" which is connected to FMPlus Users and will keep track of all the messages sent between two FMPlus Users. To purchase a particular item, FMPlus Users can have accounts. We could have kept account information inside FMPlus Users but as a user can have multiple accounts (set valued attribute), we used normalization approach to reduce duplicity in FMPlus Users and segregated accounts as a new entity "**Accounts**". Now to represent and keep a history of all the transactions involved in the purchases we made our final entity "**Sales**" which is connected to both Accounts and Advertisements. Since Accounts already contains the FMPlus User's UserId, there is no need to connect Sales with FMPlus Users as well.

This sums up all the rationale behind our E-R Model.

**Relational Model**

CREATE SCHEMA FacemashPlus;

CREATE TABLE Users (

UserId INTEGER,

FirstName VARCHAR(30) NOT NULL,

LastName VARCHAR(30) NOT NULL,

Address VARCHAR(150) NOT NULL,

City VARCHAR(30) NOT NULL,

State VARCHAR(30) NOT NULL,

Zipcode INTEGER NOT NULL,

Telephone INTEGER,

UserType ENUM('Employee', 'FMUser') NOT NULL,

PRIMARY KEY (UserId)

);

CREATE TABLE Employees (

UserId INTEGER,

SsnNo INTEGER NOT NULL,

StartDate DATE NOT NULL,

HourlyRate INTEGER,

EmpType ENUM('Manager', 'Representative') NOT NULL,

PRIMARY KEY(UserId),

FOREIGN KEY (UserId) REFERENCES Users(UserId)

);

CREATE TABLE FMPlusUsers (

UserId INTEGER,

EmailId VARCHAR(60) NOT NULL,

CreationDate DATE NOT NULL,

CreaditCardNumber INTEGER,

Rating INTEGER,

PRIMARY KEY (UserId),

FOREIGN KEY (UserId) REFERENCES Users(UserId)

);

CREATE TABLE Messages (

MessageId INTEGER,

SentDate DATE NOT NULL,

MsgSubject VARCHAR(50),

MsgContent VARCHAR(250),

SenderId INTEGER,

ReceiverId INTEGER,

CHECK (SenderId != ReceiverId),

PRIMARY KEY (MessageId),

FOREIGN KEY (SenderId) REFERENCES Users(UserId),

FOREIGN KEY (ReceiverId) REFERENCES Users(UserId)

);

CREATE TABLE Groups (

GroupId INTEGER,

GroupName VARCHAR(60) NOT NULL,

GroupType ENUM('Club', 'Organization', 'Association'),

OwnerId INTEGER,

PRIMARY KEY (GroupId),

FOREIGN KEY (OwnerId) REFERENCES Users(UserId)

);

CREATE TABLE Group\_Members(

MemberId INTEGER,

GroupId INTEGER,

MemberType ENUM('Member', 'Owner') NOT NULL,

PRIMARY KEY (MemberId, GroupId),

FOREIGN KEY (MemberId) REFERENCES Users(UserId),

FOREIGN KEY (GroupId) REFERENCES Groups(GroupId)

);

CREATE TABLE Pages(

PageId INTEGER,

PageType ENUM('Personal', 'Group'),

OwnerId INTEGER,

GroupId INTEGER,

PostCount INTEGER,

PRIMARY KEY (PageId),

FOREIGN KEY (OwnerId) REFERENCES Users(UserId),

FOREIGN KEY (GroupId) REFERENCES Groups(GroupId)

);

CREATE TABLE Posts (

PostId INTEGER,

AuthorId INTEGER,

PageId INTEGER,

PostDate DATE NOT NULL,

PostContent VARCHAR(250) NOT NULL,

CmntCount INTEGER DEFAULT 0,

LikeCount INTEGER DEFAULT 0,

PRIMARY KEY (PostId),

FOREIGN KEY (AuthorId) REFERENCES Users(UserId),

FOREIGN KEY (PageId) REFERENCES Pages(PageId));

CREATE TABLE Comments (

CommentId INTEGER,

AuthorId INTEGER,

PostId INTEGER,

CmntDate DATE NOT NULL,

CmntContent VARCHAR(250) NOT NULL,

LikeCount INTEGER DEFAULT 0,

PRIMARY KEY (CommentId),

FOREIGN KEY (AuthorId) REFERENCES Users(UserId),

FOREIGN KEY (PostId) REFERENCES Posts(PostId)

);

CREATE TABLE Advertisements (

AdvId INTEGER,

EmployeeId INTEGER,

AdvType ENUM('Cars', 'Clothing', 'Computer') NOT NULL,

AdvDate DATE NOT NULL,

Company VARCHAR(60) NOT NULL,

ItemName VARCHAR(60),

Price INTEGER,

Content VARCHAR (250),

UnitsAvailable INTEGER,

PRIMARY KEY (AdvId),

FOREIGN KEY (EmployeeId) REFERENCES Users(UserId)

);

CREATE TABLE Preferences (

UserId INTEGER,

PrefCategory ENUM('Cars', 'Clothing', 'Computer') NOT NULL,

PRIMARY KEY (UserId, PrefCategory),

FOREIGN KEY (UserId) REFERENCES Users(UserId));

CREATE TABLE Accounts (

AccountNo INTEGER,

UserId INTEGER,

PRIMARY KEY (AccountNo),

FOREIGN KEY (UserId) REFERENCES Users(UserId)

);

CREATE TABLE Sales (

TransactionId INTEGER,

AdvId INTEGER,

AccountNo INTEGER,

TransactionDate DATE NOT NULL,

NoOfUnits INTEGER NOT NULL,

PRIMARY KEY (TransactionId),

FOREIGN KEY (AdvId) REFERENCES Advertisements(AdvId),

FOREIGN KEY (AccountNo) REFERENCES Accounts(AccountNo)

);

**Rationale behind our Relational model**

Once our E-R diagram was ready, we followed that and converted it into the Relational model step by step. First we created a "**Users**" table which contains all the users of our FacemashPlus site which includes FMPlus users and Employees.

Next we created "**Employees**" and " **FMPlusUsers**" tables to store the specific attributes to each group of users. Both these tables have a foreign key UserId from the parent table Users and this is the primary key of the tables as well which maintains IsA relationship. We are considering employee and managers as single entity, since they are all working in FacemashPlus and sharing similar functionalities. Besides, employees are capable to post advertisement on FacemashPlus while Managers have privileges to add, delete and update other tables. The Primary Key is UserId, and its UserId refers UserId in Users table.

Table "**Messages**" is created to store all the messages which has primary key as MessageId. Since both the sender and the receiver are Users, SenderId and ReceiverId are foreign keys referencing to UserId of Users.

Next we created "**Groups**" table which will hold all the groups created by the users of the website. There can be three group types, ‘Club’, ‘Organization’ or ‘Association’. A user could be the owner of group and hence OwnerId references UserId in Users table.

In "**Group\_Members**" table, primary key is combination of MemberId and GroupId. Every user who is member of a group is identified by its MemberId which refers to UserId in Users table, and the GroupId is associated to GroupId in Groups table. MemberType attribute denotes whether the user is a 'member' or 'owner' of the group.

Next in "**Pages**" table, there can be two types pages, 'Personal' or 'Group' denoted by PageType attribute. User has its own personal page, and the group has its group’s page. Every page has its PageId, type, OwnerId or GroupId, and number of posts. The primary key for page table is PageId. There are two foreign keys OwnerId and GroupId referring to UserId in Users table and GroupId in Groups respectively.

"**Post**" table, contains every post’s ID, date, poster, content and comments. The primary key is PostId. Since a post will be contained inside a page and author will be a User, there are two foreign keys, one is AuthorId which refers to the UserId in User Table and other one is PageId referring to PageId in Pages table

"**Comments**" table includes CommentId, AuthorId, CmntDate, and LikeCount. The primary key is CommentId, and since a comment will be a part of a post and will have a user as the author, there are two foreign keys. First one is AuthorId which refers to the UserId in User Table and second one is PostId referring to PostId in Posts table.

Next is "**Advertisements**" table which will store all the advertisement related data. An employee is allowed to post different type advertisements which will be shown to the FMPlus Users according to their preferences. AdvType attribute is kept to keep track of the type of advertisement. The table also contains the price and item information for a particular advertisement. The primary key is AdvId and foreign key is EmployeeId which refers to UserId in Users.

Next is "**Preferences**" table which will store advertisement preferences of a FMPlus User. We have PrefCategory attribute as ENUM type to denote Preference category. UserId denotes the foreign key which refers to the UserId of Users table. Both PrefCategory and UserId combined comprise the primary key for this table.

"**Accounts**" table contains Account number and UserId. The account is created by users, the primary key is AccountNo, and the foreign key is UserId which refers to UserId of Users.

"**Sales**" table contains every transaction and its associated AccountNo data and number of units sold. Primary key for this table is TransactionId and since every transaction is associated with an advertisement and an account we included two foreign keys AdvId referring to AdvId of Advertisements table and AccountNo referring to AccountNo of Accounts table.

**Collaboration plan**

We first split our work as follows -

1. Jonathan - groups, pages, postings, and comments.
2. Tarun - messaging, searching, liking and unliking postings and comments, and group-related privacy issues.
3. Qiuyao - all aspects of targeted advertising and sales.

Then we came up with our modular designs and explained it to each other. We sorted all the mistakes and dependencies by discussion. After that we divided our work as follows -

1. Tarun - created the E-R diagram using draw.io.
2. Jonathan - Wrote the queries in the SQL workbench.
3. Qiuyao - Wrote the rationale for E-R diagram, rationale behind Relational model.

Finally we clubbed all these things to prepare the final report together.

**Transactions**

There are mainly three categories of transactions which can be done in our "FacemashPlus" social networking site -

1. User-Level Transactions
2. Manager-Level Transactions
3. Customer-Representative-Level Transactions

We uploaded some dummy data in our database and executed all the transactions. They are explained in the following sections

**1. User-Level Transactions -**

1. Users can perform the following **general transactions**:

**1.A. Register :** Users can register themselves in our Social networking site. Following transactions would be performed in our database system in backend.

**Transactions :**

|  |  |  |
| --- | --- | --- |
| Table Name | Operation | Input Parameters |
| Users | Insert | EmailId (String), Password (String), FirstName (String), LastName (String), Address (String), City (String), State (String), Zipcode (Integer), Telephone (Integer) |
| FMPlusUsers | Insert | Credit Card Number (Integer) |
| Pages | Insert | - |

**Query :**

SET @EmailId = "t.lohani@gmail.com";

SET @Psswd = "abcdefgh";

SET @FirstName = "Tarun";

SET @LastName = "Lohani";

SET @Address = "Shivalik Nagar";

SET @City = "Haridwar";

SET @State = "Uttarakhand";

SET @Zipcode = "249403";

SET @Telephone = "9458944328";

SET @CreditCardNumber = NULL;

INSERT INTO Users(EmailId, Psswd, FirstName, LastName, Address, City, State, Zipcode, Telephone, UserType)

VALUES (@EmailId, @Psswd, @FirstName, @LastName, @Address, @City, @State, @Zipcode, @Telephone, "FMUser");

INSERT INTO FMPlusUsers VALUES (LAST\_INSERT\_ID(), CURDATE(), @CreditCardNumber, 0);

INSERT INTO Pages(PageType, OwnerId, GroupId, PostCount)

VALUES ("Personal", LAST\_INSERT\_ID(), NULL, 0)

**1.B. Sign in and sign out :** Users can sign in and sign out from the website. No database transactions are required for these actions. These will be handled by the session control..

**1.C. Post messages in their personal pages :** Once a user is registered into our site, they can post messages on their own page. Following transactions would be performed in our database system in backend.

**Transactions :**

|  |  |  |
| --- | --- | --- |
| Table Name | Operation | Input Parameters |
| Posts | Insert | Post Content (String) |

**Query :**

SET @Author = (SELECT UserId FROM Users WHERE UserId=1);

SET @Page = (Select PageId FROM Pages WHERE PageId=1);

SET @PostContent = "NEW POST TEST CONTENT";

INSERT INTO Posts(AuthorId, PageId, PostDate, PostContent)

VALUES(@Author, @Page, CURDATE() , PostContent);

**1.D. Send a message :** User can send a message to another user. Following transactions would be performed in our database system in backend.

**Transactions :**

|  |  |  |
| --- | --- | --- |
| Table Name | Operation | Input Parameters |
| Messages | Insert | Message Subject (String), Message Content (String), Addressee First Name (String), Addressee Last Name (String) |

**Query :**

SET @MsgSubject = "Hello";

SET @MsgContent = "Hi Prince, How are you??";

SET @SenderId = 1; #Will be fetched from the session.

SET @ReceiverId = (SELECT UserId FROM Users WHERE FirstName ="Prince" AND LastName = "Arora");

INSERT INTO Messages(SentDate, MsgSubject, MsgContent, SenderId, ReceiverId)

VALUES(CURDATE(), @MsgSubject, @MsgContent, @SenderId, @ReceiverId);

**1.E. Receive a message :** Once a user receives a message, it can be viewed in the messages section. Following transactions would be performed in our database system in backend.

**Transactions :**

|  |  |  |
| --- | --- | --- |
| Table Name | Operation | Input Parameters |
| Messages | Read | - |

**Query :**

CREATE VIEW Received\_Messages AS

SELECT \* FROM Messages WHERE ReceiverId = 2; # ReceiverId will fetched from the session.

**1.F. Delete a message :** User can delete a message which was sent to him/her by some other user. Following transactions would be performed in our database system in backend.

**Transactions :**

|  |  |  |
| --- | --- | --- |
| Table Name | Operation | Input Parameters |
| Messages | Delete | Message Subject (String), Message Content (String), Addressee First Name (String), Addressee Last Name (String) |

**Query :**

SET @SentDate = 2016-01-09;

SET @MsgSubject = "Hello";

SET @MsgContent = "Hi Prince, How are you??";

SET @SenderId = 1;

SET @ReceiverId = 2; # All these parameters will be fetched from the session and UI

DELETE FROM Messages

WHERE SentDate = @SentDate AND MsgSubject = @MsgSubject AND MsgContent = @MsgContent

AND SenderId = SenderId AND ReceiverId = @ReceiverId;

2. Users can perform the following transactions with regard to their **own groups**:

**2.A. Create a group :** User can create a new group. Following transactions would be performed in our database system in backend.

**Transactions :**

|  |  |  |
| --- | --- | --- |
| Table Name | Operation | Input Parameters |
| Groups | Insert | Group Name (String), Group Type ('Club'/'Organization'/'Association') |
| Group\_Members | Insert | - |

**Query :**

SET @GroupName = "CSE 305, Fall 2016 SBU";

SET @GroupType = "Club";

INSERT INTO Groups(GroupName, GroupType, OwnerId)

VALUES(GroupName, @GroupType, 1);

SET @GroupToAddTo = (SELECT GroupId FROM Groups WHERE GroupId = LAST\_INSERT\_ID());

INSERT INTO Group\_Members(MemberId, GroupId, MemberType)

VALUES(1, @GroupToAddTo, "Owner");

**2.B. Search for a user and add him to a group :** A group owner can search for other users and add them to a group. Following transactions would be performed in our database system in backend.

**Transactions :**

|  |  |  |
| --- | --- | --- |
| Table Name | Operation | Input Parameters |
| Group\_Members | Insert | - |

**Query :**

SET @SearchId = (SELECT UserId FROM Users WHERE UserId = 5); # Will be fetched from the search results

SET @GroupId = (SELECT GroupId FROM Groups WHERE GroupId = 1); # Will be fetched at run time

INSERT INTO Group\_Members(MemberId, GroupId, MemberType)

VALUES(@SearchId, @GroupId, "Member");

**2.C. Make a post :** A group user can create a new post on a group or some other page of another user. Following transactions would be performed in our database system in backend.

**Transactions :**

|  |  |  |
| --- | --- | --- |
| Table Name | Operation | Input Parameters |
| Posts | Insert | Post Content (String) |

**Query :**

SET @AuthorId = 1; # Will be fetched from the session

SET @PageId = 10; # Will be fetched from the UI

SET @PostContent = "Happy Birthday";

INSERT INTO Posts(AuthorId, PageId, PostDate, PostContent, CmntCount, LikeCount)

VALUES (@AuthorId, @PageId, CURDATE(), @PostContent, 0, 0);

**2.D. Comment on a post :** A user can comment on a post. Following transactions would be performed in our database system in backend.

**Transactions :**

|  |  |  |
| --- | --- | --- |
| Table Name | Operation | Input Parameters |
| Comments | Insert | Post Content (String) |

**Query :**

SET @Author = (SELECT UserId FROM Users WHERE UserId=1);

SET @CommentParentId = (SELECT PostId FROM Posts WHERE PostId = 2);

SET @CommentContent = "Thanks a lot";

INSERT INTO Comments(AuthorId, PostId, CmntDate, CmntContent)

VALUES(@Author, @CommentParentId, CURDATE(), @CommentContent);

**2.E. Like a post :** A user can like a post. Following transactions would be performed in our database system in backend.

**Transactions :**

|  |  |  |
| --- | --- | --- |
| Table Name | Operation | Input Parameters |
| Posts | Update | - |

**Query :**

SET @PostToLike = (SELECT PostId FROM Posts WHERE PostId = 1);

UPDATE Posts SET LikeCount = LikeCount + 1 WHERE PostId=@PostToLike;

**2.F. Like a comment :** A user can like a comment. Following transactions would be performed in our database system in backend.

**Transactions :**

|  |  |  |
| --- | --- | --- |
| Table Name | Operation | Input Parameters |
| Comments | Update | - |

**Query :**

SET @CommentToLike = (SELECT CommentId FROM Comments WHERE CommentId = 1);

UPDATE Comments SET LikeCount = LikeCount + 1 WHERE CommentId=@CommentToLike;

**2.G. Remove a user from a group :** Owner of a group can remove a user from the group. Following transactions would be performed in our database system in backend.

**Transactions :**

|  |  |  |
| --- | --- | --- |
| Table Name | Operation | Input Parameters |
| Group\_Members | Delete | User first name (String), User second name (String) |

**Query :**

SET @UserIdToRemove = (SELECT UserId FROM Users WHERE UserId=2);

SET @GroupToRemoveFrom = (SELECT GroupId FROM Groups WHERE GroupId=4);

DELETE FROM Group\_Members WHERE MemberId=@UserIdToRemove AND GroupId=@GroupToRemoveFrom;

**2.H. Remove a post :** Owner of a can remove a post whenever required. Following transactions would be performed in our database system in backend.

**Transactions :**

|  |  |  |
| --- | --- | --- |
| Table Name | Operation | Input Parameters |
| Posts | Delete | - |

**Query :**

SET @PostToRemove = (SELECT PostId FROM Posts WHERE PostId=1);

DELETE FROM Posts WHERE PostId=@PostToRemove;

**2.I. Remove a comment :** Owner of a can remove a comment whenever required. Following transactions would be performed in our database system in backend.

**Transactions :**

|  |  |  |
| --- | --- | --- |
| Table Name | Operation | Input Parameters |
| Comments | Delete | - |

**Query :**

SET @CommentToRemove = (SELECT CommentId FROM Comments WHERE CommentId=1);

DELETE FROM Comments WHERE CommentId=@CommentToRemove;

**2.J. Unlike a post :** A user can unlike a post which was liked by him/her in past. Following transactions would be performed in our database system in backend.

**Transactions :**

|  |  |  |
| --- | --- | --- |
| Table Name | Operation | Input Parameters |
| Post | Update | - |

**Query :**

SET @PostToUnlike = (SELECT PostId FROM Posts WHERE PostId=1);

UPDATE Posts SET LikeCount = LikeCount - 1 WHERE PostId=@PostToUnlike;

**2.K. Unlike a comment :** A user can unlike a comment which was liked by him/her in past. Following transactions would be performed in our database system in backend.

**Transactions :**

|  |  |  |
| --- | --- | --- |
| Table Name | Operation | Input Parameters |
| Comments | Update | - |

**Query :**

SET @CommentToUnlike = (SELECT CommentId FROM Comments WHERE CommentId=5);

UPDATE Comments SET LikeCount = LikeCount - 1 WHERE CommentId=@CommentToUnlike;

**2.L. Modify a post :** The owner of a post can modify its content at any later point of time. Following transactions would be performed in our database system in backend.

**Transactions :**

|  |  |  |
| --- | --- | --- |
| Table Name | Operation | Input Parameters |
| Posts | Update | Post Content (String) |

**Query :**

SET @NewContent = "New content";

SET @PostToModify = (SELECT PostId FROM Posts WHERE PostId=1);

UPDATE Posts SET PostContent = @NewContent WHERE PostId=@PostToModify;

**2.M. Modify a comment :** The owner of a comment can modify its content at any later point of time. Following transactions would be performed in our database system in backend.

**Transactions :**

|  |  |  |
| --- | --- | --- |
| Table Name | Operation | Input Parameters |
| Comments | Update | Comment Content (String) |

**Query :**

SET @NewContent = "New Content";

SET @CommentToModify = (SELECT CommentId FROM Comments WHERE CommentId=2);

UPDATE Comments SET CmntContent = @NewContent WHERE CommentId=@CommentToModify;

**2.N. Delete a group :** The owner of a group can delete the group. Following transactions would be performed in our database system in backend.

**Transactions :**

|  |  |  |
| --- | --- | --- |
| Table Name | Operation | Input Parameters |
| Groups | Delete | Group Name (String) |

**Query :**

SET @GroupToDelete = (SELECT GroupId FROM Groups WHERE GroupId=2); # This will be fetched from the UI.

DELETE FROM Groups WHERE GroupId=@GroupToDelete;

**2.O. Rename a group :** The owner of a group can rename the group. Following transactions would be performed in our database system in backend.

**Transactions :**

|  |  |  |
| --- | --- | --- |
| Table Name | Operation | Input Parameters |
| Groups | Update | New Group Name (String) |

**Query :**

SET @NewGroupName = "New group name";

SET @GroupToRename = (SELECT GroupId FROM Groups WHERE GroupId=1);

UPDATE Groups SET GroupName = @NewGroupName WHERE GroupId=@GroupToRename;

3. User can perform the similar operations with regard to the other groups. Some of the extra transactins with regard to their **other groups** are as follows:

**3.A. Join a group :** The user can put a request to join a particular group. Following transactions would be performed in our database system in backend.

**Transactions :**

|  |  |  |
| --- | --- | --- |
| Table Name | Operation | Input Parameters |
| Group\_Members | Insert | Group Name (String) |

**Query :**

SET @MemberCreatingGroup = 3;

SET @GroupToAddTo = (SELECT GroupId FROM Groups WHERE GroupName="Arsenal");

INSERT INTO Group\_Members(MemberId, GroupId, MemberType)

VALUES(@MemberCreatingGroup, @GroupToAddTo, "Member");

**3.B. Unjoin a group :** The user can remove their subscription from a particular group. Following transactions would be performed in our database system in backend.

**Transactions :**

|  |  |  |
| --- | --- | --- |
| Table Name | Operation | Input Parameters |
| Group\_Members | Delete | Group Name (String) |

**Query :**

SET @MemberCreatingGroup = 3;

SET @GroupToDelete = (SELECT GroupId FROM Groups WHERE GroupName="Arsenal");

DELETE FROM Group\_Members WHERE GroupToDelete = @ GroupToDelete;

**2. Manager-Level Transactions -**

Managers can perform the following transactions :

**A. Add, Edit and Delete employee information :** A manager can add, edit and delete Information for an Employee working under him/her. Following transactions would be performed in our database system in backend.

**Query :**

DROP VIEW IF EXISTS l;

CREATE VIEW l AS

SELECT User.\*, Employee.SsnNo, Employees.StartDate, Employees.HourlyRate, Employees.EmpType

FROM Users

INNER JOIN Employees

ON Users.UserId = Employees.UserId;

SET @UserId = (SELECT UserId FROM Employees WHERE UserId=1);

SET @EmailId = "testmail.gmail.com";

SET @Psswd = "testpass123";

SET @FirstName = "Testfirstname";

SET @LastName = "Testlastname";

SET @Address = "123 Test Address Lane";

SET @City = "Test City";

SET @State = "Test State";

SET @Zipcode = "12435";

SET @Telephone = "123456789";

SET @UserType = "Employee";

SET @SsnNo = "1234567890";

SET @SDate = "1999-01-01";

SET @HrRate = "12";

SET @EmpType = "representative";

UPDATE Users SET EmailId=@EmailId, Psswd=@Psswd, FirstName=@FirstName,

LastName=@LastName, Address=@Address, City=@City, State=@State,

Zipcode=@Zipcode, Telephone=@Telephone, UserType=@UserType

WHERE UserId=@UserId;

UPDATE Employees SET SsnNo = @SsnNo, SDate = @SDate, HrRate = @HrRate,

EmpType = @EmpType

WHERE UserId = @UserId;

**B. Obtain sales report :** Managers can obtain a sales report for a particulr month. Following transactions would be performed in our database system in backend.

**Query :**

SET @TransactionId = (SELECT TransactionId FROM Sales WHERE TransactionId = 1);

SET @TransactionDate = (SELECT TransactionDate FROM Sales WHERE TransactionDate = 2);

SELECT \* FROM Sales

WHERE @TransactionDate = TransactionDate AND @TransactionId = TransactionId;

UPDATE Users SET EmailId=@EmailId, Psswd=@Psswd, FirstName=@FirstName,

LastName=@LastName, Address=@Address, City=@City, State=@State,

Zipcode=@Zipcode, Telephone=@Telephone, UserType=@UserType WHERE UserId=@UserId;

UPDATE Employees SET SsnNo = @SsnNo, SDate = @SDate, HrRate = @HrRate,

EmpType = @EmpType WHERE UserId = @UserId;

**C. List all items being advertised :** Managers can obtain a sales report for a particular month. Following transactions would be performed in our database system in backend.

**Query :**

DROP VIEW IF EXISTS AdvItemsView;

CREATE VIEW AdvItemsView AS

SELECT Advertisements.ItemName, Advertisements.Price, Advertisements.Content, Advertisement.UnitsAvailable

FROM Advertisements

**D. List of transactions by item name or user name :** Managers can produce a list of transactions by item name or by user name. Following transactions would be performed in our database system in backend.

**Query :**

SELECT Sales.TransactionId, Sales.AccountNo, Accounts.UserId, Sales.AdvId, Advertisements.ItemName, Sales.NoOfUnits, Users.FirstName, Users.LastName

FROM Advertisements, Sales, Users, Accounts

WHERE Advertisements.AdvId = Sales.AdvId AND Sales.AccountNo = Accounts.AccountNo AND Accounts.UserId = Users.UserId

ORDER BY Users.LastName ASC;

**E. List of revenue :** Managers can produce a summary listing of revenue generated by a particular item, item type, or customer. Following transactions would be performed in our database system in backend.

**Query :**

SELECT Sales.TransactionId, Advertisements.ItemName, Advertisements.Price, Sales.NoOfUnits

FROM Advertisements, Sales

WHERE Advertisements.ItemName = Sales.ItemName AND Advertisements.AdvId = Sales.AdvId

SELECT Sales.TransactionId, Advertisements.AdvType, Advertisements.ItemName, Advertisements.Price, Sales.NoOfUnits

FROM Advertisement, Sales

WHERE Advertisement.AdvType = ItemType AND Advertisement.AdvId = Sales.AdvId;

SELECT Sales.TransactionId, Sales.AccountNo, Users.LastName, Users.FirstName, Accounts.UserId, Advertisements.ItemName, Advertisements.Price, Sales.NoOfUnits

FROM Advertisements, Sales, Users, Account

WHERE Sales.AccountNo = Accounts.AccountNo AND Accounts.UserId = Users.UserId AND Sales.AdvId = Advertisements.AdvId;

**F. List of most active items :** Managers can produce a list of most active items. Following transactions would be performed in our database system in backend.

**Query :**

SELECT Sales.AdvId, COUNT(\*) AS Active\_item, Advertisements.ItemName, Advertisements.Price, Advertisements.Company

FROM Sales, Advertisements

WHERE Sales.AdvId = Advertisements.AdvId

GROUP BY AdvId

Order BY Active\_item DESC

LIMIT 3;

**G. Customers who purchased a particular item :** Managers can produce a list of all customers who have purchased a particular item. Following transactions would be performed in our database system in backend.

**Query :**

SELECT U.Users, U.LastName, U.FirstName, U.Address, U.EmailId, U.City, U.State, U.Zipcode, U.Telephone

FROM Users U, Sales S, Advertisements A

WHERE U.UserId = Sales.AdvId AND Sales.AdvId = A.AdvId AND A.ItemName = ItemName;

**H. List of all items for a given company :** Managers can produce a list of all items for a given company. Following transactions would be performed in our database system in backend.

**Query :**

SELECT Advertisements.Company, Advertisments.ItemName, Advertisements.Price, Advertisements.UnitsAvailable, Advertisements.AdvDate

FROM Advertisements

WHERE Advertisements.Company = Company;

**3. Customer-Representative-Level Transactions -**

Customer-Representatives can perform the following transactions :

**A. Create an advertisement :** A Customer-representative can create an advertisement. Following transactions would be performed in our database system in backend.

**Query :**

SET @EmployeeId = 1;

SET @AdvType = "Cars";

SET @Company = "BMW";

SET @ItemName = "Sedan";

SET @Price = 35000;

SET @Content = "i 18";

SET @UnitsAvailable = 10;

INSERT INTO Advertisements(EmployeeId, AdvType, Company, ItemName, Price, Content, UnitsAvailable)

VALUES (@EmployeeId, @AdvType, @Company, @ItemName, @Price, @Content, @UnitsAvailable);

**B. Delete an advertisement :** A Customer-representative can also delete an advertisement. Following transactions would be performed in our database system in backend.

**Query :**

SET @AdvId = 1;

SET @AccountNo = 5;

SET @NoOfUnits = 10;

INSERT INTO Sales(AdvId, AccountNo, TransactionDate, NoOfUnits)

VALUES (@AdvId, @AccountNo, CURDATE(), @NoOfUnits);

**C. Record a transaction :** A Customer-representative can record a transaction. Following transactions would be performed in our database system in backend.

**Query :**

SET @AdvId = 1;

SET @AccountNo = 5;

SET @NoOfUnits = 10;

INSERT INTO Sales(AdvId, AccountNo, TransactionDate, NoOfUnits)

VALUES (@AdvId, @AccountNo, CURDATE(), @NoOfUnits);

**D. Produce customer mailing lists :** A Customer-representative can produce customer mailing lists. Following transactions would be performed in our database system in backend.

**Query :**

SELECT EmailId FROM Users U, Sales S, Accounts A

WHERE S. AccountNo = A. AccountNo AND A.UserId = U.UserId AND U.UserType = "FMUser";

**E. Produce item suggestions :** A Customer-representative can produce a list of item suggestions for a given customer based on that customer's past transactions. Following transactions would be performed in our database system in backend.

**Query :**

SELECT \* FROM Advertisements A, FMPluUsers F, Preferences P

WHERE A.AdvType = P. PrefCategory AND F.UserId = P.UserId AND P.UserdId = 1;

**F. Customer's current groups :** A Customer-representative can view a customer's current groups. Following transactions would be performed in our database system in backend.

**Query :**

SELECT GroupName FROM Groups G, Group\_Members GM, FMPlusUsers F

WHERE F.UserId = 1 AND GM.MemberId = F.UserId AND GM. GroupId = G. GroupId;

**G. Account history :** A Customer-representative can view the account history for each of a customer's accounts. Following transactions would be performed in our database system in backend.

**Query :**

SELECT \* FROM SALES S, FMPlusUsers F, Accounts A

WHERE F.UserID = 1 AND F.UserID = A.UserId AND A. AccountNo = S. AccountNo;

**H. Best-Seller list of items:** A Customer-representative can view Best-Seller list of items. Following transactions would be performed in our database system in backend.

**Query :**

SELECT A.ItemName , S.AdvId, SUM(S.NoOfUnits) as Sale FROM SALES S, Advertisements A

GROUP BY (AdvId)

WHERE A. AdvId = S. AdvId

ORDER BY Sale DESC

**I Personalized item suggestion list:** A Customer-representative can show personalized item suggestion list to a user. Following transactions would be performed in our database system in backend.

**Query :**

SELECT \* FROM Advertisements A, FMPluUsers F, Preferences P

WHERE A.AdvType = P. PrefCategory AND F.UserId = P.UserId AND P.UserdId = 1;