Programming Assignment 2 Report

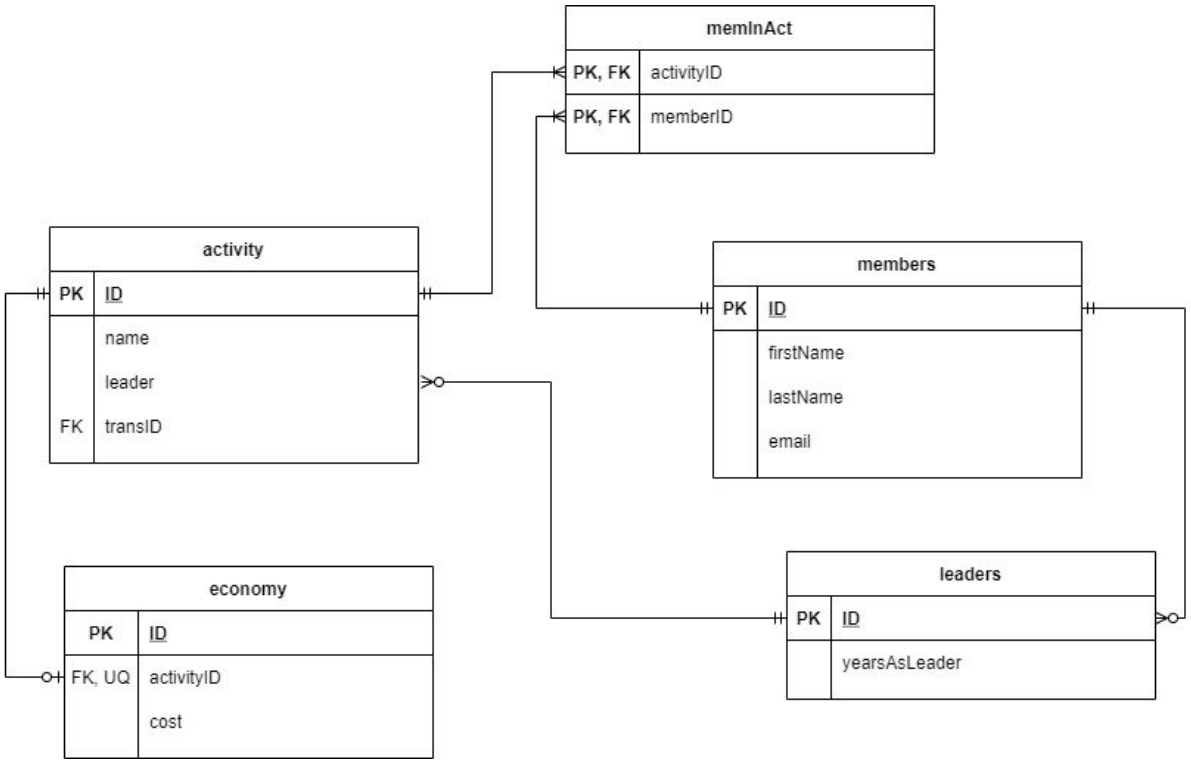
Student(s):	Saleh Shalabi - ss225bx@student.lnu.se Emma Lövgren - el222wg@student.lnu.se

1. Project Idea

The idea for this assignment was to design and implement a member activity system for an association. Here is a description of the idea for the system:

Main user(s): Administrator(s) of the system and the board members of the association.
Problem solved: View statistics about member activities, leaders and cost of the activities.
Main features: Store data about member activities and show statistics for example: show how many members that have participated in an activity, leaders of the activities and the cost of different activities.

2. Schema Design



The design consists of 5 tables (activity, economy, memInAct, members and leaders).

activity
Attributes: ID, name, leader, transID

This table holds information about the activity. The ID for a specific activity will also be found as the attribute *activityID* in the tables *economy* and *memInAct* (member in activity). This will make it possible to get information about the cost for the activity and which members who participated in the activity. In the activity table will also have the attribute *leader* that holds an ID. The leaders ID will also be found in the table *leaders* and in the table *members* because a leader is also a member of the association.

economy
Attributes: ID, activityID, cost.

Some activities are free to arrange and some activities have a cost. The values for the free events are null and the activities that cost will have a value for the attribute cost. The transID (transaction ID) is connected to the ID in the table *economy*, we write more about this later in the report under “4 Discussion and Resources”. The activityID is unique because one transaction is connected to one specific activity and has the relation zero or one.

memInAct
Attributes: activityID, memberID

“memInAct” stands for “member in activity”. This table keeps track of which members have participated in which activities. This table will have duplicates of the *activityID* if more than one member has participated in the activity and duplicates of the *memberID* if one member has participated in more than one activity. ActivityID and memberID together is a primary key. This makes it possible to count how many times the ID for the activity occurs and that shows how many members that have participated in the different activities.

members
Attributes: ID, firstName, lastName, email

This table holds information about the members. Because leaders are members we can get their names and email through this table as well. The ID for the member is also found in the table *memInAct*.

leaders
Attributes: ID, yearAsLeader

This table holds information about the leaders and how many years of experience they have. To get the name of the leader we must use the table *members*, and to see which activity this leader is leader for we must use the table *activity*.

3. SQL Queries

Q: **List all names of the members that participate in an activity**
The following query is a multirelation query that uses GROUP BY. If a member has participated in an activity we will find their *memberID* in the table *memInAct*. Therefore we look where the ID and memberID is equal. Some members participate in multiple activities so we group by the ID as well.

```
================

SELECT firstName, lastName, ID

FROM members, memInAct

WHERE members.ID = memInAct.memberID

GROUP BY ID;

================
```

Q: **Sum the cost of all activities for each leader**
In this query we take data from three tables (economy, activity and members). In the select section we have “leader as ID” to change the name in the column in the table from “leader” to “ID”. We look for matching pairs in the ID from *economy* and the *transID* from *activity* to get the cost, and also where the leader from activity and the ID from members matches to be able to get the names of the leaders. At last we GROUP BY leaders.

```
================

SELECT leader as ID, firstName, lastName, sum(cost) as sumOfCost

FROM economy, members, activity

WHERE economy.ID = activity.transID AND activity.leader = members.ID

GROUP BY  activity.leader;

================
```

Q: **List all names of the members that have participated in more than 2 activities**
In this query we have two select queries, one outer and one inner. In the inner one we count how many activities the different members participate in. In the outer we join the *memberID* from the inner on the matching members ID and add a condition that selects those who have more than 2 activities.

```
================

SELECT firstName, lastName, activitys
```

```
FROM members

JOIN (SELECT COUNT(activityID) AS activitys, memberID

      FROM memInAct

      GROUP BY memberID) AS particByMem ON particByMem.memberID = ID

WHERE activitys > 2;
```

==========

Q: **Select the activity name and the leader's name for the activity with most members in it**

For this task we created a view called *memParAct* that will show us a table with the numbers of members in each activity, and also joins the name for the *activity* and the ID for the *leaders*. From this view we have all we need except the first and last name from leaders so we join the table *members* to get the name of the leader. The WHERE condition helps us to select only the activity which has the most numbers of participants.

==========

```
CREATE VIEW memParAct AS SELECT COUNT(memberID) AS membersParAct,
activityID, name, leader

FROM memInAct

JOIN activity ON activity.ID = activityID

GROUP BY activityID;
```

==========

```
SELECT membersParAct, activityID, name, leader AS leaderID,
firstName, lastName

FROM memParAct

JOIN members ON members.ID = leader

WHERE membersParAct = SELECT MAX(membersParAct) FROM memParAct;
```

==========

Q: **Select the leaders that don't leads any activity**

We select the leaders that do not exist in the activity table as leader then we join this new table on *members* where the ID is the same and finally we select the name of them all.

==========

```
SELECT firstName, lastName, members.ID AS ID

FROM members

JOIN (SELECT * ON leaders WHERE ID  NOT IN (SELECT leader FROM
activity))

AS leadersNoLead ON leadersNoLead.ID = members.ID
```

==========

Q: **Select the sum of Amount for all activity with name X**
This query will select the name of the activity and use SUM() to sum the cost from the economy table. To get the cost from a specific activity we have a WHERE condition that checks where the economy ID matches the transaction ID in the table *activity* and at the same time checks for a specific name in *activity* such as “Fika”

==========

```
SELECT neme, SUM(cost)

FROM activity, economy

WHERE economy.ID = activity.transID AND activity.name = ‘fika’
```

==========

4. Discussion and Resources

We had an issue connecting the economy table with the activity table. The solution to this was to use the following query to update the values.
SInce every activity should only have one *transactionID* and in the table *economy* we could insert few rows with the same *activityID*. We came up with a solution by making the activity in the economy table a unique attribute.

```
UPDATE activity, economy

SET activity.transID = economy.ID

WHERE activity.ID = economy.activityID
```

The project uses CSV files with data for all tables that the python program reads at start.
Clone code from: https://github.com/saleh1shalabi/programing_assignement_2.git
Or download it at : https://github.com/saleh1shalabi/programing_assignement_2
Video demonstration: https://youtu.be/C7QpBZ8ehT0

The code should work properly when the workspace in the IDE is open in the same folder as the code and the other CSV files.