CSE 5330 Project Part 1 , 2 , & 3. Fall 2018

Social Network Database

Team 9:

Golay Nie 1001678015,

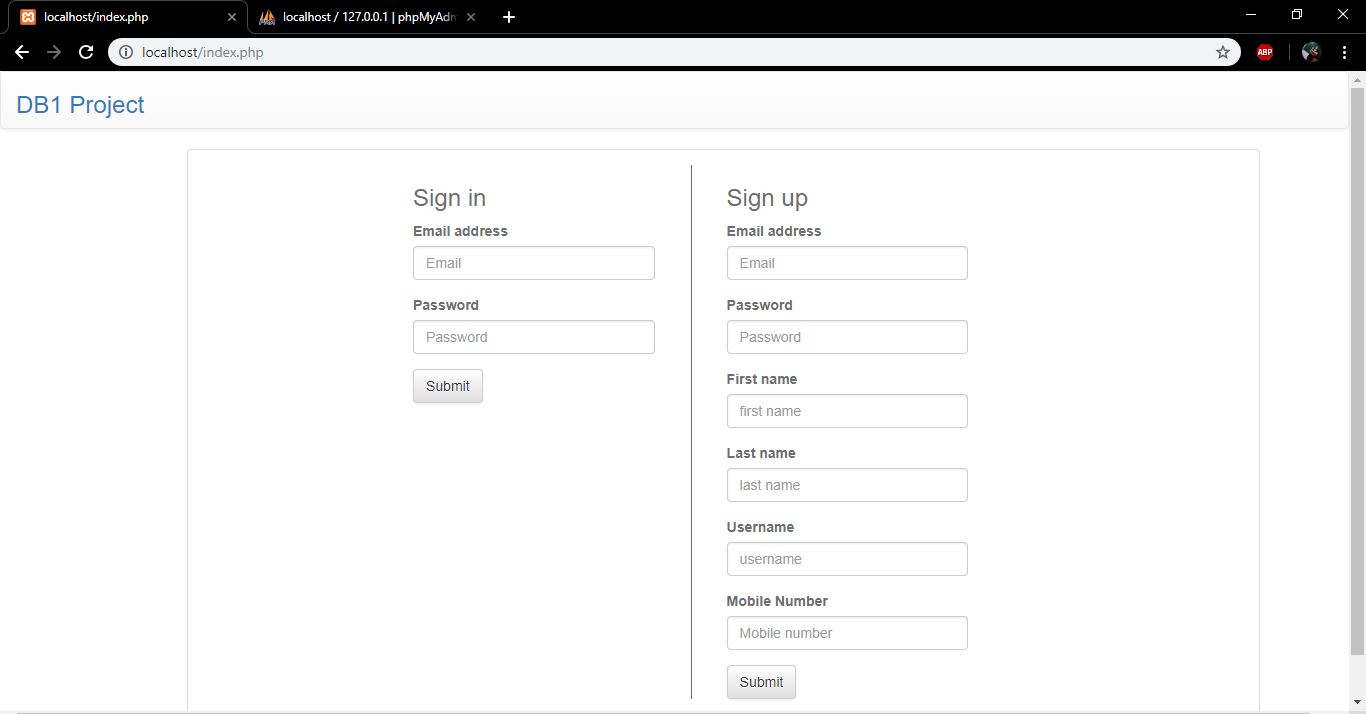
Ketaki Shintre 1001573518,

Eapen Ittiyera Vazhappilly 1001672516

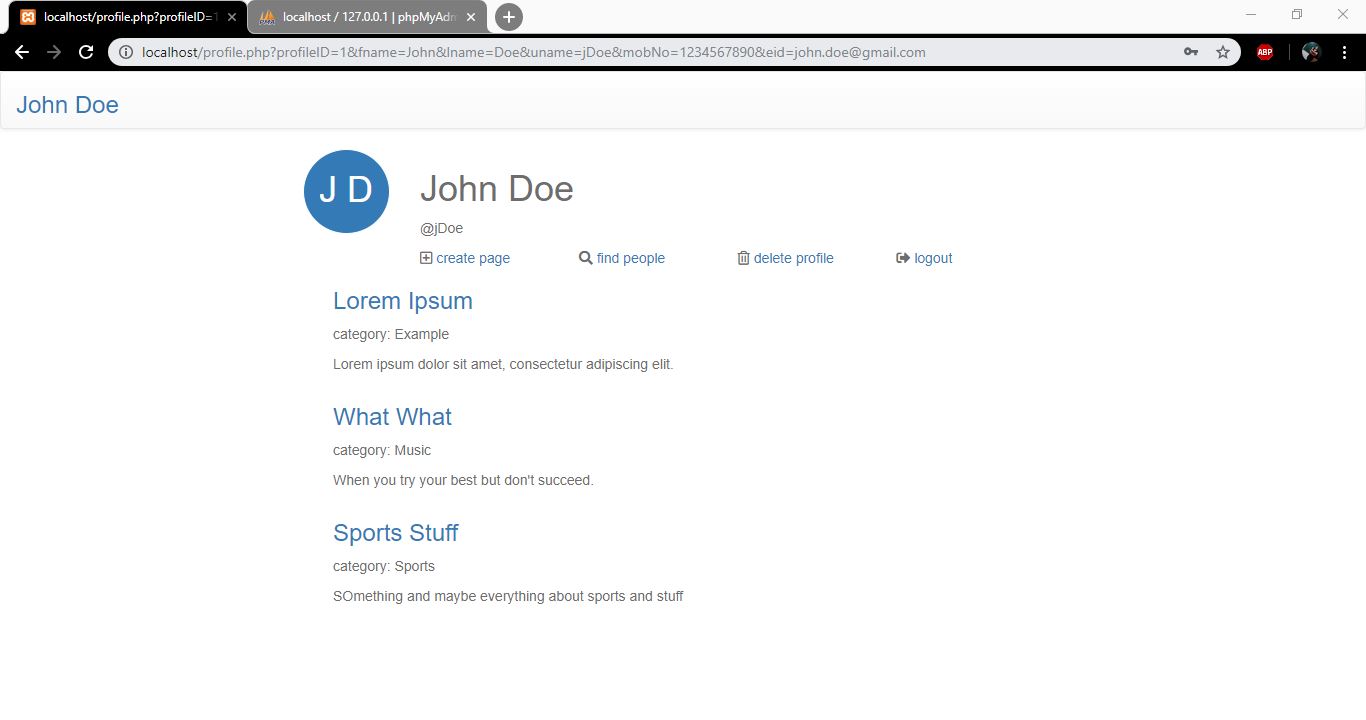
|  |  |
| --- | --- |
| Contents | Page |
| Screen Shots | 2 |
| Printout of Data | 5 |
| EER | 7 |
| Relational Schema Diagram | 8 |
| Documentation and Assumption | 9 |
| Project Setup | 11 |
| Create Table Statements | 12 |
| Transaction Code | 15 |

Screenshots:

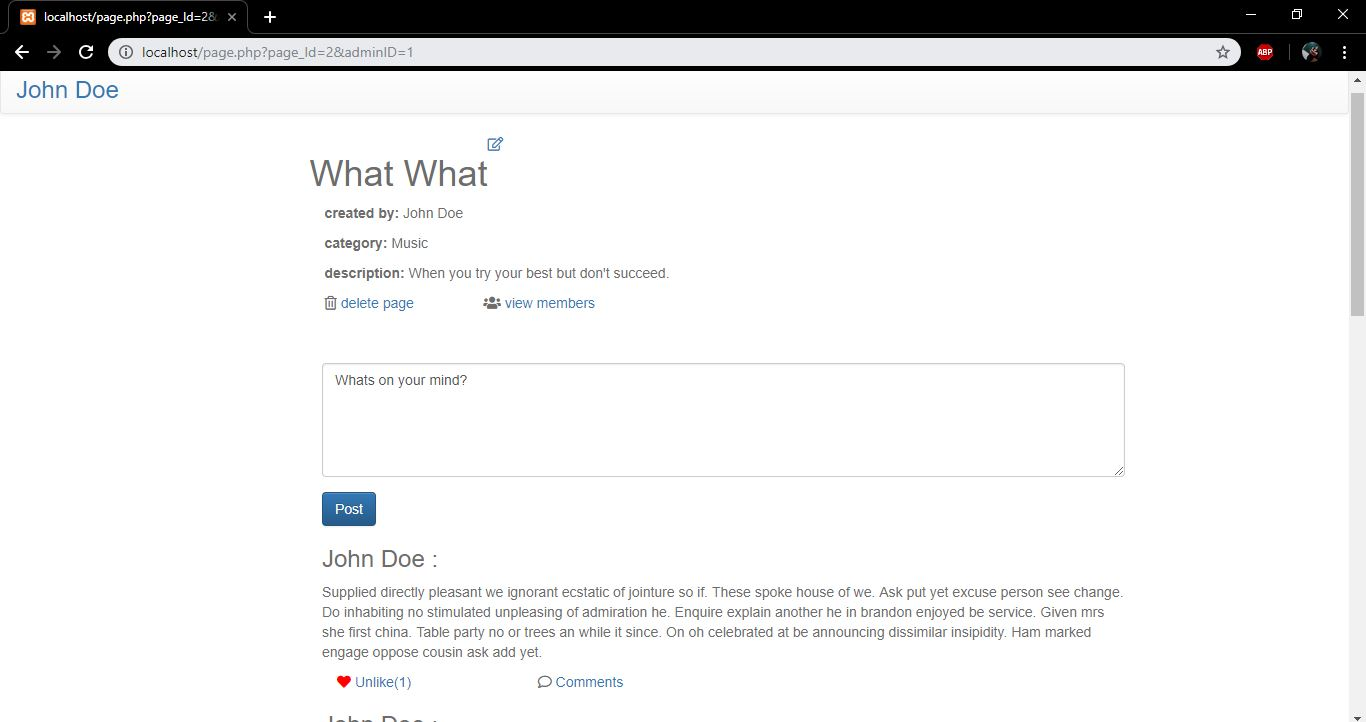
Sign in page:

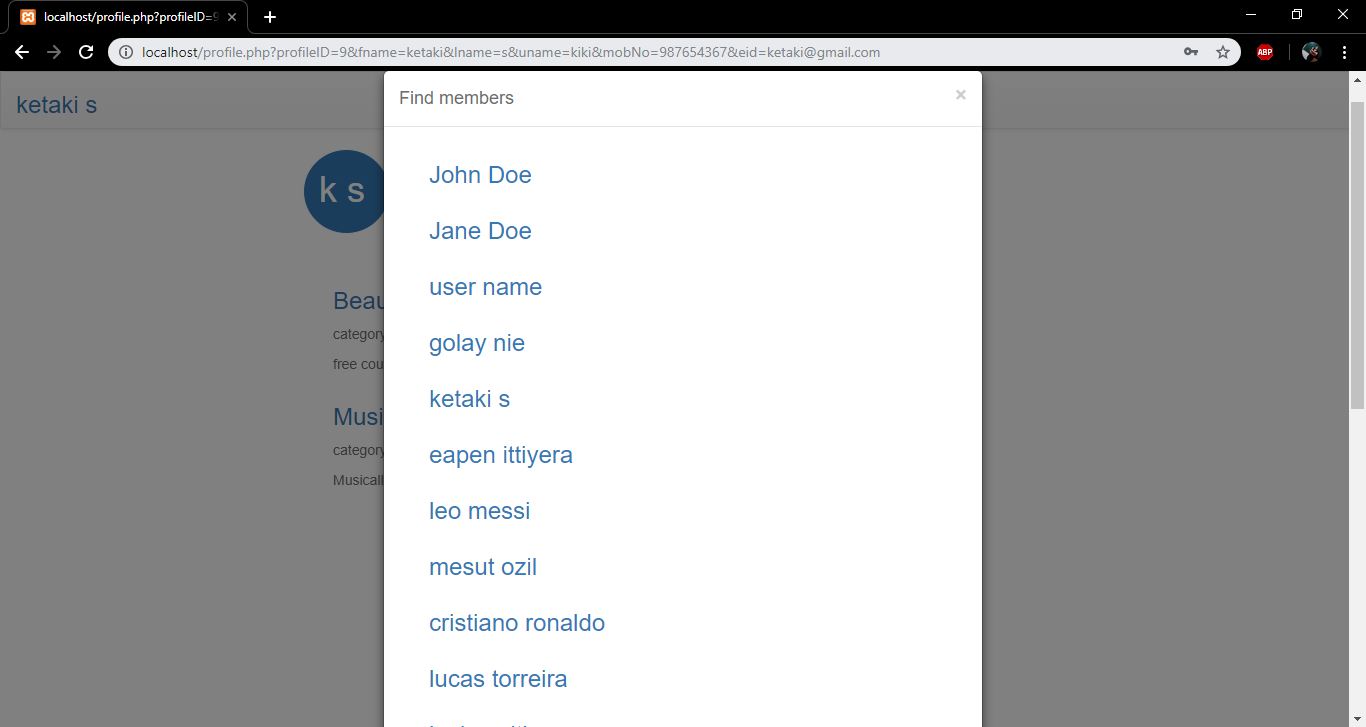
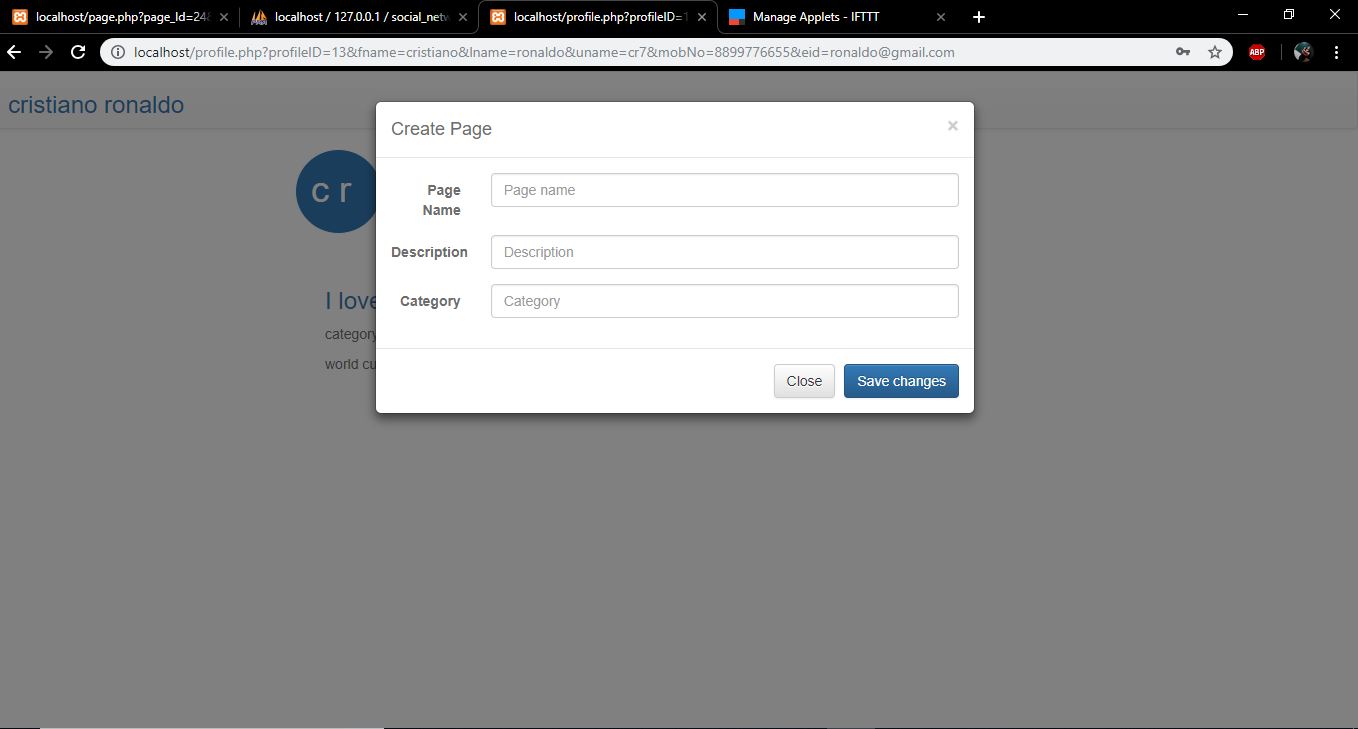


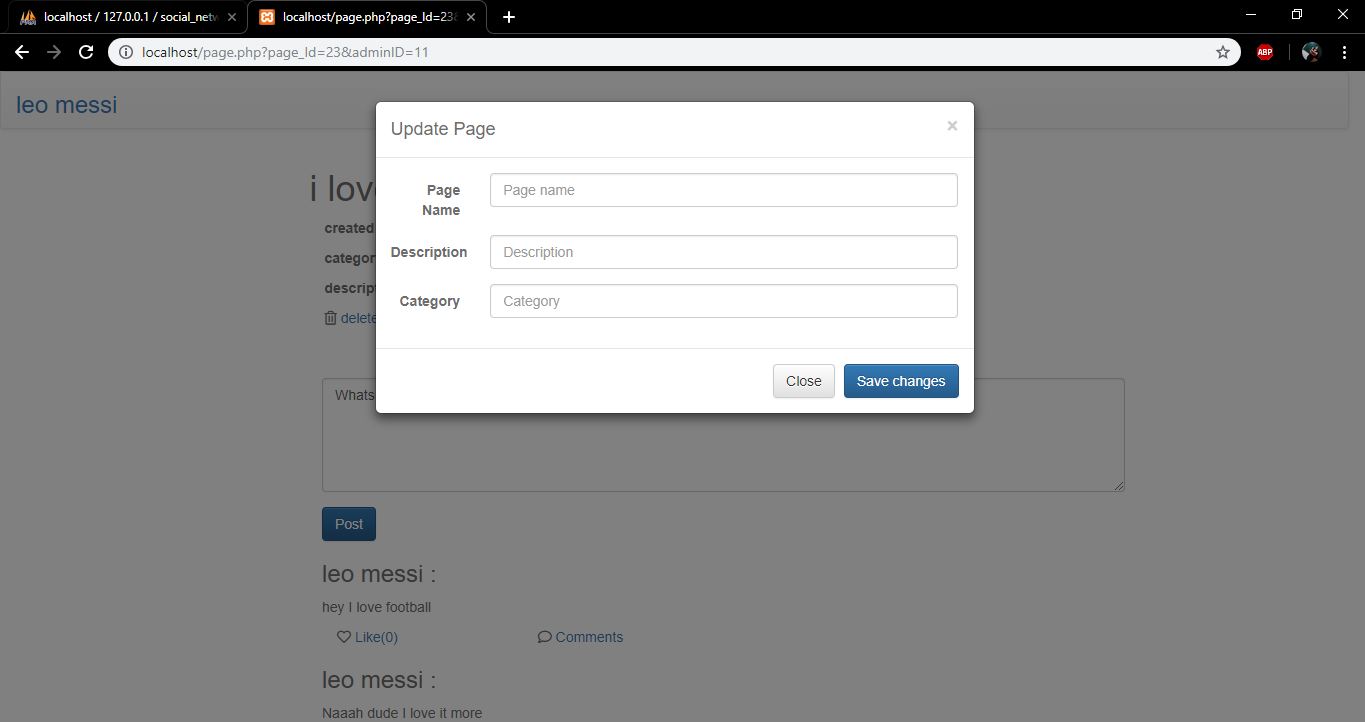
Profile with 3 Pages created by John Doe listed:

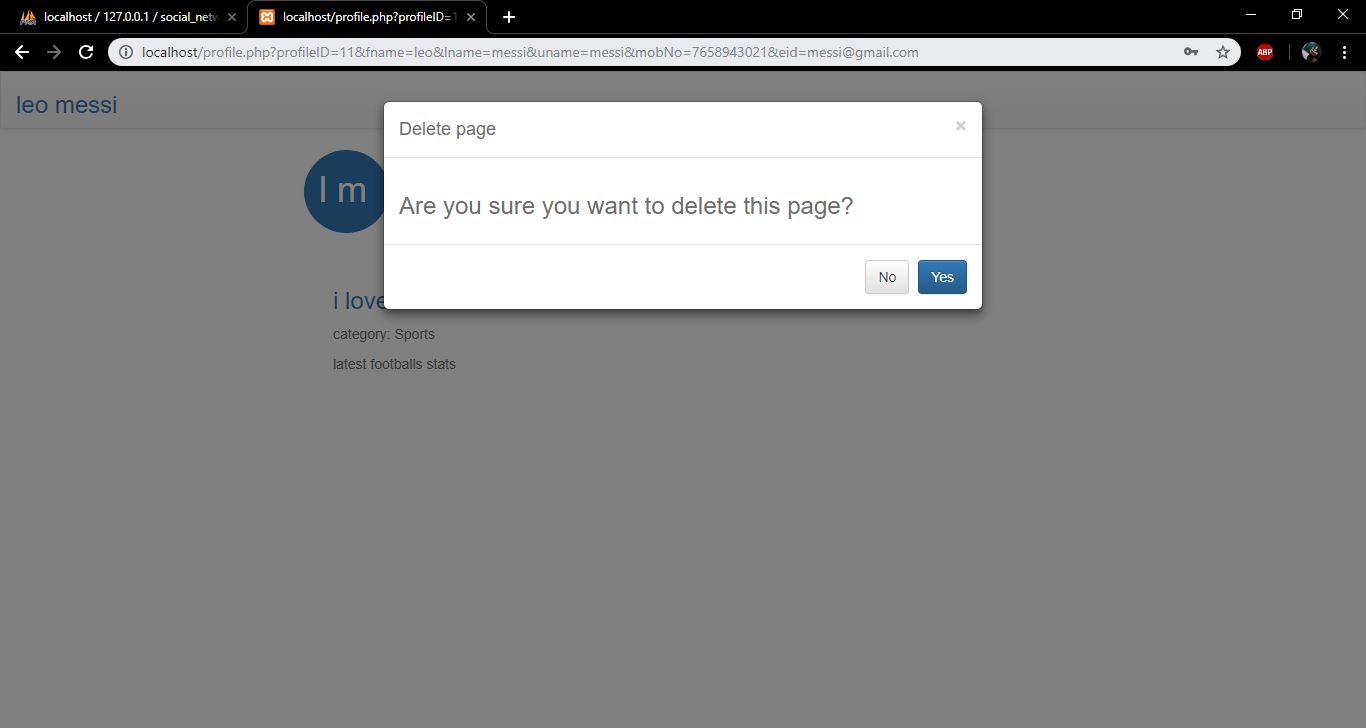
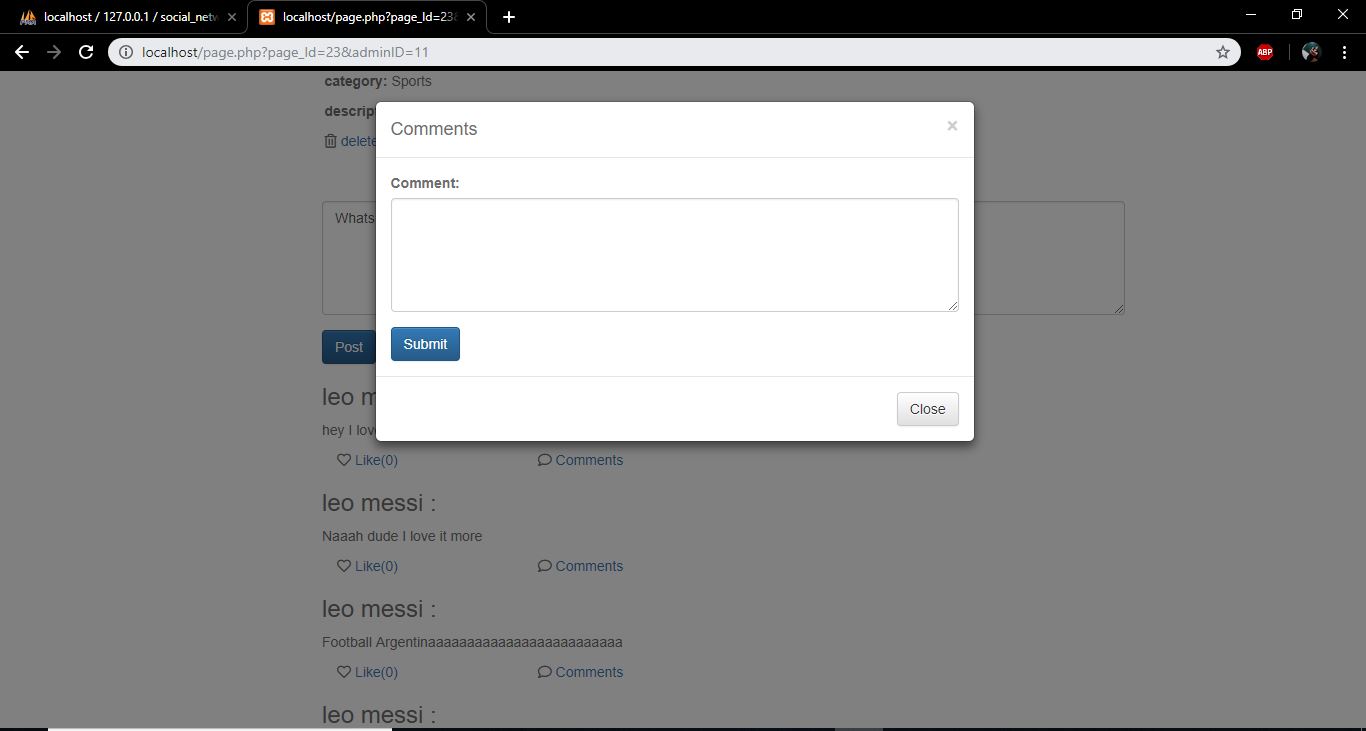


Page with Posts, Comments, likes:



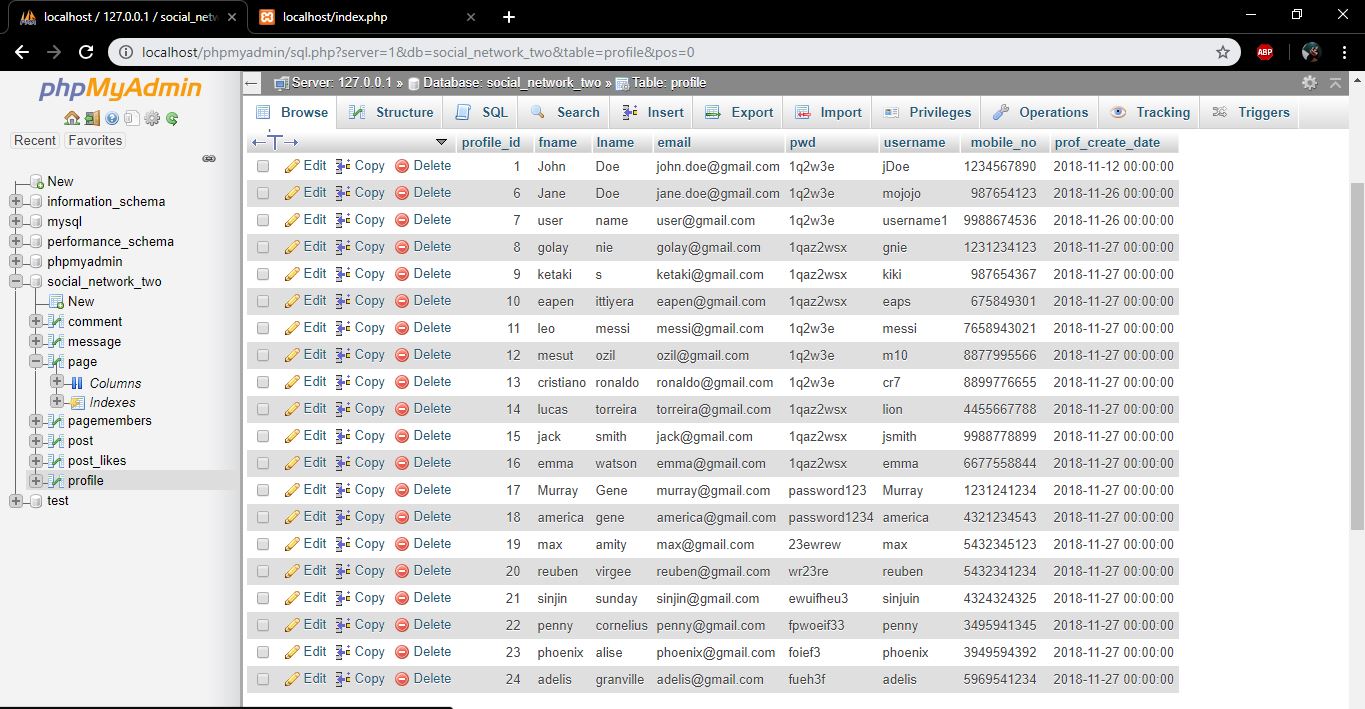
Other functionalities: finding Profiles and their Pages, create/update/delete Page, Comments: 



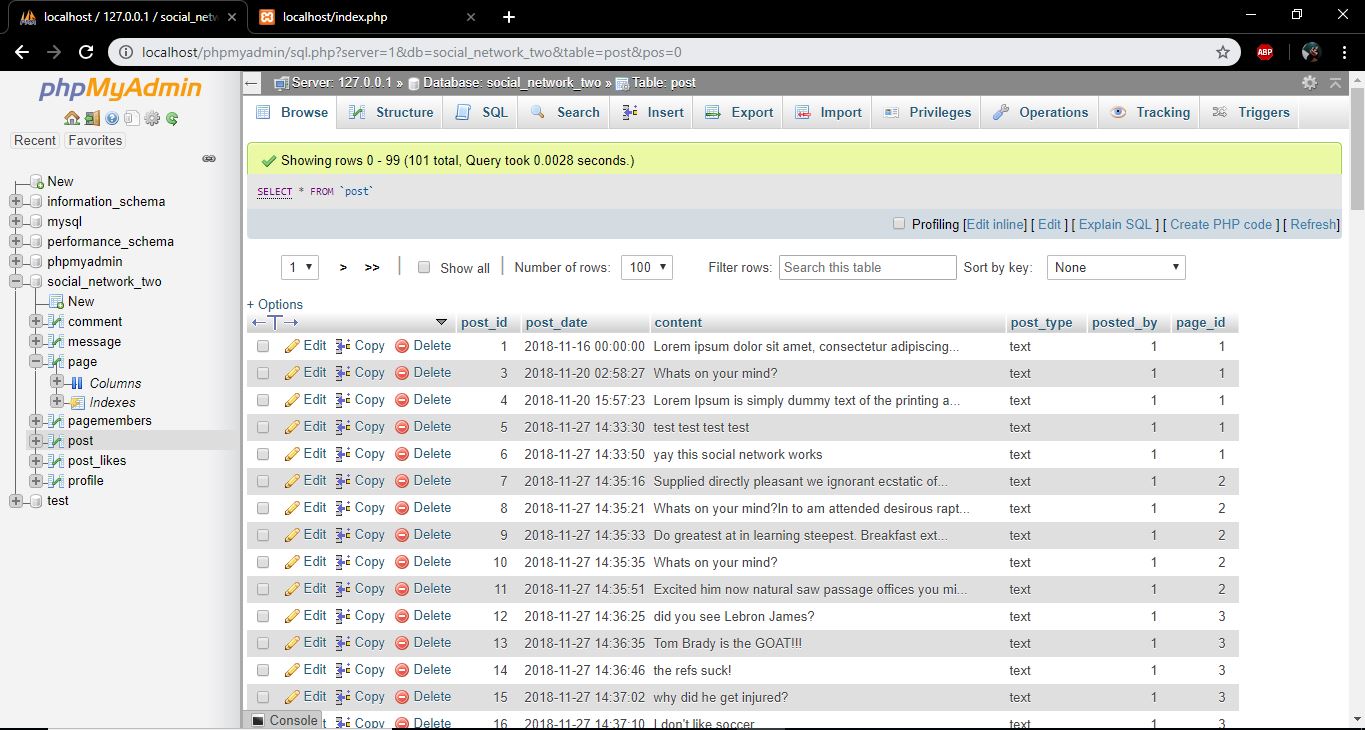
 

Printout of data:

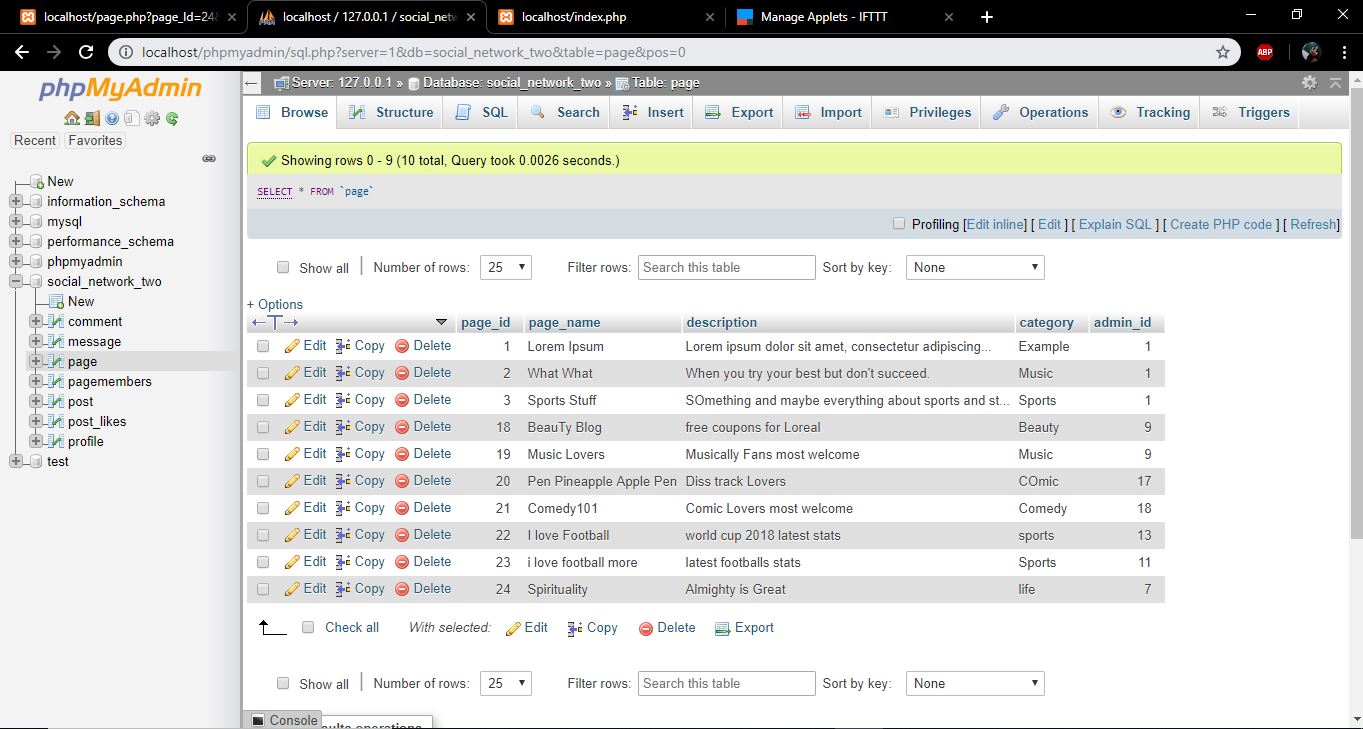
Data in Profile table:



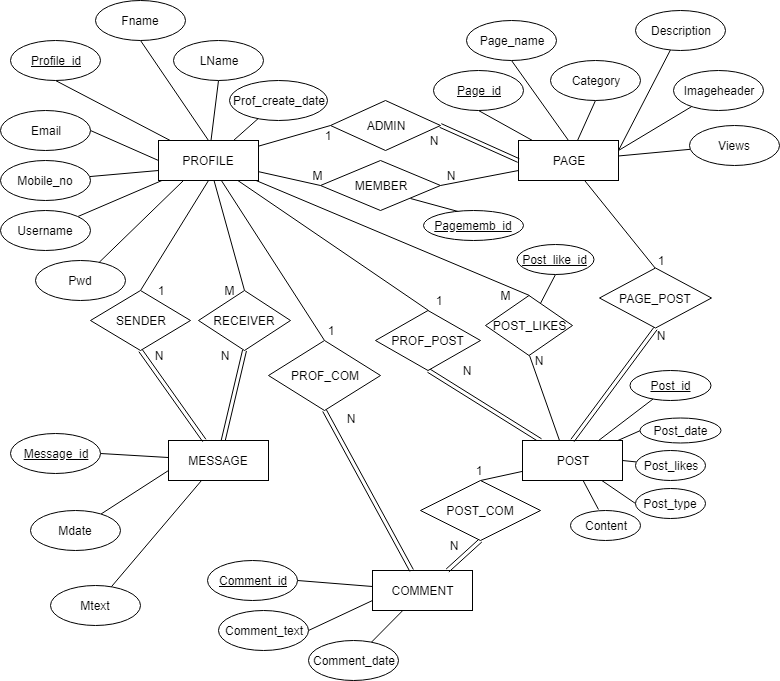
1st page of Data in Post table:



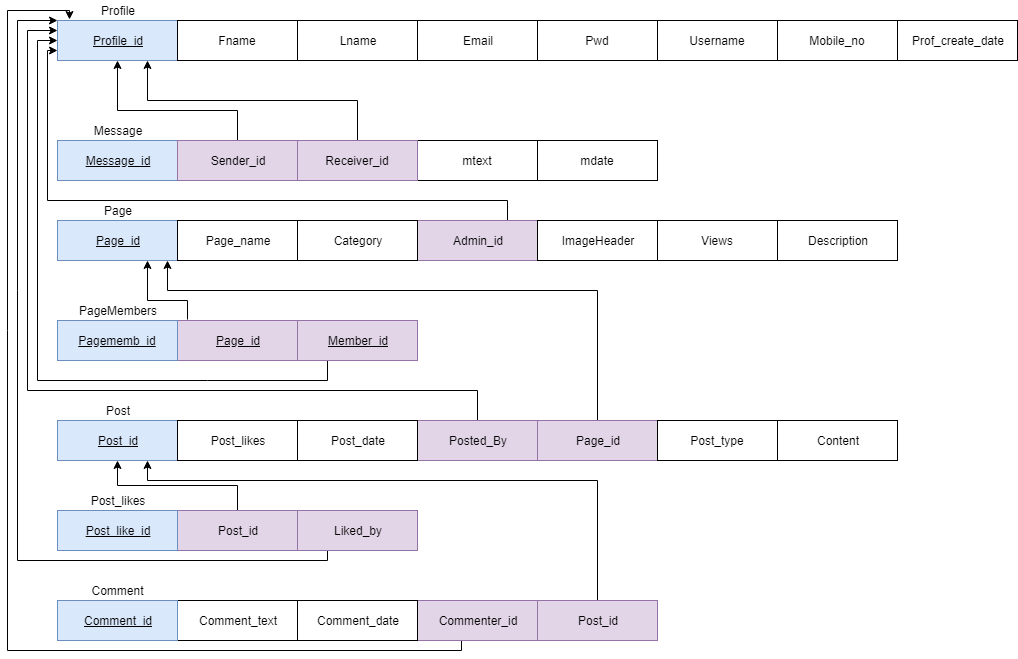
Data in Pages table:



EER (See attached file for larger image):



Relational Schema (See attached file for larger image):



Documentation:

* Profile Entity:
  + created all the attributes listed in the requirements with profile\_id as the auto-incremented primary key.
* Message entity:
  + created all attributes mentioned in the requirements
  + created additional attribute message\_id that is auto-incremented and primary key.
  + created the Sender and Receiver relationship between Profile and Message.
    - Each Message has 1 Profile as sender, and 1 Profile can be Sender of multiple Messages.
    - Each Profile can receive multiple Messages, and each Message can have multiple recipients.
  + a Profile does not need to send or receive messages, and therefore has partial participation, but a Message must have a sender and receiver and has total participation.
  + EER-to-relational mapping:
    - Sender and Receiver relationships infer a sender\_id and receiver\_id which are foreign keys referring to profile\_id.
    - through normalization we created a new table Receiver\_messages with just message\_id and receiver\_id, since each message can have multiple receiver\_ids,
  + We were unable to implement messaging in PHP
* Page entity:
  + created all the attributes mentioned in the requirements, with page\_id as an auto-incremented primary key
  + Admin and Member relationships between Profile and Page:
    - each Page must have 1 Admin Profile (complete participation) and may have multiple Member Profiles (partial).
    - The Member relationship has its own table and primary key to implement tracking of which profiles are members of a page.
    - each Profile can be Admin or Member of any number of Pages (partial participation)
    - EER-to-relational mapping:
      * Admin and Member relationships infer the admin\_id and member\_id attributes which are foreign keys referencing profile\_id.
      * normalization moves member\_id to a new table Page\_Members with just page\_id and member\_id as attributes, since each Page can have multiple members
  + Not mentioned in requirements: If an Admin Profile of a Page is deleted, we decided to delete all the data corresponding to that Page. This is implemented with the “on delete cascade” command listed in the Create Table statements.
  + We were unable to implement the ImageHeader and Views attributes.
* Post entity:
  + created all the attributes mentioned in the requirements
  + created additional attribute post\_id as an auto-incremented primary key
  + created post\_date and post\_likes attributes, which was vague in the requirements
  + Post\_Likes relationship
    - the post\_likes attribute tracks the number of likes, but the Post\_likes relationship has its own table and primary id. It tracks which profiles liked which posts.
  + Page\_Post relationship:
    - each Page may have multiple Posts; A Post must belong to a Page
  + Prof\_Post relationship:
    - each Profile can make any number of Posts; A Post must belong to a single Profile
  + The Content attribute stores the post as text
  + The Post\_type attribute tells if post is text, video, audio, or image. We could not implement these types.
  + EER-to-Relational mapping:
    - Prof\_Post relationship infers the posted\_by attribute which is foreign key referencing a profile\_id in Profile
    - Page\_Post relationship infers the page\_id attribute which is foreign key referencing page\_id in Page
* Comment entity:
  + created comment\_id as an auto-incremented primary key, and comment\_date attribute
  + created comment\_text, which is a necessity to store the comment content but not mentioned in requirements
  + Post\_Com relationship:
    - A Post can have any number of Comments; Each Comment must belong to a Post
  + Prof\_Com reliationship:
    - A Profile can make any number of comments; Each Comment must belong to a Profile
  + EER-to-relational mapping:
    - Post\_Com relationship infers the post\_id attribute, a foreign key referencing post\_id in Post
    - Prof\_Com relationship infers the commenter\_id attribute, a foreign key referencing the profile\_id in Profile
* Delete Cascade:
  + We used the ON DELETE CASCADE action on all the foreign keys.
    - When a Profile is deleted, its corresponding Posts, Comments, Messages, and Pages where it is Admin are all deleted.
    - When a Page is deleted, its corresponding Posts are deleted
    - When a Post is deleted, its corresponding Comments are deleted

Project Setup: -  
According to deliverables We’ve submitted the part1 & part2 of the project also create table SQL queries, Fully filled SQL database, php files etc  
Install XAMP  
Create a database in phpMyAdmin by name social\_network\_two  
Import the SQL database file.  
Go the browser and type localhost.  
And Now you’ll be able to see the project   
For Dummy Trial  
Email – [ronaldo@gmail.com](mailto:ronaldo@gmail.com)  
Password – 1q2w3e

**Create Table Statements (sql files are attached):**

CREATE TABLE profile(

profile\_id int PRIMARY key AUTO\_INCREMENT,

fname varchar(30),

lname varchar(30),

email varchar(50) UNIQUE,

pwd varchar(30),

username varchar(30) UNIQUE,

mobile\_no BIGINT UNIQUE,

prof\_create\_date datetime

);

CREATE TABLE message(

message\_id int PRIMARY KEY AUTO\_INCREMENT,

sender\_id int REFERENCES profile(profile\_id) ON DELETE CASCADE,

receiver\_id int REFERENCES profile(profile\_id) ON DELETE CASCADE,

mtext text,

mdate datetime

);

CREATE TABLE page(

page\_id int PRIMARY KEY AUTO\_INCREMENT,

page\_name varchar(50),

description text,

category varchar(30),

admin\_id int REFERENCES profile(profile\_id) ON DELETE CASCADE

);

CREATE table pageMembers(

pagememb\_id int PRIMARY KEY AUTO\_INCREMENT,

page\_id int REFERENCES page(page\_id) ON DELETE CASCADE,

member\_id int REFERENCES profile(profile\_id) ON DELETE CASCADE

);

CREATE TABLE post(

post\_id int PRIMARY KEY AUTO\_INCREMENT,

post\_likes int,

post\_date datetime,

content text,

post\_type varchar(50),

posted\_by int REFERENCES profile(profile\_id) ON DELETE CASCADE,

page\_id int REFERENCES page(page\_id) ON DELETE CASCADE

);

CREATE TABLE comment(

comment\_id int PRIMARY KEY AUTO\_INCREMENT,

comment\_text text,

comment\_date datetime,

commenter\_id int REFERENCES profile(profile\_id) ON DELETE CASCADE,

post\_id int REFERENCES post(post\_id) ON DELETE CASCADE

);

CREATE TABLE post\_likes(

post\_like\_id int PRIMARY KEY AUTO\_INCREMENT,

post\_id int REFERENCES post(post\_id) ON DELETE CASCADE,

page\_id int REFERENCES page(page\_id) ON DELETE CASCADE,

liked\_by int REFERENCES profile(profile\_id) ON DELETE CASCADE

);

Transaction Code (According to requirements)

**3.1 The first transaction is to add information about a new PROFILE (ACCOUNT).**

INSERT INTO profile ( fname, lname, email, pwd, username, mobile\_no, prof\_create\_date) VALUES ('".$firstName."','".$lastName."','".$email."','".$password."','".$userName."',".$number.",'".$creationDate."');

**3.2 The second transaction is to add all the information about a new PAGE.**

INSERT INTO page (page\_name, description, category, admin\_id) VALUES ("'.$pageName.'", "'.$pageDesc.'", "'.$pageCat.'", '.$\_SESSION['userid'].')

**3.3 The third transaction is to create a new POST (this must find an available account and page).**

INSERT INTO post (post\_date, content, post\_type, posted\_by, page\_id) VALUES (CURRENT\_TIMESTAMP, "'.$newPost.'", "text", '.$\_SESSION['userid'].', '.$pageId.')

**3.4 The fourth transaction is to show the details of a page (by PAGE\_ID).**

SELECT \* from page where page\_Id =".$pageId."

SELECT \* from profile where profile\_id =".$profID."

**3.5 The fifth transaction is to return a list of posts by specific account (AccountID) in different pages.**

SELECT \* FROM `post` WHERE posted\_by = 1

**3.6 The sixth transaction is to return the total posts/comments in of each page by all accounts(users) on specific date (e.g. PostDate)**

SELECT \* from post where page\_id = '.$pageID.' and post\_date like "%'.$date.'%"

**3.7 The seventh transaction is to update information of pages (e.g. update the Page name(s) or description of page(s))**

UPDATE `page` SET `page\_name`="'.$pageName.'", `description`="'.$pageDesc.'",`category`="'.$pageCat.'" WHERE `admin\_id`='.$\_SESSION['userid'].' and page\_id ='.$pageId.'

**3.8 The eighth transaction is to delete a PROFILE (What if a Profile is the Admin of at least one page?!)**

DELETE FROM `profile` WHERE profile\_id = ".$profile\_id."