Society Management System

1. Description of the project:

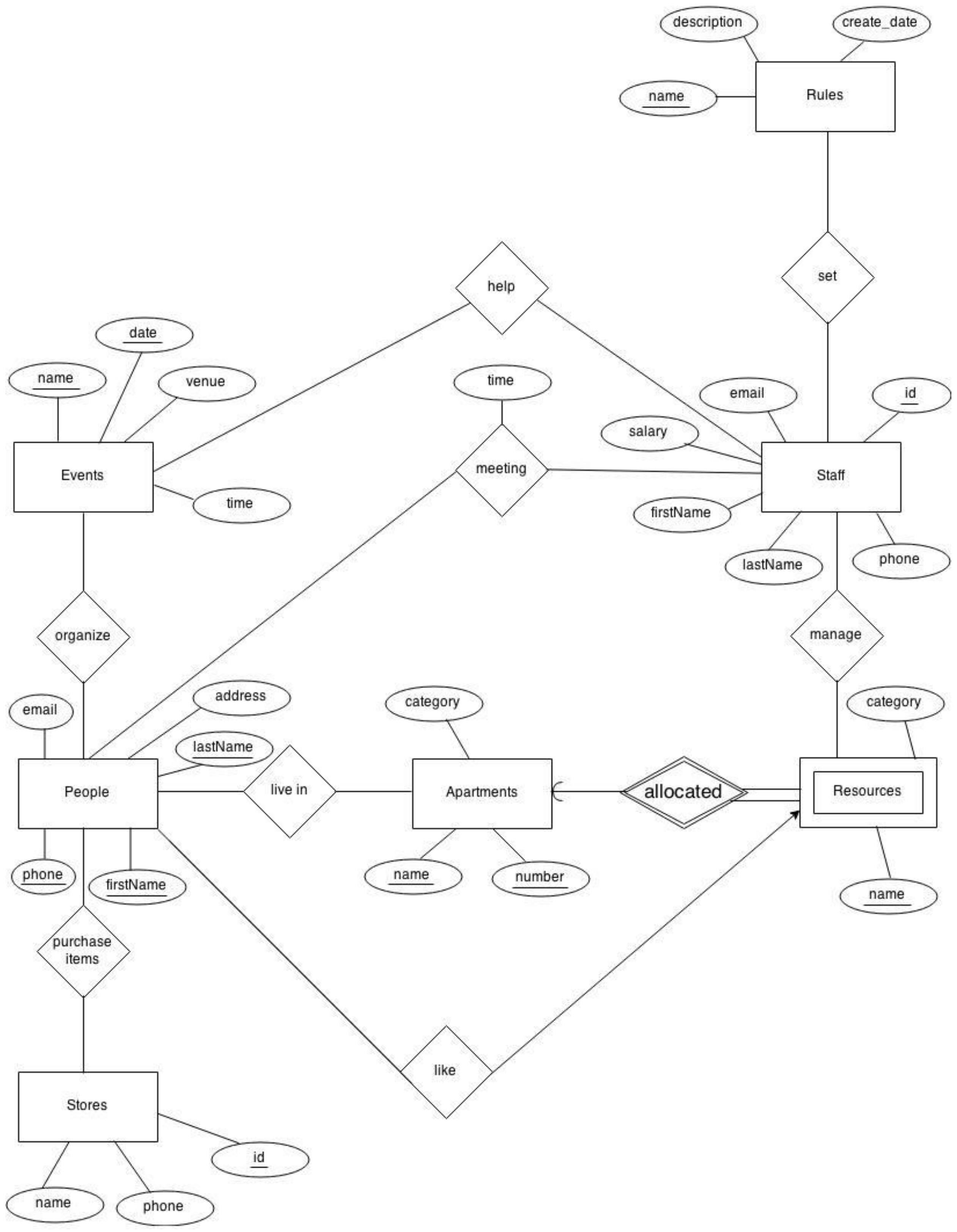
The "Society Management System" is a system in which it shows and takes care of each and every things which we see in our general society. People need to do various activities, internal as well as external. Different things; for example to organize events, have meetings, maintaining a convenient store, use and maintenance of utilities like parking area, swimming pool, gym, sports club and a garden. The system will show relationships between various entities with various attributes attached on each entity set.

The main purpose of this database design is to understand the functionality of large society and to understand how data flow from our end to another. The system will focus on people who live in society and with the various user cases scenario, it will become a database system. There may be the case, where one user involve in one entity and do some types of task then it may be effect on another cases. We will overcome all problems in different phases of project and will make efficiency system to access very large amount of data.

2. Main functionalities:

- ⇨ Manage user accounts.
- ⇨ Details of apartments with various resources management.
- ⇨ Events management.
- ⇨ Staff management.
- ⇨ Rules and regulations of society.
- ⇨ Access convenient store data.

3. Entity-Relationship diagram:



[Fig, E/R diagram]

4. Relational Schema derived from E-R model of the database:

- ⇨ People(<u>firstname</u>, <u>lastname</u>, <u>phone</u>, email, address)
- ⇨ Apartments(<u>name</u>, <u>number</u>, category)
- ⇨ Live_in(pe-firstname, pe-lastname, pe-phone, ap-name, ap-number)
- ⇨ Resources(<u>name</u>, category, <u>ap-name</u>, <u>ap-number</u>)
- ⇨ Staff(<u>id</u>, firstname, lastname, phone, email, salary)
- ⇨ Manage(re-name, ap-name, ap-number, st-id)
- ⇨ Rules(<u>name</u>, description, create_date)
- ⇨ Set(st-id, ru-name)
- ⇨ Events(<u>name</u>, <u>date</u>, venue, time)
- ⇨ Organize(ev-name, ev-date, pe-firstname, pe-lastname, pe-phone)
- ⇨ Stores(<u>id</u>, name, phone)
- ⇨ Purchase_items(sto-id, pe-firstname, pe-lastname, pe-phone)
- ⇨ Likes(pe-firstname, pe-lastname, pe-phone, re-name, ap-name, ap-number)
- ⇨ Meeting(pe-firstname, pe-lastname, pe-phone, st-id, time)
- ⇨ Help(ev-name, ev-date,st-id)

5. Database design:

- ⇨ People table:

  CREATE TABLE people(firstname varchar(20), lastname varchar(20), address varchar(30), phone varchar(15), email varchar(30) PRIMARY KEY(firstname, lastname, phone));

- ⇨ Apartments table:

  CREATE TABLE apartments(name varchar(20), number varchar(15), category varchar(20) PRIMARY KEY(name, number));

- ⇨ Live_in table:

  CREATE TABLE live_in(

  pe-firstname varchar(20) REFERENCES people(firstname) ON DELETE SET NULL, ON UPDATE CASCADE, pe-lastname varchar(20) REFERENCES people(lastname) ON DELETE SET NULL, ON UPDATE CASCADE,

pe-phone varchar(15) REFERENCES people(phone) ON DELETE SET NULL, ON UPDATE CASCADE,

ap-name varchar(20) REFERENCES apartments(name) ON DELETE SET NULL, ON UPDATE CASCADE,

ap-number varchar(15) REFERENCES apartments(number) ON DELETE SET NULL, ON UPDATE CASCADE);

- ⇨ Resources table:

  CREATE TABLE resources(name varchar(20), category varchar(20),

  ap-name varchar(20) REFERENCES apartments(name) ON DELETE SET NULL, ON UPDATE CASCADE,

  ap-number varchar(15) REFERENCES apartments (number) ON DELETE SET NULL, ON UPDATE CASCADE,

  PRIMARY KEY(name, ap-name, ap-number));

- ⇨ Staff table:

  CREATE TABLE staff(id varchar(20) PRIMARY KEY, firstname varchar(20), lastname varchar(20), phone varchar(15), email varchar(30), salary DECIMAL(10,2));

- ⇨ Manage table:

  CREATE TABLE manage(

  re-name varchar(20) REFERENCES resources(name) ON DELETE SET NULL, ON UPDATE CASCADE,

  ap-name varchar(20) REFERENCES apartments(name) ON DELETE SET NULL, ON UPDATE CASCADE,

  ap-number varchar(15) REFERENCES apartments(number) ON DELETE SET NULL, ON UPDATE CASCADE,

  st-id varchar(20) REFERENCES staff(id) ON DELETE SET NULL, ON UPDATE CASCADE);

- Rules table:

  CREATE TABLE rules(name varchar(20) PRIMARY KEY, description varchar(50), create_date date);

- Set table:

  CREATE TABLE set(

  st-id varchar(20) REFERENCES staff(id) ON DELETE SET NULL, ON UPDATE CASCADE,

  ru-name varchar(20) REFERENCES rules(name) ON DELETE SET NULL, ON UPDATE CASCADE);

- Events table

  CREATE TABLE events(name varchar(20), date date, venue varchar(20), time time, PRIMARY KEY(name, date));

- Organize table:

  CREATE TABLE organize(

  ev-name varchar(20) REFERENCES events(name) ON DELETE SET NULL, ON UPDATE CASCADE,

  ev-date date REFERENCES events(date) ON DELETE SET NULL, ON UPDATE CASCADE,

  pe-firstname varchar(20) REFERENCES people(firstname) ON DELETE SET NULL, ON UPDATE CASCADE,

  pe-lastname varchar(20) REFERENCES people(lastname) ON DELETE SET NULL, ON UPDATE CASCADE,

  pe-phone varchar(15) REFERENCES people(phone) ON DELETE SET NULL, ON UPDATE CASCADE);

- Store table:

  CREATE TABLE store(id varchar(20) PRIMARY KEY, name varchar(20), phone);

⇨ Purchase_items table

CREATE TABLE purchase_item(sto-id varchar(20) REFERENCES store(id) ON DELETE SET NULL, ON UPDATE CASCADE,
pe-firstname varchar(20) REFERENCES people(firstname) ON DELETE SET NULL, ON UPDATE CASCADE,
pe-lastname varchar(20) REFERENCES people(lastname) ON DELETE SET NULL, ON UPDATE CASCADE,
pe-phone varchar(15) REFERENCES people(phone) ON DELETE SET NULL, ON UPDATE CASCADE);

⇨ Likes table:

CREATE TABLE likes(pe-firstname varchar(20) REFERENCES people(firstname) ON DELETE SET NULL, ON UPDATE CASCADE,
pe-lastname varchar(20) REFERENCES people(lastname) ON DELETE SET NULL, ON UPDATE CASCADE,
pe-phone varchar(15) REFERENCES people(phone) ON DELETE SET NULL, ON UPDATE CASCADE,
re-name varchar(20) REFERENCES resources(name) ON DELETE SET NULL, ON UPDATE CASCADE,
ap-name varchar(20) REFERENCES apartments(name) ON DELETE SET NULL, ON UPDATE CASCADE,
ap-number varchar(15) REFERENCES apartments(number) ON DELETE SET NULL, ON UPDATE CASCADE);

⇨ Meeting table:

CREATE TABLE meeting(pe-firstname varchar(20) REFERENCES people(firstname) ON DELETE SET NULL, ON UPDATE CASCADE,
pe-lastname varchar(20) REFERENCES people(lastname) ON DELETE SET NULL, ON UPDATE CASCADE,

pe-phone varchar(15) REFERENCES people(phone) ON DELETE SET NULL, ON UPDATE CASCADE,
st-id varchar(20) REFERENCES staff(id) ON DELETE SET NULL, ON UPDATE CASCADE,
time time);

⇨ Help table:
CREATE TABLE help(ev-name varchar(20) REFERENCES events(name) ON DELETE SET NULL, ON UPDATE CASCADE,
ev-date date REFERENCES events(date) ON DELETE SET NULL, ON UPDATE CASCADE,
st-id varchar(20) REFERENCES staff(id) ON DELETE SET NULL, ON UPDATE CASCADE);

6. Examples of SQL queries:

❖ Simple queries:

⇨ **1.** Find out all the first and last names of people who likes gym.
Query: SELECT pe_firstname, pe_lastname FROM likes WHERE re_name = 'gym';

⇨ **2.** Find all the details of staff members who's salary is above 3000;
Query: SELECT * FROM staff WHERE salary > 3000;

⇨ **3.** Find the name of apartments who comes under penthose category.
Query: SELECT name FROM apartments WHERE category = 'penthouse';

⇨ **4.** Find the events names organize by archer(last name of people).
Query: SELECT e_name FROM organize WHERE pe_lastname = 'archer';

⇨ **5.** List out all the names of store.

Query: SELECT name FROM store;

- **6.** Find the people's first name and last name whos first name ends with "n".
  Query: SELECT firstname, lastname FROM people WHERE firstname like '%n';

- **7.** Find the number of people who lives in park apartment.
  Query: SELECT count(ap_name) FROM live WHERE ap_name = 'park';

- **8.** Select the highest salary among all the staff members.
  Query: SELECT max(salary) FROM staff;

❖ **Multi-relational queries:**

- **9.** Find out the addresses of those people's who organize "yoga" event.
  Query: SELECT address FROM people, organize WHERE e_name = 'yoga' AND pe_firstname = firstname;

- **10.** Find the people's first name who likes gym or garden.
  Query: SELECT pe_firstname FROM likes WHERE re_name = 'gym' OR re_name = 'garden';

- **11.** Find staff members first name, last name and salary who has salary more than 2000 and set rules for the society.
  Query: SELECT firstname, lastname, salary FROM staff, set WHERE salary > 2000 AND id = sta_id;

- **12.** Find all the people who live in park 1 apartment.
  Query: SELECT * FROM live WHERE ap_name = 'park' AND ap_number = '1';

- **13.** Find out the first name and email of person who organize event on 23rd July.

Query: SELECT firstname, emailid FROM people, organize WHERE pe_firstname = firstname AND e_date = '2015-07-23';

⇨ **14.** Find out all the details of people who's firstname and lastname is not containing "a" in their names.
Query: SELECT * FROM people WHERE firstname NOT like '%a%' AND lastname NOT like '%a%';

❖ **Sub-queries:**

⇨ **15.** Find staff ids and staff phone numbers who set rules for organization; with the help of subquery IN.
Query: SELECT id, email FROM staff WHERE id IN (SELECT sta_id FROM set WHERE id = sta_id);

⇨ **16.** Select the first name and salary of the staff member who's salary is hightest from average salary; with the help of subquery ALL.
Query: SELECT firstname, salary FROM staff where salary >= ALL(select avg(salary) from staff);

⇨ **17.** Find first name, phone and email id for those people who like gym; with the help of subquery EXISTS.
Query: SELECT firstname, phone, emailid FROM people WHERE EXISTS(select * from likes WHERE pe_firstname = firstname AND re_name = 'gym');

⇨ **18.** Select all the staff's data except those person who has minimum salary with the use of ANY.
Query: SELECT * FROM staff where salary > ANY(select salary from staff);

⇨ **19.** Find the number of apartments that are come in penthouse category with the use of GROUP BY and HAVING and AGGREGATE function's subquery.

Query: SELECT count(*) FROM apartments GROUP BY category HAVING category = (SELECT category FROM apartments WHERE number = 1 AND name = 'creek');

⇨ **20.** Find the emaild of staff who are not in managing the resources with the use of NOT IN subquery.
Query: SELECT email FROM staff WHERE id NOT IN (SELECT sta_id FROM manage);

❖ **Examples of union, intersect and difference (except):**

⇨ **21.** Find the first name of people who live in "Creek" apartment and like gym; with the help of INTERSECT.
Query: (SELECT pe_firstname FROM live WHERE ap_name = 'creek') INTERSECT (SELECT pe_firstname FROM likes WHERE re_name = 'gym');

⇨ **22.** Find first name and last name of those staff members who set rules for oraganization OR also help in events; with the help of UNION.
Query: (SELECT firstname, lastname FROM staff, set WHERE sta_id = id) UNION (SELECT firstname, lastname FROM staff, help WHERE sta_id = id);

⇨ **23.** Find the salary of those staff members who set rules for oraganization but not involved in events with the help of EXCEPT.
Query: (SELECT salary FROM staff, set WHERE sta_id = id) EXCEPT (SELECT salary FROM staff, help WHERE sta_id = id);

⇨ **24.** Find last name and phone number of those people who likes "gym" OR they organize "yoga" event with the help of UNION subquery.
Query: SELECT lastname, phone FROM people, likes WHERE lastname = pe_lastname AND re_name = 'gym') UNION

SELECT lastname, phone FROM people, organize WHERE lastname = pe_lastname AND e_name = 'yoga');

- ⇨ **25.** Find apartment names who has "gym" and they come under "fitness" category with the help of INTERSECT subquery.
  Query: SELECT ap_name FROM resources WHERE name = 'gym') INTERSECT SELECT ap_name FROM resources WHERE category = 'fitness');

❖ **Join queries:**

- ⇨ **26.** Select all the data of people and their resources with the use of CROSS JOIN.
  Query: SELECT * FROM people CROSS JOIN resources;

- ⇨ **27.** Find the firstname, lastname of those staff members who maintanin "ground" resource with the help of INNER JOIN.
  Query: SELECT firstname, lastname, re_name FROM staff INNER JOIN manage ON id = sta_id WHERE re_name = 'ground';

- ⇨ **28.** Find all the data of staff members whose salary is more than the minimum salary of all the staff members with the use of THETA JOIN.
  Query: SELECT * FROM staff WHERE salary > (SELECT min(salary) from staff);

- ⇨ **29.** Find all the data from staff and resources with the help of CROSS JOIN.
  Query: SELECT * FROM staff CROSS JOIN manage;

- ⇨ **30.** Find all the staff data whose salary is greater than the average salary with the help of THETA JOIN.
  Query: SELECT * FROM staff WHERE salary > (SELECT AVG(salary) FROM staff);

- ⇨ **31.** Find the people's firstname and apartment's name with the help of LEFT JOIN.

Query: SELECT firstname, ap_name FROM people LEFT JOIN live ON lastname = pe_lastname;

- **32.** Find out all the resources with their respective management staff members with the help of RIGHT JOIN.
  Query: SELECT re_name, firstname, lastname FROM manage RIGHT JOIN staff ON id = sta_id;

- **33.** Find the people's and staff's firstname, lastname and phone with the help of FULL OUTER JOIN.
  Query: SELECT pe_firstname, pe_lastname, pe_phone, firstname, lastname, phone from meeting FULL OUTER JOIN staff ON 'sta_id' = 'id';

- **34.** Find the people who are involved in organize events with people's firstname, lastname, event name and event venue with the help of LEFT JOIN.
  Query: SELECT e_name, pe_firstname, pe_lastname, venue FROM organize LEFT JOIN events ON e_name = name;

- **35.** Find people's firstname, lastname, phone, email with their relative apartment name and apartment number with the help of RIGHT JOIN.
  Query: SELECT firstname, lastname, phone, email, ap_name, ap_number FROM people RIGHT OUTER JOIN live ON firstname = pe_firstname;

- **Aggregate Functions:**

- **36.** Find the second smallest salary data from staff with the use of MIN aggregate function.
  Query: SELECT MIN(salary) FROM staff WHERE salary > (SELECT MIN(salary) FROM staff);

- ⇨ **37.** Count the number of people who live in "park" apartment with the use of COUNT, GROUP BY and HAVING.
  Query: SELECT COUNT(*) FR OM live group by ap_name having ap_name = 'park';

- ⇨ **38.** Find the maximum salary among all the staff members with the use of MAX aggregate function.
  Query: SELECT MAX(salary) FROM staff;

- ⇨ **39.** Find those staff's data whose salary is above than the average salary of all the staff members with the use of AVG aggregate function.
  Query: SELECT * FROM staff WHERE salary > (SELECT AVG(salary) from staff);

- ⇨ **40.** Find the number of rules that are created for society's people with the use of COUNT aggregate function.
  Query: SELECT COUNT(*) FROM rules;

❖ **Modification queries:**

- ⇨ **41.** Insert one row in manage table.
  Query: INSERT INTO manage (sta_id, re_name, ap_name, ap_number) VALUES (2, 'garden', 'creek', 2);

- ⇨ **42.** Insert one row in manage table but insert staff id with the use of subquery.
  Query: INSERT INTO manage (sta_id, re_name, ap_name, ap_number) VALUES ((SELECT id FROM staff WHERE firstname = 'steeve'), 'garden', 'creek', 2);

- ⇨ **43.** Delete one row from purchase table where firstname is "adam".
  Query: DELETE FROM purchase WHERE pe_firstname = 'adam';

- ⇨ **44.** Delete all the data from manage table where id is "1".
  Query: DELETE FROM manage WHERE sta_id = 1;

- ⇨ **45.** Update adam's emil id "yrxs@gmail.com" to "adam@yahoo.com".
  Query: UPDATE people SET emailid = 'adam@yahoo.com' where firstname = 'adam';

- ⇨ **46.** Yoga event will start on evening 5 O'clock. Update the time in table.
  Query: UPDATE events set time = '17:00:00' where name = 'yoga';

❖ **View query:**

- ⇨ **47.** Create a view for staff data which includes firstname, lastname and email.
  Query: CREATE VIEW "Staff Data" AS SELECT firstname, lastname, email FROM staff;

❖ **PSM(Persistent Stored Modules) in SQL:**

- ⇨ **48.** Create one trigger which will save the staff table's data after insertion data into it and save into the another table.
  Solution:
- ⇨ Insert staff Table:
  CREATE TABLE staff(id number(10), firstname varchar(25), lastname varchar(25), phone number(20), email varchar(20), salary decimal(10,2));

- ⇨ Insert staff_backup Table:
  CREATE TABLE staff_backup(id number(10), firstname varchar(25), lastname varchar(25), phone number(20), email varchar(20), salary decimal(10,2));

- ⇨ Create Trigger:

```
CREATE or REPLACE TRIGGER staff_after_insert_trigger AFTER INSERT ON
staff
FOR EACH ROW
DECLARE
BEGIN
INSERT INTO staff_backup VALUES (:new.id, :new.firstname, :new.lastname,
:new.phone, :new.email, :new.salary);
END;
```

⇨ Insert Data:

```
INSERT INTO staff VALUES(1, 'jeff', 'chris', 870123, 'jeff@gmail.com', 1200);
```

❖ **Procedures in SQL:**

⇨ **49.** Create one procedure for insertion the data into the table and store in database. Then call that procedure to insert the data. (PSM)

Solution:

```
CREATE OR REPLACE PROCEDURE insert_staff(a IN number, b IN varchar, c IN
varchar, d IN number, e IN varchar, f IN decimal)
AS
BEGIN
INSERT INTO staff VALUES(a,b,c,d,e,f);
dbms_output.put_line('Successful! Row added in staff table.');
END;
/
```

⇨ Execute/Call Procedure:

```
execute insert_staff(1,'Jeff', 'Hay', 870321, 'jeff@gmail.com', 1000);
```

❖ **Constraints in SQL:**

⇨ **50.** CREATE TABLE and add constraint primary key.

Query: CREATE TABLE staff (id varchar(5), firstname varchar(20), lastname varchar(20), phone varchar(20), email varchar(30), salary decimal(10,2), primary key(id));

⇨ **51.** CREATE TABLE and add foreign key constraint.

Query: CREATE TABLE manage (sta_id varchar(5), re_name varchar(20), ap_name varchar(20), ap_number varchar(10), foreign key(sta_id) references staff(id));

⇨ **52.** CREATE TABLE with attribute constraint. Check the condition for phone number, it must be start from 870.

Query: CREATE TABLE store(id varchar(5), name varchar(20), phone varchar(20) CHECK (phone like '870%'));

⇨ **53.** Modify staff table to add tuple constraint of email and salary. In email address "@" must be there and staff salary must be more than 1000.00.

Query: ALTER TABLE staff ADD constraint emailSalary check (email like '%@%' AND salary > 1000.00);

7. Normalization:

Database **normalization** is the process of organizing the attributes and tables of a relational database to minimize data redundancy.
[source: http://en.wikipedia.org/wiki/Database_normalization]

❖ Example:

⇨ On "Rules" table, we have relation R(name, description, create_date);

Here FD is name $\rightarrow$ create_date

To find super keys and keys, first we have to find closures:
name$^+$ = {name, create_date}
description$^+$ = {lastname}

create_date$^+$ = {create_date}

nameDescription$^+$ = {name, create_date, description} *** super key and key**

nameCreate_date$^+$ = {name, create_date}

description_Create_date$^+$ = {description, create_date}

- ⇨ Super key is: nameDescription.
- ⇨ Key is: nameDescription.

❖ **Check whether the FD is in BCNF(Boyce–Codd Normal Form) and 3NF:**

- ⇨ Check BCNF:

  FD's name $\rightarrow$ create_date.

  Here, left hand side of FD "name" is not a super key. So, it violates the BCNF.

- ⇨ Check 3NF:

  FD's name $\rightarrow$ create_date.

  Here, left hand side of FD "name" is not a super key and right hand side "create_date" is not a prime. Hence, it violates the 3NF.

❖ **Decompose the relation according to the BCNF decomposition:**

- ⇨ Name $\rightarrow$ create_date

  x = name;
  R1 = x$^+$ = {name, create_date};
  R2 = (R-X$^+$) U X = {name, description);

- ⇨ Final, decompose relations are R1(name, create_date), R2(name, description).

8. PHP programming code with SQL:

❖ **connection string:**

```
<?php

        $hostname = "localhost";       //host name
        $dbname = "adil.memon";     //db name
        $username = "adil.memon";     //user name
        $password = "adil_db";             //user password

                Try {
                        $conn = new PDO("pgsql:host=$hostname; dbname = $dbname",
                        $username, $password);
                }
                catch (PDOException $e) {
                        echo "Database connection failed!";
                        exit(1);
                }
?>
```

❖ **Fetch data from database with the help of SQL query and display the data in table:**

```
<?php
        $query = "Select * from people";
        $sqlquery = $conn->prepare($query, array(PDO::ATTR_CURSOR =>
        PDO::CURSOR_FWDONLY));
        $sqlquery -> execute();
?>
```

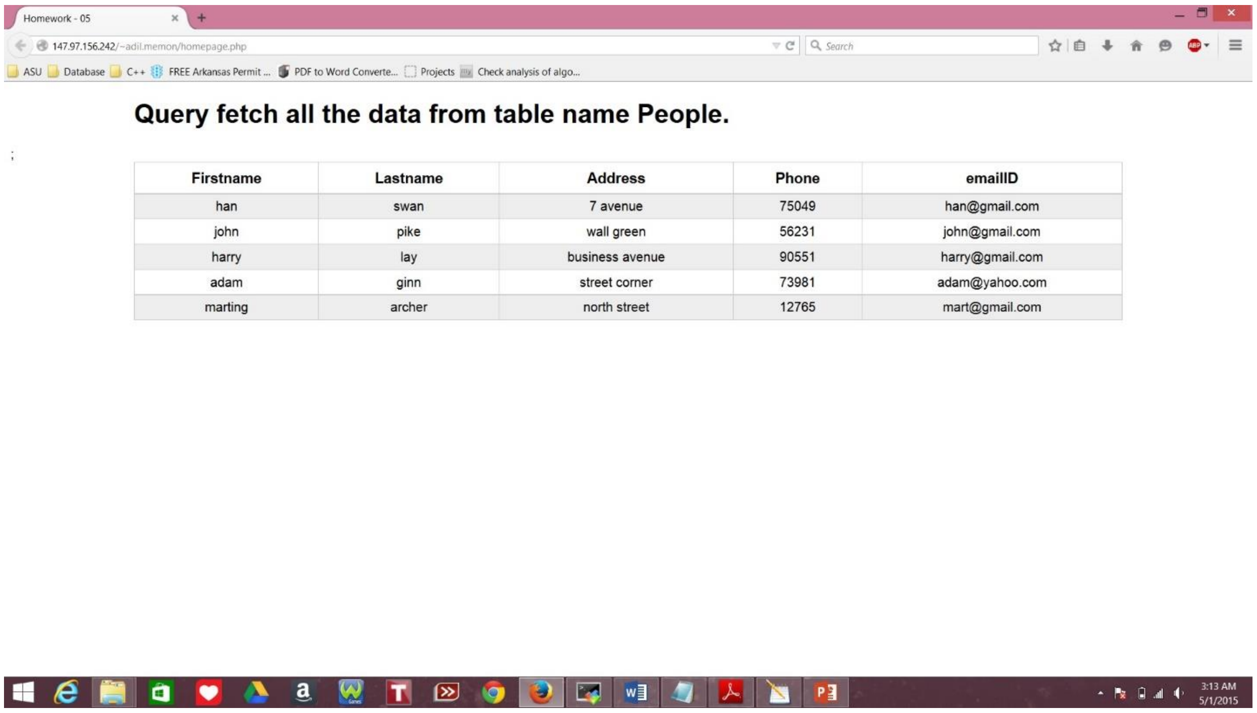
```
<h1>Query fetch all the data from table name People.</h1>;

<table> <!-- table starts -->
        <tr>
                <th> Firstname </th>
                <th> Lastname </th>
                <th> Address </th>
                <th> Phone </th>
                <th> emailID </th>
        </tr>

<?php
while($row = $sqlquery->fetch(PDO::FETCH_ASSOC))
{
        echo "<tr><td>".$row['firstname']."</td>";
        echo "<td>".$row['lastname']."</td>";
        echo "<td>".$row['address']."</td>";
        echo "<td>".$row['phone']."</td>";
        echo "<td>".$row['emailid']."</td></tr>";
}
?>

</table> <!-- table ends -->
```

9. Screenshot of webpage after fetching data with the help of SQL and display in table:


Homework - 05

147.97.156.242/~adil.memon/homepage.php

Query fetch all the data from table name People.

Firstname	Lastname	Address	Phone	emailID
han	swan	7 avenue	75049	han@gmail.com
john	pike	wall green	56231	john@gmail.com
harry	lay	business avenue	90551	harry@gmail.com
adam	ginn	street corner	73981	adam@yahoo.com
marting	archer	north street	12765	mart@gmail.com