Course Number and Name: CSE 4308 Database Management Systems Lab	
Student Name: Nafisa Maliyat	**Student ID:** 200042133
Report Submission Date: 23 October, 2022	**Name of Lab Instructor:** Md. Bakhtiar Hasan, Lecturer, CSE Zannatun Naim Sristy, Lecturer, CSE

Lab 7: Entity Relationship (ER) Data Model II

Overview:

This lab provided us with a given scenario and system requirements had to be extracted. A ER-Diagram was created based on the requirements and consecutive DDL statements had to be written based on the diagram. Additionally, queries had to be written based on the information required by the problem statement.

On the next pages, I have mentioned the following :

- the scenario,
- the ER-Diagram of the given scenario
- part by part problem analysis,
- SQL statements written based on the queries given in the problem statements,
- problems faced (if any) during solution of the tasks,
- their results on the SQL command line.

Scenario:

National ID (NID) is an integrated collection of citizens' information such as Name, Date of Birth, Occupation, Blood Group. Each citizen has his/her own NID. In order to investigate the population density, the country has been divided into divisions. Each division has its name, size (in square KM), and a brief description. Again, each division has a number of districts with similar attributes. Citizen information must be connected to its corresponding division and district.

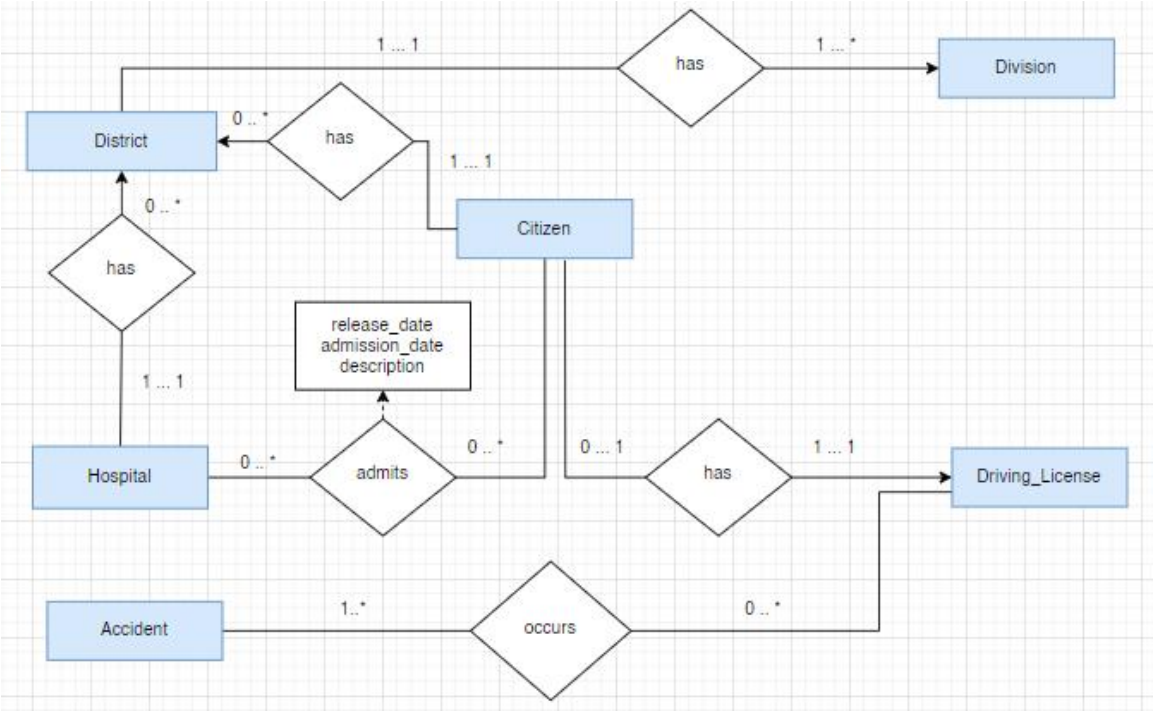
Each citizen may have exactly one driving license where information such as type of license, issue date, expiration date are maintained. Whenever any accident occurs, it is logged in the central system. The system stores relevant information such as date and time of accident, location of accident, number of deaths (if any), etc.

There are a number of hospitals in the country having name and contact information. Each hospital may have more than one contact number. Citizens may avail treatment in any hospitals they prefer. Whenever any patient (i.e., citizen) is admitted, the system keeps the record of his/here date of admission, a brief description, and release date.

Task-1:

Problem Statement:

Draw an ER Diagram, without any data redundancy, specifying the cardinality explicitly. You may add additional attributes only if it is needed.



Task 2:

Part 1:

National ID (NID) is an integrated collection of citizens' information such as Name, Date of Birth, Occupation, Blood Group. Each citizen has his/her own NID. In order to investigate the population density, the country has been divided into divisions. Each division has its name, size (in square KM), and a brief description. Again, each division has a number of districts with similar attributes. Citizen information must be connected to its corresponding division and district.

Analysis of the problem:

This requires a citizen table that has a primary attribute NID and other attributes name, date of birth, occupation and blood group.
Another table division has attributes division name, size and description. Similarly district table also has attributes district name, size and description.
The citizen table will also have division name. Here the division table, district table and citizen table will have to be created in this order.

Any problems faced and how it was solved:

There were no problems faced since the query was straightforward and simple.

Query:

```
create table division
(
    name varchar2(30),
    description varchar2(50),
    constraint pk_division primary key(name)
);
```

```
create table district
(
    name varchar2(30),
    description varchar2(50),
    division_name varchar2(20),
    constraint pk_district primary key(name),
    constraint fk_district_division foreign key(division_name) references
division(name)
);
```

```
create table citizen
(
    NID varchar2(13),
    name varchar2(30),
    date_of_birth date,
    occupation varchar2(20),
    blood_group varchar2(5),
    district_name varchar2(20),
    division_name varchar2(20),
    constraint pk_citizen primary key(NID),
    constraint fk_citizen_district foreign key(district_name) references
district(name)
);
```

```
Table created.
Table created.
Table created.
```

Part 2:

Each citizen may have exactly one driving license where information such as type of license, issue date, expiration date are maintained. Whenever any accident occurs, it is logged in the central system. The system stores relevant information such as date and time of accident, location of accident, number of deaths (if any), etc.

Analysis of the problem:

A table called driving license will have to have attribute driving license ID which will be the primary key and other attributes type of license, issue date and expiration date.
Another table accident will have attribute accident ID as primary key and attributes date, location and description of accident and number of deaths.
Since citizen may have exactly one driving license, there will be a one to one relationship between citizen and driving license table. However, multiple drivers could be involved in one accident and multiple accidents might involve one driver thus there is a need for a junction table 'occurs'.

Any problems faced and how it was solved:

The number of deaths in accident could have been calculated in a view versus storing it in a table as an attribute but since number of deaths in an accident after it has been stored in system is most likely not to change, it was added as an attribute.
Since the SQL command line showed error whenever time or datetime was used, the attribute time of accident could not be implemented. This problem was not solved.

Results:

```
create table driving_license
(
    id varchar2(20),
    type_of_license varchar2(20),
    issue_date date,
    expire_date date,
    NID varchar2(13),
    constraint pk_driving_license primary key(id),
    constraint fk_driving_license_citizen foreign key(NID) references
citizen(NID)
```

```
);
```

```
create table accident
(
    id varchar2(20),
    date_of_accident date,
    location varchar2(20),
    description varchar2(50),
    number_of_deaths int,
    constraint pk_accident primary key(id)
);
```

```
create table occurs
(
    accident_id varchar2(20),
    driving_license_id varchar2(20),
    constraint fk_accidents_driving_license foreign key(accident_id)
references accident(id),
    constraint fk_accidents_accident foreign key(driving_license_id)
references driving_license(id)
);
```

```
Table created.
Table created.
Table created.
```

Part 3:

There are a number of hospitals in the country having name and contact information. Each hospital may have more than one contact number. Citizens may avail treatment in any hospitals they prefer. Whenever any patient is admitted, the system keeps the record of his/here date of admission, a brief description, and release date.

Analysis of the problem:

A new table for hospital should store name and contact information such as phone number and address. Hospital name is the primary key and hospital table holds name of the district it is in.
Citizen table should be connected to hospital table since citizens get admitted to hospitals. A junction table is needed to connect and additionally, date of admission, description and release date should also be stored as extra attributes of that table.
Since hospitals can have multiple numbers, varray of phone numbers is declared where it is assumed that hospitals do have more than 5 phone numbers (as is usually the case).

Any problems faced and how it was solved:

The first line did not compile in SQL and thus the rest of the tables did not run. Thus a semi solution was to assume a single phone number for hospital.

Results:

```
create or replace type vmobiles as varray(5) of varchar2(20);
```

(does not run)

```
create table hospital
(
    name varchar2(30),
    phone_number varchar2(20),
    address varchar2(30),
    district_name varchar2(20),
    constraint pk_hospital primary key(name),
    constraint fk_hospital_district foreign key(district_name) references
district(name)
);
```

```
create table admits
(
    hospital_name varchar2(20),
```

```
    NID varchar2(13),
    description varchar2(50),
    admission_date date,
    release_date date,
    constraint fk_admits_hospital foreign key(hospital_name) references
hospital(name),
    constraint fk_admits_citizen foreign key(NID) references citizen(NID)
);
```

```
Table created.
Table created.
```

Task 3:

(a):

Find the list of divisions along with its total number of districts.

Analysis of the problem:

This information can be found by selecting division name from districts table using group by on division name.

Any problems faced and how it was solved:

There were no problems faced since the query was simple and straightforward.

Query:

```
select division_name, count(name)
from district
group by division_name;
```

(b):

Find the list of districts having at least 20,000 people living there.

Analysis of the problem:

This information can be found by selecting district name from citizen table using group by on division name and including a having clause to impose condition of at least 20,000 people using count function.

Any problems faced and how it was solved:

There were no problems faced since the query was simple and straightforward.

Query:

```
select district_name
from citizen
group by district_name
having count(NID) >= 20000;
```

(c):

Find the number of accidents that involved a citizen whose NID is 210.

Analysis of the problem:

This requires selecting count of accident ID from occurs table where the corresponding driving license ID matches with the ID from driving license table having NID of 210.

Any problems faced and how it was solved:

There were no problems faced since the query was simple and straightforward.

Query:

```
select count(accident_id)
from occurs
where driving_license_id = (select id
                            from driving_license
                            where NID = 210);
```

(d):

Find the list of top 5 hospitals based on the number of patients admitted so far.

Analysis of the problem:

First a list of name of hospital was selected from admits by using group by on hospital_name attribute and next the list was sorted by the number of patients in descending order. Next the top 5 rows were selected.
Here it is assumed that 'based on the number of patients' means the hospitals with the most patients.

Any problems faced and how it was solved:

There were no problems faced since the query was simple and straightforward.

Query:

```
select hospital_name from
(select hospital_name
from admits
group by hospital_name
order by count(NID) desc)
where rownum=5;
```

(e):

Find the blood group of all the patients admitted to different hospitals.

Analysis of the problem:

The blood group is to be selected from citizen table where the NID is cross checked against NID stored in admits table to ensure the citizen was a patient.

Any problems faced and how it was solved:

There were no problems faced since the query was simple and straightforward.

Query:

```
select blood_group
from citizen
where NID in (select NID
              from admits);
```

(f):

Find the population density for each division.

Analysis of the problem:

A list of attributes in citizen and district tables was selected and grouped by the district and division name. Population density was calculated using number of people in each division and number of districts in each division.

Any problems faced and how it was solved:

There were confusions about how to calculate population density without area of a division. At last, it was assumed to be the number of people per district of a division.

Query:

```
select count(citizen.NID)/count(district.name)
from citizen, district
where citizen.district_name = district.name
group by district.division_name, district.name;
```

(g):

Find the top 3 densely populated districts.

Analysis of the problem:

Using group by on district name, order by on count of NID and rownum statements, the required information could be found.

Any problems faced and how it was solved:

There were no problems faced since the query was simple and straightforward.

Query:

```
select district_name from
(select district_name
from citizen
group by district_name
order by count(NID) desc)
where rownum = 3;
```

(h):

Find the number of accidents that occurred in each district.

Analysis of the problem:

This requires a count of accident ID in each district. Since district is not connected to accident table, a series of tables had to be included as well to ensure the correct rows are being selected. The queries were grouped by district name before selecting the number of accident ID for the district.

Any problems faced and how it was solved:

Since the information required is in different tables, nested queries were getting complicated to keep track of. Thus all of these were combined in from and where clauses which made it easier to follow.

Query:

```
select count(accident.id)
from accident, occurs, driving_license, citizen
where accident.id = occurs.accident_id
    and occurs.driving_license_id = driving_license.id
    and driving_license.NID = citizen.NID
group by citizen.district_name;
```

(i):

Find the division where the least amount of accidents occurred.

Analysis of the problem:

First a list of divisions ordered by the number of accidents that occurred in the division. Next rownum was used to select the top row (i.e. the division with the least number of accidents).

Any problems faced and how it was solved:

There were no problems faced since the query was simple and straightforward.

Query:

```
select division_name
from
(
    select district.division_name
    from accident, occurs, driving_license, citizen, district
    where accident.id = occurs.accident_id
    and occurs.driving_license_id = driving_license.id
    and driving_license.NID = citizen.NID
    and citizen.district_name = district.name
    group by district.division_name
    order by count(accident.id)
)
where rownum = 1;
```

(j):

Find the number of accidents caused by 'non-professional' and 'professional' license holders.

Analysis of the problem:

The count of accident_id was selected from occurs where driving license is cross checked with driving_license where license type was 'non-professional' or 'professional'.

Any problems faced and how it was solved:

There were no problems faced since the query was simple and straightforward.

Query:

```
select count(accident_id)
from occurs
where driving_license_id in (select id
                             from driving_license
                             where type_of_license = 'non-professional' or
                             type_of_license = 'professional');
```

(k):

Find the person who was admitted to the hospital for the longest period of time.

Analysis of the problem:

From the admits table, the NID was selected and condition applied afterwards to check if the person with the NID had the longest period of admission in hospital.

Any problems faced and how it was solved:

There were no problems faced since the query was simple and straightforward.

Query:

```
select NID
from admits
group by NID
having NID in ( select NID from
                (select NID, max(release_date - admission_date)
                from admits
                group by NID));
```

(1):

Find the division where the number of young people (15 ⩽ age ⩽ 30) is the lowest.

Analysis of the problem:

First an ordered list of divisions and the number of young people living there are selected. Next, the top row is selected (since the order is ascending, the division with lowest population was selected.

Any problems faced and how it was solved:

There were no problems faced since the query was simple and straightforward.

Query:

```
select name from
(
    select division.name, count(citizen.NID) as population
    from citizen, district, division
    where citizen.district_name = district.name
        and district.division_name = division.name
        and (sysdate - citizen.date_of_birth) between 15 and 30
    group by division.name
    order by population
)
where rownum = 1;
```

(m):

Find the people whose licenses expired.

Analysis of the problem:

The list of citizens that have driving license and the expire date is less than current date should be selected.

Any problems faced and how it was solved:

There were no problems faced since the query was simple and straightforward.

Query:

```
select citizen.name
from citizen, driving_license
where citizen.NID = driving_license.NID
    and driving_license.expire_date < sysdate;
```

(n):

Find the number of accidents caused by people whose licenses expired.

Analysis of the problem:

The count of accidents of citizens having driving license whose expire date is less than current date is selected.

Any problems faced and how it was solved:

There were no problems faced since the query was simple and straightforward.

Query:

```
select count(occurs.accident_id)
from driving_license, occurs, accident
where driving_license.id = occurs.driving_license_id
    and accident.id = occurs.accident_id
    and driving_license.expire_date < sysdate;
```

(o):

Find the license holders who were not involved in any accident so far.

Analysis of the problem:

This requires license holders from driving_license table whose id did not appear in occurs table.

Any problems faced and how it was solved:

There were no problems faced since the query was simple and straightforward.

Query:

```
select id
from driving_license
where id not in (select occurs.driving_license_id
                                 from occurs);
```

(p):

Find the number of deaths due to any accident for each division.

Analysis of the problem:

A sum of the attribute number_of_deaths is selected from a list of the accidents in each division.

Any problems faced and how it was solved:

There were no problems faced since the query was simple and straightforward.

Query:

```
select sum(accident.number_of_deaths)
from accident, occurs, driving_license, citizen, district
where accident.id = occurs.accident_id
    and occurs.driving_license_id = driving_license.id
    and driving_license.NID = citizen.NID
    and citizen.district_name = district.name
group by district.division_name;
```

(q):

Find the name of the people who got their license before the age of 22 or after the age of 40.

Analysis of the problem:

This requires finding citizens who has a driving license and their age is within the limits mentioned.

Any problems faced and how it was solved:

There were no problems faced since the query was simple and straightforward.

Query:

```
select name
from citizen, driving_license
where citizen.NID = driving_license.NID
    and sysdate - citizen.date_of_birth < 22
    and sysdate - citizen.date_of_birth > 40;
```

(r):

Find the list of citizens who were admitted to the hospital on the same day they got into an accident.

Analysis of the problem:

This essentially asks for the citizen in admits table whose admission date matches the date of accident in occurs table.

Any problems faced and how it was solved:

There were no problems faced since the query was simple and straightforward.

Query:

```
select citizen.NID
from citizen, admits, occurs, driving_license, accident
where citizen.NID = admits.NID
    and admits.admission_date = accident.date_of_accident
    and accident.id = occurs.accident_id
    and occurs.driving_license_id = driving_license.id
    and driving_license.NID = citizen.NID;
```

(s):

Find the hospital where people from Dhaka division were admitted the most.

Analysis of the problem:

First the hospital name and NID was selected where the citizen with that NID lives in Dhaka division. Next, the group by is used on the hospital name and the number of patients (who lives in Dhaka division) is selected and the list is ordered by the number of these patients in descending order. Next the top row i.e. the maximum count is selected which gives the desired results.

Any problems faced and how it was solved:

There were no problems faced since the query was simple and straightforward.

Query:

```
select name from
(
    (select name, count(NID)
    from
    (
    select hospital.name, admits.NID
    from hospital, admits, citizen, district
    where hospital.name = admits.hospital_name
        and citizen.NID = admits.NID
        and citizen.district_name = district.name
        and district.division_name = 'Dhaka'
    )
    group by name
    order by count(NID) desc)
)
where rownum = 1;
```

(t):

Find the list of people who caused an accident outside their own district.

Analysis of the problem:

The list required is a combination of people who have been in accident and accident location is not the name of the district they live in.This requires information from a combination of different tables.

Any problems faced and how it was solved:

There were no problems faced since the query was simple and straightforward.

Query:

```
select accident.location, driving_license.NID
from driving_license, occurs, accident, district, citizen
where driving_license.id = occurs.driving_license_id
    and accident.id = occurs.accident_id
    and district.name = citizen.district_name
    and district.name != location
    and citizen.NID = driving_license.NID;
```