Islamic University of Technology (IUT)

CSE 4308: Database Management Systems Lab

Lab Report # 7

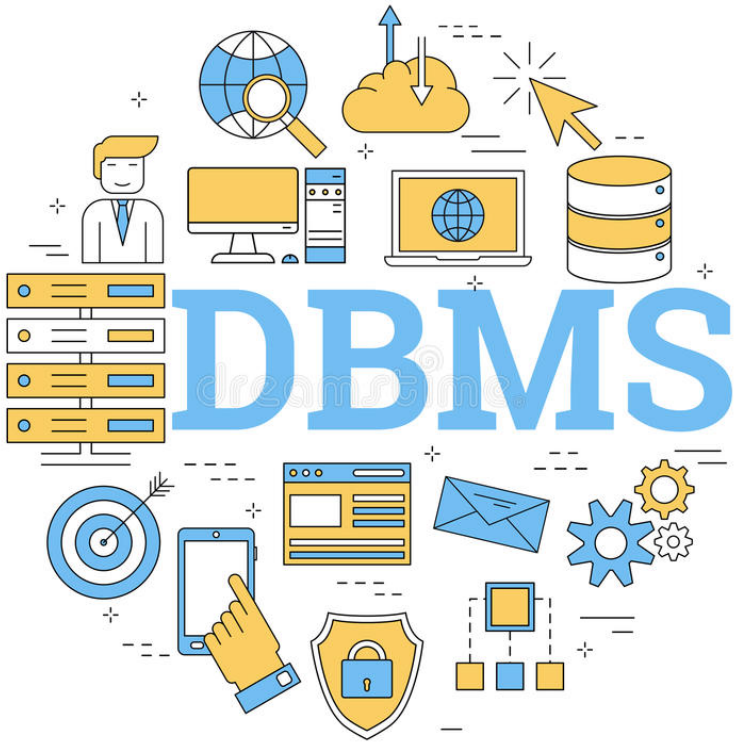
Submitted to:

Md. Bakhtiar Hasan,

Asst. Professor, CSE.

Zannatun Naim Srsity,

Lecturer, CSE.



Submitted by:

M M Nazmul Hossain
ID 200042118
CSE (SWE)

Submission Date:

26.10.2022

Introduction

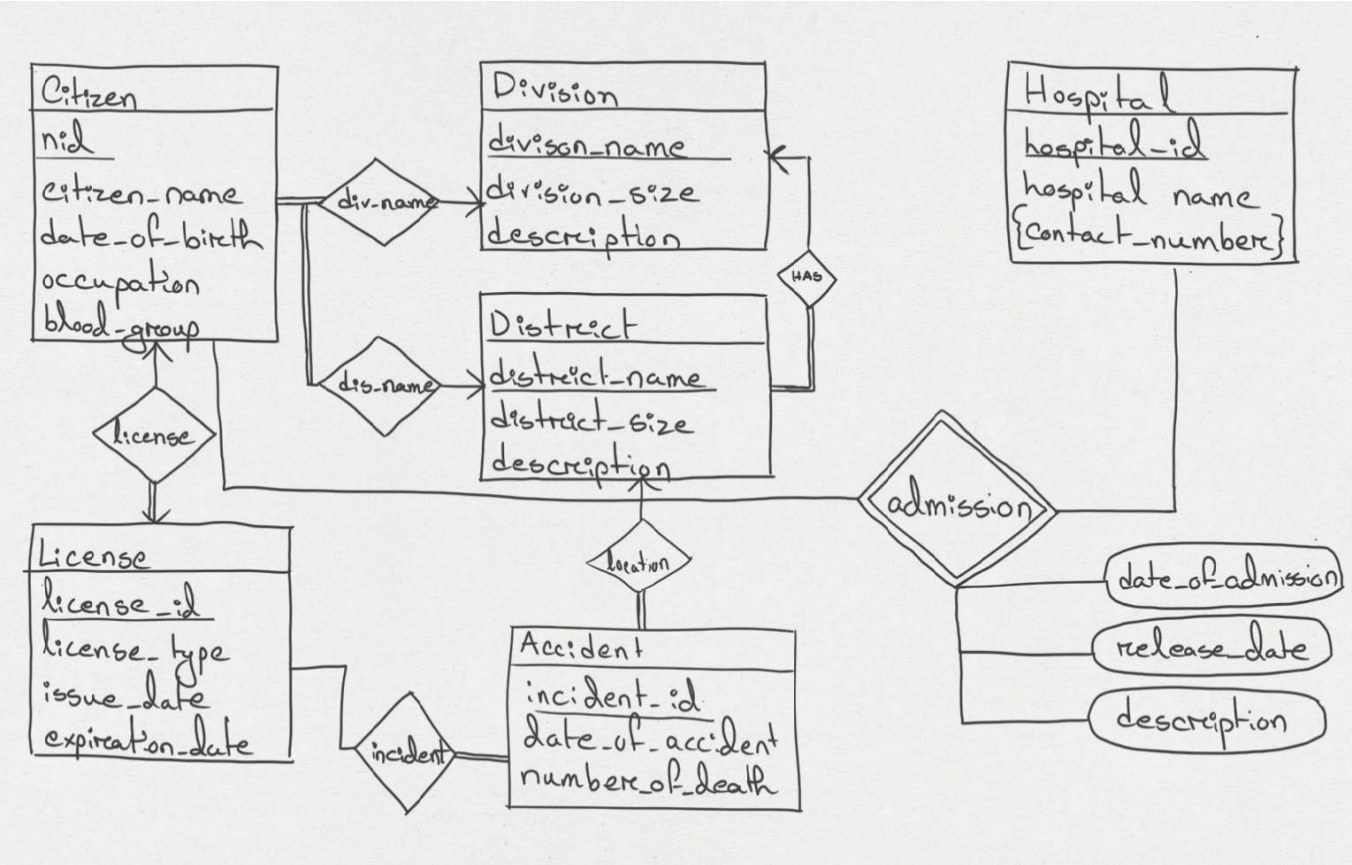
The seventh Database Management Systems Lab was about Entity Relationship Diagram, Data Definition Language, and subsequent queries on the created tables. Entity Relationship allows proper organization of Entities and helps avoid data redundancy.

Method

Analysis of the problem of Task 1

The lab task provided a scenario where the relations between a government system were described. Entities had to be identified from that brief description and a complete Entity Relationship Diagram had to be drawn which kept data redundancy at a minimum.

To implement that, the following Entity Relationship Diagram involving the entities: citizen, division, district, license, accident, and hospital, was drawn:



Analysis of the problem of Task 2

The following task was to covert the previously implemented Entity Relationship Diagram and Define the Schema through Database Definition Language. The following working code is the submitted solution:

```
-- DDL Implementation
drop table division cascade constraints;
drop table district cascade constraints;
drop table citizen cascade constraints;
drop table license cascade constraints;
drop table accident cascade constraints;
drop table incident cascade constraints;
drop table hospital cascade constraints;
drop table admission cascade constraints;
drop type vmobiles;

Create Table division (
    division_name varchar2(20),
    division_size int,
    description varchar2(20),
    primary key (division_name)
);

Create Table district(
    district_name varchar2(20),
    district_size int,
    description varchar2(20),
    division_name varchar2(20) not null,
    primary key (district_name),
    foreign key (division_name) references division(division_name) on delete
cascade
);

Create Table citizen (
    nid  varchar2(20),
    name  varchar2(20),
    date_of_birth date,
    occupation  varchar2(20),
    blood_group  varchar2(20),
    division_name  varchar2(20) not null,
    district_name  varchar2(20) not null,
    primary key(nid),
```

```
    foreign key (division_name) references division(division_name) on delete
cascade,
    foreign key (district_name) references district(district_name) on delete
cascade
);

Create Table license (
    license_id  varchar2(20),
    license_type  varchar2(20),
    issue_date date,
    expiration_date date,
    nid  varchar2(20) unique not null,
    primary key(license_id),
    foreign key (nid) references citizen(nid) on delete cascade
);

Create Table accident(
    incident_id varchar2(20),
    date_of_accident date,
    district_name  varchar2(20),
    number_of_death int,
    primary key (incident_id),
    foreign key (district_name) references district (district_name)
);

-- incident is a junction table
Create Table incident(
    incident_id varchar2(20),
    license_id varchar2(20),
    foreign key(incident_id) references accident (incident_id) on delete cascade,
    foreign key(license_id) references license (license_id) on delete cascade
);

create or replace type vmobiles as varray(10) of varchar2(20)
/
-- i do not know why this multi-value attribute creation breaks Oracle it was the
"/"
Create Table hospital(
    hospital_id  varchar2(20),
    hospital_name  varchar2(20),
    contact_number vmobiles,
    primary key (hospital_id)
);

--admission is a junction table
```

```
Create Table admission(
    admission_id   varchar2(20),
    date_of_admission date,
    description  varchar2(20),
    release_date date,
    hospital_id  varchar2(20),
    nid  varchar2(20),
    primary key (admission_id),
    foreign key(hospital_id) references hospital(hospital_id),
    foreign key(nid) references citizen(nid)
);
```

Explanation of Solution

The ER Diagram contains the database schema of a government system database. The solution Diagram has 6 entities and 2 junction tables.

1. Division: Division table essentially stores the division_name, division_size, and a description of the division. Here, the primary key is the division_name since it would be able to uniquely identify all records. It has a One-to-One relation with District and a One-to-Many relation with the Citizen table.

    a. Primary Key: division_name
    b. Foreign Key: (none)
    c. Relationships:
        i. One-to-One: District
        ii. One-to-Many: Citizen

2. District: District table stores the district_name, district_size, a short description of the district, and the division_name of the division in which it is located. The district_name being unique is taken as the primary key for the table. The division_name attribute is a foreign key ensuring a One-to-One relationship with the Division table. It also has a One-to-Many relationship with the Citizen table. The division_name attribute is set to not

null to ensure total participation.

a. Primary Key: district_name
b. Foreign Key:
    i. division_name from Division table
c. Relationships:
    i. One-to-One: Division
    ii. One-to-Many: Citizen

3. Citizen: The citizen table stores data about the nid, name, date_of_birth, occupation, blood_group, division_name, and the district_name of the registered citizen. The nid being unique has been chosen as the primary key for the table. It has the foreign keys division_name and district_name referencing Division and District table respectively. It ensures a Many-to-One relationship with both tables. The attributes are also given the constraint of not null to ensure total participation. It also has a One-to-One relationship with the license table and a Many-to-Many relationship with Hospital table ensured by the Incident junction table.

    a. Primary Key: nid
    b. Foreign Key:
        i. division_name from Division
        ii. district_name from District
    c. Relationships:
        i. One-to-One: License
        ii. Many-to-One: Division, District
        iii. Many-to-Many: Hospital junction table Admission

4. License: The license table contains information about the license_id, license_type, the issue_date and expiration_date of the license, and the nid of the person whose license it is. The license_id is designed as the primary key of the table since it would be unique for all records. The nid attribute is a foreign key referencing the Citizen table. The attribute is set to not null to

ensure total participation. It also has a Many-to-Many relationship with the Accident table.

a. Primary Key: license_id
b. Foreign Key:
    i. nid from Citizen
c. Relationships:
    i. One-to-One: Citizen
    ii. Many-to-Many: Accident junction table Incident

5. Accident: The accident table stores an incident_id, date_of_accident, district_name as the location, and the number_of_death. It has a foreign key, district_name referencing the District table, which ensures a Many-to-One relationship. It is also not null to implement total participation. It also has a Many-to-Many relationship with the License table.

    a. Primary Key: incident_id
    b. Foreign Key:
        i. district_name from District
    c. Relationships:
        i. Many-to-One: District
        ii. Many-to-Many: License junction table Incident

6. Hospital: The Hospital table stores information about the hospital_id, hospital_name, and contact_number which can store multiple phone numbers. The primary key is the hospital_id. The hospital entity has a Many-to-Many relationship with the Citizen table which is ensured by the admission junction table

    a. Primary Key: hospital_id
    b. Foreign Key: (none)
    c. Relationships:

i. Many-to-Many: Citizen junction table Admission

7. Incident: The incident table is a junction table that ensures a Many-to-Many relationship between the License and Accident table. It has two attributes, incident_id and license_id both of which are foreign keys referencing the Accident and License table respectively.

8. Admission: The admission table is a junction table that ensures a Many-to-Many relationship between Citizen and Hospital table. It has two foreign key attributes, hospital_id and nid referencing the Hospital table and Citizen table respectively. It also records additional attributes like date_of_admission, release_date, and a short description of the admission. To make searching easier, an additional attribute admission_id was introduced as the primary key converting this weak entity to a strong entity.

Analysis of the problem of Task 3

Twenty (20) new queries were provided after the lab related to the described government system. The solutions to those queries are provided below:

```sql
-- SQL Queries

-- Task 1
select division_name, count(district_name) as no_of_districts
from district
group by division_name
order by no_of_districts desc;

-- Task 2
select district_name, count(nid) as no_of_citizen
from citizen
group by district_name
having count(nid) >= 20000
order by count(nid) desc;
```

```sql
-- Task 3
select c.name as citizen_name, count(i.incident_id) as no_of_accidents
from citizen c, license l, incident i
where c.nid = l.nid and l.license_id = i.license_id and c.nid = '210'
group by c.name;

-- Task 4
select *
from (select hospital_id, max(hospital_name) as hospital_name,
count(admission_id) as no_of_patients
        from admission natural join hospital
        group by hospital_id
        order by no_of_patients desc)


where rownum <=5;

-- Task 5
select h.hospital_name as hospital_name, c.blood_group as patient_blood_group
from admission a, citizen c, hospital h
where a.hospital_id = h.hospital_id and a.nid = c.nid
order by h.hospital_id;

-- Task 6
select division_name, (division_size/no_of_citizen) as population_density
from (select division_name, count(nid) as no_of_citizen, max(division_size) as
division_size
        from citizen natural join division
        group by division_name
        order by count(nid) desc
    );

-- Task 7
select *
from (select division_name, (division_size/no_of_citizen) as population_density
        from (select division_name, count(nid) as no_of_citizen,
max(division_size) as division_size
                from citizen natural join division
                group by division_name
                order by count(nid) desc
            )
    )
where rownum<=3;
```

```sql
-- Task 8
select district_name, count(incident_id) as no_of_accidents
from accident
group by district_name
order by no_of_accidents desc;

-- Task 9
select division_name , no_of_accidents
from (select division_name, count(incident_id) as no_of_accidents
        from district natural join accident
        group by division_name
        order by no_of_accidents
        )
where rownum <=1;

-- Task 10
select license_type, count(incident_id) as no_of_accidents
from license natural join incident
where license_id = license_id and (license_type = 'professional' or license_type
= 'non-professional')
group by license_type
order by no_of_accidents;

-- Task 11
select *
from (select nid, name, (to_date(release_date)- to_date(date_of_admission)) as
admission_time
        from citizen natural join admission
        order by admission_time desc)
where rownum<=1;

-- Task 12
select *
from (select division_name, count(nid) as no_of_young_people
        from citizen
        where trunc(months_between(current_date,date_of_birth)/12)>=15 and
trunc(months_between(current_date,date_of_birth)/12)<=30


        group by division_name
        order by no_of_young_people)
where rownum<=1;

-- Task 13
select nid, name
```

```sql
from citizen natural join license
where current_date>expiration_date
order by nid;

-- Task 14
select c.nid as nid, max(c.name) as name, count(a.incident_id) as no_of_accidents
from citizen c, license l, incident a
where c.nid = l.nid and
        l.license_id = a.license_id and
        l.license_id =  (select license_id
                        from citizen natural join license
                        where current_date>expiration_date)
group by c.nid
order by no_of_accidents desc;

-- Task 15
select license_id, name
from license natural join citizen
where license_id not in (select license_id
                        from incident)
order by nid;

-- Task 16
select division_name, sum(number_of_death) as total_number_of_deaths
from district natural join accident
group by division_name
order by total_number_of_deaths;

-- Task 17
select name
from citizen natural join license
where trunc(months_between(issue_date, date_of_birth)/12) <= 22 or
        trunc(months_between(issue_date, date_of_birth)/12)>=40
order by license_id;

-- Task 18
select c.name as citizen_name
from citizen c, license l, incident i, accident a, admission ad
where c.nid = l.nid and
        l.license_id = i.incident_id and
        i.incident_id = a.incident_id and
        ad.nid = c.nid and
        a.date_of_accident = ad.date_of_admission
group by c.name
order by c.name;
```

```sql
-- Task 19
select *
from (select h.hospital_name as hospital_name, count(c.nid) as
number_of_people_from_dhaka
from citizen c, admission a, hospital h
where c.nid = a.nid and
        h.hospital_id = a.hospital_id and
        c.division_name = 'Dhaka'
group by h.hospital_name
order by number_of_people_from_dhaka desc)
where rownum<=1;

-- Task 20
select c.nid, max(c.name) as citizen_name
from citizen c, license l, incident i, accident a
where c.nid = l.nid and
        l.license_id = i.license_id and
        i.incident_id = a.incident_id and
        c.district_name != a.district_name
group by c.nid
order by citizen_name;

--  ;-;
```

Explanation of Solution

1. Select division_name and count the district_name in the district table, grouped by division_name. Show the result in descending order to show larger divisions first.

2. Select district_name and count the nid from the citizen table, grouped by district_name. Introduce a having condition where the count of nid has to be greater than or equal to 20000 to be selected. Display the results in descending order to show the most populated districts first.

3. Joining three tables is difficult, so the cartesian product of three tables with the necessary where conditions to make it similar to a natural join. Then select the name of the citizen table, and count incident_id for the number

of incidents in the accidents table grouped by the name of the citizen table. Introduce a where condition to display the name and number of accidents of only that person, whose nid is equal to 210.

4. Join the admission and hospital table. Then select hospital_name, hospital_name, and count the admission_id for the number of admitted patients, grouped by hospital_id. Apply a dummy aggregate function of hospital_name like max( ). The result will be ordered by the no_of_patients and finally nest it all in a subquery, displaying only the top 5 rows.

5. Joining three tables is difficult. So, make a cartesian product between admission, citizen, and hospital table and apply the necessary conditions to make it similar to a natural join. Then select the hospital_name from the hospital table, and the blood group from the citizen table and display it ordered by the hospital_name.

6. Create a subquery to select the no_of_citizen in a division by counting the nid and the division size and division_name by joining the citizen and division table grouped by division_name and ordered by the no_of_citizen in descending order. Then calculate the population density and display it alongside the division_name.

7. Implement the entirety of the previous query in the form of a subquery. Select all the attributes from there and display only the top 3 rows.

8. Select district_name and count the incident_id from the accident table grouped by district_name. Display the results ordered by no_of_accidents to show areas with the higher number of accidents first.

9. Select division_name and no_of_accidents from a resultant subquery where you join the district and accident table, and select the division_name, count the incident_id as the no_of_accidents joining the

district and accident table, grouped by division_name. Finally display only the first row.

10. Join the license and incident table, select the license_type and count incident_id for the number_of_accidents grouped by license_type. Implement a where condition when it will only select the license_type which is professional or non-professional. Display the results in descending order ordered by no_of_accidents.

11. In a subquery, select the nid, and name and calculate the difference between release_date and date_of_admission to determine the admission_period. Then order it in descending order ordered by the admission_period. Finally display only the top row of the subquery for the person with the longest admission_period.

12. Similar to a previous query, count the number of citizens in a division in a subquery with a condition that the age has to be greater than equal to 15 and lower than equal to 30. To calculate the age, calculate the months between current_date and date_of_birth and divide it by 12 and truncate the value.

13. Join citizen and license and select name and nid where the current_date is greater than the expiration_date.

14. Joining three tables is difficult, so create the cartesian product of citizen, license, and incident with the appropriate where conditions to simulate a natural join. In a subquery, input the query made in task 13, which matches with the license_id in the license table, and then select nid from a citizen, name from a citizen in a dummy aggregate max( ) function, and count the incident_id in the accident table for the no_of_accidents which is grouped by the nid in citizen table. Finally, order the no_of_accidents in descending order to display the higher no_of_accidents.

15. Join the license and citizen table. Select the license_id and name from the citizen table. In a subquery, select the license_id present in the incident table. Implement a where condition which selects only the license_id, not in the subquery.

16. Join the accident and district table. Select the division_name and use the sum( ) aggregate function to calculate the total number of deaths, grouped by division_name.

17. Join the citizen and license table. Select the citizen name where the ages are less than equal to 22 or greater than equal to 40. To calculate the age, calculate the months between issue_date and the date_of_birth.

18. Joining 5 tables is a difficult task, so create a cartesian product of the citizen, license, incident, accident, and admission tables. Apply necessary where conditions to simulate a natural join. Introduce a new where condition where the date_of_accident in the accident table is the same as the date_of_admission in the admission table and select the name of those citizens.

19. Join citizen admission and hospital via cartesian product. Select the hospital_name, and count the nid for the number_of_people where the division_name is equal to 'Dhaka', grouped by the hospital_name and ordered by the number_of_people in descending order. Then take the entire query as a subquery, and display only the top row.

20. Join the citizen, license, incident, and accident tables via cartesian product. Then select the nid and name in a dummy max aggregate function from the citizen table, and introduce a new where condition which makes it so that only the citizen names where the district_name in the accident table and the district_name in the citizen table do not match. Group by nid and order by citizen_name to avoid duplicate names.

Problems

While working in the lab, operating functions on date variables proved to be a challenge. There are still many parts of the date variable that remain unknown.

Sometimes, implementing group by is difficult when selecting multiple variables which aren't in the group by or in an aggregate function. For those dummy aggregate functions had to be introduced.

During the lab task, implementing the multi-value attribute type vmobiles proved to be a challenge as an incorrect syntax for multi-value attribute declaration was shown. This impacted the submission time during the lab.

Findings

Entity Relationship Diagrams are immensely helpful in visualizing the DDL and during creating queries. Looking at the ERD while writing the queries helped significantly in determining which tables to join, which attributes are present where etc.

The date is a very unique attribute in a database that can be used in many different ways.

Junction tables need not be shown as separate tables in ERD.

Conclusion

The lab was a fruitful experience, though the sheer volume of tasks was quite taxing, which has helped develop our understanding of Entity Relations. It shows the importance of ERD in visualizing the relationship between all the different entities present in a database.