DBMS Project Report

PES University

Database Management Systems

UE18CS252

Submitted By

PES1201801455 Prajwal K Naik

The database chosen was the pandemic database.
The database contains data about the patients. This includes name, address, gender, age, phone, testing details, the ward he/she belongs to and the hospital he/she is admitted in.
The country relation contains the name of the countries around the world.
The ward relation contains details about the ward in the city.
The hospital relation contains details about the hospitals in which the patients have been admitted in.
The patient contact relation has been included to keep a record of the people that the patient has come in contact with. This is a n:n relationship.
The contact relation holds details about the different people who have been contacted by the details. This includes the name, address, gender, age, phone, the date on which he/she has been contacted on, the ward he/she belongs to and the details of the test and the type of contact(1 for primary or 2 for secondary).
Finally, the travel history relation, describes the travel history of the patient if any, including the arrival date and the country from which he/she has arrived from.
This database can be used in times of pandemics, like the ongoing COVID-19 pandemic, to keep a systematic report of the spread of diseases. This pandemic database could be integrated with the hospital database to keep track of the ongoing medication on each of the patients.

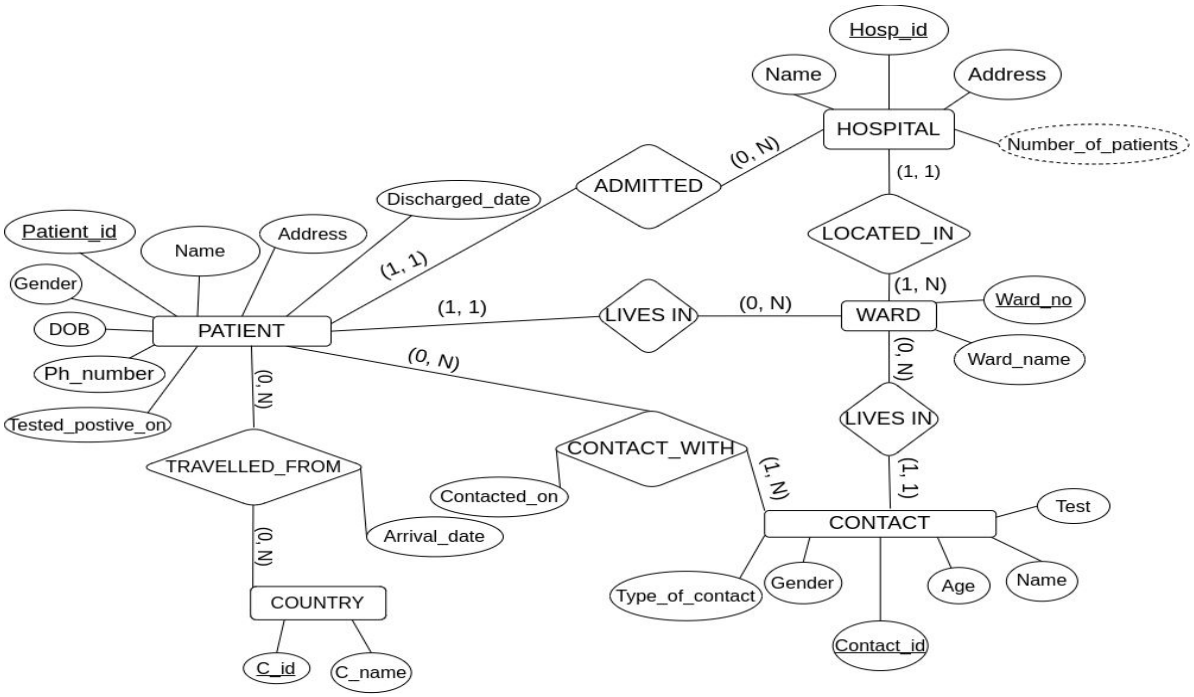
Index

Introduction

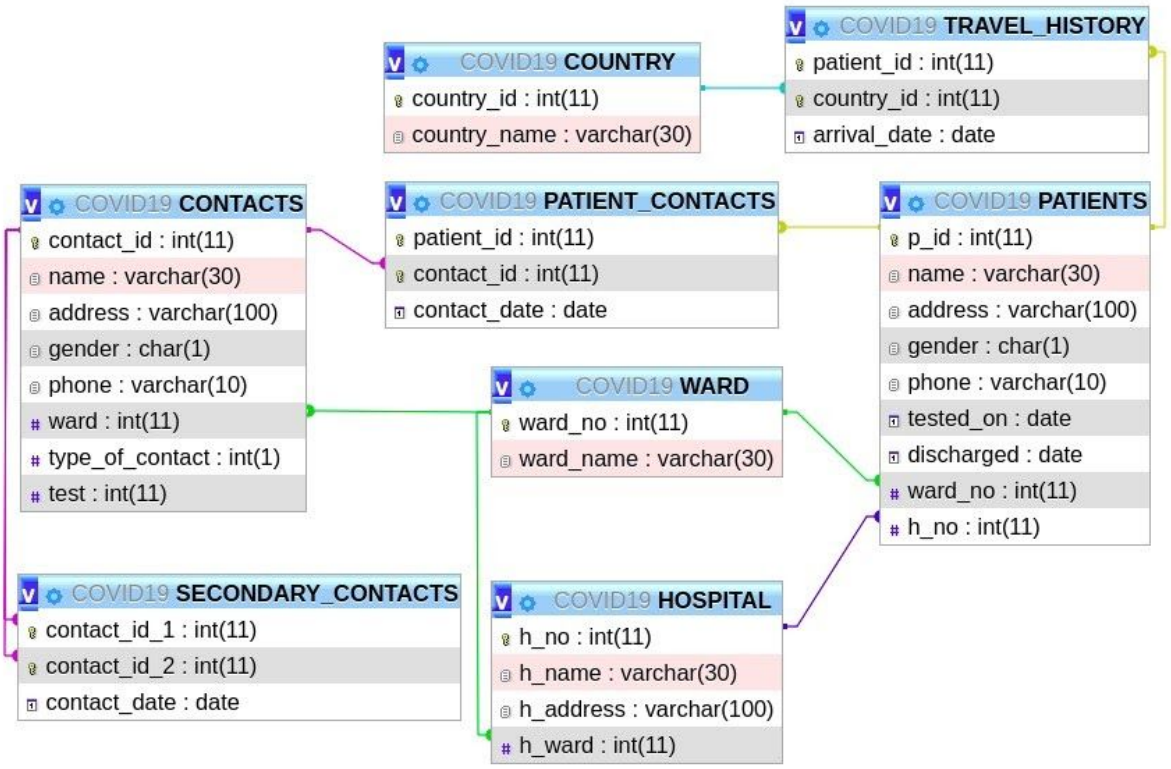
The database modelled in this project is the pandemic database. The database is composed of the patients, contacts, travel history and hospital details. Everytime a patient is tested positive for the disease, his/her details are entered into the patient table. The contacts of the patient are identified and are filled in the contact details table. To keep a record of the contacts of a particular patient, the patient_contact relation is updated. Everytime a contact is tested positive, his or her details are entered into the patients table. This way, the disease history of the contact-turned-patient can be preserved.

Data Model

The ER diagram is as follows :



The relational schema is as follows :



The keys for each relation are as follows :

PATIENTS : For this relation the p_id attribute was chosen as the primary key because each patient is given a unique patient identification number. No two patients can have the same number which gives the attribute the uniqueness.

The data types for this relation are as follows :

- p_id : int
- name : varchar
- address : varchar
- gender : char
- phone : varchar
- tested_on : date
- discharged : date
- ward_no : int
- h_no : int

CONTACTS : The contact_id attribute has been chosen as the primary key for this relation because it is easier to keep track of each contact by assigning them a unique identification number. No two contacts can have the same number.

The data types for this relation are as follows :

- contact_id : int
- name : varchar
- address : varchar
- gender : char
- phone : varchar
- ward : int
- type_of_contact : int
- test : int

HOSPITAL : The h_no is chosen as the primary key for this relation because hospitals are given a unique identification number by the healthcare authorities which is never repeated.
The data types for this relation are as follows:

- h_no : int
- h_name : varchar
- h_address : varchar
- h_ward : int

WARD : Each ward has a ward_no given by the city administration. This helps identify each of the ward by it's number even if they have the same name.
The data types for the following relation are as follows:

- ward_no : int
- ward_name : int

PATIENT_CONTACTS : Since this relation hold details about the individuals who have come in contact with the infected patients, it has the set of patient_id and contact_id as it's primary key because each of the patients could have come in contact with more than one individual and vice-versa.
The data types for this relation are as follows:

- patient_id : int
- contact_id : int
- contact_date : date

SECONDARY_CONTACTS : This relation hold details about the contacts who have come in contact with the primary contacts included in the previous relation. The set of contact_id_1 and contact_id_2 is the primary key for the relation.
The data types for this relation are as follows:

- contact_id_1 : int
- contact_id_2 : int
- contact_date : date

COUNTRY : Each country has a number associated with it, the country_id which acts as the primary key even though two countries might have the same name.
The data types for this relation as as follows :

- country_id : int
- country_name : int

TRAVEL_HISTORY : For this relation, the set of patient_id and country_id is chosen as the primary key because a patient could have travelled in from many countries and vice-versa.
The data types for this relation are as follows :

- patient_id : int
- country_id: int

FD and Normalization

The functional dependencies of the relations are as follows :

PATIENTS :

p_id -> {name, address, gender, phone, tested_on, discharged, ward_no, h_no}

CONTACTS :

contact_id -> {name, address, gender, phone, ward, type_of_contact, contact_date, test}

COUNTRY :

country_id -> country_name

HOSPITAL :

h_no -> {h_no, h_name, h_address, h_ward}

WARD :

ward_no -> ward_name

PATIENT CONTACTS :

{patient_id, contact_id} -> contact_date

SECONDARY_CONTACTS:

{contact_id_1, contact_id_2} -> contact_date

TRAVEL_HISTORY :

{patient_id, country_id} -> arrival_date

All FDs are in BCNF

The candidate keys of the relations mentioned above, have been obtained by the attribute closure of the functional dependencies that have been listed above.

TEST FOR LOSSLESS JOIN

The lossless join property is violated if natural join of relations in the database results in spurious tuples, that is it results in additional tuples representing erroneous and invalid information or results in loss of tuples.

Consider the relation MASTER_PATIENT with the following relational schema and functional dependencies.

P_id	Name	Addr	Gender	Age	Ph	Tested_date	Discharged	Ward	H_no	H_nam	H_addr	H_ward

1. p_id -> {name, address, gender, phone, tested_on, discharged, ward_no, h_no}
2. h_no -> {h_no, h_name, h_address, h_ward}

Because of the above functional dependencies, the relation violates the 3NF. So, the relation has been decomposed into the following two sub-relations.

P_id	Name	Address	Gender	Age	Ph	Tested_on	Discharged	Ward_no	Hosp_id

and

Hosp_id	H_name	Address	H_ward

Let's call the two decompositions as R1 and R2 respectively.
It is observed that $R1 \cap R2$ is Hosp_id and R1 - R2 is {H_name, Address, H_ward}
Since the functional dependency, $R1 \cap R2 \rightarrow$R1-R2 i.e h_no -> {h_no, h_name, h_address, h_ward}, it can be concluded that the decomposition is lossless or non-additive.

DDL

```
CREATE DATABASE COVID19;
USE COVID19;

CREATE TABLE COUNTRY(
   country_id int NOT NULL,
   country_name varchar(30) NOT NULL,
   PRIMARY KEY(country_id)
);

CREATE TABLE WARD(
   ward_no int NOT NULL PRIMARY KEY,
   ward_name VARCHAR(30) NOT NULL
);

CREATE TABLE HOSPITAL(
   h_no int PRIMARY KEY,
   h_name varchar(30) NOT NULL,
   h_address varchar(100),
   h_ward int NOT NULL,
   FOREIGN KEY(h_ward) REFERENCES WARD(ward_no)
      ON DELETE RESTRICT ON UPDATE CASCADE
);

CREATE TABLE PATIENTS(
   p_id int PRIMARY KEY AUTO_INCREMENT,
   name varchar(30) NOT NULL,
   address varchar(100),
   gender char,
   phone VARCHAR(10) NOT NULL,
   tested_on date NOT NULL,
   discharged date,
   ward_no int NOT NULL,
   h_no int NOT NULL DEFAULT 1,
   FOREIGN KEY(ward_no) REFERENCES WARD(ward_no)
      ON DELETE RESTRICT      ON UPDATE CASCADE,
```

```
    FOREIGN KEY(h_no) REFERENCES HOSPITAL(h_no)
        ON DELETE RESTRICT    ON UPDATE CASCADE
);

CREATE TABLE TRAVEL_HISTORY(
    patient_id int NOT NULL,
    country_id int NOT NULL,
    arrival_date date,
    PRIMARY KEY(patient_id, country_id),
    FOREIGN KEY(patient_id) REFERENCES PATIENTS(p_id)
        ON DELETE RESTRICT      ON UPDATE CASCADE,
    FOREIGN KEY(country_id) REFERENCES COUNTRY(country_id)
        ON DELETE RESTRICT      ON UPDATE CASCADE
);

CREATE TABLE CONTACTS(
    contact_id int PRIMARY KEY,
    name varchar(30) NOT NULL,
    address varchar(100) NOT NULL,
    gender char,
    phone VARCHAR(10) NOT NULL,
    ward int,
    type_of_contact int(1) NOT NULL,
    test int NOT NULL,
    FOREIGN KEY(ward) REFERENCES WARD(ward_no)
        ON DELETE RESTRICT     ON UPDATE CASCADE,
    CONSTRAINT chk_phone
          CHECK (phone NOT LIKE '%[^0-9]%'),
          CONSTRAINT chk_type
          CHECK (type_of_contact=1 or type_of_contact=2 or type_of_contact=3),
    CONSTRAINT chk_test
          CHECK (test=0 or test=1)
);

CREATE TABLE PATIENT_CONTACTS(
    patient_id int NOT NULL,
    contact_id int NOT NULL,
    contact_date date NOT NULL,
    PRIMARY KEY(patient_id, contact_id),
    FOREIGN KEY(patient_id) REFERENCES PATIENTS(p_id)
        ON DELETE RESTRICT   ON UPDATE CASCADE,
    FOREIGN KEY(contact_id) REFERENCES CONTACTS(contact_id)
        ON DELETE CASCADE       ON UPDATE CASCADE
);

CREATE TABLE SECONDARY_CONTACTS(
    contact_id_1 int NOT NULL,
```

```
    contact_id_2 int NOT NULL,
    contact_date date NOT NULL,
    PRIMARY KEY(contact_id_1, contact_id_2),
    FOREIGN KEY(contact_id_1) REFERENCES CONTACTS(contact_id)
        ON DELETE CASCADE   ON UPDATE CASCADE,
    FOREIGN KEY(contact_id_2) REFERENCES CONTACTS(contact_id)
        ON DELETE CASCADE   ON UPDATE CASCADE
);
```

Triggers

The trigger in the database updates the PATIENTS relation with the patient details when any of the already existing contacts or if a newly added contact is tested positive.

Trigger for new entry into CONTACTS:

```
DELIMITER $$
CREATE TRIGGER CONTACT_INSERT
AFTER INSERT
ON CONTACTS FOR EACH ROW
BEGIN
        IF NEW.test=1 THEN
        INSERT INTO PATIENTS(name, address, gender, phone, tested_on,
discharged, ward_no)
      VALUES(new.name, new.address, new.gender, new.phone, CURDATE(),
NULL, new.ward);
   END IF;
END $$
```

Trigger for update of an existing tuple in CONTACTS:

```
DELIMITER $$
CREATE TRIGGER CONTACT_UPDATE
AFTER INSERT
ON CONTACTS FOR EACH ROW
BEGIN
        IF NEW.test=1 THEN
        INSERT INTO PATIENTS(name, address, gender, phone, tested_on,
discharged, ward_no)
      VALUES(new.name, new.address, new.gender, new.phone, CURDATE(),
NULL, new.ward);
   END IF;
END $$
```

SQL Queries

1. Finding the number of cases

```
SELECT a.active_cases, d.discharged_cases, t.total_cases
FROM
(
   SELECT
      count(*) as active_cases
   FROM
      PATIENTS
   WHERE
      discharged IS NULL
)AS a,
(
   SELECT
      count(*) as discharged_cases
   FROM
      PATIENTS
   WHERE
      discharged IS NOT NULL
) AS d,
(
   SELECT
      count(*) as total_cases
   FROM
      PATIENTS
) AS t;
```

2. Selecting patients who have travelled in from China

```
SELECT
   P.p_id, P.name, A.arrival_date
FROM
   PATIENTS P,
   (
      SELECT
         patient_id, arrival_date
      FROM
      (
         TRAVEL_HISTORY NATURAL JOIN COUNTRY
      )
   WHERE
      country_name='China'
   ) AS A
```

```
WHERE
        P.p_id=A.patient_id;
```

3. Select the contacts who have come in contact with more than one patients

```
SELECT *
FROM
   CONTACTS
WHERE
   contact_id
IN
(
   SELECT contact_id
        FROM
      PATIENT_CONTACTS
        GROUP BY
      contact_id
        HAVING
      count(*)>1
);
```

4. Select the patient details of the patient who has travelled the most

```
SELECT * FROM
PATIENTS
NATURAL JOIN
(
   SELECT
      r.p_id
   FROM
   (
      SELECT
         p_id, count(*) as travel_count
      FROM
      (
         SELECT *
         FROM
         (
            PATIENTS as P LEFT OUTER JOIN TRAVEL_HISTORY AS T ON
P.p_id=T.patient_id
         )
      ) AS j
      WHERE
         j.country_id IS NOT NULL
      GROUP BY
         p_id
   ) AS r
```

```
   WHERE
   r.travel_count=
   (
      SELECT
         MAX(x.travel_count)
      FROM
      (
      SELECT
         count(*) as travel_count
      FROM
      (
         SELECT *
         FROM
         (
            PATIENTS as P LEFT OUTER JOIN TRAVEL_HISTORY AS T ON
P.p_id=T.patient_id
         )
      ) AS j
      WHERE
         j.country_id IS NOT NULL
      GROUP BY
         p_id
      )AS x
   )
) AS Y
WHERE
   p_id=Y.p_id;
```

5. Ward details of ward having most number of patients

```
SELECT * FROM
   WARD NATURAL JOIN
(
   SELECT
      b.ward_no
   FROM
   (
      SELECT
         ward_no, count(*) as count_ward
      FROM
         PATIENTS
      GROUP BY
         ward_no
   ) AS b
   WHERE
      b.count_ward=
      (
```

```
        SELECT
            max(a.count_ward)
        FROM
        (
            SELECT
                count(*) as count_ward
            FROM
                PATIENTS
            GROUP BY
                ward_no
        ) AS a
    )
) AS j
WHERE
    ward_no=j.ward_no;
```

6. Finding hospitals with number of patients greater than average

```
SELECT * FROM
    HOSPITAL NATURAL JOIN
(
    SELECT
        b.h_no
    FROM
    (
        SELECT
            h_no, count(*) as count_hosp
        FROM
            PATIENTS
        GROUP BY
            h_no
    )AS b
    WHERE
        b.count_hosp >
        (
            SELECT
                avg(a.pat_count)
            FROM
            (
                SELECT
                    COUNT(*) as pat_count
                FROM
                    PATIENTS
                GROUP BY
                    h_no
            ) AS a
        )
```

)AS j;

7. Find ward with most number of people who haven't travelled abroad

```
SELECT * FROM
WARD
NATURAL JOIN
(
   SELECT
      b.ward_no
   FROM
   (
      SELECT
         a.ward_no, count(*) as pat_count
      FROM
      (
         SELECT
            ward_no
         FROM
            PATIENTS
         WHERE
            p_id
         NOT IN
         (
            SELECT DISTINCT
               patient_id
            FROM
               TRAVEL_HISTORY
         )
      ) as a
      GROUP BY
         a.ward_no
   ) as b
   WHERE
         b.pat_count=
      (
         SELECT
           max(c.pat_count)
         FROM
         (
            SELECT
                  COUNT(*) as pat_count
            FROM
                  PATIENTS
            WHERE
                  p_id
            NOT IN
```

```
            (
               SELECT DISTINCT
                    patient_id
               FROM
                    TRAVEL_HISTORY
            ) group by ward_no
           ) as c
      )
) as j
WHERE ward_no=j.ward_no;
```

Conclusion

This database can be used in times of pandemics, like the ongoing COVID-19 pandemic, to keep a systematic report of the spread of diseases. This pandemic database could be integrated with the hospital database to keep track of the ongoing medication on each of the patients.
The limitations of this model is the lower level of automation. The link between patients and contacts can be identified only after the contact data has been updated in the CONTACTS relation. Moreover, there are certain drawbacks while scaling up this particular schema to include patients from different cities as the wards in those cities will have to be identified correctly.