CSE 4020/5231 - Database Systems Project

Lingjing Huang, Shreyas Ugemuge

December 1, 2017

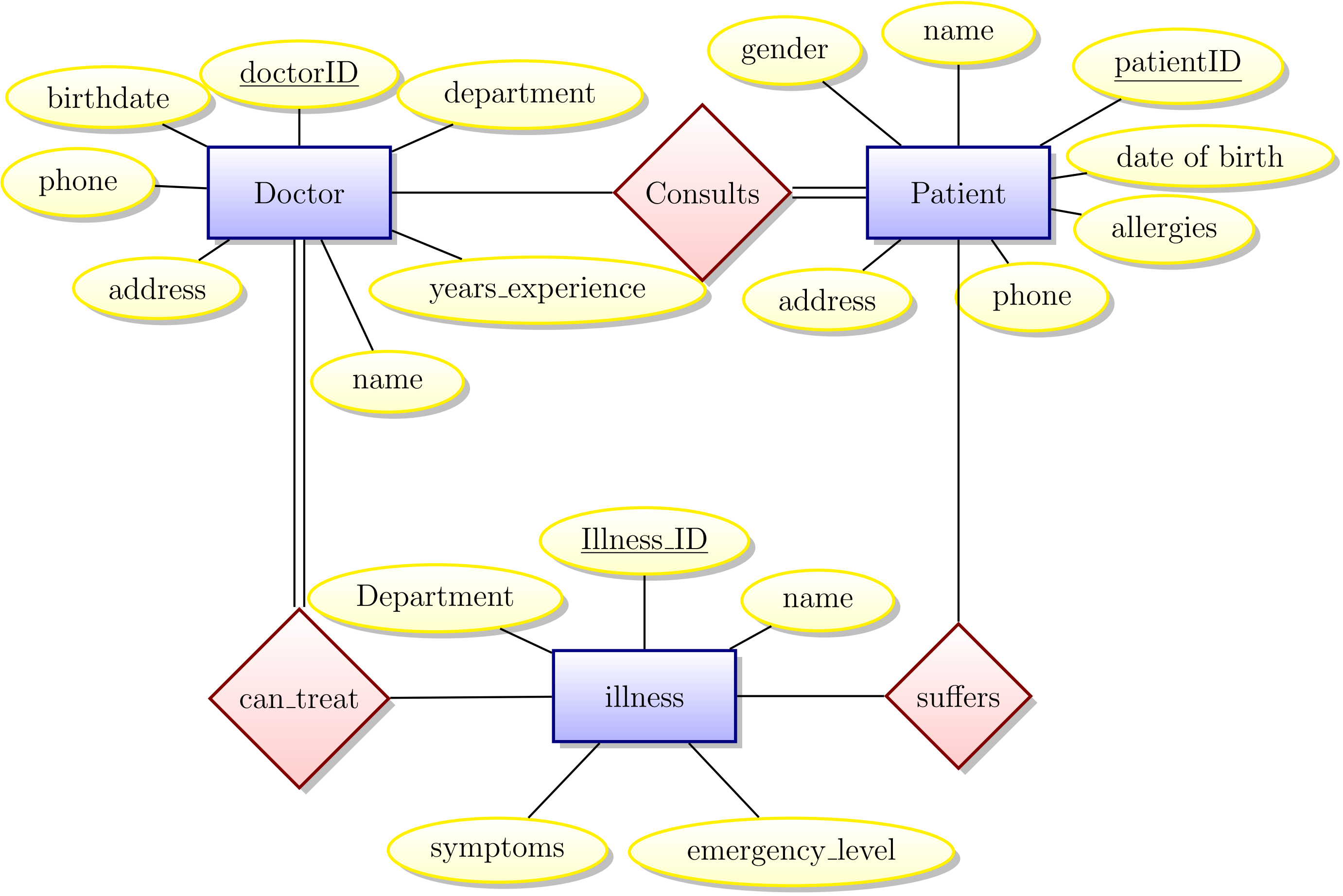
Contents

Problem Statement

Design a database for storing information about medical doctors, their patients, and ilnesses (using sqlite3):

1. Draw its E-R diagram.
2. Give a relational representation of the E-R diagram that allows for 2 functional preserving and lossless join decompositions. Describe its functional dependencies.
3. Give the two decompositions and prove that they are lossless join and functional preserving.
4. Give the SQL DDLs for creating the three versions of the database.
5. Fill the first database with data (at least 100 rows in each table).
6. Give the SQL queries to copy the data from the first version into its 2 decompositions.
7. Propose in english 3 queries that require at least 2 table joins each and such that all tables are involved in at least 2 queries.
8. Propose SQL implementations of the 3 queries on all three versions of the database.
9. Test the time in ns for exeuting the 3 queries on each database, by running each of them 1000 times.

1 ER-Diagram



2 Relational Representation and Functional Dependencies

2.1 Relations

$Doctor(\underline{Doctor\_ID}, Doctor\_Name, Phone, Address, Birthday, ExperiencesYears, Department)$
$Patient(\underline{Patient\_ID}, Patient\_Name, Phone, Address, Birthday,$
$Gender, AllergiesHistory, Doctor\_ID, Illness\_ID)$
$Illness(\underline{Illness\_ID}, Illness\_Name, Department, Symptoms, EmergencyLevel)$

2.2 Functional Dependencies

2.2.1 Doctor

$Doctor\_Name, Phone \rightarrow Doctor\_ID, Department, ExperienceYears$
$Doctor\_ID \rightarrow Doctor_{N}ame, Address, Phone, Birthday$

2.2.2 Patient

$Patient\_Name, Phone \rightarrow Patient_ID, Address, Birthday, AllergiesHistory$
$Patient\_ID \rightarrow Patient_Name, Doctor\_ID, Illness\_ID, Phone, Gender$

2.2.3 Illness

$Illness\_Name \rightarrow Illness_ID, Department, Symptoms$
$Illness\_ID \rightarrow Illness\_Name, EmergencyLevel$

3 Decompositions

3.1 Decomposition 1

3.1.1 Decomposing Doctor

$R1$: Doctor Contact Info ($Doctor\_Name, Phone, Address, Department$)
$R2$: Doctor Personal Info ($Doctor\_ID, Doctor\_Name, Phone, Department, Birthday$)

All other relations remain the same as the original schema

3.1.2 Proof: Lossless Join and Functional Preserving

First condition holds true as
Att(R1) ∪ Att(R2)
= (Doctor_Name, Phone, Address, Department) ∪ (Doctor_ID, Doctor_Name, Phone, Department, Birthday)
= (Doctor_ID, Doctor_Name, Phone, Address, Birthday, Experiences Years, Department)
= Att(R).

Second condition holds true as
Att(R1) ∩ Att(R2)
= (Doctor_Name, Phone, Address, Department) ∩ (Doctor_ID, Doctor_Name, Phone, Department, Birthday)
= (Doctor_Name, Phone)
$\neq \phi$

Third condition holds true as
Att(R1) ∩ Att(R2) = (Doctor_Name, Phone, Department) is a key of R1 (Doctor_Name, Phone, Address, Department) because of the FD's given

Furthermore, all dependencies of R either can be a part of R1 or R2 or must be derivable from combination of FDs of R1 and R2.

3.2 Decomposition 2

3.2.1 Decomposing Patient

R1: Patient Contact Info $(Patient\_Name, Phone, Address, Gender)$
R2: Patient Personal Info $(Patient\_Name, Phone, Patient\_ID,$
$Birthday, AllergiesHistory, Doctor\_ID, Illness\_ID)$

All other relations remain the same as the original schema

3.2.2 Proof: Lossless Join and Functional Preserving

First condition holds true as
Att(R1) ∪ Att(R2)
= (Patient_Name, Phone, Address, Gender) ∪ (Patient_Name, Phone, Patient_ID, Birthday, Allergies History, Doctor_ID, Illness_ID)
= (Patient_ID, Patient_Name, Phone, Address, Birthday, Gender, Allergies History, Doctor_ID, Illness_ID)
= Att(R).

Second condition holds true as
Att(R1) ∩Att(R2)
= (Patient_Name, Phone, Address,Gender) ∩ (Patient_Name, Phone, Patient_ID, Birthday, Allergies History, Doctor_ID, Illness_ID)
$\neq \phi$.

Third condition holds true as
Att(R1) ∩ Att(R2)
= (Patient_Name, Phone) is a key of
R1(Patient_Name, Phone, Address, Patient_ID) because of the given FD's

Furthermore, all dependencies of R either can be a part of R1 or R2 or must be derivable from combination of FDs of R1 and R2.

4 SQL DDLs

4.1 Version 1

This version is the original database as defined in section 2.1.
Filename: `create_d.ddl`

```
CREATE TABLE ‘Patient‘ (
‘Patient_ID‘  NUMERIC NOT NULL,
‘Patient_Name‘  TEXT NOT NULL,
‘Phone‘ NUMERIC,
‘Address‘ TEXT,
‘Birthday‘  NUMERIC,
  ‘Gender‘  TEXT,
  ‘Allergies_History‘ TEXT,
  ‘Doctor_ID‘ NUMERIC NOT NULL,
  ‘Illness_ID‘  NUMERIC,
  PRIMARY KEY(‘Patient_ID‘)
);
CREATE TABLE IF NOT EXISTS ”Doctor” (
  ‘Doctor_ID‘ NUMERIC NOT NULL UNIQUE,
  ‘Doctor_Name‘ TEXT NOT NULL,
  ‘Phone‘ NUMERIC,
  ‘Address‘ TEXT,
  ‘Birthday‘  NUMERIC,
  ‘Experiences_Year‘  INTEGER,
  ‘Department‘  TEXT,
  PRIMARY KEY(‘Doctor_ID‘)
);
CREATE TABLE ‘Illness‘ (
  ‘Illness_ID‘  NUMERIC NOT NULL UNIQUE,
  ‘Illness_Name‘  TEXT NOT NULL UNIQUE,
  ‘Department‘  TEXT,
  ‘Symptom‘ TEXT,
  ‘Emergency_Level‘ INTEGER,
  PRIMARY KEY(‘Illness_ID‘)
);
```

4.2 Version 2

This version was created using Decomposition 1 (Section 3.1)
Filename: `create_d2.ddl`

```
CREATE TABLE IF NOT EXISTS ”Doctor Personal Info” (
  ‘Doctor_ID‘ NUMERIC NOT NULL UNIQUE,
  ‘Doctor_Name‘ TEXT NOT NULL,
  ‘Phone‘ NUMERIC,
  ‘Department‘  TEXT,
  ‘Birthday‘  INTEGER,
  PRIMARY KEY(‘Doctor_ID‘)
);
CREATE TABLE IF NOT EXISTS ”Doctor Contact Info” (
  ‘Doctor_Name‘ TEXT NOT NULL,
  ‘Phone‘ NUMERIC,
  ‘Address‘ TEXT,
  ‘Department‘  TEXT
);
CREATE TABLE IF NOT EXISTS ”Illness” (
  ‘Illness_ID‘  NUMERIC NOT NULL UNIQUE,
  ‘Illness_Name‘  TEXT NOT NULL,
  ‘Department‘  TEXT,
  ‘Symptom‘ TEXT,
  ‘Emergency_Level‘ INTEGER,
  PRIMARY KEY(‘Illness_ID‘)
);
CREATE TABLE IF NOT EXISTS ”Patient” (
  ‘Patient_ID‘  NUMERIC NOT NULL UNIQUE,
  ‘Patient_Name‘  TEXT NOT NULL,
  ‘Phone‘ NUMERIC,
  ‘Address‘ TEXT,
  ‘Birthday‘  NUMERIC,
  ‘Gender‘  TEXT,
  ‘Allergies_History‘ TEXT,
  ‘Doctor_ID‘ NUMERIC,
  ‘Illness_ID‘  NUMERIC,
  PRIMARY KEY(‘Patient_ID‘)
);
```

4.3 Version 3

This version was created using Decomposition 2 (Section 3.2)
Filename: `create_d3.ddl`

```
CREATE TABLE ‘Doctor‘ (
  ‘Doctor_ID‘ NUMERIC NOT NULL UNIQUE,
  ‘Doctor_Name‘ TEXT NOT NULL,
  ‘Phone‘ NUMERIC,
  ‘Address‘ TEXT,
  ‘Birthday‘  NUMERIC,
  ‘Experiences_Years‘ INTEGER,
  ‘Department‘  TEXT,
  PRIMARY KEY(‘Doctor_ID‘)
);
CREATE TABLE ‘Illness‘ (
  ‘Illness_ID‘  NUMERIC NOT NULL UNIQUE,
  ‘Illness_Name‘  TEXT NOT NULL,
  ‘Department‘  TEXT,
  ‘Symptom‘ TEXT,
  ‘Emergency_Level‘ INTEGER,
  PRIMARY KEY(‘Illness_ID‘)
);
CREATE TABLE IF NOT EXISTS ”Patient Personal Info” (
  ‘Patient_Name‘  TEXT NOT NULL,
  ‘Phone‘ NUMERIC,
  ‘Gender‘  TEXT,
  ‘Birthday‘  NUMERIC,
  ‘Allergies_History‘ TEXT,
  ‘Doctor_ID‘ NUMERIC,
  ‘Illness_ID‘  NUMERIC
);
CREATE TABLE IF NOT EXISTS ”Patient Contact Info” (
  ‘Patient_Name‘  TEXT NOT NULL,
  ‘Phone‘ NUMERIC,
  ‘Address‘ TEXT,
  ‘Patient_ID‘  INTEGER NOT NULL UNIQUE,
  PRIMARY KEY(‘Patient_ID‘)
);
```

5 Filling The database with data

Java code was used to generate and insert data into d.db. Filename: `DataGen.java`
Below are some relevant snippets:

5.1 Data Generation

Below is an example for java code used to generate symptoms for the illness table. It used a text file with a list of generic symptoms and randomly picks out a number of symptoms represented by i. Similar code was used to generate all other fields.

```
private static ArrayList<String> GenerateSymptoms (int i) throws IOException {
        ArrayList<String> Diseases = new ArrayList<>();
        Random r = new Random(System.currentTimeMillis());
        FileReader f = new FileReader(new File(”etc/sym.txt”));
        BufferedReader br = new BufferedReader(f);
        String s;
        for (int j = 0; j < i; j++) {
            for (int k = 0; k < r.nextInt(20); k++)
                br.readLine();
            s = br.readLine();
            Diseases.add(s.split(”^[^\\s]*\\s”)[1]);
        }
        br.close();
        return Diseases;
    }
```

5.2 Data Insertion

sqlite JDBC for Java was used to insert the generated data, here is the code for inserting data into the Patient table

```
public static void insertDoctor (int id, String name, long phone, String add, String birthday, int exp, String dept) {
        String sql = ”INSERT INTO Doctor(Doctor_ID,Doctor_Name,Phone,Address,Birthday,Experiences_Year,Department) VALUES(?,?,?,?,?,?,?)”;
        try (Connection conn = connect(); PreparedStatement pstmt = conn.prepareStatement(sql)) {
            pstmt.setInt(1, id);
            pstmt.setString(2, name);
            pstmt.setLong(3, phone);
            pstmt.setString(4, add);
            pstmt.setString(5, birthday);
            pstmt.setInt(6, exp);
            pstmt.setString(7, dept);
            pstmt.executeUpdate();
        } catch (SQLException e) {
            System.out.println(e.getMessage());
        }
    }
```

6 Copy the data from the first version to the decompositions

6.1 Copy data to Version 2

This queries was using to copy data from version 1. Filename: `copy_d_to_d2.sql`

```sql
.open d.db
ATTACH 'd2.db' as d2;

INSERT INTO d2.'Doctor Personal Info'
SELECT Doctor_ID, Doctor_Name, Phone, Department, Birthday FROM Doctor;

INSERT INTO d2.'Doctor Contact Info'
SELECT Doctor_Name, Phone, Address, Department FROM Doctor;

INSERT INTO d2.Illness
SELECT * FROM Illness;

INSERT INTO d2.Patient
SELECT * FROM Patient;
```

6.2 Copy data to Version 3

This queries was using to copy data from version 1. Filename: `copy_d_to_d3.sql`

```sql
.open d.db
ATTACH 'd3.db' as d2;

INSERT INTO d2.'Patient Personal Info'
SELECT Patient_Name, Phone, Gender, Birthday,Allergies_History,Doctor_ID, Illness_ID FROM Patient;

INSERT INTO d2.'Patient Contact Info'
SELECT Patient_Name, Phone, Address, Patient_ID FROM Patient;

INSERT INTO d2.Illness
SELECT * FROM Illness;

INSERT INTO d2.Doctor
SELECT * FROM Doctor;
```

7 English Queries

7.1 First English Query

Give Doctor_Name = ’Taren Batarse’, find all his patients Patient_ID and Patient_Name. (Relate to Doctor, Patient table)

7.2 Second English Query

Give Patient_Name is ’Erika Heuberger’ and Phone is ’2449717107’, find Doctor_Name, and her symptom. (Relate to Doctor, Illness, Patient table)

7.3 Third English Query

Give Illness_ID = ’1’, find all patient has this kind of illness Patient_Name and their Doctor_Name. (Relate to Doctor, Illness, Patient table)

8 SQL implementations of the queries

8.1 First Query

Given a Doctor_Name = ’Taren Batarse’, find all his patient’s Patient_ID and Patient_Name.

8.1.1 Version 1

Filename: `query_1_db1.sql`

```
select p.Patient_ID, p.Patient_Name
from ‘Patient‘ as p
join ‘Doctor‘ as d on d.Doctor_ID = p.Doctor_ID
where d.Doctor_Name = ’Taren Batarse’;
```

8.1.2 Version 2

Filename: `query_1_db2.sql`

```
select p.Patient_ID , p.Patient_Name
from Patient as p
join ‘Doctor Personal Info‘ as d on d.Doctor_ID = p.Doctor_ID
where d.Doctor_Name = ’Taren Batarse’;
```

8.1.3 Version 3

Filename: `query_1_db3.sql`

```
select p.Patient_ID, p.Patient_Name
from ’Patient Personal Info’ as pp
join Doctor as d on d.Doctor_ID = pp.Doctor_ID
join ’Patient Contact info’ as p on pp.Patient_Name = p.Patient_Name
where d.Doctor_Name=’Taren Batarse’;
```

8.2 Second Query

Given Patient_Name is ’Erika Heuberger’ and Phone is ’2449717107’, find their doctors Name, and her symptoms.

8.2.1 Version 1

Filename: `query_2_db1.sql`

```sql
select d.Doctor_Name, i.symptom
from Patient as p
join Doctor as d on d.Doctor_ID = p.Doctor_ID
join Illness as i on i.Illness_ID = p.Illness_ID
where p.Patient_Name = 'Erika Heuberger' and p.Phone = '2449717107'
```

8.2.2 Version 2

Filename: `query_2_db2.sql`

```sql
select d.Doctor_Name , i.symptom
from Patient as p
join ‘Doctor Personal Info‘ as d on d.Doctor_ID = p.Doctor_ID
join Illness as i on i.Illness_ID = p.Illness_ID
where p.Patient_Name = 'Erika Heuberger' and p.Phone = '2449717107'
```

8.2.3 Version 3

Filename: `query_2_db3.sql`

```sql
select d.Doctor_Name , i.Illness_ID
from ‘Patient Contact Info‘ as p
join ‘Patient Personal Info‘ as pp on p.Phone = pp.Phone
join Doctor as d on pp.Doctor_ID = d.Doctor_ID
join Illness as i on pp.Illness_ID = i.Illness_ID
where p.Patient_Name = 'Erika Heuberger' and p.Phone = '2449717107';
```

8.3 Third Query

Given Illness_ID = ’1’, find all patient that have this of illness Patient_Name along with their Doctor_Name.
This is second of the two queries that include all the tables in a the database.

8.3.1 Version 1

Filename: `query_3_db1.sql`

```sql
select p.Patient_Name, d.Doctor_Name
from Patient as p
join Doctor as d on d.Doctor_ID = p.Doctor_ID
join Illness as i on i.Illness_ID = p.Illness_ID
where p.Illness_ID = '1';
```

8.3.2 Version 2

Filename: `query_3_db2.sql`

```sql
select p.Patient_Name , d.Doctor_Name
from Patient as p
join `Doctor Personal Info` as d on d.Doctor_ID = p.Doctor_ID
join Illness as i on i.Illness_ID = p.Illness_ID
where i.Illness_ID = '1';
```

8.3.3 Version 3

Filename: `query_3_db3.sql`

```sql
select p.Patient_Name , d.Doctor_Name
from `Patient Personal Info` as pp
join `Patient Contact Info` as p on p.Phone = pp.Phone
join Doctor as d on pp.Doctor_ID = d.Doctor_ID
join Illness as i on i.Illness_ID = pp.Illness_ID
where i.Illness_ID = '1';
```

9 Conclusion: Query Execution Times

Results Filename: `results.csv`
Code Filename: `Analyze.java`

9.1 Method

Java along with SQLite JDBC was used for this step. 3 other methods were tried and rejected namely

- ”.timer on” command in the sqlite3 command line interface
- ”time” utility for linux
- GNU time utility

All 3 of the alternatives only had millisecond precision and hence were rejected.

The following Subsections explore some of the relevant code snippets

9.1.1 Retrieving sql queries from file and recording timings in a file

```java
    private static String queryFromFile (String filename) throws Exception {
        String query = "", temp;
        BufferedReader br = new BufferedReader(new FileReader(new File(filename)));
        while ((temp = br.readLine()) != null)
            query += temp + " ";
        br.close();
        return query;
    }

    public static void main (String[] args) throws Exception {
        long[][] times = new long[1000][9];
        for (int i = 1; i < 4; i++) {
            for (int j = 1; j < 4; j++) {
                for (int k = 0; k < 1000; k++) {
                    times[k][(3 * (i - 1)) + (j - 1)] = timeQuery(i, j);
                }
            }
        }
        toCSV(times);
    }
```

9.1.2 Connecting to database and timing the queries

```java
  private static Connection connect (int i) {
        // a ternary operator to make sure 1st database is read as d.db
        String url = "jdbc:sqlite:../d" + (i == 1 ? "" : i) + ".db";
        Connection conn = null;
        try {
            conn = DriverManager.getConnection(url);
        } catch (SQLException e) {
            System.out.println(e.getMessage());
        }
        return conn;
    }

    private static long timeQuery (int databaseNumber, int queryNumber) throws Exception {
        String query = queryFromFile("../Queries/query_" + queryNumber + "_db" + databaseNumber + ".sql");
        Connection c = connect(databaseNumber);
        Statement s = c.createStatement();
        long start = System.nanoTime();
        s.executeQuery(query);
        long end = System.nanoTime() - start;
        return end;

    }
```

9.2 Analysis

Below is a summary table with some statistics based on the 9*1000 - 1 data points. The only outlier removed was the first reading that was taken, this was because java adds a overhead the first time it runs the first query for the first database. All values are rounded to the ones place

	Query 1 DB 1	Query 1 DB 2	Query 1 DB 3	Query 2 DB 1	Query 2 DB 2	Query 2 DB 3	Query 3 DB 1	Query 3 DB 2	Query 3 DB 3
Average time (ns)	144837	162274	166437	131116	133930	175845	897936	246981	761884
Standard Deviation (%)	20	32	25	100	45	7	24	40	19

9.3 Conclusion

Query 1 and 2 perform better on the original version of the database by a small margin. Query 3 sees a significant performance gain in database 2. This is due to the doctor table being decomposed into doctor personal info and hence reducing the query time.

10 References

1. http://www.texample.net/tikz/examples/entity-relationship-diagram/
2. http://www.mockaroo.com - used this to generate random data
3. https://www.cdc.gov/datastatistics/index.html