Assignment – Database Design and Development and Data Management

Table of Contents

Problem 1

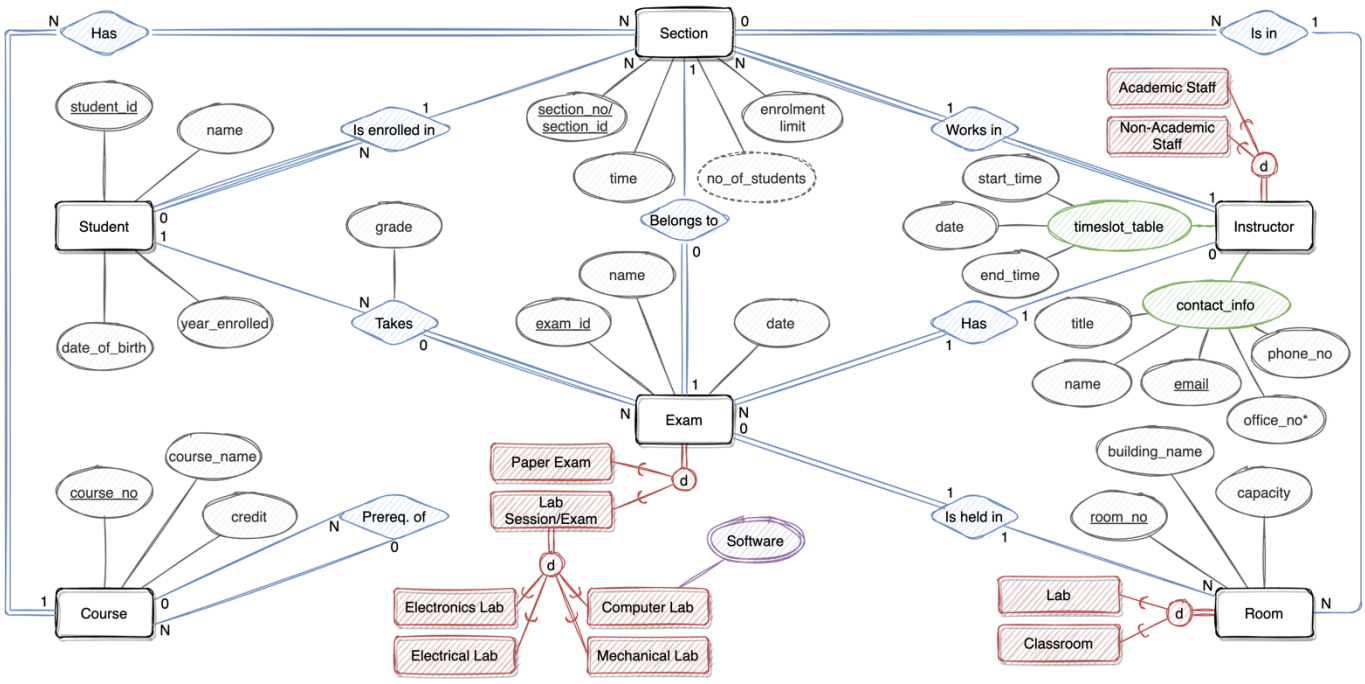

Figure 1: Enhanced Entity Relationship Diagram for the scenario described in Problem 1.

Problem 2

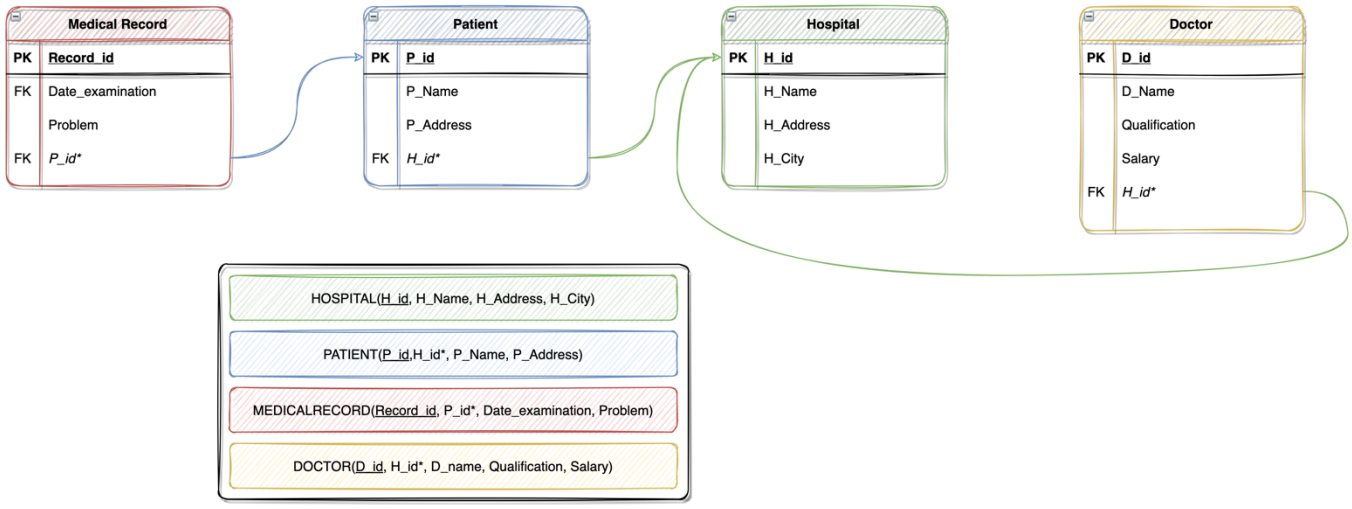

Figure 2: Relational Schema representing the database displayed in Problem 2.

Firstly, I identified all the entities, their respective attributes, including those which are to be used as primary keys. In the original diagram, the tables are identified by rectangles, and their attributes are represented by circles. Next, I identified the relationships between the entities, which are represented by diamonds. This is shown in the upper diagram in *Figure 2* by the lines linking each foreign key, shown with asterisks (*) to the appropriate primary key (represented by underlines) in its respective table. I then inserted the primary key from one entity into the entities that uphold relationships with it in the form of foreign keys. The cardinalities in each relationship in this scenario show 'One-to-Many' relationships, so the primary key from the 'One' side goes into the entitiy on the 'Many' side. Lastly, I aggregated the attributes of each entity, and their foreign keys and formatted them into a relational schema, displayed in the lower diagram in *Figure 2*. The upper diagram in *Figure 2* is present to show the process of arriving at the lower diagram in *Figure 2*, which is the final solution for Problem 2.

Problem 3

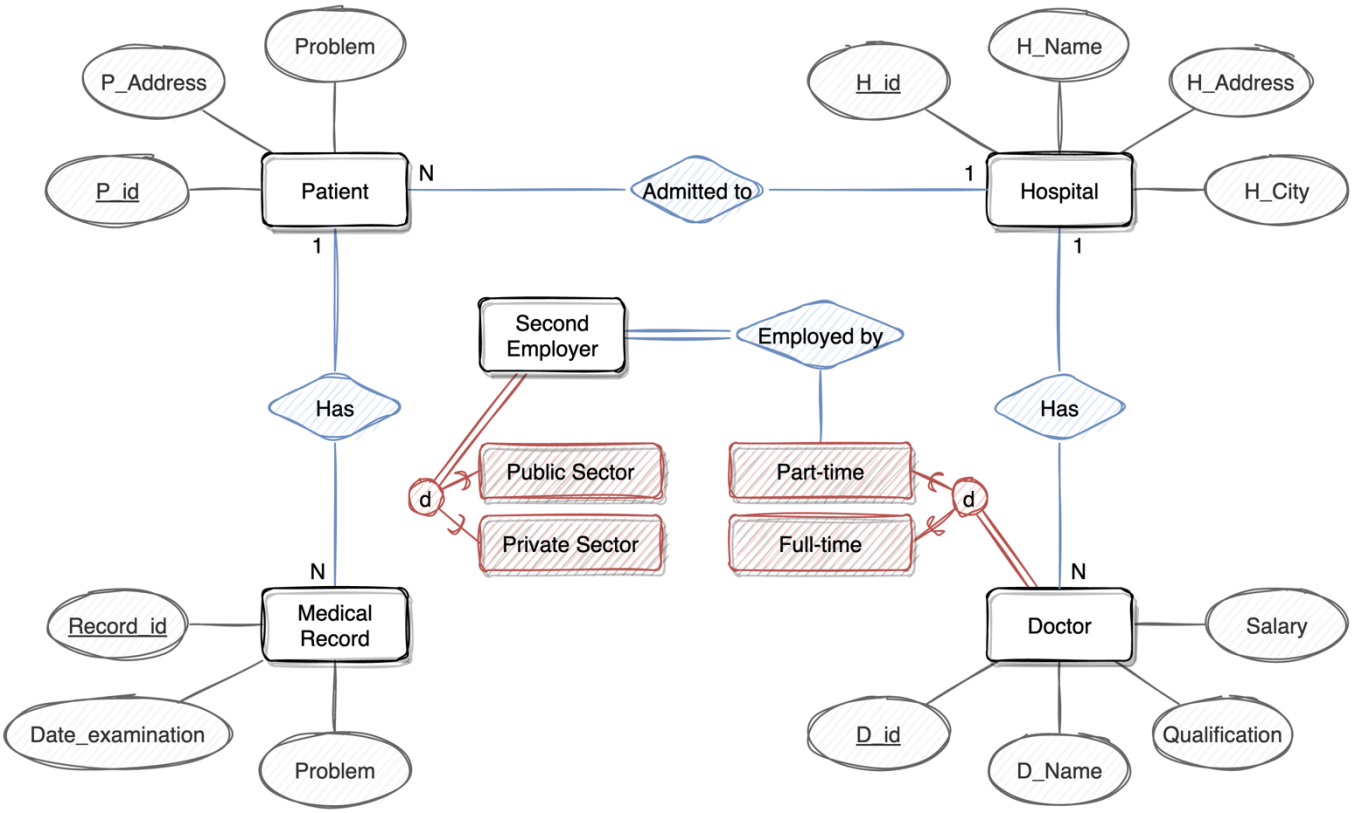

Figure 3: Revised diagram shown in Problem 2 including second employer and both doctor and employer distincts.

1. In my revised diagram, I used total distincts because a doctor has mandatory participation in being either a part-time or a full-time employee. Likewise, the second employer is mandatorily either a public sector employer or a private sector employer. Partial distincts would allow both doctor and second employer to not specialise in the subclasses and be present in the superclasses of doctor and second employer. This does not fit the scenario, so I have used total disjoints to force the entities to specialise.

   Two relevant implementation options to consider when converting EERDs to relational schemas are using superclasses with subclasses and using subclasses alone. You can use subclasses alone when your entities need to explicitly specialise. This means that they must mandatorily specialise in one or more subclasses of the original entity. However, to view all instances of the entity, you need to employ a full outer join. You can use superclasses with subclasses when your entities don't need to specialise explicitly and mandatorily. This enables entity relations to both specialise and generalise, existing outside of specialisations. This makes it easier for users to view all instances with a simple union or select query rather than a computationally expensive join query.

2. The table I have identified as the best candidate for horizontal partitioning is the 'Medical Record' table. Horizontal partitioning involves splitting the table into chunks that would now appear to be vertically stacked, so an implementation of this could look like separate tables for each year of the examination, using the 'date_examination' attribute. This would split what could be millions of records into a few tens of thousands per table. This provides the fastest access due to the reduced number of records returned per query, and the highest security because if a table is compromised, only the records within that single year will be affected. However, the regulator will be able to return all medical records if needed with a union.

```
-- Example of creating table for hospital records created in 2019.
-- Would be performed for every year present in HOSPITALRECORD.
ALTER TABLE medicalrecords DROP PRIMARY KEY,
ADD PRIMARY KEY (Record_id, Date_examination);
ALTER TABLE medicalrecords PARTITION BY RANGE (YEAR (Date_examination)) (
PARTITION mr_2018 VALUES LESS THAN (2019),
PARTITION mr_2019 VALUES LESS THAN (2020),
PARTITION mr_2020 VALUES LESS THAN (2021),
PARTITION mr_current VALUES LESS THAN MAXVALUE);
```

3. The table I have identified as the best candidate for vertical partitioning is the 'Doctor' table. Vertical partitioning involves splitting the table into separate tables in which each includes a share of the columns in the original table. An implementation of this would look like a table that holds information of the doctors themselves with the columns 'd_id', 'd_name', and 'qualification', and a table that holds information of their roles with the columns 'h_id*', and 'salary'. This would split the original table including both static and dynamic data into their respective tables. This provides the fastest access because the regulator would likely be accessing the static data like the doctors' names more frequently than the dynamic data like their salaries. However, the regulator will be able to return all columns to create the original table by creating a view in which all columns are displayed together.

```
-- Example of creating table without sensitive salary data.
-- Regulators would see the table without salaries.
CREATE TABLE doctorspublic AS (
SELECT D_id, H_id, D_name, Qualification FROM doctors);
CREATE TABLE doctorsprivate AS (
SELECT D_id, Salary FROM doctors);
DROP TABLE doctors;
```

Problem 4

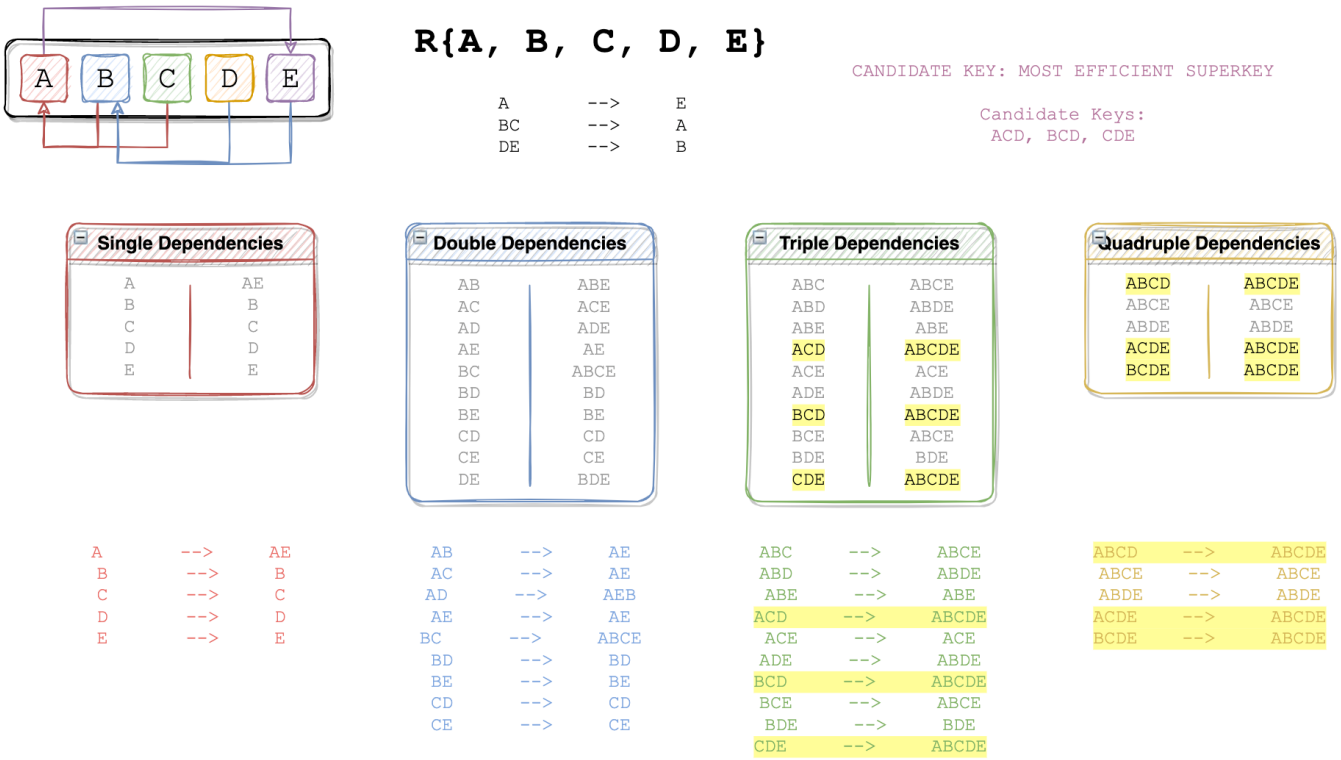

Figure 4: Determination of the candidate keys for relation R described in Problem 4 by finding closures for all sets of attributes.

1. The superkeys I have identified are ACD, BCD, CDE, ABCD, ACDE, and BCDE. However, the candidate keys I have identified are **ACD**, **BCD**, and **CDE**.
2. The relation R is not in BCNF due to the functional dependency A → E, which is also a transitive dependency because BC → A. BCNF does not allow this, so I have removed it into its own relation **{A, E}**, which now obeys BCNF. The relation R – E {A, B, C, D} violates BCNF due to the functional dependency BC → A because BC is not a key for the full set, therefore I needed to further decompose it. I did this by making BC → A into its own relation **{B, C, A}**. This left me with the remaining attributes in relation R – A, so I collated them into their own relation **{B, C, D}**. This relation follows BCNF because no functional dependencies given in the key relate, therefore it is not trivial. Furthermore, {B, C, D} is a candidate key of itself, meaning no functional or transitive dependencies exist. Finally, the final decomposition of relation R consists of three separate relations **{A, E}**, **{B, C, A}**, and **{B, C, D}**.

Word count: 859