The University Case Study

Zvi M. Kedem

Contents

1 Introduction

Our client, who does not know anything about ER (Entity-relationship) diagrams or relational database management systems asked us to build a system for maintaining its data.

Below is the description, *which on purpose*, was written so that effort is needed to understand what is going on. It is written in this way for us to practice understanding such informal descriptions so as to convert them into clear, formal, actionable specifications. The bold font does not have a significant meaning and it is there to make following the description easier.

Try to understand and build "a mental picture" of the application.

To reiterate: you are not asked to produce any drawing or implementation for this application—just think about it.

Our plans:

1. In Unit 2, from the client's narrative, we will create an Entity-Relationship diagram. After we do that, you could come back to this description.
2. In Unit 3, from the Entity-Relationship diagram built in Unit 2 we will create a relational database implementation.

2 Application: Client's Narrative

We maintain information about female **Horses** (mares). We know the **Name** of the **Horse** and it uniquely identifies the **Horse**. We also, sometimes, know the **Name** of its **Mother**, if it's in the database. A formal term for a **Mother** of a **Horse** is **Dam**. Of course, obvious biological properties are satisfied such as, a **Horse** cannot have more than one **Mother** and a **Horse** cannot be its own **Mother**. We need to think about what exactly makes (or does not make) sense. The client used to assign **Horses** for transportation but now they are just retired and are enjoying themselves.

We maintain information about **Persons**. For each **Person** we may need to maintain the properties **ID#**, **SSN**, **Name**, **DOB**, **Children**. **Name** consists of two parts: **FN** and **LN**. We do not necessarily know the value of **FN** for everybody, but we know the values of all the other attributes. No 2 **Persons** can have the same value of **SSN**. No two **Persons** can have the same value of **ID#**.

A **Person** may have 0 or more **Children**. All we need to know about the **Children** of a **Person** is their first names. No 2 **Children** of a **Person** can have the same first name. Even if a **Child** has more than 1 **Person** as a parent in our database, the **Child** is "assigned" to only one parent.

We maintain information about **Cars**. A **Car** has two attributes **VIN** and **Color**. **VIN** is like an N-Number for **Cars** and is always known, and a **Car** can be uniquely identified by it.

We maintain information about **Automobiles**. In contrast with **Cars**, these are abstract objects. An **Automobile** is a description of a type of an **Automobile** and not a specific physical object. An example would be an entry for a model of Honda. Such an entry would store **Model** with the value of "Honda CR", **Year** with the value of "2018", and **Weight** with the value of "3358". All these properties of an entry are always known and for each pair of values of **Model** and **Year** there is exactly one value of **Weight**.

We maintain information about which **Persons Like** which **Automobiles**.

Typeset: 2021-09-11T13:49:19Z MD5: A8C679E180A545C40455DC3872A9C294

Each **Car** is related through **Type** with exactly 1 **Automobile**. Through this relation we can find out for each **Car** what **Model Year** it is and what its **Weight** is.

A **Person** has at least 2 **Cars** but no 2 **Persons** can have the same **Car**. We keep information about that, but we may also keep information about the **Date** on which a **Car** was acquired by a **Person** who currently has it.

Some of the **Persons** are **Students** and some are **Professors**. A **Student** has an attribute **GPA**. A **Professor** has an attribute **Salary**, which is always known. **GPA** will need to be computed based on the student performance in some known way, to be specified later in the implementation process.

We maintain the following information about **Courses**: **C#** and **Title**. **C#** identifies a specific **Course** and is always known. **Title** is also always known.

We need to maintain required **Prerequisites**. For example, **C#** 101 might be a **Prerequisite** for **C#** 102. Some sensible restrictions on **Prerequisites** need to be specified. For example, we cannot have all the following together: **C#** 101 is **Prerequisite** for **C#** 102, **C#** 102 is **Prerequisite** for **C#** 103, and **C#** 103 is **Prerequisite** for **C#** 101, as that would be a circular dependency. We need to think about what exactly makes (or does not make) sense.

We maintain information about **Books**. For each **Book**, **Title** and **Author** are always known and together identify that **Book**.

We maintain information about required **Books**. A **Requirement** for a **Book** specifies which **Professor** required it for which **Course**.

Courses are listings in a university catalog. When an **Offering** of a **Course** is taught, a **Section** is generated for the **Course**. For example your **Section** is one of 001, 002, or 003 in Fall of 2021 for the **C#** CSCI-GA 2433 or DS-GA 2433. A **Section** has natural attributes **Year**, **Semester**, and **Sec#**, which are always known and which are needed for its identification. It also has attribute **Room**, which is sometimes known. No **Course** is listed in a catalog until at least one **Section** has been offered.

We keep information about which **Students Took** which **Sections**. For each enrollment the **Grade** may be known. A **Student** may have a known **GPA**, which is computed by adding all the known numeric **Grades**, dividing by the number of **Sections** that the **Student Took**. A **Section** has between 3 and 50 **Students**.

We keep information which **Professors Taught** which **Sections**. Such a teaching assignment may be **Monitored** by at most 1 **Professor** but a **Professor** cannot be **Monitored** by itself.

