CSCI 3410 - Database Systems

Lecture Notes

Clément Aubert

August 8, 2022 (05:47:16 PM)

Contents

List of Problems

Preamble

Disclaimer

As of August 2022, the main author of those notes (Dr. Aubert[1]) is not scheduled to teach CSCI 3410 - Database Systems in the forseeable future. As a result, those notes are archived.

Please, also note that pandoc-include-code[2] and pandoc-numbering[3], required to compile those notes, are not compatible with the current version of pandoc[4] (cf. the INSTALL.md[5] instructions). As a result, compiling those notes will require increasing version tinkering.

How to Use This Guide

How to Read This Guide

These lecture notes are written in an elusive style: they are a support for the explanations that will be made at the board. Reading them before coming to the lecture will help you getting a sense of the next topic we will be discussing, but you may sometimes have trouble deciphering their … *unique* style.

On top of the notes, you will find in this document:

- References, at the very end of this document,
- and for each chapter,
    - A list of additional resources,
    - A list of short exercises,
    - Solution to those exercises,
    - A list of problem,
    - Solution to some of those problems.

Any feedback is greatly appreciated. Please refer to https://spots.augusta.edu/caubert/db/ln/README.html#contributing for how to contribute to those notes. The syllabus is at https://spots.augusta.edu/caubert/db/, and the webpage for those notes is at https://spots.augusta.edu/caubert/db/ln/.

Please, refer to those notes using this entry (Aubert 2019):

[1] https://spots.augusta.edu/caubert/
[2] https://github.com/owickstrom/pandoc-include-code
[3] https://github.com/chdemko/pandoc-numbering
[4] https://pandoc.org/installing.html
[5] ../install/INSTALL.html#speed-run

```
@report{AubertCSCI3410-DatabaseSystems,
        author={Aubert, Clément},
        title={CSCI 3410 - Database Systems},
        url={https://spots.augusta.edu/caubert/db/ln/},
        urldate={2019-11-03},
        year={2019},
        institution={{School of Computer and Cyber Sciences,
         ↪  Augusta University}},
        location={Augusta, Georgia, USA},
        langid={en},
        type={Lecture notes}
}
```

entry.bib[6]

How to Access the Code in This Guide

There are four way to access the code shared in those lecture notes:

1. You can simply copy-and-paste it from the document and use it as it is.
2. You can browse the source code of the code snippets at https://rocketgit.com/user/caubert/CSCI_3410/source/tree/branch/master/tree/notes/code to download them directly.
3. You can clone the repository containing the notes, figures and code snippets to have a local copy of it. You can find instructions on how to do that at https://spots.augusta.edu/caubert/db/ln/README.html. Instructions on how to compile those notes and how to contribute are linked from this document, if you are curious.
4. You can use the links enclosed in the document.

For this latter aspect, note that some portion of code starts with a path in comment, and are followed by a link, like so:

```sql
/* code/sql/HW_HelloWorld.sql */
SELECT "Hello World!";
```

HW_HelloWorld.sql[7]

This means that this code can be found at

https://rocketgit.com/user/caubert/CSCI_3410/source/tree/branch/master/
blob/notes/**code/sql/HW_HelloWorld.sql**[8]

and that you can click the link below the code directly to access it[9].

The `SQL` code frequently starts with

```sql
DROP SCHEMA IF EXISTS HW_NAME_OF_SCHEMA;
CREATE SCHEMA HW_NAME_OF_SCHEMA;
USE HW_NAME_OF_SCHEMA;
```

[6] https://rocketgit.com/user/caubert/CSCI_3410/source/tree/branch/master/blob/notes/bib/entry.bib
[7] https://rocketgit.com/user/caubert/CSCI_3410/source/tree/branch/master/blob/notes/code/sql/HW_HelloWorld.sql
[8] https://rocketgit.com/user/caubert/CSCI_3410/source/tree/branch/master/blob/notes/code/sql/HW_HelloWorld.sql
[9] This feature was actually implemented by a student![10].

This parts starts by deleting the schema `HW_NAME_OF_SCHEMA` if it exists, then create and use it: it allows the code to run independently of your installation. It needs to be used with care, though, since it would delete everything you have in the `HW_NAME_OF_SCHEMA` schema before re-creating it, but empty.

Finally, the comments

```
-- start snippet something
```

and

```
-- end snippet something
```

can be ignored, as their are an artifice from pandoc-include-code[11] to select which portion of the code to display in those notes.

Planned Schedule

A typical (meeting twice a week, ±17 weeks, ±30 classes) semester is divided as follows:

- Lecture 1: Presentation and Syllabus[12]
- Lecture 2: Introduction
- Lecture 3–5: The Relational Model
- Lecture 6–9: The SQL Programming Language
- Lecture 10–11: Review session and Exam #1
- Lecture 12: Introduction to High-Level Design
- Lecture 13–15: Entity-Relationship Model
- Lecture 16: ER-to-Relational Models Mapping
- Lecture 17–20: Guidelines and Normal Form
- Lecture 21–22: Unified Modeling Language Diagram
- Lecture 23–24: Review session and Exam #2
- Lecture 25–28: Database Applications
- Lecture 29–30: Presentation of NoSQL

For information purposes, an indication like this:

---

marks the (usual) separation between two lectures.

Exams Yearbooks

To give you a sense of what you will be asked during the exams, quizzes and projects, or simply to practise, please find below the exams given previous semesters, in reverse chronological order. The quizzes are not indicated, but were generally a mix of up to five exercises and one problem from the relevant chapter(s).

[11] https://github.com/owickstrom/pandoc-include-code
[12] https://spots.augusta.edu/caubert/db/

Fall 2021

- Project #1: A variation on Problem 3.18 (A simple database for vaccines)
- Exam #1:
  - A variation on Problem 3.13 (A simple database for books)
  - A variation on Problem 2.2 (Design a relational model for a cinema company)
  - A variation on Problem 2.1 (Find a candidate key for the CLASS relation): the students were given the four candidate keys detailled in the solution and asked to justify when they could be valid candidate keys.
  - A series of small questions, Exercise 3.14, Exercise 3.19, Exercise 3.30 and Exercise 3.32

Spring 2021

- Project #1: Problem 3.18 (A simple database for vaccines)
- Exam #1:
  - A series of small commands to explain, like Exercise 3.7, Exercise 3.40 and Exercise 3.41
  - Problem 3.21 (Improving a Relational Model for a Printing Station)
  - Problem 3.22 (Write select queries for a (third!) variation of the COMPUTER table)
- Exam #2:
  - A series of questions about database application and security
  - Problem 4.7 (ER diagram for Undergraduate Conference)
  - Converting the ER diagram in *the solution* of Problem 4.4 (ER diagram for job and offers) into a relational model
  - Problem 4.25 (Normal form for the GRADE_REPORT Relation)
- Final:
  - Problem 4.16 (ER Diagram for Friendship, Dishes and Pets)
  - Problem 4.24 (Perfecting a Relational Model for File Systems)
  - Problem 7.2 (From MongoDB to SQL)
  - Problem 5.1 (Classes Relationships)

Fall 2020

- Project #1: Problem 3.17 (A simple database for capstone projects)
- Exam #1:
  - Problem 3.20 (A database for research fundings)
  - Problem 2.6 (Relational model for a pet shelter)
- Exam #2:
  - Problem 4.35 (From Business Statement to Functional Dependencies to Normal Form – TEACHING)
  - A variation on Problem 4.25 (Normal form for the GRADE_REPORT Relation)
  - Problem 4.5 (ER diagram for cellphones)
  - Five small exercises about data base application.

- Final:
  - Problem 4.15 (Studying an Accident ER Diagram)
  - (rest to come)

Spring 2020

Due to the Covid-19 pandemic, only one exam took place, and the final exam was taken remotely on D2L. A second project, more ambitious, was also asked from the students, and accounted for a large portion of their grade.

- Project #1: Problem 3.16 (A simple database for authors of textbooks)
- Exam #1:
  - Problem 3.19 (A database for residencies)
  - Problem 2.5 (Relational model for an auction website)
- Final:
  - Problem 3.10 (Write select queries for the SocialMedia schema), where the last query was treated as a bonus, due to its difficulty.
  - Problem 7.1 (Explaining NoSQL)
  - Problem 4.28 (Normal form of the DELIVERY relation)
  - Problem 4.6 (Incorrect ER diagram)

Fall 2019

- Exam #1:
  - Problem 3.15 (A simple database for published pieces of work)
  - Three exercises (Exercise 3.14, Exercise 3.32 but for the `OR` operator, and Exercise 3.30)
  - Problem 2.4 (Relational model for universities)
- Exam #2:
  - Problem 4.37 (From Business Statement to ER Diagram to Relational Model – A Network of Libraries)
  - Problem 4.22 (From business statement to dependencies, ROUTE)
  - Problem 5.3 (A GUEST Java Program)
  - Problem 4.31 (PRINT relation in third normal form)
- Final:
  - Problem 7.4 (ER Diagram from XML File – Award)
  - Problem 4.18 (Normal form of a simple relation)
  - Problem 4.23 (From business statement to dependencies, ISP)
  - Problem 3.14 (A database for website certificates)
  - Three small exercises about security (Exercise 6.2, Exercise 6.4, Exercise 6.5)

Spring 2019

- Exam #1:
  - Problem 3.13 (A simple database for books)
  - Five exercises (Exercise 1.5, Exercise 2.13, Exercise 3.32, Exercise 2.15, Exercise 3.39)
  - A variation on Problem 2.3 (Design a relational model for bills)
- Exam #2:
  - Problem 4.4 (ER diagram for job and offers)
  - Problem 4.12 (From ER diagram to Relational model – RECORD)
  - Problem 4.19 (Normal form of a SCHEDULE relation)
  - Problem 4.30 (Normal form of the MESSAGE relation)
- Final:
  - A variation on Problem 7.1 (Explaining NoSQL)
  - Problem 4.27 (Normal form of the BOOK relation)
  - Problem 4.34 (A Relation for Network Cards)
  - Problem 3.11 (Write select queries for a variation of the COMPUTER table)
  - A variation on Problem 4.37 (From Business Statement to ER Diagram to Relational Model – A Network of Libraries)
  - Five exercises from the Database Application chapter.

Spring 2018

- Exam #1:
  - Problem 2.2 (Design a relational model for a cinema company), except that I gave some of the relations and attributes, to help getting started with the problem.
  - Problem 4.1 (Design for your professor)
  - Problem 3.6 (TRAIN table and more advanced `SQL` coding)
  - Problem 3.7 (Read, correct, and write `SQL` statements for the COFFEE database)
- Exam #2:
  - A variation on Problem 4.29 (Normal form of the CONTACT relation)
  - A variation on Problem 4.32 (CONSULTATION relation: justification, primary key and normal form)
  - Problem 3.12 (Improving a role-playing game with a relational model)
  - A variation on Problem 4.39 (From business statements to dependencies – KEYBOARD)
  - Problem 4.13 (ER-to-Relation mapping for Country)
- Final:
  - Take the relational model of the solution of Problem 2.2 (Design a relational model for a cinema company), and “reverse-engineer” it to obtain a ER diagram (this problem was probably too hard).
  - Six small exercises (Exercise 4.37, Exercise 7.2, Exercise 7.3, Exercise 7.4, Exercise 7.5)
  - Problem 4.20 (Normalizing the FLIGHT relation)
  - A variation on Problem 4.14 (From business statements to ER diagram – UNIVERSITY)
  - A variation on Problem 4.40 (From UML to relational model – DRIVER): students were asked to draw the ER diagram for that schema.

Fall 2017

- Exam #1:[13]
  - Six small exercises (Exercise 1.11, Exercise 2.5, Exercise 2.9, Exercise 3.6, Exercise 3.8 and Exercise 3.12)
  - (A variation on) Problem 2.1 (Find a candidate key for the CLASS relation)
  - Problem 2.2 (Design a relational model for a cinema company)
  - A variation on (Elmasri and Navathe 2010, Exercise 3.11), (Elmasri and Navathe 2015, Exercise 5.11)
  - Problem 3.6 (TRAIN table and more advanced `SQL` coding)
- Exam #2:
  - Six small exercises, (Exercise 4.10, Exercise 4.16, Exercise 4.45, Exercise 4.42, Exercise 4.7, Exercise 4.29)
  - Problem 4.14 (From business statements to ER diagram – UNIVERSITY)
  - Problem 4.21 (From business statement to dependencies, BIKE)
  - Problem 4.29 (Normal form of the CONTACT relation)
  - A variation on Problem 4.13 (ER-to-Relation mapping for Country)
- Final:
  - A variation on (Exercise 5.20)
  - A variation on Problem 3.7 (Read, correct, and write `SQL` statements for the COFFEE database)
  - A variation on Problem 4.40 (From UML to relational model – DRIVER): students were asked to draw the ER diagram for that schema.
  - Problem 4.27 (Normal form of the BOOK relation)
  - Problem 4.32 (CONSULTATION relation: justification, primary key and normal form)

Typesetting and Acknowledgments

The source code for those notes is hosted at rocketgit[14], typeset in markdown[15], and then compiled using pandoc[16] and multiple filters (pandoc-numbering[17], the citeproc library[18], pandoc-include-code[19]). The drawings use various LaTeX[20] packages, including PGF, TikZ[21], tikz-er2[22], pgf-umlcd[23] and tikz-dependency[24]. The help from the TeX - LaTeX Stack Exchange[25] community greatly improved this document. The underline[26] text is obtained using YayText[27], the uni-

[13] This exam was probably a bit too long, but students managed it pretty well.
[14] https://rocketgit.com/user/caubert/CSCI_3410
[15] https://commonmark.org/
[16] https://pandoc.org/
[17] https://github.com/chdemko/pandoc-numbering
[18] https://github.com/jgm/citeproc
[19] https://github.com/owickstrom/pandoc-include-code
[20] https://www.latex-project.org/
[21] https://sourceforge.net/projects/pgf/
[22] https://bitbucket.org/pavel_calado/tikz-er2/raw/da9f9f7f169647cad6d91df7975400b1605ae67a/tikz-er2.sty
[23] https://github.com/pgf-tikz/pgf-umlcd/
[24] https://ctan.org/pkg/tikz-dependency
[25] https://tex.stackexchange.com/
[26] For technical reasons, underlined words cannot be searched in the document.
[27] https://yaytext.com/underline/

code symbols are searched in the "Unicode characters and corresponding LaTeX math mode commands"[28]. Finally, the `pdf` version of the document uses Linux Libertine fonts[29], the `html` version uses Futura[30].

Those lecture notes were created under an Affordable Learning Georgia[31] Mini-Grant for Ancillary Materials Creation and Revision[32] (Proposal M71[33]).

Those lecture notes have greatly benefited from the contributions of many students, included but not limited to Crystal Anderson, Bobby Mcmanus, Minh Nguyen and Poonam Veeral. Additionally, (Redacted), Mark Holcomb, Assya Sellak, Sydney Strong and Patrick Woolard helped smash some bugs in the tools used to produce this document.

Please refer to https://spots.augusta.edu/caubert/db/ln/README.html#authors-and-contributors for a detail of the contributions.

Resources

You can find at the end of this document the list of references, and some particular resources listed at the beginning of each chapter. Let me introduce some of them:

- (Elmasri and Navathe 2010) and (Elmasri and Navathe 2015) are two editions of an excellent and detailled book on Databases. It is commonly used, cover almost every aspect in a fairly accessible way.
- (Watt and Eng 2014) is an open-source, cost-free textbook on Database design that can be of good support.
- (Sadalage and Fowler 2012) and (Sullivan 2015) are two textbooks on the NoSQL approach that are short and good introductions.
- To get started on Java and how it interfaces with databases, (Gaddis 2014) is a good introduction.
- awesome-mysql[34] is a "curated list of awesome MySQL free and opensource software, libraries and resources" that is definitely worth checking out. Among other resources, there is this bank of SQL programming exercises[35].

[28] http://milde.users.sourceforge.net/LUCR/Math/unimathsymbols.html
[29] http://libertine-fonts.org/
[30] https://en.wikipedia.org/wiki/Futura_(typeface)
[31] https://www.affordablelearninggeorgia.org/
[32] https://www.affordablelearninggeorgia.org/about/r13_grantees
[33] https://affordablelearninggeorgia.org/documents/M71_Augusta_Aubert.pdf
[34] https://github.com/shlomi-noach/awesome-mysql#readme
[35] https://github.com/XD-DENG/SQL-exercise

Those resources are listed as complements, but it is not require to read them to understand the content of those notes. (Watt and Eng 2014) –being available free of charge– is more descriptive than the current notes, and as such can constitutes a great complement. Unfortunately, it lacks some technical aspects, and the database program aspect is not discussed in detail.

Copyright

[36] https://creativecommons.org/licenses/by/4.0/

1 Introduction

Resources

- (Elmasri and Navathe 2010, ch. 1.1–1.6)
- (Elmasri and Navathe 2015, ch. 1.1–1.6)
- (Watt and Eng 2014, ch. 2–3)

1.1 The Need for a Specialized Tool

There is a good chance that any programming language you can think of is Turing complete[1]. Actually, even some of the extremely basic tools you may be using may be Turing complete[2]. However, being complete does not mean being good at any task: it just means that any computable problem can be solved, but does not imply anything in terms of efficiency, comfort, or usability.

In theory, pretty much any programming language can be used to

- Store, retrieve and update data,
- Have accessible catalog describing the metadata,
- Support transactions and concurrency,
- Support authorization of access and update of data,
- Enforce constraints.

But to obtain a system that is fast in reading and writing on the disk, convenient to search in the data, and that provides as many "built-in" tools as possible, one should use a specialized tool.

In those lecture notes, we will introduce one of this tool–the `SQL` programming language– and the theory underneath it–the relational model–. We will also observe that a careful design is a mandatory step before implementing a catalog, and that how good a catalog is can be assessed, and introduce the tools to do so. Finally, we will discuss how an application interacting with a database can be implemented and secured, and the alternatives to `SQL` offered by the NoSQL approach, as well as the limitations and highlights of both models.

1.2 What is a Database?

A database (DB) is **a collection of related data**.

It has two components, the *data* (= information, can be anything, really) and the *management* (= logical organization) of the data, generally through a **D**ata**b**ase **M**anagement **S**ystem.

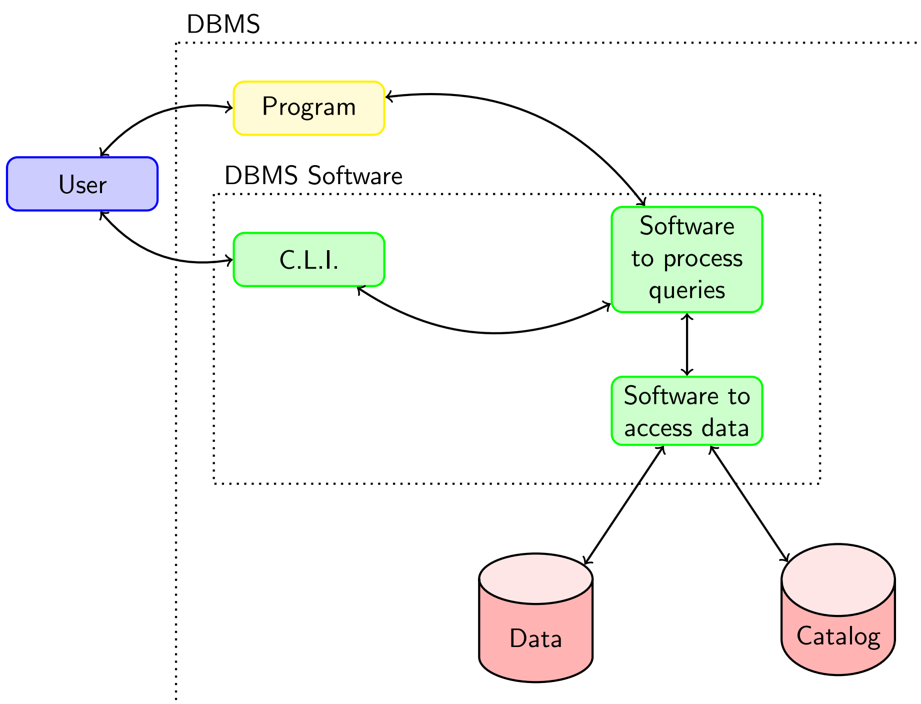
A database

[1] https://en.wikipedia.org/wiki/Turing-completeness
[2] https://www.gwern.net/Turing-complete

1. Represents a mini-world, a "Universe of Disclosure" (UoD).
2. Is logically coherent, with a meaning.
3. Has been populated for a purpose.

The mini-world is the part of the world, or universe, that will be represented in the database: as we can not represent the whole universe (every position of every atom at any given moment since the big-bang!), we must agree on what "slice" of it we should represent. Typically, a data-base designed to help in calculating students' grades will include students' names, transcript, classes taken, etc., but certainly not their height, favorite color or where they usually seat in class: altough all of this information is "part of the universe", we will not need it and decide to exclude it from our data.

A DBMS has multiple components, as follows:



Note that

- The program can be written in any language, be a web interface, etc. It is sometimes part of the software shipped with the DBMS, but not necessarily (you can, and we will, develop your own program to interact with the DBMS).
- Most DBMS software includes a Command-Line Interface (CLI).
- The catalog (or schema, meta-data[3]) contains the description of how the data is stored, i.e., the datatypes, nature of the attributes, etc.
- Sometimes, catalog and data are closer than pictured (you can have "self-describing meta-data", that is, they cannot be distinguished), this is typically the case in some of the NoSQL approaches.

[3]The term "meta-data" has numerous definition ("data about the data"): we use it here to refer to the description of the organization of the data, and not e.g. to statistical data about the data.

1.3 Database Management System (DBMS)

A DBMS contains a *general purpose* software that is used to

1. Define (= datatype, constraints, structures, etc.)
2. Construct / Create the data (= store the data)
3. Manipulate / Maintain (= change the structure, query the data, update it, etc.)
4. Share / Control access (= among users, applications)

You can think of a tool to

1. Specify a storage unit,
2. Fill it,
3. Allow to change its content, as well as its organization,
4. Allow multiple persons to access all or parts of it at the same time.

1.4 How Are the Tasks Distributed?

Exactly like a program can have

- clients, that specify the requirements,
- designers, that define the overall architecture of a program,
- programmers, that implement the details of the program,
- testers, that make sure the program is free of bugs, and
- users, that actually use the program,

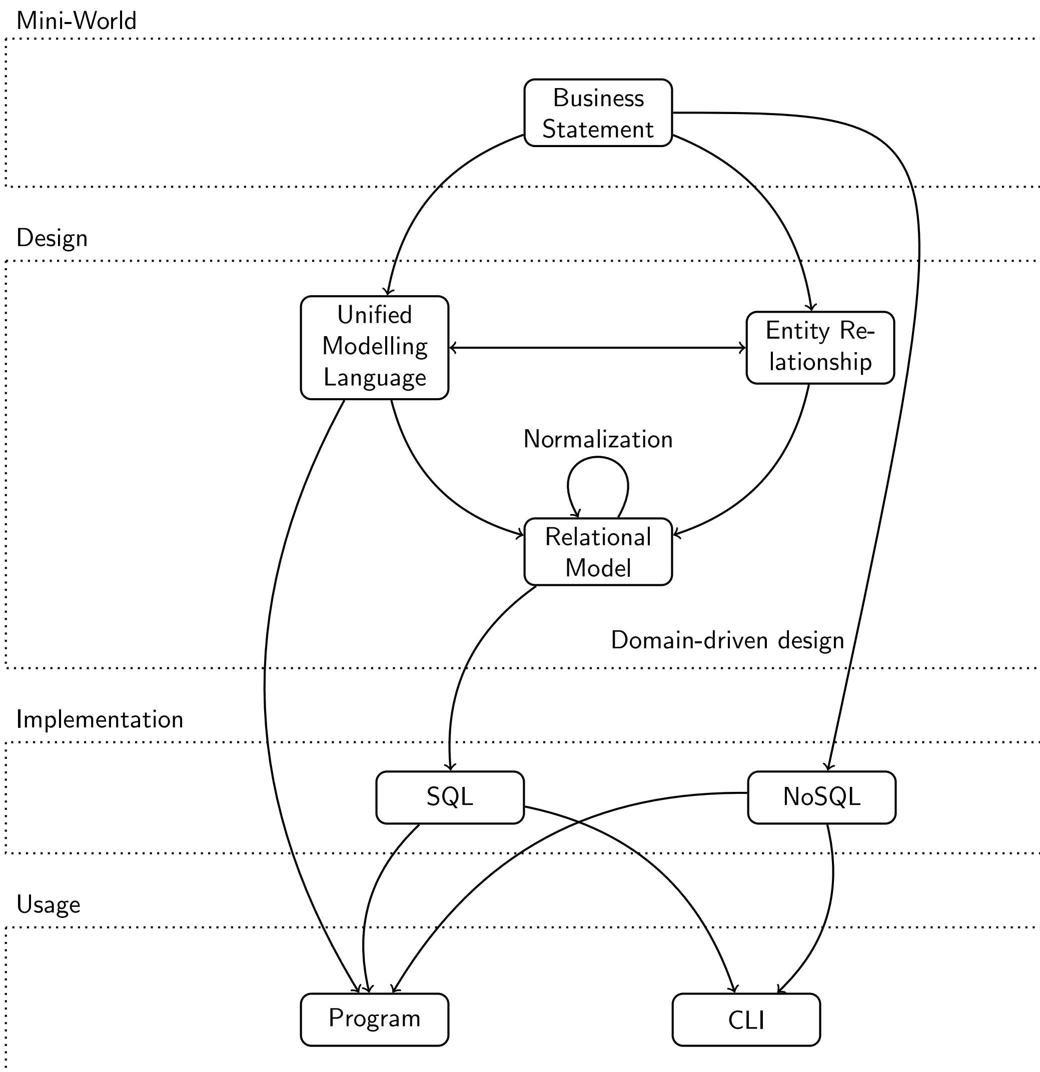
a DBMS offers multiple (sub)tasks and can be interacted with different persons with different roles.

Role	Task
Client	Specify the business statement, the specifications
DB Administrator	Install, configure, secure and maintain up-to-date the DBMS
Designer	Lay out the global organization of the data
Programmer	Implement the database, work on the programs that will interface with it
User	Provide, search, and edit the data (usually)

In those lecture notes, the main focus will be on design and implementation, but we will have to do a little bit of everything, without forgetting which role we are currently playing.

1.5 Life of a Project

From the business statement to the usage, a project generally follows one of this path:



Note that reverse-engineering can sometimes happen, i.e., if you are given a poor implementation and want to extract a relational model from it, to normalize it.

1.6 An Example

Let us consider the following:

STUDENT

Name	Student_number	Class	Major
Morgan	18	2	IT

Name	Student_number	Class	Major
Bob	17	1	CS

COURSE

Course_name	Course_number	Credit_hours	Department
Intro. to CS	1301	4	CS
DB Systems	3401	3	CS
Principles of Scripting and Automation	2120	3	AIST

SECTION

Section_identifier	Course_num	Semster	Year	Instructor
2910	1301	Fall	2019	Kate
9230	2103	Spring	2020	Todd

GRADE_REPORT

Student_number	Section_identifier	Grade
17	2910	A
18	2910	B

PREREQUISITE

Course_number	Prerequisite_number
2120	1301
1302	1301

You can describe the structure as a collection of relations, and a collection of columns:

RELATIONS

Relation Name	Number of Columns
STUDENT	4
COURSE	4
SECCTION	5
GRADE_REPORT	3
PREREQUISITE	2

COLUMNS

Column Name	Datatype	Belongs to relation
Name	String	STUDENT
Student_number	Integer	STUDENT
Class	String	STUDENT
Major	String	STUDENT
Course_name	String	COURSE
Course_number	Integer	COURSE
Credit_hours	Integer	COURSE
Department	String	COURSE
…	…	…
Prerequisite_number	Integer	PREREQUISITE

1.6.1 Structure

- Database structure and records, 5 files (=collection of records), each containing data records of the same type, stored in a persistent way.
- Each record has a structure, different data elements, each has a data type.
- Records have relationships between them (for instance, you expect the Course_number of PREREQUISITE to occur as a Course_number in COURSE).

1.6.2 Interactions

- This organization will allow some interactions. For instance, we can obtain the answer to questions like

  > "What is the name of the course whose number is 1301?",
  > "What courses is Kate teaching this semester?",
  > "Does Bob meets the pre-requisite for 2910?"

  Note that this last query is a bit different, as it forces us to look up information in multiple relations.

- We should also be able to perform updates, removal, addition of records in an efficient way (using auxiliary files (indexes), optimization).

- Finally, selection (for any operation) requires care: do we want all the records, some of them, exactly one?

1.6.3 Organization

Why are the files separated like that? Why do not we store the section with the course with the students? For multiple reasons:

- To **avoid redundancy** ("data normalization"), or having it controlled,
- To controle multiple **levels of access** (multiple user interface),
- Without sacrificing the **usability**!

In separating the datae, we also need to remember to be careful about **consistency** and **referential integrity**, which is a topic we will discuss in detail.

1.6.4 How Is a Database Conceived?

1. Specification and analysis. "Each student number will be unique, but they can have the same name. We want to access the letter grade, but not the numerical grade", etc. This gives the businnes statement.
2. Conceptual design
3. Logical design
4. Physical design

There is a gradation, from really abstract specification that is easy to modify, to more solidified description of what needs to be coded. When we will be discussing high-level models, we will come back to those notions. The global idea is that it is easier to move things around early in the conception, and harder once everything is implemented.

1.7 Characteristics of the Database Approach

1. A database is more than just data: it also contains a complete description of the structure and constraints. We generally have a catalog (a.k.a. the meta-data, the schema) and the data (we can also have self-describing data, where meta-data and data are interleaved, but note that both are still present).
2. Data-abstraction: A DBMS provides a conceptual representation, and hides implementation details. This implies that changing the internals of the database should not require to change the application (the DBMS) or the way any of the client (program, or CLI) was interacting with the data.
3. Support of multiple views of the data: view is a subset of the database, or virtual data.
4. Sharing and multiuser transaction processing: concurrency control using transactions (= series of instructions that is supposed to execute a logically correct database access if executed in its entirety). Isolation, atomicity (all or nothing): cf. the ACID principles.

Exercises

Exercise 1.1 What is the difference between a database and the meta-data of the database?

Exercise 1.2 Is a pile of trash a database? Why, or why not?

Exercise 1.3 Define the word "miniworld".

Exercise 1.4 Expand the acronym "DBMS".

Exercise 1.5 Name two DBMS.

Exercise 1.6 Name the four different kinds of action that can be performed on data.

Exercise 1.7 Assign each of the following task to one of the "character" (administrator, client, etc.) we introduced:

Task	Assigned to
Install a DBMS on a server.	
Sketch the schema so that the data will not be redundant.	
Write client-side application that uses the DBMS API.	

Task	Assigned to
Establish the purpose of the database.	

Exercise 1.8 List some of the tasks assigned to the Database Administrator.

Exercise 1.9 Why do DBMS include concurrency control?

Exercise 1.10 Do I have to change my DBMS if I want to change the structure of my data?

Exercise 1.11 What is independence between program and data? Why does it matter?

Exercise 1.12 Assume that I have a file where one record corresponds to one student. Should the information about the classes a student is taking (e.g. room, instructor, code, etc.) being stored in the same file? Why, or why not?

Exercise 1.13 Which one comes first, the physical design, the conceptual design, or the logical design?

Exercise 1.14 What is a virtual data? How can I access it?

Solution to Exercises

Solution 1.1 The data is the information we want to store, the meta-data is its organization, how we are going to store it. Meta-data is information about the data, but of no use on its own.

Solution 1.2 No, because it lacks a logical structure.

Solution 1.3 The mini-world is the part of the universe we want to represent in the database. It is supposed to be meaningful and will serve a purpose.

Solution 1.4 Database Management System

Solution 1.5 Oracle RDBMS, IBM DB2, Microsoft SQL Server, MySQL, PostgreSQL, Microsoft Access, etc., are valid answers. Are not valid "SQL", "NoSQL", "Relational Model", or such: we are asking for the names of actual softwares!

Solution 1.6 The four actions are:

- Add / Insert
- Update / Modify
- Search / Query
- Delete / Remove

Solution 1.7 We can have something like:

Task	Assigned to
Install a DBMS on a server.	Administrator, IT service
Sketch the schema so that the data will not be redundant.	Designer
Write client-side application that uses the DBMS API.	Programmer, Developer
Establish the purpose of the database.	Client, business owner

Solution 1.8 The database administrator is in charge of installing, configuring, securing and keeping up-to-date the database management system. They also control the accesses and the performance of the system, troubleshoot it, and create backup of the data.

Solution 1.9 DBMS have concurrency control to ensure that several users trying to update the same data will do so in a controlled manner. It is to avoid inconsistency to appear in the data.

Solution 1.10 Normally no, data and programs are independent. But actually, this is true only if the model does not change: shifting to a "less structured model", e.g., one of the NoSQL models, can require to change the DBMS.

Solution 1.11 The application should not be sensible to the "internals" of the definition and organization of the data. It matters because having this independence means that changing the data will not require to change the programs.

Solution 1.12 If we were to store all the information about the classes in the student records, then we would have to store it as many time as its number of students! It is better to store it in a different file, and then to "link" the two files, to avoid redundancy.

Solution 1.13 The conceptual design.

Solution 1.14 It is a set of information that is derived from the database but not directly stored in it. It is accessed through queries. For instance, we can infer the age of a person if their date of birth is in the database, but strictly speaking the age is not an information stored in the database.

Problems

Problem 1.1 ***(Define a database for `CAMPUS`)*** Define a `CAMPUS` database organized into three files as follows:

- A `BUILDING` file storing the name and GPS coordinates of each building.
- A `ROOM` file storing the building, number and floor of each room.
- A `PROF` file storing the name, phone number, email and room number where the office is located for each professor.

Pb 1.1 – Question 1 A database catalog is made of two part: a table containing the relations' name and their number of columns, and a table containing the columns' name, their data type, and the relation to which they belong. Refer to the example we made previously or consult, e.g., (Elmasri and Navathe 2010, fig. 1.3) or (Elmasri and Navathe 2015, fig. 1.3). Write the database catalog corresponding to the `CAMPUS` database.

Pb 1.1 – Question 2 Invent data for such a database, with two buildings, three rooms and two professors.

Pb 1.1 – Question 3 Answer the following, assuming all the knowledge you have of the situation comes from the `CAMPUS` database, which is an up-to-date and accurate representation of its miniworld:

1. Is it possible to list all the professors?
2. Is it possible to tell in which department is a professor?

3. Is it possible to get the office hours of a professor?
4. Is it possible to list all the professors whose offices are in the same building?
5. Is it possible to list all the rooms?
6. If a new professor arrives, and has to share his office with another professor, do you have to revise your database catalog?
7. Can you list which professors are at the same floor?
8. Can you tell which professor has the highest evaluations?

Solutions to Selected Problems

Solution to Problem 1.1 (Define a database for CAMPUS) Pb 1.1 – Solution to Q. 1

The database catalog should be similar to the following:

RELATIONS

Relation name	Number of columns
BUILDING	3
ROOM	3
PROF	4

COLUMNS

Column name	Datatype	Belongs to relation
Building_Name	Character(30)	Building
GPSLat	Decimal(9,6)	Building
GPSLon	Decimal(9,6)	Building
Building_Name	Character(30)	ROOM
Room_Number	Integer(1)	ROOM
Floor	Integer (1)	ROOM
Prof_Name	Character (30)	PROF
Phone	Integer (10)	PROF
Email	Character (30)	PROF
Room_Number	Integer (1)	PROF

Pb 1.1 – Solution to Q. 2

For the data, you could have:

- For the BUILDING file, we could have:

```
(Allgood Hall, 33.47520, -82.02503)
(Institut Galilé,  48.959001, 2.339999)
```

- For the ROOM file, we could have:

```
(Allgood Hall, 128, 1)
(Institut Galilé, 205, 3)
(Allgood Hall, 228, 2)
```

- For the `PROF` file, we could have:

```
(Aubert, 839401, dae@ipn.net, 128)
(Mazza, 938130, Dm@fai.net, 205)
```

Pb 1.1 – Solution to Q. 3

If everything we knew about the campus came from that database, then

1. Yes, we could list all the professors.
2. No, we could not tell in which department is a professor.
3. No, we could not get the office hours of a professor.
4. Yes, we could list all the professors whose offices are in the same building.
5. Yes, we could list all the rooms.
6. If a new professor arrives, and has to share his office with another professor, we would not have to revise our database catalog (it is fine for two professor to have the same room number, in our model).
7. Yes, we could list which professors are at the same floor.
8. No, we could not tell which professor has the highest evaluations.

2 The Relational Model

Resources

- (Elmasri and Navathe 2010, ch. 3), (Elmasri and Navathe 2015, ch. 5), including the exercises (look at the exercises 15 and 16, for instance).
- The wikipedia page for Relational model[1] and the category "Relational database management systems"[2].

2.1 Concepts

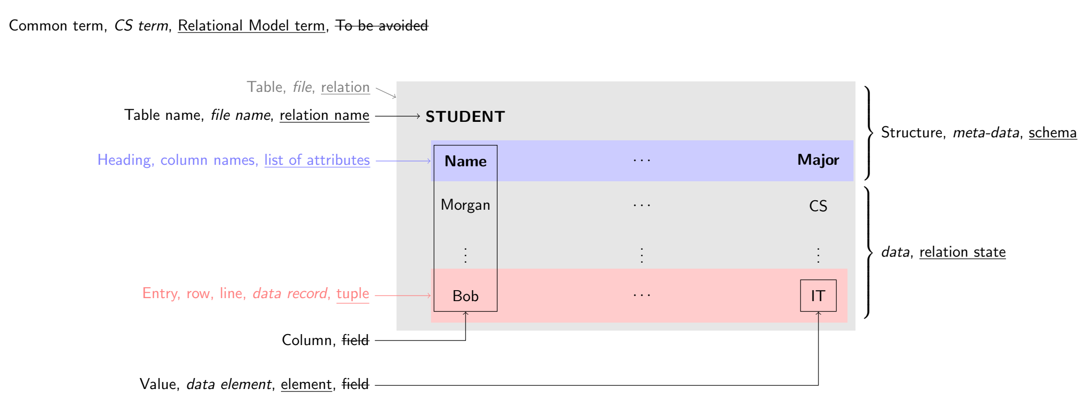


Figure 2.1: Terminology

The relational data model (or relational database schema) is:

- a mathematical model (use mathematical relations, set-theory, first-order predicate logic)
- with multiple implementations ("engineering approximation")

2.2 Domains, Attributes, Tuples and Relations

2.2.1 Definitions

- **Domain** (or type) = set of atomic (as far as the relation is concerned) values. You can compare it to datatype and literals, and indeed it can be given in the form of a data type, but it can be named and carry a logical definition (i.e., `List_of_major` as an enumerated

[1]https://en.wikipedia.org/wiki/Relational_model
[2]https://en.wikipedia.org/wiki/Category:Relational_database_management_systems

data type, instead of just `String`), enforce some constraints (i.e., **`UNIQUE`**, to force all the values to be different), or even have a default value.

- **Attribute** = Attribute name + attribute domain (but we'll just write the name).
- **Relation Schema** (or scheme) = description of a relation, often written "RELATION_NAME(Attribute$_1$, ..., Attribute$_n$)", where n is the degre (arity) of the relation, and the domain of Attribute$_i$ is written dom(Attribute$_i$).
- **Tuple** t of the schema R(A$_1$, ..., A$_n$) is an ordered list of values <v$_1$, ..., v$_n$> where v$_i$ is in dom(A$_i$) or a special **`NULL`** value.
- **Relation** (or relation state) r of the schema R(A$_1$, ..., A$_n$), also written r(R), is the set of n-tuples t$_1$, ..., t$_m$ where each t$_i$ is a tuple of the schema R(A$_1$, ..., A$_n$).

2.2.2 Characteristics of Relations

- In a relation, the order of tuples does not matter (a relation is *a set*). Order *in* tuple *do* matter (alternate representation where this is not true exist, cf. self-describing data).
- Value is atomic = "flat relational model", we will always be in the first normal form (not composite, not multi-valued).
- **`NULL`** is N/A, unknown, unavailable (or withheld).
- While a relation schema is to be read like an assertion (e.g., *"Every student has a name, a SSN, ..."*) a tuple is a fact (e.g., *"The student Bob Taylor has SSN 12898, ..."*).
- Relations represents uniformly entities (STUDENT(...)) and relations (PREREQUISITE(Course_number, Prerequisite_number)).

2.2.3 Notation

- STUDENT = relation schema + current relation state
- STUDENT(Name, ..., Major) = relation schema only
- STUDENT.Name = Attribute Name in the relation STUDENT
- t[Name], t[Name, Major], t.Name (overloading the previous notation) for the value of Name (and Major) in the tuple t.

2.3 Constraints

We now study constraints *on the tuples*. There are constraints on the scheme, for instance, "a relation cannot have two attributes with the same name", but we studied those already. The goal of those constraints is to maintain the validity of the relations, and to enforce particular connexions between relations.

2.3.1 Inherent Model-Based Constraints (implicit)

Those are part of the definition of the relational model and are independent of the particular relation we are looking at.

- You can not have two identical tuples in the same relation,
- The arity of the tuple must match the arity of the relation.

2.3.2 Schema-Based Constraints (explicit)

Those constraints are parts of the schema.

- The value must match its domain ("Domain constraint"), knowing that a domain can have additional constraints (`NOT NULL`, `UNIQUE`).
- The entity integrity constraint: no primary key value can be `NULL`[3].
- The referential integrity constraint: referred values must exist.

Those last two constraints will be studied in the next section.

2.3.3 Application-Based Constraints (semantics)

Constraints that cannot be expressed in the schema, and hence must be enforced by

- the application program,
- or the database itself, using triggers or assertions.

Examples: "the age of an employee must be greater than 16", "this year's salary increase must be more than last year's".

2.4 Keys

Since we can not have two identical tuples in the same relation, there must be a subset of values that distinguish them. We study the corresponding subset of attributes.

- A **superkey** is the subset of attributes for which no two tuples have the same values. Formally, the set of attributes SK is a superkey for the relation R, if for all relation state r of R, all tuples t_1, t_2 in r are such that $t_1[SK] \neq t_2[SK]$.
- A **key** is a minimal superkey (i.e., removing any attribute from SK would break the uniqueness property).
- A **candidate key** is a key, a **primary key** is the selected candidate key (it is underlined).

Let us consider the following example:

A	B	C	D
Yellow	Square	10	(5, 3)
Blue	Rectangle	10	(3, 9)
Blue	Circle	9	(4, 6)

and the following sets of attributes:

	{A, B, C, D}	{A}	{B, C}	{D}
Superkey ?	✔	✘	✔	✔
Key ?	✘	✘	✘	✔

[3]This is also the way this is implemented in MySQL: no part of the primary key can have for value `NULL`. Cf. the "Declaring Constraints" Section.

Note that here we “retro-fit” those definitions, in database design, they come first (i.e., you define what attributes should always distinguish between tuples before populating your database). We are making the assumption that the data pre-exist to the specification to make the concept clearer.

2.5 Foreign Keys

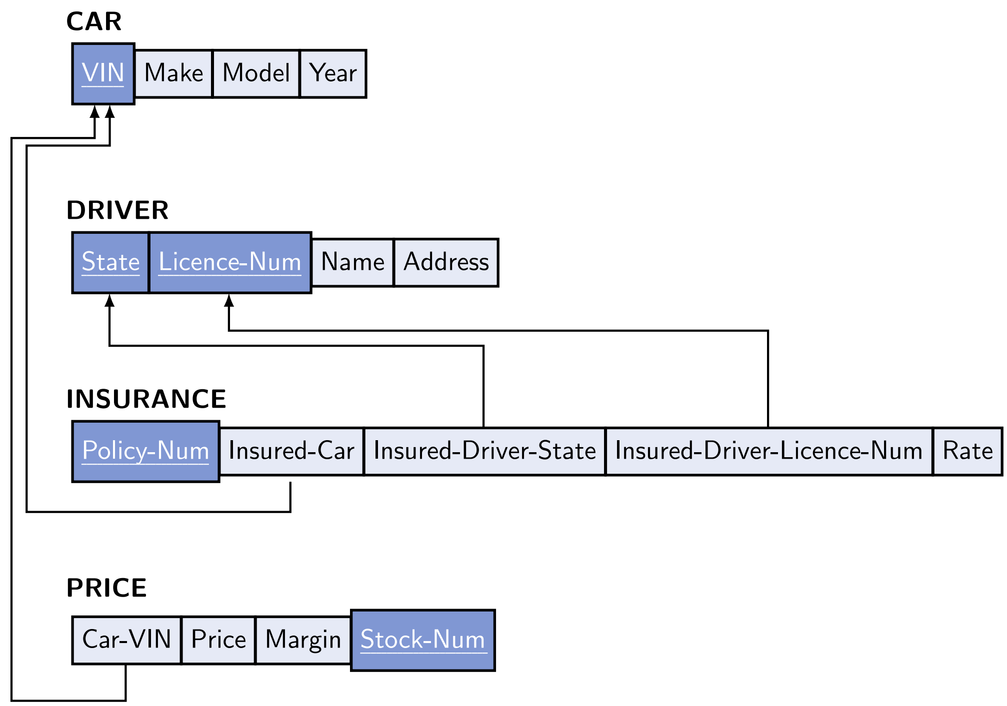
A **foreign key** (FK) is a set of attributes whose values must match the value in a tuple in another, pre-defined relation. Formally, the set of attributes FK in the relation schema R_1 is a foreign key of R_1 (“referencing relation”) that references R_2 (“referenced relation”) if

- FK refers to R_2 (i.e., the attributes in FK have the same domain(s) as the primary key PK of R_2)
- a value of FK in a tuple t_1 of $r_1(R_1)$ either
  - occurs as a value of PK for some tuple t_2 of $r_2(R_2)$, i.e., $t_1[FK] = t_2[PK]$
  - is `NULL`

  in the first case, we say that “t_1 refers to t_2”.

If there is a foreign key from R_1 to R_2, then we say that there is a referential integrity constraint from R_1 to R_2. We draw it with an arrow from the FK to the PK. Note that it is possible that $R_1 = R_2$.

2.6 Example



- In CAR, VIN is the primary key, so it must satisfy the entity integrity constraint, and its value can not be **`NULL`**. Note also that all the values must be different, as the same value cannot occur twice as the primary key of tuples: we don't want to enter the same VIN twice, that would mean we are registering a car that was already registered in our database!
- In DRIVER, State **and** Licence-num are the primary key[4], so they must *together* satisfy the intergrity constrant: neither of them can be **`NULL`**. Furthermore, their *pair* must be different from all the other values. Stated differently, you can have `<GA, 1234>`, `<GA, 0000>` and `<NC, 1234>` as values for the <State, Licence-num> pair, even if they have one element in common, what is forbidden is to have *both* element in common (i.e., you cannot have `<GA, 1234>` twice). If both elements were common, that would mean that we are registering a driver that was already in the database.
- Insurance has a primary key, and three foreign keys. The foreign keys must satisfy the referential integrity constraint: if the value stored in Insured-Car is not **`NULL`** (which it could be), then it has to be a value that occurs as the VIN value of some tuple in the CAR relation. For the Insured-Driver-State and Insured-Driver-Licence-Num, the situation is similar: they must either both be **`NULL`**, or be values that occurs *paired together* as the values for State and Licence-Num in a tuple in the CAR relationship. If e.g. Insured-Car was containing the VIN of a car not in the CAR relation, that would mean we are trying to insure a car that is "not known" from the database's perspective, something we certainly want to avoid.
- In Price, we have a primary key and a foreign key that obey similar requirements as before.

[4]Yes, we do need the state *and* the licence number to uniquely identify a driver's licence, since many states use the same licence format[5].

2.7 Transactions and Operations

The operations you can perform on your data are of two kinds: retrievals and updates.

- Retrievals leave the relation(s) state as it is and output a result relation. That is, retrieval: relation(s) state $\longrightarrow$ result relation
- Updates change the relation(s) state. That is, update: relation(s) state $\longrightarrow$ relation(s) state

They are two constraints for updates:

1. The new relation state must be "valid" (i.e., comply with the constraints).
2. There might be transition constraints (your balance cannot become negative, for instance).

A transaction is a series of retrievals and updates performed by an application program, that leaves the database in a consistent state.

In the following, we give examples of insertion, deletion and update that could be performed, as well as how they could lead a database to become inconsistent. The annotations (1.), (2.) and (3.) refer to the "remedies", discussed afterward.

2.7.1 Insert

```
Insert <109920, Honda, Accord, 2012> into CAR
```

How things can go wrong:

- Inserting the values in the wrong order (meta)
- `NULL` for any value of the attributes of the primary key (1.)
- Duplicate value for all the values in the primary key (1.)
- Wrong number of arguments (1.)
- Fail to reference an existing value for a foreign key (1.)

2.7.2 Delete

```
Delete the DRIVER tuple with State = GA and Licence_number = 123
```

How things can go wrong:

- Deleting tuples inadvertently (meta)
- Deleting tuples that are referenced (1., 2., 3.)

2.7.3 Update (a.k.a. Modify)

```
Update Name of tuple in DRIVER where State = GA and Licence_number = 123 to Ge
```

How things can go wrong:

- `NULL` for the any value of the attributes of the primary key (1.)
- Duplicate value for the primary key (1.)
- Change value that are referenced (1., 2., 3.)
- Change foreign key to a non-existing value (1.)

2.7.4 Dealing with Violations

When the operation leads the database to become inconsistent, you can either:

1. Reject (restrict) the operation,
2. Cascade (propagate) the modification,
3. Set default, or set **NULL**, the corresponding value(s).

Exercises

Exercise 2.1 What are the meta-data and the data called in the relational model?

Exercise 2.2 Connect the dots:

Row •	• Attribute
Column header •	• Tuple
Table •	• Relation

Exercise 2.3 What do we call the number of attributes in a relation?

Exercise 2.4 At the logical level, does the order of the tuples in a relation matter?

Exercise 2.5 What is the difference between a database schema and a database state?

Exercise 2.6 What should we put as a value in an attribute if its value is unknown?

Exercise 2.7 What, if any, is the difference between a superkey, a key, and a primary key?

Exercise 2.8 Name the two kinds of integrity that must be respected by the tuples in a relation.

Exercise 2.9 What is entity integrity? Why is it useful?

Exercise 2.10 Are we violating an integrity constraint if we try to set the value of an attribute that is part of a primary key to **NULL**? If yes, which one?

Exercise 2.11 If in a relation R_1, an attribute A_1 is a foreign key referencing an attribute A_2 in a relation R_2, what does this implies about A_2?

Exercise 2.12 Give three examples of operations.

Exercise 2.13 What is the difference between an operation and a transaction?

Exercise 2.14 Consider the following two relations:

COMPUTER(Owner, RAM, Year, Brand)
OS(Name, Version, Architecture)

For each, give

1. The arity of the relation,
2. A (preferably plausible) example of tuple to insert.

Exercise 2.15 Give three different ways to deal with operations whose execution in isolation would result in the violation of one of the constraint.

Exercise 2.16 Define what is the domain constraint.

Exercise 2.17 Circle the *correct* statements:

- Every key is a superkey.
- Every superkey is a singleton.
- Every singleton is either a superkey, or a key.
- Every primary key is a key.
- Every superkey with one element is a key.

Exercise 2.18 Consider the following three relations:

AUTHOR

Ref	Name	Address

BOOK

ISSN	AuthorRef	Title

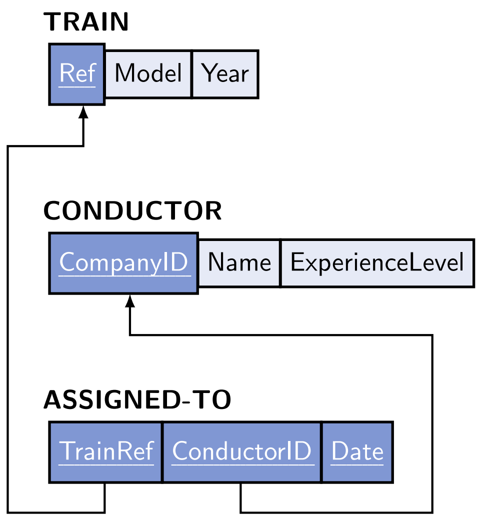
GAINED-AWARD

Ref	Name	BookISSN	Year

For each relation, answer the following:

1. What is, presumably, the primary key?
2. Are they, presumably, any foreign key?
3. Using the model you defined, could we determine which author won the greatest number of awards a particular year?

Exercise 2.19 Consider the following three relations



1. What are the foreign keys in the ASSIGNED-TO relation? What are they refering?
2. In the ASSIGNED-TO relation, explain why the Date attribute is part of the primary key. What would happen if it was not?
3. Assuming the database is empty, are the following instructions valid? If not, what integrity constraint are they violating?

   a) `Insert <'AM-356', 'Surfliner', 2012> into TRAIN`
   b) `Insert <NULL, 'Graham Palmer', 'Senior'> into CONDUCTOR`
   c) `Insert <'XB-124', 'GPalmer', '02/04/2018'> into ASSIGNED-TO`
   d) `Insert <'BTed, 'Bobby Ted', 'Senior'> and <'BTed', 'Bobby Ted Jr.',`

Exercise 2.20 Consider the following relation schema and state:

A	B	C	D
2	Blue	Austin	`true`
1	Yellow	Paris	`true`
1	Purple	Pisa	`false`
2	Yellow	Augusta	`true`

Assuming that this is all the data we will ever have, discuss whenever $\{A, B, C, D\}$, $\{A, B\}$ and $\{B\}$ are superkeys and/or keys.

Exercise 2.21 Consider the following relation and possible state. Assuming that this is all the data we will ever have, give two superkeys, and one key, for this relation.

A	B	C	D
1	Austin	`true`	Shelly
1	Paris	`true`	Cheryl
3	Pisa	`false`	Sheila
1	Augusta	`true`	Ash

A	B	C	D
1	Pisa	`true`	Linda

Exercise 2.22 Consider the following relation and possible state. Assuming that this is all the data we will ever have, give three superkeys for this relation, and, for each of them, indicate if they are a key as well.

A	B	C	D
1	A	Austin	`true`
2	B	Paris	`true`
1	C	Pisa	`false`
2	C	Augusta	`true`
1	B	Augusta	`true`

Exercise 2.23 Consider the following two relations:

1. Give two possible tuples for the BUILDING relation, and two possible tuples for the ROOM relation such that the state is consistent.
2. Based on the data you gave previously, write (in pseudo-code) one `INSERT` and one `UPDATE` instruction. Both should violate the integrity of your database.

Exercise 2.24 Consider the following two relations:

- A **Movie** relation, with attributes "Title" and "Year". The "Title" attribute should be the primary key.
- A **Character** relation, with attributes "Name", "First_Appearance". The "Name" attribute should be the primary key, and the "First_Appearance" attribute should be a foreign key referencing the **Movie** relation.

1. Draw its relational model.
2. Give an example of data that would violate the integrity of your database, and name the kind of integrity you are violating.

Solution to Exercises

Solution 2.1 The meta-data is called the *schema*, and the data is called the *relation state*. You can refer to the diagram we studied at the beginnig of the Chapter for a reminder.

Solution 2.2 Row is Tuple, Column header is Attribute, Table is Relation.

Solution 2.3 The degree, or arity, of the relation.

Solution 2.4 No, it is a set.

Solution 2.5 The schema is the organization of the database (the meta-data), while the state is the state is the content of the database (the data).

Solution 2.6 `NULL`

Solution 2.7 A superkey is a subset of attributes such that no two tuples have the same combination of values for all those attributes. A key is a minimal superkey, i.e., a superkey from which we cannot remove any attribute without losing the uniqueness constraint. The primary key is one of the candidate key, i.e., the key that was chosen.

Solution 2.8 Referential integrity and entity integrity.

Solution 2.9 Entity integrity ensures that each row of a table has a unique and non-null primary key value. It allows to make sure that every tuple is different from the others, and helps to "pick" elements in the database.

Solution 2.10 Yes, the entity integrity constraint.

Solution 2.11 Then we know that A_2 is the primary key of R_2, and that A_1 and A_2 have the same domain.

Solution 2.12 Reading from the database, performing `UPDATE` or `DELETE` operations.

Solution 2.13 An operation is an "atomic action" that can be performed on the database (adding an element, updating a value, removing an element, etc.). A transaction is a series of such operations, and the assumption is that, even if it can be made of operations that, taken individually, could violate a constraint, the overall transaction will leave the database in a consistent state.

Solution 2.14

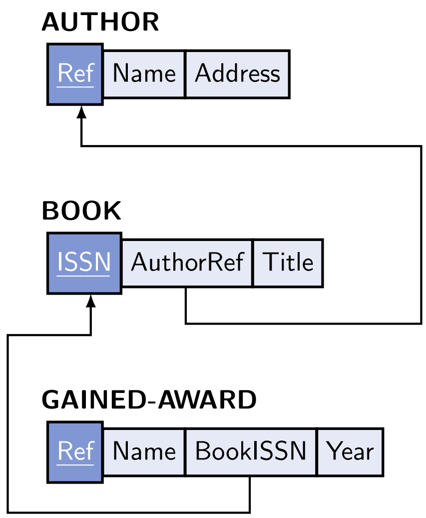
1. The arities of the relations are: COMPUTER has for arity 4, and OS has for arity 3.
2. Examples of tuple to insert are ("Linda McFather", 32, 2017, "Purism"), and ("Debian", "Stable", "amd64").

Solution 2.15 An operation whose execution in isolation would result in the violation of a constraint can either a) be "restricted" (i.e., not executed), b) result in a propagation (i.e., the tuples that would violate a constraint are updated or deleted accordingly), or c) result in some values in tuples that would violate a constraint to be set to a default value, or the `NULL` value (this last option works only if the constraint violated is the referential entity constraint).

Solution 2.16 The requirement that each tuple must have for an attribute A an atomic value from the domain dom(A), or `NULL`.

Solution 2.17 "Every key is a superkey.", "Every primary key is a key." and "Every superkey with one element is a key." are correct statements.

Solution 2.18 To answer 1 and 2, the diagram would become:



For the last question, the answer is yes: based on the ISSN of the book, we can retrieve the author of the book. Hence, knowing which book was awarded which year, by looking in the GAINED-AWARD table, gives us the answer to that question.

Solution 2.19

1. In ASSIGNED-TO, TrainRef is a FK to TRAIN.Ref, and ConductorID is a FK to CONDUCTOR.CompanyID.
2. In this model, a conductor can be assigned to different trains on different days. If Date was not part of the PK of ASSIGNED-TO, then a conductor could be assigned to only one train.
3.
   a) Yes, this instruction is valid.
   b) No, it violates the entity integrity constraint: **`NULL`** can be given as a value to an attribute that is part of the PK.
   c) No, it violates the referential integrity constraint: `'XB-124` and `'GPalmer'` are not values in `TRAIN.Ref` and `CONDUCTOR.CompanyID`.
   d) No, it violates the key constraint: two tuples cannot have the same value for the values of the primary key.

Solution 2.20

- $\{A, B, C, D\}$ is a superkey (the set of all the attributes is always a superkey), but not a superkey, as removing e.g. D would still make it a superkey.
- $\{A, B\}$ is a superkey **and** a key, as neither $\{A\}$ nor $\{B\}$ are keys.
- $\{A\}$ is not a key, and not a superkey: multiple tuples have the value 1.

Solution 2.21 For this relation, $\{A, B, C, D\}$, $\{A, B, C\}$, and $\{D\}$ are superkey. Only the latter, $\{D\}$, is a key (for $\{A, B, C\}$, removing either A or C still gives a superkey).

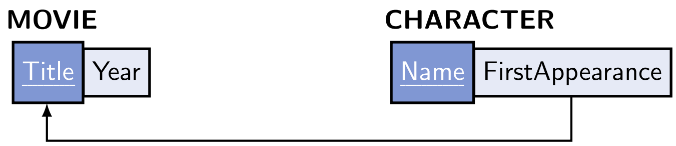
Solution 2.22 Possible superkeys are $\{A, B, C, D\}$, $\{A, B, C\}$, $\{A, C, D\}$, $\{B, C, D\}$, $\{A, B\}$, $\{B, C\}$. The possible keys are $\{A, B\}$ $\{A, C\}$, and $\{B, C\}$.

Solution 2.23

1. For the BUILDING relation: <“A.H”, “123 Main St.”>, <“U.H.”, “123 Main St.”>. For the ROOM relation: <12, “A.H.”>, <15, “A.H.”>.
2. `INSERT <"A.H.", NULL>` would violate the requirement not to have two tuples with the same value for the attributes that constitute the primary key in the BUILDING relation. `UPDATE ROOM with CODE = 12 to Building = "G.C.C."` would create an entry referencing a name in the BUILDING relation that does not exist.

Solution 2.24

1. The relations would be drawn as follows:



1. Inserting <“Ash”, “Evil Dead”> into the **CHARACTER** relation would cause an error if the database was empty, since no movie with the primary key “Evil Dead” has been introduced yet: this would be a referential integrity constraint violation. To violate the entity integrity constraint, it would suffice to insert the value <NULL, 2019> into the **MOVIE** relation.

Problems

Problem 2.1 ***(Find a candidate key for the CLASS relation)*** Consider the relation representing classes taught in a university:

CLASS(Major, Number, Section, Instructor, Term, Year, Time, Weekdays, Room)

The goal is to be able to have multiple offerings (classes) of courses over several semesters. Here are some examples of values for the attributes:

Attribute	Possible Value
Major	CSCI, AIST, CYBER, HIST, …
Number	1301, 3401, 1201, …
Section	A, B, C, …
Instructor	John Smith, Sophie Adams, …
Term	Spring, Fall, …
Year	1990, 2010, …
Time	1400, 1230, 0900, …
Weekdays	M, MW, MWF, …
Room	UH 120, GCC 3014, …

List three possible candidate keys and describe under what conditions each candidate key would be valid.

Problem 2.2 ***(Design a relational model for a cinema company)*** A cinema company wants you to design a relational model for the following set-up:

- The company has movie stars. Each star has a name, birth date, and unique ID.
- The company has the following information about movies: title, year, length, and genre. Each movie has a unique ID and features multiple stars.
- The company owns movie theaters as well. Each theater has a name, address, and a unique ID.
- Furthermore, each theater has a set of auditoriums. Each auditorium has a unique number, and seating capacity.
- Each theater can schedule movies at show-times. Each show-time has a unique ID, a start time, is for a specific movie, and is in a specific theater auditorium.
- The company sells tickets for scheduled show-times. Each ticket has a unique ticket ID and a price.

Problem 2.3 ***(Design a relational model for bills)*** Propose a relational model for the following situation:

- The database will be used to store all of the bills that are debated and voted on by the U.S. House of Representatives (HR). Each bill has a name, a unique sponsor who must be a member of the HR, and an optional date of when it was discussed.
- It must record the name, political group, and beginning and expected end-of-term dates for each HR member.
- It will also record the names of the main HR positions: Speaker, Majority Leader, Minority Leader, Majority Whip, and Minority Whip.
- Finally, it will record the vote of every member of the HR for each bill.

Problem 2.4 ***(Relational model for universities)*** Propose a relational model for the following situation:

- You want to store information about multiple universities. A university has multiple departments, a name and a website.
- Each department offers multiple courses. A course has a name, one (or multiple, when it is cross-listed) code, a number of credit hours.
- A campus has a name, an address, and belong to one university.
- A department has a contact address, a date of creation and a unique code.

Problem 2.5 ***(Relational model for an auction website)*** We want to design a relational model for an auction website. Members (that can be buyers, sellers, both or neither) can participate in the sale of items.

- Members are identified by a unique identifier and have an email address and a nickname.
- Buyers have a unique identifier, a preferred method of payment and a shipping address.

- Sellers have a unique identifier, a rating and a bank account number.
- Items are offered by a seller for sale and are identified by a unique item number. Items also have a name and a starting bid price.
- Members make bids for items that are for sale. Each bid has a unique identifier, a bidding price and a timestamp.

When creating your schema, do not add any new information, and try as much as possible to avoid relations that will create redundant data and **NULL** entries. Note that we should be able to uniquely determine the member account linked to the seller account, and similarly for buyers accounts. Furthermore, members can have at most one buyer and one seller account.

Problem 2.6 ***(Relational model for a pet shelter)*** We want to design a relational model for an animal shelter, with three goals in mind: to keep track of the pets currently sheltered, of the veterinarian for each type of pet, and of each pet's favorite toy (needed during a visit to the veterinarian!).

Follow the specification below:

- An animal has a type (cat, fish, dog, etc.), an arrival date, a name, and an id number.
- Every type of animal has a veterinarian.
- A veterinarian has a name, a phone number, an email address, and a postal address.
- Multiple types of animals can have the same veterinarian.
- A toy has a location, a description, a name, and is best suited for a particular type of animal.
- Each animal has at most one preferred toy.

When creating your schema (that you can draw at the back of previous page), do not add any new information (except possibly "id" attributes), and try as much as possible to avoid relations that will create redundant data and **NULL** entries. Identify the primary key for each relation that you create. When you are done, answer the true / false question below.

With your model ...	Yes	No
...it is possible to determine which pet don't have a favorite toy.		
...it is possible to determine what is the average stay in the shelter.		
...it is possible to determine if a pet's favorite toy is best suited for their type.		
...it is possible for multiple types of animal to have the same veterinarian.		
...it is possible for multiple veterinarians to be attributed to the same type.		

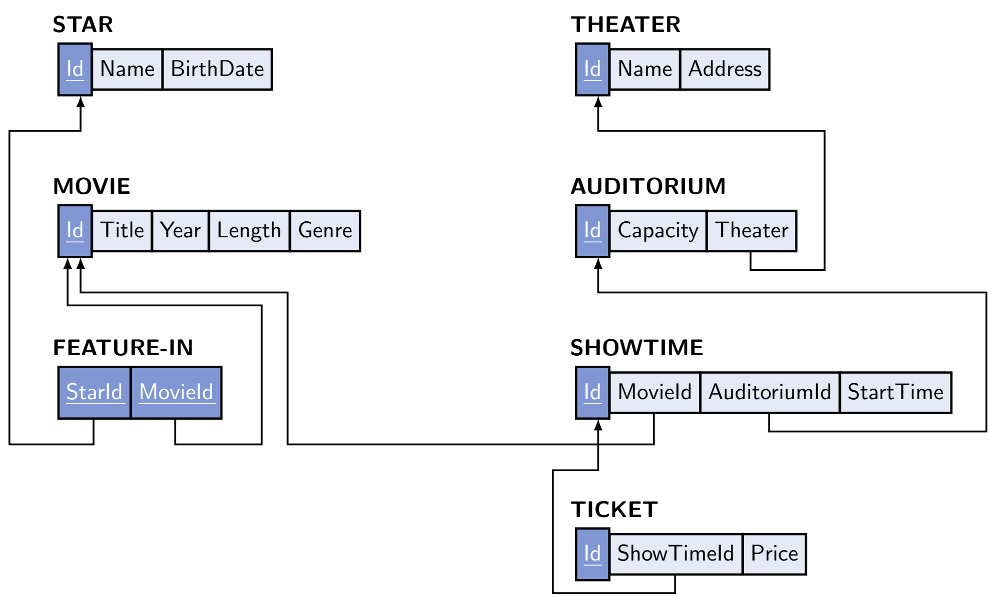
Solutions to Selected Problems

Solution to Problem 2.1 (Find a candidate key for the CLASS relation) We discuss four possible choices:

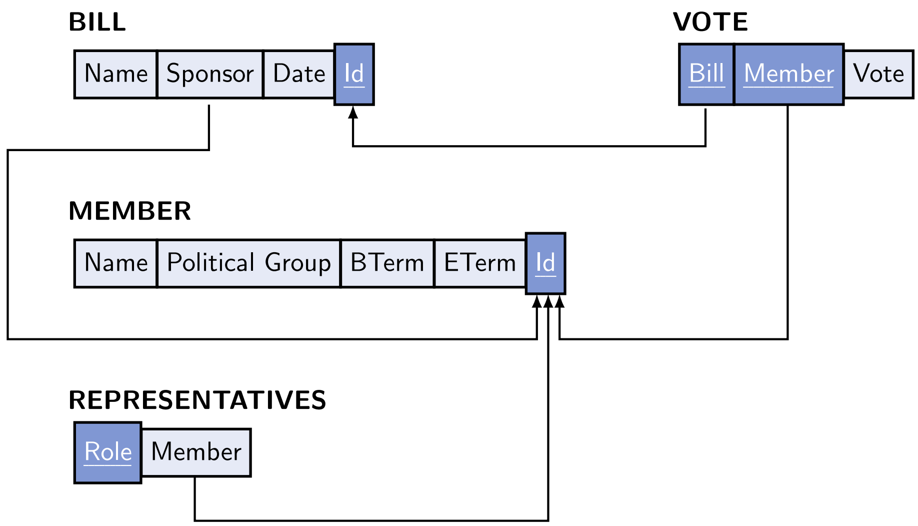
1. {Major, Number, Section, Year} This key would be valid if there was only 1 semester per year.
2. {Instructor, Term} This key would be valid if instructors were always teaching the same unique class each term (i.e., an instructor only teaching CSCI 3410 in the Fall, and nobody else teaching it during Fall).
3. {Room, Weekdays, Time} This key would be valid if the same room was used all the time (accross years, and terms) for the same class. Note also that remote classes would probably become problematic.
4. {Major, Number, Term, Year} This key would be valid if no two sections of the same class was offered at the same time.

All in all, {Major, Number, Term, Year, Section} seems like the safest choice.

Solution to Problem 2.2 (Design a relational model for a cinema company) A possible solution is:

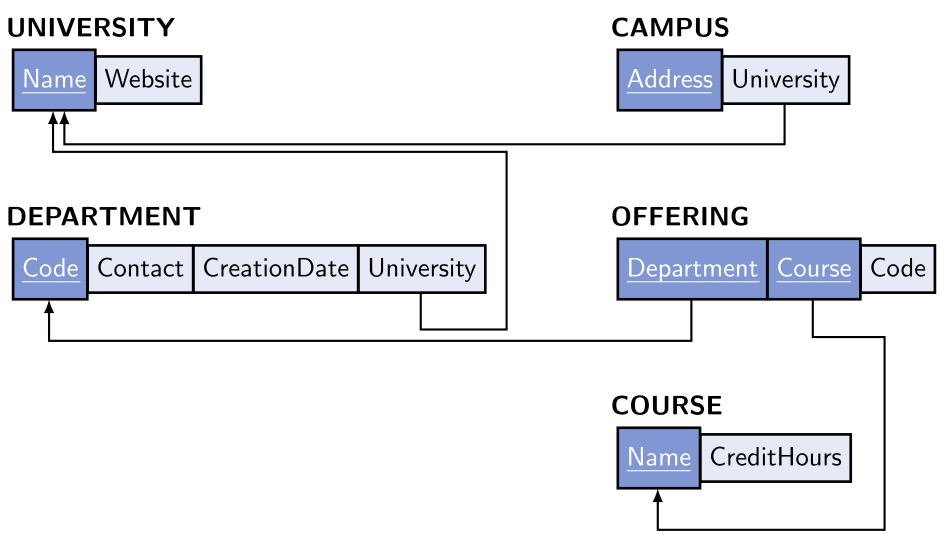


Solution to Problem 2.3 (Design a relational model for bills) Be careful: saying that a bill has a unique sponsor does *not* imply that a the sponsor is a good primary key for the bills: a house member could very well be the sponsor of multiple bills! It just implies that a single attribute is enough to hold the name of the sponsor.

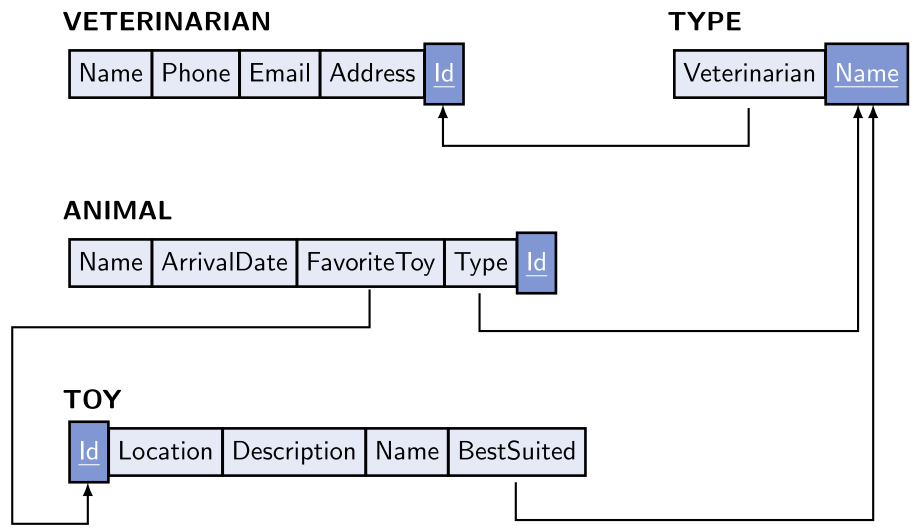


For simplicity, we added an `ID` to our `MEMBER` and `BILL` relations. Note that having a "role" in the `MEMBER` relation to store the information about speaker, etc., would be extremely inefficient, since we would add an attribute to the ~435 members that would be **`NULL`** in ~430 of them.

Solution to Problem 2.4 (Relational model for universities) A possible solution follows. The part that is the hardest to accomodate is the fact that a course can have multiple codes. We are reading here "cross-listed" as "a course that is offered under more than one departmental heading and can receive different codes (e.g., CSCI XXXX and AIST YYYY)".



Solution to Problem 2.6 (Relational model for a pet shelter) A possible solution follows.



In this model,
...it **is** possible to determine which pet don't have a favorite toy.
...it **is not** possible to determine what is the average stay in the shelter, *because their exit date is not stored.*
...it **is** possible to determine if a pet's favorite toy is best suited for their type.
...it **is** possible for multiple types of animal to have the same veterinarian, *as the same value for "Veterinarian" could occur in multiple tuples in the TYPE relation. If both "Veterinarian" and "Name" were parts of the primary key, then that would not be the case.*
...it **is not** possible for multiple veterinarians to be attributed to the same type, *as the name of the type is the primary key in the TYPE relation.*

3 The SQL Programming Language

Resources

- (Elmasri and Navathe 2010, ch. 4–5), (Elmasri and Navathe 2015, ch. 6–7) describes `SQL`, but none of its implementation.
- To compare DBMS, you can look at their features, at https://en.wikipedia.org/wiki/Comparison_of_relational_database_management_systems, and at their popularity at https://db-engines.com/en/ranking and https://insights.stackoverflow.com/survey/2020/#technology-databases (if you sum up MySQL and MariaDB, they are first in both).
- MySQL and MariaDB have some differences, you can look them up at https://mariadb.com/kb/en/mariadb-vs-mysql-features/ and https://mariadb.com/kb/en/mariadb-vs-mysql-compatibility/.

This chapter will be "code-driven": the code will illustrate and help you understand some concepts. You may want to have a look at the "Setting Up Your Work Environment" Section as early as possible in this lecture. On top of being a step-by-step guide to install and configure a relational database managment system, it contains a list of useful links.

3.1 Actors

3.1.1 Technologies

- There are other models than relational: document-based, graph, column-based, and key-value models. Those corresponds to the "NoSQL" data-model, that are often more flexible, but only defined by opposition. They will be studied separately, in the Presentation of NoSQL Chapter.
- The most commons DBMS are relational database management system (RDBMS):
  - Oracle Database[1]
  - MySQL[2] and its fork, MariaDB[3]
  - Microsoft SQL Server[4]
  - PostgreSQL[5]
  - IBM DB2[6]
  - Microsoft Access[7]
  - SQLite[8]

[1] https://www.oracle.com/database/technologies/oracle-database-software-downloads.html
[2] https://www.mysql.com/
[3] https://mariadb.org/
[4] https://www.microsoft.com/en-us/sql-server/sql-server-2017
[5] https://www.postgresql.org/
[6] https://www.ibm.com/analytics/db2
[7] https://products.office.com/en-us/access
[8] https://www.sqlite.org/index.html

Most of them supports semi-structured data, i.e., models that are not strictly speaking relational, some are "multi-model DBMS".
- The **S**tructured **Q**uery **L**anguage (SQL) is *the* language for RDBMS, it is made of 4 sublanguages:
  - **D**ata **Q**uery **L**anguage,
  - **D**ata **D**efinition **L**anguage (schema creation and modification),
  - **D**ata **C**ontrol **L**anguage (authorizations, users),
  - **D**ata **M**anipulation **L**anguage (insert, update and delete).

3.1.2 SQL

3.1.2.1 Yet Another Vocabulary

"Common" / Relational	SQL
"Set of databases"	Catalog (named collection of schema)[9]
"Database"	Schema
Relation	Table
Tuple	Row
Attribute	Column, or Field

3.1.2.2 Schema Elements

A schema is made of

- Tables (≈ relation)
- Type (≈ datatype)
- Domain (≈ more complex datatype)
- View (result set of a stored query on the data, ≈ saved search)
- Assertion (constraints, transition constraints)
- Triggers (tool to automate certain actions after pre-defined operations are performed)
- Stored procedures (≈ functions)

Type and domains are two different things in some implementations, cf. for instance PostgreSQL, where a domain is defined to be essentially a datatype with constraint.[10]

3.1.2.3 Syntax

`SQL` *is* a programming language: it has a strict syntax, sometimes cryptic error messages, it evolves, etc. Some of its salient aspects are:

[9]For a clarification on the distinction between catalog and schemas, you can refer to e.g. https://stackoverflow.com/q/7022755.

[10]Cf. https://www.postgresql.org/docs/9.2/sql-createtype.html and https://www.postgresql.org/docs/9.2/sql-createdomain.html.

- SQL is "kind of" case-insensitive[11], does not care about spaces and new lines.
- In-line comments are what is after `--`, multi-line comments uses `/* … */`.
- Every statement ends with a `;`.
- The exact syntax is left as an exercise in Pb 3.1.
- The list of reserved words can be found at https://dev.mysql.com/doc/refman/8.0/en/keywords.html or https://mariadb.com/kb/en/reserved-words/
- We will focus in this chapter to MariaDB and MySQL (no domain, limited data type definition).

3.1.2.4 Datatypes

The following is an adaptation of w3resource.com[12], the canonical source being MySQL's documentation[13]:

- For integer types, you can use `INTEGER` (or its short-hand notation `INT`) or `SMALLINT`.
- For floating-point types, you can use `FLOAT` and `DOUBLE` (or its synonym, `REAL`). MySQL also allows the syntax `FLOAT(M,D)` or `REAL(M,D)`, where the values can be stored up to `M` digits in total where `D` represents the decimal point.
- For monetary amounts, it is recommended[14] to use `DECIMAL(10, 2)` (or its synonym in MySQL `NUMERIC`).
- Characters can be stored using `CHAR` and `VARCHAR`: the length (resp. maximal length) of the `CHAR` (resp. `VARCHAR`) has to be declared, and `CHAR` are right-padded with spaces to the specified length. Historically, 255[15] was the size used, because it is the largest number of characters that can be counted with an 8-bit number, but, whenever possible, the "right size[16]" should be used.
- You can store a single bit using `BIT(1)`, and a boolean using `BOOLEAN` (or `BOOL`, both actually being aliases for `TINYINT(1)`).
- For date and time types, you can use `DATE`, `TIME`, `DATETIME` and `TIMESTAMP` (which convert the current day / time to from the current time zone to UTC).

There are many other datatypes, but they really depends on the particular implementation, so we will not consider them too much.

3.2 First Commands

```sql
/* code/sql/HW_Faculty.sql */
-- We first drop the schema if it already exists:
DROP SCHEMA IF EXISTS HW_Faculty;

-- Then we create the schema:
CREATE SCHEMA HW_Faculty;
```

[11] The SQL keywords are case-insensitive, but the table and schema names are sometimes case-sensitive, it depends of the actual implementation. For instance, MySQL is completely case-insensitive (reserved words, tables, attributes), MariaDB is not (the case for table names matter).
[12] https://www.w3resource.com/mysql/mysql-data-types.php
[13] https://dev.mysql.com/doc/refman/8.0/en/data-types.html
[14] https://stackoverflow.com/a/4397416
[15] https://stackoverflow.com/q/1217466
[16] https://dba.stackexchange.com/a/183277

```


/*
Or we could have use the syntax:

CREATE DATABASE HW_FACUTLY;
 */
-- Now, let us create a table in it:
CREATE TABLE HW_Faculty.PROF (
  Fname VARCHAR(15),
  /*
   No String!
   The value "15" vas picked randomly, any value below 255 would
   more or less do the same. Note that declaring extremely large
   values without using them can impact the performance of
   your database, cf. for instance https://dba.stackexchange.com/a/162117/
   */
  Room INT,
  /*
   shorthand for INTEGER, are also available: SMALLINT, FLOAT, REAL, DEC
   The "REAL" datatype is like the "DOUBLE" datatype of C# (they are actually synonyms in SQL):
   more precise than the "FLOAT" datatype, but not as exact as the "NUMERIC" datatype.
   cf. https://dev.mysql.com/doc/refman/8.0/en/numeric-types.html
   */
  Title CHAR(3),
  -- fixed-length string, padded with blanks if needed
  Tenured BIT(1),
  Nice BOOLEAN,
  -- True / False (= 0) / Unknown
  Hiring DATE,
  /*
   The DATE is always supposed to be entered in a YEAR/MONTH/DAY variation.
   To tune the way it will be displayed, you can use the "DATE_FORMAT" function
   (cf. https://dev.mysql.com/doc/refman/8.0/en/date-and-time-functions.html#function_date-format),
   but you can enter those values only using the "standard" literals
   (cf. https://dev.mysql.com/doc/refman/8.0/en/date-and-time-literals.html)
   */
  Last_seen TIME,
  FavoriteFruit ENUM ('apple', 'orange', 'pear'),
```

```sql
  PRIMARY KEY (Fname, Hiring)
);


/*
 Or, instead of using the fully qualified name HW_Faculty.PROF,
 we could have done:

 USE HW_Faculty;
 CREATE TABLE PROF(…)
 */
-- Let us use this schema, from now on.
USE HW_Faculty;

-- Let us insert some "Dummy" value in our table:
INSERT INTO PROF
VALUES (
  "Clément", -- Or 'Clément'.
  290,
  'PhD',
  0,
  NULL,
  '19940101', -- Or '940101',  '1994-01-01',  '94/01/01'
  '090500', -- Or '09:05:00', '9:05:0',  '9:5:0',  '090500'
  -- Note also the existence of DATETIME, with 'YYYY-MM-DD
  --                   HH:MM:SS'
  'Apple' -- This is not case-sensitive, oddly enough.
);
```

HW_Faculty.sql[17]

3.3 Useful Commands

The following commands are particularly useful. They allow you to get a sense of the current state of your databases.

3.3.1 For Schemas

In the following, <SchemaName> should be substituted with an actual schema name.

```sql
SHOW SCHEMAS; -- List the schemas.
SHOW TABLES; -- List the tables in a schema.
DROP SCHEMA <SchemaName>; -- "Drop" (erase) SchemaName.
```

You can also use the variation

```sql
DROP SCHEMA IF EXISTS <SchemaName>;
```

[17] https://rocketgit.com/user/caubert/CSCI_3410/source/tree/branch/master/blob/notes/code/sql/HW_Faculty.sql

that will not issue an error if <SchemaName> does not exist.

3.3.2 For Tables

In the following, <TableName> should be substituted with an actual table name.

```
SHOW CREATE TABLE <TableName>-- Gives the command to
↪  "re-construct" TableName.
DESCRIBE <TableName>; -- Show the structure of TableName.
DROP TABLE <TableName>; -- "Drop" (erase) TableName.
```

Note that if the table <TableName> you are trying to erase is referenced by other tables through foreign keys, you will obtain an error

```
ERROR 1451 (23000): Cannot delete or update a parent row: a
↪  foreign key constraint fails
```

you must delete first the table containing the foreign key, as by default this operation is restricted.

If you want to erase a table *if it exists*, you can use the variation

```
DROP TABLE IF EXISTS <TableName>;
```

that will not issue an error if <TableName> does not exist.

3.3.3 See Also

```
SELECT * FROM <TableName> -- List all the rows in TableName.
SHOW WARNINGS; -- Show the content of the latest warning issued.
```

3.4 Overview of Constraints

There are six different kind of constraints that one can add to an attribute:

1. Primary Key
2. Foreign Key
3. **NOT NULL**
4. **UNIQUE**
5. **DEFAULT**
6. CHECK

We already know the first two from the relational model. The other four are new, and could not be described in this model.

We will review them below, and show how they can be specified at the time the table is declared, or added and removed later. For more in-depth examples, you can refer to https://www.w3resource.com/mysql/creating-table-advance/constraint.php.

Note that all of them but **DEFAULT** *are* indeed, constraints, as they prevent the user from inserting some data (i.e. you can not insert **NULL** if the attribute has the constraint **NOT NULL**). **DEFAULT** is *not* a constraint in that sense, as it does not prevent some data from being inserted, but it *is* called a constraint nevertheless. We will see another example of such "helper" qualification with `AUTO-INCREMENT`.

3.4.1 Declaring Constraints

We will now see how to declare those constraints when we create the table (except for the foreign key, which we save for later).

```sql
/* code/sql/HW_ConstraintsPart1.sql */
DROP SCHEMA IF EXISTS HW_ConstraintsPart1;

CREATE SCHEMA HW_ConstraintsPart1;

USE HW_ConstraintsPart1;

CREATE TABLE HURRICANE (
  Name VARCHAR(25) PRIMARY KEY,
  WindSpeed INT DEFAULT 76 CHECK (WindSpeed > 74 AND
    WindSpeed < 500),
  -- 75mph is the minimum to be considered as a hurricane
  --                    cf.
   ↪  https://www.hwn.org/resources/bws.html
  Above VARCHAR(25)
);

CREATE TABLE STATE (
  Name VARCHAR(25) UNIQUE,
  Postal_abbr CHAR(2) NOT NULL
);
```

HW_ConstraintsPart1.sql[18]

If we wanted to combine multiple constraints, we could[19], but we would have to follow the order described at https://dev.mysql.com/doc/refman/8.0/en/create-table.html, which is **NOT NULL**, **DEFAULT**, **AUTO_INCREMENT**, **UNIQUE**, **PRIMARY KEY**, `CHECK` (even if, in practise, derivation from this order is oftentimes accepted by DBMSes).

`MySQL` can output a description of those tables for us:

[18]https://rocketgit.com/user/caubert/CSCI_3410/source/tree/branch/master/blob/notes/code/sql/HW_ConstraintsPart1.sql

[19]Yes, we can even add a **DEFAULT** value to a **PRIMARY KEY**, even if that's of little interest. You can see an example ate code/sql/HW_Default_On_PK.sql[20].

```
MariaDB [HW_ConstraintsPart1]> DESCRIBE HURRICANE;
+-----------+-------------+------+-----+---------+-------+
| Field     | Type        | Null | Key | Default | Extra |
+-----------+-------------+------+-----+---------+-------+
| Name      | varchar(25) | NO   | PRI | NULL    |       |
| WindSpeed | int(11)     | YES  |     | 76      |       |
| Above     | varchar(25) | YES  |     | NULL    |       |
+-----------+-------------+------+-----+---------+-------+
3 rows in set (0.01 sec)

MariaDB [HW_ConstraintsPart1]> DESCRIBE STATE;
+-------------+-------------+------+-----+---------+-------+
| Field       | Type        | Null | Key | Default | Extra |
+-------------+-------------+------+-----+---------+-------+
| Name        | varchar(25) | NO   | PRI | NULL    |       |
| Postal_abbr | char(2)     | NO   | UNI | NULL    |       |
+-------------+-------------+------+-----+---------+-------+
2 rows in set (0.00 sec)
```

Note that more than one attribute can be the primary key, in which case the syntax needs to be something like the following:

```
/* code/sql/HW_PKtest.sql */
DROP SCHEMA IF EXISTS HW_PKtest;

CREATE SCHEMA HW_PKtest;

USE HW_PKtest;

CREATE TABLE TEST (
  A INT,
  B INT,
  PRIMARY KEY (A, B)
);
```

HW_PKtest.sql[21]

Note that in this case, a statement like

```
INSERT INTO TEST VALUE (1, NULL);
```

would result in an error: all the values that are part of the primary key needs to be non-**NULL**.

For the **UNIQUE** constraint, note that **NULL** can be inserted: the rationale is that all the values need to be different from one another or **NULL**.

A couple of comments about the `CHECK` constraint:

- Before MariaDB 10.2.1, `WindSpeed INT CHECK (WindSpeed > 74 AND WindSpeed < 500)` would have been parsed but would not have any effect, cf. https://mariadb.com/kb/en/constraint/#check-constraints. Since MariaDB 10.2.1, the `CHECK` constraint are enforced.
- If we try to violate the `CHECK` constraint, with a command like

[21] https://rocketgit.com/user/caubert/CSCI_3410/source/tree/branch/master/blob/notes/code/sql/HW_PKtest.sql

```
INSERT INTO HURRICANE VALUES ("Test1", 12, NULL);
```

then the insertion would not take place, and the system would issue an error message:

```
ERROR 4025 (23000): CONSTRAINT `HURRICANE.WindSpeed` failed for
↪ `HW_ConstraintsPart1]>`.`HURRICANE`
```

- Note that you could still insert a value of NULL for the wind, and it would not triggered the error.

To use the **DEFAULT** value, use

```
INSERT INTO HURRICANE VALUES ("Test2", DEFAULT, NULL);
```

Note that, by default, the **DEFAULT** value is **NULL**, regardless of the datatype. You can experiment it by running the following code:

```
/* code/sql/HW_DefaultTest.sql */
CREATE TABLE TEST (
  TestA VARCHAR(15),
  TestB INT,
  TestC FLOAT,
  TestD BOOLEAN,
  TestE BIT(1),
  TestF DATE
);

INSERT INTO TEST
VALUES (
  DEFAULT,
  DEFAULT,
  DEFAULT,
  DEFAULT,
  DEFAULT,
  DEFAULT);

SELECT *
FROM TEST;

```

HW_DefaultTest.sql[22]

3.4.2 Editing Constraints

Let us know pretend that we want to edit some attributes, by either adding or removing constraints. SQL's syntax is a bit inconsistent on this topic, because it treats the constraints as being of different natures.

[22]https://rocketgit.com/user/caubert/CSCI_3410/source/tree/branch/master/blob/notes/code/sql/HW_DefaultTest.sql

3.4.2.1 Primary Keys

Adding a primary key:

```
ALTER TABLE STATE ADD PRIMARY KEY (Name);
```

Removing the primary key:

```
ALTER TABLE STATE DROP PRIMARY KEY;
```

3.4.2.2 UNIQUE Constraint

Adding a **UNIQUE** constraint:

```
ALTER TABLE STATE ADD UNIQUE (Postal_abbr);
```

Removing a UNIQUE constraint:

```
ALTER TABLE STATE DROP INDEX Postal_abbr;
```

Note the difference between adding and removing the **UNIQUE** constraint: the parenthesis around `(Postal_abbr)` are mandatory when adding the constraint, but would cause an error when removing it!

3.4.2.3 NOT NULL Constraint

Adding the **NOT NULL** constraint:

```
ALTER TABLE STATE MODIFY Postal_abbr CHAR(2) NOT NULL;
```

Removing the **NOT NULL** constraint:

```
ALTER TABLE STATE MODIFY Postal_abbr CHAR(2);
```

The syntax of **NOT NULL** comes from the fact that this constraint is taken to be part of the datatype.

3.4.2.4 Default value

Changing the default value:

```
ALTER TABLE HURRICANE ALTER COLUMN WindSpeed SET DEFAULT 74;
```

Removing the default value:

```
ALTER TABLE HURRICANE ALTER COLUMN  WindSpeed DROP DEFAULT;
```

Note that if you change the default value, it does *not* change the values you inserted retro-actively. To resume on our previous example, the values inserted with **DEFAULT** as a value would still be **NULL** even after executing the following instruction:

```sql
/* code/sql/HW_DefaultTest.sql */
ALTER TABLE TEST
  ALTER COLUMN TestA SET DEFAULT "A";

SELECT *
FROM TEST;

```

HW_DefaultTest.sql[23]

3.4.2.5 Foreign key

Adding a foreign key constraint:

```sql
ALTER TABLE HURRICANE ADD FOREIGN KEY (Above) REFERENCES
 ↪  STATE(Name);
```

Removing a foreign key constraint is out of the scope of this lecture. If you are curious, you can have a look at https://www.w3schools.com/sql/sql_foreignkey.asp: dropping a foreign key constraint requires your constraint to have a name, something we did not introduce.

Two important remarks:

- The datatype of the foreign key has to be the exactly the same as the datatype of the attribute that we are referring.
- The target of the foreign key *must be* the primary key.

Refer to Problem 3.4 (Constraints on foreign keys) for a slightly more accurate picture of the constraints related to the creation of foreign keys. Note that a foreign key could be declared at the time of creation of the table as well, using the syntax we will introduce below.

3.4.3 Testing the Constraints

Let us test our constraints:

```sql
INSERT INTO STATE VALUES('Georgia', 'GA');
INSERT INTO STATE VALUES('Texas', 'TX');
INSERT INTO STATE VALUES('FLORIDA', 'FL');
UPDATE STATE SET Name = 'Florida'
    WHERE Postal_abbr = 'FL';

-- There's an error with the following request. Why?
INSERT INTO HURRICANE VALUES('Irma', 150, 'FL');

/*
ERROR 1452 (23000): Cannot add or update a child row: a foreign
 ↪  key constraint fails (`HW_ConstraintsPart1`.`HURRICANE`,
 ↪  CONSTRAINT `HURRICANE_ibfk_1` FOREIGN KEY (`Above`)
 ↪  REFERENCES `STATE` (`Name`))
```

[23] https://rocketgit.com/user/caubert/CSCI_3410/source/tree/branch/master/blob/notes/code/sql/HW_DefaultTest.sql

```
*/

INSERT INTO HURRICANE VALUES('Harvey', DEFAULT, 'Texas');
INSERT INTO HURRICANE VALUES('Irma', 150, 'Florida');
DELETE FROM HURRICANE
    WHERE Name = 'Irma';
INSERT INTO HURRICANE VALUES('Irma', 150, 'Georgia');

UPDATE HURRICANE SET Above = 'Georgia'
    WHERE Name = 'Irma';

/*
MariaDB [HW_ConstraintsPart1]> SELECT * FROM HURRICANE;
+---------+-----------+---------+
| Name    | WindSpeed | Above   |
+---------+-----------+---------+
| Harvey  |        74 | Texas   |
| Irma    |       150 | Georgia |
+---------+-----------+---------+
*/

-- There's an error with the following request. Why?
UPDATE HURRICANE SET Above = 'North Carolina'
    WHERE Name = 'Irma';

-- Let's patch it, by adding North Carolina to our STATE table.
INSERT INTO STATE VALUES('North Carolina', 'NC');
UPDATE HURRICANE SET Above = 'North Carolina'
    WHERE Name = 'Irma';
```

3.5 Foreign Keys

Let us come back more specifically to foreign key.

3.5.1 A First Example

In the example below, we introduce the foreign key update and delete rules. We also introduce, passing by, the enumerated data type, and how to edit it.

```
CREATE TABLE STORM (
  NAME VARCHAR(25) PRIMARY KEY,
  Kind ENUM ("Tropical
    Storm", "Hurricane"),
  WindSpeed INT,
  Creation DATE
);

-- We can change the enumerated datatype:
```

```
ALTER TABLE STORM MODIFY Kind ENUM ("Tropical Storm",
  "Hurricane", "Typhoon");

CREATE TABLE STATE (
  NAME VARCHAR(25) UNIQUE,
  Postal_abbr CHAR(2) PRIMARY KEY,
  Affected_by VARCHAR(25),
  FOREIGN KEY (Affected_by) REFERENCES STORM (NAME) ON
    DELETE SET NULL ON UPDATE CASCADE
);

```

HW_Storm.sql[24]

Note that we can “inline” the foreign key constraint like we “inlined” the primary key constraint (cf. https://stackoverflow.com/q/24313143/), but that **it will not be enforced**!

Let us now illustrate this table by introducing some data in it:

```
INSERT INTO STORM
VALUES (
  "Harvey",
  "Hurricane",
  130,
  "2017-08-17");

-- In the following, the entry gets created, but date is
--                     "corrected" to "2017-17-08"!
--                     INSERT INTO STORM
--                       VALUES ("Dummy", "Hurricane", 120,
--             "2017-17-08");
--                      The error message returned is
--                      ERROR 1292 (22007) at line 34:
-- Incorrect
--      date
--             value:
--                     "2017-17-08" for column
--               `HW_STORM`.`STORM`.`Creation`
--                  at
--                     row 1
--                      In the following, we explicitely use
--     "DATE",
--         and
--                since
--                     the date is incorrect, nothing gets
--     inserted.
--                     INSERT INTO STORM
--                       VALUES ("Dummy2", "Hurricane", 120,
--   DATE
--                   "2017-17-08");
```

[24] https://rocketgit.com/user/caubert/CSCI_3410/source/tree/branch/master/blob/notes/code/sql/HW_Storm.sql

```
--                    ERROR 1525 (HY000) at line 40:
-- Incorrect
--      DATE
--             value:
--                   "2017-17-08"
--                    The next one sets NULL for DATE.
INSERT INTO STORM
VALUES (
  "Irma",
  "Tropical Storm",
  102,
  DEFAULT);

```

HW_Storm.sql[25]

MySQL will always notify you if there is an error in a date attribute when you use the `DATE` prefix.

```
INSERT INTO STATE
VALUES (
  "Georgia",
  "GA",
  NULL);

INSERT INTO STATE
VALUES (
  "Texas",
  "TX",
  NULL);

INSERT INTO STATE
VALUES (
  "Florida",
  "FL",
  NULL);

-- This instruction is not using the primary key, is that a
--                   problem?
UPDATE
  STATE
SET Affected_by = "Harvey"
WHERE Name = "Georgia";

UPDATE
  STORM
SET Name = "Harley"
WHERE Name = "Harvey";

```

[25] https://rocketgit.com/user/caubert/CSCI_3410/source/tree/branch/master/blob/notes/code/sql/HW_Storm.sql

```
DELETE FROM STORM
WHERE Name = "Harley";

```

HW_Storm.sql[26]

We will see in the "Reverse-Engineering" section why this schema is poorly designed, but for now, let's focus on the foreign keys and their restrictions.

3.5.2 Foreign Keys Restrictions

The following is a code-driven explanation of the foreign key update and delete rules (or "restrictions"). It is intended to make you understand the default behavior of foreig keys, and to understand how the system reacts to the possible restrictions.

```
CREATE TABLE F_Key(
    Attribute VARCHAR(25) PRIMARY KEY
    );

CREATE TABLE Table_default(
    Attribute1 VARCHAR(25) PRIMARY KEY,
    Attribute2 VARCHAR(25),
    FOREIGN KEY (Attribute2) REFERENCES F_Key(Attribute)
    );

-- By default, this foreign key will restrict.

CREATE TABLE Table_restrict(
    Attribute1 VARCHAR(25) PRIMARY KEY,
    Attribute2 VARCHAR(25),
    FOREIGN KEY (Attribute2) REFERENCES F_Key(Attribute)
        ON DELETE RESTRICT
        ON UPDATE RESTRICT
    );

CREATE TABLE Table_cascade(
    Attribute1 VARCHAR(25) PRIMARY KEY,
    Attribute2 VARCHAR(25),
    FOREIGN KEY (Attribute2) REFERENCES F_Key(Attribute)
        ON DELETE CASCADE
        ON UPDATE CASCADE
    );

CREATE TABLE Table_set_null(
    Attribute1 VARCHAR(25) PRIMARY KEY,
    Attribute2 VARCHAR(25),
    FOREIGN KEY (Attribute2) REFERENCES F_Key(Attribute)
        ON DELETE SET NULL
        ON UPDATE SET NULL
```

[26] https://rocketgit.com/user/caubert/CSCI_3410/source/tree/branch/master/blob/notes/code/sql/HW_Storm.sql

```
    );

/*
* You might encounter a
* ON UPDATE SET DEFAULT
* but this reference option (cf.
 ↪  https://mariadb.com/kb/en/library/foreign-keys/ )
* worked only with a particular engine (
 ↪  https://mariadb.com/kb/en/library/about-pbxt/ )
* and will not be treated here.
*/

INSERT INTO F_Key VALUES('First Test');
INSERT INTO Table_default VALUES('Default', 'First Test');
INSERT INTO Table_restrict VALUES('Restrict', 'First Test');
INSERT INTO Table_cascade VALUES('Cascade', 'First Test');
INSERT INTO Table_set_null VALUES('Set null', 'First Test');

SELECT * FROM Table_default;
SELECT * FROM Table_restrict;
SELECT * FROM Table_cascade;
SELECT * FROM Table_set_null;

-- The following will fail because of the Table_default table:
UPDATE F_Key SET Attribute = 'After Update'
    WHERE Attribute = 'First Test';
DELETE FROM F_Key
    WHERE Attribute = 'First Test';

-- Let us drop this table, and try again.
DROP TABLE Table_default;
```

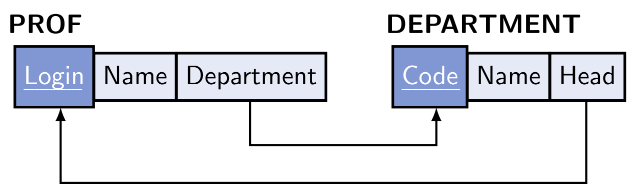
```
-- The following fails too, this time because of the
 ↪  Table_restrict table:
UPDATE F_Key SET Attribute = 'After Update'
    WHERE Attribute = 'First Test';
DELETE FROM F_Key
    WHERE Attribute = 'First Test';

-- Let us drop this table, and try again.
DROP TABLE Table_restrict;

-- Let's try again:
UPDATE F_Key SET Attribute = 'After Update' WHERE Attribute =
 ↪  'First Test';

-- And let's print the situation after this update:
SELECT * FROM Table_cascade;
```

```
SELECT * FROM Table_set_null;

/*
MariaDB [HW_CONSTRAINTS_PART3]> SELECT * FROM Table_cascade;
+------------+--------------+
| Attribute1 | Attribute2   |
+------------+--------------+
| Cascade    | After Update |
+------------+--------------+
1 row in set (0.00 sec)

MariaDB [HW_CONSTRAINTS_PART3]> SELECT * FROM Table_set_null;
+------------+------------+
| Attribute1 | Attribute2 |
+------------+------------+
| Set null   | NULL       |
+------------+------------+
1 row in set (0.00 sec)
*/

-- Let's make a second test.
INSERT INTO F_Key VALUES('Second Test');
INSERT INTO Table_cascade VALUES('Default', 'Second Test');
INSERT INTO Table_set_null VALUES('Restrict', 'Second Test');

DELETE FROM F_Key
    WHERE Attribute = 'Second Test';

/*
MariaDB [HW_CONSTRAINTS_PART3]> SELECT * FROM Table_cascade;
+------------+--------------+
| Attribute1 | Attribute2   |
+------------+--------------+
| Cascade    | After Update |
+------------+--------------+
1 row in set (0.00 sec)

MariaDB [HW_CONSTRAINTS_PART3]> SELECT * FROM Table_set_null;
+------------+------------+
| Attribute1 | Attribute2 |
+------------+------------+
| Restrict   | NULL       |
| Set null   | NULL       |
+------------+------------+
2 rows in set (0.00 sec)
*/
```

3.5.3 Constructing and Populating a New Example

3.5.3.1 Construction

- Remember, we start by creating a schema and tables inside of it.
- What if foreign keys are mutually dependent? What if we have something like:



Then note that we cannot create both tables as pictured directly, as PROF requires DEPARTMENT to exist, to have a foreign key referencing it, and similarly for DEPARTMENT: it is an egg and chicken situation! Hence, we have to first create a table without the foreign key, and then add it later on, as described below:

```sql
/* code/sql/HW_ProfExample.sql */
CREATE TABLE PROF (
  Login VARCHAR(25) PRIMARY KEY,
  NAME VARCHAR(25),
  Department CHAR(5)
);

CREATE TABLE DEPARTMENT (
  Code CHAR(5) PRIMARY KEY,
  NAME VARCHAR(25),
  Head VARCHAR(25),
  FOREIGN KEY (Head) REFERENCES PROF (LOGIN) ON UPDATE CASCADE
);

ALTER TABLE PROF
  ADD FOREIGN KEY (Department) REFERENCES DEPARTMENT (Code);

```

HW_ProfExample.sql[27]

Note the structure of the **ALTER TABLE** command:

- ... **KEY** `Department` **REFERENCES** `Code;` ⇒ error
- ... **KEY** `(Department)` **REFERENCES** `(Code);` ⇒ error
- ... **KEY** `PROF(Department)` **REFERENCES** `DEPARTMENT(Code);` ⇒ ok

```sql
CREATE TABLE STUDENT (
  Login VARCHAR(25) PRIMARY KEY,
  NAME VARCHAR(25),
```

[27] https://rocketgit.com/user/caubert/CSCI_3410/source/tree/branch/master/blob/notes/code/sql/HW_ProfExample.sql

```
  Registered DATE,
  Major CHAR(5),
  FOREIGN KEY (Major) REFERENCES DEPARTMENT (Code)
);

CREATE TABLE GRADE (
  Login VARCHAR(25),
  Grade INT,
  PRIMARY KEY (LOGIN, Grade),
  FOREIGN KEY (LOGIN) REFERENCES STUDENT (LOGIN)
);

```

HW_ProfExample.sql[28]

On a side note, note that we do not have the same difficulty when inserting a value in a table that contains a foreign key referencing itself: it is accepted to insert a value that is referencing itself, as illustrated below.

```
CREATE TABLE TEST (
  ID INT PRIMARY KEY,
  Reference INT,
  FOREIGN KEY (Reference) REFERENCES TEST (ID)
);

INSERT INTO TEST
VALUES (
  1,
  1);

```

HW_FK_Self_Reference.sql[29]

3.5.3.2 Populating

We can insert multiple values at once:

```
INSERT INTO DEPARTMENT
VALUES (
  "MATH",
  "Mathematics",
  NULL),
(
  "CS",
  "Computer
    Science",
```

[28] https://rocketgit.com/user/caubert/CSCI_3410/source/tree/branch/master/blob/notes/code/sql/HW_ProfExample.sql

[29] https://rocketgit.com/user/caubert/CSCI_3410/source/tree/branch/master/blob/notes/code/sql/HW_FK_Self_Reference.sql

```
  NULL);

```

HW_ProfExample.sql[30]

We can specify which attributes we are giving:

```
INSERT INTO DEPARTMENT (
  Code,
  Name)
VALUES (
  "CYBR",
  "Cyber Secturity");

```

HW_ProfExample.sql[31]

And we can even specify the order (even the trivial one):

```
INSERT INTO PROF (
  LOGIN,
  Department,
  Name)
VALUES (
  "caubert",
  "CS",
  "Clément Aubert");

INSERT INTO PROF (
  LOGIN,
  Name,
  Department)
VALUES (
  "aturing",
  "Alan Turing",
  "CS"),
(
  "perdos",
  "Paul
    Erdős",
  "MATH"),
(
  "bgates",
  "Bill Gates",
  "CYBR");

INSERT INTO STUDENT (
  LOGIN,
```

[30]https://rocketgit.com/user/caubert/CSCI_3410/source/tree/branch/master/blob/notes/code/sql/HW_ProfExample.sql

[31]https://rocketgit.com/user/caubert/CSCI_3410/source/tree/branch/master/blob/notes/code/sql/HW_ProfExample.sql

```
  Name,
  Registered,
  Major)
VALUES (
  "jrakesh",
  "Jalal Rakesh",
  DATE "2017-12-01",
  "CS"),
(
  "svlatka",
  "Sacnite Vlatka",
  "2015-03-12",
  "MATH"),
(
  "cjoella",
  "Candice Joella",
  "20120212",
  "CYBR"),
(
  "aalyx",
  "Ava Alyx",
  20121011,
  "CYBR"),
(
  "caubert",
  "Clément Aubert",
  NULL,
  "CYBR");

INSERT INTO GRADE
VALUES (
  "jrakesh",
  3.8),
(
  "svlatka",
  2.5);

```

HW_ProfExample.sql[32]

(Note the date literals)

By default, the values that are not given are set to their respective **DEFAULT** values.

```
/* code/sql/HW_DefaultTest.sql */
INSERT INTO TEST (
  TestB)
VALUES (
  1);
```

[32] https://rocketgit.com/user/caubert/CSCI_3410/source/tree/branch/master/blob/notes/code/sql/HW_ProfExample.sql

```sql

SELECT *
FROM TEST;

-- The value of TestA is set to "A",
--      all the other values are set to NULL.
```

HW_DefaultTest.sql[33]

3.5.4 A Bit More on Foreign Keys

Note that we can create foreign keys to primary keys made of multiple attributes, and to the primary key of the table we are currently creating.

```sql
/* code/sql/HW_AdvancedFK.sql */
CREATE TABLE T1 (
  A1 INT,
  A2 INT,
  B INT,
  PRIMARY KEY (A1, A2)
);

CREATE TABLE T2 (
  A1 INT,
  A2 INT,
  B1 INT PRIMARY KEY,
  B2 INT,
  -- We can create a "pair" of foreign key in one line, as
  -- follows:
  FOREIGN KEY (A1, A2) REFERENCES T1 (A1, A2),
  -- We can create a foreign key that references the primary
  -- key of the table we are currently creating, and name
  -- it, as follows:
  CONSTRAINT My_pk_to_T1 FOREIGN KEY (B2) REFERENCES T2 (B1)
);
```

HW_AdvancedFK.sql[34]

In the example, we are also naming our foreign key. The benefit of naming our fk constraint is that, if we violate it, for instance with

```sql
INSERT INTO T2 VALUES (1, 1, 1, 3);
```

then the name of the constraint (here "My_pk_to_T1") will be displayed in the error message:

```
Cannot add or update a child row: a foreign key constraint fails
 ↪  (`db_9_9837c1`.`t2`, CONSTRAINT
`My_pk_to_T1` FOREIGN KEY (`B2`) REFERENCES `t2` (`B1`))
```

33 https://rocketgit.com/user/caubert/CSCI_3410/source/tree/branch/master/blob/notes/code/sql/HW_DefaultTest.sql

34 https://rocketgit.com/user/caubert/CSCI_3410/source/tree/branch/master/blob/notes/code/sql/HW_AdvancedFK.sql

3.6 A First Look at Conditions

Order of clauses does not matter, not even for optimization purpose.

```
UPDATE <table>
SET <attribute1> = <value1>, <attribute2> = <value2>, …
WHERE <condition>;
```

```
SELECT <attribute list, called projection attributes>
FROM <table list>
WHERE <condition>;
```

```
DELETE FROM <table list>
WHERE <condition>;
```

Conditions can

- use the comparison operators:
  - =, equal to
  - >, greater than,
  - < less than,
  - >= greater than or equal to,
  - <= less than or equal to,
  - <> not equal to.
- be compounded:
  - `condition1 AND condition2`
  - `condition1 OR condition2`
  - `NOT condition`
  - Usage of parenthesis is possible
- be trivial or even absent,
- use regular expressions:
  - uses the expression **LIKE**,
  - escape character is \,
  - _ will match one character (any character), % will match any number of character,
  - advanced regular expression possible using the **REGEXP** keyword.

```
SELECT LOGIN
FROM STUDENT;

UPDATE
  DEPARTMENT
SET Head = "aturing"
WHERE Code = "MATH";

UPDATE
  DEPARTMENT
SET Head = "bgates"
WHERE Code = "CS"
  OR Code = "CYBR";
```

```
SELECT LOGIN
FROM STUDENT
WHERE NOT Major = "CYBR";

SELECT LOGIN,
  Name
FROM PROF
WHERE Department = "CS";

SELECT LOGIN
FROM STUDENT
WHERE Major = "CYBR"
  AND Registered > DATE "20121001";

SELECT LOGIN
FROM STUDENT
WHERE Name LIKE "Ava%";

SELECT Name
FROM PROF
WHERE LOGIN LIKE "_aubert";
```

HW_ProfExample.sql[35]

Note that **LIKE** is by default case-insensitive, both in MariaDB[36] and in MySQL[37]. The `COLLATE` operator can be used to force the search to be case-sensitive, as well as **LIKE** `BINARY`.

3.7 Three-Valued Logic

Cf. (Elmasri and Navathe 2010, 5.1.1), (Elmasri and Navathe 2015, 7.1.1)

The Boolean logic in SQL is three-valued: a statement can be `true`, `false` or `unknown`. If you pick the following two commands:

```
/* code/sql/HW_DefaultTest.sql */
SELECT *
FROM TEST
WHERE TestA = "A";

SELECT *
FROM TEST
```

[35] https://rocketgit.com/user/caubert/CSCI_3410/source/tree/branch/master/blob/notes/code/sql/HW_ProfExample.sql

[36] https://mariadb.com/kb/en/like/

[37] https://dev.mysql.com/doc/refman/8.0/en/case-sensitivity.html

```
WHERE TestA <> "A";

```

HW_DefaultTest.sql[38]

you may believe that they will capture all the tuples in the `TEST` table, as the value for `TestA` is either `"A"` or not `"A"`, but you would be wrong. If the value of `TestA` is **`NULL`**, then both conditions would fail, as SQL cannot say that the value is or is not `"A"`: it is simply undefined!

3.7.1 Meaning of `NULL`

`NULL` is

1. Unknown value ("*Nobody knows*")

    What is the date of birth of Jack the Ripper[39]?

    Does P equal NP?[40]

2. Unavailable / Withheld ("*I do not have that information with me at the moment*")

    What is the number of english spies in France?

    What is the VIN of your car?

    What is the identity of the Tiananmen Square person?

3. Not Applicable ("*Your question does not make sense*")

    What is the US SSN of a French person?

    What is the email address of an author of the XIX$^{\text{th}}$ century?

3.7.2 Comparison with Unknown Values

If **`NULL`** is involved in a comparison, the result evaluates to "Unknown."

AND	T	F	U
T	T	F	U
F	F	F	F
U	U	F	U

OR	T	F	U
T	T	T	T
F	T	F	U
U	T	U	U

[38] https://rocketgit.com/user/caubert/CSCI_3410/source/tree/branch/master/blob/notes/code/sql/HW_DefaultTest.sql
[39] https://en.wikipedia.org/wiki/Jack_the_Ripper
[40] https://en.wikipedia.org/wiki/List_of_unsolved_problems_in_computer_science

NOT	
T	F
F	T
U	U

You can test if a value is `NULL` with `IS NULL`.

```sql
INSERT INTO DEPARTMENT
VALUES (
  "Hist",
  "History",
  NULL);

SELECT *
FROM DEPARTMENT
WHERE Head IS NULL;

SELECT *
FROM DEPARTMENT
WHERE Head IS NOT NULL;

SELECT COUNT(*)
FROM GRADE
WHERE Grade IS NULL;
```

HW_ProfExample.sql[41]

Note that you can not use `IS` to compare values: this key word is reserved to test if a value is (not) `NULL`, and nothing else.

This means that if you want to capture all the tuples, you cannot write

```sql
/* code/sql/HW_DefaultTest.sql */
SELECT *
FROM TEST
WHERE TestA = "A";

SELECT *
FROM TEST
WHERE TestA <> "A";
```

HW_DefaultTest.sql[42]

but should have something like

[41] https://rocketgit.com/user/caubert/CSCI_3410/source/tree/branch/master/blob/notes/code/sql/HW_ProfExample.sql

[42] https://rocketgit.com/user/caubert/CSCI_3410/source/tree/branch/master/blob/notes/code/sql/HW_DefaultTest.sql

```sql
/* code/sql/HW_DefaultTest.sql */
SELECT *
FROM TEST
WHERE TestA IS NULL;

SELECT *
FROM TEST
WHERE TestA IS NOT NULL;

```

HW_DefaultTest.sql[43]

3.7.3 Trivia

There are no `if…then…else` statements in `SQL`, but you can do something similar with **CASE** (cf. https://dev.mysql.com/doc/refman/8.0/en/case.html). However, note that `SQL` is probably *not* the right place to try to control the flow of execution.

This probably depends on the system a lot, but one could wonder if MySQL uses some form of short-cut evaluation when comparing with **NULL**. Unfortunately, even with three times (!) the verbose option, MySQL does not give more insight as to whenever it drops comparing values once a NULL was encountered (cf. https://dev.mysql.com/doc/refman/8.0/en/mysql-command-options.html#option_mysql_verbose, you can log-in using `mysql -u testuser -p --password=password -v -v -v` to activate the most verbose mode). Normally, **EXPLAIN** (https://dev.mysql.com/doc/refman/8.0/en/explain.html) should be useful in answering this question, but failed to answer it as well.

3.8 Various Tools

For **DISTINCT**, **ALL** and **UNION**, cf. (Elmasri and Navathe 2010, 4.3.4) or (Elmasri and Navathe 2015, 6.3.4). For **ORDER BY**, cf. (Elmasri and Navathe 2010, 4.3.6) or (Elmasri and Navathe 2015, 6.3.6). For aggregate functions, cf. (Elmasri and Navathe 2010, 5.1.7) or (Elmasri and Navathe 2015, 7.1.7).

3.8.1 AUTO_INCREMENT

Something that is not exactly a constraint, but that can be used to "qualify" domains, is the **AUTO_INCREMENT** feature of MySQL. Cf. https://dev.mysql.com/doc/refman/8.0/en/example-auto-increment.html, you can have MySQL increment a particular attribute (most probably intended to be your primary key, or some form of counter) for you.

A typical example could be:

[43] https://rocketgit.com/user/caubert/CSCI_3410/source/tree/branch/master/blob/notes/code/sql/HW_DefaultTest.sql

```sql
/* code/sql/HW_AutoIncrement.sql */
CREATE TABLE PERSON (
  PersonID INT AUTO_INCREMENT,
  Name VARCHAR(255),
  PRIMARY KEY (PersonID)
);

INSERT INTO PERSON (
  Name)
VALUES (
  'Lars'),
(
  'Kristina'),
(
  'Sophie');

SELECT *
FROM PERSON;
```

HW_AutoIncrement.sql[44]

This way, the burden of having to keep track of the persons' ids is left to the program, and not to the person inserting data in the table.

3.8.2 Transactions

We can save the current state, and start a series of transactions, with the command

```
START TRANSACTION;
```

All the commands that follows are "virtually" executed: you can undo them all using

```
ROLLBACK;
```

This puts you back in the state you were in before starting the transaction. If you want all the commands you typed in-between to be actually enforced, you can use the command

```
COMMIT;
```

Nested transactions are technically possible, but they are counter-intuitive and should be avoided, cf. https://www.sqlskills.com/blogs/paul/a-sql-server-dba-myth-a-day-2630-nested-transactions-are-real/.

[44]https://rocketgit.com/user/caubert/CSCI_3410/source/tree/branch/master/blob/notes/code/sql/HW_AutoIncrement.sql

3.8.3 DISTINCT / ALL

The result of a **SELECT** query, for instance, is a table, and SQL treats tables as multi-set, hence there can be repetitions in the result of a query, but we can remove them:

```
SELECT DISTINCT Major FROM STUDENT;
```

The default behaviour is equivalent to specifying **ALL**, and it display the duplicates. In this case, it would be

```
> SELECT Major FROM STUDENT;
+-------+
| Major |
+-------+
| CS    |
| CYBR  |
| CYBR  |
| CYBR  |
| MATH  |
+-------+
```

3.8.4 UNION

Set-theoretic operations are available as well. For instance, one can use:

```
(SELECT Login FROM STUDENT) UNION (SELECT Login FROM PROF);
```

to collect all the logins from both tables.

There is also INTERSECT and EXCEPT in the specification, but MySQL does not implement them (cf. https://en.wikipedia.org/wiki/Comparison_of_relational_database_management_systems#Database_capabilities).

3.8.5 ORDER BY

You can have **ORDER BY** specifications:

```
SELECT LOGIN
FROM GRADE
WHERE Grade > 2.0
ORDER BY Grade;

SELECT LOGIN
FROM GRADE
WHERE Grade > 2.0
ORDER BY Grade DESC;

SELECT LOGIN,
  Major
FROM STUDENT
ORDER BY Major,
```

```
  Name;

```

HW_ProfExample.sql[45]

`ORDER BY` order by ascending order by default.

3.8.6 Aggregate Functions

You can use `MAX`, `SUM`, `MIN`, `AVG`, `COUNT` to peform simple operations.

`SELECT MAX(Registered) FROM STUDENT;`

returns the "greatest" date of registration of a student, i.e., the date of the latest registration.

`SELECT COUNT(Name) FROM STUDENT;`

returns the number of names, i.e., the number of students.

`SELECT COUNT(DISTINCT Name) FROM STUDENT;`

returns the number of *different names* (which in this case is the same as the number of names, since we have no homonyms).

Note that `AVG` returns the average of all **non-**`NULL` values, as we can see on the following example:

```sql
/* code/sql/HW_Avg.sql */
CREATE TABLE TEST (
  Test INT
);

INSERT INTO TEST
VALUES (
  NULL),
(
  0),
(
  10);

SELECT AVG(Test)
FROM TEST;

-- Returns 5.0
```

HW_Avg.sql[46]

The same goes for e.g. MAX:

[45] https://rocketgit.com/user/caubert/CSCI_3410/source/tree/branch/master/blob/notes/code/sql/HW_ProfExample.sql

[46] https://rocketgit.com/user/caubert/CSCI_3410/source/tree/branch/master/blob/notes/code/sql/HW_Avg.sql

```sql
/* code/sql/HW_Max.sql */
CREATE TABLE TEST (
  A DATE
);

INSERT INTO TEST
VALUES (
  DATE "2020-01-01"),
(
  DATE "2019-01-01"),
(
  NULL);

SELECT MAX(A)
FROM TEST;

-- Returns 2020-01-01
```

HW_Max.sql[47]

3.8.7 Aliases for Columns

We can use aliases for the columns. Compare

```
SELECT Login FROM PROF;
+----------+
| Login    |
+----------+
| aturing |
| caubert |
| bgates   |
| perdos   |
+----------+
```

with

```
SELECT Login AS "Username" FROM PROF;
+-----------+
| Username |
+-----------+
| aturing   |
| caubert   |
| bgates    |
| perdos    |
+-----------+
```

Aliases can also be used on table names. Adiases for columns are a helpful way of describing the result of the query, while alias on table have a specific purpose that will be clearer as we study select-project-join queries.

[47] https://rocketgit.com/user/caubert/CSCI_3410/source/tree/branch/master/blob/notes/code/sql/HW_Max.sql

3.9 More Select Queries

For select-project-join, cf. (Elmasri and Navathe 2010, 4.3.1) or (Elmasri and Navathe 2015, 6.3.1). For aliases, cf. (Elmasri and Navathe 2010, 4.3.2) or (Elmasri and Navathe 2015, 6.3.2), For nested queries, cf. (Elmasri and Navathe 2010, 5.1.2) or (Elmasri and Navathe 2015, 7.1.2).

3.9.1 Select-Project-Join

```
SELECT LOGIN
FROM PROF,
  DEPARTMENT
WHERE DEPARTMENT.Name = "Mathematics"
  AND Department = Code;

```

HW_ProfExample.sql[48]

- `Department.Name = 'Mathematics'` is the selection condition
- `Department = Code` is the join condition, because it combines two tuples.
- Why do we use the fully qualified name attribute for `Name`?
- We have to list all the tables we want to consult, even if we use fully qualified names.

```
SELECT Name
FROM STUDENT,
  GRADE
WHERE Grade > 3.0
  AND STUDENT.Login = GRADE.Login;

```

HW_ProfExample.sql[49]

- `Grade > 3.0` is the selection condition
- `STUDENT.Login = GRADE.Login` is the join condition

We can have two join conditions!

```
SELECT PROF.Name
FROM PROF,
  DEPARTMENT,
  STUDENT
WHERE STUDENT.Name = "Ava Alyx"
  AND STUDENT.Major = DEPARTMENT.Code
  AND DEPARTMENT.Head = PROF.Login;

```

[48] https://rocketgit.com/user/caubert/CSCI_3410/source/tree/branch/master/blob/notes/code/sql/HW_ProfExample.sql

[49] https://rocketgit.com/user/caubert/CSCI_3410/source/tree/branch/master/blob/notes/code/sql/HW_ProfExample.sql

HW_ProfExample.sql[50]

Note that for the kind of join we are studiying (called "inner joins"), the order does not matter[51].

In Problem 3.3 (Duplicate rows in SQL), we saw that SQL was treating tables as multi-sets, i.e., repetitions are allowed. This can lead to strange behaviour when performing Select-Project-Join queries. Consider the following example:

3.9.2 Aliasing Tuples

We can use aliases on tables to shorten the previous query:

```
SELECT PROF.Name
FROM PROF,
  DEPARTMENT,
  STUDENT AS B
WHERE B.Name = "Ava Alyx"
  AND B.Major = DEPARTMENT.Code
  AND DEPARTMENT.Head = PROF.Login;

```

HW_ProfExample.sql[52]

We can use multiple aliases to make it even shorter (but less readable):

```
SELECT A.Name
FROM PROF AS A,
  DEPARTMENT AS B,
  STUDENT AS C
WHERE C.Name = "Ava Alyx"
  AND C.Major = B.Code
  AND B.Head = A.Login;

```

HW_ProfExample.sql[53]

For those two, aliases are convenient, but not required to write the query. In some cases, we cannot do without aliases. For instance if we want to compare two rows in the same table:

```
SELECT Other.Login
FROM GRADE AS Mine,
  GRADE AS Other
WHERE Mine.Login = "aalyx"
  AND Mine.Grade < Other.Grade;

```

[50] https://rocketgit.com/user/caubert/CSCI_3410/source/tree/branch/master/blob/notes/code/sql/HW_ProfExample.sql
[51] https://stackoverflow.com/q/9614922
[52] https://rocketgit.com/user/caubert/CSCI_3410/source/tree/branch/master/blob/notes/code/sql/HW_ProfExample.sql
[53] https://rocketgit.com/user/caubert/CSCI_3410/source/tree/branch/master/blob/notes/code/sql/HW_ProfExample.sql

HW_ProfExample.sql[54]

Generally, when you want to perform a join *within the same table*, then you have to "make two copies of the tables" and name them differently using aliases. Let us try to write a query that answers the question

> What are the login of the professors that have the same department as the professor whose login is caubert?

We need a way of distinguising between the professors we are projecting on (the one whole login is caubert) and the one we are joining with (the ones that have the same department). This can be done using something like:

```
SELECT JOINT.Login
FROM PROF AS PROJECT,
  PROF AS JOINT
WHERE PROJECT.Login = "caubert"
  AND PROJECT.Department = JOINT.Department;

```

HW_ProfExample.sql[55]

Note that we are "opening up two copies of the PROF tables", and naming them differently (`PROJECT` and `JOINT`).

Another (improved) example of a similar query is

```
SELECT Fellow.Name AS "Fellow of Ava"
FROM STUDENT AS Ava,
  STUDENT AS Fellow
WHERE Ava.Name = "Ava Alyx"
  AND Fellow.Major = Ava.Major
  AND NOT Fellow.Login = Ava.Login;

```

HW_ProfExample.sql[56]

A couple of remarks about this query:

- At the beginning of the query, `AS "Fellow of Ava"` is *another* kind of aliasing, mentioned in a previous section.
- In the condition, `NOT Fellow.Login = Ava.Login` guarantees that we will not select Ava *again*, and exclude her from the results (Ava is not supposed to be a fellow of herself).

[54] https://rocketgit.com/user/caubert/CSCI_3410/source/tree/branch/master/blob/notes/code/sql/HW_ProfExample.sql
[55] https://rocketgit.com/user/caubert/CSCI_3410/source/tree/branch/master/blob/notes/code/sql/HW_ProfExample.sql
[56] https://rocketgit.com/user/caubert/CSCI_3410/source/tree/branch/master/blob/notes/code/sql/HW_ProfExample.sql

- In the (unlikely, but possible) case of an homonym, writing **NOT** `Fellow.Name = Me.Name;` instead of **NOT** `Fellow.Login = Ava.Login` would prevent the homonym from occuring in the results.
- In the condition, substituting **AND NOT** `Me = Fellow` by **NOT** `Fellow.Login = Ava.Login` would *not* work: you have to compare attributes of the tuples, not the tuples.

3.9.3 Nested Queries

Let us look at a first example

```
SELECT LOGIN
FROM GRADE
WHERE Grade > (
    SELECT AVG(Grade)
    FROM GRADE);

```

HW_ProfExample.sql[57]

A nested query is made of an outer query (**SELECT** `Login`...) and an inner query (**SELECT** `AVG(Grade)`...). Note that the inner query does not terminate with a `;`.

Logical operators such as **ALL** or **IN** can be used in nested queries. To learn more about those operators, refer to https://www.w3schools.com/sql/sql_operators.asp.

An example could be

```
SELECT LOGIN
FROM GRADE
WHERE Grade >= ALL (
    SELECT Grade
    FROM GRADE
    WHERE Grade IS NOT NULL);

```

HW_ProfExample.sql[58]

Note that

- We have to use >=, and not >, since no grade is strictly greater than itself.
- The part **IS NOT NULL** is needed: otherwise, if one of the grade is **NULL**, then the comparison would yelds “unknown”, and no grade would be greater than all of the others.
- This query could be simplified, using `MAX`:

```
SELECT LOGIN
FROM GRADE
WHERE Grade >= (
    SELECT MAX(Grade)
```

57 https://rocketgit.com/user/caubert/CSCI_3410/source/tree/branch/master/blob/notes/code/sql/HW_ProfExample.sql

58 https://rocketgit.com/user/caubert/CSCI_3410/source/tree/branch/master/blob/notes/code/sql/HW_ProfExample.sql

```
    FROM GRADE);

```

HW_ProfExample.sql[59]

Answering the question

> What are the logins of the professors belonging to a department that is the major of at least one student whose name ends with an "a"?

–that sounds like the what would ask a police officer in a whodunit– could be answer using

```
SELECT LOGIN
FROM PROF
WHERE DEPARTMENT IN (
    SELECT Major
    FROM STUDENT
    WHERE LOGIN LIKE "%a");

```

HW_ProfExample.sql[60]

For this query, we could not use =, since more than one major could be returned.

Furthermore, nested query that uses = can often be rewritten without being nested. For instance,

```
SELECT LOGIN
FROM PROF
WHERE DEPARTMENT = (
    SELECT Major
    FROM STUDENT
    WHERE LOGIN = "cjoella");

```

HW_ProfExample.sql[61]

becomes

```
SELECT PROF.Login
FROM PROF,
  STUDENT
WHERE DEPARTMENT = Major
  AND STUDENT.Login = "cjoella";

```

[59] https://rocketgit.com/user/caubert/CSCI_3410/source/tree/branch/master/blob/notes/code/sql/HW_ProfExample.sql

[60] https://rocketgit.com/user/caubert/CSCI_3410/source/tree/branch/master/blob/notes/code/sql/HW_ProfExample.sql

[61] https://rocketgit.com/user/caubert/CSCI_3410/source/tree/branch/master/blob/notes/code/sql/HW_ProfExample.sql

HW_ProfExample.sql[62]

Conversly, you can sometimes write select-project-join as nested queries For instance,

```sql
SELECT Name
FROM STUDENT,
  GRADE
WHERE Grade > 3.0
  AND STUDENT.Login = GRADE.Login;

```

HW_ProfExample.sql[63]

becomes

```sql
SELECT Name
FROM STUDENT
WHERE LOGIN IN (
    SELECT LOGIN
    FROM GRADE
    WHERE Grade > 3.0);

```

HW_ProfExample.sql[64]

3.10 Procedures

A "stored" procedure is a SQL function statements that can take arguments and may be called from another part of your program. Stated differently, a procedure is a serie of statements stored in a schema, that can easily be executed repeatedly, cf. https://dev.mysql.com/doc/refman/8.0/en/create-procedure.html or https://mariadb.com/kb/en/library/create-procedure/.

Imagine we have the following:

```sql
/* code/sql/HW_ProcedureExamples.sql */
CREATE TABLE STUDENT (
  Login INT PRIMARY KEY,
  NAME VARCHAR(30),
  Major VARCHAR(30),
  Email VARCHAR(30)
);

INSERT INTO STUDENT
VALUES (
  123,
  "Test A",
```

[62]https://rocketgit.com/user/caubert/CSCI_3410/source/tree/branch/master/blob/notes/code/sql/HW_ProfExample.sql

[63]https://rocketgit.com/user/caubert/CSCI_3410/source/tree/branch/master/blob/notes/code/sql/HW_ProfExample.sql

[64]https://rocketgit.com/user/caubert/CSCI_3410/source/tree/branch/master/blob/notes/code/sql/HW_ProfExample.sql

```
  "CS",
  "a@a.edu"),
(
  124,
  "Test B",
  "IT",
  "b@a.edu"),
(
  125,
  "Test C",
  "CYBR",
  "c@a.edu");

```

HW_ProcedureExamples.sql[65]

SQL is extremely litteral: when it reads the delimiter `;`, it *must* execute the command that was shared. But a procedure, being composed of commands, will contain the `;` symbol. To "solve" this (weird) issue, and be able to define a procedure, we have to "temporarily alter the language", using `DELIMITER //` that makes the delimiter being `//` instead of `;`[66].

In any case, we can then define and execute a simpe procedure called `STUDENTLIST` as follows:

```
DELIMITER $$
CREATE PROCEDURE STUDENTLIST ()
BEGIN
  SELECT *
  FROM STUDENT;

-- This ";" is not the end of the procedure definition!
END;
$$
-- This is the delimiter that marks the end of the procedure
--                  definition.
DELIMITER ;

-- Now, we want ";" to be the "natural" delimiter again.
CALL STUDENTLIST ();

-- Now, we want ";" to be the "natural" delimiter again.
CALL STUDENTLIST ();

```

[65]https://rocketgit.com/user/caubert/CSCI_3410/source/tree/branch/master/blob/notes/code/sql/HW_ProcedureExamples.sql

[66]The symbols `$$` are often used too, and the documentation, at https://dev.mysql.com/doc/refman/8.0/en/stored-programs-defining.html, reads:

> You can redefine the delimiter to a string other than `//` and the delimiter can consist of a single character or multiple characters. You should avoid the use of the backslash (`\`) character because that is the escape character for MySQL.

The minus sign twice is also a poor choice, since it is used for commenting.

HW_ProcedureExamples.sql[67]

A procedure an also take arguments, and an example could be:

```
DELIMITER $$
CREATE PROCEDURE STUDENTLOGIN (
  NameP VARCHAR(30)
)
BEGIN
  SELECT LOGIN
  FROM STUDENT
  WHERE NameP = Name;

END;
$$
DELIMITER ;

SHOW CREATE PROCEDURE STUDENTLOGIN;

-- This display information about the procedure just created.
--                     We can pass quite naturally an argument
--   to
--       our
--                    procedure.
CALL STUDENTLOGIN ("Test A");

```

HW_ProcedureExamples.sql[68]

3.11 Triggers

A trigger is a series of statements stored in a schema that can be automatically executed whenever a particular event in the schema occurs. Triggers are extremely powerfull, and are a way of automating part of the work in your database. In MariaDB, you could have the following program.

Imagine we have the following:

```
CREATE TABLE STUDENT (
  Login VARCHAR(30) PRIMARY KEY,
  Average FLOAT
);

CREATE TABLE GRADE (
  Student VARCHAR(30),
  Exam VARCHAR(30),
```

[67] https://rocketgit.com/user/caubert/CSCI_3410/source/tree/branch/master/blob/notes/code/sql/HW_ProcedureExamples.sql

[68] https://rocketgit.com/user/caubert/CSCI_3410/source/tree/branch/master/blob/notes/code/sql/HW_ProcedureExamples.sql

```
  Grade INT,
  PRIMARY KEY (Student, Exam),
  FOREIGN KEY (Student) REFERENCES STUDENT (LOGIN)
);

```

HW_TriggerExample.sql[69]

We want to create a trigger that counts the number of times something was inserted in our `STUDENT` table. `SQL` supports some primitive form of variables (cf. https://dev.mysql.com/doc/refman/8.0/en/user-variables.html and https://mariadb.com/kb/en/library/user-defined-variables/). There is no "clear" form of type, https://dev.mysql.com/doc/refman/8.0/en/user-variables.html reads:

> In addition, the default result type of a variable is based on its type at the beginning of the statement. This may have unintended effects if a variable holds a value of one type at the beginning of a statement in which it is also assigned a new value of a different type. To avoid problems with this behavior, either do not assign a value to and read the value of the same variable within a single statement, or else set the variable to 0, 0.0, or '' to define its type before you use it.

In other words, `SQL` just "guess" the type of your value and go with it. Creating a simple trigger that increment a variable every time an insertion is performed in the `STUDENT` table can be done as follows:

```
SET @number_of_student = 0;

CREATE TRIGGER NUMBER_OF_STUDENT_INC
  AFTER INSERT ON STUDENT
  FOR EACH ROW
  SET @number_of_student = @number_of_student + 1;

```

HW_TriggerExample.sql[70]

Now, assume we want to create a trigger that calculates the average for us. Note that the trigger will need to manipulate two tables (`STUDENT` and `GRADE`) at the same time.

```
CREATE TRIGGER STUDENT_AVERAGE
  AFTER INSERT ON GRADE
  FOR EACH ROW -- Woh, a whole query inside our trigger!
  UPDATE STUDENT
  SET STUDENT.Average = (
  SELECT AVG(Grade)
  FROM GRADE
  WHERE GRADE.Student = STUDENT.Login)
WHERE STUDENT.Login = NEW.Student;

-- The "NEW" keyword here refers to the "new" entry
```

[69] https://rocketgit.com/user/caubert/CSCI_3410/source/tree/branch/master/blob/notes/code/sql/HW_TriggerExample.sql

[70] https://rocketgit.com/user/caubert/CSCI_3410/source/tree/branch/master/blob/notes/code/sql/HW_TriggerExample.sql

```
--                    that is being inserted by the INSERT
--         statement
--                   triggering
--                    the trigger.
```

HW_TriggerExample.sql[71]

The source code contains examples of insertion and explanations on how to witness the trigger in action.

3.12 Setting Up Your Work Environment

This part is a short tutorial to install and configure a working relational DBMS. We will proceed in 5 steps:

1. Install the required software,
2. Create a user,
3. Log-in as this user,
4. Create and populate our first database,
5. Discuss the security holes in our set-up.

3.12.1 Installation

You will install the MySQL[72] DataBase Managment System, or its community-developed fork, MariaDB[73]. Below are the instruction to install MySQL Community Edition on Windows 10 and macOS, and MariaDB on Linux-based distribution, but both are developped for every major operating system (macOS, Windows, Debian, Ubuntu, etc.): feel free to pick one or the other, it will not make a difference in this course (up to some minor aspects). MySQL is more common, MariaDB is growing, both are released under GNU General Public License[74], well-documented and free of charge for their "community" versions.

It is perfectly acceptable, and actually encouraged, to install MySQL or MariaDB on a virtual machine for this class. You can use the Windows Subsystem for Linux[75], VMware[76] or Virtual Box[77] to run a "sandboxed" environment[78] that should keep your data isolated from our experimentations.

Below are precise and up-to-date instructions, follow them carefully, read the messages displayed on your screen, make sure a step was correctly executed before moving to the next one, and everything should be all right. Also, remember:

1. Do not wait, set your system early.
2. To look for help, be detailed and clear about what you think went wrong.

[71] https://rocketgit.com/user/caubert/CSCI_3410/source/tree/branch/master/blob/notes/code/sql/HW_TriggerExample.sql
[72] https://www.mysql.com/
[73] https://mariadb.org/
[74] https://www.gnu.org/licenses/licenses.html#GPL
[75] https://docs.microsoft.com/en-us/windows/wsl/install-win10
[76] https://www.vmware.com/
[77] https://www.virtualbox.org/wiki/Downloads
[78] https://en.wikipedia.org/wiki/Sandbox_(software_development)

The following links could be useful:

- https://mariadb.com/kb/en/getting-installing-and-upgrading-mariadb/
- https://dev.mysql.com/doc/refman/8.0/en/mysql-installer-workflow.html, and particularly the page on windows installation[79] and the page on Linux installation using package-managers[80],
- https://dev.mysql.com/doc/refman/8.0/en/connecting-disconnecting.html
- https://www.linode.com/docs/databases/mysql/how-to-install-mysql-on-debian-8/

3.12.1.1 Installing MySQL on Windows 10

1. Visit https://dev.mysql.com/downloads/installer/, click on "Download" next to

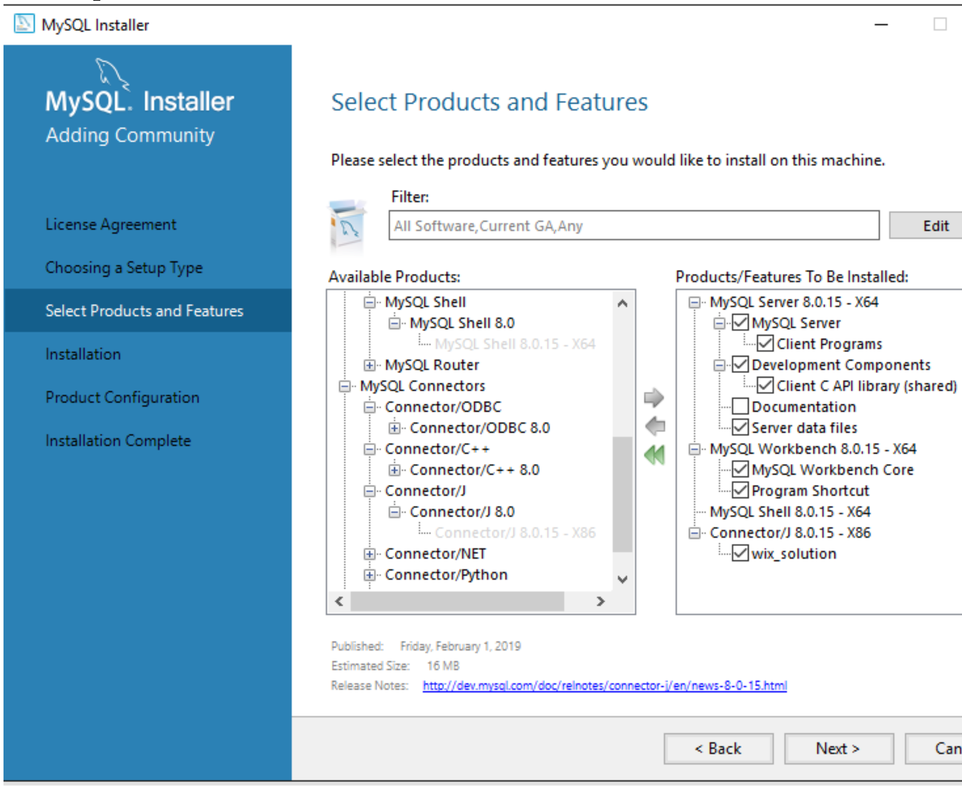
   > Windows (x86, 32-bit), MSI Installer XXX YYY (mysql-installer-web-community-XXX.msi)

   where XXX is a number version (e.g., 8.0.13.0.), and YYY is the size of the file (e.g., 16.3M). On the next page, click on the (somewhat hidden) "No thanks, just start my download." button.

2. Save the "mysql-installer-web-community-XXX.msi" file, and open it. If there is an updated version of the installer available, agree to download it. Accept the license term.

3. We will now install the various components needed for this class, leaving all the choices by defaults. This means that you need to do the following:

   a) Leave the first option on "Developer Default" and click on "Next", or click on "Custom", and select the following:



[79]https://dev.mysql.com/doc/refman/8.0/en/windows-installation.html
[80]https://dev.mysql.com/doc/refman/8.0/en/linux-installation-native.html

b) Click on “Next” even if you do not meet all the requirements
c) Click on “Execute”. The system will download and install several softwares (this may take some time).
d) Click on “Next” twice, leave “Type and Networking” on “Standalone MySQL Server / Classic MySQL Replication” and click “Next”, and leave the next options as they are (unless you know what you do and want to change the port, for instance) and click on “Next”.
e) You now need to choose a password for the MySQL root account. It can be anything, just make sure to memorize it. Click on “Next”.
f) On the “Windows Service” page, leave everything as it is and click on “Next”.
g) On the “Plugins and Extensions” page, leave everything as it is and click on “Next”.
h) Finally, click “Execute” on the “Apply Configuration” page, and then on “Finish”.
i) Click on “Cancel” on the “Product Configuration” page and confirm that you do not want to add products: we only need to have MySQL Server XXX configured.

4. We now want to make sure that MySQL is running: launch Windows’ “Control Panel”, then click on “Administrative Tools”, and on “Services”. Look for “MySQLXX”, its status should be “Running”. If it is not, right-click on it and click on “Start”.

5. Open a command prompt (search for `cmd`, or use PowerShell[81]) and type

   ```
   cd "C:\Program Files\MySQL\MySQL Server 8.0\bin"
   ```

   If this command fails, it is probably because the version number changed: open the file explorer, go to `C:\Program Files\MySQL\`, look for the right version number, and update the command accordingly.

   Then, enter

   ```
   mysql -u root -p
   ```

   and enter the password you picked previously for the root account. You are now logged as root in your database management system, you should see a brief message, followed by a prompt

   ```
   mysql >
   ```

6. Now, move on to “Creating a User”.

3.12.1.2 Installing MySQL on macOS

The instructions are almost the same as for Windows. Read https://dev.mysql.com/doc/refman/8.0/en/osx-installation-pkg.html and download the file from https://dev.mysql.com/downloads/mysql/ once you selected “macOS” as your operating system. Install it, leaving everything by default but adding a password (refer to the instructions for windows). Then, open a command-line interface (the terminal), enter

```
mysql -u root -p
```

and enter the password you picked previously for the root account. You are now logged as root in your database management system, you should see a brief message, followed by a prompt

[81] https://docs.microsoft.com/en-us/powershell/scripting/getting-started/getting-started-with-windows-powershell?view=powershell-6

```
mysql >
```

Now, move on to "Creating a User".

3.12.1.3 Installing MariaDB on Linux

1. Install, through your standard package management system (`apt` or `aptitude` for debian-based systems, `pacman` for Arch Linux, etc.), the packages `mysql-client` and `mysql-server` (or `default-mysql-client` and `default-mysql-server`) as well as their dependencies[82].
2. Open a terminal and type

   ```
   /etc/init.d/mysql status
   ```

   or, as root,

   ```
   service mysql status
   ```

   to see if MySQL is running: if you read something containing

   ```
   Active: active (running)
   ```

   then you can move on to the next step, otherwise run (as root)

   ```
   service mysqld start
   ```

   and try again.
3. As root, type in your terminal

   ```
   mysql_secure_installation
   ```

   You will be asked to provide the current password for the root MySQL user: this password has not be defined yet, so just hit "Enter". You will be asked if you want to set a new password (that you can freely chose, just make sure to memorize it). Then, answer "n" to the question "Remove anonymous users?", "Y" to "Disallow root login remotely?", "n" to "Remove test database and access to it?" and finally "Y" to "Reload privilege tables now?".
4. Still as root, type in your terminal

   ```
   mysql -u root -p
   ```

   and enter the password you picked previously for the root account. You are now logged as root in your database management system: you should see a brief message, followed by a prompt

   ```
   MariaDB [(none)]>
   ```

5. Now, move on to "Creating a User".

[82]Yes, the package is called `mysql-server`, but it actually install the package `mariadb-server-10.3` or higher… So do not be confused: *we are, indeed, installing MariaDB*!

3.12.2 Creating a User

This step will create a non-root user[83] and grant it some rights. Copy-and-paste or type the following three commands, one by one (that is, enter the first one, hit "enter", enter the second, hit "enter", etc.). This step will create a non-root user[84] and grant it some rights. Copy-and-paste or type the following three commands, one by one (that is, enter the first one, hit "enter", enter the second, hit "enter", etc.).

We first create a new user called `testuser` on our local installation, and give it the password `password`:

```
CREATE USER 'testuser'@'localhost' IDENTIFIED BY 'password';
```

Then, we grant the user all the privileges on the databases whose name starts with `HW_`:

```
GRANT ALL PRIVILEGES ON  `HW\_%` . * TO 'testuser'@'localhost';
```

Be careful: backticks (`` ` ``) are surrounding `HW\_%` whereas single quotes (`'`) are surrounding `testuser` and `localhost`.

And then we quit the DBMS, using

```
EXIT;
```

The message displayed after the two first commands should be

```
Query OK, 0 rows affected (0.00 sec)
```

and the message displayed after the last command should be

```
Bye
```

3.12.3 Logging-In as testuser

We now log in as the normal user called "testuser".

Linux users should type *as a normal user, i.e., not as root*, in their terminal the following, and Windows users should type in their command prompt the following[85]:

```
mysql -u testuser -p
```

Enter `password` as your password. If you are prompted with a message

```
ERROR 1045 (28000): Access denied for user
 ↪ 'testuser'@'localhost' (using password: YES)
```

[83] By default, MySQL and MariaDB only create a root user with all privileges and no password, but we added a password at the previous step.

[84] By default, MySQL and MariaDB only create a root user with all privileges and no password, but we added a password at the previous step.

[85] Provided the working directory is still `C:\Program Files\MySQL\MySQL Server 8.0\bin` or similar. Cf. https://dev.mysql.com/doc/mysql-windows-excerpt/8.0/en/mysql-installation-windows-path.html to add the MySQL bin directory to your Windows system `PATH` environment variable. For MacOS user, something like `sudo sh -c 'echo /usr/local/mysql/bin > /etc/paths.d/mysql'` should do.

then you probably typed the wrong password. Otherwise, you should see a welcoming message from MySQL or MariaDB and a prompt.

To save yourself the hassle of typing the password, you can use

```
mysql -u testuser -ppassword
```

or

```
mysql -u testuser -p --password=password
```

to log-in as testuser immediately.

If at some point you want to know if you are logged as root or testuser, simply enter

```
\s;
```

3.12.4 Creating Our First Database

Now, let us create our first schema, our first table, populate it with data, and display various information.

We first create the schema (or database) `HW_FirstTest`:

```
CREATE DATABASE HW_FirstTest; -- Or CREATE SCHEMA HW_FirstTest;
```

Let us make sure that we created it:

```
SHOW DATABASES;
```

Let us use it:

```
USE HW_FirstTest;
```

And see what it contains now:

```
SHOW TABLES;
```

We now create a table called `TableTest`, wtih two integer attributes called `Attribute1` and `Attribute2`:

```
CREATE TABLE TableTest (Attribute1 INT, Attribute2 INT);
```

And can make sure that the table was indeed created:

```
SHOW TABLES;
```

We can further ask our DBMS to display the structure of the table we just created:

```
DESCRIBE TableTest; -- Can be abbreviated as DESC TableTest;
```

And even ask to get back the code that would create the exact same structure (but without the data!):

```
SHOW CREATE TABLE TableTest;
```

Now, let us populate it with some data:

```
INSERT INTO TableTest
    VALUES (1,2),
           (3,4),
           (5,6);
```

Note that the SQL syntax and your DBMS are completely fine with your statement spreading over multiple lines. Let us now display the data stored in the table:

```
SELECT * FROM TableTest;
```

After that last command, you should see

```
+------------+------------+
| Attribute1 | Attribute2 |
+------------+------------+
|          1 |          2 |
|          3 |          4 |
|          5 |          6 |
+------------+------------+
```

Finally, we can erase the content of the table, then erase ("drop") the table, and finally the schema:

```
DELETE FROM TableTest; -- Delete the rows
DROP TABLE TableTest; -- Delete the table
DROP DATABASE HW_FirstTest; -- Delete the schema
```

You're all set! All you have to do is to quit, using the command

```
EXIT;
```

3.12.5 Security Concerns

Note that we were quite careless when we set-up our installation:

- We installed a software without checking its signature. MySQL has a short tutorial[86] on how to check the signature of their packages.
- We did not impose any requirement on the root password of our installation. Using a good, secure, and unique password, should have been required / advised.
- We left all the options on default, whereas a good, secure, installation, always fine-tune what is enabled and what is not.
- We chosed a very weak password for `testuser` that is common to all of our installation.
- Using the command `mysqldump -u testuser -ppassword` means that the password will be stored in the history of your command-line interface (that you should be able to access using `history` or `Get-History` for Powershell) and could be accessed by anyone having access to it.

All of those are obvious security risks, and **make this installation unsafe to be a production environment**. We will only use it as a testing / learning environment, but it is strongly recommended to:

[86] https://dev.mysql.com/doc/refman/8.0/en/verifying-package-integrity.html

- Install it on a virtual machine, so that your personal files would not be impacted by any mis-use of your DBMS,
- Perform a fresh, secured installation if you want to use a DBMS for anything but testing / learning purposes.

Exercises

Exercise 3.1 For each of the following, fill in the blanks:

- In `SQL`, a relation is called a__________________________.
- In `SQL`, every statement ends with________, and in-line comments start with a___________.
- In `SQL`, there is no `string` datatype, so we have to use_____________.
- The Data Control Language of `SQL`'s role is to__________________________.

Exercise 3.2 What does it mean to say that `SQL` is at the same time a "data definition language" and a "data manipulation language"?

Exercise 3.3 Name three kind of objects (for lack of a better word) a `CREATE` statement can create.

Exercise 3.4 Write a `SQL` statement that adds a primary key constraint to an attribute named `ID` in an already existing table named `STAFF`.

Exercise 3.5 Complete each row of the following table with either a datatype or two different examples:

Data type	Examples
	`4`, `-32`
Char(4)	
VarChar(10)	`'Train'`, `'Michelle'`
Bit(4)	
	`TRUE`, `UNKNOWN`

Exercise 3.6 In the datatype `CHAR(3)`, what does the `3` indicate?

Exercise 3.7 Explain this query: `CREATE SCHEMA FACULTY;`.

Exercise 3.8 Write code to

- declare a first table with two attributes, one of which is the primary key,
- declare a second table with two attributes, one of which is the primary key, and the other references the primary key of the first table,
- insert one tuple in the first table,
- insert one tuple in the second table, referencing the only tuple of the first table,

You are free to come up with an example (even very simple or cryptic) or to re-use an example from class.

Exercise 3.9 Explain this query:

```
ALTER TABLE TABLEA
    DROP INDEX Attribute1;
```

Exercise 3.10 If I want to enter January 21, 2016, as a value for an attribute with the `DATE` datatype, what value should I enter?

Exercise 3.11 Write a statement that inserts the values `"Thomas"` and `4` into the table `TRAINS`.

Exercise 3.12 If `PkgName` is the primary key in the table `MYTABLE`, what can you tell about the number of rows returned by the following statement?

```
SELECT * FROM MYTABLE WHERE PkgName = 'MySQL';.
```

Exercise 3.13 If you want that every time a referenced row is delted, all the refering rows are deleted as well, what mechanism should you use?

Exercise 3.14 By default, does the foreign key restrict, cascade, or set null on update? Can you justify this choice?

Exercise 3.15 If a database designer is using the `ON UPDATE SET NULL` for a foreign key, what mechanism is (s)he implementing (i.e., describe how the database will react a certain operation)?

Exercise 3.16 If the following is part of the design of a table:

```
FOREIGN KEY (DptNumber) REFERENCES DEPARTMENT(Number)
    ON DELETE SET DEFAULT
    ON UPDATE CASCADE;
```

What happen to the rows whose foreign key `DptNumber` are set to `3` if the row in the `DEPARTEMENT` table with primary key `Number` set to `3` is...

1. ... deleted?
2. ...updated to `5`?

Exercise 3.17 If the following is part of the design of a `WORKER` table:

```
FOREIGN KEY WORKER(DptNumber) REFERENCES
↪   DEPARTMENT(DptNumber)
    ON UPDATE CASCADE;
```

What happen to the rows whose foreign key `DptNumber` are set to `3` if the row in the `DEPARTMENT` table with primary key `Number` set to `3` is...

1. ... deleted?
2. ... updated to `5`?

Exercise 3.18 Given a relation `TOURIST(Name, EntryDate, Address)`, write a `SQL` statement printing the name and address of all the tourists who entered the territory after the 15 September, 2012.

Exercise 3.19 Describe what the star do in the statement

```
SELECT ALL * FROM MYTABLE;
```

Exercise 3.20 What is the fully qualified name of an attribute? Give an example.

Exercise 3.21 If DEPARTMENT is a database, what is DEPARTMENT.*?

Exercise 3.22 What is a multi-set? What does it mean to say that MySQL treats tables as multi-sets?

Exercise 3.23 What is the difference between

```
SELECT ALL * FROM MYTABLE;
```

and

```
SELECT DISTINCT * FROM MYTABLE;
```

How are the results the same? How are they different?

Exercise 3.24 What is wrong with the statement

```
SELECT * WHERE Name = 'CS' FROM DEPARTMENT;
```

Exercise 3.25 Write a query that returns the number of row (i.e., of entries, of tuples) in a table named BOOK.

Exercise 3.26 When is it useful to use a select-project-join query?

Exercise 3.27 When is a tuple variable useful?

Exercise 3.28 Write a query that changes the name of the professor whose Login is 'caubert' to 'Hugo Pernot' in the table PROF.

Exercise 3.29 Can an **UPDATE** statement have a **WHERE** condition using an attribute that is not the primary key? If no, justify, if yes, tell what could happen.

Exercise 3.30 Give the three possible meaning of the **NULL** value, and an example for each of them.

Exercise 3.31 What are the values of the following expressions (i.e., do they evaluate to TRUE, FALSE, or UNKNOWN)?

- TRUE **AND** FALSE
- TRUE **AND** UNKNOWN
- **NOT** UNKNOWN
- FALSE **OR** UNKNOWN

Exercise 3.32 Write the truth table for **AND** for the three-valued logic of SQL.

Exercise 3.33 What comparison expression should you use to test if a value is different from **NULL**?

Exercise 3.34 Explain this query:

```
SELECT Login
    FROM PROF
    WHERE Department IN ( SELECT Major
                          FROM STUDENT
                          WHERE Login = 'jrakesh');
```

Can you rewrite it without nesting queries?

Exercise 3.35 What is wrong with this query?

```
SELECT Name FROM STUDENT
    WHERE Login IN
    ( SELECT Code FROM Department WHERE head = 'aturing');
```

Exercise 3.36 Write a query that returns the sum of all the values stored in the `Pages` attribute of a `BOOK` table.

Exercise 3.37 Write a query that adds a `Pages` attribute of type `INT` into a (already existing) `BOOK` table.

Exercise 3.38 Write a query that removes the default value for a `Pages` attribute in a `BOOK` table.

Exercise 3.39 Under which conditions does `SQL` allow you to enter the same row in a table twice?

Exercise 3.40 Explain this query: `ROLLBACK;`.

Exercise 3.41 Explain this query: `DELIMITER ;`.

Solution to Exercises

Solution 3.1 The blanks can be filled as follow:

- In `SQL`, a relation is called a__table_____.
- In `SQL`, every statement ends with__a semi-colon (`;`)__, and in-line comments start with a__two minus signs (`--`)__.
- In `SQL`, there is no `string` datatype, so we have to use__VARCHAR(x) or CHAR(x) where x is an integer reflecting the maximum (or fixed) size of the string___.
- The Data Control Language of `SQL`'s role is to__control access to the data stored, by creating users and granting them rights__.

Solution 3.2 It can specify the conceptual and internal schema, **and** it can manipulate the data.

Solution 3.3 Database (schema), table, view, assertion, trigger, etc.

Solution 3.4 `ALTER TABLE STAFF ADD PRIMARY KEY(ID);`

Data type	Examples
`INT`	`4`, `-32`
`CHAR(4)`	`'abCD'`, `"dEfG"`
`VARCHAR(10)`	`'Train'`, `'Michelle'`
`BIT(4)`	`B'1010'`, `B'0101'`

Solution 3.5 `BOOL`` |`TRUE, FALSE, NULL‘

`NULL` is actually a valid answer for every single type of

Solution 3.6 That we can store exactly three characters.

Solution 3.7 It creates a schema, i.e., a database, named `Faculty`.

Solution 3.8 A simple and compact code could be:

```
/* code/sql/HW_Short.sql */
CREATE TABLE A (
  Att1 INT PRIMARY KEY,
  Att2 INT
);

CREATE TABLE B (
  Att3 INT PRIMARY KEY,
  Att4 INT,
  FOREIGN KEY (Att4) REFERENCES A (Att1)
);

INSERT INTO A
VALUES (
  1,
  2);

INSERT INTO B
VALUES (
  3,
  1);

```

HW_Short.sql[87]

Solution 3.9 It removes the `UNIQUE` constraint on the `Attribute1` in the `TABLEA` table.

Solution 3.10 `DATE'2016-01-21'`, `'2016-01-21'`, `'2016/01/21'`, `'20160121'`.

Solution 3.11 `INSERT INTO TRAINS VALUES('Thomas', 4);`

Solution 3.12 We know that at most one (but possibly 0) row will be returned.

Solution 3.13 We should use a referential triggered action clause, `ON DELETE CASCADE`.

Solution 3.14 By default, the foreign key restricts updates. This prevents unwanted update of information: if an update needs to be propagated, then it needs to be "acknowledged" and done explicitely.

Solution 3.15 If the referenced row is updated, then the attribute of the referencing rows are set to `NULL`.

Solution 3.16 In the referencing rows,

1. the department number is set to the default value.
2. the department number is updated accordingly.

Solution 3.17

1. This operation is rejected: the row in the `DEPARTMENT` table with primary key `Number` set to `3` cannot be deleted if a row in the `WORKER` table references it.
2. In the referencing rows, the department number is updated accordingly.

Solution 3.18 We could use the following:

[87] https://rocketgit.com/user/caubert/CSCI_3410/source/tree/branch/master/blob/notes/code/sql/HW_Short.sql

```
SELECT Name, Address
    FROM TOURIST
    WHERE EntryDate > DATE'2012-09-15';
```

Solution 3.19 It selects all the attributes, it is a wildcard.

Solution 3.20 The name of the relation with the name of its schema and a period beforehand. An example would be `EMPLOYEE.Name`.

Solution 3.21 All the tables in that database.

Solution 3.22 A multiset is a set where the same value can occur twice. In MySQL, the same row can occur twice in a table.

Solution 3.23 They both select all the rows in the `MYTABLE` table, but `ALL` will print the duplicate values, whereas `DISTINCT` will print them only once.

Solution 3.24 You cannot have the `WHERE` before `FROM`.

Solution 3.25 `SELECT COUNT(*) FROM BOOK;`

Solution 3.26 We use those query that projects on attributes using a selection and join conditions when we need to construct for information based on pieces of data spread in multiple tables.

Solution 3.27 It makes the distinction between two different rows of the same table, it is useful when we want to select a tuple in a relation that is in a particular relation with a tuple in the same relation. Quoting https://stackoverflow.com/a/7698796/:

> They are useful for saving typing, but there are other reasons to use them:
>
> - If you join a table to itself you must give it two different names otherwise referencing the table would be ambiguous.
> - It can be useful to give names to derived tables, and in some database systems it is required... even if you never refer to the name.

Solution 3.28 We could use the following:

```
UPDATE PROF SET Name = 'Hugo Pernot'
    WHERE Login = 'caubert';
```

Solution 3.29 Yes, we can have select condition that does not use primary key. In that case, it could be the case that we update more than one tuple with such a command (which is not necessarily a bad thing).

Solution 3.30 Unknown value ("Will it rain tomorrow?"), unavailable / withheld ("What is the phone number of Harry Belafonte?"), N/A ("What is the email address of Abraham Lincoln?").

Solution 3.31

- `TRUE AND FALSE` → `FALSE`
- `TRUE AND UNKNOWN` → `UNKNOWN`
- `NOT UNKNOWN` → `UNKNOWN`
- `FALSE OR UNKNOWN` → `FALSE`

Solution 3.32

- `TRUE AND TRUE` → `TRUE`

- TRUE **AND** FALSE ⟶ FALSE
- TRUE **AND** UNKNOWN ⟶ UNKNOWN
- FALSE **AND** FALSE ⟶ FALSE
- UNKNOWN **AND** UNKNOWN ⟶ UNKNOWN
- FALSE **AND** UNKNOWN ⟶ FALSE
- The other cases can be deduced by symmetry.

For a more compact presentation, refer to the three-valued truth table.

Solution 3.33 `IS NOT`

Solution 3.34 It list the login of the professors teaching in the department where the student whose login is "jrakesh" is majoring. It can be rewritten as

```
SELECT PROF.Login
    FROM PROF, STUDENT
    WHERE Department = Major
    AND STUDENT.Login = 'jrakesh';
```

Solution 3.35 It tries to find a `Login` in a `Code`.

Solution 3.36 `SELECT SUM(Pages) FROM BOOK;`

Solution 3.37 `ALTER TABLE BOOK ADD COLUMN Pages INT;`

Solution 3.38 `ALTER TABLE BOOK ALTER COLUMN Pages DROP DEFAULT;`

Solution 3.39 Essentially, if there are no primary key in the relation, and if no attribute has the `UNIQUE` constraint. Cf. also this previous problem.

Solution 3.40 This command, `ROLLBACK`, undoes the last transaction, i.e. it allows to "roll back" to a previous point identified by `START TRANSACTION;`: everything that happened between those two commands is "undone", as if it had never been executed.

Solution 3.41 This command, `DELIMITER ;`, changes the SQL interpreter delimiter back to being "`;`". It is useful when defining procedures, as the traditional delimiter's role have to be "suspended" to wrap a series of commands inside a function.

Problems

Problem 3.1 ***(Discovering the documentation)*** The goal of this problem is to learn where to find the documentation for your DBMS, and to understand how to read the syntax of `SQL` commands.

You can consult (Elmasri and Navathe 2010, Table 5.2, p. 140) or (Elmasri and Navathe 2015, Table 7.2, p. 235), for a very quick summary of the most common commands. Make sure you are familiar with the Backus–Naur form (BNF) notation commonly used:

- non-terminal symbols (i.e., variables, parameters) are enclosed in angled brackets,

<...>

- optional parts are shown in square brackets,

[...]

- repetitons are shown in braces

`{…}`

- alternatives are shown in parenthesis and separated by vertical bars,

`(…|…|…)`

The most complete lists of commands are probably at

- https://mariadb.com/kb/en/sql-statements/ and
- https://dev.mysql.com/doc/refman/8.0/en/sql-statements.html

Those are the commands implemented in the DBMS you are actually using. Since there are small variations from one implementation to the other, it is better to take one of this link as a reference in the future.

As a starting point, looking at the syntax for `CREATE TABLE` commands is probably a good start, cf. https://mariadb.com/kb/en/create-table/ or https://dev.mysql.com/doc/refman/8.0/en/create-table.html.

Problem 3.2 ***(Create and use a simple table in*** **`SQL`*)*** This problem will guide you in manipulating a very simple table in `SQL`.

Pb 3.2 – Question 1 Log in as `testuser`, create a database named `HW_Address`, use it, and create two tables:

```
CREATE TABLE NAME(
    FName VARCHAR(15),
    LName VARCHAR(15),
    ID INT,
    PRIMARY KEY(ID)
);

CREATE TABLE ADDRESS(
    StreetName VARCHAR(15),
    Number INT,
    Habitants INT,
    PRIMARY KEY(StreetName, Number)
);
```

Pb 3.2 – Question 2 Observe the output produced by the command `DESC ADDRESS;`.

Pb 3.2 – Question 3 Add a foreign key to the `ADDRESS` table, using

```
ALTER TABLE ADDRESS
    ADD FOREIGN KEY (Habitants)
    REFERENCES NAME(ID);
```

And observe the new output produced by the command `DESC ADDRESS;`.

Is it what you would have expected? How informative is it? Can you think of a command that would output more detailled information, including a reference to the existence of the foreign key?

Pb 3.2 – Question 4 Draw the relational model corresponding to that database and identify the primary and foreign keys.

Pb 3.2 – Question 5 Add this data to the NAME table:

```
INSERT INTO NAME VALUES ('Barbara', 'Liskov', 003);
INSERT INTO NAME VALUES ('Tuong Lu', 'Kim', 004);
INSERT INTO NAME VALUES ('Samantha', NULL, 080);
```

What command can you use to display this infomation back? Do you notice anything regarding the values we entered for the ID attribute?

Pb 3.2 – Question 6 Add some data into the ADDRESS table:

```
INSERT INTO ADDRESS
    VALUES
    ('Armstrong Drive', 10019, 003),
    ('North Broad St.', 23, 004),
    ('Robert Lane', 120, NULL);
```

What difference do you notice with the insertions we made in the NAME table? Which syntax seems more easy to you?

Pb 3.2 – Question 7 Write a **SELECT** statement that returns the ID number of the person whose first name is "Samantha".

Pb 3.2 – Question 8 Write a statement that violates the entity integrity constraint. What is the error message returned?

Pb 3.2 – Question 9 Execute an **UPDATE** statement that violates the referential integrity constraint. What is the error message returned?

Pb 3.2 – Question 10 Write a statement that violates another kind of constraint. Explain what constraint you are violating and explain the error message.

Problem 3.3 *(Duplicate rows in SQL)* Log in as testuser and create a database titled HW_REPETITION. Create in that database a table (the following questions refer to this table as EXAMPLE, but you are free to name it whatever you want) with at least two attributes that have different data types. Do not declare a primary key yet. Answer the following:

Pb 3.3 – Question 1 Add a tuple to your table using

```
INSERT INTO EXAMPLE VALUES(X, Y);
```

where the X and Y are values have the right datatype. Try to add this tuple again. What do you observe? (You can use **SELECT * FROM** EXAMPLE; to observe what is stored in this table.)

Pb 3.3 – Question 2 Alter your table to add a primary key, using

```
ALTER TABLE EXAMPLE ADD PRIMARY KEY (Attribute);
```

where Attribute is the name of the attribute you want to be a primary key. What do you observe?

Pb 3.3 – Question 3 Empty your table using

```
DELETE FROM EXAMPLE;
```

and alter your table to add a primary key, using the command we gave at the previous step. What do you observe?

Pb 3.3 – Question 4 Try to add the same tuple twice. What do you observe?

Problem 3.4 ***(Constraints on foreign keys)*** From the notes, recall the following about foreign keys:

> Two important remarks:
>
> - The datatype of the foreign key has to be the exactly the same as the datatype of the attribute to which we are referring.
> - The target of the foreign key *must be* the primary key.

But, the situation is slightly more complex. Test for yourself by editing the following code as indicated:

```sql
/* code/sql/HW_FKtest.sql */
DROP SCHEMA IF EXISTS HW_FKtest;

CREATE SCHEMA HW_FKtest;

USE HW_FKtest;

CREATE TABLE TARGET (
  Test VARCHAR(15) PRIMARY KEY
);

CREATE TABLE SOURCE (
  Test VARCHAR(25),
  FOREIGN KEY (Test) REFERENCES TARGET (Test)
);
```

HW_FKtest.sql[88]

1. Remove the `PRIMARY KEY` constraint.
2. Replace `PRIMARY KEY` with `UNIQUE`.
3. Replace one of the `VARCHAR(25)` with `CHAR(25)`.
4. Replace one of the `VARCHAR(25)` with `INT`.
5. Replace one of the `VARCHAR(25)` with `VARCHAR(15)`
6. Once you have edited and run the program in all of its modified versions, adjust the remarks above to better reflect the reality of the implementation we are using.

---

[88] https://rocketgit.com/user/caubert/CSCI_3410/source/tree/branch/master/blob/notes/code/sql/HW_FKtest.sql

Problem 3.5 ***(Revisiting the PROF table)*** Create the PROF, DEPARTMENT, STUDENT and GRADE tables as in the "Constructing and populating a new example" section. Populate them with some data (copy it from the notes or come up with your own data).

To obtain exactly the same schema as the one we developped and edited, you can use mysqldump to "dump" this table, with a command like

```
mysqldump -u testuser -ppassword\
    -h localhost --add-drop-database\
    --skip-comments --compact\
    HW_ProfExample > dump.sql
```

The code we studied during the lecture is more or less the following.

```sql
/* code/sql/HW_ProfExampleRevisitedRevisited.sql */
DROP SCHEMA IF EXISTS HW_ProfExampleRevisited;

CREATE SCHEMA HW_ProfExampleRevisited;

USE HW_ProfExampleRevisited;

CREATE TABLE PROF (
  Login VARCHAR(25) PRIMARY KEY,
  NAME VARCHAR(25),
  Department CHAR(5)
);

CREATE TABLE DEPARTMENT (
  Code CHAR(5) PRIMARY KEY,
  NAME VARCHAR(25),
  Head VARCHAR(25),
  FOREIGN KEY (Head) REFERENCES PROF (LOGIN) ON UPDATE CASCADE
);

ALTER TABLE PROF
  ADD FOREIGN KEY (Department) REFERENCES DEPARTMENT (Code);

CREATE TABLE STUDENT (
  Login VARCHAR(25) PRIMARY KEY,
  NAME VARCHAR(25),
  Registered DATE,
  Major CHAR(5),
  FOREIGN KEY (Major) REFERENCES DEPARTMENT (Code)
);

CREATE TABLE GRADE (
  Login VARCHAR(25),
  Grade INT,
  PRIMARY KEY (LOGIN, Grade),
  FOREIGN KEY (LOGIN) REFERENCES STUDENT (LOGIN)
);
```

```

INSERT INTO DEPARTMENT
VALUES (
  'MATH',
  'Mathematics',
  NULL),
(
  'CS',
  'Computer
    Science',
  NULL);

INSERT INTO DEPARTMENT (
  Code,
  Name)
VALUES (
  'CYBR',
  'Cyber Secturity');

INSERT INTO PROF (
  LOGIN,
  Department,
  Name)
VALUES (
  'caubert',
  'CS',
  'Clément Aubert');

INSERT INTO PROF (
  LOGIN,
  Name,
  Department)
VALUES (
  'aturing',
  'Alan Turing',
  'CS'),
(
  'perdos',
  'Paul
    Erdős',
  'MATH'),
(
  'bgates',
  'Bill Gates',
  'CYBR');

INSERT INTO STUDENT (
  LOGIN,
  Name,
  Registered,
```

```
  Major)
VALUES (
  'jrakesh',
  'Jalal Rakesh',
  DATE '2017-12-01',
  'CS'),
(
  'svlatka',
  'Sacnite Vlatka',
  '2015-03-12',
  'MATH'),
(
  'cjoella',
  'Candice Joella',
  '20120212',
  'CYBR'),
(
  'aalyx',
  'Ava Alyx',
  20121011,
  'CYBR'),
(
  'caubert',
  'Clément Aubert',
  NULL,
  'CYBR');

INSERT INTO GRADE
VALUES (
  'jrakesh',
  3.8),
(
  'svlatka',
  2.5);

```

HW_ProfExampleRevisitedRevisited.sql[89]

We will resume working on this model, and enhance it.

Pb 3.5 – Question 1 Draw the *complete* relational model for this database (i.e., for the PROF, DEPARTMENT, STUDENT and GRADE relations).

Pb 3.5 – Question 2 Create and populate a LECTURE table as follows:

- It should have four attributes: Name, Instructor, Code, and Year, of types VARCHAR(25) for the first two, CHAR(5) for Code, and YEAR(4) for Year.
- The Year and Code attributes should be the primary key (yes, have *two* attributes be the primary key).

[89] https://rocketgit.com/user/caubert/CSCI_3410/source/tree/branch/master/blob/notes/code/sql/HW_ProfExampleRevisitedRevisited.sql

- The `Instructor` attribute should be a foreign key referencing the `Login` attribute in `PROF`.
- Populate the `LECTURE` table with some made-up data.

Try to think about some of the weakenesses of this representation. For instance, can it accomodate two instructors for the same class? Write down two possibles scenarios in which this schema would not be appropriate.

Pb 3.5 – Question 3 The `GRADE` table had some limitations too. For example, every student could have only one grade. Add two columns to the `GRADE` table using:

```
ALTER TABLE GRADE
    ADD COLUMN LectureCode CHAR(5),
    ADD COLUMN LectureYear YEAR(4);
```

Add a foreign key:

```
ALTER TABLE GRADE
    ADD FOREIGN KEY (LectureYear, LectureCode)
    REFERENCES LECTURE(Year, Code);
```

Use **`DESCRIBE`** and **`SELECT`** to observe the schema of the `GRADE` table and its rows. Is it what you would have expected?

Pb 3.5 – Question 4 Update the tuples in `GRADE` with some made-up data. Now every row should contain, in addition to a login and a grade, a lecture year and a lecture code.

Pb 3.5 – Question 5 Update the relational model you previously drew to reflect the new situation of your tables.

Pb 3.5 – Question 6 Write **`SELECT`** statements answering the following questions (where `PROF.Name`, `LECTURE.Name`, `YYYY`, `LECTURE.Code` and `STUDENT.Login` should be relevant values considering your data):

1. "Could you give me the logins and grades of the students who took `LECTURE.Name` in `YYYY`?"
2. "Could you list the instructors who taught in year `YYYY` without any duplicates?"
3. "Could you list the name and grade of all the student who ever took the class `LECTURE.Code`?"
4. "Could you tell me which years was the class `LECTURE.Code` taught?"
5. "Could you list the other classes taught the same year as the class `LECTURE.Code`?"
6. "Could you print the names of the students who registered after `STUDENT.Login`?"
7. "Could you tell me how many departments' heads are teaching this year?"

Problem 3.6 ***(TRAIN table and more advanced SQL coding)*** Look at the `SQL` code below and then answer the following questions.

```
/* code/sql/HW_Train.sql */
CREATE TABLE TRAIN (
  ID VARCHAR(30),
  Model VARCHAR(30),
  ConstructionYear YEAR (4)
```

```
13 );
14
15 CREATE TABLE CONDUCTOR (
16   ID VARCHAR(20),
17   NAME VARCHAR(20),
18   ExperienceLevel VARCHAR(20)
19 );
20
21 CREATE TABLE ASSIGNED_TO (
22   TrainId VARCHAR(20),
23   ConductorId VARCHAR(20),
24   Day DATE,
25   PRIMARY KEY (TrainId, ConductorId)
26 );
27
```

HW_Train.sql[90]

Pb 3.6 – Question 1 Modify the **CREATE** statement that creates the `TRAIN` table (lines 1–5), so that `ID` would be declared as the primary key. It is sufficient to only write the line(s) that need to change.

Pb 3.6 – Question 2 Write an **ALTER** statement that makes `ID` become the primary key of the `CONDUCTOR` table.

Pb 3.6 – Question 3 Modify the **CREATE** statement that creates the `ASSIGNED_TO` table (lines 13–18), so that it has two foreign keys: `ConductorId` references the `ID` attribute in `CONDUCTOR` and `TrainId` references the `ID` attribute in `TRAIN`. It is sufficient to only write the line(s) that need to change.

Pb 3.6 – Question 4 Write **INSERT** statements that insert one tuple of your choosing in each relation (no **NULL** values). These statements should respect all the constraints (including the ones we added in the previous questions) and result in actual insertions. (Remember that four digits is a valid value for an attribute with the `YEAR(4)` datatype.)

Pb 3.6 – Question 5 Write a statement that sets the value of the `ExperienceLevel` attribute to “Senior” in all the tuples where the `ID` attribute is “GP1029” in the `CONDUCTOR` relation.

Pb 3.6 – Question 6 Write a **SELECT** statement that answers each of the following questions:

1. “What are the identification numbers of the trains?”
2. “What are the names of the conductors with a”Senior” experience level?”
3. “What are the construction years of the”Surfliner” and “Regina” models that we have?”
4. “What is the ID of the conductor that was responsible of the train referenced”K-13” on 2015/12/14?”
5. “What are the models that were ever conducted by the conductor whose ID is”GP1029”?”

---

[90] https://rocketgit.com/user/caubert/CSCI_3410/source/tree/branch/master/blob/notes/code/sql/HW_Train.sql

Problem 3.7 ***(Read, correct, and write `SQL` statements for the COFFEE database)***

Suppose we have the relational model depicted below, with the indicated data in it:

COFFEE

Ref	Origin	TypeOfRoast	PricePerPound
001	Brazil	Light	8.90
121	Bolivia	Dark	7.50
311	Brazil	Medium	9.00
221	Sumatra	Dark	10.25

CUSTOMER

CardNo	Name	Email
001	Bob Hill	b.hill@isp.net
002	Ana Swamp	swampa@nca.edu
003	Mary Sea	brig@gsu.gov
004	Pat Mount	pmount@fai.fr

SUPPLY

Provider	Coffee
Coffee Unl.	001
Coffee Unl.	121
Coffee Exp.	311
Johns & Co.	221

PROVIDER

Name	Email
Coffee Unl.	bob@cofunl.com
Coffee Exp.	pat@coffeex.dk
Johns & Co.	NULL

In the following, we will assume that this model was implemented in a DBMS (MySQL or MariaDB), the primary keys being `COFEE.Ref`, `CURTOMER.CardNo`, `SUPPLY.Provider` and `SUPPLY.Coffee`, and `PROVIDER.Name`, and the foreign keys being as follows:

FavCoffee in the **CUSTOMER** relation	refers to	**Ref** in the **COFFEE** relation
Provider in the **SUPPLY**	refers to	**Name** in the **PROVIDER** relation
Coffee in the **SUPPLY**	refers to	**Ref** in the **COFFEE** relation

Read and write `SQL` commands for the following "what-if" scenarios. Assume that:

1. Datatype do not matter: we use only strings and appropriate numerical datatypes.

2. Every statement respects SQL's syntax (there's no "a semi-colon is missing" trap).
3. None of these commands are actually executed; the data is always in the state depicted above.

You can use **COFFEE**.1 to denote the first tuple (or row) in **COFFEE**, and similarly for other relations and tuples (so that, for instance, **SUPPLY**.4 corresponds to `"Johns & Co"., 221`).

Pb 3.7 – Question 1 Draw the relational model of this table.

Pb 3.7 – Question 2 Determine if the following insertion statements would violate the the entity integrity constraint, ("primary key cannot be `NULL` and should be unique"), the referential integrity constraint ("the foreign key must refer to something that exists"), if there would be some other kind of error (ignoring the plausability / revelance of inserting that tuple), or if it would result in successful insertion.

```
INSERT INTO CUSTOMER VALUES(005, 'Bob Hill', NULL, 001);
INSERT INTO COFFEE VALUES(002, "Peru", "Decaf", 3.00);
INSERT INTO PROVIDER VALUES(NULL, "contact@localcof.com");
INSERT INTO SUPPLY VALUES("Johns  Co.", 121);
INSERT INTO SUPPLY VALUES("Coffee Unl.", 311, 221);
```

Pb 3.7 – Question 3

Assuming that the referential triggered action clause `ON UPDATE CASCADE` is used for each of the foreign keys in this database, list the tuples modified by the following statements:

```
UPDATE CUSTOMER SET FavCoffee = 001
    WHERE CardNo = 001;

UPDATE COFFEE SET TypeOfRoast = 'Decaf'
    WHERE Origin = 'Brazil';

UPDATE PROVIDER SET Name = 'Coffee Unlimited'
    WHERE Name = 'Coffee Unl.';

UPDATE COFFEE SET PricePerPound = 10.00
    WHERE PricePerPound > 10.00;
```

Pb 3.7 – Question 4 Assuming that the referential triggered action clause `ON DELETE CASCADE` is used for each of the foreign keys in this database, list the tuples modified by the following statements:

```
DELETE FROM CUSTOMER
    WHERE Name LIKE '%S%';

DELETE FROM COFFEE
    WHERE Ref = 001;

DELETE FROM SUPPLY
    WHERE Provider = 'Coffee Unl.'
        AND Coffee = 001;
```

```sql
DELETE FROM PROVIDER
    WHERE Name = 'Johns & Co.';
```

Pb 3.7 – Question 5 Assume that there is more data in our table than what was given at the beginning of the problem. Write SQL queries that answer the following questions:

1. "What are the origins of your dark coffees?"
2. "What is the reference of Bob's favorite coffee?" (note: it does not matter if you return the favorite coffee of all the Bobs in the database.)
3. "What are the names of the providers who did not give their email?"
4. "How many coffees does Johns & co. provide us with?"
5. "What are the names of the providers of my dark coffees?"

Problem 3.8 ***(Write select queries for the DEPARTMENT table)*** Consider the following SQL code:

```sql
/* code/sql/HW_Department.sql */
CREATE TABLE DEPARTMENT (
  ID INT PRIMARY KEY,
  NAME VARCHAR(30)
);

CREATE TABLE EMPLOYEE (
  ID INT PRIMARY KEY,
  NAME VARCHAR(30),
  Hired DATE,
  Department INT,
  FOREIGN KEY (Department) REFERENCES DEPARTMENT (ID)
);

INSERT INTO DEPARTMENT
VALUES (
  1,
  "Storage"),
(
  2,
  "Hardware");

INSERT INTO EMPLOYEE
VALUES (
  1,
  "Bob",
  20100101,
  1),
(
  2,
  "Samantha",
  20150101,
```

```
  1),
(
  3,
  "Mark",
  20050101,
  2),
(
  4,
  "Karen",
  NULL,
  1),
(
  5,
  "Jocelyn",
  20100101,
  1);

```

HW_Department.sql[91]

Write queries that return the following information. The values returned *in this set-up* will be in parenthesis, but keep the queries general.

1. The name of the employees working in the `Storage` department (`"Bob"`, `"Samantha"`, `"Karen"` and `"Jocelyn"`),
2. The name of the employee that has been hired for the longest period of time (`"Mark"`),
3. The name(s) of the employee(s) from the `Storage` department who has(have) been hired for the longest period of time. Phrased differently, the oldest employees of the `Storage` department (`"Bob"` and `"Jocelyn"`).

Problem 3.9 ***(Write select queries for the COMPUTER table)*** Consider the following SQL code:

```
/* code/sql/HW_Computer.sql */
DROP SCHEMA IF EXISTS HW_Computer;

CREATE SCHEMA HW_Computer;

USE HW_Computer;

CREATE TABLE COMPUTER (
  ID VARCHAR(20) PRIMARY KEY,
  Model VARCHAR(40)
);

CREATE TABLE PRINTER (
  ID VARCHAR(20) PRIMARY KEY,
```

[91] https://rocketgit.com/user/caubert/CSCI_3410/source/tree/branch/master/blob/notes/code/sql/HW_Department.sql

```
  Model VARCHAR(40)
);

CREATE TABLE CONNEXION (
  Computer VARCHAR(20),
  Printer VARCHAR(20),
  PRIMARY KEY (Computer, Printer),
  FOREIGN KEY (Computer) REFERENCES COMPUTER (ID),
  FOREIGN KEY (Printer) REFERENCES PRINTER (ID)
);

INSERT INTO COMPUTER
VALUES (
  'A',
  'DELL A'),
(
  'B',
  'HP X'),
(
  'C',
  'ZEPTO D'),
(
  'D',
  'MAC Y');

INSERT INTO PRINTER
VALUES (
  '12',
  'HP-140'),
(
  '13',
  'HP-139'),
(
  '14',
  'HP-140'),
(
  '15',
  'HP-139');

INSERT INTO CONNEXION
VALUES (
  'A',
  '12'),
(
  'A',
  '13'),
(
  'B',
  '13'),
(
```

```
  'C',
  '14');
```

HW_Computer.sql[92]

Write queries that return the following information. The values returned *in this set-up* will be in parenthesis, but keep the queries general.

1. The number of computers connected to the printer whose ID is '13' (2).
2. The number of different models of printers (2).
3. The model(s) of the printer(s) connected to the computer whose ID is 'A' ('HP-140' and 'HP-139').
4. The ID('s) of the computer(s) not connected to any printer ('D').

Problem 3.10 *(Write select queries for the SocialMedia schema)* Consider the following SQL code:

```
/* code/sql/HW_SocialMedia.sql */
CREATE TABLE ACCOUNT (
  ID INT PRIMARY KEY,
  NAME VARCHAR(25),
  Email VARCHAR(25) UNIQUE
);

CREATE TABLE SUBSCRIBE (
  Subscriber INT,
  Subscribed INT,
  DATE DATE,
  FOREIGN KEY (Subscriber) REFERENCES ACCOUNT (ID),
  FOREIGN KEY (Subscribed) REFERENCES ACCOUNT (ID),
  PRIMARY KEY (Subscriber, Subscribed)
);

CREATE TABLE VIDEO (
  ID INT PRIMARY KEY,
  Title VARCHAR(25),
  Released DATE,
  Publisher INT,
  FOREIGN KEY (Publisher) REFERENCES ACCOUNT (ID)
);

CREATE TABLE THUMBS_UP (
  Account INT,
  Video INT,
  DATE DATE,
  PRIMARY KEY (Account, Video),
  FOREIGN KEY (Account) REFERENCES ACCOUNT (ID),
  FOREIGN KEY (Video) REFERENCES VIDEO (ID)
```

[92] https://rocketgit.com/user/caubert/CSCI_3410/source/tree/branch/master/blob/notes/code/sql/HW_Computer.sql

```
);

INSERT INTO ACCOUNT
VALUES (
  1,
  "Bob Ross",
  "bob@ross.com"),
(
  2,
  NULL,
  "anon@fai.com"),
(
  3,
  "Martha",
  NULL);

INSERT INTO SUBSCRIBE
VALUES (
  2,
  1,
  DATE "2020-01-01"),
(
  3,
  1,
  DATE "2019-03-03"),
(
  3,
  2,
  DATE "2019-03-03"),
(
  2,
  2,
  DATE "2019-03-03"),
(
  1,
  2,
  DATE "2019-03-03");

-- The first entry means that 2 subscribed to 1, not the
--                   other way around.
--                    And similarly for the other entries.
INSERT INTO VIDEO
VALUES (
  10,
  "My first video!",
  DATE "2020-02-02",
  1),
(
  20,
  "My second video!",
```

```
  DATE "2020-02-03",
  1),
(
  30,
  "My vacations",
  DATE "2020-02-04",
  2);

INSERT INTO THUMBS_UP
VALUES (
  2,
  10,
  DATE "2020-02-02"),
(
  3,
  10,
  DATE "2020-02-02"),
(
  2,
  20,
  DATE "2020-02-02"),
(
  1,
  30,
  DATE "2020-02-05");
```

HW_SocialMedia.sql[93]

Write queries that return the following information. The values returned *in this set-up* will be in parenthesis, but keep the queries general.

1. The title of all the videos (`"My first video!"`, `"My second video!"`, `"My vacations"`).
2. The release date of the video whose title is `"My first video!"` (`"2020-02-02"`).
3. The ID of the account(s) where the "Name" attribute was not given (`"2"`).
4. The ID of the videos whose title contains the word `"video"` (`"10"`, `"20"`).
5. The number of thumbs up for the video with title `"My vacations"` (`"1"`).
6. The title of the oldest video (`"My first video!"`).
7. The names of the accounts who gave a thumbs up to the video with ID 30 (`"Bob Ross"`).
8. The ID of the account with the greatest number of subscribers (`"2"`).

Problem 3.11 ***(Write select queries for a variation of the COMPUTER table)***
Consider the following `SQL` code:

```
CREATE TABLE COMPUTER (
  ID VARCHAR(20) PRIMARY KEY,
```

[93] https://rocketgit.com/user/caubert/CSCI_3410/source/tree/branch/master/blob/notes/code/sql/HW_SocialMedia.sql

```
  Model VARCHAR(40)
);

CREATE TABLE PERIPHERAL (
  ID VARCHAR(20) PRIMARY KEY,
  Model VARCHAR(40),
  TYPE ENUM ('mouse', 'keyboard', 'screen', 'printer')
);

CREATE TABLE CONNEXION (
  Computer VARCHAR(20),
  Peripheral VARCHAR(20),
  PRIMARY KEY (Computer, Peripheral),
  FOREIGN KEY (Computer) REFERENCES COMPUTER (ID),
  FOREIGN KEY (Peripheral) REFERENCES PERIPHERAL (ID)
);

INSERT INTO COMPUTER
VALUES (
  'A',
  'Apple IIc Plus'),
(
  'B',
  'Commodore SX-64');

INSERT INTO PERIPHERAL
VALUES (
  '12',
  'Trendcom Model',
  'printer'),
(
  '14',
  'TP-10
    Thermal Matrix',
  'printer'),
(
  '15',
  'IBM Selectric',
  'keyboard');

INSERT INTO CONNEXION
VALUES (
  'A',
  '12'),
(
  'B',
  '14'),
(
  'A',
  '15');
```

61

HW_ComputerVariation.sql[94]

Write queries that return the following information. The values returned *in this set-up* will be in parenthesis, but keep the queries general.

1. The model of the computer whose `ID` is `'A'` (`'Apple IIc Plus'`).
2. The type of the peripheral whose `ID` is `'14'` (`printer`).
3. The model of the printers (Trendcom Model, TP-10 Thermal Matrix).
4. The model of the peripherals whose `NAME` starts with `'IBM'` (`'IBM Selectric'`).
5. The model of the peripherals connected to the computer whose `ID` is `'A'` (`Trendcom Model, IBM Selec`
6. The number of peripheral connected to the computer whose model is Apple IIc Plus (`2`).

Problem 3.12 ***(Improving a role-playing game with a relational model)*** A friend of yours wants you to review and improve the code for a role-playing game.

The original idea was that each character has a name, a class (e.g., Bard, Assassin, Druid), a certain amount of experience, a level, one or more weapons (providing bonuses) and the ability to complete quests. A quest has a name and rewards the characters who completed it with a certain amount of experience and, on rare occaisions, with a special item.

Your friend came up with the following code:

```sql
CREATE TABLE CHARACTER(
    Name VARCHAR(30) PRIMARY KEY,
    Class VARCHAR(30),
    XP INT,
    LVL INT,
    Weapon_Name VARCHAR(30),
    Weapon_Bonus INT,
    Quest_Completed VARCHAR(30)
);

CREATE TABLE QUEST(
    ID VARCHAR(20) PRIMARY KEY,
    Completed_By VARCHAR(30),
    XP_Gained INT,
    Special_Item VARCHAR(20),
    FOREIGN KEY (Completed_By) REFERENCES CHARACTER(Name)
);

ALTER TABLE CHARACTER
    ADD FOREIGN KEY (Quest_Completed) REFERENCES QUEST(ID);
```

However, there are several problems with the code:

- A character can have only one weapon. (All the attempts to "hack" the `CHARACTER` table to add an arbitrary number of weapons ended up creating horrible messes.)

[94] https://rocketgit.com/user/caubert/CSCI_3410/source/tree/branch/master/blob/notes/code/sql/HW_ComputerVariation.sql

- Every time a character completes a quest, a copy of the quest must also be created. (Your friend is not so sure why, but nothing else works. Also it seems that a character can complete only one quest, but your friend is not sure about that either.)
- It would be nice to be able to store features that are tied to the class, not the character, like the bonuses they provide and their associated elements (e.g., all bards use fire, all assassins use wind, etc.), but your friend simply cannot figure out how to make that happen.

Can you provide a *relational model* (there is no need to write the SQL code, but do remember to indicate the primary and foreign keys) that would solve all of your friend's troubles?

Problem 3.13 ***(A simple database for books)*** Consider the following code:

```sql
/* code/sql/HW_SimpleBook.sql */
DROP SCHEMA IF EXISTS HW_SimpleBook;

CREATE SCHEMA HW_SimpleBook;

USE HW_SimpleBook;

CREATE TABLE AUTHOR (
  FName VARCHAR(30),
  LName VARCHAR(30),
  Id INT PRIMARY KEY
);

CREATE TABLE PUBLISHER (
  NAME VARCHAR(30),
  City VARCHAR(30),
  PRIMARY KEY (NAME, City)
);

CREATE TABLE BOOK (
  Title VARCHAR(30),
  Pages INT,
  Published DATE,
  PublisherName VARCHAR(30),
  PublisherCity VARCHAR(30),
  FOREIGN KEY (PublisherName, PublisherCity) REFERENCES
    PUBLISHER (NAME, City),
  Author INT,
  FOREIGN KEY (Author) REFERENCES AUTHOR (Id),
  PRIMARY KEY (Title, Published)
);

INSERT INTO AUTHOR
VALUES (
  "Virginia",
  "Wolve",
  01),
(
```

```
  "Paul",
  "Bryant",
  02),
(
  "Samantha",
  "Carey",
  03);

INSERT INTO PUBLISHER
VALUES (
  "Gallimard",
  "Paris"),
(
  "Gallimard",
  "New-York"),
(
  "Jobs Pub.",
  "New-York");

INSERT INTO BOOK
VALUES (
  "What to eat",
  213,
  DATE '20170219',
  "Gallimard",
  "Paris",
  01),
(
  "Where to live",
  120,
  DATE '20130212',
  "Gallimard",
  "New-York",
  02),
(
  "My Life, I",
  100,
  DATE '18790220',
  "Gallimard",
  "Paris",
  03),
(
  "My Life, II",
  100,
  DATE '18790219',
  "Jobs Pub.",
  "New-York",
  03);
```

HW_SimpleBook.sql[95]

The values inserted in the database is just to provide some examples; you should assume there is more data in it than what we have inserted. In this long problem, you will be asked to write commands to select, update, delete, insert data, and to improve upon the relational model.

Pb 3.13 – Question 1 Write a command that selects:

1. The `Title` of all the books.
2. The *distinct* `Name` of the publishers.
3. The `Titles` and `Published` dates of the books published since January 31, 2012.
4. The first and last names of the authors published by `"Gallimard"` (from any city).
5. The first and last names of the authors who were not published by an editor in `"New-York"`.
6. The `ID` of the authors who published a book whose name starts with `"Where"`.
7. The total number of pages in the database.
8. The number of pages in the longest book written by the author whose last name is `"Wolve"`.
9. The titles of the books published in the 19th century.

Pb 3.13 – Question 2 Write a command that updates the title of all the books written by the author whose `ID` is `3` to `"BANNED"`. Is there any reason for this command to be rejected by the system? If yes, explain the reason.

Pb 3.13 – Question 3 Write one or multiple commands that would delete the author whose `ID` is `3` and all the books written by that author. Make sure you do not violate any foreign key constraints.

Pb 3.13 – Question 4 Write a command that would create a table used to record the awards granted to authors for particular books. Assume that each award has its own name, is awarded every year, and that it is awarded to an author for a particular book. Pick appropriate attributes, datatypes[96], primary and foreign keys, and, as always, avoid redundancy.

Pb 3.13 – Question 5 Draw the relational model of the database you created (including all the relations given in the code and the ones you added).

Pb 3.13 – Question 6 Discuss two limitations of the model and how to improve it.

Problem 3.14 ***(A database for website certificates)*** A certificate for a website has a serial number (`SN`) and a common name (`CN`). It must belong to an organization and be signed by a certificate authority (`CA`). The organization and `CA` must both have an `SN` and a `CN`. A `CA` can be trusted, not trusted, or not evaluated. The code below is an attempt to represent this situation and is populated with examples.

```
CREATE TABLE ORGANIZATION(
    SN VARCHAR(30) PRIMARY KEY,
    CN VARCHAR(30)
    );
```

[95] https://rocketgit.com/user/caubert/CSCI_3410/source/tree/branch/master/blob/notes/code/sql/HW_SimpleBook.sql

[96] You can use the `DATE` datatype to store a year.

```sql
CREATE TABLE CA(
    SN VARCHAR(30) PRIMARY KEY,
    CN VARCHAR(30),
    Trusted BOOL
    );

CREATE TABLE CERTIFICATE(
    SN VARCHAR(30) PRIMARY KEY,
    CN Varchar(30),
    Org VARCHAR(30) NOT NULL,
    Issuer VARCHAR(30) NOT NULL,
    Valid_Since DATE,
    Valid_Until DATE,
    FOREIGN KEY (Org)
        REFERENCES ORGANIZATION(SN)
        ON DELETE CASCADE,
    FOREIGN KEY (Issuer) REFERENCES CA(SN)
    );

INSERT INTO ORGANIZATION VALUES
    ('01', 'Wikimedia Foundation'),
    ('02', 'Free Software Foundation');

INSERT INTO CA VALUES
    ('A', "Let's Encrypt", true),
    ('B', 'Shady Corp.', false),
    ('C', 'NewComer Ltd.', NULL);

INSERT INTO CERTIFICATE VALUES
    ('a', '*.wikimedia.org', '01', 'A',
          20180101, 20200101),
    ('b', '*.fsf.org', '02', 'A',
          20180101, 20191010),
    ('c', '*.shadytest.org', '02', 'B',
          20190101, 20200101),
    ('d', '*.wikipedia.org', '01', 'C',
          20200101, 20220101);
```

1. Write queries that return the following information. The values returned *in this set-up* will be in parenthesis, but keep the queries general.
   a) The `CN`'s of all certificates (`"*.wikimedia.org, \*.fsf.org, \*.shadytest.org, \`
   b) The `SN`'s of the organizations whose `CN` contains `"Foundation"` (`"01, 02"`).
   c) The `CN`'s and expiration dates of all the certificates that expired, assuming today is the 6th of December 2019 (`"\*.fsf.org"`, 2019 − 10 − 10).
   d) The `CN`'s of the `CA`'s that are not trusted (`"Shady Corp., NewComer Ltd."`),

e) The CN's of the certificates that are signed by a CA that is not trusted ("*.shadytest.org, *.wikipedia.org").

f) The number of certificates signed by the CA whose CN is "Let's encrypt" (2).

g) A table listing the CN's of the organizations along with the CN's of their certificates ("Wikimedia Foundation, *.wikimedia.org, Free Software Foundat

2. *In this set-up*, what happens if the following commands are issued? List all the entries that are modified or deleted, or specify if the command would not change anything and explain why.

   a) `DELETE FROM CA WHERE SN = 'A';`

   b) `UPDATE ORGANIZATION SET CN = "FSF" WHERE SN = '02';`

   c) `UPDATE ORGANIZATION SET SN = "01" WHERE SN = '02';`

   d) `DELETE FROM ORGANIZATION;`

Problem 3.15 ***(A simple database for published pieces of work)*** Consider the following code:

```
/* code/sql/HW_Work.sql */
CREATE TABLE AUTHOR (
  NAME VARCHAR(30) PRIMARY KEY,
  Email VARCHAR(30)
);

CREATE TABLE WORK (
  Title VARCHAR(30) PRIMARY KEY,
  Author VARCHAR(30),
  FOREIGN KEY (Author) REFERENCES AUTHOR (NAME) ON DELETE
    CASCADE ON UPDATE CASCADE
);

CREATE TABLE BOOK (
  ISBN INT PRIMARY KEY,
  Work VARCHAR(30),
  Published DATE,
  Price DECIMAL(10, 2),
  FOREIGN KEY (WORK) REFERENCES WORK (Title) ON DELETE
    RESTRICT ON UPDATE CASCADE
);

CREATE TABLE EBOOK (
  ISBN INT PRIMARY KEY,
  Work VARCHAR(30),
  Published DATE,
  Price DECIMAL(10, 2),
  FOREIGN KEY (WORK) REFERENCES WORK (Title) ON DELETE
    RESTRICT ON UPDATE CASCADE
);

```

```
39 INSERT INTO AUTHOR
40 VALUES (
41   "Virginia W.",
42   "vw@isp.net"), -- A.1
43 (
44   "Paul B.", "pb@isp.net"), -- A.2
45 (
46   "Samantha T.", "st@fai.fr") -- A.3
47 ;
48
49 INSERT INTO WORK
50 VALUES (
51   "What to eat",
52   "Virginia W.") -- W.1
53 ;
54
55 INSERT INTO BOOK
56 VALUES (
57   15155627,
58   "What to eat",
59   DATE '20170219',
60   12.89) -- B.1
61 ;
62
63 INSERT INTO EBOOK
64 VALUES (
65   15155628,
66   "What to eat",
67   DATE '20170215',
68   9.89) -- E.1
69 ;
70
```

HW_Work.sql[97]

Assume the following:

1. Every statement respects `SQL`'s syntax (there's no "a semi-colon is missing" trap).
2. None of the commands in the rest of this problem are actually executed; they are for hypothetical "what if" questions.

Also, note that each row inserted between line 39 and 50 is given a name in comment (`"A.1, A.2, A.3, W.1"`, etc.).

Pb 3.15 – Question 1 Draw the relational model corresponding to this series of commands.

Pb 3.15 – Question 2 Determine if the following insertion statements would violate the entity integrity constraint, the referential integrity constraint, if there would be some other kind of error, or if it would result in successful insertion.

[97] https://rocketgit.com/user/caubert/CSCI_3410/source/tree/branch/master/blob/notes/code/sql/HW_Work.sql

```sql
INSERT INTO EBOOK VALUES (0, NULL, 20180101, 0);
INSERT INTO AUTHOR VALUES("Mary B.", "mb@fai.fr", NULL);
INSERT INTO WORK VALUES("My Life", "Claude A.");
INSERT INTO BOOK VALUES(00000000, NULL, DATE'20001225',
 ↪ 90.9);
INSERT INTO AUTHOR VALUES("Virginia W.", "alt@isp.net");
```

Pb 3.15 – Question 3 List the rows (A.2, W.1, etc.) modified by the following statements. Be careful about the conditions on foreign keys!

```sql
UPDATE AUTHOR SET Email = 'Deprecated' WHERE Email LIKE
 ↪ '%isp.net';
UPDATE WORK SET Title = "How to eat" WHERE Title = "What to
 ↪ eat";
DELETE FROM WORK;
DELETE FROM AUTHOR WHERE Name = "Virginia W.";
```

Pb 3.15 – Question 4 Assume that there is more data than what we inserted. Write a command that selects:

- The prices of all the ebooks.
- The *distinct* names of the authors who have authored a piece of work.
- The names of the authors using fai.fr for their email domain.
- The prices of the ebooks published after 2018.
- The price of the most expensive book.
- The number of the pieces of work written by the author whose name is "Virginia W."
- The email of the author who wrote the piece called "My Life."
- The ISBN's of the books containing a work written by the author whose email is "vw@isp.net."

Pb 3.15 – Question 5 Write a command that updates the title of all the pieces of work written by the author whose name is "Virginia W." to "BANNED." Is there any reason for this command to be rejected by the system? If yes, explain the reason.

Pb 3.15 – Question 6 Write one or multiple commands that would delete the work whose title is "My Life", as well as all of the book and ebook versions of it.

Pb 3.15 – Question 7 Discuss two limitations of the model and how to improve it.

Problem 3.16 ***(A simple database for authors of textbooks)*** Consider the following code:

```sql
/* code/sql/HW_TextbookAuthored.sql */
DROP SCHEMA IF EXISTS HW_TEXTBOOKAUTHORED;

CREATE SCHEMA HW_TEXTBOOKAUTHORED;

USE HW_TEXTBOOKAUTHORED;

CREATE TABLE TEXTBOOK (
  Title VARCHAR(50),
```

```
  ISBN CHAR(13) PRIMARY KEY,
  Price DECIMAL(10, 2)
);

CREATE TABLE AUTHOR (
  LName VARCHAR(30),
  FName VARCHAR(30),
  Email VARCHAR(30),
  PRIMARY KEY (Lname, Fname)
);

CREATE TABLE AUTHORED (
  Book CHAR(13),
  FOREIGN KEY (Book) REFERENCES TEXTBOOK (ISBN),
  AuthorLName VARCHAR(30),
  AuthorFName VARCHAR(30),
  FOREIGN KEY (AuthorLName, AuthorFName) REFERENCES AUTHOR
    (LName, Fname)
);

INSERT INTO TEXTBOOK
VALUES (
  'Starting Out with Java: Early Objects',
  9780133776744,
  30.00),
(
  'NoSQL for Mere Mortals',
  9780134023212,
  47.99);

INSERT INTO AUTHOR
VALUES (
  'Sullivan',
  'Dan',
  NULL),
(
  'Gaddis',
  'Tony',
  NULL);

INSERT INTO AUTHORED
VALUES (
  9780134023212,
  'Sullivan',
  'Dan'),
(
  9780133776744,
  'Gaddis',
  'Tony');
```

HW_TextbookAuthored.sql[98]

The meaning of the `AUTHORED` table is that a tuple $< I, L, F >$ represents that the author whose last name is L and whose first name is F wrote the textbook whose ISBN is I.

Answer the following:

1. Write a command that updates the email address of 'Gaddis', 'Tony.'
2. Write a command that inserts a textbook of your choice into the `TEXTBOOK` table. No value should be **NULL**.
3. Write a command that makes 'Gaddis', 'Tony' the author of the textbook you just added to our database.
4. Write a command that makes "0.01" the default value for the `Price` attribute of the `TEXTBOOK` relation.
5. Write a command that inserts a textbook of your choice in the `TEXTBOOK` table and have the price set to the default value.
6. Write a command that creates a table called `EDITOR` with three attributes: `Name`, `Address`, and `Website`. The `Name` attribute should be the primary key. Insert two tuples in the `EDITOR` table, making sure that one should has the `Name` attribute set to "Pearson."
7. Write a command that creates a table called `PUBLISHED` with two attributes: `Editor` and `Textbook`. The `Editor` attribute should reference the `EDITOR` table and the `Textbook` attribute should reference the `TEXTBOOK` table.
8. Write a command that makes "Pearson" the editor of the textbook whose ISBN is 9780133776744.

Answer the following short questions based on what is in our model so far:

1. Can an author have authored more than one textbook?
2. Can a textbook have more than one author?
3. Can a textbook without an `ISBN` be inserted in the `TEXTBOOK` relation?
4. Can the price of a textbook be negative?
5. Can two authors have the same first and last names?
6. Can two textbooks have the same title?
7. Can two editors have the same address?

Problem 3.17 ***(A simple database for capstone projects)*** Consider the following code:

```sql
/* code/sql/HW_Capstone.sql */
DROP SCHEMA IF EXISTS HW_CAPSTONE;

CREATE SCHEMA HW_CAPSTONE;

USE HW_CAPSTONE;

CREATE TABLE STUDENT (
  FName VARCHAR(50),
  Id CHAR(13) PRIMARY KEY,
```

[98] https://rocketgit.com/user/caubert/CSCI_3410/source/tree/branch/master/blob/notes/code/sql/HW_TextbookAuthored.sql

```
  GraduationYear INT,
  GraduationSemester ENUM ("Fall", "Spring", "Summer")
);

CREATE TABLE PROGRAMMING_LANGUAGE (
  NAME VARCHAR(50) PRIMARY KEY,
  Licence VARCHAR(50)
);

CREATE TABLE PROJECT (
  CodeName VARCHAR(50),
  Leader CHAR(13),
  PRIMARY KEY (CodeName, Leader),
  FOREIGN KEY (Leader) REFERENCES STUDENT (Id)
);

CREATE TABLE USED_LANGUAGE (
  ProjectCodeName VARCHAR(50),
  ProjectLeader CHAR(13),
  UsedLanguage VARCHAR(50),
  PRIMARY KEY (ProjectCodeName, ProjectLeader, UsedLanguage),
  FOREIGN KEY (ProjectCodeName, ProjectLeader) REFERENCES
    PROJECT (CodeName, Leader),
  FOREIGN KEY (UsedLanguage) REFERENCES PROGRAMMING_LANGUAGE (NAME)
);


/*
 */
INSERT INTO STUDENT
VALUES (
  "Mary",
  "0123456789100",
  2025,
  "Summer"),
(
  "Steve",
  "0000000000000",
  2025,
  "Fall"),
(
  "Claude",
  "9999999999999",
  2024,
  "Fall"),
(
  "Meghan",
  "0987654321098",
  2023,
```

```sql
  "Spring");

INSERT INTO PROGRAMMING_LANGUAGE
VALUES (
  "Rust",
  "MIT"),
(
  ".NET Core",
  "MIT"),
(
  "Racket",
  "LGPL"),
(
  "Python",
  "PSF");

-- Taken from
-- https://en.wikipedia.org/wiki/Comparison_of_open-source_programming_language_licensing
INSERT INTO PROJECT
VALUES (
  "Brick Break",
  "0123456789100"),
(
  "Brick Break",
  "0000000000000"),
(
  "Grade Calculator",
  "0123456789100"),
(
  "Undecided",
  "9999999999999");

INSERT INTO USED_LANGUAGE
VALUES (
  "Brick Break",
  "0123456789100",
  "Rust"),
(
  "Brick Break",
  "0000000000000",
  ".NET Core"),
(
  "Brick Break",
  "0000000000000",
  "Python"),
(
  "Grade Calculator",
  "0123456789100",
  "Racket");
```

HW_Capstone.sql[99]

The meaning of the USED_LANGUAGE table is that a tuple $< N, L, U >$ represents the fact that the project whose code name is N and whose leader is L uses the programming language U.

Pb 3.17 – Question 1 Answer the following short questions based on the model implemented above. You can simply answer “True” or “False”, or justify your reasoning (e.g. with code).

1. Can a project uses multiple programming languages?
2. Can a student be the leader of multiple projects?
3. Can multiple projects have the same code name?
4. Could Claude simply enter NULL for the value of his project’s code name, since he’s undecided?
5. Can a project be created without project leader?
6. Can we know who is working on a project without being its leader?

Pb 3.17 – Question 2 Draw the relational model corresponding to this code.

Pb 3.17 – Question 3 Write the following commands.

1. Write a command that insert a new student in the STUDENT table.
2. Write a command that updates the code name of the project (“Undecided”, “9999999999999”) to “VR in ER”.
3. Write a command that updates the graduation year of the student whose id is “0987654321098” to 2024, and the semester to “Fall”.
4. Write a command that changes the STUDENT table to make it impossible to enter NULL for the first name of a student, without changing the primary key.
5. Write a command that changes the datatype of GraduationYear to SMALLINT.
6. Write a command that adds an attribute “ReleaseDate” to the PROJECT table.
7. If you managed to write the previous command correctly, write a command that sets the release date of the project (“Brick Break”, “0123456789100”) to the 26th of November 2022.
8. Write a command that makes it impossible for a student to be the leader in more than one project

---

Problem 3.18 ***(A simple database for vaccines)*** Consider the following code:

```sql
/* code/sql/HW_Vaccine.sql */
CREATE TABLE COMPANY (
  Name VARCHAR(50) PRIMARY KEY,
  Website VARCHAR(255) CHECK (Website LIKE "https://%")
);

CREATE TABLE DISEASE (
  Name VARCHAR(50) PRIMARY KEY,
  Communicable BOOL,
  -- Whether the disease can be transmitted from a human to
  --       another.
  TYPE ENUM ("infectious", "deficiency", "hereditary")
```

[99] https://rocketgit.com/user/caubert/CSCI_3410/source/tree/branch/master/blob/notes/code/sql/HW_Capstone.sql

```sql
);

CREATE TABLE VACCINE (
  Name VARCHAR(50) PRIMARY KEY,
  Manufacturer VARCHAR(50) NOT NULL,
  FOREIGN KEY (Manufacturer) REFERENCES COMPANY (NAME) ON
    UPDATE CASCADE
);

CREATE TABLE EFFICACY (
  DiseaseName VARCHAR(50),
  VaccineName VARCHAR(50),
  Efficacy DECIMAl(5, 2),
  PRIMARY KEY (DiseaseName, VaccineName),
  FOREIGN KEY (DiseaseName) REFERENCES DISEASE (NAME),
  FOREIGN KEY (VaccineName) REFERENCES VACCINE (NAME)
);

INSERT INTO COMPANY
VALUES (
  "Moderna",
  "https://www.modernatx.com/");

INSERT INTO DISEASE
VALUES (
  "Coronavirus disease 2019",
  TRUE,
  "infectious");

INSERT INTO VACCINE
VALUES (
  "mRNA-1273",
  "Moderna");

INSERT INTO EFFICACY
VALUES (
  "Coronavirus disease 2019",
  "mRNA-1273",
  94.1);
```

HW_Vaccine.sql[100]

Pb 3.18 – Question 1 Answer the following short questions. In our implementation...

1. ... can two companies have exactly the same name?
2. ... can two companies have the same website?
3. ... can a company not have a website?
4. ... can the same vaccine be manufactured by multiple companies?

[100] https://rocketgit.com/user/caubert/CSCI_3410/source/tree/branch/master/blob/notes/code/sql/HW_Vaccine.sql

5. ... can a vaccine not have a manufacturer?
6. ... can a disease being neither communicable nor not communicable?
7. ... can the same vaccine have different efficacies for different diseases?

Pb 3.18 – Question 2 Answer the following questions:

1. What does `CHECK (Website LIKE "https://*")` do?
2. Why did we picked the `DECIMAl(5,2)` datatype?
3. What is the benefit / are the benefits of having a separate EFFICACY table over having something like

```
CREATE TABLE VACCINE(
    Name VARCHAR(50) PRIMARY KEY,
    Manufacturer VARCHAR(50),
    Disease VARCHAR(50),
    Efficacy DECIMAl(5,2),
    FOREIGN KEY (Manufacturer) REFERENCES COMPANY (Name)
);
```

?

Pb 3.18 – Question 3 Draw the relational model corresponding to this code.

Pb 3.17 – Question 4 Write the following commands.

1. Write a command that insert "Pfizer" in the COMPANY table (you can make up the website or look it)
2. Write a command that insert the "Pfizer-BioNTech COVID-19 Vaccine" in the VACCINE table, and a command that store the efficacy of that vaccine against the "Coronavirus disease 2019" disease (you can make up the values or look them up).
3. Write a command that updates the name of the company "Moderna" to "Moderna, Inc." everywhere.
4. Write a command that lists the name of all the companies.
5. Write a command that deletes the "Coronavirus disease 2019" entry from the DISEASE table (if only!). This command should return an error. Explain it and leave the command commented.
6. Write two commands: one that adds "physiological" to the possible types of diseases, and one that inserts a physiological disease in the DISEASE table.
7. Write a command that return the list of all the companies that manufacture a vaccine against "Coronavirus disease 2019".

Problem 3.19 ***(A database for residencies)*** Consider the following code:

```
/* code/sql/HW_Residency.sql */
DROP SCHEMA IF EXISTS HW_RESIDENCY;

CREATE SCHEMA HW_RESIDENCY;

USE HW_RESIDENCY;

```

```
CREATE TABLE PERSON (
  FName VARCHAR(40),
  LName VARCHAR(40),
  SSN VARCHAR(11) PRIMARY KEY,
  Birthdate DATE
);

CREATE TABLE HOUSE (
  Address VARCHAR(40) PRIMARY KEY,
  Color ENUM ("blue", "white", "green")
);

CREATE TABLE RESIDENCY (
  Person VARCHAR(11),
  House VARCHAR(40),
  PrincipalResidence BOOLEAN,
  Status ENUM ("own", "rent", "squat", "other"),
  FOREIGN KEY (Person) REFERENCES PERSON (SSN),
  FOREIGN KEY (House) REFERENCES HOUSE (Address) ON DELETE CASCADE
);

INSERT INTO PERSON
VALUES (
  NULL,
  "Doe",
  "000-00-0000",
  NULL), -- P.1
(
  "Michael", "Keal", "000-00-0001", DATE "1983-02-11"), -- P.2
(
  "James", "Baldwin", "000-00-0002", DATE
    "1967-01-01"), -- P.3
(
  "Mridula", "Warrier", "000-00-0003", DATE "1990-02-11");

-- P.4
INSERT INTO HOUSE
VALUES (
  "123 Main St.",
  "blue"), -- H.1
(
  "456 Second St.", "white"), -- H.2
(
  "11 Third St.", "blue");

-- H.3
INSERT INTO RESIDENCY
VALUES (
  "000-00-0001",
  "123 Main St.",
```

```
  TRUE,
  "own"), -- R.1
(
  "000-00-0001", "456 Second St.", FALSE, "own"), -- R.2
(
  "000-00-0002", "123 Main St.", TRUE, "rent"), -- R.3
(
  "000-00-0003", "456 Second St.", TRUE, "own");

-- R.4
```

HW_Residency.sql[101]

Note that each row inserted in the `PERSON`, `HOUSE` and `RESIDENCY` tables is given the name and noted as afterwards as a comment (`"P.1, P.2, P.3, P.4, H.1"`, etc.).

Answer the following questions and problems, assuming that none of the commands in the rest of the problem are actually executed.

Pb 3.19 – Question 1 Draw the relational model corresponding to this series of commands (it is not necessary to include the state).

Pb 3.19 – Question 2 Write a command that violates the entity integrity constraint.

Pb 3.19 – Question 3 Write a command that violates the referential integrity constraint.

Pb 3.19 – Question 4 List the rows (e.g. “P.2”, “H.1”, or “none”) modified by the following statements:

1. `UPDATE HOUSE SET COLOR = "green";`
2. `DELETE FROM RESIDENCY WHERE House LIKE "1%";`
3. `DELETE FROM HOUSE WHERE Address = "456 Second St.";`
4. `DELETE FROM PERSON WHERE Birthdate=DATE"1990-02-11";`

Pb 3.19 – Question 5 Write queries that return the following information. The values returned *in this set-up* will be in parenthesis, but keep the queries general.

1. The `Addresses' of the houses in the system (`“11 Third St., 123 Main St., 456 Second St.”‘).
2. The `SSN`’s of the people whose first name was not entered in the system (`"000-00-0000"`).
3. All the different colors of houses (`"white, blue"`).
4. The `Address` of the residency of `"James Baldwin"` (`"123 Main St."`).
5. The first name of the oldest person in the database (`"James"`).
6. `"Michael Keal"`’s principal residency address (`"123 Main St."`).
7. The *distinct* first and last names of the homeowners (`"Michael Keal, Mridula Warrier"`).
8. The `SSN`’s of the people that have the same principal residency as `"James Baldwin"` (`"000-00-0001"`).

Pb 3.19 – Question 6 Write a command that updates the `SSN` of `"James Baldwin"` to `"000-00-0010"`. Is there any reason for this command to be rejected by the system? If yes, explain the reason.

[101] https://rocketgit.com/user/caubert/CSCI_3410/source/tree/branch/master/blob/notes/code/sql/HW_Residency.sql

Pb 3.19 – Question 7 Answer the following short questions from the data in our model, as it is currently:

1. Is it possible for two people to have the same last name?
2. Is it possible for a person to have multiple principal residencies?
3. Is it possible for a house to not be yellow?
4. Is it possible for the `SSN` to be any series of 11 characters?
5. Is it possible for a person to own any number of houses?
6. Is it possible for a person to rent at most one house?

Pb 3.19 – Question 8 Consider the data currently in the `RESIDENCY` table and give a possible primary key.

Pb 3.19 – Question 9 Discuss why the primary key identified from the previous question for the `RESIDENCY` table is a good choice.

Problem 3.20 ***(A database for research fundings)*** Consider the following code:

```
/* code/sql/HW_ScientificResearch.sql */
CREATE TABLE SCIENTIST (
  SSN INT PRIMARY KEY,
  Name VARCHAR(30) NOT NULL
);

CREATE TABLE PROJECT (
  Code CHAR(4) PRIMARY KEY,
  Name VARCHAR(150) NOT NULL
);

CREATE TABLE CONTRIBUTESTO (
  Scientist INT,
  Project CHAR(4),
  Hours INT,
  PRIMARY KEY (Scientist, Project),
  FOREIGN KEY (Scientist) REFERENCES SCIENTIST (SSN),
  FOREIGN KEY (Project) REFERENCES PROJECT (Code) ON DELETE
    CASCADE ON UPDATE CASCADE
);

CREATE TABLE FUNDINGAGENCY (
  Name VARCHAR(150) PRIMARY KEY,
  TYPE ENUM ("State", "Federal", "Foundation"),
  Creation YEAR
);

CREATE TABLE FUNDS (
  Agency VARCHAR(150),
  Project CHAR(4),
  Amount DECIMAL(12, 2),
```

```
  FOREIGN KEY (Agency) REFERENCES FUNDINGAGENCY (NAME) ON
    UPDATE CASCADE ON DELETE RESTRICT,
  FOREIGN KEY (Project) REFERENCES PROJECT (Code)
);

INSERT INTO SCIENTIST
VALUES (
  "000000000",
  "Mike"), -- S.1
(
  "000000001", "Sabine"), -- S.2
(
  "000000002", "James"), -- S.3
(
  "000000003", "Emily"), -- S.4
(
  "000000004", "Claire");

-- S.5
INSERT INTO PROJECT
VALUES (
  "AA",
  "Advancing Airplanes"), -- P.1
(
  "BA", "Better Airplanes"), -- P.2
(
  "BB", "Better Buildings"), -- P.3
(
  "CC", "Creative Creation");

-- P.4
INSERT INTO CONTRIBUTESTO
VALUES (
  "000000001",
  "AA",
  12), -- C.1
(
  "000000001", "BB", 10), -- C.2
(
  "000000002", "AA", 5), -- C.3
(
  "000000003", "BA", 3), -- C.4
(
  "000000000", "BB", 1), -- C.5
(
  "000000000", "AA", 1);

-- C.6
INSERT INTO FUNDINGAGENCY
VALUES (
```

```
89   "National Science Foundation",
90   "Federal",
91   1950), -- FA.1
92 (
93   "French-American Cultural Exchange", "Foundation", 2017);
94
95 -- FA.2
96 INSERT INTO FUNDS
97 VALUES (
98   "National Science Foundation",
99   "AA",
100  100000), -- F.1
101 (
102  "French-American Cultural Exchange", "CC", 10000);
103
104 -- F.2
```

HW_ScientificResearch.sql[102]

Note that each row inserted in the tables is given a name and noted as afterwards as a comment (`"S.1, S.2, P.1, C.1, FA.1"`, etc.).

Answer the following questions and problems, assuming that none of the commands in the rest of the problem are actually executed.

Pb 3.20 – Question 1 Draw the relational model corresponding to this series of commands (it is not necessary to include the state).

Pb 3.20 – Question 2 Draw the relational model corresponding to this series of commands (no need to include the state).

Pb 3.20 – Question 3 How could you edit line 12 so that negative values and `NULL` would not be admitted as values for `Hours`?

Pb 3.20 – Question 4 Write a command that would violate the referential integrity constraint.

Pb 3.20 – Question 5 List the rows affected (updated or deleted) by the following commands. If no rows are affected because the command would would violate the entity integrity constraint, the referential integrity constraint, or if there would be some other kind of error, please indicate it.

1. `UPDATE SCIENTIST SET SSN = "000000001" WHERE Name = "Claire";`
2. `UPDATE FUNDINGAGENCY SET Name = "NSF" WHERE Name = "National Science Fo`
3. `DELETE FROM FUNDINGAGENCY WHERE Name = "French-American Cultural Exchar`

Pb 3.20 – Question 6 Write a query that selects ...(In parenthesis, the values returned *in this set-up*, but you have to be general.)

1. ...the name of the funding agencies created after 2000 ("French-American Cultural Exchange")
2. ...the code of the projects that contains the word "Airplanes" ("AA", "BA")

[102] https://rocketgit.com/user/caubert/CSCI_3410/source/tree/branch/master/blob/notes/code/sql/HW_ScientificResearch.sql

3. ...the number of hours scientists contributed to the project ”AA” (18)
4. ...the code of the projects to which the scientist named Sabine contributed (”AA”, ”BB”)
5. ...the name of the projects who benefited from federal funds (”Advancing Airplanes”)
6. ...the name of the scientist who contributed to the same project as Mike (”Sabine”, ”James”)
7. ...the name of the projects that are not funded by an agency (”Better Airplanes”, ”Better Buildings”)
8. ...the name of the scientist who contributed the most (in terms of hours) to the project named ”Advancing Airplanes” (Sabine).

Pb 3.20 – Question 7 Identify and discuss two limitations of this model, and offer a way to remedy at least one of them.

Problem 3.21 *(Improving a Relational Model for a Printing Station)* Consider the following code:

```
CREATE TABLE ROOM(
     Nickname VARCHAR(40) PRIMARY KEY,
     Size INT,
     ComputerOrPhoneInIt BOOL NOT NULL
 );

 CREATE TABLE COMPUTER(
     Nickname VARCHAR(40) PRIMARY KEY,
     OperatingSystem VARCHAR(50),
     Room VARCHAR(40),
     FOREIGN KEY (Room) REFERENCES ROOM(Nickname)
 );

 CREATE TABLE PHONE(
     Nickname VARCHAR(40) PRIMARY KEY,
     OperatingSystem VARCHAR(50),
     Room VARCHAR(40),
     FOREIGN KEY (Room) REFERENCES ROOM(Nickname)
 );


 CREATE TABLE PRINTER(
     Nickname VARCHAR(40) PRIMARY KEY,
     ConnectedTo VARCHAR(40),
     Room VARCHAR(40),
     FOREIGN KEY (Room) REFERENCES ROOM(Nickname)
 );
```

It was written by some friends of yours to store data for their printing station: their shop offers computers, phones and printers located in different rooms, and they want to keep track of some information about those. They have multiple issues with this implementation, and require your help to identify them and design a new model addressing those.

Pb 3.21 – Question 1 For each issue listed below, explain what causes it, and if there is a way to address it (you do not need to write actual code, simply explain how you would proceed, using keywords if it clarifies).

- They have a hard time coming up with different nicknames for their printers, computers, rooms and phones every time they add one.
- The attribute `ROOM.ComputerOrPhoneInIt` is a bit cumbersome, as they have to remember to update it to `FALSE` if the last computer or phone is removed from a room.
- The `OperatingSystem` attributes in `PHONE` and `COMPUTER` are not very convenient, as they do not provide an easy way for instance to list all the windows computers and phones, or all the 64-bits architectures. Examples of values are "Windows 10 IoT Core version 1.8.9, 64 bits","Android Pie", "macOS 11 Big Sur, updated last week", etc.
- It seems that they cannot record when a printer is connected to more than one device.

Pb 3.21 – Question 2 Draw a *relational model* (no need to write `sql` code) that would improve their implementation. You are free (and encouraged!) to create new relations and alter the original attributes. No need to specify the domains unless you want to add particular constraints, but remember to draw the primary and foreign keys.

Problem 3.22 ***(Write select queries for a (third!) variation of the COMPUTER table)***
Consider the following code:

```
CREATE TABLE COMPUTER (
    ID VARCHAR(20) PRIMARY KEY,
    Model VARCHAR(40)
);

CREATE TABLE PERIPHERAL (
    ID VARCHAR(20) PRIMARY KEY,
    Model VARCHAR(40),
    Type ENUM ('mouse', 'keyboard', 'screen', 'printer'),
    LastConnexion DATETIME
);

CREATE TABLE CONNEXION (
    Computer VARCHAR(20),
    Peripheral VARCHAR(20),
    PRIMARY KEY (Computer, Peripheral),
    FOREIGN KEY (Computer) REFERENCES COMPUTER (ID)
        ON DELETE CASCADE
        ON UPDATE CASCADE,
    FOREIGN KEY (Peripheral) REFERENCES PERIPHERAL (ID)
        ON DELETE RESTRICT
        ON UPDATE CASCADE
);
```

```sql
CREATE TRIGGER last_connexion_update
    BEFORE INSERT ON CONNEXION
    FOR EACH ROW
        UPDATE PERIPHERAL
        SET LastConnexion = NOW()
        WHERE NEW.Peripheral = PERIPHERAL.ID;

INSERT INTO COMPUTER
VALUES
    ('A','Apple IIc Plus'),   -- C.1
    ('B','Commodore SX-64'); -- C.2

INSERT INTO PERIPHERAL(ID, Model, Type)
VALUES
    ('12', 'Trendcom Model', 'printer'),        -- P.1
    ('14', 'TP-10 Thermal Matrix', 'printer'), -- P.2
    ('15', 'IBM Selectric', 'keyboard');        -- P.3

INSERT INTO CONNEXION
VALUES
    ('A', '12'),  -- X.1
    ('B', '14'),  -- X.2
    ('A', '15');  -- X.3
```

Pb 3.22 – Question 1 Draw the relational model corresponding to this series of commands (no need to include the state).

Pb 3.22 – Question 2 Fill the following table.

	True	False
A peripheral can be connected to multiple computers		
The `ID` of a computer must be a letter		
Whether the connexion is wired or wireless can be determined		
Every computer must have a different model		
A peripheral can be a mouse and a keyboard at the same time		
A computer can be connected to multiple peripheral		
A computer can be connected to another computer		
A peripheral can be connected to another peripheral		

Pb 3.22 – Question 3 List the rows (i.e., C.2, X.1, or even "none") deleted by the following statements:

1. `DELETE FROM CONNEXION WHERE Computer = 'A';`
2. `DELETE FROM COMPUTER WHERE ID = 'A';`
3. `DELETE FROM PERIPHERAL WHERE ID = '15';`
4. `DELETE FROM CONNEXION WHERE Computer <> 'A';`

Pb 3.22 – Question 4 Write a query that selects …(In parenthesis, the values returned *in this set-up*, but you have to be general.)

1. …the type of the peripheral with `Id` 12 (printer)
2. …`ID` of the computer whose model name contain "Apple" (A).

3. ...the number of computer in the database (2).
4. ...all the different kind of peripheral, without duplication (printer, keyboard).
5. ...the ID of the computer connected to a keyboard (A)
6. ...the model of the computer connected to the "TP-10 Thermal Matrix" peripheral (Commodore SX-64).

Pb 3.22 – Question 5 Discuss what would happen after the command `INSERT INTO CONNEXION VALU` is executed.

Solutions to Selected Problems

Solution to Problem 3.2 (Create and use a simple table in SQL) This problem is supposed to be a straightforward application of what we studied in class. Look back at Setting Up Your Work Environment if you feel like you are stuck before referencing this solution.

Pb 3.2 – Solution to Q. 1 We simply log-in as indicated in the "Logging-in as testuser" section. Then we enter:

```
CREATE DATABASE HW_Address;
USE HW_Address;
```

This creates the tables asked for in the problem.

Pb 3.2 – Solution to Q. 2 Ommiting the Extra column, we have:

```
MariaDB [HW_Address]>     DESC ADDRESS;
+------------+-------------+------+-----+---------+
| Field      | Type        | Null | Key | Default |
+------------+-------------+------+-----+---------+
| StreetName | varchar(15) | NO   | PRI | NULL    |
| Number     | int(11)     | NO   | PRI | NULL    |
| Habitants  | int(11)     | YES  |     | NULL    |
+------------+-------------+------+-----+---------+
```

Pb 3.2 – Solution to Q. 3 We add the foreign key, still omitting the Extra column:

```
MariaDB [HW_Address]> DESC ADDRESS;
+------------+-------------+------+-----+---------+
| Field      | Type        | Null | Key | Default |
+------------+-------------+------+-----+---------+
| StreetName | varchar(15) | NO   | PRI | NULL    |
| Number     | int(11)     | NO   | PRI | NULL    |
| Habitants  | int(11)     | YES  | MUL | NULL    |
+------------+-------------+------+-----+---------+
```

The only difference is the MUL value, which is a bit surprising: quoting https://dev.mysql.com/doc/refman/8.0/en/show-columns.html,

> If **Key** is MUL, then the column is the first column of a nonunique index in which multiple occurrences of a given value are permitted within the column.

In other words, this does not carry any information about the fact that ADDRESS.Habitants is now a foreign key referencing NAME.ID. A way of displaying information about that foreign key is using **SHOW CREATE TABLE**:

```
MariaDB [HW_Address]> SHOW CREATE TABLE ADDRESS;
+---------+----------------------+
| Table   | Create Table
+---------+----------------------+
| ADDRESS | CREATE TABLE `ADDRESS` (
`StreetName` varchar(15) NOT NULL,
`Number` int(11) NOT NULL,
`Habitants` int(11) DEFAULT NULL,
PRIMARY KEY (`StreetName`,`Number`),
KEY `Habitants` (`Habitants`),
CONSTRAINT `ADDRESS_ibfk_1` FOREIGN KEY (`Habitants`)
 ↪ REFERENCES `NAME` (`ID`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 |
+---------+----------------------+
1 row in set (0.01 sec)
```

Pb 3.2 – Solution to Q. 4

Pb 3.2 – Solution to Q. 5 To display the information back, we can use

```
SELECT * FROM NAME;
```

We should notice that the ID attribute values lost their leading zeros[103].

Pb 3.2 – Solution to Q. 6 This syntax is better for "bulk insertion" since it allows for us to write fewer commands and to focus on the data being inserted. However, if an error occurs, then nothing gets inserted.

Pb 3.2 – Solution to Q. 7 `SELECT ID FROM NAME WHERE FName = 'Samantha';`

Pb 3.2 – Solution to Q. 8 This is a command that violates the entity integrity constraint:

```
INSERT INTO NAME VALUES ('Maria', 'Kashi', NULL);
```

The error message that it returns is:

```
ERROR 1048 (23000): Column 'ID' cannot be null
```

Another way of violating the entity integrity constraint is:

```
INSERT INTO NAME VALUES ('Maria', 'Kashi', 80);
```

The error message that it returns is:

[103] https://en.wikipedia.org/wiki/Leading_zero

```
ERROR 1062 (23000): Duplicate entry '80' for key
 ↪ 'PRIMARY'
```

Pb 3.2 – Solution to Q. 9 This is an `UPDATE` statement that violates the entity integrity constraint:

```sql
UPDATE ADDRESS SET Habitants = 340 WHERE Number = 120;
```

The error message that it returns is:

```
ERROR 1452 (23000): Cannot add or update a child row: a
 ↪ foreign key constraint fails
 ↪ (`HW_Address`.`ADDRESS`, CONSTRAINT `ADDRESS_ibfk_1`
 ↪ FOREIGN KEY (`Habitants`) REFERENCES `NAME` (`ID`))
```

Pb 3.2 – Solution to Q. 10 Here is the query that violates another type of constraint:

```sql
INSERT INTO NAME VALUE ('Hi');
```

The error message that it returns is:

```
ERROR 1136 (21S01): Column count does not match value
 ↪ count at row 1
```

The query statement violates the implicit constraint by trying to insert a row with fewer values than there are attributes in the table.

Another example of a statement that violates another type of constraint is:

```sql
INSERT INTO ADDRESS VALUES ('Maria', 'Random', 98);
```

This is a violation of an explicit constraint, which is that the value must match the domain (datatype) of the attribute where it is inserted. However, MySQL and MariaDB do not return an error, they simply replace `'Random'` with `0`.

Solution to Problem 3.3 (Duplicate rows in SQL) Here is how we created our table:

```sql
CREATE SCHEMA HW_REPETITION;
USE HW_REPETITION;

CREATE TABLE EXAMPLE(
    X VARCHAR(15),
    Y INT
);
```

Pb 3.3 – Solution to Q. 1 The command to add a tuple to our table is:

```sql
INSERT INTO EXAMPLE VALUES('Train', 4);
```

If we execute this command twice, then SQL is OK with it, and inserts the same tuple twice:

```sql
SELECT * FROM EXAMPLE;
```

Displays:

```
+-------+------+
| X     | Y    |
+-------+------+
| Train |    4 |
| Train |    4 |
+-------+------+
```

This is an illustration of the fact that the data in a table in SQL is *not* a set, as opposed to a state in a relation in the relational model.

Pb 3.3 – Solution to Q. 2 The command:

```
ALTER TABLE EXAMPLE ADD PRIMARY KEY (X);
```

Should return:

```
ERROR 1062 (23000): Duplicate entry 'Train' for key
  'PRIMARY'
```

We tried to declare that X was a primary key, but SQL disagreed, since two rows have the same value for that attribute.

Pb 3.3 – Solution to Q. 3 Once the table is empty, X now qualifies as a candidate key, and can now be made a primary key. SQL stops complaining and lets us assign it as a primary key.

Pb 3.3 – Solution to Q. 4 After trying this insertion statement twice:

```
INSERT INTO EXAMPLE VALUES('Train', 4);
```

SQL refuses to insert the tuple after the second attempt:

```
ERROR 1062 (23000): Duplicate entry 'Train' for key
  'PRIMARY'
```

Notice that this is exactly the same error message as before, when we tried to add the primary key while we had a duplicate row of tuples!

Solution to Problem 3.4 (Constraints on foreign keys)

1. Removing the `PRIMARY KEY` constraint, SQL throws the following error message:

```
ERROR 1005 (HY000): Can't create table
  `HW_FK_test`.`SOURCE` (errno: 150 "Foreign key
  constraint is incorrectly formed")
```

2. Replacing `PRIMARY KEY` with `UNIQUE` does not generate any error messages.
3. Replacing one of the `VARCHAR(25)` with `CHAR(25)` does not generate any error messages.
4. Replacing `VARCHAR(25)` with `INT` results in this error message:

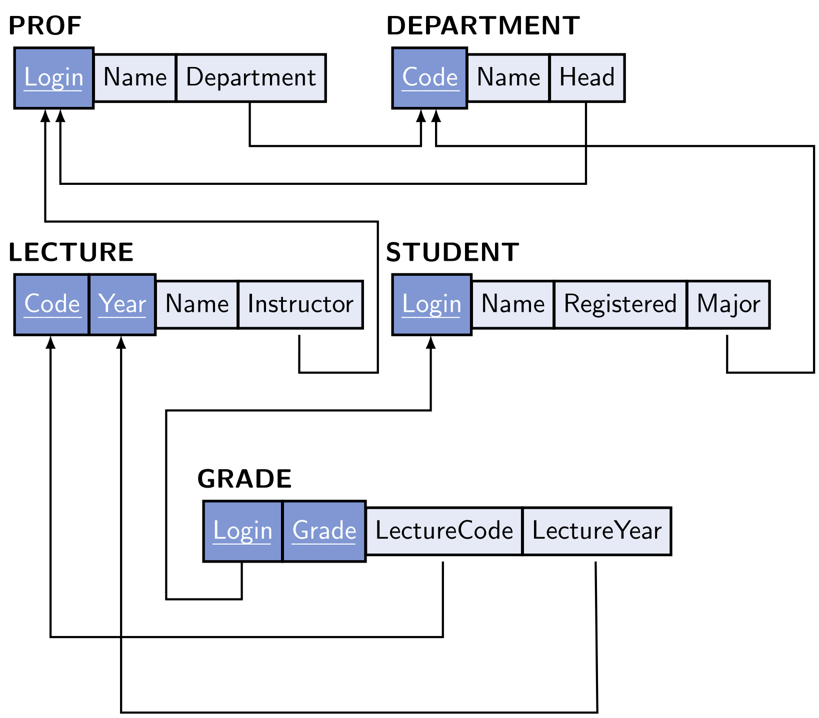
```
ERROR 1005 (HY000): Can't create table
  `HW_FK_test`.`SOURCE` (errno: 150 "Foreign key
  constraint is incorrectly formed")
```

5. Replacing one of the `VARCHAR(25)` with `VARCHAR(15)` does not generate any error messages.

6. The remarks become:

   - The datatype of the foreign key has to **be "compatible"** with the datatype of the attribute to which we are referring.
   - The target of the foreign key *must be* the primary key **or have the `UNIQUE` constraint**.

Solution to Problem 3.5 (Revisiting the PROF table) Pb 3.5– Solution to Q. 1 Ignoring the LECTURE relation, we have:



Pb 3.5– Solution to Q. 2 The code is straightforward:

```
CREATE TABLE HW_Lecture (
  NAME VARCHAR(25),
  Instructor VARCHAR(25),
  Year YEAR (4),
  Code CHAR(5),
  PRIMARY KEY (Year, Code),
  FOREIGN KEY (Instructor) REFERENCES PROF (LOGIN)
);

INSERT INTO HW_Lecture
```

```
VALUES (
  'Intro to CS',
  'caubert',
  2017,
  '1304'),
(
  'Intro
    to Algebra',
  'perdos',
  2017,
  '1405'),
(
  'Intro to
    Cyber',
  'aturing',
  2017,
  '1234');

```

HW_ProfExampleRevisitedRevisited.sql[104]

However, this representation can not handle the following situations:

- If multiple instructors teach the same class,
- If the lecture is taught more than once a year (either because it is taught in the Fall, Spring and Summer, or if multiple sections are offered at the same time),
- If a Lecture is cross-listed, then some duplication of information will be needed.

We come back to those short-coming in the “Reverse-Engineering” section, using more abstract tools (such as Entity Diagrams) that have not been introduced yet.

Pb 3.5– Solution to Q. 3

The statements are immediate:

```
DESCRIBE GRADE;

SELECT *
FROM GRADE;

```

HW_ProfExampleRevisitedRevisited.sql[105]

What may be surprising is that the values for `LectureCode` and `LectureYear` are set to **`NULL`** in all the tuples.

Pb 3.5– Solution to Q. 4

We use **`UPDATE`** statements:

[104] https://rocketgit.com/user/caubert/CSCI_3410/source/tree/branch/master/blob/notes/code/sql/HW_ProfExampleRevisitedRevisited.sql

[105] https://rocketgit.com/user/caubert/CSCI_3410/source/tree/branch/master/blob/notes/code/sql/HW_ProfExampleRevisitedRevisited.sql

```
UPDATE
  GRADE
SET LectureCode = '1304',
  LectureYear = 2017
WHERE LOGIN = 'jrakesh'
  AND Grade = '2.85';

UPDATE
  GRADE
SET LectureCode = '1405',
  LectureYear = 2017
WHERE LOGIN = 'svlatka'
  OR (LOGIN = 'jrakesh'
    AND Grade = '3.85');

UPDATE
  GRADE
SET LectureCode = '1234',
  LectureYear = 2017
WHERE LOGIN = 'aalyx'
  OR LOGIN = 'cjoella';
```

HW_ProfExampleRevisitedRevisited.sql[106]

Pb 3.5– Solution to Q. 5 We refer back to the solution to Q. 1.
Pb 3.5– Solution to Q. 6

We use **SELECT** statements:

```
SELECT LOGIN,
  Grade
FROM GRADE
WHERE Lecturecode = '1304'
  AND LectureYear = '2017';

SELECT DISTINCT Instructor
FROM HW_Lecture
WHERE Year = 2017;

SELECT Name,
  Grade
FROM STUDENT,
  GRADE
WHERE GRADE.LectureCode = 1405
  AND STUDENT.Login = GRADE.Login;

SELECT Year
FROM HW_Lecture
```

[106] https://rocketgit.com/user/caubert/CSCI_3410/source/tree/branch/master/blob/notes/code/sql/HW_ProfExampleRevisitedRevisited.sql

```
WHERE Code = '1234';

SELECT Name
FROM HW_Lecture
WHERE Year IN (
    SELECT Year
    FROM HW_Lecture
    WHERE CODE = '1234');

SELECT B.name
FROM STUDENT AS A,
  STUDENT AS B
WHERE A.Name = 'Ava Alyx'
  AND A.Registered > B.Registered;

SELECT COUNT(DISTINCT PROF.Name) AS 'Head Teaching This Year'
FROM HW_Lecture,
  DEPARTMENT,
  PROF
WHERE Year = 2017
  AND Instructor = Head
  AND Head = PROF.Login;

```

HW_ProfExampleRevisitedRevisited.sql[107]

Solution to Problem 3.6 (TRAIN table and more advanced SQL coding) The code below includes the answers to all of the questions for this problem:

```
-- Question 1:
CREATE TABLE TRAIN (
  Id VARCHAR(30) PRIMARY KEY, -- This line was changed.
  Model VARCHAR(30),
  ConstructionYear YEAR (4)
);

-- Question 2 :
CREATE TABLE CONDUCTOR (
  Id VARCHAR(20),
  NAME VARCHAR(20),
  ExperienceLevel VARCHAR(20)
);

ALTER TABLE CONDUCTOR
  ADD PRIMARY KEY (Id);

```

[107] https://rocketgit.com/user/caubert/CSCI_3410/source/tree/branch/master/blob/notes/code/sql/HW_ProfExampleRevisitedRevisited.sql

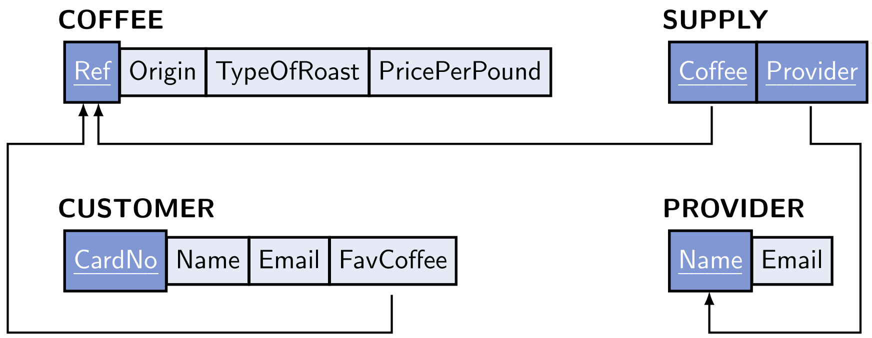
```sql
-- Question 3
CREATE TABLE ASSIGNED_TO (
  TrainId VARCHAR(20),
  ConductorId VARCHAR(20),
  Day DATE,
  PRIMARY KEY (TrainId, ConductorId),
  FOREIGN KEY (TrainId) REFERENCES TRAIN (Id), -- This line was changed
  FOREIGN KEY (ConductorId) REFERENCES CONDUCTOR (Id) -- This line was changed
);

-- Question 4:
/*
 We insert more than one tuple, to make the SELECT statements that follow easier
 to test and debug.
 */
INSERT INTO TRAIN
VALUES (
  'K-13',
  'SurfLiner',
  2019),
(
  'K-12',
  'Regina',
  2015);

INSERT INTO CONDUCTOR
VALUES (
  'GP1029',
  'Bill',
  'Junior'),
(
  'GP1030',
  'Sandrine',
  'Junior');

INSERT INTO ASSIGNED_TO
VALUES (
  'K-13',
  'GP1029',
  DATE '2015/12/14'),
(
  'K-12',
  'GP1030',
  '20120909');

-- Question 5:
UPDATE
```

```sql
  CONDUCTOR
SET ExperienceLevel = 'Senior'
WHERE Id = 'GP1029';

-- Question 6:
-- 1.
SELECT Id
FROM TRAIN;

-- 2.
SELECT Name
FROM CONDUCTOR
WHERE ExperienceLevel = 'Senior';

-- 3.
SELECT ConstructionYear
FROM TRAIN
WHERE Model = 'SurfLiner'
  OR Model = 'Regina';

-- 4.
SELECT ConductorId
FROM ASSIGNED_TO
WHERE TrainId = 'K-13'
  AND Day = '2015/12/14';

-- 5.
SELECT Model
FROM TRAIN,
  ASSIGNED_TO
WHERE ConductorID = 'GP1029'
  AND TrainId = TRAIN.ID;
```

HW_Train.sql[108]

Solution to Problem 3.7 (Read, correct, and write SQL statements for the COFFEE database)
Solution to Question 1:

[108] https://rocketgit.com/user/caubert/CSCI_3410/source/tree/branch/master/blob/notes/code/sql/HW_Train.sql



The answers to the rest of the questions are in the following code:

```sql
/* code/sql/HW_DBCoffee.sql */
-- Question 2:
START TRANSACTION;

INSERT INTO CUSTOMER
VALUES (
  005,
  'Bob Hill',
  NULL,
  001);

INSERT INTO COFFEE
VALUES (
  002,
  "Peru",
  "Decaf",
  3.00);

-- The following statement raises an error.
--                    INSERT INTO PROVIDER
--                      VALUES (NULL, "contact@localcof.com");
--                    ERROR 1048 (23000) at line 68: Column
--      'Name'
--              cannot
--                 be
--                    null
INSERT INTO SUPPLY
VALUES (
  "Johns & Co.",
  121);

-- The following statement raises an error.
--                    -INSERT INTO SUPPLY
--                       VALUES ("Coffee Unl.", 311, 221);
--                     ERROR 1136 (21S01): Column count
```

```
-- doesn't
--        match
--              value
--                    count at row 1
--                     Rest the changes:
ROLLBACK;

-- Question 3:
START TRANSACTION;

UPDATE
  CUSTOMER
SET FavCoffee = 001
WHERE CardNo = 001;

-- Rows matched: 1  Changed: 1  Warnings: 0
SELECT *
FROM CUSTOMER;

ROLLBACK;

START TRANSACTION;

UPDATE
  COFFEE
SET TypeOfRoast = 'Decaf'
WHERE Origin = 'Brazil';

-- Rows matched: 2  Changed: 2  Warnings: 0
SELECT *
FROM COFFEE;

ROLLBACK;

START TRANSACTION;

UPDATE
  PROVIDER
SET Name = 'Coffee Unlimited'
WHERE Name = 'Coffee Unl.';

-- Rows matched: 1  Changed: 1  Warnings: 0
SELECT *
FROM PROVIDER;

SELECT *
FROM SUPPLY;

ROLLBACK;
```

```sql
START TRANSACTION;

UPDATE
  COFFEE
SET PricePerPound = 10.00
WHERE PricePerPound > 10.00;

-- Rows matched: 1  Changed: 1  Warnings: 0
SELECT *
FROM COFFEE;

ROLLBACK;

-- Question 4:
START TRANSACTION;

DELETE FROM CUSTOMER
WHERE Name LIKE '%S%';

-- Query OK, 2 rows affected (0.01 sec)
SELECT *
FROM CUSTOMER;

ROLLBACK;

START TRANSACTION;

DELETE FROM COFFEE
WHERE Ref = 001;

-- Query OK, 1 row affected (0.00 sec)
SELECT *
FROM COFFEE;

SELECT *
FROM SUPPLY;

ROLLBACK;

START TRANSACTION;

DELETE FROM SUPPLY
WHERE Provider = 'Coffee Unl.'
  AND Coffee = '001';

-- Query OK, 1 row affected (0.00 sec)
SELECT *
FROM SUPPLY;

ROLLBACK;
```

```sql

START TRANSACTION;

DELETE FROM PROVIDER
WHERE Name = 'Johns & Co.';

-- Query OK, 1 row affected (0.00 sec)
SELECT *
FROM PROVIDER;

SELECT *
FROM SUPPLY;

ROLLBACK;

-- Question 5:
-- 1.
SELECT Origin
FROM COFFEE
WHERE TypeOfRoast = 'Dark';

-- 2.
SELECT FavCoffee
FROM CUSTOMER
WHERE Name LIKE 'Bob%';

-- 3.
SELECT Name
FROM PROVIDER
WHERE Email IS NULL;

-- 4.
SELECT COUNT(*)
FROM SUPPLY
WHERE Provider = 'Johns & Co.';

-- 5.
SELECT Provider
FROM COFFEE,
  SUPPLY
WHERE TypeOfRoast = 'Dark'
  AND Coffee = Ref;

```

HW_DBCoffee.sql[109]

Solution to Problem 3.8 (Write select queries for the DEPARTMENT table)

[109] https://rocketgit.com/user/caubert/CSCI_3410/source/tree/branch/master/blob/notes/code/sql/HW_DBCoffee.sql

```
SELECT EMPLOYEE.Name
FROM EMPLOYEE,
  DEPARTMENT
WHERE DEPARTMENT.Name = "Storage"
  AND EMPLOYEE.Department = DEPARTMENT.ID;

```

HW_Department.sql[110]

```
SELECT Name
FROM EMPLOYEE
WHERE Hired <= ALL (
    SELECT Hired
    FROM EMPLOYEE
    WHERE Hired IS NOT NULL);

```

HW_Department.sql[111]

```
SELECT EMPLOYEE.Name
FROM EMPLOYEE,
  DEPARTMENT
WHERE Hired <= ALL (
    SELECT Hired
    FROM EMPLOYEE
    WHERE Hired IS NOT NULL
      AND DEPARTMENT.Name = "Storage"
      AND EMPLOYEE.Department = DEPARTMENT.ID)
  AND DEPARTMENT.Name = "Storage"
  AND EMPLOYEE.Department = DEPARTMENT.ID;

```

HW_Department.sql[112]

Solution to Problem 3.11 (Write select queries for a variation of the COMPUTER table)

```
SELECT Model
FROM COMPUTER
WHERE ID = 'A';

SELECT TYPE
FROM PERIPHERAL
WHERE ID = '14';
```

[110]https://rocketgit.com/user/caubert/CSCI_3410/source/tree/branch/master/blob/notes/code/sql/HW_Department.sql

[111]https://rocketgit.com/user/caubert/CSCI_3410/source/tree/branch/master/blob/notes/code/sql/HW_Department.sql

[112]https://rocketgit.com/user/caubert/CSCI_3410/source/tree/branch/master/blob/notes/code/sql/HW_Department.sql

```sql

SELECT Model
FROM PERIPHERAL
WHERE TYPE = 'printer';

SELECT Model
FROM PERIPHERAL
WHERE Model LIKE 'IBM%';

SELECT Model
FROM PERIPHERAL,
  CONNEXION
WHERE Computer = 'A'
  AND Peripheral = PERIPHERAL.ID;

SELECT COUNT(Computer)
FROM CONNEXION,
  COMPUTER
WHERE Model = 'Apple IIc Plus'
  AND Computer = COMPUTER.ID;

```

HW_ComputerVariation.sql[113]

Solution to Problem 3.10 (Write select queries for the SocialMedia schema)

```sql
/* code/sql/HW_SocialMedia.sql */
-- … the title of all the videos ("My first video!", "My
--                    second video!", "My vacations").
SELECT TITLE
FROM VIDEO;

-- … the release date of the video whose title is "My first
--                    video!" ("2020-02-02").
SELECT Released
FROM VIDEO
WHERE Title = "My first video!";

-- … the ID of the account(s) where the "Name" attribute
--                    was not given ("2").
SELECT ID
FROM ACCOUNT
WHERE Name IS NULL;

-- … the ID of the videos whose title contains the word
--                    "video" ("10", "20").
```

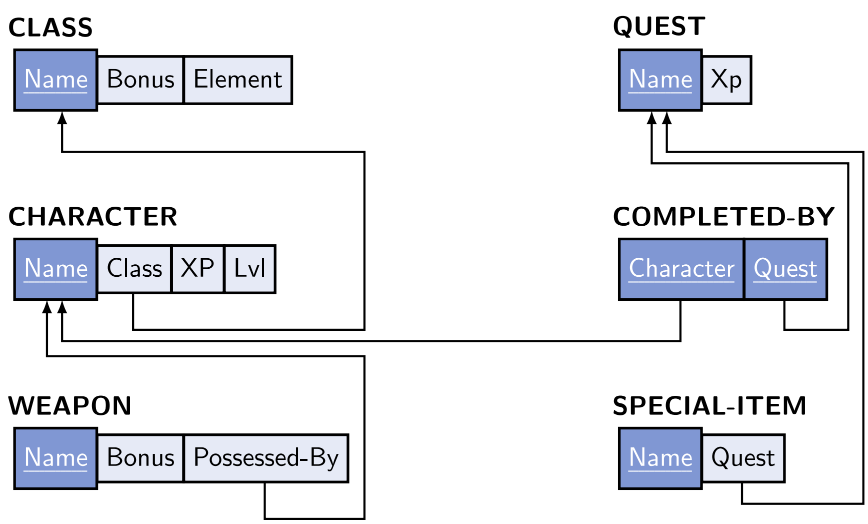
113 https://rocketgit.com/user/caubert/CSCI_3410/source/tree/branch/master/blob/notes/code/sql/HW_ComputerVariation.sql

```sql
SELECT ID
FROM VIDEO
WHERE TITLE LIKE "%video%";

-- or
SELECT ID
FROM VIDEO
WHERE Title REGEXP 'video';

-- … the number of thumbs up for the video with title "My
--                      vacations" ("1").
SELECT COUNT(*)
FROM THUMBS_UP,
  VIDEO
WHERE VIDEO.Title = "My vacations"
  AND VIDEO.ID = THUMBS_UP.Video;

-- … the title of the oldest video ("My first video!").
SELECT Title
FROM VIDEO
WHERE Released <= ALL (
    SELECT Released
    FROM VIDEO);

-- or
SELECT Title
FROM VIDEO
WHERE Released = (
    SELECT Min(Released)
    FROM VIDEO);

-- or even
SELECT Title
FROM VIDEO
ORDER BY Released ASC
LIMIT 1;

-- … the names of the accounts who gave a thumbs up to the
--                      video with id 30 ("Bob Ross").
SELECT Name
FROM ACCOUNT,
  THUMBS_UP
WHERE THUMBS_UP.Video = 30
  AND THUMBS_UP.Account = ACCOUNT.ID;

-- … the ID of the account with the greatest number of
--                      subscribers ("2").
SELECT Subscribed
FROM SUBSCRIBE
GROUP BY Subscribed
```

```sql
ORDER BY COUNT(Subscriber) DESC
LIMIT 1;

```

HW_SocialMedia.sql[114]

Solution to Problem 3.12 (Improving a role-playing game with a relational model)
The following solves all the issues with your friend's code design. As quests only rarely provide a special item, we added a relation to avoid having a `Special-item` in the `QUEST` table since that would be **`NULL`** too often.



Solution to Problem 3.13 (A simple database for books) **Pb 3.13 – Solution to Q. 1**
Here are possible ways of getting the required information:

1. The `Title` of all the books:

   ```sql
   SELECT Title FROM BOOK;
   ```

2. The *distinct* `Name` of the publishers.

   ```sql
   SELECT DISTINCT Name FROM PUBLISHER;
   ```

3. The `Titles` and `Published` dates of the books published since January 31, 2012.

   ```sql
   SELECT Title, Published FROM BOOK
   WHERE Published > DATE'20120131';
   ```

[114] https://rocketgit.com/user/caubert/CSCI_3410/source/tree/branch/master/blob/notes/code/sql/HW_SocialMedia.sql

4. The first and last names of the authors published by "Gallimard" (from any city).

```
SELECT FName, LName FROM AUTHOR, BOOK
WHERE PublisherName = "Gallimard"
    AND Author = ID;
```

5. The first and last names of the authors who were not published by an editor in "New-York".

```
SELECT FName, LName FROM AUTHOR, BOOK
WHERE NOT PublisherCity= "New-York"
    AND Author = ID;
```

6. The ID of the authors who published a book whose name starts with "Where".

```
SELECT Author FROM BOOK
WHERE Title LIKE 'Where%';
```

7. The total number of pages in the database.

```
SELECT SUM(Pages) FROM BOOK;
```

8. The number of pages in the longest book written by the author whose last name is "Wolve".

```
SELECT MAX(PAGES) FROM BOOK, AUTHOR
WHERE LName = "Wolve"
    AND Author = ID;
```

9. The title of the books published in the 19th century.

```
SELECT Title FROM BOOK
WHERE Published >= DATE'18010101'
    AND Published <= DATE'19001231';
```

Pb 3.13 – Solution to Q. 2 We can use the following command:

```
UPDATE BOOK SET Title = "BANNED"
WHERE Author = 3;
```

The pair (title, publication date) is the primary key in the BOOK table, so if the author whose ID is 3 has published more than one book at a particular date, then our update will be rejected, as applying it would result in violating the entity integrity constraint.

Pb 3.13 – Solution to Q. 3 To delete the required rows, we can use:

```
DELETE FROM BOOK WHERE Author = 3;
DELETE FROM AUTHOR WHERE ID = 3;
```

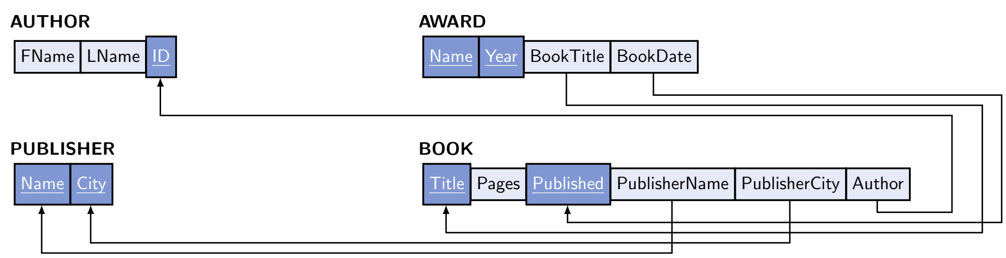
Note that trying to delete the rows in the AUTHOR table before deleting the rows in the BOOK table could cause a referential integrity violation, since the BOOK table has a foreign key assigned to the AUTHOR table's Id field.

Pb 3.13 – Solution to Q. 4 We could design that table as follows:

```
CREATE TABLE AWARD(
    Name VARCHAR(30),
    Year DATE,
    BookTitle VARCHAR(30),
    BookPubDate DATE,
    FOREIGN KEY (BookTitle, BookPubDate)
        REFERENCES BOOK(Title, Published),
    PRIMARY KEY (Name, Year)
);
```

Note that there is no need to store the name of the author in this relation: this information can be recovered by looking in the BOOK table for the name of the author of the awarded book.

Pb 3.13 – Solution to Q. 5 We obtain something as follows:



Note that having two attributes as the primary key makes the referencing of foreign keys more cumbersome.

Pb 3.13 – Solution to Q. 6 Two of the flaws that come to mind are:

1. The choice of the primary key for the BOOK relation: two books with the same title cannot be published on the same day, which is a serious limitation. Using a primary key like ISBN would be much more appropriate.
2. This design makes it impossibile to deal with books written by multiple authors or published by multiple publishers. We could address this by having two separate tables, IS_THE_AUTHOR_OF and PUBLISHED_BY, that "maps" the book's ISBN with author's or editor's primary key.

Solution to Problem 3.14 (A database for website certificates) The solution can be read from the following code:

```
/* code/sql/HW_Certificate.sql */
DROP SCHEMA IF EXISTS HW_Certificate;

CREATE SCHEMA HW_Certificate;

USE HW_Certificate;

```

```sql
/*
SN = Serial Number
CN = Common Name
CA = Certificate Authority
 */
CREATE TABLE ORGANIZATION (
  SN VARCHAR(30) PRIMARY KEY,
  CN VARCHAR(30)
);

CREATE TABLE CA (
  SN VARCHAR(30) PRIMARY KEY,
  CN VARCHAR(30),
  Trusted BOOL
);

CREATE TABLE CERTIFICATE (
  SN VARCHAR(30) PRIMARY KEY,
  CN VARCHAR(30) NOT NULL,
  Org VARCHAR(30) NOT NULL,
  Issuer VARCHAR(30),
  Valid_Since DATE,
  Valid_Until DATE,
  FOREIGN KEY (Org) REFERENCES ORGANIZATION (SN) ON DELETE CASCADE,
  FOREIGN KEY (Issuer) REFERENCES CA (SN)
);

INSERT INTO ORGANIZATION
VALUES (
  '01',
  'Wikimedia Foundation'),
(
  '02',
  'Free
    Software Foundation');

INSERT INTO CA
VALUES (
  'A',
  "Let's Encrypt",
  TRUE),
(
  'B',
  'Shady Corp.',
  FALSE),
(
  'C',
  'NewComer Ltd.',
```

```
  NULL);

INSERT INTO CERTIFICATE
VALUES (
  'a',
  '*.wikimedia.org',
  '01',
  'A',
  20180101,
  20200101),
(
  'b',
  '*.fsf.org',
  '02',
  'A',
  20180101,
  20191010),
(
  'c',
  '*.shadytest.org',
  '02',
  'B',
  20190101,
  20200101),
(
  'd',
  '*.wikipedia.org',
  '01',
  'C',
  20200101,
  20220101);

-- CN of all certificates.
SELECT CN
FROM CERTIFICATE;

-- (*.wikimedia.org | *.fsf.org | *.shadytest.org |
--                    *.wikipedia.org)
--                     The SN of the organizations whose CN
--       contains
--                    "Foundation"
SELECT SN
FROM ORGANIZATION
WHERE CN LIKE "%Foundation%";

-- (01 | 02)
--                     The CN and expiration date of all the
--           certificates
--                that
--                    expired (assuming we are the 6th of
```

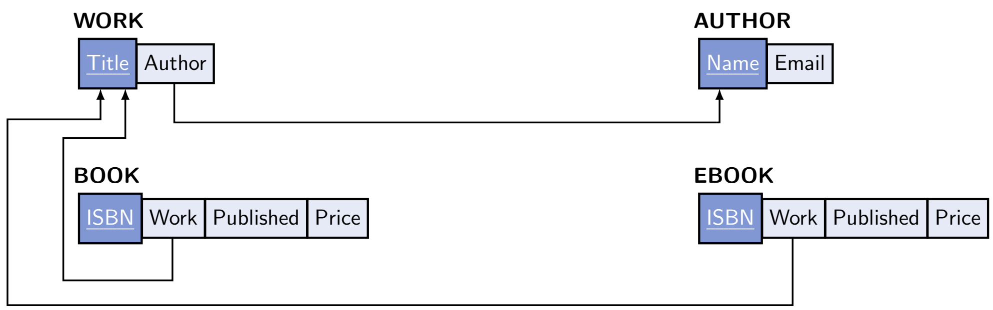
```
--    December
--            2019).
SELECT CN,
  Valid_Until
FROM CERTIFICATE
WHERE Valid_Until < DATE '20191206';

-- (*.fsf.org,  2019-10-10)
--                        The CN of the CA that are not trusted.
SELECT CN
FROM CA
WHERE Trusted IS NOT TRUE;

-- (Shady Corp. |  NewComer Ltd.)
--                        The CN of the certificates that are
--   signed
--       by
--         a
--            CA
--                 that
--                    is not trusted.
SELECT CERTIFICATE.CN
FROM CERTIFICATE,
  CA
WHERE Trusted IS NOT TRUE
  AND CA.SN = CERTIFICATE.Issuer;

-- (Shady Corp. | NewComer Ltd.)
--                        The number of certificates signed by
-- the
--    CA
--          whose
--             CN
--                is
--                   "Let's encrypt".
SELECT COUNT(CERTIFICATE.SN) AS "Number of certificates signed
    by Let's encrypt"
FROM CERTIFICATE,
  CA
WHERE CERTIFICATE.Issuer = CA.SN
  AND CA.CN = "Let's encrypt";

-- (2)
--                        A table listing the CN of the
--    organizations
--          along
--               with
--                   the CN of their certificates.
SELECT ORGANIZATION.CN AS Organization,
  CERTIFICATE.CN AS Certificate
```

```
FROM ORGANIZATION,
  CERTIFICATE
WHERE CERTIFICATE.Org = ORGANIZATION.SN;

-- ( Wikimedia Foundation,  *.wikimedia.org | Free Software
--                   Foundation, *.fsf.org | Free Software
--         Foundation
--           ,
--                   *.shadytest.org | Wikimedia Foundation ,
--                *.wikipedia.org
--                  )
/*
DELETE FROM CA WHERE SN = 'A';
ERROR 1451 (23000): Cannot delete or update a parent row: a foreign key constraint fails (`HW_Certificate`.`CERTIFICATE`, CONSTRAINT `CERTIFICATE_ibfk_2` FOREIGN KEY (`Issuer`) REFERENCES `CA` (`SN`))

=> Rejected, because an entry in CERTIFICATE references this tuple (referential integrity constraint).

UPDATE ORGANIZATION SET CN = "FSF" WHERE SN = '02';
Query OK, 1 row affected (0.008 sec)
Rows matched: 1  Changed: 1  Warnings: 0

=> Ok, change
('02', 'Free Software Foundation');
into
('02', 'FSF');
in ORGANIZATION

MariaDB [HW_Certificate]> UPDATE ORGANIZATION SET SN = "01" WHERE SN = '02';
ERROR 1451 (23000): Cannot delete or update a parent row: a foreign key constraint fails (`HW_Certificate`.`CERTIFICATE`, CONSTRAINT `CERTIFICATE_ibfk_1` FOREIGN KEY (`Org`) REFERENCES `ORGANIZATION` (`SN`) ON DELETE CASCADE)

=> Rejected, because an entry in CERTIFICATE references this tuple (referential integrity constraint).
This query would have been rejected even if this tuple was not referenced, since it would have violated the entity integrity constraint.

DELETE FROM ORGANIZATION;

=> Deletes all the content of organization and of certificate.
 */
```

HW_Certificate.sql[115]

Solution to Problem 3.15 (A simple database for published pieces of work) **Pb 3.15 – Solution to Q. 1**

The relational model for this code is:



Pb 3.15 – Solution to Q. 2

The solution to the next questions can be read from the following code:

```
/* code/sql/HW_Work.sql */
/*
 Determine if the following insertion statements would violate
 ↪  the the Entity integrity constraint,
 the Referential integrity constraint, if there would be some
 ↪  Other kind of error, or if it would
 result in uccessful insertion.
 */
START TRANSACTION;

-- We don't want to perform the actual insertions.
INSERT INTO EBOOK
VALUES (
  0,
  NULL,
  20180101,
  0);


/*
 Query OK, 1 row affected (0.003 sec)
 So, "Successful insertion".
 */
-- The following statement raises an error.
```

[115] https://rocketgit.com/user/caubert/CSCI_3410/source/tree/branch/master/blob/notes/code/sql/HW_Certificate.sql

```
-- INSERT INTO AUTHOR VALUES ("Mary B.", "mb@fai.fr", NULL);
/*
 ERROR 1136 (21S01): Column count doesn't match value count at row 1
 So, "Other kind of error".
 */
-- The following statement raises an error.
-- INSERT INTO WORK VALUES ("My Life", "Claude A.");
/*
 ERROR 1452 (23000): Cannot add or update a child row: a foreign key constraint fails
 (`HW_EXAM_1`.`WORK`, CONSTRAINT `WORK_ibfk_1` FOREIGN KEY (`Author`) REFERENCES `AUTHOR` (`Name`)
 ON DELETE CASCADE ON UPDATE CASCADE)
 So, "Referential integrity constraint"
 */
INSERT INTO BOOK
VALUES (
  00000000,
  NULL,
  DATE '20001225',
  90.9);


/*
 Query OK, 1 row affected (0.000 sec)
 So, "Successful insertion".
 */
-- The following statement raises an error.
--  INSERT INTO AUTHOR VALUES ("Virginia W.", "alt@isp.net");
/*
 ERROR 1062 (23000): Duplicate entry 'Virginia W.' for key 'PRIMARY'
 So, "Entity integrity constraint".
 */
ROLLBACK;

-- We go back to the previous state.
/*
 List the rows (i.e., A.2, W.1, etc.) modified by the following statements
 (be careful about the conditions on foreign keys!):
 */
START TRANSACTION;

-- We don't want to perform the following operations.
UPDATE
  AUTHOR
SET Email = 'Deprecated'
WHERE Email LIKE '%isp.net';
```

```sql

/*
 Query OK, 2 rows affected (0.010 sec)
 Rows matched: 2  Changed: 2  Warnings: 0
 This changed A.1 and A.2
 */
UPDATE
  WORK
SET Title = "How to eat"
WHERE Title = "What to eat";


/*
 Rows matched: 1  Changed: 1  Warnings: 0
 SQL returns only the number of row changed in the WORK table,
 but other rows have been changed as well.
 This changed W.1, B.1, E.1.
 */
-- The following statement raises an error.
-- DELETE FROM WORK;
/*
 ERROR 1451 (23000): Cannot delete or update a parent row: a foreign key constraint fails
 (`HW_EXAM_1`.`BOOK`, CONSTRAINT `BOOK_ibfk_1` FOREIGN KEY (`Work`) REFERENCES `WORK` (`Title`) ON UPDATE CASCADE)
 Does not change any row.
 */
-- The following statement raises an error.
--  DELETE FROM AUTHOR WHERE Name = "Virginia W.";
/*
 ERROR 1451 (23000): Cannot delete or update a parent row: a foreign key constraint fails
 (`HW_EXAM_1`.`BOOK`, CONSTRAINT `BOOK_ibfk_1` FOREIGN KEY (`Work`) REFERENCES `WORK` (`Title`) ON UPDATE CASCADE)
 Does not change any row.
 */
ROLLBACK;

-- We go back to the previous state.
-- You can now assume that there is more data than
-- what we inserted, if that helps you. Write a
-- command that selects …

--  We insert some dummy values for this
--  next part.
INSERT INTO WORK
VALUES (
  "My Life",
  "Paul B."),
(
```

```
  "What to eat, 2",
  "Virginia W.");

INSERT INTO BOOK
VALUES (
  15355627,
  "My Life",
  DATE '20180219',
  15.00),
(
  12912912,
  "What to eat, 2",
  DATE '20200101',
  13);

INSERT INTO EBOOK
VALUES (
  15150628,
  "My Life",
  DATE '20190215',
  10.89),
(
  42912912,
  "What to eat, 2",
  DATE '20200115',
  12);

-- … the price of all the ebooks.
SELECT Price
FROM EBOOK;

-- … the (distinct) names of the authors who have authored
-- a piece of work.
SELECT DISTINCT Author
FROM WORK;

-- … the name of the authors using fai.fr for their email.
SELECT Name
FROM AUTHOR
WHERE Email LIKE '%fai.fr';

-- … the price of the ebooks published after 2018.
SELECT Price
FROM BOOK
WHERE Published >= 20180101;


/*
 Note that
 SELECT Price FROM BOOK WHERE Published > 2018;
```

```
 would return all the prices, along with a warning:
 Incorrect datetime value: '2018'
 */
-- … the price of the most expensive book.
SELECT MAX(Price)
FROM BOOK;

-- … the number of pieces of work written by the author
-- whose name is “Virginia W.”.
SELECT COUNT(*)
FROM WORK
WHERE WORK.Author = "Virginia W.";

-- … the email of the author who wrote the piece of work
-- called “My Life”.
SELECT Email
FROM AUTHOR,
  WORK
WHERE WORK.Title = "My Life"
  AND WORK.Author = AUTHOR.Name;

-- the isbn(s) of the book containing a work written by the
-- author whose email is "vw@isp.net".
SELECT ISBN
FROM BOOK,
  WORK,
  AUTHOR
WHERE AUTHOR.Email = "vw@isp.net"
  AND WORK.Author = AUTHOR.Name
  AND BOOK.Work = WORK.Title;


/*
 Write a command that updates the title of all the pieces of work
 ↪  written by the author whose name is “Virginia W. to”BANNED".
 Is there any reason for this command to be rejected by the
 ↪  system? If yes, explain which one.
 */
-- The following statement raises an error.
/*
UPDATE
 WORK
SET
 Title = "BANNED"
WHERE
 Author = "Virginia W.";
 */
/*
```

```
 Gives an error, since "Title" is the primary key in the WORK table, and Virginia W. has authored two pieces of work or more,
 they are both given the title "BANNED", which violates the unicity of value in primary keys.
 */
-- Write one or multiple commands that would delete the work
-- whose title is “My Life”, as well as
--    all
--       of
--           the
--                 books
-- and ebooks versions of it.
--  The following statement raises an
-- error.
-- DELETE FROM WORK
-- WHERE Title = "My Life";
/*
 Fails
 ERROR 1451 (23000): Cannot delete or update a parent row: a foreign key constraint fails
 (`HW_EXAM_1`.`BOOK`, CONSTRAINT `BOOK_ibfk_1` FOREIGN KEY (`Work`) REFERENCES `WORK` (`Title`) ON UPDATE CASCADE)
 */
-- We have to first delete the corresponding publications:
DELETE FROM BOOK
WHERE WORK = "My Life";

DELETE FROM EBOOK
WHERE WORK = "My Life";

-- And then we can delete the work:
DELETE FROM WORK
WHERE Title = "My Life";


/*
 And, no, we cannot delete "simply" from multiple tables in one command.
 Some workaround exists, cf. https://stackoverflow.com/q/1233451/ .
 */
```

HW_Work.sql[116]

Pb 3.15 – Solution to Q. 3

Finally, to answer the last question, here is a list of the possible limitations:

[116] https://rocketgit.com/user/caubert/CSCI_3410/source/tree/branch/master/blob/notes/code/sql/HW_Work.sql

1. Having the name or the title as a primary key (in the `AUTHOR` and `WORK` tables) is not a good idea: we cannot have two authors with the same name or two pieces of work with the same title!
2. If all the attributes in the `BOOK` and the `EBOOK` tables are going to be the same, then we should probably have only one table called `PUBLICATION` with a boolean to indicate whenever the publication is digital or on paper.
3. Having a mix of **`ON DELETE CASCADE`** and **`ON DELETE RESTRICT`** is not really justified and makes the tables harder to use. We should have used the same update policy on both tables.

Solution to Problem 3.16 (A simple database for authors of textbooks) The answers can be found in the following snippet:

```sql
/*
 code/sql/HW_TEXTBOOK_AUTHORED_SOL.sql
 */
/*
 EXERCISE 1

 Write a command that updates the email address of 'Gaddis',
 ↪ 'Tony' to "tgaddis@pearson.com"
 */
UPDATE
  AUTHOR
SET Email = "tgaddis@pearson.com"
WHERE LName = 'Gaddis'
  AND FName = 'Tony';


/*
 You can use
 SELECT * FROM AUTHOR;
 to check that the modification took place.
 */
/*
 EXERCISE 2

 Write a command that inserts the textbook of your choice in the
 TEXTBOOK table. No value should be NULL, but you can invent
 the values.
 */
INSERT INTO TEXTBOOK
VALUES (
  'Fundamentals of Database Systems',
  9780133970777,
  165.89);
```

```sql
/*
 You can use
 SELECT * FROM TEXTBOOK;
 to check that the insertion was correctly made.
 */
/*
 EXERCISE 3

 Write a command that makes 'Gaddis', 'Tony' the author of the
 textbook you just added to our database.
 */
INSERT INTO AUTHORED
VALUES (
  9780133970777,
  'Gaddis',
  'Tony');


/*
 You can use
 SELECT * FROM AUTHORED;
 to check that the insertion was correctly made.


 EXERCISE 4

 Write a command that makes "0.01" becomes the
 default value for the Price attribute of the
 TEXTBOOK relation.
 */
ALTER TABLE TEXTBOOK
  ALTER COLUMN Price SET DEFAULT 0.01;


/*
 You can use
 DESCRIBE TEXTBOOK;
 to check that the Price attribute now has a default
 value.


 EXERCISE 5

 Write a command that insert a textbook of
 your choice in the TEXTBOOK table, with the
 price set to the default value.
 */
INSERT INTO TEXTBOOK
VALUES (
  'Proof Theory',
```

```sql
  9780486490731,
  DEFAULT);


/*
 You can use
 SELECT * FROM TEXTBOOK;
 to check that the insertion was correctly made.


 EXERCISE 6

 Write a command that creates a table called EDITOR
 with 3 attributes, "Name", "Address" and "Website".
 The "Name" attribute should be the primary key.
 Then, insert two tuples in the EDITOR table, one
 should have the "Name" attribute set to "Pearson".
 */
CREATE TABLE EDITOR (
  NAME VARCHAR(30) PRIMARY KEY,
  Address VARCHAR(255),
  Website VARCHAR(100)
);

INSERT INTO EDITOR
VALUES (
  'Pearson',
  NULL,
  'http://pearsoned.com/'),
(
  'Dover',
  NULL,
  'https://store.doverpublications.com/');


/*
 You can use
 DESCRIBE EDITOR;
 to check that the table was actually created, and
 SELECT * FROM EDITOR;
 to check that the values were inserted.


 EXERCISE 7

 Write a command that creates a table called PUBLISHED
 with 2 attributes, "Editor", and "Textbook".
 The "Editor" attribute should references the EDITOR
 table, and the "Textbook" attribute should reference
 the TEXTBOOK table.
```

```
 */
CREATE TABLE PUBLISHED (
  Editor VARCHAR(30),
  FOREIGN KEY (Editor) REFERENCES EDITOR (NAME),
  Textbook CHAR(13),
  FOREIGN KEY (Textbook) REFERENCES TEXTBOOK (ISBN)
);


/*
 You can use
 DESCRIBE PUBLISHED;
 to check that the table was actually created.

 EXERCISE 8

 Write a command that makes "Pearson" the editor of
 the textbook whose ISBN is 9780133776744.
 */
INSERT INTO PUBLISHED
VALUES (
  "Pearson",
  9780133776744);


/*
 You can use
 SELECT * FROM PUBLISHED;
 to check that the table was actually created.


 EXERCISE 9

 Answer the following short questions. In our model, as it is, …

 Can an author have authored more than one textbook?
 Yes.

 Can a textbook have more than one author?
 Yes.

 Can a textbook without ISBN be inserted in the TEXTBOOK relation?
 No, unless you create a "dummy" (fake) value for it,
 like 0000000000000, but this value can be used only
 once, since ISBN is the primary key.

 Can the price of a textbook be negative?
 Yes. We can actually test it:
 INSERT INTO TEXTBOOK VALUES ("Test", 0000000000000, -1);
```

```
 Can two author have the same first and last name?
 No. The query:
 INSERT INTO AUTHOR VALUES ('Smith', 'Bob', NULL), ('Smith',
 ↪ 'Bob', NULL);
 returns
 ERROR 1062 (23000): Duplicate entry 'Smith-Bob' for key
 ↪ 'PRIMARY'

 Can two textbooks have the same title?
 Yes, as long as they have different ISBN. The command
 INSERT INTO TEXTBOOK VALUES ("Test", 0000000000001, NULL),
 ↪ ("Test", 0000000000002, NULL);
 is processed just fine.

 Can two editiors have the same address?
 Yes. The command:
 INSERT INTO EDITOR VALUES ("Test 1", "123 Main St.", NULL),
 ↪ ("Test 2", "123 Main St.", NULL);
 is processed just fine.

```

HW_TextbookAuthoredSol.sql[117]

Solution to Problem 3.17 (A simple database for capstone projects) The answers can be found in the following snippet:

```
/*
code/sql/HW_CapstoneSol.sql
 */
/*

I. Short Questions (6 pts)

Answer the following short questions based on the model
 ↪ implemented above.
You can simply answer "True" or "False", or justify your
 ↪ reasoning (e.g. with code).
 */
-- 1. Can a project uses multiple programming languages?
--                Yes.
--       2. Can a student be the leader of multiple
--          projects?
--                Yes.
--       3. Can multiple projects have the same code name?
--                Yes.
```

[117] https://rocketgit.com/user/caubert/CSCI_3410/source/tree/branch/master/blob/notes/code/sql/HW_TextbookAuthoredSol.sql

```
--        4. Could Claude simply enter NULL for the value
--           of his pproject's code name, since he's undecided?
--        No.
--        5. Can a project be created without project
--           leader?
--        No.
--       6. Can we know who is working on a project
--          without being its leader?
--        No.
/*

II. Relational Model (6 pts.)

Draw the relational model corresponding to this code.
You can hand-draw it and join a scan or a picture, or simply hand me back the sheet where you drew it.
 */
/*

III. Simple Commands (8 pts.)

Below, you are asked to write commands that perform various actions.
Please, leave them uncommented, unless you can't write them correctly, in which case it's ok to leave them commented.
The first question is answered as an example.
 */
-- 0. Write a command that list all the names of the
--                programming languages.
SELECT Name
FROM PROGRAMMING_LANGUAGE;

-- 1. Write a command that insert a new student in the
--                STUDENT table.
--                 (You should invent the values).
INSERT INTO STUDENT
VALUES (
  "Bob",
  "0987654321234",
  NULL,
  NULL);

-- 2. Write a command that updates the code name of the
--           project ("Undecided", "9999999999999") to "VR in
--           ER".
UPDATE
  PROJECT
SET CodeName = "VR in ER"
WHERE CodeName = "Undecided"
  AND Leader = "9999999999999";
```

```sql

-- 3. Write a command that updates the graduation year of the
--           student whose id is "0987654321098" to 2024, and
--           the semester to "Fall".
UPDATE
  STUDENT
SET GraduationYear = 2024,
  GraduationSemester = "Fall"
WHERE id = "0987654321098";

-- 4. Write a command that changes the STUDENT table to make
--           it impossible to enter NULL for the first name of
--           a student, without changing the primary key.
ALTER TABLE STUDENT MODIFY FName VARCHAR(50) NOT NULL;

-- 5. Write a command that changes the datatype of
--           GraduationYear to SMALLINT.
ALTER TABLE STUDENT MODIFY GraduationYear SMALLINT;

-- 6. Write a command that adds an attribute "ReleaseDate" to
--           the PROJECT table.
ALTER TABLE PROJECT
  ADD COLUMN ReleaseDate DATE;

-- 6.bis If you managed to write the previous command
--          correctly, write a command that sets the release
--          date of the project ("Brick Break",
--   "0123456789100")
--      to
--          the 26th of November 2022.
UPDATE
  PROJECT
SET ReleaseDate = DATE "20221126"
WHERE CodeName = "Brick Break"
  AND Leader = "0123456789100";

-- 7. Write a command that makes it impossible for a student
--           to be the leader in more than one project
--           (This command should return an error)
--           ALTER TABLE PROJECT ADD UNIQUE (Leader);
```

HW_CapstoneSol.sql[118]

Solution to Problem 3.18 (A simple database for vaccines) The answers can be found in the following snippet:

[118] https://rocketgit.com/user/caubert/CSCI_3410/source/tree/branch/master/blob/notes/code/sql/HW_CapstoneSol.sql

```
/* code/sql/HW_VaccineSol.sql */
/*

I. Short Questions (3 pts.)

Answer the following short questions. In our implementation…

1. … can two companies have exactly the same name?

No, as COMPANY.Name is the only attribute in the primary key of COMPANY.

2. … can two companies have the same website?

Yes, nothing prevents it.

3. … can a company not have a website?

Yes, the domain of COMPANY.Website is "VARCHAR(255)", without a constraint preventing it from being "NULL".

4. … can the same vaccine be manufactured by multiple companies?

No, as VACCINE.Manufacturer is an attribute in VACCINE that accepts only one value.

5. … can a vaccine not have a manufacturer?

No, as VACCINE.Manufacturer bears the "NOT NULL" constraint.

6. … can a disease being neither communicable nor not communicable?

Yes, as DISEASE.Communicable is of type "BOOL", it accepts the "NULL" value.

7. … can the same vaccine have different efficacies for different diseases?

Yes, the EFFICACY table has for primary key VaccineName and DiseaseName, which implies that the same vaccine can occur repeatedly as long as it is associated with different diseases.
 */
/*

II. Longer Questions (6 pts.)

Answer the following questions:

```

```
1. What does `CHECK (Website LIKE "https://*")` do?

It refrains any value not starting with  "https://" to be
↪  inserted as a value for the COMPANY.Website attribute.
Note that in particular it forbids a website from not being
↪  secured (that is, http:// is not a valid protocol).

2. Why did we picked the `DECIMAl(5,2)` datatype?

It is the appropriate datatype to represent percentage values
↪  represented as ranging from 100.00 to 0.00.
The discussion at https://stackoverflow.com/a/2762376/ also
↪  highlights that percent can be represented as decimal(5,4)
↪  with a check to insure that the value will range between
↪  1.0000 and 0.0000.

3. What is the benefit / are the benefits of having a separate
↪  EFFICACY table over having something like

CREATE TABLE VACCINE(
 Name VARCHAR(50) PRIMARY KEY,
 Manufacturer VARCHAR(50),
 Disease VARCHAR(50),
 Efficacy DECIMAl(5,2),
 FOREIGN KEY (Manufacturer) REFERENCES COMPANY (Name)
);

?

This implementation does not allow to record that the same
↪  vaccine can have different efficacies for different
↪  diseases.
Stated differently, it forbids to represent vaccines efficient
↪  against multiple diseases faitfully.
 */
/*

III. Relational Model (6 pts.)

Draw the relational model corresponding to this code.
You can hand-draw it and join a scan or a picture, or simply
↪  hand me back a sheet.
 */
/*

IV. Simple Commands (5 pts.)

Below, you are asked to write commands that perform various
↪  actions.
Please, leave them uncommented, unless
```

```
 - you can not write them correctly, but want to share your attempt,
 - it is specified that it should return an error.

The first question is answered as an example.
 */
-- 0. Write a command that list the names of
--        all the diseases.
SELECT Name
FROM DISEASE;

-- 1. Write a command that insert "Pfizer" in the
--         COMPANY table (you can make up the website or look
--    it)
INSERT INTO COMPANY
VALUES (
  "Pfizer",
  "https://www.pfizer.com/");

--  2. Write a command that insert the "Pfizer-BioNTech
--          COVID-19 Vaccine" in the VACCINE table, and a
--    command
--          that store the efficacy of that vaccine against
--          the "Coronavirus disease 2019" disease
--         ( you can make up the values or look them up).
INSERT INTO VACCINE
VALUES (
  "Pfizer-BioNTech COVID-19 Vaccine",
  "Pfizer");

INSERT INTO EFFICACY
VALUES (
  "Coronavirus disease 2019",
  "Pfizer-BioNTech COVID-19 Vaccine",
  89);

--  3. Write a command that updates the name of the
--          company "Moderna" to "Moderna, Inc." everywhere.
UPDATE
  COMPANY
SET Name = "Moderna, Inc."
WHERE Name = "Moderna";

--  4. Write a command that lists the name of all the
--          companies.
SELECT Name
FROM COMPANY;

--  5. Write a command that deletes the "Coronavirus disease
--          2019" entry from the DISEASE table (if only!).
```

```
/*
DELETE FROM DISEASE
WHERE Name = "Coronavirus disease 2019";
 */
--  This command should return an error. Explain it and leave
--       the command commented.
--      The "Coronavirus disease 2019" value in DISEASE.Name
-- is
--     refereed to by two entries in the EFFICACY table.
--      As the foreign key from EFFICACY.DiseaseName to
--     DISEASE.Name does not specify its policy "ON DELETE",
--   its
--     default behavior is to restrict deletion, causing the
--     error.
--        6. Write two commands: one that adds "physiological"
--    to
--           the possible types of diseases, and one that
--    inserts
--           a physiological disease in the DISEASE table.
ALTER TABLE DISEASE MODIFY TYPE ENUM ("infectious",
  "deficiency", "hereditary", "physiological");

INSERT INTO DISEASE
VALUES (
  "Asthma",
  FALSE,
  "physiological");

--   7 (difficult). Write a command that return the list of
--                      all the companies that manufacture a
--                      vaccine against "Coronavirus disease
--     2019".
SELECT VACCINE.Manufacturer
FROM VACCINE,
  EFFICACY
WHERE VACCINE.Name = EFFICACY.VaccineName
  AND EFFICACY.DiseaseName = "Coronavirus disease 2019";
```

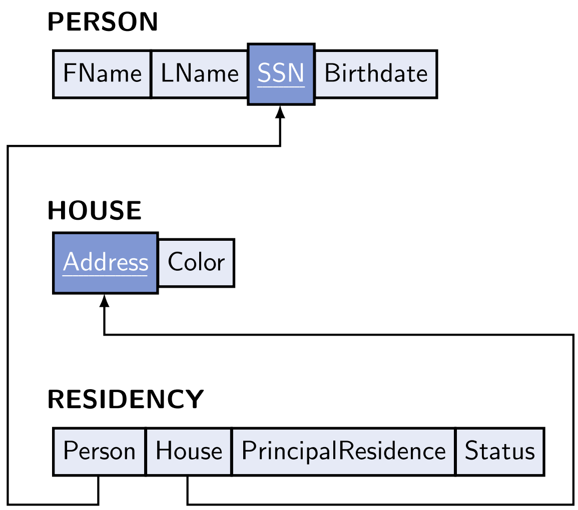
HW_VaccineSol.sql[119]

Solution to Problem 3.19 (A database for residencies) The file code/sql/HW_ResidencySol.sql[120] contains the solution to the code part of this problem.

Pb 3.19 – Solution to Q. 1 The relational model is:

[119] https://rocketgit.com/user/caubert/CSCI_3410/source/tree/branch/master/blob/notes/code/sql/HW_VaccineSol.sql

[120] https://rocketgit.com/user/caubert/CSCI_3410/source/tree/branch/master/blob/notes/code/sql/HW_ResidencySol.sql



Pb 3.19 – Solution to Q. 2

To violate the entity integrity constraint, it suffices to insert a tuple with **NULL** as a value for one of the attributes of a primary key or to insert a value that was already inserted.

Two examples are:

```
INSERT INTO PERSON VALUES ("Bob", "Ross", NULL,
 ↪  DATE"1942-10-29");
```

which would return `ERROR 1048 (23000): Column 'SSN' cannot be null`.

```
INSERT INTO HOUSE VALUES ("123 Main St.", "green");
```

which would return `ERROR 1062 (23000): Duplicate entry '123 Main St.' for key '`

Pb 3.19 – Solution to Q. 3

To violate the referential integrity constraint, it suffices to insert a tuple where the value for one of the attributes of a foreign key does not exist in the referenced table.

For instance,

```
INSERT INTO RESIDENCY VALUES ("999-99-9999", NULL, NULL,
 ↪  NULL);
```

would return

```
ERROR 1452 (23000): Cannot add or update a child row: a
 ↪  foreign key constraint fails
 ↪  (`HW_Residency_SOL`.`RESIDENCY`, CONSTRAINT
 ↪  `RESIDENCY_ibfk_1` FOREIGN KEY (`Person`) REFERENCES
 ↪  `PERSON` (`SSN`))
```

Since there is no row in the `PERSON` table with the value `"999-99-9999"` for `SSN`.

Pb 3.19 – Solution to Q. 4

The answers can be found in the following snippet:

```
/*
 In the following we use transactions
 to be able to simulate the "what if"
 aspect of the questions: we will not
 commit the changes we are testing,
 and roll back on them before moving to
 the next question.
 */
-- Exercise 4
--                    List the rows (i.e., P.2, H.1, or even
--           “none”)
--                   modified by the following statements:
START TRANSACTION;

UPDATE
  HOUSE
SET COLOR = "green";

-- H.1, H.2 and H.3
ROLLBACK;

START TRANSACTION;

DELETE FROM RESIDENCY
WHERE House LIKE "1%";

-- R.1,and R.3
ROLLBACK;

START TRANSACTION;

DELETE FROM HOUSE
WHERE Address = "456 Second St.";

-- H.2, R.2 and R.4 (because of the foreign key).
ROLLBACK;

START TRANSACTION;

-- Commented, because it causes an error.
--                    DELETE FROM PERSON
--                    WHERE Birthdate = DATE "1990-02-11";
--                    None, because of the foreign key and
-- the
--             referential
--                   integrity constraint.
--                    ERROR 1451 (23000): Cannot delete or
--    update
```

```
--       a
--              parent
--                   row:
--                    a foreign key constraint fails
--                    (`HW_RESIDENCY_SOL`.`RESIDENCY`,
--   CONSTRAINT
--                    `RESIDENCY_ibfk_1` FOREIGN KEY
-- (`Person`)
--             REFERENCES
--                    `PERSON` (`SSN`))
ROLLBACK;

```

HW_ResidencySol.sql[121]

Pb 3.19 – Solution to Q. 5

The answers can be found in the following snippet:

```
-- Exercise 5
/* Write a query that selects …
 … the addresses of the houses in the system (11 Third St., 123
 ↪ Main St., 456 Second St.).
 */
SELECT Address
FROM HOUSE;

-- … the SSN of the persons whose first name was not
--                    entered in the system (000-00-0000).
SELECT SSN
FROM PERSON
WHERE FName IS NULL;

-- … all the different colors of houses (white, blue).
SELECT DISTINCT COLOR
FROM HOUSE;

-- … the address of the residency of James Baldwin (123
--                    Main St.).
SELECT House
FROM RESIDENCY,
  PERSON
WHERE PERSON.Fname = "James"
  AND PERSON.LName = "Baldwin"
  AND PERSON.SSN = RESIDENCY.Person;

-- … the first name of the oldest person in the database
--                    (James).
SELECT FName
```

[121] https://rocketgit.com/user/caubert/CSCI_3410/source/tree/branch/master/blob/notes/code/sql/HW_ResidencySol.sql

```
FROM PERSON
WHERE Birthdate = (
    SELECT MIN(Birthdate)
    FROM PERSON
    WHERE Birthdate IS NOT NULL);

-- … Michael Keal’s principal residency address (123 Main
--                  St.).
SELECT RESIDENCY.House
FROM RESIDENCY,
  PERSON
WHERE PERSON.FName = "Michael"
  AND PERSON.LName = "Keal"
  AND PERSON.SSN = RESIDENCY.Person
  AND RESIDENCY.PrincipalResidence = TRUE;

-- … the (distinct) first and last names of the homeowners
--                  (Michael Keal, Mridula Warrier).
SELECT DISTINCT (PERSON.FName),
  PERSON.LName
FROM PERSON,
  RESIDENCY
WHERE RESIDENCY.Status = "own"
  AND RESIDENCY.Person = PERSON.SSN;

-- cf comment at snippet homonyms
SELECT PERSON.FName,
  PERSON.LName
FROM PERSON
WHERE SSN IN ( SELECT DISTINCT (RESIDENCY.Person)
    FROM RESIDENCY
    WHERE RESIDENCY.Status = "own");

-- … the SSN of the persons that have the same principal
--                  residency as James Baldwin
-- (000-00-0001).
SELECT RoomMate.Person
FROM RESIDENCY AS James,
  RESIDENCY AS RoomMate,
  PERSON
WHERE PERSON.FName = "James"
  AND PERSON.LName = "Baldwin"
  AND PERSON.SSN = James.Person
  AND James.House = RoomMate.House
  AND NOT James.Person = RoomMate.Person
  AND RoomMate.PrincipalResidence = TRUE;
```

HW_ResidencySol.sql[122]

[122] https://rocketgit.com/user/caubert/CSCI_3410/source/tree/branch/master/blob/notes/code/sql/HW_ResidencySol.s

Note that the query that returns the name of the homeowners can be improved.

```sql
--                    If we have homonymns in our database,
--   e.g.
INSERT INTO PERSON
VALUES (
  "A",
  "B",
  "000-00-0010",
  NULL),
(
  "A",
  "B",
  "000-00-0011",
  NULL);

INSERT INTO HOUSE
VALUES (
  "H",
  NULL);

-- H.3
INSERT INTO RESIDENCY
VALUES (
  "000-00-0010",
  "H",
  TRUE,
  "own"),
(
  "000-00-0011",
  "H",
  TRUE,
  "own");

-- Then the query below fails, in the sense that it reports
--                     the name "A, B" only once.
SELECT DISTINCT (PERSON.FName),
  PERSON.LName
FROM PERSON,
  RESIDENCY
WHERE RESIDENCY.Status = "own"
  AND RESIDENCY.Person = PERSON.SSN;

-- A better (and not much more complicated) solution would
--                     have been
SELECT PERSON.FName,
  PERSON.LName
FROM PERSON
WHERE SSN IN ( SELECT DISTINCT (RESIDENCY.Person)
```

ql

```
    FROM RESIDENCY
    WHERE RESIDENCY.Status = "own");

```

HW_ResidencySol.sql[123]

Pb 3.19 – Solution to Q. 6

To update the `SSN` of `"James Baldwin"` to `"000-00-0010"`, we could use:

```
UPDATE PERSON SET SSN = "000-00-0010" WHERE FName = "James" AND
    LName = "Baldwin";
```

However, this command would be rejected because of the foreign key constraint. On **UPDATE**, the foreign key from `RESIDENCY.Person` to `PERSON.SSN` restricts by default. The error would be:

```
ERROR 1451 (23000) at line 75: Cannot delete or update a parent
    row: a foreign key constraint fails
    (`HW_Residency_SOL`.`RESIDENCY`, CONSTRAINT
    `RESIDENCY_ibfk_1` FOREIGN KEY (`Person`) REFERENCES
    `PERSON` (`SSN`))
```

Pb 3.19 – Solution to Q. 7

In our model, as it is currently,

1. It is **possible** for two people to have the same last name.
2. It is **possible** for a person to have multiple principal residencies.
3. It is **not possible** for a house to not be yellow.
4. It is **possible** for the SSN to be any series of 11 characters.
5. It is **possible** for a person to own any number of houses.
6. It is **possible** for a person to rent any number of houses.

Pb 3.19 – Solution to Q. 8

Considering the given state for the `RESIDENCY` table, the following two are possible primary keys:

1. `Person` and `PrincipalResidence`
2. `Person` and `House`

Pb 3.19 – Solution to Q. 9

The first key would not accomodate a person with multiple secondary residencies, which is not a good thing. The second key could make sense, since it would refrain a person from declaring the same address twice as their residency. The only case that could be hard to work around is if a person was trying to own multiple units at the same address; however, this is more an issue with the primary key of `HOUSE` than an issue with the primary key we suggested for `RESIDENCY`.

---

[123] https://rocketgit.com/user/caubert/CSCI_3410/source/tree/branch/master/blob/notes/code/sql/HW_ResidencySol.sql

Solution to Problem 3.20 (A database for research fundings) (Some of) the answers can be found in the following snippet:

```
/* code/sql/HW_ScientificResearchSol.sql */
-- List the rows affected (updated or deleted) by the
--           following commands.
--            If no rows are affected because the command would
--        would
--           violate the entity integrity constraint, the
--        referential
--           integrity constraint, or if there would be some
--     other
--          kind
--           of error, please indicate it.
START TRANSACTION;


/*
UPDATE
 SCIENTIST
SET SSN = "000000001"
WHERE Name = "Claire";
 */
-- ERROR 1062 (23000) at line 106: Duplicate entry '1'
-- for
--           key 'PRIMARY'
ROLLBACK;

START TRANSACTION;

UPDATE
  FUNDINGAGENCY
SET Name = "NSF"
WHERE Name = "National Science Foundation";

SELECT *
FROM FUNDINGAGENCY;

-- FA. 1
SELECT *
FROM FUNDS;

-- F.1
ROLLBACK;

START TRANSACTION;


/*
DELETE FROM FUNDINGAGENCY
```

```
WHERE Name = "French-American Cultural Exchange";
 */
-- ERROR 1451 (23000): Cannot delete or update a parent row:
--          a foreign key constraint fails
--          (`HW_SCIENTIFIC_RESEARCH`.`FUNDS`, CONSTRAINT
--          `FUNDS_ibfk_1` FOREIGN KEY (`Agency`) REFERENCES
--          `FUNDINGAGENCY` (`Name`) ON UPDATE CASCADE)
ROLLBACK;

-- List the name of the funding agencies created after 2000
--          ("French-American Cultural Exchange")
SELECT Name
FROM FUNDINGAGENCY
WHERE Creation >= 2000;

-- List the code of the projects that contains the word
--          "Airplanes" ("AA", "BA")
SELECT CODE
FROM PROJECT
WHERE Name LIKE ("%Airplanes%");

-- List the number of hours scientists contributed to the
--          project "AA" (18)
SELECT SUM(Hours)
FROM CONTRIBUTESTO
WHERE Project = "AA";

-- List the code of the projects to which the scientist named
--          Sabine contributed ("AA", "BB")
SELECT Project
FROM CONTRIBUTESTO,
  SCIENTIST
WHERE SCIENTIST.Name = "Sabine"
  AND SCIENTIST.SSN = CONTRIBUTESTO.Scientist;

-- Give the name of the projects who benefited from federal
--          funds ("Advancing Airplanes")
SELECT PROJECT.Name
FROM PROJECT,
  FUNDS,
  FUNDINGAGENCY
WHERE FUNDINGAGENCY.Type = "Federal"
  AND FUNDINGAGENCY.Name = FUNDS.Agency
  AND FUNDS.Project = PROJECT.Code;

-- Give the name of the scientist who contributed to the same
--          project as Mike ("Sabine", "James")
SELECT DISTINCT (Fellow.Name) AS "Mike's fellow"
FROM SCIENTIST AS Mike,
  SCIENTIST AS Fellow,
```

```sql
  CONTRIBUTESTO AS A,
  CONTRIBUTESTO AS B
WHERE Mike.Name = "Mike"
  AND Mike.SSN = A.Scientist
  AND A.Project = B.Project
  AND B.Scientist = Fellow.SSN
  AND NOT Fellow.Name = "Mike";

-- List the name of the projects that are not funded by an
--          agency ("Better Airplanes", "Better Buildings")
SELECT DISTINCT (PROJECT.Name)
FROM PROJECT,
  FUNDS
WHERE NOT PROJECT.Code IN (
    SELECT FUNDS.Project
    FROM FUNDS);

-- Give the name of the scientist who contributed the most
--          (in terms of hours) to the project named
-- "Advancing
--          Airplanes" (Sabine)
SELECT SCIENTIST.Name
FROM SCIENTIST,
  CONTRIBUTESTO
WHERE CONTRIBUTESTO.Hours >= (
    SELECT MAX(Hours)
    FROM CONTRIBUTESTO,
      PROJECT
    WHERE PROJECT.Name = "Advancing Airplanes"
      AND PROJECT.Code = CONTRIBUTESTO.Project)
  AND CONTRIBUTESTO.Scientist = SCIENTIST.SSN;
```

HW_ScientificResearchSol.sql[124]

Solution to Problem 3.21 (Improving a Relational Model for a Printing Station) Pb 3.21 – Solution to (

- Instead of making the nickname attribute being the primary key, they could have it as a non-prime attribute, and use some `id` with auto-increment as a primary key for the computers, rooms and phones without having to come up with new names all the times.
- They should simply remove that attribute, and write a **`SELECT`** query that returns this information whenever they need it.
- The best way to address this issue is probably to have a separate table for operating systems, with the attributes they are interested in (architecture, manufacturer of the OS, etc.), and foreign keys from Computer and Phone to it.

[124] https://rocketgit.com/user/caubert/CSCI_3410/source/tree/branch/master/blob/notes/code/sql/HW_ScientificResearchSol.sql

- Making both the `Nickname` and the `ConnectedTo` attributes be the primary key would solve their issue, but could potentiall introduce a lot of reduncdancy. The best way is probably to have a separate table that list the connections. Since computers and phones are in two different tables, this creates an additional challenge, since we would need to have a "connection table for computers", and a "connection table for phones". We recommend actually merging those two tables into one, that would additionaly have an attribute to set if the device is a phone or a computer.

Pb 3.21 – Solution to Q. 2 A possible solution would consist in

- Have a `OS` relation with attributes such as manufacturer, architecture, last update, and an `id` attribute for the primary key,
- Merging `Computer` and `Phone` into a single `Device` relation, which would contain an additional attribute **`Type`** to distinguish between computers and phones, and a foreign key to OS,
- Having a `Connection` relation whose attributes would be foreign keys to `Device` and `Printer`,
- Remove the `ComputerOrPhoneInIt` attribute,
- Have id attributes in Room and Device, which would be their sole primary key, but leaving the Nickname attribute in case they would like to store that information.

Solution to Problem 3.22 (Write select queries for a (third!) variation of the COMPUTER table)
(Some of) the answers can be found in the following snippet:

```
/* code/sql/HW_ComputerVariationAdvancedSol.sql */
START TRANSACTION;

DELETE FROM CONNEXION
WHERE Computer = 'A';

SELECT *
FROM COMPUTER;

SELECT *
FROM PERIPHERAL;

SELECT *
FROM CONNEXION;

ROLLBACK;

START TRANSACTION;

DELETE FROM COMPUTER
WHERE ID = 'A';

SELECT *
```

```
FROM COMPUTER;

SELECT *
FROM PERIPHERAL;

SELECT *
FROM CONNEXION;

ROLLBACK;

START TRANSACTION;

DELETE FROM PERIPHERAL
WHERE ID = '15';

SELECT *
FROM COMPUTER;

SELECT *
FROM PERIPHERAL;

SELECT *
FROM CONNEXION;

ROLLBACK;

START TRANSACTION;

DELETE FROM CONNEXION
WHERE Computer <> 'A';

SELECT *
FROM COMPUTER;

SELECT *
FROM PERIPHERAL;

SELECT *
FROM CONNEXION;

ROLLBACK;

SELECT TYPE
FROM PERIPHERAL
WHERE ID = '12';

SELECT ID
FROM COMPUTER
WHERE Model LIKE '%Apple%';

```

```sql
SELECT COUNT(ID)
FROM COMPUTER;

SELECT DISTINCT (TYPE)
FROM PERIPHERAL;

SELECT CONNEXION.Computer
FROM CONNEXION,
  PERIPHERAL
WHERE PERIPHERAL.Type = 'keyboard'
  AND PERIPHERAL.ID = CONNEXION.Peripheral;

SELECT COMPUTER.Model
FROM CONNEXION,
  PERIPHERAL,
  COMPUTER
WHERE PERIPHERAL.Model = 'TP-10 Thermal Matrix'
  AND PERIPHERAL.ID = CONNEXION.Peripheral
  AND CONNEXION.Computer = COMPUTER.ID;

INSERT INTO CONNEXION
VALUES (
  'B',
  '12');

SELECT *
FROM COMPUTER;

SELECT *
FROM PERIPHERAL;

-- Note that the "LastConnexion" attribute has been updated.
SELECT *
FROM CONNEXION;

```

HW_ComputerVariationAdvancedSol.sql[125]

[125] https://rocketgit.com/user/caubert/CSCI_3410/source/tree/branch/master/blob/notes/code/sql/HW_ComputerVariationAdvancedSol.sql

4 Designing a Good Database

Resources

This part of the lecture covers significantly more material than the other, hence we give the details of the references below:

- ER models: (Elmasri and Navathe 2010, ch. 7) or (Elmasri and Navathe 2015, ch. 3)
- The ER to Relational model: (Elmasri and Navathe 2010, ch. 9.1) or (Elmasri and Navathe 2015, ch. 9.1)
- Normalization: (Elmasri and Navathe 2010, ch. 7) or (Elmasri and Navathe 2015, ch. 3)
- UML: not so much in the textbook, but you can look at (Elmasri and Navathe 2010, ch. 7.8, 10.3) or (Elmasri and Navathe 2015, ch. 3.8), or at https://oxygen.informatik.tu-cottbus.de/IT/Lehre/MDD-Tutorial/#d5e126.

4.1 Interest for High-Level Design

Previous relational models have mistakes and limitations:

- What if a hurricane is over more than one state?
- What if an insurance covers more than one car, more than one driver?
- Changing the code "on the fly", as we did for the `HW_Lecture` and `Grade` tables, is difficult and error-prone.

We could go back and forth between relational models (~ logical level) and `SQL` implementations (~ physical level), but we will use even more high-level tools (~ conceptual level):

- Entity Relationship Models (ER, static: DB)
- Unified Modelling Diagrams (UML, dynamic: DB + software)
- Enhanced Entity Relationship Models (EER, adds operations to ER)

Feature	Conceptual	Logical	Physical
(Main) Audience	Business	Designer	Programmer
Entity Names	✔	✔	
Entity Relationships	✔	✔	
Attributes	(✔)	✔	
Cardinalities		✔	✔
Primary Keys		✔	✔
Foreign Keys		✔	✔
Column Data types (Domain)		(✔)	✔
Table Names			✔

Feature	Conceptual	Logical	Physical
Column Names			✔

The conceptual data model is (in theory at least) independent of the choice of database technology.

Remember that in relational models, relations were representing entities (`Student`) and relationships (`Majors_In`). At the conceptual level, and more particularly in ER diagram, the distinction is made between entities and relationship.

4.2 Entity-Relationship Model

Data is organized into **entities** (with **attributes**), **relationships** between entities (with **attributes** as well).

4.2.1 Entities

- Entity = Thing, object, with independent existence.
- Each entity has attributes (properties)

Entity A :

- Name = Clément Aubert
- Address = HCOB, HA, E. 128 ; Invented St., Auguta, GA
- Diploma = Ph.D in CS; BS in Math
- Highest Diploma = Ph.D in CS
- Favorite Class = CSCI 1301
- Favorite Sport = NULL

Some vocabulary:

- Entity = actual thing (individual)
- Entity type = collection of entities with the same attributes
- Entity set (or collection) = collection of all entities of a particular entity type.

4.2.1.1 Attributes

Attributes can be

- Composite (divided in smaller parts) or simple (atomic)
- Single-valued or multi-valued
- Stored vs derived
- Nested!

{...} = multi-valued

(...) = complex

For instance, one could

- store the name using a composite attribute (First Name, {Middle Name}, Last Name),
- store multiple addresses using the "schema" `{Address(Street, Number, Apt, City, State, ZI`
- derive the value of "Highest Diploma" using the value(s) stored in "Diploma".

4.2.1.2 Key Attributes

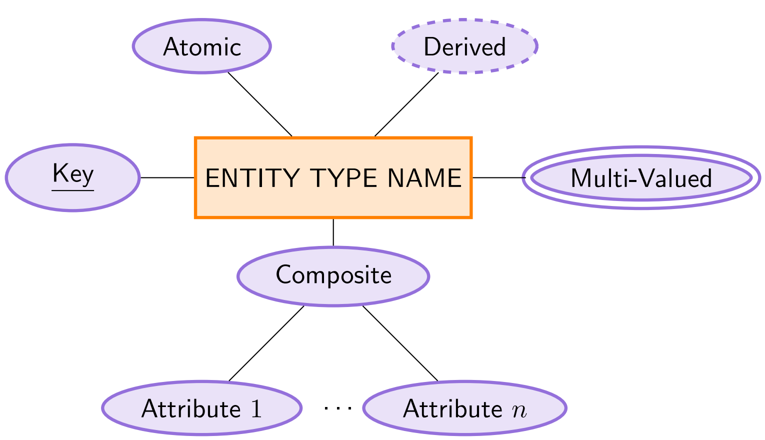
A key attribute is an attribute whose value is distinct for each entity in the entity set.

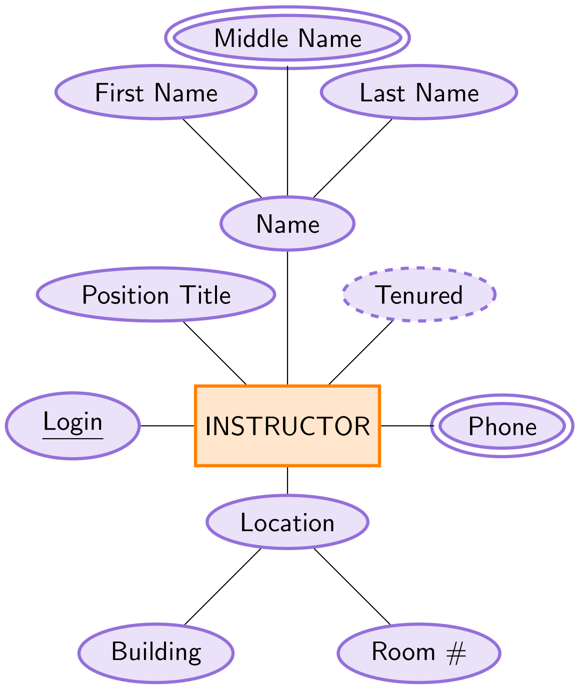
- Serve to identify an entity,
- Can be more than one such attribute (and we leave the options open),
- Cannot be multiple attributes: if more than one attribute is needed to make a key attribute, combine them into a composite attribute and make it the key.
- A composite attribute that is a key attribute should not still be a key attribute if we were to remove one of the attribute (similar to the minimality requirement).
- An entity with no key is called a weak entity type: it is an entity that will be identified thanks to its relation to other entities, and thanks to its partial key (we will discuss this later).

4.2.1.3 Drawing Entity Types

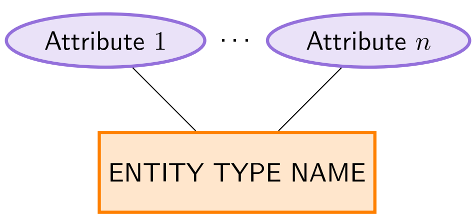
- Entity = squared box (name in upper case)
- Attribute = rounded box connected to square box (name in lower case)

If the attribute is ...,	then...
composite	other attributes are connected to it
multi-valued	the box have double lines
derived	the box have dotted lines
a key	the name of the attribute is underlined





In the following, we'll focus on the relationship between the entities more than on the attributes of particular entities, so we'll sometimes simply draw



leaving the attributes un-specified (but that does not mean that they all have to be atomic) or even just

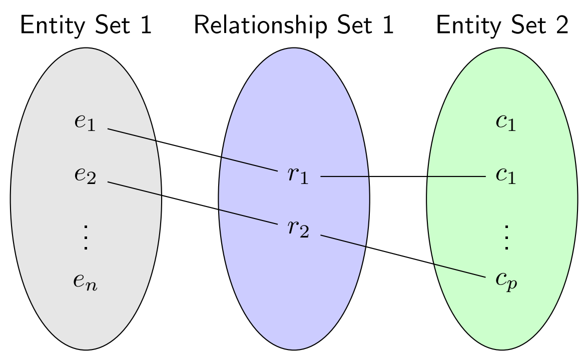
but that does not mean that the entity type have no attribute!

4.2.2 Relationships

4.2.2.1 Vocabulary

- Relationship = actual relation (or action) between entities (“teaches”, “loves”, “possesses”, etc.).
- Relationship instance = r_1 associates n entities e_1, ..., e_n (“Pr. X teaches CSCI YYY”, “There is love between Mary and Paul”, etc.)
- Relationship set = collection of instances
- Relationship type = abstraction (“Every course belong to one instructor”, “Love is a relation between two persons”, etc).

E_1, ... E_n *participate* in R, e_1, ..., e_n *participate* in r_1, n is the degree.

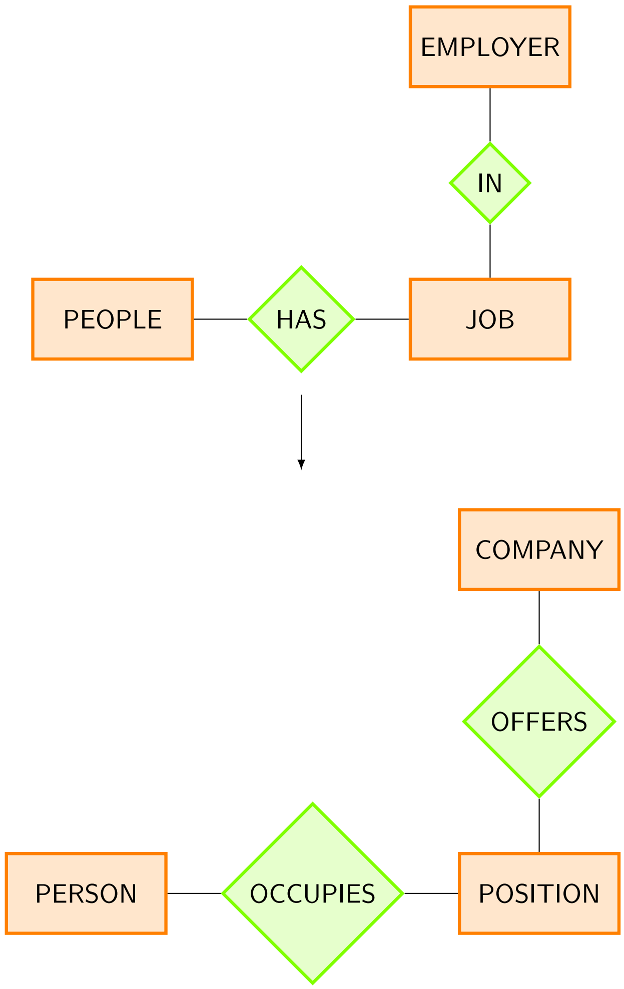


Note that we can have Entity Set 1 = Entity Set 2, in which case we say the relation is recursive[1].

Naming convention:

- Use a singular name for entity types.
- Use a verb for relationship.
- Relationship types are drawn in diamonds.
- Drawing usually reads left to right, and top-down.

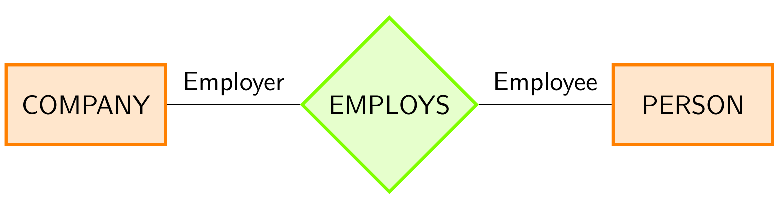
[1] Some sources call the relationships between an entity and itself “unary.” Note that with our convention, it does not make sense to speak of a unary relationship.



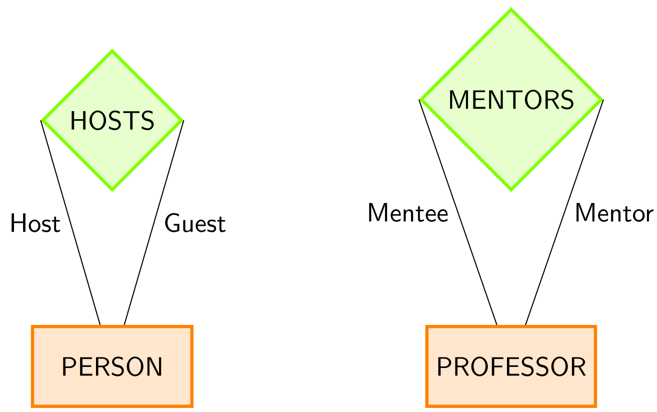
4.2.2.2 Role Names and Recursive Relations

Convenient, and sometimes mandatory, to give role names.

If we want to stress that we are considering only one aspect of an entity type (that is, a person is not only an employee, a company is not only an employer, but this aspect is crucial for the "EMPLOYS" relation):



We can also use it to make the "right-side" and the "left-side" of a recursive relationship explicit:



Finally, we will sometimes use "Role Name of Entity 1 : Role Name of Entity 2" as a notation for the relation between them. For instance, we can write "Employer:Employee" to denote the "EMPLOYS" relation, and we will also use this notation when the relationship is between different entities, and write e.g. "PERSON:POSITION" for the "OCCUPIES" relation.

4.2.2.3 Constraints

Two constraints, called "structural constraints", applies to relationship types: cardinality ratio and participation constraint. They both concerns the number of relationship instances an entity can participate in (which is different from the cardinality of a relationship type).

4.2.2.3.1 Cardinality Ratio **Maximum** number of relationships instances that an entity can participate in.

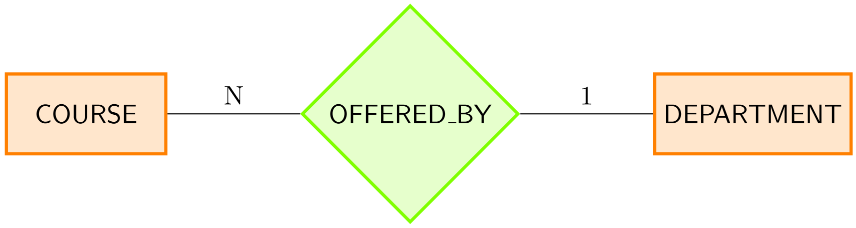
For binary relations, can be $1 : 1$, $N : 1$, $1 : N$, or $M : N$. The 1 stands for "at most 1", and the M, N, and P stand for "possibly more than 1", or "no maximum". In ER diagram, we do not count, and do not make the distinction between "at most 5" and "at most 10", for instance[2].

Possible examples include:

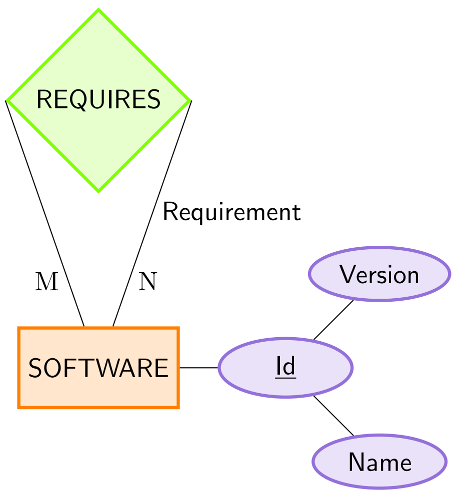
Relation	Possible Ratio	Explanation
MENTOR : MENTEE	$1 : N$	"A mentor can have multiple mentees, a mentee has at most one mentor."
PERSON : SSN	$1 : 1$	"A person has one SSN, a SSN belongs to one person."
COURSE : DEPARTMENT	$N : 1$	"A course is offered by one department, a department can offer any number of courses."
STUDENT : TEAM	$M : N$	"A student can participate in multiple team, a team can have multiple students."

We indicate the ratio on the edges:

[2] An alternative notation, detailled later on, will address this shortcoming.



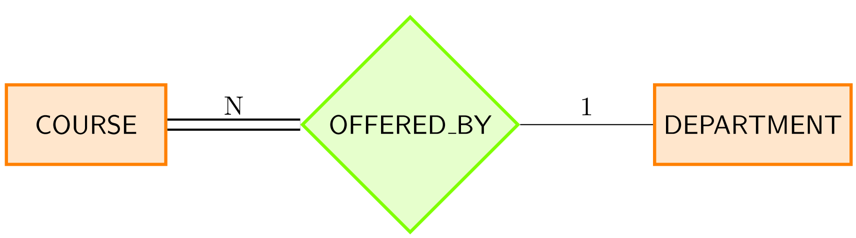
Note that reflexive relations can have any ratio as well. An example of $M : N$ recursive relation could be:



4.2.2.3.2 Participation Constraint **Minimum** number of relationships instances that an entity can participant it, a.k.a. "minimum cardinality constraint."

The participation can be total (a.k.a. existence dependency, the entity **must** be in that relationship at least once) or partial (the entity may or may not be in that relationship).

Total is drawn with a double line, partial is drawn with a single line:



This reads "a course **must** be offered by a department, but a department may or may not offer courses."

4.2.2.4 Attributes

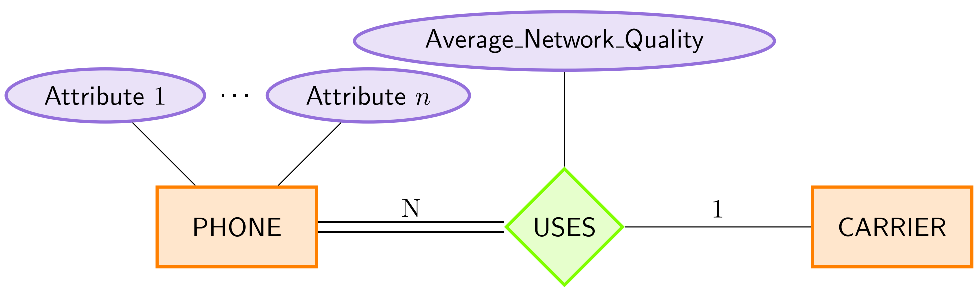
Relationships can have attributes too. The typical example is a date attribute, but other examples include

- TEACHING relation between PROF and CLASS ($N : M$) could have a "Quarter" attribute.
- MENTORING relation between MENTOR and MENTEE ($1 : N$) could have a "Since" attribute.
- EMITED_DRIVING_LICENCE between DMV and PERSON ($N : 1$) could have a "Date" attribute.

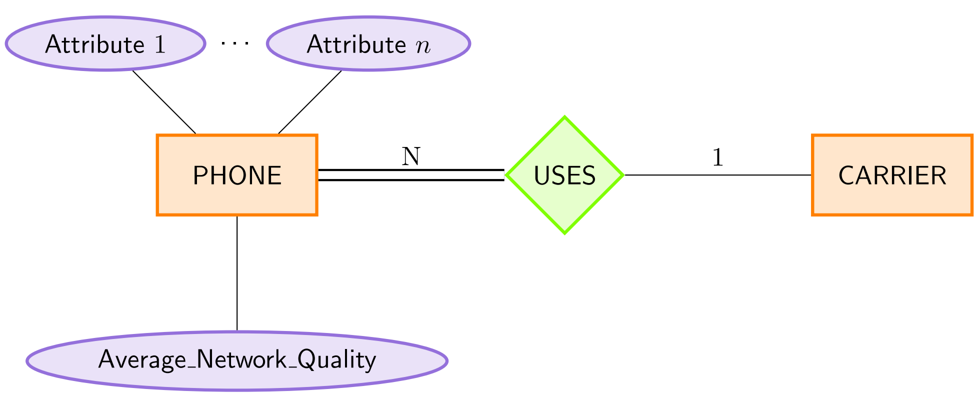
Note that an attribute on a relationship type can be atomic or composite, single or multi-valued, stored or derived, but that it cannot be a key attribute (after all, there are no entity to identify!).

Note that there are some moving aspects here: atributes on $1 : 1$, $1 : N$, $N : 1$ relationships can be migrated (to the N side when there is one, or to either side where there is none).

For instance, imagine that every phone uses exactly (= "at most and at least") one carrier, that a carrier can provide network to multiple phones, and that the average quality of the network is an attribute in this relationship:



Then each instance of the relation would be of the form ("Phone X", "Carrier Y", "9/10") for some way of ranking the average quality from 0 to 10. Note that, from the fact that the relationship is $N : 1$, this means that there is only one tuple involving "Phone X": this means that the average quality could actually be seen as a property of *the phone*, and hence be migrated as an attribute to the phone side:

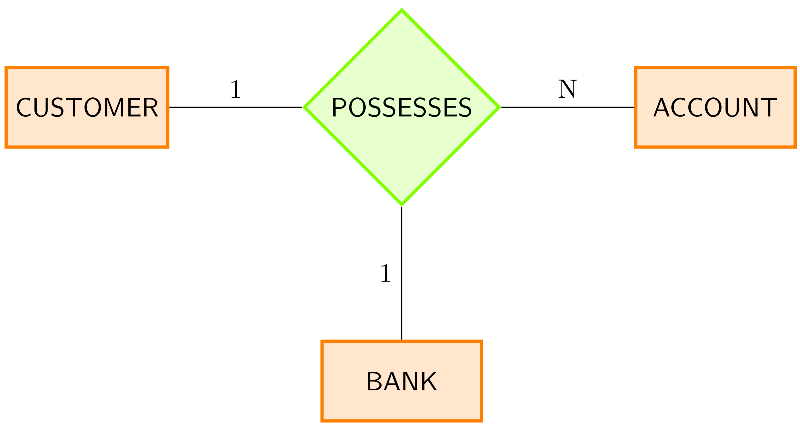


Note that we could *not* migrate the "average phone quality" to the "Carrier" side: imagine if we had the instances ("Phone X", "Carrier Y", "9/10") and ("Phone Z", "Carrier Y", "3/10"), then should the attribute of "Carrier Y" be "9/10" or "3/10": we have no way of deciding based on this model. Whenever it is a good choice to migrate this attribute or not will depend on the requirement of the models, and it may not always be appropriate to migrate the attribute to the entity. In the case of $1 : 1$ relationship, migrating the attribute to *both sides* (i.e., to both entities) would be a mistake, since it would introduce redundancy in your model.

As an exercise, you can look at the relationships TEACHING, MENTORING and EMITED_DRIVING_LICENCE that are listed above, and see if the attributes can be migrated or not, and if yes, on which side.

4.2.2.5 Relationships of Degree Higher than Two

Of course, relationships can have a degree higher than two. An example of a ternary relation could be:

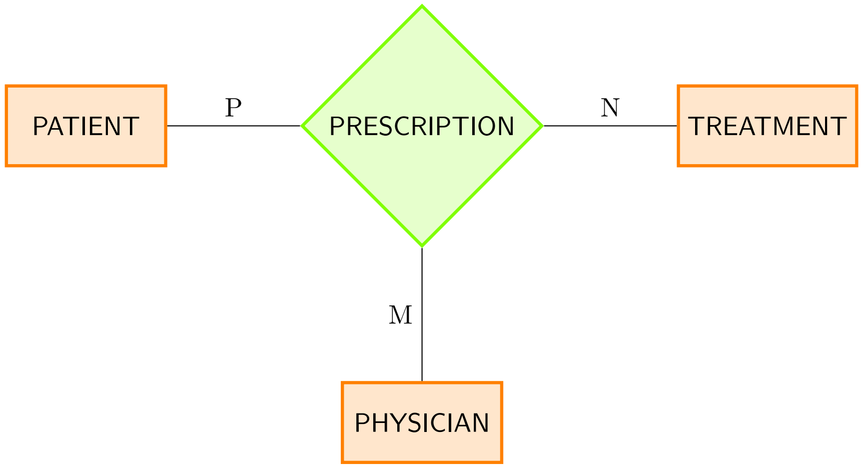


To determine cardinality ratio, one should fix all but one parameters, and wonder how many values of the remaining parameter can be in that relationship. Another wording for the same idea can be found in this thread[3].

Four our example, Customer Y and Bank Z could be in relationship with more than one account (hence the "N"). On the opposite, Customer Y and Account K would be in relationship with only one bank (hence the "1" on the bottom), and Bank Z and Account K would belong to only one customer (hence the "1" on the left).

Let us look at two other examples. First, assume we want to collect information about the treatment prescribed by physicians to patients, we could use a relationship like the following one:

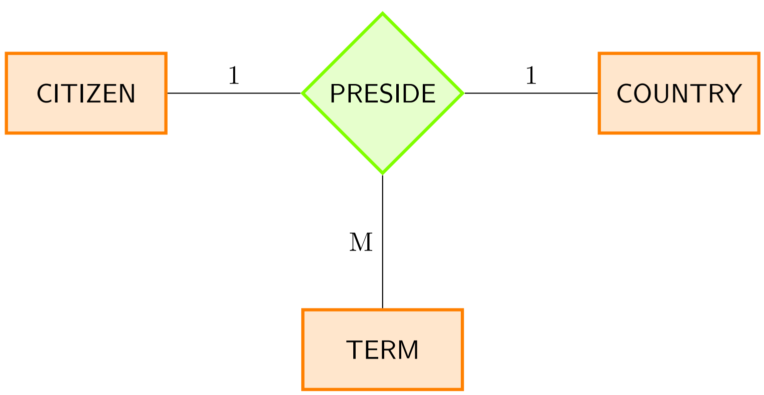
[3]https://stackoverflow.com/a/38395315



Where

- The “P” stands for the fact that the same physician can prescribe the same treatment to multiple patients,
- The “N” stands for the fact that different treatment can be prescribe by the same physiciant to the same patient,
- The “M” stands for the fact that the same patient can get the same treatment from different physicians.

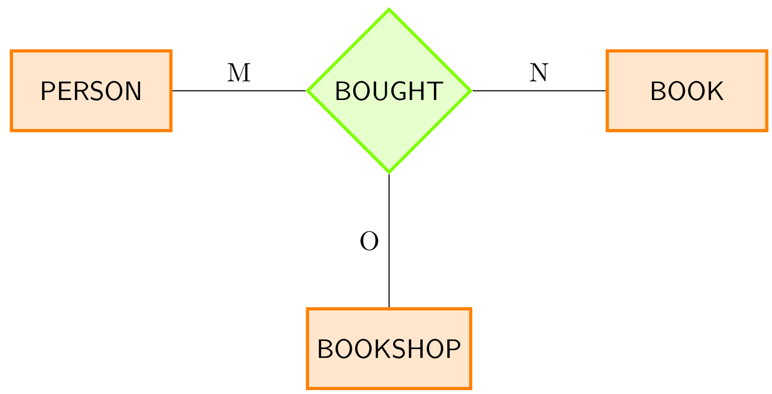
Now, if we want to store information about who is the president of a country during a term, we could get something like:

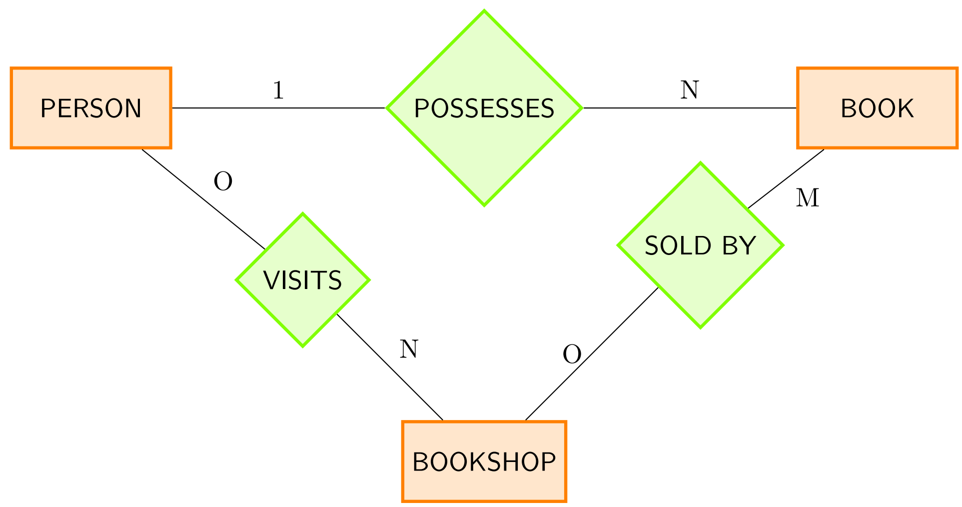


Note that this representation of the data assumes that a citizen cannot be the president of two different countries during the same term (the right 1), which could be debatable.

It is sometimes impossible to do without relations with arity greater than 2. For instance, consider the following two diagrams[4]:

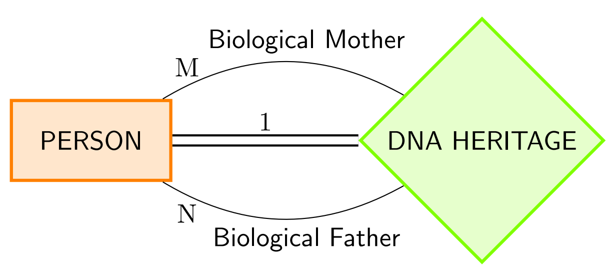
[4]Where the “BOOK” entity does not refer to one particular physical copy of a book, but to books in general, i.e., “The book on my shelf” (physical copy) as opposed to “*The Wizard of Oz*” (general).



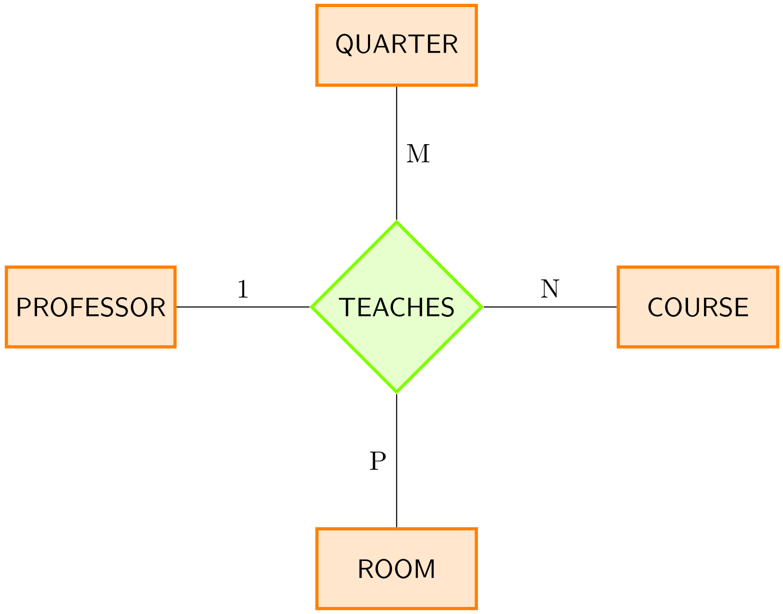


You should realize that they convey different information. For instance, you can know for a fact that a person visit a bookshop only if they bought something in it, while the second diagram de-correlate the act of buying with the visit to a bookshop. Similarly, the second diagram could give you a hint that a person that owns a copy of a book Z and visits a bookshop X that sells it could also visit it, but you will not know that for sure.

An example of recursive ternary relation could be:



An example of relation of degree 4 could be:



The cardinality ratio are computed using the same method as described before.

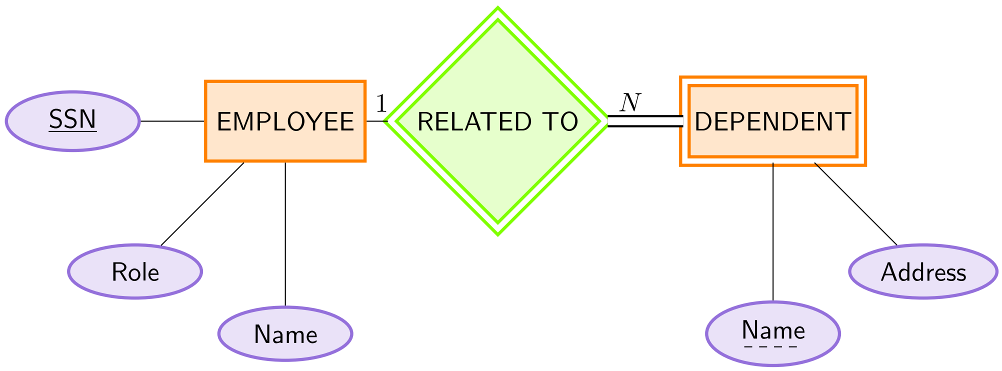
4.2.3 Weak Entity Types

There are actually two sorts of entity types:

- Strong (a.k.a. regular, the ones we studied so far), with a key attribute,
- Weak, without key attribute.

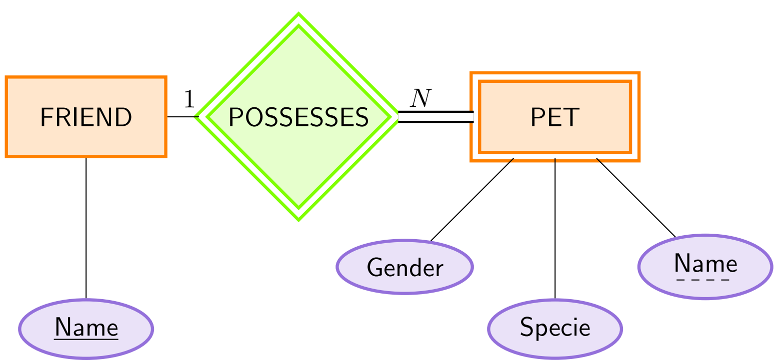
Weak (or **child**) entity types are identified by **identifying / owner** type that is related to it, in conjunction with one attribute (**the partial key**). That relation is called **identifying** (or **supporting**) **relationship**, and weak entities have a total participation constraint. The **partial key** is an attribute, that, *when paired with an entity with which they are in relation through their identifying relationship*, allows to identify a particular entity.

Weak entities and identifying relationships have a double border, and partial key have a dotted underline, as follows:

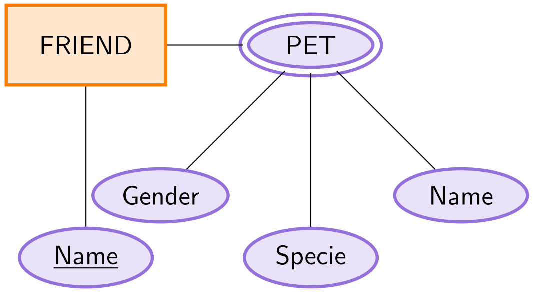


The idea here is that we do not need to gather data about all the dependent in the world, or in isolation, but are interested in dependent only if they are related to en employee in our database. Just having the name of a dependent is not enough to identify them, but having their name *and* the SSN of the employee they are related to is enough. The identifying relation always have ratio $1 : M$ or $1 : 1$: a weak entity cannot be related to more than one entity of the owner type, so that $M : N$ ratio are not possible (cf. e.g. https://dba.stackexchange.com/q/17207). If you need to have, for instance, a dependant connected to multiple employees, then that means that your dependent entity should be strong, because it has an existence "of its own".

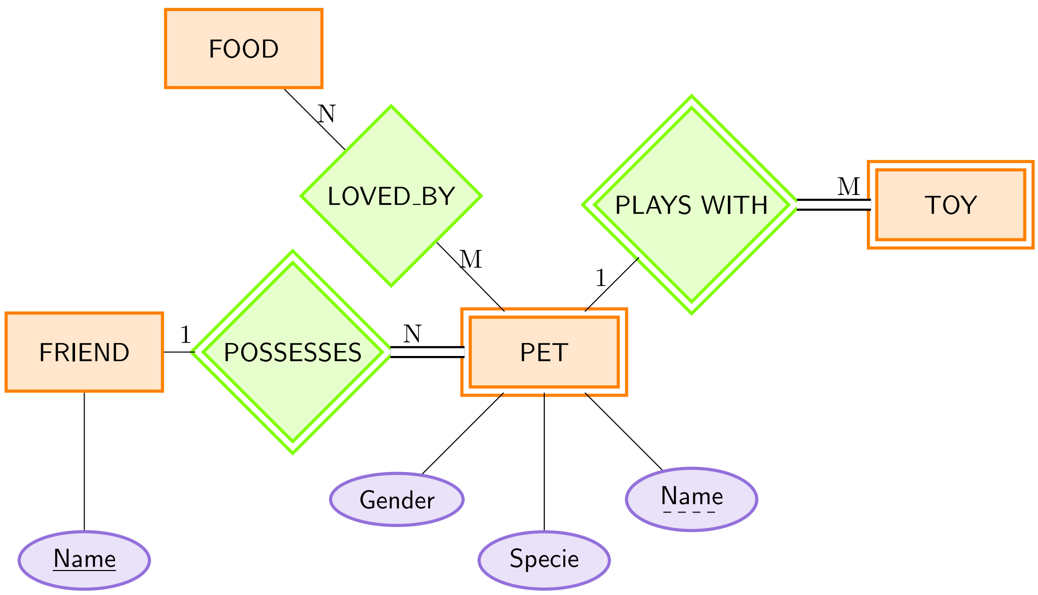
You may wonder why we do not represent weak entities simply as (composite, multi-valued) attributes of their owner type. For instance, why would we use



instead of



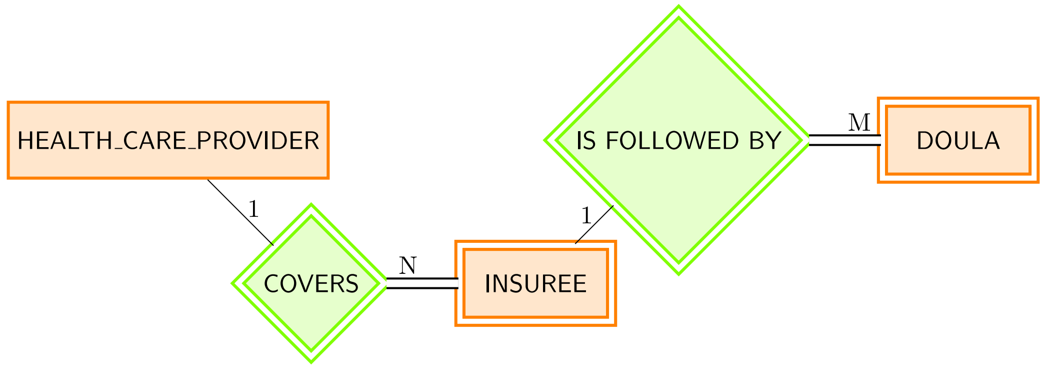
? The answer depends whenever we need to have the ability to represent our weak entities (here, PET) as being in relationship with other entities (that can themselves be weak!), as follows:



This would be impossible if PET was an attribute of FRIEND! Whenever the pet entity type is involved in other relationships or not should help you in deciding which representation to choose.

- Weak entities types can sometimes be replaced by complex (composite, multi-valued) attributes, unless they are involved in other relationships.
- Owner can itself be weak!
- The degree of the identifying relationship can be more than 2 (cf. e.g., https://stackoverflow.com/q/15393587/).

Another example of weak entity whose owner is weak as well could be:



The idea being that the Health care provider cares about an insure only if they are covered by them, and that they care about the doula only if they are currently helping one of their insure.

4.2.4 Alternative Notations

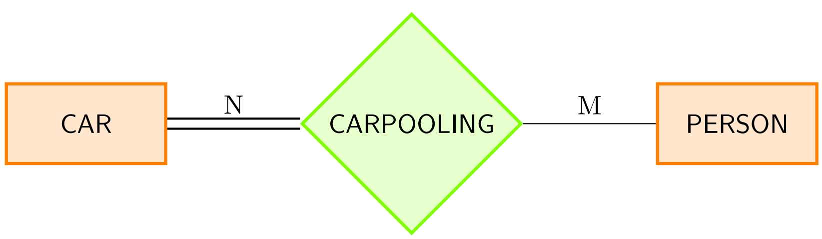
Multiple notations have been used to represent the ratio and constraint on relationship.

In the following, we introduce two of them: the Min/Max and the Crow's foot notations.

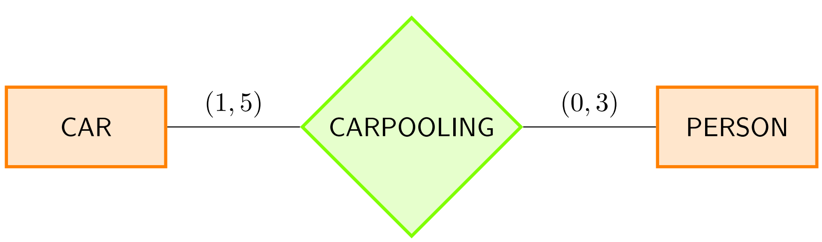
4.2.4.1 Notation with Explicit Maximal (Min/Max Notation)

The two constraints can be written on the same side, and the N, M, P ratio can be replaced by actual number, providing more information.

For instance,



could be drawn as



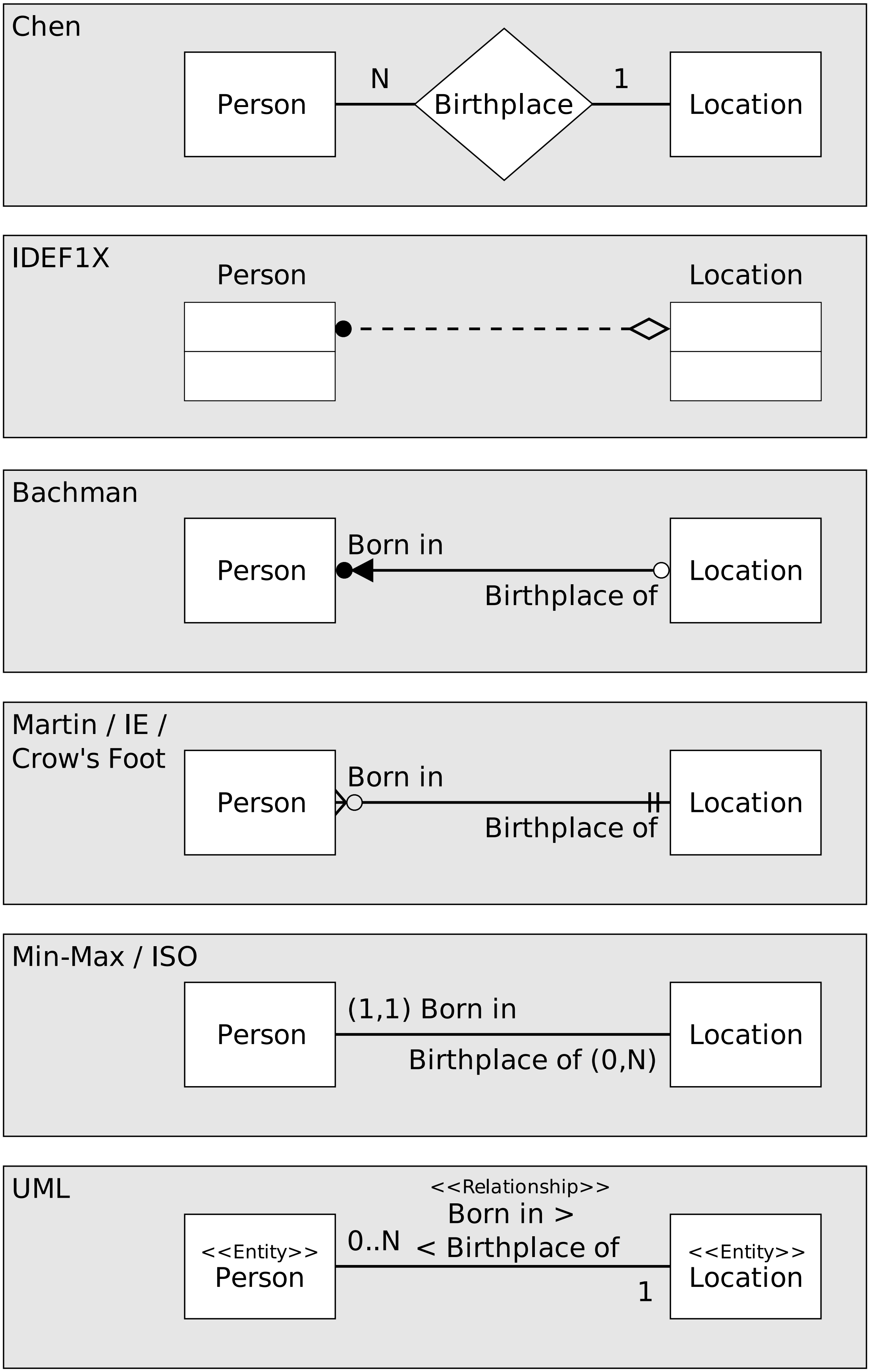
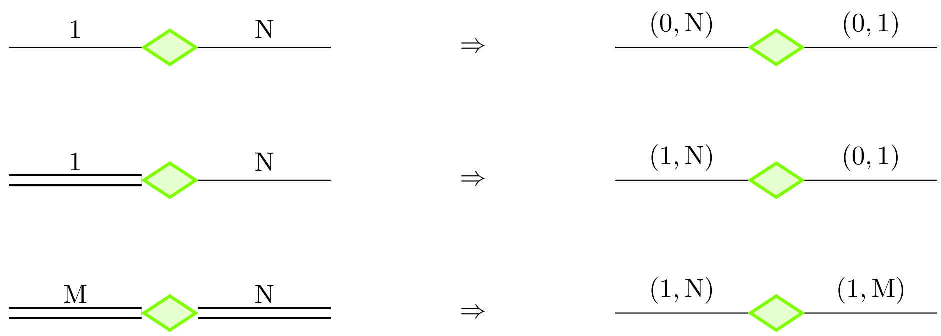


Figure 4.1: A Quick Overview of the Notations for ER Diagram (courtesy of wikipedia[5])

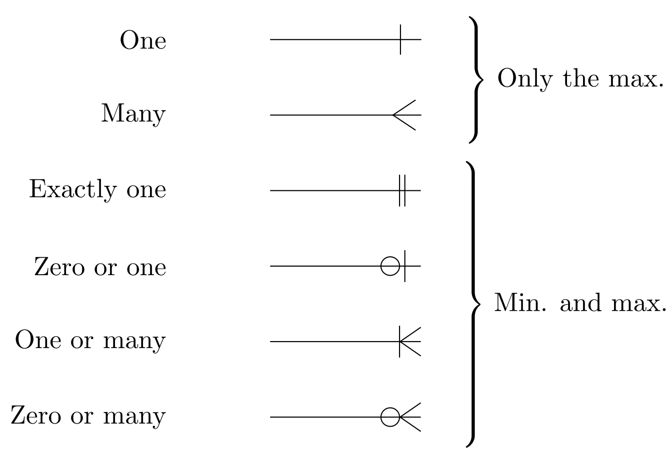
meaning that

- A car can be used to carpool between 1 and 5 persons (and that *it must* be used for at least 1 person),
- A person can be registered for 0, 1, 2 or 3 carpool at the same time.

More generally, we have the following:



4.2.4.2 Crow's Foot Notation



4.2.5 Enhanced Entity–Relationship Model

Extended (or Enhanced) ER Models (EER) have additionaly:

- Subtype / Subclass: "every professor is an employee". There is a class / subclass relationship (you can proceed by specialization or generalization).
- Category (to represent UNION): an OWNER entity that can be either a PERSON, a BANK, or a COMPANY entity type.

Closer to object-oriented programming.

4.2.6 Reverse Engineering

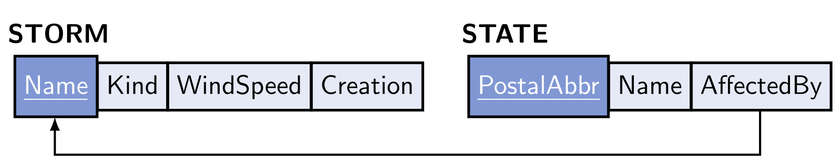
It is possible to go from relational models to ER models, and sometimes needed: if you are given an implementation that seems poorly design, this can be a way of "backing up" and thinking about the (sometimes implicit) choices that were made during the implementation, to eventually correct them.

For instance, consider the code we studied in "A First Example":

```sql
CREATE TABLE STORM (
  NAME VARCHAR(25) PRIMARY KEY,
  Kind ENUM ("Tropical
    Storm", "Hurricane"),
  WindSpeed INT,
  Creation DATE
);

-- We can change the enumerated datatype:
ALTER TABLE STORM MODIFY Kind ENUM ("Tropical Storm",
  "Hurricane", "Typhoon");

CREATE TABLE STATE (
  NAME VARCHAR(25) UNIQUE,
  Postal_abbr CHAR(2) PRIMARY KEY,
  Affected_by VARCHAR(25),
  FOREIGN KEY (Affected_by) REFERENCES STORM (NAME) ON
    DELETE SET NULL ON UPDATE CASCADE
);

```

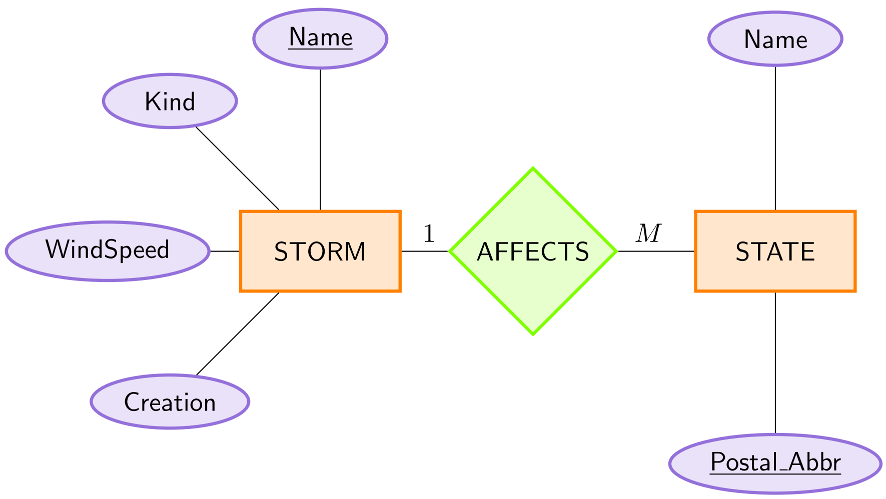
HW_Storm.sql[6]

It corresponds to the following relational model:



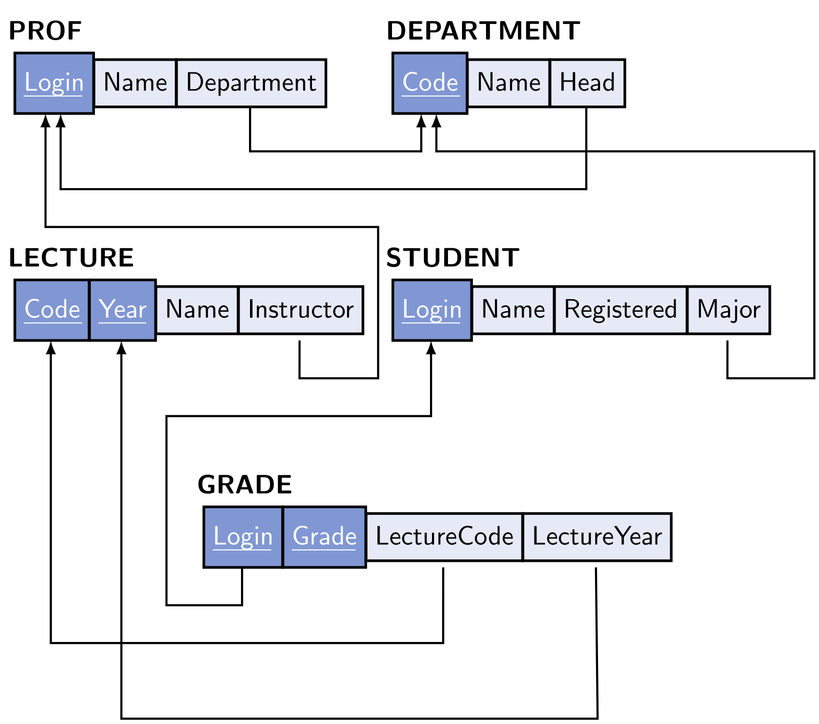
which in turn corresponds to the following ER diagram:

[6]https://rocketgit.com/user/caubert/CSCI_3410/source/tree/branch/master/blob/notes/code/sql/HW_Storm.sql

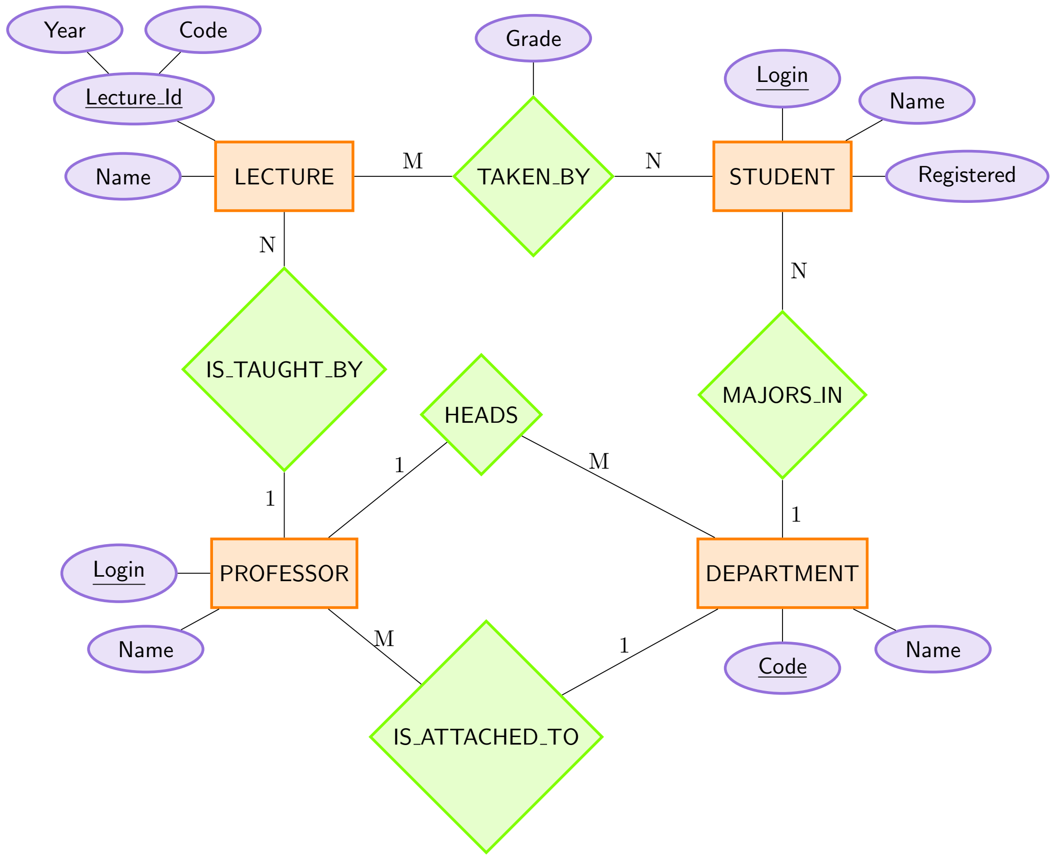


Looking at this diagram made it obvious that our code has a flaw: a stom can affect more than one state! Turning the 1 on the left-hand side of the "AFFECTS" relationship into a M is immediate on the diagram, but, of course, mapping it back to a relational model, and then implementing it correctly, will require more work. In any case, if you had not noted already this flaw, reverse-engineering this code highlighted it quite clearly.

If we look back at Problem 3.5 (Revisiting the PROF table), we had already made a first step, since we converted the code into the following relational model:



Going a bit further, we could extrapolate just a little bit and get the following ER diagram:



As we noted in our solution to the second question, this model has several limitations. To list a few, this representation can not handle the following situations:

- If multiple instructors teach the same class,
- If the lecture is taught more than once a year (either because it is taught in the Fall, Spring and Summer, or if multiple sections are offered at the same time),
- If a Lecture is cross-listed, then some duplication of information will be needed.

Looking at it as an ER diagram should help you in understanding why we have those flaws, and how they could be addressed, and "testing" the model should be made easier in its ER form than as SQL code.

4.3 ER-to-Relational Models Mapping

4.3.1 Intro

We have to map all of the following:

Entity	Strong, Weak
Attributes	Composite, Key, Atomic, Multi-valued, Partial Key

Relationships	Binary ($1:1$, $N:1$, $1:N$, $N:M$), n-ary

Using four tools: Relations, Attributes, Primary Keys, Foreign Keys.

4.3.2 Algorithm

We will use three techniques to represent some of the relationships, the foreign key approach, the merged relations approach and the cross-reference approach. They are detailed and illustrated after the algorithm, which goes as follows:

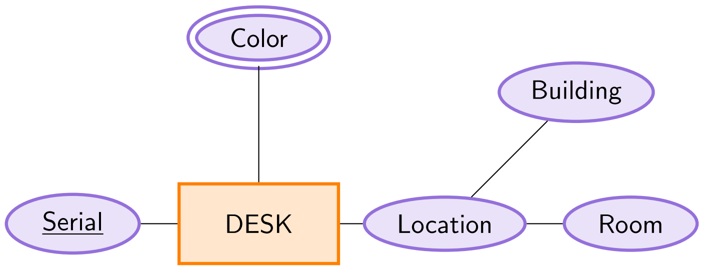
#		is mapped to
1	Strong Entity	Relation with all the simple attributes. Decompose complex (composite) attributes. Pick a key to be the PK, if it is composite, take its elements.
2	Weak Entity	Relation with all the simple attributes. Decompose complex attributes. Add as a foreign key the primary key of the relation corresponding to the owner entity type, and make it a primary key, in addition to the partial key of the weak entity. If the owner entity type is itself weak, start with it.
3	Binary $1:1$ Relationship Types	Foreign Key, Merge Relations or Cross-Reference approach
4	Binary $1:N$ Relationship Types	Foreign Key or Cross-Reference approach
5	Binary $M:N$ Relationship Types	Cross-Reference approach
6	n-ary Relationship Types	Cross-Reference approach
7	Multi-valued Attributes	Create a new relation, add as a foreign key the primary key of the relation corresponding to the original strong entity type. Make all the attributes be the primary key.

whose primary key is the foreign key to the relation corresponding to the entity.

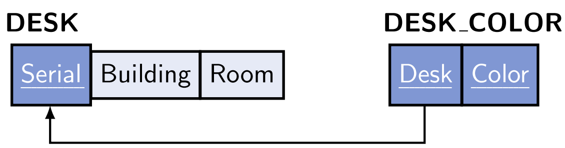
1. Foreign Key Approach: choose one of the relation (preferably with total participation constraint, or on the N side), add a foreign key and all the attributes of the relationship.
2. Merged Relation Approach: If both participation constraints are total, just merge them. Primary key = just pick one (or take both). If we were working on the implementation, we would add a `NOT NULL` constraint on the attribute that is not part of the primary key anymore.
3. Cross-Reference or Relationship Relation Approach: Create a lookup table with an appropriate number of foreign keys, pick some of them (the one on the N side, both if the ratio is $M:N$, for n-ary it is a bit more complex, cf. example below) as the primary key.

Every time a relationships have attributes, they are mapped to the resulting relation.

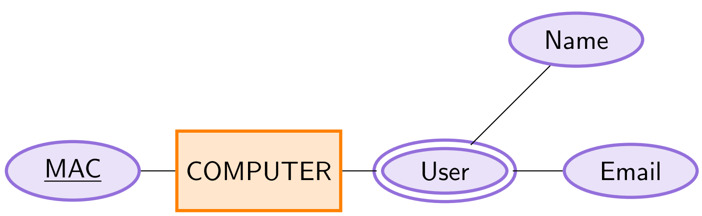
Let us look in more details at some of those steps. For strong entities, using steps 1 and 7, the following:



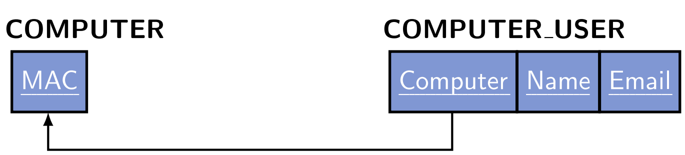
would give:



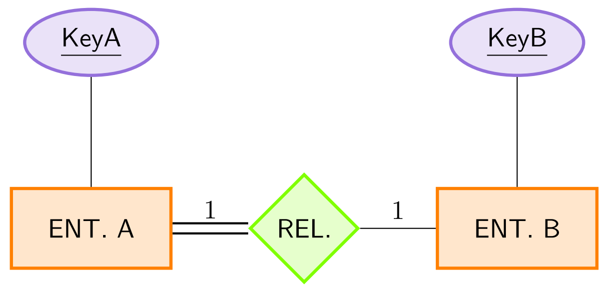
And note that if Serial was a complex attribute, we would just "unfold" it, or decompose it, and make all the resulting attributes the primary key of the relation. If one of the attribute was at the same time multi-valued and composite, as follows:



Then we would obtain:



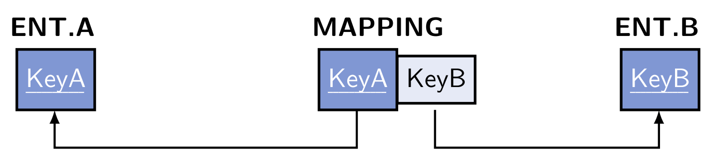
For relationships, things are a bit more complicated. Consider the following:



Since it is a $1:1$ relationship where one of the side has a partial constraint, we have the choice between two approaches. The foreign key approach would give:

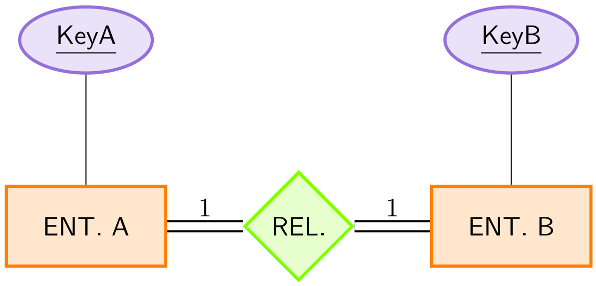
Note that we could also have added the foreign key on the side of ENT.B, referencing the key of ENT.A. But since ENT.A has a total participation constraint, we know that the value of FK will always exist, whereas some entities in ENT.B may not be in relationship with an entity from ENT.A, creating the (nefast) need for **`NULL`** values.

For the same diagram, the cross-reference approach would give:

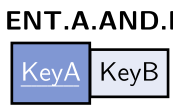


Similarly, note that, in MAPPING, KeyB, or KeyA and KeyB, would also be valid primary keys, but that it makes more sense to have KeyA being the primary key, since we know that ENT.A has a total participation constraint, but ENT.B does not.

If both participation constraints were total, as follows:

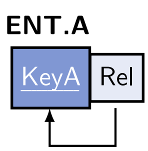


Then we could use the merged relations approach, and get:

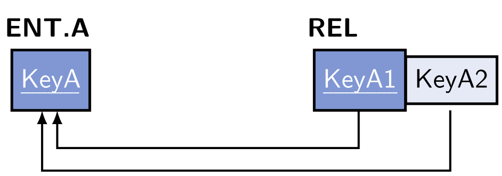


We picked KeyA to be the primary key for the same reason as before. Note that merging the two entities into one relation also means that you have eventually to do some work on the relations that were referring to them.

Of course, if ENT.A and ENT.B are the same entity (that is, REL is recursive), we would get:



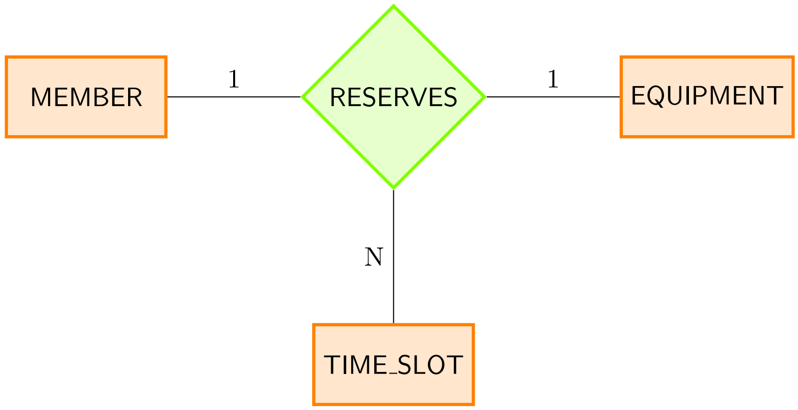
or



depending on the approach we chose.

Binary $1 : N$ and binary $M : N$ relationships are dealt with in a similar way, using foreign key or cross-reference approaches. The most difficult part of the mapping is with n-ary relationships: we have to use cross-reference approaches, but determining the primary key is not an easy task. Consider the following[7]:

[7] This developement was actually asked at https://dba.stackexchange.com/q/232068/.



The arity constraints here can be rephrased as:

- A member can reserve a particular equipment at multiple time slots (the N),
- An equipment can be reserved at a particular time slot by only one member (the 1 on the left),
- A member can reserve only one equipment per time slot (the 1 on the right).

And note that there is no total participation constraint.

To reprent the RESERVES relationship, we need to create a relation with attributes referencing the primary key of MEMBER, the primary key of TIME_SLOT, and the primary key of EQUIPMENT. Making them all the primary key does not represent the fact that the same equipment cannot be booked twice during the same slot, nor that a member can book only one equipment per slot, but allows members to reserve a particular equipment at multiple time slots. To improve this situation, we can either

1. take the foreign key to MEMBER and the foreign key to TIME_SLOT to be the primary key of this relation,
2. or take the foreign key to EQUIPMENT and the foreign key to TIME_SLOT to be the primary key of this relation.

Both solutions enforce only *some* of the requirement expressed by the ER diagram.

4.3.3 Outro

ER Model	Relational Model
Entity type	Entity relation
$1:1$ or $1:N$ relationship type	Foreign key (or relationship relation)
$M:N$ relationship type	Relationship relation and two foreign keys
n-ary relationship type	Relationship relation and n foreign keys
Simple attribute	Attribute
Composite attribute	Set of simple component attributes
Multivalued attribute	Relation and foreign key
Value set	Domain
Key attribute	Primary key

You can have a look at e.g. http://holowczak.com/converting-e-r-models-to-relational-models/ to get a slightly different explanation of this conversion, and additional pointers.

4.4 Guidelines and Normal Form

What makes a good database? At the logical (conceptual) and physical (implementation) levels. We will answer belowe this question broadly, and will then use the concept of *functional dependency* to capture some of those notions precisely, mathematically, and be able to detect issues preventing our database from meeting the usual goals.

In general, a good data base should:

1. Enforces information preservation (and avoid loss of information)
2. Have minimum redundancy
3. Makes queries easy (avoid redundant work, make **`SELECT`** and select-project-join easy)

Normally, consistency will simply follows if those goals are met.

For ER diagrams, some of the usual techniques[8] are:

- Limit the use of weak entity sets.
- Do not use an entity when an attribute will do.

4.4.1 General Rules

4.4.1.1 Semantics

1 relation corresponds to 1 entity or 1 relationship type

4.4.1.2 No Anomalies

1. **Insertion Anomalies** Having to invent values or to put **`NULL`** to insert tuples, especially on a key attribute!
2. **Deletion Anomalies** Loosing information inadvertently
3. **Modification Anomalies** Updates have to be consistent.

(Bad!) Example:

```
---------- (Login, Name, AdvisoryName, AdvisorOffice, Major,
↪  MajorHead)

-----------(Office, PhoneNumber, Building)
```

1. Advisor without student
2. Delete last student of advisor
3. Advisor change name.

[8] Cf. for instance http://infolab.stanford.edu/~ullman/fcdb/aut07/slides/er.pdf.

4.4.1.3 `NULL` Should Be Rare

`NULL` has 3 meanings, wastes space, and makes join / nested projections harder.

Example:

```
STUDENT(Login, …, siblingEnrolled)
```

Transform into “Emergency Contact in University” relation (bonus: allow multiple contacts).

4.4.1.4 Identical Attributes in Different Tables Should Be (Primary, Forgein) Key Pairs

Example with advisorOffice and Office: if we try to write a join to obtain the phone number of a student’s advisor, we will obtain all the phone.

4.4.2 Example

```
MARKER(Owner, Color, OwnerOffice, Brand, BrandEmail)

TEACHER(Office, Name, Phone)
```

Corrected to:

```
MARKER(Owner, Color, Brand)

TEACHER(Office, Name, Phone)

BRAND(Name, Email)
```

4.4.3 Functional Dependencies

Functional dependencies (FD) is a formal tool used to assess how “good” a database is, a property of the relation schema. Functional dependencies list the constraints between two sets of attributes from the database. For instance, if X and Y are (sets of) attributes, $X \to Y$ reads “X fixes Y”, and implies that the value(s) of Y is fixed by the value(s) of X.

4.4.3.1 Using Semantics of Attributes

“What *should* be.”

Let us list all the attributes of our previous example:

```
MARKER.Owner, MARKER.Color, MAKER.Brand, TEACHER.Office,
 ↪  TEACHER.Name,
TEACHER.Phone, BRAND.Name, BRAND.Email
```

Think about their dependencies, and list them:

- `TEACHER.Name` ⟶ `TEACHER.Office`
- `BRAND.Name` ⟶ `BRAND.Email`
- `TEACHER.Office` ⟶ `TEACHER.Name`
- `TEACHER.Office` ⟶ `TEACHER.Phone`
- `MAKER.Owner` and `MARKER.Color` ⟶ `MARKER.Brand` ?

4.4.3.2 Using Relation States

"What *is*.", can disprove some of the assumptions made previously, but should not add new dependencies based on it (they may be by chance!).

- Maybe `TEACHER.Office` ⟶ `TEACHER.Name` does not hold, because teachers share offices?
- Maybe `TEACHER.Name` ⟶ `MARKER.Brand` and `MARKER.Color` seemed to be enforced by the state, but we should not add a functional dependency based on that: there are no "requirement" that a Teacher must always buy the same brand and color, this could simply true be by chance so far and should not be imposed to the teachers.

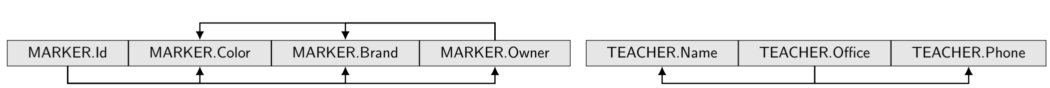
A particular state cannot enforce a FD, but it can negate one.

Example:

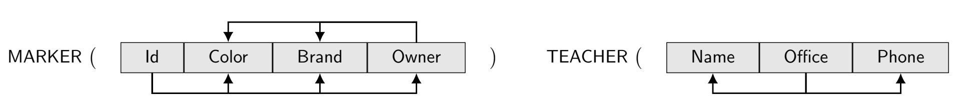
Att. 1	Att. 2	Att. 3
Bob	15	Boston
Bob	13	Boston
Jane	12	Augusta
Emily	12	Augusta

May hold	Will not hold
Att. 2 ⟶ Att. 3	Att1 ⟶ Att2
Att. 3 ⟶ Att. 2	Att. 3 ⟶ Att. 2
Att. 1 ⟶ Att. 3	Att. 2 ⟶ Att. 1
{Att. 1, Att. 2} ⟶ Att. 3	{Att. 3, Att. 2} ⟶ Att. 1

4.4.3.3 Notations



Or, more conveniently:



If an attribute is a foreign key to another, we will draw an arrow between relations:

Note that:

- X and Y are sets, we will write A instead of $\{A\}$, but keep writing $\{A, B\}$ for $\{A, B\}$.
- $\{A_1, ..., A_n\} \to \{B_1, ..., B_m\}$ means that A_1 and ... and A_n fix B_1, and that A_1 and ... and A_n fix B_n, etc.
- FD_1, FD_2, ..., FD_n for the list of functional dependencies, F for all of them.
- $A \to B$ does not imply nor refute $B \to A$.
- We will not write the FD that are implied by (this variation of) Armstrong's axioms[9]:
  - **Reflexivity**: If Y is a subset of X, then $X \to Y$
  - **Augmentation**: If $X \to Y$, then $\{X, Z\} \to Y$
  - **Transitivity**: If $X \to Y$ and $Y \to Z$, then $X \to Z$

We will assume that the consequence of those axioms always hold ("closure under those rules"), but will generaly not write them explicitely, since they do not carry any new or additional information.

4.4.3.4 Definitions

Remember superkey (not minimal key), key, candidate key, secondary key? We now have a formal definition.

In one particular relation $R(A_1, ..., A_n)$,

- If $\{A_1, ..., A_n\} \to Y$ for all attribute Y, then $\{A_1, ..., A_n\}$ is a superkey.
- If $\{A_1, ..., A_n\}/A_i$ is not a superkey anymore for all A_i, then $\{A_1, ..., A_n\}$ is a key.
- We will often discard candidate keys and focus on one primary key.
- If A_i is a member of some candidate key of R, it is a **prime attribute** of R. It is a **non-prime attribute** otherwise.

Given a FD $\{A_1, ..., A_n\} \to Y$,

- It is a **full functional dependency** if for all A_i, $\{A_1, ..., A_n\}/A_i \to Y$, does not hold.
- It is a **partial dependency** otherwise.

A FD : $X \to Y$ is a **transivive dependency** if there exist a set of attribute B s.t.

- $B \neq X, B \neq Y$
- B is not a candidate key,
- B is not a subset of any candidate key,
- $X \to B$ and $B \to Y$ hold

[9] https://en.wikipedia.org/wiki/Armstrong%27s_axioms

4.4.4 Normal Forms and Keys

There exists multiple normal forms: First, Second, Third, Fourth, Fifth normal form ("X"NF), … Stronger than the Third, there is the Boyce-Codd NF (BCNF), but we will focus on the first three, that are "cumulative": to you satisfy N, a relation have to satisfy $N-1$, $N-2$, etc. The *normal form* of a relation is the highest normal form condition that it meets.

4.4.4.1 Fist Normal Form

4.4.4.1.1 Definition The domain of all attributes must be atomic (simple, indivisible): exclude multi-valued and composite attributes.

Sometimes, additional requirement that every relation has a primary key. We will take this requirement to be part of the definition of 1NF, but some authors take a relation to be in 1NF if it has at least candidate keys (i.e., multiple possible keys, but no primary key, which makes their definition more general, cf. (Elmasri and Navathe 2015, 14.4.1)). Hence, we will always assume that a primary key is given, and it will be underlined.

4.4.4.1.2 Normalization This essentially consists in

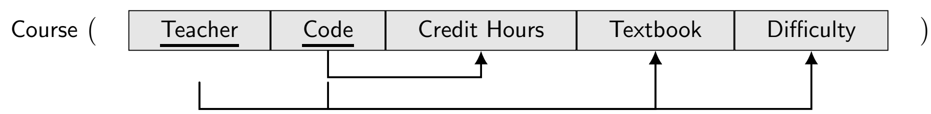
- Picking a primary key,
- Making the complex and multi-valued attributes atomic, following what was done when mapping entity-relationship models to relational models: by either "flattening" the complex attribute (i.e., picking the attributes composing it) or by creating a relation that will allow to store multiple values and linking it to the original relation.

4.4.4.2 Second Normal Form

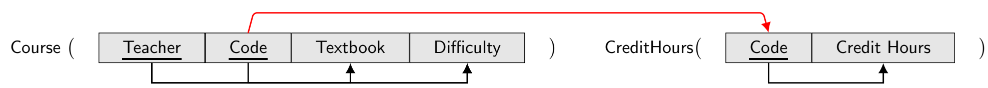
4.4.4.2.1 Definition 1NF + Every non-prime attribute is fully functionnaly dependent on the primary key.

4.4.4.2.2 Normalization For each attribute A of the relation whose primary key is $A_1, \ldots, A_n$:

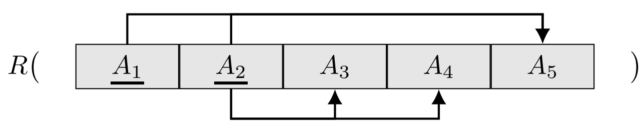
- Is it prime (i.e., is $A \in \{A_1, \ldots, A_n\}$)?
  - Yes $\longrightarrow$ Done.
  - No $\longrightarrow$ Is it partially dependent on the primary key ?
    - No, it is fully dependent on the primary key $\longrightarrow$ Done
    - Yes, it depends only of $\{A'_1, \ldots, A'_k\}$ $\longrightarrow$ Do the following:
      - Create a new relation with A and $\{A'_1, \ldots, A'_k\}$, make $\{A'_1, \ldots, A'_k\}$ the primary key, and "import" all the functional dependencies,
      - Remove A from the original relation, and all the functional dependencies that implied it,
      - Add a foreign key to $\{A'_1, \ldots, A'_k\}$ from their original counterparts in the original relation.



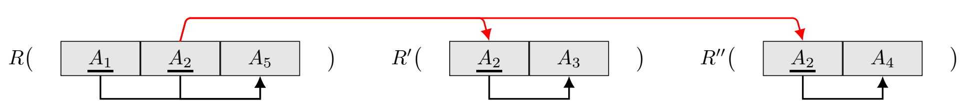
becomes



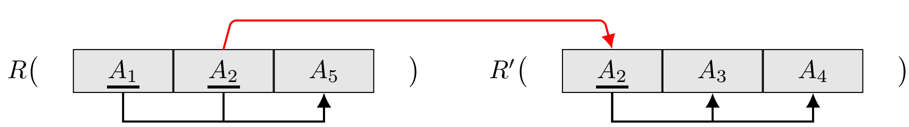
Refinment: note that if more than one attribute depends of the same subset $\{A'_1, \ldots, A'_k\}$, we will create two relations: that is useless, we could have created just one. For instance, considering



applying the algorithm would give (the incorrect, since a foreign key can not refer two attributes in two different tables)



whereas a more subtle algorithm would give



Note that in both cases, all the relations are in Second Normal Form, though (and valid if we ignore the foreign key issue discussed above).

Note also that, sometimes, removing the “original” relation may be preferable: cf. an example in Problem 4.33 (COFFEE relation: primary key and normal form).

Note also that if our primary key is a singleton (a set with only one element), then there is nothing to do, we are in 2NF as soon as we are in 1NF: every functional dependency from a single element is always full!

4.4.4.3 Third Normal Form

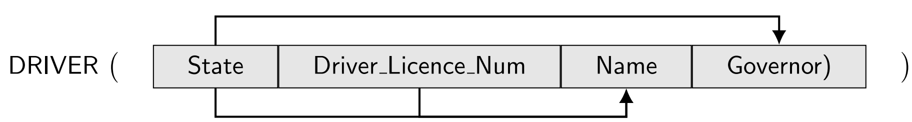
4.4.4.3.1 Definition 2NF + no non-prime attribute is transitively dependent on the primary key.

4.4.4.3.2 Normalization For each attribute A of the relation whose primary key is $A_1, \ldots, A_n$:

- Is it prime (i.e., is $A \in \{A_1, \ldots, A_n\}$)?
  - Yes $\longrightarrow$ Done.
  - No $\longrightarrow$ Is it transitively dependent on the primary key ?
    * No, there is no $\{A'_1, \ldots, A'_k\}$ such that $\{A_1, \ldots, A_n\} \to \{A'_1, \ldots, A'_k\} \to A$ and $\{A'_1, \ldots, A'_k\} \nsubseteq \{A_1, \ldots, A_n\}$ and $A \notin \{A'_1, \ldots, A'_k\} \longrightarrow$ Done
    * Yes, there is such a $\{A'_1, \ldots, A'_m\} \longrightarrow$ Do the following:
      - Create a new relation with A and $\{A'_1, \ldots, A'_k\}$, make $\{A'_1, \ldots, A'_k\}$ the primary key, and import all the functional dependencies,
      - Remove A from the original relation, as well as all the functional dependencies involving it,
      - Add a foreign key from $\{A'_1, \ldots, A'_k\}$ to their original counterparts in the original relation.

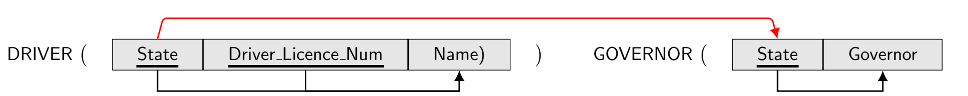
4.4.4.4 Examples

We can have a look at another example:



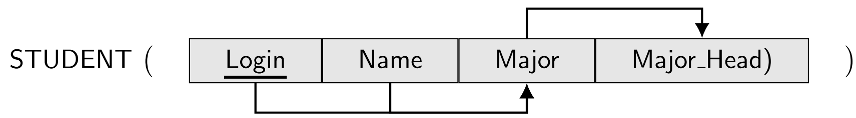
Note that {State, Driver_Licence_Num}, would be a valid primary key for this relation, and that adding it would make it a relation in 1NF.

As we can see, the name "Driver" is somehow counter-intuitive, since the relation also carries information about Governors. This relation is actually not in 2NF, because the FD {State, Driver_Licence_Num} $\longrightarrow$ Governor is not fully functional. A possible way to fix it is to get:

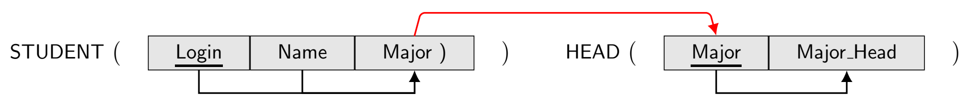


As you can see, the 2NF helped us in separating properly the entities.

An example of a relation that is in 2NF but not in 3NF could be:



As we can see, all the non-prime attributes are fully functionally dependent from Login, which is our primary key. But, obviously, one of this dependecy is transitive, and breaks the 3NF. A way to fix it is:



As we can see, 3NF also helped us in separating properly the entities, in a slightly different way.

In conclusion, we can observe that every FD $X \to Y$ s.t. X is a proper subset of the primary key, or a non-prime attribute, is problematic. 2NF is a guarantee that every entity has its own relation, 3NF is a way to avoid data inconsistency.

4.5 Unified Modeling Language Diagrams

4.5.1 Overview

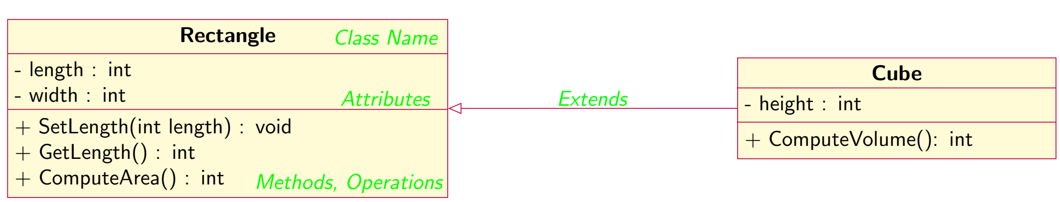
One approach for analysis, design, implementation and deployment of databases and their applications. Databases interact with multiple softwares and users, we need a common language.

Unified **M**odeling **L**anguage[10] is *a standard*:

- Generic
- Language-independent
- Platform-independent

Wide, powerful, but also intimidating.

You know UML from object-oriented programming language:



[10] http://uml.org

That is an example of a class diagram (with class name, attributes and operators, as well as a particular way to represent that a class extends another) , there are other types of diagrams, they are not unrelated! For instance, using communication diagrams, deployment diagrams, and state chart diagrams, you can collect the requirements needed to draw a class diagram! They each offer a viewpoint on a software that will help you in making sure the various pieces will fit together: it is a tool commonly used in software engineering, and useful in database design.

4.5.2 Types of Diagrams

There are 14 different types of diagrams, divided between two categories: structural and behavioral.

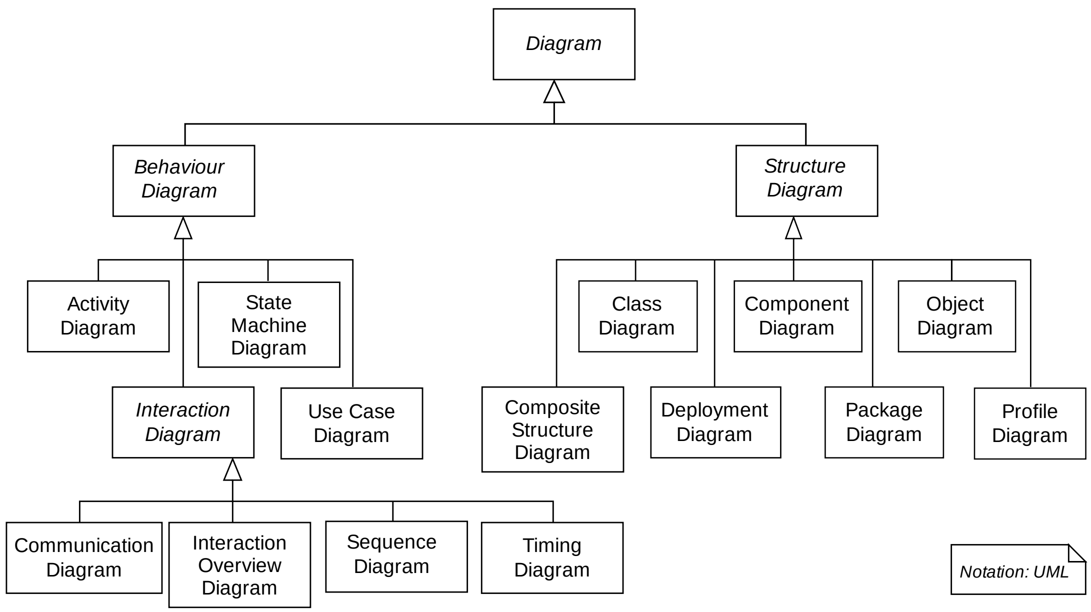


Figure 4.2: UML Diagram Hierarchie

(Source: https://commons.wikimedia.org/wiki/File:UML_diagrams_overview.svg)

4.5.2.1 Structural UML Diagrams

They describe structural, or static, relationships between objects, softwares.

- **Class diagram** describes static structures: classes, interfaces, collaborations, dependencies, generalizations, etc. We can represent conceptual data base schema with them!
- **Object diagram**, a.k.a. instance diagram, represents the static view of a system at a particular time. You can think of a “freeze” of a program, to be able to observe the value of the variables and the objects (or instances) created.
- **Component diagram** describes the organization and the dependencies among software components (e.g., executables, files, libraries, etc.), to describe how an arbitrary large software system is split into pieces.

- **Deployment diagram** is the description of the physical deployment of artifacts (i.e., software components) on nodes (i.e., hardware). If your program runs on a local computer, fetching data from the Internet, and storing output on a server, you may describe this situation using this sort of diagram.

In this category also exist **Composite structure diagram**, **Package diagram** and **Profile diagram**.

4.5.2.2 Behavioral UML diagrams

They describe the behavioral, or dynamic, relationship, between components.

- **Use case diagram** describes the interaction between the user and the system. Supposedly, it is the privileged tool to communicate with end-users.
- **State machine diagram**, a.k.a., state chart diagram, describes how a system react to external events. You can picture yourself a complex form of finite state automata diagram.
- **Activity diagram** is a flow of control between activities. You may have seen them already, they are supposedly easy to follow:

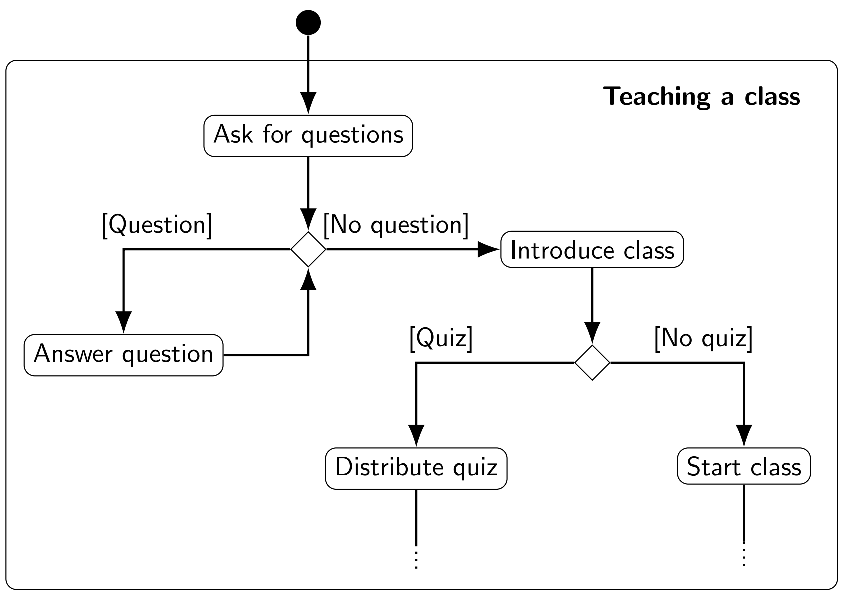


Figure 4.3: Activity Diagram Quiz Example

Then there is the sub-category of "Interaction diagrams":

- **Sequence diagram** describes the interactions between objects over time, the flow of information or messages between objects. It is helpful to grasp the time ordering of the interactions.
- **Communication diagram**, a.k.a., collaboration diagram, describes the interactions between objects as a serie of sequenced messages. It is helpful to grasp the structure of the objects, who is interacting with who.

This sub-category also comprise **Timing diagram** and **Interaction overview diagram**.

4.5.3 Zoom on Classes Diagrams

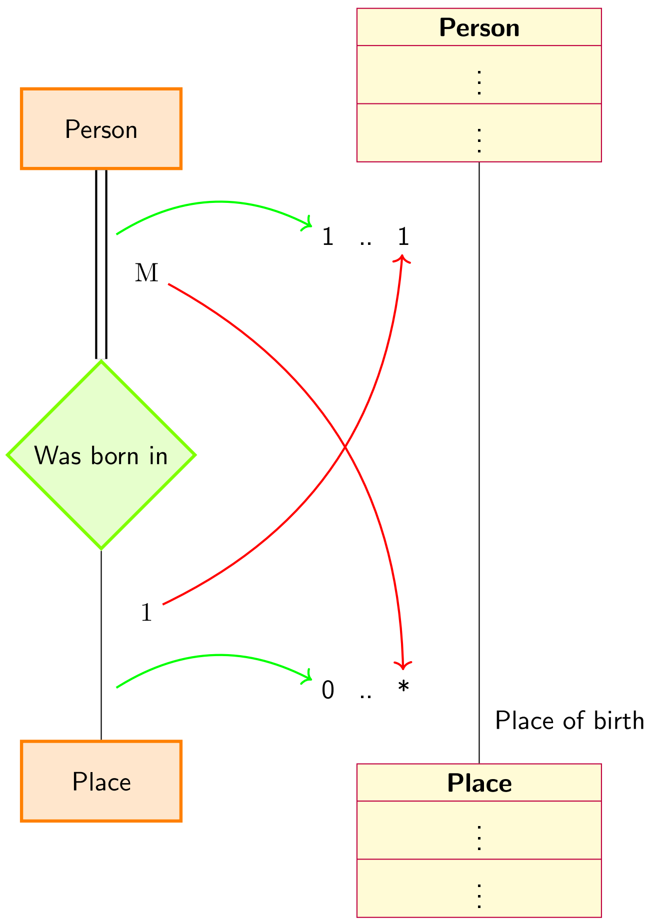
Looking at the "COMPANY conceptual schema in UML class diagram notation", and comparing it with the "ER schema diagram for the COMPANY database" from the textbook, can help you in writing your own "Rosetta Stone" between ER and UML diagram. Let us introduce some UML terminology for the class diagrams.

UML	ER
Class	Entity Type
Class Name	Entity Name
Attributes	Attributes
Operations (or Method)	*Sometimes* Derived Attributes
Association	Relationship Type
Link	Relationship Instance
Multiplicities	Structural Constraint

As well as for ER diagram, the domain (or data type) of the attributes is optional. A composite attribute in a ER diagram can be interpreted as a structured domain in a UML diagram (think of a `struct`), and a multi-valued attribute requires to create a new class.

Associations are, to some extend, more expressive than relationship types:

- **As for relationship types**, they can be recursive (or reflexive), and uses role names to clarify the roles of both parties.
- **As for relationship types** they can have attributes: actually, a whole class can be connected to an association.
- **As for relationship types**, they can express a cardinality constraint on the relation between classes. They are written as `min .. max`, with `*` for "no maximum", and the following shorthands: `*` stands for `0..*` and `1` stands for `1..1`. An association with `1` on one side and `*` on the other (resp. `1` and `1`, `*` and `1`, `*` and `*`) is sometimes called "one-to-many" (resp., "one-to-one", "many-to-one", "many-to-many"). The notation in partially inverted w.r.t. ER diagrams:



Additionally, associations can be “extended”, and they are not the only kind of relationship that can be expressed between two classes.

- **As opposed to the relationship types**, they can be given a direction, indicating that the user should be able to navigate them only in one direction, or in two (which is the default). This is used for security or privacy purposes.
- **As opposed to the relationship types**, they can be *qualified*, implying that a class is not connected to the other class as a whole, but to one particular attribute, called *the qualifier*, or *discriminator*.
- **As opposed to the relationship types**, they are part of a bigger collection of relationships. Other relationships include:
  - *Aggregation*, a.k.a. “is part of” relationship, between a whole object and its component (that have their own existence).
  - *Composition*, which is the particular case of aggregation where the component does not have an existence of their own.
  - *Generalization*, a.k.a. inheritance, that eliminates redundancy and makes a class a *specialization* of another one.
  - A full list can be consulted, e.g., at https://www.ibm.com/support/knowledgecenter/SS8PJ7_9.5.0/com.ibm.xtools.modeler.doc/topics/rreltyp.html.

Qualified associations can be used for weak entities, but not only.

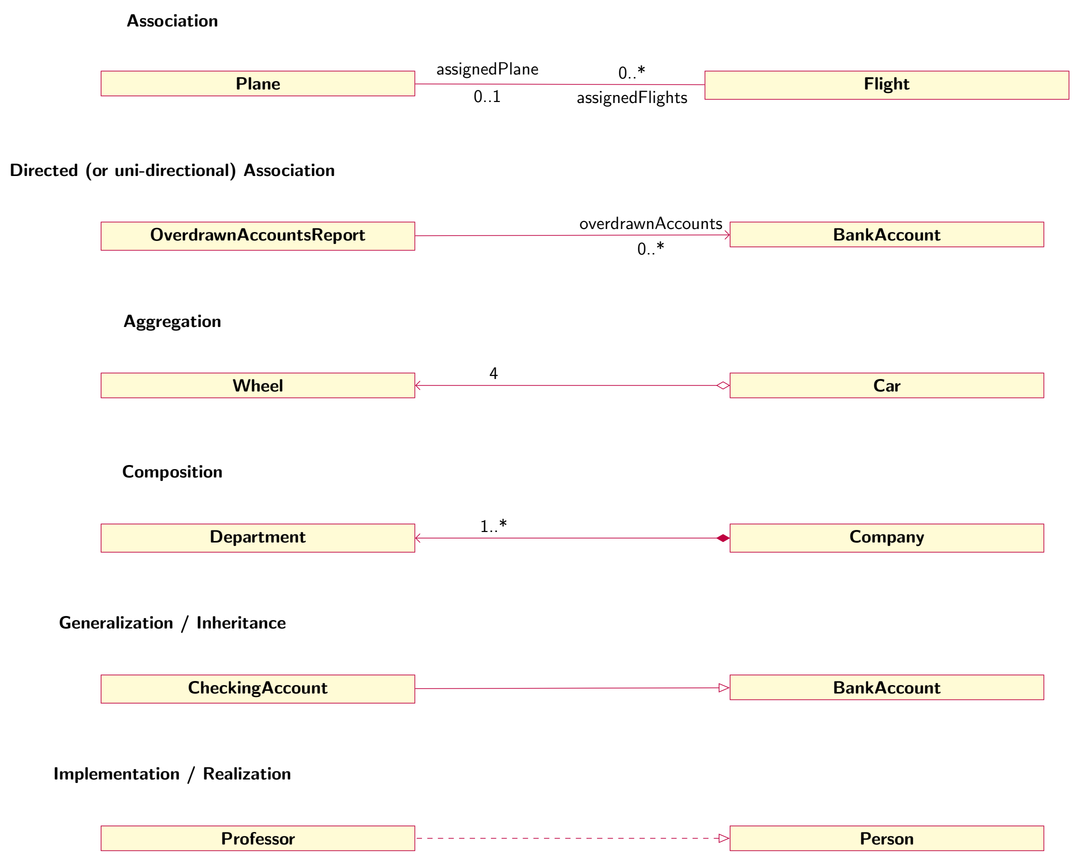


Figure 4.4: Class Diagram Relationships

Some of those subtleties depend on your need, and are subjective, but are important tool to design properly a database, and relieving the programmer from the burden of figuring out many details.

Exercises

Exercise 4.1 Name the three high-level models we will be learning about in this class (expand the acronyms).

Exercise 4.2 What could be the decomposition of an attribute used to store an email address? When could that be useful?

Exercise 4.3 What would be the benefit of having a composite attribute "Phone" made of two attributes (Number and Description) being multi-valued? Answer this question, and draw the resulting attribute.

Exercise 4.4 Draw the ER diagram for a "COMPUTER" entity that has one multivalued attribute "Operating_System", a composite attribute "Devices" (decomposed into "Keyboard" and "Mouse") and an "ID" key attribute.

Exercise 4.5 Draw the ER diagram for a "CELLPHONE" entity that has a composite attribute "Plan" (decomposed into "Carrier" and "Price"), an "MIN" (Mobile Identification Number) key attribute, and a multi-valued "App_Installed" attribute.

Exercise 4.6 Name one difference between a primary key in the relational model and a key attribute in the ER model.

Exercise 4.7 What is a derived attribute? Give two examples and justify them.

Exercise 4.8 Invent an entity type with at least one composite attribute and one atomic attribute, but no multi-valued attributes. Identify a possible key attribute and draw the entity type you obtained using the conventions we used in class.

Exercise 4.9 What is the degree of a relationship type?

Exercise 4.10 What is a self-referencing, or recursive, relationship type? Give two examples.

Exercise 4.11 What does it mean for a binary relationship type "Owner" between entity types "Person" and "Computer" to have a cardinality ratio of $M : N$?

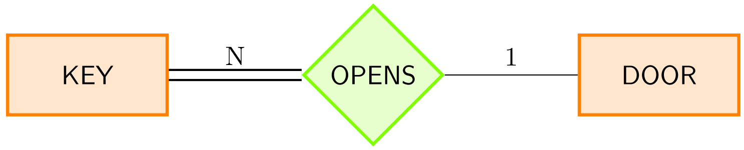
Exercise 4.12 What are the two possible structural constraints on a relationship type?

Exercise 4.13 Draw the diagram for a "VideoGame" entity that would allow to store the name of the game, the supported platform(s) and the release date. Then, add a recursive relationship on that entity called "Is the sequel of" and specify all the constraints.

Exercise 4.14 Draw a diagram to represent a relationship type R between two entities types A and B such that:

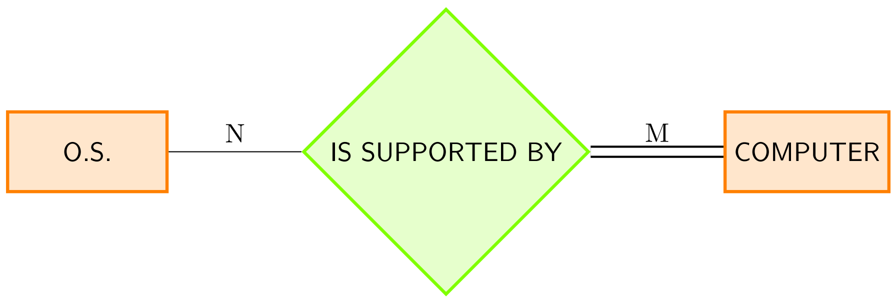
- An entity in A may or may not be in relationship R with an entity in B.
- An entity in B must be in relationship R with an entity in A.
- An entity in A can be in relationship R with at most one entity in B.
- An entity in B can be in relationship R with any number of entities in A.

Exercise 4.15 Express the constraints represented in the following diagram in plain English.



Exercise 4.16 What does it mean for a binary relationship type "is the Chair of" between entity types "Professor" and "Department" to have a cardinality ratio of 1:N? Would it make sense to be have a total participation constraint on one side, and if yes, on which side?

Exercise 4.17 Express the constraints represented in the following diagram in plain English.



Exercise 4.18 For the following binary relationships, suggest cardinality ratios based on the common-sense meaning of the entity types.

Entity 1	Cardinality Ratio	Entity 2
STUDENT	:	MAJOR
CAR	:	TAG
INSTRUCTOR	:	LECTURE
INSTRUCTOR	:	OFFICE
COMPUTER	:	OPERATING_SYSTEM

Exercise 4.19 Give an example of a binary relationship type of cardinality $1 : N$.

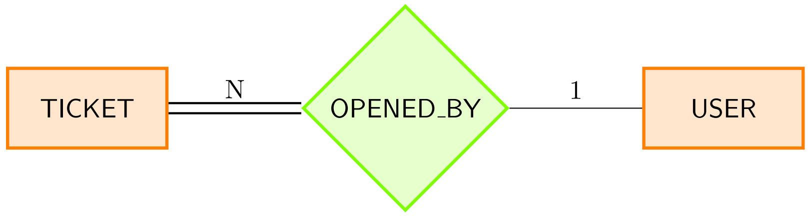
Exercise 4.20 Give an example of a binary relationship type of cardinality $N : 1$ and draw the corresponding diagram (you do not have to include details on the participating entity types).

Exercise 4.21 Draw an ER diagram with a single entity type, with two stored attributes, and one derived attribute. In your answer, it should be clear that the value for the derived attribute can always be obtained from the value(s) of the other attribute(s).

Exercise 4.22 Draw an ER diagram expressing the total participation of an entity type "BURGER" in a binary relation "CONTAINS" with an entity type "INGREDIENT". What would be the cardinality ratio of such a relation?

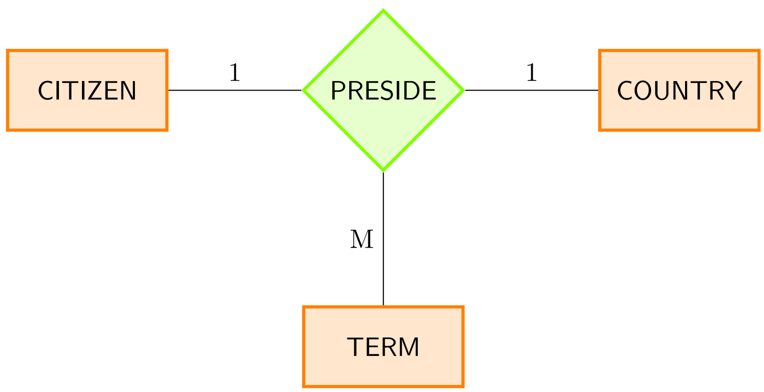
Exercise 4.23 Under what condition(s) can an attribute of a binary relationship type be migrated to become an attribute of one of the participating entity types?

Exercise 4.24 Consider the following diagram:



1. Express the constraints represented in the diagram in plain English.
2. Imagine there is a "FromIP" attribute on the "OPENED_BY" relationship that stores the IP used by the user to open the ticket. Could you migrate the attribute to one of the entity? Explain how you would do it, or why it is impossible.

Exercise 4.25 Consider the following diagram:



1. Express the constraints about the maximums represented in the diagram in plain English.
2. Briefly explain why there are no total participation constraints on this diagram.

Exercise 4.26 Suppose a "PRODUCES" relationship with an attribute "Amount" exists between a "PRODUCER" entity type and a "MOVIE" entity type, with ratio $1 : M$. Migrate the "Amount" attribute to one of the entity types and draw the resulting diagram.

Exercise 4.27 Suppose a "MEMBERSHIP relationship with an attribute"Level" (e.g., "silver", "platinium", etc.) exists between a "PERSON" entity type and a "CLUB" entity type, with ratio $M : 1$. Migrate the "Level" attribute to one of the entity types and draw the resulting diagram.

Exercise 4.28 Assume with have three entity types, "Lecture Notes", "Class" and "Professor."

1. Draw a diagram of a ternary relationship between the three entities.
2. Draw a diagram that has two binary relationships from one of the three entities to the other two entities.

3. Come up with a question that could be answered using one model but not the other from the previous steps (specify which relationship would be able to answer your question).

You can specify role names in your diagrams for added clarity, and remember to list **all** the constraints.

Exercise 4.29 Can we always replace a ternary relationship with three binary relationships? Give an example.

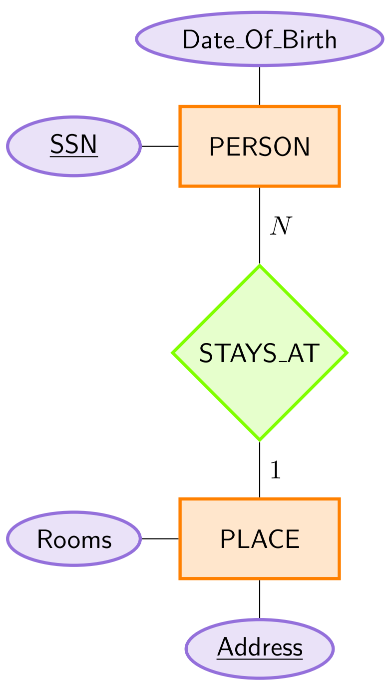
Exercise 4.30 What is the difference between an entity type and a weak entity type?

Exercise 4.31 What is a partial key?

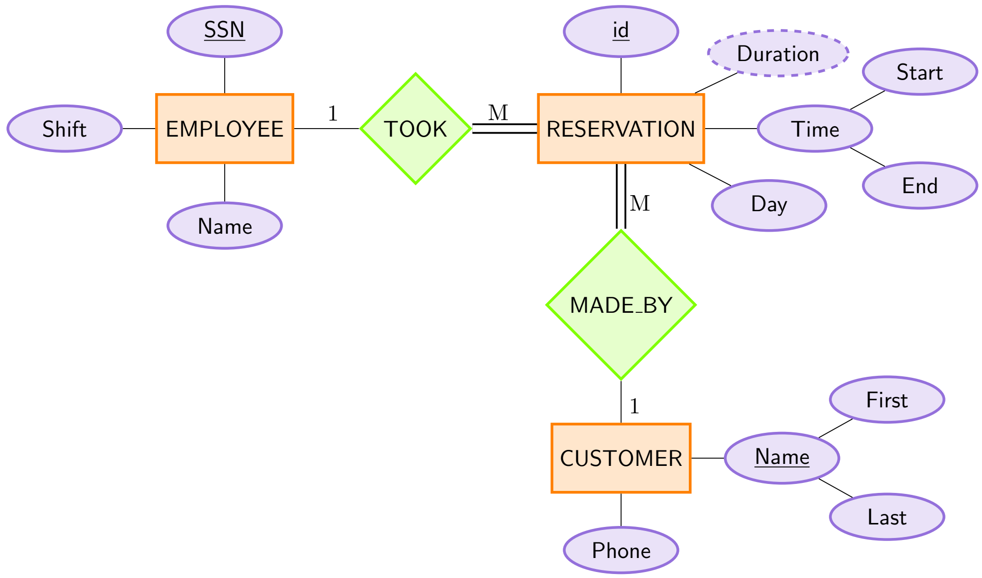
Exercise 4.32 Why do weak entity type have a total participation constraint?

Exercise 4.33 Invent a weak entity type, its identifying (owner) entity type and the identifiying (or supporting) relationship. Both entities should have (partial) key, and each should have at least one composite attribute.

Exercise 4.34 Convert the following ER diagram into a relational model:



Exercise 4.35 Convert the following ER diagram into a relational model:



Exercise 4.36 What is insertion anomaly? Give an example.

Exercise 4.37 What is deletion anomaly? Is it a desirable feature?

Exercise 4.38 Why should we avoid attributes whose value will often be **NULL**? Can the usage of **NULL** be completely avoided?

Exercise 4.39 Consider the following relation:

PROF(<u>SSN,</u> Name, Department, Bike_brand)

Why is it a poor design to have a "Bike_brand" attribute in such a relation? How should we store this information?

Exercise 4.40 Consider the following relation:

STUDENT(<u>SSN,</u> Name, ..., Sibling_On_Campus)

Why is it a poor design to have a "Sibling_On_Campus" attribute in such a relation? How should we store this information?

Exercise 4.41 Consider the following relational database schema:

STUDENT(<u>Login,</u> Name, ..., Major, Major_Head)
DEPARTMENT(<u>Code,</u> Name, Major_Head)

Assuming that "Major" is a foreign key referencing "DEPARTMENT.Code", what is the problem with that schema? How could you address it?

Exercise 4.42 Why can we not infer a functional dependency automatically from a particular relation state?

Exercise 4.43 Consider the relation $R(A, B, C, D, E, F)$ and the following functional dependencies:

1. $F \to \{D, C\}, D \to \{B, E\}, \{B, E\} \to A$
2. $\{A, B\} \to \{C, D\}, \{B, E\} \to F$
3. $A \to \{C, D\}, E \to F, D \to B$

For each set of functional dependency, give a key for R. We want a key, so it has to be minimal.

Exercise 4.44 Consider the relation $R(A, B, C, D, E, F)$ and the following functional dependencies:

$A \to \{D, E\}, D \to \{B, F\}, \{B, E\} \to A, \{A, C\} \to \{B, D, F\}, A \to F$

Answer the following:

1. How many candidate keys is there? List them.
2. How many transitive dependencies can you find? Give them and justify them.

Exercise 4.45 What is a composite attribute in a ER diagram? Can a relational schema with composite attribute be in Second Normal Form?

Exercise 4.46 Consider the relation $R(A, B, C, D)$ and answer the following:

1. If $\{A, B\}$ is the only key, is $\{A, B\} \to \{C, D\}, \{B, C\} \to D$ a 2NF? List the nonprime attributes and justify.
2. If $\{A, B, C\}$ is the only key, is $A \to \{B, D\}, \{A, B, C\} \to D$ a 2NF? List the nonprime attributes and justify.

Exercise 4.47 Consider the relation $R(A, B, C, D, E, F)$ with candidate keys $\{A, B\}$ and C. Remember that, in all generality, to be a prime attribute, you just need to be part of *a possible* candidate key. Answer the following:

1. What are the prime attributes in R?
2. Is $\{C, D\} \to E$ a fully functional dependency?
3. Write a set of functional dependencies containing at least one transitive depency, and justify your answer.

Exercise 4.48 Consider the relation $R(A, B, C, D, E)$ and the following functional dependencies:

1. $C \to D, \{C, B\} \to A, A \to \{B, C, D\}, B \to E$
2. $A \to \{C, D\}, C \to B, D \to E, \{E, C\} \to A$
3. $\{A, B\} \to D, D \to \{B, C\}, E \to C$

For each one, give one candidate key for R.

Exercise 4.49 Consider the relation $R(A, B, C, D, E)$ and answer the following:

1. If $\{A, B\}$ is the primary key, is $B \to E, C \to D$ a 2NF? List the nonprime attributes and justify.
2. If $\{A\}$ is the primary key, is $B \to C, B \to D$ a 2NF? List the nonprime attributes and justify.

Exercise 4.50 Consider the relation $R(A, B, C, D, E, F)$, and let $\{B, D\}$ be the primary key, and have additionnaly the functional dependencies $\{A, D\} \to E, C \to F$. This relation is not in 3NF, can you tell why?

Exercise 4.51 Consider the relation $R(A, B, C, D)$ and answer the following:

1. If A is the only key, is $A \to \{B, C, D\}, \{A, B\} \to C, \{B, C\} \to D$ a 3NF? List the nonprime attributes and justify.

2. If B is the only key, is $B \rightarrow \{A, C, D\}, A \rightarrow \{C, D\}, \{A, C\} \rightarrow D$ a 3NF? List the nonprime attributes and justify.

Exercise 4.52 Consider the relation $R(A, B, C, D, E)$ and the functional dependencies $\{A, B\} \rightarrow C, B \rightarrow D, C \rightarrow E$. Answer the following:

1. A by itself is not a primary key, but what is the only key that contains A?
2. List the non-prime attributes.
3. This relation is not in 2NF: what transformation can you operate to obtain a 2NF?
4. One of the relation you obtained at the previous step is likely not to be in 3NF. Can you normalize it? If yes, how?

Exercise 4.53 What are the two different categories of UML diagram?

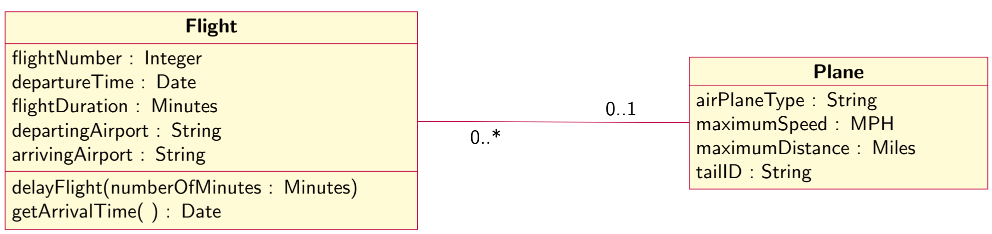
Exercise 4.54 Can a `C++` developer working on Linux and a `Java` developer working on MacOS use the same class diagram as a basis to write their programs? Justify your answer.

Exercise 4.55 What kind of diagram should we use if we want to ...

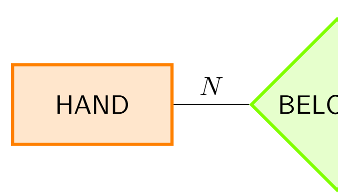
1. describe the functional behavior of the system as seen by the user?
2. capture the flow of messages in a software?
3. represent the workflow of actions of an user?

Exercise 4.56 Name two reasons why one would want to use a UML class diagram over an ER diagram to represent a conceptual schema.

Exercise 4.57 Consider the following diagram:



Give the number of attributes for both classes, and suggest two operations for the class that does not have any. Discuss the multiplicities: why did the designer picked those values?

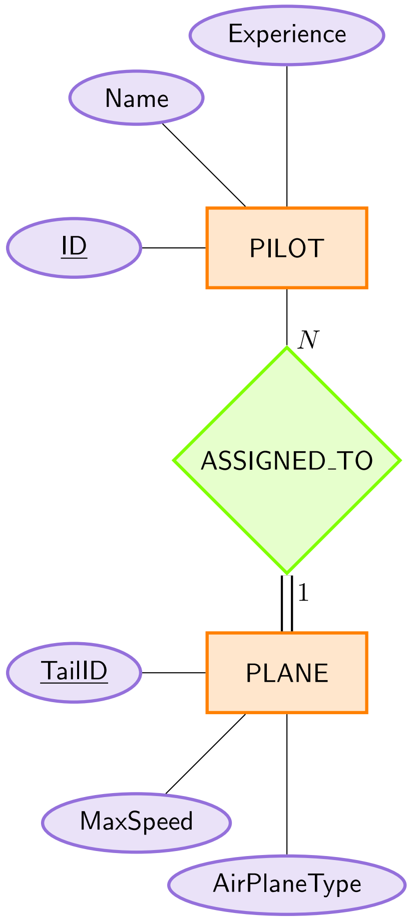


Exercise 4.58 Convert the following ER diagram to a UML class diagram.

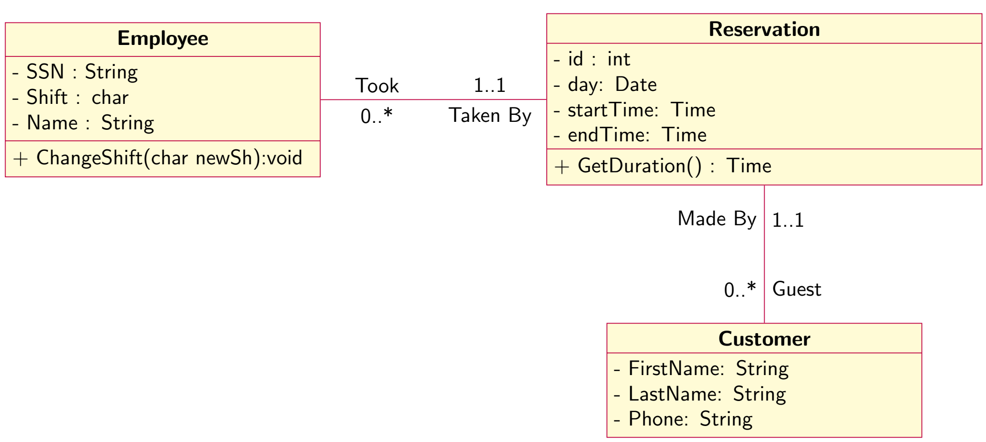
Exercise 4.59 Briefly explain the difference between an aggregation and a composition association.

Exercise 4.60 How is generalization (or inheritance) represented in a UML class diagram? Why is such a concept useful?

Exercise 4.61 Convert the following ER diagram into a UML class diagram:



Exercise 4.62 Convert the following UML class diagram into an ER diagram:

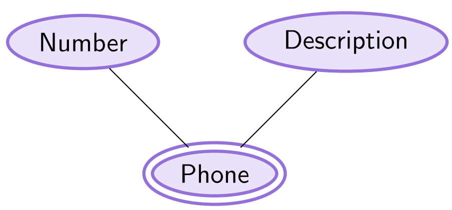


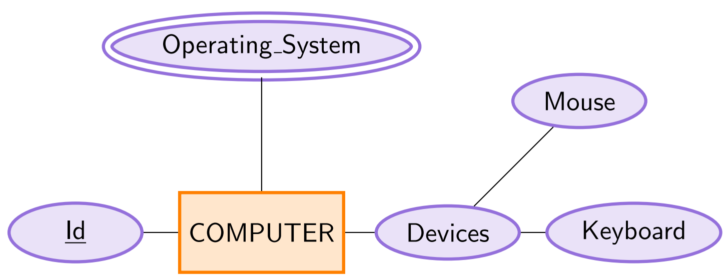
Solution to Exercises

Solution 4.1 The three high-level models we will be learning about are the **U**nified **M**odeling **L**anguage, **E**ntity **R**elationship, and **E**nhanced **E**ntity–**R**elationship models.

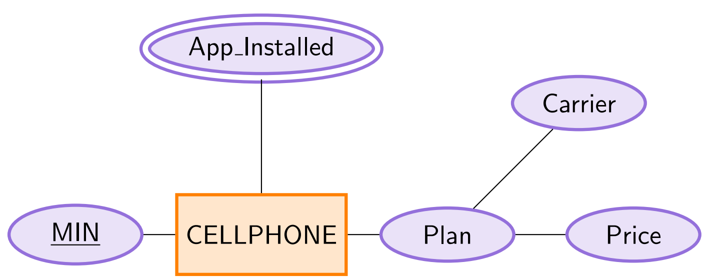
Solution 4.2 A useful decomposition of an email address attribute could be: the username part before the `@` sign, and the domain part afterwards (that could even be sub-divided between the domain name and its top-level domain). It might be useful to have statistics about the domains of the users or to sort the usernames by length, etc.

Solution 4.3 Having a "Phone" attribute being multi-valued would allow to store multiple phone numbers for the same entity. Typically, one would want to store a pair (Number, Description) for their office phone, their cell, etc. The resulting attribute would be drawn as follows:





Solution 4.4



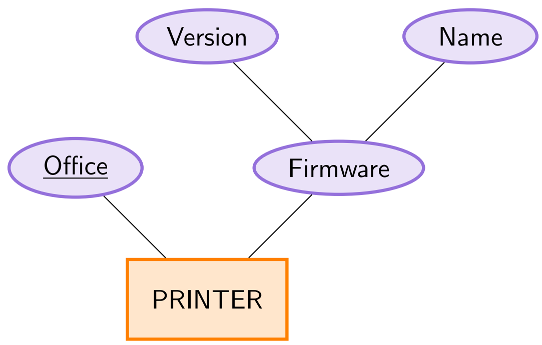
Solution 4.5

Solution 4.6 There can be more than one key in the ER model, but it has to be made of a single attribute, whereas a primary key can be made of multiple attributes.

Solution 4.7 A derived attribute is an attribute whose value can be determined by the value of other attributes. For instance:

- The value of an "Age" attribute could be determined from the value of an "Date of birth" attribute and the current day.

- The value of a "State" attribute can be determined from the value of a "Zip code" attribute.
- The value of a "Body Mass Index" attribute could be calculated from the values of height and weight attributes.
- The value of an "Initials" attribute could be determined using the values of the "First Name", "Middle Name", and "Last Name" attributes.



Solution 4.8

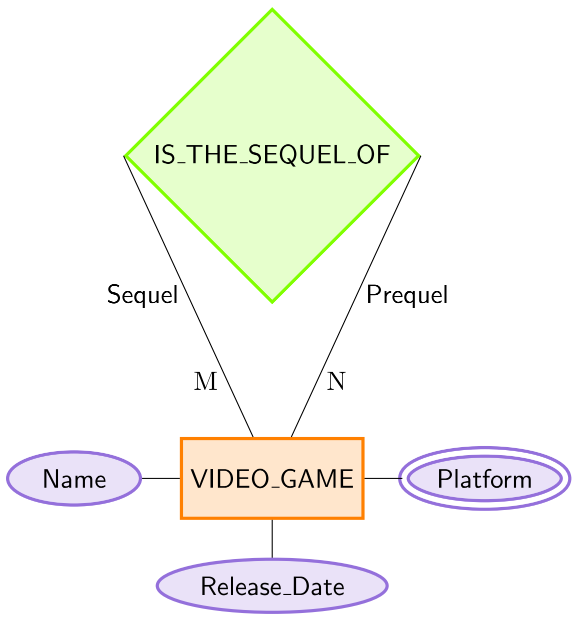
Solution 4.9 The degree of a realationship type is the number of its participating entity types.

Solution 4.10 A self-referencing relationship type is where the same entity type participates more than once. On a SEATS entity type, it would be an attribute like "is to the left of" or on a PERSONS entity type, it would be and attribute like "is married to".

Solution 4.11 The cardinality ratio on the binary relationship type "Owner" between the entity types "Person" and "Computer" means that a person can own multiple computers, and a computer can have multiple owners.

Solution 4.12 The two possible structural constraints on a relationship type are the cardinality ratio and participation constraints.

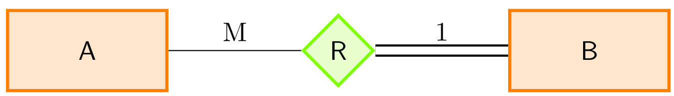
Solution 4.13 We would obtain the following diagram:



Note that the "M / N" part could be discussed: having 1 instead of M would mean that a videogame that is a sequel is the sequel of at most one videogame. This could make sense, but would forbid for instance to register "Battletoads & Double Dragon - The Ultimate Team" as the sequel of both Double Dragon and Battletoads. Having 1 instead of N would mean that every videogame has at most one sequel: it would prevent from registering both "Super Mario Land" and "Super Mario World" as the sequels of "Super Mario Bros. 3".

However, note that all the participation constraints are partial: having a total participation constraint would mean (on the left side) that every game is a sequel, or (on the right side) that every game have at least one sequel, two statements that are obviously wrong.

Solution 4.14 We would obtain the following diagram:



Solution 4.15 A key opens only one door, and every key must open at least one door. A door can be opened by multiple keys, and some doors may not be opened by any key (think of doors that do not have a lock).

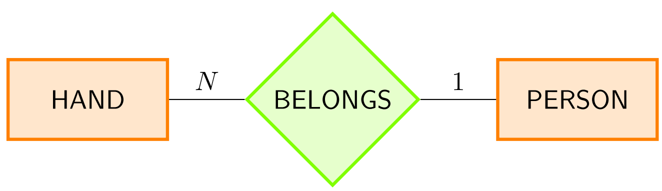
Solution 4.16 The binary relation type "is the Chair of" with a cardinality ratio of 1:N between entity types "Professor" and "Department" means that a department can have at most one professor as its chair, but that a professor can be the chair of multiple departments. It could make sense to require that every department has a chair, hence writing a double line between the Department entity and the "is the Chair of" relationship, but it would not make sense to have a total participation constraint on the side of the professor (which would mean that every professor *has to* be the chair of a department).

Solution 4.17 An operating system may be supported by many computers, but it is also possible that no computer supports it (think of an operating system in development, or developed for embeded devices). A computer must support at least one operating system and can support multiple operating systems.

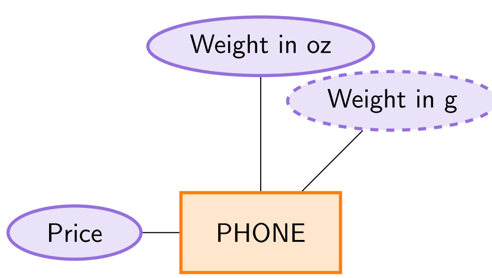
Entity 1	Cardinality Ratio	Entity 2	Explanation
STUDENT	$N:1$	MAJOR	"A student has one major, but multiple students can have the same major"
CAR	$1:1$	TAG	"A car has exactly one tag, a tag belongs to one particular car."
INSTRUCTOR	$1:N$	LECTURE	"An instructor can teach multiple lecture, but a lecture is taught by only one person."
INSTRUCTOR	$1:N$	OFFICE	"An instructor can have multiple office, but an office belongs to only one instructor"
COMPUTER	$M:N$	OPERATING_SYSTEM	"A computer can have multiple operating system, the same operating system can be installed on more than one computer."

Solution 4.18 Some of these choices are debatable (typically, almost any combination seems reasonable for the INSTRUCTOR : OFFICE relation).

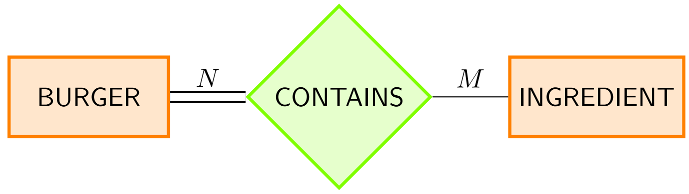
Solution 4.19 A binary of relationship of SUPERVISOR as a recursive relationship on EMPLOYEE.



Solution 4.20



Solution 4.21



Solution 4.22

Solution 4.23 An attribute of a binary relationship type can be migrated to one of the participating entity types when the cardinality ratio is $1 : N$, $1 : 1$, or $N : 1$. It can be migrated "to the N side" or, if there is no N side, to either side. Note that for n-ary relationships, at least one ratio needs to be 1 for the attribute to be allowed to migrate (and "to the N side", or, if there is no N side, to any side).

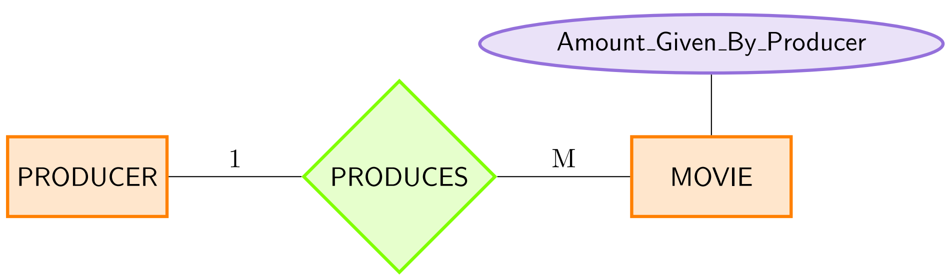
Solution 4.24

1. A ticket must be opened by a exactly one user, and an user can open any number of tickets (including 0).
2. We could migrate the "FromIP" attribute to the "TICKET" entity: the intuition is that while the IP adress of a user can evolve through time, the IP used at the time of creation of the ticket is unique to the ticket, and hence can become an attribute of the ticket.

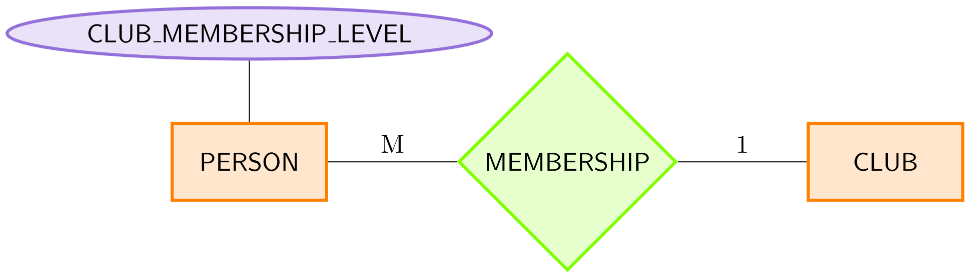
Solution 4.25

1. A citizen can be the president of at most one country during a given term. A country can have only one citizen as their president during a given term. A citizen can be the president of the same country over multiple terms.
2. Some citizen will never be president, some country may not have presidents (think royalty), some terms may not relate to any presidency (e.g. "3rd century BC").

Solution 4.26 We could have the following:

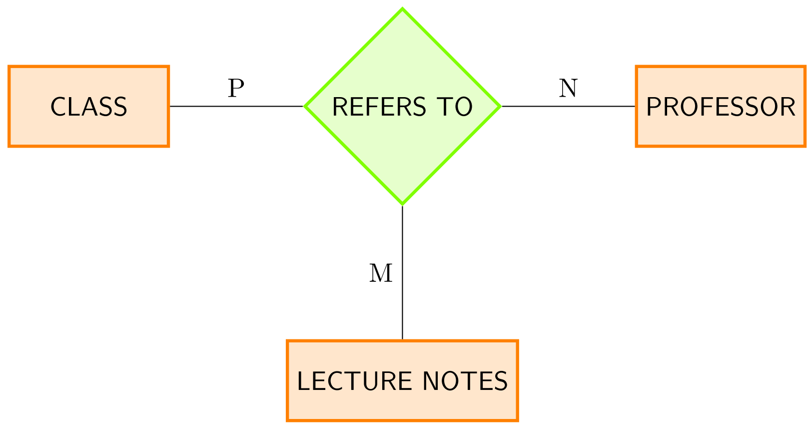


Solution 4.27 We could have the following:

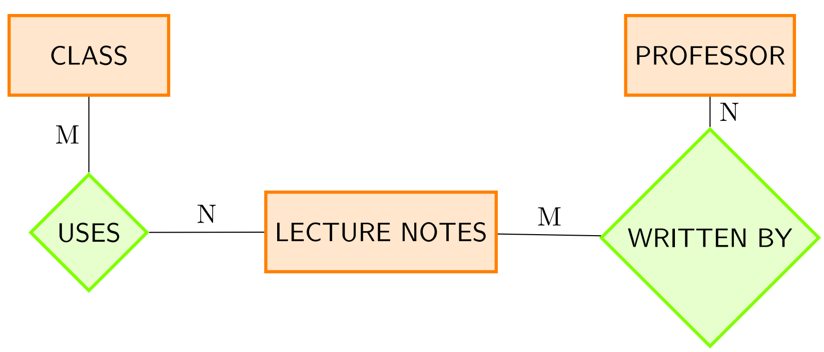


Solution 4.28

1. A possible example of ternary relationship is:



2. One example of two binary relationships could be:



3. A question like

   "Who wrote the lecture notes X?"

   could be answered with the binary relationships but not the ternary. Conversely, a question like

"What are the lecture notes refered to by Prof. X in their class Y?"

could not be answered using the binary relationships (since we do not know what classes are taught by Prof. X).

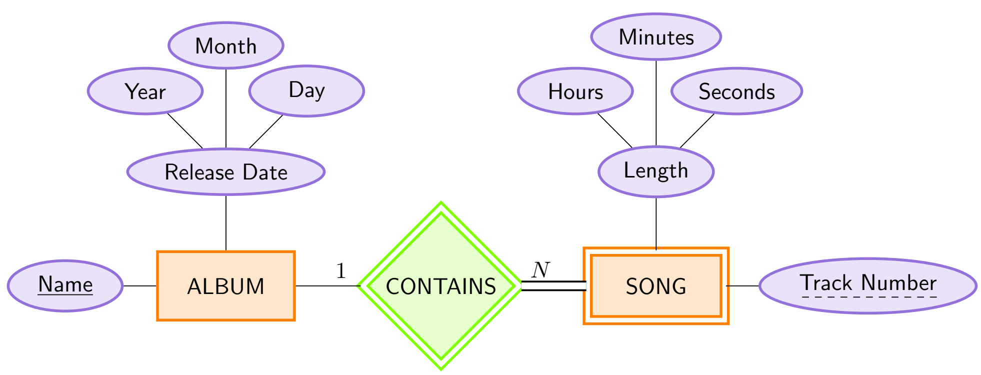
Solution 4.29 No, a ternary relationship cannot always be replaced by three binary relationship. For instance, if I have a "Travelling to" relationship between a "Person", a "City" and a "Transport mode", to represent the fact that a person is travelling to a city using a particular mode of transportation, there is no way I can convey the same information using binary relationships.

Solution 4.30 The weak entity type does not have a key attribute, it cannot be distinguised from the other weak entities based on a single attribute, for that we also need to know its relationship to some other entity type.

Solution 4.31 For a weak entity attribute, it is the attribute that can uniquely identify weak entites that are related to the same owner entity.

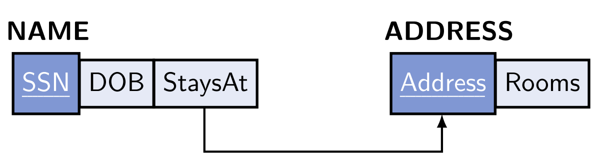
Solution 4.32 Otherwise, we could not identify entities in it without owner entity.

Solution 4.33 A possible solution is:



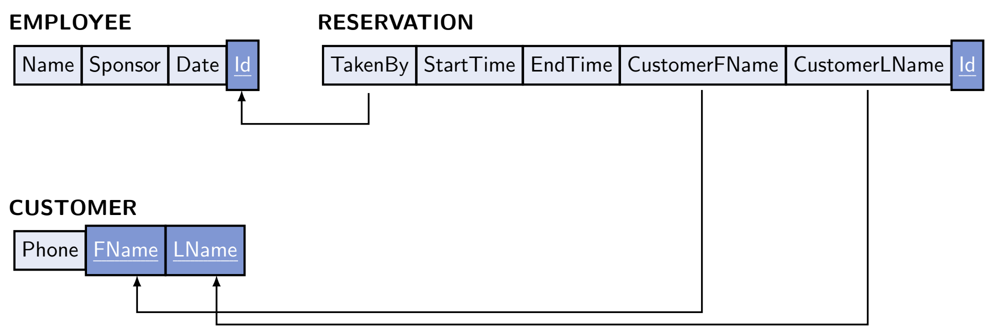
Note that the two composite attributes are "generic", in the sense that you can re-use those examples easily.

Solution 4.34 A possible option is:



Note that "Stays_At" could also be a separate relation, with two attributes, "Address" and "Person", linked to respectively PLACE.Address and PERSON.SSN, and both being the primary key of the relation.

Solution 4.35 A possible option is:



Note that to more faithfully represent the total participation constraints, one could add **NOT NULL** attributes to TakenBy, CustomerFName and CustomerLName in RESERVATION.

Solution 4.36 When you have to invent a primary key or add a lot of **NULL** value to be able to add a tuple. I want to add a room in my DB, but the only place where rooms are listed are as an attribute on a Instructor table, so I have to “fake” an instructor to add a room.

Solution 4.37 A delete anomaly exists when certain attributes are lost because of the deletion of other attributes. It is not desirable, since it can lead to the loss of information.

Solution 4.38 Because they waste space, they are ambiguous (N/A, or unknown, or not communicated?), and they make querries harder. No, it is necessary sometimes.

Solution 4.39 Because it will be **NULL** most of the time. In a separate relation, e.g. a “BIKE” relation, with two attributes, “Owner” and “Brand”, “Owner” being a foreign key referencing the SSN attribute of PROF.

Solution 4.40 Because it will be **NULL** most of the time, and because students could have more than one sibling on campus. In a separate relation, e.g. in a “EMERGENCY_CONTACT” relation, with two attributes, “Student” (refercing the SSN attribute of STUDENT), and “Contact”. If the emergency contacts are not related to the student, or if we want to preserve the fact that one student is a sibling to another, we can create another relation to store that information.

Solution 4.41 Major_Head will give update anomalies. By putting the Head of the department in the DEPARTMENT relation only, i.e., removing it from STUDENT.

Solution 4.42 Just because a coincidence exists (i.e., “in my data set, no android user is color-blind”) does not mean that it will always be true (i.e., “no color-blind person will ever use android”). Functional dependencies should come from a principled reasoning about the attributes, and not from the observation of the data.

Solution 4.43

1. F
2. $\{A, B, E\}$
3. $\{A, E\}$

Solution 4.44

1. Only one: $\{A, C\}$,

2. $A \to F$ by $A \to D, D \to F$.

Solution 4.45 A composite attribute is an attribute made of multiple attributes, like an “Address” attribute could be composed of the “sub”-attributes “Street”, “City”, “Zip” and “State. A relational schema needs a primary key and to have only atomic domains to be in first normal form, so, no, a relational schema with composite attributes can not be in second normal form.

Solution 4.46

1. Yes. C and D are non prime, and they fully depend on $\{A, B\}$.
2. No. D is the only non prime, and it depends only on A.

Solution 4.47

1. A, B and C.
2. No, because we can remove D,
3. $A \to D$, $D \to E$ and $A \to E$

Solution 4.48

1. $\{B, C\}$, A
2. A, $\{C, E\}$,
3. $\{A, D, E\}$, $\{A, B, E\}$

Solution 4.49

1. No. C, D, E, and E has a partial relation to B
2. Yes. Since the primary key is a singleton, it is obvious.

Solution 4.50 $\{B, D\} \to C \to F$ breaks the 3NF.

Solution 4.51

1. No. B, C and D are non prime, $A \to \{B, C\} \to D$ breaks the 3NF.
2. No. A, B and D are non prime, $B \to \{A, C\} \to D$ breaks the 3NF.

Solution 4.52

1. $\{A, B\}$,
2. C, D, E,
3. $R_1(A, B, C, E)$ and $R_2(B, D)$
4. $R_1(A, B, C)$, $R_2(C, E)$ and $R_3(B, D)$

Solution 4.53 The two different categories of UML diagram are behaviour and structure.

Solution 4.54 Yes, UML diagram is language-independent and platform-independent.

Solution 4.55

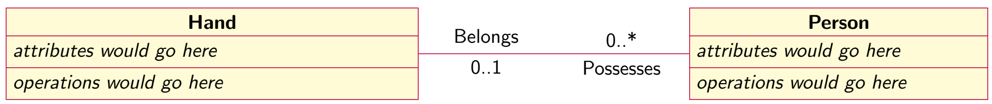
1. Use-case
2. Sequence diagram
3. Activity diagram

Solution 4.56 To use direction for association, to have a common language with someone less knowledgeable of other diagrammatic notations. For the concept of integration.

Solution 4.57 `Flight` has 5 attributes, `Plane` has 4. The `Plane` class could have the operations `getLastFlightNumber() : Integer` and `setMaximumSpeed(MPH) : void`.

For the multiplicities: A flight could not have a plane assigned, and a plane could not be assigned to a flight. A plane can be assigned to multiple (or no) flights, but a flight must have at most one plane (and could have none).

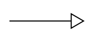
Solution 4.58 The absence of total participation constraint on the left side of the diagram may seem odd: what would be a hand not belonging to a person? Still, we have to accept it: we do not know what the requirements are, or the precise nature of the entities. As far as we know "hand" could refer to a card game, and "person" could refer to players. A straightforward representation of the same diagram as a UML class diagram could be:



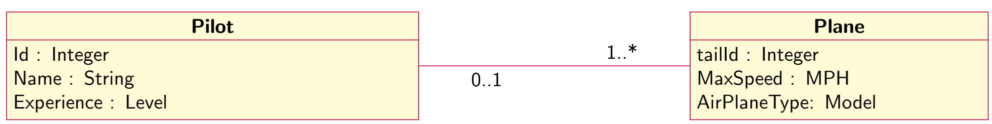
Note that we could convey more information, for instance by using aggregation, or even composition, but, without more information about those entities and this relationship, it may be safer not to make any additional supposition.

Solution 4.59 Aggregation: associated class can have an existence of its own.

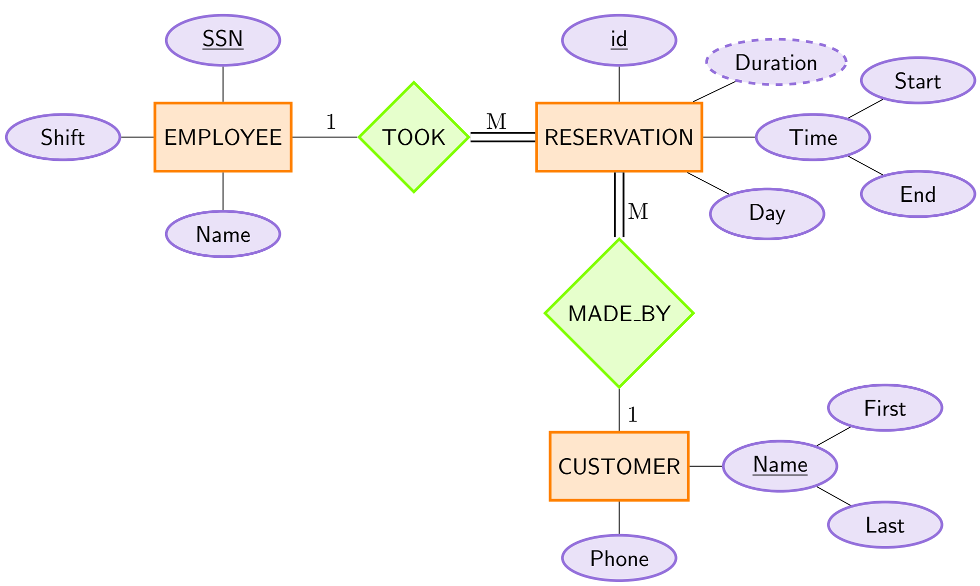
Composition association: class does not exist without the association.



Solution 4.60 Because it avoids redundancy.



Solution 4.61



Solution 4.62

Even though entity type do not *need* a key, it is generally good to include one, and we picked the "obvious" ones (even if `Phone` could have been a good choice for `CUSTOMER` as well).

Problems

Problem 4.1 ***(Design for your professor)*** Your professor designed the following relational model, at some point in his career, to help him organize his exams and the students' exam grades:

Table Name and Attributes	Example of Value
EXAM(Number, Date, Course)	< 1, '2018-02-14', 'CSCI3410'>
PROBLEM(Statement, Points, Length, Exam)	< 'Your professor designed…', 10, '00:10:00', 1>
STUDENT_GRADE(Login, Exam, Grade)	< 'aalyx', 1, 83>

`EXAM.Number`, `PROBLEM.Statement`, `STUDENT_GRADE.Login` and `STUDENT_GRADE.Exam` are all the primary key, and `STUDENT_GRADE.Exam` and `PROBLEM.Exam` are foreign keys that both refer to `EXAM.Number`.

The idea was to have the following design elements:

- The `EXAM` table for storing information about exams.
- The `PROBLEM` table for storing each problem as its' own entry and to associate every problem to an exam.
- The `STUDENT_GRADE` table for storing the grade of one student for one particular exam.

Unfortunately, this design turned out to be terrible.

1. Describe at least one common and interesting situation where this model would fail to fulfill its purpose.
2. Propose a way to correct the particular problem you identified.

Problem 4.2 ***(Reading the MOVIES database ER schema)*** Consider the ER schema for the MOVIES database (inspired from (Elmasri and Navathe 2010, fig. 7.24)):

Where the attributes are omitted, and separate entities are created for actors, producers and directors even if they happen to be the same person (to deal with e.g. pseudonyms or different attributes, like agent or address).

Given the constraints shown in the ER schema, respond to the following statements with *True* or *False*. Justify each answer.

1. There are no actors in this database that have been in no movies.
2. There might be actors who have acted in more than ten movies.
3. Some actors could have done a lead role in multiple movies.
4. A movie can have only a maximum of two lead actors.
5. Every director has to have been an actor in a movie.
6. No producer has ever been an actor.
7. A producer cannot be an actor in some other movie.
8. There could be movies with more than a dozen actors.
9. Producers can be directors as well.
10. A movie can have one director and one producer.

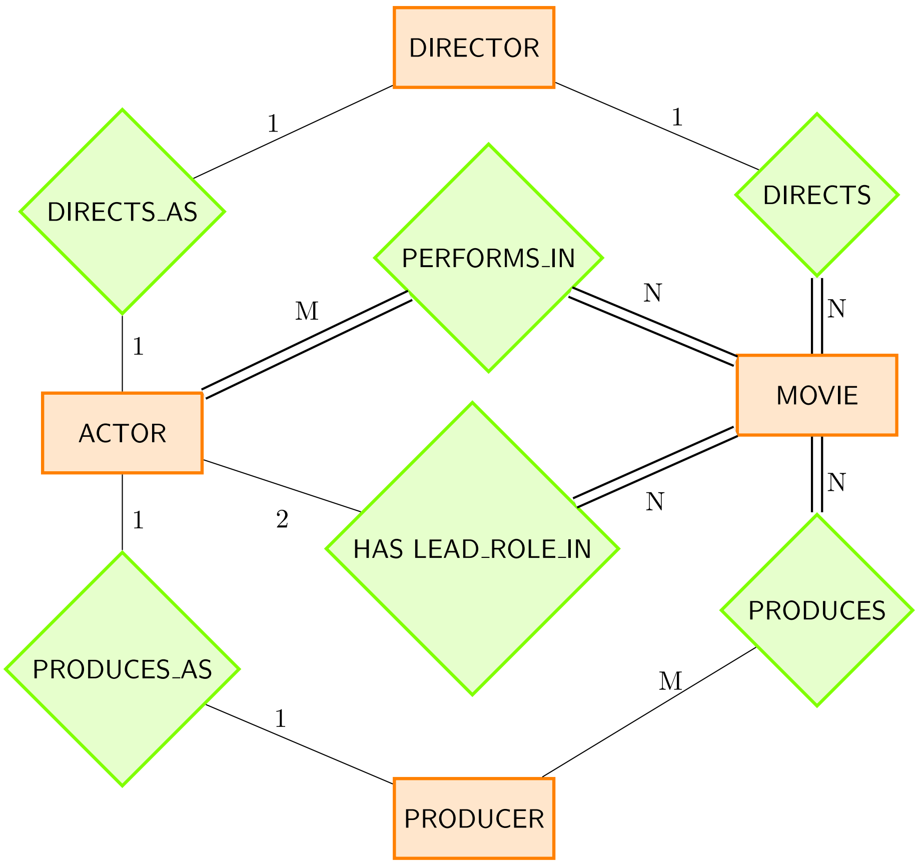


Figure 4.5: Movies Database Example

11. A movie can have one director and several producers.
12. There could be actors who have perfomed a lead role, directed a movie, and produced a movie.
13. It is impossible for a director to play in the movie (s)he directed.

Problem 4.3 ***(ER diagram for car insurance)*** Draw the ER diagram for the following situation:

1. A car insurance company wants to have a database of accidents.
2. An accident involves cars, drivers, and several aspects: the time and location of where it took place, the amount of damages, and a unique report number.
3. A car has a license, a model, a year, and an owner.
4. A driver has an ID, an age, a name, and an address.

One of the interesting choices is: should "accident" be an entity type or a relationship type?

Problem 4.4 ***(ER diagram for job and offers)*** You want to design a database to help you apply for jobs and to compare offers. Every job has a salary range, a title, multiple requirements (like languages known, years of experience, etc.) and was advertised by a company at a particular url. Every company has a physical and numerical address, provides some benefits (assuming they provide the same benefits to all their employees). Sometimes you know one or multiple persons working there, and you want to keep track of their names, role, and (if this is the case) of the job they told you about. Finally, you want to keep track of the offers you received: the job they correspond to, the actual salary offered and the possible starting date.

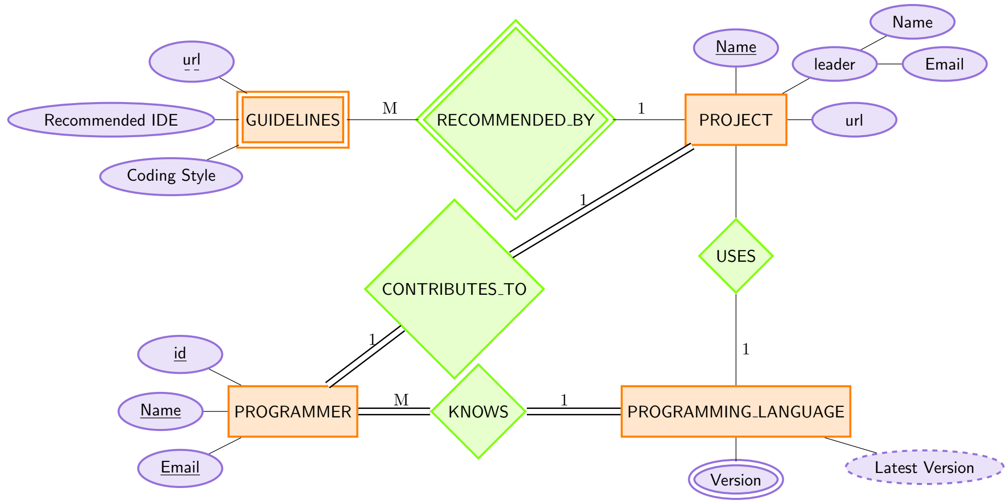
1. Draw the ER diagram for this situation.
2. Add attributes for the key attributes if needed.
3. Specify the cardinality ratios and participation constraints.

Problem 4.5 ***(ER diagram for cellphones)*** Draw an entity-relationship diagram for the following situation.

A sim card have a format (mini-SIM, micro-SIM, etc.) and unique ICCID and IMSI numbers. A cellular network have an ITU region and a frequency band. A phone has one or two IMEI number, can connect to one or multiple cellular networks, and can hold zero, one or two sim cards. A contact (which is made of a name, a phone number and an email) can be stored either in the sim card or in the phone (in which case you can add a picture to the contact). Every phone must have at least one operating system installed on it, and an operating system has a name, a version and licence. Finally, an application (which is made of a name, a version number and a url) may or may not be compatible with a phone / operating system pair.

Problem 4.6 ***(Incorrect ER diagram)*** A company wants to develop a database to keep track of the programmers, projects and programming languages they know of. They are not willing to store guidelines for the sake of it, but believe that if a project requires a particular guideline (like, which IDE to use, what spacing convention they use, etc.), it should be stored somewhere. They want to accommodate the fact that a project can use multiple programming languages (and sometimes even multiple versions of the same language), and keep track of which programmer is leading which project. To ease "match making", they also want to track which programmer is knowledgeable of what programming language. They would also like to store links to the specifications of programming languages, as well as urls of the projects and their guidelines.

They came up with the following ER diagram:



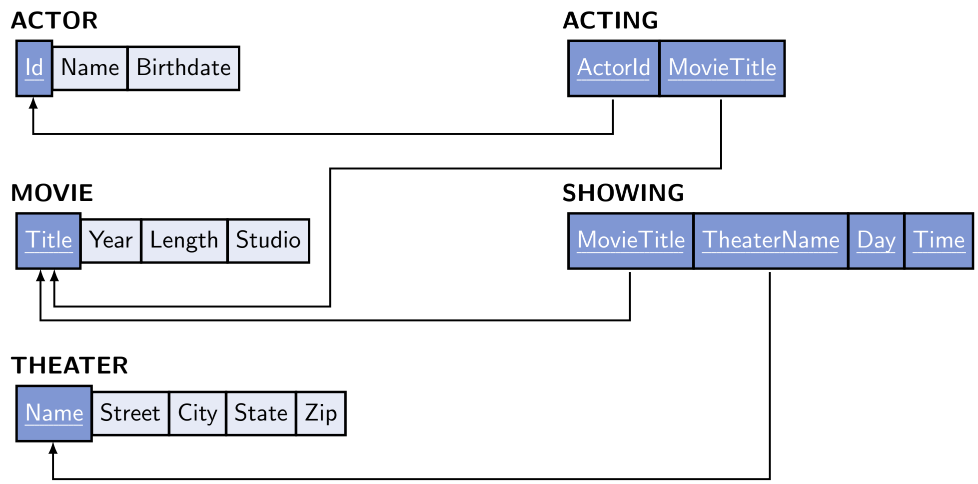
This diagram, to your expert eyes, has multiple flaws, missing constraints, and has some inconsistencies with their requirements. List as many as you can, and suggest improvments or solution when you can think of one.

Problem 4.7 ***(ER diagram for Undergraduate Conference)*** Draw the ER diagram corresponding to the following situation for conferences on undergraduate research:

> Every conference has a name, an edition ("First", "Second", etc.), and it takes place during particular days. Students can submit abstracts (made of a title, multiple keywords and a content) and, if accepted, they will give talks (that have a title and a length) during particular sessions. Note that an abstract can have multiple students as authors, but that a talk is given by exactly one student. A session must have exactly one moderator (who is a Faculty member), multiple judges (that are Faculty members as well), and a time frame. Faculty members have an email, a name, a title, and they can also mentor zero, one or multiple students.

Indicate all the assumptions or choices you are making, but try to make as few assumptions as possible.

Problem 4.8 ***(Reverse engineering by hand)*** Look at the following relational model and "reverse-engineer" it to obtain an ER diagram:



Problem 4.9 ***(Discovering MySQL Workbench)*** In this problem, we will install and explore the basic functionalities of MySQL Workbench, which is a cross-platform, open-source, and free graphical interface for database design.

1. Install MySQL Workbench if needed. Maybe you already included it in the packages to install when you installed MySQL (cf. the instructions to install MySQL on Windows): try to find if this is the case before trying to install it. Otherwise, use your package manager, or download the binaries from https://dev.mysql.com/downloads/workbench/. The installation should be straightforward for all operating systems.
2. Once installed, execute the software. The instructions below were tested for the 6.3.8 version on Debian and 8.0.19 version on Windows. The trouble with GUI-software is that the menus may differ slightly with what you see, but the core tools we will be using should still be there under a similar name, if not the same name.
3. Under the panel "MySQL Connections", you should see your local installation listed as "Local instance 3306." Click on the top-right corner of that box and then on "Edit Connections." Alternatively, click on "Database", then "Manage Connections", and then on "Local instance 3306."
4. Check that all the parameters are correct. Normally, you only have to change the name of the user to "testuser" and leave the rest as it is. Click on "Test the connection" and enter your password (which should be "password") when prompted. If you receive a warning about "Incompatible/nonstandard server version or connection protocol detected", click on "Continue anyway."
5. Now, click on the box "Local instance 3306" and enter your password. A new tab appears in which you can see the list of schemas in the bottom part of the left panel.
6. Click on "Database", then on "Reverse Engineering" (or hit `ctrl` + `r`), then click on "next", enter your password, and click on "next." You should see the list of the schemas stored in your database. Select one (any one, we are just exploring the functionalities

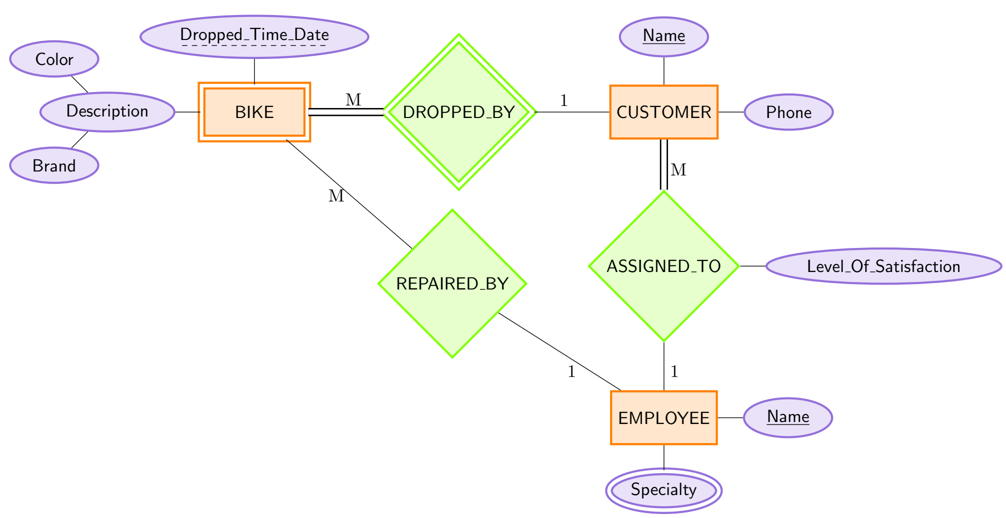
at that point. You can pick, for instance, HW_DB_COFFEE from Problem 3.7 (Read, correct, and write SQL statements for the COFFEE database)), click on "next", then click on "execute", "next", and "close."

7. You're back on the previous view, but you should now see "EER diagram" on the top of the middle panel. Click on "EER diagram" twice, scroll down if needed, and you should see the EER diagram.
8. This diagram is not exaclty an ER diagram and it is not a UML diagram either: it is an EER diagram, that uses Crow's foot notation. Make sure you understand it.
9. Try to modify the EER diagram. Make some relations mandatory, change their name, add an attribute, change the name of another relation, insert a couple of elements in an entity, add a row in a table, etc. Make sure you understand the meaning of the lines between the entities.
10. Once you're done, try to "Forward Engineer" by hitting "Ctrl" + "G." Click on "next" twice, enter your password, click on "next" once more and you should see the SQL code needed to produce the table you just designed using the graphical tool.

Problem 4.10 ***(ER-to-Relation mapping for car insurance)***

Apply the ER-to-Relation mapping to your ER diagram from Problem 4.3 (ER diagram for car insurance).

Problem 4.11 ***(From ER diagram to Relational model – BIKE)*** Consider the following ER diagram:

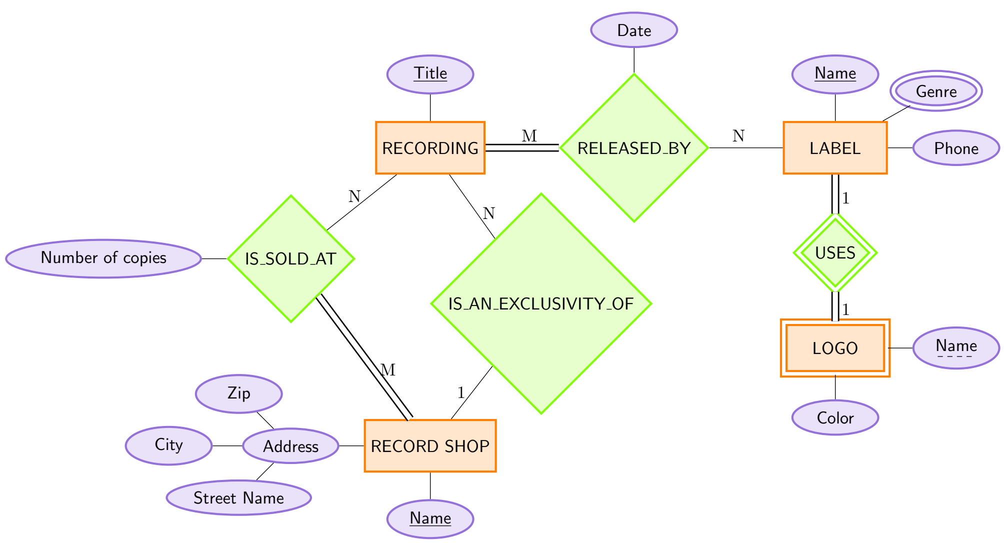


1. *Using this diagram*, answer the following:

Is it true that ...	Yes	No
... a customer cannot drop two bikes at the exact same time and date?		
... two different customers cannot drop two different bikes at the exact same time and date?		
... an employee cannot repair two bikes at the same time?		
... a customer can be assigned to more than one employee?		
... a customer can have a bike repaired by an employee that is not assigned to him/her?		
... a bike can be in the database without having been dropped by a customer?		
... an employee can be asked to repair a bike without having that type of bike as one of their specialties?		

2. Convert that ER diagram into a relational model. Try to make as few assumptions as possible.

Problem 4.12 ***(From ER diagram to Relational model – RECORD)*** Consider the following ER diagram:



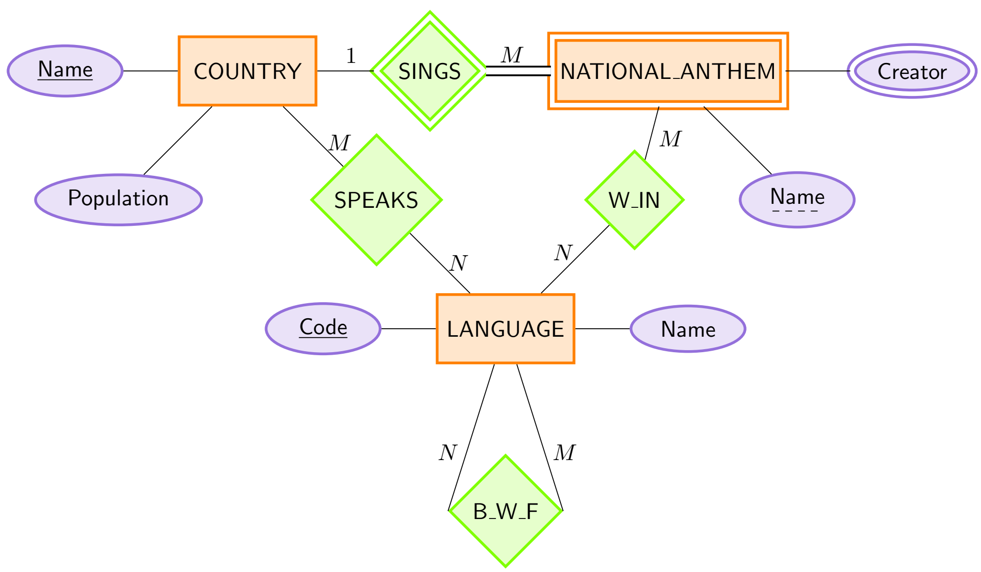
1. *Using this diagram*, answer the following:

Is it true that ...	Yes	No
... a label can have multiple logos?		
... a recording can be released by multiple labels and at different dates?		
... a record shop can have multiple exclusivities?		
... two record shops can have the same address?		

Is it true that ...	Yes	No
... two logos can have the same name?		
... two recordings can have the same title?		
... a record shop must sell at least one recording?		

2. Convert that ER diagram into a relational model. Try to make as few assumptions as possible.

Problem 4.13 ***(ER-to-Relation mapping for Country)*** Consider the following ER schema:



where

- "W_IN" stands for "WRITTEN_IN", and
- "B_W_F" stands for "BORROWS_WORDS_FROM."
- "W_IN" stands for "WRITTEN_IN", and
- "B_W_F" stands for "BORROWS_WORDS_FROM".

For this relationship, on the left-hand side is the language that borrows a word and on the right-hand side is the language that provides the loanword.

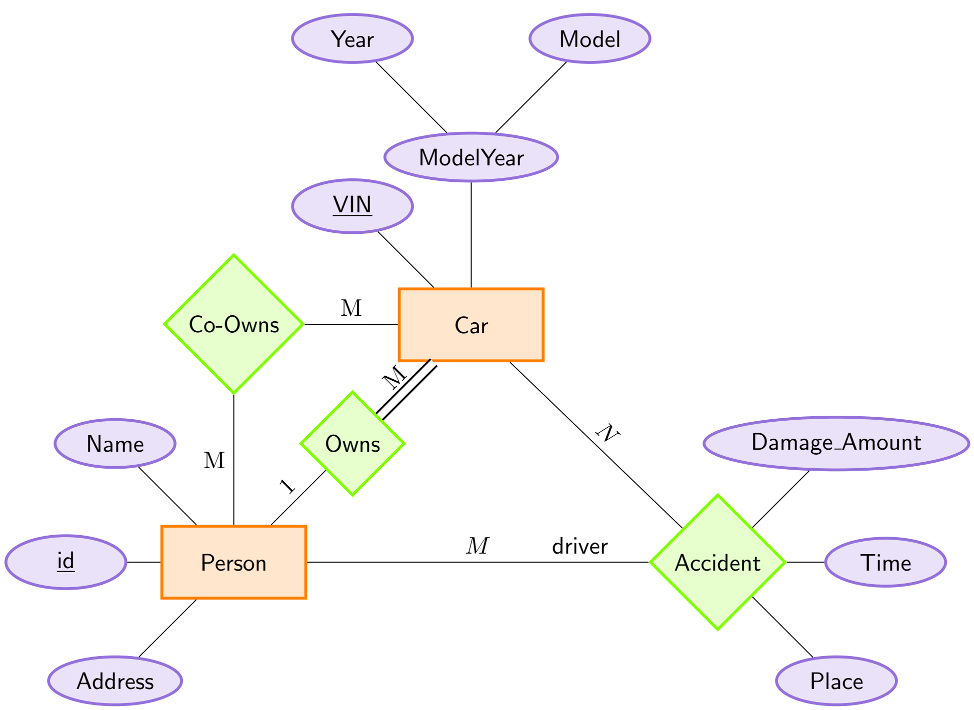
Map that ER diagram to a relational database schema.

Problem 4.14 ***(From business statements to ER diagram – UNIVERSITY)*** Consider the following requirements for a UNIVERSITY database used to keep track of students' transcripts.

- The university keeps track of each student's name, student number, class (freshman, sophomore, ..., graduate), major department, minor department (if any), and degree program (BA, BS, ..., PhD). Student number has unique values for each student.
- Each department is described by a name and has a unique department code.
- Each course has a course name, a course number, credit hours, and is offered by at least one department. The value of course number is unique for each course. A course has at least one section.
- Each section of a course has an instructor, a semester, a year, and a section number. The section number distinguishes different sections of the same course that are taught during the same semester/year; its values are 1, 2, 3, ..., up to the number of sections taught during each semester. Students can enroll in sections and receive a letter grade and grade point (0, 1, 2, 3, 4 for F, D, C, B, A, respectively).

1. Draw an ER diagram for this schema.
2. Specify the key attributes of each entity type and the structural constraints on each relationship type.
3. Note any unspecified requirements and make appropriate assumptions to complete the specification.

Problem 4.15 ***(Studying an Accident ER Diagram)*** Have a look at the diagram below:



The assumption is that a car has exactly one (primary) owner, but can additionally have multiple co-owners, and that the "Accident" relationship gathers information about who was driving which car when accident happened, along with some other information about the accident.

1. Answer the following short questions, justifying your answers:

   a) Can this model be used to determine if the driver involved in the accident was owning the car?
   b) What could justify having the "Address" attribute being composite?
   c) Is it possible to determine if multiple cars were involved in the same accident?

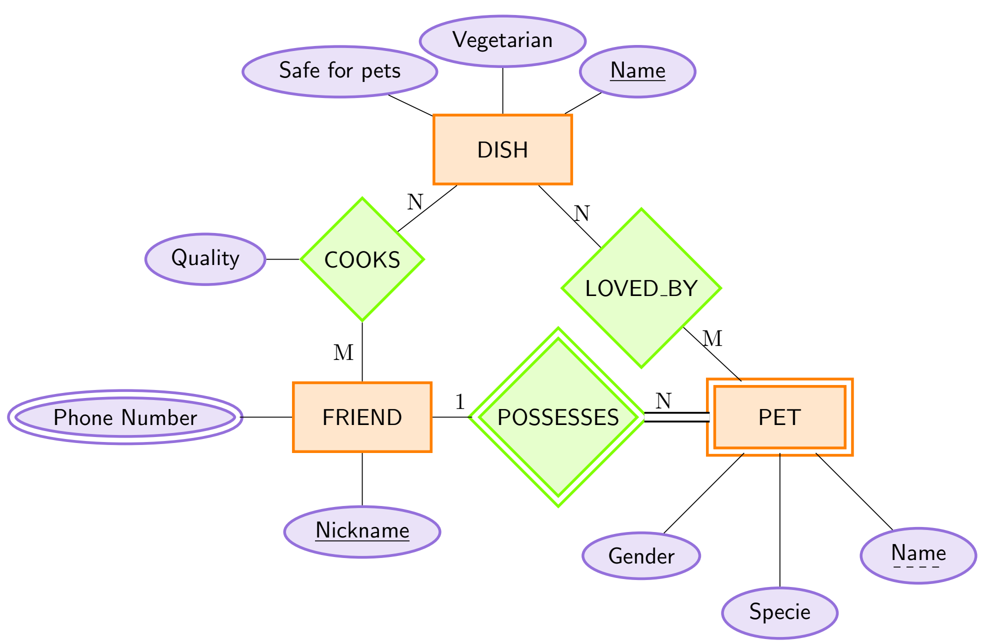
2. Convert the diagram to a relational model.

3. We want to edit the ER diagram to be able to record multiple quotes per accident instead of having a single "Damage_Amount". To do that, we would like to create a "Quote" weak entity with attributes for the amount and the contact information of the garage who wrote it. Your task is to

   a) Have Accident becomes an entity,
   b) Create a relationship between Accident, Car and Person to convey the information that was previously conveyed by the Accident relationship,
   c) Create a weak entity Quote with the required attributes and relationship.

You should draw only the part of the ER diagram that will change, no need to copy all of it.

Problem 4.16 ***(ER Diagram for Friendship, Dishes and Pets)*** Have a look at the diagram below:



The assumption is that a dish has two boolean attributes (vegetarian and safe for pets), and that the "Quality" attribute of the "COOKS" relationship stores how good a dish is when a particular friend cooks it.

1. Answer the following short questions, justifying your answers:

a) Can this model be used to determine if a friend can cook something his or her pet like?
b) Is it possible to determine if a pet loves a dish only when a particular friend cooks it?
c) Is it possible for a pet to belong to two friends at the same time?
d) What could justify having the "Phone Number" attribute being composite?
e) Is it possible to determine if a pet loves a dish that is not safe for pets?

2. Convert the diagram to a relational model.

Problem 4.17 ***(Normal form of a CAR_SALE relation)*** Consider the following relation and its functional dependencies:

CAR_SALE(Car_no, Date_sold, Salesman_no, Commission, Discount_amt)

{Car_no, Salesman_no}	$\longrightarrow$	{Date_sold, Commission, Discount_amt}
Date_sold	$\longrightarrow$	Discount_amt
Salesman_no	$\longrightarrow$	Commission

and let {Car_no, Salesman_no} be the primary key of this relation.

1. Based on the given primary key, is this relation in 1NF, 2NF, or 3NF? Why or why not?
2. Normalize it to its third normal form.

Problem 4.18 ***(Normal form of a simple relation)*** Consider the following relation:

REL(A, B, C, D, E)

Suppose we have the following dependencies:

A	$\longrightarrow$	D
{A, B}	$\longrightarrow$	C
D	$\longrightarrow$	E

1. What would be a suitable key for this relation?
2. How could this relation not be in first normal form?
3. Assume that it is in first normal form, and normalize it to the third normal form.

Problem 4.19 ***(Normal form of a SCHEDULE relation)*** Consider the following relation:

SCHEDULE(Period_Start, Period_End, Date, Room, Building, Organizer, Length)

And the following dependencies:

{Period_Start, Date}	→	{Room, Period_End}
{Period_Start, Length}	→	Period_End
{Period_Start, Period_End}	→	Length
{Period_End, Length}	→	Period_Start
{Date, Period_Start}	→	Organizer
Room	→	Building

1. Based on those functional dependencies, what would be a suitable primary key?
2. If this relation is not in second normal form, normalize it to the second normal form.
3. If this relation or the relation(s) you obtained previously is (are) not in third normal form, then normalize it (them) to the third normal form.

Problem 4.20 ***(Normalizing the FLIGHT relation)*** Consider the following relation:

FLIGHT(From, To, Airline, Flight#, Date_Hour, HeadQuarter, Pilot, TZDifference)

A tuple in the FLIGHT relation contains information about an airplane flight: the airports of departure and arrival, the airline carrier, the number of the flight, its time of departure, the headquarter of the company chartering the flight, the name of the pilot(s), and the time zone difference between the departure and arrival airports.

- The "Pilot" attribute is multi-valued (so that between 1 and 4 pilot's names can be stored in it).
- Given an airline and a flight number, one can determine the departure and arrival airports, as well as the date, time, and the pilot(s).
- Given the airline carrier, one can determine the headquarter.
- Finally, given the departure and arrival airports, one can determine their time zone difference.

Normalize the "FLIGHT" relation to its third normal form. You can indicate your steps, justify your reasoning, and indicate the foreign keys if you want to, but you do not have to.

Problem 4.21 ***(From business statement to dependencies, BIKE)*** This problem asks you to convert business statements into dependencies. Consider the following relation:

BIKE(Serial_no, Manufacturer, Model, Batch, Wheel_size, Retailer)

Each tuple in the relation BIKE contains information about a bike with a serial number, made by a manufacturer, with a particular model number, released in a certain batch, which has a certain wheel size, and is sold by a certain retailer.

1. Write each of the following dependencies as a functional dependency (the first one is given as an example):
   a) A retailer cannot have two bikes of the same model from different batches. *solution:* {Retailer, Model} → Batch
   b) The manufacturer and serial number uniquely identifies the bike and where it is sold.

c) A model number is registered by a manufacturer and therefore cannot be used by another manufacturer.
d) All bikes in a particular batch are of the same model.
e) All bikes of a certain model have the same wheel size.

2. Based on those statements, what could be a key for this relation?
3. Assuming all those functional dependencies hold, and taking the primary key you identified at the previous step, what is the degree of normality of this relation? Justify your answer.

Problem 4.22 ***(From business statement to dependencies, ROUTE)*** This problem asks you to convert business statements into dependencies. Consider the following relation:

ROUTE(Name, Direction, Fare_zone, Ticket_price, Type_of_vehicle, Hours_of_operations)

A tuple in the ROUTE relation contains information about a public transportation route: its name (e.g. "Gold", "Green", ...), its direction (e.g., "Medical Campus", "GCC", ...), the fare zone where the route operates (e.g., "Zone 1", "Zone 2", ...), the price of a ticket, the nature of the vehicles assuring the route (e.g., "subway", "bus", ...) and the time of operations (e.g., "24 hours a day", "from 0600 to 2200", etc.).

1. Write each of the following business statement as a functional dependency:
   a) Two different types of vehicle can not operate on routes with the same name.
   b) The ticket price depends of the fare zone and the type of vehicule.
   c) Both the name and the direction are needed to determine the hours of operations.
   d) Two routes with the same name and the same direction must have the same fare zone.
2. Based on those statements, what could be a key for this relation?

Problem 4.23 ***(From business statement to dependencies, ISP)*** Consider the following business statement:

> We want to represent the market of **Internet Service Providers** (ISP). Each ISP offers multiple **bundles**, that have a maximum **bandwith** and a **price**. Some ISP uses the same name for their bundles (e.g. "premium", or "unlimited"). Each ISP is given multiple **Internet Protocol addresses** (IP), and those never change. Every client has a **ID** that is proper to the ISP (i.e., ISP A and ISP B could both have a client with ID "00001"), an **email** and subscribes to a particular bundle from a particular ISP. The IP of a client changes over the **time**.

1. Assuming we have a relation with all the attributes written in bold in the business statement, list all the functional dependencies given by the statement.
2. Based on the functional dependencies you identified at the first step, construct a collection of relations, all in 3rd normal form, that would represent this situation.

Problem 4.24 ***(Perfecting a Relational Model for File Systems)*** We want to establish the relational model for the (idealized representation of) Unix files and file-system permissions. We obtained the following relation:

FILESYSTEM(FileName, Size, Extension, Uid, OwnerName, HomeFolder, FilePath, GroupName, Gid)

We wanted to represent the following situation: each file has a name, an extension (like `zip`, `cs`, etc.), a size and a path. The combination of the path, name and extension is unique to each file. Each file belongs to a particular user, that have a name, a (unique) Uid, and a home folder. Also, every file can be accessed by the members of particular groups. Finally, a group has a (unique) Gid and a name. For simplicity, we assume that a user can belong to at most one group, and that a group can have any number of users.

1. Give at least one high-level reason why this model is – to say the least – not ideal.
2. Give three functional dependencies between the attributes listed above based on our specification.
3. "Fix" our model: introduce as many relations and attributes as needed, and edit our FILESYSTEM relation any way you see fit. Specify the primary and foreign keys, and make it plausible that your model is in third normal form (you don't have to draw the functional dependencies, but intuitive understanding of the functional dependencies should not break the third normal form).

Problem 4.25 ***(Normal form for the GRADE_REPORT Relation)*** Consider the following relation:

GRADE_REPORT(StudentID, Term, StudentName, Department, Major, CourseNum, CourseTitle, LetterGrade, Grade)

and the following functional dependencies:

StudentID	→	StudentName
Major	→	Department
{Major, CourseNum}	→	CourseTitle
Grade	→	LetterGrade
{StudentID, Term, CourseNum, Major}	→	Grade

Identify a possible primary key for that relation. Then, using that primary key, decide if this relation is in second normal form. If it is not, identify a functional depency that prevents the relation from being in second normal form. Finally, normalize it to the third normal form. Try to pick meaningful names for your relations.

Problem 4.26 ***(Normalization)*** Consider the relations R and T below and their functional dependencies (as well as the one induced by the primary keys):

R(<u>EventId, Email,</u> Time, Date, Location, Status)

T(Invno, Subtotal, Tax, Total, Email, Lname, Fname, Phone)

{EventId, Email}	$\longrightarrow$	Status
EventId	$\longrightarrow$	{Time, Date, Location}
Invno	$\longrightarrow$	{Subtotal, Tax, Total, Email}
Email	$\longrightarrow$	{Fname, Lname, Phone}

Normalize the relations to 2NF and 3NF. Show all relations at each stage (2NF and 3NF) of the normalization process.

Problem 4.27 ***(Normal form of the BOOK relation)*** Consider the following relation for published books:

BOOK(Book_title, Book_type, Author_name, List_price, Author_affil, Publisher)

Suppose we have the following dependencies:

Book_title	$\longrightarrow$	{ Publisher, Book_type }
Book_type	$\longrightarrow$	List_price
Author_name	$\longrightarrow$	Author_affil

1. What would be a suitable key for this relation?
2. Explain how this relation could not be in first normal form.
3. This relation is not in second normal form: explain why and normalize it.
4. Are the relations you obtained in the previous step in third normal form? Explain why and normalize them if needed.

Problem 4.28 ***(Normal form of the DELIVERY relation)*** Consider the following relation for deliveries:

DELIVERY(Shipment, PackageNumber, RecipientName, Weight, DriverName, DriverPhone, RecipientPhone)

Suppose we have the following functional dependencies:

Shipment	$\longrightarrow$	DriverName
PackageNumber	$\longrightarrow$	Shipment
PackageNumber	$\longrightarrow$	{RecipientName, RecipientPhone}
PackageNumber	$\longrightarrow$	Weight
DriverName	$\longrightarrow$	DriverPhone

Answer the following three questions:

1. Reply Yes / No to the following: In this model...

a) ... can the shipment being handled by two drivers?
b) ... can a package have two different recipient?
c) ... can a package being in different shipments?
d) ... can a recipient have two different phone numbers for the same name?
e) ... can a driver have two different phone numbers for the same name?

2. Find a primary key for this relation.
3. Normalize this relation to the third normal form.

Problem 4.29 ***(Normal form of the CONTACT relation)*** Consider the relation

CONTACT(Phone, Call_center, Email, Zip, Brand, Website)

and the following functional dependencies:

{Zip, Brand}	→	{Phone}
{Brand}	→	{Email}
{ Brand}	→	{Website}
{Phone}	→	{Call_center}

Assume that {Zip, Brand} is the primary key. Normalize this relation to the second normal form, and *then* to the third normal form. Give the relations, their primary keys, and functional dependencies for both steps.

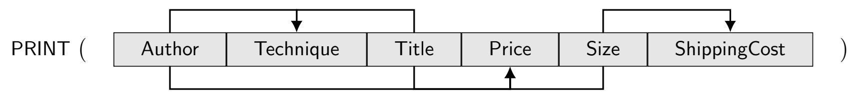
Problem 4.30 ***(Normal form of the MESSAGE relation)*** This exercise asks you to convert business statements into dependencies. Consider the following relation:

MESSAGE(SenderId, Time, Date, ReceiverId, Content, Length, Attachment, Size)

A tuple in the MESSAGE relation contains information about a text message: its sender, the time and date when it was sent, the receiver, the content, the length (in characters), the attachment, and the size (in bytes).

1. Write each of the following business statements as a functional dependency:
   a) The length of a message, which can be computed from its content.
   b) The content and attachment, which determines the size of a message.
   c) A sender can send the same content and attachment to multiple receivers at the exact same time and date, but cannot send two different contents and attachments at the exact same time and date.
2. Assuming all the functional dependencies you identified at the previous step hold, determine a suitable primary key for this relation.
3. Taking the primary key you identified at the previous step, what is the degree of normality of this relation? Justify your answer.
4. If needed, normalize this relation to the third normal form.

Problem 4.31 ***(PRINT relation in third normal form)*** Normalize the following relation to the third normal form.



Do not forget to indicate all the primary keys in your relations.

Problem 4.32 ***(CONSULTATION relation: justification, primary key and normal form)*** Consider the relation

CONSULTATION(Doctor_no, Patient_no, Date, Diagnosis, Treatment, Charge, Insurance)

with the following functional dependencies:

{Doctor_no, Patient_no, Date}	⟶	{Diagnosis}
{Doctor_no, Patient_no, Date}	⟶	{Treatment}
{Treatment, Insurance}	⟶	{Charge}
{Patient_no}	⟶	{Insurance}

1. The designer decided not to add the functional dependency {Diagnosis} ⟶ {Treatment}. Explain what could be the designer's justification (at the level of the mini-world).
2. Identify a primary key for this relation.
3. What is the degree of normalization of this relation? Normalize it to the third normal form if necessary.

Problem 4.33 ***(COFFEE relation: primary key and normal form)*** Consider the relation

COFFEE(Origin, Type_Of_Roast, Price, Roasted_Date, Best_Before, Color, Customer, Rating)

with the following functional dependencies:

{Origin, Type_Of_Roast}	⟶	Price
{Origin, Type_Of_Roast, Customer}	⟶	Rating
{Origin, Type_Of_Roast, Roasted_Date}	⟶	Color
Roasted_Date	⟶	Best_Before

Assume that all the attributes are atomic and answer the following.

1. Based on those functional dependencies, what would be a suitable primary key?
2. What is the degree of normalization of this relation? Justify your answer.
3. Normalize this relation to the *third* normal form, and do not forget to indicate all the functional dependencies. You can indicate the second normal form if that helps you.

Problem 4.34 ***(A Relation for Network Cards)*** A network card (NIC) has a **manufacturer**, a **model**, and a **unique serial number** (MAC address). It offers one or multiple **network technologies** (ethernet, wi-fi, bluetooth, etc.), and can be connected to the motherboard using one or multiple **connections** (PCI connector, FireWire, usb, etc.).

1. Assuming we have a NIC relation with all the attributes emphasized in the business statement, list all the functional dependencies.
2. The relation you obtained is not in 1st normal form, since two of the attributes (**network technology** and **connection**) are multi-valued. Propose a way to fix it, and suggest a primary key for the relation(s) you obtained.
3. Based on the functional dependencies you identified in the first step, is (are) the relation(s) you constructed in the second step in second normal form? If yes, explain why, if no, normalize it (them).

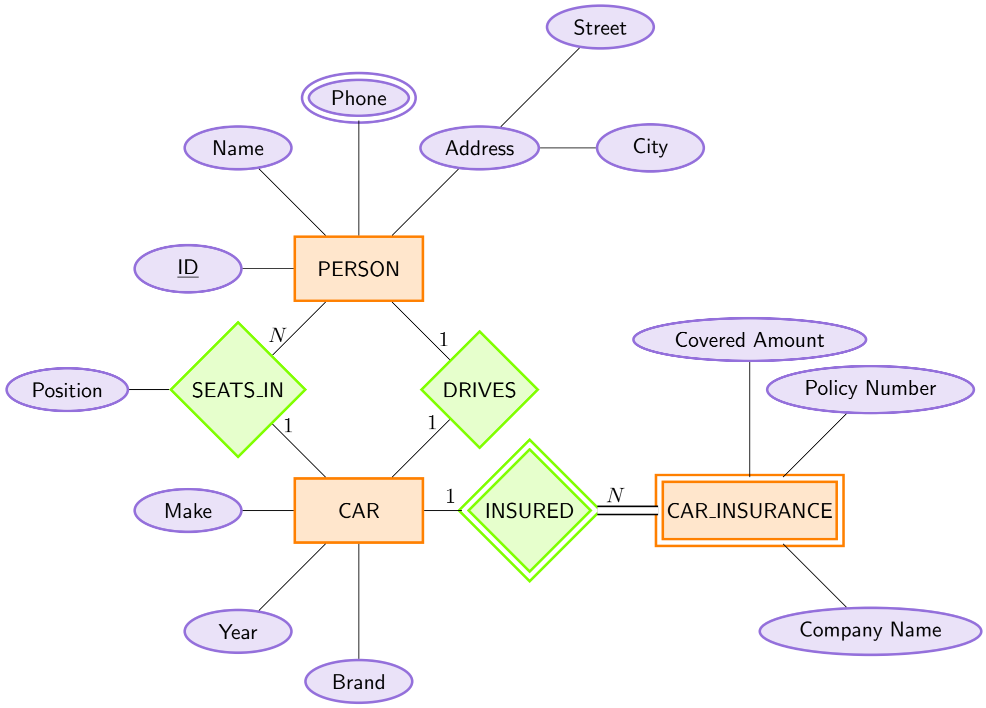
Problem 4.35 ***(From Business Statement to Functional Dependencies to Normal Form – TEACHING)*** This exercise asks you to convert business statements into dependencies, to identify a possible primary key, and to normalize the resulting relation.

Consider the following relation:

TEACHING(Class, Section, Instructor, Assistant, Office_Hours, Meeting_Hour)

- Write each of the following business statements as a functional dependency:
    1. The meeting hour is determined by the class and section.
    2. An assistant is an assistant to an instructor, no matter the class or section.
    3. A section of a class is taught by one instructor ; different section of the same class can be taught by different instructors.
    4. The office hours depends of the instructor and class.
- Assuming all the functional dependencies you identified at the previous step hold, determine a suitable primary key for this relation.
- Taking the primary key you identified at the previous step, what is the degree of normality of this relation? Justify your answer.
- If needed, normalize this relation to the third normal form.

Problem 4.36 ***(From ER to relational schema and UML class diagram – CAR_INFO)*** Consider the following ER schema for the CAR_INFO database:



Note that a car can have at most one driver, N passengers, N insurances, and that the car insurance entity exists only if it is "tied up" to a car (i.e., it is a weak entity, and its identifying relationship is called "Insured").

1. Find the key attribute for Car, and the partial key for Car Insurance. If you cannot think of any, add a dummy attribute and make it be the key.
2. Convert the ER diagram to a relational database schema.
3. Convert the ER diagram to a UMLclass diagram. Hint: Comparing Figure 7.16 with Figure 7.2 from your textbook should guide you.

Problem 4.37 ***(From Business Statement to ER Diagram to Relational Model – A Network of Libraries)*** You are asked to design a database for a network of libraries.

> Each library has a name, an address (made of a number, a street, and a zip), and have *copies* of documents available to borrow and to reserve. A document is of a particular kind (book, video, or disk), has a title, and an internal catalog number (that can be the ISBN, a barcode, etc.). There can be multiple copies of a document in the network, and each copy has a particular unique code. A copy of a document always "belongs" to a particular library, even when it is checked out.
>
> Furthermore, you want to be able to add the patrons in your database. A patron has a name, a unique library card number, and an email. A patron can reserve (put a hold on) multiple copies of documents for up to two weeks, and can borrow multiple copies of documents for one week if it is a video or a disk, and one month if it is a book. Of course, a copy can be borrowed by only one patron, but it can be put on hold for one patron while being borrowed.

1. Draw the ER diagram for this situation. Remember to add all the constraints on your relations.
2. Convert your ER diagram to the relational model.

Problem 4.38 ***(Using MySQL Workbench's reverse engineering)*** *This problem requires you to have successfully completed Pb 4.9 and Pb 4.40.*

Using the relational database schema you obtained in Pb 4.40, write the `SQL` implementation of that database. Then, using MySQL Workbench, use the "Reverse Engineering" function to obtain an EER diagram of your database and compare it with the UML diagram from Pb 4.40. Apart from the difference inherent to the nature of the diagram (i.e., UML vs EER), how else are they different? How are they the same? Is the automated tool as efficient and accurate as you are?

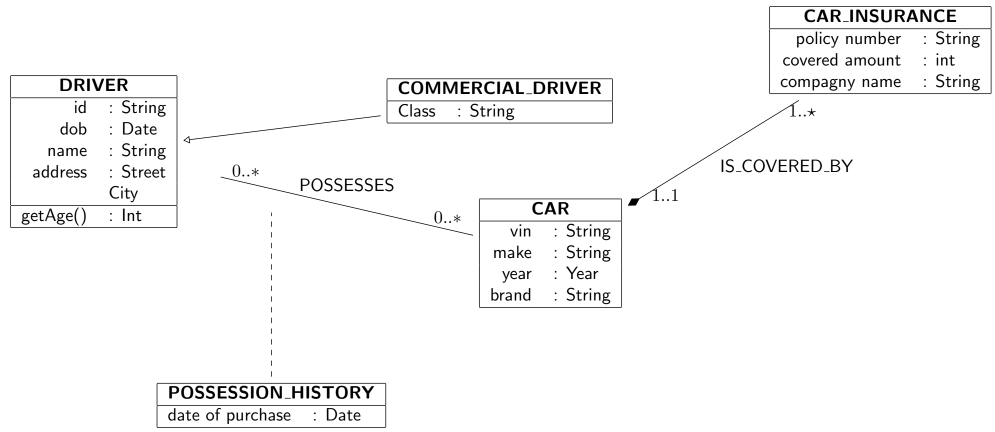
Problem 4.39 ***(From business statements to dependencies – KEYBOARD)*** This exercise asks you to convert business statements into dependencies. Consider the following relation:

KEYBOARD(Manufacturer, Model, Layout, Retail_Store, Price)

A tuple in the KEYBOARD relation contains information about a computer keyboard; its manufacturer, its model, its layout (AZERTY, QWERTY, etc.), the place where it is sold, and its price.

1. Write each of the following business statements as a functional dependency:
   - A model has a fixed layout.
   - A retail store cannot have two different models produced by the same manufacturer.
2. Based on those statements, what could be a key for this relation?
3. Assuming all those functional dependencies hold, and taking the primary key you identified at the previous step, what is the degree of normality of this relation? Justify your answer.

Problem 4.40 ***(From UML to relational model – DRIVER)*** Consider the UML diagram below, and convert it to the relational model. Do not forget to indicate primary and foreign keys.

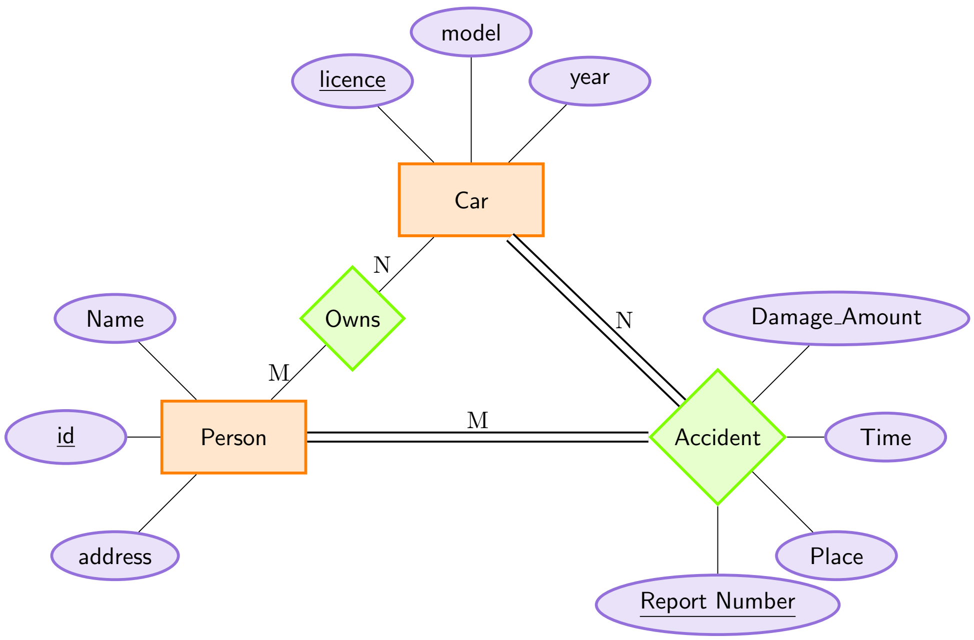


Solutions to Selected Problems

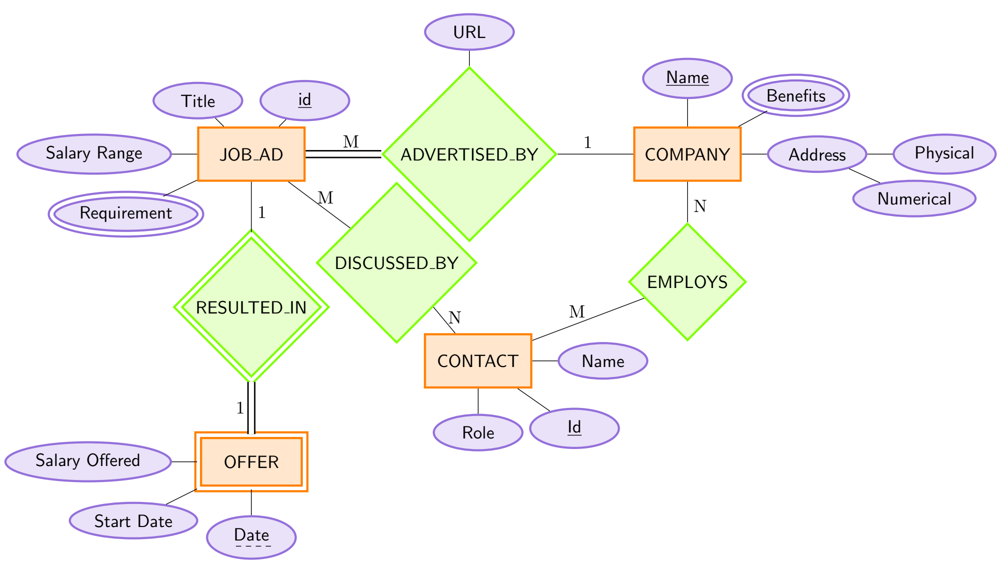
Solution to Problem 4.2 (Reading the MOVIES database ER schema)

1. True. The double lines on either side of the "PERFORMS IN" relation denote a total participation constraint. This means in particaular that an actor must have performed in at least one movie *and* that a movie must have had at least one actor who performed in it.
2. True. The arity on the right side of the "PERFORMS IN" relation is "N." This means there are an unlimited amount of movies in which an actor may perform.
3. True. The arity on the right side of "LEAD ROLE" is "N." This means that an actor may have performed in multiple lead roles for movies.
4. True. The arity on the left side of "LEAD ROLE" is "2." This means that a move can have a maximum of two lead roles.
5. False. The single line from the "DIRECTOR" to "ALSO A DIRECTOR" is unconstrained and means that the relation is optional for that entity.
6. False. There exists an "ACTOR PRODUCER" realtion between the "ACTOR" and "PRODUCER" entities to facilitate this situation.
7. False. Although the single line between "PRODUCER" and "ACTOR PRODUCER" means the relationship is optional, it does not mean it cannot happen.
8. True. The arity on the left side of "PERFORMS IN" is "M." This means that a movie can have an unlimited amount of actors in it.
9. True. It is not explicit in this schema, but an actor can be both a director and a producer, which implies that a producer can also be a director by transitivity.
10. True. In fact, there is a partial participation constraint that means a movie must have one director and one producer at least.
11. True. The arity on the left of "DIRECTS" is "1" and is "M" on the left of "PRODUCES." This means that a movie can only have up to one director, but have an unlimited amuont of producers.
12. True. All of these relations exist to enable an actor to act in a lead role, be a director, and be a producer.
13. False. There exists a relation for a director to be an actor *and* there are no constraints keeping the director to perform in the movie they are directing.

Solution to Problem 4.3 (ER diagram for car insurance) A possible solutions is

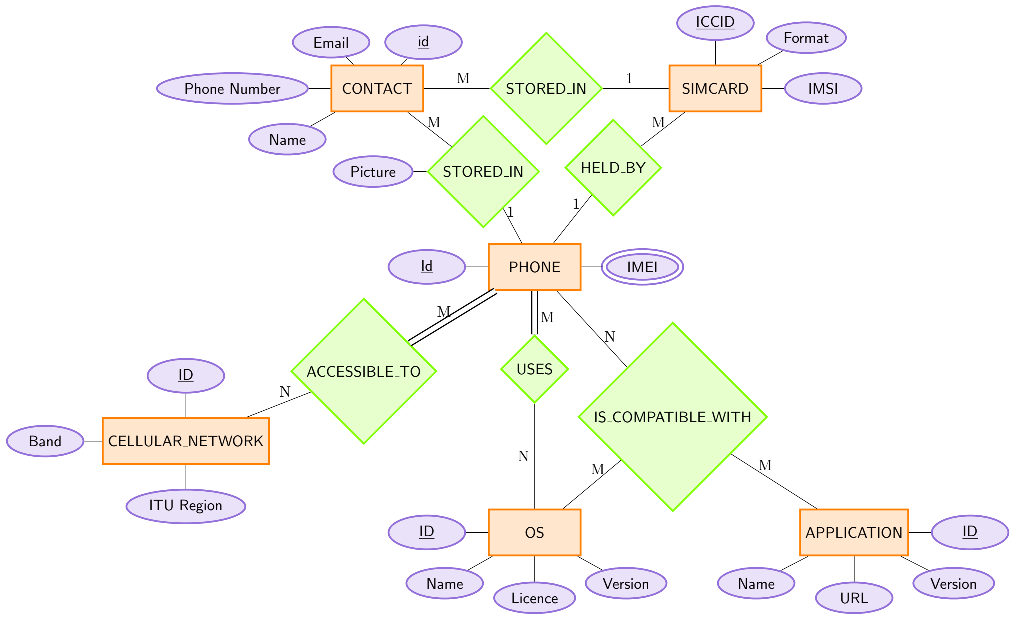


Solution to Problem 4.4 (ER diagram for job and offers) A possible solution is:



Note that CONTACT could be a weak entity with the identifying relationship being either DISCUSSED_BY or EMPLOYS, but both have disadvantages: they would not allow a contact to discuss more than one offer or to be hired by more than one company.

Solution to Problem 4.5 (ER diagram for cellphones) A possible solution is:



Note that we sometimes introduced `ID` attributes, but could have done without as well (typically, by taking the name and version attributes and gathering them in the same attribute that could be used as a key, for the OS and APPLICATION entities).

Solution to Problem 4.6 (Incorrect ER diagram) Among the numerous flaws, come to mind:

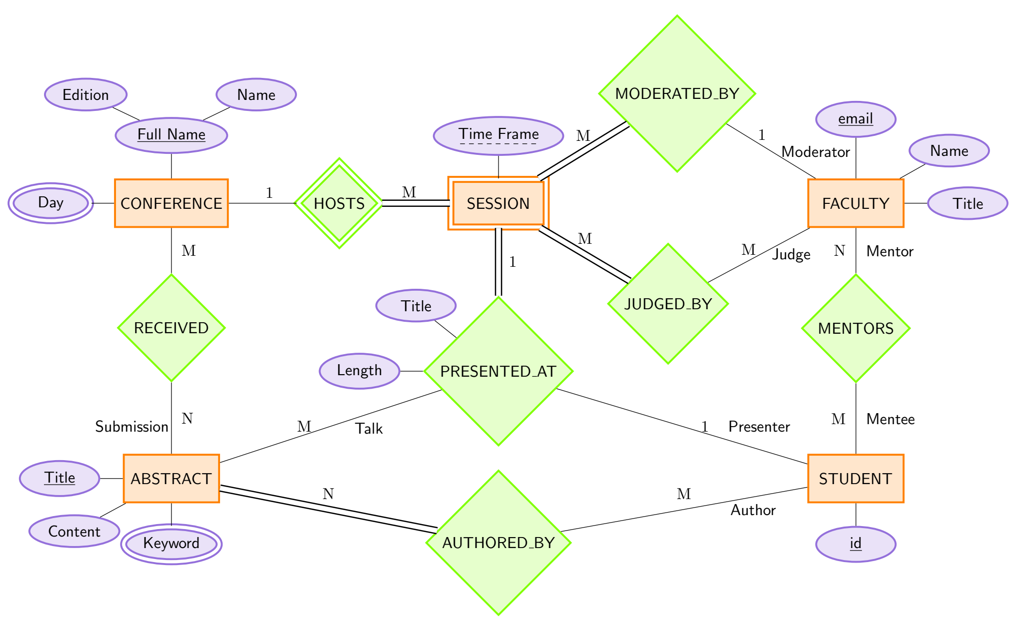
1. "Technical errors" (making the diagram incorrect):
   a) Multiple keys for an entity ("PROGRAMMER"),
   b) Absence of a key ("PROGRAMMING_LANGUAGE"),
   c) Absence of the total participation constraint between the weak entity and its identifying relationship ("GUIDELINES" and "RECOMMENDED_BY"),
   d) Absence of an arity (between the "USES" relationship and the "PROJECT" entity),
   e) To a certain extent, inconsistent naming (spaces / underscore, absence of capital letter for "leader" or "url")
2. Violations of the business statement:
   a) "They want to accommodate the fact that a project can use multiple programming languages (and sometimes even multiple versions of the same language)": this is not possible. There are two ways of adding this feature:
      i. Make the "PROGRAMMING_LANGUAGE" entity become a "VERSION_OF_PROGRAMMING_LANGUAGE" entity, and make the "USE" relationship M:N.

ii. Leave the "PROGRAMMING_LANGUAGE" entity as it is, make the "USE" relationship M:N, and add a "Version" attribute to it. The first option being a bit more elegant, to some extend (but we lose the capacity of deriving the latest version).

b) "keep track of which programmer is leading which project.": As a leader is supposed to be a programmer as well, there should be a "IS_THE_LEADER_OF" relationship between "PROGRAMMER" and "PROJECT", instead of an attribute on the "PROJECT" entity.
c) "they also want to track which programmer is knowledgeable of what programming language": the "KNOWS" relationship should not be "M:1", as a programmer can probably being knowledgeable in more than one programming language.
d) "if a project requires a particular guideline[...], it should be stored somewhere": The "RECOMMENDED_BY" relationship should be 1:1.

3. Basic understanding errors:
   a) The lack of "Name" attribute in the "PROGRAMMING_LANGUAGE" entity is puzzling.
   b) "CONTRIBUTES_TO" should probably not being total on any side, as you may want to record programmers even if they are not contributing to any project, and project even if nobody is actively working on them. Also, it should not be 1:1, but M:N.
   c) "RECOMMENDED_BY" could probably be renamed "GUIDE_THE_DEVELOPMENT_OF", for added clarity.

Solution to Problem 4.7 (ER diagram for Undergraduate Conference) A possible solution is:



Where we made the following assumptions:

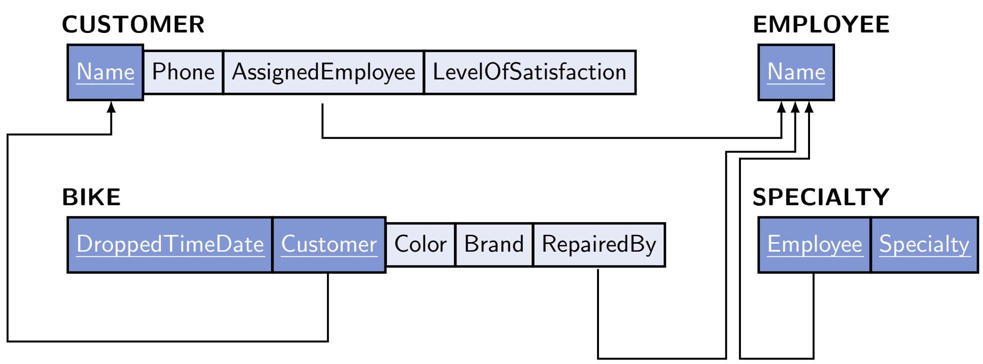
- A student has an id (note that having an entity type without attributes is improper, so we had to make an attribute),
- No two sessions for the same coneference can take place at the exact same time,
- Conferences cannot have joint sessions.

Also, note that we could have decided to make Presentation an entity instead of a relationship: both choices were correct. To determine if an abstract was accepted, one has to "track" whenever it is in the PRESENTED_AT relationship. Another option would have been to add a "Accepted" attribute to the RECEIVED relationship.

Solution to Problem 4.11 (From ER diagram to Relational model – BIKE)

Is it true that ...	Yes	No
... a customer cannot drop two bikes at the exact same time and date?	✔	
... two different customers cannot drop two different bikes at the exact same time and date?		✔
... an employee cannot repair two bikes at the same time?		✔
... a customer can be assigned to more than one employee?		✔
... a customer can have a bike repaired by an employee that is not assigned to him/her?	✔	
... a bike can be in the database without having been dropped by a customer?		✔
... an employee can be asked to repair a bike without having that type of bike as one of their specialties?	✔	

1.
2. For the `1:M` relationships that are not identifying, we can choose between the foreign key and the cross-reference approaches. If we use the former, we obtain:



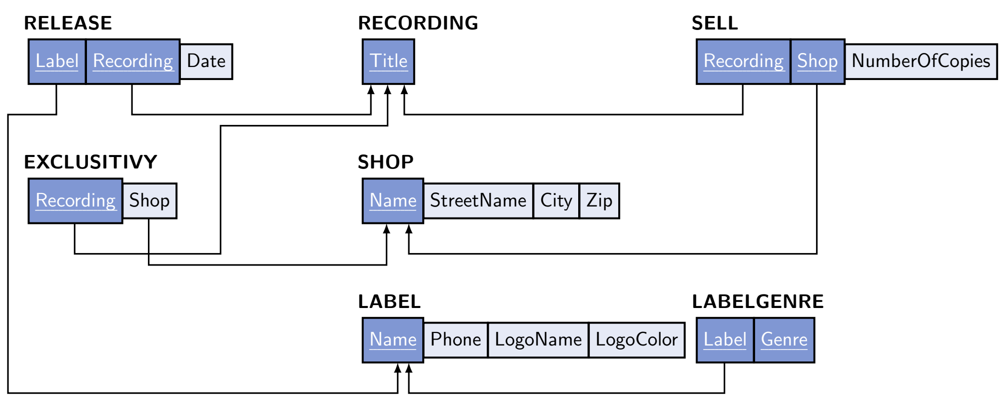
We could also have used a combination of both!

Solution to Problem 4.12 (From ER diagram to Relational model – RECORD)

Is it true that …	Yes	No
a label can have multiple logos?		✔
a recording can be released by multiple labels and at different dates?	✔	
a record shop can have multiple exclusivities?	✔	
two record shops can have the same address?	✔	
two logos can have the same name?	✔	
two recordings can have the same title?		✔
a record shop must sell at least one recording?	✔	

1.

2. For the `1:M` relationship `IS_AN_EXCLUSIVITY_OF`, we can choose between the foreign key and the cross-reference approaches. For the `1:1` relationship `USES`, we can use any approach we want (foreign key, merged relation, or cross-reference). We will choose to merge the two relations `LABEL` and `LOGO` and to have a look-up table for the `IS_AN_EXCLUSIVITY_OF` relation. This obtains:



Solution to Problem 4.16 (ER Diagram for Friendship, Dishes and Pets)

1. We have the following answers:

   a) Yes, we can determine if a friend can cook something his or her pet likes by first looking at the dishes in the "COOKS" relationship with that friend, and then looking if any of those dishes are in the "LOVED_BY" relationship with their pet (and we can determine if that pet belongs to them using "POSSESSES").

   b) No, we cannot determine if a pet loves a dish only when a particular friend cooks it: the "LOVED_BY" relationship expresses the connection between a pet and a dish regardless of who cooked it.

   c) No, a pet cannot belong to two friends at a time, since the "POSSESES" relationship is 1:N between the FRIEND and the PET entitiy types.

d) A "Phone number" attribute could be composite (on top of being multi-valued, as it is now) to e.g. separate the area code from the rest of the phone, or have a label to indicate if the phone number corresponds to a cellphone or a landline.

e) Following "LOVED_BY", it is possible to list the dishes loved by a pet and then to inspect the value of their "Safe for pets" attribute: however, if that attribute's value is `NULL`, then the database does not really says if that dish is safe or not.

2. The main difficulty in this diagram is to represent correctly the weak entity type "PET": it should have *two* attributes in its primary key, "Name" and "Owner" (or something similar), that latter attribute being a foreign key to the primary key of "FRIEND". As a consequence, the look-up table representing "LOVED_BY" should have *three* attributes: a foreign key to the primary key for "DISH", and *two* foreign keys to the two attributes that constitute the primary key for "PET". On a side note, "Quality" should become an attribute of the look-up table representing "COOKS".

Solution to Problem 4.17 (Normal form of a CAR_SALE relation)

1. The CAR_SALE relation is in 1st normal form, since it has a primary key, and by assuming that all the attributes are atomic. This relation is not is 2nd Normal Form: since Date_sold → Discount_amount and Salesman_no → Commission, then some attributes (namely Discount_amount and Commission) are not fully functional dependent on the primary key. Hence, this relation cannot be in 3rd normal form either.

2. To normalize,

2NF:

Relations	Functional Dependencies
Car_Sale1(Car_no, Date_sold, Discount_amt)	Car_no → {Date_Sold, Discount_amt} and Date_Sold → Discount_amt
Car_Sale2(Car_no, Salesman_no)	Car_no → Salesman_no
Car_Sale3(Salesman_no, Commission)	Salesman_no → Commission

3NF:

Relations	Functional Dependencies
Car_Sale1-1(Car_no, Date_sold)	Car_no → Date_Sold
Car_Sale1-2(Date_sold, Discount_amt)	Date_Sold → Discount_amt
Car_Sale2(Car_no, Salesman_no)	Car_no → Salesman_no
Car_Sale3(Salesman_no,Commission)	Salesman_no → Commission

Solution to Problem 4.18 (Normal form of a simple relation)

1. {A, B} would be a suitable primary key (actually, it is the only one).

2. If no key was selected, or if an attribute has a multi-valued domain, then this relation would not be in first normal form.

3. The following three relations are in third normal form:

   a) R1(A, B, C)

   b) R2(D, E)

   c) R3(A, D)

Solution to Problem 4.19 (Normal form of a SCHEDULE relation)

1. {Period_Start, Date} would be a suitable primary key.
2. This relation is already in second normal form: there are no non-prime attributes that are not fully dependent of the primary key. Stated differently, there are no non-prime A such that {Period_Start} ⟶ A or {Date} ⟶ A.
3. This relation is not in 3rd normal form. Consider the following relation: {Period_Start, Date} ⟶ {Period_Start, Period_End} ⟶ Length. {Period_Start, Period_End} is different from {Period_Start, Date} and from Length, and it is not included in a candidate key. The same goes for {Period_Start, Date} ⟶ Room ⟶ Building.

Once normalized to the third normal form, we get:

Solution to Problem 4.21 (From business statement to dependencies, BIKE)

1. The functional dependencies we obtain are:

   a) { Manufacturer, Serial_no } ⟶ { Model, Batch, Wheel_size, Retailer}
   b) Model ⟶ Manufacturer
   c) Batch ⟶ Model
   d) {Model, Manufacturer} ⟶ Wheel_size
2. {Manufacturer, Serial_no }
3. If every attribute is atomic, it is in second normal form. { Manufacturer, Serial_no } ⟶ Batch ⟶ Model breaks the 3NF.

Solution to Problem 4.22 (From business statement to dependencies, ROUTE) The relation we consider is:

ROUTE(Name, Direction, Fare_zone, Ticket_price, Type_of_vehicle, Hours_of_operations)

1. This problem asks to convert business statements into dependencies.
   a) Two different types of vehicles can not operate on routes with the same name. Name ⟶ Type_of_vehicle

b) The ticket price depends of the fare zone and the type of vehicule. {Fare_zone, Type_of_vehicle} ⟶ Ticket_price,

2. Both the name and the direction are needed to determine the hours of operations. {Name, Direction} ⟶ Hours_of_operations

3. Two routes with the same name and the same direction must have the same fare zone. {Name, Direction} ⟶ Fare_zone
4. Based on those statements, {Name, Direction} is the only key for this relation.

Solution to Problem 4.23 (From business statement to dependencies, ISP)

1. The relation we consider is:

ISP(ISP, bunle, bandwith, price, IP, ID, email, time)

The functional dependencies suggested by the business statement are:

{ISP, bundle}	⟶	{bandwidth, price}
IP	⟶	ISP
{ISP, ID}	⟶	{email, bundle}
{ISP, ID, time}	⟶	IP

1. We obtain the following four relations when we normalize it to the third normal form:

   a) BUNDLE(<u>ISP, bundle,</u> bandwidth, price)

   b) IP(<u>ISP,</u> IP)

   c) CLIENT(<u>ISP, id,</u> email, bundle)

   d) CLIENT_IP(<u>ISP, id, time,</u> IP)

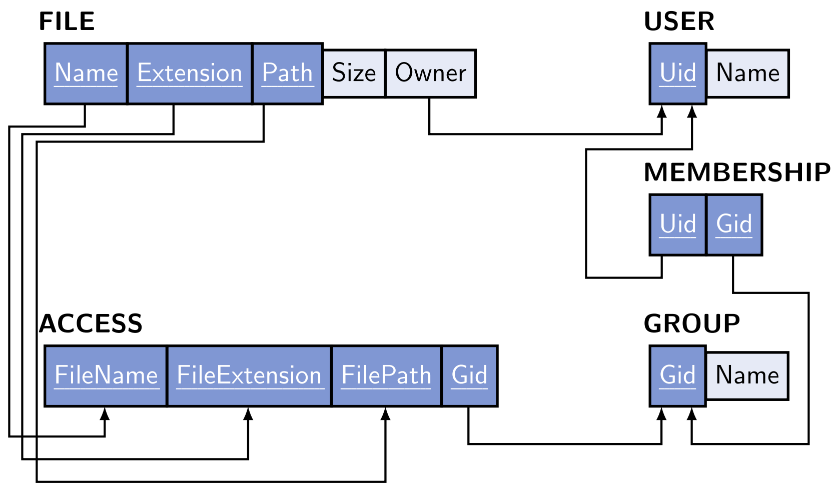
Solution to Problem 4.24 (Perfecting a Relational Model for File Systems)

1. The first criticism against that model that comes to mind is that it represents multiple entities (user, group, files, at least) into one relationship. As a consequence, a lot of redundancy is to be expected: typically, the OwnerName and HomeFolder values needs to match every time they occur with the same UID. If they do not, then this means that some inconsistency occurs in the database.

2. We can have:

{FileName, FilePath, Extension}	⟶	{Size, OwnerName}
Uid	⟶	HomeFolder
Uid	⟶	OwnerName
Gid	⟶	GroupName

3. We could obtain something like



An important aspect to remember is that relations with multiple attributes for their primary key needs as many foreign keys as attributes in their primary key to be referenced. Hennce, the "ACCESS" relationship is a bit cumbersome, as to "point" to a file, it needs three attributes. However, it would be a mistake to make "Gid" an attribute of "FILE", as the same file can be accessed by multiple groups.

Solution to Problem 4.25 (Normal form for the GRADE_REPORT Relation) The primary key would be {StudentID, Term, Major, CourseNum}.

Numerous functional dependencies prevent it from being in second normal form, for instance StudentID $\to$ StudentName prevents the functional dependency {StudentID, Term, CourseNum, Major} $\to$ StudentName from being full.

We obtain the following five relations when we normalize it to the third normal form:

GRADE_REPORT(<u>StudentID, Term, CourseID, Major,</u> Grade)
COURSE_INFO(<u>CourseID, Major,</u> CourseTitle)
STUDENT_INFO(<u>StudentID,</u> StudentName)
MAJOR_INFO(<u>Major,</u> Address)
GRADE_SCALE(<u>Grade,</u> LetterGrade)

Solution to Problem 4.27 (Normal form of the BOOK relation)

1. {Book Title, Author Name}
2. If an attribute is composite or multi-valued, then the relation would not be in first normal form.
3. It is not in second normal form because of { Book_title } $\longrightarrow$ { Publisher, Book_type }. We can normalize it like so: (Book Title, Publisher, Book Type, List Price), (Author Name, Author Affiliation), (Author Name, Book Title).

4. The relations are in third normal form because of {Book title} ⟶ { Book_type} ⟶ { List_price}| (Book Title, Publisher, Book Type) and (Book Type, List Price), (Author Name, Author Affiliation), (Author Name, Book Title).

Solution to Problem 4.28 (Normal form of the DELIVERY relation)

1. Reply Yes / No to the following: In this model...

   a) ... can the shipment being handled by two drivers? **No**
   b) ... can a package have two different recipient? **No**
   c) ... can a package being in different shipments? **No**
   d) ... can a recipient have two different phone numbers for the same name? **Yes**
   e) ... can a driver have two different phone numbers for the same name? **No**

2. PackageNumber would be a suitable primary key for this relation.
3. A possible third normal form is (where the only functional dependencies are the one given by the primary keys): SHIPMENT(<u>Shipment,</u> DriverName)
   DRIVER(<u>DriverName,</u> DriverPhone)
   PACKAGE(<u>PackageNumber,</u> Shipment, Weight, RecipientName, RecipientNumber)

Solution to Problem 4.31 (PRINT relation in third normal form) After normalizing PRINT to the second normal form (by adding the primary key {Author, Title, Size}, and working on dependencies like {Author, Title} ⟶ Technique, which does not fully depend of the primary key), we would obtain three relations that are already in third normal form:

1. PRICING(<u>Author,</u> <u>Title,</u> <u>Size</u>, Price)

2. ART(<u>Author,</u> <u>Title,</u> Technique)

3. SHIPPING_COSTS(<u>Size,</u> Price)

Solution to Problem 4.32 (CONSULTATION relation: justification, primary key and normal form)

1. The treatment for a particular disease can vary with the patient (for instance, his age can be a crucial parameter).
2. $\{Doctor\_no, Patient\_no, Date\}$ is a primary key for this relation.
3. Since we have $\{Patient\_no\} \rightarrow \{Insurance\}$, $\{Doctor\_no, Patient\_no, Date\} \rightarrow \{Insurance\}$ is a partial dependency, and this relation is not in 2NF. As we fixed a primary key in the previous step, it is in 1NF. As Charge is a non-key attribute that is determined by non-key attributes (Treatment and Insurance), we must decompose the relation further:

CONSULTATION (<u>Doctor*no,* *Patient* no,</u> <u>Date,</u> Diagnosis, Treatment)
PRICE_LISTING (<u>Treatment, Insurance,</u> Charge)
PATIENT_INFO(<u>Patient_no,</u> Insurance)

Solution to Problem 4.33 (COFFEE relation: primary key and normal form) The original relation is:

COFFEE(Origin, Type_Of_Roast, Price, Roasted_Date, Best_Before, Color, Customer, Rating)

1. A suitable primary key would be $PK_{COFFEE} = \{Origin, Type\_Of\_Roast, Roasted\_Date, Customer\}$ Note that it is the minimal and only primary key.
2. This relation is in first normal form because it has a primary key (the one we just defined), and because all the attributes are atomic. It is not in second normal form, because, for example, the functional dependency $PK_{COFFEE} \to Price$ is not fully functionally dependent, since $\{Origin, Type\_Of\_Roast\} \to Price$ holds.
3. Normalizing to the second normal form actually gives us relations in third normal form:

   -CLIENT_RATING(Origin, Type_Of_Roast, Customer, Rating)
   -PRICING(Origin, Type_Of_Roast, Price)
   -EXPIRATION_DATE(Roasted_Date, Best_Before)
   -COFFEE_BATCH(Origin, Type_Of_Roast, Roasted_Date, Color)

Where the functional dependencies always are in such a way that all the attributes but the last one fix the value of the last one, and are taken to be the primary key.

Checking that they are all in third normal form is straightforward. Note that the "original" relation was somewhat lost, since we do not have a relation whose primary key is PK_{COFFEE} anymore. We could have re-introduced a relation with only the attributes of PK_{COFFEE} to be on the "safe side", but the benefit would not have been clear.

Solution to Problem 4.35 (From Business Statement to Functional Dependencies to Normal Form – TE

- The functional dependencies given by the four statements are as follows:

{Class, Section}	→	Meeting_Hour
Assistant	→	Instructor
{Class, Section}	→	Instructor
{Instructor, Class}	→	Office_Hours

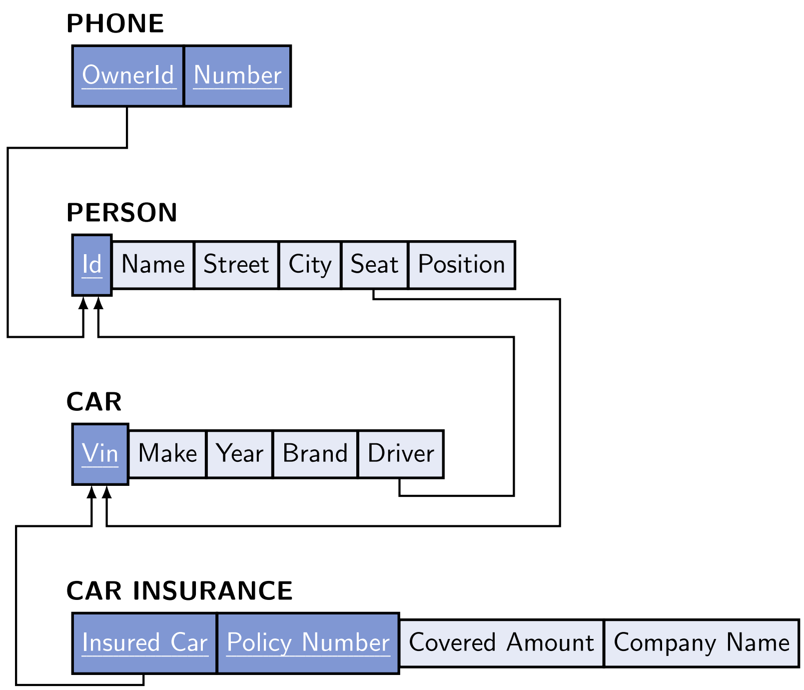
  Note that the statement reads "*An assistant is an assistant to an instructor*", which implies that an assistant can assist at most one instructor, but does not imply that an instructor can have at most one assistant: hence, the dependency is from Assistant to Instructor, and not the other way around.

- Based on the dependencies identified at the previous step, {Class, Section, Assistant} is the primary key.

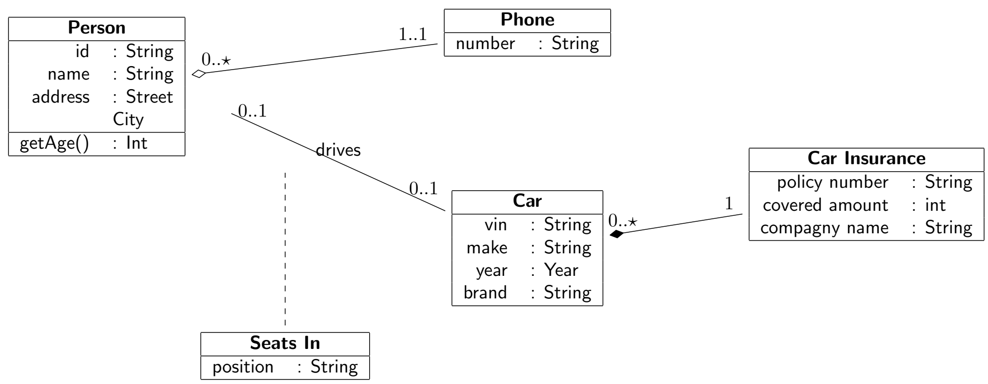
- This relation is in 1st normal form: we make the assumption that all the attributes are atomic, and we identified a primary key. However, it is not in second normal form: Assistant ⟶ Instructor, for instance, makes that Instructor is not fully functionally dependent on the primary key.
- In third normal form, we would get:

CLASS_INFO(<u>Class,</u> <u>Section,</u> Instructor, Meeting_Hour)
ASSISTANTSHIP(<u>Assistant,</u> Instructor)
OFFICE_HOURS(<u>Instructor,</u> <u>Class,</u> Office_Hours)

Solution to Problem 4.36 (From ER to relational schema and UML class diagram – CAR_INFO)
For Car, we need to create an attribute, like VIN. For Car Insurance, Policy Number is the perfect key attribute.

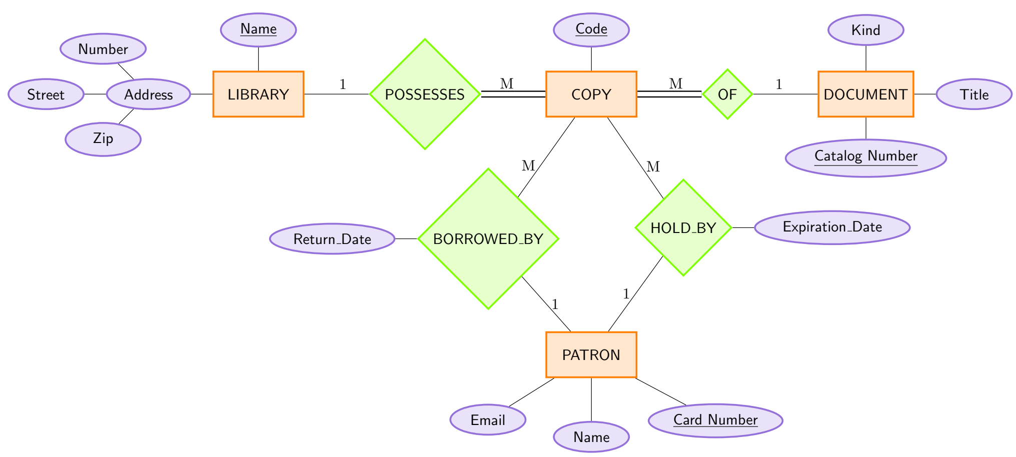


Note that, during the coversion, we had to make Insured Car part of the primary key of CAR INSURANCE.



Solution to Problem 4.37 (From Business Statement to ER Diagram to Relational Model – A Network o

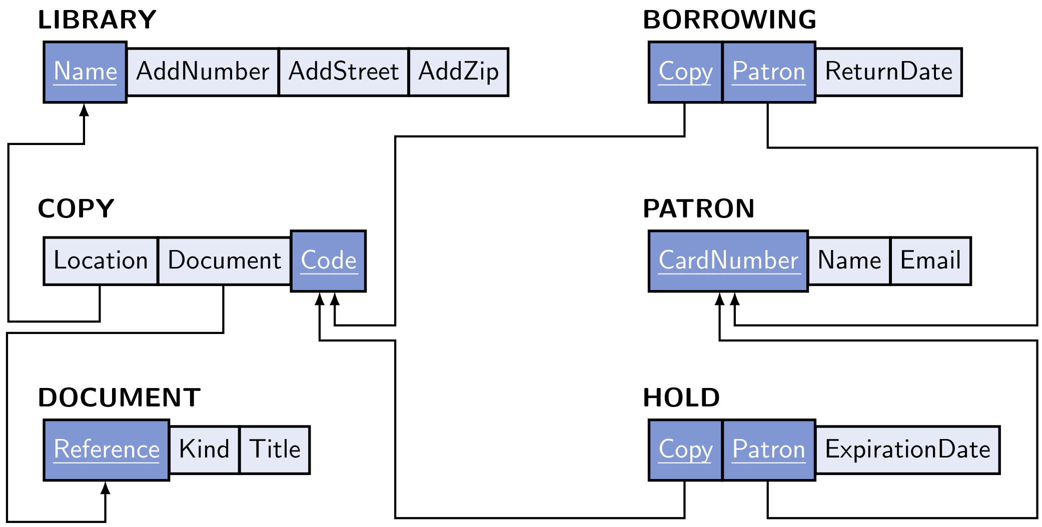
1. For the ER diagram, we could get something like:



Note that:

- We want to represent the fact that a *single* document can have *multiple* copies, which suggests that DOCUMENT and COPY are two separate entities.
- COPY could be made into a weak entity, OF being the identifying relation.
- Nothing in the statement *forces* a relationship between the patron and the library to exist, so, by simplicity, we do not add it. However, adding it would not have been a mistake.
- The fact that a COPY has to be of a particular kind does not force the kind attribute to be multi-valued or composite: it just means that if we were representing the domains as well, this attribute would have a particular domain that restricts the values to three possibilities (book, video or disk).
- POSSESS is total on the COPY side because the statement reads “A copy of a document always ‘belongs’ to a particular library.”
- HOLD_BY could be $N : M$, since nothing in the statement says that a document can be put on hold by only one patron.

1. Its mapping to a relational model could be:



Note that:

- The relationships HOLD_BY and BORROWED_BY could be represented using the foreign key approach. However, their value would be **NULL** most of the time, so it would not be very efficient.
- COPY could be the only attribute in the primary key of BORROWING and HOLD because a copy can be borrowed or put on hold only once. Having both attributes being the primary key could allow for more flexibility (typically, a copy could be put on hold by multiple patrons at the same time).

Solution to Problem 4.38 (Using MySQL Workbench's reverse engineering) We give the code first, then the drawing:

```
/* code/sql/HW_Person.sql */
DROP SCHEMA IF EXISTS HW_Person;

CREATE SCHEMA HW_Person;

USE HW_Person;

CREATE TABLE PERSON (
  ID VARCHAR(25) PRIMARY KEY,
  NAME VARCHAR(25),
  Street VARCHAR(25),
  City VARCHAR(25),
  Seat VARCHAR(25),
  Position VARCHAR(25)
```

```
);

CREATE TABLE CAR (
  Vin VARCHAR(25) PRIMARY KEY,
  Make VARCHAR(25),
  Model VARCHAR(25),
  Year DATE,
  Driver VARCHAR(25),
  FOREIGN KEY (Driver) REFERENCES PERSON (ID) ON UPDATE CASCADE
);

ALTER TABLE PERSON
  ADD FOREIGN KEY (Seat) REFERENCES CAR (Vin);

CREATE TABLE CAR_INSURANCE (
  Policy_number VARCHAR(25) PRIMARY KEY,
  Company_name VARCHAR(25),
  Insured_car VARCHAR(25),
  FOREIGN KEY (Insured_car) REFERENCES CAR (Vin)
);

CREATE TABLE PHONE (
  ID VARCHAR(25),
  Number VARCHAR(25),
  FOREIGN KEY (ID) REFERENCES PERSON (ID),
  PRIMARY KEY (ID, number)
);
```

HW_Person.sql[11]

[11]https://rocketgit.com/user/caubert/CSCI_3410/source/tree/branch/master/blob/notes/code/sql/HW_Person.sql

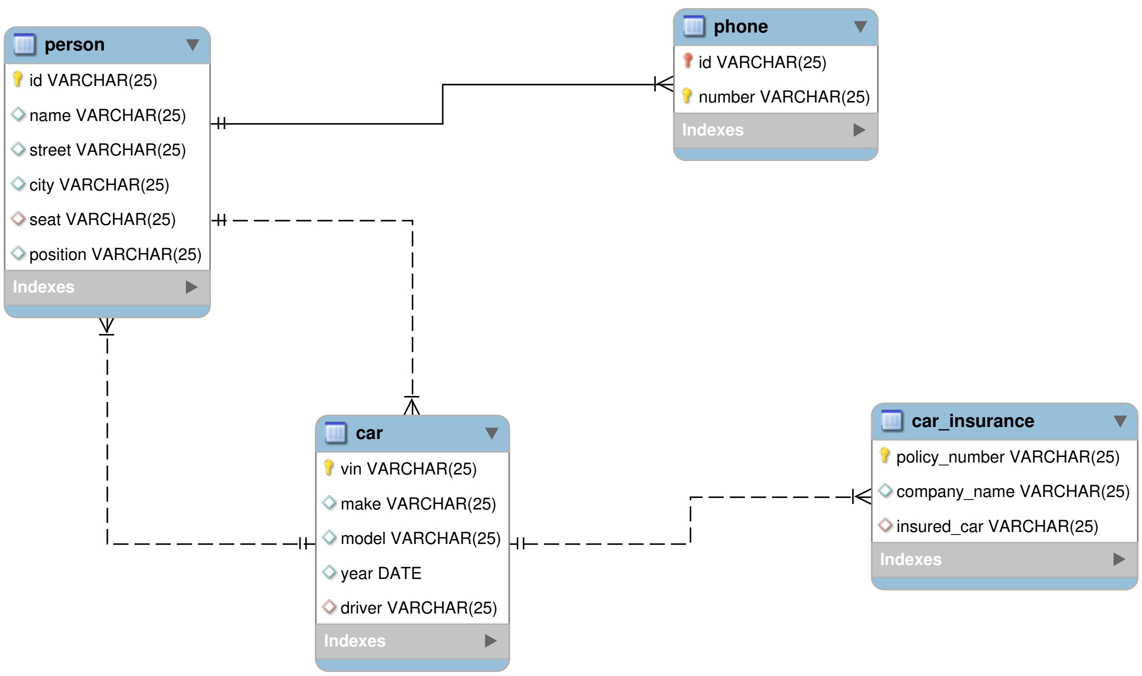


Figure 4.6: mysql Workbench Diagram

5 Database Applications

Resources

- https://spots.augusta.edu/caubert/teaching/general/java/
- If you experience any trouble, https://www.ntu.edu.sg/home/ehchua/programming/howto/ErrorMessages.html#JDBCErrors might be a good read.
- (Elmasri and Navathe 2010, 13.3.2) or (Elmasri and Navathe 2015, chap. 10) is a condensed, but good, read.
- Many textbooks on Java include a part on Databases, just like this one: (Gaddis 2014, chap. 16).

5.1 Overview

Two options to interact with a database:

- Interactive interfaces (command line interface and graphical user interface like workbench[1]): what we used so far.
- Application program / Database application, what we will study. They can be of several types:

  1. Embeded `SQL`[2]: the idea is to embed `SQL` commands directly in the program: a pre-compiler scans the code, extact the `SQL` commands, execute them on the DBMS. This system is used primarily for C, C++, COBOL or Fortran, and **L**anguage **In**tegrated **Q**uery[3] to some extend is part of this approach.
  2. Use a library, or **A**pplication **P**rogramming **I**nterface for accessing the database from application programs.
  3. Create a new language that extends `SQL` (for instance, `PL/SQL`[4]).

In this chapter, we will study how to develop a database application that uses a library.

Every database application follows the same routine:

1. Establish / open the connection with the DBMS,
2. Interact with the DBMS (Update, Query, Delete, Insert),
3. Terminate / close the connection with the DBMS.

Which API is used vary with the pair Language / DBMS. Here are some of the most commonly used pairs for MySQL (that may be compatible with other DBMS in some cases):

[1] https://en.wikipedia.org/wiki/MySQL_Workbench
[2] https://en.wikipedia.org/wiki/Embedded_SQL
[3] https://docs.microsoft.com/en-us/dotnet/standard/using-linq
[4] https://www.oracle.com/database/technologies/appdev/plsql.html

Language	API	Website
Python	Python Database API	https://www.python.org/dev/peps/pep-0249/
C, C++	MySQL C API	https://dev.mysql.com/doc/refman/8.0/en/c-api.html
C#	MySQL Connector/Net	https: //dev.mysql.com/downloads/connector/net/8.0.html
Java	Java DataBase Connectivity	https://docs.oracle.com/javase/9/docs/api/java/sql/package-summary.html

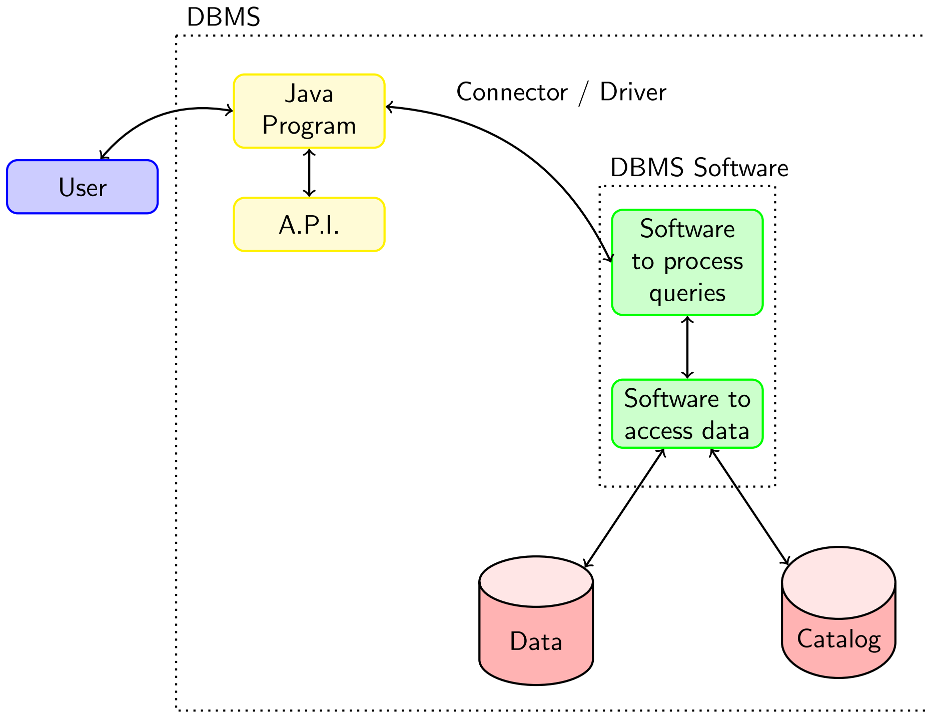
In this chapter, we will *more precisely* study how to develop a database application *coded in Java* that uses *the Java DataBase Connectivity* library. If you were to work with a different API in your future life, you would likely realize that most of what we will be studying remains true: reading the documentation and understanding the general strategy is what matters in this chapter, to build comfidence in your capacities.

5.2 Java's Way

Java actually uses

- A **protocol** (the API, a class library), Java DataBase Connectivity (JDBC), common to all DBMS and used at *compilation time*. Essentially, it is a collection of classes to send `SQL` statements, retrieve and update the results of a query, handle exceptions, etc.
- A **subprotocol** (the driver, connector), Connector/J for MySQL, that is used at *execution time*[5].

[5] The situation is similar e.g. in Python, where you have to use an API *and* a connector. Among Python's connector compatible with MySQL's API, there is PyMySQL[6] or mysql-connector-python[7].



Note that the A.P.I. is needed *when you write and compile* your program, and the driver / connector is needed *when you execute it*. We will come back to this when we explore our first program.

5.3 Flash Intro to Java

For a quick introduction to Java, cf. https://spots.augusta.edu/caubert/teaching/general/java/.

5.4 A First Program

We will write and compile a simple java program that manipulates a simple database[8]. Even if the creation and population of the database could have been done from within the program, we will do it as a preliminary step, using the C.L.I., to make our program simpler (and also because it generally match usage: schemas are usually created before the program is executed).

[8]This program ows a lot to the one presented at http://www.ntu.edu.sg/home/ehchua/programming/java/jdbc_basic .html.

5.4.1 The Database (SQL)

For this program, we will use the following database:

```sql
/* code/sql/HW_EBookshop.sql */
DROP SCHEMA IF EXISTS HW_EBookshop;

CREATE DATABASE HW_EBookshop;

USE HW_EBookshop;

CREATE TABLE BOOKS (
  ID INT PRIMARY KEY,
  title VARCHAR(50),
  author VARCHAR(50),
  price DECIMAL(10, 2),
  qty INT
);

-- Cf. https://en.wikipedia.org/wiki/List_of_best-selling_books
INSERT INTO BOOKS
VALUES (
  1,
  'The Communist Manifesto',
  'Karl Marx and
    Friedrich Engels',
  11.11,
  11);

INSERT INTO BOOKS
VALUES (
  2,
  'Don Quixote',
  'Miguel de Cervantes',
  22.22,
  22);

INSERT INTO BOOKS
VALUES (
  3,
  'A Tale of Two Cities',
  'Charles Dickens',
  33.33,
  33);

INSERT INTO BOOKS
VALUES (
  4,
  'The Lord of the Rings',
  'J. R. R. Tolkien',
```

```
  44.44,
  44);

INSERT INTO BOOKS
VALUES (
  5,
  'Le Petit Prince',
  'Antoine de
    Saint-Exupéry',
  55.55,
  55);

SELECT *
FROM BOOKS;
```

HW_EBookshop.sql[9]

```
MariaDB [HW_EBookshop]> SELECT * FROM BOOKS;
+----+-------------------------+------------------------------
-+-------+------+
| ID | title                   | author                        | price | qty  |
+----+-------------------------+------------------------------
-+-------+------+
|  1 | The Communist Manifesto | Karl Marx and Friedrich Engels | 11.11 |   11 |
|  2 | Don Quixote             | Miguel de Cervantes           | 22.22 |   22 |
|  3 | A Tale of Two Cities    | Charles Dickens               | 33.33 |   33 |
|  4 | The Lord of the Rings   | J. R. R. Tolkien              | 44.44 |   44 |
|  5 | Le Petit Prince         | Antoine de Saint-
Exupéry      | 55.55 |   55 |
+----+-------------------------+------------------------------
-+-------+------+
5 rows in set (0.00 sec)
```

You can copy and paste the code, then execute it, or use MySQL's batch mode: you can find the code previously given at `code/sql/HW_EBookshop.sql`, i.e., at https://rocketgit.com/user/caubert/CSCI_3410/source/tree/branch/master/blob/notes/code/sql/HW_EBookshop.sql. Open a terminal (or command-line interpreter), navigate to the folder where you stored that file (using `cd`), and type

```
mysql -u testuser -p < HW_EBookshop.sql
```

for linux, or (something like)

```
"C:\Program Files\MySQL\MySQL Server 5.7\bin\mysql.exe" -u
 ↪ testuser -p < HW_EBookshop.sql
```

for Windows. Refer to the *Logging-In as testuser* section if you forgot how to log-in to your database.

[9] https://rocketgit.com/user/caubert/CSCI_3410/source/tree/branch/master/blob/notes/code/sql/HW_EBookshop.sql

You just discovered MySQL's batch mode, that perform *series* of instructions from a file. You can easily make sure that the database and the table were indeed created, and the values inserted, by logging the way you used to, and executing the usual commands.

5.4.2 Executing Database Application

As we are about to see, a database application needs to be written following this order:

1. Load the API,
2. Try to open the connection (i.e., create `Connection` and `Statement` objects), using a try/catch statement,
3. Perform the required actions on the database (using `Statement` object),
4. Close the connection.

and the program needs to load the driver (which is specific to DBMS) at execution time.

Of course, if the second step failed, then the program needs to exit gracefully, or to provide debugging information to the user. The program we will obtain can (normally) be compiled, using something like `javac FirstProg.java` (or an equivalent command for windows). But another refinment is needed when you want to execute it. We need to set up the *driver* (or *connector*) to make the java `SQL` API and MySQL communicate. To do so,

- Go to https://dev.mysql.com/downloads/connector/j/
- Select "Platform Independent",
- Click on "Download" in front of "Platform Independent (Architecture Independent), ZIP Archive"
- Look for the (somewhat hidden) "No thanks, just start my download."
- Download the file named "mysql-connector-java-***.zip", where `***` is the version number.
- Unzip the file, and locate the "mysql-connector-java-***.jar" file (normally, in the root folder).
- Copy that file in the same folder as where you intend to compile your program.

Once this is done and your program was compiled, you can execute it using (where you replace `***` with the actual number, of course, e.g. `8.0.22`):

```
java -cp .:mysql-connector-java-***.jar FirstProg
```

in Linux, or

```
java -cp .;mysql-connector-java-***.jar FirstProg
```

in Windows. The `-cp` option lists the places where java should look for the class used in the program: we are explicitely asking java to use the `mysql-connector-java-***.jar` executable (the driver) to execute our `FirstProg` executable.

If we try to execute `FirstProg` without that flag, we obtain the following error message:

```
$ java FirstProg
java.sql.SQLException: No suitable driver found for
 ↪  jdbc:mysql://localhost:3306/HW_EBOOKSHOP
at java.sql.DriverManager.getConnection(DriverManager.java:689)
at java.sql.DriverManager.getConnection(DriverManager.java:247)
at FirstProg.main(FirstProg.java:9)
```

Two additional observations:

- There are other ways to connect a Java application to a database, cf. https://www.baeldung.com/java-connect-mysql or https://stackoverflow.com/a/2839563 for some examples and guides.
- You can include a routine to check if the driver was correctly loaded with (courtesy of BalusC[10]):

```
    try {
      Class.forName("com.mysql.cj.jdbc.Driver");
      System.out.println("Driver loaded!");
    } catch (ClassNotFoundException e) {
      throw new IllegalStateException("Cannot find the driver in the classpath!", e);
    }
```

TestDriver.java[11]

5.4.3 The Application Program (java)

```
// code/java/FirstProg.java

import java.sql.*;

public class FirstProg {
  public static void main(String[] args) {
    try (Connection conn =
           DriverManager.getConnection(
               "jdbc:mysql://localhost:3306/HW_EBookshop", "testuser", "password");
      Statement stmt = conn.createStatement(); ) {
      String strSelect = "SELECT title, price, qty FROM BOOKS WHERE qty > 40";
      System.out.print("The SQL query is: " + strSelect + "\n");
      ResultSet rset = stmt.executeQuery(strSelect);

      System.out.println("The records selected are:");
      int rowCount = 0;
      String title;
      double price;
      int qty;

      while (rset.next()) {
        title = rset.getString("title");
        price = rset.getDouble("price");
        qty = rset.getInt("qty");
        System.out.println(title + ", " + price + ", " + qty);
        rowCount++;
```

[10] https://stackoverflow.com/a/2840358

[11] https://rocketgit.com/user/caubert/CSCI_3410/source/tree/branch/master/blob/notes/code/java/TestDriver.java

```
        }

        System.out.println("Total number of records = " +
         ↪ rowCount);
        conn.close();

      } catch (SQLException ex) {
        ex.printStackTrace();
      }
    }
}
```

FirstProg.java[12]

Please, note that if at execution time you receive an error that starts with "java.sql.SQLException: The server time zone value 'EDT' is unrecognized or represents more than one time zone. You must configure either the server …" add `?serverTimezone=UTC` at the end of `jdbc:mysql://localhost:3306/HW_EBookshop` i.e., replace the line that creates the `Connection` object with

```
Connection conn =
    DriverManager.getConnection(

         ↪ "jdbc:mysql://localhost:3306/HW_EBookshop?serverTimezone=UTC",
        "testuser","password");
```

For more information, refer to https://stackoverflow.com/q/26515700. You can also change your server's configuration "once and for all", cf.https://stackoverflow.com/a/44720416. On my personnal set-up (Debian with MariaDB), this required to:

- Open as root the file `/etc/mysql/mariadb.conf.d/50-server.cnf`,
- Look for `[mysqld]`,
- Insert below

```
default_time_zone='-04:00'
```

(you can look up your time zone at https://time.is if you are unsure)

- Restart the server, using (still as root)

```
service mysql restart
```

A couple of comments:

- `java.sql.*`, whose documentation is at https://docs.oracle.com/javase/8/docs/api/java/sql/package-summary.html, contains the following classes that we will use in this chapter:
  - `DriverManager`, used for managing a set of JDBC drivers,
  - `Connection`, used to make a connection with a database via `DriverManager` objects,
  - `Statement`, used to send basic `SQL` statements via `Connection` objects,

[12] https://rocketgit.com/user/caubert/CSCI_3410/source/tree/branch/master/blob/notes/code/java/FirstProg.java

- `ResultSet`, to retrieve and update the results of a query, returned by a `Statement` object,
- `ResultSetMetadata`, to get information about a `ResultSet` object,
- `SQLException`, a class of exceptions relative to `SQL`.

- Intuitively, a `Connection` is a bridge (the physical connection), and `Statement` is a lane (a symbolic, or logic, path on the bridge).
- In the string `"jdbc:mysql://localhost:3306/HW_EBOOKSHOP"`,

  - `jdbc` is the protocol,
  - `mysql` is the subprotocol,
  - `localhost` is the url of the database,
  - `3306` is the port, and
  - `HW_EBOOKSHOP` is the schema (that needs to already exist in this case).

- Note that `strSelect` does not end with `;` (it could, but does not have to).
- `next()` returns `true` if there is something left in the set of result, and move to the next line if it is the case. It ressembles what we would use to read from a file. If you try to use `getString` *before* moving to the first row, you'll get an error like

```
java.sql.SQLException: Before start of result set
```

Undeed, the cursor is "above" the first row of results when the `ResultSet` object is created. - We could use `1`, `2`, and `3` instead of `"title"`, `"price"` and `"qty"` in the `while` loop: the `getString`, `getDouble` and `getInt` are overloaded, and have versions that take one integer as input, corresponding to the position of the attribute in the result set.

5.4.4 The Result

If you store the program in `FirstProg.java`, compile it, with

```
javac FirstProg.java
```

and then execute it, with

```
java -cp .:mysql-connector-java-***.jar FirstProg
```

(refer back to "Executing Database Application" for more details) then you should obtain:

```
The `SQL` query is: SELECT title, price, qty FROM BOOKS WHERE
 ↪  qty > 40
The records selected are:
The Lord of the Rings, 44.44, 44
Le Petit Prince, 55.55, 55
Total number of records = 2
```

Take the time to make sure you have the same result on your installation, and that you understand how the code works before moving on.

5.4.5 A Variation

If you were to replace the body of `try` in the previous program with

```
String strSelect = "SELECT * FROM BOOKS";
ResultSet rset = stmt.executeQuery(strSelect);

System.out.println("The records selected are:");

ResultSetMetaData rsmd = rset.getMetaData();
int columnsNumber = rsmd.getColumnCount();
String columnValue;
while (rset.next()) {
  for (int i = 1; i <= columnsNumber; i++) {
    if (i > 1) System.out.print(",  ");
    columnValue = rset.getString(i);
    System.out.print(columnValue + " " + rsmd.getColumnName(i));
  }
  System.out.println();
```

FirstProgBis.java[13]

You would obtain:

```
The records selected are:
1 ID,  The Communist Manifesto title,  Karl Marx and Friedrich
 ↪ Engels author,  11.11 price,  11 qty
2 ID,  Don Quixote title,  Miguel de Cervantes author,  22.22
 ↪ price,  22 qty
3 ID,  A Tale of Two Cities title,  Charles Dickens author,
 ↪ 33.33 price,  33 qty
4 ID,  The Lord of the Rings title,  J. R. R. Tolkien author,
 ↪ 44.44 price,  44 qty
5 ID,  Le Petit Prince title,  Antoine de Saint-Exupéry author,
 ↪ 55.55 price,  55 qty
```

In that code, please note:

- the use of `ResultSetMetadata`,
- that we could "extract" the number of columns in the `ResultSet` using the `getColumnCount` method,
- that we used the `getString` method with integer input to read *all* the data in the table, no matter its "original" data type.

Overall, this code would work equally well if the table had a different number of columns, as opposed to our first program. Note also that `ResultSetMetadata` does *not* contain a method to count the number of rows in the result set: to obtain it, either use a counter like we did with `rowCount` before, or execute a query to obtain this value (using MySQL's `count` aggregate function).

[13]https://rocketgit.com/user/caubert/CSCI_3410/source/tree/branch/master/blob/notes/code/java/FirstProgBis.java

5.5 Mapping Datatypes

Note that in the previous code, we read everything as a string. But, actually, `SQL` and `JAVA` datatypes can be mapped as follows:

SQL	JAVA
`INTEGER`	`int`
`CHARACTER(n)`	`String`
`VARCHAR(n)`	`String`
`REAL`	`float`
`DOUBLE`	`double`
`DECIMAL(t,d)`	`java.math.BigDecimal`
`DATE`	`java.sql.Date`
`BOOLEAN`	`boolean`
`BIT(1)`	`byte`

Remember that in `DECIMAL(t,d)` the `t` stands for the number of digits, the `d` for the precision.

However, we cannot always have a correspondance going the other way around (from Java to SQL): what would correspond to a reference variable? To a private attribute? This series of problems is called "object-relational impedance mismatch", it can be overcomed, but at a cost. We will come back to this in the Presentation of NoSQL Chapter.

5.6 Differences Between `executeQuery`, `executeUpdate` and `execute`

Previously, we used `executeQuery` to send a SQL command to the DBMS. This method is tailored for **`SELECT`** statement, and it is not the only method we can use.

Name	`executeQuery`	`executeUpdate`	`execute`
Used for	**`SELECT`**	**`INSERT`**, **`UPDATE`**, **`DELETE`**	Any type
Input Type	`string`	`string`	`string`
Return Type	`ResultSet`	`int`, the number of rows affected by the querry	`boolean`, `true` if the query returned a `ResultSet`, `false` if the query returned an `int` or nothing

To retrieve the `ResultSet` obtained by an `execute` statement, you need to use `getResultSet` or `getUpdateCount`. For more details, consult https://docs.oracle.com/javase/7/docs/api/java/sql/Statement.html.

5.7 A Second Program

The program in Problem 5.2 (Advanced Java Programming) uses the modifications discussed below. Please refer to it once you are done with this section.

5.7.1 Passing Options

We can pass options (values of fields) when connecting to the database:

```
Connection conn =
      DriverManager.getConnection(
          "jdbc:mysql://localhost:3306/HW_DBPROG"
              + "?user=testuser"
              + "&password=password"
              + "&allowMultiQueries=true"
              + "&createDatabaseIfNotExist=true"
              + "&useSSL=true");
```

AdvancedProg.java[14]

On top of `user` and `password` (which are self-explanatory), setting `allowMultiQueries` to `true` allows to pass multiple queries with one `executeUpdate` statement, and `createDatabaseIfNotExist` creates the *schema* passed in the `url` (so, here, `HW_DBPROG`) if it does not already exists.

The syntax used is the syntax of querying strings[15], i.e., it follows the pattern

```
?field1=value1&field2=value2…&fieldN=valueN
```

That is, it starts with an `?` and then "pile up" the field / value pairs with `&`. In particular, if you needed to add `?serverTimezone=UTC` in the first application program we used, you will need here to replace

```
+ "&useSSL=true");
```

with

```
+ "&useSSL=true"
+ "&serverTimezone=UTC");
```

You can read about other options at https://dev.mysql.com/doc/connector-j/8.0/en/connector-j-reference-configuration-properties.html or https://jdbc.postgresql.org/documentation/head/connect.html. Please, note that `useSSL` does not apply to the inital handshake[16] and that there are no good ways to hide the password from the application user[17]. Using SSL here simply guarantee that the application user will interact in a secure manner with the database, not that the password is secured.

[14]https://rocketgit.com/user/caubert/CSCI_3410/source/tree/branch/master/blob/notes/code/java/AdvancedProg.java

[15]https://en.wikipedia.org/wiki/Query_string#Structure

[16]https://dev.mysql.com/doc/connector-j/8.0/en/connector-j-reference-using-ssl.html

[17]https://stackoverflow.com/a/6981725/

5.7.2 Creating a Table

We can create a table with the method `stmt.execute`.

```
stmt.execute(
    "CREATE TABLE DVD ("
        + "Title CHAR(25) PRIMARY KEY, "
        + "Minutes INTEGER, "
        + "Price DOUBLE)");
```

AdvancedProg.java[18]

If we were to execute **`SHOW TABLES;`** after this `execute` instruction directly in the MySQL interpreter, this would display at the screen:

```
+----------------------+
| Tables_in_HW_DBPROG |
+----------------------+
| DVD                  |
+----------------------+
```

But here, to access this information, we will use the connection's metadata. The `DatabaseMetaData` is a class used to get information about the database: the driver, the user, the versions, etc. We can use the `getMetaData()` method of this class to obtain information about the schema we just created:

```
DatabaseMetaData md = conn.getMetaData();

ResultSet rs = md.getTables("HW_DBPROG", null, "%", null);
```

AdvancedProg.java[19]

The first parameter of `getMetaData()` is the schema's name, as you probably guessed, and the the third parameter is `String tableNamePattern`, i.e., what must match the table name stored in the database to be selected. Here, by using the wildcard `%`, we select all the table names (which is only "DVD" at this point).

The `getMetaData()` method returns a `ResultSet` (here named `rs`), where `3` is the `TABLE_NAME`. We can now iterate over this `rs` object to list all the elements in it, as we would with any `ResultSet` object:

```
while (rs.next()) {
  System.out.println(rs.getString(3));
}
```

[18] https://rocketgit.com/user/caubert/CSCI_3410/source/tree/branch/master/blob/notes/code/java/AdvancedProg.java

[19] https://rocketgit.com/user/caubert/CSCI_3410/source/tree/branch/master/blob/notes/code/java/AdvancedProg.java

AdvancedProg.java[20]

You can read at https://docs.oracle.com/javase/7/docs/api/java/sql/DatabaseMetaData.html#getTables(java.lang.String,%20java.lang.String,%20java.lang.String,%20java.lang.String%5B%5D) the full specification of this method.

5.7.3 Inserting Values

To insert values in our table, we can use `stmt.executeUpdate`:

```
String sqlStatement = "INSERT INTO DVD VALUES ('Gone With The
 ↪ Wind', 221, 3);";
int rowsAffected = stmt.executeUpdate(sqlStatement);
System.out.print(sqlStatement + " changed " + rowsAffected + "
 ↪ row(s).\n");
```

AdvancedProg.java[21]

Note that the `executeUpdate` returns an integer, the number of rows changed. We can even use this method to perform multiple insertions at the same time, if `allowMultiQueries` was set to true, cf. https://stackoverflow.com/a/10804730/:

```
String insert1 = "INSERT INTO DVD VALUES ('Aa', 129, 0.2)";
String insert2 = "INSERT INTO DVD VALUES ('Bb', 129, 0.2)";

stmt.executeUpdate(insert1 + ";" + insert2);
```

AdvancedProg.java[22]

Another way of "batch processing" statements (i.e., of executing multiple insertions at the same time) is to use `addBatch` (that "loads" statements in the `statement` object) and `executeBatch()` (that execute all the statement loaded):

```
String insert3 = "INSERT INTO DVD VALUES ('Cc', 129, 0.2)";
String insert4 = "INSERT INTO DVD VALUES ('DD', 129, 0.2)";
stmt.addBatch(insert3);
stmt.addBatch(insert4);
stmt.executeBatch();
```

AdvancedProg.java[23]

Note that the database is not sollicited until the `executeBatch` method is called: we simply loaded the instruction in the program, and connect to the database only once, with all the instructions, when this `executeBatch()` instruction is met.

Note also that `executeBatch` may be used, per https://docs.oracle.com/javase/tutorial/jdbc/basics/retrieving.html#batch_updates:

[20] https://rocketgit.com/user/caubert/CSCI_3410/source/tree/branch/master/blob/notes/code/java/AdvancedProg.java

[21] https://rocketgit.com/user/caubert/CSCI_3410/source/tree/branch/master/blob/notes/code/java/AdvancedProg.java

[22] https://rocketgit.com/user/caubert/CSCI_3410/source/tree/branch/master/blob/notes/code/java/AdvancedProg.java

[23] https://rocketgit.com/user/caubert/CSCI_3410/source/tree/branch/master/blob/notes/code/java/AdvancedProg.java

> for updating, inserting, or deleting a row; and it may also contain DDL statements such as **CREATE TABLE** and **DROP TABLE**. It cannot, however, contain a statement that would produce a `ResultSet` object, such as a **SELECT** `statement`.

Note that using batches does not require to set `allowMultiQueries` to true[24].

Also, the name suggests that it should be possible to fetch the `SQL` instructions from a file and load them in your `Java` program, but there is actually no easy way to do this, c.f. https://stackoverflow.com/q/2071682/.

5.7.4 Prepared Statements

A prepared statement is "a query with a slot": it is a query that takes one or multiple parameters, is parsed and stored on the database, but not executed. It is only *after* the value of the slot(s) are fixed by the program that this query can be executed. The program can re-use the same prepared statement with multile (different) values multiple times.

Compared to executing `SQL` statements directly, prepared statements have three main advantages:

- They reduce parsing time (we store the prepared statement only once, VS as many time as there are values),
- They minimize bandwidth usage (once the prepared statement is sent, the server needs only the parameters, and not the whole query again),
- They protect against `SQL` injections (cf. A Bit About Security).

Let us look at a first example:

```

/*
 * We create a string with an empty slot,
 * represented by "?".
 */
sqlStatement = "SELECT title FROM DVD WHERE Price <= ?";
/*
 * We create a PreparedStatement object, using that string with an
 * empty slot.
 */
PreparedStatement ps = conn.prepareStatement(sqlStatement);

/*
 * Then, we "fill" the first slot with the value of a variable.
 */
double maxprice = 0.5;
ps.setDouble(1, maxprice);
/*
 * Finally,  we can execute the query, and display the results.
 */
ResultSet result = ps.executeQuery();
```

[24] https://dev.mysql.com/doc/connector-j/8.0/en/connector-j-connp-props-security.html

```
System.out.printf("For %.2f you can get:\n", maxprice);

while (result.next()) {
  System.out.printf("\t %s \n", result.getString(1));
}
```

AdvancedProg.java[25]

Note that once the `ps PreparedStatement` object is created, we cannot change the content of the query, beside instantiating the slot. cf. e.g. the discussion at https://stackoverflow.com/q/25902881/.

As we said earlier, a prepared statement can have multiple “slots”, as we can see in that second example:

```
sqlStatement = "INSERT INTO DVD VALUES (?, ?, ?)";
// Now, our string has 3 empty slots, and it is an INSERT statement.
PreparedStatement preparedStatement = conn.prepareStatement(sqlStatement);

preparedStatement.setString(1, "The Great Dictator");
preparedStatement.setInt(2, 124);
preparedStatement.setDouble(3, 5.4);

rowsAffected = preparedStatement.executeUpdate();
/* You can check "by hand" that this statement was correctly
 * executed. Note that the toString method is quite verbose.
 */
System.out.print(preparedStatement.toString() + " changed " + rowsAffected + " row(s).\n");
```

AdvancedProg.java[26]

Where we stored the integer value returned by `executeUpdate` and displayed the the prepared statement using the`toString` method.

If we try to mess things up, i.e., provide wrong datatypes:

```
preparedStatement.setString(1, "The Great Dictator");
preparedStatement.setString(2, "Not-an-integer");
preparedStatement.setString(3, "Not-a-double");

/* This command will make your program crash:
 * rowsAffected = preparedStatement.executeUpdate();
 */
```

[25] https://rocketgit.com/user/caubert/CSCI_3410/source/tree/branch/master/blob/notes/code/java/AdvancedProg.java

[26] https://rocketgit.com/user/caubert/CSCI_3410/source/tree/branch/master/blob/notes/code/java/AdvancedProg.java

AdvancedProg.java[27]

Java compiler will be ok, but we'll have an error at execution time when executing the query.

Executing `rowsAffected = preparedStatement.executeUpdate();` would return an error containing

```
com.mysql.cj.jdbc.exceptions.MysqlDataTruncation: Data truncation: Incorrect ir
an-integer' for column `HW_DBPROG`.`DVD`.`Minutes` at row 1
```

since `"Not-an-integer"` is not … a valid integer!

Of course, prepared statements are particularly convenient when you want to automate some tasks or repeat them multiple times, as you write the query only once, and then re-use it. For instance, inserting the whole "Saw" franchise can be made into a loop:

```java
for (int i = 1; i < 5; i++) {
  preparedStatement.setString(1, "Saw " + i);
  preparedStatement.setInt(2, 100);
  preparedStatement.setDouble(3, .5);
  preparedStatement.executeUpdate();
}
```

AdvancedProg.java[28]

5.7.5 More Complex Statement Objects

When you create the `Statement` objects, you can give two arguments to the `createStatement` method:

```java
Statement stmtNew =
    conn.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,
 ↪  ResultSet.CONCUR_UPDATABLE);
```

AdvancedProg.java[29]

Those options change two things about the ResultSet we obtain using this statement The first argument indicates whenever you can scroll (go forward *and* backward) in the `ResultSets` objects that will be created using this `Statement` object:

- `TYPE_FORWARD_ONLY` is the default (you can only move forward).
- `TYPE_SCROLL_INSENSITIVE` means that you can scroll, but that updates don't impact result set.
- `TYPE_SCROLL_SENSITIVE` means that you can scroll, and that updates impact result set.

[27] https://rocketgit.com/user/caubert/CSCI_3410/source/tree/branch/master/blob/notes/code/java/AdvancedProg.java

[28] https://rocketgit.com/user/caubert/CSCI_3410/source/tree/branch/master/blob/notes/code/java/AdvancedProg.java

[29] https://rocketgit.com/user/caubert/CSCI_3410/source/tree/branch/master/blob/notes/code/java/AdvancedProg.java

Allowing to go in both direction extends the methods one can use in the `ResultSet` class: now, to scrool through the results, one can use:

- `first()`
- `last()`
- `next()`
- `previous()`
- `relative(x)` : move cursor x times (positive = forward, negative = backward)
- `absolute(x)`: move to the row number x, where 1 is the first.

The second argument is the concurrency level, it indicates whenever you can update the values into the `ResultSet` directly.

- `CONCUR_READ_ONLY` is the default.
- `CONCUR_UPDATABLE` means that we can change the database without issuing SQL statement.

In other terms, manipulting the `ResultSet` object will *directly* impact the data stored in the database if we set the second parameter to `CONCUR_UPDATABLE`.

This `createStatement` method is documented at https://docs.oracle.com/javase/7/docs/api/java/sql/Connection.html#createStatement(int,%20int).

You can find below a simple example of "scrollable" `ResultSet`:

```
// code/java/ScrollingProgram.java

import java.sql.*;

public class ScrollingProgram {
  public static void main(String[] args) {
    try (Connection conn =
            DriverManager.getConnection(
                // We connect to the database, not to a
                 ↪ particular schema.
                "jdbc:mysql://localhost:3306/"
                    + "?user=testuser"
                    + "&password=password"
                    + "&allowMultiQueries=true"
                /*
                 * We want to allow multiple statements
                 * to be shipped in one execute() call.
                 */
                );
        Statement stmt =
          conn.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,
  ↪ ResultSet.CONCUR_READ_ONLY);
        /*
         * Finally, we want to be able to move back and forth in
  ↪ our
         * ResultSets. This implies that we have to also chose
  ↪ if the
```

```
         * ResultSets will be updatable or not: we chose to have them
         * to be "read-only".
         */
        ) {
      /*
       * Before you ask: no, there are no "simple" way of
       * constructing a string over multiple lines,
       * besides concatenating them,
       * cf. e.g. https://stackoverflow.com/q/878573
       */

      stmt.execute(
          "DROP SCHEMA IF EXISTS HW_SCROLLABLE_DEMO;"
              +
              /*
               * We drop the schema we want to use if it already exists.
               * (This allows to execute the same program multiple times.)
               */
              "CREATE SCHEMA HW_SCROLLABLE_DEMO;"
              + "USE HW_SCROLLABLE_DEMO;"
              +
              // We create and use the schema.
              "CREATE TABLE TEST("
              + "    Id INT"
              + ");"
          // The schema contains only one very simple table.
          );
      /*
       * We can execute all those queries at once
       * because we passed the "allowMultiQueries=true"
       * token when we created the Connection object.
       */

      // Let us insert some dummy values in this dummy table:
      for (int i = 0; i < 10; i++) stmt.addBatch("INSERT INTO TEST VALUES (" + i + ")");
      /*
       * no ";" in the statements that we add
       * to the batch!
       */
      stmt.executeBatch();
      // We execute the 10 statements that were loaded at once.

      // Now, let us write a simple query, and navigate in the result:
      ResultSet rs = stmt.executeQuery("SELECT * FROM TEST");
      /*
```

```
       * We select all the tuples in the table.
       * If we were to execute this instruction on the
       * command-line interface, we would get:

       * MariaDB [HW_SCROLLABLE_DEMO]> SELECT * FROM TEST;
       * +----+
       * | Id |
       * +----+
       * | 0  |
       * | 1  |
       * | 2  |
       * | 3  |
       * | 4  |
       * | 5  |
       * | 6  |
       * | 7  |
       * | 8  |
       * | 9  |
       * +----+
       * 10 rows in set (0.001 sec)
       */

      // We can "jump" to the 8th result in the set:
      rs.absolute(8);
      System.out.printf("%-22s %s %d.\n", "After absolute(8),",
        ↪ "we are at Id", rs.getInt(1));
      /* Note that this would display "7" since the
       * 8th result contains the value 7 (sql starts
       * counting at 1.
       */

      // We can move back 1 item:
      rs.relative(-1);
      System.out.printf("%-22s %s %d.\n", "After relative(-1),",
        ↪ "we are at Id", rs.getInt(1));

      // We can move to the last item:
      rs.last();
      System.out.printf("%-22s %s %d.\n", "After last(),", "we
        ↪ are at Id", rs.getInt(1));

      // We can move to the first item:
      rs.first();
      System.out.printf("%-22s %s %d.\n", "After first(),", "we
        ↪ are at Id", rs.getInt(1));

      conn.close();
    } catch (SQLException ex) {
      ex.printStackTrace();
    }
```

```
  }
}
```

ScrollingProgram.java[30]

You can also have a look at the end of `code/java/AdvancedProg.java`, which creates a second `Statement` object is created and used.

5.8 A Delicate Balance

Forgetting about the technical difficulties for a minute, there is always the issue of finding the right balance between what the application, and what the database, should do. Any control structure should be dealt with by the application, and queries (such as select project joins) should be done by the DBMS, but the line may be a bit blurry at times. For instance, should the schema being created from the application? Probably yes if this is an operation that needs to be performed repeatedly (to "reboot" your schema), or if you want your application to be as portable as possible. Otherwise, it may make little sense.

Another question is: if a task need to be performed repeatedly, should you create a method in the application, or a procedure in the DBMS? Once again, it will depend: if you need to read information from the user or if using control flow is crucial to your task, then a method seems more adequate. But if the task is essentially a series of queries, then creating a procedure may have benefits:

- A procedure will be accessible from any application, and from the command-line as well,
- A procedure will be pre-parsed and optimized on the database side,
- A procedure can serve as a "bridge" between a schema that may evolve and an application that requires resilience from the DBMS. Typically, if two attributes are swapped in the schema, it may not impact the order in which the arguments are passed to the procedure, which ultimately save from having to edit the application.

As an example of how to technically declare and use a procedure from an application, refer to the following code:

```
// code/java/CallProcedure.java

import java.sql.*;

public class CallProcedure {
  public static void main(String[] args) {
    try (Connection conn =
            DriverManager.getConnection(
                "jdbc:mysql://localhost:3306/HW_CALL_TEST"
                    + "?user=testuser"
                    + "&password=password"
                    + "&allowMultiQueries=true"
                    + "&createDatabaseIfNotExist=true"
                    + "&useSSL=true");
```

[30] https://rocketgit.com/user/caubert/CSCI_3410/source/tree/branch/master/blob/notes/code/java/ScrollingProgram.java

```
          Statement stmt = conn.createStatement(); ) {
        stmt.execute(
            "DROP SCHEMA IF EXISTS HW_CALL_TEST;"
                + "CREATE SCHEMA HW_CALL_TEST;"
                + "USE HW_CALL_TEST;");

        stmt.execute("CREATE TABLE Test1 (A INT PRIMARY KEY);");

        stmt.execute("INSERT INTO Test1 VALUES (1), (2), (3);");
        // To create a procedure, we don't need to change the delimiter!
        // Cf. https://stackoverflow.com/a/5314879/ for instance.
        stmt.execute(" CREATE PROCEDURE List () BEGIN SELECT * FROM Test1; END; ");
        // We create a CallabaleStatement object
        // https://docs.oracle.com/javase/7/docs/api/java/sql/CallableStatemen
        // that extends PreparedStatements and allow to call procedures.
        CallableStatement cs = conn.prepareCall("CALL List()");
        ResultSet rset = cs.executeQuery();
        while (rset.next()) {
          System.out.println("The value of A is " + rset.getInt("A") + ".");
        }

        // Second example of procedure, with arguments
        stmt.execute(
            " CREATE PROCEDURE ListGreaterThan(arg INT) "
                + " BEGIN "
                + "   SELECT * "
                + "   FROM Test1 "
                + "   WHERE A > arg; "
                + " END; ");
        cs = conn.prepareCall("CALL ListGreaterThan(?)");
        // Note that we use the same "?" placeholder
        // than we used for prepared statement.

        // We declare an int variable for the argument
        // for the sake of clarity, but don't need to.
        int x = 2;
        // We set the value of the first "? slot"
        // using setInt as for prepared statements.
        cs.setInt(1, x); // 1 is the position

        rset = cs.executeQuery();
        System.out.println("The values of A greater than " + x + " are:");
        while (rset.next()) {
```

```java
        System.out.println("The value of A is " + rset.getInt("A") + ".");
      }

      conn.close();
    } catch (SQLException ex) {
      ex.printStackTrace();
    }
  }
}
```

CallProcedure.java[31]

5.9 Making It Your Own

Now that you have reviewed some of the tools and options in the API, you can start programming and navigating the documentation[32]. As an exercise, you can try to combine prepared statements and batch processing on your own. It is actually fairly immediate!

```java
      PreparedStatement ps = conn.prepareStatement("INSERT INTO TEST VALUES (?);");
      for (int i = 0; i < 10; i++) {
        ps.setInt(1, i);
        ps.addBatch();
      }

      ps.executeBatch();
```

BatchPreparedStatements.java[33]

Exercises

Exercise 5.1 What are the technologies that make it possible for a Java application to communicate with a DBMS?

Exercise 5.2 Why is it important to have the statements creating the connection to the database inside a `try`…`catch` statement?

Exercise 5.3 Name three classes in the SQL API of java.

Exercise 5.4 What JDBC method do you call to get a connection to a database?

Exercise 5.5 Why would somebody want to create multiple `Statement` objects?

Exercise 5.6 What is the class of the object used to create a ResultSet object?

[31] https://rocketgit.com/user/caubert/CSCI_3410/source/tree/branch/master/blob/notes/code/java/CallProcedure.java
[32] on%20your%20own
[33] https://rocketgit.com/user/caubert/CSCI_3410/source/tree/branch/master/blob/notes/code/java/BatchPreparedStatements.java

Exercise 5.7 Briefly explain what the `next()` method from the `ResultSet` class does and give its return type.

Exercise 5.8 Write a statement that execute `SELECT * FROM TEST;` in the DBMS and store the result given by the database in an object. Assume that there is a `Statement` object called `stmt`.

Exercise 5.9 What method should be used to perform an `INSERT` command from your program? In which class is it?

Exercise 5.10 Where is a `ResultSet` object's cursor initially pointing? How do you move the cursor forward in the result set?

Exercise 5.11 Give three navigation methods provided by `ResultSet`.

Exercise 5.12

Explain this JDBC URL format:

```
jdbc:mysql://localhost:3306/HW_NewDB?createDatabaseIfNotExist=true&useSSL=tru
```

Exercise 5.13 In what class is the `getColumnName()` method?

Exercise 5.14 Assuming `stmt` is a `Statement` object, in the statement:

```
modif = stmt.executeUpdate(strC);
```

What is...

1. ... the datatype of `modif`?
2. ... the datatype of `strC`?
3. ... a possible value for `strC`?

Exercise 5.15 Let `stmt` be a statement object, and consider the following:

```
ResultSet rset = stmt.executeQuery("SELECT Name, Price FROM
 ↪ DVD");
```

Write a piece of code that would display at the screen the name and price of the rows present in the `rset` object.

Exercise 5.16 What is a prepared statement?

Exercise 5.17 Give three reasons why prepared statements are preferable over statements.

Exercise 5.18 Assume `ps` is the prepared statement:

```
INSERT INTO EXAM VALUES (?, ?);
```

Write the three statements needed to allocate "Quiz" and "5" to the two slots and to execute the prepared statement in the database.

Exercise 5.19 Briefly explain what `ResultSet.TYPE_SCROLL_SENSITIVE` enables, and where / when it is used.

Exercise 5.20 In the code below, there are five errors between line 13 and line 32. They are *not* subtle Java errors (like misspelling a key word) and do not come from the DBMS (so you should assume that the password is correct, that the database exists, etc.). Highlight each error and explain why it is an error.

```
// code/java/ProgWithErrors.java

import java.sql.*;

public class ProgWithErrors {
  public static void main(String[] args) {
    try (Connection conn =
            DriverManager.getConnection(
                "jdbc:mysql://localhost:3306/" +
                 ↪  "HW_TestDB?user=testuser&password=password");
          Statement stmt = conn.createStatement(); ) {

        // Errors after this point.

        String strSelect = "SELECT title FROM DISKS WHERE qty >
         ↪  40;";
        ResultSet rset = stmt.executeUpdate(strSelect);

        System.out.println("The records selected are: (listed last
         ↪  first):");
        rset.last();

        while (rset.previous()) {
          String title = rset.getDouble("title");
          System.out.println(title + "\n");
        }

        String sss = "SELECT title FROM DISKS WHERE Price <= ?";
        PreparedStatement ps = conn.prepareStatement(sss);
        ResultSet result = ps.executeQuery();

        conn.close();

        // Errors before this point.

      } catch (SQLException ex) {
        ex.printStackTrace();
      }
  }
}
```

ProgWithErrors.java[34]

Exercise 5.21 Write a program that determines if the **null** value from Java code is equal to the **NULL** value in the DBMS.

[34] https://rocketgit.com/user/caubert/CSCI_3410/source/tree/branch/master/blob/notes/code/java/ProgWithErrors.java

Solution to Exercises

Solution 5.1 The technologies theat make it possible for a Java application to communicate with the DBMS are API's and the drivers to implement them.

Solution 5.2 It is important to put the statements that create the connection to the database inside the `try...catch` statement because the program will interact with the environment if this interraction fails (typically, if the connection does not succeed), for which we want to be able to catch the exception and recover from that failure.

Solution 5.3 There are many classes in the SQL API if Java. There are `Connection`, `DatabaseMetaData`, `ResultSetMetaData`, `PreparedStatement`, and `Statement` to name a few. You can find them listed at https://docs.oracle.com/javase/7/docs/api/java/sql/package-summary.html.

Solution 5.4 The JDBC method that must be called to connect to a database is DriverManager.getConnection()

Solution 5.5 You may want to create multiple `Statement` object for multiple reasons: to use parallelism (you could use a statement while another is still being processed), to have different policies (some objects could have update rights, some could not), to connect to multiple databases.

Solution 5.6 The class of the object used to create a `ResultSet` object is the `Statement` class. A `Statement` object is used to create a `ResultSet` object, e.g. by calling the `executeQuery` method.

Solution 5.7 The `next()` method checks if there is data to read and, if there is, it moves the cursor to read it. Its return type is Boolean.

Solution 5.8 You execute a `SELECT` statement and store its returning value in a `ResultSet` object using

```
{.java}. ResultSet rset = stmt.executeQuery("SELECT * FROM TEST;");
```

Solution 5.9 The `executeUpdate()` or `execute()` methods can be used to perform an `INSERT` command from our program. They are in the `Statement` and in the `PreparedStatement` classes.

Solution 5.10 The `ResultSet` object's cursor is initially pointing at the position before the first line. We move the cursor forward by using the `next()` method.

Solution 5.11 There are many navigation methods provided by `ResulSet`. They are the `first()`, `last()`, `next()`, `previous()`, `relative()`, and `absolute()` methods.

Solution 5.12 This JDBC URL format connects to `localhost:3306`, creates a new database if needed, and uses the secure SSL connection.

Solution 5.13 The `getColumnName()` method is in the `ResultSetMetaData` class.

Solution 5.14 In the statement `modif = stmt.executeUpdate(strC);`...

1. `modif` is an integer (the number of rows modified by the query).
2. `strC` is a `String` (a `SQL` command).
3. A possible value for `strC` is `DELETE FROM BOOKS Where Price > 0.5`.

Solution 5.15 We could use the following:

```
while(rset.next()){
    System.out.println("The name is "
    + rset.GetString("Name") + " and the price is "
    + rset.GetDouble("Price") + ".");
}
```

Solution 5.16 A prepared statement is a feature used to execute `SQL` statements repeatedly with high efficiency that protects against `SQL` injections.

Solution 5.17 A prepared statement offers a protection against `SQL` injection, mutualize the work (you write the query only once, and then re-use it), reduces the bandwith usage, and reduce the parsing time on the DBMS (the query is parsed only once as opposed to every time a statement is sent, no matter how similar to the previous one it is).

Solution 5.18

```
ps.setString(1, "Quiz");
ps.setInt(2, 5);
ps.execute();
```

Solution 5.19 `ResultSet.TYPE_SCROLL_SENSITIVE` is used as the first argument of the `createStatement` method from the `Connection` class. The official documentation[35] reads

> The result can be scrolled; its cursor can move both forward and backward relative to the current position, and it can move to an absolute position. The result set reflects changes made to the underlying data source while the result set remains open.

which means that it is possible to use e.g. the `previous()` method to scroll *up* in the ResultSet objects created using that statement.

Solution 5.20 The errors are:

- The first highlighted portion is found in the `ResultSet` object creation line and should only be this part:

```
stmt.executeUpdate(strSelect);
```

The error is that the `executeUpdate()` method cannot be used to perform **`SELECT`** statements.

- The second highlighted portion is found in the while loop condition statement:

```
rset.previous()
```

This error *is* subtle: we need to display the last record before using the `previous()` method, otherwise it would be skipped. We can fix this using a **`do`**…**`while`** loop.

- The third highlighted portion is found in the creation statement for the `String` object named `title`:

[35] https://docs.oracle.com/javase/tutorial/jdbc/basics/retrieving.%20html

```
String title = rset.getDouble("title");
```

The error is that the `getDouble()` method returns a `double`, which cannot be stored as a `String`.

- The fourth highlighted portion is found in the creation statement for the `ResulSet` object named `result`:

```
ps.executeQuery();
```

The error here comes from the previous prepared statement that did not receive a value for the ?.

~~~~
You can find the corrected program in `code/java/ProgWithErrorsPatched.java`, which looks like:

```{.bash}
16c16
<       ResultSet rset = stmt.executeUpdate(strSelect);
---
>      ResultSet rset = stmt.executeQuery(strSelect); // Error 1
21,24c21,24
<       while(rset.previous()) {
<         String title = rset.getDouble("title");
<         System.out.println(title + "\n");
<       }
---
>       do { // Error 2
>         String title = rset.getString("title"); // Error 3
>         System.out.println(title); // Not an error, but we probably do not need two new lines.
>       }while(rset.previous()); // Error 2 bis
27a28
>       ps.setInt(1, 10); // Error 4
```
~~~~

Solution 5.21 Here is what the program should look like:

```java
// code/java/TestForNull.java

import java.sql.*;

public class TestForNull {
  public static void main(String[] args) {
    try (Connection conn =
            DriverManager.getConnection(
                    "jdbc:mysql://localhost:3306/HW_DBPROG?user=testuser&pass
        Statement stmt = conn.createStatement(); ) {
      stmt.execute("CREATE TABLE Test (" + "A CHAR(25), " + "B INTEGER, " + "C DOUBLE)");

```

```
      String strAdd = "INSERT INTO Test VALUES (NULL, NULL, NULL);";
      int number_of_row_changed = stmt.executeUpdate(strAdd);
      System.out.print("This last query changed " + number_of_row_changed + " row(s).\n");

      ResultSet result = stmt.executeQuery("SELECT * FROM Test");

      if (result.next()) {
        System.out.print(result.getString(1) + " " + result.getDouble(2) + " " + result.getInt(3));
        if (result.getString(1) == null) {
          System.out.print("\nAnd null for CHAR in SQL is null for String in Java.\n");
        }
      }
      conn.close();
    } catch (SQLException ex) {
      ex.printStackTrace();
    }
  }
}
```

TestForNull.java[36]

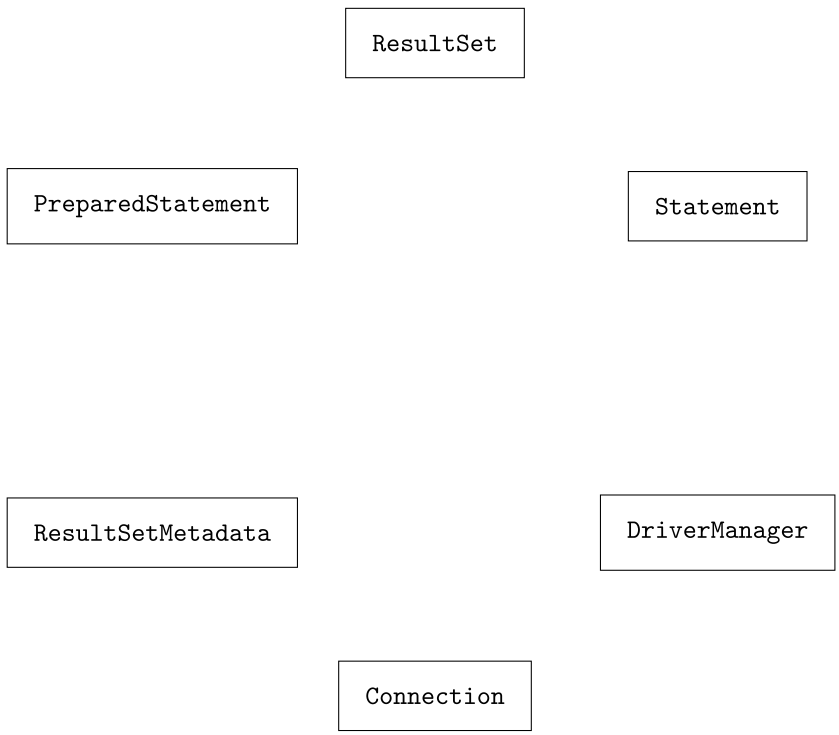
This program should display:

```
This last query changed 1 row(s).
null 0.0 0
And null for CHAR in `SQL` is null for String in Java.
```

5.10 Problems

Problem 5.1 ***(Classes Relationships)*** Draw an arrow from class `A` to class `B` when in the Java `SQL` API a method from class `A` can be used to create an object from class `B`.

[36] https://rocketgit.com/user/caubert/CSCI_3410/source/tree/branch/master/blob/notes/code/java/TestForNull.java



Problem 5.2 ***(Advanced Java Programming)*** Read, execute, break, edit, compile, patch, hack and (most importantly) understand the following program:

```java
// code/java/AdvancedProg.java

/*
 * This is a long program, introducing:
 * I. How to pass options when connecting to the database,
 * II. How to create a table and read its meta-data,
 * III. How to insert values,
 * IV. How to use prepared statements,
 * V. How to read backward and write in ResultSets.
 *
 * To be able to execute this program multiple times, the schema is dropped and re-created.
 *
 */

import java.sql.*;

public class AdvancedProg {
  public static void main(String[] args) {
    try (
```

```
    // I. Passing options to the database

    // start snippet passing-options
    Connection conn =
            DriverManager.getConnection(
                "jdbc:mysql://localhost:3306/HW_DBPROG"
                    + "?user=testuser"
                    + "&password=password"
                    + "&allowMultiQueries=true"
                    + "&createDatabaseIfNotExist=true"
                    + "&useSSL=true");
        // end snippet passing-options

        Statement stmt = conn.createStatement(); ) {
      /*
       * Below, we drop the schema and re-create it to allow multiple execution of the
       * program. You can ignore this part if you want.
       */

      stmt.execute(
          "DROP SCHEMA IF EXISTS HW_DBPROG;" + "CREATE SCHEMA HW_DBPROG;" + "USE HW_DBPROG;");

      // II. Creating a table and reading its meta-data

      // start snippet table-creation
      stmt.execute(
          "CREATE TABLE DVD ("
              + "Title CHAR(25) PRIMARY KEY, "
              + "Minutes INTEGER, "
              + "Price DOUBLE)");
      // end snippet table-creation

      // start snippet table-metadata-1
      DatabaseMetaData md = conn.getMetaData();

      ResultSet rs = md.getTables("HW_DBPROG", null, "%", null);
      // end snippet table-metadata-1

      // start snippet table-metadata-2
      while (rs.next()) {
        System.out.println(rs.getString(3));
      }
      // end snippet table-metadata-2

      // III. Inserting values

      // start snippet inserting-1
```

```
        String sqlStatement = "INSERT INTO DVD VALUES ('Gone With
         ↪  The Wind', 221, 3);";
        int rowsAffected = stmt.executeUpdate(sqlStatement);
        System.out.print(sqlStatement + " changed " + rowsAffected
         ↪  + " row(s).\n");
        // end snippet inserting-1

        // start snippet inserting-2
       String insert1 = "INSERT INTO DVD VALUES ('Aa', 129, 0.2)";
       String insert2 = "INSERT INTO DVD VALUES ('Bb', 129, 0.2)";

        stmt.executeUpdate(insert1 + ";" + insert2);
        // end snippet inserting-2

        // start snippet inserting-3
       String insert3 = "INSERT INTO DVD VALUES ('Cc', 129, 0.2)";
       String insert4 = "INSERT INTO DVD VALUES ('DD', 129, 0.2)";
        stmt.addBatch(insert3);
        stmt.addBatch(insert4);
        stmt.executeBatch();
        // end snippet inserting-3

        // IV. Prepared Statements

        // start snippet prepared-queries-1

        /*
         * We create a string with an empty slot,
         * represented by "?".
         */
        sqlStatement = "SELECT title FROM DVD WHERE Price <= ?";
        /*
         * We create a PreparedStatement object, using that string
↪   with an
         * empty slot.
         */
        PreparedStatement ps =
         ↪  conn.prepareStatement(sqlStatement);

        /*
         * Then, we "fill" the first slot with the value of a
↪   variable.
         */
        double maxprice = 0.5;
        ps.setDouble(1, maxprice);
        /*
         * Finally,  we can execute the query, and display the
↪   results.
         */
        ResultSet result = ps.executeQuery();
```

```

    System.out.printf("For %.2f you can get:\n", maxprice);

    while (result.next()) {
      System.out.printf("\t %s \n", result.getString(1));
    }
    // end snippet prepared-queries-1

    // start snippet prepared-queries-2
    sqlStatement = "INSERT INTO DVD VALUES (?, ?, ?)";
    // Now, our string has 3 empty slots, and it is an INSERT statement.
    PreparedStatement preparedStatement = conn.prepareStatement(sqlStatement);

    preparedStatement.setString(1, "The Great Dictator");
    preparedStatement.setInt(2, 124);
    preparedStatement.setDouble(3, 5.4);

    rowsAffected = preparedStatement.executeUpdate();
    /* You can check "by hand" that this statement was correctly
     * executed. Note that the toString method is quite verbose.
     */
    System.out.print(preparedStatement.toString() + " changed " + rowsAffected + " row(s).\n");
    // end snippet prepared-queries-2

    // start snippet prepared-queries-3
    preparedStatement.setString(1, "The Great Dictator");
    preparedStatement.setString(2, "Not-an-integer");
    preparedStatement.setString(3, "Not-a-double");

    /* This command will make your program crash:
     * rowsAffected = preparedStatement.executeUpdate();
     */
    // end snippet prepared-queries-3

    // start snippet prepared-queries-4
    for (int i = 1; i < 5; i++) {
      preparedStatement.setString(1, "Saw " + i);
      preparedStatement.setInt(2, 100);
      preparedStatement.setDouble(3, .5);
      preparedStatement.executeUpdate();
    }
    // end snippet prepared-queries-4

    // V. Reading backward and writing in ResultSets

```

```
    // start snippet new-statement-1
    Statement stmtNew =
        conn.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE, ResultSet.CONCUR_UPDATABLE);
    // end snippet new-statement-1

    // Reading backward
    sqlStatement = "SELECT title FROM DVD WHERE Price < 1;";
    result = stmtNew.executeQuery(sqlStatement);

    System.out.println("For $1, you can get:");

    if (result.last()) {
      // We can jump to the end of the ResultSet
      System.out.print(result.getString("Title") + " ");
    }

    System.out.print("and also, (in reverse order)");

    while (result.previous()) {
      // Now we can scroll back!
      System.out.print(result.getString("Title") + " ");
    }

    // Changing the values
    System.out.print("\n\nLet us apply a 50% discount. Currently, the prices are:\n");

    sqlStatement = "SELECT title, price FROM DVD;";
    result = stmtNew.executeQuery(sqlStatement);
    while (result.next()) {
      System.out.printf("%20s \t $%3.2f\n", result.getString("title"), result.getDouble("price"));
    }

    // We need to scroll back!
    result.absolute(0);

    while (result.next()) {
      double current = result.getDouble("price");
      result.updateDouble("price", (current * 0.5));
      result.updateRow();
    }
    System.out.print("\n\nAfter update, the prices are:\n");

    // We need to scroll back!
    result.absolute(0);

    while (result.next()) {
```

```java
          System.out.printf("%20s \t $%3.2f\n", result.getString("title"), result.getDouble("price"));
        }

        conn.close();
      } catch (SQLException ex) {
        ex.printStackTrace();
      }
    }
}
```

AdvancedProg.java[37]

Problem 5.3 ***(A GUEST Java Program)*** Consider the code below:

```java
// code/java/GuestProgram.java

// java.util.Scanner is an API to read from the keyboard.
import java.sql.*;
import java.util.Scanner;

// This first part is "standard". Just note that we allow multiple statements.
public class GuestProgram {
  public static void main(String[] args) {
    try (Connection conn =
            DriverManager.getConnection(
                    "jdbc:mysql://localhost:3306/?user=testuser&password=pass
                    + "&allowMultiQueries=true");
          Statement stmt = conn.createStatement(); ) {
        // We create a schema, use it, create two tables, and insert a value in the second one.
        stmt.execute(
            "CREATE SCHEMA HW_GUEST_PROGRAM;"
                + "USE HW_GUEST_PROGRAM;"
                + "CREATE TABLE GUEST("
                + "Id INT PRIMARY KEY,"
                + "Name VARCHAR(30),"
                + "Confirmed BOOL"
                + ");"
                + "CREATE TABLE BLACKLIST("
                + "Name VARCHAR(30)"
                + ");"
```

[37] https://rocketgit.com/user/caubert/CSCI_3410/source/tree/branch/master/blob/notes/code/java/AdvancedProg.java

```java
                + "INSERT INTO BLACKLIST VALUES (\"Marcus
                 ↪  Hells\");");

        /*
         * INSERT HERE Solution to exercises 1, 2 and 3.
         * Tip for Exercise 1, this solves the first item.
         */
        System.out.print("How many guests do you have?\n");
        Scanner key = new Scanner(System.in);
        int guest_total = key.nextInt();

      } catch (SQLException ex) {
        ex.printStackTrace();
      }
    }
}
```

GuestProgram.java[38]

In the following three exercises, you will add some code below the comment `// INSERT HERE Solution to e` in order to obtain a behavior like the following one (you do not have to reproduce it exactly!). The user input is underlined, and hitting "enter" is represented by ↵:

```
How many guests do you have?
2↵
Enter name of guest 1.
Marcus Hells↵
Enter name of guest 2.
Cynthia Heavens↵
..............⌛..............
Oh no, (at least) one of the guest from the black list confirmed their presence!
The name of the first one is Marcus Hells.

Do you want to remove all the guests that are on the black list and who have confir
their presence? Enter "Y" for yes, anything else for no.
```

You should suppose that `BLACKLIST` contains more than one name, and that some other operations are performed where ....⌛.............. is (typically, some guests will confirm their presence). Using batch processing or prepared statements will be a plus, but is not mandatory to solve these exercises.

1. Write a snippet that
    a) Asks the user how many guests they have,
    b) For each guest, asks their name (using `key.nextLine()`, that returns the `String` entered by the user),
    c) For each guest name entered, inserts in the `GUEST` table an integer that is incremented after each insertion, the name entered by the user, and **NULL**.

[38] https://rocketgit.com/user/caubert/CSCI_3410/source/tree/branch/master/blob/notes/code/java/GuestProgram.java

2. Write a snippet such that if there is at least one guest who confirmed their presence and whose name is on the blacklist, a message will be displayed on the screen containing the name of (at least) one of those guests.
3. Write a snippet that asks the user whenever they want to remove from the guest list all the persons on the blacklist that confirmed their presence, and do so if they enter "yes" (or some variation).

Solutions to Selected Problems

Solution to Problem 5.3 (A GUEST Java Program) The file `code/java/GuestProgramSolution.ja` contains the whole code for you to compile and test.

Pb 5.3 – Solution to Q. 1

We explore two solutions, one with batch processing, the second with prepared statement.

They both starts with:

```java
int guest_id;
String guest_name;
int counter = 0;
```

GuestProgramSolution.java[39]

Then the solution using batch processing could be:

```java
while (counter < guest_total) {
  // Ask the name of the guest.
  System.out.print("Enter name of guest " + (counter + 1) +
    ↪  ".\n");
  // Read the name of the guest.
  guest_name = key.nextLine();
  stmt.addBatch("INSERT INTO GUEST VALUES (" + counter + ", \""
 ↪  + guest_name + "\", NULL)");
  // Add to the batch the statement to insert the required data
    ↪  in the table
  counter++;
}
stmt.executeBatch(); // Execute the batch statement.
```

GuestProgramSolution.java[40]

while the solution using prepared statements could be:

```java
PreparedStatement ps = conn.prepareStatement("INSERT INTO GUEST
 ↪  VALUES(?, ?, NULL);");
while (counter < guest_total) {
    System.out.print("Enter name of guest " + (counter + 1) +
      ↪  ".\n");
```

[39]https://rocketgit.com/user/caubert/CSCI_3410/source/tree/branch/master/blob/notes/code/java/GuestProgramSolution.java

[40]https://rocketgit.com/user/caubert/CSCI_3410/source/tree/branch/master/blob/notes/code/java/GuestProgramSolution.java

```
    guest_name = key.nextLine();
    ps.setInt(1, counter);
    ps.setString(2, guest_name);
    ps.executeUpdate();
    counter++;
}
```

Pb 5.3 – Solution to Q. 2

We let `SQL` do all the hard work:

```
ResultSet rset =
    stmt.executeQuery(
        "SELECT * FROM GUEST, BLACKLIST WHERE GUEST.Name =
          ↪  BLACKLIST.Name AND"
            + " GUEST.Confirmed = true");
if (rset.next()) {
  System.out.print(
      "Oh no, (at least) one of the guest from the black list
        ↪  confirmed their presence!\n"
          + "The name of the first one is "
          + rset.getString(2)
          + ".\n");
}
```

GuestProgramSolution.java[41]

Pb 5.3 – Solution to Q. 3

Similarly, we let `SQL` do all the hard work:

```
System.out.print(
    "Do you want to remove all the guests that are on the black
      ↪  list and confirmed their"
        + " presence? Enter \"Y\" for yes, anything else for
          ↪  no.\n");
if (key.nextLine().equals("Y")) {
  stmt.execute(
      "DELETE FROM GUEST WHERE NAME IN (SELECT NAME FROM
        ↪  BLACKLIST) AND Confirmed = true;");
}
```

GuestProgramSolution.java[42]

[41] https://rocketgit.com/user/caubert/CSCI_3410/source/tree/branch/master/blob/notes/code/java/GuestProgramSolution.java

[42] https://rocketgit.com/user/caubert/CSCI_3410/source/tree/branch/master/blob/notes/code/java/GuestProgramSolution.java

6 A Bit About Security

DBMS, as any software, needs to be secured. DBMS, as any online service, needs to be well secured. DBMS, as any place where (possibly confidential) data is stored, needs to be extremely well secured.

In this Chapter, we review some "usual" aspects of security, before focusing on one particular type of attack on DBMS, `SQL` injections.

6.1 Usual Aspects

6.1.1 Threat Model

As usual, a threat model needs to be sketched when designing how your DBMS will be used. It should answer questions like

- Who is threatening you?
- What are the risks?
- What are the type of attacks?

The first question is of importance, as you will not be securing your application the same way depending of if you fear attack from script kiddies[1], competitors, former employee, or government. However, thinking "this system can not possibly be secured against Google's quantum computer, so let's do nothing" is probably giving your system too much importance (Google is not going to waste its resources to hack your database), and counter-productive (you should protect your database against low-level threats in any case).

Risks generally include

- Loss of integrity (improper modification),
- Loss of availability,
- Loss of confidentiality (unauthorized disclosure).

About the type of attacks, DBMS are exposed to many channels. Indeed, they can be targeted by

- the "usual" attacks on programs (e.g. buffer overflow),
- the "usual" attacks on online services (e.g. denial of service),
- the "usual" attacks on systems (e.g. weak authentication, privilege escalation),
- **and** some particular attacks (e.g. `SQL` injections).

We will study those in the second part of this Chapter, but do not forget that other types of vulnerabilities exist as well.

[1] https://en.wikipedia.org/wiki/Script_kiddie

6.1.2 Control Measures

It can be useful to design your control measures for your DBMS, which can include, e.g.

- Access control (user account, passwords, restrictions),
- Inference control (cannot access information about a particular "case"),
- Flow control (prevent indirect access).

6.1.3 Protections

Protection measures are principled and technological. You should always have in mind principles like

- "You are as strong as your weakest link."
- Never trust the user or their computer.
- Systems needs to be up-to-date.
- Options that are not used should be desactivated.
- Use dual-factor authentication when available.
- Stay informed (e.g. read newsfeeds).

Technological measure of protections exist, and should be used. For instance,

- Use mysqldump[2] to create backups of your tables. On our system, it would be something like

  ```
  mysqldump --all-databases - u testuser -p password - h
   ↪  localhost > dump.sql
  ```

- Use encryption, salting *and* hashing when it comes to password and other sensitive data.

- **Do not** let the users connect directly to your database, even through a piece of software you wrote (refer e.g. to https://security.stackexchange.com/q/229954 for a discussion on why this is not a good idea).

If you are not familiar with the concepts of salting and hashing, you can consult e.g. https://crackstation.net/hashing-security.htm. In a nutshell, this is a measure of prevention to protect your users against weak passwords, and to make sure that only an encrypted version of their password will be stored in your database.

6.1.4 How to Recover?

Generally, people are in agreement that the question is not *if* a security vulnerability will be exploited on your system, but *when*. The general strategy is to … have a plan. How can you recover, where is your backup stored, is it versioned (i.e., multiple versions of the data exist), do you have a backup of your configuration files, how to restore access quickly, etc.

[2]https://dev.mysql.com/doc/refman/8.0/en/mysqldump.html

6.2 SQL Injections

The global idea behind this particular type of attack is the attacker mixing instructions with the data. Imagine, during a process, the following conversation:

> *(At the court)*
> Judge — *What is your name?*
> Attacker — *Bill, you are free to go. This court is adjourned.*
> Judge — *We are here to today to judge Bill, you are free to go. This court is adjourned*

And the attacker can now leave, since the judge said that he was free to go, and that court was adjourned. This is exactly how SQL injections work.

Prepared statement makes it impossible to mix data and instructions, and are the "go-to" solution to protect from this attack. Note that, however, if they are used improperly, they could still be exploited to perform SQL injections.

6.2.1 First Example

Let us look at a first simple example with `ASP`, Active Server Pages, a server-side scripting language. Imagine your code contains:

```
txtUserId = getRequestString("UserId");
txtSQL = "SELECT * FROM Users WHERE UserId = " + txtUserId;
```

that

1. reads a `string` from the user, supposedly their `id`,
2. create a `SQL` statement using this `string` *as is*.

Then, a user can

1. **execute remote command**, entering e.g. `105; DROP TABLE Suppliers;`,
2. **bypass login screen**, entering e.g. `105 or 1 = 1` (so that the `WHERE` condition is now always true),
3. **escalate privileges**, entering e.g. `admin'--` (so that the rest of the line is commented, possibly des-activating other tests on the password, for instance).

This type of attack can also be used for DBMS fingerprinting, i.e., to get a more precise picture of the type of architecture your victim is using.

6.2.2 Second Example

The situation is the following: we are having a party with a secret VIP guest (Marcus Hells). The other guests can try to guess the name of the secret guest. If they succeed, we tell them so, if they don't, we simply display that they do not know who the secret guest is.

An improper program would allow the name of the secret guest to be displayed even if the user does not know that Marcus Hells is the secret VIP. We will see two examples of insecure programs (`code/java/SimpleInjection01.java` and `code/java/SimpleInjection02.java`), where `SQL` injection are possible, and a possible fix (`code/java/SimpleInjection03.java`), using prepared statements.

The gist of `code/java/SimpleInjection01.java` is that writing a statement like

```
ResultSet rset =
    stmt.executeQuery("SELECT * FROM SECRETVIP WHERE Name ='" +
    entered + "';");
```

SimpleInjection01.java[3]

leaves the door open for an attacker to enter `n' OR '1' = '1` as a value for `entered`, so that the condition would always be true.

For `code/java/SimpleInjection02.java`, it shows how

```
stmt.execute("SELECT * FROM SECRETVIP WHERE Name ='" + entered +
    "';");
```

SimpleInjection02.java[4]

could be a serious issue if `nope'; DROP SCHEMA HW_SIMPLE_INJECTION_2;` was entered as a value for `entered`, destroying the whole schema `HW_SIMPLE_INJECTION_2`.

In the second cases, **`INSERT`** or even **`UPDATE`** statements can be executed as well, and a careful SQL injection can even perform its task without crashing the program. As an example, you can try `nope'; UPDATE SECRETVIP SET Name="Me!";` or even `nope'; UPDATE SECRETVIP SET Name="Me!"; SELECT * FROM SECRETVIP WHERE Name='n` for the second program, and see that we successfully modify the data (without the program crashing in the second case).

Finally, `code/java/SimpleInjection03.java` shows how to use proper statements to avoid this situation. Note that the attacks we discussed are no longer possible with this program.

6.2.3 Protections

Possible protections from SQL injections (-like) includes:

1. Prepared statements (a.k.a. stored procedures),
2. White list input validation,
3. Escaping (AT YOUR OWN RISK).

If parts of your prepared statement is determined by the user, then SQL injection could still be possible. For instance, having

```
PreparedStatement ps =
    conn.prepareStatement("SELECT * FROM " + table_given_by_user
    + " WHERE Name = ?;");
```

would still leave you exposed, as `table_given_by_user` could mix instructions with data.

[3] https://rocketgit.com/user/caubert/CSCI_3410/source/tree/branch/master/blob/notes/code/java/SimpleInjection01.java

[4] https://rocketgit.com/user/caubert/CSCI_3410/source/tree/branch/master/blob/notes/code/java/SimpleInjection02.java

Exercises

Exercise 6.1 For each of the following service, indicate the possible consequences attached to the type of loss given, or the type of loss that would result in the consequence given.

Service	Type of loss	Consequence
GPS Satellite	Availability	
Pounce	Integrity	
Patient Database		Everybody knows your uncle had an appendicitis when he was 12.
Bank Service		Your professor is now a millionaire.
Facebook		Mark Zucherberg's phone number is now public.
Booking Website		You can't book your Summer vacations.

Exercise 6.2 You forgot your password for an online service, and click on their "Forgot your password?" link. You enter your email and a few seconds later receive an email with your original password in it. What is the issue here? What are the next steps you should take?

Exercise 6.3 Briefly explain what a SQL injection is.

Exercise 6.4 Briefly explain what a prepared statement is and the benefits it provides.

Exercise 6.5 You are using a software that is directly connected to a database. You do not have access to the source code, but you suspect it is vulnerable to `SQL` injections. How do you proceed to test if injections are possible?

Exercise 6.6 What is fingerprinting?

Solution to Exercises

Solution 6.1 A possible solution is

Service	Type of loss	Consequence
GPS Satellite	Availability	*The navigation system in your car goes blank.*
Pounce	Integrity	*Your grades have been changed.*
Patient Database	*Confidentiality*	Everybody knows your uncle had an appendicitis when he was 12.
Bank Service	*Integrity*	Your professor is now a millionaire.
Facebook	*Confidentiality*	Mark Zucherberg's phone number is now public.
Booking Website	*Availability*	You can't book your Summer vacations.

Solution 6.2 The issue is that they are storing your password in clear text, which is an extremely bad security practice. This suggests that this service does not care about the security of their users, and that all the data in it should be considered compromised. The next steps are:

- If the same password was used on different websites, change it immediately.
- Change the password on this website.

- Delete your account on this website, or, if that is not possible, remove as much information as possible (credit card, address, email, etc.).
- Contact them to express your worries about this security flaw.
- (Optional) See if your account has already been hacked using a service like: https://haveibeenpwned.com/.

Solution 6.3 An SQL injection[5] is a type of attack targeting DBMS using the SQL programming langugae. It consists in mixing conditions (e.g., `' OR '1'='1' --`) or commands (e.g., `DROP TABLE users;`) into the data asked to the user with the goal of executing malicious code on the targeted DBMS. It can result in loss of confidentiality, availability, or integrity, and is a common vector[6] of attack on DBMS.

Solution 6.4 A prepared statement is stored in a DBMS as a "query with parameters," or a template waiting for values to be passed to fill those placeholders, or slots, and then is executed all together as one statement. It is used to execute the same or similar statements repeatedly and with high efficiency, since it is pre-compiled, and compiled only once, it takes less computational resources to be executed. Also, in the case where the arguments are transmitted over the network, it means that only the arguments, and not the whole query, has to be sent, which may result in a increase in speed.

Moreover, since only the arguments are passed, it prevents `SQL` injection, when properly utilized.

Solution 6.5 There are two ways to test if `SQL` injections are possible:

- Look for places where the program is asking for user input and enter values like `1 OR 1 = 1` or `; DROP TABLE Users;--`
- Look for an automated tool (like http://sqlmap.org/) that will test the server to which we are connecting.

Note that both options can be explored in parallel. You can also check out coder resoures, e.g. https://sqa.stackexchange.com/q/1527/, for more ideas on how to test for injections.

Solution 6.6 In general, fingerprinting means accessing information to uniquely identify something. In this particular context, it means attacking the DBMS using multiple techniques (among which SQL injections) to obtain more information about it (software, version, plug-in, etc.). You can read more about it in this article[7].

Problems

Problem 6.1 ***(Insecure Java Programming)*** Consider the following code:

```java
Scanner key = new Scanner(System.in);
System.out.print(
    "Do you want to browse the table containing "
        + "DISK, BOOK or VINYL? (please enter exactly the table
         ↪  name)?\n");
String table = key.nextLine();
```

[5]https://en.wikipedia.org/wiki/SQL_injection
[6]https://en.wikipedia.org/wiki/SQL_injection#Examples
[7]https://null-byte.wonderhowto.com/how-to/sql-injection-101-fingerprint-%20databases-perform-general-reconnaissance-for-more-successful-attack-0184562/

```
System.out.print("How much money do you have?\n");
String max = key.nextLine();
ResultSet rst =
    stmt.executeQuery("SELECT Title FROM " + table + " WHERE PRICE <= " + max + ";");
System.out.printf("Here are the %s you can afford with %s: \n", table, max);
while (rst.next()) {
  System.out.printf("\t- %s \n", rst.getString(1));
}
```

InsecureProgram.java[8]

Assume this software is connecting to a schema in a database hosted at http://example.com/ using:

```
Connection conn = DriverManager.getConnection(
    "jdbc:mysql://example.com/:3306/?user=admin&password=admin");
```

The schema contains three tables (`DISK`, `BOOK` and `VINYL`), each with `Title` and `Price` attributes. The compiled version is then shared with customers all around the world.

You can find a program in a compilable state at `code/java/InsecureProgram.java` that connects to localhost, if you want to test it.

Question 1 The authors of this program believe that the top-secret title of the next disk by a secret group will not be accessible to the user of this program because its price is set to **`NULL`** in the `DISK` table. Prove them wrong.

Question 2 This database application and the whole set-up contains at least three vulnerabilities. List as many as you can think of, and, when relevant, describe how to fix them.

Solutions to Selected Problems

Solution to Problem 6.1 (Insecure Java Programming) **Pb 6.1 – Solution to Q. 1** This program is vulnerable to SQL injection. A user entering "DISK" followed by `0 OR PRICE IS NULL OR PRICE IS NOT NULL` would have access to all the entries, no matter their price tag or lack of one.

Pb 6.1 – Solution to Q. 2 Some of the issues are:

- Disclosing the name of the tables to the user (DISK, BOOK and VINYL). It would be preferable to use some other name in the program.
- Not asking explicitly for a secure connection is probably not a good idea. Using the default port can sometimes be problematic as well.
- Reading a figure as a string is a bad idea, since the user can try to manipulate the content of that field. The datatype read in the application should match the datatype we are trying to get.

[8] https://rocketgit.com/user/caubert/CSCI_3410/source/tree/branch/master/blob/notes/code/java/InsecureProgram.java

- Having `admin` / `admin` as a login / password is unforgivable. The login and password should be changed. And, at least, the application should not connect to the database with admin rights!
- Giving the credentials in the source code is not a good idea. The application should connect to another application, hosted on the the server-side, that performs the connection to the database. Refer e.g. to https://security.stackexchange.com/q/229954 for explanations on why users should not be allowed to connect directly to your database.
- Not using a prepared statement is a huge mistake. This can lead to `SQL` injection like the one we saw above.

7 Presentation of NoSQL

Resources

To write this chapter, were used

- (Sadalage and Fowler 2012),
- https://en.wikipedia.org/wiki/NoSQL,
- (Sullivan 2015),
- (Elmasri and Navathe 2015, chap. 24),
- (Pavlo and Aslett 2016)
- (Lakshman and Malik 2010) and
- (Ellis 2013)

7.1 A Bit of History

This part is partially inspired from (Sadalage and Fowler 2012, chap. 1), but it has been further updated.

7.1.1 Database Applications and Application Databases

When you write a database application, you have two options:

1. One database for multiple applications,
2. One database for each application,
3. Multiple databases for each application.

The first option can cause severe impacts on the efficiency of your system: since multiple clients, different in nature, access the same DBMS, it can become a bottleneck. On the plus side, there is no need to synchronize or duplicate the information, as everything is already in one place.

The second option sidesteps the "bottleneck" issue if the number of user is reasonnable, but may require a lot of synchronization if multiple application needs to share some information. It can also generate a lot of duplication, if the databases need to have some data in common. But, with that second option, you develop an "application database" (i.e., a database dedicated to a particular application), and you have more freedom in the design, schema, and even DBMS (you can use one particular software solution for one particular database application, and a different one for a different database application).

The third option can become a requirement if a large number of clients are using your application and your database become flooded with requests. This is mostly this need to distribute the "same" data accross databases that we will be discussing below.

7.1.2 Clusters, Clusters...

The increase in everything (traffic, size of data, number of clients, etc.) means "up or out", and raises numerous challenges for the "one database for multiple application" option. There is two ways to increase the resources and to scale up:

1. Bigger machines,
2. More machines.

The second option is generally less expensive (compare buying 1,000 raspberry pi VS buying 1 supercomputer that is not a cluster of more modest computers), but came with two drawbacks w.r.t. databases:

1. The cost of licences can be excessive (indeed, you have to buy one licence per computer),
2. and it generally forces to perform "unnatural acts": relational model used to not be really made to be distributed.

7.1.3 A First Shift

Developping DBMS more suited for distributed architectures became growingly important, and some comanies took at stab at it. The more important attemts were

- Google Big Table[1], 2004 (made public in ... 2015!) (Chang et al. 2006)
- Amazon DynamoDB[2], 2004 (used in Simple Storage Service (S3) in 2007)
- Facebook's Cassandra is sometimes mentioned, but it came later on, around 2009 (Lakshman and Malik 2009).

It was solutions suited to the needs of those big companies, that were very specific. But it was interresting to see SQL's supremacy being questionned.

One of the goal was to get rid of "impedance mismatch": mapping classes or objects to database tables defined by a relational schema is complex and cumbersome. However, if you want your database application to go naturally from their data representation to the representations in the DBMS, solving this issue becomes critical. Among the issues,

- There is no absolute notion of "private" and "public" in RDBMS (relative to needs),
- There are many differences in the data-type (no pointer, weird way of defining string, etc.),
- The values in a relational structure have to be simple (no complex datatype, no structure).

The term "impedance mismatch" describes that annoying need for a translation, and one of the goal of this first shift was to get rid of it.

Also, the data is now moving, growing fast, extremely diverse, and traditional relational DBMS seemed not necessarily wel-suited to hande those changes.

[1] https://cloud.google.com/bigtable/
[2] https://aws.amazon.com/dynamodb/

7.1.4 Gathering Forces

To renew the world of DBMS, there were multiple attempts, going in multiple directions. A meetup to discuss them coined the term “NoSQL” in an attempt to have a “twittable” hashtag, and it stayed (even it is as specific as describing a dog as “not being a cat”). The original meet-up asked for “open-source, distributed, nonrelational database”. Today, there is no “official” definition of NoSQL, but NoSQL often implies the following:

- No relational model,
- Not using `SQL`. Some still have a query language, and it ressembles `SQL` (to minimize learning cost), for instance Cassandra’s CQL.,
- Run well on clusters,
- Schemaless: you can add records without having to define a change in the structure first,
- Open source.

Another important notion that emerged was the notion of “polyglot persistence”, which is the idea of “using different data storage technologies to handle varying data storage needs.” In other terms, if you adopt the “application database” approach (i.e., one database dedicated to one particular application), the you can use the DBMS A for your application 1, and the DBMS B for your application 2, or even use A and B for the same application!

7.1.5 The Future or the Past?

There was a lot of enthusiasm, also because this approach “frees the data” (and, actually, the metadata, cf. application/ld+json, JavaScript Object Notation for Linked Data, schema.org, etc.): sharing e.g. a `json` file is much easier that sharing a `SQL` view along with its schema (the example in the Document-Oriented Database will make it clearer).

Some of it will last for sure: polyglot persistency, the possibility of being schema-less, being “distributed first”, the possibility of sacrificing consistency for greater good, etc. This does not mean that `SQL` (“OldSQL”) and relational database are over: there are still useful in many scenario, and the powerfull query language is great (writing your own every time is a nightmare…).

Starting ~ 2010, one reaction was to develop “NewSQL”, which would combine aspects of both approaches. For instance, having to drop the ACID requirements (detailled in this Section) was often seen as a major drawback, but, for instance, MongoDB announced[3] that it would have more and more of the ACID properties!

Also, a really great use of NoSQL is to adopt it at an early stage of the development, when it is not clear what the schemas should be. When the schemas are final, then you can shift to relational DBMS!

The retro-acronym “Not Only `SQL`” emphasizes that `SQL` will still be one of the principal actor, but that developer should be aware of other solutions for other needs.

[3]https://www.mongodb.com/blog/post/multi-document-transactions-in-mongodb

7.1.6 Co-Existing Technologies

It should also be remembered that multiple technologies can and should co-exist. As an example, the hierarchical database model[4] is a type of DBMS dating back to the 60's that has some advantages (high performance and availability) but one major drawback: as the data is represented as trees, the only type of relationship that can be represented is one-to-many ($1 : M$). However, this tree-like structure is still relevant today in some particular applications: for file systems or geographical information, or because of its qualities, it is still used for e.g. file systems or in the windows registery.

7.2 Comparison

`SQL` and the NoSQL approach can be compared in many different ways. Note that there is no "best tool": it would be like trying to decide if a hammer is better than a saw, the answer is "it depends of what you want to do with it!". But you can use one relational or non-relational DBMS for different purposes, sometimes, again, within the same application ("polyglot persistency").

7.2.1 Overview

« Comparaison n'est pas raison »[5]

NoSQL

- Semi-structured data (no schema)
- High performance
- Availability
- Data Replication (improves availability and performance)
- Scalability (horizontal scalabality (add nodes) instead of vertical (add memory))
- Eventual Consistency
- Natively versionning

SQL

- Immediate data consistency
- Powerfull query language (for instance, join is often missing in NoSQL, has to be implemented on the application-side)
- Structured data storage (can be too restrictive)

7.2.2 ACID vs CAP vs BASE

ACID and BASE are three acronyms capturing desirable features of DBMS, while CAP is a theorem stating the impossibility to have some desirable properties at the same time in distributed systems.

ACID is the guarantee of validity even in the event of errors, power failures, etc.

- Atomicity → Transactions are all or nothing
- Consistency → Transactions maintains validity

[4] https://en.wikipedia.org/wiki/Hierarchical_database_model

[5] A French proverb, meaning that "things should be judged on the individual qualities they posses, rather than by comparing one with another." (Manser 2007)

- Isolation → Executing two transactions in parallel or one after the other would have the same result
- Durability → Once a transaction has been commited, it is stored in non-volatile memory.

CAP (a.k.a. Brewer's theorem): Roughly, "In a distributed system, one has to choose between consistency (every read receives the most recent write or an error) and availability (every request receives a (non-error) response, without guarantee that it contains the most recent write)" (the P. standing for "Partition tolerance", a guarantee of availability).

BASE (also formulated by Brewer) corresponds to Basic Availability, Soft state, Eventual consistency. It is a series of properties that can be reached by distributed systems, including NoSQL systems, and is often seen as the "NoSQL's version of ACID". This answer[6] for answer, gives some insight on its meaning.

7.3 Categories of NoSQL Systems

There are multiple ways to be "non-relational". A rough hierarchy of the different approaches can be sketched as follows.

Model	Description	Examples
Document-based	Data is stored as "documents" (JSON, for instance), accessible via their ID (other indexes).	Apache CouchDB[7] (simble for web applications, and reliable), MongoDB[8] (easy to operate), Couchbase[9] (high concurrency, and high availability).
Key-value stores	Fast access by the key to the value. Value can be a record, an object, a document, or be more complex.	Redis[10] (in-memory but persistent on disk database, stores everything in the RAM!)
Column-based (a.k.a. wide column)	Partition a table by colmuns into column families, where each column family is stored in its own files.	Cassandra[11], HBase[12] (both for huge amount of data)
Graph-based	Data is represented as graphs, and related nodes can be found by traversing the edges using path expressions.	Neo4J[13] (excellent for pattern recognition, and data mining)
Multi-model	Support multiple data models	Apache Ignite[14], ArangoDB[15], etc.

[6] https://stackoverflow.com/a/3382260
[7] https://couchdb.apache.org/
[8] https://www.mongodb.com/
[9] https://www.couchbase.com/
[10] https://redis.io/
[11] https://cassandra.apache.org/
[12] https://hbase.apache.org/
[13] https://neo4j.com/
[14] https://ignite.apache.org/
[15] https://www.arangodb.com/

7.4 MongoDB

7.4.1 Resources

- https://www.mongodb.com/
- https://university.mongodb.com
- https://en.wikipedia.org/wiki/MongoDB
- (Sadalage and Fowler 2012, ch. 9)
- (Sullivan 2015, ch. 6)

7.4.2 Introduction

MongoDB is

- Free (i.e., provided at no cost). Their business model leverages training, support, and DB as service. They actually developped MongoDB because they wanted a good solution for a cloud solution!
- Open-source, even if recents changes makes their licence not really open source[16], they share most of their code[17].
- Cross-platform: their community server[18], for instance, runs on all major operating systems.
- Document-oriented: it uses a JSON-like documents with schemas, and it is the most popular[19] DBMS using documents (the next ones are Amazon DynamoDB, Couchbase, CouchDB).

7.4.2.1 Technologies

MongoDB is endowed with

- API[20] and drivers[21] for C, C++, C#, Hadoop Connector, Haskell, Java, node.js, PHP, Perl, Python, Ruby, Scala (Casbah),
- a "mongo shell" (a command-line interface), which is an interactive JavaScript interface to MongoDB. You can try it on-line[22].

[16]https://opensource.org/LicenseReview122018
[17]https://github.com/mongodb
[18]https://www.mongodb.com/download-center/community
[19]https://db-engines.com/en/ranking/document+store
[20]https://api.mongodb.com/
[21]https://docs.mongodb.com/drivers/
[22]https://docs.mongodb.com/manual/tutorial/getting-started/

7.4.2.2 Design

Note that while the design of your database becomes a "second class citizen", as you can start manipulating data before a schema has been defined, this does not mean that design became irrelevant. General design principles *still needs to be adopted*, and everything that was said about design remains true. The key difficulty is that there is no foreign key, in MongoDB, or at least no contraints attached to the relationships two documents can have, except for the one you implement. This is generally considered to be a downside in terms of consistency, and an advantage in terms of flexibility and scalability.

7.4.2.3 Security

Mongodb is vulnerable to `SQL` injection (cf. https://zanon.io/posts/nosql-injection-in-mongodb) and should respect the same general guidelines as discussed in A Bit About Security (cf. https://docs.mongodb.com/manual/administration/security-checklist/).

And additional challenge is that e.g. since **`JOIN`** operations need to be performed "by hand", in the application program (cf. https://www.w3schools.com/nodejs/nodejs_mongodb_join.asp), your attack surface grows.

7.4.3 Document

Let us start by detailling what a "document" is. There are multiple different implementations and definition of what a document is, but at the core of all of them are the followings:

- Documents encapsulate and encode data (Self-Describing Data),
- Documents do not *need* to adhere a standard schema (but they can, if you want),
- One program can have many different types of objects, and those objects often have many optional fields.

Among the formats of documents, there is `XML`, `YAML`, `JSON` (JavaScript Object Notation), `PDF`, etc. You can generally convert from one format to the others, which is an important feature.

An example of XML (Extensible Markup Languag) document, storing information about what Martin and Pradmod like, which cities they visited, etc.:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!-- code/xml/person.xml -->
<root>
   <element>
      <firstname>Martin</firstname>
      <lastVisited>Paris</lastVisited>
      <lastcity>Boston</lastcity>
      <likes>
         <element>Biking</element>
         <element>Photography</element>
      </likes>
   </element>
   <element>
      <firstname>Pramod</firstname>
```

```
        <lastcity>Chicago</lastcity>
        <addresses>
            <element>
                <city>DILLINGHAM</city>
                <state>AK</state>
            </element>
            <element>
                <city>PUNE</city>
                <state>MH</state>
            </element>
        </addresses>
        <citiesvisited>
            <element>Chicago</element>
            <element>London</element>
            <element>Pune</element>
            <element>Bangalore</element>
        </citiesvisited>
    </element>
</root>

```

person.xml[23]

As you can see, from this document:

- The two `element` (person) contains different information: we know the first name of both, but not the address of Martin, nor the `lastVisited` of Pradmod.
- Tags can have an internal structure (like `addresses`), but their order does not matter.
- Invalid document exists! Imagine if one tag is not properly closed, then the parsing would fail.
- Documents are somehow human *and* computer-readable.
- There are no or little predefined tags: the `shiporder` or `item` tags are made-up!
- Documents are extensible, as one can invent new tags, re-fine the organization inside an item, etc.

A more detailled example, including the design of a schema, can be found at w3schools.com[24].

The kind of document MongoDB uses is called `BSON` (portmanteau of the words "binary" and "`JSON`"), and it actually extends `JSON`. Think of `BSON` as a binary representation of `JSON` documents.

7.4.4 Document-Oriented Database

Mongodb is a document-oriented database[25] (document store), which means that the databases contain semi-structured data. It is a subclass of the key-value store:

- Relational databases (RDB) pre-define the data structure (i.e., the schema) in the database (fields + data type).

[23]https://rocketgit.com/user/caubert/CSCI_3410/source/tree/branch/master/blob/notes/code/xml/person.xml
[24]https://www.w3schools.com/xml/schema_example.asp
[25]https://en.wikipedia.org/wiki/Document-oriented_database

- Key-value (KV) treats the data as a single opaque collection, which may have any number (including 0) fields for every record.
- Document-oriented (DO) system relies on internal structure in the data to extract metadata.

RDB is excellent for optimization, but sometimes waste space (placeholders for optional values) and is sometimes too rigid. KV does not allow any optimization, but provides flexibility and follows more closely modern programming concepts. DO has the flexibility of KV, and allows for some optimization.

One important difference: in RDB, data is stored in separate tables, and a single object (entity) may be spread across several tables. In DO, one object = one instance, and every stored object can be different from every other. There are pros to this approach:

- Mapping objects to a DB simpler,
- Change “in place”,
- Increase speed of deployment.

7.4.5 General Organization of MongoDB Databases

Let us start by mapping the common notions of RDBMS to the mongoDB ecosystem:

RDBMS	MongoDB
database instance	MongoDB instance
schema	database
table	collection
row	document

Each MongoDB instance has multiple databases, each database can have multiple collections.

Our previous `XML` “person” example can be converted into two documents[26] delimited by `[...]`, used to delimit an array of document.

```
[
  {
    "_comment": "code/json/person.json"
  },
  {
    "firstname": "Martin",
    "likes": [
      "Biking",
      "Photography"
    ],
    "lastcity": "Boston",
    "lastVisited": "Paris"
  },
  {
    "firstname": "Pramod",
```

[26] We actually had to have three documents: as `JSON` does not really have comments (cf. https://stackoverflow.com/q/244777/), we added a document containing only the attribute `"_comment"` to specify the path where that file is located.

```
    "citiesvisited": [
      "Chicago",
      "London",
      "Pune",
      "Bangalore"
    ],
    "addresses": [
      {
        "state": "AK",
        "city": "DILLINGHAM"
      },
      {
        "state": "MH",
        "city": "PUNE"
      }
    ],
    "lastcity": "Chicago"
  }
]
```

person.json[27]

Note that

- `addresses` is a document embedded in a document!
- Some attributes are common, some are not: that's fine, every document can have its own schema.

A collection should be on "related" entities (do not store server logs, store customers and list of employee in the same collection!), and not too abstract ones (no "Server stuff"). Also, if you store document that are too different, your performances will take a big hit. Bottom line: think about your usage, and the kind of queries you will perform.

So, in summary, "Schema-less" does not mean "organization-less"!

7.4.6 Set Up

The instructions are only for Linux, but should be easy to adapt.

- Download and install `mongodb` from https://www.mongodb.com/download-center/community, select the "server" and "shell" packages.
- As a normal user, type

  ```
  mkdir /tmp/mongotest
  mongod --dbpath /tmp/mongotest
  ```

  to start the server and create a "dummy" database in the folder `/tmp/mongotest`.
- Then, open another terminal, and type in, as a normal user `mongo`.

[27] https://rocketgit.com/user/caubert/CSCI_3410/source/tree/branch/master/blob/notes/code/json/person.json

The documentation is nicely written and well-organized: we'll follow parts of it, please refer to it if needed. You can start by opening the "Getting started"[28] tutorial and running its examples on your own installation.

7.4.7 First Elements of Syntax

The syntax for the command-line interface can be found at https://docs.mongodb.com/manual/reference/mongo-shell/. In a first approximation, the syntax is of the form:

```
db.<name of the collection>.<command>(<arguments>)
```

Where `db` is **not** the name of the database, it is just the prefix.

- To get information about your installation, use
  - `show dbs` to see the databases,
  - `use mydb` to use the `mydb` database,
  - `show collections` to see the collections in a particular database,
- To insert, use:

  ```
  db.books.insert({"title": "Mother Night", "author": "Kurt
   ↪  Blabal"})
  ```

  MongoDB will add a unique identifier (`_id`) if you do not provide one. You can think of that as a primary key.
- To remove an entry, use:

  ```
  db.books.remove({"title":"Mother Night"})
  ```
- To update an entry, use:

  ```
  db.books.update({"title":"Mother Night"}, {$set:
   ↪  {"quantity" : 10}})
  ```

  Other function, such as `$inc`, to increment, can be used.
- To select, use:
  - ```
    db.books.find()
    ```

    is like 'SELECT * FROM Books;
  - ```
    db.books.find({"title":"Mother Night"})
    ```

    is like `SELECT * FROM Books WHERE Title="Mother Night";` and

  ```
  db.books.find({"title":"Mother Night"}, {"author":1,
   ↪  "quantity":1})
  ```

[28] https://docs.mongodb.com/manual/tutorial/getting-started/

is like `SELECT Author, Quantity FROM Books WHERE Title="Mother Night";` Both search for the book with title “Mother Night”, and the second query displays only the author and quantity attributes (along with the `_id`, which is included by default).

- ```
  db.books.find({"title":"Mother Night"}, {"author":0,
    "quantity":0})
  ```

  display all the attributes, except the author and the quantity. You can use that tool in conjonction with the projection to exclude the `_id` from the attributes given:

```
db.books.find({"title":"Mother Night"}, {"author":1,
  "quantity":1, "_id":0})
```

- If you want to select all the entries (without condition) but only certain attributes, you can use the empty document in place of the condition:

```
db.books.find({}, {"author":1, "quantity":1})
```

- ```
  db.books.find({"quantity":{"$gte": 10, "$lt": 50}})
  ```

  displays the entries were the quantity is greater than equal to 10, and less than 50.

It is possibility to mimic some features of `SQL` (like the unique attributes), but there are no referential key integrity, for instance.

Most insert / update / detele will return success as soon as one node received your command, but you may tweak them so that success is returned only once the operation has been performed on the majority of the nodes.

Mongodb does not offer as many features as e.g. MySQL, and there is the need to write a lot on the program side. However, you can find a lot of API (i.e., it is taking the “package manager” approach to offer a modular software), cf. for instance an API over mongo-java-driver: http://jongo.org/ (support some form of prepared statement).

7.4.8 MongoDB Database Program

This section will follow Mongodb’s “quick tour” of the Java api, as discussed at https://mongodb.github.io/mongo-java-driver/3.9/driver/getting-started/quick-start/.

You will need to :

- Make sure mongodb server is up and running as noted in the “Set-up” section,
- Download https://repo1.maven.org/maven2/org/mongodb/mongo-java-driver/3.9.1/mongo-java-driver-3.9.1.jar from https://oss.sonatype.org/content/repositories/releases/org/mongodb/ or from https://github.com/mongodb/mongo-java-driver. You can also find a copy in the git repo[29].
- Download https://raw.githubusercontent.com/mongodb/mongo-java-driver/master/driver-sync/src/examples/tour/QuickTour.java, that you can also find archived in the git repo[30].

[29] https://rocketgit.com/user/caubert/CSCI_3410/source/tree/branch/master/blob/notes%2Flib/mongo-java-driver-3.9.1.jar

[30] https://rocketgit.com/user/caubert/CSCI_3410/source/tree/branch/master/blob/notes%2Fcode%2Fjava/QuickTour.java

Place those two files, `mongo-java-driver-3.9.1.jar` and `QuickTour.java` in the same folder, and run

```
java -cp .:mongo-java-driver-3.9.1.jar QuickTour.java
```

(We do not compile the file first thanks to Java's JEP 330[31]'s feature.)

You should see a large number of lines displayed at the screen, and around the top, the message `INFO: Opened connection [connectionId{localValue:2, serverValue:12}] to local`
Now, open the program file and inspect it.

After various import statement, the program create a `MongoClient` object called `mongoClient`, and connects it to the local database server:

```
MongoClient mongoClient;

if (args.length == 0) {
  // connect to the local database server
  mongoClient = MongoClients.create();
} else {
  mongoClient = MongoClients.create(args[0]);
}
```

QuickTour.java[32]

To get a database and a collection, the program uses:

```
// get handle to "mydb" database
MongoDatabase database = mongoClient.getDatabase("mydb");

// get a handle to the "test" collection
MongoCollection<Document> collection =
  ↪ database.getCollection("test");
```

QuickTour.java[33]

Note that a collection is simply an ArrayList[34] of documents.

Assume we want to create the following document:

```
{
  "name": "MongoDB",
  "type": "database",
  "count": 1,
  "info": {
    "x": 203,
    "y": 102
  }
}
```

[31]https://openjdk.java.net/jeps/330
[32]https://rocketgit.com/user/caubert/CSCI_3410/source/tree/branch/master/blob/notes/code/java/QuickTour.java
[33]https://rocketgit.com/user/caubert/CSCI_3410/source/tree/branch/master/blob/notes/code/java/QuickTour.java
[34]https://docs.oracle.com/javase/8/docs/api/java/util/ArrayList.html

(Remember: order does not matter!)

Then we can use the `Document` class to create it, and then insert the document created:

```
// make a document and insert it
Document doc =
  new Document("name", "MongoDB")
      .append("type", "database")
      .append("count", 1)
      .append("info", new Document("x", 203).append("y", 102));

collection.insertOne(doc);
```

QuickTour.java[35]

Note that we can "chain" the `append`, using `doc.append("type", "database").append("count", 1` etc.

Only at this point would the database and collection being created.

To "witness" what the program is doing from the command line, you can, for instance,

1. Edit the Java program, by commenting the statement `database.drop();`.
2. Execute the modified version,
3. Open the command-line-interface (simply type `mongo`), and run:

   ```
   use mydb
   show collections
   db.test.find()
   ```

   This last command should returns something like

   ```
   { "_id" : ObjectId("5ea72152d8b5777d53c1a148"), "name" :
    ↪ "MongoDB", "type" :   "database", "count" : 1, "info" :
    ↪ { "x" : 203, "y" : 102 } }
   ```

The program goes on and is discussed in details at https://mongodb.github.io/mongo-java-driver/3.9/driver/getting-started/quick-start/. You can see for instance that to construct lists of documents and insert them, one can use:

```
// now, lets add lots of little documents to the collection so
 ↪ we can explore queries and
// cursors
List<Document> documents = new ArrayList<Document>();
for (int i = 0; i < 100; i++) {
  documents.add(new Document("i", i));
}
collection.insertMany(documents);
```

QuickTour.java[36]

A discipline similar to what we saw on Java applications interacting with MySQL should apply:

[35] https://rocketgit.com/user/caubert/CSCI_3410/source/tree/branch/master/blob/notes/code/java/QuickTour.java
[36] https://rocketgit.com/user/caubert/CSCI_3410/source/tree/branch/master/blob/notes/code/java/QuickTour.java

- read the documentation,
- think about what should be the role of the application, and what should be left to the DBMS (knowing that mongoDB can not do as much as MySQL),
- secure your application,
- think if you need one database per application, or if you want to share a single database accross multiple users and applications.

7.5 Principles

We can summarize some of the principles we have learned, and introduce some new, as follows:

- "Schemaless means more responsibility"
- In NoSQL, it is sometimes ok to have a bit of denormalization: duplicating the information, to have everything in the same document, can save time at the price of memory. For instance, you could imagine having a table for phone number, for employee, for emergency contact. You can duplicate that information, it is not going to consume a lot of memory, but will allow you to perform fewer join operations (which are resources expensive), at the price of course of storage.
- As we said, "NoSQL injection" exists, and the same remedies apply: your application should accept only strings from your users (never allow objects by design) and sanitize the inputs before using them (mongo-sanitize[37] or content-filter[38] are good modules for this).

Exercises

Exercise 7.1 Briefly explain the term "Durability".

Exercise 7.2 What is polyglot persistence? Is it useful?

Exercise 7.3 What does it mean to be "schemaless"? What does it imply?

Exercise 7.4 What is denormalization? When could it be useful?

Exercise 7.5 What is the object-relational impedance mismatch? Is it an issue that cannot be overcome?

Exercise 7.6 For each of the following notions, indicate if they are usually an attribute of NoSQL or of "traditional" `SQL`:

	Schema First	Distributed	Relational	Scalable
NoSQL				
`SQL`				

[37] https://www.npmjs.com/package/mongo-sanitize
[38] https://www.npmjs.com/package/content-filter

Solution to Exercises

Solution 7.1 Durability[39] is a (positive) property of a DBMS: it means that if the system "commits" (aknowledges) an operation or a transaction, then it has been recorded permanently (i.e., on a hard-drive, or at least on a non-volatile memory). This implies that in case of failure, the result of the commited transaction will still be present in the system.

Solution 7.2 It is the act of picking the right DBMS for the task and involving multiple DBMS's in a single application. Yes, it is useful. Per wikipedia[40], "Polyglot persistence is the concept of using different data storage technologies to handle different data storage needs within a given software application."

Solution 7.3 "Schemaless" means hat a table can contain documents, or tuples, with different attributes. It implies more responsibilities.

Solution 7.4 Denormalization is to duplicate data about other entities in some entities. It is useful when joining is expensive.

Solution 7.5 Database and object-oriented principles are different and it requires work to make them work together. This correspondance, or matching, can be implemented in the application, or lead to the design of a new DBMS.

	Schema First	Distributed	Relational	Scalable
NoSQL		✓		✓
`SQL`	✓		✓	

Solution 7.6

Problems

Problem 7.1 ***(Explaining NoSQL)*** "NoSQL" used to mean "Non `SQL`", but was retro-actively given the meaning "Not Only `SQL`." Below, write a short essay that explains:

1. What motivated the "Non `SQL`" approach.
2. What is the meaning of "Not Only `SQL`."
3. The benefits and drawbacks of the relational approach.

Problem 7.2 ***(From MongoDB to SQL)*** Your friend has a small MongoDB application to keep track of video games high-scores that they would like to convert to a relational database. They need your help in designing a suitable model for their needs, and to get them started in translating their MongoDB code into SQL code.

1. A typical document in their database is given below. Sketch an entity-relationship diagram that could fit their needs.

[39] https://en.wikipedia.org/wiki/Durability_(database_systems)
[40] https://en.wikipedia.org/wiki/Polyglot_persistence

2. "Translate" the following commands from their workflow into SQL commands, assuming that they successfully implemented the schema you designed at the previous step.

   a)

   ```
   db.games.update(
       {"Game name": "Tetris"},
       {$set:
               {"High score" :
                   {"Points": 1399, "Date":"2021/03/29"}
               }
       }
   )
   ```

   a)

   ```
   db.games.find({"Hold by": "Aunt Minnie"}, {"Game Name":1, "Points":1}}
   ```

   a)

   ```
   db.games.find({"High score": {"$gte": 10, "$lt": 1000}, "Platform":"Nes"
   ```

```
{
  "Game name":"Tetris",
  "Platform":[
    {
      "Name":"Nes",
      "Number of controllers":2
    },
    {
      "Name":"Game boy",
      "Portable":true,
      "Quantity":2
    }
  ],
  "High score":{
    "Points":1293,
    "Hold by":"Aunt Minnie",
    "Date":"2021/02/30"
  },
  "Description":"Complete lines",
  "Genre":[
    "Puzzle",
    "Tile-matching"
  ]
}
```

video_game.json[41]

[41] https://rocketgit.com/user/caubert/CSCI_3410/source/tree/branch/master/blob/notes/code/json/video_game.json

Problem 7.3 ***(ER Diagram from XML File – Customer)*** Consider the following `xml` file:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!-- code/xml/sustomers.xml -->
<Customers>
    <Customers>
        <Customer Name="Pamela Zave" ID="C001">
            <Orders>
                <Order Date="2012-07-04T00:00:00" ID="10248">
                    <Product Quantity="5" ID="10">
                       <Description>A Box of Cereal</Description>
                        <Brand>Cereal Company</Brand>
                        <Price>$3</Price>
                    </Product>
                    <Product Quantity="10" ID="43">
                      <Description>A Box of Matches</Description>
                        <Brand>Match Ltd</Brand>
                        <Price>$1.20</Price>
                        <Caution>Not suitable for children</Caution>
                    </Product>
                </Order>
            </Orders>
            <Address>123 Main St., Augusta, GA, 30904</Address>
        </Customer>
        <Customer Name="Nancy Lynch" ID="C002">
            <Orders>
```

```
                <Order Date="2011-07-04T00:00:00" ID="10245">
                    <Product Quantity="3" ID="10">
                       <Description>A Box of Cereal</Description>
                        <Brand>Cereal Company</Brand>
                        <Price>$3</Price>
                    </Product>
                    <Product Quantity="1" ID="5">
                        <Description>A Cup</Description>
                        <Brand>Cup Company</Brand>
                        <Price>$2</Price>
                        <Material>Stoneware</Material>
                    </Product>
                </Order>
            </Orders>
            <Address> Address line 5, 6, 7</Address>
        </Customer>
        <Customer Name="Shafi Goldwasser" ID="C003">
           <Address>345 Second St., Augusta, GA, 30904</Address>
        </Customer>
    </Customers>
</Customers>
```

customers.xml[42]

Try to draw the ER model that would correspond to the relational implementation of this database. Justify your choices.

---

[42] https://rocketgit.com/user/caubert/CSCI_3410/source/tree/branch/master/blob/notes/code/xml/customers.xml

Problem 7.4 ***(ER Diagram from XML File – Award)*** Find below a mashup of actual data from the National Science Foundation (courtesy of https://www.nsf.gov/awardsearch/download.jsp):

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!-- code/xml/NSFAward.xml -->
<rootTag>
    <Award>
        <AwardTitle>CAREER: Advances in Graph Learning and Inference</AwardTitle>
        <AwardEffectiveDate>11/01/2019</AwardEffectiveDate>
        <AwardExpirationDate>01/31/2023</AwardExpirationDate>
        <AwardAmount>105091</AwardAmount>
        <Organization>
            <Code>05010000</Code>
            <Directorate>
                <Abbreviation>CSE</Abbreviation>
                <LongName>Direct For Computer &amp; Info Scie &amp; Enginr</LongName>
            </Directorate>
            <Division>
                <Abbreviation>CCF</Abbreviation>
               <LongName>Division of Computing and Communication Foundations</LongName>
            </Division>
        </Organization>
        <ProgramOfficer>
            <SignBlockName>Phillip Regalia</SignBlockName>
        </ProgramOfficer>
        <AwardID>2005804</AwardID>
        <Investigator>
            <FirstName>Patrick</FirstName>
            <LastName>Hopkins</LastName>
            <EmailAddress>phopkins@virginia.edu</EmailAddress>
            <StartDate>11/22/2019</StartDate>
            <EndDate />
            <RoleCode>Co-Principal Investigator</RoleCode>
        </Investigator>
        <Investigator>
            <FirstName>Jon</FirstName>
            <LastName>Ihlefeld</LastName>
            <EmailAddress>jfi4n@virginia.edu</EmailAddress>
            <StartDate>11/22/2019</StartDate>
            <EndDate />
            <RoleCode>Principal Investigator</RoleCode>
        </Investigator>
        <Institution>
            <Name>University of Virginia Main Campus</Name>
            <CityName>CHARLOTTESVILLE</CityName>
            <ZipCode>229044195</ZipCode>
```

```
            <PhoneNumber>4349244270</PhoneNumber>
            <StreetAddress>P.O. BOX 400195</StreetAddress>
            <CountryName>United States</CountryName>
            <StateName>Virginia</StateName>
            <StateCode>VA</StateCode>
        </Institution>
    </Award>
</rootTag>
```

NSFAward.xml[43]

It contains information about one particular award that was awarded to an institution on behalf of two researchers. Quoting the National Science Foundation[44] (NSF):

> NSF is divided into the following seven directorates that support science and engineering research and education:…. Each is headed by an assistant director and each is further subdivided into divisions like …

From this `xml` file and the information given above, draw an ER diagram for NSF's awards. Do not hesitate to comment on the choices you are making and on what justifies them.

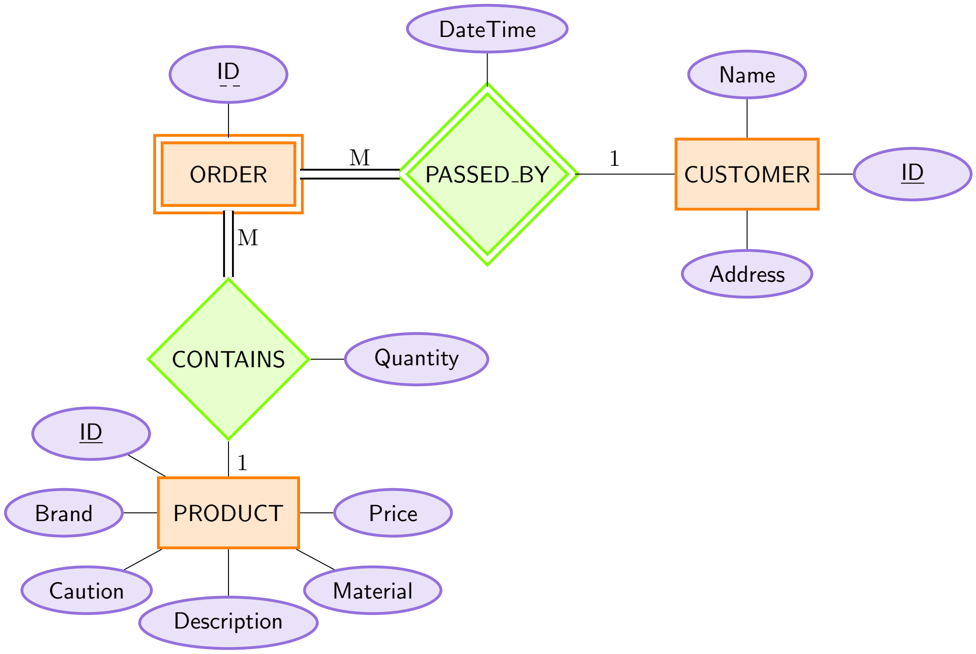
Solutions to Selected Problems

Solution to Problem 7.3 (ER Diagram from XML File – Customer) It should be clear that three entities are present in this file: Customer, Order, and Product. An order can contain a certain quantity of a product, and a customer can pass 0 or more orders. Some attributes are natural primary keys (they are named "ID" in the diagram below), and some attributes seems to be optional ("Caution", or "Material"), but should still be made an attribute.

Put together, this gives the following diagram:

[43] https://rocketgit.com/user/caubert/CSCI_3410/source/tree/branch/master/blob/notes/code/xml/NSFAward.xml

[44] https://www.nsf.gov/about/research_areas.jsp



We made further assumptions: an order cannot be empty (transcribed by the total constraint on CONTAINS), and an order does not exist if it was not passed by a customer (transcribed by the fact that ORDER is a weak entity), which also implies that an order cannot be passed by more than one customer. Note that the same product cannot be present "twice" (with the equal or different quantities) in an order: an order can contain a particular product only once in any quantity, implying that if an order had two of the product A, and three of the same product A, then those two quantities of A should be merged so that an order contains five of this product A. This is enforced by the cardinality ratio of 1 in the CONTAINS relationship.

Of course, other choices are possible.

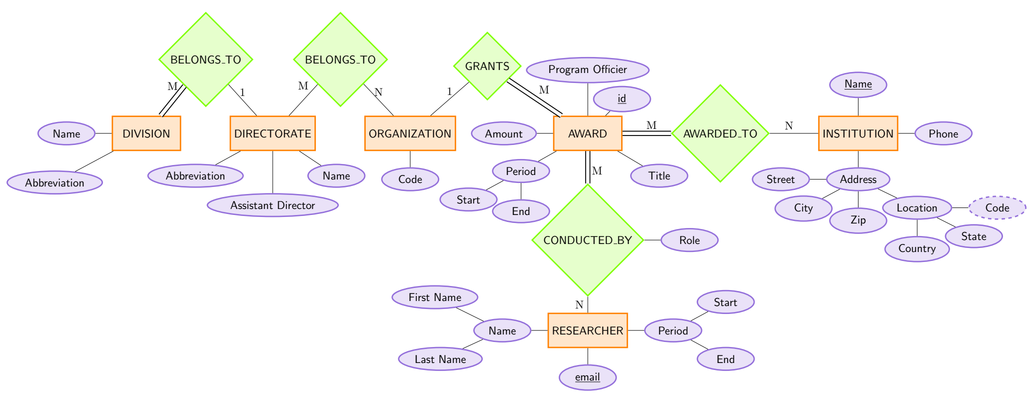
Solution to Problem 7.4 (ER Diagram from XML File – Award) Two entities are easy to distinguish: RESEARCHER (for "Investigator") and INSTITUTION. The status of the the content between the `<Organization>` tags is less clear; apparently, an organization has a code, and is made of two parts: a Directorate and a Division. Using the quote, we know that a Division should be a part of exactly one Directorate, and that a Directorate has an assistant director. But what is the status of that "Organization"? Is it subsumed by the Directorate or is it orthogonal? We decide to create an entity for it, but its precise role should be clarified. The relationship between Division and Directorate is clear, but, once again, the relationship between Division and Organization could have any constraint, we can not really infer that information from the document.

The next difficulty is the status of the award itself: should it be a relationship with many attributes, between the RESEARCHER and INSTITUTION entities? The issue with this approach is that an award can have multiple investigators, as shown in the example, and that this number can vary. Hence, fixing the arity and constraints on this relationship will be difficult. We could have a relation of arity 2, and "duplicate it" if multiple researchers are involved in the same grant, but that seems like a poor choice (since all the information about the grant will need to be duplicated). Therefore, it seems more reasonable to make the award an entity.

How should we connect the AWARD entity with the RESEARCHER and INSTITUTION entities? A ternary relation has some drawbacks, since it would require some duplication when multiple investigators are working on the same award. Instead, having one binary relationship between the award and the institution, and one binary relationship between the award and the researcher (that specifies further the role of the researcher for that particular award), seems like a safer choice. An award must be awarded to at least one researcher and one institution, but we do not know if there is a maximum number of institutions that can obtain the same award, so it is better not to restrict this arity. Whether there should be a relationship between the researcher and the institution is up in the air; we do not know if a researcher has to work for an institution to get a grant, nor if getting a grant for an institution means that you work for it, so it is probably better to refrain from adding such a relationship.

Most of the attributes are straightforward once we see that “Role” is an attribute of a relationship, not of an entity.

All together, this gives the following diagram:



References

Aubert, Clément. 2019. "CSCI 3410 - Database Systems." Lecture notes. Augusta, Georgia, USA: School of Computer and Cyber Sciences, Augusta University. https://spots.augusta.edu/caubert/db/ln/.

Chang, Fay, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach, Michael Burrows, Tushar Chandra, Andrew Fikes, and Robert Gruber. 2006. "Bigtable: A Distributed Storage System for Structured Data (Awarded Best Paper!)." In *7th Symposium on Operating Systems Design and Implementation (OSDI '06), November 6-8, Seattle, WA, USA*, edited by Brian N. Bershad and Jeffrey C. Mogul, 205–18. USENIX Association. https://www.usenix.org/legacy/events/osdi06/tech/chang.html.

Ellis, Jonathan. 2013. "Facebook's Cassandra Paper, Annotated and Compared to Apache Cassandra 2.0." 2013. https://docs.datastax.com/en/articles/cassandra/cassandrathenandnow.html.

Elmasri, Ramez, and Shamkant B. Navathe. 2010. *Fundamentals of Database Systems (6th Edition)*. Pearson.

———. 2015. *Fundamentals of Database Systems (7th Edition)*. Pearson.

Gaddis, Tony. 2014. *Starting Out with Java: Early Objects (5th Edition)*. Pearson.

Lakshman, Avinash, and Prashant Malik. 2009. "Cassandra - a Decentralized Structured Storage System." In *LADIS 2009*. https://research.cs.cornell.edu/ladis2009/papers/lakshman-ladis2009.pdf.

———. 2010. "Cassandra: A Decentralized Structured Storage System." *SIGOPS Oper. Syst. Rev.* 44 (2): 35–40. https://doi.org/10.1145/1773912.1773922.

Manser, Martin H. 2007. *The Facts on File Dictionary of Proverbs*. Facts on File.

Pavlo, Andrew, and Matthew Aslett. 2016. "What's Really New with NewSQL?" *SIGMOD Record* 45 (2): 45–55. https://doi.org/10.1145/3003665.3003674.

Sadalage, Pramod J., and Martin Fowler. 2012. *NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence*. Addison-Wesley Professional.

Sullivan, Dan. 2015. *NoSQL for Mere Mortals*. Addison-Wesley Professional.

Watt, Adrienne, and Nelson Eng. 2014. *Database Design (2nd Edition)*. Victoria, B.C.: BCcampus. https://opentextbc.ca/dbdesign01/.