<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.12 Setting Up Your Work Environment</td>
<td>72</td>
</tr>
<tr>
<td>3.12.1 Installation</td>
<td>72</td>
</tr>
<tr>
<td>3.12.2 Creating a User</td>
<td>76</td>
</tr>
<tr>
<td>3.12.3 Logging-In as testuser</td>
<td>77</td>
</tr>
<tr>
<td>3.12.4 Creating Our First Database</td>
<td>77</td>
</tr>
<tr>
<td>3.12.5 Security Concerns</td>
<td>79</td>
</tr>
<tr>
<td>Exercises</td>
<td>79</td>
</tr>
<tr>
<td>Solution to Exercises</td>
<td>82</td>
</tr>
<tr>
<td>Problems</td>
<td>85</td>
</tr>
<tr>
<td>Solutions to Selected Problems</td>
<td>107</td>
</tr>
<tr>
<td>4 Designing a Good Database</td>
<td>138</td>
</tr>
<tr>
<td>4.1 Interest for High-Level Design</td>
<td>138</td>
</tr>
<tr>
<td>4.2 Entity-Relationship Model</td>
<td>139</td>
</tr>
<tr>
<td>4.2.1 Entities</td>
<td>139</td>
</tr>
<tr>
<td>4.2.2 Relationships</td>
<td>142</td>
</tr>
<tr>
<td>4.2.3 Weak Entity Types</td>
<td>149</td>
</tr>
<tr>
<td>4.2.4 Alternative Notations</td>
<td>152</td>
</tr>
<tr>
<td>4.2.5 Enhanced Entity-Relationship Model</td>
<td>154</td>
</tr>
<tr>
<td>4.2.6 Reverse Engineering</td>
<td>155</td>
</tr>
<tr>
<td>4.3 E.R.-to-Relational Models Mapping</td>
<td>155</td>
</tr>
<tr>
<td>4.3.1 Intro</td>
<td>155</td>
</tr>
<tr>
<td>4.3.2 Algorithm</td>
<td>156</td>
</tr>
<tr>
<td>4.3.3 Outro</td>
<td>160</td>
</tr>
<tr>
<td>4.4 Guidelines and Normal Form</td>
<td>161</td>
</tr>
<tr>
<td>4.4.1 General Rules</td>
<td>161</td>
</tr>
<tr>
<td>4.4.2 Example</td>
<td>162</td>
</tr>
<tr>
<td>4.4.3 Functional Dependencies</td>
<td>162</td>
</tr>
<tr>
<td>4.4.4 Normal Forms and Keys</td>
<td>165</td>
</tr>
<tr>
<td>4.5 Unified Modeling Language Diagrams</td>
<td>168</td>
</tr>
<tr>
<td>4.5.1 Overview</td>
<td>168</td>
</tr>
<tr>
<td>4.5.2 Types of Diagrams</td>
<td>169</td>
</tr>
<tr>
<td>4.5.3 Zoom on Classes Diagrams</td>
<td>171</td>
</tr>
<tr>
<td>Exercises</td>
<td>173</td>
</tr>
<tr>
<td>Solution to Exercises</td>
<td>180</td>
</tr>
<tr>
<td>Problems</td>
<td>187</td>
</tr>
<tr>
<td>Solutions to Selected Problems</td>
<td>202</td>
</tr>
<tr>
<td>5 Database Applications</td>
<td>213</td>
</tr>
<tr>
<td>5.1 Overview</td>
<td>213</td>
</tr>
<tr>
<td>5.2 Java’s Way</td>
<td>214</td>
</tr>
<tr>
<td>5.3 Flash Intro to Java</td>
<td>215</td>
</tr>
<tr>
<td>5.4 A First Program</td>
<td>215</td>
</tr>
<tr>
<td>5.4.1 The Database (sql)</td>
<td>215</td>
</tr>
<tr>
<td>5.4.2 Executing Database Application</td>
<td>216</td>
</tr>
<tr>
<td>5.4.3 The Application Program (java)</td>
<td>217</td>
</tr>
<tr>
<td>5.4.4 The Result</td>
<td>219</td>
</tr>
<tr>
<td>5.4.5 A Variation</td>
<td>219</td>
</tr>
</tbody>
</table>
5.5 Mapping Datatypes ........................................... 220
5.6 Differences Between `executeQuery`, `executeUpdate` and `execute` ........................................ 221
5.7 A Second Program ........................................... 221
5.7.1 Passing Options ........................................ 221
5.7.2 Creating a Table ......................................... 221
5.7.3 Inserting Values ......................................... 222
5.7.4 Prepared Statements .................................. 222
5.7.5 More Complex Statement Objects ................. 222
Exercises ......................................................... 225
Solution to Exercises ........................................ 227
5.8 Problems ..................................................... 229
Solutions to Selected Problems ......................... 237

6 A Bit About Security ........................................... 240
Resources ........................................................ 240
6.1 Usual Aspects ................................................ 240
6.1.1 Threat Model ........................................... 240
6.1.2 Control Measures ....................................... 240
6.1.3 How to Recover? ........................................ 240
6.2 Specificities Of Databases ................................. 240
6.2.1 Attack ..................................................... 240
6.2.2 Protections .............................................. 241
Exercises ......................................................... 242
Solution to Exercises ........................................ 242
Problems ........................................................ 243
Solutions to Selected Problems ......................... 243

7 Presentation of NoSQL .......................................... 245
Resources ........................................................ 245
7.1 A Bit of History ............................................. 245
7.1.1 Database Applications and Application Databases ........................................... 245
7.1.2 Clusters, clusters... .................................... 245
7.1.3 A First Shift ............................................. 246
7.1.4 Gathering Forces ...................................... 246
7.1.5 The Future or the Past? .............................. 247
7.2 Comparison ................................................... 247
7.2.1 Overview ................................................ 247
7.2.2 ACID vs CAP vs BASE ............................... 247
7.3 Categories of NoSQL Systems ......................... 248
7.4 MongoDB ...................................................... 249
7.4.1 Resources ................................................. 249
7.4.2 Introduction ............................................. 249
7.4.3 Document-Oriented Database ....................... 249
7.4.4 General Organization of MongoDB Databases ........................................... 251
7.4.5 Set Up .................................................... 252
7.4.6 First Elements of Syntax ............................. 252
7.4.7 MongoDB Database Program ....................... 253
7.5 Principles ...................................................... 255
Exercises ......................................................... 255
Solution to Exercises ........................................ 255
# List of Problems

1.1 Define a database for **CAMPUS** .................................................. 23
2.1 Find a candidate key for the **CLASS** relation .............................. 37
2.2 Design a relational model for a cinema company ............................ 37
2.3 Design a relational model for bills .............................................. 38
2.4 Relational model for universities .............................................. 38
2.5 Relational model for an auction website .................................... 38
3.1 Discovering the documentation ................................................ 85
3.2 Create and use a simple table in **SQL** ..................................... 86
3.3 Duplicate rows in **SQL** ........................................................ 87
3.4 Constraints on foreign keys ..................................................... 88
3.5 Revisiting the **PROF** table ................................................... 88
3.6 **TRAIN** table and more advanced **SQL** coding ...................... 89
3.7 Read, correct, and write **SQL** statements for the **COFFEE** database ......................... 90
3.8 Write select queries for the **DEPARTMENT** table ..................... 93
3.9 Write select queries for the **COMPUTER** table ......................... 94
3.10 Write select queries for a variation of the **COMPUTER** table ........ 95
3.11 Improving a role-playing game with a relational model .................. 96
3.12 A simple database for books .................................................. 97
3.13 A database for website certificates ....................................... 99
3.14 A simple database for published pieces of work .......................... 101
3.15 A simple database for authors of textbooks ............................... 103
3.16 A database for residencies .................................................... 105
4.1 Design for your professor ....................................................... 187
4.2 Reading the **MOVIES** database **ER** schema ......................... 188
4.3 **ER** diagram for car insurance ........................................... 188
4.4 **ER** diagram for job and offers ............................................ 189
4.5 Reverse engineering by hand .................................................. 189
4.6 Discovering MySQL Workbench .............................................. 189
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.7</td>
<td>ER-to-Relation mapping for car insurance</td>
<td>190</td>
</tr>
<tr>
<td>4.8</td>
<td>From E.R. diagram to Relational model – BIKE</td>
<td>190</td>
</tr>
<tr>
<td>4.9</td>
<td>From E.R. diagram to Relational model – RECORD</td>
<td>191</td>
</tr>
<tr>
<td>4.10</td>
<td>ER-to-Relation mapping for Country</td>
<td>192</td>
</tr>
<tr>
<td>4.11</td>
<td>From business statements to E.R. diagram – UNIVERSITY</td>
<td>193</td>
</tr>
<tr>
<td>4.12</td>
<td>Normal form of a CAR_SALE relation</td>
<td>194</td>
</tr>
<tr>
<td>4.13</td>
<td>Normal form of a simple relation</td>
<td>194</td>
</tr>
<tr>
<td>4.14</td>
<td>Normal form of a SCHEDULE relation</td>
<td>194</td>
</tr>
<tr>
<td>4.15</td>
<td>Normalizing the FLIGHT relation</td>
<td>195</td>
</tr>
<tr>
<td>4.16</td>
<td>From business statement to dependencies, BIKE</td>
<td>195</td>
</tr>
<tr>
<td>4.17</td>
<td>From business statement to dependencies, ROUTE</td>
<td>196</td>
</tr>
<tr>
<td>4.18</td>
<td>From business statement to dependencies, ISP</td>
<td>196</td>
</tr>
<tr>
<td>4.19</td>
<td>Normalization</td>
<td>196</td>
</tr>
<tr>
<td>4.20</td>
<td>Normal form of the BOOK relation</td>
<td>197</td>
</tr>
<tr>
<td>4.21</td>
<td>Normal form of the CONTACT relation</td>
<td>197</td>
</tr>
<tr>
<td>4.22</td>
<td>Normal form of the MESSAGE relation</td>
<td>197</td>
</tr>
<tr>
<td>4.23</td>
<td>PRINT relation in third normal form</td>
<td>198</td>
</tr>
<tr>
<td>4.24</td>
<td>CONSULTATION relation: justification, primary key and normal form</td>
<td>198</td>
</tr>
<tr>
<td>4.25</td>
<td>COFFEE relation: primary key and normal form</td>
<td>199</td>
</tr>
<tr>
<td>4.26</td>
<td>A Relation for Network Cards</td>
<td>199</td>
</tr>
<tr>
<td>4.27</td>
<td>From E.R. to relational schema and UML class diagram – CAR_INFO</td>
<td>199</td>
</tr>
<tr>
<td>4.28</td>
<td>From Business Statement to ER Diagram to Relational Model – A Network of Libraries</td>
<td>200</td>
</tr>
<tr>
<td>4.29</td>
<td>Using MySQL Workbench’s reverse engineering</td>
<td>201</td>
</tr>
<tr>
<td>4.30</td>
<td>From business statements to dependencies – KEYBOARD</td>
<td>201</td>
</tr>
<tr>
<td>4.31</td>
<td>From UML to relational model – DRIVER</td>
<td>201</td>
</tr>
<tr>
<td>5.1</td>
<td>Advanced Java Programming</td>
<td>229</td>
</tr>
<tr>
<td>5.2</td>
<td>A GUEST Java Program</td>
<td>235</td>
</tr>
<tr>
<td>6.1</td>
<td>Insecure Java Programming</td>
<td>243</td>
</tr>
<tr>
<td>7.1</td>
<td>Explaining NoSQL</td>
<td>256</td>
</tr>
<tr>
<td>7.2</td>
<td>E.R. Diagram from XML File – Customer</td>
<td>256</td>
</tr>
<tr>
<td>7.3</td>
<td>E.R. Diagram from XML File – Award</td>
<td>257</td>
</tr>
</tbody>
</table>
Preamble

How to Use This Guide

These lecture notes are written in an elusive style: their are a support for the explanations I will be making at the board. They are not designed to be self-contained: they are rather a list of topics and reminders, along with handy examples, code and drawings. 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.

When it comes to code, you can normally copy-and-paste it from the document and use it as it is. Or, 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 it directly. Some portion of code starts with a path in comment, like so:

```sql
/* code/sql/HELLO_WORLD.sql */

SELECT "Hello World!";
```

This means that this code can be found at https://rocketgit.com/user/caubert/CSCI_3410/source/tree/branch/master/blob/notes/code/sql/HELLO_WORLD.sql.

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;
```

This part 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 they are an artifice from pandoc-include-code to select which portion of the code to display.

To clone this source of those notes and have a local copy of it, please refer to the instructions at http://spots.augusta.edu/caubert/db/ln/README.html. Instructions on how to compile those notes are available at the same place.

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

- References, at the very end of this document
- and for each chapter,

---

2. https://github.com/owickstrom/pandoc-include-code
- A list of additional resources,
- A list of short exercises,
- Solution to those exercises,
- A list of problems,
- Sometimes, solution to some of those problems.

Any feedback is greatly appreciated. Please refer to http://spots.augusta.edu/caubert/db/ln/RE ADME.html#contributing for how to contribute to those notes.

The syllabus is at http://spots.augusta.edu/caubert/db/, and the webpage for this notes is at http://spots.augusta.edu/caubert/db/ln/. Please, refer to those notes using this entry (Aubert 2019):

```latex
@report{AubertCSCI3410-DatabaseSystems,
  author={Aubert, Clément},
  title={CSCI 3410 - Database Systems},
  url={http://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}
}
```

**Planned Schedule**

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

- Lecture 1: Presentation and Syllabus\(^3\)
- 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: E.R.-to-Relational Models Mapping
- Lecture 17–20: Guidelines and Normal Form
- Lecture 23–24: Review session and Exam #2
- Lecture 25–18: Database Applications
- Lecture 29–30: Presentation of NoSQL

For information purposes, an indication like this:

```
```

marks the (usual) separation between two lectures.

\(^3\)http://spots.augusta.edu/caubert/db/
Contents

Previous Exams

To give you a sense of what you will be asked during the exams, 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).

Spring 2020

- Project #1: Problem 3.15 (A simple database for authors of textbooks)
- Exam #1:
  - Problem 3.16 (A database for residencies)
  - Problem 2.5 (Relational model for an auction website)

Fall 2019

- Exam #1:
  - Problem 3.14 (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 4.28 (From Business Statement to ER Diagram to Relational Model – A Network of Libraries)
  - Problem 4.17 (From business statement to dependencies, ROUTE)
  - Problem 5.2 (A GUEST Java Program)
  - Problem 4.23 (PRINT relation in third normal form)
- Final:
  - Problem 7.3 (E.R. Diagram from XML File – Award)
  - Problem 4.13 (Normal form of a simple relation)
  - Problem 4.18 (From business statement to dependencies, ISP)
  - Problem 3.13 (A database for website certificates)
  - Three small exercises about security (Exercise 6.1, Exercise 6.2, Exercise 6.3)

Spring 2019

- Exam #1:
  - Problem 3.12 (A simple database for books)
  - Five exercises (Exercise 1.5, Exercise 2.12, Exercise 3.32, Exercise 2.14, 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 4.9 (From E.R. diagram to Relational model – RECORD)
  - Problem 4.14 (Normal form of a SCHEDULE relation)
  - Problem 4.22 (Normal form of the MESSAGE relation)
• Final:
  - Problem 7.1 (Explaining NoSQL)
  - Problem 4.20 (Normal form of the BOOK relation)
  - Problem 4.26 (A Relation for Network Cards)
  - Problem 3.10 (Write select queries for a variation of the COMPUTER table)
  - A variation on Problem 4.28 (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 4.21 (Normal form of the CONTACT relation)
  - A variation on Problem 4.24 (CONSULTATION relation: justification, primary key and normal form)
  - Problem 3.11 (Improving a role-playing game with a relational model)
  - A variation on Problem 4.30 (From business statements to dependencies – KEYBOARD)
  - Problem 4.10 (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 4.32, Exercise 7.1, Exercise 7.2, Exercise 7.3, Exercise 7.4)
  - Problem 4.15 (Normalizing the FLIGHT relation)
  - A variation on Problem 4.11 (From business statements to E.R. diagram – UNIVERSITY)
  - A variation on Problem 4.31 (From UML to relational model – DRIVER): students were asked to draw the ER diagram for that schema.

Fall 2017

• Exam #1:4
  - Six small exercises (Exercise 1.11, Exercise 2.4, Exercise 2.8, Exercise 3.6, Exercise 3.9 and Exercise 3.12)
  - Problem 2.1 (Find a candidate key for the CLASS relation)
  - Problem 2.2 (Design a relational model for a cinema company)

---

4This exam was probably a bit too long, but students managed it pretty well.
– 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:
– Problem 4.11 (From business statements to E.R. diagram – UNIVERSITY)
– Problem 4.16 (From business statement to dependencies, BIKE)
– Problem 4.21 (Normal form of the CONTACT relation)
– A variation on Problem 4.10 (ER-to-Relation mapping for Country)

• Final:
– A variation on (Exercise 5.16)
– A variation on Problem 3.7 (Read, correct, and write SQL statements for the COFFEE database)
– A variation on Problem 4.31 (From UML to relational model – DRIVER): students were asked to draw the ER diagram for that schema.
– Problem 4.20 (Normal form of the BOOK relation)
– Problem 4.24 (CONSULTATION relation: justification, primary key and normal form)

Typesetting and Acknowledgments

The source code for those notes is hosted at rocketgit\(^5\), typeset in markdown\(^6\), and then compiled using pandoc\(^7\) and multiple filters (pandoc-numbering\(^8\), pandoc-citeproc\(^9\), pandoc-include-code\(^10\)). The drawings use various LaTeX\(^11\) packages, including PGF, TikZ\(^12\), tikz-er2\(^13\) and tikz-dependency\(^14\). The help from the TeX - LaTeX Stack Exchange\(^15\) community greatly improved this document. The underlined\(^16\) text is obtained using YayText\(^17\), the unicode symbols are searched in the "Unicode characters and corresponding LaTeX math mode commands"\(^18\). Finally, the pdf version of the document uses Linux Libertine fonts\(^19\), the html version uses Futura\(^20\).

Those lecture notes were created under an Affordable Learning Georgia\(^21\) Mini-Grant for Ancillary Materials Creation and Revision\(^22\) (Proposal M71\(^23\)).
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 detailed 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.
- (Sadlage 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, I believe (Gaddis 2014) is a good introduction.
- awesome-mysql is a “curated list of awesome MySQL free and opensource software, libraries and resources” that is definitely worth checking out. Among other resources, note this bank of SQL programming exercises.

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

This work is under Creative Commons Attribution 4.0 International License or later.

Some figures and resources are borrowed from other sources, in which case it is indicated clearly.

---

24https://github.com/shlomi-noach/awesome-mysql#readme
25https://github.com/XD-DENG/SQL-exercise
26https://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 mean 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 this 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 of both models.

1.2 Database

A database (DB) is a collection of related data. Data (= information, can be anything, really) + management (= logical organization of the data), through Database Management System.

1. Represent a mini-world, a Universe of Disclosure (UoD).
2. Logically coherent, with a meaning.

\(^1\)https://en.wikipedia.org/wiki/Turing-completeness
\(^2\)https://www.gwern.net/Turing-complete
3. Populated for a purpose.

A DBMS has multiple components, as follows:

- 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.
- Most DBMS software include a Command-Line Interface (C.L.I.).
- 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).

### 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, softwares)

You can think of a tool to

\(^3\)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 our data.
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 Subtasks

Exactly like a program can have

- a client, 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.

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

In those lecture notes, the main focus will be on design, 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:
1.6 An Example

Let us consider the following:

**STUDENT**

<table>
<thead>
<tr>
<th>Name</th>
<th>Student_number</th>
<th>Class</th>
<th>Major</th>
</tr>
</thead>
<tbody>
<tr>
<td>Morgan</td>
<td>18</td>
<td>2</td>
<td>IT</td>
</tr>
<tr>
<td>Bob</td>
<td>17</td>
<td>1</td>
<td>CS</td>
</tr>
</tbody>
</table>

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

### COURSE

<table>
<thead>
<tr>
<th>Course_name</th>
<th>Course_number</th>
<th>Credit_hours</th>
<th>Department</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intro. to CS</td>
<td>1301</td>
<td>4</td>
<td>CS</td>
</tr>
<tr>
<td>DB Systems</td>
<td>3401</td>
<td>3</td>
<td>CS</td>
</tr>
<tr>
<td>Principles of Scripting and Automation</td>
<td>2120</td>
<td>3</td>
<td>AIST</td>
</tr>
</tbody>
</table>

### SECTION

<table>
<thead>
<tr>
<th>Section_identifier</th>
<th>Course_num</th>
<th>Semster</th>
<th>Year</th>
<th>Instructor</th>
</tr>
</thead>
<tbody>
<tr>
<td>2910</td>
<td>1301</td>
<td>Fall</td>
<td>2019</td>
<td>Kate</td>
</tr>
<tr>
<td>9230</td>
<td>2103</td>
<td>Spring</td>
<td>2020</td>
<td>Todd</td>
</tr>
</tbody>
</table>

### GRADE_REPORT

<table>
<thead>
<tr>
<th>Student_number</th>
<th>Section_identifier</th>
<th>Grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>17</td>
<td>2910</td>
<td>A</td>
</tr>
<tr>
<td>18</td>
<td>2910</td>
<td>B</td>
</tr>
</tbody>
</table>

### PREREQUISITE

<table>
<thead>
<tr>
<th>Course_number</th>
<th>Prerequisite_number</th>
</tr>
</thead>
<tbody>
<tr>
<td>2120</td>
<td>1301</td>
</tr>
<tr>
<td>1302</td>
<td>1301</td>
</tr>
</tbody>
</table>

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

#### RELATIONS

<table>
<thead>
<tr>
<th>Relation Name</th>
<th>Number of Columns</th>
</tr>
</thead>
<tbody>
<tr>
<td>STUDENT</td>
<td>4</td>
</tr>
<tr>
<td>COURSE</td>
<td>4</td>
</tr>
<tr>
<td>SECTION</td>
<td>5</td>
</tr>
<tr>
<td>GRADE_REPORT</td>
<td>3</td>
</tr>
<tr>
<td>PREREQUISITE</td>
<td>2</td>
</tr>
</tbody>
</table>

#### COLUMNS

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Datatype</th>
<th>Belongs to relation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>String</td>
<td>STUDENT</td>
</tr>
<tr>
<td>Student_number</td>
<td>Integer</td>
<td>STUDENT</td>
</tr>
<tr>
<td>Class</td>
<td>String</td>
<td>STUDENT</td>
</tr>
<tr>
<td>Major</td>
<td>String</td>
<td>STUDENT</td>
</tr>
<tr>
<td>Course_name</td>
<td>String</td>
<td>COURSE</td>
</tr>
</tbody>
</table>
1.6 An Example

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Datatype</th>
<th>Belongs to relation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Course_number</td>
<td>Integer</td>
<td>COURSE</td>
</tr>
<tr>
<td>Credit_hours</td>
<td>Integer</td>
<td>COURSE</td>
</tr>
<tr>
<td>Department</td>
<td>String</td>
<td>COURSE</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>Prerequisite_number</td>
<td>Integer</td>
<td>PREREQUISITE</td>
</tr>
</tbody>
</table>

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?", etc. 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?

- Avoiding redundancy ("data normalization"), or having it controlled,
- Levels of access (multiple user interface),
- And we still have the same usability!

But need to be carefull about consistency / referential integrity.

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
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:

<table>
<thead>
<tr>
<th>Task</th>
<th>Assigned to</th>
</tr>
</thead>
<tbody>
<tr>
<td>Install a DBMS on a server.</td>
<td></td>
</tr>
<tr>
<td>Sketch the schema so that the data will not be redundant.</td>
<td></td>
</tr>
<tr>
<td>Write client-side application that uses the DBMS API.</td>
<td></td>
</tr>
<tr>
<td>Establish the purpose of the database.</td>
<td></td>
</tr>
</tbody>
</table>

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:

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

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, Figure 1.3) or (Elmasri and Navathe 2015, Figure 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**

<table>
<thead>
<tr>
<th>Relation name</th>
<th>Number of columns</th>
</tr>
</thead>
<tbody>
<tr>
<td>BUILDING</td>
<td>3</td>
</tr>
<tr>
<td>ROOM</td>
<td>3</td>
</tr>
<tr>
<td>PROF</td>
<td>4</td>
</tr>
</tbody>
</table>

**COLUMNS**

<table>
<thead>
<tr>
<th>Column name</th>
<th>Datatype</th>
<th>Belongs to relation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Building_Name</td>
<td>Character(30)</td>
<td>Building</td>
</tr>
<tr>
<td>GPSLat</td>
<td>Decimal(9,6)</td>
<td>Building</td>
</tr>
<tr>
<td>GPSLon</td>
<td>Decimal(9,6)</td>
<td>Building</td>
</tr>
<tr>
<td>Building_Name</td>
<td>Character(30)</td>
<td>ROOM</td>
</tr>
<tr>
<td>Room_Number</td>
<td>Integer(1)</td>
<td>ROOM</td>
</tr>
<tr>
<td>Floor</td>
<td>Integer(1)</td>
<td>ROOM</td>
</tr>
<tr>
<td>Prof_Name</td>
<td>Character(30)</td>
<td>PROF</td>
</tr>
<tr>
<td>Phone</td>
<td>Integer(10)</td>
<td>PROF</td>
</tr>
<tr>
<td>Email</td>
<td>Character(30)</td>
<td>PROF</td>
</tr>
<tr>
<td>Room_Number</td>
<td>Integer(1)</td>
<td>PROF</td>
</tr>
</tbody>
</table>

**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

![Terminology](image)

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
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.
- 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:

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yellow</td>
<td>Square</td>
<td>10</td>
<td>(5, 3)</td>
<td></td>
</tr>
<tr>
<td>Blue</td>
<td>Rectangle</td>
<td>10</td>
<td>(3, 9)</td>
<td></td>
</tr>
<tr>
<td>Blue</td>
<td>Circle</td>
<td>9</td>
<td>(4, 6)</td>
<td></td>
</tr>
</tbody>
</table>

and the following sets of attributes:

<table>
<thead>
<tr>
<th></th>
<th>{A, B, C, D}</th>
<th>{A}</th>
<th>{B, C}</th>
<th>{D}</th>
</tr>
</thead>
<tbody>
<tr>
<td>Superkey ?</td>
<td>✔</td>
<td>✘</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Key ?</td>
<td>✘</td>
<td>✘</td>
<td>✔</td>
<td>✔</td>
</tr>
</tbody>
</table>

Note that here we “retro-fit” those definitions, in database design, they come first (i.e., you de-
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 R₁ of R₁ ("referencing relation") that references R₂ ("referenced relation") if

- FK refers to R₂ (i.e., the attributes in FK have the same domain(s) as the primary key PK of R₂)
- a value of FK in a tuple t₁ of r₁(R₁) either
  - occurs as a value of PK for some tuple t₂ of r₂(R₂), i.e., t₁[FK] = t₂[PK]
  - is **NULL**

in the first case, we say that “t₁ refers to t₂”.

If there is a foreign key from R₁ to R₂, then we say that there is a referential integrity constraint from R₁ to R₂. We draw it with an arrow from the FK to the PK. Note that it is possible that R₁ = R₂.

2.6 Example
(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\(^3\).

List all the entity integrity constraint and referential integrity constraint.

### 2.7 Transactions and Operations

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

- Retrievals leave the relation state as it is and output a result relation. That is, retrieval: \(\text{relation state} \rightarrow \text{result relation}\)
- Updates change the relation state. That is, update: \(\text{relation state} \rightarrow \text{relation state}\)

They are two constraints for updates:

1. The new relation state must be “valid” (i.e., comply with the state 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.)

\(^3\)https://ntsi.com/drivers-license-format/
2.7 Transactions and Operations

2.7.3 Update (a.k.a. Modify)

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

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** Connect the dots:

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

**Exercise 2.2** What do we call the number of attributes in a relation?

**Exercise 2.3** At the logical level, does the order of the tuples in a relation matter?

**Exercise 2.4** What is the difference between a database schema and a database state?

**Exercise 2.5** What should we put as a value in an attribute if its value is unknown?

**Exercise 2.6** What, if any, is the difference between a superkey, a key, and a primary key?

**Exercise 2.7** Name the two kinds of integrity that must be respected by the tuples in a relation.

**Exercise 2.8** What is entity integrity? Why is it useful?

**Exercise 2.9** 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 2.10** 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 2.11** Give three examples of operations.

**Exercise 2.12** What is the difference between an operation and a transaction?
Exercise 2.13 Consider the following two relations:

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

For each, give a) The arity of the relation, b) A (preferably plausible) example of tuple to insert.

Exercise 2.14 Give three different ways to deal with operations whose execution in isolation would result in the violation of one of the constraint.

Exercise 2.15 Define what is the domain constraint.

Exercise 2.16 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 2.17 Consider the following three relations
1. What are the foreign keys in the ASSIGNED-TO relation? What are they referring to?

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.', 'Junior'> into CONDUCTOR

**Exercise 2.18** Consider the following relation schema and state:

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>Blue</td>
<td>Austin</td>
<td>true</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Yellow</td>
<td>Paris</td>
<td>true</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Purple</td>
<td>Pisa</td>
<td>false</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Yellow</td>
<td>Augusta</td>
<td>true</td>
<td></td>
</tr>
</tbody>
</table>

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 2.19** 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.
Exercise 2.20 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.

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Austin</td>
<td>true</td>
<td>Shelly</td>
</tr>
<tr>
<td>1</td>
<td>Paris</td>
<td>true</td>
<td>Cheryl</td>
</tr>
<tr>
<td>3</td>
<td>Pisa</td>
<td>false</td>
<td>Sheila</td>
</tr>
<tr>
<td>1</td>
<td>Augusta</td>
<td>true</td>
<td>Ash</td>
</tr>
<tr>
<td>1</td>
<td>Pisa</td>
<td>true</td>
<td>Linda</td>
</tr>
</tbody>
</table>

Exercise 2.21 Consider the following two relations:

![Diagram of BUILDING and ROOM 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 2.22 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 Row is Tuple, Column header is Attribute, Table is Relation.
Solution 2.2  The degree, or arity, of the relation.

Solution 2.3  No, it is a set.

Solution 2.4  The schema is the organization of the database (the meta-data), while the state is the content of the database (the data).

Solution 2.5  NULL

Solution 2.6  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 keys, i.e., the key that was chosen.

Solution 2.7  Referential integrity and entity integrity.

Solution 2.8  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 2.9  Yes, the entity integrity constraint.

Solution 2.10  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 2.11  Reading from the database, performing UPDATE or DELETE operations.

Solution 2.12  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 2.13  a) COMPUTER has for arity 4, and OS has for arity 3.

b) ("Linda McFather", 32, 2017, "Purism"), and ("Debian", "Stable", "amd64")

Solution 2.14  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 2.15  The requirement that each tuple must have for an attribute $A$ an atomic value from the domain dom($A$), or NULL.

Solution 2.16  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 2.17
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 2.18
- \{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 2.19 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 2.20 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 2.21
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 2.22 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:

CLASS(Course_Number, Univ_Section_Number, Instructor_Name, Semester, Building_Code, Room_Number, Time, Weekdays, Credit_Hours)

- This relation represents classes taught in a university.
- The goal is to be able to have multiple offerings (classes) of courses over several semesters.
- List three possible candidate keys and describe under what conditions each candidate key would be valid.
- Each candidate key should have between one and three attributes.

Here are some examples of values for the attributes:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Possible Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Course_Number</td>
<td>CSCI3410, CSCI1302</td>
</tr>
<tr>
<td>Building_Code</td>
<td>AH, UH, ECC</td>
</tr>
<tr>
<td>Univ_Section_Number</td>
<td>1, 2, 3</td>
</tr>
<tr>
<td>Room_Number</td>
<td>E127, N118</td>
</tr>
<tr>
<td>Instructor_Name</td>
<td>John Smith, Sophie Adams</td>
</tr>
<tr>
<td>Time</td>
<td>1400, 1230, 0900</td>
</tr>
<tr>
<td>Semester</td>
<td>Spring 2015, Fall 2010, Summer 2012</td>
</tr>
<tr>
<td>Weekdays</td>
<td>M, MW, MWF, T, TH</td>
</tr>
<tr>
<td>Credit_Hours</td>
<td>1, 2, 3, 4</td>
</tr>
</tbody>
</table>

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.

Solutions to Selected Problems

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 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 accommodate 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 2.5 (Relational model for an auction website) A possible solution follows. Two aspects are worth stressing:

- It is possible for a primary key to be a foreign key. It is a rare construct, but useful in this case, where a member may or may not have a seller or buyer profile. You can see this approach detailed at https://stackoverflow.com/a/47563393/. In this case, we make the choice of making the ID attributes of SELLER and BUYER a foreign key to the primary key of MEMBER. However, adding an attribute MEMBER_ID to those tables, that references the primary key of MEMBER, would also be acceptable.
- A bid is placed by a member: the idea is that any member can place a bid, but if a member wins the auction, then that member have to use a buyer account, and to create it if need be.
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.

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 management 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\(^8\)

\(^2\)https://www.mysql.com/
\(^3\)https://mariadb.org/
\(^5\)https://www.postgresql.org/
\(^7\)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 Structured Query Language (SQL) is *the* language for RDBMS, it is made of 4 sublanguages:
  - Data Query Language,
  - Data Definition Language (schema creation and modification),
  - Data Control Language (authorizations, users),
  - Data Manipulation Language (insert, update and delete).

The three last sublanguages being dubbed “Data Manipulation Language”.

### 3.1.2 SQL

#### 3.1.2.1 Yet Another Vocabulary

<table>
<thead>
<tr>
<th>“Common” / Relational</th>
<th>SQL</th>
</tr>
</thead>
<tbody>
<tr>
<td>“Set of databases”</td>
<td>Catalog (named collection of schema)</td>
</tr>
<tr>
<td>“Database”</td>
<td>Schema</td>
</tr>
<tr>
<td>Relation</td>
<td>Table</td>
</tr>
<tr>
<td>Tuple</td>
<td>Row</td>
</tr>
<tr>
<td>Attribute</td>
<td>Column, or Field</td>
</tr>
</tbody>
</table>

#### 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.

#### 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](https://stackoverflow.com/q/7022755).

• SQL is “kind of” case-insensitive\textsuperscript{11}, does not care about spaces and new lines
• In-line comments are what is after \texttt{--}, multi-line comments uses \texttt{/* ...*/}.
• 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/library/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\textsuperscript{12}, the canonical source being MySQL’s documentation\textsuperscript{13}:

• For integer types, you can use \texttt{INTEGER} (or its short-hand notation \texttt{INT}) or \texttt{SMALLINT}.
• For floating-point types, you can use \texttt{FLOAT} and \texttt{DOUBLE} (or its synonym, \texttt{REAL}). MySQL also allows the syntax \texttt{FLOAT (M, D)} or \texttt{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\textsuperscript{14} to use \texttt{DECIMAL(10, 2)} (or its synonym in MySQL \texttt{NUMERIC}).
• Characters can be stored using \texttt{CHAR} and \texttt{VARCHAR}: the length (resp. maximal length) of the \texttt{CHAR} (resp. \texttt{VARCHAR}) has to be declared, and \texttt{CHAR} are right-padded with spaces to the specified length. Historically, 255\textsuperscript{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\textsuperscript{16}” should be used.
• You can store a single bit using \texttt{BIT(1)}, and a boolean using \texttt{BOOLEAN}.
• For date and time types, you can use \texttt{DATE, TIME, DATETIME} and \texttt{TIMESTAMP}.

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

\begin{verbatim}
/* code/sql/HW_FACULTY.sql */

DROP SCHEMA IF EXISTS HW_FACULTY;
CREATE SCHEMA HW_FACULTY;

/*
Or we could have use the syntax:
\end{verbatim}

\footnotetext{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).
\footnotetext{12}{https://www.w3resource.com/mysql/mysql-data-types.php
\footnotetext{14}{https://stackoverflow.com/a/4397416
\footnotetext{15}{https://stackoverflow.com/q/1217466
\footnotetext{16}{https://dba.stackexchange.com/a/183277}
CREATE DATABASE HW_FACULTY;
/
-- Now, let us create a table in it:
CREATE TABLE HW_FACULTY.PROF(
  Fname VARCHAR(15),
  -- No String!
  -- The value "15" was 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.
  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
  -- but you can enter those values only using the "standard"
  -- literals
  -- )
  Last_seen TIME,
  FavoriteFruit ENUM('apple', 'orange', 'pear'),
  PRIMARY KEY(Fname, Hiring)
);
/

Or, instead of using the fully qualified name HW_FACULTY.PROF,
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.

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

DROP SCHEMA IF EXISTS <SchemaName>;

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.

You can also 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.

#### 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_CONSTRAINTS_PART1.sql */
DROP SCHEMA IF EXISTS HW_CONSTRAINTS_PART1;
CREATE SCHEMA HW_CONSTRAINTS_PART1;
USE HW_CONSTRAINTS_PART1;

CREATE TABLE HURRICANE(  
    Name VARCHAR(25) PRIMARY KEY,
```
3.4 Overview of Constraints

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

If we wanted to combine multiple constraints, we would have to follow the order described at https://dev.mysql.com/doc/refman/8.0/en/create-table.html.

MySQL can output a description of those tables for us:

```sql
MariaDB [HW_CONSTRAINTS_PART1]> 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_CONSTRAINTS_PART1]> 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:

```sql
/* code/sql/HW_PK_test.sql */

DROP SCHEMA IF EXISTS HW_PK_test;
CREATE SCHEMA HW_PK_test;
USE HW_PK_test;

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

Note that in this case, a statement like
3.4 Overview of Constraints

**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/library/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

**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_CONSTRAINTS_PART1>`.

* 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:

```sql
/* code/sql/HW_DEFAULT_test.sql */

DROP SCHEMA IF EXISTS HW_DEFAULT_test;
CREATE SCHEMA HW_DEFAULT_test;
USE HW_DEFAULT_test;

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;
```
3.4 Overview of Constraints

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.

3.4.2.1 Primary Keys

Adding a primary key:

\texttt{ALTER TABLE STATE ADD PRIMARY KEY (Name);}

Removing the primary key:

\texttt{ALTER TABLE STATE DROP PRIMARY KEY;}

3.4.2.2 UNIQUE Constraint

Adding a \texttt{UNIQUE} constraint:

\texttt{ALTER TABLE STATE ADD UNIQUE (Postal_abbr);}  

Removing a \texttt{UNIQUE} constraint:

\texttt{ALTER TABLE STATE DROP INDEX Postal_abbr;}

Note the difference between adding and removing the \texttt{UNIQUE} constraint: the parenthesis around \texttt{(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 \texttt{NOT NULL} constraint:

\texttt{ALTER TABLE STATE MODIFY Postal\_abbr CHAR(2) NOT NULL;}  

Removing the \texttt{NOT NULL} constraint:

\texttt{ALTER TABLE STATE MODIFY Postal\_abbr CHAR(2);}  

The syntax of \texttt{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:

\texttt{ALTER TABLE HURRICANE ALTER COLUMN WindSpeed SET DEFAULT 74;}  

Removing the default value:

\texttt{ALTER TABLE HURRICANE ALTER COLUMN WindSpeed DROP DEFAULT;}
3.4 Overview of Constraints

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_CONSTRAINTS_PART1`.`HURRICANE`, CONSTRAINT `HURRICANE_ibfk_1` FOREIGN KEY (`Above`) REFERENCES `STATE` (`Name`))
*/

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';
/*
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.

```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
);
```
3.5 Foreign Keys

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:

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

-- In the following, the entry gets created, but date is 0000-00-00!
INSERT INTO STORM VALUES ('Dummy', 'Hurricane', 120, '2017-17-08');

-- In the following, there's an error, and nothing gets inserted.
INSERT INTO STORM VALUES ('Dummy2', 'Hurricane', 120, DATE '2017-17-08');

-- The next one sets NULL for DATE.
INSERT INTO STORM VALUES ('Irma', 'Tropical Storm', 102, DEFAULT);
```

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

```sql
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?
```sql
UPDATE STATE SET Affected_by = 'Harvey'
    WHERE Name = 'Georgia';
```

```sql
UPDATE STORM SET Name = 'Harley' WHERE Name = 'Harvey';
DELETE FROM STORM
    WHERE Name = 'Harley';
```

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 foreign keys, and to understand how the system reacts to the possible restrictions.

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

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

/*
 * You might encounter a
 * ON UPDATE SET DEFAULT
 * but this reference option (cf.
 * worked only with a particular engine ( 
 * 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;
+------------+--------------+
<table>
<thead>
<tr>
<th>Attribute1</th>
<th>Attribute2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cascade</td>
<td>After Update</td>
</tr>
</tbody>
</table>
+------------+--------------+
1 row in set (0.00 sec)

MariaDB [HW_CONSTRAINTS_PART3]> SELECT * FROM Table_set_null;
+------------+------------+
<table>
<thead>
<tr>
<th>Attribute1</th>
<th>Attribute2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Set null</td>
<td>NULL</td>
</tr>
</tbody>
</table>
+------------+------------+
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;
+------------+--------------+
<table>
<thead>
<tr>
<th>Attribute1</th>
<th>Attribute2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cascade</td>
<td>After Update</td>
</tr>
</tbody>
</table>
+------------+--------------+
1 row in set (0.00 sec)

MariaDB [HW_CONSTRAINTS_PART3]> SELECT * FROM Table_set_null;
+------------+------------+
<table>
<thead>
<tr>
<th>Attribute1</th>
<th>Attribute2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Restrict</td>
<td>NULL</td>
</tr>
<tr>
<td>Set null</td>
<td>NULL</td>
</tr>
</tbody>
</table>
+------------+------------+
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:

![Diagram of PROF and DEPARTMENT tables]

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

CREATE TABLE DEPARTMENT(
    Code CHAR(5) PRIMARY KEY,
3.5 Foreign Keys

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);

Note the structure of the ALTER TABLE command:

• ...KEY Department REFERENCES Code; \Rightarrow error
• ...KEY (Department) REFERENCES (Code); \Rightarrow error
• ...KEY PROF(Department) REFERENCES DEPARTMENT(Code); \Rightarrow ok

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)
);

3.5.3.2 Populating

We can insert multiple values at once:

INSERT INTO DEPARTMENT VALUES
  ('MATH', 'Mathematics', NULL),
  ('CS', 'Computer Science', NULL);

We can specify which attributes we are giving:

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

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'),
  ('caubert', 'CS', 'Clément Aubert');
3.6 A First Look at Conditions

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

\[
\text{UPDATE} \ <\text{table}> \\
\text{SET} \ <\text{attribute1}> = <\text{value1}>, <\text{attribute2}> = <\text{value2}>, \ldots \\
\text{WHERE} \ <\text{condition}>;
\]

\[
\text{SELECT} \ <\text{attribute list, called projection attributes}> \\
\text{FROM} \ <\text{table list}> \\
\text{WHERE} \ <\text{condition}>;
\]

\[
\text{DELETE} \text{ FROM} \ <\text{table list}> \\
\text{WHERE} \ <\text{condition}>;
\]

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.

```sql
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';
```

Note that LIKE is by default case-insensitive, both in MariaDB\(^{17}\) and in MySQL\(^{18}\). 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)

\(^{17}\)https://mariadb.com/kb/en/library/like/

3.7 Three-Valued Logic

3.7.1 Meaning of NULL

**NULL** is

1. Unknown value (“Nobody knows”)
   
   What is the date of birth of Jack the Ripper?\(^{19}\)
   
   Does P equal NP?\(^{20}\)
   
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 XIXth century?

3.7.2 Comparison with Unknown Values

If **NULL** is involved in a comparison, the result evaluates to “Unknown”.

<table>
<thead>
<tr>
<th>AND</th>
<th>T</th>
<th>F</th>
<th>U</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>U</td>
<td>U</td>
<td>F</td>
<td>U</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>OR</th>
<th>T</th>
<th>F</th>
<th>U</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>U</td>
<td>T</td>
<td>F</td>
<td>U</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>NOT</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
</tr>
<tr>
<td>F</td>
</tr>
<tr>
<td>U</td>
</tr>
</tbody>
</table>

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

```
INSERT INTO DEPARTMENT Values ('Hist', 'History', NULL);
SELECT * FROM DEPARTMENT WHERE Head IS NULL;
```

\(^{19}\)https://en.wikipedia.org/wiki/Jack_the_Ripper

SELECT * FROM DEPARTMENT WHERE Head IS NOT NULL;
SELECT COUNT(*) FROM GRADE WHERE Grade IS NULL;

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.

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) for you.

#### 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/.
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;
```

ORDER BY order by ascending order by default.
3.8.6 Aggregate Functions

You can use MAX, SUM, MIN, AVG, COUNT:

```
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:

```
/* code/sql/HW_AVG.sql */

DROP SCHEMA IF EXISTS HW_AVG;
CREATE SCHEMA HW_AVG;
USE HW_AVG;

CREATE TABLE TEST(
    Test INT
);

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

SELECT AVG(Test) FROM TEST;
-- Returns 5.0
```

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. Aliases 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.

### 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

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

• 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.
```

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

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

We can have two join conditions!

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

Note that for the kind of join we are studying (called “inner joins”), the order does not matter\textsuperscript{21}.

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:

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

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

```sql
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;
```

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:

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

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 distinguishing 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:

\textsuperscript{21}https://stackoverflow.com/q/9614922
### 3.9 More Select Queries

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

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;

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).
- 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 occurring 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);  

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);

Note that

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

  SELECT Login FROM GRADE
  WHERE Grade >= (SELECT MAX(Grade) FROM GRADE);

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');

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");

becomes

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

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

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

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

### 3.10 Procedures

A “stored” procedure is a series of SQL statements that can be “called” from another part of your program. It is pretty much like defining a function, and, exactly like a function, it can take arguments. In MariaDB, you could have the following program.

```sql
/* code/sql/HW_PROCEDURE_EXAMPLES.sql */

DROP SCHEMA IF EXISTS HW_PROCEDURE_EXAMPLES;
CREATE SCHEMA HW_PROCEDURE_EXAMPLES;
USE HW_PROCEDURE_EXAMPLES;

/*
A "procedure" is a serie of statements stored in a schema, that can easily be executed repeatedly.
*/

CREATE TABLE STUDENT(
    Login INT PRIMARY KEY,
    Name VARCHAR(30),
    Major VARCHAR(30),
    Email VARCHAR(30)
);

INSERT INTO STUDENT VALUES (123, "Test A", "CS", "a@a.edu"),
    (124, "Test B", "IT", "b@a.edu"),
    (125, "Test C", "CYBR", "c@a.edu");

DELIMITER // -- This tells mysql not to mistake the ; below for the end of the procedure definition.
-- We temporarily alter the language, and make the delimiter being //.
-- $$ is 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.
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 powerful, and are a way of automating part of the work in your database. In MariaDB, you could have the following program.

```sql
/* code/sql/HW_TRIGGER_EXAMPLE.sql */

DROP SCHEMA IF EXISTS HW_TRIGGER_EXAMPLE;
CREATE SCHEMA HW_TRIGGER_EXAMPLE;
USE HW_TRIGGER_EXAMPLE;

-- I am assuming that using the minus sign twice is also a poor choice.
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();

/*
As the "()" suggests, a procedure can take argument(s).
*/
DELIMITER //
CREATE PROCEDURE STUDENTLOGIN(NameP VARCHAR(30))
BEGIN
    SELECT Login
    FROM STUDENT WHERE NameP = Name;
END;
//
DELIMITER ;

SHOW CREATE PROCEDURE STUDENTLOGIN; -- You can ask the system to give you information
-- About the procedure you just created.
CALL STUDENTLOGIN("Test A"); -- We can pass quite naturally an argument to our procedure.
```
```sql
CREATE TABLE STUDENT(
    Login VARCHAR(30) PRIMARY KEY,
    Average Float
);

SET @number_of_student = 0;

/*
 * SQL supports some primitive form of variables.
 * cf.
 * https://mariadb.com/kb/en/library/user-defined-variables/
 * There is no "clear" form of type
 * 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, mysql just "guess" the type of your value and
    go with it.
 */

CREATE TRIGGER NUMBER_OF_STUDENT
AFTER INSERT ON STUDENT
FOR EACH ROW SET @number_of_student = @number_of_student + 1;
-- As far as I know, this is the only way to increment a
-- variable.

INSERT INTO STUDENT(Login) VALUES ("A"), ("B"), ("C"), ("D");
```
3.11 Triggers

```sql
SELECT COUNT(*) FROM STUDENT; -- We now have four value inserted in the table.
SELECT @number_of_student AS 'Total number of student'; -- And the counter knows it.

/*
* We should not forget to update our counter when a student is removed from our table!
*/

CREATE TRIGGER NUMBER_OF_STUDENT
AFTER DELETE ON STUDENT
FOR EACH ROW SET @number_of_student = @number_of_student - 1;
DELETE FROM STUDENT WHERE Login = "D" || Login = "E";
SELECT COUNT(*) FROM STUDENT; -- Note that our previous query deleted only one student.
SELECT @number_of_student AS 'Total number of student'; -- And the counter knows it.

/*
* Let us now create a table for each individual grade, and a trigger to calculate the average for us.
* Note that the trigger will need to manipulate two tables at the same time.
*/

CREATE TABLE GRADE(
    Student VARCHAR(30),
    Exam VARCHAR(30),
    Grade INT,
    PRIMARY KEY (Student, Exam),
    FOREIGN KEY (Student) REFERENCES STUDENT(Login)
);

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

INSERT INTO GRADE VALUES
    ("A", "Exam 1", 50),
```
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 DataBase Management System, or its community-developed fork, MariaDB. Below are the instruction to install MySQL Community Edition on Windows 10 and macOS, and MariaDB on Linux-based distribution, but both are developed 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, 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, VMware or Virtual Box to run a “sandboxed” environment that should keep your data isolated from our experiments.

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.

The following links could be useful:

22https://www.mysql.com/
23https://mariadb.org/
24https://www.gnu.org/licenses/licenses.html#GPL
26https://www.vmware.com/
27https://www.virtualbox.org/wiki/Downloads
28https://en.wikipedia.org/wiki/Sandbox_(software_development)
3.12 Setting Up Your Work Environment

- https://dev.mysql.com/doc/refman/8.0/en/mysql-installer-workflow.html, and particularly the page on windows installation\(^\text{29}\) and the page on Linux installation using package-managers\(^\text{30}\),
- https://www.linode.com/docs/databases/mysql/how-to-install-mysql-on-debian-8

3.12.1.1 Installing MySQL on Windows 10


   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:

   ![MySQL Installer](image)


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 PowerShell31) 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](https://dev.mysql.com/doc/refman/8.0/en/osx-installation-pkg.html) and download the file from [https://dev.mysql.com/download](https://dev.mysql.com/download) 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

```
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.

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?”.

---

32Yes, 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!*
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”.

3.12.2 Creating a User

This step will create a non-root user\(^{33}\) 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 \texttt{testuser} on our local installation, and give it the password \texttt{password}:

\begin{verbatim}
CREATE USER 'testuser'@'localhost' IDENTIFIED BY 'password';
\end{verbatim}

Then, we grant the user all the privileges on the databases whose name starts with \texttt{HW\_}:

\begin{verbatim}
GRANT ALL PRIVILEGES ON `HW\_%` . * TO 'testuser'@'localhost';
\end{verbatim}

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

And then we quit the DBMS, using

\begin{verbatim}
EXIT;
\end{verbatim}

The message displayed after the two first commands should be

\begin{verbatim}
Query OK, 0 rows affected (0.00 sec)
\end{verbatim}

and the message displayed after the last command should be

\begin{verbatim}
Bye
\end{verbatim}

\(^{33}\)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.
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\(^\text{34}\):

```
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)
```

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;
```

\(^{34}\)Provided the working directory is still `C:\Program Files\MySQL\MySQL Server 8.0\bin` or similar. Cf. \url{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.
We now create a table called TableTest, with 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\(^\text{35}\) 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 chose a very weak password for testuser that is common to all of our installation.

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:

- 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:

<table>
<thead>
<tr>
<th>Data type</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>Char(4)</td>
<td>4, -32</td>
</tr>
<tr>
<td>VarChar(10)</td>
<td>'Train', 'Michelle'</td>
</tr>
<tr>
<td>Bit(4)</td>
<td>TRUE, UNKNOWN</td>
</tr>
</tbody>
</table>

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 deleted, 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 multisets?

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:

```sql
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?

```sql
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?

**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);`
Solution 3.5

<table>
<thead>
<tr>
<th>Data type</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>Int</td>
<td>4, -32</td>
</tr>
<tr>
<td>Char(4)</td>
<td>'trai', 'plol'</td>
</tr>
<tr>
<td>VarChar(10)</td>
<td>'Train', 'Michelle'</td>
</tr>
<tr>
<td>Bit(4)</td>
<td>B'1010', B'0101'</td>
</tr>
<tr>
<td>Boolean</td>
<td>TRUE, UNKNOWN</td>
</tr>
</tbody>
</table>

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:

```
-- You can ignore the first three lines.
DROP SCHEMA IF EXISTS HW_SHORT;
CREATE SCHEMA HW_SHORT;
USE HW_SHORT;
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);
```

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 explicitly.

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:

```
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  \texttt{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:

\texttt{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 \rightarrow FALSE
• TRUE AND UNKNOWN \rightarrow UNKNOWN
• NOT UNKNOWN \rightarrow UNKNOWN
• FALSE OR UNKNOWN \rightarrow FALSE

Solution 3.32  

• TRUE AND TRUE \rightarrow TRUE
• TRUE AND FALSE \rightarrow FALSE
• TRUE AND UNKNOWN \rightarrow UNKNOWN
• FALSE AND FALSE \rightarrow FALSE
• UNKNOWN AND UNKNOWN \rightarrow UNKNOWN
• FALSE AND UNKNOWN \rightarrow 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 lists the login of the professors teaching in the department where the student whose login is “jrakesh” is majoring. It can be rewritten as

```sql
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**

```sql
SELECT SUM(Pages) FROM BOOK;
```

**Solution 3.37**

```sql
ALTER TABLE BOOK ADD COLUMN Pages INT;
```

**Solution 3.38**

```sql
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.

**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, `[...]`
- repetitions 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/library/sql-statements/ and
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.


---

**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:

```sql
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

```sql
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 detailed 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:

```sql
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 information back? Do you notice anything regarding the values we entered for the `ID` attribute?
3.12 Setting Up Your Work Environment

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_FK_test.sql */

DROP SCHEMA IF EXISTS HW_FK_test;
CREATE SCHEMA HW_FK_test;
USE HW_FK_test;

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

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

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.

Problem 3.5 (Revisiting the PROF table) Create the PROF, STUDENT, DEPARTMENT 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).

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).
- 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 weaknesses of this representation. For instance, can it accommodate two instructors for the same class? Write down two possible 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 .3.5 ~**: 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 .3.5 ~**: Update the relational model you previously drew to reflect the new situation of your tables.

**Pb 3.5 – Question 4**

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.

```
CREATE TABLE TRAIN(
    ID VARCHAR(30),
    Model VARCHAR(30),
    ConstructionYear YEAR(4)
);

CREATE TABLE CONDUCTOR(
```
3.12 Setting Up Your Work Environment

ID VARCHAR(20),
Name VARCHAR(20),
ExperienceLevel VARCHAR(20)
);

CREATE TABLE ASSIGNED_TO(
    TrainId VARCHAR(20),
    ConductorId VARCHAR(20),
    Day DATE,
    PRIMARY KEY(TrainId, ConductorId)
);

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’s 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’s 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”?”

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
3.12 Setting Up Your Work Environment

<table>
<thead>
<tr>
<th>Ref</th>
<th>Origin</th>
<th>TypeOfRoast</th>
<th>PricePerPound</th>
</tr>
</thead>
<tbody>
<tr>
<td>001</td>
<td>Brazil</td>
<td>Light</td>
<td>8.90</td>
</tr>
<tr>
<td>121</td>
<td>Bolivia</td>
<td>Dark</td>
<td>7.50</td>
</tr>
<tr>
<td>311</td>
<td>Brazil</td>
<td>Medium</td>
<td>9.00</td>
</tr>
<tr>
<td>221</td>
<td>Sumatra</td>
<td>Dark</td>
<td>10.25</td>
</tr>
</tbody>
</table>

CUSTOMER

<table>
<thead>
<tr>
<th>CardNo</th>
<th>Name</th>
<th>Email</th>
</tr>
</thead>
<tbody>
<tr>
<td>001</td>
<td>Bob Hill</td>
<td><a href="mailto:b.hill@isp.net">b.hill@isp.net</a></td>
</tr>
<tr>
<td>002</td>
<td>Ana Swamp</td>
<td><a href="mailto:swampa@nca.edu">swampa@nca.edu</a></td>
</tr>
<tr>
<td>003</td>
<td>Mary Sea</td>
<td><a href="mailto:brig@gsu.gov">brig@gsu.gov</a></td>
</tr>
<tr>
<td>004</td>
<td>Pat Mount</td>
<td><a href="mailto:pmount@fai.fr">pmount@fai.fr</a></td>
</tr>
</tbody>
</table>

SUPPLY

<table>
<thead>
<tr>
<th>Provider</th>
<th>Coffee</th>
</tr>
</thead>
<tbody>
<tr>
<td>Coffee Unl.</td>
<td>001</td>
</tr>
<tr>
<td>Coffee Unl.</td>
<td>121</td>
</tr>
<tr>
<td>Coffee Exp.</td>
<td>311</td>
</tr>
<tr>
<td>Johns &amp; Co.</td>
<td>221</td>
</tr>
</tbody>
</table>

PROVIDER

<table>
<thead>
<tr>
<th>Name</th>
<th>Email</th>
</tr>
</thead>
<tbody>
<tr>
<td>Coffee Unl.</td>
<td><a href="mailto:bob@cofunl.com">bob@cofunl.com</a></td>
</tr>
<tr>
<td>Coffee Exp.</td>
<td><a href="mailto:pat@coffeex.dk">pat@coffeex.dk</a></td>
</tr>
<tr>
<td>Johns &amp; Co.</td>
<td>NULL</td>
</tr>
</tbody>
</table>

In the following, we will assume that this model was implemented in a DBMS (MySQL or MariaDB), the primary keys being COFFEE.Ref, CUSTOMER.CardNo, SUPPLY.Provider and SUPPLY.Coffee, and PROVIDER.Name, and the foreign keys being as follows:

<table>
<thead>
<tr>
<th>FavCoffee in the CUSTOMER relation refers to</th>
<th>Ref in the COFFEE relation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Provider in the SUPPLY refers to</td>
<td>Name in the PROVIDER relation</td>
</tr>
<tr>
<td>Coffee in the SUPPLY refers to</td>
<td>Ref in the COFFEE relation</td>
</tr>
</tbody>
</table>

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.
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 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.

```sql
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:

```sql
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:

```sql
DELETE FROM CUSTOMER
    WHERE Name LIKE '%S%';

DELETE FROM COFFEE
    WHERE Ref = 001;

DELETE FROM SUPPLY
    WHERE Provider = 'Coffee Unl.'
        AND Coffee = 001;

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),
```
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:

```sql
/* 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(20)
);

CREATE TABLE PRINTER(
    ID VARCHAR(20) PRIMARY KEY,
    Model VARCHAR(20)
);

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'),
```
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 a variation of the COMPUTER table)

Consider the following SQL code:

```sql
/* code/sql/HW_COMPUTER_Variation.sql */
DROP SCHEMA IF EXISTS HW_COMPUTER_Variation;
CREATE SCHEMA HW_COMPUTER_Variation;
USE HW_COMPUTER_Variation;
CREATE TABLE COMPUTER(
    ID VARCHAR(20) PRIMARY KEY,
    Model VARCHAR(20)
);
CREATE TABLE PERIPHERAL(
    ID VARCHAR(20) PRIMARY KEY,
    Model VARCHAR(20),
    Type ENUM ('mouse', 'keyboard', 'screen', 'printer')
);
CREATE TABLE CONNEXION(
    Computer VARCHAR(20),
    Peripheral VARCHAR(20),
    PRIMARY KEY (Computer, Peripheral),
    FOREIGN KEY (Computer)
);```
3.12 Setting Up Your Work Environment

```sql
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');
```

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 Selectric).
6. The number of peripheral connected to the computer whose model is Apple IIc Plus (2).

---

**Problem 3.11 (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 occasions, 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,
```
3.12 Setting Up Your Work Environment

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.)
- 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 3.12 (A simple database for books) Consider the following code:

```sql
/* code/sql/HW_SIMPLE_BOOK.sql */
DROP SCHEMA IF EXISTS HW_SIMPLE_BOOK;
CREATE SCHEMA HW_SIMPLE_BOOK;
USE HW_SIMPLE_BOOK;

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),
```
3.12 Setting Up Your Work Environment

```
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);
```

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.

1. Write a command that selects:
   a) The Title of all the books.
   b) The distinct Name of the publishers.
   c) The Titles and Published dates of the books published since January 31, 2012.
   d) The first and last names of the authors published by "Gallimard" (from any city).
   e) The first and last names of the authors who were not published by an editor in "New-York".
   f) The ID of the authors who published a book whose name starts with "Where".
   g) The total number of pages in the database.
   h) The number of pages in the longest book written by the author whose last name is "Wolve".
   i) The titles of the books published in the 19th century.

2. Write a command that updates the title of all the books written by the author whose ID is 1 to "BANNED". Is there any reason for this command to be rejected by the system? If yes, explain the reason.
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.

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, primary and foreign keys, and, as always, avoid redundancy.

5. Draw the relational model of the database you created (including all the relations given in the code and the ones you added).

6. Discuss two limitations of the model and how to improve it.

---

**Problem 3.13 (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.

```sql
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),
    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');
```

---

You can use the DATE datatype to store a year.
1. Write queries that return the following information. The values returned in this set-up will be in parenthesis, but keep the queries general.
   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.),
### Problem 3.14 (A simple database for published pieces of work)

Consider the following code:

```sql
/* code/sql/HW_WORK.sql */

DROP SCHEMA IF EXISTS HW_WORK;
CREATE SCHEMA HW_WORK;
USE HW_WORK;

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
);```

---

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 Foundation, *.fsf.org, Free Software Foundation, *.shadytest.org, Wikimedia Foundation, *.wikipedia.org).

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;

---

Problems in Database Management Systems

3.12 Setting Up Your Work Environment

101
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.).

• Draw the relational model corresponding to this series of commands.

• 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 unsuccessful insertion.

    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");

• List the rows (A.2, W.1, etc.) modified by the following statements. Be careful about the conditions on foreign keys!


**Problem 3.15 (A simple database for authors of textbooks)** Consider the following code:

```sql
/* code/sql/HW_TEXTBOOK_AUTHORED.sql */
DROP SCHEMA IF EXISTS HW_TEXTBOOK_AUTHORED;
CREATE SCHEMA HW_TEXTBOOK_AUTHORED;
USE HW_TEXTBOOK_AUTHORED;

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)
);
```

- 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”.

- 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.

- 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.

- Discuss two limitations of the model and how to improve it.
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');

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 3.16 (A database for residencies) Consider the following code:

```sql
/* 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
```
3.12 Setting Up Your Work Environment

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.

1. Draw the relational model corresponding to this series of commands (it is not necessary to include the state).
2. Write a command that violates the entity integrity constraint.
3. Write a command that violates the referential integrity constraint.
4. List the rows (P.2, H.1, or "none") modified by the following statements:
   a) UPDATE HOUSE SET COLOR = "green";
   b) DELETE FROM RESIDENCY WHERE House LIKE "1%";
   c) DELETE FROM HOUSE WHERE Address = "456 Second St.";
   d) DELETE FROM PERSON WHERE Birthdate=DATE"1990-02-11";
5. 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 addresses of the houses in the system (11 Third St., 123 Main St., 456 Second St.).
   b) The SSN’s of the people whose first name was not entered in the system (000-00-0000).
   c) All the different colors of houses (white, blue).
   d) The address of the residency of James Baldwin (123 Main St.).
   e) The first name of the oldest person in the database (James).
   f) Michael Keal’s principal residency address (123 Main St.).
   g) The distinct first and last names of the homeowners (Michael Keal, Mridula Warrier).
   h) The SSN’s of the people that have the same principal residency as James Baldwin (000-00-0001).
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.
7. Answer the following short questions from the data in our model, as it is currently:
   a) Is it possible for two people to not have the same last name?
   b) Is it possible for a person to have multiple principal residencies?
   c) Is it possible for a house to not be yellow?
   d) Is it possible for the SSN to be any series of 11 characters?
   e) Is it possible for a person to own any number of houses?
   f) Is it possible for a person to rent at most one house?
8. Consider the data currently in the RESIDENCY table and give a possible primary key.
9. Discuss when the primary key you identified from the previous question for the RESIDENCY table would be a good choice.
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:

```sql
CREATE DATABASE HW_Address;
USE HW_Address;
```

This creates the tables asked for in the problem.

Pb 3.2 – Solution to Q. 2 Omitting the Extra column, we have:

```sql
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    |
```

Pb 3.2 – Solution to Q. 3 We add the foreign key, still omitting the Extra column:

```sql
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`:

```sql
MariaDB [HW_Address]>
SHOW CREATE TABLE ADDRESS;
+-------------------------+
| Table                   |
+-------------------------+
| ADDRESS                 |
| CREATE TABLE `ADDRESS` (`StreetName` varchar(15) NOT NULL, `Number` int(11) NOT NULL, |
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\(^\text{37}\).

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:

```
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:

```
UPDATE ADDRESS SET Habitants = 340 WHERE Number = 120;
```

The error message that it returns is:

\(^{37}\)https://en.wikipedia.org/wiki/Leading_zero
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:

```
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:

```
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:

```
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:

```
INSERT INTO EXAMPLE VALUES ('Train', 4);
```

If we execute this command twice, then SQL is OK with it, and inserts the same tuple twice:

```
SELECT * FROM EXAMPLE;
```

Displays:

```
+-------+------+
| X     | Y    |
| 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) For Questions 1 and 5, we should have:

For the other questions, refer to this code.

```sql
/* code/sql/LECTURE.sql */

-- Question 2
CREATE TABLE 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 LECTURE VALUES
    ('Intro to CS', 'caubert', 2017, '1304'),
    ('Intro to Algebra', 'perdos', 2017, '1405'),
    ('Intro to Cyber', 'aturing', 2017, '1234');

-- 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.

-- Question 3

```
ALTER TABLE GRADE
  ADD COLUMN LectureCode CHAR(5),
  ADD COLUMN LectureYear YEAR(4);

DESCRIBE GRADE;

SELECT * FROM GRADE;

ALTER TABLE GRADE
  ADD FOREIGN KEY (LectureYear, LectureCode)
    REFERENCES LECTURE(Year, Code);

-- The values for LectureCode and LectureYear are set to NULL in all the tuples.

-- Question 4

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';

-- Question 6

SELECT Login, Grade
  FROM GRADE
  WHERE LectureCode='1304'
  AND LectureYear = '2017';

SELECT DISTINCT Instructor
  FROM LECTURE
  WHERE Year = 2017;

SELECT Name, Grade
  FROM STUDENT, GRADE
  WHERE GRADE.LectureCode = 1405
  AND STUDENT/Login = GRADE/Login;
```
3.12 Setting Up Your Work Environment

```
SELECT Year
    FROM LECTURE
    WHERE Code = '1234';

SELECT Name
    FROM LECTURE
    WHERE Year IN
        (SELECT Year
            FROM 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 LECTURE, DEPARTMENT, PROF
    WHERE Year = 2017
    AND Instructor = Head
    AND Head = PROF.Login;
```

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:

```
/* code/sql/TRAIN.sql */

-- 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);

-- 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 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;
Solution to Problem 3.7 (Read, correct, and write SQL statements for the COFFEE database)

Solution to Question 1:

The answers to the rest of the questions are in the following code:

```sql
/* code/sql/HW_DB_COFFEE.sql */
/* Setting up the data */
DROP SCHEMA IF EXISTS HW_DB_COFFEE;
CREATE SCHEMA HW_DB_COFFEE;
USE HW_DB_COFFEE;

CREATE TABLE COFFEE(
    Ref VARCHAR(30) PRIMARY KEY,
    Origin VARCHAR(30),
    TypeOfRoast VARCHAR(30),
    PricePerPound DOUBLE
);

CREATE TABLE CUSTOMER(
    CardNo VARCHAR(30) PRIMARY KEY,
    Name VARCHAR(30),
    Email VARCHAR(30),
    FavCoffee VARCHAR(30),
    FOREIGN KEY (FavCoffee) REFERENCES COFFEE(Ref)
        ON UPDATE CASCADE
        ON DELETE CASCADE
);

CREATE TABLE PROVIDER(
    Name VARCHAR(30) PRIMARY KEY,
    Email VARCHAR(30)
);

CREATE TABLE SUPPLY(
    Provider VARCHAR(30),
    Coffee VARCHAR(30),
    Provider VARCHAR(30)
);
```
3.12 Setting Up Your Work Environment

```sql
COFFEE VARCHAR(30),
PRIMARY KEY (Provider, Coffee),
FOREIGN KEY (Provider) REFERENCES PROVIDER(Name)
ON UPDATE CASCADE
ON DELETE CASCADE,
FOREIGN KEY (Coffee) REFERENCES COFFEE(Ref)
ON UPDATE CASCADE
ON DELETE CASCADE
);

INSERT INTO COFFEE VALUES
(001, 'Brazil', 'Light', 8.9),
(121, 'Bolivia', 'Dark', 7.5),
(311, 'Brazil', 'Medium', 9.0),
(221, 'Sumatra', 'Dark', 10.25);

INSERT INTO CUSTOMER VALUES
(001, 'Bob Hill', 'b.hill@sip.net', 221),
(002, 'Ana Swamp', 'swampa@nca.edu', 311),
(003, 'Mary Sea', 'brig@gsu.gov', 121),
(004, 'Pat Mount', 'pmount@fai.fr', 121);

INSERT INTO PROVIDER VALUES
('Coffee Unl.', 'bob@cofunl.com'),
('Coffee Exp.', 'pat@coffeeex.dk'),
('Johns & Co.', NULL);

INSERT INTO SUPPLY VALUES
('Coffee Unl.', 001),
('Coffee Unl.', 121),
('Coffee Exp.', 311),
('Johns & Co.', 221);

/*
   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.
*/

-- Question 2:
START TRANSACTION;

INSERT INTO CUSTOMER VALUES(005, Bob Hill, NULL, 001);

INSERT INTO COFFEE VALUES(002, "Peru", "Decaf", 3.00);

-- INSERT INTO PROVIDER VALUES(NULL, "Coffee Unl.", "contact@localcof.com");
-- ERROR 1048 (23000): Column 'Name' cannot be null

INSERT INTO SUPPLY VALUES("Johns & Co.", 121);
```
-- INSERT INTO SUPPLY VALUES("Coffee Unl.", 311, 221);
-- ERROR 1136 (21S01): Column count doesn't match value
  count at row 1

-- COMMIT;
-- 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;

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)


3.12 Setting Up Your Work Environment

```sql
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;

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;
```

Solution to Problem 3.8 (Write select queries for the DEPARTMENT table)

1.

```sql
FROM EMPLOYEE, DEPARTMENT
WHERE DEPARTMENT.Name = 'Storage'
    AND EMPLOYEE.Department = DEPARTMENT.ID;
```

2. `SELECT Name

FROM EMPLOYEE`
3.12 Setting Up Your Work Environment

WHERE Hired <= ALL ( SELECT Hired 
FROM EMPLOYEE 
WHERE Hired IS NOT NULL )

3. 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;

Solution to Problem 3.10 (Write select queries for a variation of the COMPUTER table)

SELECT Model FROM COMPUTER WHERE ID = 'A';
SELECT Type FROM PERIPHERAL WHERE ID = '14';
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;

Solution to Problem 3.11 (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 3.12 (A simple database for books) Pb 3.12 – Solution to Q. 1
Here are possible ways of getting the required information:

1. The Title of all the books:
   ```
   SELECT Title FROM BOOK;
   ```

2. The distinct Name of the publishers.
   ```
   SELECT DISTINCT Name FROM PUBLISHER;
   ```

3. The Title and Published date of the books published since January 31, 2012.
   ```
   SELECT Title, Published FROM BOOK
   WHERE Published > DATE'20120131';
   ```

4. The first and last names of the authors published by "Gallimard" (no matter the 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;


   SELECT Title FROM BOOK
   WHERE Published >= DATE'18010101'
   AND Published <= DATE'19001231';

Pb 3.12 – Solution to Q. 2 We can use the following command:

   UPDATE BOOK SET Title = "BANNED"
   WHERE Author = 3;

But, as the pair (title, publication date) is the primary key in the BOOK table, 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 3.12 – 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 some of the books would be
“authorless”.

Pb 3.12 – 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 that relation: this information
can be recovered by looking in the BOOK table for the name of the author of the awarded
book.

Pb 3.12 – Solution to Q. 5 We obtain something as follows:
Note that having two attributes as primary key makes all the referencing more cumbersome.

**Pb 3.12 – Solution to Q. 6** Two of the flaws that come to mind are:

- The choice of the primary key for the BOOK relation: two books with the same title cannot be published on the same day, and that is a serious limitation. Using a primary key like ISBN would be much more appropriate.
- The impossibility 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" book’s ISBN with author’s or editor’s primary key.

---

**Solution to Problem 3.13 (A database for website certificates)** The solution can be read from the following code:

```sql
/* code/sql/HW_CERTIFICATE.sql */

DROP SCHEMA IF EXISTS HW_CERTIFICAT;
CREATE SCHEMA HW_CERTIFICAT;
USE HW_CERTIFICAT;

/*
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_CERTIFICAT`.`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_CERTIFICAT]> UPDATE ORGANIZATION SET SN = "01"
WHERE SN = '02';
ERROR 1451 (23000): Cannot delete or update a parent row: a
foreign key constraint fails
(`HW_CERTIFICAT`.`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.

/* code/sql/WORK_SOL.sql */

For this code to work, you need to execute
the code in
code/sql/HW_WORK.sql
first.

/*
*/

/*
* 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.

START TRANSACTION; -- We don't want to perform the actual insertions.

```
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".

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".

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 (0.003 sec)
-- So, "Successful insertion".

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';
-- 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.

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.

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);
3.12 Setting Up Your Work Environment

```sql
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.Author = "Virginia W." 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;

UPDATE WORK SET Title = "BANNED" WHERE Author = "Virginia W.";
```
-- Does not give an error with the that we currently have.
-- However, since "Title" is the primary key in the WORK
   table, if Virginia W. had authored two pieces of work or
   more, then this command would give an error.

/*
 * 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.
 *
*/

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/ .

Finally, to answer the last question, we could list, among the possible limitations:

- 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!
- 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 e.g. PUBLICATION, with a boolean to
  indicate whenever the publication is numeric or on paper.
- 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 the same update policy
  on both tables.

Solution to Problem 3.15 (A simple database for authors of textbooks) The answers
can be read off from the following snippet:

/* 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);

-- 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;
3.12 Setting Up Your Work Environment

107  -- You can use
108  -- DESCRIBE TEXTBOOK;
109  -- to check that the Price attribute now has a default
110  -- value.
111  
112  -- EXERCISE 5
113  
114  -- Write a command that insert a textbook of
115  -- your choice in the TEXTBOOK table, with the
116  -- price set to the default value.
117  INSERT INTO TEXTBOOK VALUES ('Proof Theory', 9780486490731,
118  DEFAULT);
119  
120  -- You can use
121  -- SELECT * FROM TEXTBOOK;
122  -- to check that the insertion was correctly made.
123  
124  -- EXERCISE 6
125  
126  -- Write a command that creates a table called EDITOR
127  -- with 3 attributes, "Name", "Address" and "Website".
128  -- The "Name" attribute should be the primary key.
129  -- Then, insert two tuples in the EDITOR table, one
130  -- should have the "Name" attribute set to "Pearson".
131  CREATE TABLE EDITOR(
132    Name VARCHAR(30)
133    PRIMARY KEY,
134    Address VARCHAR(255),
135    Website VARCHAR(100)
136  );
137  
138  INSERT INTO EDITOR VALUES
139    ('Pearson', NULL, 'http://pearsoned.com/'),
140    ('Dover', NULL, 'https://store.doverpublications.com/');
141  
142  -- You can use
143  -- DESCRIBE EDITOR;
144  -- to check that the table was actually created, and
145  -- SELECT * FROM EDITOR;
146  -- to check that the values were inserted.
147  
148  -- EXERCISE 7
149  
150  -- 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)
);
Solution to Problem 3.16 (A database for residencies) The file code/sql/HW_RESIDENCY_SOL.sql contains the solution to the code part of this problem.

1. The relational model is:

```
PERSON
  FName  LName  SSN  Birthdate

HOUSE
  Address  Color

RESIDENCY
  Person  House  PrincipalResidence  Status
```

1. To violate the entity integrity constraint, it suffices to insert a tuple with NULL as a value for one of the attribute of a primary key, or a value that was already inserted. Two examples are:
3.12 Setting Up Your Work Environment

```
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`.

2. To violate the referential integrity constraint, it suffices to insert a tuple where the value for one of the attribute of a foreign key does not exist in the referenced table. For instance,

```
INSERT INTO RESIDENCY VALUES ("999-99-9999", NULL, NULL,
  NULL);
```

which 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" at SSN.

3. The answers can be read off from 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).
```
3.12 Setting Up Your Work Environment

ROLLBACK;

START TRANSACTION;

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;

4. The answers can be read off from 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
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;

Note that the query that returns the name of the homeowners can be improved.

-- If we have homonymms 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
3.12 Setting Up Your Work Environment

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)
     FROM RESIDENCY
     WHERE RESIDENCY.Status = "own");

5. 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

1. In our model, as it is, ...
   a) ... two persons can have the same last name.
   b) ... a person can have multiple principal residencies.
   c) ... a house can not be yellow.
   d) ... the SSN can be any series of 11 characters.
   e) ... a person can own any number of houses.
   f) ... a person can rent any number of houses.

2. Considering the given state for the RESIDENCY table, the following two are possible
   primary keys:
   • Person and PrincipalResidence,
   • Person and House

3. 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 e.g. owning multiple units at the same
   address: but this is more an issue with the primary key of HOUSE that an issue with
   the primary key we suggested for RESIDENCY.
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:

- E.R. models: (Elmasri and Navathe 2010, ch. 7) or (Elmasri and Navathe 2015, ch. 3)
- The E.R. 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).

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 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)

<table>
<thead>
<tr>
<th>Feature</th>
<th>Conceptual</th>
<th>Logical</th>
<th>Physical</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Main) Audience</td>
<td>Business</td>
<td>Designer</td>
<td>Programmer</td>
</tr>
<tr>
<td>Entity Names</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Entity Relationships</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Attributes</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Cardinalities</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Primary Keys</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Foreign Keys</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Column Data types (Domain)</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Table Names</td>
<td></td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Column Names</td>
<td></td>
<td></td>
<td>✓</td>
</tr>
</tbody>
</table>
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., Augusta, 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, ZIP)\},
• 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)

<table>
<thead>
<tr>
<th>ENTITY TYPE NAME</th>
<th>Key</th>
<th>Atomic</th>
<th>Multi-Valued</th>
<th>Composite</th>
<th>Derived</th>
<th>Attribute 1</th>
<th>…</th>
<th>Attribute n</th>
</tr>
</thead>
</table>

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 Entity-Relationship Model

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 right to left, and up to 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 Entity-Relationship Model

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 E.R. 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:

<table>
<thead>
<tr>
<th>Relation</th>
<th>Possible Ratio</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>MENTOR : MENTEE</td>
<td>$1 : N$</td>
<td>“A mentor can have multiple mentees, a mentee has at most one mentor.”</td>
</tr>
<tr>
<td>PERSON : SSN</td>
<td>$1 : 1$</td>
<td>“A person has one SSN, a SSN belongs to one person.”</td>
</tr>
<tr>
<td>COURSE : DEPARTMENT</td>
<td>$N : 1$</td>
<td>“A course is offered by one department, a department can offer any number of courses”.</td>
</tr>
<tr>
<td>STUDENT : TEAM</td>
<td>$M : N$</td>
<td>“A student can participate in multiple team, a team can have multiple students.”</td>
</tr>
</tbody>
</table>

We indicate the ratio on the edges:

---

$^2$An alternative notation, detailed later on, will address this shortcoming.
4.2 Entity-Relationship Model

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: attributes 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:

![Diagram](diagram.png)

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:

![Diagram](diagram2.png)
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:

![Diagram of a ternary relationship]

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.

For 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).

It is sometimes impossible to do without relations with arity greater than 2. For instance, consider the following two diagrams³:

---

³Where 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:
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 an 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

```plaintext
FRIEND
Name
POSSESSES
1
PET
Name
Gender
Specie
```

instead of

```plaintext
FRIEND
Name
RELATED TO
1
DEPENDENT
Name
Address
```
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 chose.

- 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,

\[
\begin{align*}
\text{CAR} & \quad \overset{N}{\longrightarrow} \quad \text{CARPOOLING} & \quad \overset{M}{\longrightarrow} \quad \text{PERSON}
\end{align*}
\]

could be drawn as

\[
\begin{align*}
\text{CAR} & \quad (1, 5) \quad \overset{}{\longrightarrow} \quad \text{CARPOOLING} & \quad (0, 3) \quad \overset{}{\longrightarrow} \quad \text{PERSON}
\end{align*}
\]
Figure 4.1: A Quick Overview of the Notations for ER Diagram (courtesy of wikipedia)
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:

\[
\begin{align*}
1 \rightarrow N & \Rightarrow (0, N) \rightarrow (0, 1) \\
1 \rightarrow N & \Rightarrow (1, N) \rightarrow (0, 1) \\
M \rightarrow N & \Rightarrow (1, N) \rightarrow (1, M)
\end{align*}
\]

### 4.2.4.2 Crow’s Foot Notation

- **One**
- **Many**
- **Exactly one**
- **Zero or one**
- **One or many**
- **Zero or many**

Only the max.

Min. and max.

### 4.2.5 Enhanced Entity–Relationship Model

Extended (or Enhanced) E.R. Models (E.E.R.) have additionally:

- **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

From relational models to E.R. models (sometimes needed)

4.3 E.R.-to-Relational Models Mapping

4.3.1 Intro

We have to map all of the following:

<table>
<thead>
<tr>
<th>Entity</th>
<th>Strong, Weak</th>
</tr>
</thead>
<tbody>
<tr>
<td>Attributes</td>
<td>Composite, Key, Atomic, Multi-valued, Partial Key</td>
</tr>
<tr>
<td>Relationships</td>
<td>Binary (1 : 1, N : 1, 1 : N, N : M), n-ary</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Step</th>
<th>Relationship Type</th>
<th>Mapping</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Strong Entity</td>
<td>Relation with all the simple attributes. Decompose complex (composite) attributes. Pick a key to be the PK, if it is composite, take its elements.</td>
</tr>
<tr>
<td>2</td>
<td>Weak Entity</td>
<td>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.</td>
</tr>
<tr>
<td>3</td>
<td>Binary 1 : 1 Relationship Types</td>
<td>Foreign Key, Merge Relations or Cross-Reference approach</td>
</tr>
<tr>
<td>4</td>
<td>Binary 1 : N Relationship Types</td>
<td>Foreign Key or Cross-Reference approach</td>
</tr>
<tr>
<td>5</td>
<td>Binary $M : N$ Relationship Types</td>
<td>Cross-Reference approach</td>
</tr>
<tr>
<td>6</td>
<td>$n$-ary Relationship Types</td>
<td>Cross-Reference approach</td>
</tr>
<tr>
<td>7</td>
<td>Multi-valued Attributes</td>
<td>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.</td>
</tr>
</tbody>
</table>

whose primary key is the foreign key to the relation corresponding to the entity.

1. Foreign Key Approach: Chose 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:

As shown in the diagram, the DESK entity would be split into two relations: DESK and DESK_COLOR.

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:

ENT.A.AND.B.

KeyA KeyB

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:

ENT.A

KeyA

Rel

or

ENT.A

KeyA

REL

KeyA1 KeyA2

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:\footnote{This development 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 represent 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 E.R. diagram.

### 4.3.3 Outro

<table>
<thead>
<tr>
<th>E.R. Model</th>
<th>Relational Model</th>
</tr>
</thead>
<tbody>
<tr>
<td>Entity type</td>
<td>Entity relation</td>
</tr>
<tr>
<td>$1 : 1$ or $1 : N$</td>
<td>Foreign key (or relationship relation)</td>
</tr>
<tr>
<td>$M : N$</td>
<td>Relationship relation and two foreign keys</td>
</tr>
<tr>
<td>$n$-ary relationship type</td>
<td>Relationship relation and $n$ foreign keys</td>
</tr>
<tr>
<td>Simple attribute</td>
<td>Attribute</td>
</tr>
<tr>
<td>Composite attribute</td>
<td>Set of simple component attributes</td>
</tr>
<tr>
<td>Multivalued attribute</td>
<td>Relation and foreign key</td>
</tr>
<tr>
<td>Value set</td>
<td>Domain</td>
</tr>
<tr>
<td>Key attribute</td>
<td>Primary key</td>
</tr>
</tbody>
</table>
4.4 Guidelines and Normal Form

What makes a good database? At the logical (conceptual) and physical (implementation) levels.

Goals:

1. Information preservation (and avoid loss of information)
2. Minimum redundancy
3. Make queries easy (avoid redundant work, make \texttt{SELECT} and select-project-join easy)

For E.R. diagrams, some of the usual techniques\footnote{Cf. for instance \url{http://infolab.stanford.edu/~ullman/fcdb/aut07/slides/er.pdf}.} 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 \texttt{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:

\begin{verbatim}
---------- (Login, Name, AdvisoryName, AdvisorOffice, Major, MajorHead)
----------(Office, PhoneNumber, Building)
\end{verbatim}

1. Advisor without student
2. Delete last student of advisor
3. Advisor change name.
4.4.1.3 **NULL Should Be Rare**

**NULL** has 3 meanings, wastes space, and makes join / nested projections harder.

Example:

\[ \text{STUDENT}(\text{Login}, \ldots, \text{siblingEnrolled}) \]

Transform into “Emergency Contact in University” relation (bonus: allow multiple contacts).

4.4.1.4 **Identical Attributes in Different Tables Should Be (Primary, Foreign) 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**

\[ \text{MARKER}(\text{Owner}, \text{Color}, \text{OwnerOffice}, \text{Brand}, \text{BrandEmail}) \]

\[ \text{TEACHER}(\text{Office}, \text{Name}, \text{Phone}) \]

Corrected to:

\[ \text{MARKER}(\text{Owner}, \text{Color}, \text{Brand}) \]

\[ \text{TEACHER}(\text{Office}, \text{Name}, \text{Phone}) \]

\[ \text{BRAND}(\text{Name}, \text{Email}) \]

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, \( X \rightarrow Y \) 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:

\[
\text{MARKER.Owner, MARKER.Color, MAKER.Brand, TEACHER.Office, TEACHER.Name, TEACHER.Phone, BRAND.Name, BRAND.Email}
\]

Think about their dependencies, and list them:

- TEACHER.Name $\rightarrow$ TEACHER.Office
- BRAND.Name $\rightarrow$ BRAND.Email
- TEACHER.Office $\rightarrow$ TEACHER.Name
- TEACHER.Office $\rightarrow$ TEACHER.Phone
- MAKER.Owner and MARKER.Color $\rightarrow$ 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 $\rightarrow$ TEACHER.Name does not hold, because teachers share offices?
- Maybe TEACHER.Name $\rightarrow$ 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:

<table>
<thead>
<tr>
<th>Att. 1</th>
<th>Att. 2</th>
<th>Att. 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bob</td>
<td>15</td>
<td>Boston</td>
</tr>
<tr>
<td>Bob</td>
<td>13</td>
<td>Boston</td>
</tr>
<tr>
<td>Jane</td>
<td>12</td>
<td>Augusta</td>
</tr>
<tr>
<td>Emily</td>
<td>12</td>
<td>Augusta</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>May hold</th>
<th>Will not hold</th>
</tr>
</thead>
<tbody>
<tr>
<td>Att. 2 $\rightarrow$ Att. 3</td>
<td>Att1 $\rightarrow$ Att2</td>
</tr>
<tr>
<td>Att. 3 $\rightarrow$ Att. 2</td>
<td>Att. 3 $\rightarrow$ Att. 2</td>
</tr>
<tr>
<td>Att. 1 $\rightarrow$ Att. 3</td>
<td>Att. 2 $\rightarrow$ Att. 1</td>
</tr>
<tr>
<td>{Att. 1, Att. 2} $\rightarrow$ Att. 3</td>
<td>{Att. 3, Att. 2} $\rightarrow$ Att. 1</td>
</tr>
</tbody>
</table>
4.4 Guidelines and Normal Form

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 \( \{A, B\} \).
- \( \{A_1, ..., A_n\} \rightarrow \{B_1, ..., B_m\} \) means that \( A_1 \) and ... and \( A_n \) fix \( B_1 \), and that \( A_1 \) and ... and \( A_n \) fix \( B_n \), etc.
- \( \text{FD}_1, \text{FD}_2, ..., \text{FD}_n \) for the list of functional dependencies, \( F \) for all of them.
- \( A \rightarrow B \) does not imply nor refute \( B \rightarrow A \).
- We will not write the FD that are implied by (this variation of) Armstrong’s axioms\(^7\):
  - Reflexivity: If \( Y \) is a subset of \( X \), then \( X \rightarrow Y \)
  - Augmentation: If \( X \rightarrow Y \), then \( \{X, Z\} \rightarrow Y \)
  - Transitivity: If \( X \rightarrow Y \) and \( Y \rightarrow Z \), then \( X \rightarrow Z \)

We will assume that the consequence of those axioms always hold (“closure under those rules”), but will generally not write them explicitly, 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 \( \{A_1, ..., A_n\} \rightarrow Y \) 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.

\(^7\)https://en.wikipedia.org/wiki/Armstrong%27s_axioms
Given a FD \( \{A_1, \ldots, A_n\} \rightarrow Y \),

- It is a full functional dependency if for all \( A_i \), \( \{A_1, \ldots, A_n\}/A_i \rightarrow Y \), does not hold.
- It is a partial dependency otherwise.

A FD \( X \rightarrow Y \) is a transitive 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,
- \( X \rightarrow B \) and \( B \rightarrow Y \) hold

---

### 4.4.4 Normal Forms and Keys

First, Second, Third, Fourth, Fifth normal form ("X"NF). Stronger than the Third, there is the Boyce-Codd NF (BCNF)

If you satisfy \( N \), you 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 First Normal Form

**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.

**Normalization** To be written

#### 4.4.4.2 Second Normal Form

**Definition** 1NF + Every non-prime attribute is fully functionnaly dependent on the primary key.
4.4.4.2 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 $\rightarrow$ Done.
  - No $\rightarrow$ Is it partially dependent on the primary key ?
    * No, it is fully dependent on the primary key $\rightarrow$ Done
    * Yes, it depends only of $\{A'_1, \ldots, A'_k\}$ $\rightarrow$ 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 from $\{A'_1, \ldots, A'_k\}$ to their original counterparts in the original relation.

Course

<table>
<thead>
<tr>
<th>Teacher</th>
<th>Code</th>
<th>Credit Hours</th>
<th>Textbook</th>
<th>Difficulty</th>
</tr>
</thead>
</table>

becomes

Course

<table>
<thead>
<tr>
<th>Teacher</th>
<th>Code</th>
<th>Textbook</th>
<th>Difficulty</th>
</tr>
</thead>
</table>

CreditHours

<table>
<thead>
<tr>
<th>Code</th>
<th>Credit Hours</th>
</tr>
</thead>
</table>

Refinement: 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

$R(\begin{array}{c} A_1 \quad A_2 \quad A_3 \quad A_4 \quad A_5 \end{array})$

applying the algorithm would give

$R(\begin{array}{c} A_1 \quad A_2 \quad A_5 \end{array})$

$R'(\begin{array}{c} A_2 \quad A_3 \end{array})$

$R'(\begin{array}{c} A_2 \quad A_4 \end{array})$

whereas a more subtle algorithm would give
4.4 Guidelines and Normal Form

Note that in both cases, all the relations are in Second Normal Form, though.

Note also that, sometimes, removing the "original" relation may be preferable: cf. an example in Problem 4.25 (COFFEE relation: primary key and normal form).

Note also that if our primary key is a singleton, then there is nothing to do, we are in 2NF as soon as we are in 1NF.

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 $A_1, ..., A_n$:

- Is it prime (i.e., is $A \in \{A_1, ..., A_n\}$)?
  - Yes → Done.
  - No → Is it transitively dependent on the primary key ?
    * No, there is no $\{A'_1, ..., A'_k\}$ such that $\{A_1, ..., A_n\} \rightarrow \{A'_1, ..., A'_k\} \rightarrow A$ and $\{A'_1, ..., A'_k\} \nsubseteq \{A_1, ..., A_n\}$ and $A \notin \{A'_1, ..., A'_k\} \rightarrow$ Done
    * Yes, there is such a $\{A'_1, ..., A'_m\} \rightarrow$ Do the following:
      - Create a new relation with $A$ and $\{A'_1, ..., A'_k\}$, make $\{A'_1, ..., A'_k\}$ 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 $\{A'_1, ..., A'_k\}$ to their original counterparts in the original relation.

4.4.4.4 Examples

We can have a look at another example:

Note that $\{\text{State}, \text{Driver\_Licence\_Num}\}$, 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 \{\text{State, Driver\_Licence\_Num}\} → Governor is not fully functional. A possible way to fix it is to get:

\[
\begin{align*}
\text{DRIVER} & \left( \text{State, Driver\_Licence\_Num, Name} \right) \\
\text{GOVERNOR} & \left( \text{State, Governor} \right)
\end{align*}
\]

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:

\[
\begin{align*}
\text{STUDENT} & \left( \text{Login, Name, Major, Major\_Head} \right) \\
\text{HEAD} & \left( \text{Major, Major\_Head} \right)
\end{align*}
\]

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 dependency is transitive, and breaks the 3NF. A way to fix it is:

\[
\begin{align*}
\text{STUDENT} & \left( \text{Login, Name, Major} \right) \\
\text{HEAD} & \left( \text{Major, Major\_Head} \right)
\end{align*}
\]

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 \rightarrow Y \text{ s.t. } X \text{ 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 Modeling Language\(^8\) is a standard:

- Generic
- Language-independent
- Platform-independent

\(^8\)http://uml.org
Wide, powerful, but also intimidating.

You know UML from object-oriented programming language:

That’s a class diagram, 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.
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 database 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:

![Activity Diagram Example](image)

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 series 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.

<table>
<thead>
<tr>
<th>UML</th>
<th>ER</th>
</tr>
</thead>
<tbody>
<tr>
<td>Class</td>
<td>Entity Type</td>
</tr>
<tr>
<td>Class Name</td>
<td>Entity Name</td>
</tr>
<tr>
<td>Attributes</td>
<td>Attributes</td>
</tr>
<tr>
<td>Operations (or Method)</td>
<td><em>Sometimes</em> Derived Attributes</td>
</tr>
<tr>
<td>Association</td>
<td>Relationship Type</td>
</tr>
<tr>
<td>Link</td>
<td>Relationship Instance</td>
</tr>
<tr>
<td>Multiplicities</td>
<td>Structural Constraint</td>
</tr>
</tbody>
</table>
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 $\text{min .. max}$, 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:

![ER Diagram](image)

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.

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 looking at in this class (expanding 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** 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 4.4** Draw the ER diagram for a “CELLPHONE” entity that has a composite attribute “Plan” (decomposed into ”Carrier” and ”Price”), a “Mobile_identification_number” key attribute, and a multi-valued “App_Installed” attribute.

**Exercise 4.5** Name one difference between a primary key in the relational model, and a key attribute in the ER model.

**Exercise 4.6** What is a derived attribute? Give two examples and justify them.
Exercise 4.7 Invent an entity type with at least one composite attribute and one atomic attribute, but no multi-valued attribute, identify a possible key attribute, and draw the entity type you obtained using the conventions we used in class.

Exercise 4.8 What is the degree of a relationship type?

Exercise 4.9 What is a self-referencing, or recursive, relationship type? Give two examples.

Exercise 4.10 What does it mean for a binary relationship type "Owner" between entity types "Person" and "Computer" to have a cardinality ratio $M : N$?

Exercise 4.11 What are the two possible structural constraints on a relationship type?

Exercise 4.12 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 through $R$ with at most one entity in $B$,
- An entity in $B$ can be in relationship through $R$ with any number of entities in $A$.

Exercise 4.13 Express the constraints represented in the following diagram in plain English.

Exercise 4.14 What does it mean for a binary relationship type “Chair” between entity types “Professor” and “Department” to have a cardinality ratio $1:N$? Would it make sense to have a total participation constraint on one side, and if yes, on which one?

Exercise 4.15 Express the constraints represented in the following diagram in plain English.

Exercise 4.16 For the following binary relationships, suggest cardinality ratios based on the common-sense meaning of the entity types.

<table>
<thead>
<tr>
<th>Entity 1</th>
<th>Cardinality Ratio</th>
<th>Entity 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>STUDENT</td>
<td>:</td>
<td>MAJOR</td>
</tr>
<tr>
<td>CAR</td>
<td>:</td>
<td>TAG</td>
</tr>
</tbody>
</table>
4.5 Unified Modeling Language Diagrams

<table>
<thead>
<tr>
<th>Entity 1</th>
<th>Cardinality Ratio</th>
<th>Entity 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>INSTRUCTOR</td>
<td>:</td>
<td>LECTURE</td>
</tr>
<tr>
<td>INSTRUCTOR</td>
<td>:</td>
<td>OFFICE</td>
</tr>
<tr>
<td>COMPUTER</td>
<td>:</td>
<td>OPERATING_SYSTEM</td>
</tr>
</tbody>
</table>

**Exercise 4.17** Give an example of a binary relationship type of cardinality $1 : N$.

**Exercise 4.18** 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 4.19** 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) for the other attribute(s).

**Exercise 4.20** Draw an ER diagram expressing the total participation of an entity type “BURGER” in a binary relation “CONTAINS” between “BURGER” and “INGREDIENT”. What would be the ratio of such a relation?

**Exercise 4.21** Under what condition(s) can an attribute of a binary relationship type be migrated to become an attribute of one of the participating entity type?

**Exercise 4.22** 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 type and draw the resulting diagram.

**Exercise 4.23** Suppose a “MEMBERSHIP” relationship with an attribute “Level” (e.g., “silver”, “platinum”, 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 type and draw the resulting diagram.

**Exercise 4.24** Below, you can specify roles names in your diagrams for added clarity, and remember to list all the constraints. Assume with have three entity types, “Lecture Notes”, “Class” and “Professor”. Give

1. One example of a ternary relationship between them,
2. Two examples of binary relationships between any two of them,
3. A question that could be answered using one model but not the other (and specify which relationship would be able to answer your question).

**Exercise 4.25** Can we always replace a ternary relationship with three binary relationships? Give an example.

**Exercise 4.26** What is the difference between an entity type and a weak entity type?

**Exercise 4.27** What is a partial key?

**Exercise 4.28** Why do weak entity type have a total participation constraint?

**Exercise 4.29** Invent a weak entity type, its identifying (owner) entity type and the identifying (or supporting) relationship. Both entities should have (partial) key, and each should have at least one composite attribute.
Exercise 4.30 Convert the following ER diagram into a relational model:

```
+----------------+     +----------------+     +----------------+
| Date_Of_Birth | --> | PERSON          | --> | PLACE           |
| SSN           |     | N               |     | Rooms           |
|               |     | STAYS_AT        |     | Address         |
+----------------+     +----------------+     +----------------+
```

Exercise 4.31 What is insertion anomaly? Give an example.

Exercise 4.32 What is deletion anomaly? Is it a desirable feature?

Exercise 4.33 Why should we avoid attributes whose value will often be NULL? Can the usage of NULL be completely avoided?

Exercise 4.34 Consider the following relation:

PROF(SSN, 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 4.35 Consider the following relation:

STUDENT(SSN, 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 4.36 Consider the following relational database schema:

STUDENT(Login_Name, ..., Major, Major_Head)
DEPARTMENT(Code, 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 4.37 Why can we not infer a functional dependency automatically from a particular relation state?
**Exercise 4.38** Consider the relation \( R(A, B, C, D, E, F) \) and the following functional dependencies:

1. \( F \rightarrow \{D, C\}, D \rightarrow \{B, E\}, \{B, E\} \rightarrow A \)
2. \( \{A, B\} \rightarrow \{C, D\}, \{B, E\} \rightarrow F \)
3. \( A \rightarrow \{C, D\}, E \rightarrow F, D \rightarrow B \)

For each set of functional dependency, give a key for \( R \). We want a key, so it has to be minimal.

**Exercise 4.39** Consider the relation \( R(A, B, C, D, E, F) \) and the following functional dependencies:

\[
A \rightarrow \{D, E\}, D \rightarrow \{B, F\}, \{B, E\} \rightarrow A, \{A, C\} \rightarrow \{B, D, F\}, A \rightarrow F
\]

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 4.40** What is a composite attribute in a E.R. diagram? Can a relational schema with composite attribute be in Second Normal Form?

**Exercise 4.41** Consider the relation \( R(A, B, C, D) \) and answer the following:

1. If \( \{A, B\} \) is the only key, is \( \{A, B\} \rightarrow \{C, D\}, \{B, C\} \rightarrow D \) a 2NF? List the nonprime attributes and justify.
2. If \( \{A, B, C\} \) is the only key, is \( A \rightarrow \{B, D\}, \{A, B, C\} \rightarrow D \) a 2NF? List the nonprime attributes and justify.

**Exercise 4.42** 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 \( \{C, D\} \rightarrow E \) a fully functional dependency?
3. Write a set of functional dependencies containing at least one transitive dependency, and justify your answer.

**Exercise 4.43** Consider the relation \( R(A, B, C, D, E) \) and the following functional dependencies:

1. \( C \rightarrow D, \{C, B\} \rightarrow A, A \rightarrow \{B, C, D\}, B \rightarrow E \)
2. \( A \rightarrow \{C, D\}, C \rightarrow B, D \rightarrow E, \{E, C\} \rightarrow A \)
3. \( \{A, B\} \rightarrow D, D \rightarrow \{B, C\}, E \rightarrow C \)

For each one, give one candidate key for \( R \).

**Exercise 4.44** Consider the relation \( R(A, B, C, D, E) \) and answer the following:

1. If \( \{A, B\} \) is the primary key, is \( B \rightarrow E, C \rightarrow D \) a 2NF? List the nonprime attributes and justify.
2. If \( \{A\} \) is the primary key, is \( B \rightarrow C, B \rightarrow D \) a 2NF? List the nonprime attributes and justify.
Exercise 4.45 Consider the relation $R(A, B, C, D, E, F)$, and let $\{B, D\}$ be the primary key, and have additionally the functional dependencies $\{A, D\} \rightarrow E, C \rightarrow F$. This relation is not in 3NF, can you tell why?

Exercise 4.46 Consider the relation $R(A, B, C, D)$ and answer the following:

1. If $A$ is the only key, is $A \rightarrow \{B, C, D\}, \{A, B\} \rightarrow C, \{B, C\} \rightarrow D$ 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 4.47 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 4.48 What are the two different categories of U.M.L. diagram?

Exercise 4.49 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 4.50 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 4.51 Name two reasons why one would want to use a U.M.L. class diagram over an E.R. diagram to represent a conceptual schema.

Exercise 4.52 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 4.53 Convert the following E.R. diagram to a U.M.L. class diagram.

![Diagram showing a relationship between HAND and PERSON]

Exercise 4.54 Briefly explain the difference between an aggregation and a composition association.

Exercise 4.55 How is generalization (or inheritance) represented in a U.M.L. class diagram? Why is such a concept useful?

Exercise 4.56 Convert the following E.R. diagram into a U.M.L. class diagram:

![Diagram showing a relationship between PILOT and PLANE]
Solution to Exercises


Solution 4.2 Name / extension. To have statistics about the extensions, to sort the username by length, etc.

Solution 4.3

Solution 4.4

Solution 4.5 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 4.6 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 an “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 a “Initials” attribute could be determined using the values of the “First Name”, “Middle Name” and “Last Name” attributes.
Solution 4.7

Solution 4.8 The number of participating entity types.

Solution 4.9 A relationship type where the same entity type participates more than once. On seats, "is to the left of", on persons, "is married to".

Solution 4.10 That a person can own multiple computers, and that a computer can have multiple owners.

Solution 4.11 Cardinality ration and participation constraints.

Solution 4.12 We would obtain the following drawing:

Solution 4.13 A key opens only one door, and every key must open at least one door. A door can be opened by multiple key, and some doors may not be opened by keys (think of doors that cannot be locked).

Solution 4.14 It means that a department can have at most one professor being 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 Chair 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 chair a department).

Solution 4.15 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.

Solution 4.16

<table>
<thead>
<tr>
<th>Entity 1</th>
<th>Cardinality Ratio</th>
<th>Entity 2</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>STUDENT</td>
<td>$N : 1$</td>
<td>MAJOR</td>
<td>&quot;A student has one major, but multiple students can have the same major&quot;</td>
</tr>
<tr>
<td>Entity 1</td>
<td>Cardinality Ratio</td>
<td>Entity 2</td>
<td>Explanation</td>
</tr>
<tr>
<td>----------</td>
<td>-------------------</td>
<td>----------</td>
<td>-------------</td>
</tr>
<tr>
<td>CAR</td>
<td>1 : 1</td>
<td>TAG</td>
<td>“A car has exactly one tag, a tag belongs to one particular car.”</td>
</tr>
<tr>
<td>INSTRUCTOR</td>
<td>1 : N</td>
<td>LECTURE</td>
<td>“An instructor can teach multiple lecture, but a lecture is taught by only one person.”</td>
</tr>
<tr>
<td>INSTRUCTOR</td>
<td>1 : N</td>
<td>OFFICE</td>
<td>“An instructor can have multiple office, but an office belongs to only one instructor.”</td>
</tr>
<tr>
<td>COMPUTER</td>
<td>𝑀 ∶ 𝑁</td>
<td>OS</td>
<td>“A computer can have multiple operating system, the same operating system can be installed on more than one computer.”</td>
</tr>
</tbody>
</table>

Some of those choices, of course, are arguable (typically, almost any combination seems reasonable for the INSTRUCTOR : OFFICE relation).

**Solution 4.17** SUPervision, a recursive relationship on EMPLOYEE.

![Hand Belongs Person Diagram](image)

**Solution 4.18**

![Phone Price Weight Diagram](image)

**Solution 4.19**

![Burger Contains Ingredient Diagram](image)

**Solution 4.20**

**Solution 4.21** When the cardinality is 1 : 𝑁, 1 : 1 or 𝑁 : 1, the attribute on the relationship can be migrated “to the 𝑁 side”, or to either side, if there is none. Note that for 𝑛-ary
relationships, at most all but one ratio needs to be 1 for the attribute to be allowed to migrate (and, again, “to the $N$ side”, or to any side, if there is none).

**Solution 4.22** We could have the following:

![Diagram 1](image1)

**Solution 4.23** We could have the following:

![Diagram 2](image2)

**Solution 4.24** 1. A possible example of ternary relationship is:

![Diagram 3](image3)

2. 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 not know what classes are taught by Prof. X).

Solution 4.25 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 4.26 The weak entity type does not have a key attribute, it cannot be distinguished 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 4.27 For a weak entity attribute, it is the attribute that can uniquely identify weak entites that are related to the same owner entity.

Solution 4.28 Otherwise, we could not identify entities in it without owner entity.

Solution 4.29 A possible solution is:
Note that the two composite attributes are “generic”, in the sense that you can re-use those examples easily.

**Solution 4.30** A possible option is:

```
<table>
<thead>
<tr>
<th>NAME</th>
<th>ADDRESS</th>
</tr>
</thead>
<tbody>
<tr>
<td>NAME SSN</td>
<td>Address</td>
</tr>
<tr>
<td>DOB</td>
<td>Rooms</td>
</tr>
<tr>
<td>StaysAt</td>
<td></td>
</tr>
</tbody>
</table>
```

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 4.31** 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 4.32** 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 4.33** Because they waste space, they are ambiguous (N/A, or unknown, or not communicated?), and they make queries harder. No, it is necessary sometimes.

**Solution 4.34** 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 4.35** 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” (referencing 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 4.36** 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 4.37** 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 reasonning about the attributes, and not from the observation of the data.

**Solution 4.38**
1. $F$
2. $\{A, B, E\}$
3. $\{A, E\}$

**Solution 4.39**
1. Only one: $\{A, C\}$,
2. $A \rightarrow F$ by $A \rightarrow D, D \rightarrow F$. 
Solution 4.40 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 4.41 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 4.42 1. A, B and C.
   2. No, because we can remove D,
   3. A \rightarrow D, D \rightarrow E and A \rightarrow E

Solution 4.43 1. \{B, C\}, A
   2. A, \{C, E\},
   3. \{A, D, E\}, \{A, B, E\}

Solution 4.44 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 4.45 \{B, D\} \rightarrow C \rightarrow F breaks the 3NF.

Solution 4.46 1. No. B, C and D are non prime, A \rightarrow \{B, C\} \rightarrow D breaks the 3NF.
   2. No. A, B and D are non prime, B \rightarrow \{A, C\} \rightarrow D breaks the 3NF.

Solution 4.47 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 4.48 The two different categories of U.M.L. diagram are behaviour and structure.

Solution 4.49 Yes, U.M.L. diagram is language-independent and platform-independent.

Solution 4.50 1. Use-case
   2. Sequence diagram
   3. Activity diagram

Solution 4.51 To use direction for association, to have a common language with someone less knowledgeable of other diagrammatic notations. For the concept of integration.

Solution 4.52 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 4.53 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 U.M.L. 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 4.54  Aggregation: associated class can have an existence of its own.

Composition association: class does not exist without the association.

Solution 4.55  Because it avoids redundancy.

Solution 4.56

Problems

Problem 4.1 (Design for your professor)  Your professor designed the following relational model at some point in his career, to help him organizing his exams and the students grades:

<table>
<thead>
<tr>
<th>Table Name and Attributes</th>
<th>Example of Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXAM(Number, Date, Course)</td>
<td>&lt; 1, '2018-02-14', 'CSCI3410'&gt;</td>
</tr>
<tr>
<td>PROBLEM(Statement, Points, Length, Exam)</td>
<td>'&lt;Your professor designed...', 10, '00:10:00', 1&gt;</td>
</tr>
<tr>
<td>STUDENT_GRADE(Login, Exam, Grade)</td>
<td>'&lt;aalyx', 1, 83&gt;</td>
</tr>
</tbody>
</table>

where EXAM.Number, PROBLEM.Statement, STUDENT_GRADE/Login and STUDENT_GRADE.Exam are primary key, and where STUDENT_GRADE.Exam and PROBLEM.Exam both refer to EXAM.Number.

The idea was to have

- The EXAM table storing information about exams,
- One entry per problem in the PROBLEM table, and to associate every problem to an exam,
- The grade of one student for one particular exam stored in the STUDENT_GRADE table.

Unfortunately, this design turned out to be terrible. Describe at least one common and interesting situation where this model would fail to fulfill its purpose, and 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 ((Elmasri and Navathe 2010, Figure 7.24)):

Assume that MOVIES is a populated database. ACTOR is used as a gender-neutral term. 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 have to have been an actor in some 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.
11. A movie can have one director and several producers.
12. There could be some actors who have done a lead role, directed a movie, and produced some 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: A car-insurance company wants to have a database of accidents. An accident involves
4.5 Unified Modeling Language Diagrams

cars, drivers, and it has several aspects: the moment and place where it took place, the amount of damages, and a (unique) report number. A car has a license, a model, a year, and an owner. A driver has an ID, an age, a name, and an address.

One of the interesting choice 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.

Draw the E.R. diagram for this situation. Add attributes for the key attributes if needed, and specify the cardinality ratios and participation constraints.

Problem 4.5 (Reverse engineering by hand) Look at the following relational model, and “reverse-engineer” it to obtain an E.R. diagram:

Problem 4.6 (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 system.

2. Once installed, execute the software. The instructions below were tested for the 6.3.8 version on Debian, and 8.0.15 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, and under a similar name, if not the same.

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”, on “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, you can see the list of schemas in the bottom part of the left panel.

6. Click on “Database”, and then on “Reverse Engineering” (or hit ctrl + r), 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 4.6 (Discovering MySQL Workbench)), click on “next”, and then click on “execute”, “next”, and “close”.

7. You’re back on the previous view, but you should now see “E.E.R. diagram” on the top of the middle panel. Click on “E.E.R. diagram” twice, scroll down if needed, and you should see the E.E.R. diagram.

8. This diagram is not exactly an E.R. diagram, and it is not a U.M.L. diagram either: it is an E.E.R. diagram, that uses Crow’s foot notation. Make sure you understand it.

9. Try to modify the E.E.R. diagram. Make some relations mandatory, change their name, add an attribute, change the name of another, 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 tick on “next” once more, and you should see the SQL code needed to produce the table you just designed using the graphical tool.

---

Problem 4.7 (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 4.8 (From E.R. diagram to Relational model – BIKE) Consider the following E.R. diagram:
Based on this diagram, answer the following: “Is it true that ...”

<table>
<thead>
<tr>
<th>Question</th>
<th>Yes</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;... a customer cannot drop two bikes at the exact same time and date?</td>
<td></td>
<td></td>
</tr>
<tr>
<td>&quot;... two different customers cannot drop two different bikes at the exact same time and date?</td>
<td></td>
<td></td>
</tr>
<tr>
<td>&quot;... an employee cannot repair two bikes at the same time?</td>
<td></td>
<td></td>
</tr>
<tr>
<td>&quot;... a customer can be assigned to more than one employee?</td>
<td></td>
<td></td>
</tr>
<tr>
<td>&quot;... a customer can have a bike repaired by an employee that is not assigned to them?</td>
<td></td>
<td></td>
</tr>
<tr>
<td>&quot;... a bike can be in the database without having been dropped by a customer?</td>
<td></td>
<td></td>
</tr>
<tr>
<td>&quot;... an employee can be asked to repair a bike without having that type of bike as one of their specialty?</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Then, convert that E.R. diagram to the relational model. Try to make as few assumptions as possible.

**Problem 4.9 (From E.R. diagram to Relational model – RECORD)** Consider the following E.R. diagram:
Based on this diagram, is it true that ...

<table>
<thead>
<tr>
<th>Question</th>
<th>Yes</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>a label can have multiple logos?</td>
<td></td>
<td></td>
</tr>
<tr>
<td>a recording can be released by multiple labels, at different dates?</td>
<td></td>
<td></td>
</tr>
<tr>
<td>a record shop can have multiple exclusivities?</td>
<td></td>
<td></td>
</tr>
<tr>
<td>two record shops can have the same address?</td>
<td></td>
<td></td>
</tr>
<tr>
<td>two logos can have the same name?</td>
<td></td>
<td></td>
</tr>
<tr>
<td>two recordings can have the same title?</td>
<td></td>
<td></td>
</tr>
<tr>
<td>a record shop must sell at least one recording?</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Then, convert that E.R. diagram to the relational model. Try to make as few assumptions as possible.

**Problem 4.10 (ER-to-Relation mapping for Country)** Consider the following E.R. schema:
where

- “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 E.R. diagram to a relational database schema.

Problem 4.11 (From business statements to E.R. diagram – UNIVERSITY) Consider the following requirements for a UNIVERSITY database, used to keep track of students’ transcripts.

1. The university keeps track of each student’s name, student number, class (freshman, sophomore, …, graduate), major department, minor department (if any), and degree program (B.A., B.S., …, Ph.D.). Student number has unique values for each student.
2. Each department is described by a name and has a (unique) department code.
3. 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.
4. 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).

Draw an E.R. diagram for that schema. Specify key attributes of each entity type and structural constraints on each relationship type. Note any unspecified requirements, and make appropriate assumptions to make the specification complete.
Problem 4.12 (Normal form of a CAR_SALE relation) Consider the following relation, and its functional dependencies:

\[\text{CAR_SALE}(\text{Car_no, Date_sold, Salesman_no, Commission, Discount_amt})\]

\[
\begin{align*}
\{\text{Car_no, Salesman_no}\} & \rightarrow \{\text{Date_sold, Commission, Discount_amt}\} \\
\text{Date_sold} & \rightarrow \text{Discount_amt} \\
\text{Salesman_no} & \rightarrow \text{Commission}
\end{align*}
\]

and let \{\text{Car_no, Salesman_no}\} be the primary key of this relation.

Based on the given primary key, is this relation in 1NF, 2NF, or 3NF? Why or why not? Normalize it to its third normal form.

Problem 4.13 (Normal form of a simple relation) Consider the following relation:

\[\text{REL}(A, B, C, D, E)\]

Suppose we have the following dependencies:

\[
\begin{align*}
A & \rightarrow D \\
\{A, B\} & \rightarrow C \\
D & \rightarrow E
\end{align*}
\]

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 4.14 (Normal form of a SCHEDULE relation) Consider the following relation:

\[\text{SCHEDULE}(\text{Period_Start, Period_End, Date, Room, Building, Organizer, Length})\]

And the following dependencies:

\[
\begin{align*}
\{\text{Period_Start, Date}\} & \rightarrow \{\text{Room, Period_End}\} \\
\{\text{Period_Start, Length}\} & \rightarrow \text{Period_End} \\
\{\text{Period_Start, Period_End}\} & \rightarrow \text{Length} \\
\{\text{Period_End, Length}\} & \rightarrow \text{Period_Start} \\
\{\text{Date, Period_Start}\} & \rightarrow \text{Organizer} \\
\text{Room} & \rightarrow \text{Building}
\end{align*}
\]

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,
Problem 4.15 *(Normalizing the FLIGHT relation)* Consider the following relation:

\[ \text{FLIGHT(From, To, Airline, Flight, DateHour, HeadQuarter, Pilot, TZDifference)} \]

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 and hour 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 do not have to.

Problem 4.16 *(From business statement to dependencies, BIKE)* This problem asks you to convert business statements into dependencies. Consider the following relation:

\[ \text{BIKE(Serial_no, Manufacturer, Model, Batch, Wheel_size, Retailer)} \]

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.

- Write each of the following dependencies as a functional dependency (the first one is given as an example):
  1. A retailer cannot have two bikes of the same model from different batches. *solution:* \{Retailer, Model\} \rightarrow Batch
  2. The manufacturer and serial number uniquely identifies the bike and where it is sold.
  3. A model number is registered by a manufacturer and therefore cannot be used by another manufacturer.
  4. All bikes in a particular batch are of the same model.
  5. All bikes of a certain model have the same wheel size.
- Based on those statements, what could be a key for this relation?
- 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 4.17 (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 vehicle.
   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 4.18 (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 bandwidth 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 intuitive understanding of this 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 4.19 (Normalization) Consider the relations R and T below, and their functional dependencies (on top of the one induced by the primary keys):

R(EventId, Email, Time, Date, Location, Status)
T(Invno, Subtotal, Tax, Total, Email, Lname, Fname, Phone)

\[
\begin{align*}
\{\text{EventId, Email}\} & \rightarrow \text{Status} \\
\text{EventId} & \rightarrow \{\text{Time, Date, Location}\} \\
\text{Invno} & \rightarrow \{\text{Subtotal, Tax, Total, Email}\}
\end{align*}
\]
Normalize the relations to 2NF and 3NF. Show all relations at each stage (2NF and 3NF) of the normalization process.

---

**Problem 4.20 (Normal form of the BOOK relation)** Consider the following relation for published books:

\[ \text{BOOK(Book_title, Book_type, Author_name, List_price, Author_affil, Publisher)} \]

Suppose we have the following dependencies:

\[
\begin{align*}
\text{Book_title} & \rightarrow \{ \text{Publisher, Book_type} \} \\
\text{Book_type} & \rightarrow \text{List_price} \\
\text{Author_name} & \rightarrow \text{Author_affil}
\end{align*}
\]

- What would be a suitable key for this relation?
- How could this relation not be in first normal form? Explain your answer.
- This relation is not in second normal form: explain why and normalize it.
- Is the relations you obtained at the previous step in third normal form? Explain why, and normalize them if needed.

---

**Problem 4.21 (Normal form of the CONTACT relation)** Consider the relation

\[ \text{CONTACT(Phone, Call_center, Email, Zip, Brand, Website)} \]

and the following functional dependencies:

\[
\begin{align*}
\{ \text{Zip, Brand} \} & \rightarrow \{ \text{Phone} \} \\
\{ \text{Brand} \} & \rightarrow \{ \text{Email} \} \\
\{ \text{Brand} \} & \rightarrow \{ \text{Website} \} \\
\{ \text{Phone} \} & \rightarrow \{ \text{Call_center} \}
\end{align*}
\]

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 4.22 (Normal form of the MESSAGE relation)** This exercise asks you to convert business statements into dependencies. Consider the following relation:

\[ \text{MESSAGE(SenderId, Time, Date, ReceiverId, Content, Length, Attachment, Size)} \]
4.5 Unified Modeling Language Diagrams

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 statement as a functional dependency:
   a) The length of a message can be computed from its content.
   b) The content and attachment 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 content and attachment 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 4.23 (PRINT relation in third normal form) Normalize the following relation to the third normal form.

PRINT

( Author Technique Title Price Size ShippingCost )

Do not forget to indicate all the primary keys in your relations.

Problem 4.24 (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:

\[ \{\text{Doctor}_\text{no}, \text{Patient}_\text{no}, \text{Date}\} \rightarrow \{\text{Diagnosis}\} \]
\[ \{\text{Doctor}_\text{no}, \text{Patient}_\text{no}, \text{Date}\} \rightarrow \{\text{Treatment}\} \]
\[ \{\text{Treatment}, \text{Insurance}\} \rightarrow \{\text{Charge}\} \]
\[ \{\text{Patient}_\text{no}\} \rightarrow \{\text{Insurance}\} \]

1. The designer decided not to add the functional dependency \(\{\text{Diagnosis}\} \rightarrow \{\text{Treatment}\}\). 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 4.25 (COFFEE relation: primary key and normal form) Consider the relation

\[ \text{COFFEE}(\text{Origin} , \text{Type_Of_Roast} , \text{Price} , \text{Roasted_Date} , \text{Best_Before} , \text{Color} , \text{Customer} , \text{Rating}) \]

with the following functional dependencies:

\[
\begin{align*}
\{\text{Origin, Type_Of_Roast}\} & \rightarrow \text{Price} \\
\{\text{Origin, Type_Of_Roast, Customer}\} & \rightarrow \text{Rating} \\
\{\text{Origin, Type_Of_Roast, Roasted_Date}\} & \rightarrow \text{Color} \\
\text{Roasted_Date} & \rightarrow \text{Best_Before}
\end{align*}
\]

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 4.26 (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 given by the intuitive understanding of NICs.
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 at the first step, is (are) the relation(s) you constructed at the second step in 2nd normal form? If yes, explain why, if no, normalize it (them).

Problem 4.27 (From E.R. to relational schema and UML class diagram – CAR_INFO) Consider the following E.R. schema for the CAR_INFO database:
Note that a car can have at most one driver, \( N \) passengers, \( N \) insurances, and that car insurances exist only if they are “tied up” to a car (i.e., they are weak entities, and their 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 that E.R. diagram to a relational database schema.
3. Convert the E.R. diagram to a U.M.L. class diagram. Comparing Figure 7.16 with Figure 7.2 from your textbook should guide you.

**Problem 4.28 (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 that can be reserved. A document is of a particular kind (book, video, or disk), has a title, 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’s a video or a disk, one month otherwise. Of course, a copy can be borrowed by only one patron, but it can be put on hold for one patron while being borrowed.
4.5 Unified Modeling Language Diagrams

1. Draw the E.-R. diagram for this situation. Remember to add all the constraints on your relations.
2. Convert your E.-R. diagram to the relational model.

---

**Problem 4.29 (Using MySQL Workbench’s reverse engineering)** This problem requires you to have successfully completed Pb 4.6 and Pb 4.31.

Using the relational database schema you obtained in Pb 4.31, write the SQL implementation of that database. Then, using MySQL Workbench, use the “Reverse Engineering” function to obtain a E.E.R. diagram of your database, and compare it with the U.M.L. diagram from Pb 4.31. Apart from the difference inherent to the nature of the diagram (i.e., U.M.L. Vs E.E.R.), how are they the same? How do they differ? Is the automated tool as efficient and accurate as you are?

---

**Problem 4.30 (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 statement as a functional dependency:
   a) A model has a fixed layout.
   b) 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 4.31 (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
2. true
3. true
4. true
5. false
6. false
7. false
8. true
9. true
10. true
11. true
12. true
13. false

Solution to Problem 4.3 (ER diagram for car insurance)

Two possible solutions are
**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, since they would disallow a CONTACT to discuss more than one offer, or to be hired by more than one company.
Solution to Problem 4.8 (From E.R. diagram to Relational model – BIKE) “Is it true that ...”

<table>
<thead>
<tr>
<th>Question</th>
<th>Yes</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;... a customer cannot drop two bikes at the exact same time and date?</td>
<td>✔</td>
<td></td>
</tr>
<tr>
<td>&quot;... two different customers cannot drop two different bikes at the exact same time and date?</td>
<td>✔</td>
<td></td>
</tr>
<tr>
<td>&quot;... an employee cannot repair two bikes at the same time?</td>
<td>✔</td>
<td></td>
</tr>
<tr>
<td>&quot;... a customer can be assigned to more than one employee?</td>
<td>✔</td>
<td></td>
</tr>
<tr>
<td>&quot;... a customer can have a bike repaired by an employee that is not assigned to them?</td>
<td>✔</td>
<td></td>
</tr>
<tr>
<td>&quot;... a bike can be in the database without having been dropped by a customer?</td>
<td>✔</td>
<td></td>
</tr>
<tr>
<td>&quot;... an employee can be asked to repair a bike without having that type of bike as one of their specialty?</td>
<td>✔</td>
<td></td>
</tr>
</tbody>
</table>

For the 1 : M relationships that are not identifying, we can chose between the foreign key and the cross-reference approaches. If we use the former, we obtain:

![Diagram of the ER model for BIKE]

We could also have used a combination of both!

Solution to Problem 4.9 (From E.R. diagram to Relational model – RECORD) “Is it true that ...”

<table>
<thead>
<tr>
<th>Question</th>
<th>Yes</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>a label can have multiple logos?</td>
<td>✔</td>
<td></td>
</tr>
<tr>
<td>a recording can be released by multiple labels, at different dates?</td>
<td>✔</td>
<td></td>
</tr>
<tr>
<td>a record shop can have multiple exclusivities?</td>
<td>✔</td>
<td></td>
</tr>
<tr>
<td>two record shops can have the same address?</td>
<td>✔</td>
<td></td>
</tr>
<tr>
<td>two logos can have the same name?</td>
<td>✔</td>
<td></td>
</tr>
<tr>
<td>two recordings can have the same title?</td>
<td>✔</td>
<td></td>
</tr>
<tr>
<td>a record shop must sell at least one recording?</td>
<td>✔</td>
<td></td>
</tr>
</tbody>
</table>
For the $1:M$ relationship \texttt{IS\_AN\_EXCLUSIVITY\_OF}, we can choose between the foreign key and the cross-reference approaches. For the $1:1$ relationship \texttt{USES}, we can use any approach we want (foreign key, merged relation or cross-reference). We chose to merge the two relations \texttt{LABEL} and \texttt{LOGO}, and to have a look-up table for the \texttt{IS\_AN\_EXCLUSIVITY\_OF} relation, and we obtain:

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{diagram.png}
\end{figure}

**Solution to Problem 4.12 (Normal form of a CAR\_SALE relation)** The \texttt{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 in 2nd Normal Form: since \texttt{Date\_sold} \rightarrow \texttt{Discount\_amount} and \texttt{Salesman\_no} \rightarrow \texttt{Commission}, then some attributes (namely \texttt{Discount\_amount} and \texttt{Commission}) are not fully functional dependent on the primary key. Hence, this relation cannot be in 3rd normal form either.

To normalize,

2NF:

\begin{tabular}{|l|l|}
\hline
Relations & Functional Dependencies \\
\hline
\texttt{Car\_Sale1(Car\_no, Date\_sold, Discount\_amt)} & \texttt{Car\_no \rightarrow \{Date\_Sold, Discount\_amt\}} and \\
& \texttt{Date\_Sold \rightarrow Discount\_amt} \\
\texttt{Car\_Sale2(Car\_no, Salesman\_no)} & \texttt{Car\_no \rightarrow Salesman\_no} \\
\texttt{Car\_Sale3(Salesman\_no, Commission)} & \texttt{Salesman\_no \rightarrow Commission} \\
\hline
\end{tabular}

3NF:

\begin{tabular}{|l|l|}
\hline
Relations & Functional Dependencies \\
\hline
\texttt{Car\_Sale1-1(Car\_no, Date\_sold)} & \texttt{Car\_no \rightarrow Date\_Sold} \\
\texttt{Car\_Sale1-2(Date\_sold, Discount\_amt)} & \texttt{Date\_Sold \rightarrow Discount\_amt} \\
\texttt{Car\_Sale2(Car\_no, Salesman\_no)} & \texttt{Car\_no \rightarrow Salesman\_no} \\
\texttt{Car\_Sale3(Salesman\_no, Commission)} & \texttt{Salesman\_no \rightarrow Commission} \\
\hline
\end{tabular}
Solution to Problem 4.13 (Normal form of a simple relation)  
1. \{A, B\} would be a suitable primary key (and, 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:
\[ \begin{align*}
R1(A, B, C) \\
R2(D, E) \\
R3(A, D)
\end{align*} \]

Solution to Problem 4.14 (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 attribute that is 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:

\[ \begin{align*}
\text{SCHEDULE} & (\text{Period\_Start, Date, Room, Organizer, Period\_End}) \\
\text{LENGTH} & (\text{Period\_Start, Period\_End, Length}) \\
\text{ROOM} & (\text{Room, Building})
\end{align*} \]

Solution to Problem 4.16 (From business statement to dependencies, BIKE)

The functional dependencies we obtain are:
1. \{ Manufacturer, Serial\_no \} → \{ Model, Batch, Wheel\_size, Retailer \}
2. Model → Manufacturer
3. Batch → Model
4. \{Model, Manufacturer\} → Wheel\_size

• \{Manufacturer, Serial\_no\}

• If every attribute is atomic, it is in second normal form. \{ Manufacturer, Serial\_no \} → Batch → Model breaks the 3NF.

Solution to Problem 4.17 (From business statement to dependencies, ROUTE)  
The relation we consider is:

\begin{align*}
\text{ROUTE}\text{(Name, Direction, Fare\_zone, Ticket\_price, Type\_of\_vehicle, Hours\_of\_operations)}
\end{align*} 

This problem asks to convert business statements into dependencies.
1. Two different types of vehicle can not operate on routes with the same name.
   \[ \text{Name} \rightarrow \text{Type_of_vehicle} \]

2. The ticket price depends of the fare zone and the type of vehicle.
   \[ \{\text{Fare\_zone}, \text{Type\_of\_vehicle}\} \rightarrow \text{Ticket\_price} \]

3. Both the name and the direction are needed to determine the hours of operations.
   \[ \{\text{Name}, \text{Direction}\} \rightarrow \text{Hours\_of\_operations} \]

4. Two routes with the same name and the same direction must have the same fare zone.
   \[ \{\text{Name}, \text{Direction}\} \rightarrow \text{Fare\_zone} \]

Based on those statements, \{\text{Name}, \text{Direction}\} is the only key for this relation.

---

**Solution to Problem 4.18 (From business statement to dependencies, ISP)**

The relation we consider is:

\[
\text{ISP(ISP, bundle, bandwidth, price, IP, ID, email, time)}
\]

The functional dependencies suggested by the business statement are:

\[
\begin{align*}
\{\text{ISP, bundle}\} & \rightarrow \{\text{bandwidth, price}\} \\
\text{IP} & \rightarrow \text{ISP} \\
\{\text{ISP, ID}\} & \rightarrow \{\text{email, bundle}\} \\
\{\text{ISP, ID, time}\} & \rightarrow \text{IP}
\end{align*}
\]

We obtain the following four relations when we normalize it to the third normal form:

- **BUNDLE**\(\text{(ISP, bundle, bandwidth, price)}\)
- **IP**\(\text{(ISP, IP)}\)
- **CLIENT**\(\text{(ISP, id, email, bundle)}\)
- **CLIENT\_IP**\(\text{(ISP, id, time, IP)}\)

---

**Solution to Problem 4.23 (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 the table \(\text{PRICING(Author, Title, Size, Price)}\), \(\text{ART(Author, Title, Technique)}\), \(\text{SHIPPING\_COSTS(Size, Price)}\), we would obtain three relations that are already in third normal form:

- **PRICING**\(\text{(Author, Title, Size, Price)}\)
- **ART**\(\text{(Author, Title, Technique)}\)
- **SHIPPING\_COSTS**\(\text{(Size, Price)}\)

---

**Solution to Problem 4.24 (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. \(\{\text{Doctor\_no, Patient\_no, Date}\}\) is a primary key for this relation.
3. Because there are no partial dependencies, the given relation is in 2NF already. This however is not 3NF because the Charge is a nonkey attribute that is determined by another nonkey attribute, Treatment. We must decompose further:

CONSULTATION (Doctor_no, Patient_no, Date, Diagnosis, Treatment)
PRICE_LISTING (Treatment, Charge)

---

Solution to Problem 4.25 (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 only primary (i.e., minimal) key.

2. This relation is in first normal form, because it has a primary key (the one we just define), and because all the attributes are atomic. It is not in second normal form, because e.g. the functional dependency \( PK_{COFFEE} \rightarrow Price \) is not fully functionally dependent, since \( \{Origin, Type\_Of\_Roast\} \rightarrow Price \) 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 are always that the 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 4.20 (Normal form of the BOOK relation)

- \{Book Title, Author Name\}
- If an attribute is composite or multi-valued.
- Because of \{ Book_title \} \rightarrow \{ Publisher, Book_type \}. We can normalize it as (Book Title, Publisher, Book Type, List Price), (Author Name, Author Affiliation), (Author Name, Book Title).
- Because of [Book title] \rightarrow [Book type] \rightarrow [List_price] (Book Title, Publisher, Book Type) and (Book Type, List Price), (Author Name, Author Affiliation), (Author Name, Book Title).
Solution to Problem 4.27 (From E.R. 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 perfect.

Note that, during the conversion, we had to make “Insured Car” part of the primary key of CAR INSURANCE.

Solution to Problem 4.28 (From Business Statement to ER Diagram to Relational Model – A Network of Libraries)

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: that suggests the existence of 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. 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 possible 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.

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), but should be discussed.

---

Solution to Problem 4.29 (Using MySQL Workbench’s reverse engineering) We give first the code, then the drawing.

```sql
/* 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)
);
5 Database Applications

Resources

• http://spots.augusta.edu/caubert/teaching/general/java/
• If you experience troubles, https://www.ntu.edu.sg/home/ehchua/programming/howto/ErrorMessage.html#JDBCErrors might be a good read.
• (Elmasri and Navathe 2010, 13.3.2) or (Elmasri and Navathe 2015, Chapter 10) is a condensed, but good read.
• Many textbook on Java includes a part on Databases, cf. for instance [Gaddis2014, Chapter 16].

5.1 Overview

Two options to interact with a database:

• Interactive interface (C.L.I.), what we used so far
• Application program / Database application
  1. Embed SQL commands in your program: a pre-compiler scans the code, extract the SQL commands, execute them on the DBMS. Refer to wikipedia\(^1\), this system is used primarily for C, C++, COBOL or Fortran, and Language Integrated Query\(^2\) to some extend is part of this approach.
  2. Use a library, or Application Programming Interface for accessing the database from application programs.
  3. Create a new language that extends SQL (for instance, PL/SQL\(^3\)).

We will consider the option of using a library. Every database application follows the same routine:

  1. Establish / open the connection
  2. Interact (Update, Query, Delete, Insert)
  3. Terminate / close the connection

But the API varies with the pair Language / DBMS. Here are some of the most commonly used pairs:

<table>
<thead>
<tr>
<th>Language</th>
<th>API</th>
<th>Website</th>
</tr>
</thead>
<tbody>
<tr>
<td>Python</td>
<td>Python Database API</td>
<td><a href="https://www.python.org/dev/peps/pep-0249/">https://www.python.org/dev/peps/pep-0249/</a></td>
</tr>
</tbody>
</table>

\(^1\)https://en.wikipedia.org/wiki/Embedded_SQL
\(^2\)https://docs.microsoft.com/en-us/dotnet/standard/using-linq
\(^3\)http://www.oracle.com/technetwork/database/features/plsql/index.html
5.2 Java’s Way

Java actually uses

- A **protocol** (the API, a class library), Java Database Connectivity (JDBC), common to all DBMS. Essentially, 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.

And the routine is a bit more complex:

1. Import library
2. Load driver (done at execution time)
3. Open connection (create Connection and Statement objects)
4. Interact with DB (use `Statement` object)
5. Close connection

5.3 Flash Intro to Java


5.4 A First Program

We will write and compile a simple java program that manipulates a simple database. 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.

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
);


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);
```

This program owes a lot to the one presented at http://www.ntu.edu.sg/home/ehchua/programming/java/jdbc_basic.html.
5.4.2 Executing Database Application

As we're about to see, a database application needs to be written following this order:

1. Load the API,
2. Try to open the connection, using a try/catch statement.
3. Perform the required actions on the database.
4. Close the connection.
5.4 A First Program

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 refinement is needed when you want to compile 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/
- 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-***-bin.jar” file.
- Copy that file in the same folder as where you intend to write your program.

Once this is done and your program was compiled, you can run it using (where you replace *** with the actual number, of course, e.g. 8.0.15):

```java
java -cp .:mysql-connector-java-***-bin.jar FirstProg
```

in Linux, or

```java
java -cp .;mysql-connector-java-***-bin.jar FirstProg
```

in Windows. The `-cp` option lists the places where java should look for the class used in the program: we are explicitly asking java to use the `mysql-connector-java-***-bin.jar` executable (the driver) to execute our `FirstProg` executable.

If we try to execute `FirstProg` without that flag, we obtain the following error message:

```bash
$ 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)
```

### 5.4.3 The Application Program (java)

```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"
            );
            /* If at execution time you receive an error that starts with */
```
5.4 A First Program

* "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 the previous string,
* i.e., replace the previous line of code with:
* Connection conn = DriverManager.getConnection("jdbc:mysql://localhost:3306/HW_EBOOKSHOP?serverTimezone=UTC",
"testuser","password");
* cf. for instance https://stackoverflow.com/q/26515700
*/
Statement stmt = conn.createStatement();
String strSelect = "SELECT title, price, qty FROM BOOKS WHERE qty > 40"
System.out.println("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++;
}
System.out.println("Total number of records = " + rowCount);
conn.close();

} catch (SQLException ex) {
    ex.printStackTrace();
}
}

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 classes
  - DriverManager, used for managing a set of JDBC drivers,
  - Connection, used to make a connection with a database via the DriverManager,
  - Statement, used to send basic SQL statements via the Connection,
- ResultSet, to retrieve and update the results of a query, returned by a Statement object,
- ResultSetMetadata, to get information about the columns of 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 is close to the code 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.
- We could use 1, 2, and 3 instead of "title", "price" and "qty" in the while loop: getString, getDouble and getInt also take integers, 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-8.0.15.jar FirstProg

then you would obtain:

1  The `SQL` query is: SELECT title, price, qty FROM BOOKS WHERE qty > 40
2  The records selected are:
3  The Lord of the Rings, 44.44, 44
4  Le Petit Prince, 55.55, 55
5  Total number of records = 2

5.4.5 A Variation

If you were to replace the body of try in the previous program with

1  String strSelect = "SELECT * FROM BOOKS";
2  ResultSet rset = stmt.executeQuery(strSelect);
3
4  System.out.println("The records selected are:");
5
6  ResultSetMetaData rsmd = rset.getMetaData();
7  int columnsNumber = rsmd.getColumnCount();
8  String columnValue;
5.5 Mapping Datatypes

```java
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();
}
conn.close();
```

(You can find this code in code/java/FirstProgBis.java)

You would obtain:

1. 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

5.5 Mapping Datatypes

<table>
<thead>
<tr>
<th>SQL</th>
<th>JAVA</th>
</tr>
</thead>
<tbody>
<tr>
<td>INTEGER</td>
<td>int</td>
</tr>
<tr>
<td>CHARACTER(n)</td>
<td>String</td>
</tr>
<tr>
<td>VARCHAR(n)</td>
<td>String</td>
</tr>
<tr>
<td>REAL</td>
<td>float</td>
</tr>
<tr>
<td>DOUBLE</td>
<td>double</td>
</tr>
<tr>
<td>DECIMAL(t,d)</td>
<td>java.math.BigDecimal</td>
</tr>
<tr>
<td>DATE</td>
<td>java.sql.Date</td>
</tr>
<tr>
<td>BOOLEAN</td>
<td>boolean</td>
</tr>
<tr>
<td>BIT(1)</td>
<td>byte</td>
</tr>
</tbody>
</table>

(DECIMAL(t,d) was not previously introduced: the t stands for the number of digits, the d for the precision.)

We cannot always have that correspondence: 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.
5.6 Differences Between `executeQuery`, `executeUpdate` and `execute`

<table>
<thead>
<tr>
<th>Name</th>
<th><code>executeQuery</code></th>
<th><code>executeUpdate</code></th>
<th><code>execute</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>Used for</td>
<td><code>SELECT</code></td>
<td><code>INSERT, UPDATE, DELETE</code></td>
<td><code>Any type</code></td>
</tr>
<tr>
<td>Input Type</td>
<td><code>string</code></td>
<td><code>string</code></td>
<td><code>string</code></td>
</tr>
<tr>
<td>Return Type</td>
<td><code>ResultSet</code></td>
<td><code>int, the number of rows affected by the query</code></td>
<td><code>boolean, true if the query returned a </code>ResultSet<code>, </code>false<code>if the query returned an</code>int<code> or nothing</code></td>
</tr>
</tbody>
</table>

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 5.1 (Advanced Java Programming) uses the modifications discussed below.

5.7.1 Passing Options

We can pass options when connecting to the database:

```java
Connection conn = DriverManager.getConnection("jdbc:mysql://localhost:3306/HW_DBPROG" + "?user=testuser" + "&password=password" + "&allowMultiQueries=true" + "&createDatabaseIfNotExist=true" + "&useSSL=true");
```

`allowMultiQueries` allows to pass multiple queries with one `executeUpdate` statement, and `createDatabaseIfNotExist` is about schema, actually.

5.7.2 Creating a Table

We can create a table with the method `stmt.executeUpdate`. We can use the `getMetaData()` of the `DatabaseMetaData` to obtain information about the tables.
5.7.3 Inserting Values

Use `stmt.executeUpdate` (multiple insertion possible if `allowMultiQueries` was set to true, cf. https://stackoverflow.com/a/10804730/).

Another way of batch processing statements:

```java
stmt.addBatch(insert3);
stmt.addBatch(insert4);
stmt.executeBatch();
```

Note that `executeBatch` may be used “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”, cf. https://docs.oracle.com/javase/tutorial/jdbc/basics/retrieving.html#batch_updates. 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 query with a slot”: parsed and stored on the database, but not executed. When the program gives values, it is executed.

Compared to executing SQL statements directly, prepared statements have three main advantages:

- Reduces parsing time (one time VS as many time as values)
- Minimize bandwidth (send only the parameters, and not the whole query)
- Protect against SQL injections

5.7.5 More Complex Statement Objects

When you create the `Statement` objects, you can give two arguments to the `createStatement` method. The first one will indicate whenever you can scroll in the ResultSets objects that will be created using this `Statement` object. The second indicates whenever you can update the values “from” the ResultSet directly.

The 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:

```java
/* code/java/ScrollingProgram.java */

import java.sql.*;

public class ScrollingProgram {
    public static void main(String[] args) {
        try {
            Connection conn = DriverManager.getConnection("jdbc:mysql://localhost:3306/");
```
// We connect to the database, not to a particular schema.
+ "?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));
5.7 A Second Program

```java
    conn.close();
    } catch (SQLException ex) {
        ex.printStackTrace();
    }
}
```

**Exercises**

**Exercise 5.1** What are the technologies that makes 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** What is the class of the object used to create a ResultSet object?

**Exercise 5.6** Briefly explain what the `next()` method from the ResultSet class does, and give its return type.

**Exercise 5.7** How do you submit a `SELECT` statement to the DBMS?

**Exercise 5.8** What method should be used to perform an `INSERT` command from your program?

**Exercise 5.9** Where is a ResultSet object’s cursor initially pointing? How do you move the cursor forward in the result set?

**Exercise 5.10** Give three navigation methods provided by ResultSet.

**Exercise 5.11**

Explain this JDBC URL format:

```java
jdbc:mysql://localhost:3306/HW_NewDB?createDatabaseIfNotExist=true&useSSL=true
```

**Exercise 5.12** In what class is the `getColumnName` method?

**Exercise 5.13** Assuming `stmt` is a `Statement` object, in the statement

```java
    modif = stmt.executeUpdate(strC);
```

what is

1. the datatype of `modif`?
2. the datatype of `strC`?
3. a(n example of a) possible value for `strC`?

**Exercise 5.14** What is a prepared statement?
Exercise 5.15 Assuming \( ps \) is the prepared statement \texttt{INSERT INTO EXAM VALUES (?, ?)}, write the three statements needed to allocate "Quiz" and "5" to the two slots, and then to execute the prepared statement on the database.

Exercise 5.16 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.). For each error, highlight it precisely and give a short explanation.

```java
/* 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();

            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();

        } catch(SQLException ex) {
            ex.printStackTrace();
        }
    }
}
```
Exercise 5.17  Write a small program that determine whenever the null value from Java is equal to the NULL value in your DBMS.

Solution to Exercises

Solution 5.1  API + driver

Solution 5.2  Because the program will interact with the environment: if this interaction fails (typically, if the connection does not succeed), then we want to be able to catch the exception and recover from that failure.

Solution 5.3  You can find them listed at https://docs.oracle.com/javase/7/docs/api/java/sql/package-summary.html. We used Connection, DatabaseMetaData, ResultSetMetaData, PreparedStatement, Statement, ...

Solution 5.4  DriverManager.getConnection()

Solution 5.5  A Statement object is used to create a ResultSet object, by calling e.g. the executeQuery method.

Solution 5.6  It checks if there is data to read, and if there is, it moves the cursor to read it. It returns a Boolean.

Solution 5.7  Using .executeQuery(strSelect)

Solution 5.8  The executeUpdate or execute methods can be used to perform an INSERT command from our program.

Solution 5.9  Before the first line. Using the next method.

Solution 5.10  first, last, next, previous, relative, absolute methods.

Solution 5.11  Connect to localhost:3306 and create a new database if needed, and use secure connection.

Solution 5.12  In modif = stmt.executeUpdate(strC);, modif is an integer (the number of rows modified by the query), strC is a String (a SQL command), and an example of value is DELETE FROM BOOKS Where Price > 0.5.

Solution 5.13  ResultSetMetaData

Solution 5.14  A prepared statement is a feature used to execute the same (or similar) SQL statements repeatedly with high efficiency.

Solution 5.15

    ps.setString(1, "Quiz");
    ps.setInt(2, 5);
    ps.executeUpdate();

Solution 5.16  The errors are:

- line 16, stmt.executeUpdate(strSelect): executeUpdate cannot be used to perform SELECT statements.
- line 21, this error is subtle: we need to display the last record before using previous(), otherwise it would be just skipped. We can fix this using a do...while loop.
• line 22, String title = rset.getDouble("title");: getDouble returns a double, and hence cannot be stored as a String.
• line 28, ps.executeQuery(): the prepared statement did not receive a value for the ? argument.

You can find the program patched in code/java/ProgWithErrorsPatched.java, the (relevant) diff. is:

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 5.17 Here it is:

/* 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&password=password&createDatabaseIfNotExist=true&serverTimezone=UTC");
            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();
}

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.

5.8 Problems

Problem 5.1 (Advanced Java Programming) Read, execute, break, edit, compile, patch, hack and (most importantly) understand the following program:

/* 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
III. How to insert values
IV. How to use prepared statements
V. How to read backward and write in ResultSets

If you want to run this program multiple times, you have to either:
1. Comment first statement of II. Creating a table
2. Change the name of the schema, from HW_DBPROG to whatever you want
3. Drop the DVD table: connect to your database, and then enter
   USE HW_DBPROG;
5.8 Problems

DROP TABLE DVD;
Or do it from within your program!

If you use option 1, you will keep inserting tuples in
your table: cleaning it with
DELETE FROM DVD;
can help. You can do it from within the program!

*/

import java.sql.*;

public class AdvancedProg {
    public static void main(String[] args) {
        try {
            /* I. Passing options to the database */
            /*
                * I. Passing options to the database
                */

            Connection conn = DriverManager.getConnection("jdbc:mysql://localhost:3306/HW_DBPROG"
                + "?user=testuser" + 
                "&password=password" + 
                "&allowMultiQueries=true" + 
                "&createDatabaseIfNotExist=true" + 
                "&useSSL=true");

            // Read about other options at
            // https://dev.mysql.com/doc/connector-
            // j/5.1/en/connector-j-reference-configuration-
            // properties.html
            //
            // https://jdbc.postgresql.org/documentation/head/connect.html

            Statement stmt = conn.createStatement();
        }
        /*
            * II. Creating a table
            */

        stmt.executeUpdate("CREATE TABLE DVD (" + 
            "Title CHAR(25) PRIMARY KEY, " + 
            "Minutes INTEGER, " + 
            "Price DOUBLE);"

        /* If we were to execute
            * SHOW TABLES

        */
5.8 Problems

* directly in the MySQL interpreter, this would display at the screen
* +--------------------------+
  | Tables_in_HW_NewDataBase |
  +--------------------------+
* | DVD                      |
  +--------------------------+
* But here, to access this information, we will use the connection's metadata.
*/

DatabaseMetaData md = conn.getMetaData();
// DatabaseMetaData is a class used to get information about the database: the driver, the user, the versions, etc.

ResultSet rs = md.getTables(null, null, "%", null);

/*
* You can read at
* https://docs.oracle.com/javase/7/docs/api/java/sql/DatabaseMetaData.html
* the full specification of this method.
* All you need to know, for now, is that the third parameter is
* String tableNamePattern,
* i.e., what must match the table name as it is stored in the database
* Here, by using the wildcard ">%", we select all the table names.
* We can then iterate over the ResultSet as usual:
*/

while (rs.next()) {
    System.out.println(rs.getString(3)); // In the ResultSet returned by getTables, 3 is the TABLE_NAME.
}

/*
* III. Inserting values
*/

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");
Batch Insertion

String insert1 = "INSERT INTO DVD VALUES ('Aa', 129, 0.2)";
String insert2 = "INSERT INTO DVD VALUES ('Bb', 129, 0.2)";
String insert3 = "INSERT INTO DVD VALUES ('Cc', 129, 0.2)";
String insert4 = "INSERT INTO DVD VALUES ('DD', 129, 0.2)";

// Method 1: Using executeUpdate, if the option allowMultiQueries=true was passed in the url given
to getConnection and your DBMS supports it.
stmt.executeUpdate(insert1 + ";" + insert2);

// Method 2: Using the addBatch and executeBatch methods
stmt.addBatch(insert3);
stmt.addBatch(insert4);
stmt.executeBatch();

/*
 * IV. Prepared Statements
 */

// Example 1
sqlStatement = "SELECT title FROM DVD WHERE Price <= ?";
// We have a string with an empty slot, represented by ?.
PreparedStatement ps =
    conn.prepareStatement(sqlStatement); // We create a PreparedStatement object, using that string with an empty slot.
// Note that once the 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/>.
double maxprice = 0.5;
ps.setDouble(1, maxprice); // This statement says "Fill the first slot with the value of maxprice".
ResultSet result = ps.executeQuery(); // And then we can execute the query, and display the results:
System.out.printf("For %.2f you can get:\n", maxprice);
while (result.next()) {
    System.out.printf("\t%s \n", result.getString(1));
}

// Example 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");
purchasedStatement.setInt(2, 124);
purchasedStatement.setDouble(3, 5.4);

rowsAffected = purchasedStatement.executeUpdate(); // You can check "by hand" that this statement was correctly executed.
System.out.println(purchasedStatement.toString() + 
    " changed " + rowsAffected + " row(s).\n");

// If we try to mess things up, i.e., provide wrong datatypes:
purchasedStatement.setString(1, "The Great Dictator");
purchasedStatement.setString(2, "The Great Dictator");
purchasedStatement.setString(3, "The Great Dictator");

// Java compiler will be ok, but we'll have an error at execution time when executing the query. You can uncomment the line below to see for yourself.
//rowsAffected = purchasedStatement.executeUpdate();

// Of course, we can use prepared statement inside loops.
for (int i = 1; i < 5; i++) {
    purchasedStatement.setString(1, "Saw " + i);
purchasedStatement.setInt(2, 100);
purchasedStatement.setDouble(3, .5);
purchasedStatement.executeUpdate();
}

/* V. Reading backward and writing in ResultSets */

// To read backward and write in ResultSets, you need to have a statement with certain options:
Statement stmtNew =
   conn.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,
   ResultSet.CONCUR_UPDATABLE);

/*
 * Those options change two things about the ResultSet we
 * obtain using this statement
 *
 * The first argument is the scrolling level:
 * TYPE_FORWARD_ONLY = default.
 * TYPE_SCROLL_INSENSITIVE = can scroll, but updates
don't impact result set.
 * TYPE_SCROLL_SENSITIVE = can scroll, update impact
 * result set.
 *
 * The second argument is the concurrency level:
 * CONCUR_READ_ONLY: default.
 * CONCUR_UPDATABLE: we can change the database without
issuing SQL statement.
 */

/*
 * 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") + " ");
}

/*
 * Other methods to navigate in ResultSet:
 * first()
 * last()
 * next()
 * previous()
 * relative(x) : move cursor x times (positive = forward,
 * negative = backward)
 * absolute(x) : move to the row number x. 1 is the first.
/*
  * Changing the values
  */

System.out.println("\n\nLet us apply a 50% discount.
  \o 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"));
}

result.absolute(0); // We need to scroll back!

while (result.next()) {
    double current = result.getDouble("price");
    result.updateDouble("price", (current * 0.5));
    result.updateRow();
}
System.out.println("\n\nAfter update, the prices are: \n");

result.absolute(0); // We need to scroll back!

while (result.next()) {
    System.out.printf("%20s \t $%3.2f\n",
                      result.getString("title"),
                      result.getDouble("price"));
}

conn.close();
} catch (SQLException ex) {
    ex.printStackTrace();
}
}

Problem 5.2 (A GUEST Java Program) Consider the code below:

/* code/java/GuestProgram.java */
import java.sql.*;
import java.util.Scanner; // Importing a java API to read
  \o from the keyboard.
// 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=password" + "&allowMultiQueries=true");
            Statement stmt = conn.createStatement();
        } {
            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)" + 
                           ");" + 
                           "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();
        }
    }
}

In the following three exercises, you will add some code where the comment // INSERT HERE Solution to exercises 1, 2 and 3. is to obtain a behavior like the following one (you do not have to reproduce it exactly!), where the user
input is underlined, and hitting “enter” is represented by ↵:

How many guests do you have?

Enter name of guest 1.
Marcus Hells
Enter name of guest 2.
Cynthia Heavens

Oh no, (at least) one of the guest from the blacklist confirmed their presence!
The name of the first one is Marcus Hells.

Do you want to remove all the guests that are on the blacklist and confirmed 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 those exercises.

1. Write a snippet that
   a) Ask the user how many guests they have,
   b) For each guest, ask their name (using key.nextLine(), that returns the String entered by the user),
   c) For each guest name entered, insert in the GUEST table an integer that is incremented after each insertion, the name entered by the user, and NULL.

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 at 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 5.2 (A GUEST Java Program) The file code/java/GuestProgram_Solution.java contains the whole code for you to compile and test.

Pb 5.2 – Solution to Q. 1

We explore two solutions, one with batch processing, the second with prepared statement.

They both start with:

```java
1 // Asking the number of guests.
2 int guest_id;
3 String guest_name;
4 int counter = 0;
```

Then the solution using batch processing could be:
```java
while (counter < guest_total) {
    System.out.print("Enter name of guest " + (counter + 1) + ".\n") ; // Ask the name of the guest.
    guest_name = key.nextLine(); // Read the name of the guest.
    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.
```

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");
    guest_name = key.nextLine();
    ps.setInt(1, counter);
    ps.setString(2, guest_name);
    ps.executeUpdate();
    counter++;
}
```

**Pb 5.2 – Solution to Q. 2**

We let SQL do all the hard work:

```java
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!
    The name of the first one is " + rset.getString(2) + ".\n");
}
```

**Pb 5.2 – Solution to Q. 3**

Similarly, we let SQL do all the hard work:

```java
System.out.print("Do you want to remove all the guests that
    are on the black list" + ", that confirmed their presence? Enter \"Y\" for yes, anything else for no.\n");
if (key.nextLine().equals("Y")) {
    stmt.execute(
```

\texttt{DELETE FROM GUEST}
\texttt{+ " WHERE NAME IN"
+ "(SELECT NAME FROM BLACKLIST)"
+ " AND Confirmed = true;"}
\texttt{)}
\texttt{}
6 A Bit About Security

Resources

6.1 Usual Aspects

6.1.1 Threat Model

- Who is threatening you?
- What are the risks?
  1. Loss of integrity (improper modification)
  2. Loss of availability
  3. Loss of confidentiality (unauthorized disclosure)
- "You are as strong as your weakest link."
- Never trust the user or their computer.

6.1.2 Control Measures

- Access control (user account, passwords, restrictions)
- Inference control (cannot access information about a particular “case”)
- Flow control (prevent indirect access)
- Encryption (salting + encrypting, can be a legal obligation): password + salt -> hashed.

6.1.3 How to Recover?

- Have a plan.

6.2 Specificities Of Databases

6.2.1 Attack

Attacks: buffer overflow, denial of service, weak authentication, privilege escalation, SQL injections.

“Mixing the instructions with the data”: a judge asking “what is your name”, and you answer “Bill, you are now free to go”.

Example with ASP, Active Server Pages, a server-side scripting language:
txtUserId = getRequestString("UserId");
txtSQL = "SELECT * FROM Users WHERE UserId = " + txtUserId;

1. **DROP TABLE** Suppliers; Execute remote command
2. **105 or 1 = 1** Exploit, bypass login screen
3. `admin'--` Line comment, privilege escalation

Can also be used for DBMS fingerprinting.

The source code at `code/java/SimpleInjection_1.java` and `code/java/SimpleInjection_2.java` give two examples of SQL injection for Java programs, and how to fix them using prepared statements is presented at `code/java/SimpleInjection_3.java`.

The gist of `code/java/SimpleInjection_1.java` is that writing a statement like

```java
ResultSet rset = stmt.executeQuery("SELECT * FROM SECRETVIP
← WHERE Name =" + entered + ";");
```

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/SimpleInjection_2.java`, it shows how

```java
stmt.execute("SELECT * FROM SECRETVIP WHERE Name =" + entered + 
← "';");
```

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`.

Finally, `code/java/SimpleInjection_3.java` shows how to use proper statements to avoid this situation.

### 6.2.2 Protections

1. **Backups:**

   ```bash
   mysql dump --all-databases - u testuser -p password - h localhost > dump.sql
   ```

2. **Possible protections from sql injections (-like):**

   a) Prepared Statements (a.k.a. stored procedures)
   b) White list input validation
   c) Escaping (AT YOUR OWN RISK).

3. Be up-to-date, desactivate the options you are not using, read newsfeeds,
Exercises

Exercise 6.1 You forgot your password for an on-line service, and click on their "Forgot your password?" link. You enter your email, and receive a few seconds later an email with your original password in it. What is the issue here? What are the next steps you should take?

Exercise 6.2 Briefly explain what a prepared statement is, and the benefits it provides.

Exercise 6.3 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?

Solution to Exercises

Solution 6.1 The issue is that they are storing your password in clear text, which is an extremely bad practice. This suggests that this service does not care about the security of their users, and that all the information on 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 not possible, remove as much information as possible (credit card, address, email, etc.),
- Contact them to express your worries about this flaw,
- (Optional) See if your account has already been hacked, using a service like https://haveibeenpwned.com/.

Solution 6.2 A prepared statement is stored in a DBMS as a “query with parameters”, a template, waiting for values to be passed to fill those placeholders, or slots, and being executed. It is used to execute the same or similar statements repeatedly with high efficiency:

- Since it is pre-compiled, and compiled only once, it takes less computational resources to be executed.
- 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 an increase in speed.

Moreover, since only the arguments are passed, it prevents SQL injection, when properly utilized.

Solution 6.3 There are two ways:

- I look for places where the program is asking for user-input, and I enter values like 1 OR 1 = 1, or DROP TABLE Users;
- I 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 e.g. https://sqa.stackexchange.com/q/1527/ for more ideas on how to test for injections.
6.2 Specificities Of Databases

Problems

Problem 6.1 (Insecure Java Programming) Consider the following code:

```java
1 System.out.print("Do you want to browse DISK, BOOK or VINYL? 
2 (enter the table name exactly)\n");
3 String table = key.nextLine();
4 System.out.print("How much money do you have?\n");
5 String max = key.nextLine();
6 ResultSet rst = stmt.executeQuery("SELECT Title FROM " + 
7 table + " WHERE PRICE <= " + max + ");
8 System.out.printf("Here are the %s you can afford with %s:
9 \n", table, max);
10 while (rst.next()){System.out.printf("	- %s 
11", rst.getString(1));}
```

Assume this software is connecting to a schema in a database hosted at http://example.com/ using

```java
1 Connection conn = DriverManager.getConnection(
2   "jdbc:mysql://example.com/:3306/?user=admin&password=admin");
```

that 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 security errors. 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 absence thereof.

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 connexion 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.
• Having admin / admin as a login / password is unforgivable. They 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 server-side, that performs the connexion to the database.

• Not using 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, I used

- (Sadalage and Fowler 2012), -https://en.wikipedia.org/wiki/NoSQL
- (Sullivan 2015),
- (Elmasri and Navathe 2015, Chapter 24),
- Andrew2016
  • https://dl.acm.org/doi/10.1145/1773912.1773922 and
  • https://docs.datastax.com/en/articles/cassandra/cassandrathenandnow.html

7.1 A Bit of History

Inspired from (Sadalage and Fowler 2012, Chap. 1)

7.1.1 Database Applications and Application Databases

When you write a Database application, you have two options:

1. One database for many softwares
2. One database for each softwares

The first option can cause severe impacts on the efficiency of your database: since maintaining the integrity of the database is a requirement, a lot of synchronization is needed. With the second option, you develop an “application database”, and you have more freedom of choice: since only a program interact with a database, you can chose whatever data management you want.

But people were attached to SQL and kept using it.

7.1.2 Clusters, clusters...

Increase in everything (traffic, size of data, number of clients, etc.) meant "up or out", and there was two ways to increase the resources:

1. Bigger machines
2. More machines

The second option was generally less expensive, but came with two drawbacks w.r.t. databases:

1. Cost of licences,
2. Force to perform “unnatural acts”: relational model are really not made to be distributed
7.1.3 A First Shift

- 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).

Particular, big company, with specific needs, but people interested in solving some of their problems. Now, people started to think that there could be other ways.

One goal was to get rid of “impedance mismatch”: mapping classes or objects to database tables defined by a relational schema is complex and cumbersome.

Some issues:

- No absolute notion of “private” and “public” in RDBMS (relative to needs)
- Data-type differences (no pointer, weird way of defining string, etc.)
- Value in a relational structure have to be simple (no complex datatype, no structure)

“Impedance mismatch” is that annoying need for a translation.

Also, the data is now

- moving
- growing
- too diverse

for traditional relational DBMS.

7.1.4 Gathering Forces

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 with “no-cat”). The original meet-up asked for “open-source, distributed, nonrelational database”. Today, no official definition, but NoSQL often implies the following:

- No relational model
- Not using SQL. Some still have a query language, and it resembles 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.

Most importantly: polyglot persistence, “using different data storage technologies to handle varying data storage needs.”

\(^1\)https://cloud.google.com/bigtable/
\(^2\)https://aws.amazon.com/dynamodb/
7.1.5 The Future or the Past?

A lot of enthusiasm, also because it “frees the data” (and, actually, the metadata, cf. application/ld+json, JavaScript Object Notation for Linked Data, schema.org, etc.). 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. Does not mean SQL (“OldSQL”) and relational database are over: still useful in many scenario, and the powerful 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. MongoDB announced that it would have more and more of the ACID properties! https://www.mongodb.com/blog/post/multi-document-transactions-in-mongodb

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.

7.2 Comparison

7.2.1 Overview

« Comparaison n’est pas raison » ³

- Semi-structured data (no schema)
- High performance
- Availability
- Data Replication (improves availability and performance)
- Scalability (horizontal scalability (add nodes) instead of vertical (add memory))
- Eventual Consistency
- Natively versionning

Vs

- Immediate data consistency
- Powerful 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 is the guarantee of validity even in the event of errors, power failures, etc.

- Atomicity → Transactions are all or nothing
- Consistency → Transactions maintains validity

³A 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 committed, 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 is Basic Availability, Soft state, Eventual consistency.

### 7.3 Categories of NoSQL Systems

<table>
<thead>
<tr>
<th>Model</th>
<th>Description</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>Document-based</td>
<td>Data is stored as “documents” (JSON, for instance), accessible via their ID (other indexes available).</td>
<td>Apache CouchDB⁴ (simple for web applications, and reliable), MongoDB⁵ (easy to operate), Couchbase⁶ (high concurrency, and high availability).</td>
</tr>
<tr>
<td>Key-value stores</td>
<td>Fast access by the key to the value. Value can be a record, an object, a document, or be even more complex.</td>
<td>Redis⁷ (in-memory but persistent on disk database, stores everything in the RAM!)</td>
</tr>
<tr>
<td>Column-based (a.k.a. wide column)</td>
<td>Partition a table by columns into column families, where each column family is stored in its own files.</td>
<td>Cassandra⁸, HBase⁹ (both for huge amount of data)</td>
</tr>
<tr>
<td>Graph-based</td>
<td>Data is represented as graphs, and related nodes can be found by traversing the edges using path expressions.</td>
<td>Neo⁴⁰¹⁰ (excellent for pattern recognition, and data mining).</td>
</tr>
<tr>
<td>Multi-model</td>
<td>Support multiple data models</td>
<td>Apache Ignite¹¹, ArangoDB¹², etc.</td>
</tr>
</tbody>
</table>
7.4 MongoDB

7.4.1 Resources

- https://mongodb.github.io/mongo-java-driver/3.4/driver/getting-started/quick-start/
- https://docs.mongodb.com/manual/administration/security-checklist/
- https://docs.mongodb.com/getting-started/shell/
- https://university.mongodb.com
- https://en.wikipedia.org/wiki/MongoDB
- https://www.w3schools.com/xml/schema_example.asp
- https://www.w3schools.com/nodejs/nodejs_mongodb_join.asp
- http://api.mongodb.com/
- (Sadalage and Fowler 2012, ch. 9)
- (Sullivan 2015, ch. 6)

7.4.2 Introduction

MongoDB is

- free (business model: training, support, DB as service, they actually developed MongoDB because they wanted a good solution for a cloud solution!),
- open-source (even if recent changes makes their license not really open source\textsuperscript{13})
- cross-platform
- document-oriented (JSON-like documents with schemas).

And there are drivers\textsuperscript{14} for C, C++, C#, Hadoop Connector, Haskell, Java, node.js, PHP, Perl, Python, Ruby, Scala (Casbah).

The mongo shell is an interactive JavaScript interface to MongoDB.

7.4.3 Document-Oriented Database

Document-oriented database (document store) contains semi-structured data, it is a subclass of the key-value store:

- Relational databases (RDB) pre-define the data structure in the database (fields + data type).

\textsuperscript{4}https://couchdb.apache.org/
\textsuperscript{5}https://www.mongodb.com/
\textsuperscript{6}https://www.couchbase.com/
\textsuperscript{7}https://redis.io/
\textsuperscript{8}https://cassandra.apache.org/
\textsuperscript{9}https://redis.io/
\textsuperscript{10}https://cassandra.apache.org/
\textsuperscript{11}https://hbase.apache.org/
\textsuperscript{12}https://neo4j.com/
\textsuperscript{13}https://opensource.org/LicenseReview122018
\textsuperscript{14}https://docs.mongodb.com/ecosystem/drivers/
- Key-value (KV) treats the data as a single opaque collection, which may have any number (incl. 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 waste space (placeholders for optional values). KV does not allow any optimization, but flexibility and more closely follows modern programming concepts. DO has the flexibility of KV, and allow 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.

Pro:
- Mapping objects to a DB simpler
- Change “in place”
- Increase speed of deployment

Document:
- Implementations differs on the details of the definition, but always the central notion. MongoDB has its own implementation, but there are ~ 45 others. MongoDB is the most popular one (next: Amazon DynamoDB, Couchbase, CouchDB)
- Documents encapsulate and encode data (Self-Describing Data)
- Do not need to adhere a standard schema.
- One program can have many different types of objects, and those objects often have many optional fields
- Formats: XML, YAML, JSON, PDF, etc.

MongoDB uses JSON to BSON (portmanteau of the words “binary” and “JSON”), and actually extend JSON. Think of BSON as a binary representation of JSON (JavaScript Object Notation) documents.

An example of XML (Extensible Markup Language) document (you can actually convert from XML to JSON):

```xml
<shiporder orderid="889923">
  <orderperson>John Smith</orderperson>
  <shipto>
    <name>Ola Nordmann</name>
    <address>Langgt 23</address>
    <city>4000 Stavanger</city>
    <country>Norway</country>
  </shipto>
  <item>
    <title>Empire Burlesque</title>
    <note>Special Edition</note>
    <quantity>1</quantity>
    <price>10.90</price>
  </item>
  <item>
    <title>Hide your heart</title>
    <quantity>1</quantity>
  </item>
</shiporder>
```
7.4 MongoDB

• Invalid document exists!
• Human and computer-readable
• No predefined tags
• Extensible

7.4.4 General Organization of MongoDB Databases

<table>
<thead>
<tr>
<th>RDBMS</th>
<th>MongoDB</th>
</tr>
</thead>
<tbody>
<tr>
<td>database instance</td>
<td>MongoDB instance</td>
</tr>
<tr>
<td>schema</td>
<td>database</td>
</tr>
<tr>
<td>table</td>
<td>collection</td>
</tr>
<tr>
<td>row</td>
<td>document</td>
</tr>
</tbody>
</table>

Each MongoDB instance has multiple databases, each database can have multiple collections.

Two documents (delimited by [...], used to delimit an array of document).

```plaintext
[ 
  
  "firstname": "Martin",
  "likes": [ "Biking",
             "Photography" ],
  "lastcity": "Boston",
  "lastVisited": 
  ]

, 

[ 
  
  "firstname": "Pramod",
  "citiesvisited": [ "Chicago", "London", "Pune", "Bangalore" ],
  "addresses": [ 
    { "state": "AK",
      "city": "DILLINGHAM"
    }, 
    { "state": "MH",
      "city": "PUNE" } 
  ],
  "lastcity": "Chicago"
  ]

• 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.

“Schema-less” does not mean “organization-less”!

### 7.4.5 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 (for this tutorial, we used the versions at https://repo.mongodb.org/apt/debian/dists/buster/mongodb-org/4.2/main/binary-amd64/mongodb-org-server_4.2.1_amd64.deb and https://repo.mongodb.org/apt/debian/dists/buster/mongodb-org/4.2/main/binary-amd64/mongodb-org-shell_4.2.1_amd64.deb).

- As root, 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
  ```

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” tutorial.

### 7.4.6 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>) (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"\})

  db.books.find(\{"title": "Mother Night"\}, \{"author":1, "quantity":1\}): display only the author and quantity attributes.

  db.books.find(\{"title": "Mother Night"\}, \{"author":0, "quantity":0\}): display all the attributes, except the author and the quantity.

  db.books.find(\{"quantity":\{"$gte": 10, "$lt": 50\}\}): display the entries were the quantity is greater than equal to 10, and less than 50.

Possibility to mimic some features (unique attributes), but no referential key integrity, for instance.

Most insert / update / delete 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.

Not a lot of features, need to write a lot on the program side. But there are tons of API ("package manager" approach), cf. for instance an API over mongo-java-driver: http://jongo.org/ (support some form of prepared statement)

### 7.4.7 MongoDB Database Program

Cf. "MongoDB Driver Quick Start" https://mongodb.github.io/mongo-java-driver/3.4/driver/getting-started/quick-start/#find-all-documents-in-a-collection. We use a slightly outdated version of the driver: more recent example can be found e.g. at https://github.com/mongodb/mongo-java-driver/blob/master/driver-sync/src/examples/tour/QuickTour.java. The code explained below can be found at code/java/MongoTest.java.

Compile and execute with

```
javac -cp .:mongo-java-driver-3.7.0-rc0.jar MongoTest2.java
java -cp .:mongo-java-driver-3.7.0-rc0.jar MongoTest2
```
After various import statement, and the usual header:

```java
MongoClientURI connectionString = new MongoClientURI("mongodb://localhost:27017");
MongoClient mongoClient = new MongoClient(connectionString);
```

Or, more compact:

```java
MongoClient mongoClient = new MongoClient();
```

Get a collection:

```java
MongoDatabase database = mongoClient.getDatabase("mydb");
MongoCollection<Document> collection = database.getCollection("test");
```

Assume we want to create the following document:

```json
{
    "name": "MongoDB",
    "type": "database",
    "count": 1,
    "versions": ["v3.2", "v3.0", "v2.6"],
    "info": { "level": "easy", "used": "yes" }
}
```

(Remember: order does not matter!)

Then we can use the `Document` class, and then insert it:

```java
Document doc = new Document("name", "MongoDB");
doc.append("type", "database");
doc.append("count", 1);
doc.append("versions", Arrays.asList("v3.2", "v3.0", "v2.6"));
doc.append("info", new Document("level", "easy").append("used", "yes"));
```

We can “chain” the append: `doc.append("type", "database").append("count", 1);`

And then insert:

```java
collection.insertOne(doc);
```

Only at this point would the database and collection being created. To make sure everything went right, we can open the command-line-interface, and run:

```
mongo
show dbs
use mydb
show collections
db.collection.find()
```
This last command should return something that begins with `{ "_id" : ObjectId("5ae08a7252cbeb2717712b9f"), "name" : "MongoDB" ... }.

We can construct lists of documents and insert them:

```java
List<Document> documents = new ArrayList<Document>();
for (int i = 0; i < 10; i++) {
    documents.add(new Document("i", i));
}
collection.insertMany(documents);
```

7.5 Principles

- "Schemaless means more responsibility"
- Some denormalization, sometimes: duplicate the information, to have it all in one place. Example: table for phone number, for employee, for emergency contact. You can duplicate that information, no big deal. Less join (resources expensive), but need more storage, more functions, to substitute.
- NoSQL injection: your application should accept only strings from your users (never allow objects by design) and sanitize the inputs before using them (mongo-sanitize is a good module for this).

Exercises

Exercise 7.1 What is polyglot persistence? Is it useful?

Exercise 7.2 What does it mean to be “schemaless”? What does it imply?

Exercise 7.3 What is denormalization? When could that be useful?

Exercise 7.4 What is the (object-relational) impedance mismatch? Is it an issue that cannot be overcome?

Exercise 7.5

For each of the following notion, indicate if they are usually an attribute of NoSQL or of "traditional" SQL:

<table>
<thead>
<tr>
<th></th>
<th>Schema First</th>
<th>Distributed</th>
<th>Relational</th>
<th>Scalable</th>
</tr>
</thead>
<tbody>
<tr>
<td>NoSQL</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SQL</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Solution to Exercises

Solution 7.1 It is the task of picking the right DBMS for the task, and to involve multiple DBMS in a single application. Yes, it is useful. Per Wikipedia16, "Polyglot persistence is the concept

16https://en.wikipedia.org/wiki/Polyglot_persistence
of using different data storage technologies to handle different data storage needs within a
given software application.”.

Solution 7.2 That a table can contain documents, or tuples, with different attributes. It implies
more responsibilities.

Solution 7.3 To duplicate data about other entities in some entities. It is useful when joining is
expensive.

Solution 7.4 Data-base and object-oriented principles are different and it requires work to make
them work together. This correspondance, or matching, can be implemented in the appli-
cation, or lead to the design of new DBMS.

Solution 7.5

<table>
<thead>
<tr>
<th></th>
<th>Schema First</th>
<th>Distributed</th>
<th>Relational</th>
<th>Scalable</th>
</tr>
</thead>
<tbody>
<tr>
<td>NoSQL</td>
<td></td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SQL</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
</tbody>
</table>

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 moti-
vated the “Non SQL” approach, 2. What is the meaning of “Not Only SQL”, 3. The benefits
and limits of the relational approach.

Problem 7.2 (E.R. Diagram from XML File – Customer) Consider the following xml file:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!-- code/xml/Customers.xml -->
<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>
  </Customer>
</Customers>
```
Try to draw the E.R. model that would correspond to the relational implementation of this database. Justify your choices.

**Problem 7.3 (E.R. 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/NSF_Award.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>
  </Award>
</rootTag>
```
<Organization>
  <Code>05010000</Code>
  <Directorate>
    <Abbreviation>CSE</Abbreviation>
    <LongName>Direct For Computer &amp; Info Scie &amp; Engnr</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>

It contains information about one particular award, awarded to an institution on behalf of two researchers. Quoting the National Science Foundation\(^\text{17}\) (NSF):

\(^\text{17}\)https://www.nsf.gov/about/research_areas.jsp
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 a E.-R. 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 7.2 (E.R. Diagram from XML File – Customer) It should be clear that 3 entities are present in this file: Customer, Order, and Product. A product can be part of an order in a certain quantity, and a customer can pass 0 or more orders. Some attributes are naturally good primary keys (they are named “ID”), and some attributes seems to be optional (“Caution”, or “Material”), but should still be given an attribute.

Put together, this gives the following diagram:

We made further assumptions: an order cannot be empty (transcribed by the total constraint on CONTAINS), 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 contains only once a particular product in any quantity, implying that if an order had 2 of a product A, and 3 of the same product A, those two information should be merged in the fact that an order contains 5 of product A. This is enforced by the cardinality ratio of 1 in the CONTAINS relationship.

Of course, other choices were possible.
Solution to Problem 7.3 (E.R. Diagram from XML File – Award) Two entities are easy to distinguish: RESEARCHER (for “Investigator”), 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, 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 in 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 the 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). Hence, 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. Hence, having one binary relationship between the award and the institution, and one binary relationship between the award and the researcher (that furthermore specify 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 institution that can obtain the same award, so it is better not to restrict it. Whenever 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’s probably better to refrain from adding such a relationship.

Most of the attributes are straightforward, once we noticed that “Role” was an attribute of a relationship, and not of an entity.

All together, this gives the following diagram:
References


Sullivan, Dan. 2015. NoSQL for Mere Mortals. Addison-Wesley Professional.