

---

## Master thesis : An Interactive Environment for Relational Algebra Exercises

**Auteur :** Donati, Alexia

**Promoteur(s) :** Debruyne, Christophe

**Faculté :** Faculté des Sciences appliquées

**Diplôme :** Master en sciences informatiques, à finalité spécialisée en "intelligent systems"

**Année académique :** 2025-2026

**URI/URL :** <http://hdl.handle.net/2268.2/26012>

---

### *Avertissement à l'attention des usagers :*

*Tous les documents placés en accès ouvert sur le site le site MatheO sont protégés par le droit d'auteur. Conformément aux principes énoncés par la "Budapest Open Access Initiative"(BOAI, 2002), l'utilisateur du site peut lire, télécharger, copier, transmettre, imprimer, chercher ou faire un lien vers le texte intégral de ces documents, les disséquer pour les indexer, s'en servir de données pour un logiciel, ou s'en servir à toute autre fin légale (ou prévue par la réglementation relative au droit d'auteur). Toute utilisation du document à des fins commerciales est strictement interdite.*

*Par ailleurs, l'utilisateur s'engage à respecter les droits moraux de l'auteur, principalement le droit à l'intégrité de l'oeuvre et le droit de paternité et ce dans toute utilisation que l'utilisateur entreprend. Ainsi, à titre d'exemple, lorsqu'il reproduira un document par extrait ou dans son intégralité, l'utilisateur citera de manière complète les sources telles que mentionnées ci-dessus. Toute utilisation non explicitement autorisée ci-avant (telle que par exemple, la modification du document ou son résumé) nécessite l'autorisation préalable et expresse des auteurs ou de leurs ayants droit.*

---

# An Interactive Environment for Relational Algebra Exercises

---

Alexia Donati

Thesis presented to obtain the degree of:  
**MSc. in Computer Science, professional focus in intelligent systems**

Thesis supervisor:  
Prof. Christophe Debruyne

Academic year: **2025 - 2026**

## **Abstract**

This Master's thesis investigates the conception of an interactive learning environment for relational algebra and extended relational algebra that facilitates autonomous practice and automated assessment, while being in complete accordance with the formalism of the introductory database course of Prof. Christophe Debruyne. Although relational algebra forms a fundamental component of relational database systems, students often encounter difficulties when translating their theoretical understanding into writing correct relational algebra expressions. This is amplified by the fact that existing resources for practising relational algebra frequently omit parts of the theory, adopt semantics that differ from those taught in class, or provide limited support for verifying the correctness of solutions. To address these limitations, the thesis first introduces the underlying theoretical foundations and discusses the decisions that led to the framework adopted for the tool. Then, it conducts a state-of-the-art analysis of existing educational tools that leads to the selection of RelaX as the basis for the tool. Consequently, RelaX is modified to align its syntax, interface, and semantics with the formalism of the course material. In addition, an exercise and verification system is designed and integrated into the tool to provide automated correctness checking and immediate feedback to users. Finally, the perceived usability and pedagogical usefulness of the implemented platform are evaluated through qualitative and quantitative data collected from two practical sessions of the introductory database course. The evaluation indicates that the proposed environment constitutes a promising support tool for relational algebra practice in the database course.

# Acknowledgements

*I would like to express my gratitude to my supervisor, Professor Christophe Debruyne, for his invaluable feedback, encouragement, and continuous guidance throughout this thesis.*

*I would also like to thank the students of the introductory database course who were present during the practical sessions on relational algebra for their patience and understanding.*

*Finally, I would like to thank my family, Isabelle, Olivier, Viviane, Jean, Binini, and Hanini, and my friends, Mai-Phi, Lei, and Saïd, for their support and encouragement.*

*Lastly, I acknowledge the use of tools such as Grammarly and Writefull to refine the grammar and phrasing of my writing, and ChatGPT to receive quick opinions and feedback on sections of this thesis and to generate datasets for relational algebra exercises.*

# Contents

|          |                                                                      |           |
|----------|----------------------------------------------------------------------|-----------|
| <b>1</b> | <b>Introduction</b>                                                  | <b>1</b>  |
| <b>2</b> | <b>Background: the Relational Model and Relational Algebra</b>       | <b>3</b>  |
| 2.1      | Relational Model . . . . .                                           | 3         |
| 2.2      | Relational Algebra . . . . .                                         | 4         |
| 2.2.1    | Projection . . . . .                                                 | 4         |
| 2.2.2    | Selection . . . . .                                                  | 4         |
| 2.2.3    | Rename . . . . .                                                     | 5         |
| 2.2.4    | Union, Intersection, and Difference . . . . .                        | 5         |
| 2.2.5    | Cartesian Product . . . . .                                          | 7         |
| 2.2.6    | Division . . . . .                                                   | 8         |
| 2.2.7    | Join . . . . .                                                       | 8         |
| 2.3      | Extended Relational Algebra . . . . .                                | 10        |
| 2.3.1    | Extended Operators . . . . .                                         | 10        |
| 2.3.2    | Additional Operators . . . . .                                       | 12        |
| 2.3.3    | Missing Information . . . . .                                        | 13        |
| 2.3.4    | Ordering . . . . .                                                   | 14        |
| 2.4      | Subset Used in This Work . . . . .                                   | 15        |
| 2.4.1    | Relational Algebra Used . . . . .                                    | 15        |
| 2.4.2    | Extended Relational Algebra Used . . . . .                           | 16        |
| 2.5      | Query Equivalence in Relational Algebra . . . . .                    | 17        |
| 2.5.1    | Conjunctive Query Equivalence . . . . .                              | 17        |
| 2.5.2    | Full Relational Algebra Equivalence . . . . .                        | 18        |
| 2.6      | Summary . . . . .                                                    | 18        |
| <b>3</b> | <b>Web Environment for (Extended) Relational Algebra Expressions</b> | <b>19</b> |
| 3.1      | Specifications . . . . .                                             | 19        |
| 3.2      | State of the Art . . . . .                                           | 20        |
| 3.2.1    | WinRDBI . . . . .                                                    | 20        |
| 3.2.2    | Relational . . . . .                                                 | 22        |
| 3.2.3    | DBSnap . . . . .                                                     | 22        |
| 3.2.4    | RA (radb) . . . . .                                                  | 22        |
| 3.2.5    | RelaX . . . . .                                                      | 22        |
| 3.2.6    | Scientific Calculator Online . . . . .                               | 23        |
| 3.2.7    | RALI . . . . .                                                       | 23        |
| 3.3      | Why RelaX was Chosen . . . . .                                       | 23        |
| 3.4      | Aligning RelaX Codebase with the Course Material . . . . .           | 24        |
| 3.4.1    | Implementation Overview . . . . .                                    | 24        |
| 3.4.2    | Licences . . . . .                                                   | 24        |
| 3.4.3    | Syntax Modifications . . . . .                                       | 24        |

## TABLE OF CONTENTS

---

|          |                                                    |           |
|----------|----------------------------------------------------|-----------|
| 3.4.4    | View-Level Modification . . . . .                  | 26        |
| 3.4.5    | Model-level Modifications . . . . .                | 27        |
| 3.5      | Summary . . . . .                                  | 37        |
| <b>4</b> | <b>Support for Exercises</b>                       | <b>38</b> |
| 4.1      | State of the Art . . . . .                         | 38        |
| 4.1.1    | AsseSQL . . . . .                                  | 38        |
| 4.1.2    | SQLator . . . . .                                  | 39        |
| 4.1.3    | ACME . . . . .                                     | 39        |
| 4.1.4    | ReliQ . . . . .                                    | 39        |
| 4.1.5    | Neural-network SQL Grading . . . . .               | 40        |
| 4.1.6    | Summary and Identified Limitations . . . . .       | 40        |
| 4.2      | Exercise System Design . . . . .                   | 40        |
| 4.2.1    | Definition of an Exercise . . . . .                | 40        |
| 4.2.2    | Distribution of Exercises . . . . .                | 41        |
| 4.2.3    | Exercise Mode . . . . .                            | 41        |
| 4.2.4    | Verification Process . . . . .                     | 41        |
| 4.2.5    | User Interaction Workflow . . . . .                | 43        |
| 4.2.6    | Exercise Maker . . . . .                           | 44        |
| 4.3      | Exercise System Implementation . . . . .           | 45        |
| 4.3.1    | Exercise Model Implementation . . . . .            | 45        |
| 4.3.2    | Exercise Parsing and Validation . . . . .          | 46        |
| 4.3.3    | Test Battery Parsing and Alignment . . . . .       | 47        |
| 4.3.4    | Exercise Loading Implementation . . . . .          | 48        |
| 4.3.5    | Exercise Mode Implementation . . . . .             | 49        |
| 4.3.6    | Verification Process Implementation . . . . .      | 49        |
| 4.3.7    | User Interface Integration . . . . .               | 53        |
| 4.3.8    | Exercise Maker Implementation . . . . .            | 54        |
| 4.4      | Summary . . . . .                                  | 54        |
| <b>5</b> | <b>Evaluation of the tool</b>                      | <b>55</b> |
| 5.1      | Evaluation Methodology . . . . .                   | 55        |
| 5.2      | Participation . . . . .                            | 55        |
| 5.3      | Perception . . . . .                               | 55        |
| 5.3.1    | Questionnaire Design . . . . .                     | 56        |
| 5.3.2    | Perception Results . . . . .                       | 57        |
| 5.4      | Performance . . . . .                              | 61        |
| 5.5      | Limitations . . . . .                              | 61        |
| <b>6</b> | <b>Conclusions</b>                                 | <b>63</b> |
| 6.1      | Summary . . . . .                                  | 63        |
| 6.2      | Further Improvements . . . . .                     | 63        |
|          | <b>References</b>                                  | <b>67</b> |
| <b>A</b> | <b>Questionnaire</b>                               | <b>68</b> |
| A.1      | English Translation of the Questionnaire . . . . . | 68        |
| A.2      | F-CSUQ Responses . . . . .                         | 69        |
| A.3      | CSUQ Score . . . . .                               | 71        |

# Chapter 1

## Introduction

Relational algebra is a fundamental theoretical pillar of relational database systems, serving as the formal foundation of SQL and query optimisers. However, students often struggle to translate their understanding of the theory into writing syntactically and semantically correct relational algebra expressions. This difficulty is amplified by the fact that the existing online resources students use to practice relational algebra either omit parts of the theory or adopt semantics that differ from those taught in class. As a result, students may practice with an incomplete or inconsistent model of relational algebra, leading to systematic errors that carry over to exams.

Regardless, even if the available resources were in accordance with the course material, they typically offer only a playground-style environment where students can experiment with different relational algebra expressions on provided relation samples. These tools generally do not allow students to verify that their solution is correct. Moreover, even when the output relation of a student's expression seems to match the expected result of the problem that they were trying to solve, this may simply be a coincidence, as these types of exercises are usually tested on small datasets. It is also possible for students to enter expressions that are semantically equivalent to the expected solution while being syntactically different, making it difficult for them to determine the correctness of their own answers without dedicated support.

This thesis addresses these issues by investigating the design and implementation of an interactive platform for relational algebra and extended relational algebra that is fully aligned with the formalism taught in the database course of Prof. Christophe Debruyne. The goal of this platform is to support students in practising relational algebra, receiving feedback, and verifying the correctness of their solutions.

The platform developed as part of this thesis is publicly available at <https://alexiadonati.github.io/relax-modified/> and its codebase can be found at the following Uliège GitLab repository: <https://gitlab.uliege.be/Alexia.Donati/relax-modified> or the following GitHub repository: <https://github.com/AlexiaDonati/relax-modified>.

Therefore, the central research question of this thesis is: How can an interactive platform for relational algebra and extended relational algebra be designed and implemented to support autonomous practice, automated assessment, and relational algebra learning for the introductory database course?

More specifically, the thesis examines the following research sub-questions:

- How can students author relational algebra queries in an interactive manner that is in complete accordance with the course material?
- How can the solution of relational algebra homework and tests be automatically assessed in a way that minimises the risk of false positive evaluation?
- How can such a tool affect student perception, engagement, and learning experience?

To answer these questions, the thesis begins by introducing the underlying theoretical foundations, including the relational model, relational algebra, extended relational algebra, and the problem of query equivalence, while also discussing the decisions that led to the specific frameworks adopted for the tool.

It then presents the design and implementation of the relational algebra learning environment. After defining the requirements the environment must satisfy, the thesis conducts a state-of-the-art analysis of existing relational algebra tools. Based on this analysis, RelaX was selected as the foundation for the implementation. The thesis subsequently describes the modifications introduced to the RelaX codebase to align its syntax, interface, and semantics with the formalism established previously.

The next part discusses the conception of the integrated exercise and verification system. It first examines the grading and feedback mechanisms used in existing automated relational algebra and SQL verification tools, before detailing the design and implementation of the exercise and verification system integrated into the learning environment defined previously.

Finally, the thesis performs a formative evaluation of the usability and pedagogical usefulness of the tool as perceived by students by collecting both quantitative and qualitative data through two practical exercise sessions of the introductory database course of Prof. Debruyne.

The thesis concludes with a summary of the work carried out and outlines potential improvements and directions for future research and development.

## Chapter 2

# Background: the Relational Model and Relational Algebra

This chapter introduces the theoretical foundations on which the remainder of the thesis is based. It begins with a presentation of the relational model. The discussion then turns to relational algebra, reviewing both its origin and the subsequent developments that shaped its modern interpretation. Next, the chapter examines extended relational algebra, its motivation, and the changes it introduces. The specific variants of relational algebra and extended relational algebra adopted in this thesis are then outlined. Finally, the chapter ends by discussing the problem of determining query equivalence.

### 2.1 Relational Model

The *relational model* was first introduced by Edgar F. Codd in his 1970 paper [1] "A Relational Model of Data for Large Shared Data Banks." Codd's principal objective was to define the model as a mathematical framework for data management, based on set theory and first-order logic, while remaining independent of any particular physical representation of data.

In Codd's formulation, a *relation*  $R$  of degree  $n$  is a set of  $n$ -tuples defined over a set of  $n$  typed attributes,  $S_1, S_2, \dots, S_n$ , also called domains. Because relations are sets, they possess some key mathematical properties that distinguish the relational model from earlier data models. For example, relations do not contain duplicate tuples, and the order of those tuples is irrelevant.

Although Codd informally used the term instance in [1], he did not explicitly distinguish between the definition of a relation and its content. The separation between a relation schema, the set of its attribute names and associated domains, and a relation instance, which is the finite set of tuples that it contains, was formalised in subsequent literature. In David Maier's 1983 book [2], "The Theory of Relational Databases,"<sup>1</sup> a *relation scheme*, or schema, is defined as a finite set of attributes with specified domains, and a relation on that scheme is defined as a set of mappings, or tuples, conforming to it.

In [1], Codd distinguishes between domain-ordered and domain-unordered relations. In a *domain-ordered* relation, the position of each attribute in the tuple is used to keep track of its meaning within a relation. As such, the name given to the different attributes of a relation may be non-unique. However, due to the complexity of having to remember the domain ordering of large relations, Codd introduces a *domain-unordered* counterpart, in which relations are invariant under permutations of attribute positions. In a domain-unordered relation, each attribute must be labelled with a uniquely identifiable name, since its position cannot be used to remember its meaning. Subsequent literature, such as Abiteboul et al. [3] "Foundations of Databases," refers to those two interpretations as the *unnamed* and *named perspectives*, respectively.

---

<sup>1</sup>David Maier's 1983 book [2] "The Theory of Relational Databases" is referenced as "[t]he first text on the logical level of database theory" by Serge Abiteboul, Richard Hull, and Victor Vianu [3] in their 1995 book "Foundations of Databases."

Following a meeting with Prof. Debruyne, it was decided that, for pedagogical clarity, only the named perspective would be considered in the course and in the tool developed in this thesis. Most practical query languages, such as SQL, adopt this perspective, making it the one students are most likely to encounter. As a result, it can be assumed that all relations considered in this work have uniquely named attributes and are independent of positional ordering.

## 2.2 Relational Algebra

In Codd's first introduction of the relational model [1], he also introduced a collection of basic algebraic operators. He would refer to this collection as the *relational algebra* in his 1972 follow-up paper [4], "Relational Completeness of Data Base Sublanguages." In the decades that followed, several of these operators were revised by later authors as the field grew larger and practical query languages emerged. This section presents the operators as Codd originally defined them and traces their evolution into the forms now found in contemporary presentations of relational algebra.

### 2.2.1 Projection

In [1], Codd introduces the idea of projection as follows: "*Suppose now we select certain columns of a relation (striking out the others) and then remove from the resulting array any duplication in the rows. The final array represents a relation which is said to be a projection of the given relation.*" He also designates  $\pi$  as the symbol for the corresponding operator. In Maier's formal notation [2], the projection of a relation  $r$  onto a set of attributes  $X$  is defined as " $\pi_X(r)$  is the relation  $r'(X) = \{t(X) \mid t \in r\}$ ," where  $t(X)$  is the restriction of the tuple  $t$  to the set of attributes  $X$ . This definition of the projection operator has remained essentially unchanged in all subsequent writings on relational algebra [3,5,6].

Table 2.1: Results of the Projection operator on a relation  $R$ .

|    |   |   |   |
|----|---|---|---|
| R: | A | B | C |
|    | z | b | 1 |
|    | s | h | 2 |
|    | z | h | 1 |

|                   |   |   |
|-------------------|---|---|
| $\pi_{A, C}(R)$ : | A | C |
|                   | z | 1 |
|                   | s | 2 |
|                   |   |   |

### 2.2.2 Selection

To filter tuples in a relation, Codd first introduced a binary restriction operator in [1] that restricts one relation using another as the filtering criterion. When restricting a relation  $R$  by a relation  $S$ ,  $R$  is reduced to the tuples whose values in some specified attributes match their corresponding values found in  $S$ .

In 1981, the ANSI/X3/SPARC relational model<sup>2</sup> documentation [7] explicitly defines the selection operator as a unary algebraic operator: "*The select operator refers to only one relation. The selection criterion refers only to tuple attributes and constants; it contains comparison operators, such as  $=$ ,  $<$ ,  $>$ ,  $\leq$ ,  $\geq$ ,  $\neq$ , and logical operators, such as *and*, *or*, *not*.*"

By the time of Codd's 1982 Turing Award lecture [8], he also began recasting selection as a unary operator, reflecting the common need to filter one relation by a condition. He wrote: "*The SELECT operator (also called RESTRICT) of the relational algebra takes one relation (table) as an operand and produces a new relation (table) consisting of selected tuples (rows) of the first.*" This shift simplified the algebra and made queries easier to express, while retaining full expressive power since Codd's original restriction operator remains expressible using join and projection. As will be discussed later, the restriction operator is also now referred to as the semi-join.

---

<sup>2</sup>The ANSI/X3/SPARC relational model [7] is a project to develop a relational standard that was first proposed to ANSI/X3 in 1979 by the Relational Task Group (RTG).

During the same period, textbooks and papers [2, 5] adopted the symbol  $\sigma$  to denote the unary selection operator, which filters the tuples of a relation according to a condition expressed directly on its attributes [9]. For example, Maier [2] defines unary selection with the equality comparator using mapping notation as " $\sigma_{A=a}(r)$  ('select  $A$  equal to  $a$  on  $r$ ') is the relation  $r'(R) = \{t \in r \mid t(A) = a\}$ ," where  $r$  is a relation of scheme  $R$ ,  $A$  is an attribute in  $R$ , and  $a$  is an element of the domain of  $A$ .

Table 2.2: Results of the Selection operator on a relation  $R$ .

| $R$ : | A | B | C | $\sigma_{B=h}(R)$ : | A | B | C |
|-------|---|---|---|---------------------|---|---|---|
|       | z | b | 1 |                     | s | h | 2 |
|       | s | h | 2 |                     | z | h | 1 |
|       | z | h | 1 |                     |   |   |   |

### 2.2.3 Rename

Considering that, in [1], Codd formulates the relational algebra in the unnamed perspective, it does not include a rename operator. However, as relational systems evolved toward the named perspective, the need for explicit attribute renaming became apparent.

The rename operator first emerged in the work of Patrick Hall, Peter Hitchcock, and Stephen Todd [10] in their 1975 ISBL paper "An algebra of relations for machine computation," which extends Codd's relational algebra. They introduce the generalised project operator, defined as a function that maps attributes of the input relation to attributes of the output relation. This operator can duplicate or reorder attributes, but more importantly, as they note, "*[i]t can be used to. . . rename components that are to be used in a join. It is informally called rename.*"

Independently, the need for a rename operator was also recognised in early work on relational DBMS standards. For example, the ANSI/X3/SPARC relational model documentation [7] formally defines rename as a primitive operator that takes a relation and a bijection on its attribute names, producing an identical relation with those attributes renamed accordingly.

Subsequent literature [3] generalises this definition by allowing the renaming function to be specified only on a subset of attributes, removing the redundancy of having to specify the names of attributes that keep their original names.

Table 2.3: Results of the Rename operator on a relation  $R$ .

| $R$ : | A | B | C | $\rho_{A \leftarrow D, C \leftarrow A}(R)$ : | D | B | A |
|-------|---|---|---|----------------------------------------------|---|---|---|
|       | z | b | 1 |                                              | z | b | 1 |
|       | s | h | 2 |                                              | s | h | 2 |
|       | z | h | 1 |                                              | z | h | 1 |

### 2.2.4 Union, Intersection, and Difference

In [1], Codd only briefly specified that "*Since relations are sets, all of the usual set operations are applicable to them,*" but he did not give further details on the semantics of those set operations.

In his follow-up paper [4], Codd introduces the idea of *union-compatibility* of domains as a requirement for union, intersection, and difference operations, although only in the unnamed perspective. Codd defines *union-compatibility* as requiring that the domains of corresponding attribute positions match. In his words, "*[t]wo compound domains  $D$ ,  $E$  are union-compatible if they are of the same degree (say  $m$ ) and for every  $j$  ( $j = 1, 2, \dots, m$ ) the  $j^{\text{th}}$  simple domain of  $D$  is union-compatible with the  $j^{\text{th}}$  simple domain of  $E$ .*"

Many early treatments of set operations follow this definition of *union-compatibility* in terms of two schemes having the same arity and compatible domain at corresponding attribute positions [11]. In his

1988 book "Principles of Database and Knowledge-Base Systems," Jeffrey Ullman [5] even states that *"the attribute names for the operand relations are ignored when taking the union."*

In his 1990 book, "The relational model for database management: version 2," Codd [9] explicitly abandons positional concepts and states that: *"Not only does the relation have a name, but each column has a distinct name."* This shift to the named perspective also introduces changes to the definition of *union-compatibility*. Codd writes: *"Suppose that  $S$  and  $T$  are two relations. Then,  $S$  and  $T$  are union-compatible if they are of the same degree and it is possible to establish at least one mapping between the columns of  $S$  and those of  $T$  that is one-to-one, and with the property that, for every column  $A$  of  $S$  and every column  $B$  of  $T$ , if column  $A$  is mapped onto column  $B$ , then  $A$  and  $B$  draw their values from a common domain."*

This updated definition relaxes strict positional matching by allowing arbitrary one-to-one mappings between columns, but still does not require attribute names to match. As a consequence, Codd notes that: *"Of course, the number of such mappings between  $S$  and  $T$  may be zero, one, two, or more,"* which was not the case in the original unnamed version. Since multiple mappings can exist between two relations, a mapping strategy must be chosen to determine the alignment between attributes. Codd proposes a strategy that *"pairs columns by name alphabetically (lowest alphabetically from one operand with lowest alphabetically from the other, and so on)."*

Hugh Darwen [12] raised concerns he had already formed in the mid-1970s regarding Codd's relational model in his 2015 paper, "Why Are There No Relational DBMSs?" For instance, he observes that *"his definition of the algebra was unclear concerning the attribute names of relations resulting from invocation of certain dyadic operators."* Darwen also recalls a 1978 conversation with an ISBL<sup>3</sup> designer about this same ambiguity: *"Q4. What about union, difference, and intersection? ... Answer: Corresponding attributes must have the same name as well as being based on the same domain."*

This idea that attribute names should match for the *union-compatibility* is formalised by Serge Abiteboul, Richard Hull, and Victor Vianu [3] in their 1995 book "Foundations of Databases." In their named algebra, called SPJRU, the union operator *"can only be applied to expressions having the same sort."* In this context, a sort refers to a finite set of attributes. Thus, in SPJRU, union is defined only for relations whose schemas have the same set of attribute names and domains.

Ramez Elmasri and Shamkant B. Navathe [14] give a clear definition of the semantics of the three basic relational set operations. They write: *"We can define the three operations UNION, INTERSECTION, and SET DIFFERENCE on two union-compatible relations  $R$  and  $S$  as follows:*

- *UNION: The result of this operation, denoted by  $R \cup S$ , is a relation that includes all tuples that are either in  $R$  or in  $S$  or in both  $R$  and  $S$ . Duplicate tuples are eliminated.*
- *INTERSECTION: The result of this operation, denoted by  $R \cap S$ , is a relation that includes all tuples that are in both  $R$  and  $S$ .*
- *SET DIFFERENCE (or MINUS): The result of this operation, denoted by  $R - S$ , is a relation that includes all tuples that are in  $R$  but not in  $S$ ."*

For this thesis, it was decided that the set operators, union, intersection, and difference, would be defined only in cases where the operand relations are *union-compatible*. Moreover, it was decided that, as in SPJRU, two relations would be considered *union-compatible* only when their schemas consist of the same set of attribute names and domains.

---

<sup>3</sup>ISBL (Information Systems Base Language) is a relational algebra language developed for the PRTV (Peterlee Relational Test Vehicle) [13], a high-level database system first prototyped in 1971 that implements Codd's relational model.

Table 2.4: Results of the set operators on two union-compatible relations R and S.

|    |   |   |   |    |   |   |   |              |   |   |   |              |   |   |   |           |   |   |   |
|----|---|---|---|----|---|---|---|--------------|---|---|---|--------------|---|---|---|-----------|---|---|---|
| R: | A | B | C | S: | A | B | C | $R \cup S$ : | A | B | C | $R \cap S$ : | A | B | C | $R - S$ : | A | B | C |
|    | 1 | 2 | a |    | 1 | 2 | a |              | 1 | 2 | a |              | 1 | 2 | a |           | 3 | 4 | b |
|    | 3 | 4 | b |    | 7 | 8 | b |              | 3 | 4 | b |              | 5 | 6 | c |           |   |   |   |
|    | 5 | 6 | c |    | 5 | 6 | c |              | 5 | 6 | c |              | 7 | 8 | b |           |   |   |   |

### 2.2.5 Cartesian Product

Codd did not define the semantics of a Cartesian product operator in [1]. In his follow-up paper [4], however, he distinguishes the set-theoretic Cartesian product and the relational Cartesian product. He writes: "*The usual Cartesian product  $R \times S$  of relation  $R$  (degree  $m$ ) with relation  $S$  (degree  $n$ ) is a relation of degree  $m+n$ ,  $R \times S = \{(r, s) : r \in R \wedge s \in S\}$ . The Cartesian product employed in the relational algebra, however, yields an expanded product, a relation  $R \otimes S$ , whose degree is  $m+n$ . This product is defined by  $R \otimes S = \{(r \frown s) : r \in R \wedge s \in S\}$ ." Because Codd was still working with domain-ordered relations at the time, the semantics of this concatenation relied entirely on positional alignment. The attributes of the resulting relation were simply those of  $R$  followed by those of  $S$ , without considering attribute names or possible name conflicts.*

When he shifted to the named perspective in his 1990 book [9], Codd kept this definition of the Cartesian product. However, as noted above, it does not prevent name conflicts between the attributes of the two operand relations. Codd acknowledges this problem and, consistent with his treatment of the other named set operations, introduces a renaming technique to be applied whenever such a conflict occurs. In the case where the names of two attributes  $A_i$  and  $B_j$  are identical, he proposes to prefix them with the name of their original relation in the resulting relation, such as  $S.A_i$  and  $T.B_j$ .

Later, Darwen [12] raised further concerns about Codd's treatment of the attribute names in the algebra. As he observes, Codd defined the Cartesian product "*for all pairs of relations  $r_1$  and  $r_2$ . What if some attribute in  $r_1$  had the same name as one in  $r_2$ ? Indeed, what if  $r_1$  and  $r_2$  were the same relation?*" In a fully named perspective, such situations cannot be treated as benign. For this reason, to his question "*What about Cartesian product?*" ISBL designers replied: "*It is defined only in the case where the headings of  $r_1$  and  $r_2$  have no attribute names in common.*"

Much of the later literature on relational algebra does not treat the Cartesian product as a distinct operator in the named perspective [2, 3]. Instead, these treatments consider the natural join as a natural variant of the Cartesian product operator [3], thereby avoiding attribute-name conflicts by construction.

Table 2.5: Results of the Cartesian product operator on two relations R and S.

|    |   |   |    |   |   |   |                |   |   |   |   |   |
|----|---|---|----|---|---|---|----------------|---|---|---|---|---|
| R: | A | B | S: | C | D | E | $R \times S$ : | A | B | C | D | E |
|    | 1 | 2 |    | a | 2 | b |                | 1 | 2 | a | 2 | b |
|    | 3 | 4 |    | h | 0 | n |                | 3 | 4 | h | 0 | n |
|    |   |   |    |   |   |   |                | 1 | 2 | h | 0 | n |

For this thesis, the Cartesian product operator was treated as a distinct operator, as it is a building block useful for understanding more complex operators. It was also decided that it would not allow shared attributes and would be defined only when the two operand relations do not share any attribute names.

### 2.2.6 Division

Some formalisations of relational algebra also define a relational division operator as a complement to the Cartesian product [5, 9, 11]. Elmasri et al. [14] define it as: "*the DIVISION operation is applied to two relations  $R(Z) \div S(X)$ , where the attributes of  $S$  are a subset of the attributes of  $R$ ; that is,  $X \subseteq Z$ . Let  $Y$  be the set of attributes of  $R$  that are not attributes of  $S$ ; that is,  $Y = Z - X$  (and hence  $Z = X \cup Y$ ). The result of DIVISION is a relation  $T(Y)$  that includes a tuple  $t$  if tuples  $t_R$  appear in  $R$  with  $t_R[Y] = t$ , and with  $t_R[X] = t_S$  for every tuple  $t_S$  in  $S$ . This means that, for a tuple  $t$  to appear in the result  $T$  of the DIVISION, the values in  $t$  must appear in  $R$  in combination with every tuple in  $S$ .*"

Table 2.6: Results of the Division operator on two relations  $R$  and  $S$ .

|       |   |   |   |
|-------|---|---|---|
| $R$ : | A | B | C |
|       | 1 | 2 | a |
|       | 1 | 2 | b |
|       | 5 | 4 | b |
|       | 3 | 4 | a |
|       | 3 | 7 | c |

|       |   |   |
|-------|---|---|
| $S$ : | A | B |
|       | 1 | 2 |
|       | 3 | 4 |

|              |   |
|--------------|---|
| $R \div S$ : | C |
|              | a |

### 2.2.7 Join

The motivation for introducing join operators first appears in [1], where Codd raises the question: "*Suppose we are given two binary relations, which have some domain in common. Under what circumstances can we combine these relations to form a ternary relation which preserves all of the information in the given relations?*" The family of join operators arises as a way of addressing this problem by specifying how tuples from two relations may be combined based on shared information.

#### $\theta$ -Join

The most general join operator is the theta-join ( $\theta$ -join), introduced by Codd in his 1972 paper on relational algebra [4]. A theta-join combines tuples from two relations according to a comparison condition on their attribute values, provided that the comparison is defined for each of their elements. Formally, it is defined by Raghu Ramakrishnan and Johannes Gehrke [11] as a Cartesian product followed by a selection,  $R \bowtie_{\theta} S = \sigma_{\theta}(R \times S)$ .

In his 1990 book [9], Codd further clarified that the join condition may be any Boolean combination of comparisons between attributes of the operand relations, provided that each comparison is defined for all elements of the comparands. He explicitly lists ten comparison operators that "*may be used in a join*:"

1. EQUALITY    2. INEQUALITY
3. LESS THAN    4. LESS THAN OR EQUAL TO
5. GREATER THAN    6. GREATER THAN OR EQUAL TO
7. GREATEST LESS THAN    8. GREATEST LESS THAN OR EQUAL TO
9. LEAST GREATER THAN    10. LEAST GREATER THAN OR EQUAL TO"

A common specialisation of the theta-join is the equi-join, which restricts the join condition to equality comparisons only. Codd [9] notes that an equi-join "*generates a result in which two of the columns are identical in values, although different in column names.*" Because these duplicate attributes are semantically redundant, many textbook formalisations of the equi-join operator directly include an additional projection operation to remove the redundant attribute as part of the equi-join definition [11].

Table 2.7: Results of the Theta-join of two relations R and S.

|    |   |   |
|----|---|---|
| R: | A | B |
|    | a | 2 |
|    | b | 1 |
|    |   |   |

|    |   |   |   |
|----|---|---|---|
| S: | C | D | E |
|    | z | b | 1 |
|    | c | f | 5 |
|    | b | e | 2 |

|                            |   |   |   |   |   |
|----------------------------|---|---|---|---|---|
| $R \bowtie_{B \geq E} S$ : | A | B | C | D | E |
|                            | a | 2 | z | b | 1 |
|                            | a | 2 | b | e | 2 |
|                            | b | 1 | z | b | 1 |

For this thesis, since the theta-join operator is defined using a Cartesian product, it was decided that the same constraints would be applied. As a consequence, it would not allow shared attributes and would be defined only when the two operand relations do not share any attribute names.

## Natural Join

A natural join is a further specialisation of the equi-join that relies on attribute names rather than explicit comparison conditions. Informally, it identifies attributes with the same name in both operand relations, performs an equi-join on all such attributes, and removes the duplicate columns in the result.

Codd first introduced the natural join on two binary relations in [1] as " $R * S = \{(a, b, c) : R(a, b) \wedge S(b, c)\}$  where  $R(a, b)$  has the value true if  $(a, b)$  is a member of  $R$  and similarly for  $S(b, c)$ ."

Subsequently, a precise formal definition of the natural join of two  $n$ -ary relations was given by Maier in [2]: "Start with relations  $r(R)$  and  $s(S)$ , with  $R \cup S = T$ . The join of  $r$  and  $s$ , written  $r \bowtie s$ , is the relation  $q(T)$  of all tuples  $t$  over  $T$  such that there are tuples  $t_r \in r$  and  $t_s \in s$  with  $t_r = t(R)$  and  $t_s = t(S)$ . Since  $R \cap S$  is a subset of both  $R$  and  $S$ , as a consequence of the definition  $t_r(R \cap S) = t_s(R \cap S)$ . Thus, every tuple in  $q$  is a combination of a tuple from  $r$  and a tuple from  $s$  with equal  $(R \cap S)$ -values."

In practice, the natural join is seen as the default join, and the term "join" used alone refers to it [2,8].

Table 2.8: Results of the Natural join of two relations R and S.

|    |   |   |    |   |   |   |                 |   |   |   |   |
|----|---|---|----|---|---|---|-----------------|---|---|---|---|
| R: | A | B | S: | A | C | D | $R \bowtie S$ : | A | B | C | D |
|    | a | 2 |    | b | 1 | d |                 | a | 2 | 2 | b |
|    | b | 4 |    | c | 3 | f |                 | b | 4 | 1 | d |
|    | d | 5 |    | a | 2 | b |                 |   |   |   |   |

## Semi-Join

As noted previously, in [1], Codd introduced a restriction operation that could filter one relation based on the tuples of another. Subsequent textbooks substituted this operator with the semi-join operator, which achieves the same filtering effect [3,15].

Unlike a natural join, the semi-join does not combine tuples from both operands into a larger tuple. Instead, it returns only the tuples of the left operand that successfully join with the right operand [15]. In other words, a semi-join is equivalent to taking the natural join and projecting its result onto the attributes of the left relation. Abiteboul et al. [3] give a formal definition: "Given instances  $I, J$  over  $R, S$ , the semi-join of  $I$  and  $J$  is  $I \ltimes J = \pi_R(I \bowtie J)$ ."

Table 2.9: Results of the Semi-join operator on two relations R and S.

|    |   |   |
|----|---|---|
| R: | A | B |
|    | a | 2 |
|    | b | 4 |
|    | d | 5 |

|    |   |   |   |
|----|---|---|---|
| S: | A | C | D |
|    | b | 1 | d |
|    | c | 3 | f |
|    | a | 2 | b |

|               |   |   |
|---------------|---|---|
| $R \times S:$ | A | B |
|               | a | 2 |
|               | b | 4 |

## Anti-Join

The anti-semijoin, often called anti-join and commonly written using the symbol  $\triangleright$ , selects tuples from one relation that do not join with tuples from another. While the semi-join retains tuples of the left relation that have at least one matching tuple in the right relation, the anti-join does the opposite and retains exactly the tuples that do not participate in any successful join [6].

Garcia-Molina, Ullman, and Widom [15] provide a description: "The antisemijoin  $R \bar{\bowtie} S$  is the set of tuples  $t$  in  $R$  that do not agree with any tuple of  $S$  in the attributes common to  $R$  and  $S$ ." For two relations  $R$  and  $S$ , the anti-join can be rewritten in terms of the semi-join as  $R \bar{\bowtie} S = R - (R \bowtie S)$ .

Table 2.10: Results of the Anti-join operator on two relations  $R$  and  $S$ .

|       | <table><tr><th>A</th><th>B</th></tr><tr><td>a</td><td>2</td></tr><tr><td>b</td><td>4</td></tr><tr><td>d</td><td>5</td></tr></table> | A     | B | a | 2 | b | 4 | d | 5 |  | <table><tr><th>A</th><th>C</th><th>D</th></tr><tr><td>b</td><td>1</td><td>d</td></tr><tr><td>c</td><td>3</td><td>f</td></tr><tr><td>a</td><td>2</td><td>b</td></tr></table> | A | C | D | b | 1 | d | c | 3 | f | a | 2 | b |  | $R \triangleright S$ : <table><tr><th>A</th><th>B</th></tr><tr><td>d</td><td>5</td></tr></table> | A | B | d | 5 |
|-------|-------------------------------------------------------------------------------------------------------------------------------------|-------|---|---|---|---|---|---|---|--|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---|---|---|---|---|---|---|---|---|---|---|---|--|--------------------------------------------------------------------------------------------------|---|---|---|---|
| A     | B                                                                                                                                   |       |   |   |   |   |   |   |   |  |                                                                                                                                                                             |   |   |   |   |   |   |   |   |   |   |   |   |  |                                                                                                  |   |   |   |   |
| a     | 2                                                                                                                                   |       |   |   |   |   |   |   |   |  |                                                                                                                                                                             |   |   |   |   |   |   |   |   |   |   |   |   |  |                                                                                                  |   |   |   |   |
| b     | 4                                                                                                                                   |       |   |   |   |   |   |   |   |  |                                                                                                                                                                             |   |   |   |   |   |   |   |   |   |   |   |   |  |                                                                                                  |   |   |   |   |
| d     | 5                                                                                                                                   |       |   |   |   |   |   |   |   |  |                                                                                                                                                                             |   |   |   |   |   |   |   |   |   |   |   |   |  |                                                                                                  |   |   |   |   |
| A     | C                                                                                                                                   | D     |   |   |   |   |   |   |   |  |                                                                                                                                                                             |   |   |   |   |   |   |   |   |   |   |   |   |  |                                                                                                  |   |   |   |   |
| b     | 1                                                                                                                                   | d     |   |   |   |   |   |   |   |  |                                                                                                                                                                             |   |   |   |   |   |   |   |   |   |   |   |   |  |                                                                                                  |   |   |   |   |
| c     | 3                                                                                                                                   | f     |   |   |   |   |   |   |   |  |                                                                                                                                                                             |   |   |   |   |   |   |   |   |   |   |   |   |  |                                                                                                  |   |   |   |   |
| a     | 2                                                                                                                                   | b     |   |   |   |   |   |   |   |  |                                                                                                                                                                             |   |   |   |   |   |   |   |   |   |   |   |   |  |                                                                                                  |   |   |   |   |
| A     | B                                                                                                                                   |       |   |   |   |   |   |   |   |  |                                                                                                                                                                             |   |   |   |   |   |   |   |   |   |   |   |   |  |                                                                                                  |   |   |   |   |
| d     | 5                                                                                                                                   |       |   |   |   |   |   |   |   |  |                                                                                                                                                                             |   |   |   |   |   |   |   |   |   |   |   |   |  |                                                                                                  |   |   |   |   |
| $R$ : |                                                                                                                                     | $S$ : |   |   |   |   |   |   |   |  |                                                                                                                                                                             |   |   |   |   |   |   |   |   |   |   |   |   |  |                                                                                                  |   |   |   |   |

## 2.3 Extended Relational Algebra

The relational algebra detailed previously is based on set semantics, where relations are treated as sets of tuples and duplicate tuples are not allowed. Although this model provides a rigorous mathematical foundation, it differs from the behaviour of most practical database systems, which instead rely on multi-set, or bag, semantics, in which duplicate tuples are permitted [16]. This is due both to functional requirements and to the high cost of eliminating duplicates in database operations [17]. In particular, SQL, the standard query language for relational database systems, adopts multi-set semantics by default and preserves duplicates unless they are explicitly removed [18].

In *extended relational algebra*, also called multi-set or bag relational algebra, relations are generalised from sets to multi-sets, meaning that tuples may appear multiple times and are associated with a multiplicity. The following section presents the extended relational algebra by redefining the core operators of the standard relational algebra to incorporate multi-set semantics and introducing additional operators.

### 2.3.1 Extended Operators

In 1982, Umeshwar Dayal, Nathan Goodman, and Randy H. Katz [19] extended the standard set relational algebra to multi-set relations in their paper "An extended relational algebra with control over duplicate elimination." Their work introduces modified definitions of the standard set relational algebra operators to account for tuple multiplicities. In this extended model, a relation  $R$  is mathematically defined as a set of pairs  $\langle t, i \rangle$ , where  $t$  is a tuple and  $i > 0$  denotes the number of occurrences of  $t$  in  $R$ .

The selection operator extends naturally to multi-sets. For a relation  $R$  with schema  $\underline{R}$ , an attribute  $A \in \underline{R}$ , and a value  $a \in \text{dom}(A)$ , it is defined for the equality comparator as: " $\sigma_{A=a} R = \{ \langle t, i \rangle : \langle t, i \rangle \in R \wedge t(A) = a \}$ ." Thus, the tuples that satisfy the selection condition are kept, along with their initial multiplicities.

Table 2.11: Results of the Selection operator on a multi-set relation  $R$ .

|    |   |   |   |
|----|---|---|---|
|    | A | B | C |
| R: | z | b | 1 |
|    | s | h | 2 |
|    | s | h | 2 |
|    | z | h | 1 |

|                     |   |   |   |
|---------------------|---|---|---|
|                     | A | B | C |
| $\sigma_{B=h}(R)$ : | s | h | 2 |
|                     | s | h | 2 |
|                     | z | h | 1 |

Meanwhile, extending the projection operator must account for duplicate tuples introduced when reducing the set of attributes. For a set of attributes  $X \subseteq \underline{R}$ , it is defined as: " $\pi_X(R) = \{\langle x, k \rangle : x \text{ is a map on } X \text{ and } k = \sum i \text{ for } \langle t, i \rangle \in R \text{ such that } t \text{ agrees with } x \text{ on } X\}$ ." In other words, the projection operator groups tuples by their projection on  $X$  and sums their multiplicities.

Table 2.12: Results of the Projection operator on a relation R.

|    |   |   |   |                   |   |   |
|----|---|---|---|-------------------|---|---|
| R: | A | B | C | $\pi_{A, C}(R)$ : | A | C |
|    | z | a | 1 |                   | z | 1 |
|    | z | b | 1 |                   | z | 1 |
|    | h | b | 2 |                   | h | 2 |

For binary operators, the multi-set versions of union, intersection, and difference are defined so that they reduce to their set-based definition. In the following definitions, the multiplicity of a tuple not occurring in a multi-set relation is taken to be 0. For two union-compatible multi-set relations  $R$  and  $S$ , the operators are defined in [19] as follows:

- $R \cup S = \{\langle t, k \rangle : \exists i \exists j (\langle t, i \rangle \in R \wedge \langle t, j \rangle \in S \wedge k = \max(i, j) \wedge k \neq 0)\}$
- $R \cap S = \{\langle t, k \rangle : \exists i \exists j (\langle t, i \rangle \in R \wedge \langle t, j \rangle \in S \wedge k = \min(i, j) \wedge k \neq 0)\}$
- $R - S = \{\langle t, k \rangle : \exists i \exists j (\langle t, i \rangle \in R \wedge \langle t, j \rangle \in S \wedge k = i \ominus j \wedge k \neq 0)\}$

where  $i \ominus j$  is equal to  $i - j$  if  $i > j$ , or 0 otherwise. These definitions ensure that the resulting relations remain valid multi-sets by excluding tuples with zero multiplicity.

Table 2.13: Results of the multi-set operators on two union-compatible multi-set relations R and S.

|    |   |   |   |    |   |   |   |
|----|---|---|---|----|---|---|---|
| R: | A | B | C | S: | A | B | C |
|    | 1 | 2 | a |    | 1 | 2 | a |
|    | 3 | 4 | b |    | 7 | 8 | b |
|    | 5 | 6 | c |    | 5 | 6 | c |
|    | 5 | 6 | c |    |   |   |   |

|              |   |   |   |              |   |   |   |           |   |   |   |
|--------------|---|---|---|--------------|---|---|---|-----------|---|---|---|
| $R \cup S$ : | A | B | C | $R \cap S$ : | A | B | C | $R - S$ : | A | B | C |
|              | 1 | 2 | a |              | 1 | 2 | a |           | 3 | 4 | b |
|              | 3 | 4 | b |              |   |   |   |           | 5 | 6 | c |
|              | 5 | 6 | c |              | 5 | 6 | c |           |   |   |   |
|              | 5 | 6 | c |              |   |   |   |           |   |   |   |
|              | 7 | 8 | b |              |   |   |   |           |   |   |   |

Finally, for two multi-set relations  $R_1$  and  $R_2$  with schema  $\underline{R_1}$  and  $\underline{R_2}$  respectively, the natural join operator is defined as: " $R_1 \bowtie R_2 = \{\langle t, i \rangle : t \text{ is a map on } \underline{R_1} \cup \underline{R_2} \text{ and } (\exists \langle t_1, i_1 \rangle \in R_1)(\exists \langle t_2, i_2 \rangle \in R_2) (t \text{ agrees with } t_1 \text{ on } \underline{R_1} \text{ and } t \text{ agrees with } t_2 \text{ on } \underline{R_2} \text{ and } i = i_1 \times i_2)\}$ ." In other words, it combines the tuples of both relations that agree on their common attributes and computes the multiplicities of the resulting tuples by multiplying the multiplicities of the matching tuples.

In addition, the Cartesian product extends naturally to multi-sets as it can be viewed as a special case of the natural join in which no attribute matching is required. In this case, every tuple of  $R_1$  is combined with every tuple of  $R_2$ , and the multiplicity of each resulting tuple is the product of their multiplicities.

Table 2.14: Results of the Natural join of two multi-set relations R and S.

| R: | A | B |  | S: | A | C | D |  | $R \bowtie S$ : | A | B | C | D |
|----|---|---|--|----|---|---|---|--|-----------------|---|---|---|---|
|    | a | 2 |  |    | b | 1 | d |  |                 | a | 2 | 2 | e |
|    | b | 4 |  |    | c | 3 | f |  |                 | b | 4 | 1 | d |
|    | b | 4 |  |    | a | 2 | e |  |                 | b | 4 | 1 | d |
|    | d | 5 |  |    |   |   |   |  |                 |   |   |   |   |

Table 2.15: Results of the Cartesian product operator on two multi-set relations R and S.

| R: | A | B |  | S: | C | D | E |  | $R \times S$ : | A | B | C | D | E |
|----|---|---|--|----|---|---|---|--|----------------|---|---|---|---|---|
|    | 1 | 2 |  |    | a | 1 | d |  |                | 1 | 2 | a | 1 | d |
|    | 3 | 4 |  |    | b | 2 | e |  |                | 1 | 2 | b | 2 | e |
|    | 1 | 2 |  |    |   |   |   |  |                | 3 | 4 | a | 1 | d |
|    |   |   |  |    |   |   |   |  |                | 3 | 4 | b | 2 | e |
|    |   |   |  |    |   |   |   |  |                | 1 | 2 | a | 1 | d |
|    |   |   |  |    |   |   |   |  |                | 1 | 2 | b | 2 | e |

### 2.3.2 Additional Operators

While the extended relational algebra presented above increases the expressive power of the classical model by introducing multi-set semantics, it still lacks several important features required in practical database applications. As pointed out in 1994 by Paul Grefen and Rolf de By [17] in their paper "A multi-set extended relational algebra: a formal approach to a practical issue," these limitations include the absence of arithmetic expressions on attributes, mechanisms for duplicate elimination, and aggregation operations over multi-sets. As a result, the algebra cannot express a wide range of queries commonly encountered in real-world scenarios. To address these shortcomings, the relational algebra is further extended with additional operators that enhance the expressive power of the language by introducing mechanisms for removing duplicate tuples, computing aggregate values, and supporting arithmetic operations.

#### Duplicate Elimination

First, the duplicate elimination operator, denoted by  $\Delta$  or  $\delta$ , is introduced to transform a multi-set into a set by removing duplicate tuples. For a multi-set relation  $R$ , it is formally defined in [19] as: " $\delta(R) = \{ \langle t, 1 \rangle : (\exists i)(\langle t, i \rangle \in R) \}$ ." It ensures that each tuple appears exactly once in the relation.

Table 2.16: Results of the Duplicate elimination operator on a multi-set relation R.

| R: | A | B | C |  | $\Delta(R)$ : | A | B | C |
|----|---|---|---|--|---------------|---|---|---|
|    | z | b | 1 |  |               | z | b | 1 |
|    | h | b | 2 |  |               | h | b | 2 |
|    | z | b | 1 |  |               |   |   |   |

#### Group By

Next, the group by operator, denoted by  $\gamma$  or  $\Gamma$ , is introduced to compute aggregate functions on groups of tuples. For a multi-set relation  $E$ , a list of grouping attributes  $\alpha$ , and an aggregate function  $f$  applied to an attribute  $\beta$ , the operator groups the tuples of  $E$  according to their values in  $\alpha$ , and computes  $f$  for each group. Using the notation introduced in this thesis, the group operator defined in [17] can be rewritten as:  $\gamma_{\alpha, f, \beta} E = \{ \langle x \oplus [f(E_x), \beta], 1 \rangle : x \in \pi_{\alpha}(E) \}$  where  $E_x = \{ y \in E : y(\alpha) = x \}$ . In other words, the operator partitions the relation  $E$  into groups of tuples that share the same values for the attributes in

$\alpha$ . Then, for each group  $E_x$ , a single tuple is produced that contains the grouping attributes  $x$  and the value of the aggregate function  $f$  applied to the corresponding attribute. If the grouping attribute list is empty, the aggregate function is applied to all tuples in the multi-set as a whole. In that case, the result is a single tuple containing the result of the function.

Table 2.17: Results of the Group by operator on a multi-set relation R.

|    | ID_STUDENT | ID_COURSE | GRADE |
|----|------------|-----------|-------|
| R: | 1          | ZB11306   | 15    |
|    | 2          | A2B2507   | 10    |
|    | 2          | A2B2507   | 18    |
|    | 3          | ZB11306   | 17    |

|                                                                 | ID_COURSE | AVG_COURSE |
|-----------------------------------------------------------------|-----------|------------|
| $\gamma_{ID\_COURSE ; AVG(GRADE) \rightarrow AVG\_COURSE}(R)$ : | ZB11306   | 16         |
|                                                                 | A2B2507   | 14         |

### Generalised Projection

Finally, the support for arithmetic operations is introduced through the generalised projection operator. This operator extends the standard projection operator by allowing the projection operand to contain arithmetic expressions defined on the attributes of the relation. The standard projection operator can therefore be seen as a special case of the generalised operator.

Table 2.18: Results of the Generalised projection operator on a multi-set relation R.

|    | STUDENT | COURSE  | GRADE_TH | GRADE_EX |
|----|---------|---------|----------|----------|
| R: | 1       | ZB11306 | 15       | 11       |
|    | 2       | A2B2507 | 10       | 9        |
|    | 2       | A2B2507 | 18       | 17       |
|    | 3       | ZB11306 | 17       | 19       |

|                                                                         | STUDENT | COURSE  | GRADE |
|-------------------------------------------------------------------------|---------|---------|-------|
| $\pi_{STUDENT, COURSE, (GRADE\_TH + GRADE\_EX) \rightarrow GRADE}(R)$ : | 1       | ZB11306 | 26    |
|                                                                         | 2       | A2B2507 | 19    |
|                                                                         | 2       | A2B2507 | 35    |
|                                                                         | 3       | ZB11306 | 36    |

### 2.3.3 Missing Information

Another limitation of the classical relational model comes from its assumed completeness. As pointed out by Abiteboul et al. in [3], "we have assumed that a database always records information that is completely known . . . In reality, we often must deal with incomplete information." In practice, there are several reasons why information might be missing. For example, the value may be unknown or not yet available when the tuple is inserted into the system, or the attribute may not apply to the tuple. Partial information can also occur when the data cannot be fully trusted due to potential inconsistencies, for instance, those introduced by integrating or merging data from multiple sources.

To address this limitation, Codd introduced null values to relations in his 1979 paper [20], "Extending the Database Relational Model to Capture More Meaning." Null values provide a mechanism to represent missing or incomplete information. However, their use also complicates the semantics of the model because

they do not behave like ordinary values in comparisons and logical expressions. In particular, as Codd notes, incorporating them necessitates extending the classical two-valued logic, true or false, to a three-valued logic, where a comparison involving a null value may evaluate to the additional unknown truth value.

In the same work, Codd also introduced new join operators called outer joins, written as  $\bowtie$ ,  $\ltimes$ , and  $\rtimes$  for the full, left, and right outer joins respectively. Unlike the natural join, these operators preserve tuples that do not have matching counterparts in the other relation by generating null values whenever needed.

Table 2.19: Results of the Outer-join operators on two multi-set relations R and S.

|                 | $R$ : | <table><tr><th>A</th><th>B</th></tr><tr><td>a</td><td>2</td></tr><tr><td>d</td><td>5</td></tr><tr><td>d</td><td>5</td></tr></table>                                                                                                                                                                                                  | A    | B | a | 2 | d | 5 | d | 5 | $S$ : | <table><tr><th>A</th><th>C</th><th>D</th></tr><tr><td>a</td><td>2</td><td>b</td></tr><tr><td>b</td><td>1</td><td>d</td></tr><tr><td>a</td><td>2</td><td>b</td></tr></table> | A | C | D | a | 2    | b    | b | 1 | d    | a    | 2    | b |   |   |                 |                                                                                                                                                                                                                                                                                                                                         |   |   |   |   |   |   |   |   |   |   |   |   |   |   |      |      |   |   |      |      |   |   |      |      |                 |                                                                                                                                                                                                                        |   |   |   |   |   |   |   |   |   |   |   |   |      |   |   |   |
|-----------------|-------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------|---|---|---|---|---|---|---|-------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---|---|---|---|------|------|---|---|------|------|------|---|---|---|-----------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---|---|---|---|---|---|---|---|---|---|---|---|---|---|------|------|---|---|------|------|---|---|------|------|-----------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---|---|---|---|---|---|---|---|---|---|---|---|------|---|---|---|
| A               | B     |                                                                                                                                                                                                                                                                                                                                      |      |   |   |   |   |   |   |   |       |                                                                                                                                                                             |   |   |   |   |      |      |   |   |      |      |      |   |   |   |                 |                                                                                                                                                                                                                                                                                                                                         |   |   |   |   |   |   |   |   |   |   |   |   |   |   |      |      |   |   |      |      |   |   |      |      |                 |                                                                                                                                                                                                                        |   |   |   |   |   |   |   |   |   |   |   |   |      |   |   |   |
| a               | 2     |                                                                                                                                                                                                                                                                                                                                      |      |   |   |   |   |   |   |   |       |                                                                                                                                                                             |   |   |   |   |      |      |   |   |      |      |      |   |   |   |                 |                                                                                                                                                                                                                                                                                                                                         |   |   |   |   |   |   |   |   |   |   |   |   |   |   |      |      |   |   |      |      |   |   |      |      |                 |                                                                                                                                                                                                                        |   |   |   |   |   |   |   |   |   |   |   |   |      |   |   |   |
| d               | 5     |                                                                                                                                                                                                                                                                                                                                      |      |   |   |   |   |   |   |   |       |                                                                                                                                                                             |   |   |   |   |      |      |   |   |      |      |      |   |   |   |                 |                                                                                                                                                                                                                                                                                                                                         |   |   |   |   |   |   |   |   |   |   |   |   |   |   |      |      |   |   |      |      |   |   |      |      |                 |                                                                                                                                                                                                                        |   |   |   |   |   |   |   |   |   |   |   |   |      |   |   |   |
| d               | 5     |                                                                                                                                                                                                                                                                                                                                      |      |   |   |   |   |   |   |   |       |                                                                                                                                                                             |   |   |   |   |      |      |   |   |      |      |      |   |   |   |                 |                                                                                                                                                                                                                                                                                                                                         |   |   |   |   |   |   |   |   |   |   |   |   |   |   |      |      |   |   |      |      |   |   |      |      |                 |                                                                                                                                                                                                                        |   |   |   |   |   |   |   |   |   |   |   |   |      |   |   |   |
| A               | C     | D                                                                                                                                                                                                                                                                                                                                    |      |   |   |   |   |   |   |   |       |                                                                                                                                                                             |   |   |   |   |      |      |   |   |      |      |      |   |   |   |                 |                                                                                                                                                                                                                                                                                                                                         |   |   |   |   |   |   |   |   |   |   |   |   |   |   |      |      |   |   |      |      |   |   |      |      |                 |                                                                                                                                                                                                                        |   |   |   |   |   |   |   |   |   |   |   |   |      |   |   |   |
| a               | 2     | b                                                                                                                                                                                                                                                                                                                                    |      |   |   |   |   |   |   |   |       |                                                                                                                                                                             |   |   |   |   |      |      |   |   |      |      |      |   |   |   |                 |                                                                                                                                                                                                                                                                                                                                         |   |   |   |   |   |   |   |   |   |   |   |   |   |   |      |      |   |   |      |      |   |   |      |      |                 |                                                                                                                                                                                                                        |   |   |   |   |   |   |   |   |   |   |   |   |      |   |   |   |
| b               | 1     | d                                                                                                                                                                                                                                                                                                                                    |      |   |   |   |   |   |   |   |       |                                                                                                                                                                             |   |   |   |   |      |      |   |   |      |      |      |   |   |   |                 |                                                                                                                                                                                                                                                                                                                                         |   |   |   |   |   |   |   |   |   |   |   |   |   |   |      |      |   |   |      |      |   |   |      |      |                 |                                                                                                                                                                                                                        |   |   |   |   |   |   |   |   |   |   |   |   |      |   |   |   |
| a               | 2     | b                                                                                                                                                                                                                                                                                                                                    |      |   |   |   |   |   |   |   |       |                                                                                                                                                                             |   |   |   |   |      |      |   |   |      |      |      |   |   |   |                 |                                                                                                                                                                                                                                                                                                                                         |   |   |   |   |   |   |   |   |   |   |   |   |   |   |      |      |   |   |      |      |   |   |      |      |                 |                                                                                                                                                                                                                        |   |   |   |   |   |   |   |   |   |   |   |   |      |   |   |   |
| $R \bowtie S$ : |       | <table><tr><th>A</th><th>B</th><th>C</th><th>D</th></tr><tr><td>a</td><td>2</td><td>2</td><td>b</td></tr><tr><td>a</td><td>2</td><td>2</td><td>b</td></tr><tr><td>d</td><td>5</td><td>null</td><td>null</td></tr><tr><td>d</td><td>5</td><td>null</td><td>null</td></tr><tr><td>null</td><td>b</td><td>1</td><td>d</td></tr></table> | A    | B | C | D | a | 2 | 2 | b | a     | 2                                                                                                                                                                           | 2 | b | d | 5 | null | null | d | 5 | null | null | null | b | 1 | d | $R \bowtie S$ : | <table><tr><th>A</th><th>B</th><th>C</th><th>D</th></tr><tr><td>a</td><td>2</td><td>2</td><td>b</td></tr><tr><td>a</td><td>2</td><td>2</td><td>b</td></tr><tr><td>d</td><td>5</td><td>null</td><td>null</td></tr><tr><td>d</td><td>5</td><td>null</td><td>null</td></tr><tr><td>d</td><td>5</td><td>null</td><td>null</td></tr></table> | A | B | C | D | a | 2 | 2 | b | a | 2 | 2 | b | d | 5 | null | null | d | 5 | null | null | d | 5 | null | null | $R \bowtie S$ : | <table><tr><th>A</th><th>B</th><th>C</th><th>D</th></tr><tr><td>a</td><td>2</td><td>2</td><td>b</td></tr><tr><td>a</td><td>2</td><td>2</td><td>b</td></tr><tr><td>null</td><td>b</td><td>1</td><td>d</td></tr></table> | A | B | C | D | a | 2 | 2 | b | a | 2 | 2 | b | null | b | 1 | d |
| A               | B     | C                                                                                                                                                                                                                                                                                                                                    | D    |   |   |   |   |   |   |   |       |                                                                                                                                                                             |   |   |   |   |      |      |   |   |      |      |      |   |   |   |                 |                                                                                                                                                                                                                                                                                                                                         |   |   |   |   |   |   |   |   |   |   |   |   |   |   |      |      |   |   |      |      |   |   |      |      |                 |                                                                                                                                                                                                                        |   |   |   |   |   |   |   |   |   |   |   |   |      |   |   |   |
| a               | 2     | 2                                                                                                                                                                                                                                                                                                                                    | b    |   |   |   |   |   |   |   |       |                                                                                                                                                                             |   |   |   |   |      |      |   |   |      |      |      |   |   |   |                 |                                                                                                                                                                                                                                                                                                                                         |   |   |   |   |   |   |   |   |   |   |   |   |   |   |      |      |   |   |      |      |   |   |      |      |                 |                                                                                                                                                                                                                        |   |   |   |   |   |   |   |   |   |   |   |   |      |   |   |   |
| a               | 2     | 2                                                                                                                                                                                                                                                                                                                                    | b    |   |   |   |   |   |   |   |       |                                                                                                                                                                             |   |   |   |   |      |      |   |   |      |      |      |   |   |   |                 |                                                                                                                                                                                                                                                                                                                                         |   |   |   |   |   |   |   |   |   |   |   |   |   |   |      |      |   |   |      |      |   |   |      |      |                 |                                                                                                                                                                                                                        |   |   |   |   |   |   |   |   |   |   |   |   |      |   |   |   |
| d               | 5     | null                                                                                                                                                                                                                                                                                                                                 | null |   |   |   |   |   |   |   |       |                                                                                                                                                                             |   |   |   |   |      |      |   |   |      |      |      |   |   |   |                 |                                                                                                                                                                                                                                                                                                                                         |   |   |   |   |   |   |   |   |   |   |   |   |   |   |      |      |   |   |      |      |   |   |      |      |                 |                                                                                                                                                                                                                        |   |   |   |   |   |   |   |   |   |   |   |   |      |   |   |   |
| d               | 5     | null                                                                                                                                                                                                                                                                                                                                 | null |   |   |   |   |   |   |   |       |                                                                                                                                                                             |   |   |   |   |      |      |   |   |      |      |      |   |   |   |                 |                                                                                                                                                                                                                                                                                                                                         |   |   |   |   |   |   |   |   |   |   |   |   |   |   |      |      |   |   |      |      |   |   |      |      |                 |                                                                                                                                                                                                                        |   |   |   |   |   |   |   |   |   |   |   |   |      |   |   |   |
| null            | b     | 1                                                                                                                                                                                                                                                                                                                                    | d    |   |   |   |   |   |   |   |       |                                                                                                                                                                             |   |   |   |   |      |      |   |   |      |      |      |   |   |   |                 |                                                                                                                                                                                                                                                                                                                                         |   |   |   |   |   |   |   |   |   |   |   |   |   |   |      |      |   |   |      |      |   |   |      |      |                 |                                                                                                                                                                                                                        |   |   |   |   |   |   |   |   |   |   |   |   |      |   |   |   |
| A               | B     | C                                                                                                                                                                                                                                                                                                                                    | D    |   |   |   |   |   |   |   |       |                                                                                                                                                                             |   |   |   |   |      |      |   |   |      |      |      |   |   |   |                 |                                                                                                                                                                                                                                                                                                                                         |   |   |   |   |   |   |   |   |   |   |   |   |   |   |      |      |   |   |      |      |   |   |      |      |                 |                                                                                                                                                                                                                        |   |   |   |   |   |   |   |   |   |   |   |   |      |   |   |   |
| a               | 2     | 2                                                                                                                                                                                                                                                                                                                                    | b    |   |   |   |   |   |   |   |       |                                                                                                                                                                             |   |   |   |   |      |      |   |   |      |      |      |   |   |   |                 |                                                                                                                                                                                                                                                                                                                                         |   |   |   |   |   |   |   |   |   |   |   |   |   |   |      |      |   |   |      |      |   |   |      |      |                 |                                                                                                                                                                                                                        |   |   |   |   |   |   |   |   |   |   |   |   |      |   |   |   |
| a               | 2     | 2                                                                                                                                                                                                                                                                                                                                    | b    |   |   |   |   |   |   |   |       |                                                                                                                                                                             |   |   |   |   |      |      |   |   |      |      |      |   |   |   |                 |                                                                                                                                                                                                                                                                                                                                         |   |   |   |   |   |   |   |   |   |   |   |   |   |   |      |      |   |   |      |      |   |   |      |      |                 |                                                                                                                                                                                                                        |   |   |   |   |   |   |   |   |   |   |   |   |      |   |   |   |
| d               | 5     | null                                                                                                                                                                                                                                                                                                                                 | null |   |   |   |   |   |   |   |       |                                                                                                                                                                             |   |   |   |   |      |      |   |   |      |      |      |   |   |   |                 |                                                                                                                                                                                                                                                                                                                                         |   |   |   |   |   |   |   |   |   |   |   |   |   |   |      |      |   |   |      |      |   |   |      |      |                 |                                                                                                                                                                                                                        |   |   |   |   |   |   |   |   |   |   |   |   |      |   |   |   |
| d               | 5     | null                                                                                                                                                                                                                                                                                                                                 | null |   |   |   |   |   |   |   |       |                                                                                                                                                                             |   |   |   |   |      |      |   |   |      |      |      |   |   |   |                 |                                                                                                                                                                                                                                                                                                                                         |   |   |   |   |   |   |   |   |   |   |   |   |   |   |      |      |   |   |      |      |   |   |      |      |                 |                                                                                                                                                                                                                        |   |   |   |   |   |   |   |   |   |   |   |   |      |   |   |   |
| d               | 5     | null                                                                                                                                                                                                                                                                                                                                 | null |   |   |   |   |   |   |   |       |                                                                                                                                                                             |   |   |   |   |      |      |   |   |      |      |      |   |   |   |                 |                                                                                                                                                                                                                                                                                                                                         |   |   |   |   |   |   |   |   |   |   |   |   |   |   |      |      |   |   |      |      |   |   |      |      |                 |                                                                                                                                                                                                                        |   |   |   |   |   |   |   |   |   |   |   |   |      |   |   |   |
| A               | B     | C                                                                                                                                                                                                                                                                                                                                    | D    |   |   |   |   |   |   |   |       |                                                                                                                                                                             |   |   |   |   |      |      |   |   |      |      |      |   |   |   |                 |                                                                                                                                                                                                                                                                                                                                         |   |   |   |   |   |   |   |   |   |   |   |   |   |   |      |      |   |   |      |      |   |   |      |      |                 |                                                                                                                                                                                                                        |   |   |   |   |   |   |   |   |   |   |   |   |      |   |   |   |
| a               | 2     | 2                                                                                                                                                                                                                                                                                                                                    | b    |   |   |   |   |   |   |   |       |                                                                                                                                                                             |   |   |   |   |      |      |   |   |      |      |      |   |   |   |                 |                                                                                                                                                                                                                                                                                                                                         |   |   |   |   |   |   |   |   |   |   |   |   |   |   |      |      |   |   |      |      |   |   |      |      |                 |                                                                                                                                                                                                                        |   |   |   |   |   |   |   |   |   |   |   |   |      |   |   |   |
| a               | 2     | 2                                                                                                                                                                                                                                                                                                                                    | b    |   |   |   |   |   |   |   |       |                                                                                                                                                                             |   |   |   |   |      |      |   |   |      |      |      |   |   |   |                 |                                                                                                                                                                                                                                                                                                                                         |   |   |   |   |   |   |   |   |   |   |   |   |   |   |      |      |   |   |      |      |   |   |      |      |                 |                                                                                                                                                                                                                        |   |   |   |   |   |   |   |   |   |   |   |   |      |   |   |   |
| null            | b     | 1                                                                                                                                                                                                                                                                                                                                    | d    |   |   |   |   |   |   |   |       |                                                                                                                                                                             |   |   |   |   |      |      |   |   |      |      |      |   |   |   |                 |                                                                                                                                                                                                                                                                                                                                         |   |   |   |   |   |   |   |   |   |   |   |   |   |   |      |      |   |   |      |      |   |   |      |      |                 |                                                                                                                                                                                                                        |   |   |   |   |   |   |   |   |   |   |   |   |      |   |   |   |

### 2.3.4 Ordering

Lastly, since relational algebra is defined on sets or multi-sets of tuples, relations are, by definition, unordered. If the formalism of a relation is kept unchanged, sort operators cannot be expressed [17]. However, in practice, it is common for a database system user to want the output of a query to be listed in some particular order. To formally allow an operator which sorts a relation, Giedrius Slivinskasy, Christian S. Jenseny, and Richard T. Snodgrass [21] define the extended relational algebra detailed in their 2002 paper, "Bringing Order to Query Optimization," on relations that are lists of tuples. Given a relation  $R$  with schema  $S = \{A_1, \dots, A_n\}$  and an order  $\langle (A_i : \text{ASC or DESC}), \dots, (A_k : \text{ASC or DESC}) \rangle$ , the sort operator  $\tau$  produces a list of tuples of  $R$ , lexicographically sorted by  $(A_i, \dots, A_k)$ .

Table 2.20: Results of the Sort operator on a multi-set relation R.

|    |   |   |   |   |                             |   |   |   |   |
|----|---|---|---|---|-----------------------------|---|---|---|---|
| R: | A | B | C | D | $\tau_{D \text{ ASC}}(R)$ : | A | B | C | D |
|    | 3 | h | b | 2 |                             | 2 | z | h | 1 |
|    | 1 | k | j | 8 |                             | 3 | h | b | 2 |
|    | 2 | z | h | 1 |                             | 4 | s | m | 3 |
|    | 4 | s | m | 3 |                             | 5 | s | q | 4 |
|    | 5 | k | t | 6 |                             | 8 | p | g | 5 |
|    | 5 | s | q | 4 |                             | 5 | k | t | 6 |
|    | 6 | k | g | 7 |                             | 6 | k | g | 7 |
|    | 8 | p | g | 5 |                             | 1 | k | j | 8 |
|    | 9 | h | y | 9 |                             | 9 | h | y | 9 |

## 2.4 Subset Used in This Work

This section outlines the specific framework adopted in this thesis. It first presents the standard relational algebra operators and their formal semantics, before introducing the additional constructs included in the extended relational algebra.

### 2.4.1 Relational Algebra Used

To maintain a coherent interpretation of the relational algebra across all operators, this thesis adopts the semantics of the named conjunctive algebra, referred to as the SPJRU algebra, proposed by Abiteboul et al. [3] in "Foundations of Databases." Since their algebra does not include a division operator, this framework is completed by the definition of division provided by Elmasri et al. [14] in "Fundamentals Of Database Systems."

Let  $I$  be the instance of a relation  $R$  of schema  $S$ . Where  $S$  is a finite and non-empty set of attributes  $S = \{A_1, A_2, \dots, A_n\}$ . The unary operators of the algebra are defined as:

|            |                                                                                                                                                                          |
|------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Projection | $\pi_{A_i, \dots, A_j}(I) = \{\langle t(A_i), \dots, t(A_j) \rangle \mid t \in I\}$ ,<br>where $\{A_i, \dots, A_j\}$ is a non-empty subset of $S$ .                      |
| Selection  | $\sigma_C(I) = \{t \mid t \in I \wedge t \text{ satisfies } C\}$ ,<br>where $C$ is a condition expressed on $S$ .                                                        |
| Rename     | $\rho_{A \leftarrow B}(I) = \{t \mid \exists t' \in I : t(B) = t'(A) \wedge \forall A_i \neq A : t(A_i) = t'(A_i)\}$ ,<br>where $B$ is not included in $S \setminus A$ . |

Let  $I_1$  be the instance of a relation  $R_1$  of schema  $S_1$  and  $I_2$  be the instance of a relation  $R_2$  of schema  $S_2$ . The binary operators of the algebra are defined as:

|                   |                                                                                                                                                                                                                                              |
|-------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Union             | $I_1 \cup I_2 = \{t \text{ over } S_1 \mid t \in I_1 \vee t \in I_2\}$ ,<br>where $S_1 = S_2$ .                                                                                                                                              |
| Intersection      | $I_1 \cap I_2 = \{t \text{ over } S_1 \mid t \in I_1 \wedge t \in I_2\}$ ,<br>where $S_1 = S_2$ .                                                                                                                                            |
| Difference        | $I_1 - I_2 = \{t \text{ over } S_1 \mid t \in I_1 \wedge t \notin I_2\}$ ,<br>where $S_1 = S_2$ .                                                                                                                                            |
| Cartesian product | $I_1 \times I_2 = \{t \text{ over } S_1 \cup S_2 \mid \exists u \in I_1 : \exists v \in I_2 : t(S_1) = u \wedge t(S_2) = v\}$ ,<br>where $S_1 \cap S_2 = \emptyset$ .                                                                        |
| Natural Join      | $I_1 \bowtie I_2 = \{t \text{ over } S_1 \cup S_2 \mid \exists u \in I_1 : \exists v \in I_2 : t(S_1) = u \wedge t(S_2) = v\}$ .<br>If $S_1 \cap S_2 = \emptyset$ , then $I_1 \bowtie I_2$ is the Cartesian product of $I_1$ and $I_2$ . [2] |
| Division          | $I_1 \div I_2 = \{t \text{ over } S_1 \setminus S_2 \mid \forall v \in I_2, \exists u \in I_1 : u(S_2) = v \wedge u(S_1 \setminus S_2) = t\}$ ,<br>where $S_2 \subseteq S_1$ .                                                               |

Let  $I_1$  be the instance of a relation  $R_1$  of schema  $S_1$  and  $I_2$  be the instance of a relation  $R_2$  of schema  $S_2$ . The derived join operators of the algebra are defined in terms of the previous operators as:

|            |                                                                                                                                                 |
|------------|-------------------------------------------------------------------------------------------------------------------------------------------------|
| Theta-join | $I_1 \bowtie_C I_2 = \sigma_C(I_1 \times I_2)$ ,<br>where $C$ is a condition expressed on $S_1 \cup S_2$ , and $S_1 \cap S_2 = \emptyset$ [11]. |
| Semi-join  | $I_1 \ltimes I_2 = \pi_{S_1}(I_1 \bowtie I_2)$ [3].                                                                                             |
| Anti-join  | $I_1 \rhd I_2 = I_1 - (I_1 \ltimes I_2)$ [15].                                                                                                  |

## 2.4.2 Extended Relational Algebra Used

The precise variant of the extended relational algebra adopted in this thesis combines the extensions of the standard relational algebra presented in the previous section.

First, it adopts the multi-set semantics introduced by Dayal et al. [19] in "An extended relational algebra with control over duplicate elimination," in which relations are treated as multi-sets of tuples rather than sets. Consequently, tuples may appear multiple times and are associated with a multiplicity. All of the standard operators of the relational algebra are interpreted according to their multi-set definitions. Let  $I$  be the instance of a relation  $R$  of schema  $S$ . In the extended relational algebra, a relation instance is represented as a multi-set of pairs  $\langle t, i \rangle$ , where  $t$  is a tuple over  $S$  and  $i > 0$  denotes its multiplicity. The unary operators of the multi-set algebra are defined as:

|            |                                                                                                                                                                                                                                                                                          |
|------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Projection | $\pi_{A_i, \dots, A_j}(I) = \{\langle x, k \rangle \mid x \text{ is a tuple over } \{A_i, \dots, A_j\} \text{ and } k = \sum i \text{ for all } \langle t, i \rangle \in I \text{ such that } t \text{ agrees with } x\},$<br>where $\{A_i, \dots, A_j\}$ is a non-empty subset of $S$ . |
| Selection  | $\sigma_C(I) = \{\langle t, i \rangle \mid \langle t, i \rangle \in I \wedge t \text{ satisfies } C\},$<br>where $C$ is a condition expressed on $S$ .                                                                                                                                   |

Let  $I_1$  be the instance of a relation  $R_1$  of schema  $S_1$  and  $I_2$  be the instance of a relation  $R_2$  of schema  $S_2$ . In the following definitions, the multiplicity of a tuple not occurring in a multi-set relation is taken to be 0. The binary operators of the multi-set algebra are defined as:

|                   |                                                                                                                                                                                                                                                                                                                              |
|-------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Union             | $I_1 \cup I_2 = \{\langle t, k \rangle \mid \exists \langle t, i \rangle \in I_1 : \exists \langle t, j \rangle \in I_2 : k = \max(i, j) \wedge k \neq 0\},$<br>where $S_1 = S_2$ with $t$ defined over $S_1$ .                                                                                                              |
| Intersection      | $I_1 \cap I_2 = \{\langle t, k \rangle \mid \exists \langle t, i \rangle \in I_1 : \exists \langle t, j \rangle \in I_2 : k = \min(i, j) \wedge k \neq 0\},$<br>where $S_1 = S_2$ with $t$ defined over $S_1$ .                                                                                                              |
| Difference        | $I_1 - I_2 = \{\langle t, k \rangle \mid \exists \langle t, i \rangle \in I_1 : \exists \langle t, j \rangle \in I_2 : k = i \ominus j \wedge k \neq 0\},$<br>where $i \ominus j$ is equal to $i - j$ if $i > j$ , or 0 otherwise,<br>and $S_1 = S_2$ with $t$ defined over $S_1$ .                                          |
| Cartesian product | $I_1 \times I_2 = \{\langle t, i \times j \rangle \mid \exists \langle u, i \rangle \in I_1 : \exists \langle v, j \rangle \in I_2 : t(S_1) = u \wedge t(S_2) = v\},$<br>where $S_1 \cap S_2 = \emptyset$ with $t$ defined over $S_1 \cup S_2$ .                                                                             |
| Natural Join      | $I_1 \bowtie I_2 = \{\langle t, i \times j \rangle \mid \exists \langle u, i \rangle \in I_1 : \exists \langle v, j \rangle \in I_2 : t(S_1) = u \wedge t(S_2) = v\},$<br>where $t$ is defined over $S_1 \cup S_2$ .<br>If $S_1 \cap S_2 = \emptyset$ , then $I_1 \bowtie I_2$ is the Cartesian product of $I_1$ and $I_2$ . |
| Division          | $I_1 \setminus I_2 = \{\langle t, k \rangle \mid k = \min_{\langle v, j \rangle \in I_2} [i_{t,v} \setminus j] \wedge k \neq 0\},$<br>where $i_{t,v}$ is such that $\langle t \cup v, i_{t,v} \rangle \in I_1$ ,<br>and $S_2 \subseteq S_1$ with $t$ defined over $S_1 \setminus S_2$ .                                      |

In addition to the standard operators, this thesis also adopts the additional operators introduced previously. Namely, duplicate elimination, grouping with aggregation, and generalised projection. Let  $I$  be the instance of a relation  $R$  of schema  $S$ . These additional operators are defined as:

|                        |                                                                                                                                                                                                                                                                                                                                                                                             |
|------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Duplicate elimination  | $\delta(I) = \{\langle t, 1 \rangle \mid (\exists i)(\langle t, i \rangle \in I)\}.$                                                                                                                                                                                                                                                                                                        |
| Grouping               | $\gamma_{\alpha, f, \beta}(I) = \{\langle x \oplus [f(I_x), \beta], 1 \rangle \mid x \in \pi_\alpha(I)\},$<br>where $I_x = \{y \in I \mid y(\alpha) = x\}.$                                                                                                                                                                                                                                 |
| Generalised projection | $\pi_{A_i \rightarrow B_i, \dots, A_j \rightarrow B_j, e_k \rightarrow B_k, \dots, e_l \rightarrow B_l}(I) = \{\langle t, i \rangle \mid \exists \langle u, i \rangle \in I : \forall B_m \in \{B_i, \dots, B_j\} : t(B_m) = u(A_m) \wedge \forall B_n \in \{B_k, \dots, B_l\} : t(B_n) = e_n(u),$<br>where $\{e_k, \dots, e_l\}$ are arithmetic expressions defined on attributes of $S$ . |

Finally, this thesis adopts the support for null values introduced by Codd [20] in "Extending the Database Relational Model to Capture More Meaning." Relations may contain null values to represent

incomplete information. In addition, the outer join operators  $\bowtie$ ,  $\ltimes$ , and  $\rtimes$  are included. Let  $I_1$  be the instance of a relation  $R_1$  of schema  $S_1$  and  $I_2$  be the instance of a relation  $R_2$  of schema  $S_2$ . The outer joins operators are defined as:

|                  |                                                                                                                                                                                                                                                             |
|------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Left Outer Join  | $I_1 \bowtie I_2 = (I_1 \ltimes I_2) \cup$<br>$\{\langle t, i \rangle \mid \langle t_1, i \rangle \in I_1 \wedge \nexists \langle t_2, j \rangle \in I_2 : t_1(S_1 \cap S_2) = t_2(S_1 \cap S_2) \wedge t(S_1) = t_1 \wedge t(S_2 \setminus S_1) = null\}.$ |
| Right Outer Join | $I_1 \ltimes I_2 = (I_1 \bowtie I_2) \cup$<br>$\{\langle t, j \rangle \mid \langle t_2, j \rangle \in I_2 \wedge \nexists \langle t_1, i \rangle \in I_1 : t_1(S_1 \cap S_2) = t_2(S_1 \cap S_2) \wedge t(S_2) = t_2 \wedge t(S_1 \setminus S_2) = null\}.$ |
| Full Outer Join  | $I_1 \rtimes I_2 = (I_1 \bowtie I_2) \cup (I_1 \ltimes I_2).$                                                                                                                                                                                               |

Lastly, the extended algebra is extended with a sort operator, denoted by  $\tau$ , following the ordered relational model proposed by Slivinskasy et al. [21] in "Bringing Order to Query Optimization." Let  $I$  be the instance of an ordered relation  $R$  of schema  $S$ . The sort operator is defined as:  $\tau_{(A_1, o_1), \dots, (A_n, o_n)}(I) = L$ , where  $L$  is the ordered list of tuples of  $I$  sorted lexicographically according to the attributes  $A_1, \dots, A_n$ , and each order specifier  $o_i \in \{ASC, DESC\}$ .

## 2.5 Query Equivalence in Relational Algebra

Deciding if two relational algebra queries are equivalent is an important problem in database theory. Informally, two queries are *equivalent* if they yield the same result for every possible database instance. This problem is particularly important in the context of learning tools, since a student query may use a different syntax from the reference solution, while still being semantically equivalent.

A formal definition of query equivalence was given by Ashok K. Chandra and Philip M. Merlin [22] in their 1977 paper "Optimal implementation of conjunctive queries in relational data bases." They define two queries  $Q_1, Q_2$  to be equivalent, written  $Q_1 \equiv Q_2$ , if for every database  $B$  compatible with both queries,  $Res(Q_1, B) = Res(Q_2, B)$ , where  $Res(Q, B)$  denotes the result of evaluating query  $Q$  on database  $B$ .

### 2.5.1 Conjunctive Query Equivalence

A particularly important fragment of relational algebra is the class of *conjunctive queries*. Informally, they correspond to a restricted fragment of relational expressions that use only the selection, projection, cartesian product, join, and rename operators [3]. In particular, they exclude the union and difference operators. Despite this restriction, they remain highly expressive and form the theoretical basis of many optimisation techniques. Chandra and Merlin [22] formally introduce *conjunctive queries* as first-order queries of the form  $(x_1, x_2, \dots, x_k) \cdot \exists x_{k+1} x_{k+2} \dots x_m \cdot A_1 \wedge A_2 \wedge \dots \wedge A_r$ , where each  $A_i$  is an atomic formula.

To study the equivalence of conjunctive queries, Chandra and Merlin [22] associate each query with a *natural model*. Informally, this model can be viewed as a graph-like canonical representation of the query, where the variables define the nodes and the relational atoms define the connections between them. Formally, Chandra and Merlin define the natural model  $M_Q$  of a conjunctive query  $Q$  as an algebraic structure whose domain  $D_Q$  consists of the variables and constants that occur in the query. More precisely, if an atomic formula  $R(z_1, \dots, z_r)$  appears in  $Q$ , then the corresponding relation in  $M_Q$  contains the tuple  $(z_1, \dots, z_r)$ . Using this representation, questions about query equivalence and containment can be reformulated as checking for the existence of *homomorphisms*, which are structure-preserving mappings between two models. In [3], Abiteboul et al. formally define the homomorphism from a directed graph  $G = (V, E)$  to a directed graph  $G' = (V', E')$  as a function  $h : V \rightarrow V'$  such that "for each pair  $u, v \in V$ ,  $(u, v) \in E$  implies  $(h(u), h(v)) \in E'$ " and note that "they generalize in the natural fashion to other data and algebraic structures, such as relations, algebraic groups, etc."

A key result established by Chandra and Merlin in [22] is that: "*For conjunctive queries  $Q_1, Q_2$ ,  $Q_1 \equiv Q_2$  iff there exist homomorphisms  $h_1 : M_{Q_1} \rightarrow M_{Q_2}$ ,  $h_2 : M_{Q_2} \rightarrow M_{Q_1}$ .*" Since the problem of determining whether such homomorphisms exist is NP-complete, checking for conjunctive query equivalence is also NP-complete. Therefore, although equivalence for conjunctive queries is decidable, it remains computationally expensive in the general case.

Abiteboul et al. [3] further note that many desirable theoretical properties of conjunctive queries, including the decidability of equivalence and containment, extend naturally to conjunctive queries with union.

### 2.5.2 Full Relational Algebra Equivalence

The full relational calculus and relational algebra are obtained by extending conjunctive queries with union and negation, or equivalently, the difference operator in relational algebra [3].

The *relational calculus* is a declarative query language based on the predicate calculus of first-order logic. Instead of specifying a sequence of operations, as in relational algebra, a relational calculus query specifies the properties that the tuples in the result must satisfy. A relational calculus query has the form  $x_1, \dots, x_n \mid \phi(x_1, \dots, x_n)$ , where  $\phi$  is a first-order formula constructed from relational atoms, logical operators, and quantifiers [3].

In [3], Abiteboul et al. demonstrate the undecidability of equivalence for the relational calculus by reduction from the satisfiability problem: "*For equivalence, suppose that there were an algorithm for deciding equivalence between relational calculus queries. Then the satisfiability problem can be solved as follows: For each query  $q = \{x_1, \dots, x_n \mid \phi\}$ , this is unsatisfiable if and only if it is equivalent to the empty query  $q^\emptyset$ . This demonstrates that equivalence is not decidable.*"

Since relational algebra and relational calculus have equivalent expressive power [3], this result also applies to relational algebra expressions. Therefore, no general algorithm can determine whether two arbitrary relational algebra expressions are equivalent on all databases.

## 2.6 Summary

This chapter established the theoretical foundations that underlie the remainder of the thesis.

It first introduced the relational model proposed by Edgar F. Codd, highlighting its mathematical basis in set theory, the distinction between relation schemas and instances, as well as the difference between domain-ordered and domain-unordered relations.

Next, the chapter examined relational algebra, discussing its historical development, the contemporary definitions of its operators, and the choices made for this thesis. The discussion then turned to extended relational algebra and the motivations behind the different extensions of the classical operator set. Finally, the specific subset of relational algebra and extended relational algebra adopted in this thesis was defined.

Lastly, the chapter addressed the problem of query equivalence, as it provides an important theoretical context for the design and limitations of a system that compares relational algebra expressions.

## Chapter 3

# Web Environment for (Extended) Relational Algebra Expressions

This chapter presents the design and implementation of the interactive relational algebra learning environment developed in this thesis. It begins by defining the technical and pedagogical requirements that the tool must satisfy based on the needs of the database course. Next, a state-of-the-art analysis of existing relational algebra tools is conducted. The discussion then turns to the reasons why RelaX was chosen as the foundation for the implementation. Finally, the chapter details the modifications that were applied to the codebase of RelaX to align its syntax, interface, and semantics with the relational algebra formalism adopted in this thesis.

### 3.1 Specifications

The tool that will be provided to students must satisfy several specific requirements:

- **Web-based.** The program should run entirely in the browser, as students tend to prefer Web-based tools, since they do not require any installation or setup and can even be used on mobile devices.
- **Support for all primitive operators.** All of the primitive relational algebra operators must be available to use in expressions. Namely, selection ( $\sigma$ ), projection ( $\pi$ ), union ( $\cup$ ), difference ( $-$ ), Cartesian Product ( $\times$ ), and rename ( $\rho$ ).
- **Support for derived operators.** The tool should also implement every derived operator covered in the course material. Namely, intersection ( $\cap$ ), division ( $\div$ ), join ( $\bowtie$ ), theta-join ( $\bowtie_{A_1 \theta A_2}$ ), semi-join ( $\ltimes$ ), and anti-join ( $\rhd$ ).
- **Adherence to the semantics of the course.** To guarantee an alignment with the theory presented in the course, the tool must enforce the semantics presented in the previous chapter on its operators. For example:
  - **Domain unordered relations.** The algebra must be considered in the named perspective, i.e., no operation should rely on the positional ordering of attributes.
  - **Select.** The selection operator must accept a Boolean combination of comparisons as its filter conditions and must support the full range of comparison operators.
  - **Rename.** The rename operator must allow the renaming of only a subset of attributes without requiring the entire schema to be renamed.
  - **Union compatibility.** The union, difference, and intersection operations must only be defined on union-compatible relations, i.e. relations with identical sets of attribute names and domains.

- **Cartesian product.** The Cartesian product and theta-join operations must only be defined on relations with disjoint sets of attribute names.
- **Multi-set support.** The tool must provide a separate environment for extended relational algebra expressions where relations are treated as multi-sets.
- **Extended relational algebra extensions.** To guarantee an alignment with the theory presented in the course, this separate multi-set relational algebra environment must support the extensions presented in the previous chapter. For example:
  - **NULL values support.** This environment must support the inclusion of null values.
  - **Support for all extended operators.** All of the extended relational algebra operators covered in the course must be available to use in extended expressions. Namely, duplicate elimination ( $\Delta$ ), group by ( $\gamma$ ), generalised projection, order by ( $\tau$ ), and outer joins ( $\bowtie$ ,  $\Join$ , and  $\ltimes$ ).
  - **Group by.** The group by operator must accept the following aggregate functions: SUM, AVG, MIN, MAX, and COUNT.

These requirements ensure that the tool aligns with both pedagogical needs and the formal semantics of relational algebra and extended relational algebra seen in the course material. They will serve as the basis for evaluating existing resources and designing the new tool.

## 3.2 State of the Art

Several tools already exist for practising relational algebra, each with its own strengths and limitations. Because the objective of this thesis is to provide a tool that can be used for practical exercises in database courses, only software that is publicly accessible and can be easily obtained and used in practice is considered. Therefore, tools described in the literature but for which no source code or executable version is available, such as ReliQ [23], are excluded from this analysis.

The following subsections present the characteristics of each tool in detail, and Table 3.1 provides a summary of these findings.

### 3.2.1 WinRDBI

WinRDBI (Windows Relational DataBase Interpreter) [24] is an educational tool developed by Suzanne W. Dietrich, Eric Eckert, and Kevin Piscator at Arizona State University. It can be downloaded for free to Windows systems, although it is not distributed under an open-source licence. The underlying relational database interpreter is implemented with the Amzi! Prolog Logic Server, while the latest version of its graphical user interface is built in Java. WinRDBI supports the evaluation of relational algebra expressions and is designed primarily for use in the introductory database course at Arizona State University.

A significant limitation of WinRDBI is its lack of support for the rename operator. The tool also does not provide a generalised projection operator, which makes attribute renaming impossible. As a result, queries that rely on attribute renaming, such as self-joins, disambiguation of attribute names for the Cartesian Product, or schema alignment for the set operators, cannot be expressed with the tool. In addition, the select operator lacks some comparison capabilities. In particular, inequality cannot be expressed because the tool does not support an explicit inequality operator or a Boolean negation operator to emulate it.

WinRDBI also, by design, does not provide any derived operators. The developers explicitly state: "*The  $\theta$ -join and division operators are purposely not provided by the interpreter so that students investigate the equivalent definitions of these operators in terms of the fundamental relational algebra operators.*"

Table 3.1: Comparison of the existing resources for relational algebra.

| Program                            | Open-source | Web-based | All<br>primitive<br>operators | All<br>derived<br>operators | Union-<br>compatibility | Cartesian<br>product | Select | Rename | Multi-set<br>support |
|------------------------------------|-------------|-----------|-------------------------------|-----------------------------|-------------------------|----------------------|--------|--------|----------------------|
| WinRDBI                            | ✗           | ✗         | ✗                             | ✗                           | ✓                       | ✓                    | ✗      | ✗      | ✗                    |
| Relational                         | ✓           | ✗         | ✓                             | ✗                           | ✓                       | ✓                    | ✓      | ✓      | ✗                    |
| DBSnap                             | ✓           | ✓         | ✓                             | ✗                           | ✓                       | ✓                    | ✗      | ✗      | ✗                    |
| RA (radb)                          | ✓           | ✗         | ✓                             | ✗                           | ✗                       | ✗                    | ✓      | ✗      | ✗                    |
| RelaX                              | ✓           | ✓         | ✓                             | ✓                           | ✗                       | ✗                    | ✓      | ✓      | ✓                    |
| Scientific<br>calculator<br>online | ✗           | ✓         | ✓                             | ✗                           | ✓                       | ✓                    | ✗      | ✗      | ✗                    |
| Rali                               | ✓           | ✗         | ✓                             | ✓                           | ✗                       | ✓                    | ✓      | ✓      | ✓                    |

### 3.2.2 Relational

Relational [25] is a tool developed and maintained by Salvo 'LtWorf' Tomaselli. It is available for Linux systems and is distributed under the GPLv3 licence, making it free open-source software [26]. The tool provides multiple interfaces for working with relational algebra: a graphical interface, a command-line interface, and a Python library. Together, these components aim to offer "*a workspace for experimenting with relational algebra.*"

Relational implements most of the relational algebra operators as defined in the course material, with the main exception being the absence of the  $\theta$ -join as a distinct operator. The most significant practical limitation of Relational is that it is only available as downloadable software for Linux. Since the introductory database course is sometimes given to students outside of computer science, it is not reasonable to assume that they are familiar with Linux or willing to install an additional operating system.

### 3.2.3 DBSnap

DBSnap [27] is a Web-based learning tool for relational algebra developed by Yasin N. Silva and Jaime Chon at Arizona State University. It features a block-based interface for query construction, allowing students to assemble relational algebra expressions visually rather than typing formulas directly.

A major limitation of DBSnap is the way it handles the rename operator. The user needs to provide the positional ordering of the attribute to be renamed, which implicitly assumes that relations are domain-ordered. This contrasts with the semantics adopted in this thesis, where attribute names determine schema correspondence rather than their positions. In addition, the selection operator in DBSnap cannot express inequality, since the system does not support an explicit inequality comparison operator or any form of Boolean negation that could emulate it. Lastly, DBSnap also lacks some of the derived operators covered in the courses, including division, semi-join, and anti-join.

### 3.2.4 RA (radb)

RA (radb) [28] is a command-line tool for relational algebra developed and maintained by Jun Yang at Duke University. It is written in Python 3 and distributed as open-source software under a licence equivalent to the MIT licence. Its parser is built using ANTLR 4, which is released under the BSD licence.

A major semantic limitation of RA is its definition of union compatibility. In RA, two relations are considered union-compatible only if they have the same number of attributes, even if their attribute names or domains do not match. This definition diverges significantly from the notion of union compatibility adopted in this thesis and in the course material. RA also only issues a warning when the user attempts to compute the Cartesian product of relations that share attribute names. However, the operation is still performed. The rename operator in RA requires users to provide a new name for every attribute in the relation, preventing partial renaming of only some attributes. Furthermore, like most tools surveyed, RA lacks several derived operators covered in the course, including division, semi-join, and anti-join. Lastly, from a practical perspective, another limitation of RA is that the tool is available only as a command-line application, which is often not the preferred interface for students.

### 3.2.5 RelaX

RelaX (relational algebra calculator) [29] is a Web-based relational algebra tool developed and maintained by Johannes Kessler at the University of Innsbruck under the supervision of Michael Tschuggnall and Günther Specht. It is written in Typescript and distributed under the MIT licence, making it free and open-source software.

RelaX semantics differs from the formalism defined previously in many ways. For example, RelaX implicitly renames each attribute by prefixing each attribute name with the name of its relation using dot notation. For example, if a relation  $R$  contains an attribute  $A$ , the name of the attribute will be displayed

as  $R.A$ . As a consequence of this implicit renaming strategy, the Cartesian product is defined for any pair of relations, even when they share attribute names. The only disallowed case is the Cartesian product of a relation with itself. RelaX also adopts a domain-ordered definition of union compatibility, as two relations are considered union-compatible if the domains of their attributes positionally match. This does not align with the name-based unordered definition of union compatibility used in this thesis and in the course material.

However, despite these semantic differences, RelaX has a lot of advantages: it is Web-based, open-source, distributed under a permissive licence, supports multi-sets with null values, implements all of the primitive and derived relational algebra operators defined in the course, and is the tool that students currently seem to gravitate toward the most on their own.

### 3.2.6 Scientific Calculator Online

Scientific Calculator Online [30] is a Web-based platform that offers multiple types of online calculators, including an interface for executing relational algebra queries. The site includes limited information about its developers. It contains claims, such as "*Today, our revolutionary calculator has gained international recognition, winning the prestigious Turing Award for innovation in computing,*" that could not be verified through external sources. For this reason, while the tool can be used for executing simple relational algebra expressions, its provenance and reliability remain uncertain.

### 3.2.7 RALI

RALI (Relational Algebra Learning Instrument) [31] is a command-line tool developed by Prof. Debruyne to aid students of the introductory database course in their learning of relational algebra. It is written in Java, and its parser is implemented using ANTLR 4. The tool is distributed under the GPLv3 licence, making it free and open-source software [26]. The tool also provides an extended version, ERALI, to execute extended relational algebra queries on multi-sets.

Since RALI was developed directly with the course in mind, it implements all primitive and derived relational operators defined in the course material and generally follows the corresponding semantics. However, an exception arises in the treatment of set operations. In RALI, two relations are considered union-compatible if it is possible to find at least one mapping between the domains of their attributes. This interpretation of union compatibility is the one proposed by Codd in his 1990 book [9] but it does not align with the name-based definition of union compatibility defined in the course material.

After a discussion with Prof. Debruyne, it was clarified that this discrepancy between the way RALI operates and the course material arises from the fact that RALI internally relies on SQL to execute queries. According to PostgreSQL [32], in SQL: "In order to calculate the union, intersection, or difference of two queries, the two queries must be 'union-compatible', which means that they return the same number of columns and the corresponding columns have compatible data types."

Similar to RA, another limitation of RALI is that the tool is only available as a command-line application, which is often not the preferred interface for students.

## 3.3 Why RelaX was Chosen

Among the tools surveyed in the previous section, RelaX [29] was selected as the foundation for the learning environment developed in this thesis. This choice is motivated by a combination of practical and pedagogical considerations.

First, RelaX satisfies one of the primary requirements identified at the beginning of this chapter, as it is a fully Web-based tool, meaning it requires no installation or configuration. As a result, it is immediately accessible to students regardless of their operating system or technical background. Second, it is open-source and distributed under the permissive MIT licence. This is essential to guarantee the feasibility of

this project, as it allows anyone to modify and extend its implementation. Third, it already provides a rich implementation of relational algebra, including support for both primitive and derived operators, as well as a multi-set environment that supports null values. Lastly, RelaX is already commonly used by students following the database course. Building upon a familiar platform can increase the chance that students will use the tool. In conclusion, RelaX was chosen as the foundation of the tool due to its accessibility, permissiveness towards modifications, expressive power, and existing student adoption.

However, RelaX does not adhere fully to the relational algebra semantics adopted in this thesis. In particular, its implicit renaming strategy and its domain-based definition of union compatibility differ from the approach used in the course material. For this reason, the following section consists of adapting the behaviour of RelaX to ensure consistency with the formal framework defined previously.

## 3.4 Aligning RelaX Codebase with the Course Material

The modifications presented in this section directly address the limitations identified in the state-of-the-art analysis of RelaX in Section 3.2.5. Their objective is to align both the syntax and the semantics of the tool with the relational algebra formalism used in the database course.

The modifications are organised into three complementary layers. First, the syntax layer defines how relational algebra expressions are written and interpreted by the system. Second, the view layer controls how the query results and schemas are presented to the user. Third, the model layer governs the semantic behaviour of relational algebra operators during query evaluation.

### 3.4.1 Implementation Overview

The tool developed in this thesis was implemented as an extension of the RelaX platform. It is publicly available at <https://alexiadonati.github.io/relax-modified/> and its codebase can be found at the following Uliège GitLab repository: <https://gitlab.uliege.be/Alexia.Donati/relax-modified> or the following GitHub repository as a fork of the original RelaX repository: <https://github.com/AlexiaDonati/relax-modified>.

The implementation was carried out within the existing RelaX architecture, which is primarily based on TypeScript and React. In addition, version control was performed using Git.

Overall, both the alignment of the tool with the formalism of the course material and the implementation of the exercise and verification system, which will be discussed in the following chapter, involved the modification of 80 files. According to the Git revision history, these modifications correspond to approximately 4,817 added and 2,032 removed lines of code.

### 3.4.2 Licences

RelaX's code [33] is distributed under both the MIT licence for the entire repository and the Mozilla Public Licence 2.0 for specific files. All modifications to MPL-licensed files have been made in accordance with the MPL-2.0 terms [34], preserving the original licence headers and attributions.

### 3.4.3 Syntax Modifications

This section presents modifications made to the syntax of the relational algebra language supported by RelaX. The goal of these changes is to align the syntax accepted by the tool with the notation used in the database course of Prof. Debruyne.

All modifications described here affect only the parsing layer of the program. They affect how expressions are written and interpreted by the parser but do not alter the internal representation or semantics of the language.

These changes were implemented by modifying the parser of Relax, which is generated using the PEG.js package. PEG.js [35] is a JavaScript parser generator originally developed by David Majda from 2010 to 2016 and maintained by Futago-za Ryu since 2017. It takes a grammar written in Parsing Expression Grammar (PEG) syntax and produces a parser that can read an input string and either return an AST or raise a parse error. In the context of Relax, this input corresponds to the relational algebra formula entered by the user.

### Rename Operator Syntax

This modification concerns the syntax of the rename operator  $\rho$ . In Relax, to rename the column  $a$  of the relation  $R$  to  $x$ , the following formulas can be used:  $\rho x \leftarrow a(R)$  or  $\rho a \rightarrow x(R)$ . However, in the course, the same operation is written as  $\rho_{a \leftarrow x}(R)$ . To ensure consistency with the course notation, the syntax accepted by the tool was adapted accordingly.

This is achieved by reversing the position of the original attribute name and the new attribute name around the arrow in the rename operator. In the PEG grammar, this corresponds to swapping the occurrences of the grammar rule that parses an unqualified column identifier, `unqualifiedColumnName`, and the rule that parses the column reference, `columnName`, around the arrow token. The updated version of the `colAssignment` rule that illustrates this change is shown in Listing 1. The same modification is applied across all grammar rules required to parse the rename operator.

```
// Original rule
colAssignment = dst:unqualifiedColumnName arrowLeft src:columnName

// Modified rule
colAssignment = src:columnName arrowLeft dst:unqualifiedColumnName
```

Listing 1: Modification of the `colAssignment` rule in the PEG.js grammar of Relax.

### Support for Accented Characters

In Relax, a relation or column name can contain any alphanumeric characters and underscores, but cannot contain accented characters. However, since the course is taught in French, accented characters are very common in relation and column names. Supporting accented characters is therefore necessary to remain consistent with the notation used in the course material.

This requires extending the current grammar rule to include the character range that provides accented Latin characters. In the PEG grammar, this is achieved by adding the character range `À-ÖØ-öø-ÿ` to the regular expression of the grammar rule that parses a relation or column name. The updated version of the `relationName` rule is shown in Listing 2. The same modification is applied to the `unqualifiedColumnName` rule.

```
// Original rule
relationName 'relationName'
= !(RESERVED_KEYWORD !([0-9a-zA-Z_]+))
  a:$([a-zA-Z_]+ $[0-9a-zA-Z_]*)

// Modified rule
relationName 'relationName'
= !(RESERVED_KEYWORD !([0-9A-Za-zÀ-ÖØ-öø-ÿ_]+))
  a:$([A-Za-zÀ-ÖØ-öø-ÿ_]+ $[0-9A-Za-zÀ-ÖØ-öø-ÿ_]*)
```

Listing 2: Modification of the `relationName` rule in the PEG.js grammar of RelaX.

### Allow Double Quotes for String Literals

In RelaX, as in SQL, only single quotes delimit strings. However, as will be discussed in Chapter 5, students seem to intuitively gravitate towards using double quotes. Furthermore, the solutions provided in the practical sessions of the course also use double quotes to denote string literals. Since the main focus of the tool is relational algebra rather than SQL, it was extended to allow both single and double quotes to denote string literals in the relational and extended relational algebra environments.

This modification consists of adding a parser expression that starts and ends with double quotes rather than single quotes to the current grammar rule. The updated version of the `string` rule is shown in Listing 3.

```
// Original rule
string 'string'
= '\'' a:$[^'\n]* '\''

// Modified rule
string 'string'
= '\'' a:$[^'\n]* '\''
/ '"' a:$[^"\n]* '"'
```

Listing 3: Modification of the `string` rule in the PEG.js grammar of RelaX.

### Allow Arrow for Assignment

In RelaX, assigning an expression to a variable is done using the equality symbol `=`. However, in the practical session of the course on relational algebra, the left arrow symbol `←` is used instead. To match the notation used in the course material, support for the left arrow symbol `←` was introduced as an alternative assignment operator, while keeping the original equality-based syntax currently used by RelaX.

This modification consists of extending the current parser rule for the assignment operator by adding an alternative expression that matches the left-arrow token. The updated version of the `assignmentOperator` rule is shown in Listing 4.

```
// Original rule
assignmentOperator
= _ '=' _

// Modified rule
assignmentOperator
= _ '=' _
/ _ arrowLeft _
```

Listing 4: Modification of the `assignmentOperator` rule in the PEG.js grammar of RelaX.

## 3.4.4 View-Level Modification

The modification presented in this section concerns the way relational schemas and query results are displayed to the user. Its objective is to ensure consistency between the visual representation provided by

the tool and the notation used in the course material.

### Attribute Display Without Relation Aliases

As discussed in Section 3.2.5, RelaX displays attribute names by prefixing them with the name of their origin relation. For example, if a relation  $R$  contains an attribute  $a$ , it would be displayed as  $R.a$  in the header of the resulting relation. This implicit qualification is used to distinguish attributes that originate from different relations. However, in the notation used in the course material, attributes are referred to by their name only, without relation qualifiers. To remain consistent with this notation, the display of the relation aliases was removed from the user interface.

In the implementation, relation aliases are stored in the `_relAlias` attribute of the `Column` class. The `_relAlias` attribute is used by the `toString` method through the `printColumn` function to generate the textual representation of a column. To prevent relation aliases from being displayed, the `printColumn` function was modified to always return only the attribute name, regardless of the value of `_relAlias`. The modification is shown in Listing 5, where the logic responsible for prefixing attribute names with their relation alias has been removed.

```
1 export class Column {
2   _name: string | number;
3   _relAlias: string | null;
4   _type: DataType = 'null';
5   ...
6   toString() { return Column.printColumn(this._name, this._relAlias); }
7   ...
8   static printColumn(name: string | number, relAlias: string | null): string {
9     let pName;
10    if (typeof (name) === 'number') { pName = `[${name}]`; }
11    else { pName = name; }
12
13    if (relAlias !== null) {
14      return pName;
15    }
16    return `${relAlias}.${pName}`;
17  }
18 }
```

Listing 5: Modification of the `printColumn` function in the `Column` class of RelaX.

As a result, attribute names are now displayed without relation qualifiers. This modification only affects the visual representation of the query results and does not alter the internal handling of attributes or the semantics of any relational algebra operations.

### 3.4.5 Model-level Modifications

In contrast to the previous sections, the modifications presented here affect how relational algebra expressions are evaluated by altering the semantic model of the system. These changes aim to resolve discrepancies between the original implementation of RelaX and the relational algebra semantics adopted in this thesis and in the course material.

## Cartesian Product

The previous modification only affects the way columns are displayed to the user and does not change how they are handled when executing queries. As a result, relation aliases are still taken into account when checking to see if two relations have conflicting columns.

In RelaX, the Cartesian product is defined for any pair of relations, even when they share attribute names. The only disallowed case is the product of a relation with itself. However, in the relational algebra semantics adopted in this thesis and in the course, the Cartesian product of two relations that share attribute names is not defined.

This difference comes from the way RelaX detects conflicts between attribute names. During the semantic check phase of the Cartesian product operator, the `_checkSchema` function, partially shown in Listing 6, verifies whether the two input relations contain conflicting attributes by calling the `getConflictingColumnsArray` function of the `Schema` class.

```
1 _checkSchema(schemaA: Schema, schemaB: Schema): void {
2     try {
3         const conflicts = schemaA.getConflictingColumnsArray(schemaB);
4         if (conflicts.length > 0) {
5             this.throwExecutionError(...);
6         }
7         ...
8     }
9     ...
10 }
```

Listing 6: Local function `_checkSchema` located in the `CrossJoin` class of RelaX.

In the original implementation, two columns are considered conflicting only if both their names and relation aliases match. As a result, attributes with identical names but different relation aliases are not detected as conflicts.

To enforce the detection of attribute name conflicts regardless of their relation aliases, the conflict detection mechanism was modified to ignore relation aliases. This change is shown in Listing 7, where the argument corresponding to the relation alias is removed and replaced by `null`. This instructs the `getColumnIndex` function to ignore the relation aliases entirely.

```
1 getConflictingColumnsArray(schemaB: Schema) {
2     const conflicts = [];
3     for (let i = 0; i < this._size; i++) {
4         const index = schemaB.getColumnIndex(
5             this._names[i],
6             this._relAliases[i] null,
7             false
8         );
9         if (index !== -1) {
10             conflicts.push(this.getColumn(i));
11         }
12     }
13     return conflicts;
14 }
```

Listing 7: Modification of the `getConflictingColumnsArray` function in the `Schema` class of RelaX.

As a result, two relations that contain attributes with the same name are now correctly identified as conflicting, and the Cartesian product is rejected in such cases. This modification changes the semantic model of RelaX by enforcing a fully name-based notion of attribute uniqueness, independently of relation aliases.

### Union-Compatibility

As discussed in Section 3.2.5, in RelaX, two relations are considered union-compatible if they have the same number of attributes and if their attribute types can be matched positionally. However, this definition does not align with the relational algebra semantics adopted in the course material and in this thesis.<sup>1</sup>

**Name-based Compatibility.** In RelaX, union compatibility is checked during the evaluation of set operations such as union, intersection, and difference. In the original implementation, this verification is performed using the `equalsTypeOnly` function, which compares attribute types positionally and ignores attribute names. This behaviour is illustrated in Listing 8.

```
1 equalsTypeOnly(schemaB: Schema) {  
2     if (this._size !== schemaB._size) {  
3         return false;  
4     }  
5     for (let i = 0; i < this._size; i++) {  
6         if (this._types[i] !== schemaB._types[i]) {  
7             return false;  
8         }  
9     }  
10    return true;  
11 }
```

Listing 8: Function `equalsTypeOnly` located in the `Schema` class of RelaX.

To enforce the name-based definition of union compatibility, the call to the `equalsTypeOnly` function in the `check` function of the set operations was replaced by a stricter comparison using the `equals` function, which also compares attribute names. This function, shown in Listing 9, not only verifies the type compatibility by calling `equalsTypeOnly`, but also compares the attribute names of both schemas. In its original form, `equals` also compared relation aliases. However, to remain consistent with the previously mentioned alias-independent interpretation, this comparison was removed.

---

<sup>1</sup>As explained in Section 2.1, this thesis adopts a fully named perspective on relations, where attribute names carry semantic meaning rather than serving only as positional identifiers. Therefore, as explained in Section 2.2.4, union-compatibility is restricted to relations whose schemas contain the same attribute names and domains, independently of attribute ordering.

```
1 equals(schemaB: Schema) {
2     if (this.equalsTypeOnly(schemaB) === false) {
3         return false;
4     }
5     for (let i = 0; i < this._size; i++) {
6         if (this._names[i] !== schemaB._names[i]
7             || this._relAliases[i] !== schemaB._relAliases[i]) {
8             return false;
9         }
10    }
11    return true;
12 }
```

Listing 9: Modification of the equals function in the Schema class of RelaX.

This modification ensures that union compatibility requires both matching attribute names and domain compatibility.

**Order Independence.** Although the equals function compares attribute names, its original implementation still relies on positional alignment between attributes. This means that, even if two relations contain the same set of attributes, they will not be considered union-compatible if the corresponding attributes are not positioned at the same indices in both relations. This conflicts with the domain-unordered view of relations required by the relational algebra semantics used in this thesis and taught in the course.

To address this issue, the comparison logic of the equals function was modified to match attributes by name and type, regardless of their position. The revised implementation, as shown in Listing 10, iterates through the attributes of one schema and attempts to find, for each attribute, a matching attribute with an identical name and type in the other schema. The schemas are considered equal, and therefore union-compatible, only if every attribute finds such a match.

```
1 equals(schemaB: Schema) {
2     if (this._size !== schemaB._size) { return false; }
3
4     for (let i = 0; i < this._size; i++) { // schemaA
5         let foundMatch = false;
6         for (let j = 0; j < schemaB._size; j++) { // schemaB
7             if (this._names[i] === schemaB._names[j]) { // name matches
8                 if (this._types[i] === schemaB._types[j]) { // type matches
9                     foundMatch = true;
10                    break;
11                }
12            }
13        }
14        if (foundMatch === false) { return false; }
15    }
16    return true; // all columns have a matching column
17 }
```

Listing 10: Further modification of the equals function in the Schema class of RelaX.

This change ensures that schema comparison is invariant to the ordering of attributes.

**Execution Alignment.** While the previous modification ensures correct schema comparison, it introduces a discrepancy with the execution phase. The execution model of each set operation still assumes that the matching attributes are positionally aligned. This leads to incorrect results whenever the matching attributes appear in different orders in both relations.

Fully unordered relations are not directly representable in the current implementation, since tuples are internally stored as ordered arrays of values. To resolve this issue, the execution of the set operations was modified to align the tuples of both relations with the target schema before combining them to ensure correct attribute correspondence. The modified `getResult` function, shown partially in Listing 11, replaces direct calls to the `getRows` function with calls to a new function, `getRowsMappedToSchema`.

```
1 getResult(doEliminateDuplicateRows: boolean = true, session?: Session) {
2     ...
3     const res = new Table();
4     const orgA = this.getChild().getResult(doEliminateDuplicateRows, session);
5     const orgB = this.getChild2().getResult(doEliminateDuplicateRows, session);
6     res.setSchema(this._schema);
7
8     const rowsA = orgA.getRows(); getRowsMappedToSchema(this._schema);
9     const rowsB = orgB.getRows(); getRowsMappedToSchema(this._schema);
10
11     res.addRows(rowsA);
12     res.addRows(rowsB);
13     ...
14     return res;
15 }
```

Listing 11: Modification of the `getResult` function in the `Union` class of `RelaX`.

The `getRowsMappedToSchema` function, as shown in Listing 12, performs this reordering in two steps. First, it builds a column-mapping array that indicates, for each attribute in the target schema, the position of its corresponding attribute in the schema of the source relation. Then, using this mapping, it reconstructs each row so that its values appear in the correct order expected by the target schema.

This modification ensures that the execution of union, intersection, and difference operations fully adheres to the name-based and domain-unordered definition of union compatibility adopted in the course material and in this thesis.

```
1  getRowsMappedToSchema(targetSchema: Schema): Tuple[] {
2      const colMapping: number[] = []; // build column mapping array
3      for (let i = 0; i < targetSchema.getSize(); i++) { // for each attribute
4          const col = targetSchema.getColumn(i);
5          const sourceIndex = this._schema.getColumnIndex(col.getName(), null);
6          colMapping.push(sourceIndex);
7      }
8
9      const mappedRows: Tuple[] = []; // map rows to target schema
10     for (let i = 0; i < this._rows.length; i++) { // for each row
11         const originalRow = this._rows[i];
12         const mappedRow: Tuple = [];
13         for (let j = 0; j < colMapping.length; j++) { // for each attribute
14             const sourceIndex = colMapping[j];
15             if (sourceIndex === -1) {
16                 mappedRow.push(null);
17             } else {
18                 mappedRow.push(originalRow[sourceIndex]);
19             }
20         }
21         mappedRows.push(mappedRow);
22     }
23     return mappedRows;
24 }
```

Listing 12: New function `getRowsMappedToSchema` located in the `Table` class.

## Full Outer join

In `RelaX`, the implementation of the join operators distinguishes between two join types, namely: natural and theta joins. However, in the original implementation of the full outer join operator, only theta-join semantics are supported.

Specifically, the schemas of both operand relations are always concatenated, and a check ensures that the input schemas do not contain conflicting attribute names. If such conflicts are detected, the operation is rejected. This is consistent with theta-join semantics but does not support natural-join semantics. In particular, when two relations share attribute names, the natural full outer join should merge these attributes. Although this behaviour is supported for the inner, left outer, and right outer join operators, it is not available for the full outer join.

To address this limitation, the `_checkSchema` function of the full outer join operator was modified to support both natural and theta join semantics, as shown in Listing 13. When the join condition is specified as a theta join, the original behaviour is preserved. Conflicts between attribute names are checked to prevent ambiguity, and schemas are concatenated to form the resulting schema. In contrast, when the join condition is specified as a natural join, the modified implementation constructs the resulting schema by merging attributes with identical names.

```
1 _checkSchema(schemaA: Schema, schemaB: Schema): void {
2     try {
3         if (this._joinConditionOptions.type === 'natural') { // natural join
4             const {keep, schema} = Schema.concatNatural(
5                 schemaA, schemaB, true, this._joinConditionOptions.restrictToColumns
6             );
7             this._schema = schema;
8
9             this._rowCreatorMatched = function (rowA: any[], rowB: any[]): any[] {
10                 return Join.createNaturalRowArray(
11                     rowA, rowB, keep.size, keep.keepIndicesA, keep.keepIndicesB
12                 );
13             }
14         }
15         else { // theta join (original implementation)
16             const conflicts = schemaA.getConflictingColumnsArray(schemaB);
17             if (conflicts.length > 0) { this.throwExecutionError(...); }
18
19             this._schema = Schema.concat(schemaA, schemaB);
20             this._rowCreatorMatched = function (rowA: any[], rowB: any[]): any[] {
21                 return rowA.concat(rowB);
22             };
23         }
24         ...
25     }
26     ...
27 }
```

Listing 13: Modification of the `_checkSchema` function in the `FullOuterJoin` class of `RelaX`.

This modification ensures that the full outer join operator supports both theta and natural join semantics, allowing attributes with identical names to be merged when appropriate, while preserving the original behaviour when explicit join conditions are used.

## Aggregate Functions over Value Expressions

In `RelaX`, the aggregate functions used in the group by operator can only take a single attribute reference as arguments. As a result, expressions such as `SUM(a + b)` are not supported.

To extend the expressiveness of the language and align it with the extended relational algebra used in the practical session of the course, aggregate functions were modified to operate on arbitrary value expressions rather than only column references, as shown in Listing 14.

This change requires adapting both the validation and evaluation phases of the group by operator. During the semantic check phase, each aggregate function expression is now validated against the input schema, as shown in Listing 15. Furthermore, during query evaluation, instead of directly accessing a column value by index, each aggregate function now evaluates the corresponding value expression for each tuple. This modification is illustrated for the min and max functions in Listing 16.

As a result of this modification, aggregate functions can now be applied to arbitrary value expressions instead of being limited to simple attribute references.

```

1 export interface AggregateFunction {
2     name: string | number,
3     aggFunction: string,
4     col: {
5         name: string | number,
6         relAlias: string | null,
7     }
8     col: ValueExpr | null,
9 }

```

Listing 14: Modification of the AggregateFunction interface in the GroupBy class of RelaX.

```

1 for (let i = 0; i < this.aggregateFunctions.length; i++) {
2     const f = this.aggregateFunctions[i];
3     ... // If f.aggFunction is unknown, throw an error.
4
5     if (f.aggFunction === 'COUNT_ALL') {
6         aggregateFunctionsExpr[i] = null;
7         continue;
8     }
9
10    const expr = f.col;
11    if (!expr) { throw new Error(...); }
12
13    expr.check(childSchema); // Check the value expression against the child schema.
14    aggregateFunctionsExpr[i] = expr;
15 }

```

Listing 15: Validation of the aggregate functions in the check function of the GroupBy class.

```

1 case 'MIN':
2 case 'MAX':
3     aggValue = null;
4     const c = func.aggFunction === 'MIN' ? genericMin : genericMax;
5
6     for (let j = 0; j < group.length; j++) {
7         value = group[j][funcColIndex]; expr!.evaluate(group[j], [], j, session);
8         if (value === null) {continue;}
9
10        if (aggValue === null) { aggValue = value; }
11        else { aggValue = c(aggValue, value); }
12    }
13
14    break;

```

Listing 16: Modification of the MIN and MAX functions evaluation in the getResult function.

## Date Subtraction in Value Expressions

In RelaX, the subtraction operator in value expressions only supports numerical operands. Hence, expressions involving date differences, such as computing the duration between two dates, are not supported.

To extend the expressiveness of value expressions, the subtraction operator was modified to also support date operands. When both operands are dates, the result of the subtraction is defined as the difference between their underlying timestamps, expressed as a numerical value corresponding to the elapsed time in milliseconds, as shown in Listing 17.

```
1 case 'sub':  
2     if (valueA === null || valueB === null) { return null; }  
3     if (valueA instanceof Date && valueB instanceof Date) {  
4         return valueA.getTime() - valueB.getTime();  
5     }  
6     return valueA - valueB;
```

Listing 17: Evaluation of the subtraction in the `_evaluateNumber` function of the `ValueExpr` class.

The validation logic of the subtraction operator was also extended. In the original implementation, the operator requires both operands to be of type number. In the modified version shown in Listing 18, it either accepts two numerical or two date operands.

```
1 case 'sub':  
2     this._args[0].check(schemaA, schemaB);  
3     this._args[1].check(schemaA, schemaB);  
4  
5     if ((this._args[0].getDataType() !== 'number'  
6         || this._args[1].getDataType() !== 'number')  
7         && (this._args[0].getDataType() !== 'date'  
8         || this._args[1].getDataType() !== 'date')) {  
9         this.throwExecutionError(...);  
10    }  
11    return true;
```

Listing 18: Validation of the subtraction in the `_checkNumber` function of the `ValueExpr` class.

As a result of this modification, the subtraction operator is now polymorphic as it can be applied to either numerical values or date values.

## Division in Extended Relational Algebra Environment

In RelaX, the evaluation of the division operator is not implemented as an independent process, but as a combination of primitive operators. More specifically, the division is evaluated as  $R \div S := \pi_{r'}(R) - \pi_{r'}((\pi_{r'}(R) \times S) - R)$  where  $r'$  is the set of attributes of  $R$  that do not appear in  $S$ . This rewrite of the operator is implemented in the `check` function of the operator, and the `getResult` function delegates the computation to this expression.

Although this approach correctly implements division under set semantics, it does not extend to multi-set semantics. This is because the equivalence relies on the absence of duplicates, as it only tests for the presence or absence of tuples, not whether they occur sufficiently many times. As a result, the expression may incorrectly keep tuples that do not meet the required multiplicity constraints, or eliminate tuples that should appear multiple times in the result.

To address this issue, the implementation of the `getResult` function was modified to distinguish between set and multi-set evaluation modes. When the division operator is evaluated in the standard set relational algebra environment, i.e., when duplicate elimination is enabled, the original implementation is preserved. In contrast, when the operator is evaluated in the extended relational algebra environment, i.e., when duplicate elimination is disabled, it is evaluated under multi-set semantics. In this interpretation, the division takes into account the multiplicities of the tuples and is defined as  $R \div S := \{t \mid t \in \pi_{r'}(R) \text{ and for all } s \in S, (t, s) \in R\}$ .

The modified implementation, partially shown in Listing 19, computes the result of the division operation in three steps. First, it counts the multiplicity of each tuple  $s$  in the divisor relation  $S$ . Then, it groups the tuples of  $R$  according to their projection on the result schema,  $T = \pi_{r'}(R)$ , and counts how often each tuple  $t \in T$  appears together with each tuple  $s \in S$  in  $R$ . Lastly, for each  $t$ , it determines how many times it can appear in the result by comparing this count with the required multiplicities of the tuples  $s$  in  $S$ . Only the tuples in  $T$  that satisfy the division condition for all tuples in  $S$  are included in the result, with a multiplicity equal to the minimum number of valid combinations.

```

1  getResult(doEliminateDuplicateRows: boolean = true, session?: Session) {
2      ...
3      if (doEliminateDuplicateRows === true) { // Set division implementation
4          ...
5          res = this._delegate.getResult(doEliminateDuplicateRows, session);
6          res.eliminateDuplicateRows();
7      } else { // Multi-set division implementation
8          ...
9          const requiredSCount = new Map<string, number>();
10         ... // Count multiplicities in the divisor relation (S).
11
12         const leftRowsT = dividend.getRowsMappedToSchema(schema);
13         const leftRowsS = dividend.getRowsMappedToSchema(divisor.getSchema());
14
15         interface TEntry { row: any[]; sCount: Map<string, number>; }
16         const tEntries = new Map<string, TEntry>();
17         ... // For each t, count how often it occurs with each s in R.
18
19         for (const { row, sCount } of tEntries.values()) {
20             let repetitions = Number.POSITIVE_INFINITY;
21             ... // Count how many times t can be repeated based on s-multiplicities.
22             if (repetitions > 0 && repetitions !== Number.POSITIVE_INFINITY) {
23                 for (let i = 0; i < repetitions; i++) {
24                     res.addRow(row);
25                 }
26             }
27         }
28     }
29     ...
30     return res;
31 }

```

Listing 19: Modification of the `getResult` function in the `Division` class of `RelaX`.

This modification ensures that the division operator behaves consistently under both set and multi-set semantics.

### 3.5 Summary

This chapter presented the design and implementation of the interactive relational algebra learning environment developed in this thesis.

To begin with, a set of technical and pedagogical requirements was established based on the needs of the database course. Then, a state-of-the-art analysis of existing relational algebra systems was conducted to evaluate how well each of them satisfies these requirements.

Based on this analysis, RelaX was selected as the foundation for the implementation due to its Web-based nature, open-source licence, expressive power, and existing student adoption. However, several discrepancies were identified between the behaviour of RelaX and the semantics adopted in this thesis.

To resolve these discrepancies, three types of modifications were made to the implementation of RelaX. First, changes to the syntax layer aligned the notation accepted by the parser with the notation used in the course. Then, a change to the view layer adapted the visual representation of attributes. Finally, changes to the model layer modified the semantic behaviour of multiple operators.

## Chapter 4

# Support for Exercises

This section presents the integrated exercise system of the relational algebra learning tool. The goal of this system is to provide students with structured exercises and automated feedback on their responses. The exercise system introduces mechanisms for defining, managing, and evaluating exercises within the platform. It also integrates with existing RelaX functionalities, allowing students to seamlessly transition from sandbox query execution to guided exercise practice.

### 4.1 State of the Art

Several systems already exist to provide structured query exercises with automated grading and feedback. Tools for both relational algebra and SQL were surveyed as they often share grading mechanisms. The scope and grading approach of each tool are described below.

#### 4.1.1 AsseSQL

AsseSQL [36] is an online tool developed by Julia Coleman Prior in 2002. It is used to assess the SQL query formulation skills of students in a supervised classroom setting.

For each test, the system presents a set of questions for students to answer. For each question, a natural-language description is provided along with the expected result dataset. When a student submits a query, it will be executed against the dataset corresponding to the current exercise. If the query is syntactically correct, its execution result will be displayed to the student. If this result differs from the expected one, a message is displayed to the student, and they can submit another answer after comparing the two results. If they are identical, the answer is marked as correct.

A notable feature of AsseSQL is its implementation of result-set comparison as a feedback mechanism. Rather than simply indicating correctness, the system displays both the student's query output and the expected output. This approach enables learners to identify discrepancies and iteratively refine their queries. In a later paper, Julia Coleman Prior and Raymond Lister theoretically justify this feedback mechanism [37]. They argue that one of the main difficulties in learning SQL is conceptualising and visualising query results. By executing queries, observing their outputs, and comparing their answers with the required results, students can progressively internalise query-formulation skills.

Since both queries are tested only on the sample dataset, the author also raised concerns that students might write solutions that are valid only for the specific results shown in the sample output. To address that issue, they propose extending the tool to run both the reference and student queries on an additional dataset not shown to students.

### 4.1.2 SQLator

SQLator [38] is a Web-based SQL learning workbench developed in 2004 by Shazia Sadiq, Maria Orlowska, Wasim Sadiq, and Joe Lin.

A student is given a sample dataset and a statement describing the query requirements, and is then asked to write an SQL query. Once a query has been written, the student can choose to execute it on the dataset, evaluate it against the requirements, or view a correct solution to the problem. During evaluation, SQLator executes both the student and reference queries on the sample dataset and reports binary feedback: "correct" if and only if the results match, and "incorrect" otherwise.

Despite the limitation of testing both queries on a single sample dataset, the authors report that SQLator achieves at least a 95% successful evaluation rate even in relatively easy test cases. However, the success of this method still depends, in part, on the test dataset instance, as poorly designed instances reduce the accuracy of the correction.

### 4.1.3 ACME

ACME [39] is an e-learning framework that integrates a tool designed to automatically correct relational algebra queries developed in 2007 by Josep Soler, Imma Boada, Ferran Prados, Jordi Poch, and Ramon Fabregat.

A student is randomly assigned a set of problems from a common repository and must enter the relational algebra expressions required to solve them. Once they have entered all the expressions, they can send their solution for correction. A solution is considered correct if the values that its execution returns match the correct solution values stored in the base problem. If the values do not match, the system will indicate an error, and the student will have the opportunity to submit a new solution.

The system considers two types of errors: syntactic errors, which are identified when the student writes an incorrect expression, and result errors, which occur when the expression is syntactically correct but the final result is not the desired one. In the latter case, the system executes the student's solution, displays the result, and indicates where the error occurred.

A notable feature of ACME is its feedback system. Instead of simply reporting the binary correctness of the query, the error message explains the difference between the two results when they do not match. For example, it can display "the relation has a different number of attributes." This mechanism is useful to steer students in the right direction without giving them the full solution directly.

### 4.1.4 ReliQ

ReliQ [23] is an online tutoring and assessment system for teaching relational algebra developed in 2023 by Hasan Jamil, Kallol Naha, and Farjahan Shawon. It is part of a larger database tutoring and assessment system called Project 360.

A student is given a target dataset and a natural language description. They can then use a query editor to attempt to write a relational algebra expression that, when executed, produces results that are semantically consistent with the query described in natural language. If the instructor allows it, students may run queries against the provided sample dataset before submitting their answer. The system translates relational algebra queries into MySQL SQL statements for execution. When a student submits a query, the system compares its result with the reference and marks it correct or incorrect. The comparison works by first converting each row in both tables to a string and concatenating all rows from each table into a single long string. Then, it sorts the characters in each of these long strings and calculates an MD5 hash for each. The underlying assumption is that if the two tables are identical, the resulting hash values will also be identical. This method improves robustness against changes in column names or column order.

### 4.1.5 Neural-network SQL Grading

In 2024, Donald R. Schwartz and Pablo Rivas proposed a machine-learning approach to automatically grade SQL statements using a convolutional neural network with self-attention and parameter-sharing across multiple grading tasks [40]. Their work attempts to address the challenge of assigning partial credit to queries that are partially correct but would receive a score of zero with binary grading approaches.

They proposed a shared neural network architecture consisting of embedding layers, convolutional self-attention, dropout, batch normalisation, and a bottleneck representation layer to learn features from SQL queries. The learned representation is then reused by three specialised output models trained to predict the binary correctness of the query, the grader's remark, and the numerical grade assigned to the answer. The system was trained using approximately 2700 SQL queries collected from student submissions over multiple years of their database courses.

While the approach illustrates that machine learning techniques can be applied to automated SQL grading, the reported results remain relatively limited. The model obtained an average AUC<sup>1</sup> of approximately 0.64, indicating performance better than random classification yet insufficient to outperform more deterministic grading approaches. Additionally, machine-learning approaches require large labelled datasets for training and provide less transparent grading criteria than rule-based approaches.

### 4.1.6 Summary and Identified Limitations

Most grading systems for relational algebra or SQL rely on dynamic analysis based on result equivalence. In these systems, both the student and reference queries are executed on a sample dataset, and their results are compared to determine correctness. This approach can identify queries that are definitely not equivalent, but it cannot guarantee that they are equivalent. As a result, queries that produce the expected output for the sample dataset while remaining semantically incorrect may still be accepted.

This survey also highlighted a disparity between SQL and relational algebra exercise systems. SQL grading tools are more prevalent and generally more mature, often offering more features and a wider range of grading strategies. In contrast, relational algebra systems remain comparatively limited.

Recent research additionally shows growing interest in machine-learning-based grading methods and tutoring systems. However, current machine-learning approaches for SQL grading still display limited reliability and require large labelled datasets for training. Furthermore, no completed studies that specifically applied large language models to relational algebra tutoring were identified in this survey.

These observations motivated the design choices of the implementation presented in this thesis. In particular, the proposed approach follows the widely adopted principle of result-based equivalence testing.

## 4.2 Exercise System Design

Based on the previous observations, the exercise system was designed to provide a practical environment for learning relational algebra. In particular, the system aims to support flexible exercise distribution, structured student interaction, and automated verification with progressively detailed feedback. This section presents the main concepts and design choices behind the exercise system.

### 4.2.1 Definition of an Exercise

In the context of this relational algebra learning tool, an exercise is defined as a structure composed of three main elements. First, a sample dataset that is provided by referencing an existing dataset, accessible either through a local file or via an external link, in this case, a Gist file.<sup>2</sup> This dataset serves as the basis for executing queries. Second, a natural-language description specifying the task the student is expected to

---

<sup>1</sup>The AUC (Area Under the Curve) is a performance metric used to evaluate the ability of a classification model.

<sup>2</sup>A Gist file is a publicly accessible online text file hosted on GitHub Gist.

perform on this dataset. This description corresponds to the type of instruction typically given in practical sessions and is presented to the student. Third, a reference relational algebra query that represents the expected solution to the exercise. This expression is not shown to the student and is instead used internally for verification purposes.

In addition to those three required elements, an exercise may optionally include a reference to a test battery made up of one or more datasets, which will be hidden from the students. These datasets will be used during an additional verification step to improve the robustness of correctness checking by evaluating students' queries on unseen data.

For example, an exercise could reference a local dataset containing two relations, `Student(ID, Name)` and `Enrolment(ID, Course)`. The exercise description presented to students could be: "Retrieve the names of all students enrolled in the Database course." And the corresponding hidden reference query could then be: " $\pi$  Name (Student  $\bowtie$   $\sigma$  Course='Database' (Enrolment))." Additionally, the exercise could include hidden test datasets containing different instances of the `Student` and `Enrolment` relations to verify that student submissions remain correct on unseen data.

### 4.2.2 Distribution of Exercises

There are multiple ways to make exercises accessible to students, each with different implications for the usability and maintainability of the system. The first option would be to provide exercises via predefined lists hosted directly on the platform. This approach is straightforward for students but requires modifying the codebase of the tool whenever new exercises need to be added. Alternatively, exercises could be loaded from external files. This approach improves flexibility for instructors, but introduces the need for a separate distribution system, external to the tool, to provide these files to students. A more flexible solution would be to allow instructors to upload exercises directly to the platform, making them immediately accessible to students. However, while this approach offers the most seamless workflow for both instructors and students, it would require implementing access control mechanisms on the platform to prevent unauthorised modifications.

For this tool, a hybrid approach is adopted, in which exercises can be loaded from both predefined local lists and external files, allowing instructors to easily add new content without modifying the tool, while still providing students with a list of base exercises. To support the distribution of external exercise files, the files are hosted on GitHub Gist and accessed through their corresponding Gist identifier.

### 4.2.3 Exercise Mode

In addition to the standard sandbox usage of the tool, the system introduces an exercise mode. In standard mode, students can freely select datasets and experiment with relational algebra queries without constraints. In contrast, exercise mode is activated when a specific exercise is selected and provides a more structured environment. When exercise mode is enabled, the system associates the selected exercise with the working environment: the exercise's name and description are displayed, and the active calculator is set to match the exercise type. To ensure consistency between the exercise definition and the query language used, the student cannot switch to a different query language while solving an exercise. In this mode, additional functionalities become available, most notably the ability to verify a solution against the reference query. In contrast, when no exercise is active, these functionalities are unavailable, and the interface reverts to a simplified configuration focused on query execution.

### 4.2.4 Verification Process

The verification process is designed to assess whether a student's relational algebra expression is correct for the current exercise and to provide feedback. The system performs a sequence of verification steps that combine structural analysis and result-based evaluation. This approach allows simple cases to be handled efficiently while still supporting more detailed validation.

The system first compares the submitted query and the reference query associated with the exercise. If both expressions are textually identical, the query is considered correct immediately and no additional verification is needed. If they differ, the system compares the structures of both expressions to determine whether they are equivalent. If both structures are equivalent, the query is also considered correct without requiring further evaluation. In both cases, if the queries match, the system can confidently verify the correctness of the student's answer. However, if the queries do not match, that alone does not guarantee that the answer is incorrect, as two expressions may have different structures while still being semantically equivalent.

When structural equivalence cannot be established, the system checks whether both queries produce the same result on the exercise sample dataset. If the two results differ, the query is considered incorrect and feedback is generated for the student. However, a known limitation of result-based verification is the possibility of coincidental correctness, where an incorrect query produces the expected result on the sample dataset. To reduce this risk, the verification process can optionally rely on hidden datasets associated with an exercise. If the two results differ on any test dataset, the student's query is considered incorrect. By evaluating queries over several datasets, potentially including corner cases defined by the instructor, the probability of incorrectly accepting an invalid query is significantly reduced.

If the student's query successfully passes all verification stages, it is considered most likely correct. Although this result-based approach does not formally prove semantic equivalence, it provides a reasonable compromise between correctness, implementation feasibility, and pedagogical usefulness.

## Feedbacks

The verification interface is designed to provide feedback that is adapted to the type of error encountered.

If the student query contains a syntax or semantic error, the system displays the error message linked to the execution. In this situation, no further results are displayed since the query cannot be evaluated.

If the query is valid but fails a verification step, the system provides several types of feedback to help the student understand and correct their mistake, as shown in Figure 4.1.

**(a) Results Comparison**

Execution Verification

Solution rejected: Results do not match on the reference dataset.  
Hide expected resulting relation

The relation resulting from your query:

| NOM       | PRENOM   |
|-----------|----------|
| 'Dupont'  | 'Alice'  |
| 'Martin'  | 'Bob'    |
| 'Durand'  | 'Claire' |
| 'Smith'   | 'David'  |
| 'Bernard' | 'Emma'   |

The relation resulting from the reference query:

| NOM       | PRENOM |
|-----------|--------|
| 'Bernard' | 'Emma' |

**(b) Links to theoretical explanation**

Hide the theory items that are relevant to the exercise reference solution

- projection
- innerjoin
- duplicate elimination
- selection

**(c) Reference query**

Hide the exercise reference query

```
X ← Δ(πID(σDATE_FIN=null(contract) × σTACHE='Valider du code'(tache)))
πNOM,PRENOM(personne × X)
```

Replace the current query with the reference query

Figure 4.1: Feedback displayed after a failed verification in the verification tab of the calculator page.

First, the system displays the results of the student query and the results of the reference query for the dataset over which verification failed. Following the reasoning proposed in [37], showing the difference between the obtained and expected results can help students identify what is wrong with their query.

In addition, the system also displays the expected operators in the solution, along with links to their corresponding sections of the theoretical explanation page. An example of the theoretical section for the

Projection and Selection operator is shown in Figure 4.2. This provides students with guidance without immediately revealing the full answer.

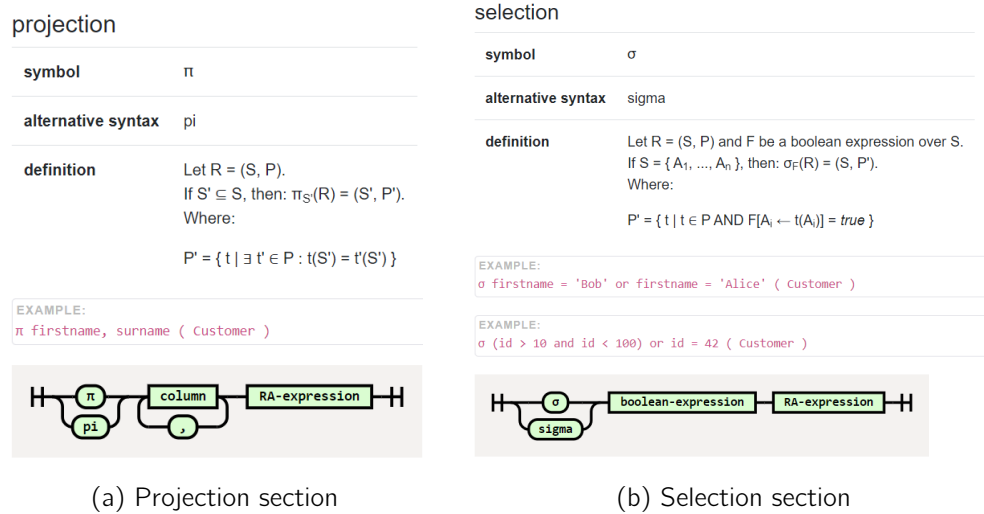


Figure 4.2: Projection and Selection section of the theory page.

Finally, if they feel unable to solve the exercise, the student may choose to reveal the reference query itself. Since the tool is intended for self-practice rather than formal assessment, access to the solution is acceptable.

To preserve the educational value of the exercise, users need to reveal these feedback elements one by one, allowing students to request increasingly explicit assistance only when necessary.

If the query passes all verification stages, the system informs the student that the solution has been validated.

#### 4.2.5 User Interaction Workflow

The exercise system is designed to provide a simple and intuitive interaction workflow for students. The main entry point for interacting with exercises is a dedicated interface element, as shown in Figure 4.3, which allows students to select and load an exercise in the working environment. Students can select an exercise from the list of loaded exercises or load a new set of exercises by providing a Gist identifier. After loading, the exercises found in the Gist file referenced by the identifier will appear in the list of exercises available for selection. These exercises are grouped according to their maintainer and sorted alphabetically.

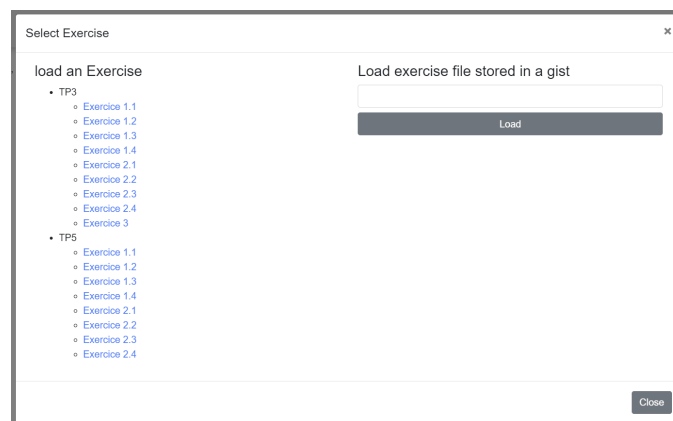


Figure 4.3: Exercise selection modal interface.

Once a student selects an exercise, it becomes the current working exercise. The interface then displays that exercise name and description on the main calculator interface, as shown in 4.4. Concurrently, the sample dataset referenced in the exercise definition is loaded as the current working dataset. At this stage, the student can write a relational algebra expression in the query editor and interact with the system using the standard execution and the new verification functionalities.

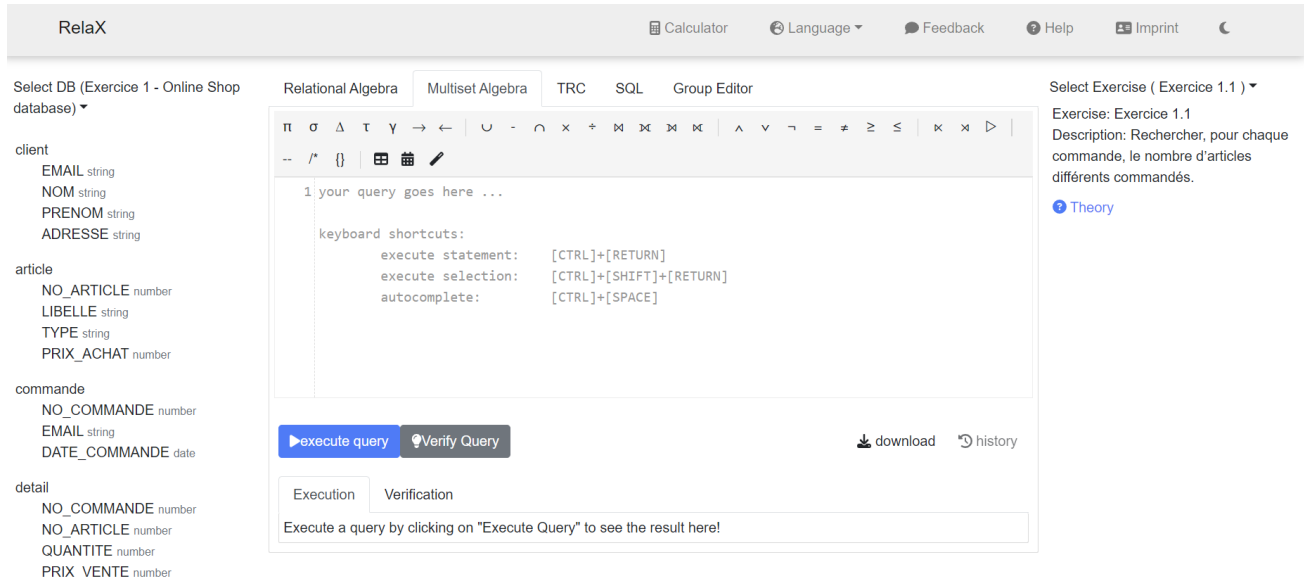


Figure 4.4: Main calculator page when an exercise is selected.

## 4.2.6 Exercise Maker

A dedicated helper tool, shown in Figure 4.5, is available to help instructors create and export new sets of exercises without having to learn how to format an exercise file.

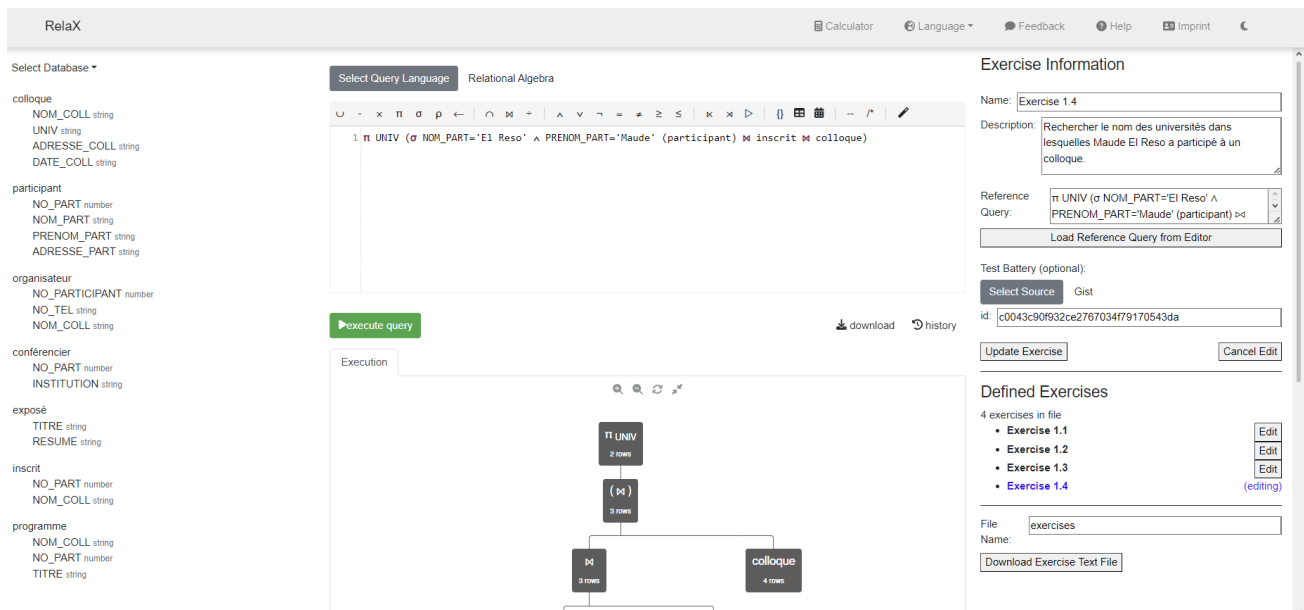


Figure 4.5: Exercise Maker interface.

To define a new exercise, the user first selects the associated sample dataset and exercise type, relational algebra or multi-set algebra. The user must then provide the name and natural-language description of the exercise.

In addition, a reference query must be defined using the same query editors as those available in the main calculator interface. This editor allows users to run and test the query on the selected dataset before adding it to the exercise. When a query is loaded into the exercise form, it is executed automatically to verify that it can be evaluated successfully. If the execution fails, a warning toast is displayed, and the query is not loaded.

The exercise form also includes an optional section to associate a test battery with the exercise. In this section, the user can select the type of source from which the test battery needs to be fetched, either locally or from a Gist repository, and provide the corresponding identifier.

Once all fields have been provided, the user can add the exercise to the current collection. If a Gist identifier was entered, the system attempts to verify that the file exists. If this verification fails, a warning toast is displayed, and the exercise stays in the editing state without being added to the collection. This process can be repeated to define multiple exercises within the same file.

All exercises in the current collection are displayed in a list within the interface, allowing the user to keep track of the exercises that will be included in the exported file. Furthermore, each exercise within that list comes with an "edit" button that reloads its information into the editor, enabling the user to modify it.

After all of the desired exercises have been added, the user can specify a file name and export the resulting file. This file can subsequently be uploaded to a Gist repository, allowing it to be shared and later loaded into the system.

Using this helper tool also helps detect malformed exercises before they are shared with students, as it performs checks on the reference query and the fields of each exercise added to the list to be downloaded.

## 4.3 Exercise System Implementation

This section presents the implementation of the exercise system as described in the previous section into the existing Relax platform.

### 4.3.1 Exercise Model Implementation

In the implementation, exercises are represented using an `Exercise` type data structure, as shown in listing 20, which encapsulates all information required to display and evaluate an exercise. Each exercise includes a `name` and a `description`, which are displayed to the student, as well as a `reference` field containing the reference relational algebra query used for verification. The `datasetPath` field specifies the location of the sample dataset on which the exercise is evaluated. The `verificationGroups` field may contain information about additional datasets used during verification, enabling the system to evaluate student queries against multiple test cases. The `type` field allows the system to distinguish between different types of exercises, such as relational algebra, multi-set algebra, or SQL. This enables the reuse of the same infrastructure for different types of exercise.

The metadata associated with each exercise are divided into two structures, as shown in Listing 21. The `ExerciseInfo` type stores information related to the location of the exercise, including its source file type, the file identifier, and the position of the exercise within the file. The `SourceInfo` type stores optional descriptive metadata, such as information about the author of the exercise, a reference URL, and the last modification date.

Overall, this structure directly implements the definition introduced in Section 4.2.1 and is designed to be modular and extensible to allow future extensions, such as additional exercise types or more advanced verification steps.

```
export type Exercise = {
  name: string,
  description: string,
  reference: string,
  datasetPath: string,
  verificationGroups: VerificationGroups,
  type: ExerciseType,

  exerciseInfo: ExerciseInfo,
  sourceInfo: SourceInfo,
};

export type ExerciseType = 'sql'
| 'relational_algebra' | 'multiset_algebra';

export type VerificationGroups = {
  source: string,
  id: string,

  groups: Group[] | undefined,
};
```

Listing 20: Exercise data structure.

```
export type ExerciseInfo = {
  source: ExerciseSourceType,
  id: string,
  filename: string,
  index: number,

  maintainer: string,
};

export type ExerciseSourceType
= 'maker' | 'gist' | 'local';

export type SourceInfo = {
  author?: string,
  authorUrl?: string,

  url?: string,
  lastModified?: Date,
};
```

Listing 21: Exercise metadata data structure.

### 4.3.2 Exercise Parsing and Validation

An exercise is defined using a lightweight text format, shown in Listing 22. The order of the fields does not matter. If the `tests` field is not included or left empty, no test battery will be considered during verification. And if the `type` field is omitted or empty, the default exercise type is standard relational algebra. Multiple exercises can be located in the same file, separated by one or more empty lines. This lightweight formatting approach was preferred over more formal formats, such as JSON, to maintain consistency with the way datasets are defined in the original Relax system.

```
exercise: Name of the exercise
description: Description of the exercise
reference: Relational Algebra expression representing the expected solution
dataset: Reference to a dataset
tests: Link to a file containing at least one group of relations.
type: Type of the exercise (relational_algebra, multiset_algebra, or sql)
```

Listing 22: Template for the definition of an exercise.

Exercise definitions provided via local files or external sources are converted into the internal data structures used by the system by a dedicated parser that converts the textual representation of one or more exercises into a list of `Exercise` instances.

This mechanism is implemented in the function `parseExercisesFromDefinition`. The input consists of a raw text file and metadata describing the source of the file, and the output is the corresponding list of parsed exercises. Each exercise is processed line by line, where each line is interpreted using a simple key-value format of the form `key: value`. To support multi-line values, the parser keeps track

of the most recently encountered key. If a line does not match the key–value pattern, it is treated as a continuation of the previous field. This mechanism is used for fields such as `description` and `reference`, which may span multiple lines.

Once all lines of an exercise have been processed, the collected values are validated. If the `type` field is not specified or contains an unsupported value, the default exercise type `relational_algebra` is used. Similarly, optional fields, such as `tests`, are initialised with empty values when not provided. If any of the required fields, `exercise`, `description`, `reference`, and `dataset`, are omitted or left empty before reaching a new line, the whole exercise will be skipped, and the parsing will move on to the next definition.

The `tests` field, when present, is interpreted as a reference to a resource containing additional datasets. This reference is split into a source identifier and a resource identifier, which are later used by the loading system to retrieve the verification datasets

After parsing and validation, each exercise is instantiated as an `Exercise` object. Metadata describing the origin of the exercise, such as its source type, file identifier, and position within the file, are attached to the object. This ensures that exercises can be uniquely identified and managed within the system.

The parsing process is designed to be tolerant of minor formatting issues. Unknown keys and malformed lines do not cause the parsing process to fail, as long as the valid parts form a valid exercise. Additionally, even a malformed exercise in the file does not affect the parsing of the remaining exercises. However, critical errors encountered during file loading, such as inaccessible external resources or invalid file formats, result in the full rejection of the parsing operation.

### 4.3.3 Test Battery Parsing and Alignment

A test dataset can be defined in a text file using the same structure as regular datasets. An example of such a structure is shown in Listing 23. The order of the information fields does not matter, and only the `group` field is mandatory.

Contrary to the way exercises are defined, the information fields of a dataset can be separated by one or more empty lines without being considered part of a new dataset. Multiple datasets can be located in the same file, separated by their relation definitions. All datasets in a file are considered part of the same test battery and will all be tested when the file is linked to an exercise.

```
group: Test1
description: Test for missing NOM_PART or PRENOM_PART in the selection condition.

participant = {
  NO_PART:number, NOM_PART:string, PRENOM_PART:string, ADRESSE_PART:string

  1, 'Kim', 'Ricky', 'Liège'
  2, 'Kim', 'Gyuvín', 'Liège'
  3, 'Shen', 'Ricky', 'Liège'
}

programme = {
  NOM_COLL:string, NO_PART:number, TITRE:string

  'BD2024', 1, 'Relational Algebra Basics'
  'Test', 2, 'Test Kim'
  'Test', 3, 'Test Ricky'
}
```

Listing 23: Example of a test dataset for an exercise whose reference query is  $\pi$  TITRE ( $\sigma$  NOM\_PART='Kim'  $\wedge$  PRENOM\_PART='Ricky' (participant)  $\bowtie$  programme).

Since they reuse the structure of regular datasets, test batteries are parsed by the original RelaX dataset parsing system. However, because a test dataset may define only a subset of the relations available in the exercise reference dataset, newly parsed test datasets must first be aligned with the schema expected by the exercise execution environment. This process is implemented by the `alignGroupToDataset` function.

The function iterates through all relations of the reference dataset and attempts to find a relation with the same name in the test dataset. When a match is found, the equality of their schemas is tested. If the schemas are compatible, the relation from the test dataset is kept as is for the aligned dataset. Otherwise, it is ignored, and a console warning is emitted. If such a match does not exist or if the matching relation was ignored, an empty relation is created using the schema of the corresponding reference relation. This guarantees that all relations students could want to use in their query are present during the query evaluation, even if they are omitted from the test dataset definition or defined with an incompatible schema. After the alignment step, the function checks whether the test dataset contains additional relations that are not present in the reference dataset. Such relations are appended unchanged to the aligned dataset.

This mechanism ensures that all datasets in a test battery remain compatible with the exercise execution environment while allowing instructors to define only the relations they find pertinent to the potential error they wish to test.

#### 4.3.4 Exercise Loading Implementation

The loading of new exercises, as described in Section 4.2.2, is implemented using a state management architecture built with Redux and Redux-Saga. This architecture is inherited from the original RelaX system, which already provides a structured state tree and asynchronous handling mechanisms for loading and managing groups of relations. In this work, this existing architecture is extended to introduce exercise management.

A branch of the global state tree, as shown in Listing 24, stores all loaded exercises in a map indexed by a unique string identifier derived from their location metadata. In addition, this branch also maintains a reference to the currently selected exercise, or the null value if there is none.

```
export type State = {  
  exercises: Immutable.Map<string, Exercise>,  
  current: { exercise: Exercise } | null,  
};
```

Listing 24: Exercise branch of the global Redux state tree

The loading of a file containing one or multiple exercises is initiated through the dispatch of an `EXERCISES_LOAD_REQUEST` action. This action specifies the source of the file, its identifier to retrieve it from that source, its maintainer for grouping purposes, and an optional parameter to indicate which exercise should be set as the current exercise after loading. Using a saga allows this process to be handled asynchronously. Upon receiving a load request, the system first checks whether the requested file source and identifier have already been loaded to avoid redundant loading. If the exercises are already present in the state, the system updates only the currently selected exercise without reloading the data. If the file information is new to the system, the exercises described in the file are retrieved using a dedicated loading function. Once retrieved, each exercise is processed individually. If the exercise definition includes a `tests` field, the corresponding verification groups are loaded through additional asynchronous requests and are aligned to the reference dataset. After loading, the exercises are stored in the global state through an `EXERCISES_LOAD_SUCCESS` action.

The selection of the current exercise is handled by the dedicated `EXERCISE_SET_CURRENT` action. This action starts by checking whether the specified exercise is in the set of all exercises. If the exercise is not

found, an error occurs. If the exercise is found, it updates the state and triggers the loading of the dataset associated with the new current exercise.

### 4.3.5 Exercise Mode Implementation

The distinction between the standard sandbox and guided exercise usage of the tool, as described in Section 4.2.3, is implemented through an `exerciseMode` flag stored in the global session state. This flag is managed within the Redux architecture and updated through the `SET_EXERCISE_MODE` action. When the user selects a dataset, the system operates in standard mode, and the `exerciseMode` flag is set to `false`. In contrast, when an exercise is selected, the flag is set to `true` to indicate that the system should operate in exercise mode.

This state variable directly influences several aspects of the tool behaviour. First, it controls the configuration of the calculator user interface. When exercise mode is enabled, the active working tab is automatically set according to the exercise type, relational algebra, multi-set algebra, or SQL, and switching between tabs is disabled to prevent inconsistencies between the exercise definition and the query language used by the student. Second, the flag also controls the availability of the user interface. In exercise mode, additional components, such as the verification button and the tab that displays the verification results, are visible. When the tool operates in standard mode, these elements are hidden from the interface. Additionally, when no exercise is selected, the interface hides the details of the previously selected exercise and displays only the button required to select a new exercise.

This approach provides a simple and effective mechanism for controlling the behaviour of the interface while maintaining a clear separation between different usage contexts.

### 4.3.6 Verification Process Implementation

The verification process, as described in Section 4.2.4, is integrated into the base editor component via the `verify` function shown in part in Listing 26. If this function is executed outside of exercise mode, it throws an error as it cannot be triggered when no exercise is selected. The overall verification workflow is illustrated in Figure 4.6.

The function first performs an exact string comparison between the student and reference queries, followed by a structural comparison of their corresponding ASTs using an equality method introduced in the `RANode` class as part of this work. This method was implemented for every relational algebra node class to determine whether two ASTs are structurally equivalent. If either check succeeds, the solution is accepted and the function returns.

When AST equivalence cannot be established, both queries are executed on the sample dataset associated with the exercise, and the resulting relations are compared using an updated version of the equality method of the `Table` class.

This modified method, shown in Listing 25, first verifies schema equality before creating copies of both relations to avoid modifying the original results. Since semantically equivalent relations may differ in attribute or tuple ordering, the second relation is remapped to the schema ordering of the first. Both relations are then sorted to obtain a deterministic tuple ordering, except when the reference query contains a sort operation, in which case the original tuple ordering is preserved because it is semantically relevant. After this normalisation step, the method compares the number of tuples in both relations and performs a tuple-by-tuple comparison. The two relations are considered equivalent only if every tuple matches exactly.

If the results are equivalent and if the exercise includes a battery of tests, the same comparison procedure is repeated for each additional dataset.

If a mismatch is detected, the process stops, and the query is rejected. Otherwise, the query is accepted after all available tests have been passed successfully.

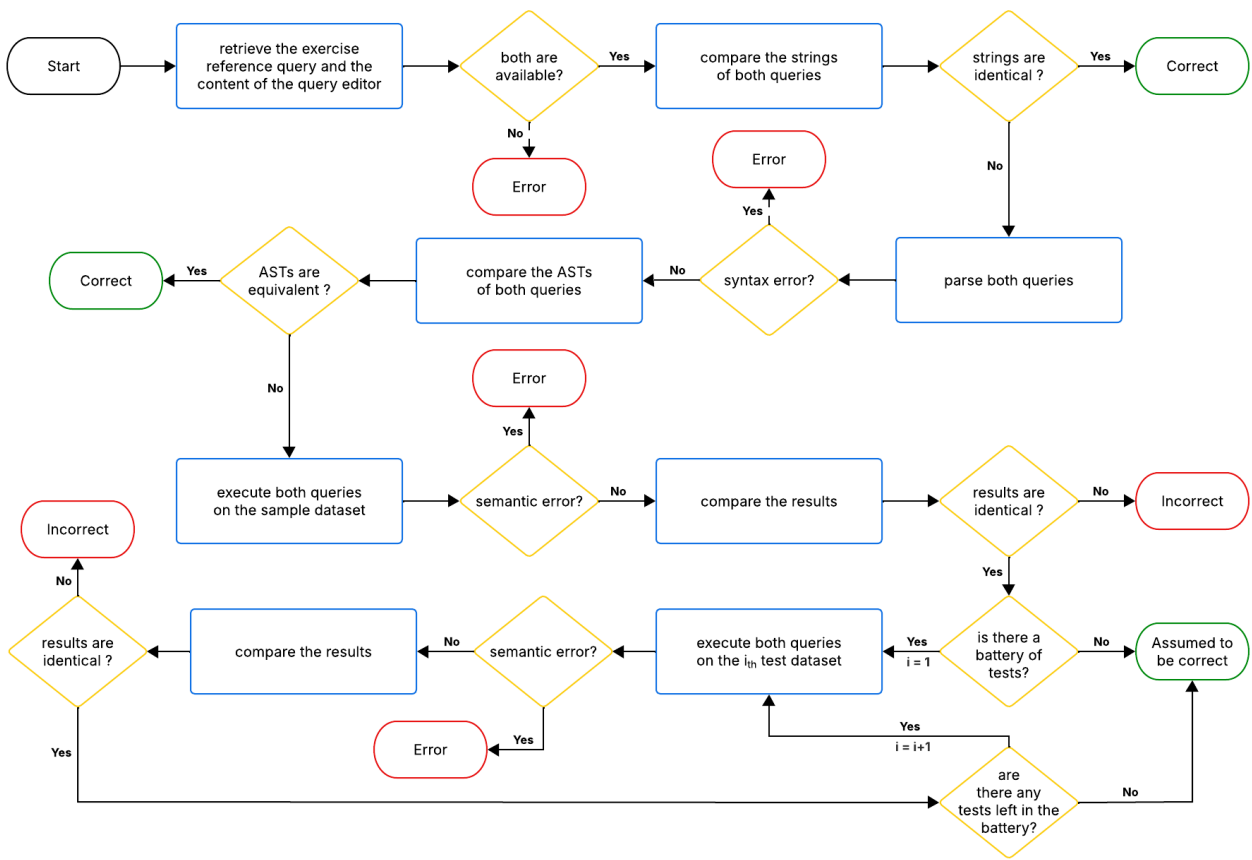


Figure 4.6: Flow chart of the verification workflow performed by the `verify` function.

```

1 equals(table: Table, sortIsRelevant: boolean = false) {
2     if (table instanceof Table === false) { throw new Error('can not compare'); }
3
4     if (this._schema.equals(table._schema) === false) { return false; }
5
6     const tableA = this.copy();
7     const tableB = new Table();
8
9     tableB.setSchema(this.getSchema().copy());
10    tableB.addRow(table.getRowsMappedToSchema(this.getSchema()));
11
12    if(!sortIsRelevant) { // if the ordering of the rows is irrelevant for equivalence
13        tableA.sort();
14        tableB.sort();
15    }
16
17    if (tableA._rows.length !== tableB._rows.length) { return false; }
18    for (let i = 0; i < tableA._rows.length; i++) { // for each row
19        if (Table.rowEqualsRow(tableA._rows[i], tableB._rows[i]) === false) {
20            return false;
21        }
22    }
23
24    return true;
25 }

```

Listing 25: Modified implementation of the equals function located in the Table class of Relax.

```

1  async verify() {
2      if(!this.props.exerciseMode){ throw new Error(...); }
3      ...
4      this.setState({
5          activeTab: 'verify'
6          verifyResult: (<div className="spinner">...</div>),
7          ... // Reset feedback-related state variables
8      }, () => {
9          ...
10         if(query === referenceQuery) { // Query strings
11             ... // Set feedback-related state variables
12             return true;
13         }
14
15         try {
16             ... // Parse both queries
17             if(execResult.props.root.equals(referenceResult.props.root)) { // ASTs
18                 ... // Set feedback-related state variables
19                 return true;
20             }
21
22             ... // Execute both queries over the sample dataset
23             const sortIsRelevant = referenceResult.props.root.find(sort) !== undefined;
24
25             if(!queryResult.equals(referenceQueryResult, sortIsRelevant)) { // Results
26                 ... // Set feedback-related state variables
27                 return false;
28             }
29
30             if(the exercise has any test dataset){
31                 for(each test dataset) {
32                     ... // Parse and execute both queries
33                     if(!queryTestResult.equals(referenceQueryTestResult, sortIsRelevant)) {
34                         ... // Set feedback-related state variables
35                         return false;
36                     }
37                 }
38             }
39
40             ... // Set feedback-related state variables for when all tests passed
41             return true;
42         } catch (e) {...}
43     });
44 }

```

Listing 26: Partial implementation of the verify function located in the EditorBase component.

## Feedbacks

The feedback from the last verification is displayed in the verification tab of the base editor component. The visibility and content of the feedback elements are controlled by the component state, which is dynamically updated during execution of the `verify` function.

Before starting the verification process, the function resets all feedback-related state variables, switches to the verification tab, and displays a loading spinner to indicate that the query is being processed. When verification is complete, the spinner is replaced with a verification message indicating whether the query was accepted or rejected and at which step the process terminated.

If the query contains a syntax or semantic error, an error message is displayed using the same error mechanism as standard query execution. In this situation, the verification process terminates, and no exercise feedback is displayed since the query result cannot be computed.

When a result-based verification fails, the function stores both the student and reference query results in the component state. The verification interface then allows the student to reveal these relations. Both results are displayed using the same paginated table component as the standard execution results.

The verification tab also provides the student with theoretical hints associated with the reference solution. To provide those hints, the AST of the reference query is stored in the component state during verification. Each relational algebra node contains an identifier that corresponds to a theory section in the theoretical documentation of the tool. When the student chooses to reveal the theory hints, the relevant identifiers are extracted from the reference AST, and a list of hyperlinks pointing to the associated sections of the help page is generated. For example, a reference query containing a selection followed by a projection will produce hyperlinks to the corresponding “Selection” and “Projection” sections.

Finally, the component allows the student to reveal the reference query associated with the exercise. It may additionally be copied into the query editor through a dedicated button, which replaces the current editor content with the reference query.

Independent toggle buttons control the visibility of each feedback element, so students can progressively reveal increasingly explicit hints rather than being forced to view the full solution immediately.

### 4.3.7 User Interface Integration

The user interface integrates the exercise system into the existing RelaX frontend using dedicated React components that connect to the global Redux state.

Selecting an exercise is initiated by a button in the interface, which opens a modal window displaying the list of available exercises. This list is generated dynamically based on the information stored in the global state and groups exercises by their maintainer.

Each exercise in that list is associated with a navigation link that encodes its source type, identifier, filename, and index. When a student selects an exercise, this navigation action triggers the loading process by dispatching an `EXERCISES_LOAD_REQUEST` action.

Furthermore, upon selection, a callback function is invoked to close the modal and update the active tab based on the exercise type. This ensures that the interface remains synchronised with the selected exercise.

In addition to selecting from already loaded exercises, the same modal also provides a mechanism to load exercises from external sources. This is achieved through an input field where students can specify an external Gist identifier. When a student clicks on the Load button associated with the input field, the same loading process as for local exercises is triggered.

Once an exercise is loaded, its name and description are displayed on the main calculator interface, its corresponding dataset is automatically selected, and the verification button becomes visible.

### 4.3.8 Exercise Maker Implementation

The exercise maker, as described in Section 4.2.6, relies on a local React state to manage both the list of complete exercises and the fields of the exercise being constructed.

The dataset selection interface reuses existing components of the system, ensuring consistency with the rest of the application. Similarly, the reference query is defined using the same query editors as those provided in the calculator. Depending on the selected exercise type, either the relational algebra or multi-set algebra editor is instantiated with the selected dataset. This guarantees that reference queries are validated in the same environment in which students will later execute them when solving the exercises. Before a query can be added as the reference query for an exercise, the system confirms that it is non-empty and valid for the associated dataset.

Additionally, before an exercise is added to the file, the system ensures that all required fields are provided. If the user selected Gist as the source for the test battery and entered a Gist identifier, the system also verifies that the resource exists and can be fetched. In case of invalid input or Gist identifier, feedback is provided to the user via toast notifications.

Finally, the collection of exercises stored in the local state are displayed to the user in a list, each with an update button. When the user clicks on an exercise's update button, its information is loaded back as the fields of the exercise to be constructed and can be modified in the form.

Lastly, when the user clicks on the download button after entering a filename, the set of exercises is encoded in the structured text format described above and downloaded through the user's web browser.

## 4.4 Summary

This chapter presented the design and implementation of the exercise and verification system integrated into the relational algebra learning tool.

It first reviewed existing feedback and grading systems for relational algebra and SQL exercises. This analysis highlighted common approaches, such as query comparison based on the resulting relations, while also identifying limitations related to feedback quality and the reliance on single test datasets. These observations motivated the design decisions adopted for the system developed in this thesis.

The chapter then described both the design and implementation of the proposed solution. This included the definition and format of an exercise, the loading of new exercises onto the platform, the verification process, the automated feedback system, and the creation of new sets of exercises.

## Chapter 5

# Evaluation of the tool

This chapter presents the evaluation of the relational algebra learning tool conducted with students of the introductory database course INFO0009. The evaluation focused on the usability and pedagogical usefulness of the tool as perceived by students.

### 5.1 Evaluation Methodology

The evaluation followed a formative approach, as is common in educational technology research [41]. Formative evaluations aim to identify usability problems, understand learner perception, and guide iterative improvement rather than proving conclusive learning outcomes, as is the case in summative studies.

The study combined quantitative and qualitative data collection. Quantitative data were obtained through a post-use questionnaire including custom Likert-scale items and a standardised usability survey, while qualitative insights were gathered through open-ended responses and direct student interactions during the supervised exercise sessions. This strategy aligns with previous studies in this domain, where post-use questionnaires are the predominant usability evaluation method [41, 42].

### 5.2 Participation

The evaluation was conducted during two practical exercise sessions of the introductory database course INFO0009. Each session was conducted under normal classroom conditions, was primarily led by Prof. Debruyne, and used exercises prepared by Fernandez-Lorden Christian, the teaching assistant for the course. During the sessions, students were encouraged to use the tool to complete the exercises.

The first session took place on 16 February 2026 and focused on standard relational algebra exercises. At the end of the session, a questionnaire was administered via an online form. To encourage honest feedback, participation was voluntary and had no impact on the students with respect to the course. The course had 72 students enrolled, 35 completed the attendance sheet, and 18 answered the questionnaire. The questionnaire response rate was therefore 25% of the enrolled students and approximately 51% of the recorded attendees.

The second session, held on 9 March 2026, focused on extended relational algebra exercises. 26 students completed the attendance sheet. However, due to a technical oversight with the online platform used to administer the form, students were not able to fill in the questionnaire again for this session.

### 5.3 Perception

This section presents the evaluation of the students' subjective perception of the tool. It first describes the design of the questionnaire used during the evaluation and then analyses the responses collected from students regarding usability, pedagogical usefulness, and overall satisfaction with the system.

### 5.3.1 Questionnaire Design

To evaluate the perception of the tool by students, a questionnaire combining a standardized usability questionnaire and custom items was designed.

Jie Lu, Matthew Schmidt, Minyoung Lee, and Rui Huang [41] observed that most usability studies for educational technology focus on technical usability and often neglect the pedagogical aspect of usability. To address this limitation, the questionnaire included several additional custom items asking students about their perception of how the tool supports their understanding of relational algebra alongside the standardised usability questions.

This combination allowed for comparability with existing usability studies, as well as evaluation of learning-specific aspects not covered by those general usability instruments.

#### Selection of a Usability Questionnaire

To evaluate the perception of the usability of the tool by students, both the System Usability Scale (SUS) [43] and the IBM usability questionnaires PSSUQ/CSUQ [44,45] (and the CSUQ French adaptation F-CSUQ [46]) were considered.

The System Usability Scale (SUS) [43] is a 10-question survey developed by John Brooke in 1996 to compare subjective usability assessments between different systems. It is made up of 5 positive and 5 negative questions. The alternation between positively and negatively phrased questions is intended to encourage respondents to read each statement carefully before answering. It represents the level of agreement of a user with a 5-point Likert scale, ranging from 1 being "strongly disagree" to 5 being "strongly agree." Every question should be answered, and if a respondent feels like a question does not apply to the system they are evaluating, they are instructed to select the centre of the scale. The SUS yields a number from 0 to 100, which represents the overall assessed usability of the system being surveyed.

The Post-Study System Usability Questionnaire (PSSUQ) [44] is a survey developed by IBM to be used after usability testing scenarios. Its first version consists of 18 questions, its second version has 19, and its third version has 16. It uses a 7-point Likert scale, ranging from 1 being "strongly agree" to 7 being "strongly disagree."

For the second version of the survey, the 19 questions form the OVERALL scale and are then grouped into three sub-scales:

- System Usefulness (SYSUSE) from questions 1 to 8;
- Information Quality (INFOQUAL) from questions 9 to 15;
- Interface Quality (INTERQUAL) from questions 16 to 18.

For this thesis, the PSSUQ structure was preferred over SUS because its sub-scales provide more detailed diagnostic information regarding the specific usability dimensions.

The Computer System Usability Questionnaire (CSUQ) [45] developed by James R. Lewis in 1995 is a slightly revised version of the PSSUQ with nearly identical items and the same subscales. The key difference is that items are rephrased into the present tense and do not refer to any specific testing situation. For this thesis, the wording of the CSUQ was chosen over that of the PSSUQ, as the context in which the tool was tested was not a formal test with specialised instructions.

In 2022, Guillaume Gronier and Laurence Johannsen proposed a French adaptation of the second version of the CSUQ questionnaire called the French-CSUQ (F-CSUQ) [46]. It includes the same 19 items and three sub-scales as the CSUQ translated into French. For this thesis, the F-CSUQ was selected over the original CSUQ due to the language context of the database course. Using F-CSUQ over CSUQ ensured that language would not be a barrier to accurately capturing student feedback.

## Structure of the Questionnaire

Student perception of the tool was evaluated through a questionnaire completed by students after the practical exercise session. The complete list of questions included in the questionnaire and their English translation can be found in Appendix A.1. This questionnaire was divided into three categories:

- presentation and organisation of the tool,
- perceived learning support,
- usability evaluation using the F-CSUQ questionnaire [46].

Lastly, an optional open-ended field allowed students to provide additional comments and suggestions.

### 5.3.2 Perception Results

This subsection presents the results collected during and after the practical sessions.

#### Observed Difficulties During the Session

During the first exercise sessions, students asked several recurring questions and encountered a number of usability difficulties related specifically to the tool, in addition to the usual questions about the exercises themselves. The main issue, as mentioned in Section 3.4.3, was that the students did not understand why the system returned a syntax error when defining string literals using double quotes. This issue was subsequently addressed by allowing string literals to be written using double quotes.

The initial version of the tool also did not yet include feedback other than simple binary correctness. As a result, students often struggled to understand why the results of their queries did not match the expected output. To address this issue, the advanced feedback system detailed in Section 4.2.4 was added between the two exercise sessions.

During the second exercise session, most of the questions asked by students were related to the exercises themselves and not to the tool.

#### Presentation

The responses related to the presentation of the tool are summarised in Figure 5.1.

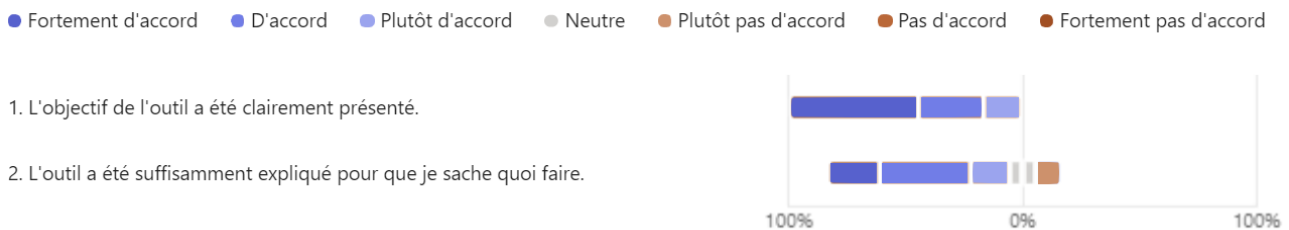


Figure 5.1: Aggregated responses (N=18) to the questions about the presentation of the tool.

The responses to the first question indicate that the students generally considered that the objective of the tool had been clearly presented, as they all (18/18) selected positive agreement levels. This suggests that they were able to quickly understand the role of the tool within the exercise session.

The second question evaluated whether the tool had been sufficiently explained for students to know how to use it. Although most of the responses remained positive (14/18), some neutral (2/18) and moderately negative responses (2/18) emerged, suggesting that providing more explanations about how to use the tool during the session might have helped students become more comfortable with it.

Overall, the results suggest that the presentation of the tool was generally well received. However, they also indicate that some improvements in the explanations of the tool could help new users to become more familiar with the system more easily.

Perceived Learning Support

The aggregated responses regarding the pedagogical usefulness of the tool are presented in Figure 5.2.

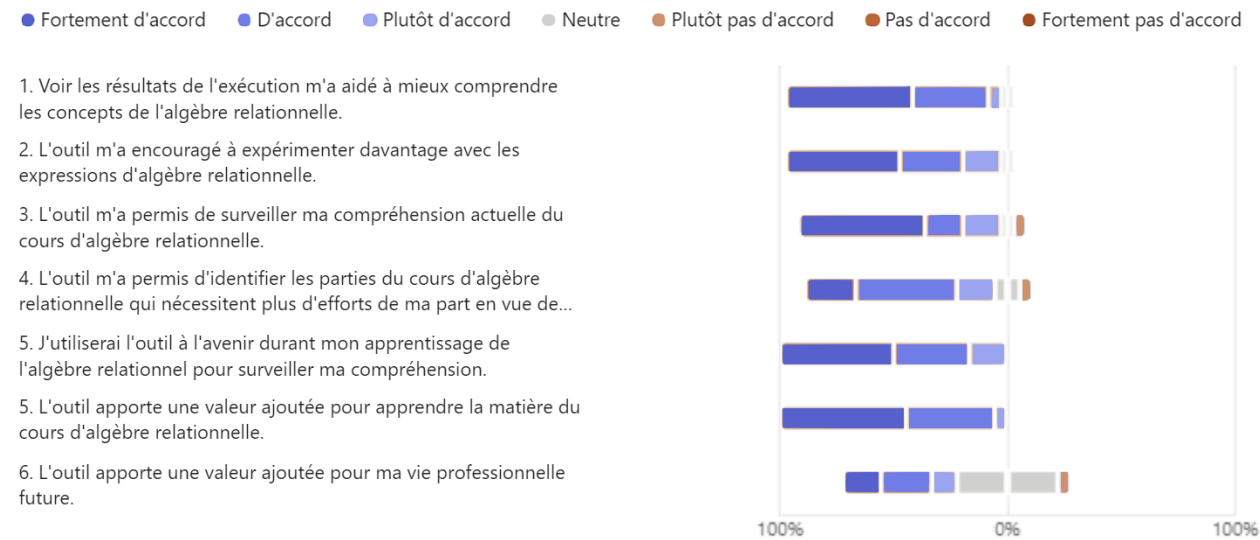


Figure 5.2: Aggregated responses (N=18) to the questions about the pedagogical usefulness of the tool.

Most of the students (17/18) agreed that visualising the results of the execution helped them better understand the concepts of relational algebra. Similarly, a large proportion of students (17/18) reported that the tool encouraged experimentation with relational algebra expressions during the exercise session.

Students also generally perceived the tool as useful for monitoring their own current understanding of the course material (16/18) and identifying the parts that required additional effort in preparation for the exam (15/18). These responses suggest that students generally perceived the verification system as helpful for self-assessment.

Additionally, the responses indicate that students (18/18) were positive about continuing to use the tool in the future as part of their learning process. This suggests that the tool was perceived by students as useful enough to justify using it beyond these supervised exercise sessions.

Finally, students (18/18) agreed that the tool provided added value for learning relational algebra. In contrast, they were substantially more neutral (8/18 neutral and 1/18 moderately negative responses) about its relevance for their future professional life. This result is expected, as the tool was designed primarily with the database course in mind.

Nevertheless, although these results generally indicate a positive perception of the tool as a learning support for relational algebra by students, they alone are not enough to draw conclusions about their actual learning, since this evaluation measures subjective perception rather than objective performance improvements.

F-CSUQ Results

The aggregated responses to the F-CSUQ questionnaire [46] are shown in Figure 5.3 and the complete overview of the responses can be found in Appendix A.2.



Figure 5.3: Aggregated responses (N=18) to the F-CSUQ questionnaire [46].

The F-CSUQ responses (N = 18) yielded an overall mean score of 2.47 (SD = 0.60) on the original 1 to 7 CSUQ scale, where lower values correspond to better perceived usability. Therefore, the results indicate a generally positive perception of the system. The subscale scores were also favourable, with a mean of 2.46 (SD = 0.84) for System Usefulness, 2.71 (SD = 0.89) for Information Quality, and 2.17 (SD = 0.70) for Interface Quality. The complete overview of the CSUQ score obtained for each respondent can be found in Appendix A.3.

To situate these results within the existing literature, the overall score obtained was compared with the results of a large-scale analysis conducted by Prokopia Vlachogianni and Nikolaos Tselios [47] across 42 research papers on educational technology (N = 58 surveys). They reported an average perceived usability score of 72.75 across all 58 surveys and 73.23 across 21 Internet platforms, with the score being

reported on a normalised 0-100 CSUQ/PSSUQ scale. Following the standard linear transformation of CSUQ scores from the original 1-7 scale to a 0-100 scale, the mean usability score obtained in this study corresponds to approximately 75.5/100. This value is slightly above the average usability level reported for educational technology platforms in the surveyed literature, suggesting that the tool achieves a usability level comparable to typical systems evaluated in this domain.

Students (16/18) reported a generally positive opinion of the general usability of the system. Most agreed that the tool was easy to use (18/18), allowed them to complete their work quickly (15/18) and efficiently (16/18), and was easy to learn (15/18). On the other hand, the question about feeling comfortable using the system received the most number (3/18) of negative responses, and the question about becoming productive fast using the system got the most number (6/18) of neutral responses out of the System Usefulness sub-scale. Nonetheless, the majority of the responses to the system usefulness questions were concentrated in the favourable response categories, with relatively few strongly negative evaluations. These results suggest that the workflow of the tool was sufficiently intuitive and useful for short supervised exercise sessions.

The lowest evaluations were observed for the questions related to error messages and error recovery. Although the responses remained globally positive (at least 10/18 positive responses for all questions), these questions showed a higher proportion of neutral or negative responses compared to the other usability categories. This tendency is reflected by the Information Quality subscale, which obtained the highest, i.e. the least favourable, average score among the three subscales.

The students particularly had negative opinions about the error messages given by the system clearly telling them how to fix problems (6/18 negative and 2/18 neutral responses) and about being able to recover easily and quickly after making a mistake (4/18 negative responses). This indicates that the way errors are processed and explained could be improved to more effectively help students when they submit incorrect relational algebra expressions.

The clarity of the documentation provided with this system was also received fairly neutrally (3/18 neutral and 2/18 negative responses). This indicates that additional guidance or more explicit explanations could further improve usability. On the other hand, the information that the system did provide was generally well perceived. Students mostly agreed (17/18 positive responses for all 3 questions) that the information was easy to understand, effective in helping them to do their work, and clearly organised on the screen.

The last sub-scale, Interface Quality, received generally positive response. Most of the students agreed that the interface of the system was pleasant (16/18), that they liked using it (17/18), and that the system provided the functionalities that they expected from such a tool (16/18).

Finally, the overall satisfaction question received positive evaluations (18/18), indicating that students were generally satisfied with the system. However, although the F-CSUQ results indicate good overall usability, the relatively small number of responses limits the extent to which these observations can be generalised to the entire student cohort.

Lastly, it should be noted that this evaluation led to the implementation of the feedback system detailed in Section 4.2.4 and to the addition of a help page containing the theory elements as seen in the course. Consequently, some of the concerns expressed by students may already have been partially addressed in later versions of the tool.

### **Additional Remarks and Suggestions**

The responses to the open-ended question, shown in Figure 5.4, mainly highlighted three recurring themes: requests for additional theoretical support, improvements to the text-editing workflow, and requests for more advanced feedback.

Several students requested additional theoretical support and concrete examples directly integrated into the platform. These remarks are consistent with the neutral responses observed for documentation-related F-CSUQ items. This feedback contributed to the addition of a theoretical help page to the platform.

4. Avez-vous des commentaires, remarques, ou propositions d'amélioration possible ? (optionnel mais fortement apprécié)

8 Réponses

| ID ↑ | Nom | Réponses                                                                                                                                                                                                                                                                                                      |
|------|-----|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 1    |     | non, c'est nickel                                                                                                                                                                                                                                                                                             |
| 2    |     | Une page exemple serait parfait (exemple concret)                                                                                                                                                                                                                                                             |
| 3    |     | c'est très intuitif, surtout avec les différentes données affichées à gauche, cela permet de les retrouver très facilement.                                                                                                                                                                                   |
| 4    |     | Quant on sélectionne un text et que l'on fait la parenthèse (, on aimerai que la parenthèse apparaisse mais cela supprime le texte sélectionné                                                                                                                                                                |
| 5    |     | J'aurais bien aimé pouvoir écrire les produits/sélection/projection/... sans avoir à cliquer avec la souris. Sinon, j'ai bien aimé.                                                                                                                                                                           |
| 6    |     | Ce serait pas mal de rajouter une fonctionnalité permettant d'afficher une solution type du problème pour les moments où on est vraiment perdu. Et si cette fonctionnalité existe déjà, il faudrait la mettre plus en valeur car je ne l'ai pas vue sur le site. A part ceci, le site est vraiment bien fait. |
| 7    |     | J'ai trouvé la documentation, Tuto pas super efficace et claire. Le fait que on est le Tp donné par MR Debruyne et l'outil en même temps est pas optimal                                                                                                                                                      |
| 8    |     | C'est chouette d'apprendre de façon interactive.                                                                                                                                                                                                                                                              |

Figure 5.4: Additional remarks and suggestions given by the students with their names redacted.

Other comments focused on usability improvements for the relational algebra editor. In particular, students requested more efficient text editing interactions, such as automatic insertion of parentheses around selected expressions and direct keyboard input for operators.

Finally, a student requested more advanced guidance mechanisms when they were unable to progress on an exercise. This comment contributed to the implementation of the feedback system described in Section 4.2.4, including the possibility of revealing the reference solution.

## 5.4 Performance

As an MSc student, I am not allowed to be involved in any certificate evaluation activities of students. While I did evaluate perception and participation, the institution does not allow me to look at student's individual performances. Moreover, at the time of writing, the deadline of this thesis' submission was before the INFO0009 and INFO3003 exams, which impedes the possibility for the Professor to cross the data I have gathered with student performances.

## 5.5 Limitations

Several limitations affect the interpretation of the evaluation results.

First, participation in the questionnaire was voluntary and resulted in a relatively small number of responses compared to the total number of enrolled students. Consequently, the collected perceptions

may not fully represent the entire student cohort. Additionally, students who chose to complete the questionnaire may have been more motivated or more engaged with the exercise session than those who did not respond, which may introduce a participation bias in the collected results.

Second, as discussed previously, a large part of the evaluation relied on the self-reported perceptions of students collected through questionnaires. Although such measures are useful for evaluating usability, they do not directly measure actual learning or understanding of relational algebra. As a result, positive perceptions of the tool should not be interpreted as direct evidence of improved academic performance.

Furthermore, the evaluation was conducted during supervised practical exercise sessions. This context may have influenced the way students interacted with the tool, as they could receive assistance immediately when they encountered difficulties. The perceived usability and usefulness of the system by students may therefore differ from those obtained in a fully autonomous learning context.

The tool also evolved during the evaluation period. Several usability improvements were implemented between the two exercise sessions, including the addition of the feedback system. However, the students who had already filled out the questionnaire during the first session could not fill it out again during the second session due to a technical oversight, which prevents the possibility of comparing observations across sessions.

Finally, there was no performance evaluation, as access to official examination results was not possible within the scope of this thesis.

Therefore, this chapter represented an initial validation of the tool rather than a controlled experimental evaluation.

## Chapter 6

# Conclusions

### 6.1 Summary

This thesis aimed to design and implement an interactive platform for relational algebra and extended relational algebra to support autonomous practice, automated assessment, and relational algebra learning for the introductory database course.

More specifically, to provide students with an interactive manner to author relational algebra queries that are in complete accordance with the course material, the thesis first discussed the theoretical background and choices that define the formalism used in the course material and for the tool.

A list of technical and pedagogical requirements was then established to define what it means for an environment to be interactive and aligned with the course material. Next, a state-of-the-art analysis of existing relational algebra learning tools was conducted to evaluate how well each of them satisfies these requirements. Based on this analysis, it was decided that RelaX would be used as a basis for the tool despite its semantic discrepancies with the formalism defined previously. To resolve those discrepancies, modifications were made to the implementation of RelaX.

Subsequently, to build a system that automatically evaluates relational algebra exercises in a way that minimises the risk of false positive evaluation, the thesis examined the grading and feedback mechanisms used in existing relational algebra and SQL automated verification tools, before detailing the design and implementation of the exercise and verification system integrated into the learning environment.

The platform resulting from this work is available at <https://alexiaDonati.github.io/relax-modified/> and its codebase can be found at the following Uliège GitLab repository: <https://gitlab.uliege.be/Alexia.Donati/relax-modified> or the following GitHub repository as a fork of the original RelaX repository: <https://github.com/AlexiaDonati/relax-modified>.

Lastly, to assess how the tool affects student perception, engagement, and learning experience, an evaluation of the usability and pedagogical usefulness of the tool as perceived by students was conducted during two exercise sessions of the introductory database course INFO0009. This evaluation allowed for the identification and implementation of several improvements to the tool. Overall, the results from the evaluation are largely positive and comparable to those of typical systems evaluated in this domain. However, this evaluation had some limitations that restrict the scope of interpretation of those results. Nevertheless, the evaluation suggests that the proposed environment constitutes a promising support tool for relational algebra practice in the database course.

### 6.2 Further Improvements

Although the proposed learning environment satisfies the objectives defined for this thesis, several possible areas of improvement were identified during the development and evaluation of the tool.

In its current form, the tool only supports autonomous practice and does not support formal assessment. Since the reference queries and verification logic are currently available on the client side, sufficiently

motivated students could potentially access the expected solutions even if they were hidden from the interface. A possible extension would therefore consist of introducing a server-side component responsible for storing the reference queries and executing the verification process. Such an architecture would make the platform more suitable for graded assignments and examinations.

The query grading system itself could also be improved. In the current implementation, a query can only be considered correct or incorrect. While this approach simplifies automated evaluation, it does not reflect the existence of partially correct solutions. A future extension could therefore investigate grading strategies capable of assigning partial scores based on the degree of similarity between the student query and the expected solution, the number of correctly implemented operators, or the correctness of intermediate results.

Aside from the verification process, another promising area concerns the addition of an authentication mechanism. Integrating a Single Sign-On (SSO) authentication mechanism would allow students to access the platform using their university accounts, while enabling the tool to track their exercise completion. Such a system would also enable instructors to check the progress of their students on the exercises.

Finally, as seen in Section 4.1.5 with Schwartz and Rivas machine-learning approach to SQL grading [40], machine learning techniques constitute an interesting research direction for future work as they could potentially improve the flexibility of the verification process. In addition, machine learning methods could also be explored for the automatic generation of test batteries for exercises.

# Bibliography

- [1] Edgar F. Codd. A relational model of data for large shared data banks. *Commun. ACM*, 13(6):377–387, June 1970.
- [2] David Maier. *The Theory of Relational Databases*. Computer software engineering series. Computer Science Press, 1983.
- [3] Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of databases*. Addison-Wesley, 1995.
- [4] Edgar F. Codd. Relational completeness of data base sublanguages. *Research Report / RJ / IBM / San Jose, California*, RJ987, 1972.
- [5] Jeffrey D. Ullman. *Principles of database and knowledge-base systems, Vol. I*. Computer Science Press, Inc., 1988.
- [6] Avi Silberschatz, Henry F. Korth, and S. Sudarshan. *Database System Concepts*. McGraw-Hill Education, 2020.
- [7] Michael L. Brodie and Joachim W. Schmidt. Final report of the ansi/x3/sparc dbs-sg relational database task group. *SIGMOD Rec.*, 12(4):1–62, July 1982.
- [8] Edgar F. Codd. Relational database: a practical foundation for productivity. *Commun. ACM*, 25(2):109–117, February 1982.
- [9] Edgar F. Codd. *The relational model for database management: version 2*. Addison-Wesley, 1990.
- [10] Patrick Hall, Peter Hitchcock, and Stephen Todd. An algebra of relations for machine computation. In *Proceedings of the 2nd ACM SIGACT-SIGPLAN symposium on Principles of programming languages - POPL '75*, pages 225–232. ACM Press, 1975.
- [11] Raghu Ramakrishnan and Johannes Gehrke. *Database Management Systems*. McGraw-Hill higher education. McGraw-Hill Education, 2003.
- [12] Hugh Darwen. Why are there no relational dbmss? <https://www.dcs.warwick.ac.uk/~hugh/TTM/Why-Are-There-No-Relational-DBMSs.pdf>, 2015.
- [13] Stephen Todd. The peterlee relational test vehicle: a system overview. *IBM Syst. J.*, 15(4):285–308, December 1976.
- [14] Ramez Elmasri and Sham Navathe. *Fundamentals of database systems*. Addison-Wesley, 2011.
- [15] Hector Garcia-Molina, Jeffrey D. Ullman, and Jennifer Widom. *Database Systems: The Complete Book*. Pearson Prentice Hall, 2002.
- [16] Marco Console, Paolo Guagliardo, and Leonid Libkin. Fragments of bag relational algebra: Expressiveness and certain answers. *Information Systems*, 105:101604, July 2020.

- [17] Paul Grefen and Rolf de By. A multi-set extended relational algebra: a formal approach to a practical issue. In *Proceedings of 1994 IEEE 10th International Conference on Data Engineering*, pages 80 – 88, March 1994.
- [18] Paolo Guagliardo and Leonid Libkin. A formal semantics of SQL queries, its validation, and applications. In *Proceedings of the VLDB Endowment*, volume 11, pages 27–39, 2017.
- [19] Umeshwar Dayal, Nathan Goodman, and Randy H. Katz. An extended relational algebra with control over duplicate elimination. In *Proceedings of the 1st ACM SIGACT-SIGMOD Symposium on Principles of Database Systems*, PODS '82, page 117–123, New York, NY, USA, 1982. Association for Computing Machinery.
- [20] Edgar F. Codd. Extending the database relational model to capture more meaning. *ACM Trans. Database Syst.*, 4(4):397–434, December 1979.
- [21] Giedrius Slivinskas, Christian S. Jensen, and Richard Thomas Snodgrass. Bringing order to query optimization. *ACM SIGMOD Record*, 31(2):5–14, 2002.
- [22] Ashok K. Chandra and Philip M. Merlin. Optimal implementation of conjunctive queries in relational data bases. In *Proceedings of the ninth annual ACM symposium on Theory of computing - STOC '77*, page 77–90, Boulder, Colorado, United States, 1977. ACM Press.
- [23] Hasan Jamil, Kallol Naha, and Farjahan Shawon. An online tutoring and assessment system for teaching relational algebra in database classes. In *International Conference on Web-Based Learning*, pages 62–76, November 2023.
- [24] Suzanne W. Dietrich, Eric Eckert, and Kevin Piscator. Winrdbi: a windows-based relational database educational tool. *Proceedings of the twenty-eighth SIGCSE technical symposium on Computer science education*, 29(1):126–130, March 1997.
- [25] Salvo 'LtWorf' Tomaselli. Relational: Educational tool for relational algebra. <https://ltworf.coderberg.page/relational>.
- [26] Brett Smith. A quick guide to GPLv3. *Free Software Foundation, Inc.*, 2007.
- [27] Yasin N. Silva and Jaime Chon. Dbsnap: Learning database queries by snapping blocks. In *Proceedings of the 46th ACM Technical Symposium on Computer Science Education*, SIGCSE '15, page 179–184, New York, NY, USA, 2015. Association for Computing Machinery.
- [28] Jun Yang. Ra (radb). <https://users.cs.duke.edu/~junyang/radb/index.html>.
- [29] Johannes Kessler, Michael Tschuggnall, and Günther Specht. Relax: A webbased execution and learning tool for relational algebra. BTW 2019, 2019.
- [30] Scientific Calculator Online. Relational algebra calculator: Solve db queries instantly. <https://scientificcalculatoronline.io/relational-algebra-calculator/>.
- [31] Christophe Debruyne. (e)rali: (extended) relational algebra learning instrument. [github.com/childebru/RALI/tree/0cb06fd6bf3f2f5f91a6f9183a0fe2d4087e93c5](https://github.com/childebru/RALI/tree/0cb06fd6bf3f2f5f91a6f9183a0fe2d4087e93c5).
- [32] postgresSQL. postgresql documentation : 7.4. combining queries (union, intersect, except). <https://www.postgresql.org/docs/18/queries-union.html>.
- [33] Johannes Kessler, Michael Tschuggnall, and Günther Specht. Relax v0.34 [source code]. <https://github.com/dbis-uibk/relax>, 2025.

- [34] Mozilla foundation. Mozilla public license version 2.0. <https://www.mozilla.org/en-US/MPL/2.0>, 2012.
- [35] David Majda and Futago-za Ryuu. Peg.js : Parser generator for javascript. [pegjs.org/](https://pegjs.org/), 2010.
- [36] Julia Prior. Online assessment of SQL query formulation skills. In *Proceedings of the fifth Australasian conference on Computing education*, pages 247–256. Australian Computer Society, 2003.
- [37] Julia Prior and Raymond Lister. The backwash effect on sql skills grading. *SIGCSE Bull.*, 36(3):32–36, June 2004.
- [38] Shazia Sadiq, Maria Orlowska, Wasim Sadiq, and Joe Lin. Sqlator: an online SQL learning workbench. In Roger D. Boyle, Martyn Clark, and Amruth N. Kumar, editors, *Proceedings of the 9th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education, ITiCSE 2004, Leeds, UK, June 28-30, 2004*, pages 223–227. ACM, 2004.
- [39] Ferran Prados, Imma Boada, Josep Soler, and Jordi Poch. An automatic correction tool for relational database schemas. In *2005 6th International Conference on Information Technology Based Higher Education and Training*, pages S3C–9–S3C–14. IEEE, 2005.
- [40] Donald R. Schwartz and Pablo Rivas. An automated sql query grading system using an attention-based convolutional neural network. *An Automated SQL Query Grading System Using An Attention-Based Convolutional Neural Network*, 2024.
- [41] Jie Lu, Matthew Schmidt, Minyoung Lee, and Rui Huang. Usability research in educational technology: a state-of-the-art systematic review. *Educational technology research and development*, 70(6):1951–1992, December 2022.
- [42] Jennifer W. M. Lai and Matt Bower. How is the use of technology in education evaluated? a systematic review. *Computers Education*, 133:27–42, May 2019.
- [43] John Brooke. *SUS – a quick and dirty usability scale*, pages 189–194. Taylor Francis, January 1996.
- [44] James Lewis. Psychometric evaluation of the post-study system usability questionnaire: The pssuq. In *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, volume 2, pages 1259–1263, January 1992.
- [45] James Lewis. Ibm computer usability satisfaction questionnaires: Psychometric evaluation and instructions for use. *International Journal of Human-Computer Interaction*, 7:57–, February 1995.
- [46] Guillaume Gronier and Laurence Johannsen. Proposition d’une adaptation française et premières validations de l’échelle d’utilisabilité computer system usability questionnaire (f-CSUQ): Proposal for a french adaptation and first validations of the computer system usability questionnaire (f-CSUQ). In *Proceedings of the 33rd Conference on l’Interaction Humain-Machine, IHM '22*, pages 1–11. Association for Computing Machinery, November 2022.
- [47] Prokopia Vlachogianni and Nikolaos Tselios. Perceived usability evaluation of educational technology using the post-study system usability questionnaire (pssuq): A systematic review. *Sustainability*, 15:12954, August 2023.

## Appendix A

# Questionnaire

### A.1 English Translation of the Questionnaire

| Number | French                                                            | English                                                       |
|--------|-------------------------------------------------------------------|---------------------------------------------------------------|
| 1      | L'objectif de l'outil a été clairement présenté.                  | The purpose of the tool was clearly stated.                   |
| 2      | L'outil a été suffisamment expliqué pour que je sache quoi faire. | The tool was explained well enough for me to know what to do. |

Table A.1: English translation of the questions regarding the presentation of the tool.

| Number | French                                                                                                                                     | English                                                                                                                                     |
|--------|--------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------|
| 1      | Voir les résultats de l'exécution m'a aidé à mieux comprendre les concepts de l'algèbre relationnelle.                                     | Seeing the results of the execution helped me to better understand the concepts of relational algebra.                                      |
| 2      | L'outil m'a encouragé à expérimenter davantage avec les expressions d'algèbre relationnelle.                                               | The tool encouraged me to experiment more with relational algebra expressions.                                                              |
| 3      | L'outil m'a permis de surveiller ma compréhension actuelle du cours d'algèbre relationnelle.                                               | The tool allowed me to monitor my current understanding of the relational algebra course.                                                   |
| 4      | L'outil m'a permis d'identifier les parties du cours d'algèbre relationnelle qui nécessitent plus d'efforts de ma part en vue de l'examen. | The tool allowed me to identify the parts of the relational algebra course that require more effort on my part in preparation for the exam. |
| 5      | J'utiliserai l'outil à l'avenir durant mon apprentissage de l'algèbre relationnelle pour surveiller ma compréhension.                      | I will use the tool in the future during my learning of relational algebra to monitor my understanding.                                     |
| 6      | L'outil apporte une valeur ajoutée pour apprendre la matière du cours d'algèbre relationnelle.                                             | The tool provides added value for learning the material of the relational algebra course.                                                   |
| 7      | L'outil apporte une valeur ajoutée pour ma vie professionnelle future.                                                                     | The tool provides added value for my future professional life.                                                                              |

Table A.2: English translation of the questions regarding the pedagogical usefulness of the tool.

| Number | F-CSUQ [46]                                                                                                                          | CSUQ [45]                                                                                                               |
|--------|--------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------|
| 1      | De manière générale, je trouve ce système facile à utiliser.                                                                         | Overall, I am satisfied with how easy it is to use this system.                                                         |
| 2      | Ce système est simple à utiliser.                                                                                                    | It is simple to use this system.                                                                                        |
| 3      | Je peux faire mon travail efficacement en utilisant ce système.                                                                      | I can effectively complete my work using this system.                                                                   |
| 4      | Je suis capable de faire mon travail rapidement en utilisant ce système.                                                             | I am able to complete my work quickly using this system.                                                                |
| 5      | Je suis capable de faire mon travail de manière efficiente en utilisant ce système.                                                  | I am able to efficiently complete my work using this system.                                                            |
| 6      | Je suis à l'aise avec l'utilisation de ce système.                                                                                   | I feel comfortable using this system.                                                                                   |
| 7      | Cela a été facile d'apprendre à utiliser ce système.                                                                                 | It was easy to learn to use this system.                                                                                |
| 8      | Je suis rapidement devenu productif-ive en utilisant ce système.                                                                     | I believe I became productive quickly using this system.                                                                |
| 9      | Les messages d'erreurs générés par le système expliquent clairement comment résoudre les problèmes.                                  | The system gives error messages that clearly tell me how to fix problems.                                               |
| 10     | Lorsque je fais une erreur en utilisant le système, je peux les rattraper facilement et rapidement.                                  | Whenever I make a mistake using the system, I recover easily and quickly.                                               |
| 11     | Les informations (telles que l'aide en ligne, les messages à l'écran, et autre documentation) fournies avec le système sont claires. | The information (such as on-line help, on-screen messages, and other documentation) provided with this system is clear. |
| 12     | Il est facile de trouver l'information dont j'ai besoin.                                                                             | It is easy to find the information I need.                                                                              |
| 13     | Les informations fournies avec le système sont faciles à comprendre.                                                                 | The information provided with the system is easy to understand.                                                         |
| 14     | Les informations sont efficaces pour m'aider à faire mon travail.                                                                    | The information is effective in helping me complete my work.                                                            |
| 15     | L'organisation des informations sur les écrans du système est claire.                                                                | The organization of information on the system screens is clear.                                                         |
| 16     | L'interface de ce système est agréable.                                                                                              | The interface of this system is pleasant.                                                                               |
| 17     | J'ai plaisir à utiliser l'interface de ce système.                                                                                   | I like using the interface of this system.                                                                              |
| 18     | Ce système possède toutes les fonctionnalités et capacités que j'en attends.                                                         | This system has all the functions and capabilities I expect it to have.                                                 |
| 19     | De manière générale, je suis satisfait-e de ce système.                                                                              | Overall, I am satisfied with this system.                                                                               |

Table A.3: Side to side comparison of the F-CSUQ and CSUQ questions.

## A.2 F-CSUQ Responses

[illegible]

Figure A.1: Overview of the responses to the F-CSUQ per respondent with their corresponding Likert scale.

### A.3 CSUQ Score

| CSUQ score:         |                  |                     |                       |             |
|---------------------|------------------|---------------------|-----------------------|-------------|
| OVERALL (Q1-19) :   | SYSUSE (Q1-Q8) : | INFOQUAL (Q9-Q15) : | INTERQUAL (Q16-Q18) : |             |
| 3,052631579         |                  | 3                   | 3,285714286           | 3           |
| 1,631578947         | 1,714285714      | 1,857142857         | 1,333333333           |             |
| 3,052631579         | 2,142857143      | 4,571428571         | 2,333333333           |             |
| 3,421052632         | 3,714285714      | 4                   |                       | 2           |
| 1,894736842         | 1,714285714      | 2,428571429         | 1,333333333           |             |
| 2                   | 1                | 3,285714286         |                       | 2           |
| 1,631578947         | 1,428571429      | 1,714285714         | 1,666666667           |             |
| 2,894736842         | 3                | 2,571428571         | 3,666666667           |             |
| 2,052631579         | 1,857142857      | 2,142857143         | 2,333333333           |             |
| 1,789473684         | 1,714285714      | 1,571428571         | 2,333333333           |             |
| 2,105263158         | 2,428571429      | 2                   | 1,666666667           |             |
| 2,578947368         | 3,571428571      | 2,428571429         |                       | 1           |
| 2,789473684         | 2,714285714      | 3,285714286         | 2,333333333           |             |
| 3,631578947         | 3,428571429      | 4,285714286         | 2,666666667           |             |
| 2,105263158         | 2                | 2                   | 2,666666667           |             |
| 2,526315789         | 2,285714286      | 2,714285714         |                       | 3           |
| 2,789473684         | 3,857142857      | 2,428571429         | 1,333333333           |             |
| 2,473684211         | 2,714285714      | 2,285714286         | 2,333333333           |             |
| Global mean :       | 2,467836257      | 2,714285714         | 2,166666667           |             |
| Standart Deviation: | 0,603581967      | 0,839532763         | 0,887420422           | 0,697802339 |

Figure A.2: Overview of the individual and global CSUQ scores.