Bases de Dados

Engenharia Informática (2ºano)

Tecnologias Digitais e Gestão (2ºano)

Capítulo 4. Álgebra Relacional e SQL - (MySQL)

João Paulo Pereira | jprp@ipb.pt
Marisa Ortega | marisa.ortega@ipb.pt
David Dias |davide.dias@ipb.pt
Tiago Santos | tiago.santos@ipb.pt

2025

Índice

- Conteúdo
  - 1. Introdução aos ambientes de Base de Dados
    - 1.1 Noção de Sistema de Informação
    - 1.2 A Informação nas Organizações
    - 1.3 Tecnologias de Informação
    - 1.4 Gestão de Informação
  - 2. Sistemas de Gestão de Bases de Dados
    - 2.1 Abordagem e Vantagens
    - 2.2 Arquitetura de um SGBD
    - 2.3 Tipos de Utilizadores num SGDB
  - 3. Modelação e Normalização de Dados
    - 3.1 Manutenção da Integridade
    - 3.2 Redundância e Chaves
    - 3.3 Diagramas E-R
    - 3.4 Modelo Relacional
  - 4. Álgebra Relacional e SQL - (MySQL)
    - 4.1 Conceitos e aplicação de Álgebra Relacional
    - 4.2 Ferramentas de Administração MySQL
    - 4.3 Comandos DDL
    - 4.4 Comandos DML
  - 5. Introdução às Bases de Dados NoSQL
    - Bases de Dados NoSQL,
    - Tipos de Bases de Dados NoSQL,
    - Bases de dados orientadas a documentos
  - 6. MongoDB
    - Estruturas JSON e BSON,
    - Modelação de dados,
    - Criação de coleções e documentos,
    - Operações CRUD e agregação,
    - Indexação e transações

4.1 Concepts and application of Relational Algebra

- What is Relational Algebra?
  - ➔ Formal data manipulation model
  - ➔ Defines operations on relations (tables)
  - ➔ Basis for languages like

- SQL Fundamental Concepts
  - ➔ Relation = Set of tuples (rows) with the same schema
  - ➔ Attributes = Relation columns
  - ➔ Schema = Relation name + attributes

- Relational Algebra
  - ➔ Operators Unary operators (1 relation)
  - ➔ Binary operators (2 relations)
  - ➔ Categories:
    1. Selection (σ)
    2. Projection (π)
    3. Union (∪)
    4. Difference (−)
    5. Cartesian product (×)
    6. Join (⋈)
    7. Rename (ρ)

  - ➔ Derivative Operations (These can be constructed with basic operators)
    1. Intersection (∩)
    2. Division
    3. Outer join

Relational algebra is a set of operators to manipulate relations. Each operator of the relational algebra takes either one or two relations as its input and produces a new relation as its output.
Codd defined 8 such operators, two groups of 4 each:
- The traditional set operations: union, intersection, difference and Cartesian product
- The special relational operations: select, project, join and divide.

CODD, E.F. *Relational completeness on data base sublanguage*, Data Base Systems, Courant Computer Science Symposia Series, Vol.6 Englewood Cliffs, N.J, Prentice-Hall, 1972

4.1 Concepts and application of Relational Algebra

- Relational Algebra Operators
  - ➔ Selection (σ)
    - ⇨ Filters rows based on a condition
    - ⇨ Notation: σ_{condition}(R)
    - ⇨ Example: σ_{age > 30}(People)
  - ➔ Projection (π)
    - ⇨ Selects specific columns (attributes)
    - ⇨ Notation: π_{attributes}(R)
    - ⇨ Example: π_{name, age}(People)
  - ➔ Union (∪)
    - ⇨ Combines two relations with the same schema
    - ⇨ Removes duplicates (by definition)
    - ⇨ Example: R ∪ S
  - ➔ Difference (−)
    - ⇨ Rows that are in R but not in S
    - ⇨ Example: R − S
  - ➔ Cartesian Product (×)
    - ⇨ Combines each row in R with all of S
    - ⇨ Can generate large volumes of data
    - ⇨ Example: R × S
  - ➔ Join (⋈)
    - ⇨ Combines tuples based on a condition
    - ⇨ Natural join: ⋈
    - ⇨ Conditional join: R ⋈_{condition} S
  - ➔ Rename (ρ)
    - ⇨ Changes the name of relationships or attributes
    - ⇨ Notation: ρ_{new_name}(R)
    - ⇨ Useful for distinguishing temporary relationships

4.1 Concepts and application of Relational Algebra

- Union
  - ➔ The union of two union-compatible relations R1 and R2, R1 UNION R2, is the set of all tuples t belonging to either R1 or R2 or both.
  - ➔ Two relations are union-compatible if they have the same degree, and the ith attribute of each is based on the same domain.
  - ➔ The formal notation for a union operation is U.
  - ➔ UNION operation is associative and commutative.
  - ➔ Next figure provides an example of a UNION operation. The operands are relation R1 and relation R2 and the result is another relation R3 with 5 tuples.

R1

Name	Age	Sex
A	20	M
C	21	M
B	21	F

R2

Name	Age	Sex
D	20	F
A	20	M
E	21	F

R3= R1 U R2

Name	Age	Sex
A	20	M
C	21	M
B	21	F
D	20	F
E	21	F

Course_1

C_id	C_name
11	Foundation C
21	C++
31	JAVA

Course_2

C_id	C_name
12	Python
21	C++

Course_1 U Course_2

C_id	C_name
11	Foundation C
21	C++
31	JAVA
12	Python

4.1 Concepts and application of Relational Algebra

- Intersection
  - ➔ The intersection of two union-compatible relations R1 and R2, R1 INTERSECT R2, is the set of all tuples t belonging to both R1 and R2.
  - ➔ The formal notation for an intersect operation is ∩.
  - ➔ INTERSECT operation is associative and commutative.
  - ➔ Next figure provides an example of an INTERSECT operation. The operands are relation R1 and relation R2 and the result is another relation R3 with only one tuple.

R1

Name	Age	Sex
A	20	M
C	21	M
B	21	F

R2

Name	Age	Sex
D	20	F
A	20	M
E	21	F

R3= R1 ∩ R2

Name	Age	Sex
A	20	M

Relação A:

ID	Nome
1	Ana
2	João
3	Marta

Relação B:

ID	Nome
2	João
3	Marta
4	Pedro

$A \cap B$

Resultado:

ID	Nome
2	João
3	Marta

$$R \cap S = R - (R - S)$$

4.1 Concepts and application of Relational Algebra

- **Difference**
  - ➔ The ***difference*** between two union-compatible relations **R1** and **R2**, **R1 MINUS R2**, is the set of all tuples **t** belonging to **R1** and not to **R2**.
  - ➔ The formal notation for a difference operation is -
  - ➔ **DIFFERENCE** operation is not associative and commutative.
  - ➔ *Next figure* provides an example of a **DIFFERECE** operation. The operands are relation **R1** and relation **R2** and the result is another relation **R3** with two tuples. As you can see, the result of **R1-R2** is different from **R2-R1**.

R1

Name	Age	Sex
A	20	M
C	21	M
B	21	F

R2

Name	Age	Sex
D	20	F
A	20	M
E	21	F

R3= R1 - R2

Name	Age	Sex
C	21	M
B	21	F

R3= R2 – R1

Name	Age	Sex
D	20	F
E	21	F

4.1 Concepts and application of Relational Algebra

- Cartesian product
  - ➔ The Cartesian product between two relations R1 and R2, R1 TIMES R2, is the set of all tuples t such that t is the concatenation of a tuple r belonging to R1 and a tuple s belonging to R2. The concatenation of a tuple r = (r1, r2, …, rm) and a tuple s = (sm+1, sm+2, …, sm+n) is the tuple t = (r1, r2, …, rm, sm+1, sm+2, …, sm+n).
  - ➔ R1 and R2 don't have to be union-compatible.
  - ➔ The formal notation for a Cartesian product operation is ×.
  - ➔ If R1 has degree n and cardinality N1 and R2 has degree m and cardinality N2 then the resulting relation R3 has degree (n+m) and cardinality (N1*N2). This is illustrated in Figure.

R1

Name	Age	Sex
A	20	M
C	21	M

R2

Name	Age	Sex
D	20	F
E	21	F

R3= R1 X R2

Name	Age	Sex	Name	Age	Sex
A	20	M	D	20	F
C	21	M	D	20	F
A	20	M	E	21	F
C	21	M	E	21	F

4.1 Concepts and application of Relational Algebra

- **Selection**
  - ➔ The ***select*** operation selects a subset of tuples from a relation. It is a unary operator, that is, it applies on a single relation. The tuples subset must satisfy a selection condition or predicate.
  - ➔ The formal notation for a select operation is:
    - ⇨ **σ <select condition> (<relation>)**
    - ⇨ where **<select condition>** is
    - ⇨ **<attribute> <comparison operator> <constant value>/<attribute> [AND/OR/NOT <attribute> <comparison operator> <constant value>/<attribute>…]**
  - ➔ The comparison operator can be **<, >, <=, >=, =, <>** and it depends on attribute domain or data type constant value.
  - ➔ The resulting relation degree is equal with the initial relation degree on which the operator is applied. The resulting relation cardinality is less or equal with the initial relation cardinality. If the cardinality is low we say that the select condition selectivity is high and if the cardinality is high we say that the select condition selectivity is low.
  - ➔ Selection is commutative.

4.1 Concepts and application of Relational Algebra

- **Selection**
  - ➔ In the next *Figure*, there are two select operation examples performed on relation **R**. First the select condition is **Age=20** and the result is relation **R1** and second the select condition is **(Sex=M) AND (Age>19)** and the result is relation **R2**.

R

Name	Age	Sex
A	20	M
M	21	F
B	20	F
F	19	M
A	20	F
R	21	F
C	21	M

R1= σ(Age=20)(R)

Name	Age	Sex
A	20	M
B	20	F
A	20	F

R2= σ(Sex=M AND Age>19)(R)

Name	Age	Sex
A	20	M
C	21	M

4.1 Concepts and application of Relational Algebra

- **Projection**
  - ➔ The ***project*** operation builds another relation by selecting a subset of attributes of an existing relation. Duplicate tuples from the resulting relation are eliminated. It is also a unary operator.
  - ➔ The formal notation for a project operation is:
    - ⇨ **π <attribute list> (<relation>)**
    - ⇨ where **<attribute list>** is the subset attributes of an existing relation.
  - ➔ The resulting relation degree is equal with the number of attributes from **<attribute list>** because only those attributes will appear in the resulting relation. The resulting relation cardinality is less or equal with the initial relation cardinality. If the list of attributes contains a relation candidate key, then the cardinality is equal with the initial relation cardinality. If it does not contain a candidate key, then the cardinality could be less because of the possibility to have duplicate tuples, which are eliminated from the resulting relation.
  - ➔ Projection is not commutative.

4.1 Concepts and application of Relational Algebra

- In the next f*igure* there are two project operation examples performed on relation **R**. First the projection is made on attributes **Name** and **Sex** and the result is relation **R1** and second the projection is made on attributes **Age** and **Sex** and the result is relation **R2**.

R

Name	Age	Sex
A	20	M
M	21	F
B	20	F
F	19	M
A	20	F

R1= π(Name, Sex)(R)

Name	Sex
A	M
M	F
B	F
F	M
A	F

R2= π(Age, Sex)(R)

Age	Sex
20	M
21	F
20	F
19	M

4.1 Concepts and application of Relational Algebra

- **Join**
  - ➔ The ***join*** operation concatenates two relations based on a joining condition or predicate. The relations must have at least one common attribute with the same underlying domain, and on such attributes a joining condition can be specified.
  - ➔ The formal notation for a join operation is:
    - ⇨ **R <join condition> ► ◄S**
    - ⇨ where **<join condition>** is
    - ⇨ **<attribute from R> <comparison operator> < <attribute from S>**
  - ➔ The comparison operator can be **<, >, <=, >=, =, <>** and it depends on attributes domain.
  - ➔ If relation **R** has attributes **A1, A2, …, An** and relation **S** has attributes **B1, B2, …, Bm** and attribute **Ai** and attribute **Bj** have the same underlying domain we can define a join operation between relation **R** and relation **S** on a join condition between attribute **Ai** and **Bj**. The result is another relation **T** that contains all the tuples **t** such that **t** is the concatenation of a tuple **r** belonging to **R** and a tuple **s** belonging to **S** if the join condition is true. This type of join operation is also called ***theta-join***. It follows from the definition that the result of a join must include two identical attributes from the point of view of their values. If one of those two attributes is eliminated the result is called ***natural join***.

4.1 Concepts and application of Relational Algebra

- Join
  - ➔ There are also other forms of join operations. The most often used is the ***equijoin***, which means that the comparison operator is =.
  - ➔ There are situations when not all the tuples from relation R have a corresponding tuple in relation S. Those tuples won't appear in the result of a join operation between R and S. In practice, sometimes it is necessary to have all tuples in the result, so, another form of join was created: the ***outer join***.
  - ➔ There are 3 forms of outer join:
    - ⇨ ***left outer join***, where all tuples from R will be in the result,
    - ⇨ ***right outer join***, where all tuples from S will be in the result, and
    - ⇨ ***full outer join***, where all tuples from R and S will be in the result. If there is not a corresponding tuple, the system considers that there is a hypothetical tuple, with all attribute values null, which will be used for concatenation.
  - ➔ In the next *figure* there are two relations **R1** and **R2** joined on a join condition where **LastName** from relation **R1** is equal with **LastName** from relation **R2**. The resulting relation is **R3**.

R1

First Name	Last Name
A	Mary
B	John
C	Ann

R2

Last Name	Sex
Ann	F
John	M
Mary	F
Bill	M

R3=R1(Last Name=Last name) R2

First Name	Last Name	Last Name	Sex
A	Mary	Mary	F
B	John	John	M
C	Ann	Ann	F

4.1 Concepts and application of Relational Algebra

- Join
  - ➔ In the next *figure* you could see the results for a natural join between **R1** and **R2** and also the results for a right outer join.

R1

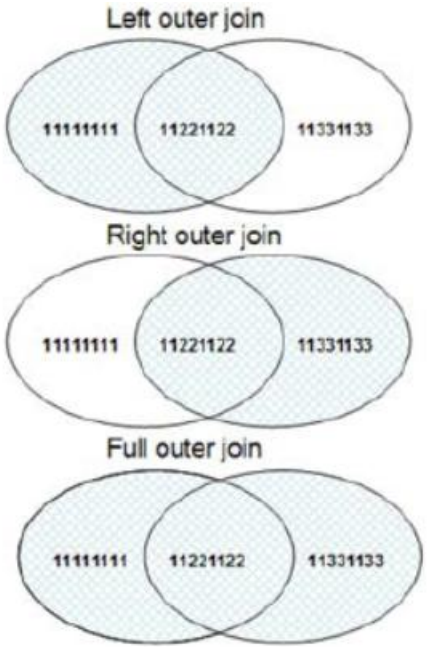
First Name	Last Name
A	Mary
B	John
C	Ann

R2

Last Name	Sex
Ann	F
John	M
Mary	F
Bill	M

R3=R1(Last Name=Last name) R2

First Name	Last Name	Last Name	Sex
A	Mary	Mary	F
B	John	John	M
C	Ann	Ann	F



Natural Join

First Name	Last Name	Sex
A	Mary	F
B	John	M
C	Ann	F

Right Outer Join

First Name	Last Name	Last Name	Sex
A	Mary	Mary	F
B	John	John	M
C	Ann	Ann	F
NULL	NULL	Bill	M

4.1 Concepts and application of Relational Algebra

- **Division**
  - ➔ The ***division*** operator divides a relation **R1** of degree **(n+m)** by a relation **R2** of degree **m** and produces a relation of degree **n**. The **(n+i)**th attribute of **R1** and the **i**th attribute from **R2** should be defined on the same domain. The result of a division operation between **R1** and **R2** is another relation, which contains all the tuples that concatenated with all **R2** tuples are belonging to **R1** relation.
  - ➔ The formal notation for a division operation is ÷.
  - ➔ *Next figure* provides an example of a division operation between relation **R1** and **R2.**

R1

Name	Sex
A	M
B	F
A	F
C	F
D	M
C	M

R2

Sex
M
F

R3= R1÷ R2

Name
A
C

4.1 Concepts and application of Relational Algebra

- Differences between Relational Algebra and SQL
  - ➔ SQL allows duplicates, relational algebra does not
  - ➔ SQL is more permissive in syntax
  - ➔ Relational algebra is more formal and mathematical

- Advantages of Relational Algebra
  - ➔ Formal clarity
  - ➔ Solid theoretical basis
  - ➔ Facilitates query optimization

- Disadvantages/Limitations
  - ➔ Less user-friendly syntax
  - ➔ Little used directly (more in theory)
  - ➔ SQL is more expressive for complex practical cases

- Summary
  - ➔ Relational algebra = basis for SQL
  - ➔ Main operators: σ, π, $\cup$, $-$, $\times$, $\bowtie$
  - ➔ Essential for understanding the logic behind the query

4.1 Concepts and application of Relational Algebra

- Example:
  - ➔ Tables:
    - ⇨ Students(Name, Course)
    - ⇨ Courses(Course, Department)
  - ➔ Query: Student names and corresponding department
    - ⇨ π_{Name, Department}(Students ⋈ Courses)
    - ⇨ Reasoning:
      - Natural join between Students and Courses
      - Projection of the Name and Department attributes

Operação	Quando usar?
σ (seleção)	Filtrar linhas com base numa condição
π (projecção)	Escolher colunas/atributos
× (produto)	Combinar todas as linhas de duas tabelas
⋈ (junção)	Combinar tabelas com base numa condição
− (diferença)	Obter elementos de uma relação que não estão noutra

- Example: Selection and Projection (Simple)
  - ➔ Statement:
    - ⇨ Given the Employees(Name, Position, Salary, Department) relation, get the names of the employees who work in the "Sales" department.
    - ⇨ Resolution:
      - Filter (selection):
        - σ_{Department = 'Sales'}(Employees)
      - Name projection:
        - π_{Name}(σ_{Department = 'Sales'}(Employees))
      - Final result:
        - π_{Name}(σ_{Department = 'Sales'}(Employees))

- Reasoning:
  - ➔ Select rows that satisfy a condition:
    - ⇨ We want only employees in the "Sales" department.
    - ⇨ We use the selection operator (σ) with the condition Department = 'Sales'.
  - ➔ Select only the "Name" column:
    - ⇨ After filtering, we want only the names.
    - ⇨ We use the projection operator (π) to extract only that column.

4.1 Concepts and application of Relational Algebra

- Example: Join and Projection
  - ➔ Statement:
    - ⇨ Given the relationships:
      - Students(StudentID, Name, CourseID)
      - Courses(CourseID, CourseName, Department)
    - ⇨ Get the names of the students and the names of their courses.
  - ➔ Resolution:
    - ⇨ Join between Students and Courses:
      - Students ⋈ Courses (Natural join using CourseID)
    - ⇨ Projection of the desired attributes:
      - π_{Name, CourseName}(Students ⋈ Courses)
    - ⇨ Final result:
      - π_{Name, CourseName}(Students ⋈ Courses)
  - ➔ Reasoning:
    - ⇨ Relate the two tables:
      - Both have the CourseID attribute.
      - We can join the tables with a natural join, since they have a common attribute.
    - ⇨ Select only the relevant data:
      - We want the names of the students and the names of the courses.
      - Projection on the attributes Name and CourseName.

4.1 Concepts and application of Relational Algebra

- Example: Difference
  - ➔ Statement:
    - ⇨ Given the relationships:
      - Customers(CustomerID, Name)
      - Purchases(PurchaseID, CustomerID, Date)
      - Get the names of customers who have never made a purchase.
    - ⇨ Resolution:
      - Getting customers who made purchases:
        - π_{CustomerID}(Purchases)
      - Get all customers:
        - π_{CustomerID}(Customers)
      - Difference:
        - π_{CustomerID}(Customers) − π_{CustomerID}(Purchases)
      - Join with the Customers relation to get the names:
        - ρ_{CustomersWithoutPurchases}(π_{CustomerID}(Customers) − π_{CustomerID}(Purchases))
        - CustomersWithoutPurchases ⋈ Customers
        - π_{Name}(CustomersWithoutPurchases ⋈ Customers)
      - Final result:
        - π_{Name}((π_{CustomerID}(Customers) − π_{CustomerID}(Purchases)) ⋈ Customers)
  - ➔ Reasoning:
    - ⇨ Customers who purchased:
      - We extract the CustomerIDs from the Purchases list.
    - ⇨ Total customers:
      - We extract the CustomerIDs from the Customers list.
    - ⇨ Difference:
      - We use the difference operator (−) to get the IDs of customers who are in Customers but not in Purchases.
    - ⇨ Get the names:
      - Since we only have the IDs, we need to reconnect to Customers to get the names — using a join.

4.1 Concepts and application of Relational Algebra

- Example: Cartesian Product + Selection
  - ➔ Statement:
    - ⇨ Given:
      - Professors(ID, Name, Department)
      - Disciplines(DisciplineID, DisciplineName, Department)
      - List all pairs (professor, discipline) from the same department.
  - ➔ Resolution:
    - ⇨ Cartesian product:
      - Teachers × Subjects
    - ⇨ Join condition:
      - σ_{Teachers.Department = Subjects.Department}
    - ⇨ Name projection:
      - π_{Name, SubjectName}(σ(...))
    - ⇨ Final result:
      - π_{Name, SubjectName}(σ_{Teachers.Department = Subjects.Department}(Teachers × Subjects))
  - ➔ Reasoning:
    - ⇨ Cartesian product (×):
      - Pairs all rows of Teachers with all rows of Subjects.
    - ⇨ Filter pairs that have the same department:
      - We use selection (σ) with the condition Teachers.Department = Subjects.Department.
    - ⇨ Projection:
      - We want to see only the name of the teacher and the subject.
      - We use π_{Name, SubjectName}

4.1 Concepts and application of Relational Algebra

- Example: Composition of multiple operations (Join + Multiple Filters)
  - ➔ Statement:
    - ⇨ Given:
      - Employees(ID, Name, Salary, DepartmentID)
      - Departments(ID, DeptName, Location)
      - List the names of employees who earn more than 2000 and who work in "Lisbon".
  - ➔ Resolution:
    - ⇨ Join:
      - Employees ⋈_{Employees.DepartmentID = Departments.ID} Departments
    - ⇨ Filter (double condition):
      - σ_{Salary > 2000 ∧ Location = 'Lisbon'}
    - ⇨ Name projection:
      - π_{Name}(...)
    - ⇨ Final result:
      - π_{Name}(σ_{Salary > 2000 ∧ Location = 'Lisbon'}(Employees ⋈_{Employees.DepartmentID = Departments.ID} Departments))
  - ➔ Reasoning:
    - ⇨ Join between Employees and Departments:
      - We link the tables by the common attribute: DepartmentID = ID.
    - ⇨ Filter by conditions:
      - Salary > 2000
      - Location = "Lisbon"
    - ⇨ Projection:
      - We only want the employee's name.

4.1 Concepts and application of Relational Algebra

- Example: Selection + Projection
  - ➔ Fictitious Schema:
    - ⇨ Employees Relationship(EmployeeID, Name, Age, Department)
  - ➔ Desired Query:
    - ⇨ Display the Name and Age of all employees over 30.
  - ➔ Algebraic Expression:
    - ⇨ $\pi_{Name, Age}(\sigma_{Age>30}(Employees))$
  - ➔ Reasoning
    - ⇨ Selection:
      - σ_{Age > 30}(Employees)
        - filters the tuples from the Employees table that satisfy the "Age > 30" condition.
    - ⇨ Projection:
      - π_{Name, Age}(…) from the filtered table, we extract only the Name and Age columns.
    - ⇨ The result is a new relationship/table with only two columns and only employees over 30.