INF 4401
Introduction To Databases

Julien Romero - Télécom SudParis

Meet Your Instructor

Julien Romero

julien.romero@telecom-sudparis.eu

Office: D307

General Organization

- **[16/01/2024]** CI1 - Introduction And Relational Algebra
  - Databases and information systems
  - Types of Databases
  - Relational Algebra
- **[23/01/2024]** CI2 - SQL 1
  - From relational algebra to SQL
  - SFW queries
  - Set operations
- **[26/02/2024]** CI3
  - Joins
  - Aggregation

General Organization

- **[27/02/2024]** CI4 - Modelisation
  - Entity-relationship diagram
- **[05/03/2024]** CI5 - RDBMS
  - Usage of an RDBMS
  - Setting up a database
  - Tables manipulation
- **[12/03/2024]** CI6 - NoSQL Databases
  - Ecosystem of NoSQL databases
  - Key-Value Storage

General Organization

- **[19/03/2024]** CI7 - Lecture Review
  - Review of all the main concepts seen during the lecture
  - Core exercises
- **[02/04/2024] CF**
  - Final Exam
  - No document allowed

Link to course content:
https://www-inf.telecom-sudparis.eu/COURS/INF4401/Supports/

Link to Moodle: https://moodle.imtbs-tsp.eu/course/view.php?id=197

Grading

Your grade will be composed of:

- One homework (10%)
- Attendance (10%)
- The final exam (80%)

Final Exam

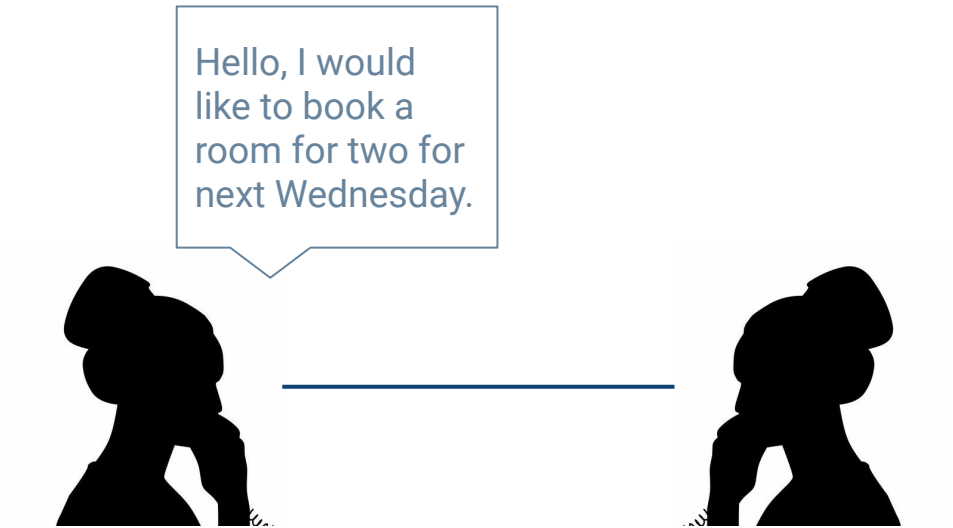
- At least half the points will be “pure knowledge”
  - MCQ, close questions
  - You know the content of the lecture, you validate the course
  - No document allowed

What We Will Learn Today

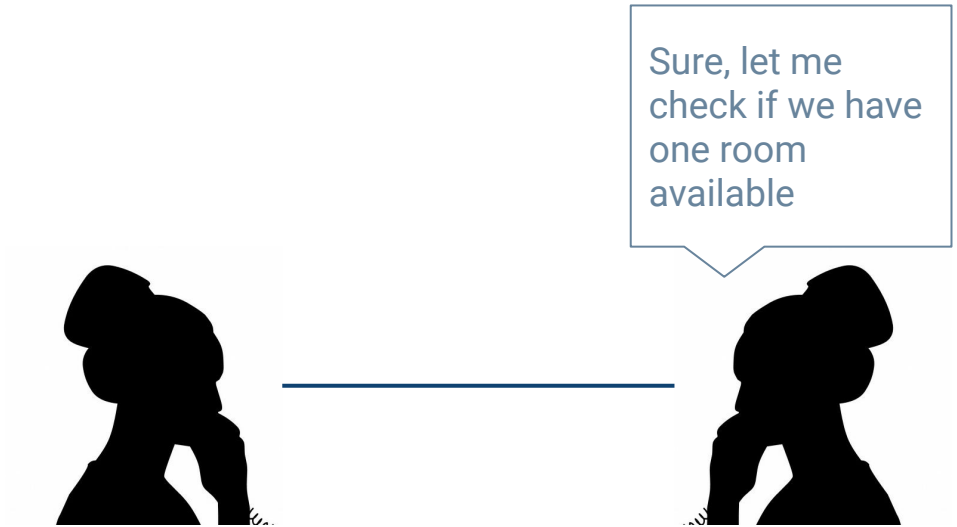
- What is a database and what it does
- What are the different types of databases
- What is a relational database and what are its advantages
- How relational databases are modeled
- Relational Algebra

Introduction To Databases

Booking a Hotel Before Computer Science...

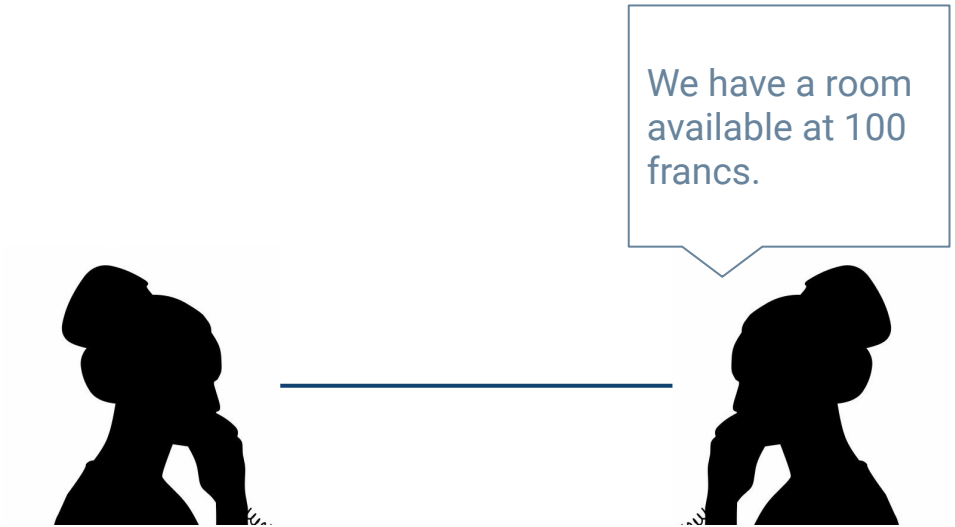


Booking a Hotel Before Computer Science...

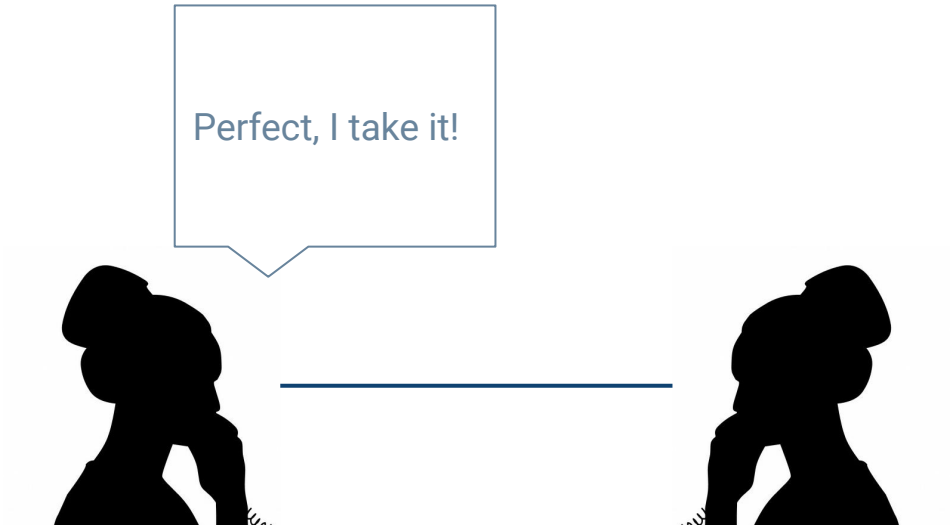


Booking a Hotel Before Computer Science...

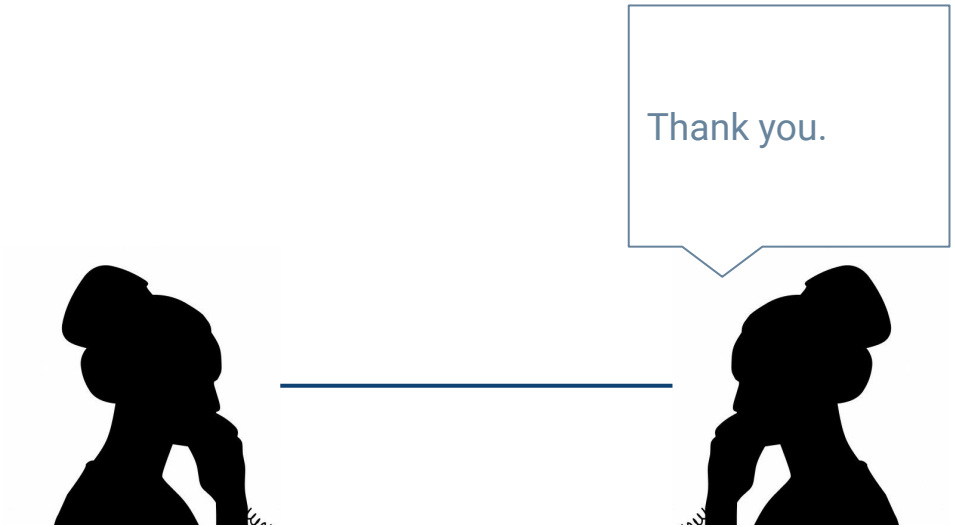
Booking a Hotel Before Computer Science...



Booking a Hotel Before Computer Science...



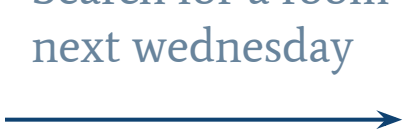
Booking a Hotel Before Computer Science...



Booking a Hotel Before Computer Science...

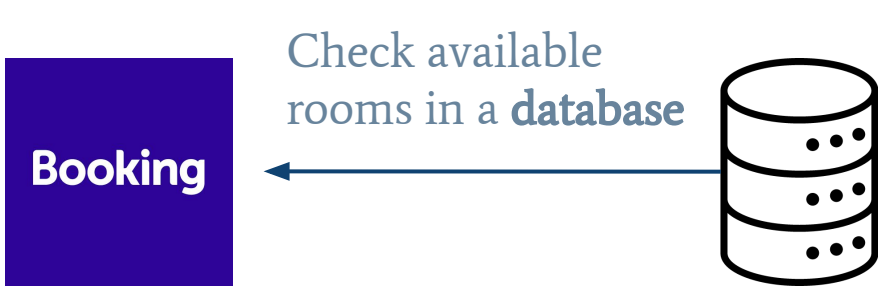
Booking a Hotel Now...

Search for a room
next wednesday



Booking

Booking a Hotel Now...



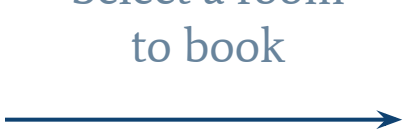
Booking a Hotel Now...

Display the information

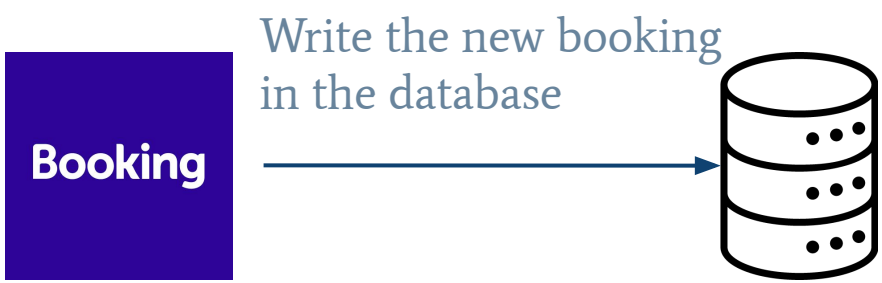
Booking

Booking a Hotel Now...

Select a room to book

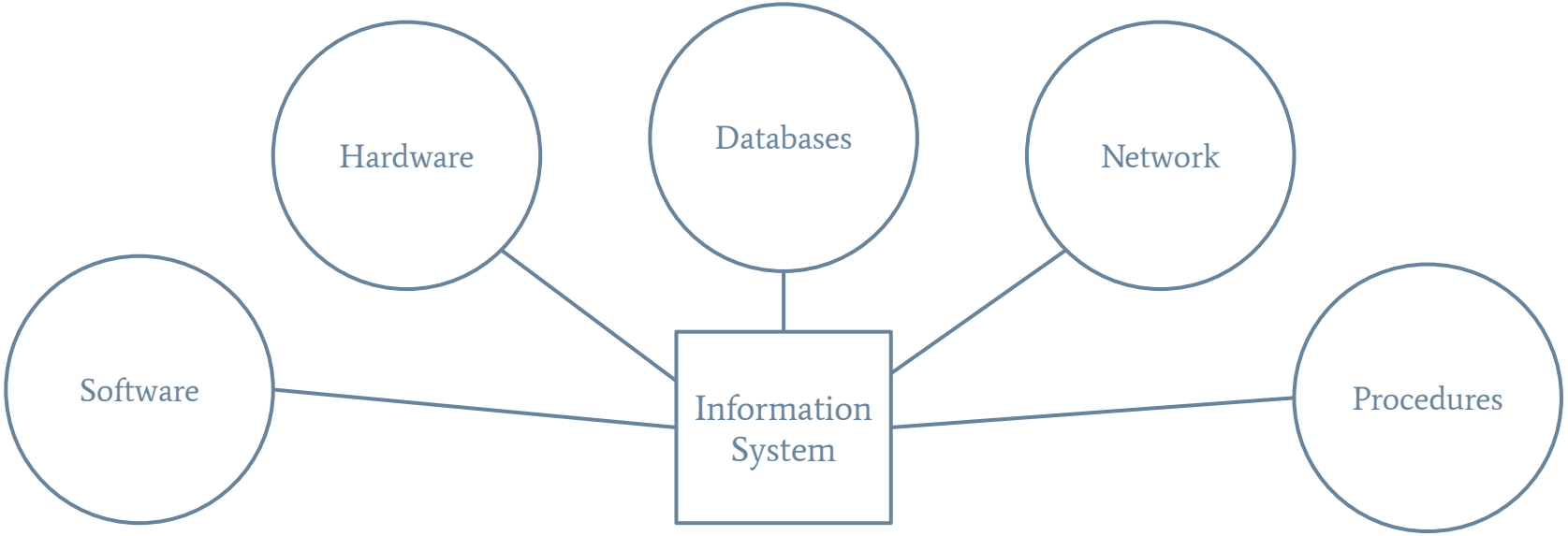


Booking a Hotel Now...



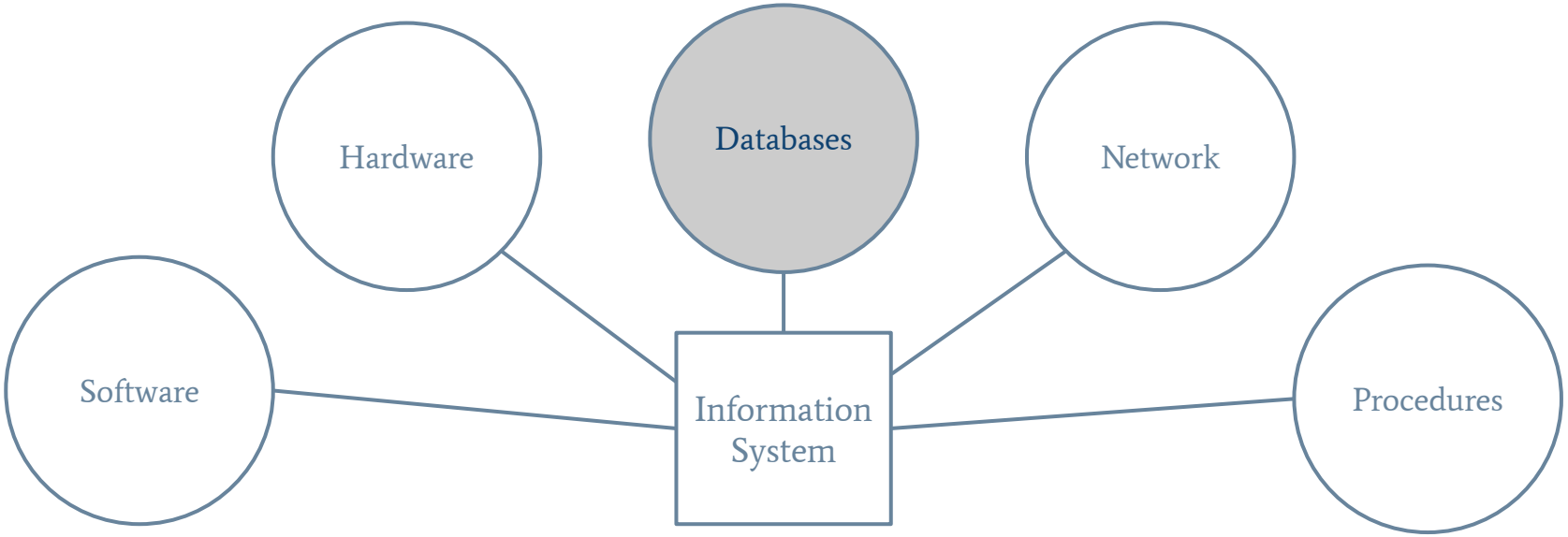
Computer-Based Information Systems

"An information system (IS) is a formal, sociotechnical, organizational system designed to **collect, process, store, and distribute information.**" [Wikipedia]



Computer-Based Information Systems

"An information system (IS) is a formal, sociotechnical, organizational system designed to **collect, process, store, and distribute information.**" [Wikipedia]

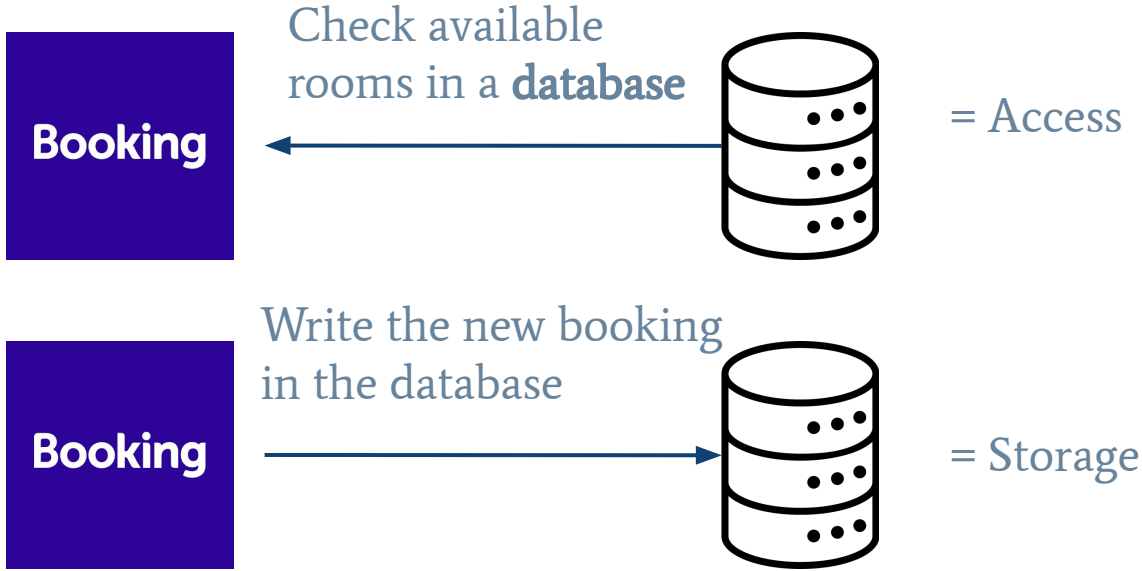


Data Is At The Core Of All Applications

- Booking.com
  - 400 million users per months
  - 200,000 hotels
  - Stores bookings, information about the hotels, information about the user, evaluations, ...
- Google
  - 3.5 billion searches daily
  - 1 trillion URLs
  - Stores queries, information about websites, user actions, ...
- WhatsApp
  - 65 billion messages per day
- Facebook
  - 2.9 billion of users
  - Store personal information, posts, friends, likes, groups, messages, ...
  - 4 petabytes of data per day (= 4,000,000 gigabytes = half a million 2h movies at 1080p)

A Database

"In computing, a database is an **organized** collection of data **stored** and **accessed** electronically." [Wikipedia]



Functionalities of a Database

Querying a Database - Access the data

When we want to read data stored on a database, we say that we **query** the database.

- For example:
  - Give me the free rooms available next Wednesday
  - Give me the list of my friends
  - Give me my unread messages
  - Give me my schedule for next week

Querying a Database - Related Problems

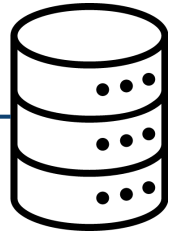
- How fast can I read the data?
  - **Latency** = delay between the request and the moment you actually have the data
  - Google Search = Low latency
  - Dropbox = High latency

Where Does Latency Come From?

Latency of the website:

- Efficient code
- Enough computers

Booking



Latency of the network:

- Slow connection
- Website hosted far

Latency of the database:

- Fast query
- Fast implementation
- Database close to the server
- Number of requests

Querying a Database - Related Problems

- How often do I read data?
  - 63,000 searches per second on Google = We need fast access to answer all of them
  - Few downloads of big files on DropBox = Speed is not the main problem
- Can I express complex queries?
  - “Give me the list of my friends”: simple
  - “Give me the list of my married friends who have at least a baby and who live in France”: complex
    - We need to access and combine different kind of information to answer this query
- Is it simple to write a query?
  - Do I need to write complex code, using a programming language like Java or C?
  - Can I use a simple description language that will manage everything for me?

Writing In a Database - Storing the Data

When we want to write in a database, we want to **insert**, **modify,** or **delete** data.

For example:

- Add a new friend = Insertion
- Remove a friend = Deletion
- Change my email address = Modification
- Book a room = Insertion (add information about the client) + Modification (change the status of the booked room)

Writing In a Database - Related Problems

- How fast can I write data?
  - Add a friend on Facebook = fast
  - Upload a video on Youtube = slow
- How often do I write data?
  - In general, we have fewer write operations than read operations
  - Writing data is slower than reading data
  - Whatsapp = many writes
  - Facebook = many reads, few writes
- What can I write?
  - Text? Images? Videos?

Organizing/Defining Data

We want to organize the data to make read and write operations faster = **data modeling**

We want to **create**, **modify** and **remove** definitions.

VS

Defining Data - Related Problems

- Can we define complex structures?
  - To store videos, we do not need a complex organisation
  - To answers queries like *"Give me the list of my married friends who have at least a baby and who live in France"*, we have to have a strict organization as we potentially combine and filter several sources of information
- Is it easy to define the structure of your data?
- Is it easy to reorganize your data?
  - Is it possible?
  - Is it fast?

Data Security

We want to make sure only authorized entities can access protected information.

For example:

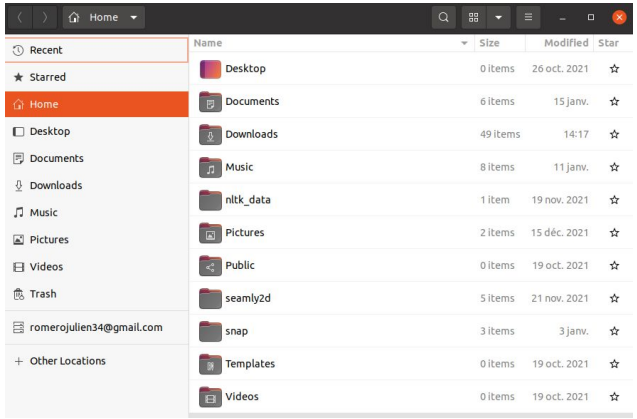
- We do not want anyone to read our Whatsapp messages
- Human resources information in a company should not be accessible by developers

Examples of Databases

Raw Files

We could use raw files to store data.

The data is stored on your hard drive and managed by the file system of your operating system (Windows, Linux, Mac)



Raw Files - Read

- How fast can I read the data?
  - It depends on the hardware
  - Fast if you know where is your data
  - Slow if you have to search for a specific information (this is generally the case)
- Can I express complex queries?
  - No, I can just access a file given its path (e.g.: /home/julien/Documents/inf4401)
- Is it simple to write a query?
  - We have to use a programming language like Java or C and explicitly code the processing of the file
  - The implementation depends on the OS and the file system
  - E.g: Read the list of my friends = Open the file + process each line to get a list of friends

Raw Files - Write

- How fast can I write data?
  - Depends on the hardware
  - Some operations can be very long (e.g. to change my email address, I need to process an entire file and rewrite it)
- How often do I write data?
  - Your file system can be very slow when writing too many files
- What can I write?
  - Everything kind of files! It is particularly good for large files.
  - Does not take into account the content of the file.

Raw Files - Defining Data

- Can we define complex structures?
  - No, we just have the paths of the files on which we can play.
  - If we have too many files and users, it is hard to understand the links between data.
  - We potentially have data redundancy and inconsistency. The file system does not provide mechanisms for this.
- Is it easy to define the structure of your data?
  - There is not much to define, so yes.
- Is it easy to reorganize your data?
  - Moving files is fast, so yes.

Raw Files = Database?

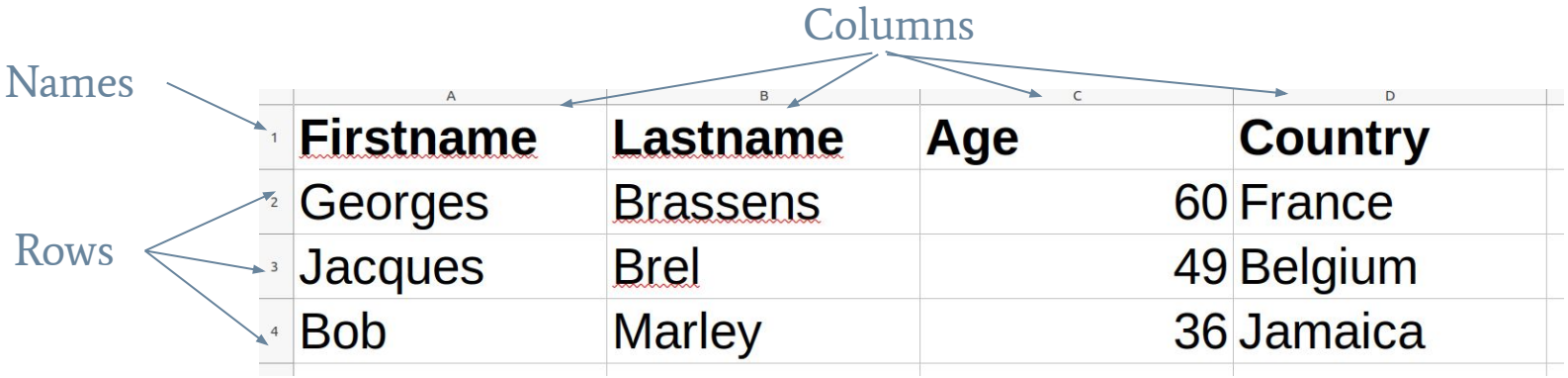
The raw files could be considered as a very simple database. However, it lacks data organisation and does not consider the content of the file, which make them hard to exploit

However, they are used as the base elements to constructed more elaborated structures!

Relational Databases - Tables

Relational databases are constructed around the notion of **table**.

A table is composed of **rows** and named **columns** = excel document



Relational Database

A relational database is an organized collection of one or several tables. It provides organizational tools such as:

- Typed and named columns (e.g. the column “age” is composed of integers)
- Unique identifiers for rows called the **keys**
- Links between tables: an identifier can be referred in another table
- Consistency checking: Checks that we do not write incorrect information (like negative age)
- Access protection
- ...

Relational Databases - Read

- How fast can I read the data?
  - Fast thanks to highly structured data
- Can I express complex queries?
  - Yes, because I can connect tables together
- Is it simple to write a query?
  - Relational Databases come with a query language called **SQL** (see in future lectures)
  - SQL allow to simply describe what we want to do without programming anything!

Relational Databases - Write

- How fast can I write data?
  - Faster than on raw files (especially for deletions/modifications)
  - Small overhead to manage the structure of the data
- How often do I write data?
  - Too many writes can be a problem, but it is generally not the case
  - Generally, we write small quantities of data
  - Writes are slower than reads, but it is faster than raw files
- What can I write?
  - In general, a column contains “small data” like a string or an integer.
  - No good for pictures and videos.

Relational Databases - Defining Data

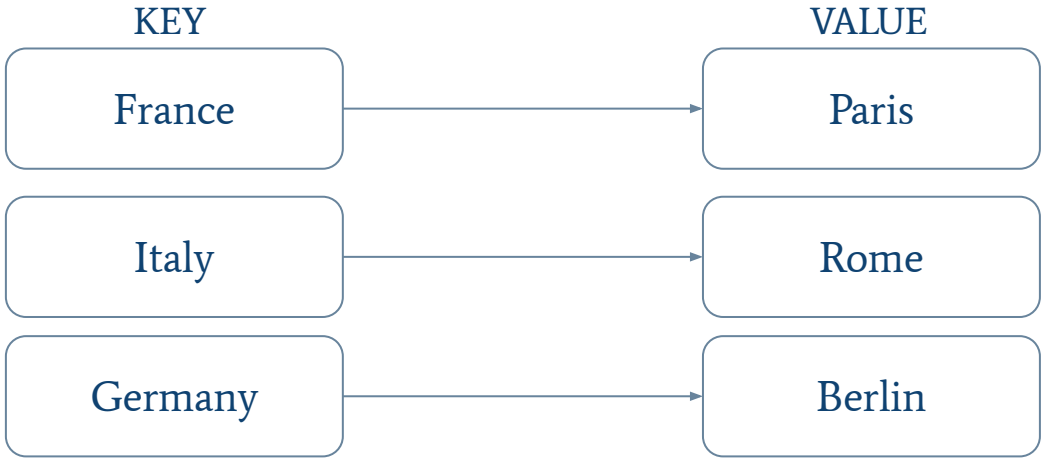
- Can we define complex structures?
  - Yes, and by using **SQL**
  - We can describe precisely tables and their links
- Is it easy to define the structure of your data?
  - Yes, the tables are easy to understand and the definition of the databases can be visually represented.
- Is it easy to reorganize your data?
  - It is feasible, but not recommended. This operation can be long to execute.

Non-Relational/NoSQL Databases

NoSQL databases are databases that are non relational.

A large family of databases:

- Key, value storage: Associate a value to a unique identifier called the key.



Non-Relational/NoSQL Databases

NoSQL databases are databases that are non relational.

A large family of databases:

- Key, value storage: Associate a value to a unique identifier called the key.
- Graph databases

Non-Relational/NoSQL Databases

NoSQL databases are databases that are non relational.

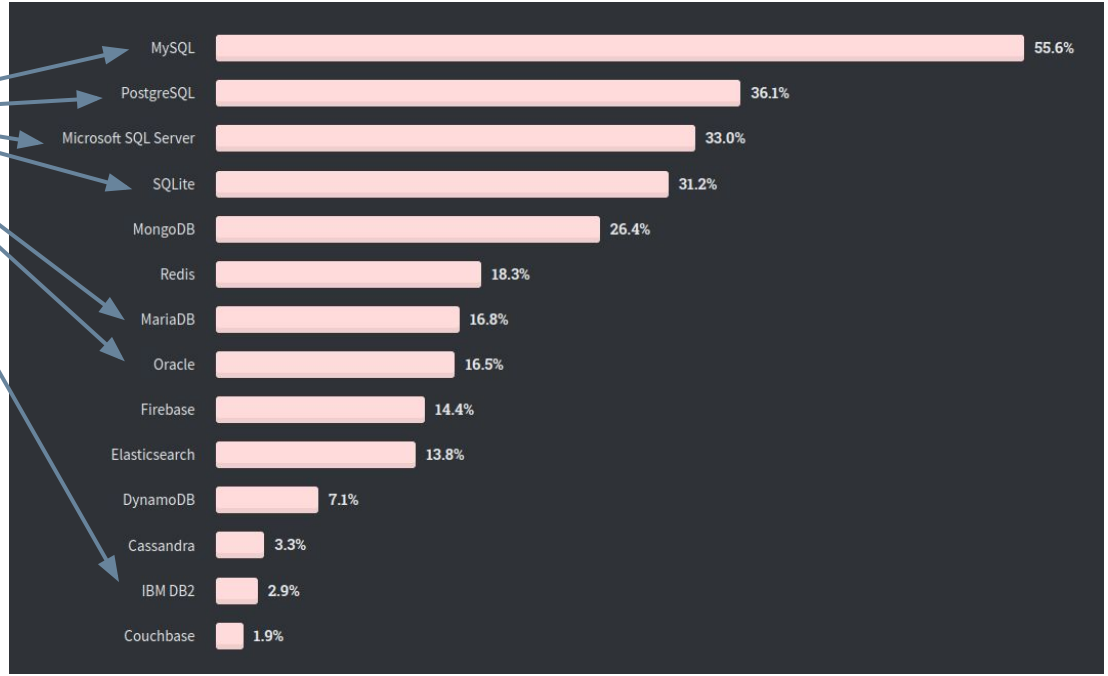
A large family of databases:

- Key, value storage: Associate a value to a unique identifier called the key.
- Graph databases
- Object databases
- Document databases
- ...

They are designed for very specific applications.

Relational Databases VS NoSQL

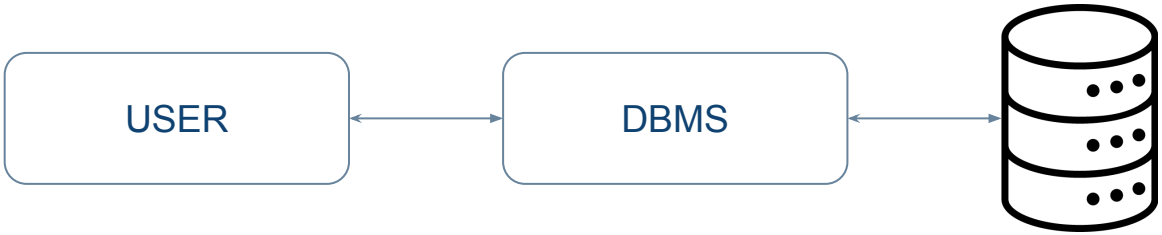
Relational Databases



Source: stackoverflow

Database Management System (DBMS)

A database generally comes with a **Database Management System (DBMS).** A DBMS is the software that actually implements the database model (relational or not) and its interactions with the users. It makes the bridge between the user and the data stored on the disk.



In contrast, the database only represents the (organized) data actually stored.

Database Management System (DBMS)

The most famous DBMS are:

- MySQL
- PostgreSQL
- Microsoft SQL Server
- SQLite

Databases And Big Data

When dealing with a large amount of data, new problems appear:

- Reading and writing become slower
- We need to **distribute** the data, i.e. put it on several computers
- Data is more often unstructured and noisy (raw text, images, ...) = non-relational
- Hard to understand the general organization

Big Data depends on specialized ecosystems (i.e set of softwares)

Most famous = **Hadoop Ecosystem**

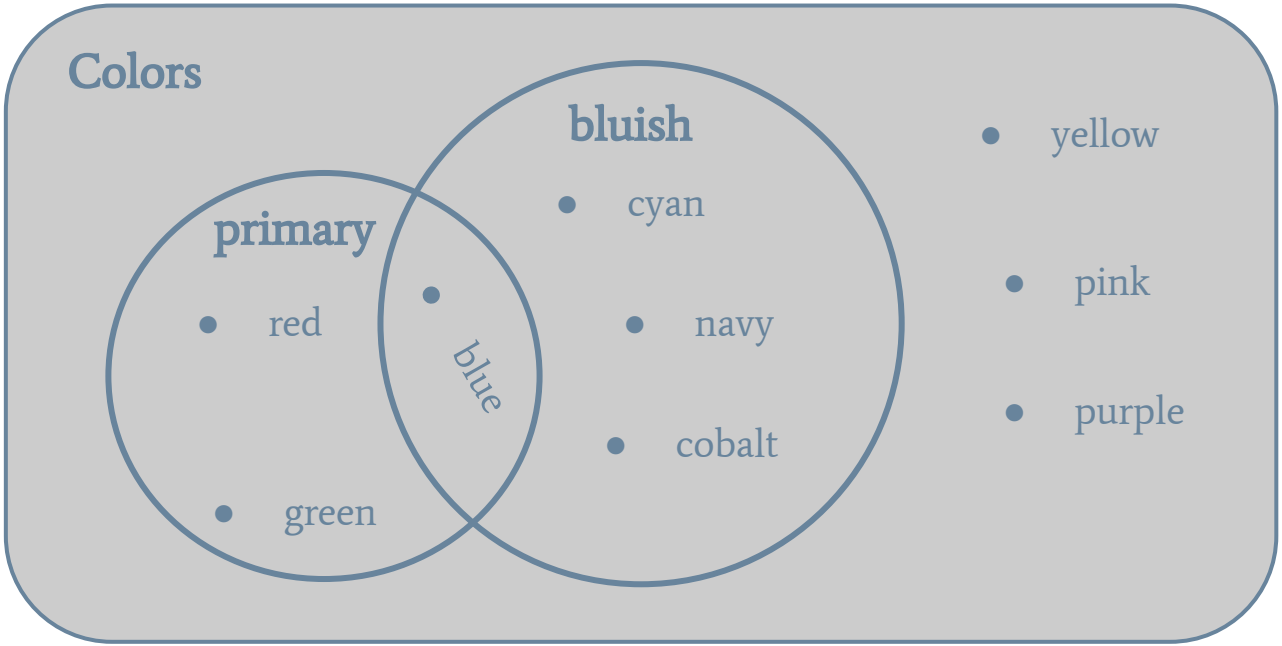
We still have relational databases (with HIVE for example)!

Preliminaries: Set Theory

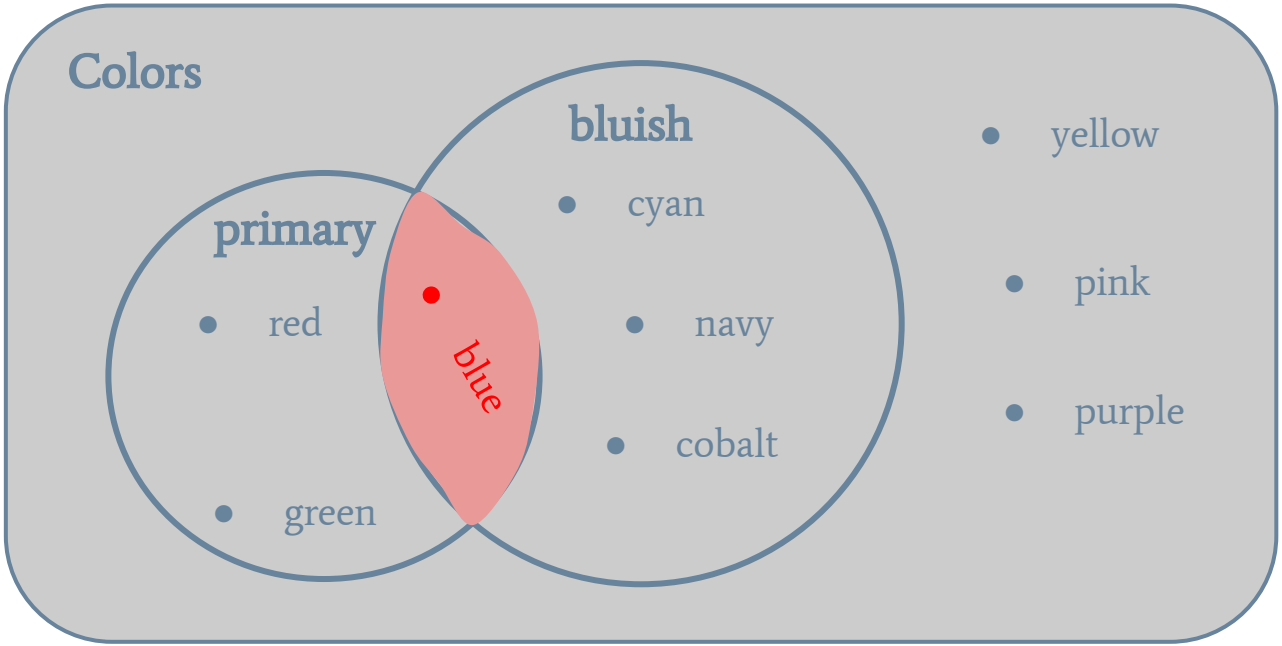
What is a set?

- A set is a group of objects.
- E.g:
  - Set of colors = {red, blue, yellow, pink, purple, green, cyan, cobalt, navy}
  - Set of primary colors = { red, green, blue }
  - Set of bluish colors = {blue, cyan, cobalt, navy}
  - Set of positive integers = {0, 1, 2, 3, 4, ...}
  - Set of even numbers = {0, 2, 4, 6, 8, ...}

Euler Diagram

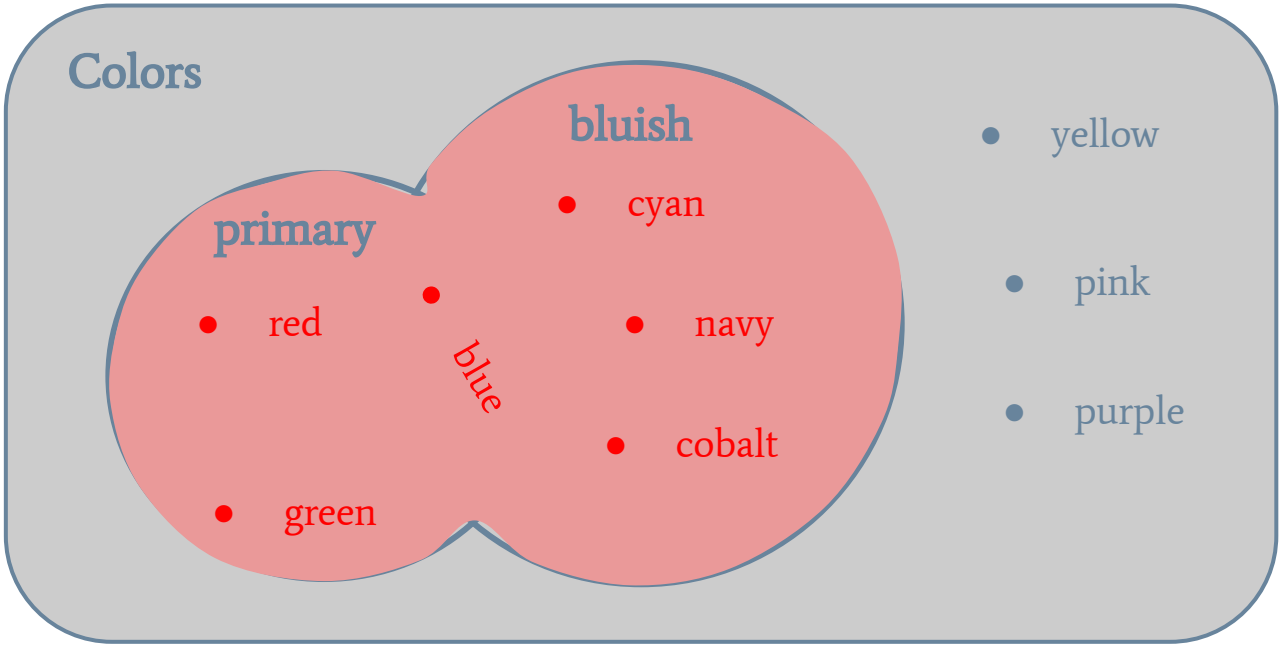


Intersection = Set of elements in both sets



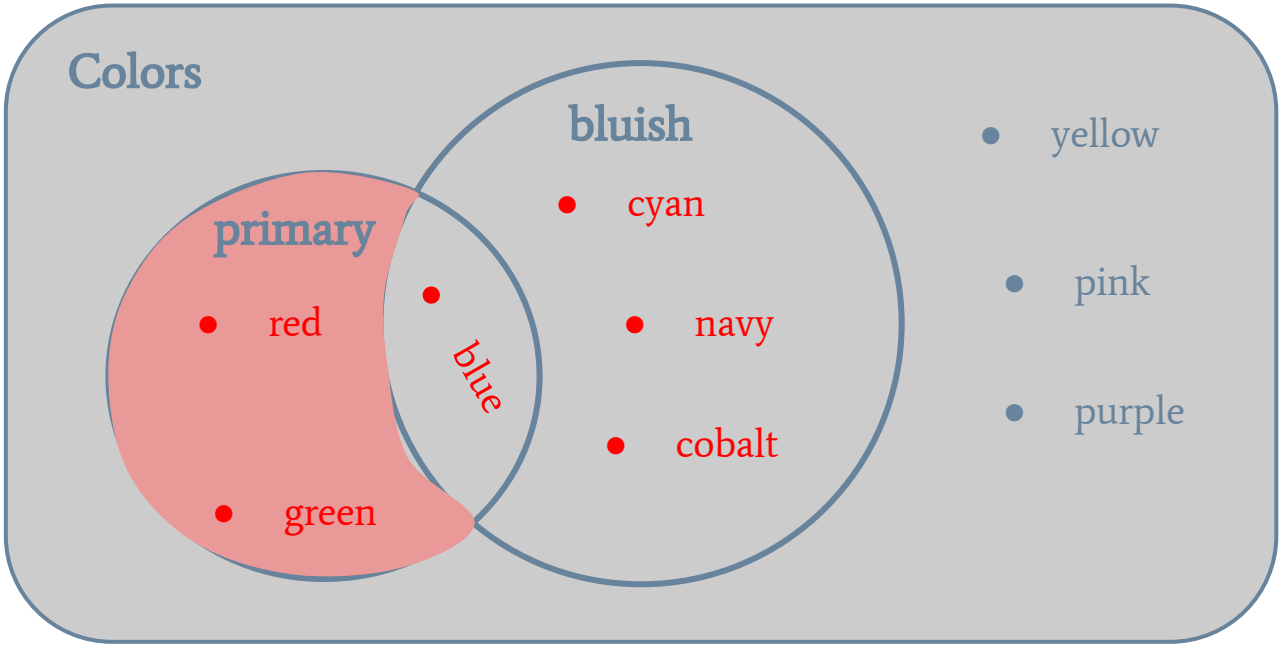
Intersection of primary and bluish = {blue}

Union = Set of elements in at least on set



Union of primary and bluish = {blue, red, green, cyan, navy, cobalt}

Difference = Remove elements of one set from another



Difference of primary - bluish = {red, green}

The Relational Model

Domain

A domain is a set of values.

Examples:

- Integers: 1, 2, 100, 12030, -283, ...
- Real numbers: 3.14, 5.2, -8.42, ...
- Strings: “John”, “Paris”, “The early bird catches the worm”
- Personalized domains: Colors = {“red”, “blue”, “green”}

Cartesian Product

The cartesian product of n domains $D_1, \ldots, D_n$ is the set of all tuples $(v_1, \ldots, v_n)$ such that $v_i \in D_i$. We write it $D_1 \times \ldots \times D_n$

Simple definition: We take all possible combinations

Examples:

- $D_1 = \{1, 2\}$
- $D_2 = \{red, blue, green\}$
- $D_1 \times D_2 = \{(1, red)\,(2, red), (1, blue), (2, blue), (1, green), (2, green)\}$

Relation

A relation is a subset of a the cartesian product of a set of domains.

Simple definition: This is the content (i.e. the rows without their names) of the table where each column can only take the values of its domain.

1	red
2	blue
1	red

Example:

- $D_1 = \{1, 2\}$
- $D_2 = \{red, blue, green\}$
- $D_1 \times D_2 = \{(1, red)\,(2, red), (1, blue), (2, blue), (1, green), (2, green)\}$

Relation

- An element of a relation (i.e. a line or a row) is called a **tuple**
- The name of a column is called the **attribute**.

Example:

- A relation to represent people could be composed of two attributes:
  - A *Name*, with strings as its domain
  - An *Age*, with positive integer as it domain

Name	Age
John	10
Bob	20
John	42

Key

A key is a minimal set of attributes that **identify uniquely** a tuple (or row).

=> A value/set of value can only appear once!

Example:

- Your social security number
- Your email address
- A random number

StudentID	Name	Age
0202	John	10
3453	Bob	20
0192	John	42

Key

Is it a key?

- Your name?
- Your passport number?
- The name of a movie?
- The name of a movie and its director?

Relation Schema

A relation schema describe a table (but is not its content). It is composed of:

- A name (the name of the table)
- A list of attributes with their domains (the columns names and types)
- A list of attributes that represent the key (we underline them)

For a relation name R, attributes $A_1, \ldots, A_n$ with domains $D_1, \ldots, D_n$ and key A_1, we write $R(\underline{A_1 : D_1}, A_2 : D_2, \ldots, A_n : D_n)$

Example: $Student(\underline{StudentID : Integer}, name : String, birthdate : Date)$

Note: If the name of an attribute is ambiguous (shared by several schemas), we write: TableName.AttributeName

Database Schema

A database schema is a set of relation schema.

In short, it is the description of all the tables.

Example: A database about student could contain a table with information about the students and one table with their grades:

$$Student(\underline{StudentID : Integer}, name : String, birthdate : Date)$$

$$Grade(\underline{ExamID : Integer}, \underline{StudentID : Integer}, grade : Integer)$$

Foreign Key

A foreign key is a set of attributes that represents **a key in another relation**.

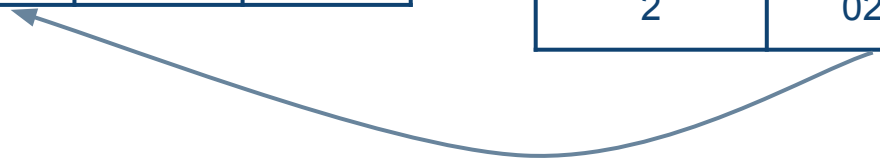
Foreign keys create connections between tables!

Students

StudentID	Name	Age
0202	John	10
3453	Bob	20
0192	John	42

Grades

ExamID	StudentID	Grade
1	0202	10
1	3453	20
2	0202	42



Operations In the Relational Model

PROJECT

The **PROJECT** operation extract given columns from a relation.

Example:

- Query: Get all the grades.

  PROJECT(Grade in the table Grades):

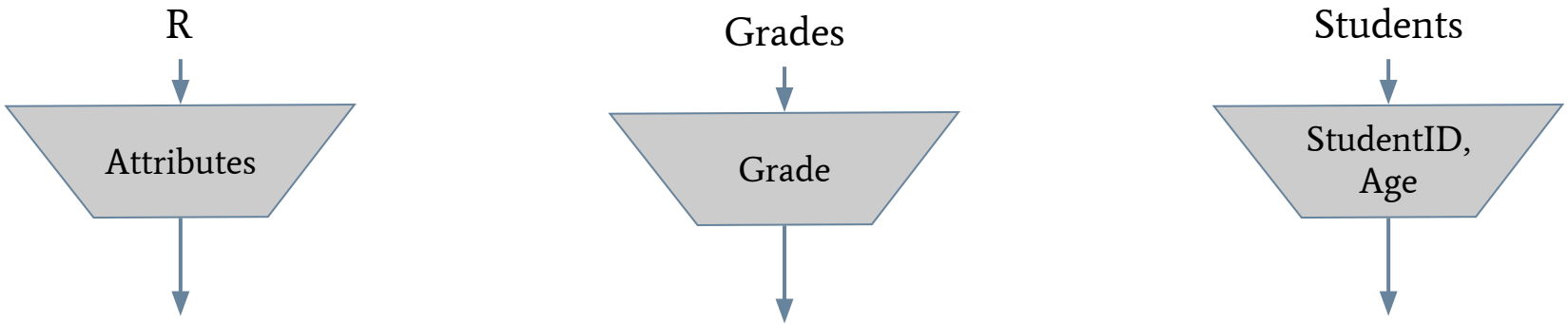
Grade
10
20
42

- Query: Get all the StudentID with the age

  PROJECT(StudentID and Age in Students):

StudentID	Age
0202	10
3453	20
0192	42

PROJECT - Schema



SELECT

The **SELECT** operations allows to filter the rows of a table with a condition.

Example:

- Query: Get all the exams results with a grade >= 20

  SELECT(Grade >= 20 in Grades)

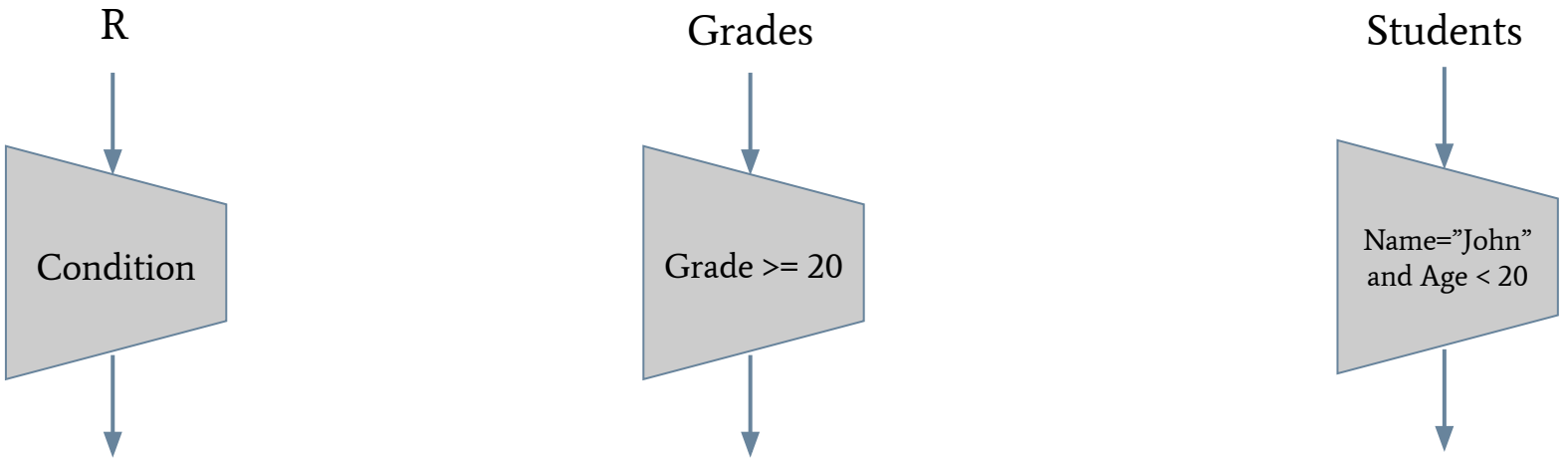
ExamID	StudentID	Grade
1	3453	20
2	0202	42

- Query: Get all the students named John who are less than 20

  SELECT(Name="John" and Age < 20 in Students)

StudentID	Name	Age
0202	John	10

SELECT - Schema



PRODUCT

The **PRODUCT** operation over one or more tables computes the cartesian product of the relations, i.e. all possible combinations of rows.

Students

StudentID	Name
0202	John
3453	Bob

X

Grades

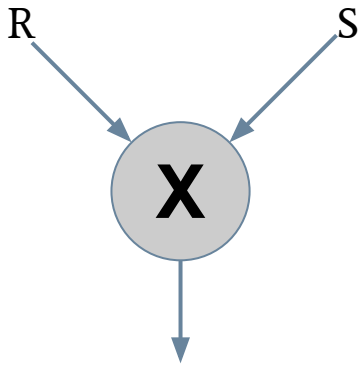
ExamID	StudentID	Grade
1	0202	10
1	3453	20
2	0202	42

=

PRODUCT(Students, Grades) =

ExamID	Grades. StudentID	Grade	Students. StudentsID	Name
1	0202	10	0202	John
1	3453	20	0202	John
2	0202	42	0202	John
1	0202	10	3453	Bob
1	3453	20	3453	Bob
2	0202	42	3453	Bob

PRODUCT - Schema



UNION

The **UNION** operations concatenate two tables that have the **same schema.**

Students

Name
John
Bob

Teachers

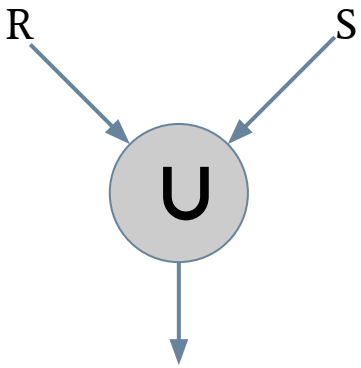
Name
John
Roger

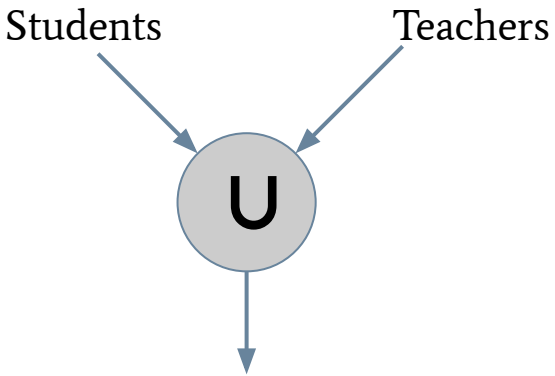
Query: Get the name of all people in the school.

UNION(Students, Teaches) = Students ∪ Teachers =

Name
John
Bob
John
Roger

UNION - Schema

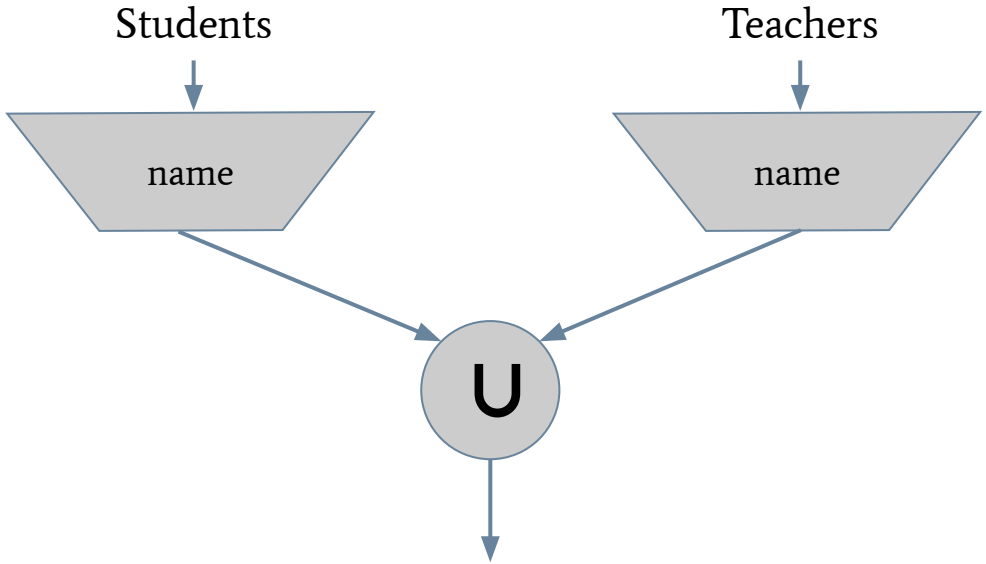




UNION - Different Relation Schema

- Students(StudentID, name, promo)
- Teachers(TeacherID, name, salary)

We cannot make the union directly!



INTERSECTION

The **INTERSECTION** operation takes the rows that are in two tables with **the same schema.**

Students

Name
John
Bob

Teachers

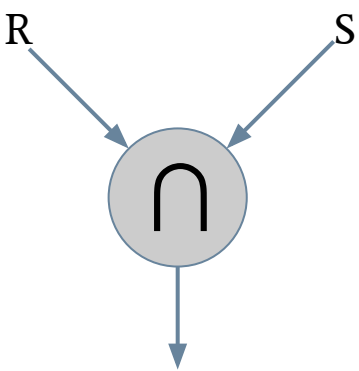
Name
John
Roger

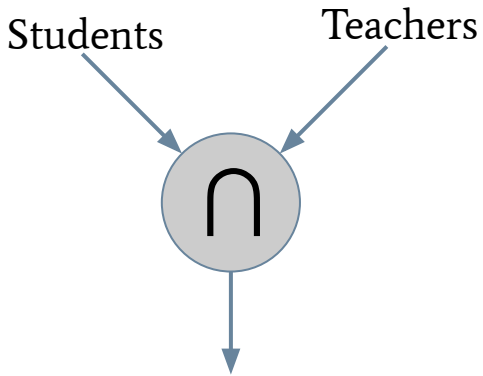
Query: Get the names that are hold by both Students and Teachers

INTERSECTION(Students, Teachers) = Students ∩ Teachers =

Name
John

INTERSECTION - Schema





DIFFERENCE

The **DIFFERENCE** operator removes from a first table rows that are in a second table **with the same schema**.

Students

Name
John
Bob

Teachers

Name
John
Roger

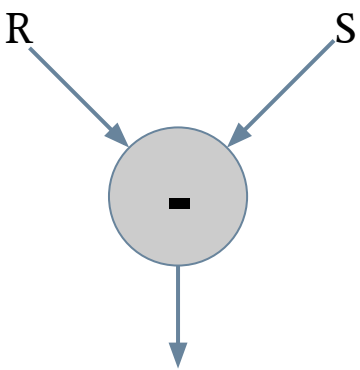
We often have a negation with a DIFFERENCE

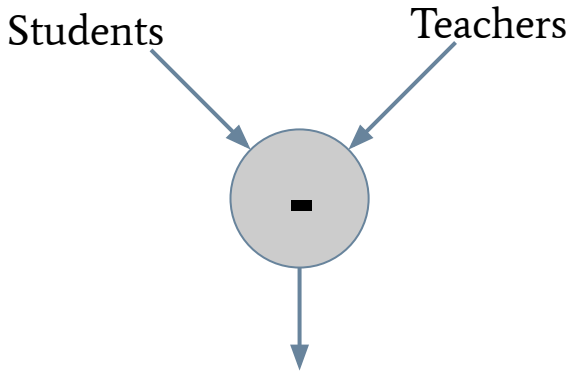
Query: Get the names of students that are **not** teachers' names

DIFFERENCE(Students, Teachers) = Students - Teachers =

Name
Bob

DIFFERENCE - Schema





RENAMING

The **RENAMING** operations allows to rename attributes.

This is very useful in practice if the names become too complex (Students.name -> SName) or if the schema are not the same because of attribute names.

Students

Name
John
Bob

Teachers

Surname
John
Roger

To align the schema, we can do **RENAME**(Name/Surname in Teachers)

RENAME - Schema

JOIN

The **JOIN** operation creates a link between two tables given a condition.

Example:

Students

StudentID	Name
0202	John
3453	Bob

Grades

ExamID	StudentID	Grade
1	0202	10
1	3453	20
2	0202	42

Result

ExamID	Grades. StudentID	Grade	Students. StudentsID	Name
1	0202	10	0202	John
2	0202	42	0202	John
1	3453	20	3453	Bob

JOIN(Students, Grades, Students.StudentID = Grades.StudentID) =

JOIN

The **JOIN** operation is in fact a **PRODUCT** followed by a **SELECT**

JOIN(Students, Grades, Students.StudentID = Grades.StudentID) =

SELECT(Students.StudentID = Grades.StudentID, **PRODUCT**(Students, Grades))

JOIN - PRODUCT

The **JOIN** operation is in fact a **PRODUCT** followed by a **SELECT**

Students

StudentID	Name
0202	John
3453	Bob

Grades

ExamID	StudentID	Grade
1	0202	10
1	3453	20
2	0202	42

ExamID	Grades. StudentID	Grade	Students. StudentsID	Name
1	0202	10	0202	John
1	3453	20	0202	John
2	0202	42	0202	John
1	0202	10	3453	Bob
1	3453	20	3453	Bob
2	0202	42	3453	Bob

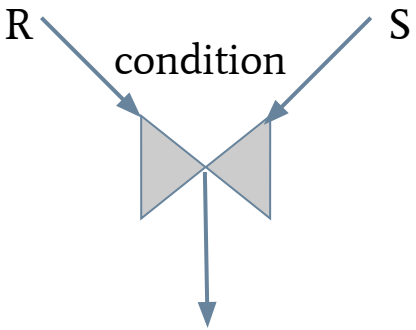
JOIN - SELECT

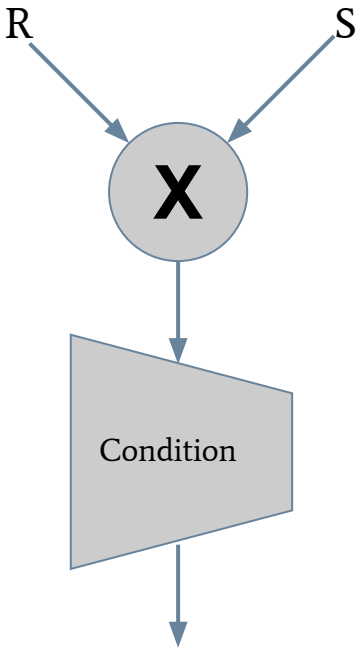
The **JOIN** operation is in fact a **PRODUCT** followed by a **SELECT**

ExamID	Grades. StudentID	Grade	Students. StudentsID	Name
1	0202	10	0202	John
1	3453	20	0202	John
2	0202	42	0202	John
1	0202	10	3453	Bob
1	3453	20	3453	Bob
2	0202	42	3453	Bob

ExamID	Grades. StudentID	Grade	Students. StudentsID	Name
1	0202	10	0202	John
2	0202	42	0202	John
1	3453	20	3453	Bob

JOIN - Schema

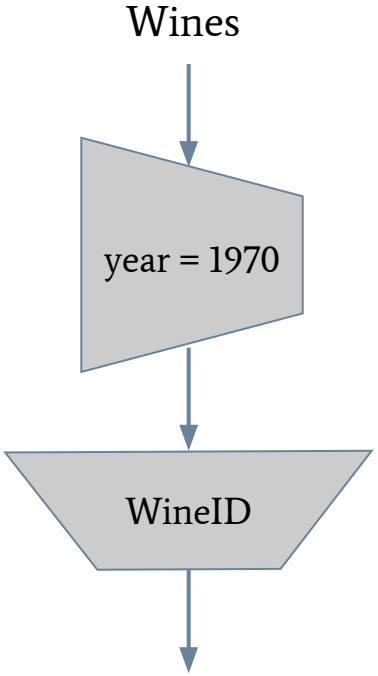




Combining Operations

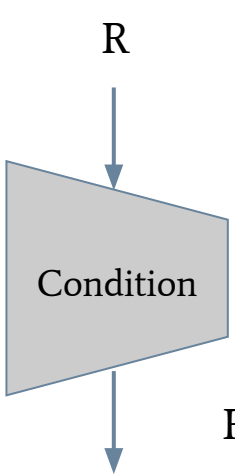
The relational operations become very powerful when combined to get the wanted information.

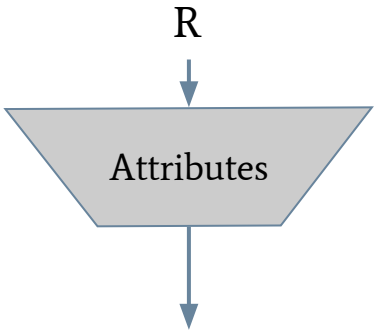
Get all the wine IDs of wines produced in 1970.
We can use two representations: the textual representation or the schema representation

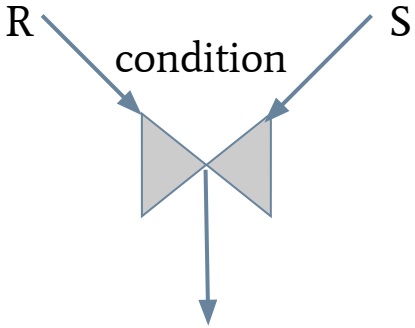


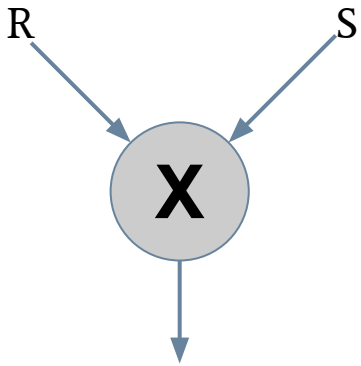
Result = **PROJECT**(WineID, **SELECT**(year = 1970 in Wine))
Or (easier to read if many operations):
Temp = **SELECT**(year = 1970 in Wine)
Result = **PROJECT**(WineID, Temp)

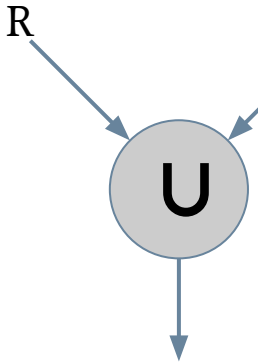
List of the Operators

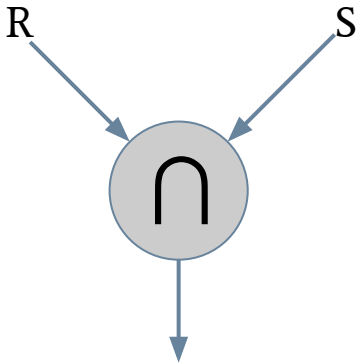


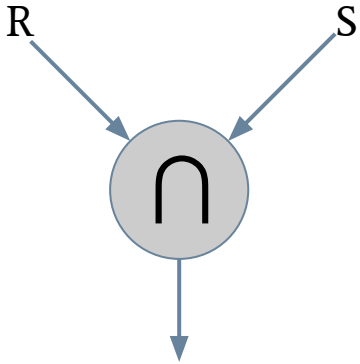


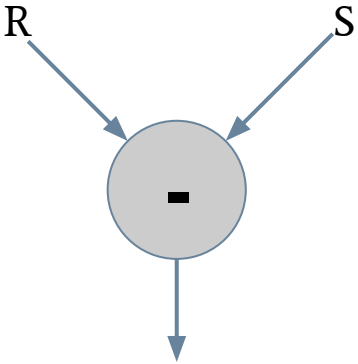


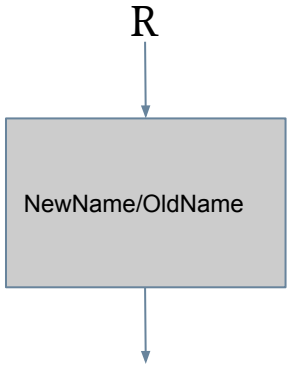












Methodology

- Identify the tables you need
  - Tables explicitly mentioned in the query
  - Tables to connect tables
- Identify what you need to output
- Filter (with **SELECT**) and keep only the columns you need (with **PROJECT**)
- Identify the links you need to create and the conditions (JOIN) or operations (UNION, DIFFERENCE, INTERSECTION)
- Filter and keep only the columns you need as you progress

If the query is too complex, try to decompose it into easier queries.

There is not a single solution!

Review Of Concepts

- What databases are and how they are integrated in an information system
- The different types of databases: files, relational databases, NoSQL
- The functionalities of a database: Query, Storying; Organizing, and Security
- Concepts of relational databases:
  - Table, row/tuple, column/attribute
  - Domain, cartesian product, relation
  - Key, foreign key
  - Relation schema, database schema
- Operations of relational algebra
  - Project, select, product
  - Set operations: union, intersection, difference (need the same schema!)
  - Rename
  - Join (= Product + Select)

Combining Operations

The relational operations become very powerful when combined to get the wanted information.

Example: We have the following database schema

- Wines(WineID, vineyard, year, degree)
- Harvests(WineID, ProducerID, weight)
- Producers(ProducerID, name, city)
- Clients(ClientID, name, city)
- Orders(OrderID, date, ClientID, WineID, quantity)
- Deliveries(OrderID, status)

Exercices

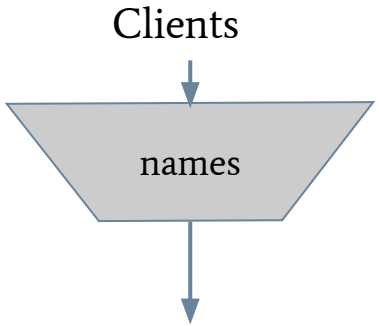
- Queries:
  - Get the name of the producers of Muscadet (type of vineyard)
  - Get the wines that were never ordered
  - Get the names of clients who live in Paris and order before 2000 at least a bottle of Macon (vineyard) from 1995
- Database schema:
  - Wines(WineID, vineyard, year, degree)
  - Harvests(WineID, ProducerID, weight)
  - Producers(ProducerID, name, city)
  - Clients(ClientID, name, city)
  - Orders(OrderID, date, ClientID, WineID, quantity)
  - Deliveries(OrderID, status)

AR Symbols

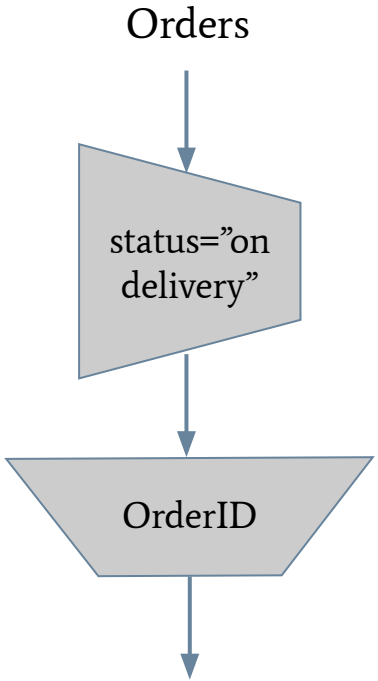
Additional Exercises

- Get the name of the clients who are currently being delivered (status = “delivering”)
- Get the names of producers living in Paris who are also clients
- Get the names of all the cities involved in the wine business
- Get the names of the cities where a producer harvested more that 1 ton of grapes.
- Get the names of the clients who bought a wine produced in Paris

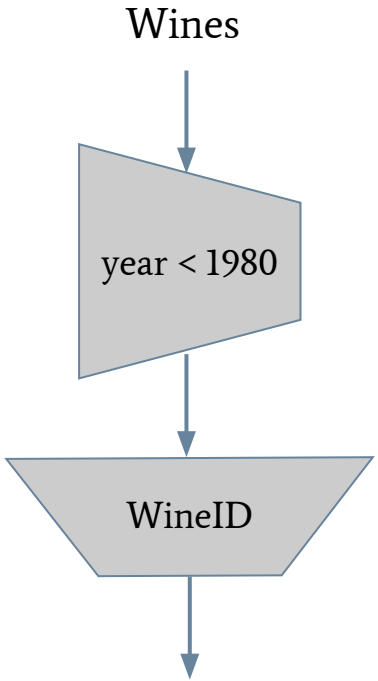
Get the name of all the clients.



Give the OrderID of all orders currently on delivery (status="on delivery").



Get the name of the WineIDs of wines produced before 1980.

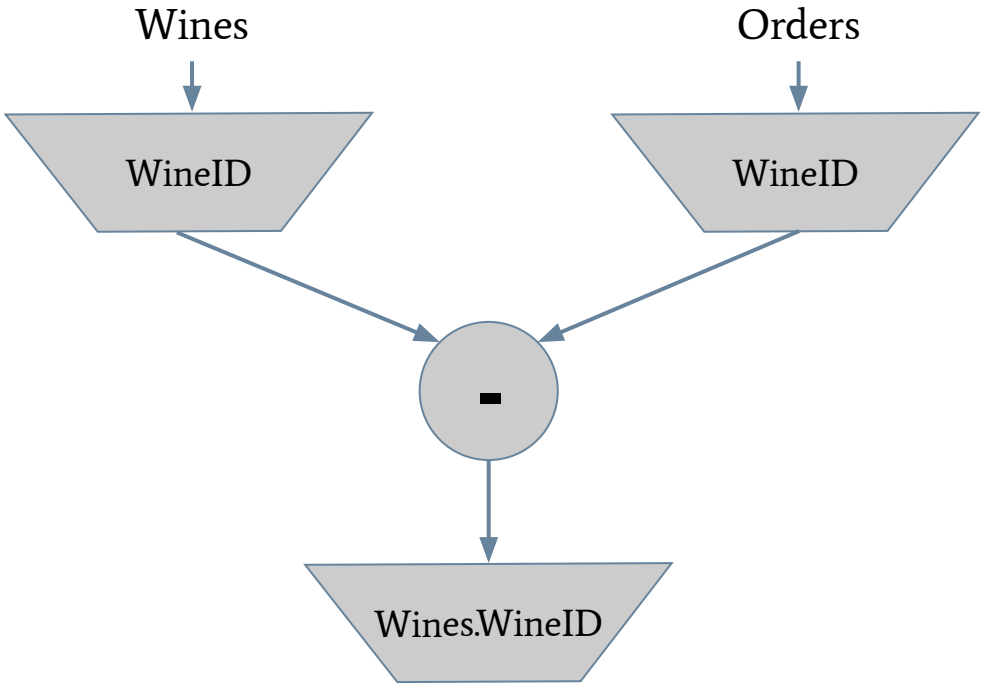


Solution - Get the wines that were never ordered

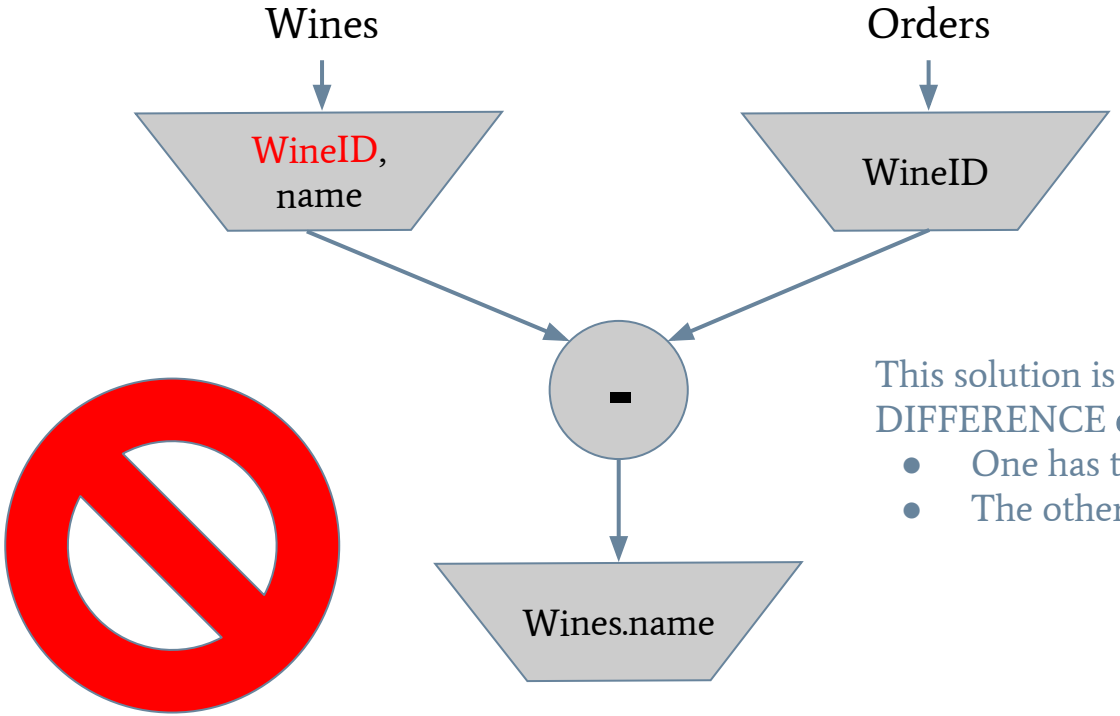
Methodology:

- Identify the tables you need
  - Tables explicitly mentioned in the query: Wines, Orders
  - Tables to connect tables: None
- Identify what you need to output: WineIDs from Wines
- Filter (with **SELECT**) and keep only the columns you need (with **PROJECT**)
  - We need WineID from Orders and Wines
- Identify the links you need to create and the conditions/operations:
  - The WineID attributes connect Orders and Wines
  - We need to remove WinesID in Orders from WinesID in Wines (DIFFERENCE)
- Filter and keep only the columns you need as you progress

Solution - Get the wines that were never ordered



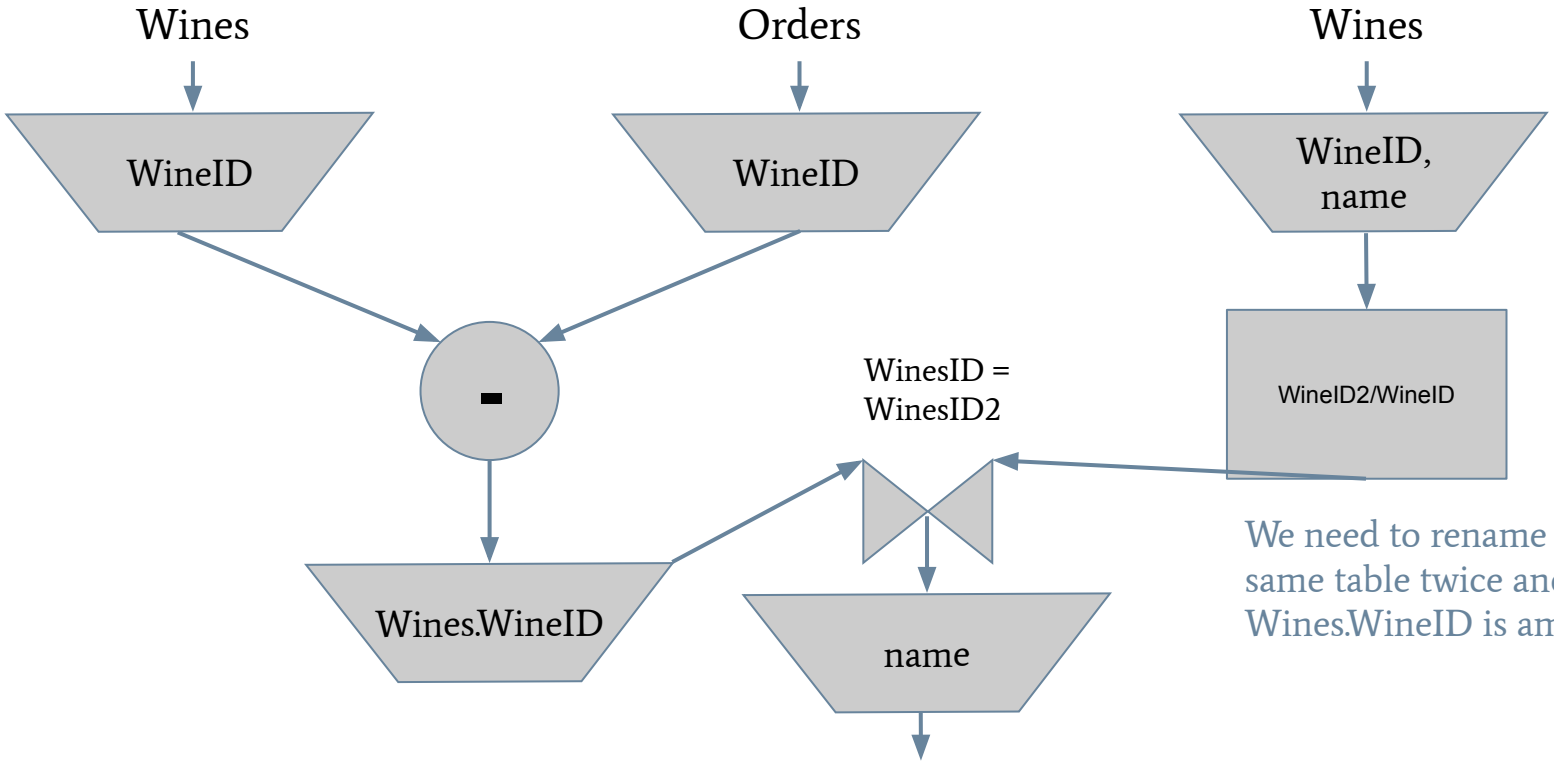
Solution - Get the **names** of the wines that were never ordered



This solution is **incorrect** has the two tables of the DIFFERENCE **do not have the same schema:**

- One has two columns (WineID and name)
- The other has one column (WineID)

Solution - Get the **names** of the wines that were never ordered



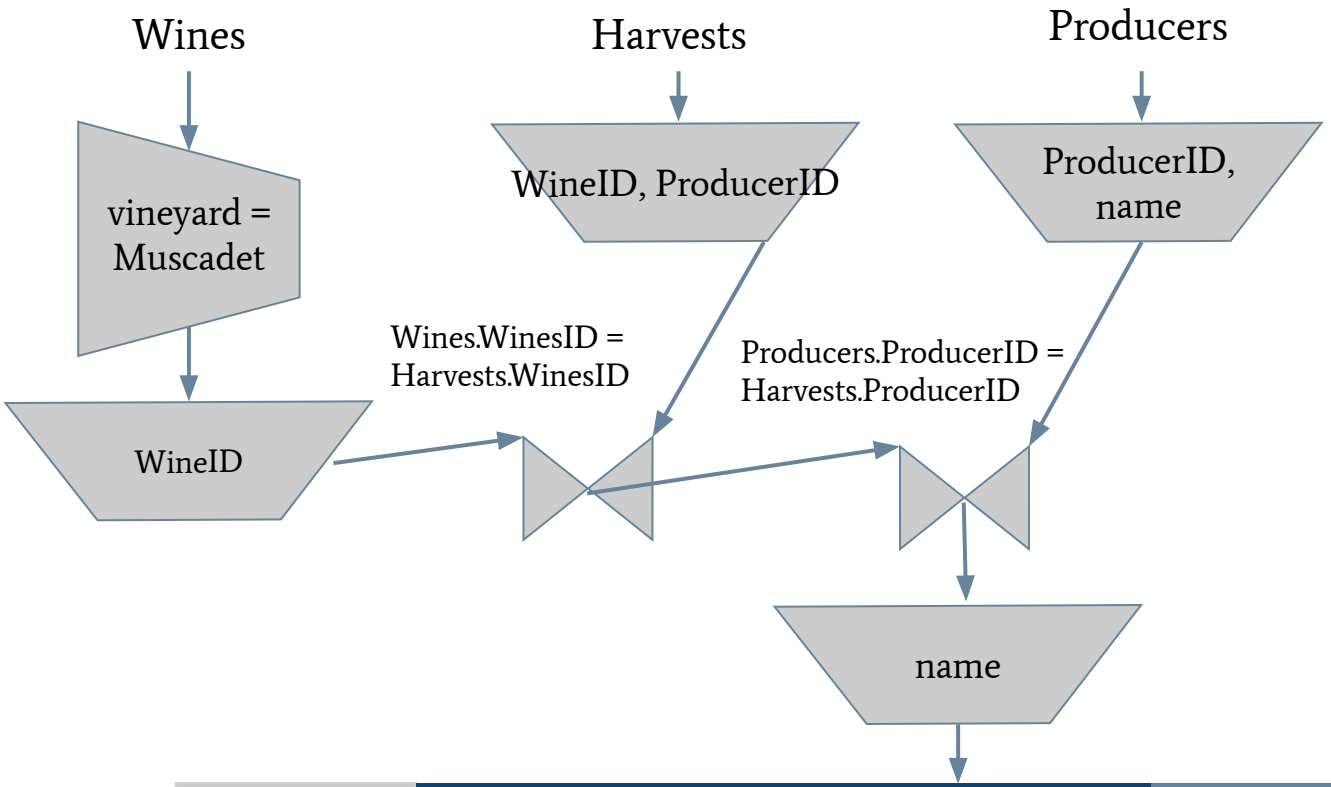
We need to rename as we use the same table twice and, therefore, Wines.WineID is ambiguous

Solution - Get the name of the producers of Muscadet

Methodology:

- Identify the tables you need
  - Tables explicitly mentioned in the query: Producers and Wines (for the vineyard)
  - Tables to connect tables: Harvest connect Producers and Wines
- Identify what you need to output: We want Producers.name
- Filter (with **SELECT**) and keep only the columns you need (with **PROJECT**)
  - In Wines, we need the WineID (to link with Harvest) and vineyard (to pick Muscadet)
  - In Harvest, we need WineID and ProducerID to connect Producers and Wines
  - In Producer, we need ProducerID (to link with Harvest) and name (the output)
- Identify the links you need to create and the conditions/operations
  - We link Wines and Harvest with the commun column WineID
  - We link Producers and Harvest with the commun ProducerID
- Filter and keep only the columns you need as you progress
  - We need to filter the final result to get only the name in Producer

Solution - Get the name of the producers of Muscadet

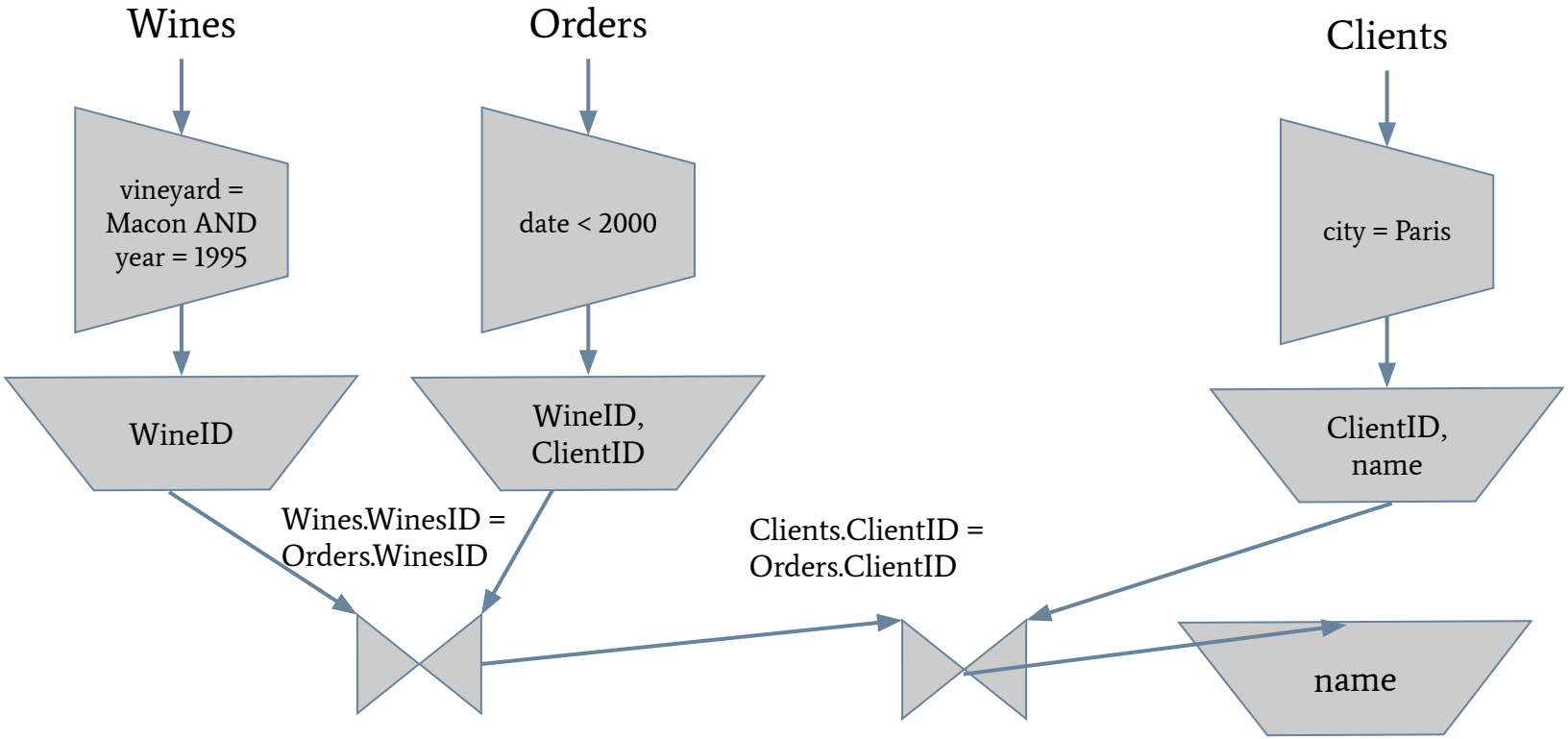


Solution - Get the names of clients who live in Paris and order before 2000 at least a bottle of Macon (vineyard) from 1995

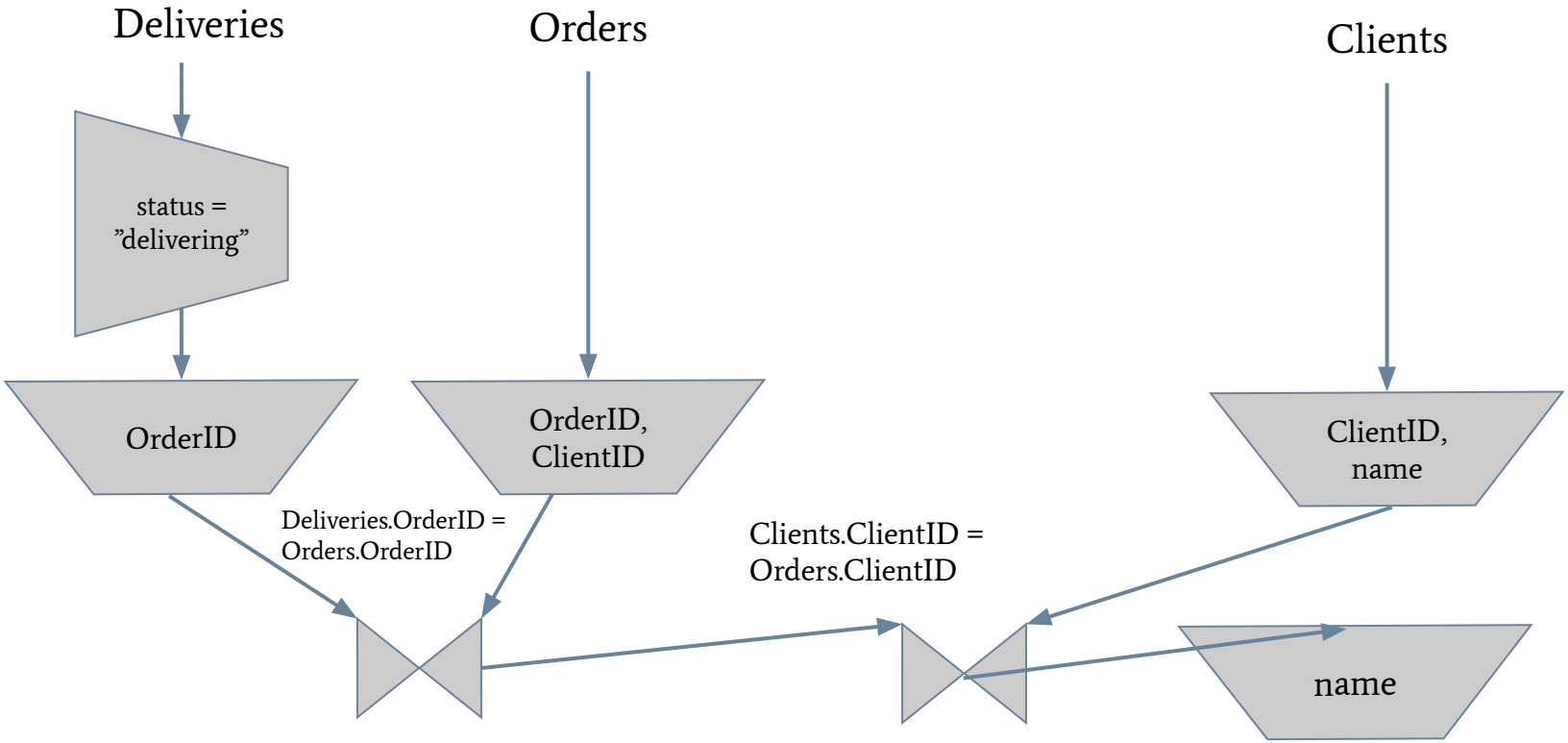
Methodology:

- Identify the tables you need
  - Tables explicitly mentioned in the query: Clients, Wines and Orders
  - Tables to connect tables: None
- Identify what you need to output: We want Client.name
- Filter (with **SELECT**) and keep only the columns you need (with **PROJECT**)
  - In Wines, we need the WineID (to link with Orders), vineyard (to pick Macon) and year (to pick 1995)
  - In Orders, we need WineID and ClientID to connect Clients and Wines
  - In Clients, we need ClientID (to link with Orders), name (the output) and city (to pick Paris)
- Identify the links you need to create and the conditions/operations
  - We link Wines and Orders with the commun column WineID
  - We link Clients and Orders with the commun ClientID
- Filter and keep only the columns you need as you progress
  - We need to filter the final result to get only the name in Client

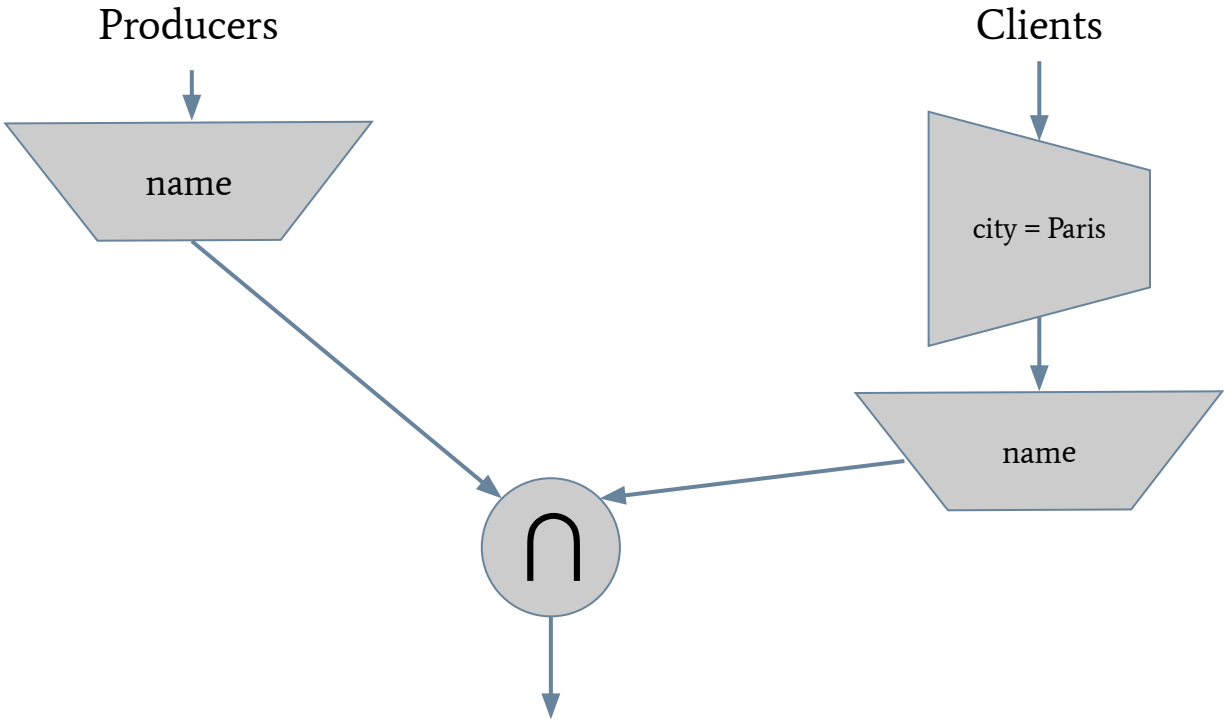
Solution - Get the names of clients who live in Paris and order before 2000 at least a bottle of Macon (vineyard) from 1995



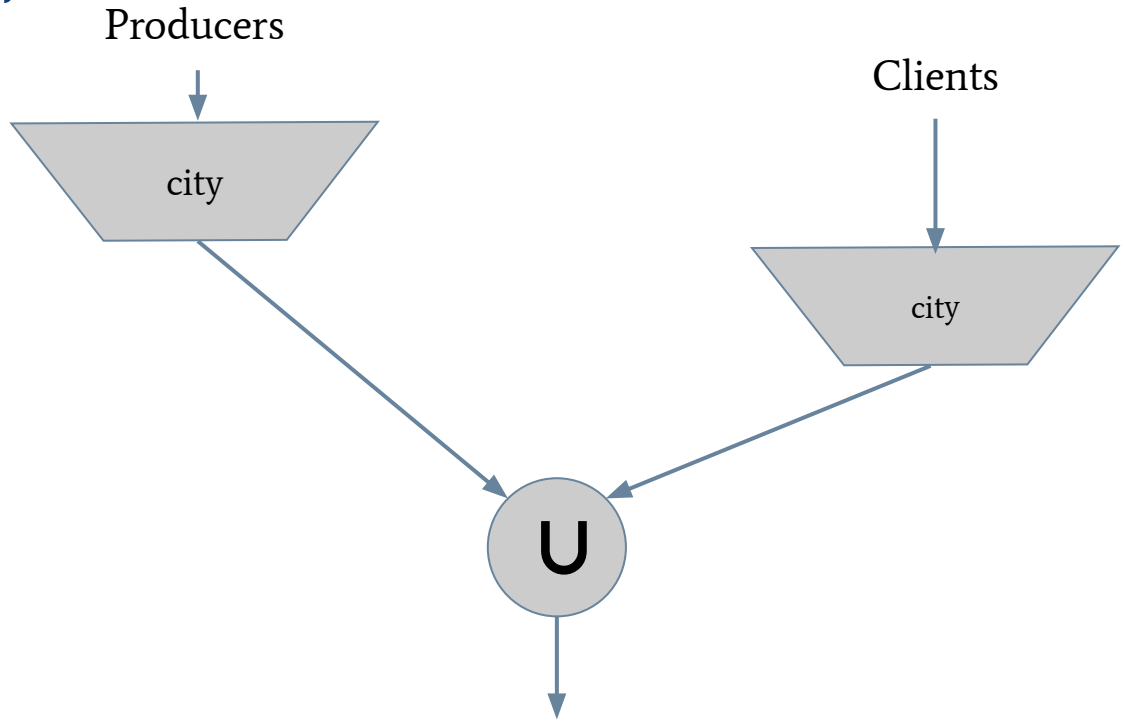
Get the name of the clients who are currently being delivered (status = “delivering”)



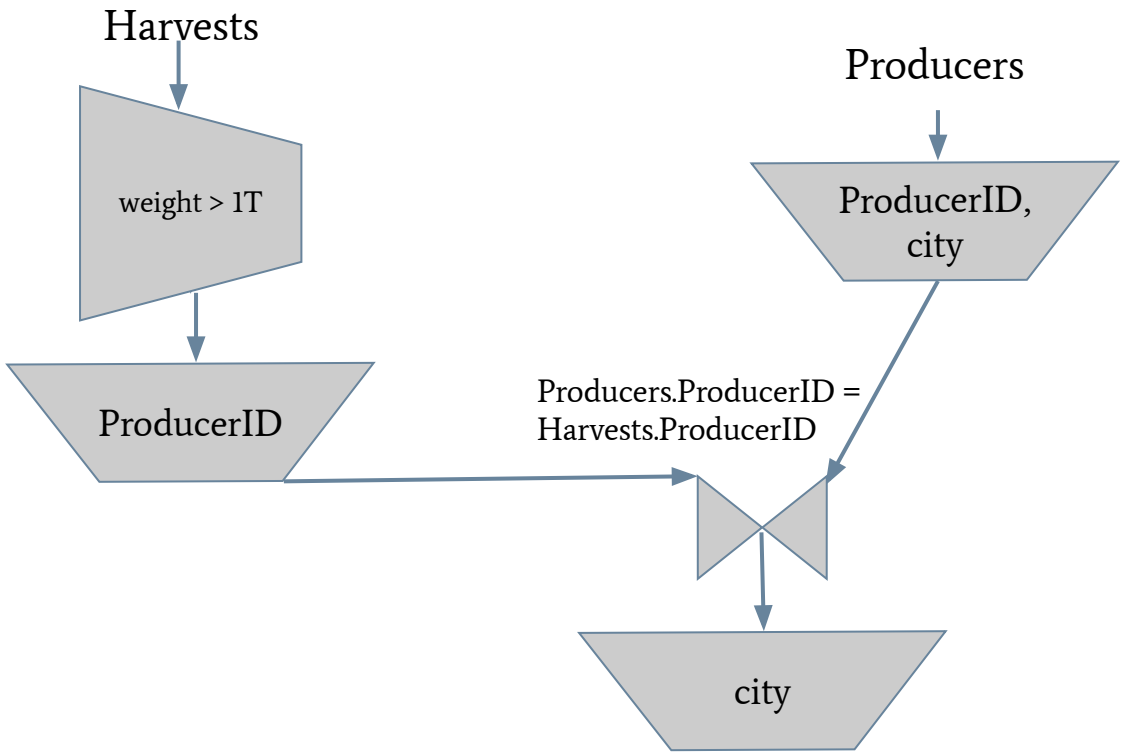
Get the names of producers living in Paris who are also clients



Get the names of all the cities involved in the wine business (i.e. where a producer or a client lives)



Get the names of the cities where a producer harvested more that 1 ton of grapes in one time.



Get the names of the clients who bought a wine produced in Paris