

Deliverable 2

Team 1 :

AJROUCHE Maël

DELAHAYE Adrien

LE GALL Paul

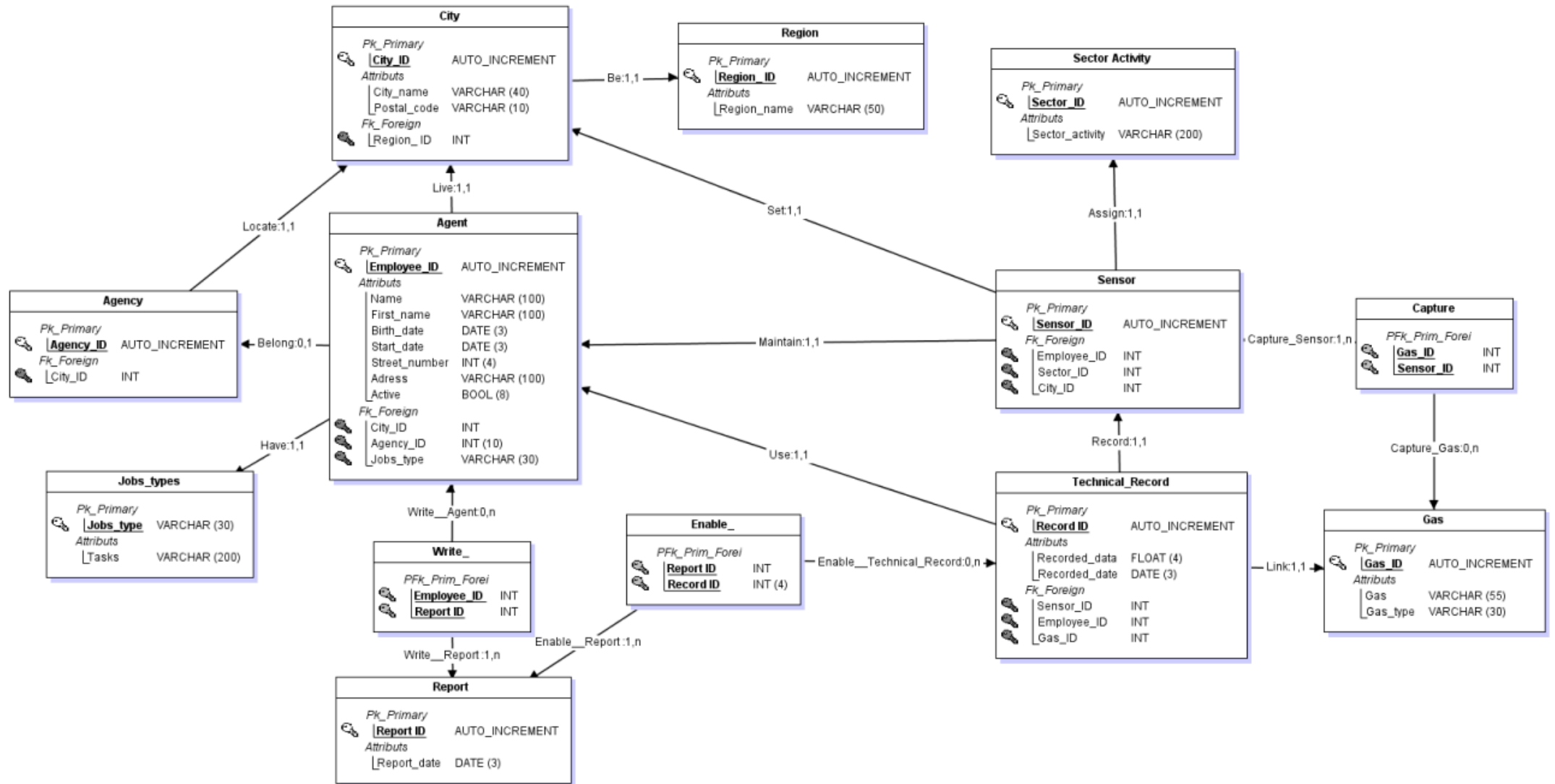
PHAN CONG Laurent



Table des matières

| | |
|-------------------------------------|----|
| Changes to Deliverable 1 | 2 |
| Creation of all tables..... | 4 |
| Populating the Database | 8 |
| For the table region:..... | 8 |
| For the table city: | 9 |
| For the table agency:..... | 10 |
| For the table gas:..... | 11 |
| For the table job types: | 11 |
| For the table sector activity | 11 |
| SQL Queries (12 queries) | 12 |
| Several specifications | 15 |
| User and root accounts | 15 |
| Storage engine | 16 |

Changes to Deliverable 1



We didn't make any big changes in the structure, just some renaming, variable type changes and size increases. These changes improve the clarity, consistency, and efficiency of the database structure, while avoiding potential problems associated with the use of SQL reserved words:

1. City Table:

The name of the "City" column has been changed to "City_name". This change aims to make the column name more explicit and avoid potential confusion when writing SQL queries.

2. Region Table:

Two modifications have been made on this table. Firstly, the name of the "Region" column has been changed to "Region_name" for added clarity. Secondly, "Region__ID" has been corrected to "Region_ID", removing the extra underscore for more consistency in column naming.

3. Jobs Types Table:

The name of the "Jobs_types" column has been modified to "Jobs_type" to reflect that a single job type value will be stored per row. In addition, the type of the "Tasks" column has been changed from TEXT(100) to VARCHAR(200), allowing for larger character strings to be stored.

4. Sector Activity Table:

A new "Sector_ID" field has been added as a primary key to uniquely identify each record in the table. In addition, the length limit of the "Sector_activity" column has been increased from VARCHAR(65) to VARCHAR(200) to allow for longer values to be recorded.

5. Write intermediary Table:

This table has been renamed to "Write_" as "Write" is a reserved word in the SQL language, which can cause errors or confusion when writing queries.

6. Enable intermediary Table:

Similarly, to the Write table, this table has been renamed to "Enable_" to avoid conflicts with SQL reserved words.

Creation of all tables

```
#-----
# Table: Gas
#-----

CREATE TABLE Gas(
    Gas_ID    Int    Auto_increment NOT NULL ,
    Gas       Varchar (55) NOT NULL ,
    Gas_type  Varchar (30) NOT NULL
    ,CONSTRAINT Gas_PK PRIMARY KEY (Gas_ID)
)ENGINE=InnoDB;

#-----
# Table: Report
#-----

CREATE TABLE Report(
    Report_ID  Int    Auto_increment NOT NULL ,
    Report_date Date NOT NULL
    ,CONSTRAINT Report_PK PRIMARY KEY (Report_ID)
)ENGINE=InnoDB;

#-----
# Table: Sector Activity
#-----

CREATE TABLE Sector_Activity(
    Sector_ID    Int    Auto_increment NOT NULL ,
    Sector_activity Varchar (200) NOT NULL
    ,CONSTRAINT Sector_Activity_PK PRIMARY KEY (Sector_ID)
)ENGINE=InnoDB;

#-----
# Table: Jobs_types
#-----

CREATE TABLE Jobs_types(
    Jobs_type Varchar (30) NOT NULL ,
    Tasks     Varchar (200) NOT NULL
    ,CONSTRAINT Jobs_types_PK PRIMARY KEY (Jobs_type)
)ENGINE=InnoDB;

#-----
# Table: Region
#-----

CREATE TABLE Region(
    Region_ID  Int    Auto_increment NOT NULL ,
    Region_name Varchar (50)
    ,CONSTRAINT Region_PK PRIMARY KEY (Region_ID)
)ENGINE=InnoDB;
```

```

#-----
# Table: City
#-----

CREATE TABLE City(
    City_ID      Int  Auto_increment NOT NULL ,
    City_name    Varchar (40) NOT NULL ,
    Postal_code  Varchar (10) NOT NULL ,
    Region_ID    Int  NOT NULL
    ,CONSTRAINT City_PK PRIMARY KEY (City_ID)

    ,CONSTRAINT City_Region_FK FOREIGN KEY (Region_ID) REFERENCES
Region(Region_ID)
)ENGINE=InnoDB;

#-----
# Table: Agency
#-----

CREATE TABLE Agency(
    Agency_ID Int  Auto_increment NOT NULL ,
    City_ID   Int  NOT NULL
    ,CONSTRAINT Agency_PK PRIMARY KEY (Agency_ID)

    ,CONSTRAINT Agency_City_FK FOREIGN KEY (City_ID) REFERENCES City(City_ID)
)ENGINE=InnoDB;

#-----
# Table: Agent
#-----

CREATE TABLE Agent(
    Employee_ID  Int  Auto_increment NOT NULL ,
    Name         Varchar (100) NOT NULL ,
    First_name   Varchar (100) NOT NULL ,
    Birth_date   Date NOT NULL ,
    Start_date   Date NOT NULL ,
    Street_number Int NOT NULL ,
    Adress       Varchar (100) NOT NULL ,
    Active       Bool NOT NULL ,
    City_ID      Int  NOT NULL ,
    Agency_ID    Int ,
    Jobs_type    Varchar (30) NOT NULL
    ,CONSTRAINT Agent_PK PRIMARY KEY (Employee_ID)

    ,CONSTRAINT Agent_City_FK FOREIGN KEY (City_ID) REFERENCES City(City_ID)
    ,CONSTRAINT Agent_Agency0_FK FOREIGN KEY (Agency_ID) REFERENCES
Agency(Agency_ID)
    ,CONSTRAINT Agent_Jobs_types1_FK FOREIGN KEY (Jobs_type) REFERENCES
Jobs_types(Jobs_type)
)ENGINE=InnoDB;

```

```

#-----
# Table: Sensor
#-----

CREATE TABLE Sensor(
    Sensor_ID      Int  Auto_increment NOT NULL ,
    Employee_ID    Int  NOT NULL ,
    Sector_ID      Int  NOT NULL ,
    City_ID        Int  NOT NULL
    ,CONSTRAINT Sensor_PK PRIMARY KEY (Sensor_ID)

    ,CONSTRAINT Sensor_Agent_FK FOREIGN KEY (Employee_ID) REFERENCES
Agent(Employee_ID)
    ,CONSTRAINT Sensor_Sector_Activity0_FK FOREIGN KEY (Sector_ID) REFERENCES
Sector_Activity(Sector_ID)
    ,CONSTRAINT Sensor_City1_FK FOREIGN KEY (City_ID) REFERENCES City(City_ID)
)ENGINE=InnoDB;

#-----
# Table: Technical_Record
#-----

CREATE TABLE Technical_Record(
    Record_ID      Int  Auto_increment NOT NULL ,
    Recorded_data   Float NOT NULL ,
    Recorded_date   Date NOT NULL ,
    Sensor_ID      Int  NOT NULL ,
    Employee_ID    Int  NOT NULL ,
    Gas_ID         Int  NOT NULL
    ,CONSTRAINT Technical_Record_PK PRIMARY KEY (Record_ID)

    ,CONSTRAINT Technical_Record_Sensor_FK FOREIGN KEY (Sensor_ID) REFERENCES
Sensor(Sensor_ID)
    ,CONSTRAINT Technical_Record_Agent0_FK FOREIGN KEY (Employee_ID) REFERENCES
Agent(Employee_ID)
    ,CONSTRAINT Technical_Record_Gas1_FK FOREIGN KEY (Gas_ID) REFERENCES
Gas(Gas_ID)
)ENGINE=InnoDB;

#-----
# Table: Capture
#-----

CREATE TABLE Capture(
    Gas_ID         Int  NOT NULL ,
    Sensor_ID      Int  NOT NULL
    ,CONSTRAINT Capture_PK PRIMARY KEY (Gas_ID,Sensor_ID)

    ,CONSTRAINT Capture_Gas_FK FOREIGN KEY (Gas_ID) REFERENCES Gas(Gas_ID)
    ,CONSTRAINT Capture_Sensor0_FK FOREIGN KEY (Sensor_ID) REFERENCES
Sensor(Sensor_ID)
)ENGINE=InnoDB;

```

```

#-----
# Table: Write_
#-----

CREATE TABLE Write_(
    Employee_ID Int NOT NULL ,
    Report_ID Int NOT NULL
    ,CONSTRAINT Write__PK PRIMARY KEY (Employee_ID,Report_ID)

    ,CONSTRAINT Write__Agent_FK FOREIGN KEY (Employee_ID) REFERENCES
Agent(Employee_ID)
    ,CONSTRAINT Write__Report0_FK FOREIGN KEY (Report_ID) REFERENCES
Report(Report_ID)
)ENGINE=InnoDB;

#-----
# Table: Enable_
#-----

CREATE TABLE Enable_(
    Report_ID Int NOT NULL ,
    Record_ID Int NOT NULL
    ,CONSTRAINT Enable__PK PRIMARY KEY (Report_ID,Record_ID)

    ,CONSTRAINT Enable__Report_FK FOREIGN KEY (Report_ID) REFERENCES
Report(Report_ID)
    ,CONSTRAINT Enable__Technical_Record0_FK FOREIGN KEY (Record_ID) REFERENCES
Technical_Record(Record_ID)
)ENGINE=InnoDB;

```

Populating the Database

Script del all bdd

```

DROP TABLE Write_;
DROP TABLE Enable_;
DROP TABLE Report;
DROP TABLE Capture;
DROP TABLE Technical_record;
DROP TABLE Gas;
DROP TABLE Sensor;
DROP TABLE Sector_activity;
DROP TABLE Agent;
DROP TABLE Jobs_types;
DROP TABLE Agency;
DROP TABLE City;
DROP TABLE Region;

```

We decided to put:

- 10 000 Technical reports
- 250 Employees
- 200 Sensors
- 100 Cities
- 50 Reports
- 25 Agencies

For the table region:

We took all the regions of France, wich is about 18 regions, to fill this table:

```

DELETE FROM livrable.Region;
ALTER TABLE livrable.Region AUTO_INCREMENT = 1;

INSERT INTO livrable.Region (Region_name) VALUES
('Auvergne-Rhône-Alpes'),
('Bourgogne-Franche-Comté'),
('Bretagne'),
('Centre-Val de Loire'),
('Corse'),
('Grand Est'),
('Hauts-de-France'),
('Île-de-France'),
('Normandie'),
('Nouvelle-Aquitaine'),
('Occitanie'),
('Pays de la Loire'),
('Provence-Alpes-Côte d\'Azur'),
('La Réunion'),
('Martinique'),
('Mayotte'),
('Guyane'),
('Guadeloupe');

```


For the table city:

We have inserted the 100 largest cities in France through a temporary table. We create it to add the data and then link the cities with the id of the regions. Then we delete it at the end:

```
DROP TEMPORARY TABLE IF EXISTS Temp_City_Region;

CREATE TEMPORARY TABLE Temp_City_Region (
    City VARCHAR(40),
    Postal_code VARCHAR(10),
    Region_name VARCHAR(50)
);

INSERT INTO Temp_City_Region (City, Postal_code, Region_name)
VALUES
    ('Paris', '75000', 'Île-de-France'),
    ('Marseille', '13000', 'Provence-Alpes-Côte d\'Azur'),
    ('Lyon', '69000', 'Auvergne-Rhône-Alpes'),
    ('Toulouse', '31000', 'Occitanie'),
    ('Nice', '06000', 'Provence-Alpes-Côte d\'Azur'),
    ('Nantes', '44000', 'Pays de la Loire'),
    ('Montpellier', '34000', 'Occitanie'),
    ('Strasbourg', '67000', 'Grand Est'),
    ('Bordeaux', '33000', 'Nouvelle-Aquitaine'),
    ('Lille', '59000', 'Hauts-de-France'),
    ('Rennes', '35000', 'Bretagne'),
    ('Reims', '51100', 'Grand Est'),
    ('Saint-Étienne', '42000', 'Auvergne-Rhône-Alpes'),
    ('Le Havre', '76600', 'Normandie'),
    ('Toulon', '83000', 'Provence-Alpes-Côte d\'Azur'),
    ..... With all the cities and regions

DELETE FROM City;
ALTER TABLE City AUTO_INCREMENT = 1;

INSERT INTO City (City_name, Postal_code, Region_ID)
SELECT TCR.City, TCR.Postal_code, R.Region_ID
FROM Temp_City_Region AS TCR
JOIN Region AS R ON TCR.Region_name = R.Region_name;

DROP TEMPORARY TABLE IF EXISTS Temp_City_Region;
```

For the table agency:

We also create an intermediate table to add the 25 agencies that will be located in the 25 largest cities in France:

```
DROP TEMPORARY TABLE IF EXISTS Temp_Agency_City;
```

```
CREATE TEMPORARY TABLE Temp_Agency_City (  
    City_name VARCHAR(50)  
);
```

```
INSERT INTO Temp_Agency_City (City_name)  
VALUES
```

```
    ('Paris'),  
    ('Marseille'),  
    ('Lyon'),  
    ('Toulouse'),  
    ('Nice'),  
    ('Nantes'),  
    ('Montpellier'),  
    ('Strasbourg'),  
    ('Bordeaux'),  
    ('Lille'),  
    ('Rennes'),  
    ('Reims'),  
    ('Saint-Étienne'),  
    ('Le Havre'),  
    ('Toulon'),  
    ('Grenoble'),  
    ('Dijon'),  
    ('Angers'),  
    ('Nîmes'),  
    ('Villeurbanne'),  
    ('Saint-Denis'),  
    ('Aix-en-Provence'),  
    ('Le Mans'),  
    ('Clermont-Ferrand'),  
    ('Brest')
```

```
DELETE FROM Agency;
```

```
ALTER TABLE Agency AUTO_INCREMENT = 1;
```

```
INSERT INTO Agency (City_ID)
```

```
SELECT City.City_ID
```

```
FROM Temp_Agency_City AS TAC
```

```
JOIN City ON City.City_name = TAC.City_name;
```

```
DROP TEMPORARY TABLE IF EXISTS Temp_Agency_City;
```

For the table gas:

We enter all the gases that are going to be studied as well as an information to know if they are acidifying or greenhouse effect. We have left the freedom to write whatever information we want about the gas if later we want to study gases that have other negative effects on the planet:

```
INSERT INTO livrable.Gas (Gas, Gas_type) VALUES
('SO2', 'Accidification'),
('NOx', 'Accidification'),
('NH3', 'Accidification'),
('CO', 'Accidification'),
('COVNM', 'Accidification'),
('CH4', 'Effet de serre'),
('N2O', 'Effet de serre'),
('PFC', 'Effet de serre'),
('HFC', 'Effet de serre'),
('SF6', 'Effet de serre');
```

For the table job types:

We add the 3 roles of which we have been informed. (It is important to note that there should only be one head agency per agency):

```
INSERT INTO livrable.Jobs_types (Jobs_type, Tasks) VALUES
('Head of agency', 'Responsible for overseeing all his agency '),
('Technical agent', 'Responsible for the proper functioning and maintenance of data sensors'),
('Administrative agent', 'Performs data analysis and writes air quality reports');
```

For the table sector activity

We add all the sectors of activity, mentioned in the document example given, which emit polluting gases:

```
INSERT INTO livrable.sector_activity (Sector_activity) VALUES
('Combustion in the energy and energy conversion industries'),
('Combustion outside industry'),
('Combustion in manufacturing industry'),
('Production processes'),
('Extraction and distribution of fossil fuels/geothermal energy'),
('Use of solvents and other products'),
('Road transport'),
('Other mobile sources and machinery'),
('Waste treatment and disposal'),
('Agriculture and forestry'),
('Other biotic sources');
```

To access the other tables: go to the file we have provided.

SQL Queries (12 queries)

1: List all agencies

```
SELECT Agency.Agency_ID, City.City_name, Region.Region_name
FROM Agency
JOIN City ON Agency.City_ID = City.City_ID
JOIN Region ON Region.Region_ID = City.Region_ID;
```

2: List all technical staff from the Bordeaux agency

```
SELECT Agent.Employee_ID, Agent.Name, Agent.First_name, Agent.Jobs_Type FROM Agent
JOIN Agency ON Agency.Agency_ID = Agent.Agency_ID
JOIN City ON City.City_ID = Agency.City_ID
WHERE Agent.Jobs_type = 'Technical agent' AND City.City_name = 'Bordeaux';
```

3: Provide the total number of deployed sensors

```
SELECT COUNT(Sensor_ID) FROM Sensor;
```

4: List the reports published between 2018 and 2022

```
SELECT * FROM Report
WHERE Report_date BETWEEN '2018-01-01' AND '2022-12-31';
```

5: Calculate the total greenhouse effect (effet de serre) gas emissions by region in 2020

```
SELECT Region.Region_name, SUM(Recorded_data) AS Emission_per_region_2020 FROM
technical_record
JOIN Sensor ON Sensor.Sensor_ID = technical_record.Sensor_ID
JOIN City ON City.City_ID = Sensor.City_ID
JOIN Region ON Region.Region_ID = City.Region_ID
JOIN Capture ON Capture.Sensor_ID = Sensor.Sensor_ID
JOIN Gas ON Gas.Gas_ID = Capture.Gas_ID
WHERE technical_record.Recorded_date BETWEEN '2020-01-01' AND '2020-12-31' AND
Gas.Gas_type = 'Effet de serre'
GROUP BY Region.Region_name;
```

6: Display the most polluting sector of activity in Île-de-France

```
SELECT Sector_activity.Sector_activity, SUM(Technical_record.Recorded_data) AS
Emission_per_sector FROM Technical_record
JOIN Sensor ON Sensor.Sensor_ID = Technical_record.Sensor_ID
JOIN Sector_activity ON Sector_activity.Sector_ID = Sensor.Sector_ID
JOIN City ON City.City_ID = Sensor.City_ID
JOIN Region ON Region.Region_ID = City.Region_ID
WHERE Region.Region_name = 'Île-de-France'
GROUP BY sector_activity.Sector_activity
ORDER BY Emission_per_sector DESC
LIMIT 1;
```

7: Rank the reports concerning NH3 emissions in chronological order

```
SELECT DISTINCT Report.Report_ID, Report.Report_date FROM Report
JOIN Enable_ ON Enable_.Report_ID = Report.Report_ID
JOIN Technical_Record ON Technical_record.Record_ID = Enable_.Record_ID
JOIN Gas ON Gas.Gas_ID
WHERE Gas.Gas = 'NH3'
ORDER BY Report_date;
```

8: Provide the names of technical agents maintaining sensors related to acidifying pollutants

```
SELECT DISTINCT Agent.Employee_ID, Agent.Name, Agent.First_name FROM Agent
JOIN Sensor ON Sensor.Employee_ID = Agent.Employee_ID
JOIN Capture ON Capture.Sensor_ID = Sensor.Sensor_ID
JOIN Gas ON Gas.Gas_ID = Capture.Gas_ID
WHERE Gas.Gas_type = 'Accidification' AND Agent.Active = True
ORDER BY Agent.Employee_ID;
```

9: For each gas, provide the sum of its emissions in the Ile de France region in 2020

```
SELECT Gas.Gas, SUM(Recorded_data) AS Emission FROM Technical_record
JOIN Gas ON Gas.Gas_ID = Technical_record.Gas_ID
JOIN Sensor ON Sensor.Sensor_ID = Technical_record.Sensor_ID
JOIN Sector_activity ON Sector_activity.Sector_ID = Sensor.Sector_ID
JOIN City ON City.City_ID = Sensor.City_ID
JOIN Region ON Region.Region_ID = City.Region_ID
WHERE Region.Region_name = 'Île-de-France' AND Technical_record.Recorded_date
BETWEEN '2020-01-01' AND '2020-12-31'
GROUP BY Gas.Gas
ORDER BY Gas.Gas;
```

10: Provide the productivity rate of administrative agents from the Toulouse agency (based on the number of written reports and their seniority in the position)

```
SELECT Agent.Employee_ID, Agent.Name, Agent.First_name, COUNT(Write_.Employee_ID)
/ (TIMESTAMPDIFF(YEAR, Agent.Start_date, CURDATE())) AS Productivity_per_Year FROM
Agent
JOIN Agency ON Agency.Agency_ID = Agent.Agency_ID
JOIN City ON City.City_ID = Agency.City_ID
JOIN Write_ ON Write_.Employee_ID = Agent.Employee_ID
WHERE City.City_name = 'Toulouse' AND Agent.Jobs_type = 'Administrative agent'
GROUP BY Agent.Employee_ID, Agent.Name, Agent.First_name
```

11: For a given gas, list the reports containing data concerning it (the name of the gas must be provided as a parameter):

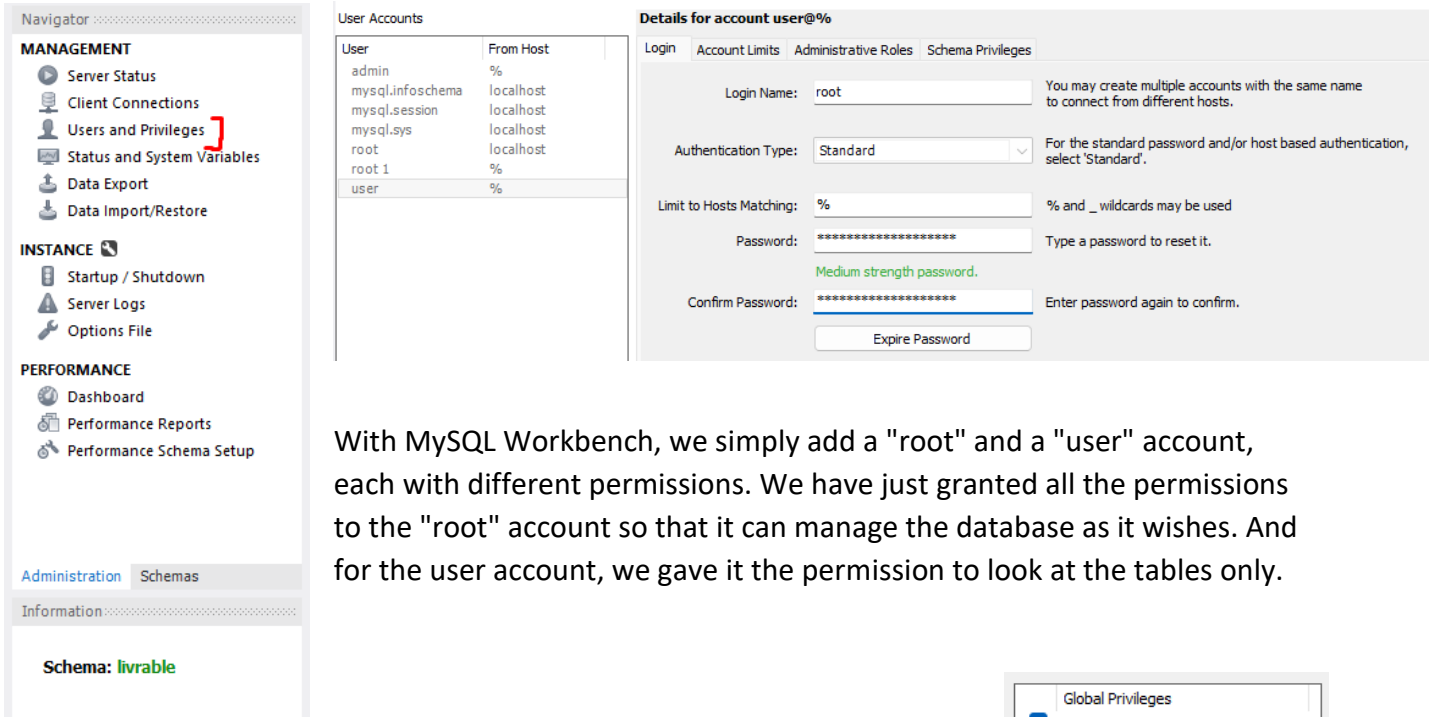
```
DROP PROCEDURE IF EXISTS Get_Report_From_Gas;
DELIMITER $$
CREATE PROCEDURE Get_Report_From_Gas(IN gas_name VARCHAR(100))
BEGIN
    SELECT DISTINCT Report.Report_ID FROM Report
    JOIN Enable_ ON Enable_.Report_ID = Report.Report_ID
    JOIN Technical_Record ON Technical_Record.Record_ID = Enable_.Record_ID
    JOIN Gas ON Gas.Gas_ID = Technical_Record.Gas_ID
    WHERE Gas.Gas = gas_name;
END $$
DELIMITER ;
```

12: List the regions where there are fewer sensors than agencies.

```
SELECT Sensor.Region_ID, Sensor.Region_name FROM (
    SELECT Region.Region_ID AS Region_ID, Region.Region_name AS Region_name,
    COUNT(Sensor.Sensor_ID) AS Sensor_NB FROM City
    JOIN Region ON Region.Region_ID = City.Region_ID
    JOIN Sensor ON Sensor.City_ID = City.City_ID
    GROUP BY Region.Region_ID
    ORDER BY Region_ID) AS Sensor
JOIN (
    SELECT Region.Region_ID AS Region_ID, COUNT(Agency.Agency_ID) AS Agency_NB
FROM City
    JOIN Region ON Region.Region_ID = City.Region_ID
    JOIN Agency ON Agency.City_ID = City.City_ID
    GROUP BY Region.Region_ID
    ORDER BY Region.Region_ID) AS Agency ON Sensor.Region_ID = Agency.Region_ID
WHERE Agency.Agency_NB > Sensor.Sensor_N;
```

Several specifications

User and root accounts



The screenshot shows the MySQL Workbench interface. On the left is the Navigator pane with sections for MANAGEMENT, INSTANCE, and PERFORMANCE. The 'Users and Privileges' icon is highlighted with a red bracket. The main area displays the 'User Accounts' table and the 'Details for account user@%' dialog.

| User | From Host |
|------------------|-----------|
| admin | % |
| mysql.infoschema | localhost |
| mysql.session | localhost |
| mysql.sys | localhost |
| root | localhost |
| root 1 | % |
| user | % |

The 'Details for account user@%' dialog shows the following fields:

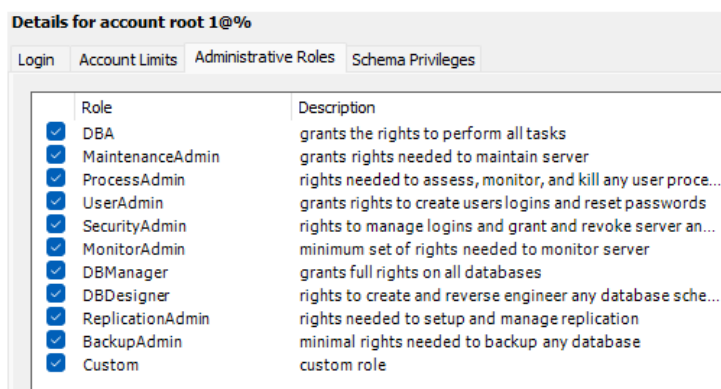
- Login Name: root
- Authentication Type: Standard
- Limit to Hosts Matching: %
- Password: (masked)
- Confirm Password: (masked)

Below the dialog, a list of 'Global Privileges' is shown, all of which are checked:

- ALTER
- ALTER ROUTINE
- CREATE
- CREATE ROUTINE
- CREATE TABLESPACE
- CREATE TEMPORARY TABLES
- CREATE USER
- CREATE VIEW
- DELETE
- DROP
- EVENT
- EXECUTE
- FILE
- GRANT OPTION
- INDEX
- INSERT
- LOCK TABLES
- PROCESS
- REFERENCES
- RELOAD
- REPLICATION CLIENT
- REPLICATION SLAVE
- SELECT
- SHOW DATABASES
- SHOW VIEW
- SHUTDOWN
- SUPER
- TRIGGER
- UPDATE

With MySQL Workbench, we simply add a "root" and a "user" account, each with different permissions. We have just granted all the permissions to the "root" account so that it can manage the database as it wishes. And for the user account, we gave it the permission to look at the tables only.

We have granted all administrative (and so all the privileges) roles to the "root" account.



The screenshot shows the 'Details for account root 1@%' dialog with the 'Administrative Roles' tab selected. The following roles are checked:

- DBA
- MaintenanceAdmin
- ProcessAdmin
- UserAdmin
- SecurityAdmin
- MonitorAdmin
- DBManager
- DBDesigner
- ReplicationAdmin
- BackupAdmin
- Custom

We can also do this directly using an SQL query:

```
GRANT SELECT ON `livrable` TO 'user'@'localhost';
GRANT ALL PRIVILEGES ON `livrable` TO 'root'@'localhost';
```

Storage engine

We decided to choose the **InnoDB** storage engine because it has many advantages:

- **Transactions and security:** InnoDB is a transaction-supported storage engine, which means it can handle multiple operations in a single transaction, ensuring data consistency. It also provides a higher level of security through the use of row locks, which allow multiple users to access the database simultaneously without risking data compromise.
- **Performance:** InnoDB is designed for performance, offering good speed in writing and reading data. It is also capable of handling large volumes of data without compromising performance.
- **High availability:** InnoDB offers high availability, thanks to the ability to set up server clusters for data replication. This means that if one of the servers fails, the data is still available on the other servers.
- **Scalability:** InnoDB is also scalable, making it easy to manage large amounts of data using efficient indexes and support